DataEng: Data Transport Activity

[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 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 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.

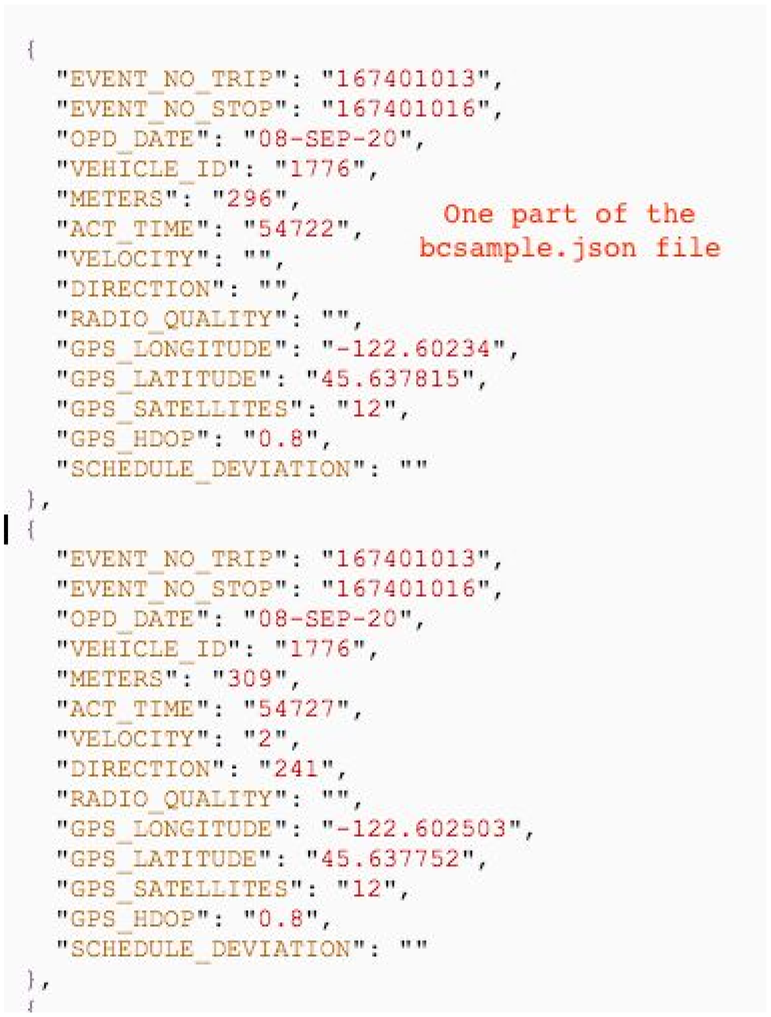
A. Initialization

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

    Done
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.

    Done

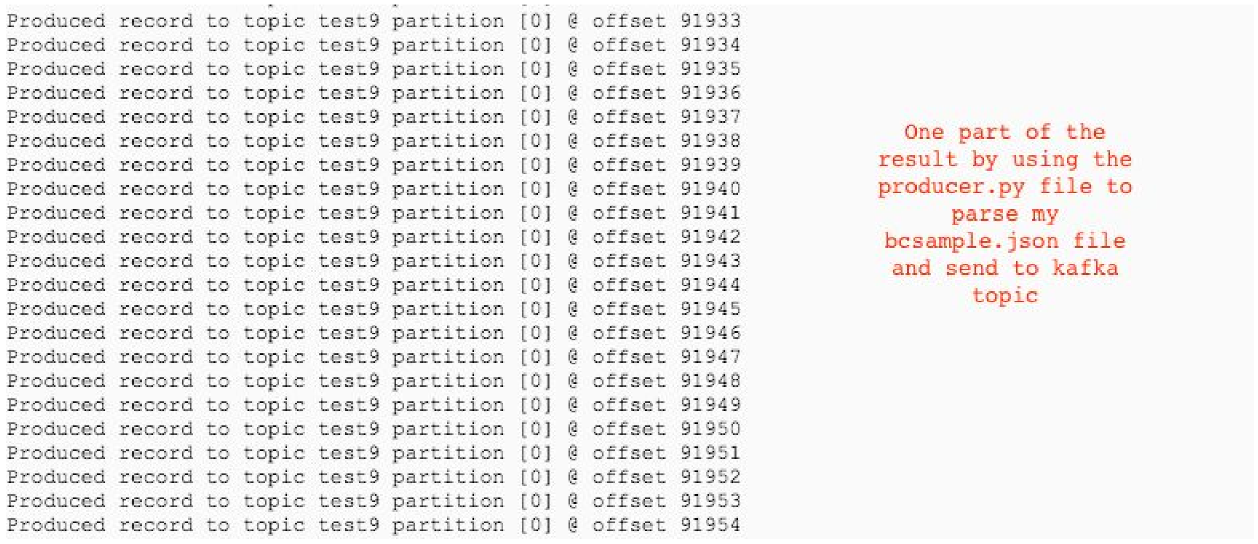
3. Create a sample breadcrumb data file (named bcsample.json) consisting of a sample of 1000 breadcrumb records. These can be any records because we will not be concerned with the actual contents of the breadcrumb records during this assignment.



