

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.

DEClassActivity

Search (/) for resources, docs, products, and more

Search

Topics

CREATE TOPIC

DELETE

LIST

METRICS

Filter

Filter topics

<input type="checkbox"/>	Topic ID ↑	Encryption key	Topic name	Retention
<input type="checkbox"/>	busbreadcrumbdata	Google-managed	projects/soy-blueprint-420318/topics/busbreadcrumbdata	—

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.

Sample File Creation and Publisher Program:

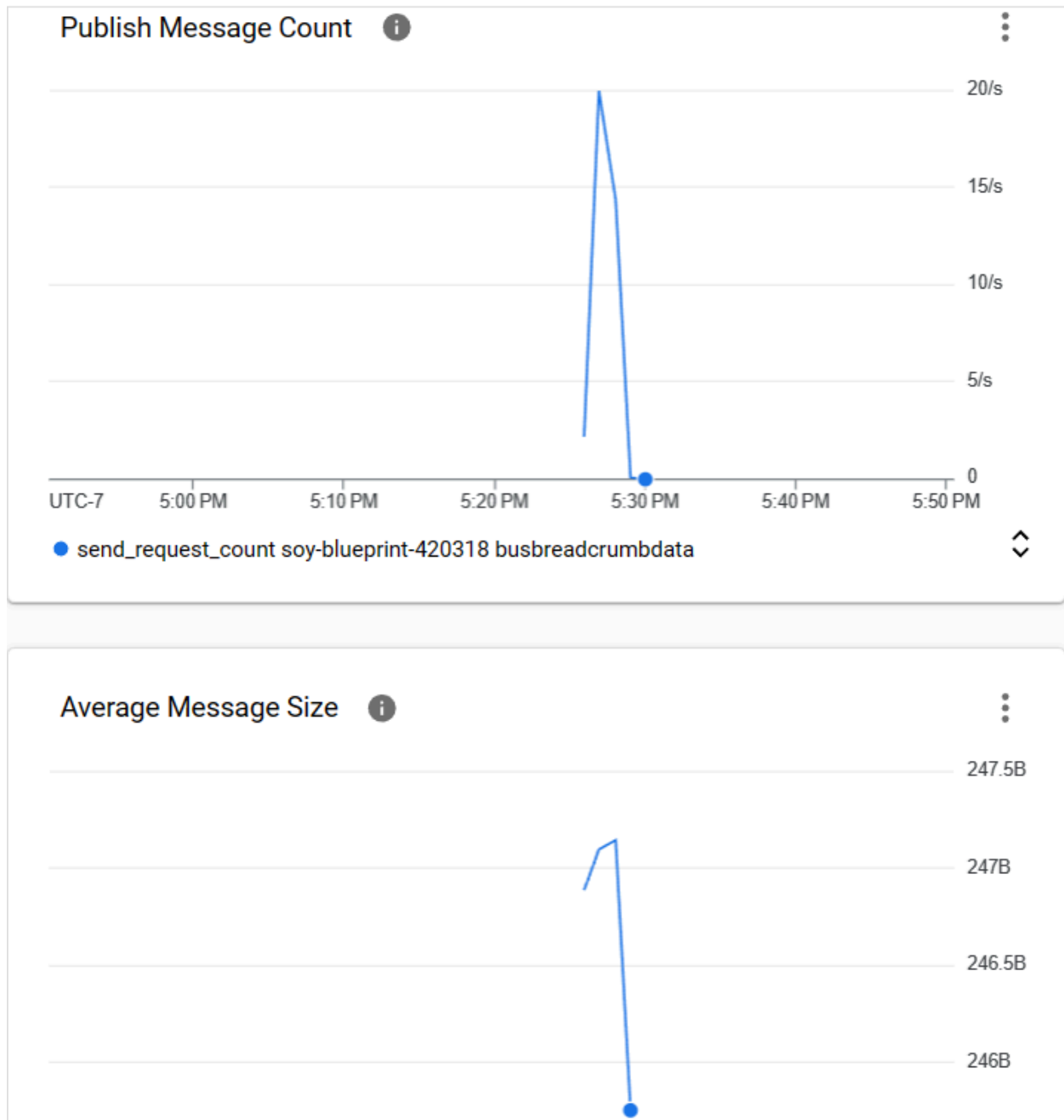
```
PS C:\Data_Engineering> & 'c:\Users\srra\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\srra\.vscode\extensions\ms-python.debugpy-2024.4.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '52817' '--' 'c:\Data_Engineering\ClassActivity_Publisher.py'
Data saved to bcsample.json
Message published. ID: 10946654856882713
Message published. ID: 10947055941028305
Message published. ID: 10946875873526471
Message published. ID: 10947345147939163
Message published. ID: 10947398180727823
Message published. ID: 10946583664532557
Message published. ID: 10947333803132563
Message published. ID: 10946699490809205
```

