DataEng S24: PubSub

Chase Verbout CS510

[this lab activity references tutorials at cloud.google.com]

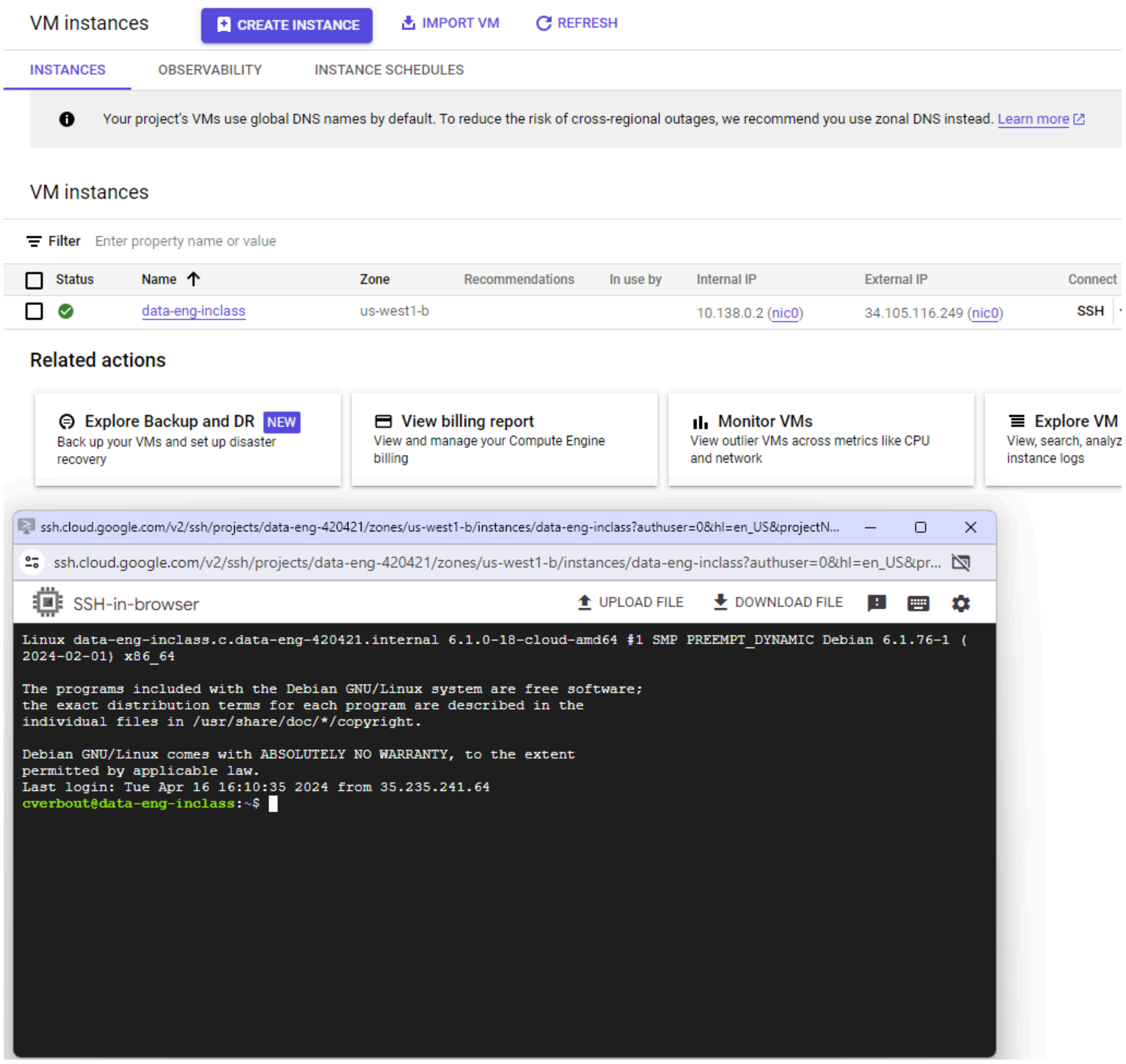
Make a copy of this document and use it to record your results. Store a PDF copy of the document in your git repository along with your code before submitting for this week. For your code, you create several publisher/receiver programs or you might make various features within one program. There is no one single correct way to do it. Regardless, store your code in your repository.

