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 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.

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
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.
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 records (you'll have to do some math!) from one of the bread crumb data files, and create a new file from that.

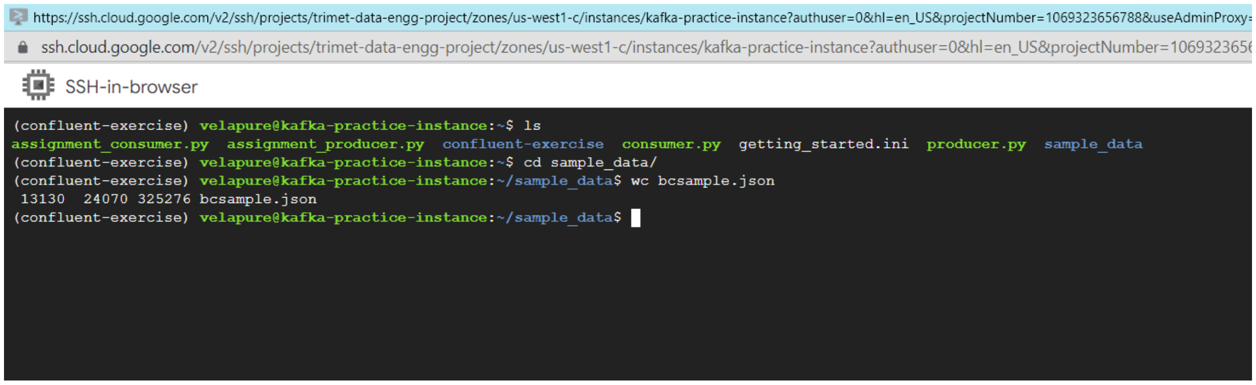
4. Update your producer to parse your bcsample.json file and send its contents, one record at a time, to the kafka topic.
5. Use your consumer.py program (from the tutorial) to consume your records.

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: I think cluster overview along with topic overview will be a better way to monitor the data flow on the topics as cluster overview displays the details such as throughput, data storage at a given time, number of topics in the cluster. To see the further details like the messages added to the topic along with their producer and consumer rates. We can use the topics→messages view to see the messages added to the topic although the messages added to the producer disappear on page refresh.
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.
   Answer: Below is the screenshot of the output of the command “wc bcsample.json”:



https://ssh.cloud.google.com/v2/ssh/projects/trimet-data-engg-project/zones/us-west1-c/instances/kafka-practice-instance?authuser=0&hl=en_US&projectNumber=1069323656788&useAdminProxy=

ssh.cloud.google.com/v2/ssh/projects/trimet-data-engg-project/zones/us-west1-c/instances/kafka-practice-instance?authuser=0&hl=en_US&projectNumber=1069323656

SSH-in-browser

```
(confluent-exercise) velapure@kafka-practice-instance:~$ ls
assignment_consumer.py  assignment_producer.py  confluent-exercise  consumer.py  getting_started.ini  producer.py  sample_data
(confluent-exercise) velapure@kafka-practice-instance:~$ cd sample_data/
(confluent-exercise) velapure@kafka-practice-instance:~/sample_data$ wc bcsample.json
 13130  24070 325276 bcsample.json
(confluent-exercise) velapure@kafka-practice-instance:~/sample_data$ 
```

2. What happens if you run your consumer multiple times while only running the producer once?
   Answer: The consumer reads the messages that were added to the topic by the producer when the producer was executed once. When the consumer is

executed or run for the second or consequent times then the consumer keeps waiting for any new messages that might get added to the producer.

3. Before the consumer runs, where might the data go, where might it be stored?
   Answer: Before the consumer runs, the data is stored in the topic in the Confluent Kafka cloud storage.

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. The cluster overview displays the amount of data stored in the cloud cluster for all the topics together in the 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.

D. Multiple Producers

1. Clear all data from the topic
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: When two versions of the producer are run concurrently then both the producers add data to the same topic. After their execution is finished, when the consumer is run once it reads all the messages added by both the producers and then goes into waiting when there are no more messages to be consumed.

E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic
2. Update your Producer code to include a 250 msec sleep after each send of a message to the topic.
3. Run two or three concurrent producers and two concurrent consumers all at the same time.
4. Describe the results.

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.

