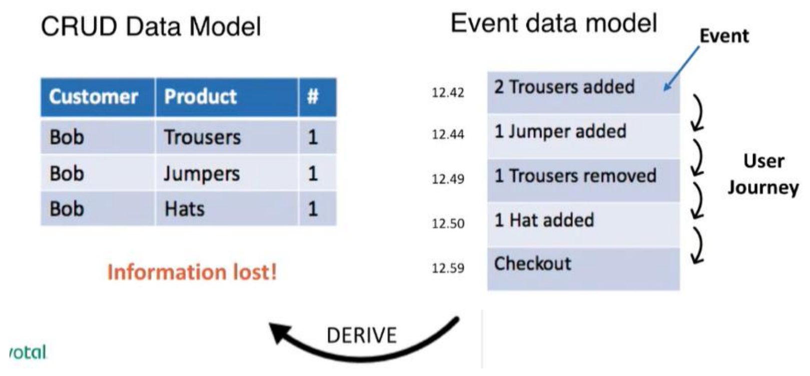
Yusuf Shaik
100 655 451

Link to Videos:
https://drive.google.com/drive/folders/19vKxMTMJUr1YVrmHRT8Uije5QRHQTp2t?usp=sharing

What is EDA? What are its advantages and disadvantages?

EDA is event driven architecture that is used to facilitate communication between decoupled micro-services within a system. An event is any action taken by the system (update status, new tweet, update page, add item to cart, remove item from cart). These items are stored in a series of events.

Event sourcing: converting events into a data model:



Customer	Product	#
Bob	Trousers	1
Bob	Jumpers	1
Bob	Hats	1

12.42	2 Trousers added
12.44	1 Jumper added
12.49	1 Trousers removed
12.50	1 Hat added
12.59	Checkout

Streaming: processing windows of the event stream

Pros: Helps reduce integration complexity, useful for big data ingestion, high scalability, easy configurability, extremely high throughput (1000's of topics per broker and millions of messages per second), custom disk-based retention, "fairly low latency".

Cons: High learning curve, not optimal for very low latency systems.

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

A **topic** in Kafka is a subject or theme relating to a group of messages. For example, a smart thermostat would send temperature information to a "temperature" topic, as well as the humidity information to a "humidity" topic.

Yusuf Shaik
100 655 451

A **broker** is a middleware between publishers and subscribers (also known as consumers and producers. **Producers** publish data to a topic, and **consumers** subscribe to that topic, receiving information from said topic. The purpose of a broker is to decouple the consumers from their producers.

Brokers contain multiple partitions, a **partition** contains either part of, or a whole topic (partitions cannot contain multiple topics). A topic is essentially split up across multiple partitions and the partitions are split across multiple brokers so that multiple clients do not decrease the throughput of the system. These split partitions are referred to as **replicas**. Partitions contain **leaders** which are the “parent” of the follower replicas. Producers send updates to this leader, and the replicas will request updates from the leader based on their configured “heartbeat”.

A **consumer group** is responsible on the consumer end for having multiple consumers to accomplish some sort of task such as gather data from all partitions of a given topic, and pushing the information into a database service.

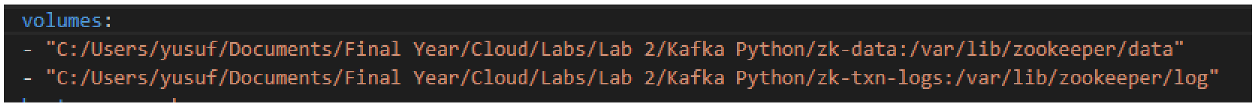
A **cluster** can be a cluster of brokers. This is to ensure high reliability and availability of a system; multiple brokers are used to make up a cluster and treated as a single system. Since there are multiple brokers in a cluster, if a broker fails, there are other brokers available to handle the data.

Zookeeper handles leader election, removal of dead brokers, topic-partition relationships, new brokers inserted into, and leaving a cluster, and new topics. Essentially keeps track of Kafka’s constantly updating topology.

