Kafka Topics

Kafka is simply a collection of topics. As topics can get quite big, they can split into partitions of an smaller size for better performance an scalability.

Topic Partition

Kafka topics are partitioned, meaning a topic is spread over a number of ***Partition/Fragments.***
Each partition can be placed on a separate Kafka Broker.
Where a new message is published on a topic, it getget appended to one of the topic's partition.
The Producer controls which partition it publishes message to based on the data.
Example :- A producer can decide that all the messages related to a particular city got to some partition.

Especially, a partition is ordered sequence of messages. Producer continuously add messages to partition. `*Kafka guarantees that all message inside a partition are stored in the sequence in which they can in. Ordering is maintained at partition level, not at topic level.'*

- A unique id/offset is assigned to every message, that enters in partition called **offset**.
- Offset Sequence are unique to each partition. This means to locate a message, we need to know topic, partitions, offset.
- Producer can choose to publish a message to any partition. In ordering with in a partition is not needed, round-robin partition strategy can be used, so records get distributed evenly accross partitions.
- Placing each partition on separate Kafka Broker enables multiple consumer to read from a topic in parallel. That means, different consumers can concurrently read different partitions present on separate Kafka Brokers.
- Placing each partition of a topic on a separate broker also enables a topic to hold more data than of the capacity of the server.
- Message once written to a part are **immutable** and can't be updated.
- A producer can add a **`key`** to any message it publishes. Kafka guarantees that message with same key are written to same partition.
- Each Kafka Broker manages a set of partitions belonging to different topics.

Kafka follows the principle of **dumb broker** and **smart consumer.** This means, It doesn't records the data read by the consumer. Instead consumer, themselves can poll the next messages from kafkaesque or can give offset from where they want to read / consume data. This also allows consumer to join cluster at any point in time.
Each topic can be replaced to multipleKafka Brokers to make the data fault-tolerant & high available. Every topic partition has one leader broker and multiple replica(followers) brokers.

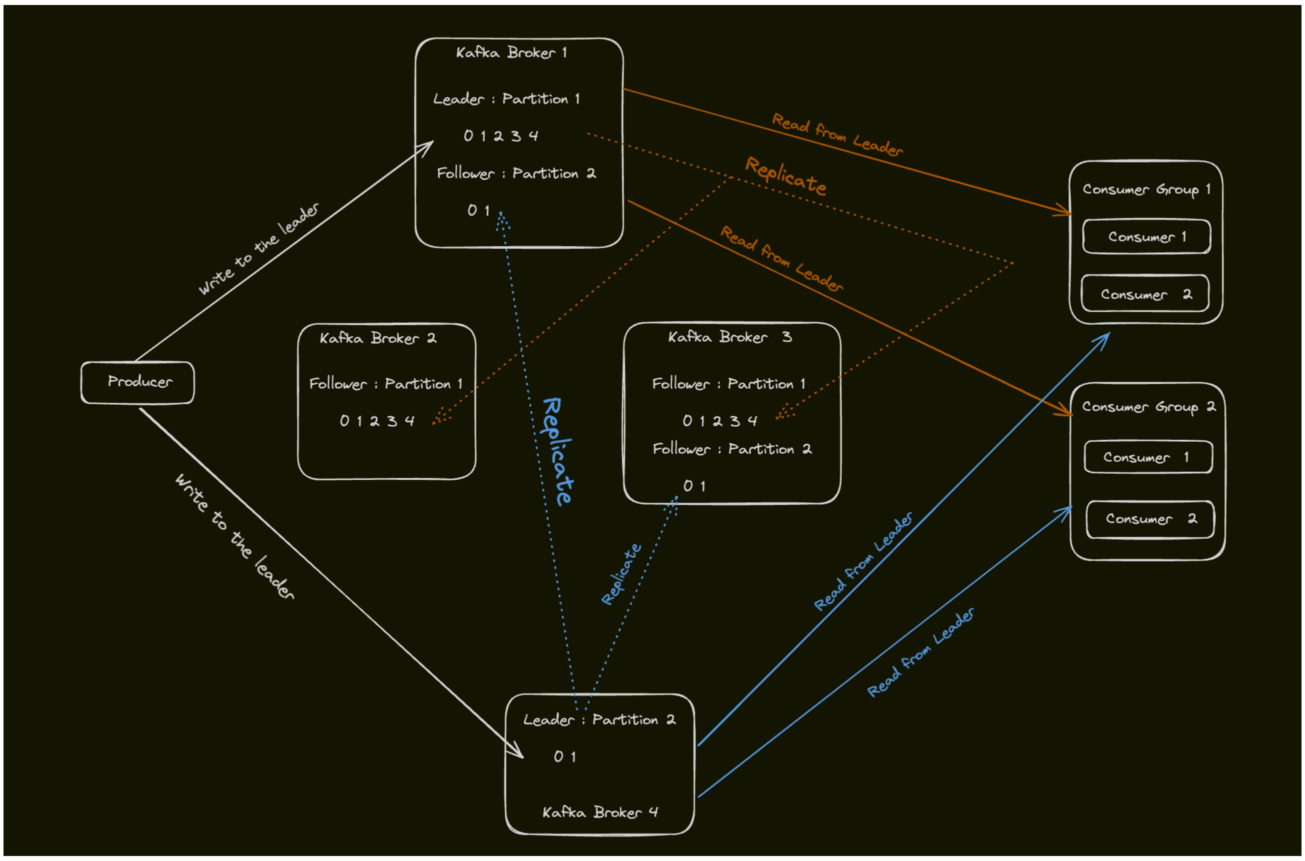
Leader

A leader is a node responsible for all read and write for the given partition. Every partition has one Kafka broker acting as a leader.

Follower

To handle single point of failure, Kafka can replicate partitions and distribute them accross multiple brokers server as followers. Each followers responsibility to have a replica of leader's data follower serve backup partition.This means any follower can take leadership if the leader goes down.

In the following diagram we have two partitions and four brokers. **Broker 1** is leader of **Partition 1** & follower of **Partition 2** . Consumers work together in group to process messages efficiently.



Leader and Followers of partition

Kafka Stores the location of the leader of each part in Zookeeper. As all the write/read happens at/from the leader Producers & Consumers directly talk to zookeeper to find a partition leader.

In Sync Replica

In sync replica is a broker that has latest data of a given partition. A leader is always an in-sync replica. A follower is In-sync replica, if it has caught up to the partition it is following. In another words, In-sync replica can never be behind on the latest record of the partition. **Only In-Rsync Replica** to become partition leader. Kafka can choose minimum numbers of In-sync replicas before the data becomes available for consumer to read.

High Water Mark

To ensure data consistency, the leader broker never return messages which has not been replicated to a minimum sets of In-sync replica. For this the broker keeps the record of the **High Water Mark** offset and propagates the **High Water Mark** offset to all the followers.

Example : If a consumer reads records with offset **7** from leader[Broker 1] and later the current leader fails , and one of the follower becomes the leader before the record is replicated to the followers, then consumer will not be able to find the message or the leader was not in total sync. Because of this, Kafka Broker only returns the records upto the **High Water Mark**.