One part of the bcsample.json file

```
{
  "EVENT_NO_TRIP": "167401013",
  "EVENT_NO_STOP": "167401016",
  "OPD_DATE": "08-SEP-20",
  "VEHICLE_ID": "1776",
  "METERS": "296",
  "ACT_TIME": "54722",
  "VELOCITY": "",
  "DIRECTION": "",
  "RADIO_QUALITY": "",
  "GPS_LONGITUDE": "-122.60234",
  "GPS_LATITUDE": "45.637815",
  "GPS_SATELLITES": "12",
  "GPS_HDOP": "0.8",
  "SCHEDULE_DEVIATION": ""
},
{
  "EVENT_NO_TRIP": "167401013",
  "EVENT_NO_STOP": "167401016",
  "OPD_DATE": "08-SEP-20",
  "VEHICLE_ID": "1776",
  "METERS": "309",
  "ACT_TIME": "54727",
  "VELOCITY": "2",
  "DIRECTION": "241",
  "RADIO_QUALITY": "",
  "GPS_LONGITUDE": "-122.602503",
  "GPS_LATITUDE": "45.637752",
  "GPS_SATELLITES": "12",
  "GPS_HDOP": "0.8",
  "SCHEDULE_DEVIATION": ""
},
{
```

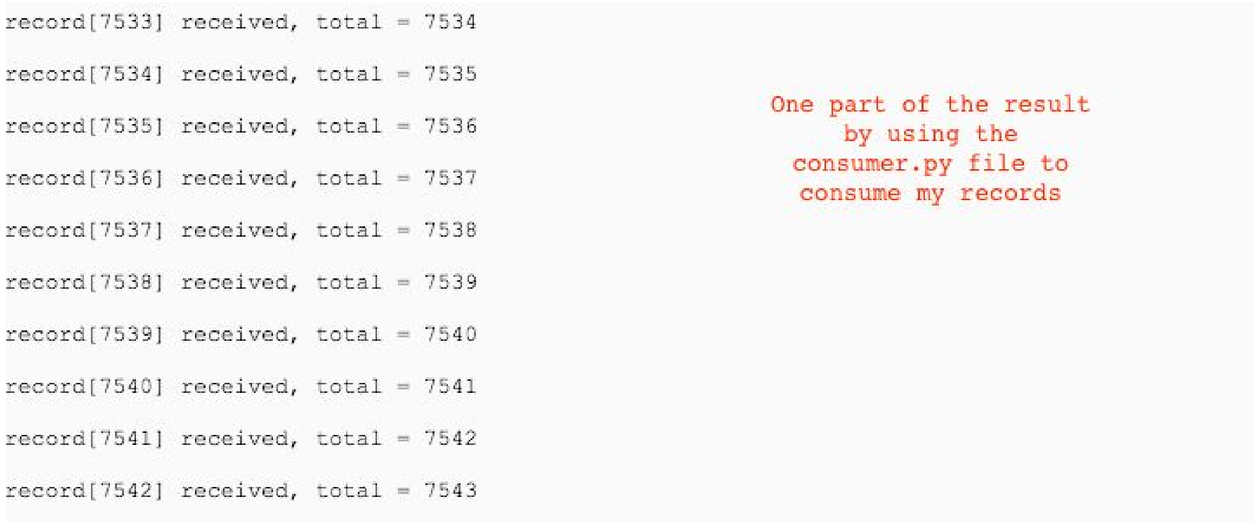
4. Update your producer to parse your sample.json file and send its contents, one record at a time, to the kafka topic.



```
Produced record to topic test9 partition [0] @ offset 91933
Produced record to topic test9 partition [0] @ offset 91934
Produced record to topic test9 partition [0] @ offset 91935
Produced record to topic test9 partition [0] @ offset 91936
Produced record to topic test9 partition [0] @ offset 91937
Produced record to topic test9 partition [0] @ offset 91938
Produced record to topic test9 partition [0] @ offset 91939
Produced record to topic test9 partition [0] @ offset 91940
Produced record to topic test9 partition [0] @ offset 91941
Produced record to topic test9 partition [0] @ offset 91942
Produced record to topic test9 partition [0] @ offset 91943
Produced record to topic test9 partition [0] @ offset 91944
Produced record to topic test9 partition [0] @ offset 91945
Produced record to topic test9 partition [0] @ offset 91946
Produced record to topic test9 partition [0] @ offset 91947
Produced record to topic test9 partition [0] @ offset 91948
Produced record to topic test9 partition [0] @ offset 91949
Produced record to topic test9 partition [0] @ offset 91950
Produced record to topic test9 partition [0] @ offset 91951
Produced record to topic test9 partition [0] @ offset 91952
Produced record to topic test9 partition [0] @ offset 91953
Produced record to topic test9 partition [0] @ offset 91954
```

One part of the result by using the producer.py file to parse my bcsample.json file and send to kafka topic

