APACHE KAFKA

1. Introduction

- Apache Kafka is a distributed publish-subscribe messaging system as well as a robust queue that can handle a high volume of data and enables you to pass messages from one endpoint to another.

1.1. Kafka Queue

- In this, a pool of Kafka consumers may read from a server, and each record goes to one of them here.
- It permits us to divide up the processing of data over multiple consumer instances, that help us scale our processing.
- It is not multi-subscriber, as soon as one process reads the data it's gone.

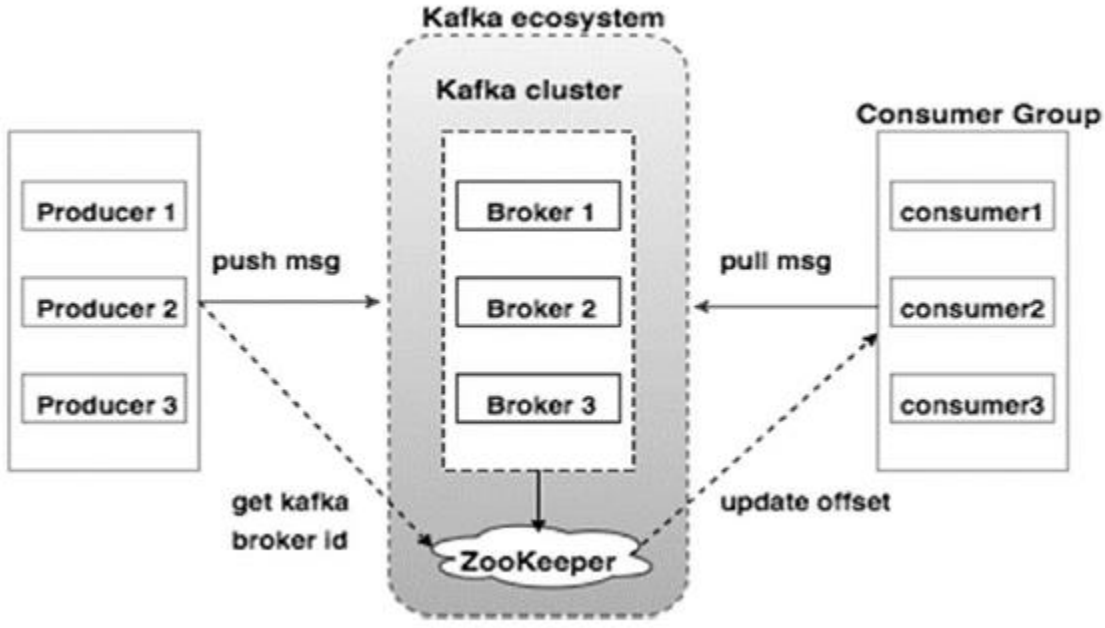
1.2 Kafka Publish-Subscribe System

- In this, the record is broadcast to all the Kafka consumers.
- It permits us to broadcast data to multiple processes.
- There is no way of scaling processing because here every message goes to every subscriber.

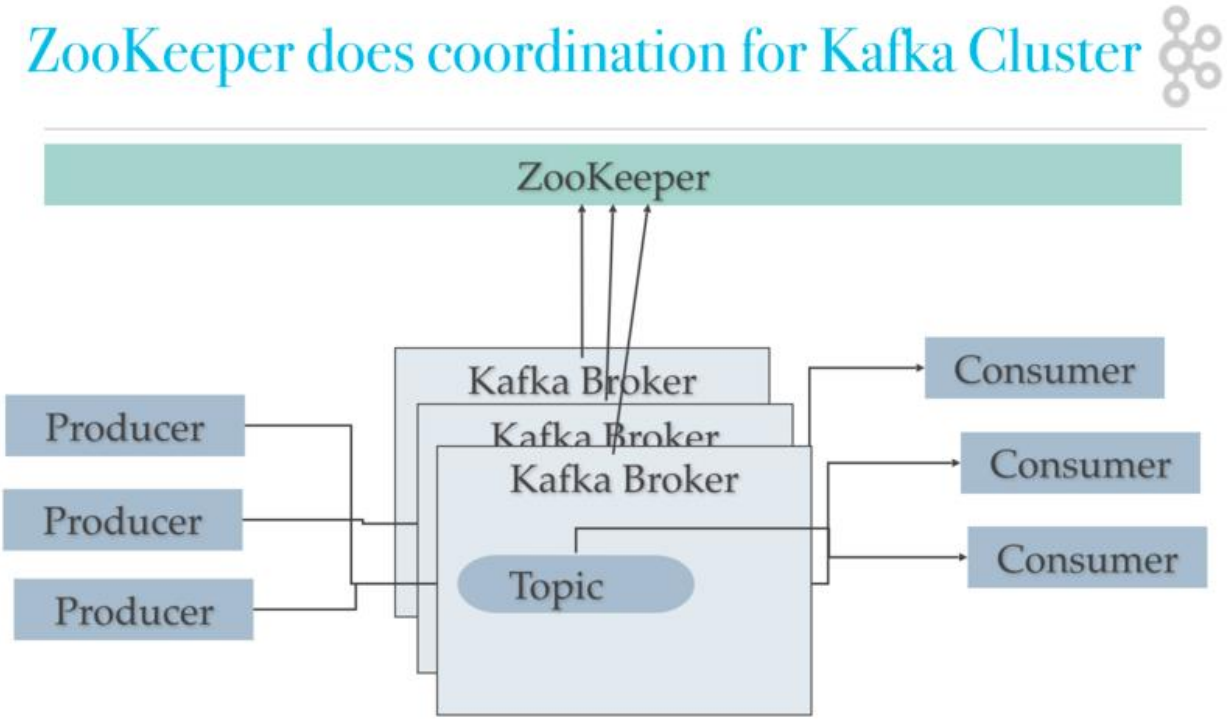
- The main benefit of Kafka's model is that both these properties are available in every Kafka topic i.e, it can scale processing as well as it is multi-subscriber. Hence, that implies we do not have to select one or the other.