Receiver Program :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Received message: b'{"EVENT_NO_TRIP": 223325282, "EVENT_NO_STOP": 223325324, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3029, "METERS": 277710, "ACT_TIME": 73134, "GPS_LONGITUDE": -122.973955, "GPS_LATITUDE": 45.513845, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.7}'
Received message: b'{"EVENT_NO_TRIP": 223325282, "EVENT_NO_STOP": 223325326, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3029, "METERS": 278210, "ACT_TIME": 73204, "GPS_LONGITUDE": -122.97388, "GPS_LATITUDE": 45.518445, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.7}'
Received message: b'{"EVENT_NO_TRIP": 223325282, "EVENT_NO_STOP": 223325324, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3029, "METERS": 277774, "ACT_TIME": 73139, "GPS_LONGITUDE": -122.973933, "GPS_LATITUDE": 45.514435, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.7}'
Received message: b'{"EVENT_NO_TRIP": 223325282, "EVENT_NO_STOP": 223325325, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3029, "METERS": 278024, "ACT_TIME": 73159, "GPS_LONGITUDE": -122.973882, "GPS_LATITUDE": 45.516723, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.8}'
Received message: b'{"EVENT_NO_TRIP": 223325282, "EVENT_NO_STOP": 223325325, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3029, "METERS": 278085, "ACT_TIME": 73169, "GPS_LONGITUDE": -122.973845, "GPS_LATITUDE": 45.51731, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.7}'
```

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 : If we run the receiver (also known as a subscriber) multiple times after publishing a message once, the message will be received by only one instance of the receiver if the messages have been acknowledged. Pub/Sub

guarantees at-least-once message delivery, which means that every message that is not acknowledged will be redelivered until it is acknowledged by a subscriber.

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

Answer:

Before the consumer runs, the messages are stored in Pub/Sub within the topic they were published to. These messages are retained by the service until they're delivered to and acknowledged by a subscriber, or until they reach the maximum message retention duration (default is 7 days, but can be configured up to 31 days).