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

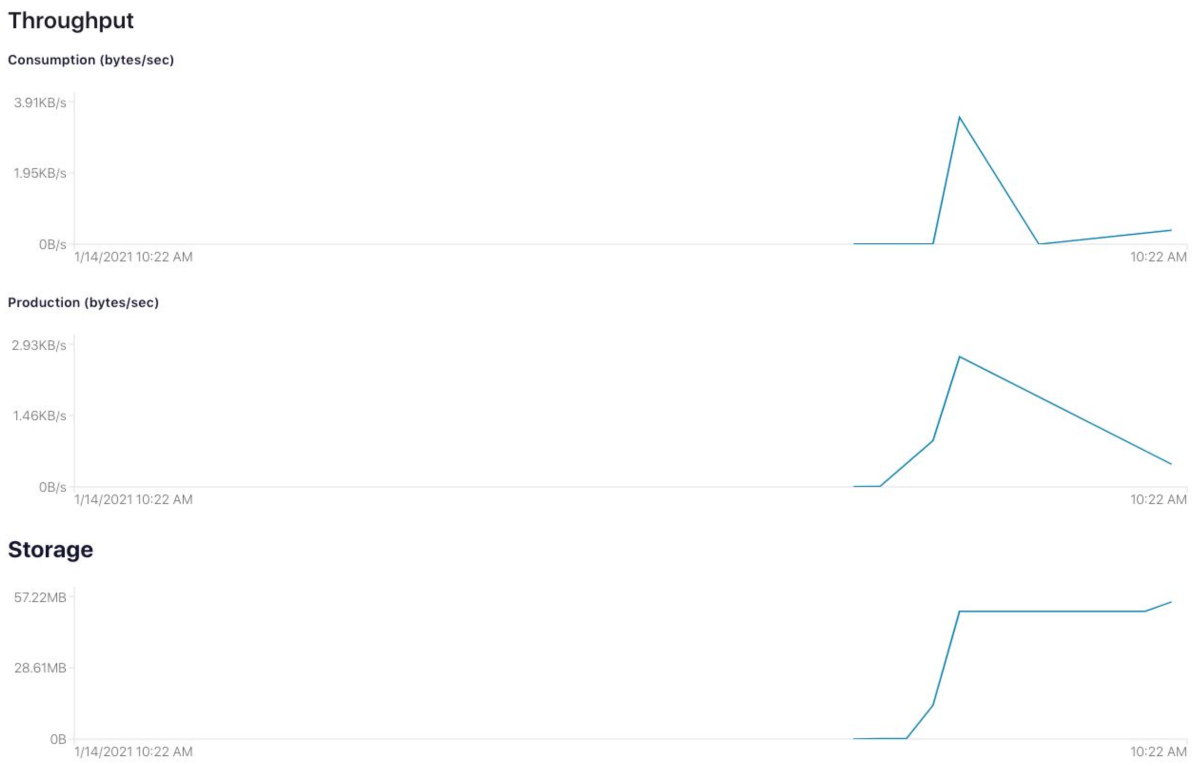


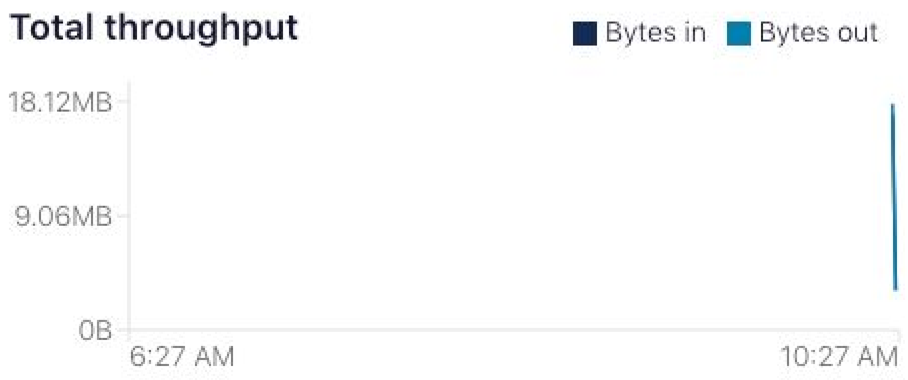
```
record[7533] received, total = 7534

record[7534] received, total = 7535

record[7535] received, total = 7536

record[7536] received, total = 7537

record[7537] received, total = 7538

