Zookeeper: It is a software which manages the metadata of Kafka cluster. It is responsible for managing the topics, partitions etc…

Chart

Description automatically generated

The topic is divided into multiple partitions over one or more brokers (brokers == servers).

The messages in the partitions are time ordered, meaning each message is assigned with a timestamp. The order of these message timestamps are guaranteed only within a partition (not across partitions)

Fault Tolerance features:

Whenever a broker goes down, all the partitions of it will also go down. The producers and consumers talking to such broker will experience the failure. In such cases the zookeeper will take out such broker out of the cluster and update the consumers and producers to use other partitions on other alive brokers. This is fine, but still there is a data loss. That is where the question of reliability arises. Reliability NFR is handled by the feature called “Replication Factor”.

Note that in the following topic creation command, we mention the reliability through the parameter: replication-factor. It should be more than 1.

Text

Description automatically generated

This will enable the Kafka to store the messages redundantly in more than one broker. This replication ensures

* Cluster Resilience
* Fault Tolerance

The replication\redundancy will actually happen for partitions of a given topic. The same partition of the topic replicates on another broker. So, if any one broker goes down (along with the partitions in it), the zookeeper will bring in another broker into the scene (which will have redundant partitions with it).

Diagram

Description automatically generated

Graphical user interface

Description automatically generated

The scalability of apache kafka is driven by the number of partitions managed by the brokers.

The partitions are mutually exclusive in nature, they receive unique messages from the producer. How the messages are distributed evenly among the partitions of a topic? Generally it will be done in a round robin fashion.

Among the available brokers\kafka servers, a leader will be chosen.

**Producer Keys:** If the publisher would like to send all the messages to one particular\same partition of the topic, then a key must be specified. This key can be like CustomerId, OrderId etc…

Diagram, text

Description automatically generated

What happens if you try to publish the messages to a non-existent\wrong topic ?

Kafka doesn’t show any errors, instead it shows a warning, saying that, there is NO\_LEADER\_AVAILABLE, as the leader election not happened between the partition(s) of this topic. Kafka internally creates this new topic.

Diagram

Description automatically generated

The consumers will always read from the topic partitions in the consumer groups. Let’s see an example.

Diagram, PowerPoint

Description automatically generated

As you can see the ‘consumer-group-application-1’ have 2 consumers, where consumer 1 reads from partitions 0 and 1, where as consumer 2 reads just from partition 2.

The ‘consumer-group-application-2’ got 3 consumers (an analytics application) where each consumer reads from each individual partition.

The ‘consumer-group-application-3’ got only 1 customer (may be a notification service) which reads from all the partitions.

If the no. of consumers are more than the no. of available partitions of the give topic, then some consumers will be inactive.

For ex:

Diagram

Description automatically generated

As you can see we got 4 consumers against 3 partitions. So the consumer-4 will be inactive\passive one. This will become active when any one of the active consumers goes down.

Table

Description automatically generated

You can see here as and when consumer reads the messages from the partition, it will commit the message offset. For example, last time when consumer reads the message (4262) before it had gone down, it committed it. When it comes back, the Kafka will let the consumer to start from the next offset, i.e. 4263.

*When to commit the consumer offset?*

*The answer depends on the delivery semantics.*

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, diagram, application

Description automatically generated

Text

Description automatically generated

Let’s have the zookeeper is hosted on a 3 server cluster.

Diagram

Description automatically generated

The leader instance of the zookeeper (server 2) manages all the writes, but other instances (1,3) the followers, will handle all the reads. (CAP theorem)

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated

Scenario: Let’s assume you have a topic with 3 partitions with it.

You will start with a producer, producing messages with keys. Imagine you have 1 consumer (with a groupid). This consumer will listen to all the partitions (0,1,2).

Net imagine if you start a new consumer under the same groupid, then the first consumer rebalances itself to listen to 2 partitions only out of 3, as the new consumer started, will listen to the other partition. Thus the first (existing) partition is left with only 2 partitions.

If you start one more consumer, the first 2 consumers will again rebalances themselves as go with 1 partition each by leaving 1 partition to this newly started consumer.

So every time when you start a new consumer, a ‘Consumer Co-ordinator’ will rebalances the existing consumers to re-assign\re-allocate the partitions among the existing and new consumers (provided all the old and new consumers are with the same groupid). This is called Rebalancing.