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.

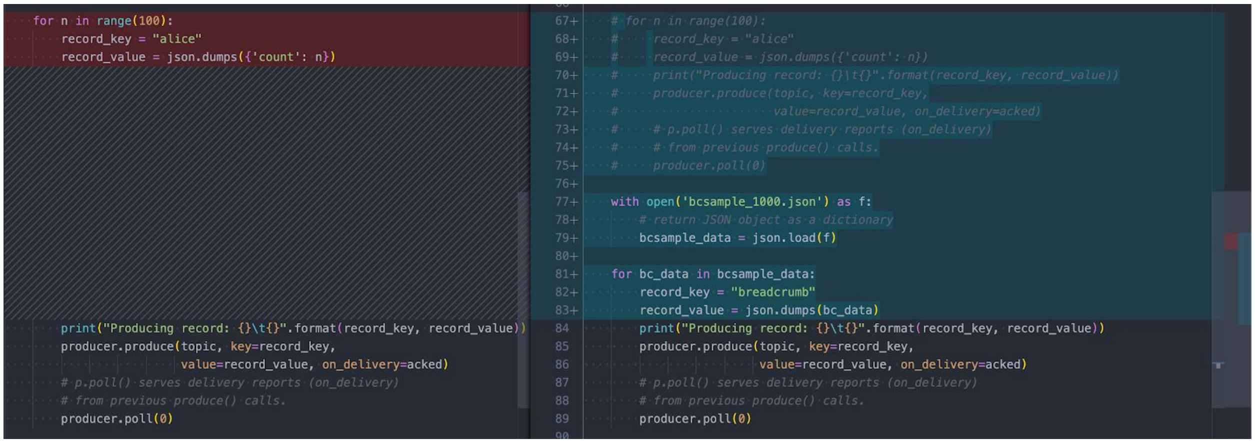
The goal for this week is to gain experience and knowledge of using a streaming data transport system (Kafka). Complete as many of the following exercises as you can. Proceed at a pace that allows you to learn and understand the use of Kafka with python.

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.

   I have created bcsample.json by extracting the first 1000 records from the actual breadcrumbs file. I used a mini python script (found in datagen.py) for this and have checked in the code.

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

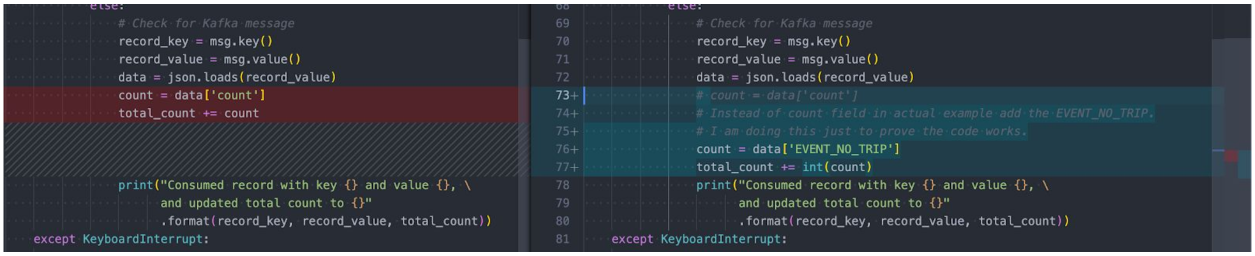


```
    for n in range(100):
        record_key = "alice"
        record_value = json.dumps({'count': n})
        print("Producing record: {}\t{}".format(record_key, record_value))
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(0)
```

```
    # for n in range(100):
    #     record_key = "alice"
    #     record_value = json.dumps({'count': n})
    #     print("Producing record: {}\t{}".format(record_key, record_value))
    #     producer.produce(topic, key=record_key,
    #                      value=record_value, on_delivery=acked)
    #     # p.poll() serves delivery reports (on_delivery)
    #     # from previous produce() calls.
    #     producer.poll(0)

    with open('bcsample_1000.json') as f:
        # return JSON object as a dictionary
        bcsample_data = json.load(f)

    for bc_data in bcsample_data:
        record_key = "breadcrumb"
        record_value = json.dumps(bc_data)
        print("Producing record: {}\t{}".format(record_key, record_value))
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(0)
```

I updated existing parser.py to send the records sequentially one at a time.

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



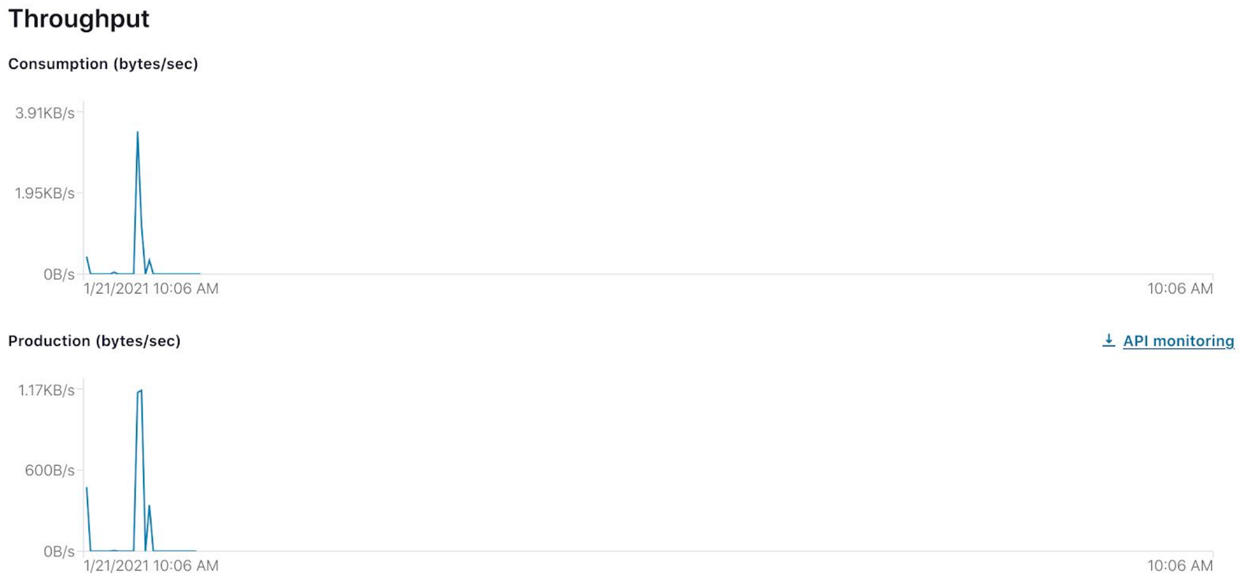
```
            else:
                # Check for Kafka message
                record_key = msg.key()
                record_value = msg.value()
                data = json.loads(record_value)
                count = data['count']
                total_count += count
                print("Consumed record with key {} and value {}, \
                      and updated total count to {}"
                      .format(record_key, record_value, total_count))
    except KeyboardInterrupt:
```

```
            else:
                # Check for Kafka message
                record_key = msg.key()
                record_value = msg.value()
                data = json.loads(record_value)
                # count = data['count']
                # Instead of count field in actual example add the EVENT_NO_TRIP.
                # I am doing this just to prove the code works.
                count = data['EVENT_NO_TRIP']
                total_count += int(count)
                print("Consumed record with key {} and value {}, \
                      and updated total count to {}"
                      .format(record_key, record_value, total_count))
    except KeyboardInterrupt:
```

In Consumer.py, the data['count'] parsing needs to be modified to some field in breadcrumb record. I randomly picked EVENT_NO_TRIP. And just to ensure my code works, I typecast EVENT_NO_TRIP field, from string to int and accumulated to value in the total_count variable.