Kafka **controller** is responsible for managing partitions/replicas within a broker cluster. There is only 1 within each cluster, and it is responsible for handling administrative tasks.

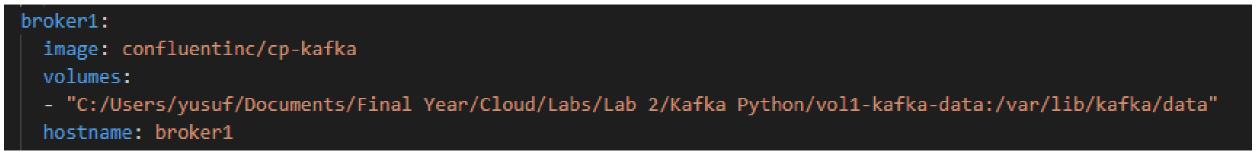
8. persistent data in yaml file:

Based on the docker documentation, created directories for zookeeper, and 1 directory for each broker. Mounted the absolute path for the host to the relative path on the container.



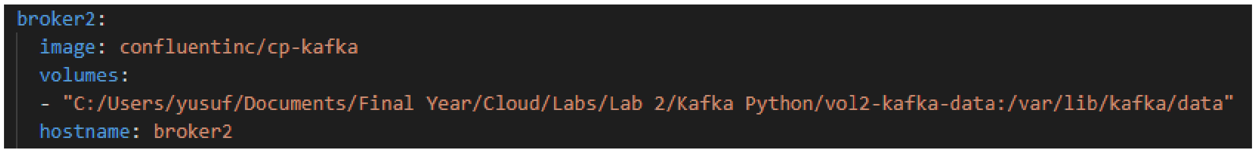
```
volumes:
- "C:/Users/yusuf/Documents/Final Year/Cloud/Labs/Lab 2/Kafka Python/zk-data:/var/lib/zookeeper/data"
- "C:/Users/yusuf/Documents/Final Year/Cloud/Labs/Lab 2/Kafka Python/zk-txn-logs:/var/lib/zookeeper/log"
```

And followed these similar steps for the remaining brokers:

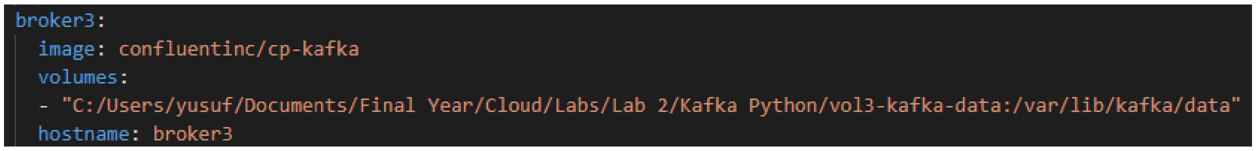


```
broker1:
  image: confluentinc/cp-kafka
  volumes:
  - "C:/Users/yusuf/Documents/Final Year/Cloud/Labs/Lab 2/Kafka Python/vol1-kafka-data:/var/lib/kafka/data"
  hostname: broker1
```

Yusuf Shaik
100 655 451



```
broker2:
  image: confluentinc/cp-kafka
  volumes:
  - "C:/Users/yusuf/Documents/Final Year/Cloud/Labs/Lab 2/Kafka Python/vol2-kafka-data:/var/lib/kafka/data"
  hostname: broker2
```



```
broker3:
  image: confluentinc/cp-kafka
  volumes:
  - "C:/Users/yusuf/Documents/Final Year/Cloud/Labs/Lab 2/Kafka Python/vol3-kafka-data:/var/lib/kafka/data"
  hostname: broker3
```

This solves the problem of containers not containing information after the container has been deleted. It creates a persistent location on the host, so that even when the container is removed, the data still remains. Every event that occurs within zookeeper and each broker is stored in a local copy on the host machine as well, so that if the container starts again, the new data will be read from these folder described in the volumes when they were mounted.

Please see linked videos for the other steps