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.

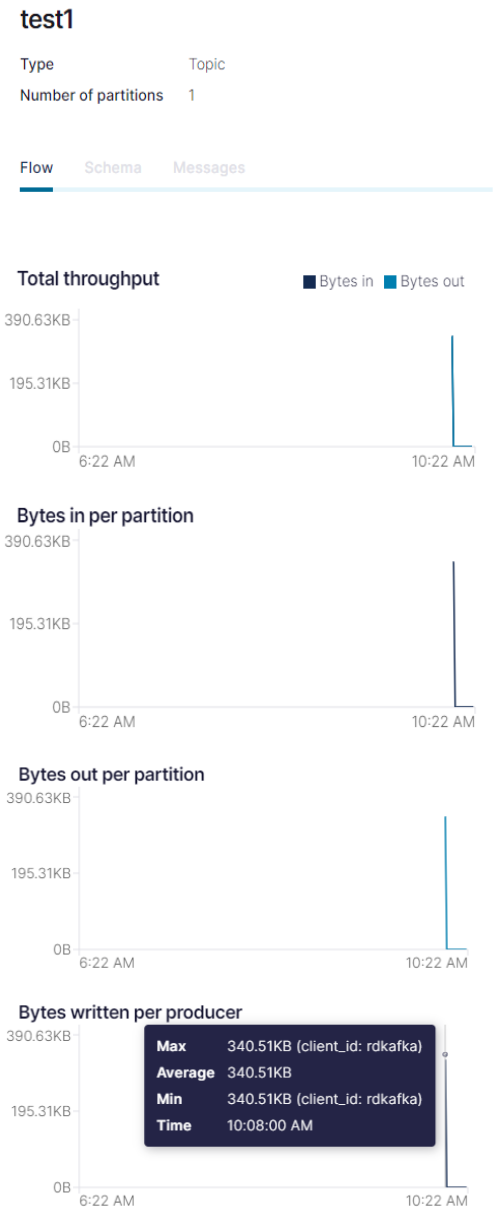
The goal for this week is to gain experience and knowledge of using a streaming data transport system (Kafka). Complete as many of the following exercises as you can. Proceed at a pace that allows you to learn and understand the use of Kafka with python.

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?



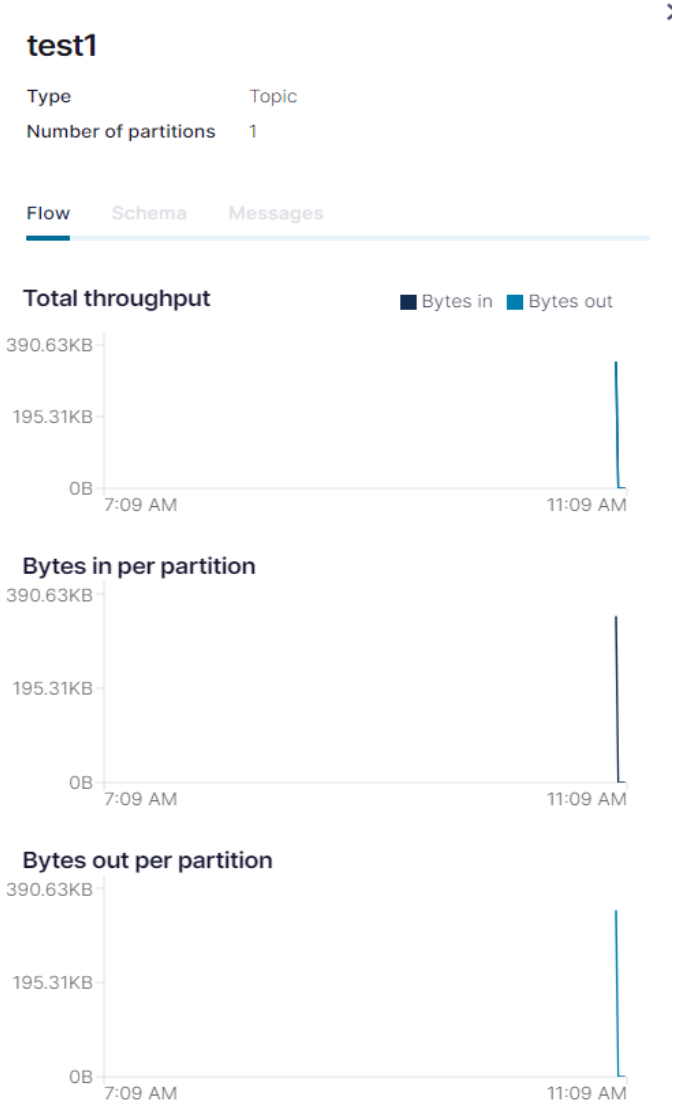
   The above information shows the data flow of test1 topic over a period of time. These values are reasonable, because the size of the json file is around 340kb.
2. Use this monitoring feature as you do each of the following exercises.

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.

