DataEng: Data Transport Activity

[this lab activity references tutorials at confluence.com]

Varun Jaisundar Raju
PSU ID: 957710312
Email: raju@pdx.edu

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: In-class Activity Submission Form

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.

4. Update your producer to parse your sample.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. Find the Kafka monitoring console for your topic. Briefly describe its contents. Do the measured values seem reasonable to you?
   Yes, the measured values seem reasonable. In my case the producer is pushing the messages into the kafka stream at the rate of 436 bytes per second and the consumer is consuming the pushed message from the queues at the rate of 533 bytes per second.
   The rate at which the producer pushes the message will always be slightly lower than the consumption rate of the consumer
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.
   ```
    465   871 11401 sample.json
   ```
2. What happens if you run your consumer multiple times while only running the producer once?
   The throughput out the consumer is slightly reduced as the bandwidth is being consumed by multiple consumers at the same time. Example producer is producing the data at the rate of 460 bytes per second while 2 consumers which are running simultaneously are consuming at a rate of 503 bytes per second. If we stop one consumer and run only one consumer there is a sudden drop in throughput and eventually the consumer throughput increases and starts consuming at a rate of 533 bytes per second which is more than previously consumed by the 2 consumers
3. Before the consumer runs, where might the data go, where might it be stored?

Data is stored in the kafka topic before the data is being consumed by the consumer. Apache Kafka's most fundamental unit of organization is the topic, which is something like a table in a relational database

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 it is possible to view the maximum amount of message bytes that the kafka confluent has allocated for a given topic. Yes, Confluent monitoring tools help to view the data allocated with the help of its configuration window.

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.
   Initially when two producers run simultaneously each producer pushes the messages into the queue at the rate of 486 bytes per second in total the production rate of messages being pushed into the queue happens at the rate of 986 bytes per second. Now we start the consumer and it starts to consume the messages at a greater rate than it used to in the case of single producer - consumer scenario, Consumption rate of consumer is 5303 bytes per second and there is a sudden drop in the consumption rate. As the consumer consumes the messages faster than the rate at which the producer’s pushes the messages into the queue

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.
   Two Producers are run at the same time with each producer pushing messages to the topic with a delay of 250 msec. And now we run two consumers at the same time. Initially the two consumer’s would not receive any messages from the queue; it would be in the waiting state. Because the consumer’s starts to consume the message before the producer has actually produced it. The consumer starts to consume the message after the messages arrive inside the message queue. There is an initial delay when the delay is added to the producer code.

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
   The Consumer does not print anything because that key value does not exist in the message queue.
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?
   No, it is not possible for the consumer to consume records with a specific key. No producer pushes the messages into the queue in random order and consumers start to consume the messages in the same order in which they arrived. Therefore it is not possible to maintain the order of messages

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 used to push all the messages the producer produced into the message queue
2. What happens if you do not call producer.flush()? new_producer.py
   If we do not call the producer.flush() all the messages that the producer produced will not be pushed into the message queue for the consumer to consume
3. What happens if you call producer.flush() after sending each record?(modified_producer.py)
   Partition will be created immediately inside the topic after each record is pushed before the producer pushes another 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?
No, the consumer does not receive the message until the producer flush has been called . Therefore the consumer starts to consume the message from the message queue when it reaches the 15th record then it calls the producer.flush() method to completely push all the messages it has produced so far into the message queue for the consumer to consume it.

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.
   consumer_group1-1.py
   consumer_group2-1.py
   consumer_group2-2.py

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.

topic_consumer.py

transactional_consumer.py

transactional_producer.py

topic_transactional_producer.py