The goal for this week is to gain experience and knowledge of using an asynchronous data transport system (Google PubSub). Complete as many of the following exercises as you can. Proceed at a pace that allows you to learn and understand the use of PubSub with python.

Submit: use the in-class activity submission form which is linked from the Materials page on the class website. Submit by 10pm PT this Friday.

A. [MUST] PubSub Tutorial

1. Get your cloud.google.com account up and running
    a. Redeem your GCP coupon
    b. Login to your GCP console
    c. Create a new, separate VM instance



Your project's VMs use global DNS names by default. To reduce the risk of cross-regional outages, we recommend you use zonal DNS instead. Learn more

VM instances

Filter Enter property name or value

Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	data-eng-inclass	us-west1-b			10.138.0.2 (nic0)	34.105.116.249 (nic0)	SSH

Related actions

Explore Backup and DR NEW
Back up your VMs and set up disaster recovery

View billing report
View and manage your Compute Engine billing

Monitor VMs
View outlier VMs across metrics like CPU and network

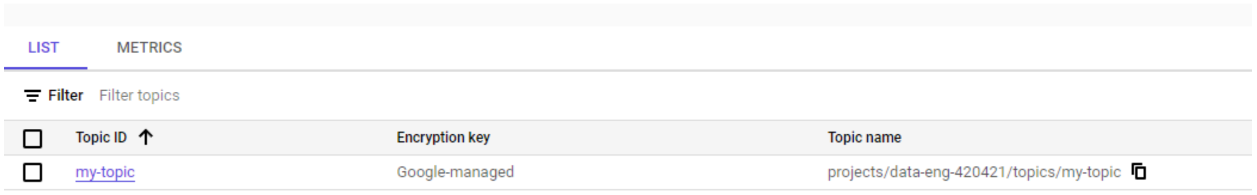
Explore VM
View, search, analyz instance logs

ssh.cloud.google.com/v2/ssh/projects/data-eng-420421/zones/us-west1-b/instances/data-eng-inclass?authuser=0&hl=en_US&projectN...

ssh.cloud.google.com/v2/ssh/projects/data-eng-420421/zones/us-west1-b/instances/data-eng-inclass?authuser=0&hl=en_US&pr...

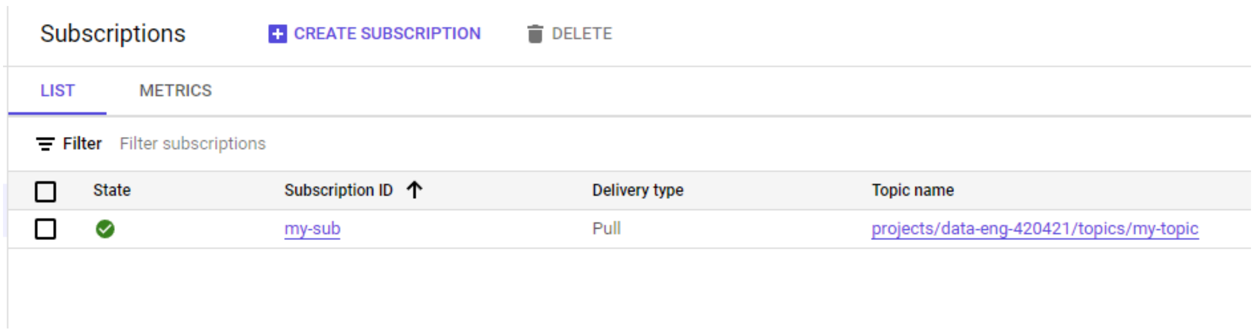
```
Linux data-eng-inclass.c.data-eng-420421.internal 6.1.0-18-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.76-1 (
2024-02-01) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr 16 16:10:35 2024 from 35.235.241.64
cverbout@data-eng-inclass:~$ 
```

2. Complete this PubSub tutorial: link Note that the tutorial instructs you to destroy your PubSub topic, but you should not destroy your topic just yet. Destroy the topic after you finish the following parts of this in-class assignment.



