DataEng S23: Kafka

[this lab activity references tutorials at confluence.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 producer/consumer programs, or you might make various features within one program. There is not 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 a streaming data transport system (Kafka). Complete as many of the following exercises as you can. Proceed at a pace that allows you to learn and understand the use of Kafka with python.

Submit: use the in-class activity submission form which is linked from the Materials page on the class website. Submit by 5pm PT this Friday.

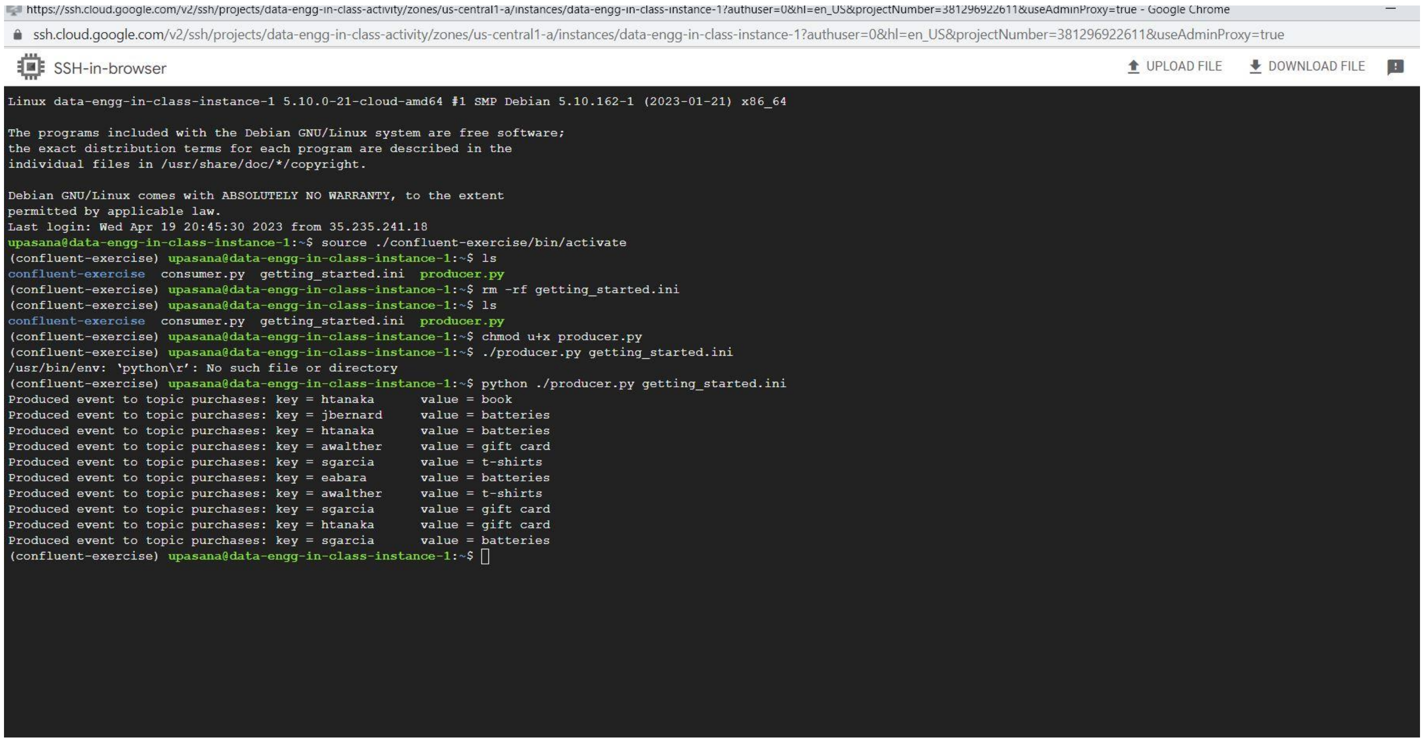
A. Initialization

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.

Answer – Completed

2. Follow the Kafka tutorial from project assignment #1
    a. Create a separate topic for this in-class activity.
    b. Make it “small” as you will not want to use many resources for this activity. By “small” I mean that you should choose medium or minimal options when asked for any configuration decisions about the topic, cluster, partitions, storage, anything. GCP/Confluent will ask you to choose the configs, and because you are using a free account you should opt for limited resources where possible.
    c. Get a basic producer and consumer working with a Kafka topic as described in the tutorials.

Answer – Completed.


https://ssh.cloud.google.com/v2/ssh/projects/data-engg-in-class-activity/zones/us-central1-a/instances/data-engg-in-class-instance-1?authuser=0&hl=en_US&projectNumber=381296922611&useAdminProxy=true - Google Chrome
ssh.cloud.google.com/v2/ssh/projects/data-engg-in-class-activity/zones/us-central1-a/instances/data-engg-in-class-instance-1?authuser=0&hl=en_US&projectNumber=381296922611&useAdminProxy=true
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE

