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.

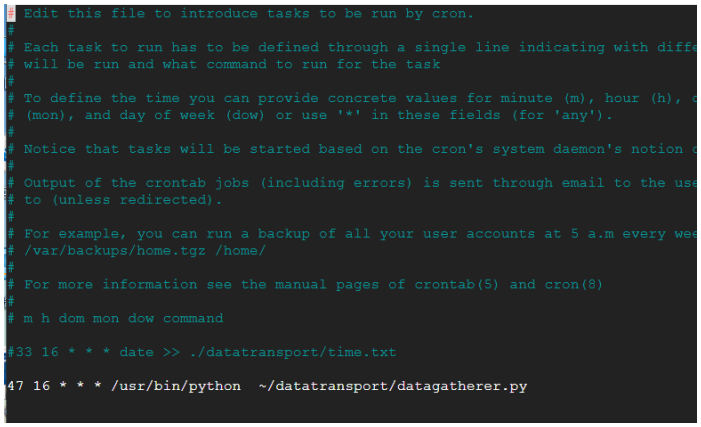
The goal for this week is to gain experience and knowledge of using a streaming data transport system (Kafka). Complete as many of the following exercises as you can. Proceed at a pace that allows you to learn and understand the use of Kafka with python.

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.



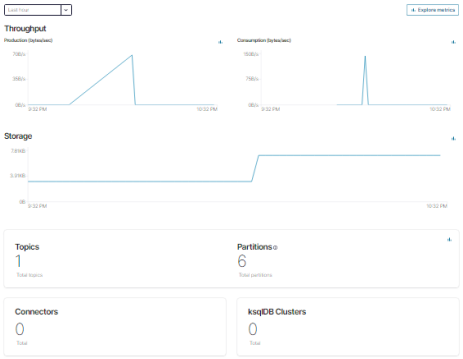
```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line indicating with diff
# will be run and what command to run for the task
#
# To define the time you can provide concrete values for minute (m), hour (h),
# (mon), and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system daemon's notion
#
# Output of the crontab jobs (including errors) is sent through email to the us
# to (unless redirected).
#
# For example, you can run a backup of all your user accounts at 5 a.m every we
# /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow command

#33 16 * * * date >> ./datatransport/time.txt

47 16 * * * /usr/bin/python  ~/datatransport/datagatherer.py
```

```
# 04/14/2022

import requests
import urllib3
import json

##urllib3 template!
# http = urllib3.PoolManager()
# r = http.request('GET', 'http://www.hubertiming.com/results/2017GPTR10K')
# print(r.status)

url = requests.get("http://www.psudataeng.com:8000/getBreadCrumbData")

text = url.text
data = json.loads(text)

with open("days.json",'w') as code:
    code.write(text)

# # TO Get First_Data
# user = data[0]
# print(user['OPD_DATE'])
# address = user['ACT_TIME']
# print(address)
```

After setting cron and finishing this file, I could get the days.json file to make sure I can download the .json file from the url.

B. Kafka Monitoring

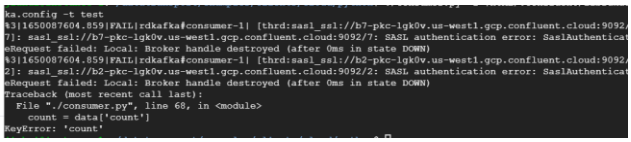
1. Tools for monitoring your Kafka topic. For example the cluster overview, or the topic overview, or the stream lineage. Which area do you think will be the best way to monitor data flow on your topic? Briefly describe its contents. Does it measure throughput, or total messages produced into Kafka and consumed out of Kafka? Do the measured values seem reasonable to you?
2. Use this monitoring feature as you do each of the following exercises.



I prefer to use Cluster setting rather than Streamlineage. First of all, as shown in the screenshot, these data can be seen intuitively and accurate data can be obtained according to the time. In Streamlineage, although information can be observed by mouse, the overall visualization is not as good as cluster setting.

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.
2. What happens if you run your consumer multiple times while only running the producer once?
3. Before the consumer runs, where might the data go, where might it be stored?
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?
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.



The previous tests passed normally, but after running and modifying producer.py, an error occurred.

I think more will probably be stored between topics. Therefore, once we upload the data, it should be observed by some metric of the topic. Since these data can be observed, they may be stored in topics.

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.

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.

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.