Name: Royyuru Sai Prasanna Gangadhar **Roll number: ME19B190**

Q1. 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.

a. Download the file from here: https://filesamples.com/samples/document/txt/sample1.txt

b. Write 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.

c. Write a python file, which acts as a subscriber to this topic and prints out the number of lines in the file in real-time.

(Documentation for pub/sub available at: https://pypi.org/project/google-cloud-pubsub/)

Answer: The following are the steps followed

1. Create a **requirements.txt** file and add all the required packages to the text file **requirements.txt**. Here we need "google-cloud-pubsub" and "google-cloud". Create a new bucket **"me19b190-ass6"** (google cloud functions gets triggered when a new file is added to this bucket).

```
me19b190@cloudshell:~/me19b190-assignment6-cs4830 (me19b190-project1)$ nano requirements.txt
```

Cloud Shell Editor

(me19b190-project1)

```
 GNU nano 5.4                                                    requirements.txt
google-cloud-pubsub
google-cloud
```

2. Before defining the google cloud function, it is important to create the Pub/Sub topic ("**me19b190_assignment6_topic**") to which the cloud function publishes the message and a Pub/Sub subscription (**"me19b190_assignment6_subscription"**) to the same topic.

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gcloud pubsub topics create me19b190_assignment6_topic
Created topic [projects/me19b190-project1/topics/me19b190_assignment6_topic].
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gcloud pubsub subscriptions create me19b190_assignment6_subscription --topic me19b190_assignment
6_topic
Created subscription [projects/me19b190-project1/subscriptions/me19b190_assignment6_subscription].
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$
```

3. Now write the main.py file that contains the google cloud function **"gcf_assignment_6_publisher"** that gets triggered when a new file is added to the bucket **"me19b190-ass6".**

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ nano main.py
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$
```

```
 GNU nano 5.4                                                        main.py
def gcf_assignment_6_publisher(data,context):

    from google.cloud import pubsub_v1


    data_topic = data['name'].encode('utf-8')

    publisher = pubsub_v1.PublisherClient()
    topic_id = "me19b190_assignment6_topic"
    project_id  = "me19b190-project1"
    topic_path = publisher.topic_path(project_id, topic_id)

    #publish the message to the topic
    publisher.publish(topic_path, data_topic)
    print(f"Published messages to {topic_path}.")
```

The the google cloud function is triggered when a new file is uploaded to the bucket "**me19b190-ass6".** It publishes the message (file name) to the topic **me19b190_assignment6_topic** that has been created.

4. The subscriber is created using the file **"me19b190_assignment_6_subscriber.py".** The function *callback()* in the subscriber reads file and prints the number of lines in the given file.

```
  GNU nano 5.4                                          me19b190_assignment_6_subscriber.py
from google.cloud import pubsub_v1
from concurrent.futures import TimeoutError

def callback(msg:pubsub_v1.subscriber.message.Message):
    file = msg.data.decode() #get the file name
    with open(file,'r') as input_file:
        num_lines = len(input_file.readlines())   #read number of lines in file
        print("Number of lines in the given file are: ",num_lines)  #printing the output
    msg.ack()

project_id = "me19b190-project1"     #cuurent project
subscription_id = "me19b190_assignment6_subscription"   #subscription

subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_id)

streaming_pull_future = subscriber.subscribe(subscription_path ,callback = callback)
print(f"Listening for messages on {subscription_path}..\n")

with subscriber:
    try:
        streaming_pull_future.result(timeout = 5.0)
    except TimeoutError:
        streaming_pull_future.cancel()
        streaming_pull_future.result()
```

5. Deploy the google cloud function using the following command, to trigger when a new file is uploaded to the bucket me19b190-ass6.

