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
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?

   The topic monitoring console contains the time and the throughput (bytes produced /sec). In the above updating, the max throughput is 564 bytes produced per second. This is really reasonable for me. Also, the consumer log is available, but I do not use it yet.

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.

   The output of the "wc" command is "0 2000 13780 bcsample.json". That means this file has one line, 2000 words, and 13780 characters.

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

   The first time the consumer runs normally, and all messages produced were consumed. However, after this time, those messages that only produced one time was marked as consumed, which means the end offset mark moved to the last message. Thus the flowering consumer cannot read any messages and continues waiting.

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

   The data will be stored in the Kafka server before and after those are consumed. After seven days, the server will delete them.

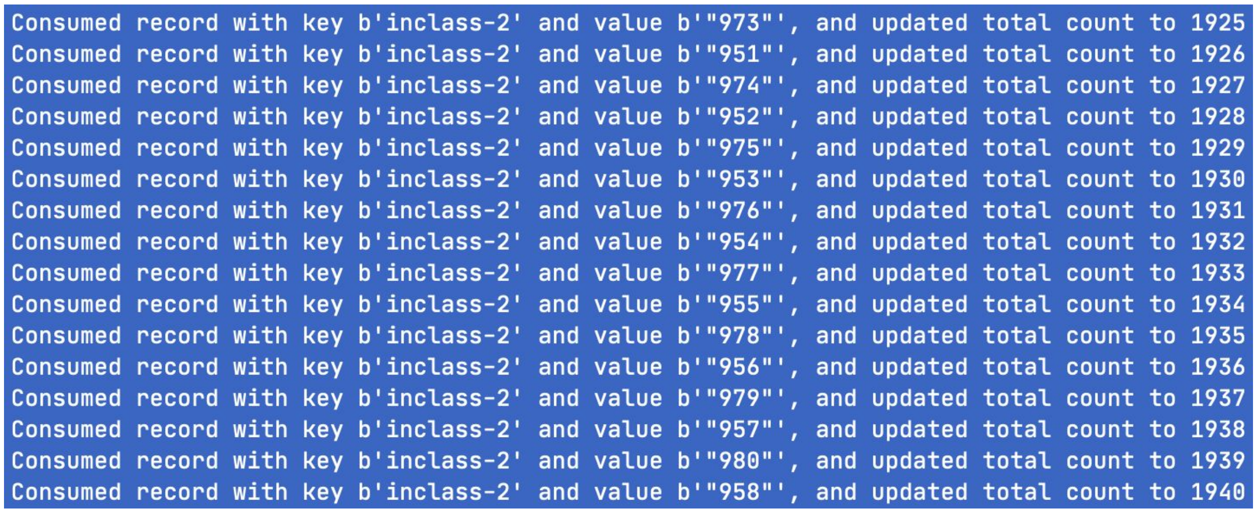
4. Is there a way to determine how much data Kafka/Confluent is storing for your topic? Do the Confluent monitoring tools help with this?

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.

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.

All the data is consumed by order of offset, not the value. The data produced by two different processes are alternately consumed.

