

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

VM instance created:

<input checked="" type="checkbox"/> Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect
<input checked="" type="checkbox"/>  instance-1	us-west1-b			10.138.0.2 (nic0)	35.233.132.86	SSH ▾ ⋮

2. Follow the Kafka tutorial from project assignment #1
 - a. Create a separate topic for this in-class activity

Done:

Topics

☒ Hide internal topics

Topic name	Partitions	Production (last min)	Consumption (last min)
first-topic	6	--	--
test1	1	--	--

- 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.

Done, screenshot showing consumer consuming data from producer via the topic:



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.

Created and added to repo.

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

1000 records sent to topic test2:

```
Produced record to topic test2 partition [0] @ offset 2970
Produced record to topic test2 partition [0] @ offset 2971
Produced record to topic test2 partition [0] @ offset 2972
Produced record to topic test2 partition [0] @ offset 2973
Produced record to topic test2 partition [0] @ offset 2974
Produced record to topic test2 partition [0] @ offset 2975
Produced record to topic test2 partition [0] @ offset 2976
Produced record to topic test2 partition [0] @ offset 2977
Produced record to topic test2 partition [0] @ offset 2978
Produced record to topic test2 partition [0] @ offset 2979
Produced record to topic test2 partition [0] @ offset 2980
Produced record to topic test2 partition [0] @ offset 2981
Produced record to topic test2 partition [0] @ offset 2982
Produced record to topic test2 partition [0] @ offset 2983
Produced record to topic test2 partition [0] @ offset 2984
Produced record to topic test2 partition [0] @ offset 2985
Produced record to topic test2 partition [0] @ offset 2986
Produced record to topic test2 partition [0] @ offset 2987
Produced record to topic test2 partition [0] @ offset 2988
Produced record to topic test2 partition [0] @ offset 2989
Produced record to topic test2 partition [0] @ offset 2990
Produced record to topic test2 partition [0] @ offset 2991
Produced record to topic test2 partition [0] @ offset 2992
Produced record to topic test2 partition [0] @ offset 2993
Produced record to topic test2 partition [0] @ offset 2994
Produced record to topic test2 partition [0] @ offset 2995
Produced record to topic test2 partition [0] @ offset 2996
Produced record to topic test2 partition [0] @ offset 2997
Produced record to topic test2 partition [0] @ offset 2998
Produced record to topic test2 partition [0] @ offset 2999
1000 messages were produced to topic test2!
```

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

1000 topics consumed from test2 topic:

[illegible]

B. Kafka Monitoring

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

The topic console has the following 4 tabs: Overview, Messages, Schema, Configuration. The measured values in terms of production and consumption graph in bytes per second seem reasonable to me based on my 1000 json records.

I can also see my messages on there:

The screenshot shows the Kafka Monitoring console with the 'Messages' tab selected. The interface includes a sidebar with 'Producers' (0 bytes in/sec) and 'Consumers' (0 bytes out/sec) sections. The 'Message fields' section lists various fields like topic, partition, offset, timestamp, timestampType, headers, key, and value. The main area displays a table of messages with columns: key, value.id, value.first_name, value.last_name, value.email, value.gender, and value.ip_address. A search bar and a 'Jump to offset' dropdown are at the top. A 'Produce a new message to this topic' button is also visible.

key	value.id	value.first_name	value.last_name	value.email	value.gender	value.ip_address
key-72	73	Alysa	Seldner	aseldner20@usat...	Female	196.86.177.142
key-71	72	Ashley	Rasmus	arasmus1z@slate...	Female	235.195.84.217
key-70	71	Rachele	Furze	rfurze1y@prweb....	Female	113.166.141.161
key-69	70	Cheslie	Rentall	crentall1x@imgur...	Female	86.70.37.244
key-68	69	Pier	Pincked	ppincked1w@goo...	Female	131.245.208.20
key-67	68	Aile	Ferrer	aferrer1v@examp...	Female	63.46.80.39
key-66	67	Hermon	Jeffcoat	hjeffcoat1u@cbs...	Male	106.213.11.250
key-65	66	Karylin	Boundley	kboundley1t@is.gd	Female	140.247.19.157
key-64	65	Barrie	Geffen	bgeffen1s@times...	Female	145.87.74.107
key-63	64	Shayla	Bacup	sbacup1r@army.mil	Female	169.152.118.244
key-62	63	Gaylord	Horder	ghorder1q@smug...	Male	87.236.203.180