In Google Cloud Platform's Pub/Sub, messages are stored in a distributed system, not reliant on traditional disks. This system spreads data across multiple servers for durability and availability. As for clustering, Pub/Sub is architected to handle high throughput and scalability demands. It automatically scales by distributing data across clusters of servers, ensuring efficient message processing. When subscribers retrieve messages, Pub/Sub utilizes clustering to evenly distribute messages among subscriber instances, maintaining high throughput and low latency. This approach enables Pub/Sub to efficiently manage large volumes of messages, providing reliable and scalable messaging infrastructure for various applications.

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:

Google Cloud Pub/Sub does provide monitoring capabilities which is now part of Google Cloud Operations Suite. We can use Cloud Monitoring to view metrics like undelivered messages, message retention duration, and message size. This can help us determine how much data Pub/Sub is storing for our topic and subscriptions.

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 :

We can use this cleaner program to read the messages in the topic and acknowledge it so that we can run it if required to clear messages in the topic.

Alternatively, We can use Console UI option to click on Purge Messages to clear the messages unacknowledged to clear it as well.

[←](#) busbreadcru...[EDIT](#)[CREATE SNAPSHOT](#)[REPLAY MESSAGES](#)[PURGE MESSAGES](#)[DETACH](#)[DELETE](#)

Subscription name	projects/soy-blueprint-420318/subscriptions/busbreadcrumbdata-sub
Subscription state	active
Topic name	projects/soy-blueprint-420318/topics/busbreadcrumbdata

busbreadcrumbdata-sub

PERMISSIONS

Labels

Edit or delete permissions below, or select "Add Principal" to grant new access.

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.

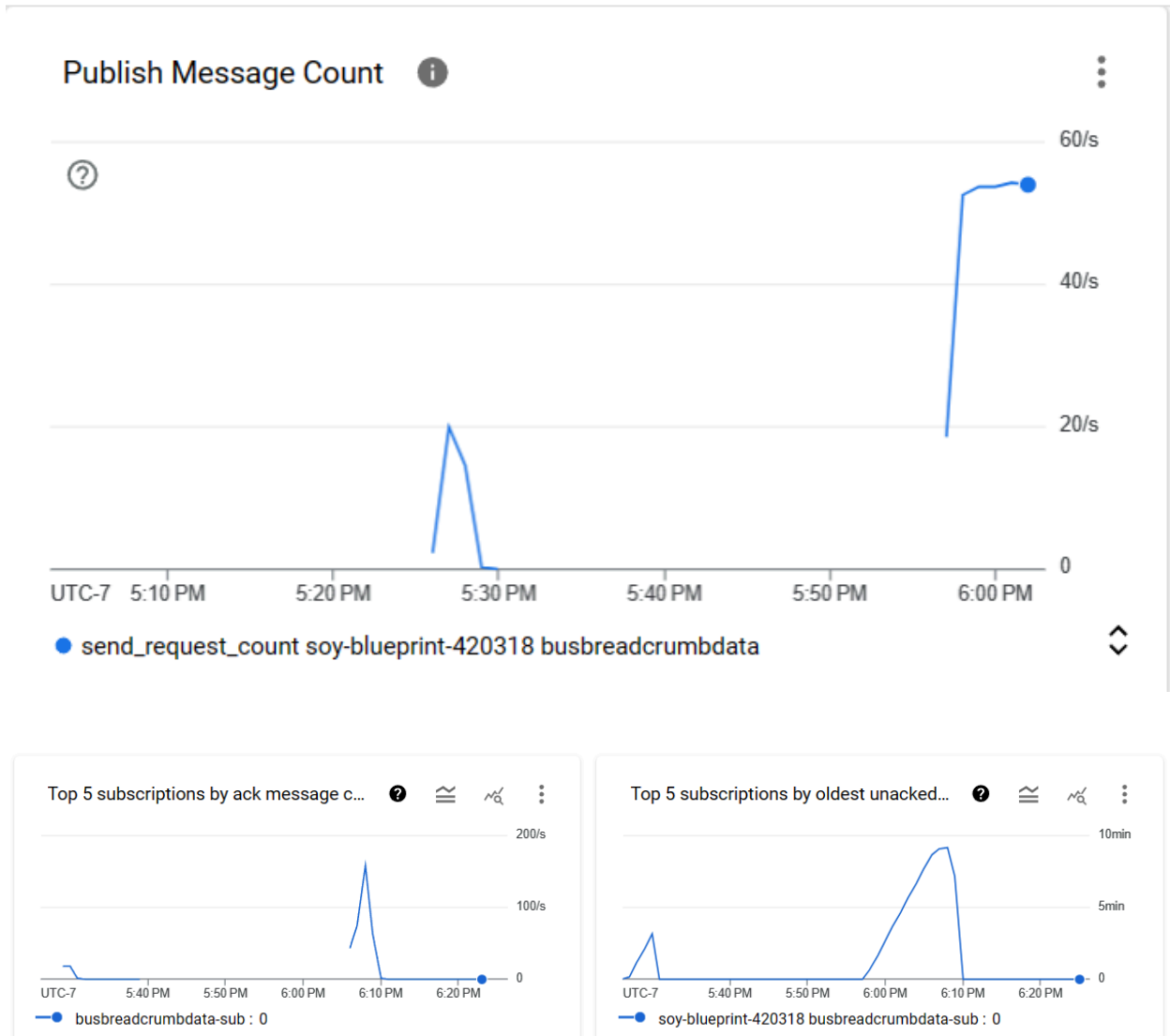
ANSWER:

Duplicate Messages got sent again and again to the receiver causing messages sent in the pipeline to be doubled tripled reflecting differences in rate especially when monitoring the Metrics of Pub/Sub in GCP.

