SOFE4630 Cloud Computing (Winter 2022 - Dr. M. El-darieby)

Lab 2: Project Milestone-- Data Ingestion Software-- Kafka Clusters

Final Submission

Date: 02/03/2022

Student Names:

Alexander Pitcher (100693749)

Mitul Patel (100700131)

Clarissa Branje (100716458)

Amin Khakpour(100669547)

Objective:

- **Understand Event-Driven Architecture (EDA).**
- **Get familiar with Kafka concepts and APIs.**
- **Using Kafka within Google Cloud Platform (GCP).**

Video 1 NodeJS Kafka:
https://drive.google.com/file/d/1CFDJ41La1-a7A2hdB-nT8V6R-l8VpGJZ/view?usp=sharing

Video 2 Python Kafka:
https://drive.google.com/file/d/1AacNqGMLiEC6S2tdpeC5RAjASNaS0uM3/view?usp=sharing

Video 3 Confluent Kafka CLI:
https://drive.google.com/file/d/1ENZKL2dkLrbL_kjgFKu-lypxsBlk27kG/view?usp=sharing

Video 4 Confluent Python Connection:

https://drive.google.com/file/d/1TgA_P6BU149moDn3nTyP1Z6N9F6I2Ndf/view?usp=sharing

What is EDA?

- An event driven architecture is a software architecture model for making a system which reacts to a particular event.
- An event is a significant change in state.
- When a state changes of a system, a reaction happens which simulates some action

Advantages

- Processing Streaming Data in real time
    - This allows for speed processing of large amount of data and perform an action in real time
- Reduced Operation Costs
    - This allows business to achieve predictable response times and processed scalability
- Scalability
    - EDA is usually an event-handling architecture which scales with the number of entities
- Enhanced Customer Experience Response
    - Allows to do real time analysis on large scale applications.

Disadvantages

- Handling Schema changes
    - Adds complexity to the overall system
    - Becomes increasingly expensive
- Dealing with complex, real world domains
    - Deals with issue regarding the problem of explanation fatigue
    - Too difficult to explain the system
- Visibility of Data
    - Data intensive applications usually cause by data anomalies rather than code

In Kafka, what's meant by cluster, broker, topic, replica, partition, zookeeper, controller, leader, consumer, producer, and consumer group?

Cluster: one or more than one server referred to as Kafka brokers running kafka.

Broker: A Kakfa server which runs in a kafka cluster. Multiple brokers form a cluster.

Topic: Defined as a virtual group which comprised of one or more partitions across kafka brokers

Replica: one replica is a leader while others are followers. If a leader is down, an election is called to make a new leader. The idea is to replicate logs in order to n servers.

Partition: Kafka topics are divided into several partitions

Zookeeper: Responsible for keeping track of the status in the kafka cluster nodes

Controller: It is a service which runs on every broker in a cluster

Leader: There is a leader in a cluster of nodes which is responsible for all read and write requests for a partition.

Consumer: Usually referred to as a subscriber which are subscribed to a topic

Producer: Usually referred to as a publisher which pushes information to certain topics.

Consumer Group: A group of subscribers which are subscribed to a topic

Follow the following video to install Kafka into your local machine and create topics, consumers, and producers using Kafka's built-in tools