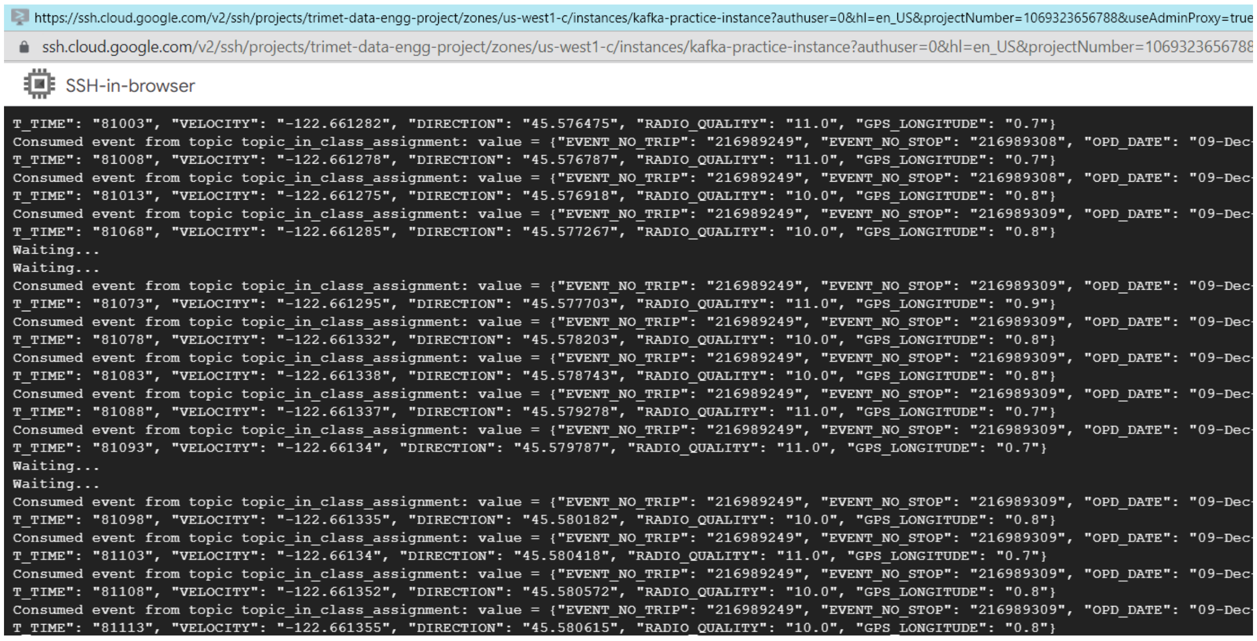
2. Update your producer code to choose a random number between 1 and 5 for each record’s key.
3. Modify your consumer to consume only records with a specific key (or subset of keys).
4. Attempt to consume records with a key that does not exist. E.g., consume records with key value of “100”. Describe 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?
   Answer: The producer.flush() ensures that the previously sent messages are delivered to the topic in kafka cloud and blocks the remaining messages until the previous ones are delivered successfully.
2. What happens if you do not call producer.flush()?
   Answer: If we do not call producer.flush() then all the messages are accumulated in a buffer which gets full after a certain number of messages are sent to the topic but the producer.flush() is not called explicitly. This results in a buffer full error message.
3. What happens if you call producer.flush() after sending each record?
   Answer: If we call producer.flush() after sending each record then the producer will make sure that each message is delivered to the kafka topic in the cloud before sending the next message. This may cause some delay.
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?
   Answer: When the producer is made to wait for 2 seconds after every 5th record is sent and when the producer.flush() is called after every 15th record is sent then the consumer does not receive each record immediately instead there is a delay and the consumer goes into ‘waiting’ state in between as the producer.flush() is called only after 15 records so records are accumulated on the producer side rather than sending them to the topic, thus consumer remains idle until the records are actually delivered to the topic in the cloud. Below is the

screenshot of the output seen after the consumer is run after making the corresponding changes in the producer:


https://ssh.cloud.google.com/v2/ssh/projects/trimet-data-engg-project/zones/us-west1-c/instances/kafka-practice-instance?authuser=0&hl=en_US&projectNumber=1069323656788&useAdminProxy=true
ssh.cloud.google.com/v2/ssh/projects/trimet-data-engg-project/zones/us-west1-c/instances/kafka-practice-instance?authuser=0&hl=en_US&projectNumber=1069323656788
SSH-in-browser

```
T_TIME": "81003", "VELOCITY": "-122.661282", "DIRECTION": "45.576475", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.7"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989308", "OPD_DATE": "09-Dec-
T_TIME": "81008", "VELOCITY": "-122.661278", "DIRECTION": "45.576787", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.7"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989308", "OPD_DATE": "09-Dec-
T_TIME": "81013", "VELOCITY": "-122.661275", "DIRECTION": "45.576918", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81068", "VELOCITY": "-122.661285", "DIRECTION": "45.577267", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Waiting...
Waiting...
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81073", "VELOCITY": "-122.661295", "DIRECTION": "45.577703", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.9"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81078", "VELOCITY": "-122.661332", "DIRECTION": "45.578203", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81083", "VELOCITY": "-122.661338", "DIRECTION": "45.578743", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81088", "VELOCITY": "-122.661337", "DIRECTION": "45.579278", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.7"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81093", "VELOCITY": "-122.66134", "DIRECTION": "45.579787", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.7"}
Waiting...
Waiting...
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81098", "VELOCITY": "-122.661335", "DIRECTION": "45.580182", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81103", "VELOCITY": "-122.66134", "DIRECTION": "45.580418", "RADIO_QUALITY": "11.0", "GPS_LONGITUDE": "0.7"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81108", "VELOCITY": "-122.661352", "DIRECTION": "45.580572", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
Consumed event from topic topic_in_class_assignment: value = {"EVENT_NO_TRIP": "216989249", "EVENT_NO_STOP": "216989309", "OPD_DATE": "09-Dec-
T_TIME": "81113", "VELOCITY": "-122.661355", "DIRECTION": "45.580615", "RADIO_QUALITY": "10.0", "GPS_LONGITUDE": "0.8"}
```

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.