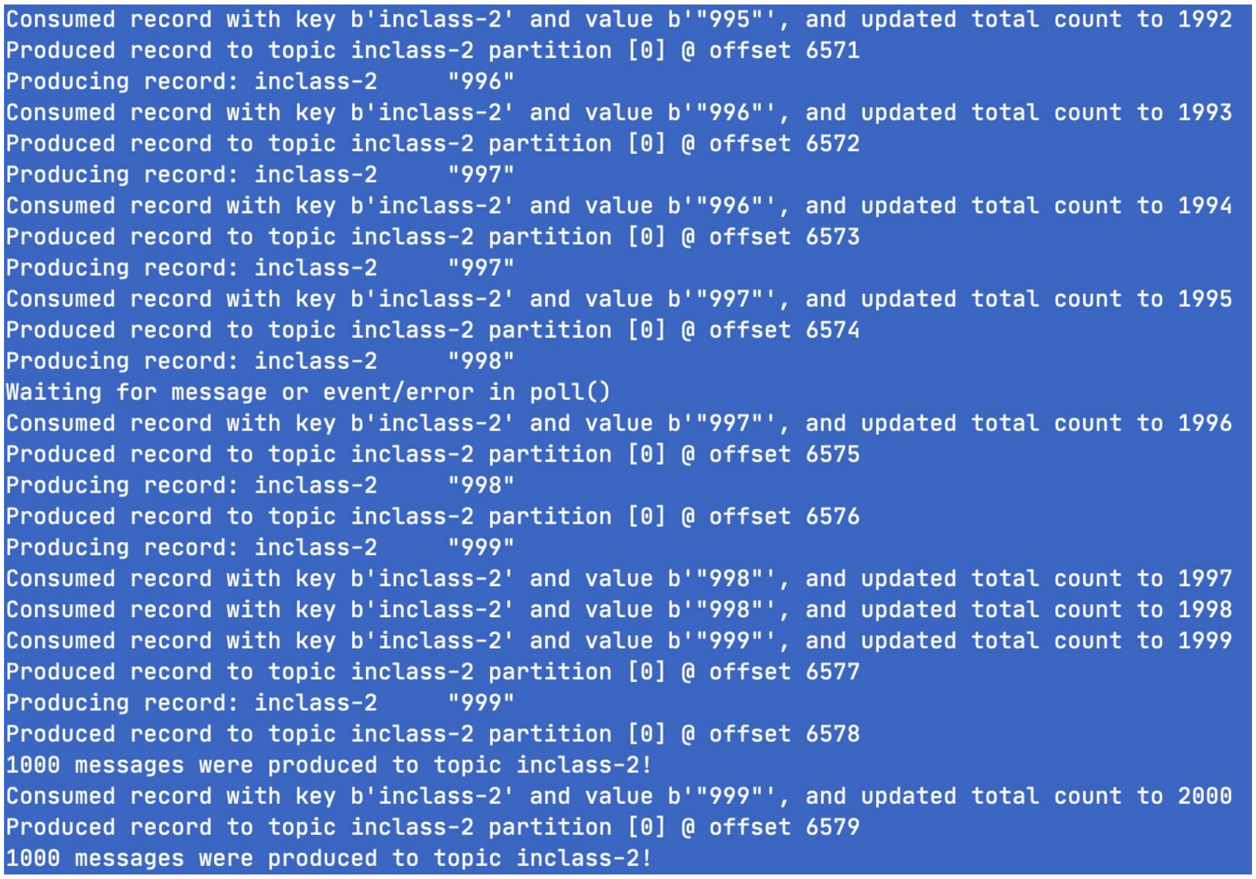


```
Consumed record with key b'inclass-2' and value b'"973"', and updated total count to 1925
Consumed record with key b'inclass-2' and value b'"951"', and updated total count to 1926
Consumed record with key b'inclass-2' and value b'"974"', and updated total count to 1927
Consumed record with key b'inclass-2' and value b'"952"', and updated total count to 1928
Consumed record with key b'inclass-2' and value b'"975"', and updated total count to 1929
Consumed record with key b'inclass-2' and value b'"953"', and updated total count to 1930
Consumed record with key b'inclass-2' and value b'"976"', and updated total count to 1931
Consumed record with key b'inclass-2' and value b'"954"', and updated total count to 1932
Consumed record with key b'inclass-2' and value b'"977"', and updated total count to 1933
Consumed record with key b'inclass-2' and value b'"955"', and updated total count to 1934
Consumed record with key b'inclass-2' and value b'"978"', and updated total count to 1935
Consumed record with key b'inclass-2' and value b'"956"', and updated total count to 1936
Consumed record with key b'inclass-2' and value b'"979"', and updated total count to 1937
Consumed record with key b'inclass-2' and value b'"957"', and updated total count to 1938
Consumed record with key b'inclass-2' and value b'"980"', and updated total count to 1939
Consumed record with key b'inclass-2' and value b'"958"', and updated total count to 1940
```

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.