record[7538] received, total = 7539

record[7539] received, total = 7540

record[7540] received, total = 7541

record[7541] received, total = 7542

record[7542] received, total = 7543
```

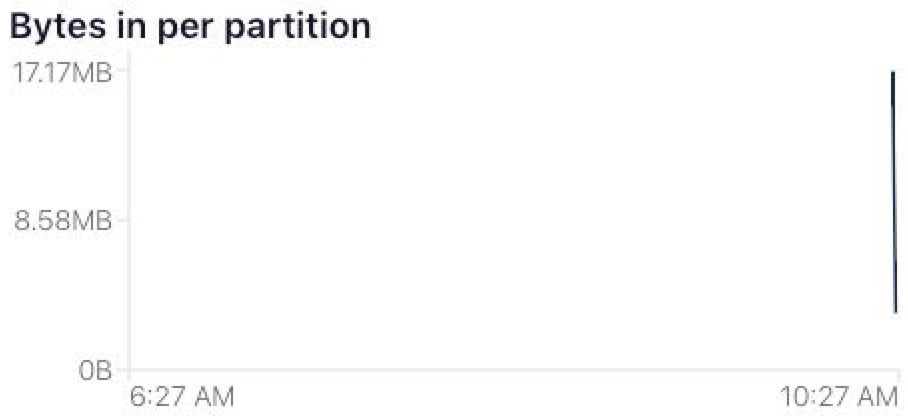
One part of the result by using the consumer.py file to consume my records