B. Kafka Monitoring

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

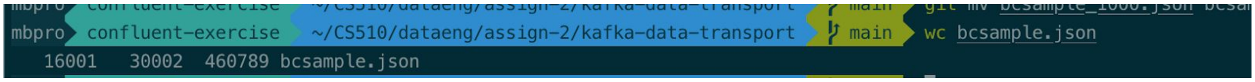


Kafka monitoring console provides a real time data flow graph to observe the throughput of both producer and consumer.

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.

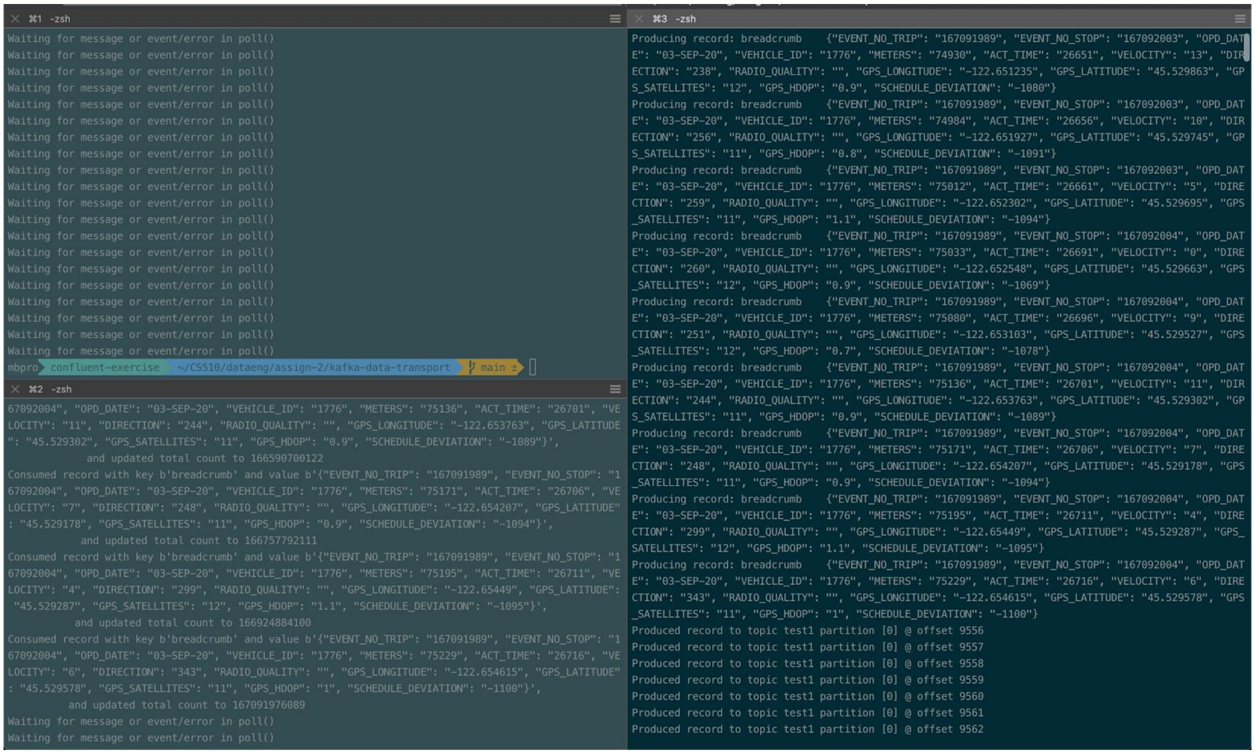


```
mbpro  confluent-exercise  ~/CS510/dataeng/assign-2/kafka-data-transport  main  git mv bcsample_1000.json bcsa
mbpro  confluent-exercise  ~/CS510/dataeng/assign-2/kafka-data-transport  main  wc bcsample.json
   16001   30002  460789 bcsample.json
```

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

When multiple consumers are running and only one producer sends data, then only one consumer consumes all the data and the other consumer just continues its polling job.

In the picture below, the terminals on left are running consumers and the terminal on right produces data.



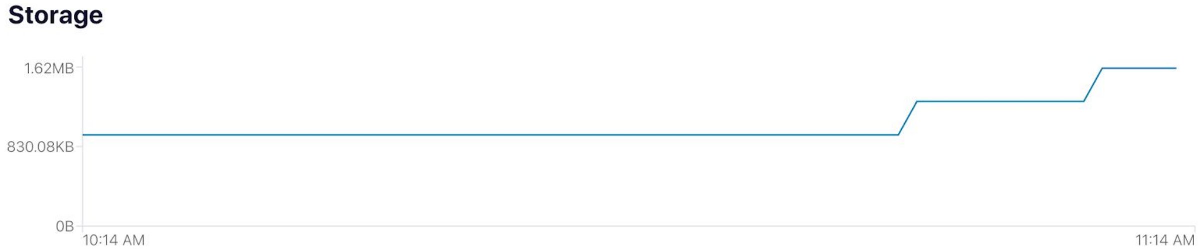
```
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
Waiting for message or event/error 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()
mbpro  confluent-exercise  ~/CS510/dataeng/assign-2/kafka-data-transport  main ±
```

```
67092004", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75136", "ACT_TIME": "26701", "VE
LOCITY": "11", "DIRECTION": "244", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.653763", "GPS_LATITUDE
": "45.529302", "GPS_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1089"}',
             and updated total count to 166590700122
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "1
67092004", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75171", "ACT_TIME": "26706", "VE
LOCITY": "7", "DIRECTION": "248", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.654207", "GPS_LATITUDE"
: "45.529178", "GPS_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1094"}',
            and updated total count to 166757792111
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "1
67092004", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75195", "ACT_TIME": "26711", "VE
LOCITY": "4", "DIRECTION": "299", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.65449", "GPS_LATITUDE":
 "45.529287", "GPS_SATELLITES": "12", "GPS_HDOP": "1.1", "SCHEDULE_DEVIATION": "-1095"}',
           and updated total count to 166924884100
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "1
67092004", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75229", "ACT_TIME": "26716", "VE
LOCITY": "6", "DIRECTION": "343", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.654615", "GPS_LATITUDE"
: "45.529578", "GPS_SATELLITES": "11", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "-1100"}',
          and updated total count to 167091976089
Waiting for message or event/error in poll()
Waiting for message or event/error in poll()
```

```
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092003", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "74930", "ACT_TIME": "26651", "VELOCITY": "13", "DIR
ECTION": "238", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.651235", "GPS_LATITUDE": "45.529863", "GP
S_SATELLITES": "12", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1080"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092003", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "74984", "ACT_TIME": "26656", "VELOCITY": "10", "DIR
ECTION": "256", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.651927", "GPS_LATITUDE": "45.529745", "GP
S_SATELLITES": "11", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "-1091"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092003", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75012", "ACT_TIME": "26661", "VELOCITY": "5", "DIRE
CTION": "259", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.652302", "GPS_LATITUDE": "45.529695", "GPS
_SATELLITES": "11", "GPS_HDOP": "1.1", "SCHEDULE_DEVIATION": "-1094"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75033", "ACT_TIME": "26691", "VELOCITY": "0", "DIRE
CTION": "260", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.652548", "GPS_LATITUDE": "45.529663", "GPS
_SATELLITES": "12", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1069"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75080", "ACT_TIME": "26696", "VELOCITY": "9", "DIRE
CTION": "251", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.653103", "GPS_LATITUDE": "45.529527", "GPS
_SATELLITES": "12", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": "-1078"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75136", "ACT_TIME": "26701", "VELOCITY": "11", "DIR
ECTION": "244", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.653763", "GPS_LATITUDE": "45.529302", "GP
S_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1089"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75171", "ACT_TIME": "26706", "VELOCITY": "7", "DIRE
CTION": "248", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.654207", "GPS_LATITUDE": "45.529178", "GPS
_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "-1094"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75195", "ACT_TIME": "26711", "VELOCITY": "4", "DIRE
CTION": "299", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.65449", "GPS_LATITUDE": "45.529287", "GPS_
SATELLITES": "12", "GPS_HDOP": "1.1", "SCHEDULE_DEVIATION": "-1095"}
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091989", "EVENT_NO_STOP": "167092004", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "75229", "ACT_TIME": "26716", "VELOCITY": "6", "DIRE
CTION": "343", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.654615", "GPS_LATITUDE": "45.529578", "GPS
_SATELLITES": "11", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "-1100"}
Produced record to topic test1 partition [0] @ offset 9556
Produced record to topic test1 partition [0] @ offset 9557
Produced record to topic test1 partition [0] @ offset 9558
Produced record to topic test1 partition [0] @ offset 9559
Produced record to topic test1 partition [0] @ offset 9560
Produced record to topic test1 partition [0] @ offset 9561
Produced record to topic test1 partition [0] @ offset 9562
```