The program runs two producers and two consumers concurrently (four threads). Those messages are produced and consumed quickly. In almost time, two or three messages are produced, then the consumers will get them at one time.



```
Consumed record with key b'inclass-2' and value b'"995"', and updated total count to 1992
Produced record to topic inclass-2 partition [0] @ offset 6571
Producing record: inclass-2     "996"
Consumed record with key b'inclass-2' and value b'"996"', and updated total count to 1993
Produced record to topic inclass-2 partition [0] @ offset 6572
Producing record: inclass-2     "997"
Consumed record with key b'inclass-2' and value b'"996"', and updated total count to 1994
Produced record to topic inclass-2 partition [0] @ offset 6573
Producing record: inclass-2     "997"
Consumed record with key b'inclass-2' and value b'"997"', and updated total count to 1995
Produced record to topic inclass-2 partition [0] @ offset 6574
Producing record: inclass-2     "998"
Waiting for message or event/error in poll()
Consumed record with key b'inclass-2' and value b'"997"', and updated total count to 1996
Produced record to topic inclass-2 partition [0] @ offset 6575
Producing record: inclass-2     "998"
Produced record to topic inclass-2 partition [0] @ offset 6576
Producing record: inclass-2     "999"
Consumed record with key b'inclass-2' and value b'"998"', and updated total count to 1997
Consumed record with key b'inclass-2' and value b'"998"', and updated total count to 1998
Consumed record with key b'inclass-2' and value b'"999"', and updated total count to 1999
Produced record to topic inclass-2 partition [0] @ offset 6577
Producing record: inclass-2     "999"
Produced record to topic inclass-2 partition [0] @ offset 6578
1000 messages were produced to topic inclass-2!
Consumed record with key b'inclass-2' and value b'"999"', and updated total count to 2000
Produced record to topic inclass-2 partition [0] @ offset 6579
1000 messages were produced to topic inclass-2!
```

