

DataEng S24: PubSub

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

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?
Ans: It does not display any data second time.
2. Before the consumer runs, where might the data go, where might it be stored?
Ans: Messages are stored in a temporary message storage system managed by pub/sub.
3. Is there a way to determine how much data PubSub is storing for your topic? Do the PubSub monitoring tools help with this?
Ans: Pub/Sub has the size of the published message. It mainly focuses on data throughput rather than data storage metrics.
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.

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.
Ans: It receives the data twice.

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.

Ans: It redirects the data into different files.

4. Describe the results.

Ans:

When running multiple concurrent publishers and receivers in Google Cloud Pub/Sub, each subscriber can potentially receive a different subset of messages from the same subscription, depending on how Pub/Sub distributes the load. This distribution is typically load-balanced to ensure no single subscriber is overwhelmed. However, due to the nature of distributed systems and the asynchronous acknowledgment of messages, there's a possibility that the same message could be delivered to more than one receiver, especially if there's a delay in acknowledging the messages or if the subscribers briefly disconnect and reconnect. Each receiver writing its output to separate files enables a clear analysis of this distribution and overlap, allowing you to see which messages each subscriber processed and to ensure that all messages are being handled efficiently without significant loss or duplication.

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.

Ans: Running the test with multiple concurrent publishers and receivers, each receiver assigned to its own dedicated subscription, resulted in each receiver obtaining a complete and identical set of messages published by the publishers. This configuration ensures that all receivers operate independently, without competing for messages, thereby receiving every message sent to the topic. Such a setup validates the robustness and scalability of Google Cloud Pub/Sub in managing distinct data streams to multiple consumers simultaneously. It confirms the system's capability to distribute identical sets of data across different subscriptions efficiently, making it ideal for scenarios where identical data needs to be processed in parallel by multiple independent systems or services. This approach is particularly beneficial for fault tolerance and redundancy, as each receiver processes the same data independently, ensuring no single point of failure in data handling and processing.