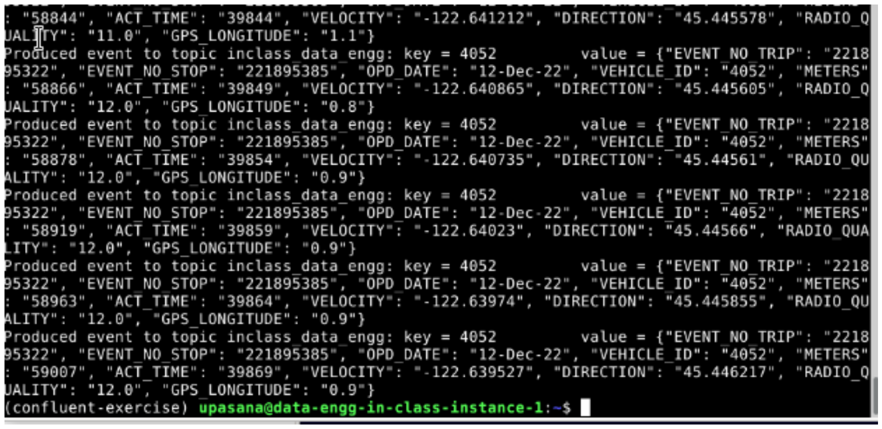
```
Linux data-engg-in-class-instance-1 5.10.0-21-cloud-amd64 #1 SMP Debian 5.10.162-1 (2023-01-21) x86_64

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: Wed Apr 19 20:45:30 2023 from 35.235.241.18
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activate
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ ls
confluent-exercise  consumer.py  getting_started.ini  producer.py
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ rm -rf getting_started.ini
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ ls
confluent-exercise  consumer.py  getting_started.ini  producer.py
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ chmod u+x producer.py
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ ./producer.py getting_started.ini
/usr/bin/env: 'python\r': No such file or directory
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.py getting_started.ini
Produced event to topic purchases: key = htanaka      value = book
Produced event to topic purchases: key = jbernard     value = batteries
Produced event to topic purchases: key = htanaka      value = batteries
Produced event to topic purchases: key = awalther     value = gift card
Produced event to topic purchases: key = sgarcia      value = t-shirts
Produced event to topic purchases: key = eabara       value = batteries
Produced event to topic purchases: key = awalther     value = t-shirts
Produced event to topic purchases: key = sgarcia      value = gift card
Produced event to topic purchases: key = htanaka      value = gift card
Produced event to topic purchases: key = sgarcia      value = batteries
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ 
```

3. Create a sample breadcrumb data file (named bcsample.json) consisting of a sample of 1000 breadcrumb records. These can be any records because we will not be concerned with the actual contents of the breadcrumb records during this assignment. One way to do this is by using the Linux command "head" to get the first n lines from one of the bread crumb data files and create a new file from that.

Answer- head -n 12002 2023-04-20.json > bcsample.json
Because selecting 12002 gives you 12002 lines of data which will be equal to 1000 records.

4. Update your producer to parse your bcsample.json file and send its contents, one record at a time, to the Kafka topic.

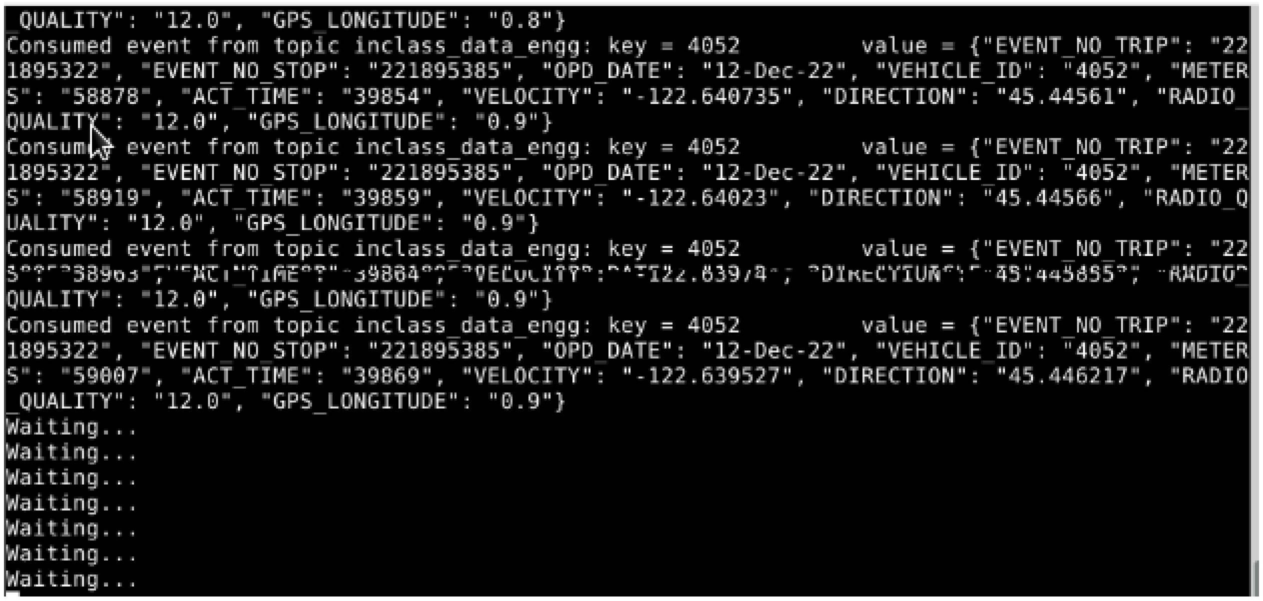
Answer – Completed.



```
: "58844", "ACT_TIME": "39844", "VELOCITY": "-122.641212", "DIRECTION": "45.445578", "RADIO_Q
UALITY": "11.0", "GPS_LONGITUDE": "1.1"}
Produced event to topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "2218
95322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METERS"
: "58866", "ACT_TIME": "39849", "VELOCITY": "-122.640865", "DIRECTION": "45.445605", "RADIO_Q
UALITY": "12.0", "GPS_LONGITUDE": "0.8"}
Produced event to topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "2218
95322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METERS"
: "58878", "ACT_TIME": "39854", "VELOCITY": "-122.640735", "DIRECTION": "45.44561", "RADIO_QU
ALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Produced event to topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "2218
95322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METERS"
: "58919", "ACT_TIME": "39859", "VELOCITY": "-122.64023", "DIRECTION": "45.44566", "RADIO_QUA
LITY": "12.0", "GPS_LONGITUDE": "0.9"}
Produced event to topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "2218
95322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METERS"
: "58963", "ACT_TIME": "39864", "VELOCITY": "-122.63974", "DIRECTION": "45.445855", "RADIO_QU
ALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Produced event to topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "2218
95322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METERS"
: "59007", "ACT_TIME": "39869", "VELOCITY": "-122.639527", "DIRECTION": "45.446217", "RADIO_Q
UALITY": "12.0", "GPS_LONGITUDE": "0.9"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ 
```

5. Use your consumer.py program (from the tutorial) to consume your records.

Answer – Python file in GitHub.