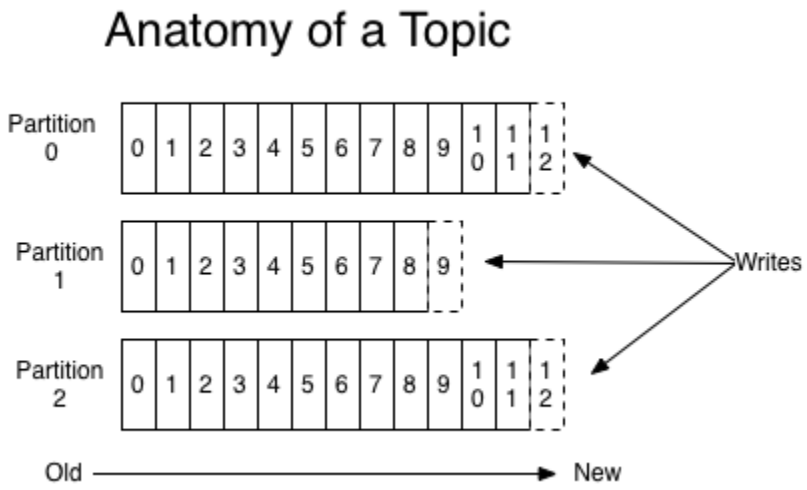
```
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$ wc /home/xiaoran/bcsample.json
 16001  30002 370892 /home/xiaoran/bcsample.json
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

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



The Producer produced 1000 records, and the consumer received 1000 records when it was first run, and the consumer will no longer receive records after running it again.

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

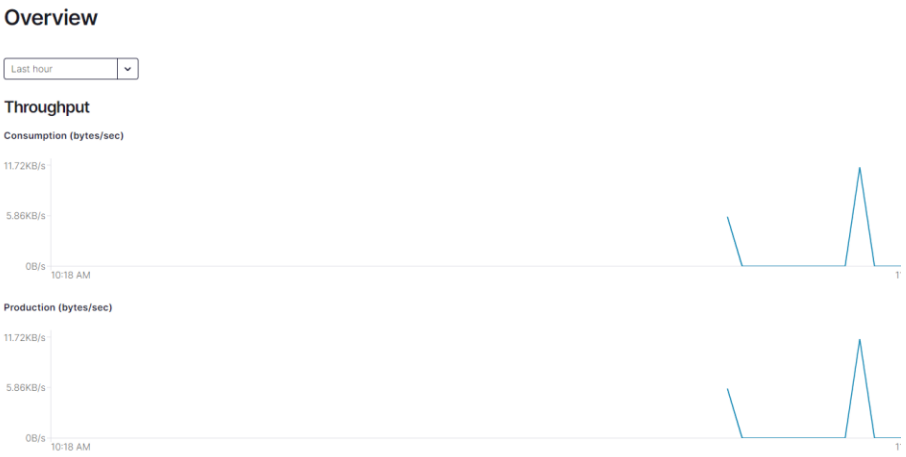


I think the data should be uploaded to Kafka's cloud server. To be precise, it should be written to the Topic's partition.

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, it can be seen from Confluent monitoring tools how much data is stored, the tools are very helpful.
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

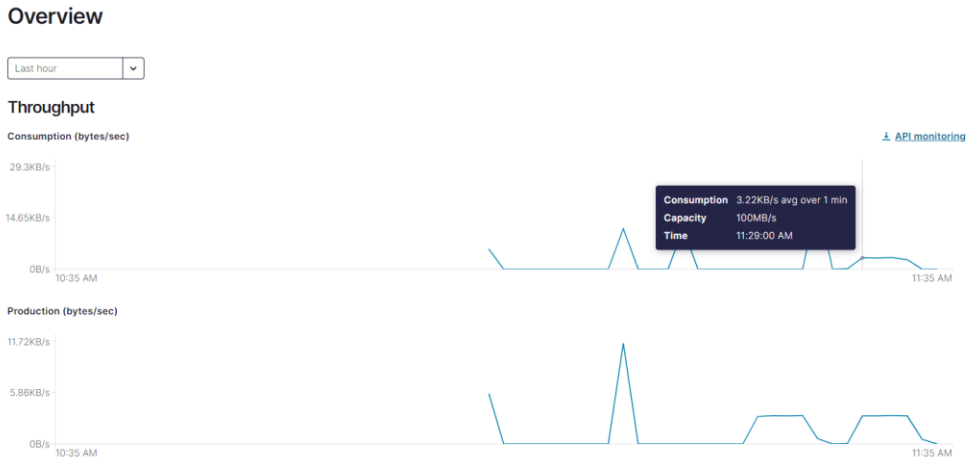
1. Clear all data from the topic
2. Run two versions of your producer concurrently, have each of them send all 1000 of your sample records. When finished, run your consumer once. Describe the results.

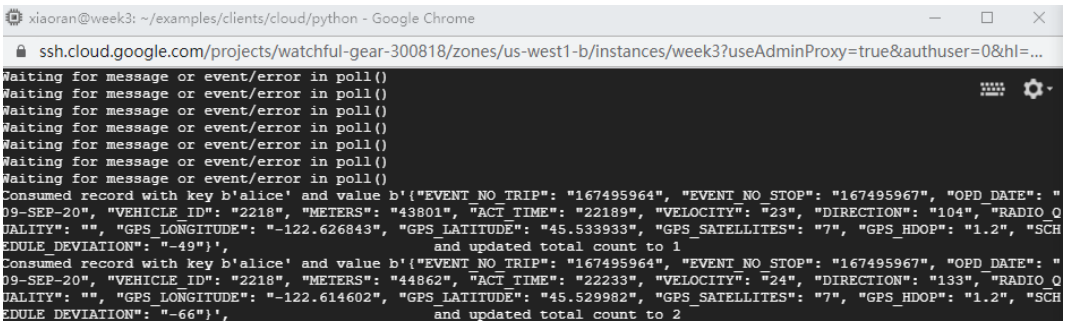


As can be seen from the figure, the producer's production speed is twice as fast as before, and the consumer's consumption speed is also twice as fast as before. I think this shows that deploying multiple producers can increase the data transfer speed.

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.

   It can be seen from the figure that the consumption speed of the consumer has not been improved, but has been reduced. I think this is because there can only be one the consumer running at the same time, and the other consumer needs to wait for the first consumer to stop working before running.

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.
   I choose a randon number as records’ keys, when the producer producing the records, the producer will sent the records to the partition has same key as records.