```
Message published. ID: 10948079752298682
Message published. ID: 10947261145725447
Message published. ID: 10947163197092978
Message published. ID: 10947088496093372
Message published. ID: 10947971193184246
Message published. ID: 10946635340031981
Message published. ID: 10947148389963462
Message published. ID: 10947769573267110
Message published. ID: 10947987556673904
Message published. ID: 10947357668287265
Message published. ID: 10947387819256008
PS C:\Data_Engineering>

Message published. ID: 10947049713829437
Message published. ID: 10947989172069819
Message published. ID: 10946928014089274
Message published. ID: 10946927174657621
Message published. ID: 10947987673651427
Message published. ID: 10947954581964380
Message published. ID: 10947387659791559
Message published. ID: 10947971193185914
Message published. ID: 10948113365240274
Message published. ID: 10947172328237395
Message published. ID: 10947962908119529
PS C:\Data_Engineering>
```

```
0, "GPS_LONGITUDE": -122.746118, "GPS_LATITUDE": 45.399633, "GPS_SATELLITES": 11.0, "GPS_HDOP": 0.8}'
Received message: b'{"EVENT_NO_TRIP": 222978556, "EVENT_NO_STOP": 222978566, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3235, "METERS": 161475, "ACT_TIME": 7171
5, "GPS_LONGITUDE": -122.744955, "GPS_LATITUDE": 45.403, "GPS_SATELLITES": 11.0, "GPS_HDOP": 1.1}'
Received message: b'{"EVENT_NO_TRIP": 222978556, "EVENT_NO_STOP": 222978566, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3235, "METERS": 162064, "ACT_TIME": 7174
0, "GPS_LONGITUDE": -122.747067, "GPS_LATITUDE": 45.397872, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.8}'
Received message: b'{"EVENT_NO_TRIP": 222978556, "EVENT_NO_STOP": 222978566, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3235, "METERS": 162143, "ACT_TIME": 7174
5, "GPS_LONGITUDE": -122.747512, "GPS_LATITUDE": 45.397223, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.8}'
Received message: b'{"EVENT_NO_TRIP": 222978556, "EVENT_NO_STOP": 222978566, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3235, "METERS": 162218, "ACT_TIME": 7175
5, "GPS_LONGITUDE": -122.747952, "GPS_LATITUDE": 45.396607, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.8}'