3. Before the consumer runs, where might the data go, where might it be stored?
   The data is stored in a queue inside the kafka cloud.

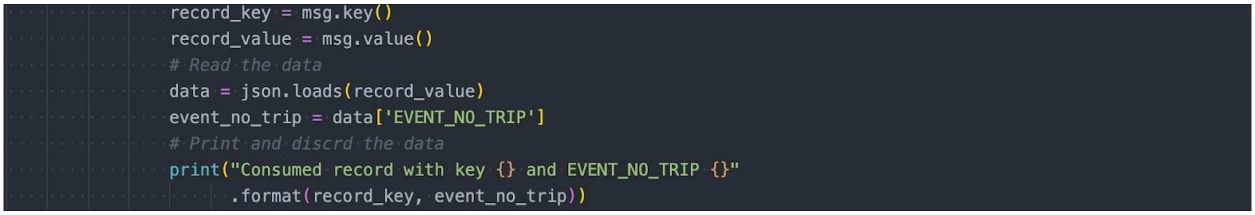
4. Is there a way to determine how much data Kafka/Confluent is storing for your topic? Do the Confluent monitoring tools help with this?



   Yes, we can see the data storage metric as a graph.

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.

I created a file named topic_clean.py by cloning the existing consumer file and changed the code as below.



```
            record_key = msg.key()
            record_value = msg.value()
            # Read the data
            data = json.loads(record_value)
            event_no_trip = data['EVENT_NO_TRIP']
            # Print and discrd the data
            print("Consumed record with key {} and EVENT_NO_TRIP {}"
                  .format(record_key, event_no_trip))
```

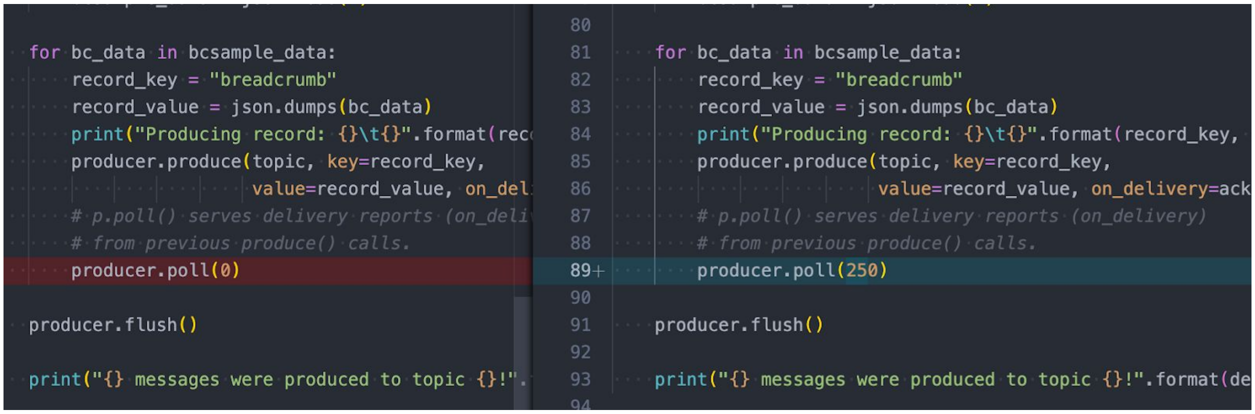
D. Multiple Producers

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.

Both the producers successfully send the data to kafka. When a consumer is started the same consumer consumes all the data irrespective of the source producer.

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.

