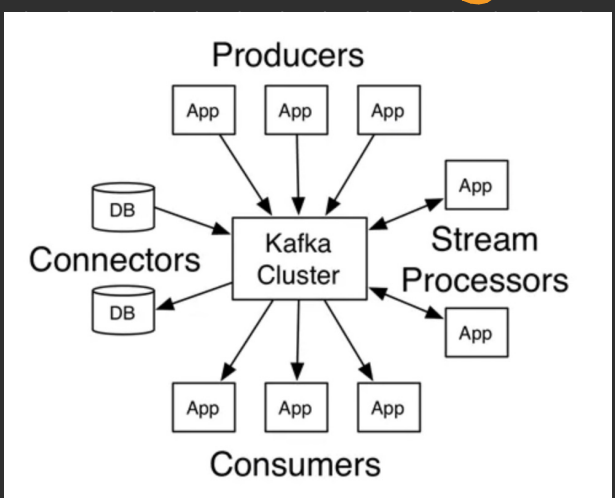
Distributed Messaging Queue (Kafka)

Kafka

Kafka Introduction:

- It is developed by Linkedin.
- It is highly scalable, distributed, durable & fault-tolerant publish-subscribe event streaming & messaging system.
- Written in scala & java.



There are two types of messaging patterns available:

i) Queuing (Point to Point)

ii) Publish- Subscribe (Pub/Sub)

- Kafka enables both above models through "consumer group" concept making it more scalable.

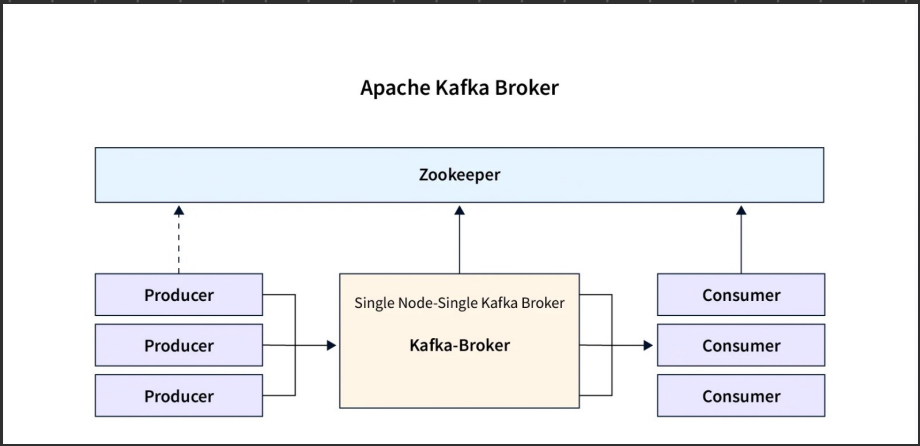
- As with queue;

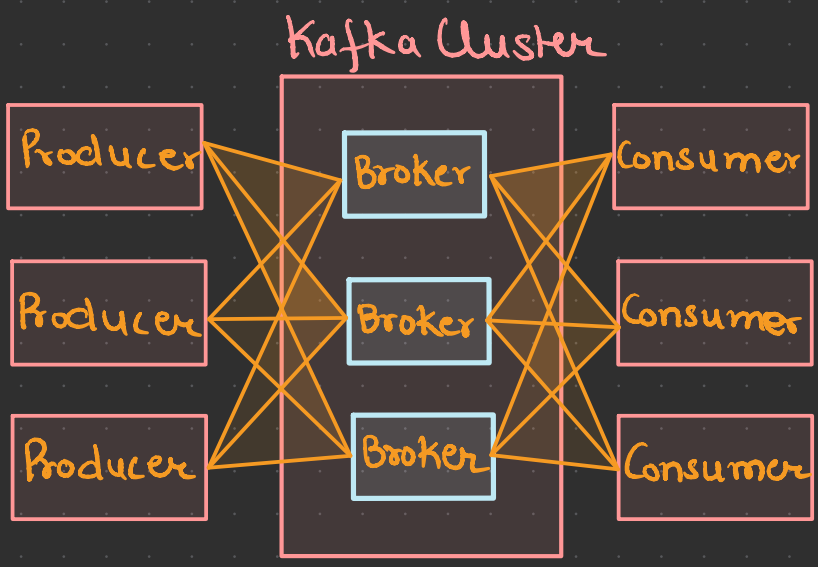
  consumer group allows you to divide up the processing over members of consumer group.