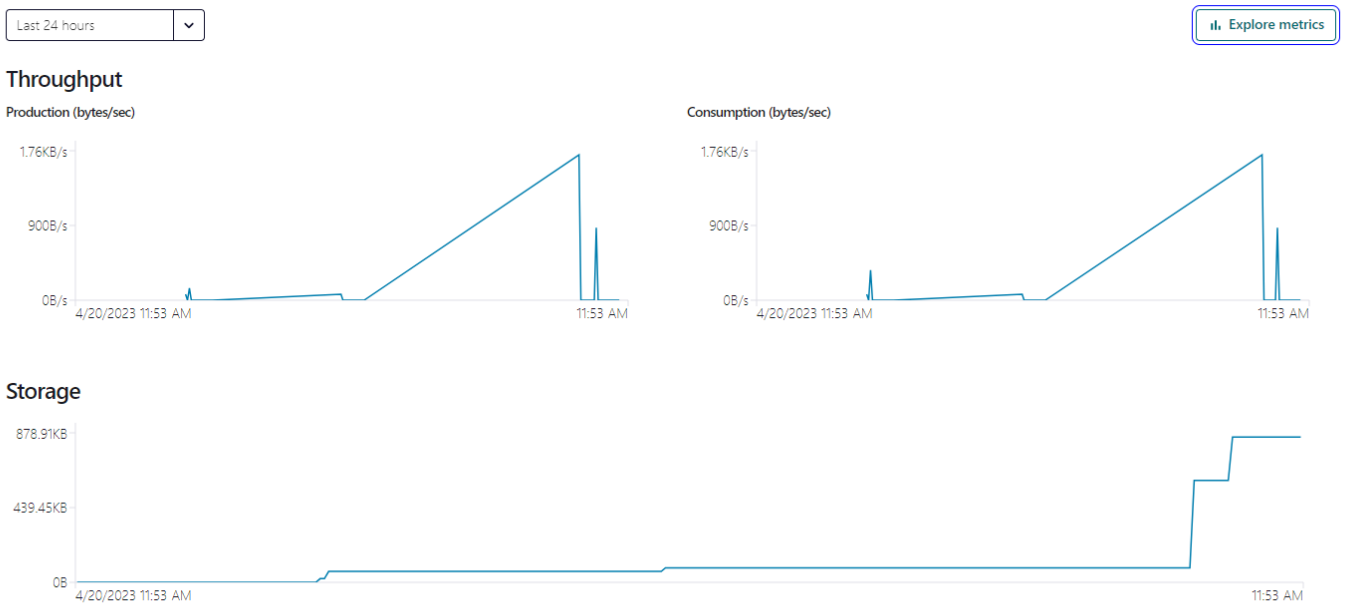
```
_QUALITY": "12.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "22
1895322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METER
S": "58878", "ACT_TIME": "39854", "VELOCITY": "-122.640735", "DIRECTION": "45.44561", "RADIO_
QUALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Consumed event from topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "22
1895322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METER
S": "58919", "ACT_TIME": "39859", "VELOCITY": "-122.64023", "DIRECTION": "45.44566", "RADIO_Q
UALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Consumed event from topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "22
S": "58963", "ACT_TIME": "39864", "VELOCITY": "-122.63974", "DIRECTION": "45.445855", "RADIO_
QUALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Consumed event from topic inclass_data_engg: key = 4052         value = {"EVENT_NO_TRIP": "22
1895322", "EVENT_NO_STOP": "221895385", "OPD_DATE": "12-Dec-22", "VEHICLE_ID": "4052", "METER
S": "59007", "ACT_TIME": "39869", "VELOCITY": "-122.639527", "DIRECTION": "45.446217", "RADIO
_QUALITY": "12.0", "GPS_LONGITUDE": "0.9"}
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
```

B. Kafka Monitoring

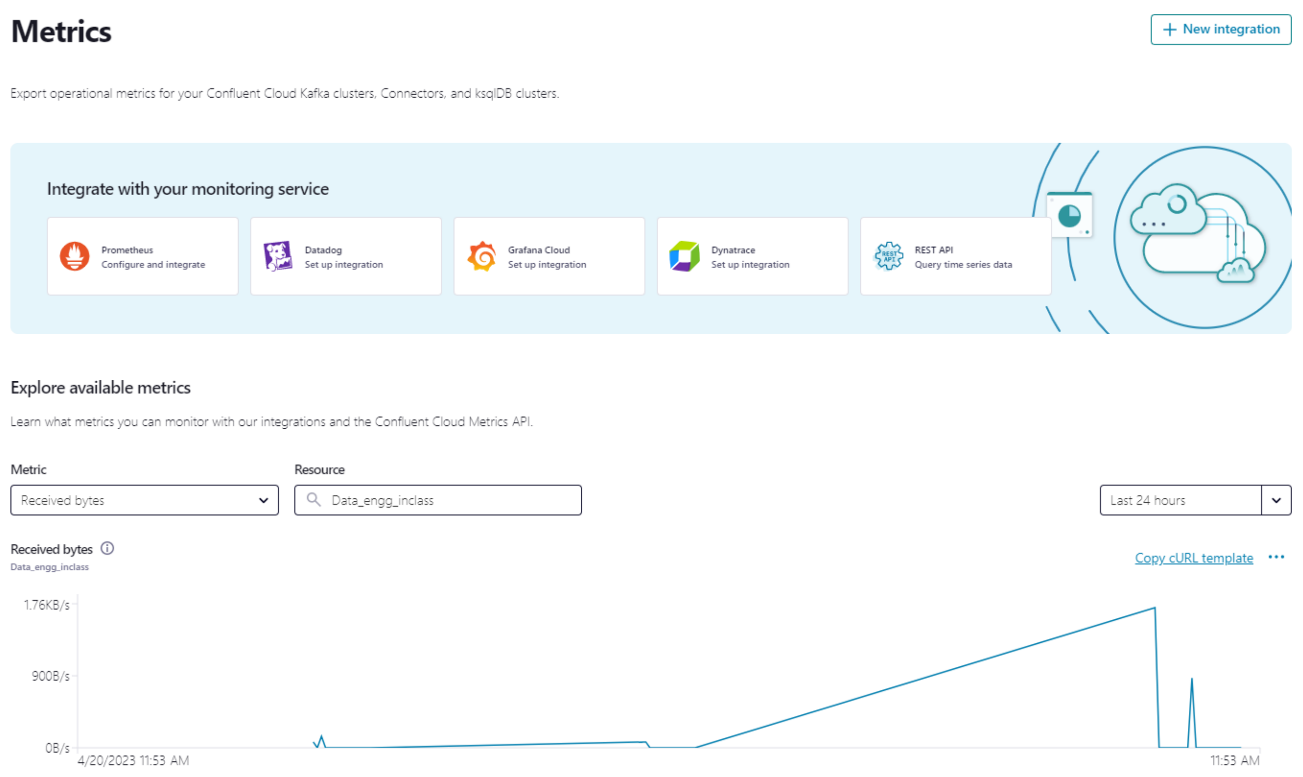
1. Tools for monitoring your Kafka topic. For example, the cluster overview, or the topic overview, or the stream lineage. Which area do you think will be the best way to monitor data flow on your topic? Briefly describe its contents. Does it measure throughput, or total messages produced into Kafka and consumed out of Kafka? Do the measured values seem reasonable to you?