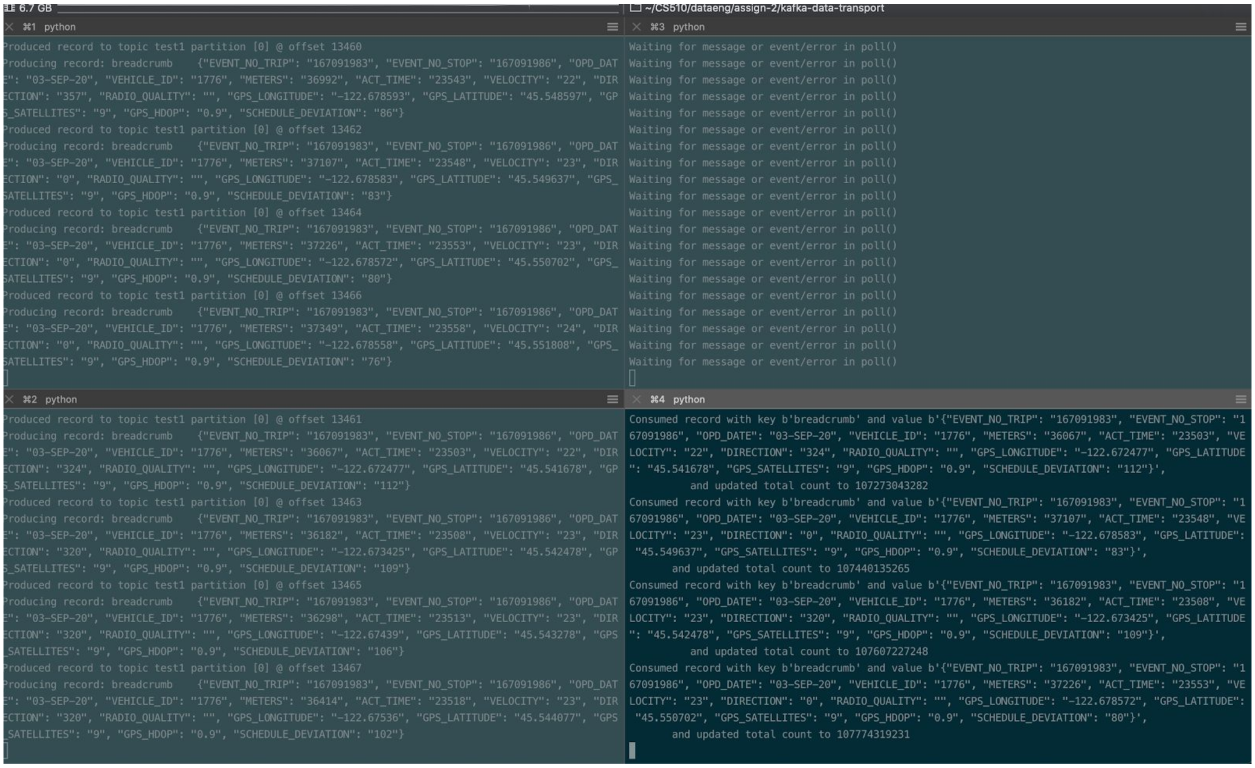


```
for bc_data in bcsample_data:
    record_key = "breadcrumb"
    record_value = json.dumps(bc_data)
    print("Producing record: {}\t{}".format(rec
    producer.produce(topic, key=record_key,
                     value=record_value, on_del
    # p.poll() serves delivery reports (on_deli
    # from previous produce() calls.
    producer.poll(0)

producer.flush()

print("{} messages were produced to topic {}!".
```

```
    for bc_data in bcsample_data:
        record_key = "breadcrumb"
        record_value = json.dumps(bc_data)
        print("Producing record: {}\t{}".format(record_key,
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=ack
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(250)

    producer.flush()

    print("{} messages were produced to topic {}!".format(de
```

3. Run two or three concurrent producers and two concurrent consumers all at the same time.
4. Describe the results.

In the below image, the left 2 terminals are producers running at 250 ms sleep. And the right 2 terminals are consumers. It is observed that only one consumer receives all the data and the other consumer simply waits and watch.



```
Produced record to topic test1 partition [0] @ offset 13460
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36992", "ACT_TIME": "23543", "VELOCITY": "22", "DIR
ECTION": "357", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678593", "GPS_LATITUDE": "45.548597", "GP
S_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "86"}
Produced record to topic test1 partition [0] @ offset 13462
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "37107", "ACT_TIME": "23548", "VELOCITY": "23", "DIR
ECTION": "0", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678583", "GPS_LATITUDE": "45.549637", "GPS_
SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "83"}
Produced record to topic test1 partition [0] @ offset 13464
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "37226", "ACT_TIME": "23553", "VELOCITY": "23", "DIR
ECTION": "0", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678572", "GPS_LATITUDE": "45.550702", "GPS_
SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "80"}
Produced record to topic test1 partition [0] @ offset 13466
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "37349", "ACT_TIME": "23558", "VELOCITY": "24", "DIR
ECTION": "0", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678558", "GPS_LATITUDE": "45.551808", "GPS_
SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "76"}
```

```
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()
```

```
Produced record to topic test1 partition [0] @ offset 13461
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36067", "ACT_TIME": "23503", "VELOCITY": "22", "DIR
ECTION": "324", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.672477", "GPS_LATITUDE": "45.541678", "GP
S_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "112"}
Produced record to topic test1 partition [0] @ offset 13463
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36182", "ACT_TIME": "23508", "VELOCITY": "23", "DIR
ECTION": "320", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.673425", "GPS_LATITUDE": "45.542478", "GP
S_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "109"}
Produced record to topic test1 partition [0] @ offset 13465
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36298", "ACT_TIME": "23513", "VELOCITY": "23", "DIR
ECTION": "320", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.67439", "GPS_LATITUDE": "45.543278", "GPS
_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "106"}
Produced record to topic test1 partition [0] @ offset 13467
Producing record: breadcrumb    {"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "167091986", "OPD_DAT
E": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36414", "ACT_TIME": "23518", "VELOCITY": "23", "DIR
ECTION": "320", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.67536", "GPS_LATITUDE": "45.544077", "GPS
_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "102"}
```

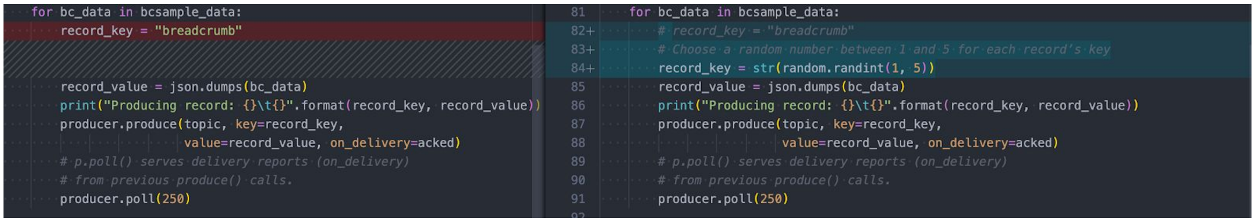
```
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "1
67091986", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36067", "ACT_TIME": "23503", "VE
LOCITY": "22", "DIRECTION": "324", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.672477", "GPS_LATITUDE
": "45.541678", "GPS_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "112"}',
          and updated total count to 107273043282
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "1
67091986", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "37107", "ACT_TIME": "23548", "VE
LOCITY": "23", "DIRECTION": "0", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678583", "GPS_LATITUDE":
 "45.549637", "GPS_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "83"}',
       and updated total count to 107440135265
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "1
67091986", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "36182", "ACT_TIME": "23508", "VE
LOCITY": "23", "DIRECTION": "320", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.673425", "GPS_LATITUDE
": "45.542478", "GPS_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "109"}',
          and updated total count to 107607227248
Consumed record with key b'breadcrumb' and value b'{"EVENT_NO_TRIP": "167091983", "EVENT_NO_STOP": "1
67091986", "OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "37226", "ACT_TIME": "23553", "VE
LOCITY": "23", "DIRECTION": "0", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.678572", "GPS_LATITUDE":
 "45.550702", "GPS_SATELLITES": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "80"}',
       and updated total count to 107774319231
```

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.

