DataEng S24: PubSub

Name: Monika Kamineni
PSU ID: 920433615
Email ID: kamineni@pdx.edu

[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
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.

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)

Answer:

```
[
    {
        "EVENT_NO_TRIP": 222754149,
        "EVENT_NO_STOP": 222754163,
        "OPD_DATE": "19DEC2022:00:00:00",
        "VEHICLE_ID": 3707,
        "METERS": 5461,
        "ACT_TIME": 56067,
        "GPS_LONGITUDE": -122.502918,
        "GPS_LATITUDE": 45.496067,
        "GPS_SATELLITES": 12.0,
        "GPS_HDOP": 0.7
    },
    {
        "EVENT_NO_TRIP": 222754149,
        "EVENT_NO_STOP": 222754163,
        "OPD_DATE": "19DEC2022:00:00:00",
        "VEHICLE_ID": 3707,
        "METERS": 5517,
        "ACT_TIME": 56072,
        "GPS_LONGITUDE": -122.502225,
        "GPS_LATITUDE": 45.495922,
        "GPS_SATELLITES": 12.0,
        "GPS_HDOP": 0.7
    },
    {
        "EVENT_NO_TRIP": 222754149,
        "EVENT_NO_STOP": 222754163,
        "OPD_DATE": "19DEC2022:00:00:00",
        "VEHICLE_ID": 3707,
        "METERS": 5573,
        "ACT_TIME": 56077,
        "GPS_LONGITUDE": -122.501535,
        "GPS_LATITUDE": 45.495777,
        "GPS_SATELLITES": 12.0,
```

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.
   Answer:

```
Published message 1: 10949252422827694
Published message 2: 10948935173338703
Published message 3: 10949107065696086
Published message 4: 10949026426767970
Published message 5: 10949505697227030
Published message 6: 10949252978098378
Published message 7: 10948855451242206
Published message 8: 10948968202161266
Published message 9: 10948935843778798
```

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

Answer:

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?

   Answer:

When running multiple receivers with only one publisher, each might receive a portion of the messages. If they share a subscription, messages are divided among them, resulting in each processing only a fraction.

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

   Answer:
   Before subscribers start processing messages, data resides in Google Cloud Pub/Sub. When messages are published to a topic, they stay in storage associated with the subscription until acknowledged by a subscriber. This ensures immediate and reliable access for consumers, even if they are not connected during publication.

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

   Answer:
   Yes, there is a way to determine how much data Pub/Sub is storing for your topic. We can use Pub/Sub monitoring tools such as Cloud Monitoring to monitor the backlog size, which indicates the amount of data waiting to be processed by subscribers. Additionally, we can use Cloud Pub/Sub quotas and limits to understand the maximum amount of data that can be stored for a topic.

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.

   Answer:
   Code submitted in the git repository.

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.