Clarissa Branje
100716458
Procedure:

1. **Watch the first 27 minutes from the following video that describes the concepts of EDA https://youtu.be/OqN6x_vNsds**
2. **Watch the following video that describes the main components of Kafka as a core of EDA https://youtu.be/JalUUBKdcA0**

Answer the following questions:

3. **What is EDA? What are its advantages and disadvantages?**

EDA stands for Event Driven Apporaches. It is utilized to aid in the examination of facts prior to making any assumptions It can aid in the detection of evident errors, as well as a better understanding of data patterns, the detection of outliers or unusual events, and the discovery of interesting relationships between variables.
Advantages

- Loose coupling
- Fault tolerance
- Reduced technical debt

Disadvantages

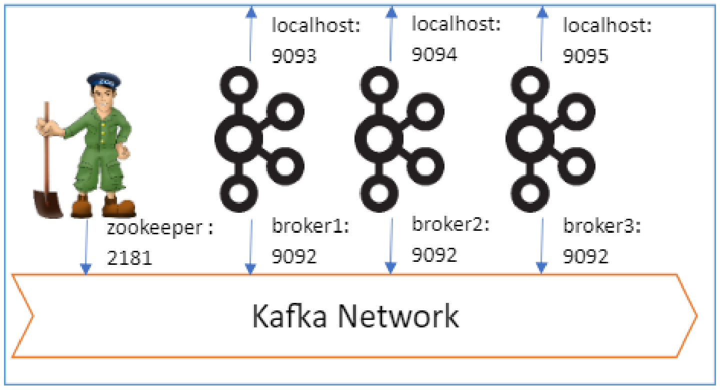
- Duplicated events
- Naming convention confusion
- Lack of clear workflow order
- Error handling and troubleshooting

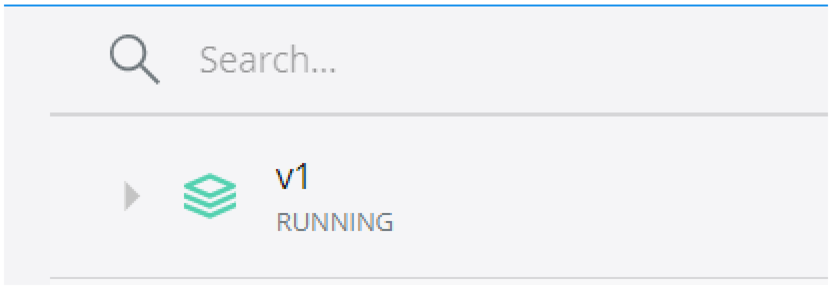
4. **In Kafka, what's meant by cluster, broker, topic, replica, partition, zookeeper, controller, leader, consumer, producer, and consumer group?**