```
    for n in range(1000):
        # F:Varying Keys
        record_key =str(random.randint(1,5))
        '''I part
        if n==0:
            producer.init_transactions();
        '''
        #record_key="alice"
        record_value = json.dumps(json_object[n])
        print("Producing record: {}\t{}".format(record_key, record_value))
        producer.produce(topic, key=record_key, value=record_value, on_delivery=acked,parti
tion=int(record_key))
```

3. Modify your consumer to consume only records with a specific key (or subset of keys).
   After I searched in google, I know I can use assign() to modify consumer to consume records with a specific key.

https://stackoverflow.com/questions/58724645/is-it-possible-to-consume-kafka-messages-using-key-and-partition

```
    #f:varify keys:
    consumer.assign([TopicPartition(topic,1)])
```

```
Consumer1:Consumed record with key b'1' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "79618", "AC
T_TIME": "25606", "VELOCITY": "25", "DIRECTION": "233", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.600583", "GPS_LATITUDE": "45.529785", "GPS_SATELLITES": "10", "GPS_HDOP": "0.8", "
SCHEDULE_DEVIATION": "608"}',                    and updated total count to 221
Consumer1:Consumed record with key b'1' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "80001", "AC
T_TIME": "25621", "VELOCITY": "25", "DIRECTION": "241", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.604643", "GPS_LATITUDE": "45.527823", "GPS_SATELLITES": "10", "GPS_HDOP": "0.8", "
SCHEDULE_DEVIATION": "602"}',                    and updated total count to 222
Consumer1:Consumed record with key b'1' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "81012", "AC
T_TIME": "25661", "VELOCITY": "25", "DIRECTION": "312", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.61588", "GPS_LATITUDE": "45.530973", "GPS_SATELLITES": "11", "GPS_HDOP": "0.7", "S
CHEDULE_DEVIATION": "588"}',                    and updated total count to 223
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

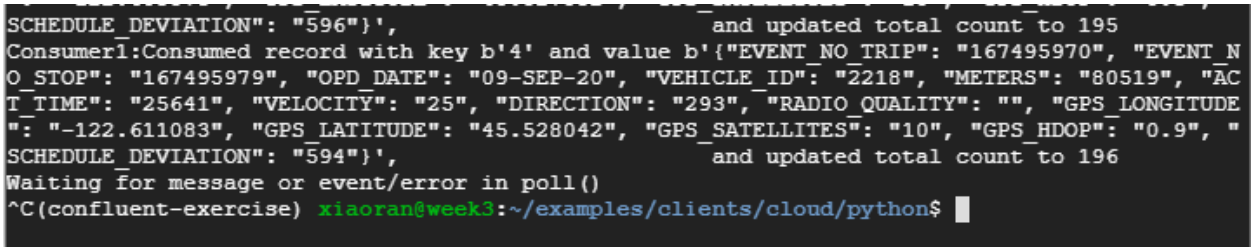
Key==1

```
Consumer1:Consumed record with key b'2' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "80261", "AC
T_TIME": "25631", "VELOCITY": "26", "DIRECTION": "266", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.607902", "GPS_LATITUDE": "45.5274", "GPS_SATELLITES": "10", "GPS_HDOP": "0.9", "SC
HEDULE_DEVIATION": "598"}',                    and updated total count to 202
Consumer1:Consumed record with key b'2' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "80763", "AC
T_TIME": "25651", "VELOCITY": "24", "DIRECTION": "313", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.613515", "GPS_LATITUDE": "45.529452", "GPS_SATELLITES": "10", "GPS_HDOP": "0.9", "
SCHEDULE_DEVIATION": "591"}',                    and updated total count to 203
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^C(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

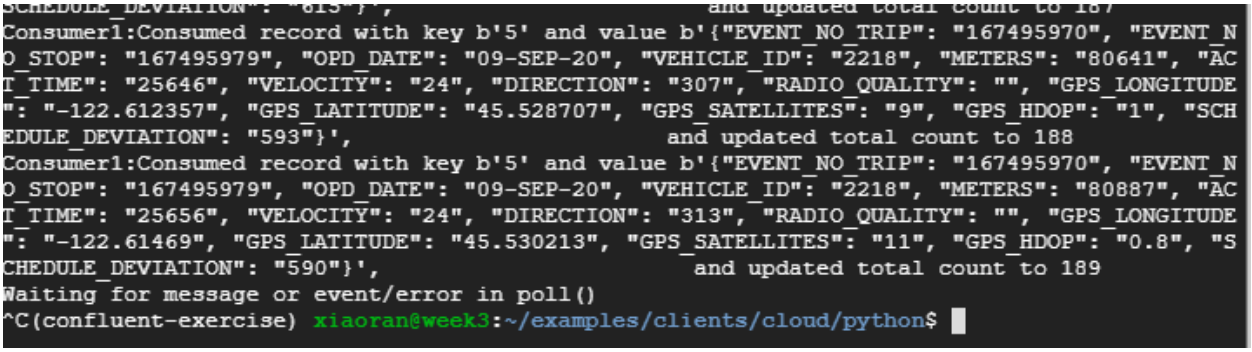
Key==2