- With publisher-subscriber,

  it allows you to bradcast msgs to multiple consumer groups.

Components of Kafka



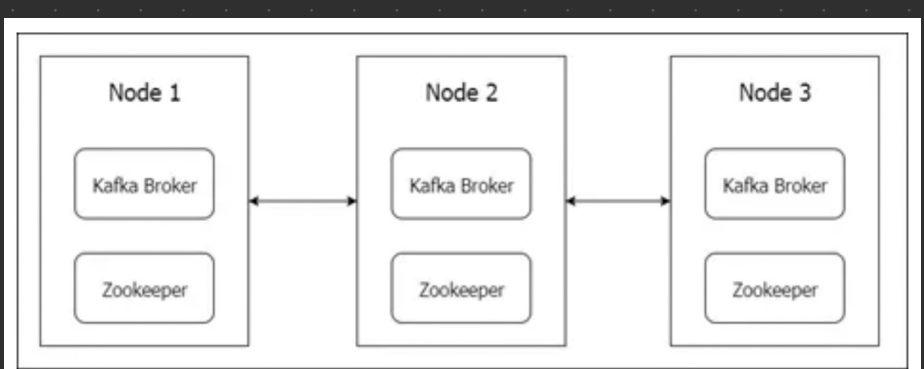


- Broker
- Messages / Records
- Topic
- Producers

- Consumers
- Streams
- Connector

- **Broker:**
  - An instance or a single server of a Kafka cluster.
  - A broker will be automatically elected as the controller.
  - Controller is responsible for admin operations such as partition assignment to broker, monitoring broker failures etc.

Controller Election:



Zookeeper Instances running on these servers.

i) First server that is registered in the cluster creates a controller node in zookeeper.

ii) other servers that are also being registered try to create a controller node in Zookeeper but get notified as "ec controller exists".

iii) These servers then create a zookeeper watch to monitor the controller node.

iv) If the controller broker breaks down, others are notified. Then each one tries to become a controller but only one wins the race.

Distributed Commit Log:

- Kafka cluster durably persists all published records/messages on the topics

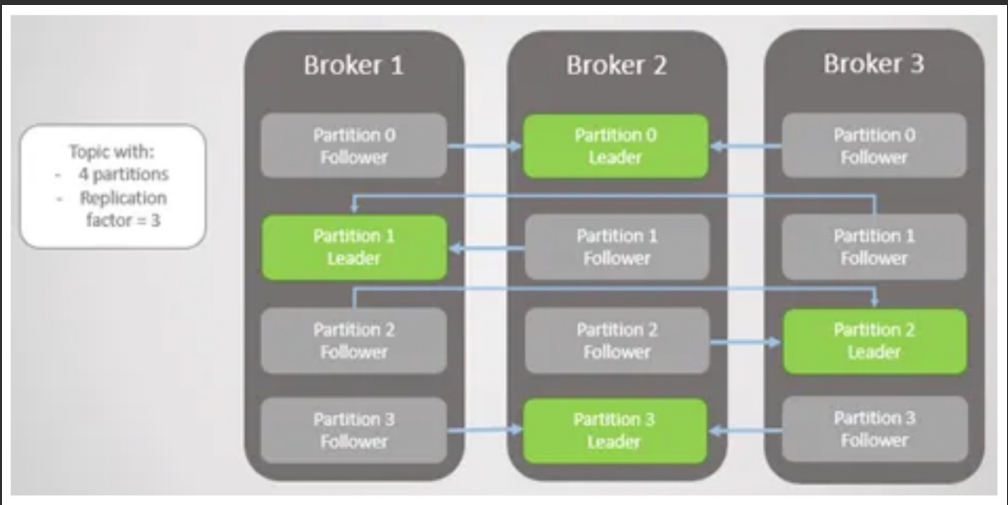
whether they have been consumed or not (using retention period or a storage size exceeded).