Filter Filter topics

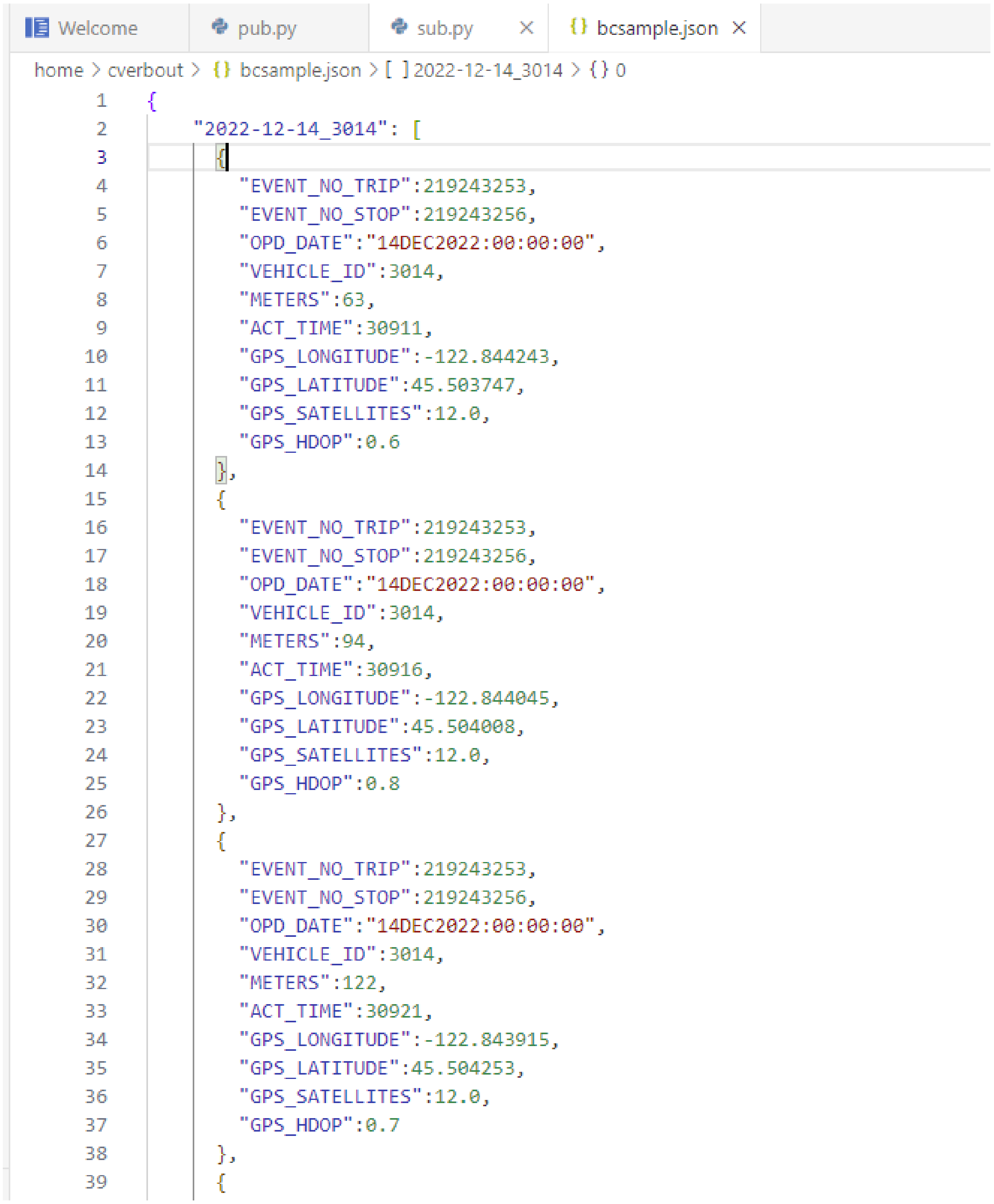
Topic ID ↑	Encryption key	Topic name
my-topic	Google-managed	projects/data-eng-420421/topics/my-topic



	State	Subscription ID ↑	Delivery type	Topic name
☐	✓	my-sub	Pull	projects/data-eng-420421/topics/my-topic