```
": "-122.594045", "GPS_LATITUDE": "45.533158", "GPS_SATELLITES": "10", "GPS_HDOP": "0.8", "
SCHEDULE_DEVIATION": "617"}',                    and updated total count to 187
Consumer1:Consumed record with key b'3' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "79233", "AC
T_TIME": "25591", "VELOCITY": "25", "DIRECTION": "233", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.596577", "GPS_LATITUDE": "45.531855", "GPS_SATELLITES": "10", "GPS_HDOP": "0.8", "
SCHEDULE_DEVIATION": "614"}',                    and updated total count to 188
Consumer1:Consumed record with key b'3' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_N
O_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "81142", "AC
T_TIME": "25666", "VELOCITY": "26", "DIRECTION": "307", "RADIO_QUALITY": "", "GPS_LONGITUDE
": "-122.617215", "GPS_LATITUDE": "45.531678", "GPS_SATELLITES": "11", "GPS_HDOP": "0.7", "
SCHEDULE_DEVIATION": "586"}',                    and updated total count to 189
Waiting for message or event/error in poll()
^C(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

Key==3

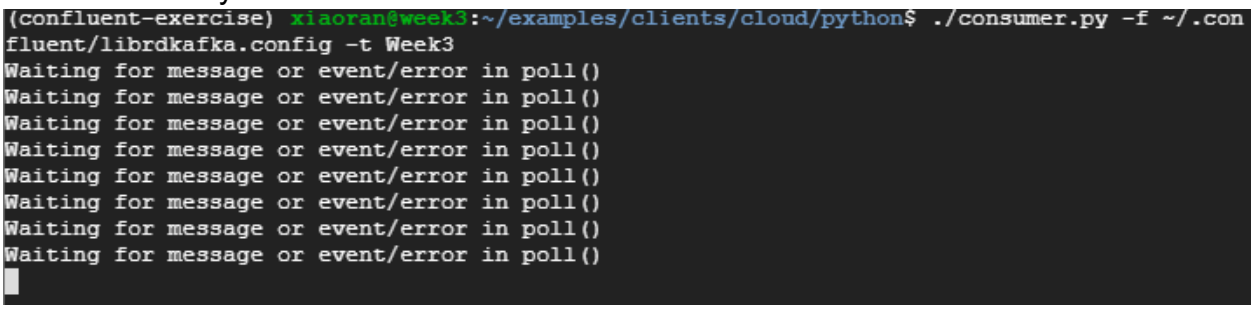


Key==4



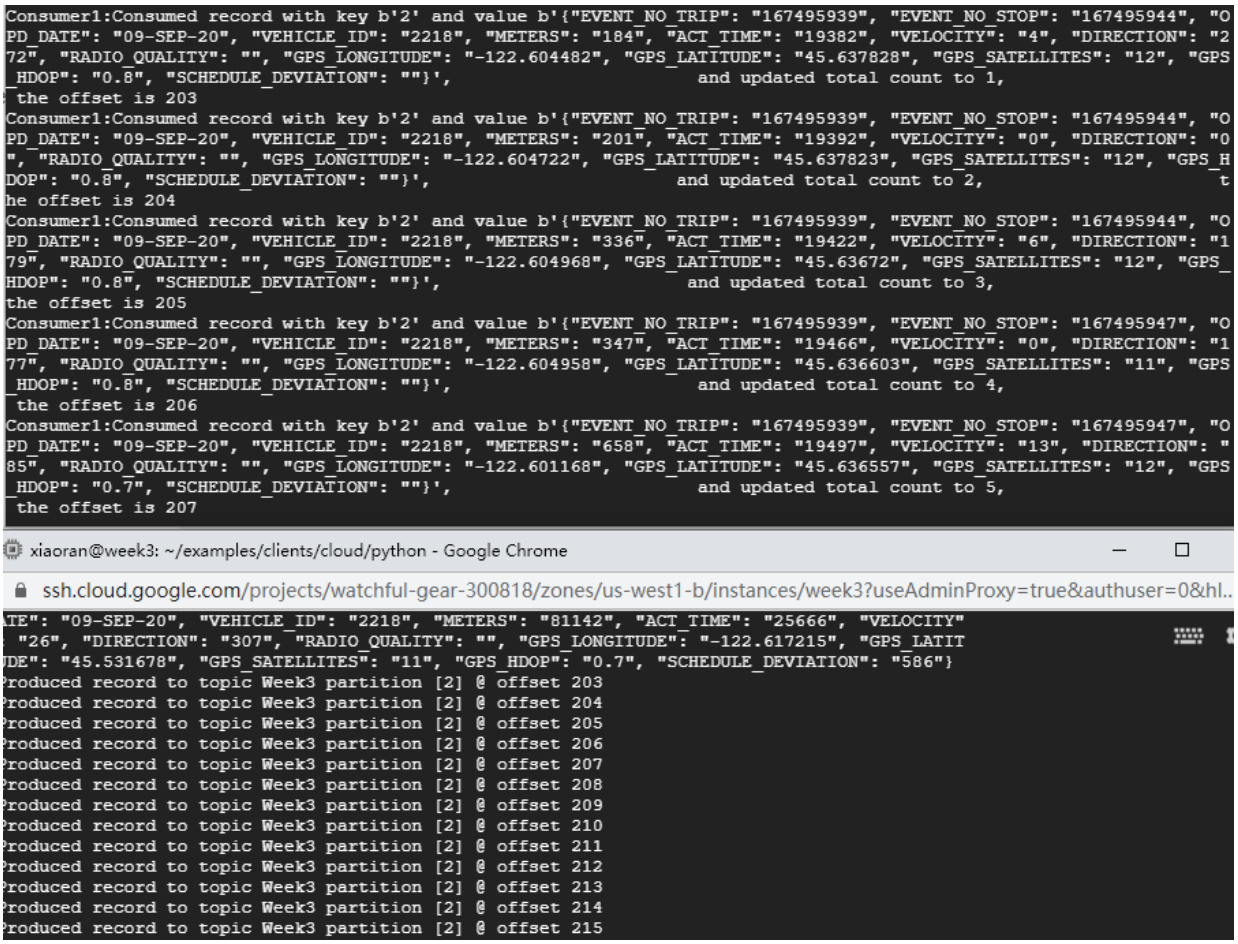
Key==5

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

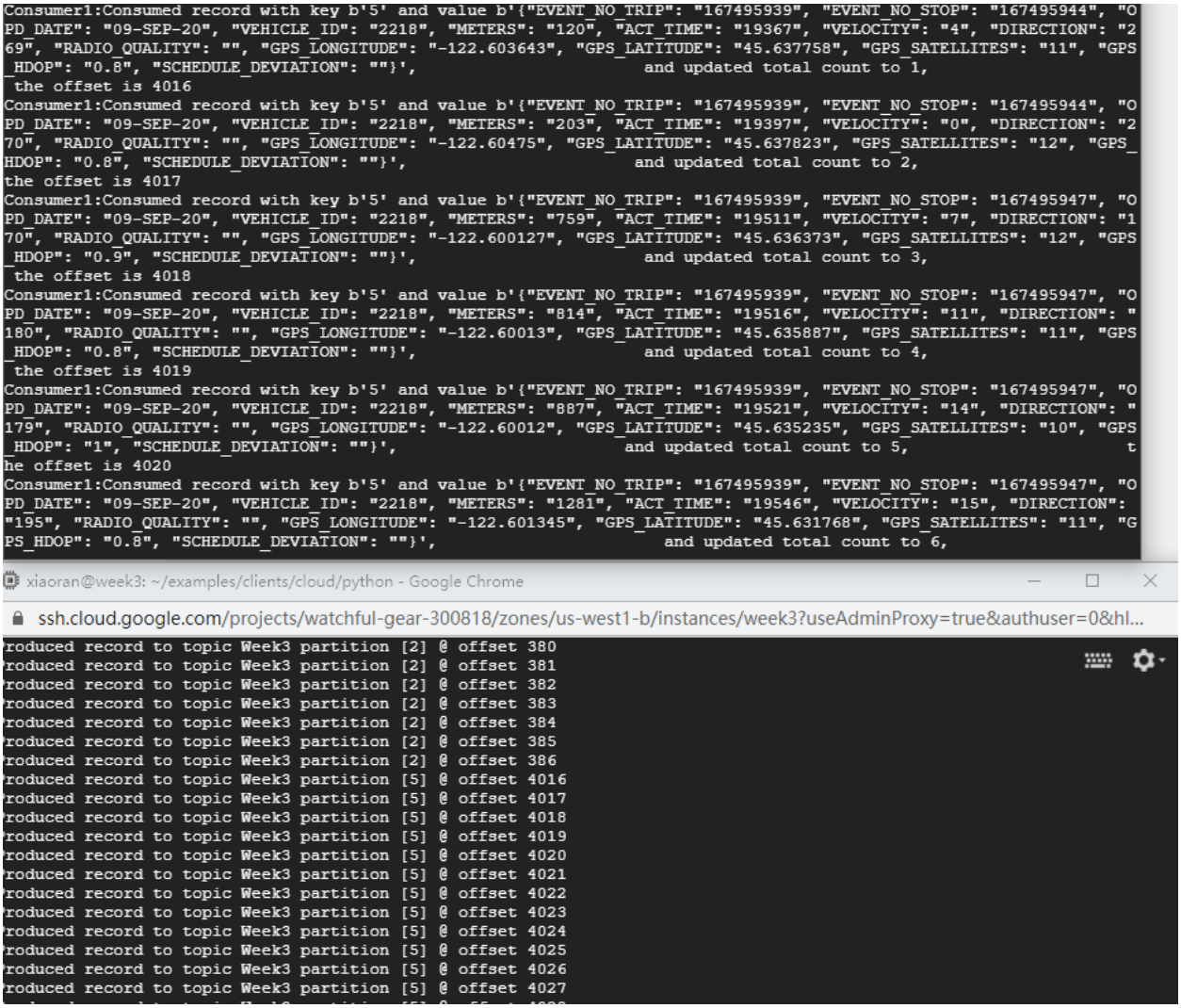


K==100 , the consumer cant recevie any records

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?
   I can create such a consumer.



Key==2 , print the offset and compare, the records maintaining the same order within key==2



Key==5 , print the offset and compare, the records maintaining the same order within key==5

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. This is a convenience method that calls poll() until len() is zero or the optional timeout elapses.
2. What happens if you do not call producer.flush()?