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.

```
(kafka_client-7zvVoRsM) root@L26861
8001 14010 161917 bcsample.json
```

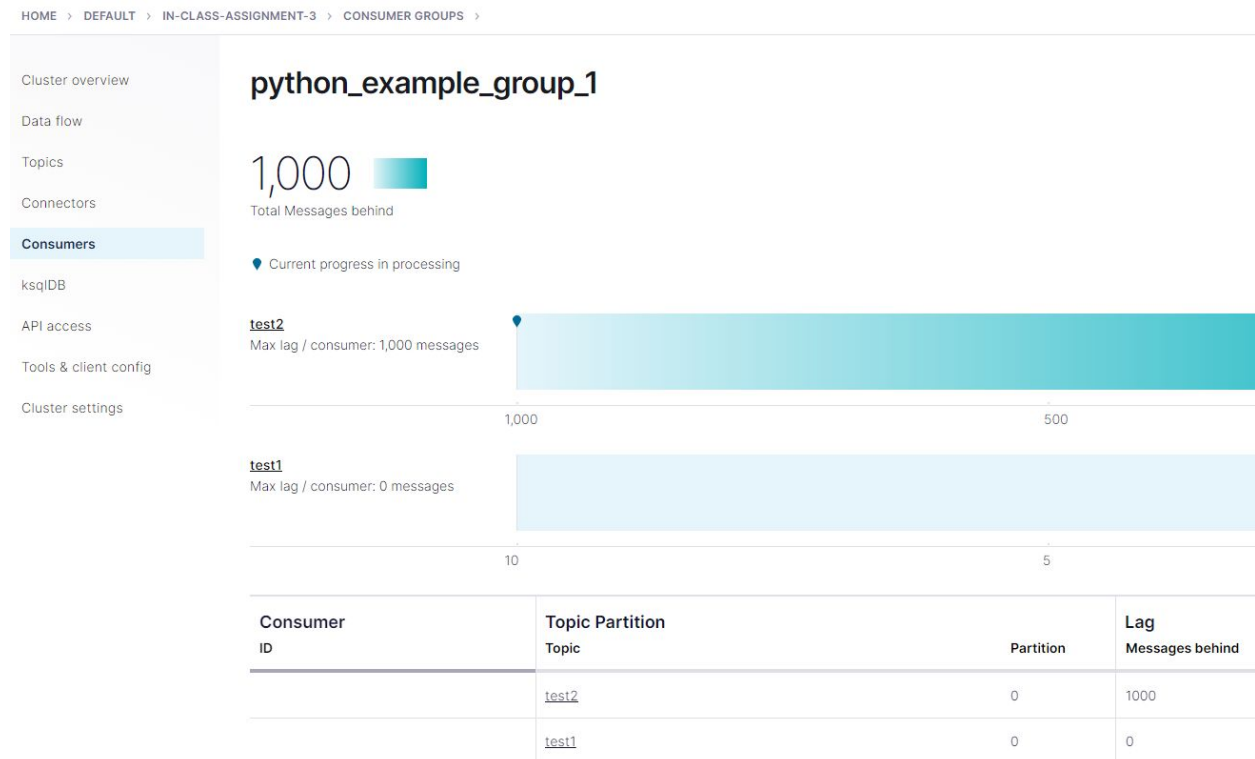
2. What happens if you run your consumer multiple times while only running the producer once?