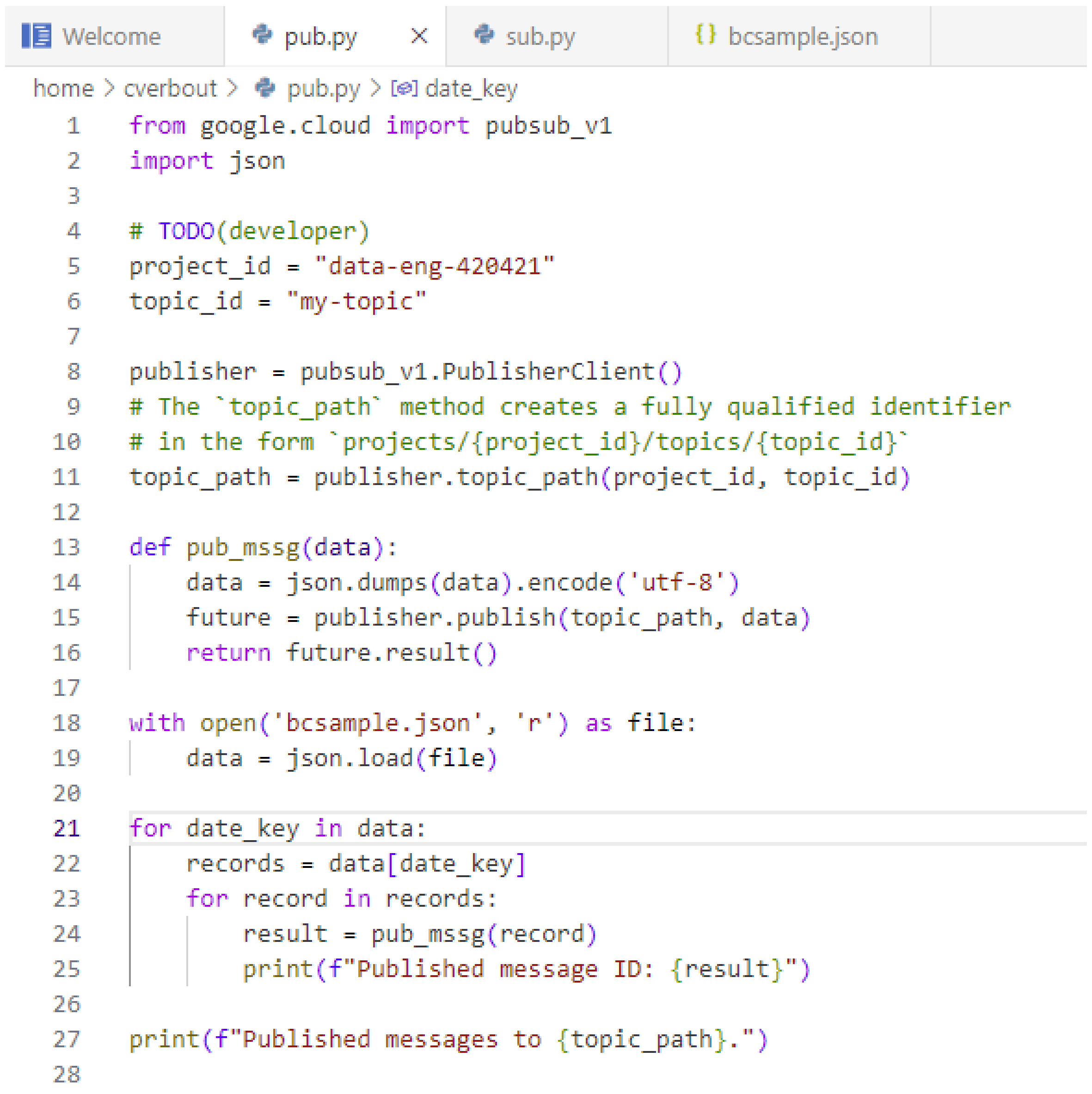
B. [MUST] Create Sample Data

1. Get data from https://busdata.cs.pdx.edu/api/getBreadCrumbs for two Vehicle IDs from among those that have been assigned to you for the class project.
2. Save this data in a sample file (named bcsample.json)


Welcome
pub.py
sub.py
bcsample.json
home > cverbout > {} bcsample.json > [ ] 2022-12-14_3014 > {} 0