Nothing was produced to the topic.

```
^C(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$ ./consumer.py -f ~/.confluent/librdkafka.config -t
test1
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
^C(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

```
<class 'int'>
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": '
D": "2218", "METERS": "81012", "ACT_TIME": "25661", "VELOCITY": "25", "DI
ITUDE": "-122.61588", "GPS_LATITUDE": "45.530973", "GPS_SATELLITES": "11'
"}
<class 'int'>
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": '
D": "2218", "METERS": "81142", "ACT_TIME": "25666", "VELOCITY": "26", "DI
ITUDE": "-122.617215", "GPS_LATITUDE": "45.531678", "GPS_SATELLITES": "1
6"}
0 messages were produced to topic test1!
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

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

```
Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "0
9-SEP-20", "VEHICLE_ID": "2218", "METERS": "81012", "ACT_TIME": "25661", "VELOCITY": "25", "DIRECTION": "312", "RADIO_QUA
LITY": "", "GPS_LONGITUDE": "-122.61588", "GPS_LATITUDE": "45.530973", "GPS_SATELLITES": "11", "GPS_HDOP": "0.7", "SCHEDU
LE_DEVIATION": "588"}',                    and updated total count to 1000
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

xiaoran@week3: ~/examples/clients/cloud/python - Google Chrome

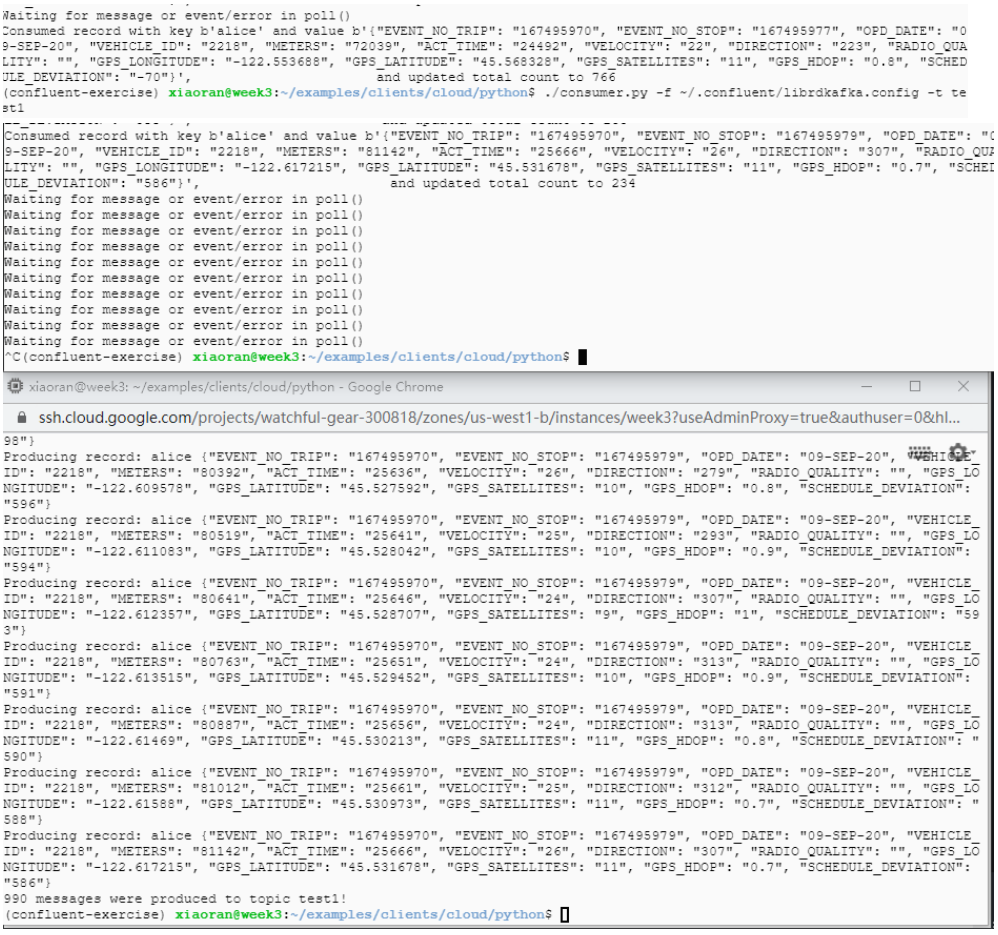
ssh.cloud.google.com/projects/watchful-gear-300818/zones/us-west1-b/instances/week3?useAdminProxy=true&authuser=0&hl...

```
ID": "2218", "METERS": "80519", "ACT_TIME": "25641", "VELOCITY": "25", "DIRECTION": "293", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.611083", "GPS_LATITUDE": "45.528042", "GPS_SATELLITES": "10", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION":
"594"}
Produced record to topic test1 partition [0] @ offset 156999
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_
ID": "2218", "METERS": "80641", "ACT_TIME": "25646", "VELOCITY": "24", "DIRECTION": "307", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.612357", "GPS_LATITUDE": "45.528707", "GPS_SATELLITES": "9", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "59
3"}
Produced record to topic test1 partition [0] @ offset 157000
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_
ID": "2218", "METERS": "80763", "ACT_TIME": "25651", "VELOCITY": "24", "DIRECTION": "313", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.613515", "GPS_LATITUDE": "45.529452", "GPS_SATELLITES": "10", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION":
"591"}
Produced record to topic test1 partition [0] @ offset 157001
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_
ID": "2218", "METERS": "80887", "ACT_TIME": "25656", "VELOCITY": "24", "DIRECTION": "313", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.61469", "GPS_LATITUDE": "45.530213", "GPS_SATELLITES": "11", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "
590"}
Produced record to topic test1 partition [0] @ offset 157002
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_
ID": "2218", "METERS": "81012", "ACT_TIME": "25661", "VELOCITY": "25", "DIRECTION": "312", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.61588", "GPS_LATITUDE": "45.530973", "GPS_SATELLITES": "11", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": "
588"}
Produced record to topic test1 partition [0] @ offset 157003
Producing record: alice {"EVENT_NO_TRIP": "167495970", "EVENT_NO_STOP": "167495979", "OPD_DATE": "09-SEP-20", "VEHICLE_
ID": "2218", "METERS": "81142", "ACT_TIME": "25666", "VELOCITY": "26", "DIRECTION": "307", "RADIO_QUALITY": "", "GPS_LO
NGITUDE": "-122.617215", "GPS_LATITUDE": "45.531678", "GPS_SATELLITES": "11", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION":
"586"}
Produced record to topic test1 partition [0] @ offset 157004
1000 messages were produced to topic test1!
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