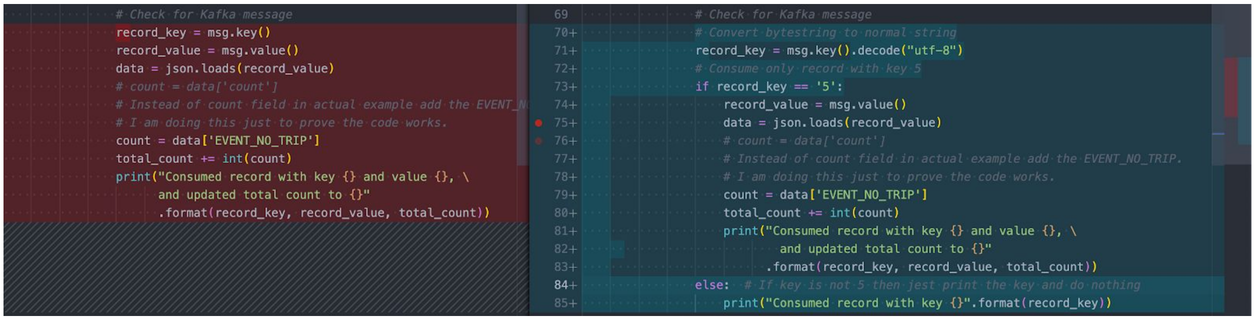
2. Update your producer code to choose a random number between 1 and 5 for each record’s key.



```python
    for bc_data in bcsample_data:
        record_key = "breadcrumb"

        record_value = json.dumps(bc_data)
        print("Producing record: {}\t{}".format(record_key, record_value))
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(250)
```

```python
    for bc_data in bcsample_data:
        # record_key = "breadcrumb"
        # Choose a random number between 1 and 5 for each record’s key
        record_key = str(random.randint(1, 5))
        record_value = json.dumps(bc_data)
        print("Producing record: {}\t{}".format(record_key, record_value))
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(250)
```

3. Modify your consumer to consume only records with a specific key (or subset of keys).

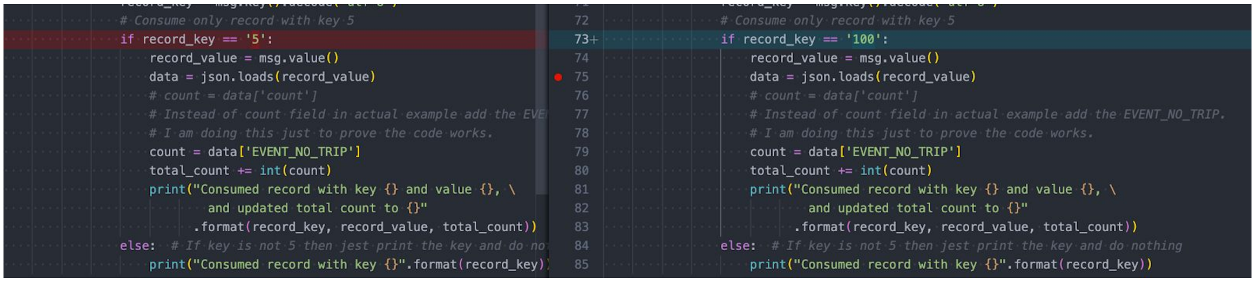


```
# Check for Kafka message
record_key = msg.key()
record_value = msg.value()
data = json.loads(record_value)
# count = data['count']
# Instead of count field in actual example add the EVENT_N
# I am doing this just to prove the code works.
count = data['EVENT_NO_TRIP']
total_count += int(count)
print("Consumed record with key {} and value {}, \
      and updated total count to {}"
     .format(record_key, record_value, total_count))
```

```
# Check for Kafka message
# Convert bytestring to normal string
record_key = msg.key().decode("utf-8")
# Consume only record with key 5
if record_key == '5':
    record_value = msg.value()
    data = json.loads(record_value)
    # count = data['count']
    # Instead of count field in actual example add the EVENT_NO_TRIP.
    # I am doing this just to prove the code works.
    count = data['EVENT_NO_TRIP']
    total_count += int(count)
    print("Consumed record with key {} and value {}, \
            and updated total count to {}"
          .format(record_key, record_value, total_count))
else:  # If key is not 5 then jest print the key and do nothing
    print("Consumed record with key {}".format(record_key))
```

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

When I change the code to consume records with a key that does not exist, the consumer silently ignores all the records and waits for the next data.

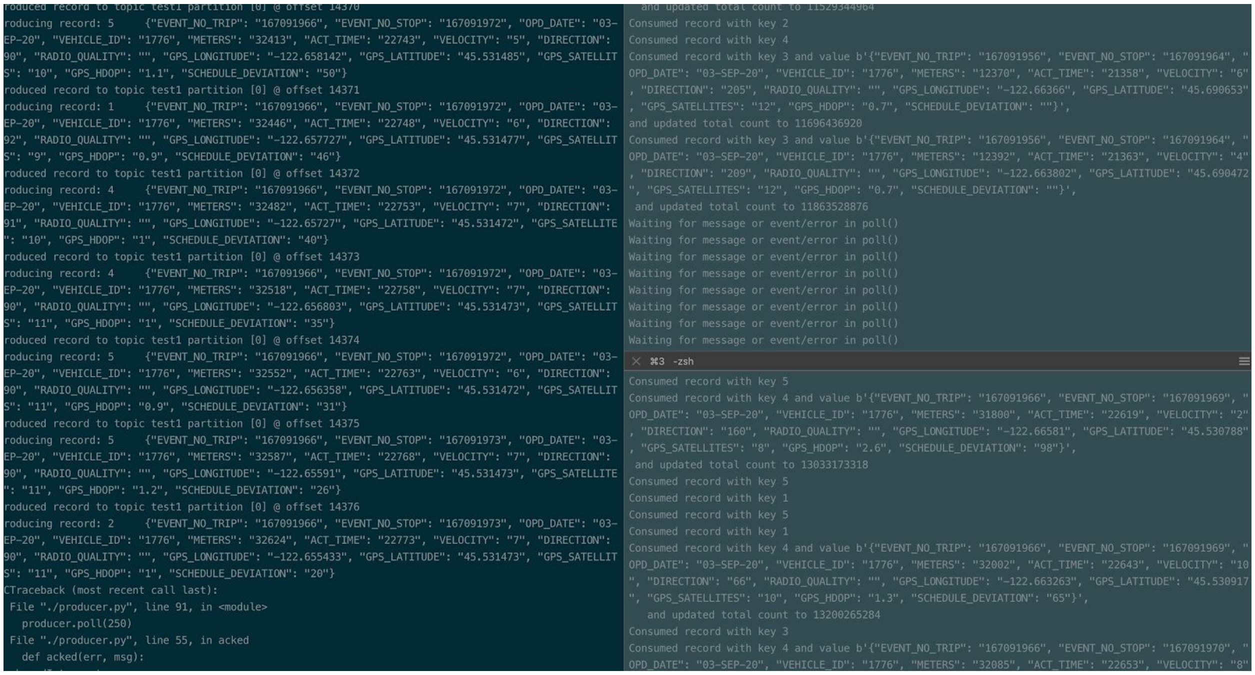


```
# Consume only record with key 5
if record_key == '5':
    record_value = msg.value()
    data = json.loads(record_value)
    # count = data['count']
    # Instead of count field in actual example add the EVE
    # I am doing this just to prove the code works.
    count = data['EVENT_NO_TRIP']
    total_count += int(count)
    print("Consumed record with key {} and value {}, \
            and updated total count to {}"
          .format(record_key, record_value, total_count))
else:  # If key is not 5 then jest print the key and do no
    print("Consumed record with key {}".format(record_key))
```

```
# Consume only record with key 5
if record_key == '100':
    record_value = msg.value()
    data = json.loads(record_value)
    # count = data['count']
    # Instead of count field in actual example add the EVENT_NO_TRIP.
    # I am doing this just to prove the code works.
    count = data['EVENT_NO_TRIP']
    total_count += int(count)
    print("Consumed record with key {} and value {}, \
            and updated total count to {}"
          .format(record_key, record_value, total_count))
else:  # If key is not 5 then jest print the key and do nothing
    print("Consumed record with key {}".format(record_key))
```

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?