Answer –

Cluster Overview –Cluster overview has dashboard where you can see a high-level view of your Kafka cluster in Confluent Cloud. It displays throughput and storage of the cluster you are using. When you click on explore metrics then you can see a little in detailed view.



As shown in the below screenshot, we can monitor our cluster and view information regarding the received bytes, sent bytes etc. The graph below shows us the received bytes in the last 24 hours. You can change the metrics using the drop-down menu.



Metrics

+ New integration

Export operational metrics for your Confluent Cloud Kafka clusters, Connectors, and ksqlDB clusters.

Integrate with your monitoring service

Explore available metrics

Learn what metrics you can monitor with our integrations and the Confluent Cloud Metrics API.

Metric: Received bytes

Resource: Data_engg_inclass

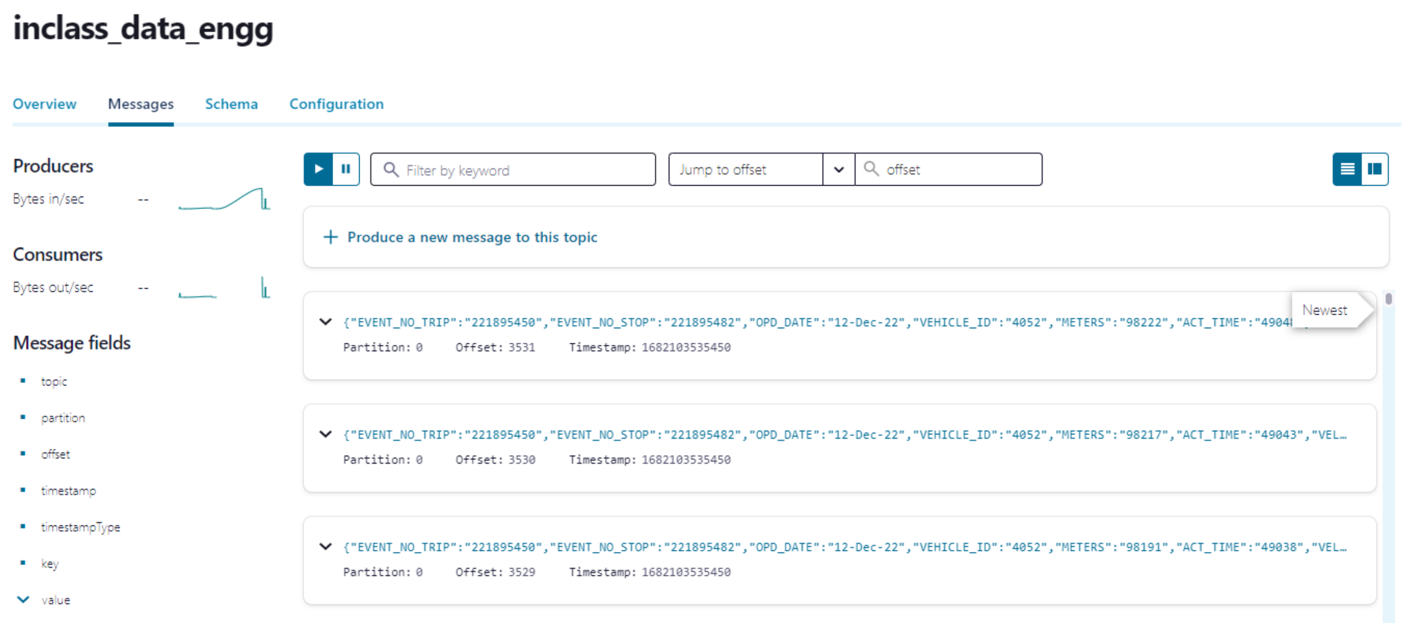
Last 24 hours

Received bytes
Data_engg_inclass

Topic Overview –

This tool allows you to monitor the performance of individual topics in your Kafka cluster. It provides detailed information about each topic, including the number of messages produced and consumed, message throughput, and message size distribution. It also allows you to monitor consumer lag and set up alerts for specific metrics.