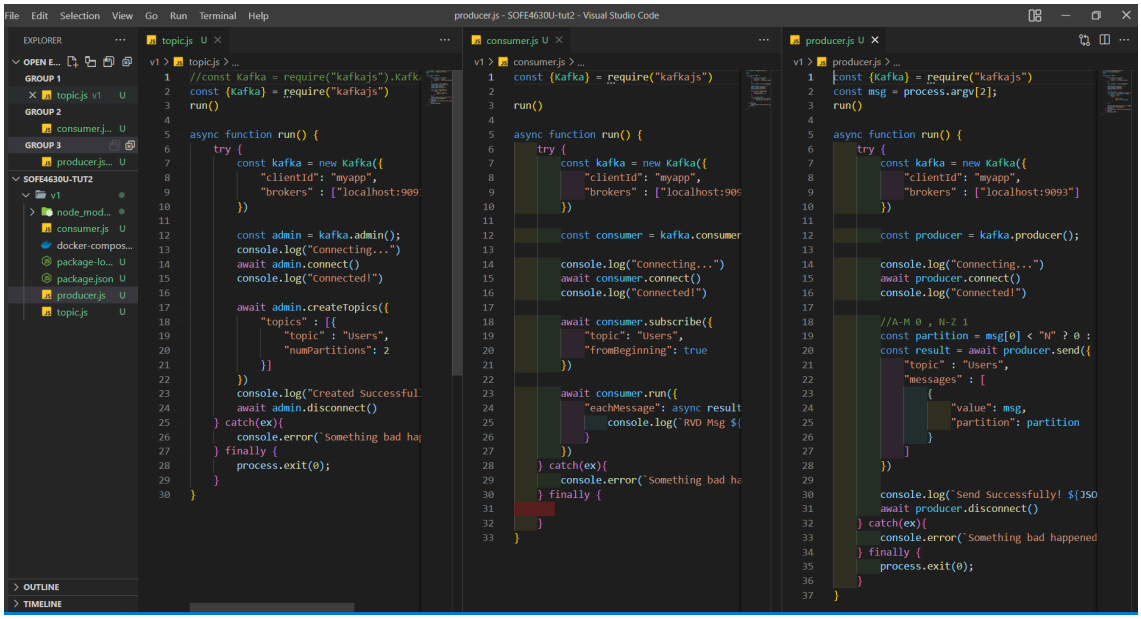
5. Follow the following video to install Kafka into your local machine and create topics, consumers, and producers using Kafka's built-in tools.

```
patel@DESKTOP-DGRS3IB MINGW64 ~/OneDrive/Documents/Year IV/Semester 2/Cl
$ docker exec broker1 kafka-topics --create --topic topic3 --partitions
er broker1:9092,broker2:9092,broker3:9092
Error while executing topic command : Timed out waiting for a node assig
$ docker exec broker1 kafka-topics --describe --bootstrap-server broker1
:9092
Topic: topic1   TopicId: 6jS6HF2-QOu44wh3Y3uu7g PartitionCount: 3
ReplicationFactor: 3    Configs:
        Topic: topic1   Partition: 0    Leader: 3       Replicas: 3,2,1
Isr: 3,2,1
        Topic: topic1   Partition: 1    Leader: 1       Replicas: 1,3,2
titionCount: 3       ReplicationFactor: 3    Configs:
        Topic: topic    Partition: 0    Leader: 3       Replicas: 3,1,2 Isr: 3,1,2
        Topic: topic    Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1
        Topic: topic    Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
Topic: topic2   TopicId: mE2vCOIPRN23E87TbFsvSQ PartitionCount: 3       ReplicationFactor: 2    Configs:
        Topic: topic2   Partition: 0    Leader: none    Replicas: 2,3   Isr: 3
        Topic: topic2   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1
        Topic: topic2   Partition: 2    Leader: 1       Replicas: 1,2   Isr: 1
```