2. Architecture

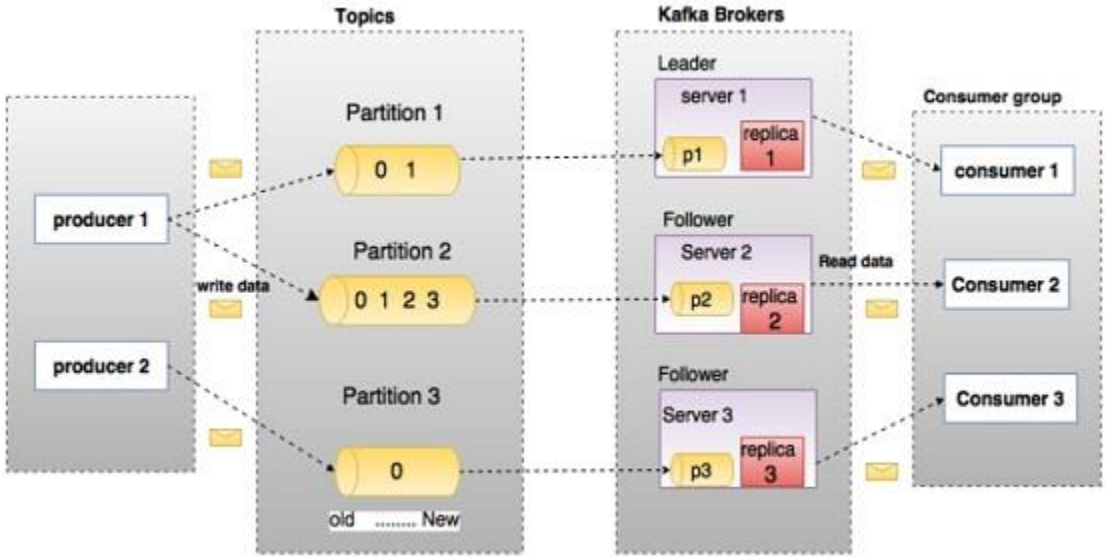
- Kafka Ecosystem



- Zookeeper does coordination for Kafka Cluster



- Kafka Brokers



2.1. Message

- A message is the data transferred from one application to another.

2.2. Topic

- A stream of messages belonging to a particular category is called a topic.
- Data is stored in topics.

2.2.1. Partition

- Topics are split into partitions, so it can handle an arbitrary amount of data.
- For each topic, Kafka keeps a minimum of one partition.
- Each such partition contains messages in an immutable ordered sequence.
- A partition is implemented as a set of segment files of equal sizes.

2.2.2. Partition Offset

- Each partitioned message has a unique sequence id called as offset.

2.2.3. Replicas of partition

- Replicas are nothing but backups of a partition.
- They never read or write data, they are only used to prevent data loss.

2.3. Brokers

- Brokers are simple system responsible for maintaining the published data.
- Each broker may have zero or more partitions per topic.
- One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact.

2.4. Producer

- Producers are the publisher of messages to one or more Kafka topics.
- Producers push data to brokers.
- When the new broker is started, all the producers search it and automatically sends a message to that new broker.
- Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.
- Every time a producer publishes a message to a broker, the broker simply appends the message to a partition.
- Producer can also send messages to a partition of their choice.

2.5. Consumer

- Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.
- Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset.
- If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages.
- The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume.
- The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.

2.6. ZooKeeper

- ZooKeeper is used for managing and coordinating Kafka broker.
- ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.
- As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.
- Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

2.7. Kafka Cluster

- Kafka’s having more than one broker are called as Kafka cluster.
- Kafka cluster typically consists of multiple brokers to maintain load balance.
- A Kafka cluster can be expanded without downtime.
- These clusters are used to manage the persistence and replication of message data.
- Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state.

2.8. Leader

- Leader is the node responsible for all reads and writes for the given partition.
- Every partition has one server acting as a leader.
- Kafka broker leader election can be done by ZooKeeper.

2.9. Follower

- Node which follows leader instructions are called as follower.
- If the leader fails, one of the followers will automatically become the new leader.
- A follower acts as normal consumer, pulls messages and updates its own data store.

3. Benefits of using Kafka vs AMQP vs JMS

- Kafka is Highly Scalable
    - Kafka is a distributed system, which is able to be scaled quickly and easily without incurring any downtime.
    - Apache Kafka is able to handle many terabytes of data without incurring much at all in the way of overhead.

- Kafka is Highly Durable
    - Kafka persists the messages on the disks, which provides intra-cluster replication.
    - This makes for a highly durable messaging system.

- Kafka is Highly Reliable1
    - Kafka replicates data and is able to support multiple subscribers.
    - Additionally, it automatically balances consumers in the event of failure.
    - That means that it's more reliable than similar messaging services available.

- Kafka Offers High Performance
    - Kafka delivers high throughput for both publishing and subscribing, utilizing disk structures that are capable of offering constant levels of performance, even when dealing with many terabytes of stored messages.