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
    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.

    Done

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

   Done

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

   Done

   Note: I had to create a second topic because I accidentally put some data in the wrong format in my first_topic

B. Kafka Monitoring

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

   I see two boxes: one that shows production data and another that shows production data. Both have a number by them (which is perhaps the number of bytes being produced and consumed per second) and a graph of the history, I think. Right now, they are showing 0 as the current value in both boxes, which makes sense since I'm not sending or consuming any data at the moment, and the graph is mostly empty and shows a dramatic spike at the beginning and a flat line afterwards, which also makes sense since I started the produce and consume 1000 from a file part of the lab on a different day than I started the first one. There are also some other tabs (messages, schema, and configuration tabs), and the messages tab seems like it displays messages that have arrived (been produced) since the page has been opened.

2. Use this monitoring feature as you do each of the following exercises.

   Alright

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.

   ```
   1002  4002 20003 bcsample.json
   ```

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

The first time the consumer is run, it consumes all the data made by the producer. The following runs of consumer don't have any data left to consume, so they just sit there and wait for more data to be produced.

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

   I would guess that it is stored in Kafka topic.

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, there is. The monitoring tools (in the cluster overview section) do help you with this by providing a nice looking, interactive graph. It looks like they let you store up to 5TB of message data for all of your topics.

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

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 processes all the entries generated by both producers--so it processes 2000 entries. Note: the way I had my producers set up, it was only doing one flush at the time (instead of doing 1 flush for each record sent--this might not have been right, but I presume the results would have been similar if they had been sent one at a time--the consumer would process the messages from both producers).

E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic

   Done

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

   Done

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

   Done

4. Describe the results.

   The consumers are eating the data as the producers are generating it, though interestingly, the two consumer scripts seem to be in sync with each other (that is, one the total counts they are keeping track of seem to be going up in sync)

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?
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.