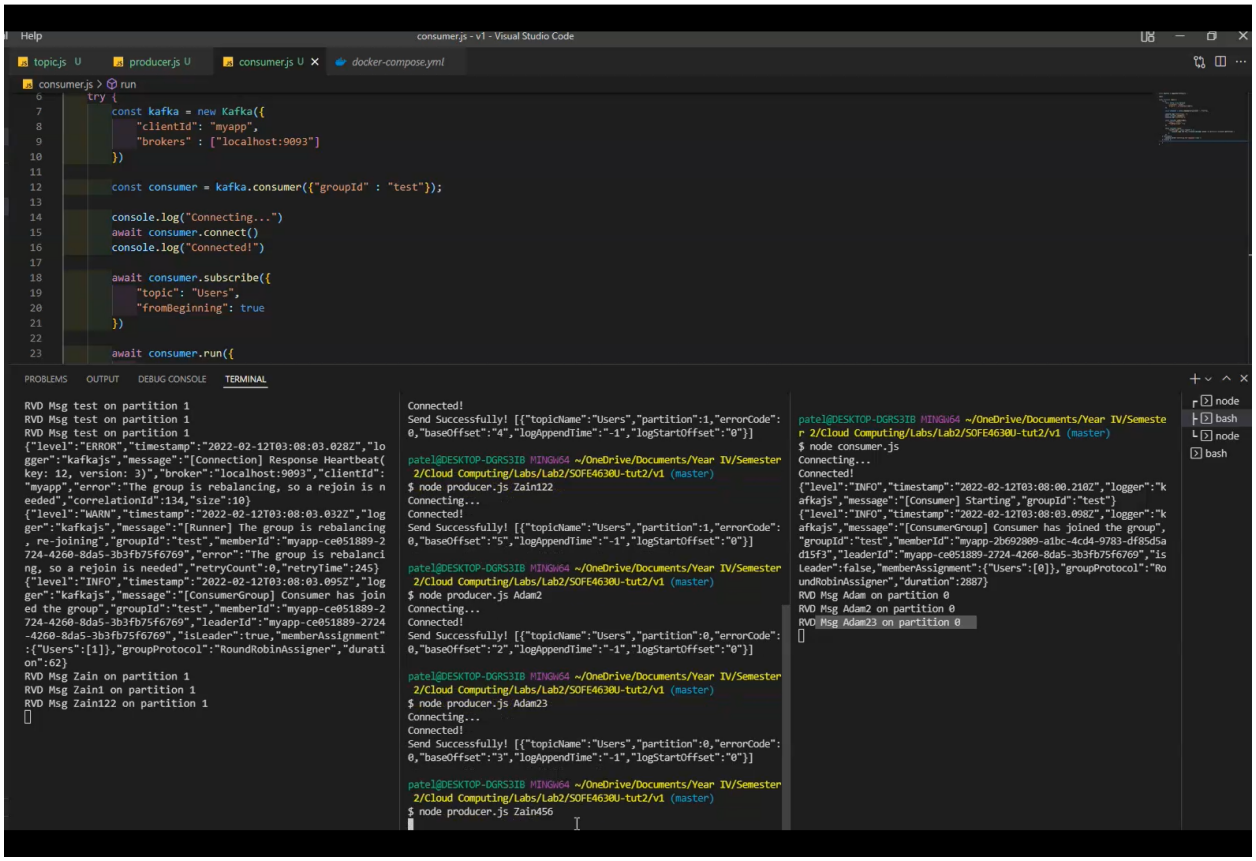
```
$ docker exec broker1 kafka-topics --list --bootstrap-server broker1:9092
topic
topic1
topic2
```

5. Follow the following video to generate NodeJS scripts for creating topics, consumers, and producers

Video Link: https://drive.google.com/file/d/1CFDJ41La1-a7A2hdB-nT8V6R-l8VpGJZ/view?usp=sharing



Output



6. Using the python library Kafka-python, write three python scripts that

Video Link:
https://drive.google.com/file/d/1AacNqGMLiEC6S2tdpeC5RAjASNaS0uM3/view?usp=sharing

consumer.py | producer.py | topic.py

```python
import random
import string


user_ids = list(range(1, 101))
recipient_ids = list(range(1, 101))

def generate_message() -> dict:
    random_user_id = random.choice(user_ids)

    # Copy the recipients array
    recipient_ids_copy = recipient_ids.copy()

    # User can't send message to himself
    recipient_ids_copy.remove(random_user_id)
    random_recipient_id = random.choice(recipient_ids_copy)

    # Generate a random message
    message = ''.join(random.choice(string.ascii_letters) for

    return {
        'user_id': random_user_id,
        'recipient_id': random_recipient_id,
        'message': message
    }

if __name__ == '__main__':
    print(generate_message())
```

