

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. One way to do this is by using the linux command “head” to get the first n lines from one of the bread crumb data files, and create new file from that.
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?*

The topic overview shows a chronological graph of production and consumption, as well as storage. The cluster overview only shows the total number, so topic overview would be more useful. You can also view metrics to see how many records have been produced/consumed among other things. The numbers seem to match what I produced and 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.*

Result: 0 28000 338588 bcsample.json

It indicates that the sample file has 28,000 words (about 28 average per record) and has a size of 338KB. This seems like a reasonable size.

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

The consumer will be polling for messages every second but not consume any because it has already consumed everything.

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

The data is stored in the Kafka cloud server waiting to be consumed.

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?*

The topic overview shows how much is currently being stored and how it changed over time.

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 2000 messages.

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.*

One of the consumers consumes the messages as the two producers produce them. The other consumer just polls for data, seemingly not able to consume anything.

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*

I could make it only record messages with a certain key but it appears that it still technically “consumes” everything.

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?*

As per question 4, it appears you have to consume everything. The key of the message determines which partition it will be put into, so in a sense you could only consume certain groups of keys by “assigning” the consumer to a particular partition. I could not get this to work with the `confluent_kafka` package but I would expect that you can consume messages out of order and they would not be in 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?*

`flush()` waits for all messages in the queue to be delivered.

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

The messages were apparently not delivered to the broker since the consumer did not consume anything afterward.

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

The console indicates that it produced the record the topic after every breadcrumb.

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?*

The consumer only receives messages after the flush. While the producer is sleeping, the consumer is polling for new messages but not getting any.

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 1 consumed all of the messages, consumer 2 and 3 consumed a total of 1000 between them.

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/Kafka 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.*

Running 4 transactions, 3 of which were aborted, only 4 transactions were consumed. Interestingly, it does not appear that they were consumed in the order they were produced.