



**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](https://drive.google.com/file/d/1ENZKL2dkLrbL_kjgFKu-lypxsBlk27kG/view?usp=sharing)

**Video 4 Confluent Python Connection:**

[https://drive.google.com/file/d/1TgA\\_P6BU149moDn3nTyP1Z6N9F6l2Ndf/view?usp=sharing](https://drive.google.com/file/d/1TgA_P6BU149moDn3nTyP1Z6N9F6l2Ndf/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 Kafka 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-Q0u44wh3Y3uu7g 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
PartitionCount: 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>

```

// topic.js
1 //const kafka = require("kafkajs").Kafka;
2 const {kafka} = require("kafkajs");
3 run()
4
5 async function run() {
6   try {
7     const kafka = new Kafka({
8       "clientId": "myapp",
9       "brokers" : ["localhost:9092"]
10    })
11
12    const admin = kafka.admin();
13    console.log("Connecting...")
14    await admin.connect()
15    console.log("Connected!")
16
17    await admin.createTopics({
18      "topics": [{
19        "topic": "Users",
20        "numPartitions": 2
21      }]
22    })
23    console.log("Created Successful")
24    await admin.disconnect()
25  } catch(ex){
26    console.error("Something bad happened")
27  } finally {
28    process.exit(0);
29  }
30 }

// consumer.js
1 const {kafka} = require("kafkajs")
2
3 run()
4
5 async function run() {
6   try {
7     const kafka = new Kafka({
8       "clientId": "myapp",
9       "brokers" : ["localhost:9092"]
10    })
11
12    const consumer = kafka.consumer
13
14    console.log("Connecting...")
15    await consumer.connect()
16    console.log("Connected!")
17
18    await consumer.subscribe({
19      "topic": "Users",
20      "fromBeginning": true
21    })
22
23    await consumer.run({
24      "eachMessage": async result
25      console.log("RVD Msg ${{result.value}}")
26    })
27  } catch(ex){
28    console.error("Something bad happened")
29  } finally {
30    process.exit(0);
31  }
32 }
33 }

// producer.js
1 const {kafka} = require("kafkajs")
2 const msg = process.argv[2];
3 run()
4
5 async function run() {
6   try {
7     const kafka = new Kafka({
8       "clientId": "myapp",
9       "brokers" : ["localhost:9092"]
10    })
11
12    const producer = kafka.producer();
13
14    console.log("Connecting...")
15    await producer.connect()
16    console.log("Connected!")
17
18    //A-M 0 , N-1
19    const partition = msg[0] < "N" ? 0 :
20    const result = await producer.send({
21      "topic": "Users",
22      "messages": [
23        {
24          "value": msg,
25          "partition": partition
26        }
27      ]
28    })
29
30    console.log("Send Successfully! ${JSON.stringify(result)}")
31    await producer.disconnect()
32  } catch(ex){
33    console.error("Something bad happened")
34  } finally {
35    process.exit(0);
36  }
37 }

```

Output

The screenshot shows a Visual Studio Code editor with a file named `consumer.js` open. The code defines a Kafka consumer and a producer. The consumer is configured with `clientId: "myapp"` and `brokers: ["localhost:9093"]`. It subscribes to the `"Users"` topic from the beginning. The producer is configured with `clientId: "myapp"` and `brokers: ["localhost:9093"]`. It sends a message to the `"Users"` topic.

```
const kafka = new Kafka({
  "clientId": "myapp",
  "brokers": ["localhost:9093"]
})

const consumer = kafka.consumer({ "groupId": "test" });

console.log("Connecting...")
await consumer.connect()
console.log("Connected!")

await consumer.subscribe({
  "topic": "Users",
  "fromBeginning": true
})

await consumer.run({
  // ...
})
```

The terminal output shows the consumer and producer logs. The consumer logs show it connecting and subscribing to the `"Users"` topic. The producer logs show it sending a message to the `"Users"` topic.

```
RVD Msg test on partition 1
RVD Msg test on partition 1
{"level":"INFO","timestamp":"2022-02-12T03:08:03.028Z","logger":"kafkajs","message":"[Consumer] Starting","groupId":"test"}
{"level":"INFO","timestamp":"2022-02-12T03:08:03.082Z","logger":"kafkajs","message":"[ConsumerGroup] Consumer has joined the group","groupId":"test","memberId":"myapp-ce051889-2724-4260-8da5-3b3fb75f6769","isLeader":false,"memberAssignment":{"Users":["0"],"groupProtocol":"RoundRobinAssigner","duration":62}
RVD Msg Zain on partition 1
RVD Msg Zain1 on partition 1
RVD Msg Zain122 on partition 1
Connected!
Send Successfully! [{"topicName":"Users","partition":1,"errorCode":0,"baseOffset":4,"logAppendTime":-1,"logStartOffset":0}]
patel@DESKTOP-DGRS31B MINGW64 ~/OneDrive/Documents/Year IV/Semester 2/Cloud Computing/Labs/Lab2/SOF4630U-tut2/v1 (master)
$ node producer.js Zain122
Connecting...
Connected!
Send Successfully! [{"topicName":"Users","partition":1,"errorCode":0,"baseOffset":5,"logAppendTime":-1,"logStartOffset":0}]
patel@DESKTOP-DGRS31B MINGW64 ~/OneDrive/Documents/Year IV/Semester 2/Cloud Computing/Labs/Lab2/SOF4630U-tut2/v1 (master)
$ node producer.js Adam2
Connecting...
Connected!
Send Successfully! [{"topicName":"Users","partition":0,"errorCode":0,"baseOffset":2,"logAppendTime":-1,"logStartOffset":0}]
patel@DESKTOP-DGRS31B MINGW64 ~/OneDrive/Documents/Year IV/Semester 2/Cloud Computing/Labs/Lab2/SOF4630U-tut2/v1 (master)
$ node producer.js Adam23
Connecting...
Connected!
Send Successfully! [{"topicName":"Users","partition":0,"errorCode":0,"baseOffset":3,"logAppendTime":-1,"logStartOffset":0}]
patel@DESKTOP-DGRS31B MINGW64 ~/OneDrive/Documents/Year IV/Semester 2/Cloud Computing/Labs/Lab2/SOF4630U-tut2/v1 (master)
$ node producer.js Zain456
```

