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

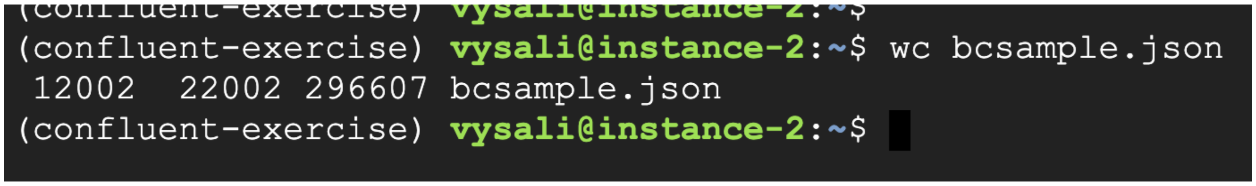
1. Tools for monitoring your Kafka topic. For example the cluster overview, or the topic overview, or the stream lineage. Which area do you think will be the best way to monitor data flow on your topic? Briefly describe its contents. Does it measure throughput, or total messages produced into Kafka and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

In my view the topic overview is the best way to monitor data flow in kafka topics. It displays key metrics such as message throughput and total messages produced and consumed, making it easy to monitor the overall health and performance of the topic. It also provides information on individual partitions, including message lag and partition size, which can help pinpoint any issues with the topic's performance.

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) vysali@instance-2:~$
(confluent-exercise) vysali@instance-2:~$ wc bcsample.json
 12002  22002 296607 bcsample.json
(confluent-exercise) vysali@instance-2:~$ 
```

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

During my test, I launched two Kafka consumers in the cloud shell and started the producer to send messages to the consumers. At first, the consumer that I opened most recently began consuming the messages, while the other consumer remained idle and waited for data. However, when I stopped the most recent consumer that was consuming messages, the other consumer began consuming the messages

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

Before the consumer runs, the data may be produced by the producer and sent to a Kafka broker, where it is temporarily stored in a topic partition before being consumed by the consumer.

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. I think the Confluent Control Center allows us to monitor Kafka clusters, topics, and partitions.

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.

The consumer consumed the data produced by 2 of the producers i.e., 2000 records

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.

I run 3 producer and 2 consumer codes concurrently. All the messages produced by the producer code are consumed by a single consumer, while the other one is waiting for messages.

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

Upon executing the consumer with a value(100) that did not exist, the consumer was waiting for messages and was not consuming anything that was being produced.

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?

Yes. we can create a consumer that only consumes specific keys. Yes the order is maintained.

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?

The Producer.flush() method is used to block until production of all the messages in the queue are completed. It also ensures that all the messages that are produced by the producer are actually sent to the Kafka broker before the program terminates.

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

Without calling producer.flush(), some or all of the messages produced may not be sent immediately.However, the producer will continue to send messages in the background until the batch buffer is full or a certain amount of time has elapsed. This can result in messages being delayed or not being sent at all

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

If flush() is called after sending each record, the producer will take a longer time blocking and waiting for acknowledgments, which can slow down the overall performance

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?

In this case the consumer receives each message as soon as it's written to the Kafka topic regardless of whether they have been flushed or not. Calling flush() doesn't affect message delivery to the consumer, it only ensures that all buffered messages are written to Kafka before the method call returns

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.
    a. Each consumer of the groups is consuming the 1000 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.

a. In Consumer group1 with 2 consumers , only one consumer consumed the 1000 messages
b. The Consumer group 2 with one consumer consumed all the 1000 messages.

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.

The consumer is only consuming the committed data produced to two different topics