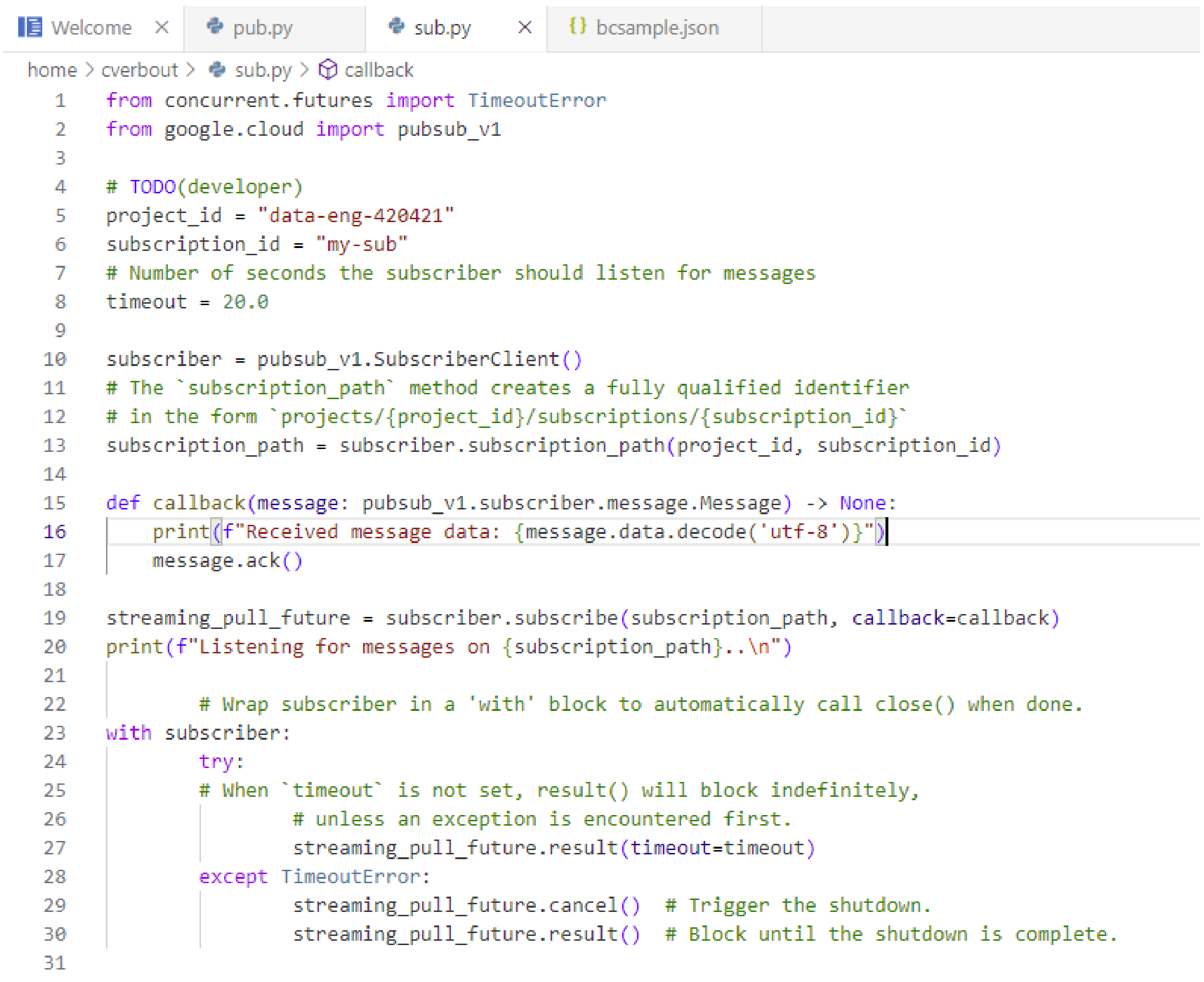
```
{
    "2022-12-14_3014": [
        {
          "EVENT_NO_TRIP":219243253,
          "EVENT_NO_STOP":219243256,
          "OPD_DATE":"14DEC2022:00:00:00",
          "VEHICLE_ID":3014,
          "METERS":63,
          "ACT_TIME":30911,
          "GPS_LONGITUDE":-122.844243,
          "GPS_LATITUDE":45.503747,
          "GPS_SATELLITES":12.0,
          "GPS_HDOP":0.6
        },
        {
          "EVENT_NO_TRIP":219243253,
          "EVENT_NO_STOP":219243256,
          "OPD_DATE":"14DEC2022:00:00:00",
          "VEHICLE_ID":3014,
          "METERS":94,
          "ACT_TIME":30916,
          "GPS_LONGITUDE":-122.844045,
          "GPS_LATITUDE":45.504008,
          "GPS_SATELLITES":12.0,
          "GPS_HDOP":0.8
        },
        {
          "EVENT_NO_TRIP":219243253,
          "EVENT_NO_STOP":219243256,
          "OPD_DATE":"14DEC2022:00:00:00",
          "VEHICLE_ID":3014,
          "METERS":122,
          "ACT_TIME":30921,
          "GPS_LONGITUDE":-122.843915,
          "GPS_LATITUDE":45.504253,
          "GPS_SATELLITES":12.0,
          "GPS_HDOP":0.7
        },
        {
```