consumer.py | producer.py | topic.py

```python
from datetime import datetime
from topic import generate_message
from kafka import KafkaProducer


# Messages will be serialized as JSON
def serializer(message):
    return json.dumps(message).encode('utf-8')


# Kafka Producer
producer = KafkaProducer(
    bootstrap_servers=['localhost:9093'],
    value_serializer=serializer
)

if __name__ == '__main__':
    # Infinite loop - runs until you kill the program
    while True:
        # Generate a message
        dummy_message = generate_message()

        # Send it to our 'messages' topic
        print(f'Producing message @ {datetime.now()}
        producer.send('messages', dummy_message)

        # Sleep for a random number of seconds
        time_to_sleep = random.randint(1, 11)
        time.sleep(time_to_sleep)
```

consumer.py | producer.py | topic.py

```python
import json


from kafka import KafkaConsumer



if __name__ == '__main__':
    # Kafka Consumer
    consumer = KafkaConsumer(
        'messages',
        bootstrap_servers='localhost:9093',
        auto_offset_reset='earliest'
    )
    for message in consumer:
        print(json.loads(message.value))
```

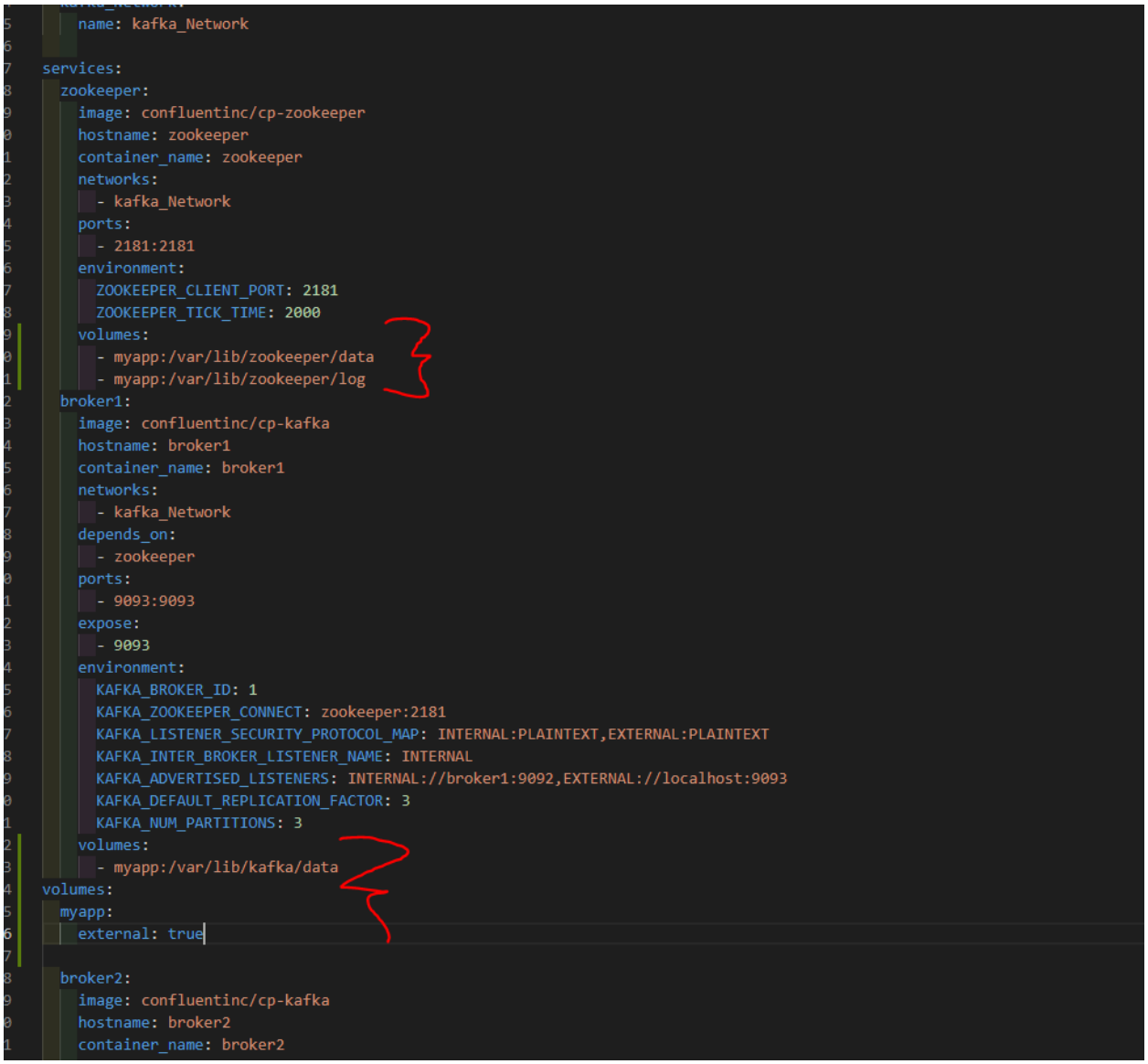
8. A problem in the used YAML file to create the docker images is that the data inside Kafka clusters are not persistent which means if the docker images are down, all its messages are lost. Update the YAML file for persistent data (hint: it's related to the volume options in Kafka brokers and zookeeper). Describe how this update solves the problem.