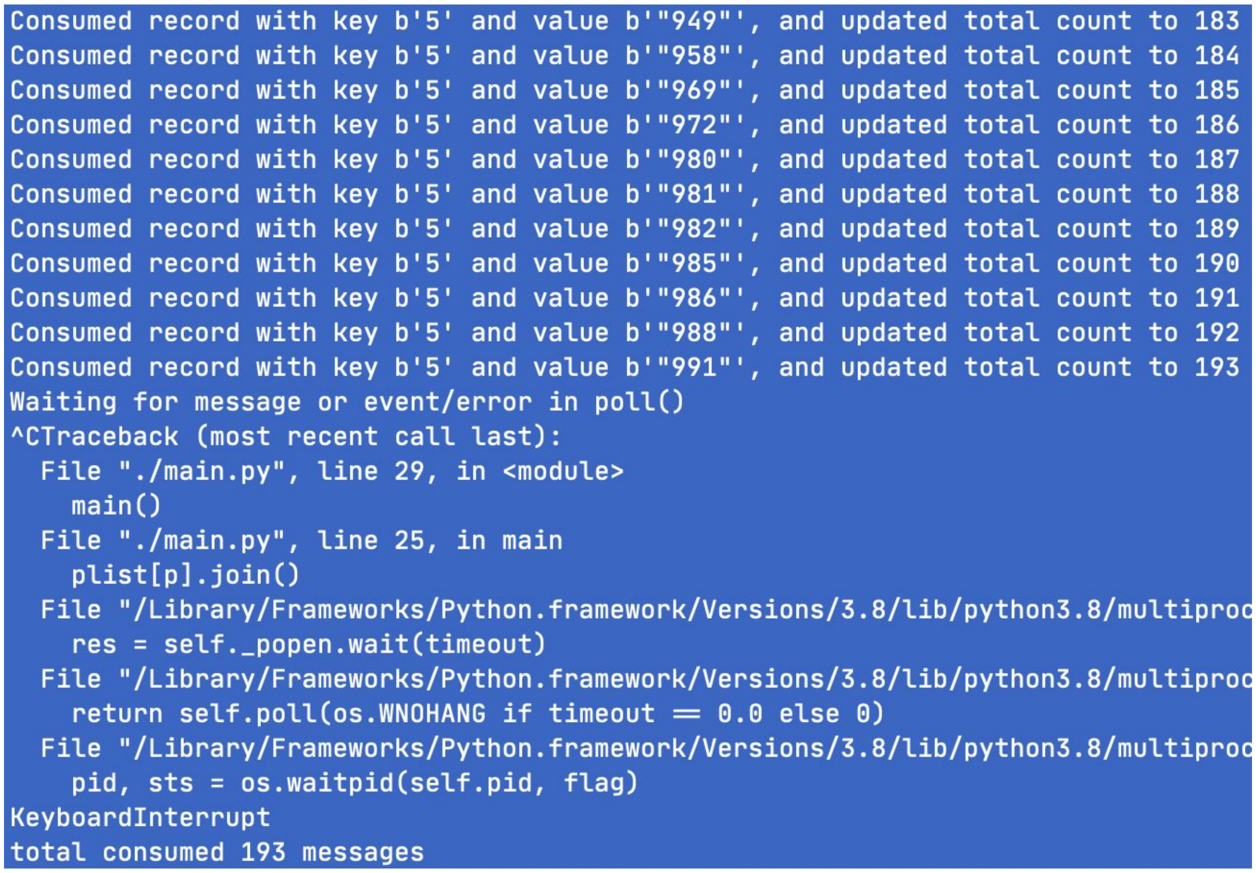
F. Varying Keys

1. Clear all data from the topic

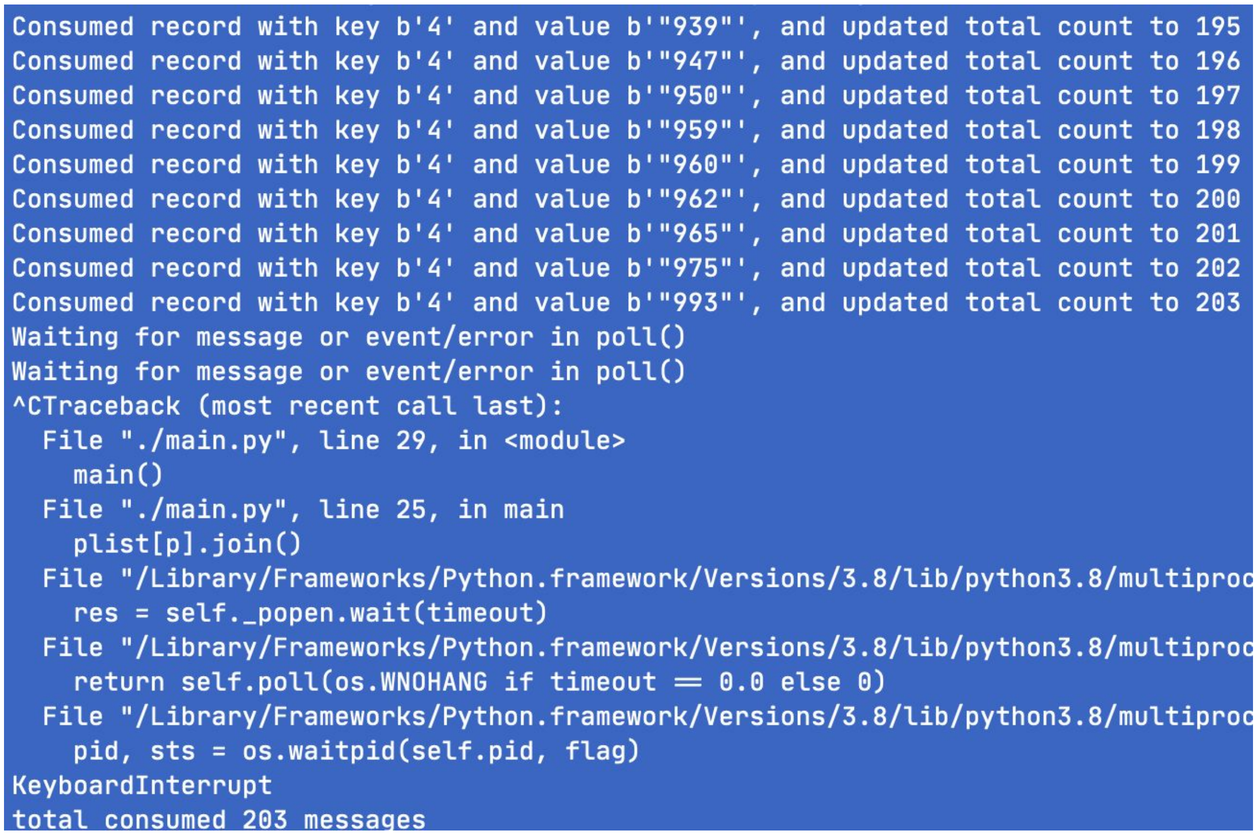
So far you have kept the “key” value constant for each record sent on a topic. But keys can be very useful to choose specific records from a stream.

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).

