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.

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. 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 the cluster overview, we have the link to the metrics page where we can monitor the cluster, topic for things like sent/received bytes, sent/received count.
   In the dashboard, we can see the throughput and storage details.
   In the monitoring console it shows the bytes being produced and bytes being consumed.
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.
   0 28000 338035 bcsample.json
2. What happens if you run your consumer multiple times while only running the producer once?
   Messages are consumed by only one consumer even when we have multiple consumers
3. Before the consumer runs, where might the data go, where might it be stored?
   Kafka stores it locally and to make data persistent even after a crash, it is stored in disk
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?
   Yeah, the metrics page shows the storage metrics per topic or per 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.
   Done

D. Multiple Producers

1. Clear all data from the topic
   Done usng topic_clean.py
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.

It was similar to sending 2000 records together. The total consumed message were at 2000.

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.
   With 3 producers and 2 consumers, only 1 of the consumer consumed all 3000 messages. The other consumer was waiting for other messages. This is because all of the messages have the same key. If we vary the key, we can use multiple consumers to consume 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
   Rest of the messages get discarded. So basically its acting like the topic_clean.py script
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 like that. Yes, the order within each key will be 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?

   As per the documentation, flush will wait for all messages in the Producer queue to be delivered
2. What happens if you do not call producer.flush()?

   Publishing a message in async, it buffers message into a group and then sends them. Since flush will make sure to wait for all messages to be sent first and then only exit the program, but if there are some messages which are not sent, not flushing will send incorrect number of messages.
3. What happens if you call producer.flush() after sending each record?

   This will make each message delivery synchronous instead of being a asynchronous one. Flush will make it wait before proceeding ahead to send the next message
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 messages are received in the order, but after every 2 seconds or so. And they aren't received only after a flush. This might be because the buffer size is small and sends the message before the 15 messages mark.

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.

   Each consumer consumes 1000 messages.
4. Create a second consumer within one of the groups so that you now have three consumers total.

   The consumer group with 2 consumers - only one of them consumed the 1000 messages

   The consumer group with 1 consumer - consumed all of the 1000 messages
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.

   The consumer group with 2 consumers - only one of them consumed the 1000 messages

   The consumer group with 1 consumer - consumed all of 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.