When running the producer, you can monitor the real time data in the topic overview. The below image shows us the messages produced by the producers.



inclass_data_engg

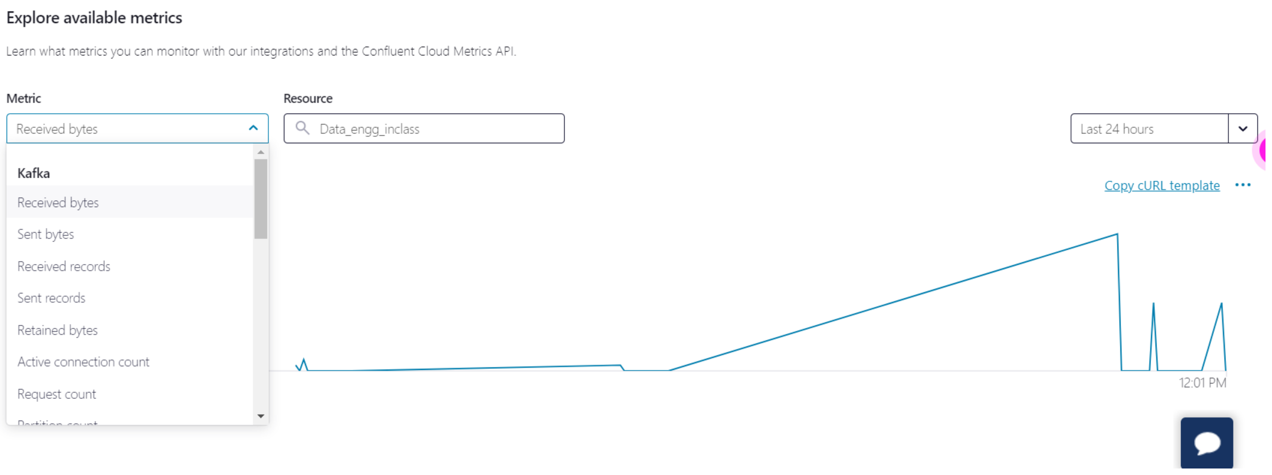
Producers
Bytes in/sec --

Consumers
Bytes out/sec --

Message fields
- topic
- partition
- offset
- timestamp
- timestampType
- key
- value

```
{"EVENT_NO_TRIP":"221895450","EVENT_NO_STOP":"221895482","OPD_DATE":"12-Dec-22","VEHICLE_ID":"4052","METERS":"98222","ACT_TIME":"4904
Partition: 0    Offset: 3531    Timestamp: 1682103535450

{"EVENT_NO_TRIP":"221895450","EVENT_NO_STOP":"221895482","OPD_DATE":"12-Dec-22","VEHICLE_ID":"4052","METERS":"98217","ACT_TIME":"49043","VEL…
Partition: 0    Offset: 3530    Timestamp: 1682103535450

{"EVENT_NO_TRIP":"221895450","EVENT_NO_STOP":"221895482","OPD_DATE":"12-Dec-22","VEHICLE_ID":"4052","METERS":"98191","ACT_TIME":"49038","VEL…
Partition: 0    Offset: 3529    Timestamp: 1682103535450
```

In the below image, we have a detailed overview of topic where we can monitor our topic and view information regarding the received bytes, sent bytes, received records, set records, retained bytes etc. The graph below shows us the received bytes in the last 24 hours. You can change the metrics using the drop-down menu.

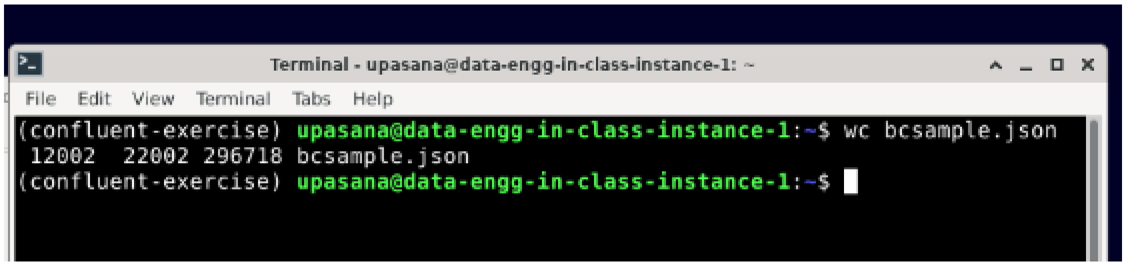


Stream lineage: This tool allows you to visualize the flow of data between Kafka topics and other data systems in your architecture. It provides a graphical representation of your data pipeline and allows you to monitor data flow and troubleshoot issues.

2. Use this monitoring feature as you do each of the following exercises.

C. Kafka Storage

1. Run the linux command “wc bcsample.json”. Record the output here so that we can verify that your sample data file is of reasonable size.



```
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ wc bcsample.json
 12002  22002 296718 bcsample.json
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$
```

2. What happens if you run your consumer multiple times while only running the producer once?

Answer – I ran producer once and consumer multiple times, what I saw is once it has consumed the data it is just waiting for more data to consume. I forced stop the consumer and ran it multiple times but once it has consumed the data it will be forever waiting to consume more data.

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

Answer – The data is stored in the partitions of the topic. The data will remain in the topic partitions until the data is consumed.

4. Is there a way to determine how much data Kafka/Confluent is storing for your topic? Do the Confluent monitoring tools help with this?

Yes, there is a way to determine how much data Kafka/Confluent is storing for your topic. Confluent Control Center and other monitoring tools can provide metrics on the size of Kafka topics and the total storage used by the Kafka cluster.

5. Create a “topic_clean.py” consumer that reads and discards all records for a given topic. This type of program can be very useful during debugging.
   Created in GitHub.