Producer sent 1000 records, consumer also received 1000 data, after each producing, immediately send the record and print:
“Produced record to topic test1 partition [0] @ offset XXXXXX"

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?

Producer sent 990 records,cause 10<15,the last 10 record can't be produced to the topic.**But** consumer also received (766+234)==1000 records, the interruption occurred when the consumer was run for the first time. I think it was this interruption that made the results strange.
And after every 15 times producing,the producer sends the record and print:
"Produced record to topic test1 partition [0] @ offset XXXXXX"
Because every five recodes need to wait for 2 seconds, the consumer has no message to recevie, and the consumer will print:
"Waiting for message or event/error in poll()" twice,
I think this period == 2 seconds-producing 15 records.
I think the consumer receives the information after producer.flush(), not immediately. If the interval between flushes is too long, the consumer will not receive the information and needs to wait for the next flush.



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.

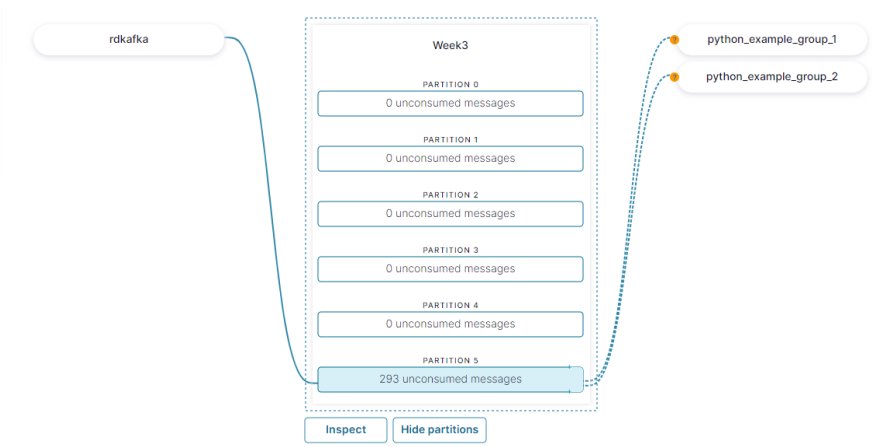
```
Consumer1:Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495939", "EVENT_NO_STOP": "167495947", "OP
D_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "2154", "ACT_TIME": "19606", "VELOCITY": "14", "DIRECTION": "180",
 "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.601787", "GPS_LATITUDE": "45.623865", "GPS_SATELLITES": "11", "GPS_HDOP": "
0.8", "SCHEDULE_DEVIATION": ""}',                    and updated total count to 49
Consumer1:Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495939", "EVENT_NO_STOP": "167495947", "OP
D_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "2225", "ACT_TIME": "19611", "VELOCITY": "14", "DIRECTION": "182",
 "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.601813", "GPS_LATITUDE": "45.62323", "GPS_SATELLITES": "12", "GPS_HDOP": "0
.8", "SCHEDULE_DEVIATION": ""}',                     and updated total count to 50
Consumer2:Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495939", "EVENT_NO_STOP": "167495944", "OP
D_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "15", "ACT_TIME": "19337", "VELOCITY": "", "DIRECTION": "", "RADIO
_QUALITY": "", "GPS_LONGITUDE": "-122.602352", "GPS_LATITUDE": "45.637868", "GPS_SATELLITES": "12", "GPS_HDOP": "0.8", "
SCHEDULE_DEVIATION": ""}',                    and updated total count to 51
```