Only one consumer consumes data while running multiple consumers. In the below screenshot, the first consumer (which consumes only odd number keys) stops consuming and goes to waiting mode, when the second consumer (which consumes only even number keys) is started.



```
roduced record to topic test1 partition [0] @ offset 14370
roducing record: 5     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091972", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32413", "ACT_TIME": "22743", "VELOCITY": "5", "DIRECTION":
90", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.658142", "GPS_LATITUDE": "45.531485", "GPS_SATELLIT
S": "10", "GPS_HDOP": "1.1", "SCHEDULE_DEVIATION": "50"}
roduced record to topic test1 partition [0] @ offset 14371
roducing record: 1     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091972", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32446", "ACT_TIME": "22748", "VELOCITY": "6", "DIRECTION":
92", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.657727", "GPS_LATITUDE": "45.531477", "GPS_SATELLIT
S": "9", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "46"}
roduced record to topic test1 partition [0] @ offset 14372
roducing record: 4     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091972", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32482", "ACT_TIME": "22753", "VELOCITY": "7", "DIRECTION":
91", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.65727", "GPS_LATITUDE": "45.531472", "GPS_SATELLITE
": "10", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "40"}
roduced record to topic test1 partition [0] @ offset 14373
roducing record: 4     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091972", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32518", "ACT_TIME": "22758", "VELOCITY": "7", "DIRECTION":
90", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.656803", "GPS_LATITUDE": "45.531473", "GPS_SATELLIT
S": "11", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "35"}
roduced record to topic test1 partition [0] @ offset 14374
roducing record: 5     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091972", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32552", "ACT_TIME": "22763", "VELOCITY": "6", "DIRECTION":
90", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.656358", "GPS_LATITUDE": "45.531472", "GPS_SATELLIT
S": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "31"}
roduced record to topic test1 partition [0] @ offset 14375
roducing record: 5     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091973", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32587", "ACT_TIME": "22768", "VELOCITY": "7", "DIRECTION":
90", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.65591", "GPS_LATITUDE": "45.531473", "GPS_SATELLITE
": "11", "GPS_HDOP": "1.2", "SCHEDULE_DEVIATION": "26"}
roduced record to topic test1 partition [0] @ offset 14376
roducing record: 2     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091973", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "32624", "ACT_TIME": "22773", "VELOCITY": "7", "DIRECTION":
90", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.655433", "GPS_LATITUDE": "45.531473", "GPS_SATELLIT
S": "11", "GPS_HDOP": "1", "SCHEDULE_DEVIATION": "20"}
CTraceback (most recent call last):
 File "./producer.py", line 91, in <module>
   producer.poll(250)
 File "./producer.py", line 55, in acked
   def acked(err, msg):
```

```
  and updated total count to 11529344964
Consumed record with key 2
Consumed record with key 4
Consumed record with key 3 and value b'{"EVENT_NO_TRIP": "167091956", "EVENT_NO_STOP": "167091964", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "12370", "ACT_TIME": "21358", "VELOCITY": "6"
, "DIRECTION": "205", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66366", "GPS_LATITUDE": "45.690653"
, "GPS_SATELLITES": "12", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": ""}',
and updated total count to 11696436920
Consumed record with key 3 and value b'{"EVENT_NO_TRIP": "167091956", "EVENT_NO_STOP": "167091964", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "12392", "ACT_TIME": "21363", "VELOCITY": "4"
, "DIRECTION": "209", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.663802", "GPS_LATITUDE": "45.690472
", "GPS_SATELLITES": "12", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": ""}',
 and updated total count to 11863528876
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()
```

```
Consumed record with key 5
Consumed record with key 4 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091969", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "31800", "ACT_TIME": "22619", "VELOCITY": "2"
, "DIRECTION": "160", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66581", "GPS_LATITUDE": "45.530788"
, "GPS_SATELLITES": "8", "GPS_HDOP": "2.6", "SCHEDULE_DEVIATION": "98"}',
 and updated total count to 13033173318
Consumed record with key 5
Consumed record with key 1
Consumed record with key 5
Consumed record with key 1
Consumed record with key 4 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091969", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "32002", "ACT_TIME": "22643", "VELOCITY": "10
", "DIRECTION": "66", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.663263", "GPS_LATITUDE": "45.530917
", "GPS_SATELLITES": "10", "GPS_HDOP": "1.3", "SCHEDULE_DEVIATION": "65"}',
   and updated total count to 13200265284
Consumed record with key 3
Consumed record with key 4 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091970", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "32085", "ACT_TIME": "22653", "VELOCITY": "8"
```

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?
   Producer.flush() waits for all messages in the Producer queue to be delivered.

2. What happens if you do not call producer.flush()?
   It does not make the producer synchronous as it's unlikely the message just sent will already have reached the broker and a delivery report was already sent back to the client.

3. What happens if you call producer.flush() after sending each record?
   It will block until the previously sent messages have been delivered (or errored), effectively making the producer synchronous.

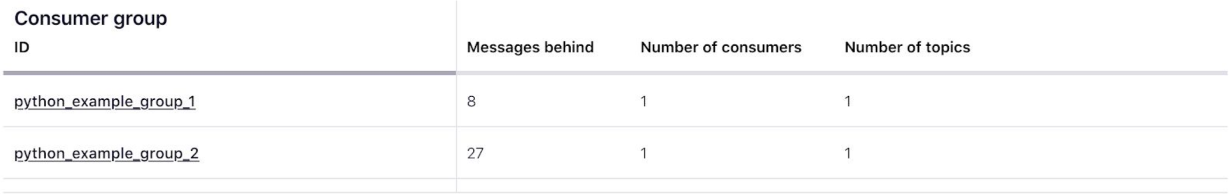
4. What happens if you wait for 2 seconds after every 5th record send, and you call flush only after every 15 record sends, and you have a consumer running concurrently? Specifically, does the consumer receive each message immediately? only after a flush? Something else?

Each time flush is executed with 15 records, consumer receives 3 batches of 5 records for every 2 sec. And these 3 batches are received asynchronously with each record inside the batch is received asynchronously.