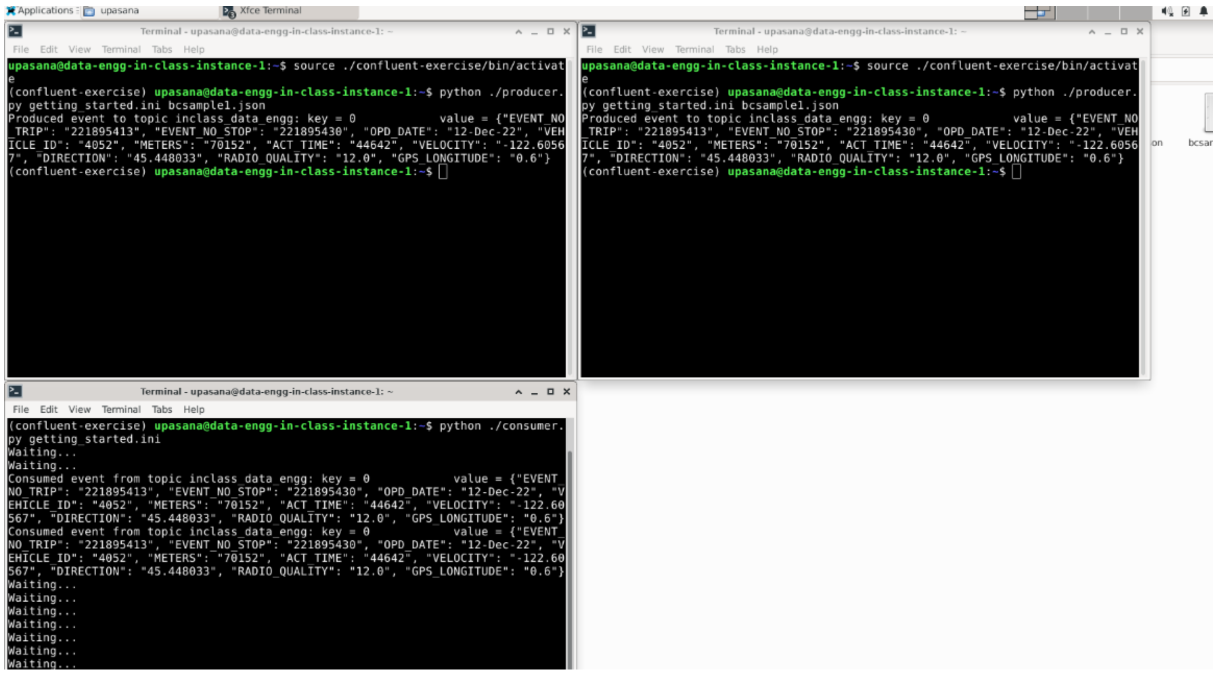
D. Multiple Producers

1. Clear all data from the topic.
   I could not clean the data from the topic with the topic_clean.py file hence, I am deleting the topic from the confluent cloud and then recreating it to check the behavior of the following question.

2. Run two versions of your producer concurrently, have each of them send all 1000 of your sample records. When finished, run your consumer once. Describe the results.

   When we run two version of producers concurrently and have each of them send 1000 records and run consumer once- what I saw is when we are done producing, consumer consumes 2000 records, and the records are duplicate. For every single key it consumes 2 records.

   As the records are 1000 and it was difficult to understand what exactly is happening, I created a sample json file with only one records in it and did the sample operation in the below image we can see how consumer consumes two records with the same key.


Applications
upasana
Xfce Terminal
Terminal - upasana@data-engg-in-class-instance-1: ~
File Edit View Terminal Tabs Help

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic inclass_data_engg: key = 0           value = {"EVENT_NO
_TRIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEH
ICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.6056
7", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$
```

Terminal - upasana@data-engg-in-class-instance-1: ~
File Edit View Terminal Tabs Help

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic inclass_data_engg: key = 0           value = {"EVENT_NO
_TRIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEH
ICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.6056
7", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$
```

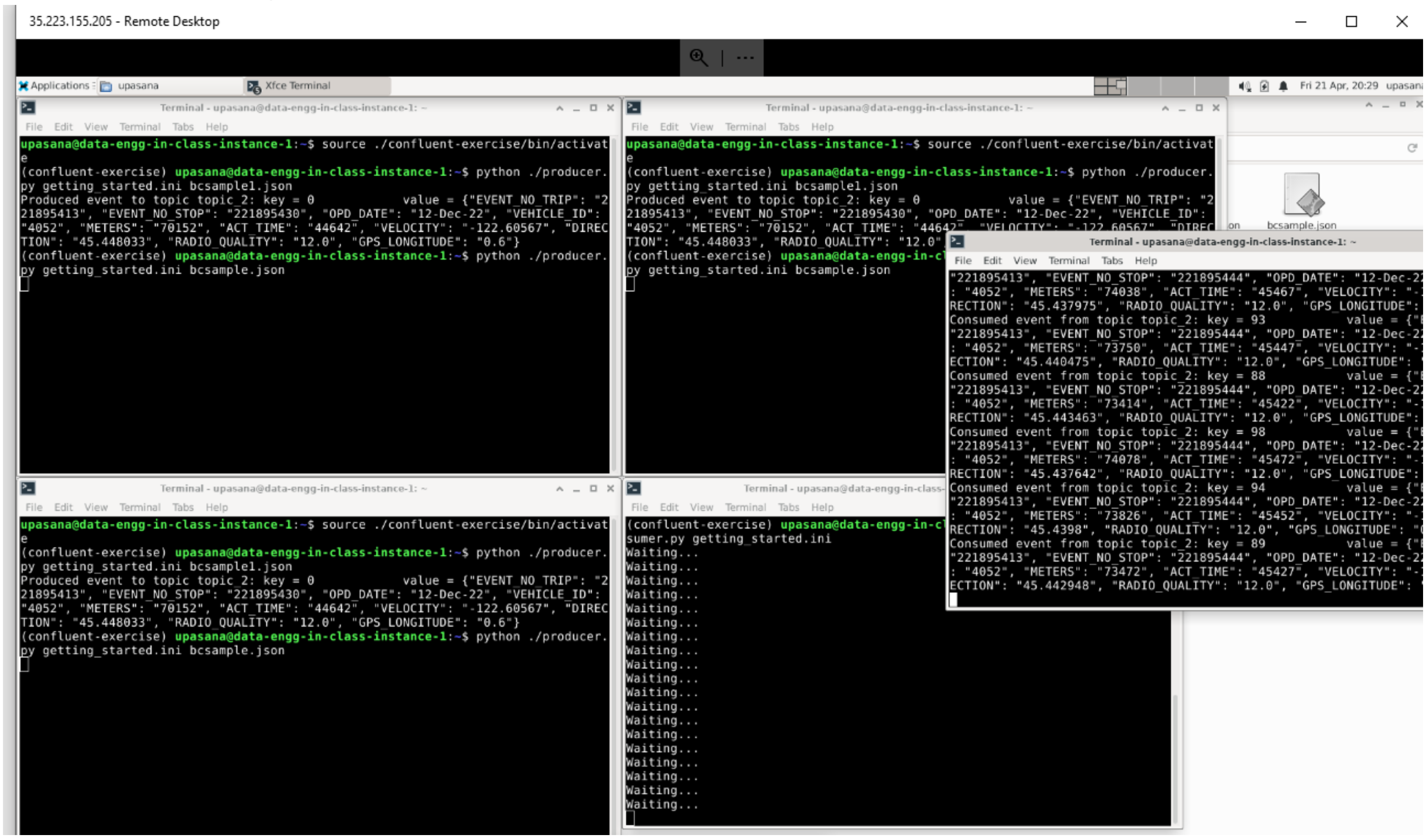
Terminal - upasana@data-engg-in-class-instance-1: ~
File Edit View Terminal Tabs Help

```
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./consumer.
py getting_started.ini
Waiting...
Waiting...
Consumed event from topic inclass_data_engg: key = 0           value = {"EVENT_
NO_TRIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "V
EHICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60
567", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
Consumed event from topic inclass_data_engg: key = 0           value = {"EVENT_
NO_TRIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "V
EHICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60
567", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
```