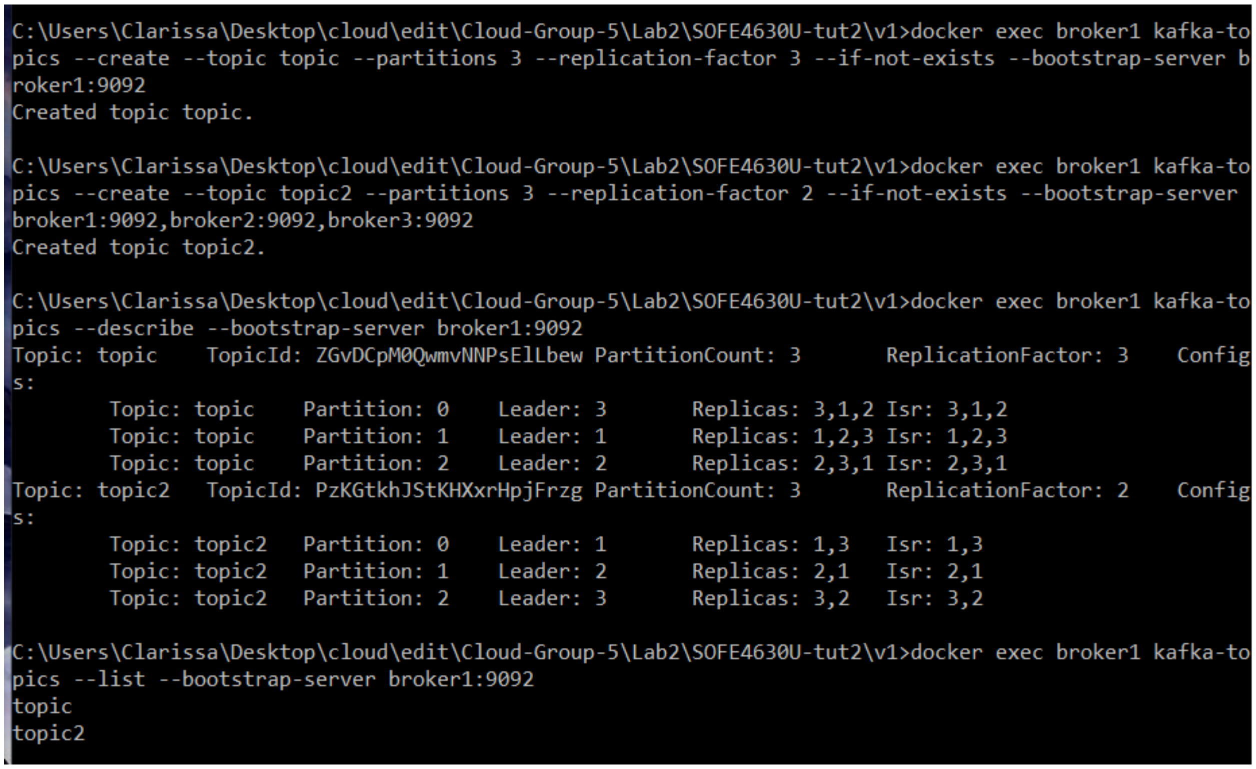
Cluster: consists of one or more brokers. Producers are processes within the broker that push records into topics.
Broker: a Kafka server that forms and runs a Kafka Cluster
Topic: a category/feed name to which records are stored and published.
Replica: Multiple copies of the data, will spread across multiple servers/brokers. This helps in maintaining high availability
Partition: this will take a single topic log and break it into multiple logs. Each of these can live on a separate node in the Kafka cluster.
Zookeeper: a replicated distributed log with a filesystem API on top
Controller: manages the states of partitions and replicas, and performs administrative tasks like reassigning partitions. There is only one controller broker in a cluster
Leader: will have followers who will replicate the leader while the leader handles all read and write requests for the partition. If the leader fails, one of the followers will automatically become the new leader.

Consumer: typically set in groups, multiple consumers will be subscribed to a topic and can receive messages from different subsets of the partitions in a topic
Producer: will take producer properties, save them as inputs, and send them to a Kafka. Producers use partitions to serialise, partition, compress, and load balance data between brokers.
Consumer Group: can create group id's and provides Kafka the flexibility to have the advantages of both message queuing and publish-subscribe models.

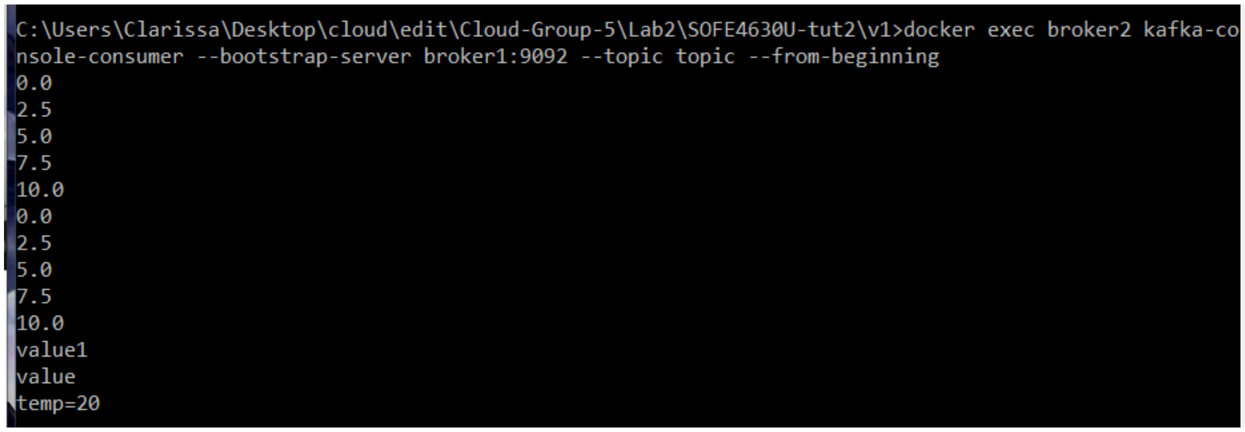
5. Follow the following video to install Kafka into your local machine and create topics, consumers, and producers using Kafka's built-in tools. https://youtu.be/Kx0o2jeMB0o