H. Consumer Groups

1. Create two consumer groups with one consumer program instance in each group. Created a new consumer group with group id : python_example_group_2.

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.



Consumer group ID	Messages behind	Number of consumers	Number of topics
python_example_group_1	8	1	1
python_example_group_2	27	1	1

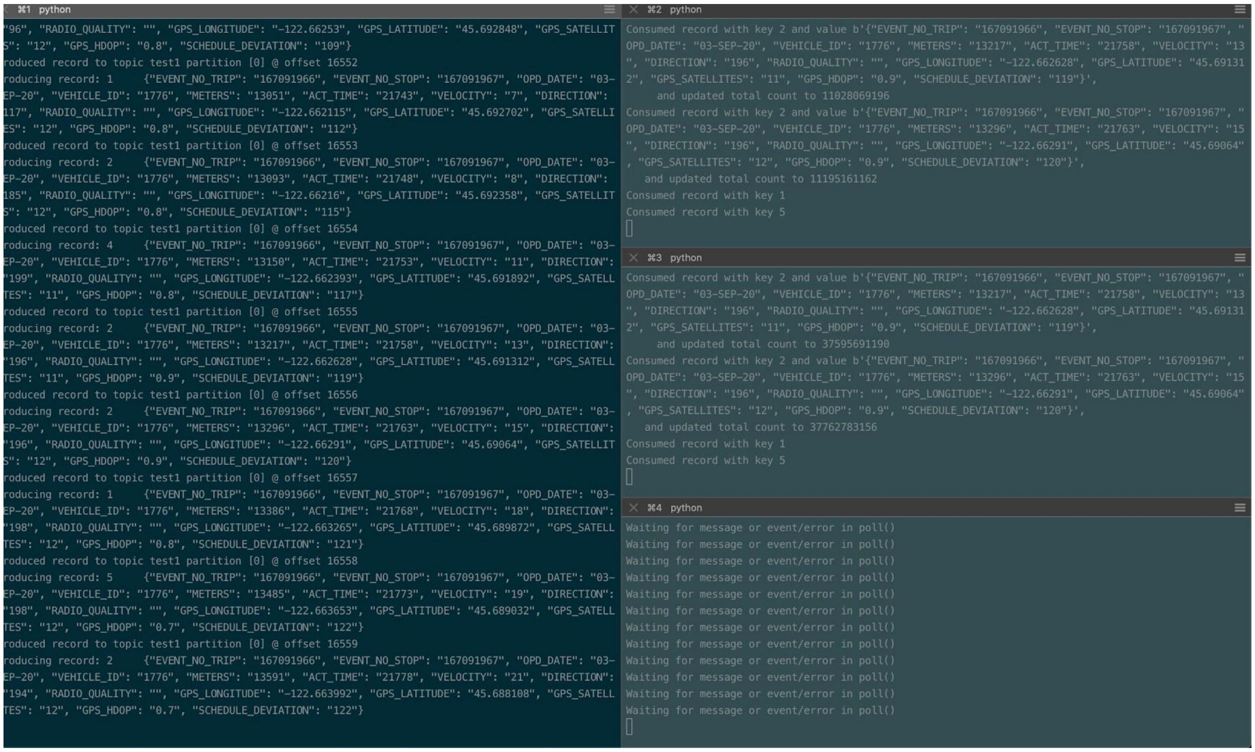
Could be seen that both the consumer instances are consuming messages.

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

   Created a consumer within group id : python_example_group_1.

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.

   It is observed that each consumer group consumes the full set of messages but when there are two consumers with the same topic, just one consumer processes all the messages.



```
"96", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66253", "GPS_LATITUDE": "45.692848", "GPS_SATELLIT
S": "12", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "109"}
roduced record to topic test1 partition [0] @ offset 16552
roducing record: 1     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13051", "ACT_TIME": "21743", "VELOCITY": "7", "DIRECTION":
117", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.662115", "GPS_LATITUDE": "45.692702", "GPS_SATELLI
ES": "12", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "112"}
roduced record to topic test1 partition [0] @ offset 16553
roducing record: 2     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13093", "ACT_TIME": "21748", "VELOCITY": "8", "DIRECTION":
185", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66216", "GPS_LATITUDE": "45.692358", "GPS_SATELLIT
S": "12", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "115"}
roduced record to topic test1 partition [0] @ offset 16554
roducing record: 4     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13150", "ACT_TIME": "21753", "VELOCITY": "11", "DIRECTION":
"199", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.662393", "GPS_LATITUDE": "45.691892", "GPS_SATELL
TES": "11", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "117"}
roduced record to topic test1 partition [0] @ offset 16555
roducing record: 2     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13217", "ACT_TIME": "21758", "VELOCITY": "13", "DIRECTION":
"196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.662628", "GPS_LATITUDE": "45.691312", "GPS_SATELL
TES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "119"}
roduced record to topic test1 partition [0] @ offset 16556
roducing record: 2     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13296", "ACT_TIME": "21763", "VELOCITY": "15", "DIRECTION":
"196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66291", "GPS_LATITUDE": "45.69064", "GPS_SATELLIT
S": "12", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "120"}
roduced record to topic test1 partition [0] @ offset 16557
roducing record: 1     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13386", "ACT_TIME": "21768", "VELOCITY": "18", "DIRECTION":
"198", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.663265", "GPS_LATITUDE": "45.689872", "GPS_SATELL
TES": "12", "GPS_HDOP": "0.8", "SCHEDULE_DEVIATION": "121"}
roduced record to topic test1 partition [0] @ offset 16558
roducing record: 5     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13485", "ACT_TIME": "21773", "VELOCITY": "19", "DIRECTION":
"198", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.663653", "GPS_LATITUDE": "45.689032", "GPS_SATELL
TES": "12", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": "122"}
roduced record to topic test1 partition [0] @ offset 16559
roducing record: 2     {"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "OPD_DATE": "03-
EP-20", "VEHICLE_ID": "1776", "METERS": "13591", "ACT_TIME": "21778", "VELOCITY": "21", "DIRECTION":
"194", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.663992", "GPS_LATITUDE": "45.688108", "GPS_SATELL
TES": "12", "GPS_HDOP": "0.7", "SCHEDULE_DEVIATION": "122"}
```

