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.

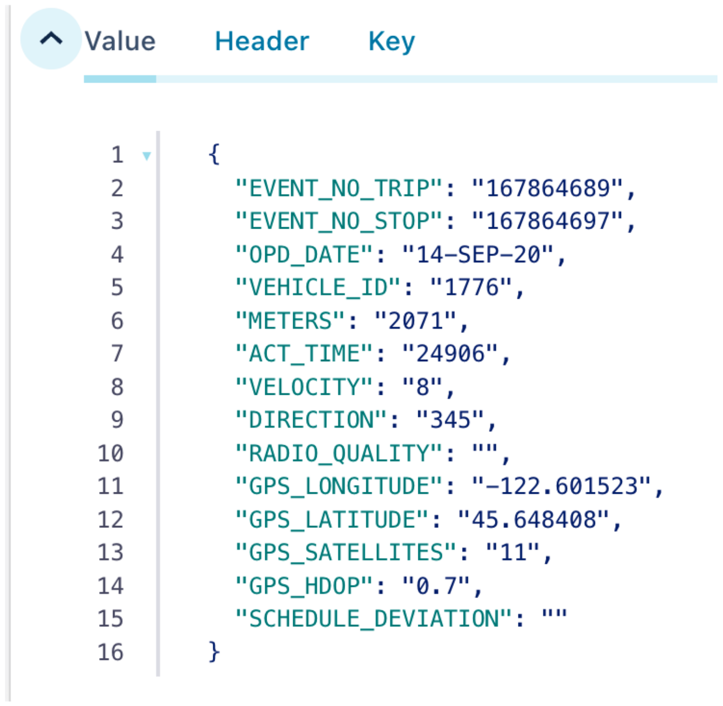
Submit: In-class Activity Submission Form

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

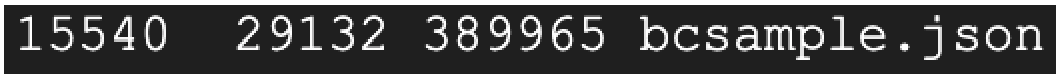
1. Find the Kafka monitoring console for your topic. Briefly describe its contents. Do the measured values seem reasonable to you?
   - We extracted each object from the bcsample.json file to a single message in Kafka as the attached screenshot below. It does seem reasonable to me.



```
{
  "EVENT_NO_TRIP": "167864689",
  "EVENT_NO_STOP": "167864697",
  "OPD_DATE": "14-SEP-20",
  "VEHICLE_ID": "1776",
  "METERS": "2071",
  "ACT_TIME": "24906",
  "VELOCITY": "8",
  "DIRECTION": "345",
  "RADIO_QUALITY": "",
  "GPS_LONGITUDE": "-122.601523",
  "GPS_LATITUDE": "45.648408",
  "GPS_SATELLITES": "11",
  "GPS_HDOP": "0.7",
  "SCHEDULE_DEVIATION": ""
}
```

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.
   - 

```
15540  29132 389965 bcsample.json
```

2. What happens if you run your consumer multiple times while only running the producer once?
   - It will output “Waiting for message or event/error in poll()”, since no message in the poll after the first run on the consumer.
3. Before the consumer runs, where might the data go, where might it be stored?
   - The data goes and store in Kafka's topic after we run the producer.

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! If we go to the Topic overview, we can see the number of bytes/sec of production and consumption. Also, if we go to the Data Integration, then go to the corresponding client ID, we can see the number of bytes that we storing for our topic.
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 this part by copying a consumer.py to topic_clean.py and resetting record_key and record_value to zero instead of loading them into a JSON file.

D. Multiple Producers

1. Clear all data from the topic
    - Done by run 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.
    - Data will be duplicated in the output JSON file.

E. Multiple Concurrent Producers and Consumers

1. Clear all data from the topic
    - Done by run topic_clean.py
2. Update your Producer code to include a 250 msec sleep after each send of a message to the topic.
    - Import time, time.sleep(0.25)
3. Run two or three concurrent producers and two concurrent consumers all at the same time.
4. Describe the results.
    - I ran 2 producers and consumers at the same time. One of the consumers get into waiting status while the one was still running. It took around 4 minutes to run the producer process, and when both producers' process ends, the other consumers get into waiting status in the poll as well.

F. Varying Keys

1. Clear all data from the topic
    - Done by run topic_clean.py

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
    - Nothing will record in the file with a key that does not exist.
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, I can create a consumer that only consumes specific keys. And it will not contains any data outside of the 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?
    - “Wait for all messages in the Producer queue to be delivered.” from https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html#confluent_kafka.Producer.flush
2. What happens if you do not call producer.flush()?
    - 0 messages were produced to topic.
3. What happens if you call producer.flush() after sending each record?
    - It takes longer time to process.
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?
    - Waited for 2 seconds after 5th record send, and flush after 15 record sends. We need to wait around 6 seconds to flush. The consumer will received 5 records immediately, then wait for 2 seconds to receive another 5 records. That would be before flush i believe.

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.