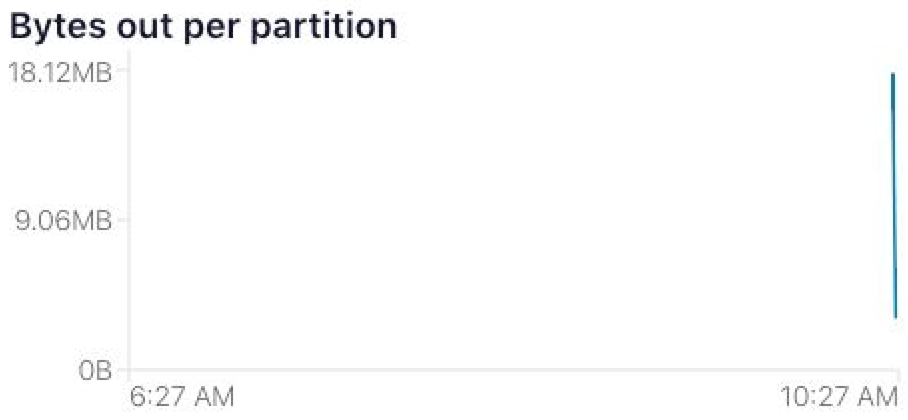
B. Kafka Monitoring

1. Find the Kafka monitoring console for your topic. Briefly describe its contents. Do the measured values seem reasonable to you?







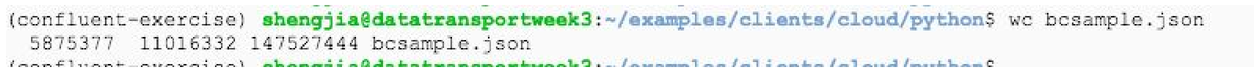


Based on the above two screenshots, the results seem reasonable to me. It shows the time and the value of storage, consumption, and production is changed in the graph.

2. Use this monitoring feature as you do each of the following exercises.
   Answer:
   Sure, that is helpful.

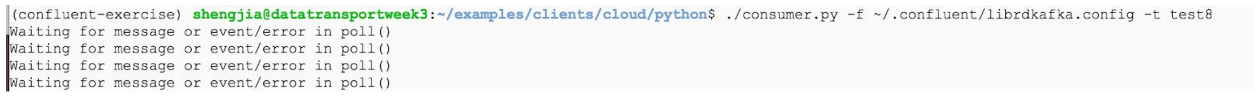
C. Kafka Storage

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) shengjia@datatransportweek3:~/examples/clients/cloud/python$ wc bcsample.json
  5875377  11016332 147527444 bcsample.json
```

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