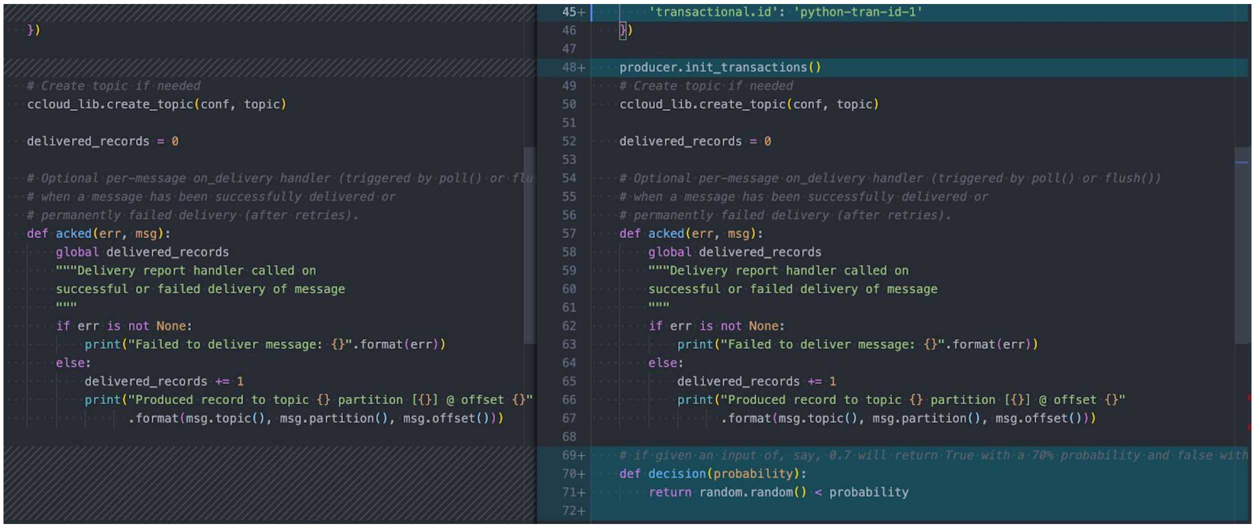
```
Consumed record with key 2 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "13217", "ACT_TIME": "21758", "VELOCITY": "13
", "DIRECTION": "196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.662628", "GPS_LATITUDE": "45.69131
2", "GPS_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "119"}',
     and updated total count to 11028069196
Consumed record with key 2 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "13296", "ACT_TIME": "21763", "VELOCITY": "15
", "DIRECTION": "196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66291", "GPS_LATITUDE": "45.69064"
, "GPS_SATELLITES": "12", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "120"}',
   and updated total count to 11195161162
Consumed record with key 1
Consumed record with key 5
```

```
Consumed record with key 2 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "13217", "ACT_TIME": "21758", "VELOCITY": "13
", "DIRECTION": "196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.662628", "GPS_LATITUDE": "45.69131
2", "GPS_SATELLITES": "11", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "119"}',
     and updated total count to 37595691190
Consumed record with key 2 and value b'{"EVENT_NO_TRIP": "167091966", "EVENT_NO_STOP": "167091967", "
OPD_DATE": "03-SEP-20", "VEHICLE_ID": "1776", "METERS": "13296", "ACT_TIME": "21763", "VELOCITY": "15
", "DIRECTION": "196", "RADIO_QUALITY": "", "GPS_LONGITUDE": "-122.66291", "GPS_LATITUDE": "45.69064"
, "GPS_SATELLITES": "12", "GPS_HDOP": "0.9", "SCHEDULE_DEVIATION": "120"}',
   and updated total count to 37762783156
Consumed record with key 1
Consumed record with key 5
```

```
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()
```

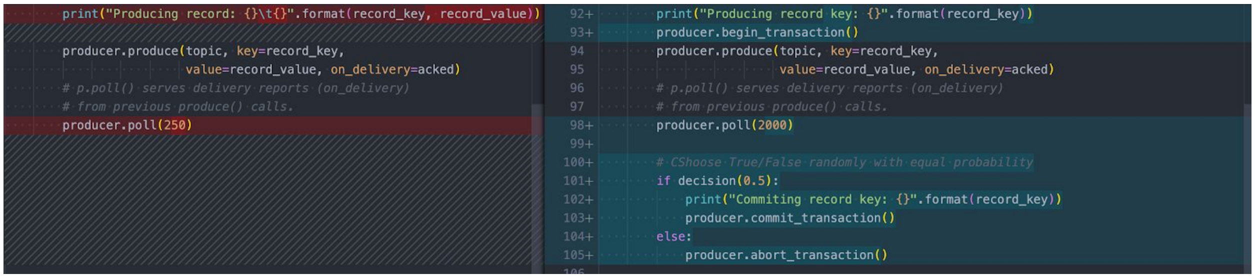
I. Kafka Transactions

6. Create a new producer, similar to the previous producer, that uses transactions.



```
  })

  # Create topic if needed
  ccloud_lib.create_topic(conf, topic)

  delivered_records = 0

  # Optional per-message on_delivery handler (triggered by poll() or flu
  # when a message has been successfully delivered or
  # permanently failed delivery (after retries).
  def acked(err, msg):
      global delivered_records
      """Delivery report handler called on
      successful or failed delivery of message
      """
      if err is not None:
          print("Failed to deliver message: {}".format(err))
      else:
          delivered_records += 1
          print("Produced record to topic {} partition [{}] @ offset {}"
                .format(msg.topic(), msg.partition(), msg.offset()))
```

```
        'transactional.id': 'python-tran-id-1'
    })

    producer.init_transactions()
    # Create topic if needed
    ccloud_lib.create_topic(conf, topic)

    delivered_records = 0

    # Optional per-message on_delivery handler (triggered by poll() or flush())
    # when a message has been successfully delivered or
    # permanently failed delivery (after retries).
    def acked(err, msg):
        global delivered_records
        """Delivery report handler called on
        successful or failed delivery of message
        """
        if err is not None:
            print("Failed to deliver message: {}".format(err))
        else:
            delivered_records += 1
            print("Produced record to topic {} partition [{}] @ offset {}"
                  .format(msg.topic(), msg.partition(), msg.offset()))

    # if given an input of, say, 0.7 will return True with a 70% probability and false with
    def decision(probability):
        return random.random() < probability

```

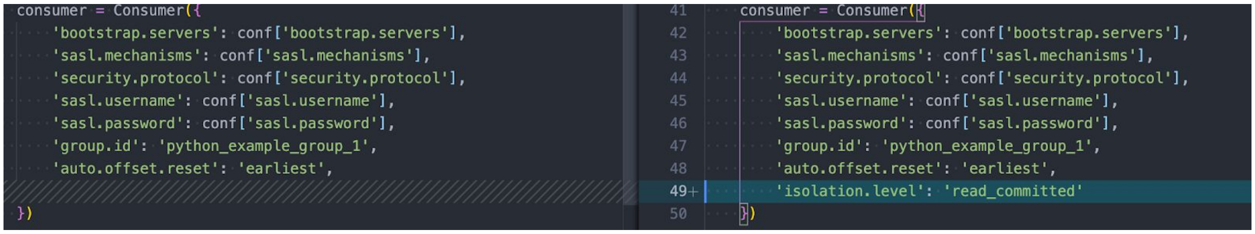


```
        print("Producing record: {}\t{}".format(record_key, record_value))

        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(250)
```

```
        print("Producing record key: {}".format(record_key))
        producer.begin_transaction()
        producer.produce(topic, key=record_key,
                         value=record_value, on_delivery=acked)
        # p.poll() serves delivery reports (on_delivery)
        # from previous produce() calls.
        producer.poll(2000)

        # CShoose True/False randomly with equal probability
        if decision(0.5):
            print("Commiting record key: {}".format(record_key))
            producer.commit_transaction()
        else:
            producer.abort_transaction()
```

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).



```
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',
})
```

```
    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'
    })
```

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.