- These records cannot be deleted or modify once they are sent to Kafka; this is known as "distributed commit log".

- Messages / Records:

  - A message is a key/value pair of data only with metadata such as a timestamp & message key.
  - Messages are stored inside topics within a log structure format.
  - Log is a persistent ordered data structure which only supports append.
    - Uniqueness of a message is determined from a tuple of (partition, topic, offset).

- Topics :



- Topic is a structure that holds the messages or the records published to the kafka broker.
- A topic can be divided into partitions, the place where data is published.
- Each partition is an ordered, immutable sequence of records that is appending to - a structured commit log.
- Each record in the partition is assigned a sequential id number

called offset that uniquely identifies each record.

- A general formula to pick up the number of partitions is based upon throughput needs.
- If partition keys are not used then Kafka assigns the key in a default manner which makes the ordering unguaranteed.
- Each partition has one server which acts as the "leader" & more servers which act as "followers".
- The leader handles all read & write requests for the partition while followers passively replicate the leader.
- If the leader fails, the kafka controller will detect the failure and elect a new leader from the pool of followers.

- Partitions allow Kafka to parallelize a topic by splitting the data of a topic across multiple brokers, thus adding an essence of parallelism to the ecosystem i.e each partition can be placed on a separate machine to allow for multiple consumers to read from a topic in parallel.
This is how kafka delivers high throughput.
On the other hand more partition sometime leads to low latency as in Kafka the message can be available to consumers until it is replicated.

- Producers:

- Producers through Producer APIs can connect to Kafka Brokers & publish

a stream of records or messages to one or more kafka topics irrespective of the desired pattern.

- **Consumers:**
  - Consumers are subscriber of kafka topics.
  - The data is extracted from a kafka topic using a Consumer Instance which is encapsulated inside a Consumer Group.
  - It is basically a collection of one or more consumers working collectively in parallel to consume messages from topic partitions.
  - Each consumer has a consumer groupID kafka stores the current offset per Consumer Group / Topic / Partition, as it

would for a single consumer.

- No two consumers having the same group-id would be assigned to the same partition.
- That means each partition is tied to only one consumer process per consumer groups, that way it avoids reading the same record twice - guaranteed each message is only processed once.

If consumers > partitions

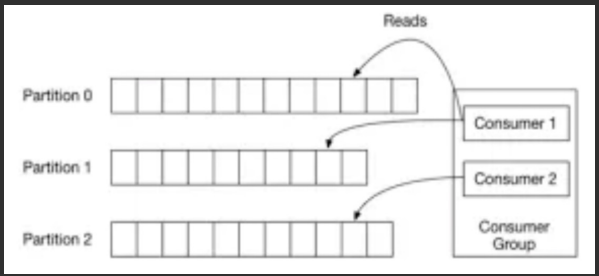
Some consumers will be idle.

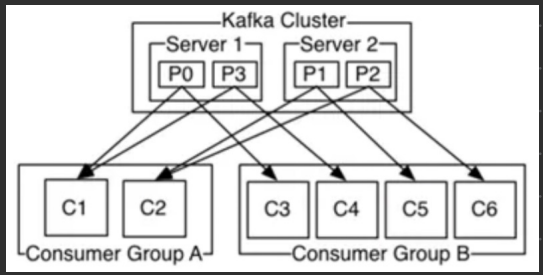
If partitions > consumers

some consumers will receive messages from multiple partitions

If consumers = partitions

each consumer reads msgs in order from exactly one partition





- Streams :

- Streams consume from one or more topics & produce an output stream to one or more output topics, effectively transforming the input streams to output streams.
- Kafka enables this with regular producers, with Streams we can achieve real-time stream processing rather than batch-processing.

- Connector :

- It is a framework for importing data into kafka from external

datasources or exporting data to external sources like databases & applications.