E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic.
   Deleted the topic and recreated one.
2. Update your Producer code to include a 250 msec sleep after each send of a message to the topic.
   Added one line of code in the producer.py
   sleep(0.25)
3. Run two or three concurrent producers and two concurrent consumers all at the same time.
   Again, I tried it both ways with 1000 records and for understanding I also tried with one record json file.
4. Describe the results.

For 1000 records, the producer sends the data with the sleep time but it is visible in the terminal after 250 seconds. In the below image the producer is yet to print because as the code is written the print statement will be executed once all the data is produced and after the wait of 250 msec after every record which is after 250 seconds. Also the first consumer consumes all the data while the second consumer is waiting for the data.


35.223.155.205 - Remote Desktop
Applications
upasana
Xfce Terminal
Fri 21 Apr, 20:29
Terminal - upasana@data-engg-in-class-instance-1: ~
File Edit View Terminal Tabs Help
bcsample.json

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample.json
```

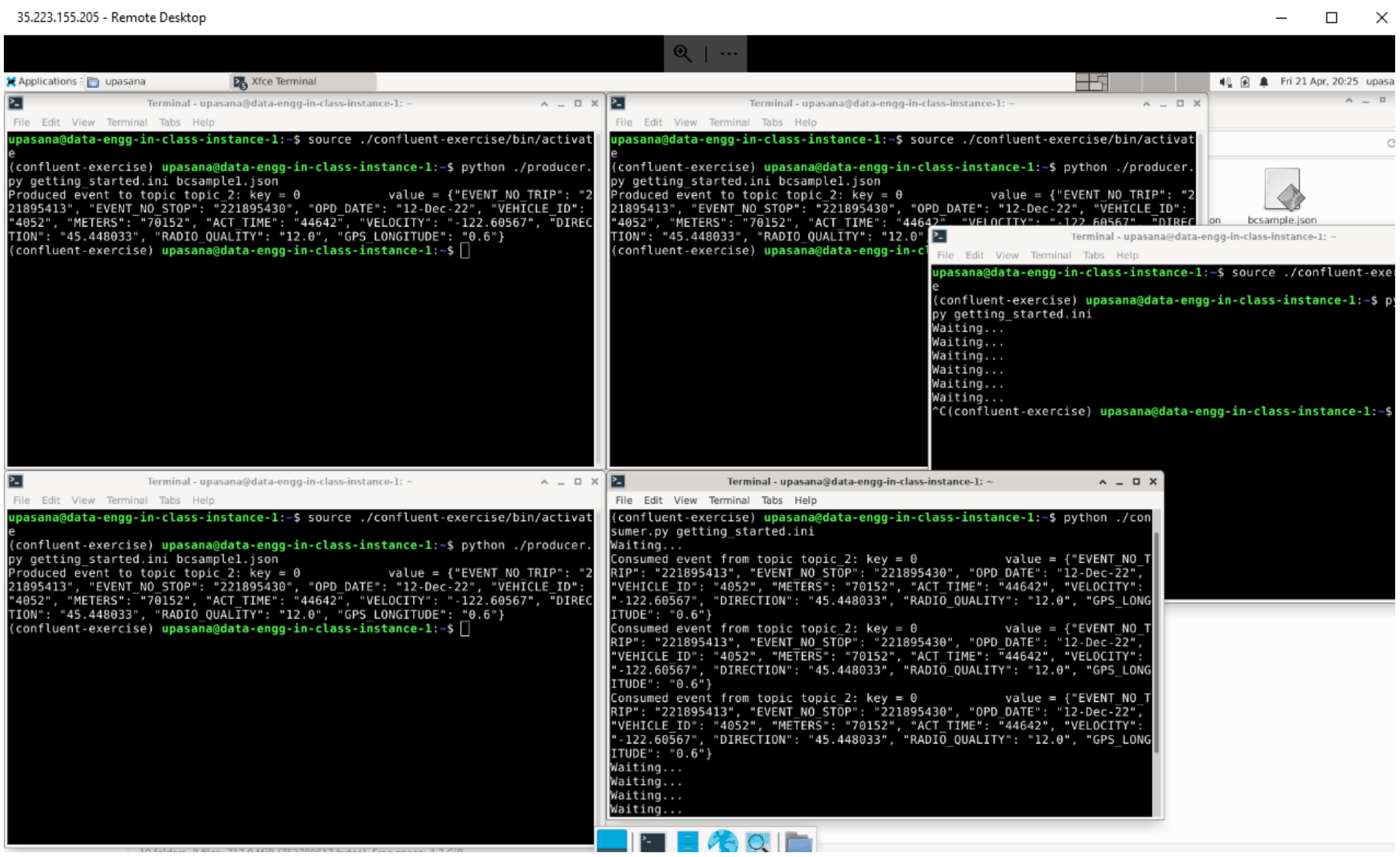
```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0"
(confluent-exercise) upasana@data-engg-in-c
py getting_started.ini bcsample.json
```