3. Update the publisher python program that you created in the PubSub tutorial to read and parse your bcsample.json file and send its contents, one record at a time, to the my-topic PubSub topic that you created for the tutorial.



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

# TODO(developer)
project_id = "data-eng-420421"
topic_id = "my-topic"

publisher = pubsub_v1.PublisherClient()
# The `topic_path` method creates a fully qualified identifier
# in the form `projects/{project_id}/topics/{topic_id}`
topic_path = publisher.topic_path(project_id, topic_id)

def pub_mssg(data):
    data = json.dumps(data).encode('utf-8')
    future = publisher.publish(topic_path, data)
    return future.result()

with open('bcsample.json', 'r') as file:
    data = json.load(file)

for date_key in data:
    records = data[date_key]
    for record in records:
        result = pub_mssg(record)
        print(f"Published message ID: {result}")

print(f"Published messages to {topic_path}.")

```

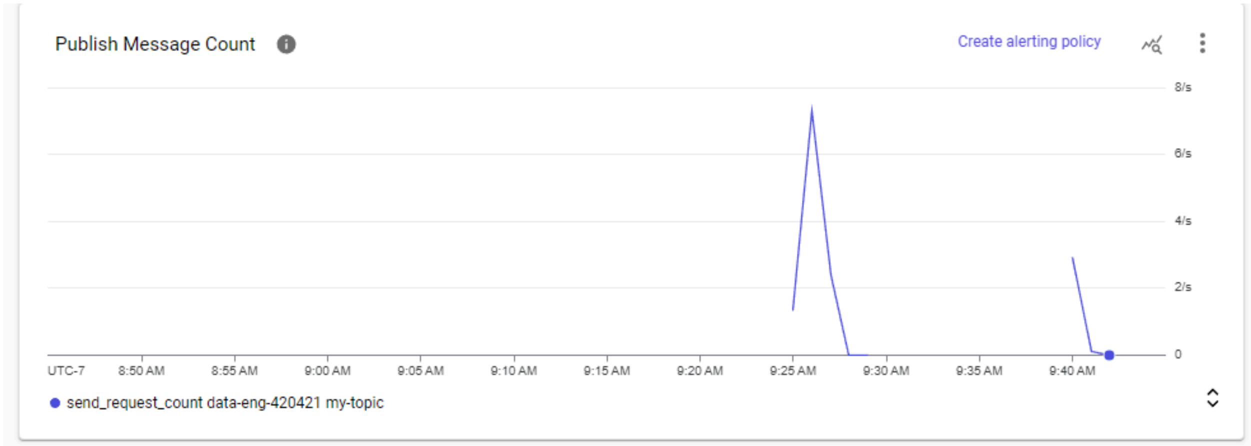
4. Use your receiver python program (from the tutorial) to consume your records.

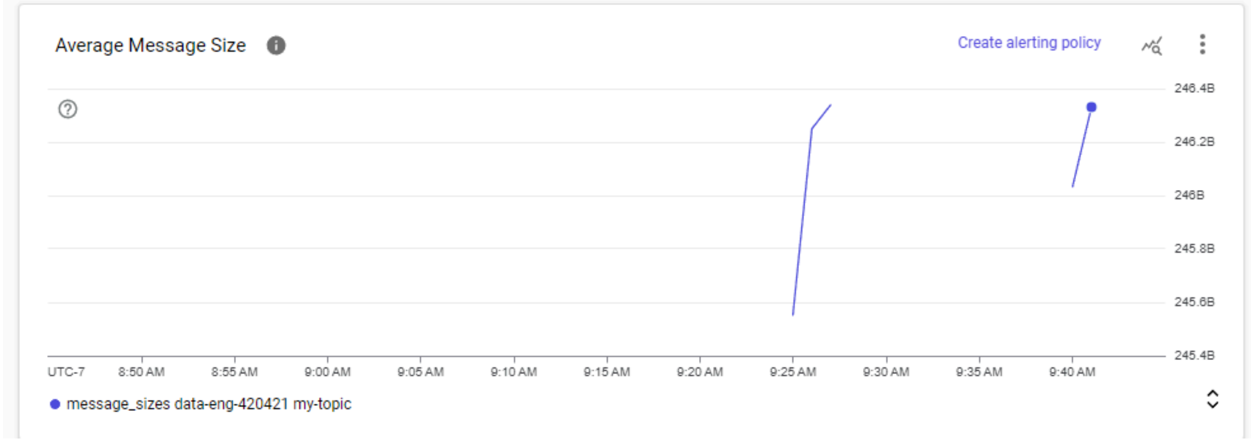