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gcloud functions deploy gcf_assignment_6_publisher --runtime python39 --trigger-resource me19b19
0-ass6 --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/b3395683-e0c3-4531-bcfb-0ba8fcffb721?project=775468331472
Deploying function (may take a while - up to 2 minutes)...done.
availableMemoryMb: 256
buildId: b3395683-e0c3-4531-bcfb-0ba8fcffb721
buildName: projects/775468331472/locations/us-central1/builds/b3395683-e0c3-4531-bcfb-0ba8fcffb721
dockerRegistry: CONTAINER_REGISTRY
entryPoint: gcf_assignment_6_publisher
eventTrigger:
  eventType: google.storage.object.finalize
  failurePolicy: {}
  resource: projects/_/buckets/me19b190-ass6
  service: storage.googleapis.com
ingressSettings: ALLOW_ALL
labels:
  deployment-tool: cli-gcloud
maxInstances: 3000
name: projects/me19b190-project1/locations/us-central1/functions/gcf_assignment_6_publisher
runtime: python39
serviceAccountEmail: me19b190-project1@appspot.gserviceaccount.com
sourceUploadUrl: https://storage.googleapis.com/uploads-132807434885.us-central1.cloudfunctions.appspot.com/d4540c9d-1660-4ee9-9b70-a7cec22b47ea.zip
status: ACTIVE
timeout: 60s
updateTime: '2023-03-21T17:09:52.291Z'
versionId: '8'
```

6. Upload the given input file to the bucket **me19b190-ass6**

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ wget https://filesamples.com/samples/document/txt/sample1.txt
--2023-03-21 16:26:47--  https://filesamples.com/samples/document/txt/sample1.txt
Resolving filesamples.com (filesamples.com)... 172.67.178.244, 104.21.17.252, 2606:4700:3035::6815:11fc, ...
Connecting to filesamples.com (filesamples.com)|172.67.178.244|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 607 [text/plain]
Saving to: 'sample1.txt'

sample1.txt                         100%[=============================================================================>]     607  --.-KB/s    in 0s

2023-03-21 16:26:47 (12.0 MB/s) - 'sample1.txt' saved [607/607]

me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gsutil cp sample1.txt gs://me19b190-ass6
Copying file://sample1.txt [Content-Type=text/plain]...
- [1 files][  607.0 B/  607.0 B]
Operation completed over 1 objects/607.0 B.
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$
```

7. Run the subscriber to observed the required output

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ python3 me19b190_assignment_6_subscriber.py
Listening for messages on projects/me19b190-project1/subscriptions/me19b190_assignment6_subscription..

Number of lines in the given file are:  4
```

8. Cross verifying our answer

```
Utilitatis causa amicitia est quaesita.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Collatio igitur ista te nihil iuvat. Honesta orati

Quamquam id quidem licebit iis existimare, qui legerint. Summum a vobis bonum voluptas dicitur. At hoc in e
```

Thus, the number of lines in the input file sample.txt is 4 and thus the output of the subscriber file is correct.

9. Now delete the subscriber and the topic using the following commands:

```
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gcloud pubsub subscriptions delete me19b190_assignment6_subscription
Deleted subscription [projects/me19b190-project1/subscriptions/me19b190_assignment6_subscription].
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$ gcloud pubsub topics delete me19b190_assignment6_topic
Deleted topic [projects/me19b190-project1/topics/me19b190_assignment6_topic].
me19b190@cloudshell:~/me19b190-ass6-cs4830 (me19b190-project1)$
```

Name: Royyuru Sai Prasanna Gangadhar **Roll number: ME19B190**

Q2. 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 delivery, Subscriber initiates the request to the Pub/Sub to receive messages and in a push delivery Cloud Pub/Sub initiates the request to the subscriber to deliver messages.

End points for push and pull:

1. Any authorized device on the internet can be an end point as it initiates requests from Cloud Pub/Sub.
2. Where as in case of a push subscriber the Cloud Pub/Sub sends the signals to the subscriber and thus the subscriber must be reachable via DNS name and have SSL.

The advantages of push subscriber is that they are decoupled from the Pub/Sub and do not have the credentials or use the client library of Pub/Sub.

Load balancing:

In pull mechanism, multiple subscribers can make pull calls to the same shared subscription and thus each subscriber will receive a subset of the messages.

In push mechanism, the push end point itself can be a load balancer distributing requests to the right subscribers.

Thus push mechanism is highly favourable when multiple subscriptions point to the same subscriber, for example front end. Thus, as new topics are introduced the subscriber need not be modified to accommodate the changes.

Flow of messages:

In a pull mechanism the subscriber controls the rate of delivery, while in push mechanism the Cloud Pub/Sub server automatically implements the flow control. Whenever the subscriber returns an error it back offs exponentially.

Efficiency and through put:

Pull mechanism achieves high throughput per CPU and bandwidth by allowing batch delivery processing. One of the drawbacks is that it needs the application to keep running to receive the messages from Pub/Sub.

With push delivery we can take advantage of scale to zero serverless pattern supported by cloud functions. This is useful when request rate is low or uneven. The push mechanism delivers one message per request and limits the number of outstanding messages.

A pull subscriber is preferred if:

1. Volume of messages is large(far greater than one per second)
2. If efficiency and throughput of processing is critical
3. No public HTTPS endpoint

A push subscriber is preferred is:

1. There are multiple topics that must be processed by the same webhook
2. With App engine standard and cloud function subscribers

3. In environments where GCP dependencies cannot be set up.