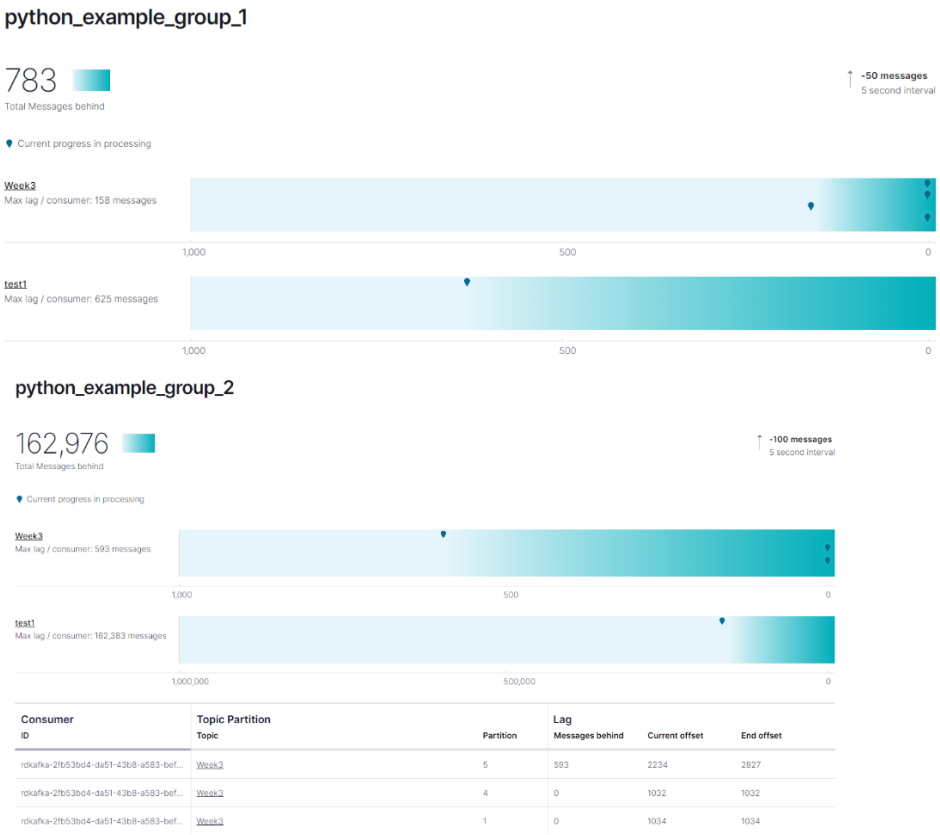
```
Consumer2:Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495939", "EVENT_NO_STOP": "167495947", "OP
D_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "2154", "ACT_TIME": "19606", "VELOCITY": "14", "DIRECTION": "180",
 "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.601787", "GPS_LATITUDE": "45.623865", "GPS_SATELLITES": "11", "GPS_HDOP": "
0.8", "SCHEDULE_DEVIATION": ""}',                     and updated total count to 99
Consumer2:Consumed record with key b'alice' and value b'{"EVENT_NO_TRIP": "167495939", "EVENT_NO_STOP": "167495947", "OP
D_DATE": "09-SEP-20", "VEHICLE_ID": "2218", "METERS": "2225", "ACT_TIME": "19611", "VELOCITY": "14", "DIRECTION": "182",
 "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.601813", "GPS_LATITUDE": "45.62323", "GPS_SATELLITES": "12", "GPS_HDOP": "0
.8", "SCHEDULE_DEVIATION": ""}',                    and updated total count to 100
(confluent-exercise) xiaoran@week3:~/examples/clients/cloud/python$
```

4. Create a second consumer within one of the groups so that you now have three consumers total.

```
consumer = Consumer({
    'bootstrap.servers': conf['bootstrap.servers'],
    'sasl.mechanisms': conf['sasl.mechanisms'],
    'security.protocol': conf['security.protocol'],
    'sasl.username': conf['sasl.username'],
    'sasl.password': conf['sasl.password'],
    'group.id': 'python_example_group_1',
    'auto.offset.reset': 'earliest',
    'isolation.level':'read_committed',
})
#H:Consumer Groups
consumer2 = Consumer({
    'bootstrap.servers': conf['bootstrap.servers'],
    'sasl.mechanisms': conf['sasl.mechanisms'],
    'security.protocol': conf['security.protocol'],
    'sasl.username': conf['sasl.username'],
    'sasl.password': conf['sasl.password'],
    'group.id': 'python_example_group_2',
    'auto.offset.reset': 'earliest',
})
consumer3 = Consumer({
    'bootstrap.servers': conf['bootstrap.servers'],
    'sasl.mechanisms': conf['sasl.mechanisms'],
    'security.protocol': conf['security.protocol'],
    'sasl.username': conf['sasl.username'],
    'sasl.password': conf['sasl.password'],
    'group.id': 'python_example_group_1',
    'auto.offset.reset': 'earliest',
})
```

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.

```
for n in range(1000):
    # F:Varying Keys
    # record_key =str(random.randint(1,5))
     if n==0:
          producer.init_transactions();
     record_key="alice"
     record_value = json.dumps(json_object[n])
     print("Producing record: {}\t{}".format(record_key, record_value))
     try:
         producer.produce(topic, key=record_key, value=record_value, on_delivery=acked)
     except confluent_kafka.KafkaException  as e:
         print(e)

     if (n+1)==4:
         sleep(2)
         p=random.randint(1,10)
         if(p>=5):
             try:
                 producer.commit_transaction()
                 print("Commit transaction sucess!")
             except confluent_kafka.KafkaException as e:
                 print(e)
         else:
             try:
                 producer.abort_transaction()
                 print("Commit transaction fail!")
             except confluent_kafka.KafkaException as e:
                 print(e)
```

Don’t know how to handle the KafkaError, KafkaError{code=_STATE,val=-172,str="Unable to produce message: Local: Erroneous state"}

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).

```
consumer = Consumer({
    'bootstrap.servers': conf['bootstrap.servers'],
    'sasl.mechanisms': conf['sasl.mechanisms'],
    'security.protocol': conf['security.protocol'],
    'sasl.username': conf['sasl.username'],
    'sasl.password': conf['sasl.password'],
    'group.id': 'python_example_group_1',
    'auto.offset.reset': 'earliest',
    'isolation.level':'read_committed',
    'transactional.id':'1',
})
```

Producer cant work, but the consumer can work.

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.