Kafka uses volumes for logging data and Zookeeper uses volumes for transaction logs. It is ideal to separate volumes on the host for these services. In order to map, you need to declare the full path. First you need to create all the directories for data mapping and give the read and write permissions.

```
# Create dirs for Kafka / ZK data.
mkdir -p /vol1/zk-data
mkdir -p /vol2/zk-txn-logs
mkdir -p /vol3/kafka-data

# Make sure the user has the read and write permissions.
chown -R 1000:1000 /vol1/zk-data
chown -R 1000:1000 /vol2/zk-txn-logs
chown -R 1000:1000 /vol3/kafka-data
```

Once this is done you can mount the volumes using the -v flag.



```
    name: kafka_Network

services:
  zookeeper:
    image: confluentinc/cp-zookeeper
    hostname: zookeeper
    container_name: zookeeper
    networks:
      - kafka_Network
    ports:
      - 2181:2181
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    volumes:
      - myapp:/var/lib/zookeeper/data
      - myapp:/var/lib/zookeeper/log
  broker1:
    image: confluentinc/cp-kafka
    hostname: broker1
    container_name: broker1
    networks:
      - kafka_Network
    depends_on:
      - zookeeper
    ports:
      - 9093:9093
    expose:
      - 9093
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://broker1:9092,EXTERNAL://localhost:9093
      KAFKA_DEFAULT_REPLICATION_FACTOR: 3
      KAFKA_NUM_PARTITIONS: 3
    volumes:
      - myapp:/var/lib/kafka/data
volumes:
  myapp:
    external: true

  broker2:
    image: confluentinc/cp-kafka
    hostname: broker2
    container_name: broker2
```

This update solves the problem by holding the data in a place where it would stay persisted. Adding the option will ensure that data inside kafka cluster are not lost