```
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "74038", "ACT_TIME": "45467", "VELOCITY": "-
RECTION": "45.437975", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE":
Consumed event from topic topic_2: key = 93            value = {"
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "73750", "ACT_TIME": "45447", "VELOCITY": "-
ECTION": "45.440475", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE":
Consumed event from topic topic_2: key = 88            value = {"
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "73414", "ACT_TIME": "45422", "VELOCITY": "-
RECTION": "45.443463", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE":
Consumed event from topic topic_2: key = 98            value = {"
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "74078", "ACT_TIME": "45472", "VELOCITY": "-
RECTION": "45.437642", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE":
Consumed event from topic topic_2: key = 94            value = {"
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "73826", "ACT_TIME": "45452", "VELOCITY": "-
RECTION": "45.4398", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "
Consumed event from topic topic_2: key = 89            value = {"
"221895413", "EVENT_NO_STOP": "221895444", "OPD_DATE": "12-Dec-2
: "4052", "METERS": "73472", "ACT_TIME": "45427", "VELOCITY": "-
ECTION": "45.442948", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE":
```

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample.json
```

```
(confluent-exercise) upasana@data-engg-in-c
sumer.py getting_started.ini
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
```

For 1 record, below is the image for understanding 3 producers produced the data concurrently and consumers were started concurrently the first consumer consumes 3 records from 3 diff producers and the second consumer is waiting for more data.


35.223.155.205 - Remote Desktop
Applications
upasana
Xfce Terminal
Fri 21 Apr, 20:25 upasa
Terminal - upasana@data-engg-in-class-instance-1: ~
File Edit View Terminal Tabs Help
bcsample.json

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ 
```

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0"
(confluent-exercise) upasana@data-engg-in-cl
```

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exe
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ py
py getting_started.ini
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
Waiting...
^C(confluent-exercise) upasana@data-engg-in-class-instance-1:~$
```

```
upasana@data-engg-in-class-instance-1:~$ source ./confluent-exercise/bin/activat
e
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./producer.
py getting_started.ini bcsample1.json
Produced event to topic topic_2: key = 0            value = {"EVENT_NO_TRIP": "2
21895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22", "VEHICLE_ID":
"4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY": "-122.60567", "DIREC
TION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONGITUDE": "0.6"}
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ 
```

```
(confluent-exercise) upasana@data-engg-in-class-instance-1:~$ python ./con
sumer.py getting_started.ini
Waiting...
Consumed event from topic topic_2: key = 0            value = {"EVENT_NO_T
RIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22",
"VEHICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY":
"-122.60567", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONG
ITUDE": "0.6"}
Consumed event from topic topic_2: key = 0            value = {"EVENT_NO_T
RIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22",
"VEHICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY":
"-122.60567", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONG
ITUDE": "0.6"}
Consumed event from topic topic_2: key = 0            value = {"EVENT_NO_T
RIP": "221895413", "EVENT_NO_STOP": "221895430", "OPD_DATE": "12-Dec-22",
"VEHICLE_ID": "4052", "METERS": "70152", "ACT_TIME": "44642", "VELOCITY":
"-122.60567", "DIRECTION": "45.448033", "RADIO_QUALITY": "12.0", "GPS_LONG
ITUDE": "0.6"}
Waiting...
Waiting...
Waiting...
Waiting...
```

F. Varying Keys

1. Clear all data from the topic.

So far you have kept the “key” value constant for each record sent on a topic. But keys can be very useful to choose specific records from a stream.

2. Update your producer code to choose a random number between 1 and 5 for each record’s key.
3. Modify your consumer to consume only records with a specific key (or subset of keys).
4. Attempt to consume records with a key that does not exist. E.g., consume records with key value of “100”. Describe the results.
5. Can you create a consumer that only consumes specific keys? If you run this consumer multiple times with varying keys then does it allow you to consume messages out of order while maintaining order within each key?

G. Producer Flush

The provided tutorial producer program calls “producer.flush()” at the very end, and presumably your new producer also calls producer.flush().

1. What does Producer.flush() do?

Producer.flush() is a method in message queuing and streaming systems that ensures all messages in the producer buffer are sent to the broker or server immediately. It blocks until all messages have been sent and acknowledged by the broker or server, ensuring data consistency, and reducing latency.

2. What happens if you do not call producer.flush()?

If you do not call producer.flush(), messages may remain in the producer buffer for a longer period of time before being sent to the broker or server. This can increase latency and may lead to data inconsistencies in certain scenarios.

3. What happens if you call producer.flush() after sending each record?

Calling producer.flush() after sending each record can increase network overhead and decrease overall throughput. This is because each call to producer.flush() will block until the message has been sent and acknowledged by the broker or server, which can be a time-consuming process.

4. What happens if you wait for 2 seconds after every 5th record send, and you call flush only after every 15 record sends, and you have a consumer running concurrently? Specifically, does the consumer receive each message immediately? only after a flush? Something else?

If you wait for 2 seconds after every 5th record send, and you call flush() only after every 15 record sends, and you have a consumer running concurrently, the consumer may receive messages immediately or after a flush, depending on the exact timing of the producer and consumer actions.

H. Consumer Groups

1. Create two consumer groups with one consumer program instance in each group.
2. Run the producer and have it produce all 1000 messages from your sample file.
3. Run each of the consumers and verify that each consumer consumes all of the messages.
4. Create a second consumer within one of the groups so that you now have three consumers total.
5. Rerun the producer and consumers. Verify that each consumer group consumes the full set of messages but that each consumer within a consumer group only consumes a portion of the messages sent to the topic.

I. Kafka Transactions

6. Create a new producer, similar to the previous producer, that uses transactions.
7. The producer should begin a transaction, send 4 records in the transactions, then wait for 2 seconds, then choose True/False randomly with equal probability. If True then finish the transaction successfully with a commit. If False is picked then cancel the transaction.
8. Create a new transaction-aware consumer. The consumer should consume the data. It should also use the Confluent/Kaka transaction API with a “read_committed” isolation level. (I can’t find evidence of other isolation levels).
9. Transaction across multiple topics. Create a second topic and modify your producer to send two records to the first topic and two records to the second topic before randomly committing or canceling the transaction. Modify the consumer to consume from the two queues. Verify that it only consumes committed data and not uncommitted or canceled data.