Received message: b'{"EVENT_NO_TRIP": 222978556, "EVENT_NO_STOP": 222978566, "OPD_DATE": "20DEC2022:00:00:00", "VEHICLE_ID": 3235, "METERS": 162198, "ACT_TIME": 7175
0, "GPS_LONGITUDE": -122.747858, "GPS_LATITUDE": 45.396785, "GPS_SATELLITES": 12.0, "GPS_HDOP": 0.8}'
```

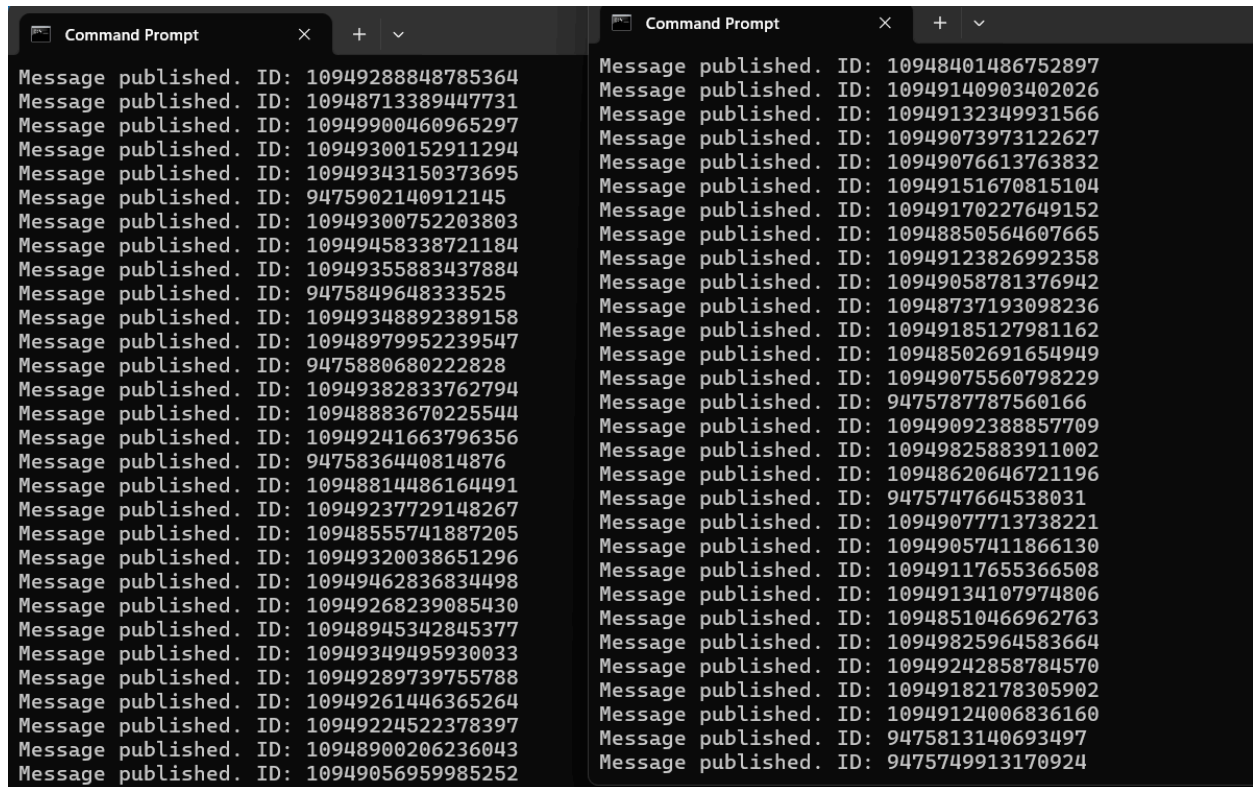


F. [SHOULD] Multiple Concurrent Publishers and Receivers

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.

ANSWER:

I modified the code to sleep for publishing for 250ms and ran 2 multiple publishers and then 2 consumers simultaneously. I noticed the metrics to be controlled now and also the 2 files separately for the receiver seemed to be near parallel with similar data.



The image shows two side-by-side Windows Command Prompt windows. Both windows display a continuous stream of log messages. Each message follows the format: "Message published. ID: [unique ID]". The IDs in the left window range from 10949288848785364 down to 10949056959985252. The IDs in the right window range from 10948401486752897 down to 9475749913170924. The windows are titled "Command Prompt" and have standard window controls (minimize, maximize, close) and a tab bar with a "+" button and a dropdown arrow.

Output File for First Subscriber Program :

https://drive.google.com/file/d/1G8kyfXIY76uPm3EgNDRlnLNorlIQVS0l/view?usp=drive_link

Output File for Second Subscriber Program:

https://drive.google.com/file/d/1G8kyfXIY76uPm3EgNDRlnLNorlIQVS0l/view?usp=drive_link

G. [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.

ANSWER:

We ran the test from previous section and noticed that both receivers are getting the same message and exact files now.

We gave first Subscription for the first instance and Second Subscription created for the second Instance Run and we found this.

LIST

METRICS

Filter

Filter subscriptions

?

⌵

<input type="checkbox"/>	State	Subscription ID ↑	Delivery type	Topic name	Ack deadline	Retention	Message ordering	Exactly once del	
<input type="checkbox"/>	✓	busbedcrumbdatasub2	Pull	projects/soy-blueprint-420318/top...	10 seconds	7 days	Disabled	Disabled	⋮
<input type="checkbox"/>	✓	busbreadcrumbdata-sub	Pull	projects/soy-blueprint-420318/top...	10 seconds	7 days	Disabled	Disabled	⋮