   When those messages are produced, the partition is allocated by the key value. In this case, the consumer will consume from a specific partition to consume the records with a specific key. In this assignment, there are 1000 total messages assigned to 1-5 different partitions. 1, 2, 3, 4, 5 each partition has 219, 203, 182, 193, and 203. 219+203+182+193+203=1000



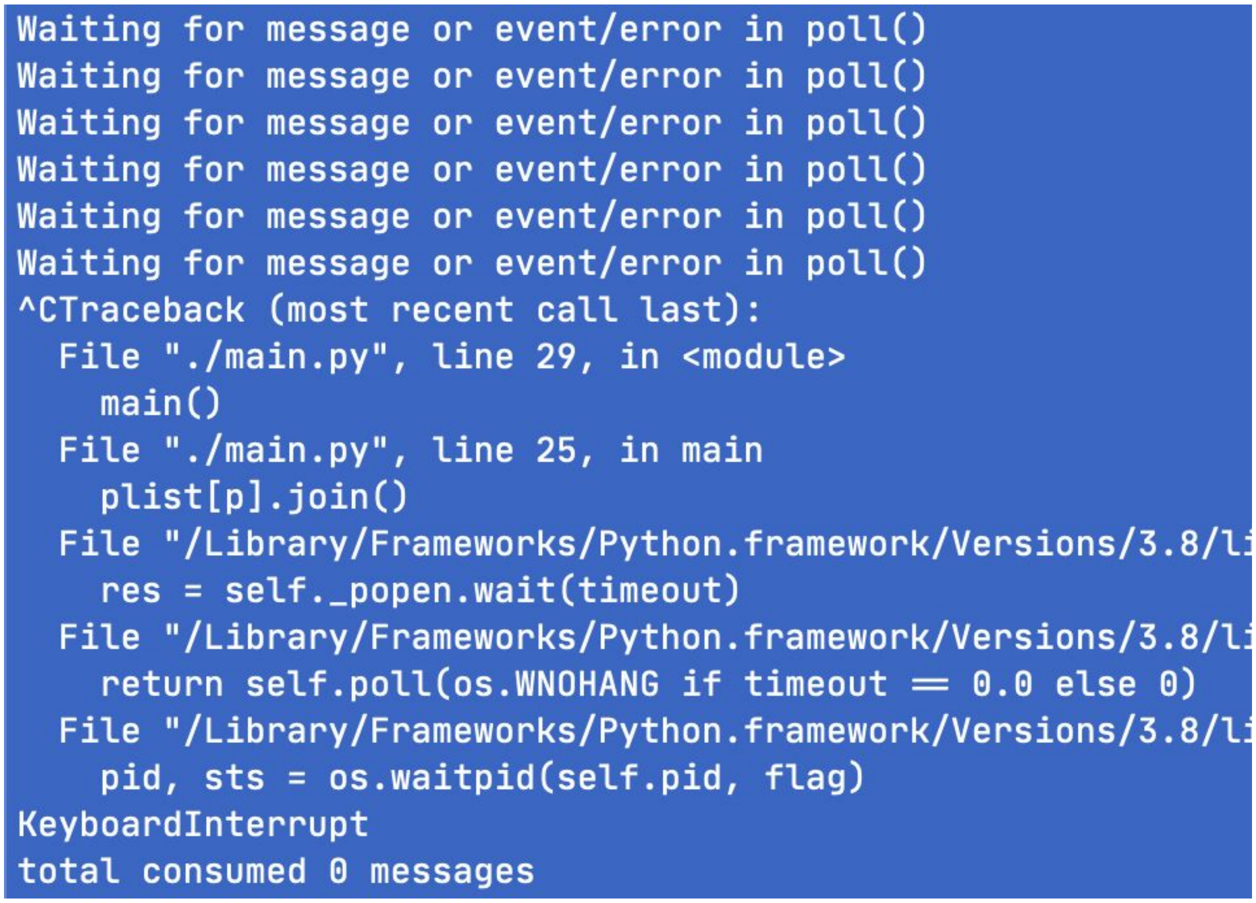
```
Consumed record with key b'5' and value b'"949"', and updated total count to 183
Consumed record with key b'5' and value b'"958"', and updated total count to 184
Consumed record with key b'5' and value b'"969"', and updated total count to 185
Consumed record with key b'5' and value b'"972"', and updated total count to 186
Consumed record with key b'5' and value b'"980"', and updated total count to 187
Consumed record with key b'5' and value b'"981"', and updated total count to 188
Consumed record with key b'5' and value b'"982"', and updated total count to 189
Consumed record with key b'5' and value b'"985"', and updated total count to 190
Consumed record with key b'5' and value b'"986"', and updated total count to 191
Consumed record with key b'5' and value b'"988"', and updated total count to 192
Consumed record with key b'5' and value b'"991"', and updated total count to 193
Waiting for message or event/error in poll()
^CTraceback (most recent call last):
  File "./main.py", line 29, in <module>
    main()
  File "./main.py", line 25, in main
    plist[p].join()
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    res = self._popen.wait(timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt
total consumed 193 messages
```



```
Consumed record with key b'4' and value b'"939"', and updated total count to 195
Consumed record with key b'4' and value b'"947"', and updated total count to 196
Consumed record with key b'4' and value b'"950"', and updated total count to 197
Consumed record with key b'4' and value b'"959"', and updated total count to 198
Consumed record with key b'4' and value b'"960"', and updated total count to 199
Consumed record with key b'4' and value b'"962"', and updated total count to 200
Consumed record with key b'4' and value b'"965"', and updated total count to 201
Consumed record with key b'4' and value b'"975"', and updated total count to 202
Consumed record with key b'4' and value b'"993"', and updated total count to 203
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^CTraceback (most recent call last):
  File "./main.py", line 29, in <module>
    main()
  File "./main.py", line 25, in main
    plist[p].join()
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    res = self._popen.wait(timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiproc
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt
total consumed 203 messages
```