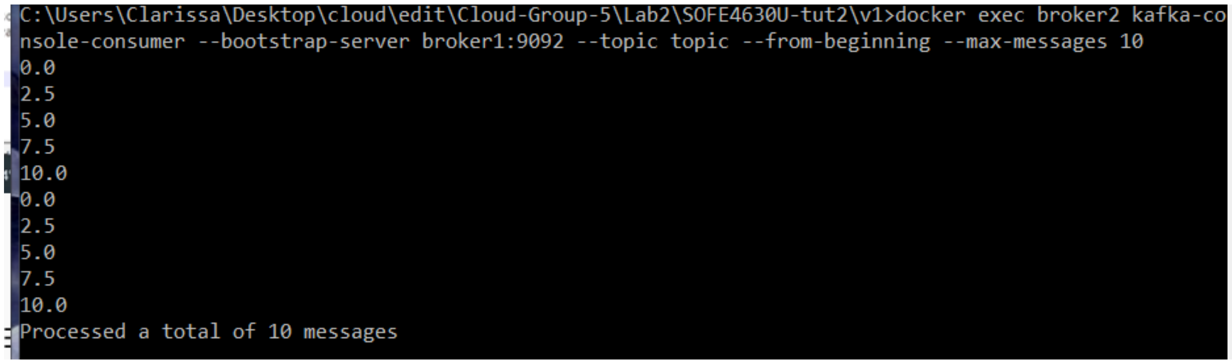




```
C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker1 kafka-to
pics --create --topic topic --partitions 3 --replication-factor 3 --if-not-exists --bootstrap-server b
roker1:9092
Created topic topic.

C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker1 kafka-to
pics --create --topic topic2 --partitions 3 --replication-factor 2 --if-not-exists --bootstrap-server
broker1:9092,broker2:9092,broker3:9092
Created topic topic2.

C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker1 kafka-to
pics --describe --bootstrap-server broker1:9092
Topic: topic    TopicId: ZGvDCpM0QwmvNNPsElLbew PartitionCount: 3       ReplicationFactor: 3    Config
s:
        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: 1,2,3
        Topic: topic    Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
Topic: topic2   TopicId: PzKGtkhJStKHXxrHpjFrzg PartitionCount: 3       ReplicationFactor: 2    Config
s:
        Topic: topic2   Partition: 0    Leader: 1       Replicas: 1,3   Isr: 1,3
        Topic: topic2   Partition: 1    Leader: 2       Replicas: 2,1   Isr: 2,1
        Topic: topic2   Partition: 2    Leader: 3       Replicas: 3,2   Isr: 3,2

C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker1 kafka-to
pics --list --bootstrap-server broker1:9092
topic
topic2
```



```
C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker2 kafka-co
nsole-consumer --bootstrap-server broker1:9092 --topic topic --from-beginning
0.0
2.5
5.0
7.5
10.0
0.0
2.5
5.0
7.5
10.0
value1
value
temp=20
```



```
C:\Users\Clarissa\Desktop\cloud\edit\Cloud-Group-5\Lab2\SOFE4630U-tut2\v1>docker exec broker2 kafka-co
nsole-consumer --bootstrap-server broker1:9092 --topic topic --from-beginning --max-messages 10
0.0
2.5
5.0
7.5
10.0
0.0
2.5
5.0
7.5
10.0
Processed a total of 10 messages
```

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

https://youtu.be/R873BlNVUB4?t=2424

Note:

- Use the docker application from the repository, not that shown in the video.
- To use the docker application from the repository, Kafka brokers' addresses will be localhost:9093,9094, and 9095.
- Follow only the NodeJS scripts, not the docker commands.
- When the partition is not specified, Kafka will decide the partition randomly or by hashing the key.

6. Using the python library Kafka-python, write three python scripts that
   - Create a topic
   - Produce messages on a topic. The message's key and the partition should be optional. Also, an error message should be displayed in the case of failure.
   - Repeatedly consume messages from a topic and display the consumed messages whenever available. The group id should be optional.

7. Prepare a video showing the codes that generated topics, produce messages, and consume them in both NodeJS and python. Your video should display the possible producer and consumer scenarios.

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.

9. Follow the following video about Kafka in Confluent Cloud https://youtu.be/vllYRz65tgA

10. Use the shown CLI to create a topic, consumer, and producer. Also update your python code to create a consumer, and producer using Kafka in Confluent Cloud (hint: only the connection information of Kafka Cluster has to be updated). Record a video illustrating those tools and showing them in action.

11. Upload the used files, reports, and videos (links) into your repository.