```
(confluent-exercise) shengjia@datatransportweek3:~/examples/clients/cloud/python$ ./consumer.py -f ~/.confluent/librdkafka.config -t test8
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
```

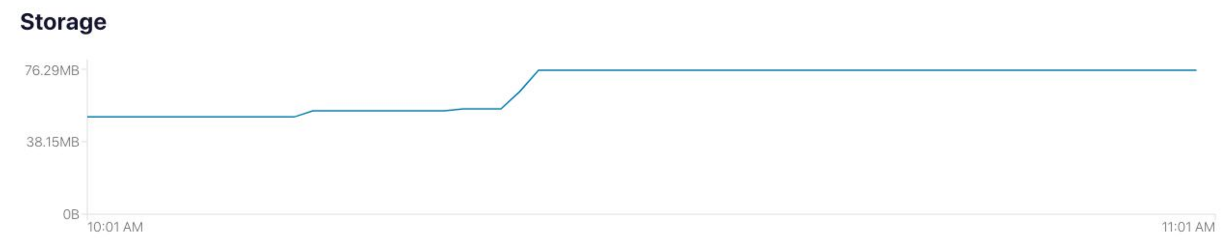
Run the consumer multiple times, it shows "Waiting for message or event/error in poll()". Need to run producer.py file and consumer.py file at the same time, then consumer.py can consume all the created messages from the producer.

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

Answer:
The data will be stored in the kafka topic after running the producer.py file.

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?

Answer:
Yes.



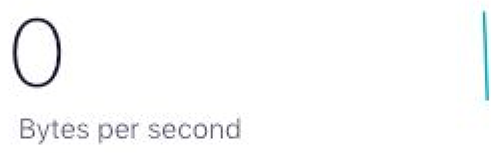
5. Create a "topic_clean.py" consumer that reads and discards all records for a given topic. This type of program can be very useful during debugging.

Answer:
Done

D. Multiple Producers

1. Clear all data from the topic
   Answer:
   Done

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.
   Answer:
   The two versions of producer can run concurrently and send all the data to same records. The consumer processes all the entries.



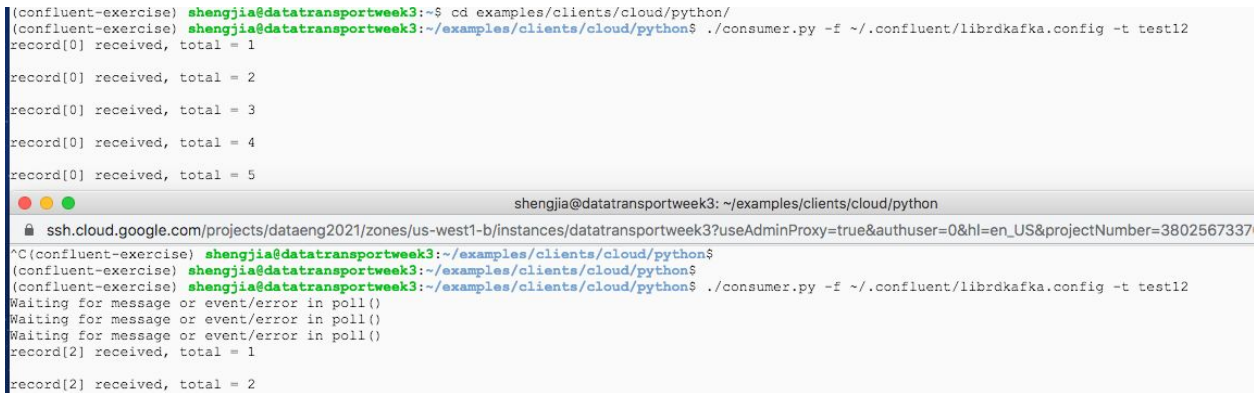
E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic
   Answer:
   Done

2. Update your Producer code to include a 250 msec sleep after each send of a message to the topic.
   Answer:
   Done

3. Run two or three concurrent producers and two concurrent consumers all at the same time.
   Answer:
   Done

4. Describe the results.
   Answer:
   I used two concurrent producers and two concurrent consumers. The consumers are consuming the data which is created by producers. And one of the consumers needs to wait for data, the other consumer consumes the data. Some part of the consumers result is as follows:



```
(confluent-exercise) shengjia@datatransportweek3:~$ cd examples/clients/cloud/python/
(confluent-exercise) shengjia@datatransportweek3:~/examples/clients/cloud/python$ ./consumer.py -f ~/.confluent/librdkafka.config -t test12
record[0] received, total = 1

record[0] received, total = 2

record[0] received, total = 3

record[0] received, total = 4

record[0] received, total = 5
```

shengjia@datatransportweek3: ~/examples/clients/cloud/python

ssh.cloud.google.com/projects/dataeng2021/zones/us-west1-b/instances/datatransportweek3?useAdminProxy=true&authuser=0&hl=en_US&projectNumber=3802567337

```
^C(confluent-exercise) shengjia@datatransportweek3:~/examples/clients/cloud/python$
(confluent-exercise) shengjia@datatransportweek3:~/examples/clients/cloud/python$
(confluent-exercise) shengjia@datatransportweek3:~/examples/clients/cloud/python$ ./consumer.py -f ~/.confluent/librdkafka.config -t test12
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
record[2] received, total = 1

record[2] received, total = 2
```

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.

Answer:

Done

----

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?
2. What happens if you do not call producer.flush()?
3. What happens if you call producer.flush() after sending each record?
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?

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 50 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.