9. **Follow the following video about Kafka in Confluent Cloud CLI**

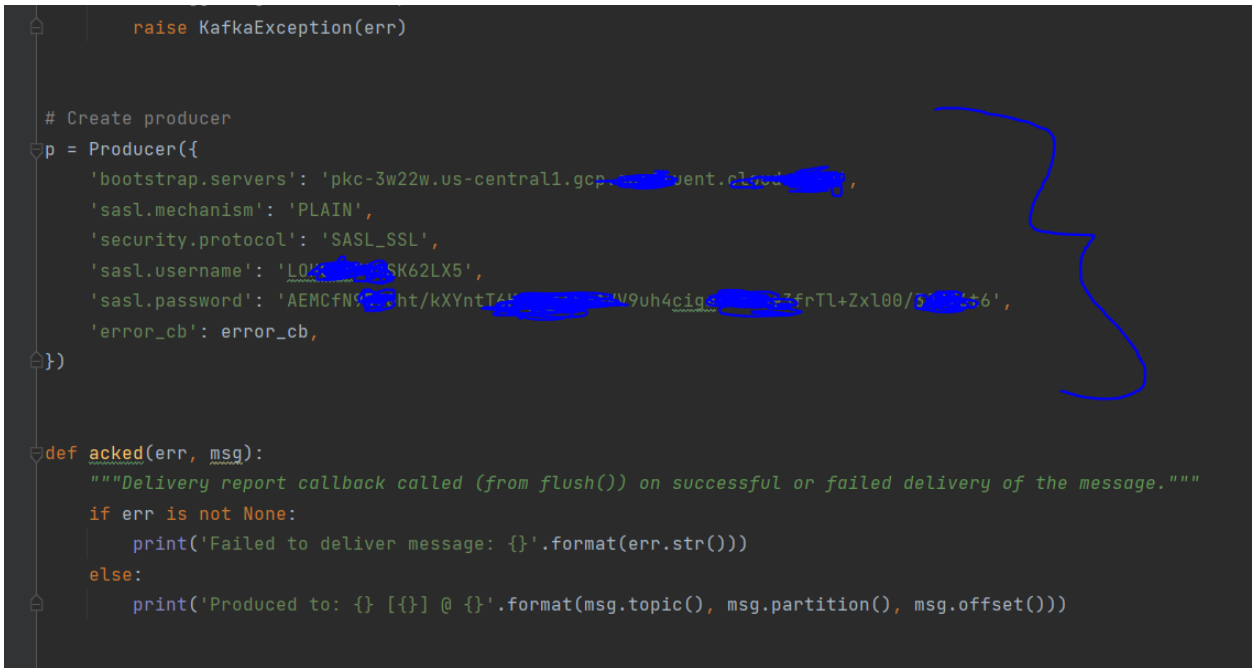
Video:
https://drive.google.com/file/d/1ENZKL2dkLrbL_kjgFKu-lypxsBlk27kG/view?usp=sharing

Connecting python to Confluent Cloud

Video:

https://drive.google.com/file/d/1TgA_P6BU149moDn3nTyP1Z6N9F6l2Ndf/view?usp=sharing

Confluent Cloud; Python Client Consumer/Producer connection



```
        raise KafkaException(err)


# Create producer
p = Producer({
    'bootstrap.servers': 'pkc-3w22w.us-central1.gcp.[redacted]',
    'sasl.mechanism': 'PLAIN',
    'security.protocol': 'SASL_SSL',
    'sasl.username': 'LO[redacted]SK62LX5',
    'sasl.password': 'AEMCfN9[redacted]ht/kXYntT4[redacted]W9uh4cig[redacted]frTl+Zxl00/5[redacted]',
    'error_cb': error_cb,
})


def acked(err, msg):
    """Delivery report callback called (from flush()) on successful or failed delivery of the message."""
    if err is not None:
        print('Failed to deliver message: {}'.format(err.str()))
    else:
        print('Produced to: {} [{}] @ {}'.format(msg.topic(), msg.partition(), msg.offset()))
```

```
C:\Users\patel\PycharmProjects\kafka\venv\Scripts\python.exe C:/Users/patel/PycharmProjects/kafka/confluent_cloud.py
Produced to: test-topic [5] @ 0
Produced to: test-topic [0] @ 1
Produced to: test-topic [4] @ 9
Produced to: test-topic [4] @ 10
Produced to: test-topic [2] @ 1
Produced to: test-topic [2] @ 2
Produced to: test-topic [2] @ 3
Produced to: test-topic [2] @ 4
Produced to: test-topic [2] @ 5
Produced to: test-topic [2] @ 6
Consumed: b'python test value nr 0'
Consumed: b'\x1b""ttest'
Consumed: b'python test value nr 4'
Consumed: b'python test value nr 5'
Consumed: b'python test value nr 6'
Consumed: b'python test value nr 7'
Consumed: b'python test value nr 8'
Consumed: b'python test value nr 9'
Consumed: b'"test"'
Consumed: b'"bar"'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":55555555,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
Consumed: b'python test value nr 1'
```


```
Consumed: b'python test value nr 3'
Consumed: b'"test"'
Consumed: b'{"ordertime":1497014222380,"orderid":18,"itemid":"Item_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}'
```