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

Video Link:

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

```

1  import random
2  import string
3
4
5  user_ids = list(range(1, 101))
6  recipient_ids = list(range(1, 101))
7
8
9  def generate_message() -> dict:
10     random_user_id = random.choice(user_ids)
11
12     # Copy the recipients array
13     recipient_ids_copy = recipient_ids.copy()
14
15     # User can't send message to himself
16     recipient_ids_copy.remove(random_user_id)
17     random_recipient_id = random.choice(recipient_ids_copy)
18
19     # Generate a random message
20     message = ''.join(random.choice(string.ascii_letters) for
21
22     return {
23         'user_id': random_user_id,
24         'recipient_id': random_recipient_id,
25         'message': message
26     }
27
28 if __name__ == '__main__':
29     print(generate_message())

```

```

4  from datetime import datetime
5  from topic import generate_message
6  from kafka import KafkaProducer
7
8
9  # Messages will be serialized as JSON
10 def serializer(message):
11     return json.dumps(message).encode('utf-8')
12
13
14 # Kafka Producer
15 producer = KafkaProducer(
16     bootstrap_servers=['localhost:9093'],
17     value_serializer=serializer
18 )
19
20 if __name__ == '__main__':
21     # Infinite loop - runs until you kill the program
22     while True:
23         # Generate a message
24         dummy_message = generate_message()
25
26         # Send it to our 'messages' topic
27         print(f'Producing message @ {datetime.now()}')
28         producer.send('messages', dummy_message)
29
30         # Sleep for a random number of seconds
31         time_to_sleep = random.randint(1, 11)
32         time.sleep(time_to_sleep)

```

```

1  import json
2
3
4  from kafka import KafkaConsumer
5
6
7
8
9  if __name__ == '__main__':
10     # Kafka Consumer
11     consumer = KafkaConsumer(
12         'messages',
13         bootstrap_servers='localhost:9093',
14         auto_offset_reset='earliest'
15     )
16     for message in consumer:
17         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.

```

5   name: kafka_Network
6
7   services:
8     zookeeper:
9       image: confluentinc/cp-zookeeper
10      hostname: zookeeper
11      container_name: zookeeper
12      networks:
13        - kafka_Network
14      ports:
15        - 2181:2181
16      environment:
17        ZOOKEEPER_CLIENT_PORT: 2181
18        ZOOKEEPER_TICK_TIME: 2000
19      volumes:
20        - myapp:/var/lib/zookeeper/data
21        - myapp:/var/lib/zookeeper/log
22
23     broker1:
24       image: confluentinc/cp-kafka
25       hostname: broker1
26       container_name: broker1
27       networks:
28        - kafka_Network
29       depends_on:
30        - zookeeper
31       ports:
32        - 9093:9093
33       expose:
34        - 9093
35       environment:
36        KAFKA_BROKER_ID: 1
37        KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
38        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
39        KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
40        KAFKA_ADVERTISED_LISTENERS: INTERNAL://broker1:9092,EXTERNAL://localhost:9093
41        KAFKA_DEFAULT_REPLICATION_FACTOR: 3
42        KAFKA_NUM_PARTITIONS: 3
43       volumes:
44        - myapp:/var/lib/kafka/data
45
46     volumes:
47       myapp:
48         external: true
49
50     broker2:
51       image: confluentinc/cp-kafka
52       hostname: broker2
53       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](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](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.confluent.cloud:9092',
    'sasl.mechanism': 'PLAIN',
    'security.protocol': 'SASL_SSL',
    'sasl.username': 'LOK:SK62LX5',
    'sasl.password': 'AEMCFN9Cht/kXYntT4K9V9uh4cig0=3frTL+Zxl00/5+6',
    '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'"ban"'
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'

```

Filter by keyword

Jump to offset

offset

Value

1

{

2

"ordertime": 1497014222380,

3

"orderid": 18,

4

"itemid": "Item\_184",

5

"address": {

6

"city": "Mountain View",

7

"state": "CA",

8

"zipcode": 94041

9

}

10

}

Key

1

18

Cancel

Produce

▼ {"ordertime":1497014222380,"orderid":18,"itemid":"Item\_184","address":{"city":"Mountain View","state":"CA","zipcode":94041}}

Partition: 4    Offset: 11    Timestamp: 1644796528760

```

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}}'

```