```python
from concurrent.futures import TimeoutError
from google.cloud import pubsub_v1

# TODO(developer)
project_id = "data-eng-420421"
subscription_id = "my-sub"
# Number of seconds the subscriber should listen for messages
timeout = 20.0

subscriber = pubsub_v1.SubscriberClient()
# The `subscription_path` method creates a fully qualified identifier
# in the form `projects/{project_id}/subscriptions/{subscription_id}`
subscription_path = subscriber.subscription_path(project_id, subscription_id)

def callback(message: pubsub_v1.subscriber.message.Message) -> None:
    print(f"Received message data: {message.data.decode('utf-8')}")
    message.ack()

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

        # Wrap subscriber in a 'with' block to automatically call close() when done.
with subscriber:
        try:
        # When `timeout` is not set, result() will block indefinitely,
                # unless an exception is encountered first.
                streaming_pull_future.result(timeout=timeout)
        except TimeoutError:
                streaming_pull_future.cancel()  # Trigger the shutdown.
                streaming_pull_future.result()  # Block until the shutdown is complete.

```

C. [MUST] PubSub Monitoring

1. Review the PubSub Monitoring tutorial: link and work through the steps listed there. You might need to rerun your publisher and receiver programs multiple times to trigger enough activity to monitor your my-topic effectively.





D. [MUST] PubSub Storage

1. **What happens if you run your receiver multiple times while only running the publisher once?**

Each subscriber receives its own copy of the message from the publisher. This allows subscribers to be independent of each other and work with the data in their own format.

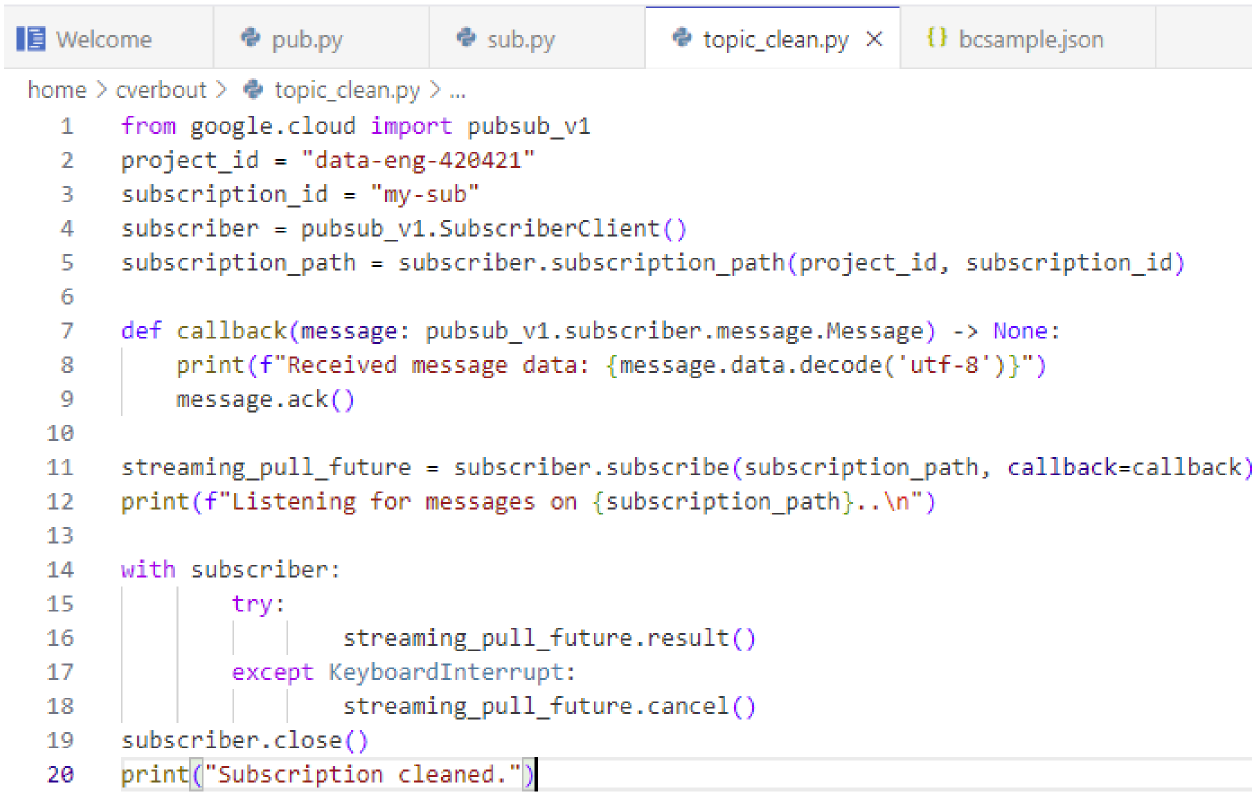
2. **Before the consumer runs, where might the data go, where might it be stored?**

The data is stored in Google's storage that is set aside for the pub/sub system. The data will remain there until it is consumed by all of the subscribers or it expires due to a time limit.

3. **Is there a way to determine how much data PubSub is storing for your topic? Do the PubSub monitoring tools help with this?**