If I run two consumers at the same time and then run the producer, then only the consumer that ran first gets messages from the topic. The other one keeps polling and does not get any messages.

3. Before the consumer runs, where might the data go, where might it be stored? The data gets sent to Kafka and it stores it.

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, we can see that consumers are 1000 messages behind



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.

Added to my github repo.

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.

Each of the two producers sent 1000 messages each for a total of 2000 messages. The consumer consumed all 2000 message:

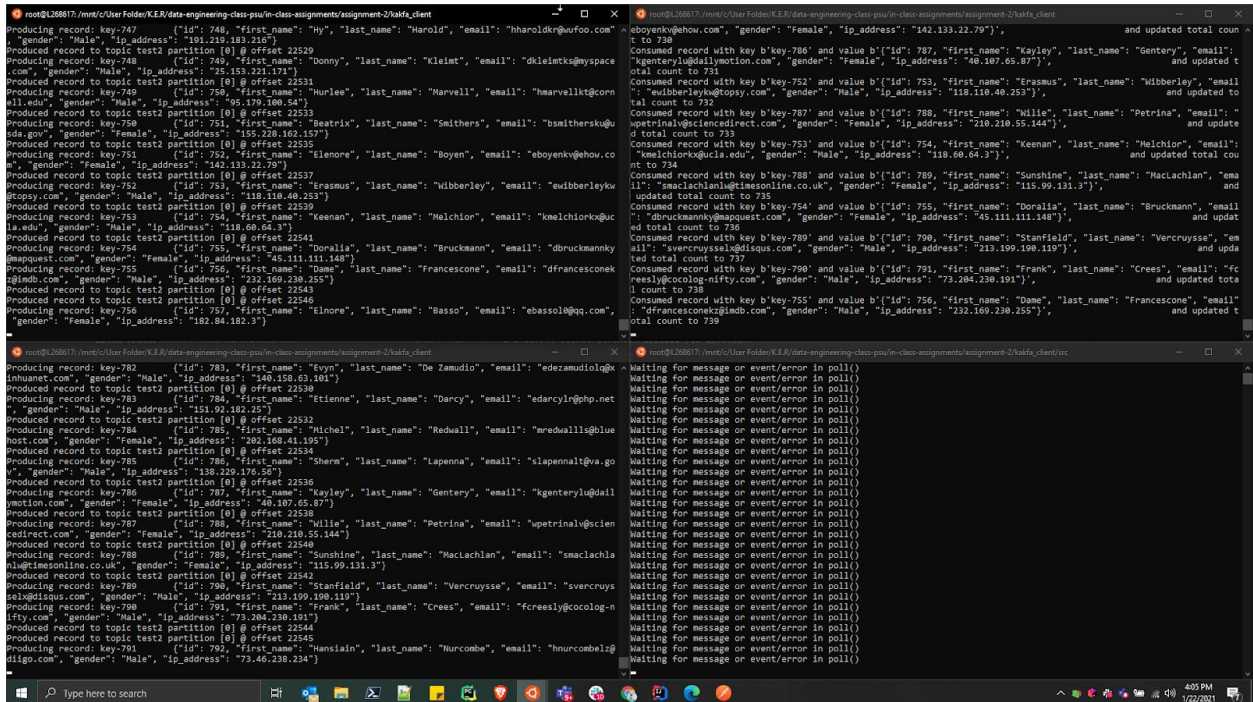
```
Consumed record with key b'key-999' and value b'{"id": 1000, "first_name": "Dionysus", "last_name": "Rapson", "email": "drapsonrrn@theforest.net", "gender": "Male", "ip_address": "175.72.229.19"}', and updated total count to 2000
```

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.

Both producers are able to send messages to the topic. However, only the consumer who started first is able to consume results.

Logs showing two producers on left and two consumers on right:



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.

Created producer-3.py and added to github

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/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.