Workflow

As of now, we discussed the core concepts of Kafka. Let us now throw some light on the workflow of Kafka.

Kafka is simply a collection of topics split into one or more partitions. A Kafka partition is a linearly ordered sequence of messages, where each message is identified by their index (called as offset). All the data in a Kafka cluster is the disjointed union of partitions. Incoming messages are written at the end of a partition and messages are sequentially read by consumers. Durability is provided by replicating messages to different brokers.

Kafka provides both pub-sub and queue based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner. In both cases, producers simply send the message to a topic and consumer can choose any one type of messaging system depending on their need. Let us follow the steps in the next section to understand how the consumer can choose the messaging system of their choice.

Workflow of Pub-Sub Messaging

Following is the step wise workflow of the Pub-Sub Messaging:

- Producers send message to a topic at regular intervals.
- Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared

between partitions. If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.

- Consumer subscribes to a specific topic.
- Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
- Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.
- Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- Consumer will receive the message and process it.
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outrages.
- This above flow will repeat until the consumer stops the request.
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

Workflow of Queue Messaging / Consumer Group

In a queue messaging system instead of a single consumer, a group of consumers having the same “Group ID” will subscribe to a topic. In simple terms, consumers subscribing to a topic with same “Group ID” are considered as a single group and the messages are shared among them. Let us check the actual workflow of this system:

- Producers send message to a topic in a regular interval.
- Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.
- A single consumer subscribes to a specific topic, assume `Topic-01` with `Group ID` as `Group-1`.
- Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, `Topic-01` with the same `Group ID` as `Group-1`.
- Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. This sharing will go on until the number of con- sumers reach the number of partition configured for that particular topic.
- Once the number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribes. This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.
- This feature is also called as `Consumer Group`. In the same way, Kafka will provide the best of both the systems in a very simple and efficient manner.

Role of ZooKeeper

A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service. Zookeeper serves as the coordination interface between the Kafka brokers and consumers.

The Kafka servers share information via a Zookeeper cluster. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

> *Note: Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker/Zookeeper does not affect the state of the Kafka cluster. Kafka will restore the state, once the Zookeeper restarts. This gives zero downtime for Kafka. The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.*

Let us continue further on how to install Java, ZooKeeper, and Kafka on your machine on Kafka in the first lab.