4. Attempt to consume records with a key that does not exist. E.g., consume records with key value of "100". Describe the results

The program will wait for messages forever since this partition does not exist.



```
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^CTraceback (most recent call last):
  File "./main.py", line 29, in <module>
    main()
  File "./main.py", line 25, in main
    plist[p].join()
  File "/Library/Frameworks/Python.framework/Versions/3.8/li
    res = self._popen.wait(timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.8/li
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/Library/Frameworks/Python.framework/Versions/3.8/li
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt
total consumed 0 messages
```

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?

This is possible. Since consumers consume specific keys from the specific partition, and each partition has a separate end offset marker, consume messages from one partition will not affect the others.

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?
   This function will wait for all messages in the Producer queue to be delivered.

2. What happens if you do not call producer.flush()?
   In this case, some messages will not be delivered. The produce is an asynchronous function that only moves the message to the internal queue and waits to send it to the broker. And the outstanding messages might not send to broker until the program exit.

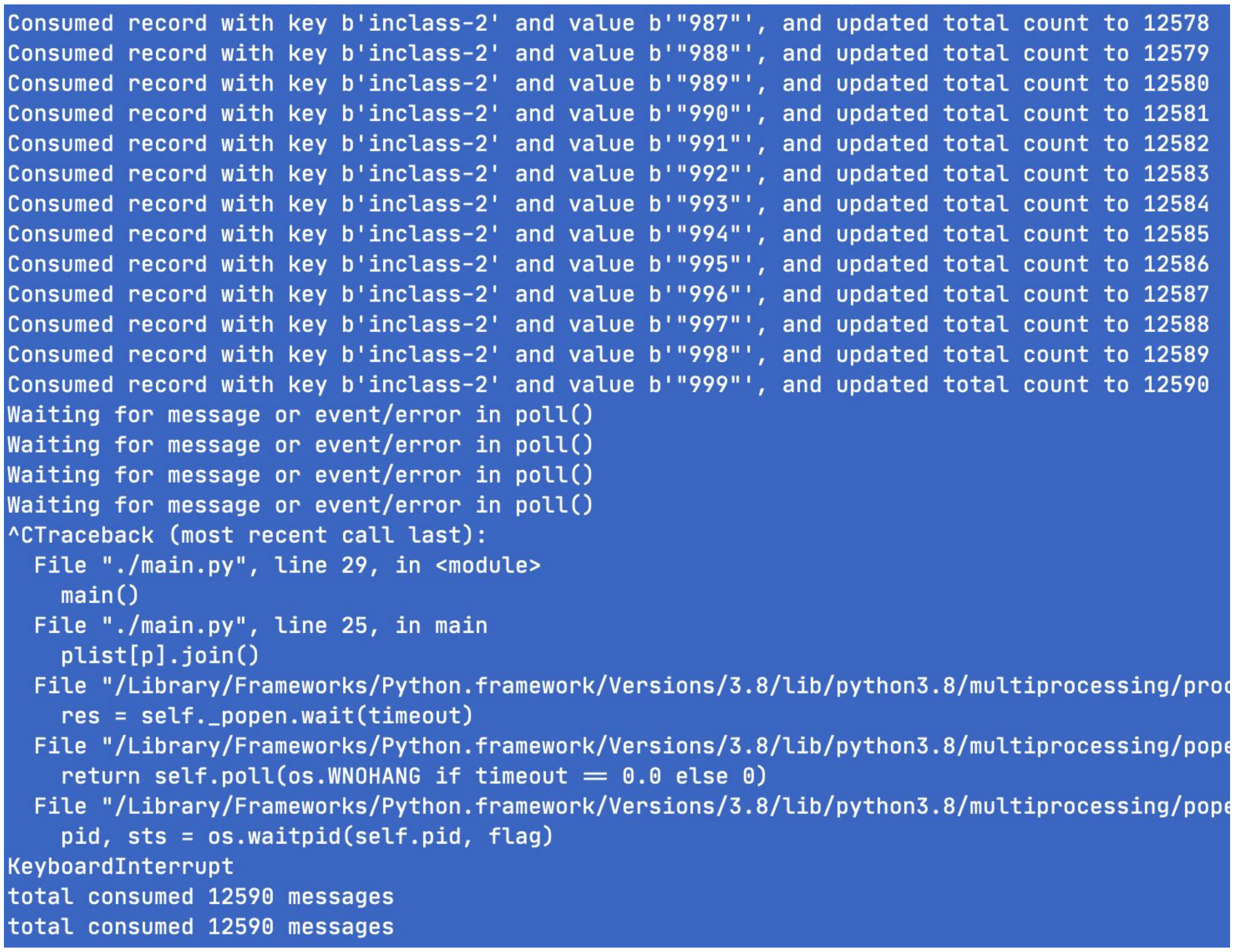
3. What happens if you call producer.flush() after sending each record?
   In this case, the producer will slow, and it like a sync function. Every time a message was sent to the internal queue by producer(), it will be sent to the broker immediately. Those cause low performance and poor scalability and make the program limited by the network and broker latency.

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

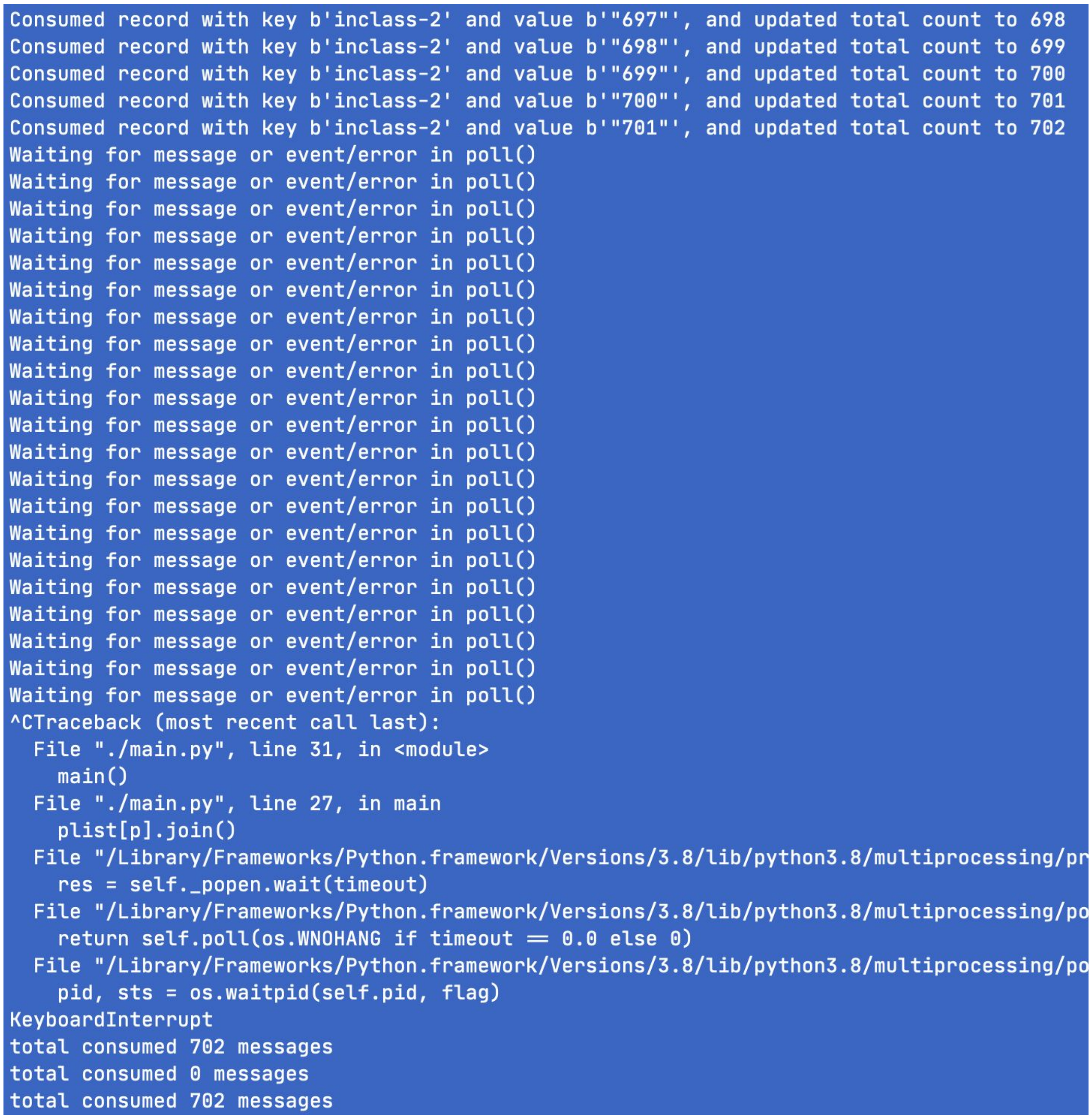
1. Create two consumer groups with one consumer program instance in each group.
   Two groups named “1” and “2” are created.