We can use the monitoring tool to infer how much data PubSub is storing for us by looking at metrics related to undelivered messages and the size of the messages being sent. I don't see a direct metric for storage unfortunately.

4. **Create a "topic_clean.py" receiver program that reads and discards all records for a given topic. This type of program can be very useful for debugging your project code.**



```python
from google.cloud import pubsub_v1
project_id = "data-eng-420421"
subscription_id = "my-sub"
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_id)

def callback(message: pubsub_v1.subscriber.message.Message) -> None:
    print(f"Received message data: {message.data.decode('utf-8')}")
    message.ack()

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()
        except KeyboardInterrupt:
                streaming_pull_future.cancel()
subscriber.close()
print("Subscription cleaned.")
```

E. [SHOULD] Multiple Publishers

1. Clear all data from the topic (run your topic_clean.py program whenever you need to clear your topic)
2. Run two versions of your publisher concurrently, have each of them send all of your sample records. When finished, run your receiver once. Describe the results.

F. [SHOULD] Multiple Concurrent Publishers and Receivers

1. Clear all data from the topic

2. Update your publisher code to include a 250 msec sleep after each send of a message to the topic.
3. Run two or three concurrent publishers and two concurrent receivers all at the same time. Have your receivers redirect their output to separate files so that you can sort out the results more easily.
4. Describe the results.

F. [ASPIRE] Multiple Subscriptions

1. So far your receivers have all been competing with each other for data. Next, create a new subscription for each receiver so that each one receives a full copy of the data sent by the publisher. Parameterize your receiver so that you can specify a separate subscription for each receiver.
2. Rerun the multiple concurrent publishers/receivers test from the previous section. Assign each receiver to its own subscription.
3. Describe the results.