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.
   Two consumers in each group also get all of the 12590 messages on this topic.


```
Consumed record with key b'inclass-2' and value b'"987"', and updated total count to 12578
Consumed record with key b'inclass-2' and value b'"988"', and updated total count to 12579
Consumed record with key b'inclass-2' and value b'"989"', and updated total count to 12580
Consumed record with key b'inclass-2' and value b'"990"', and updated total count to 12581
Consumed record with key b'inclass-2' and value b'"991"', and updated total count to 12582
Consumed record with key b'inclass-2' and value b'"992"', and updated total count to 12583
Consumed record with key b'inclass-2' and value b'"993"', and updated total count to 12584
Consumed record with key b'inclass-2' and value b'"994"', and updated total count to 12585
Consumed record with key b'inclass-2' and value b'"995"', and updated total count to 12586
Consumed record with key b'inclass-2' and value b'"996"', and updated total count to 12587
Consumed record with key b'inclass-2' and value b'"997"', and updated total count to 12588
Consumed record with key b'inclass-2' and value b'"998"', and updated total count to 12589
Consumed record with key b'inclass-2' and value b'"999"', and updated total count to 12590
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^CTraceback (most recent call last):
  File "./main.py", line 29, in <module>
    main()
  File "./main.py", line 25, in main
    plist[p].join()
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/proc
    res = self._popen.wait(timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/pope
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/pope
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt
total consumed 12590 messages
total consumed 12590 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.

Three consumers consumed 702, 0, 702 in total 702 messages. Actually, consumer in each group only consumes the message one time. Actually, the question is why only one consumer consumes successfully.



```
Consumed record with key b'inclass-2' and value b'"697"', and updated total count to 698
Consumed record with key b'inclass-2' and value b'"698"', and updated total count to 699
Consumed record with key b'inclass-2' and value b'"699"', and updated total count to 700
Consumed record with key b'inclass-2' and value b'"700"', and updated total count to 701
Consumed record with key b'inclass-2' and value b'"701"', and updated total count to 702
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^CTraceback (most recent call last):
  File "./main.py", line 31, in <module>
    main()
  File "./main.py", line 27, in main
    plist[p].join()
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/pr
    res = self._popen.wait(timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/po
    return self.poll(os.WNOHANG if timeout == 0.0 else 0)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/multiprocessing/po
    pid, sts = os.waitpid(self.pid, flag)
KeyboardInterrupt
total consumed 702 messages
total consumed 0 messages
total consumed 702 messages
```

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.