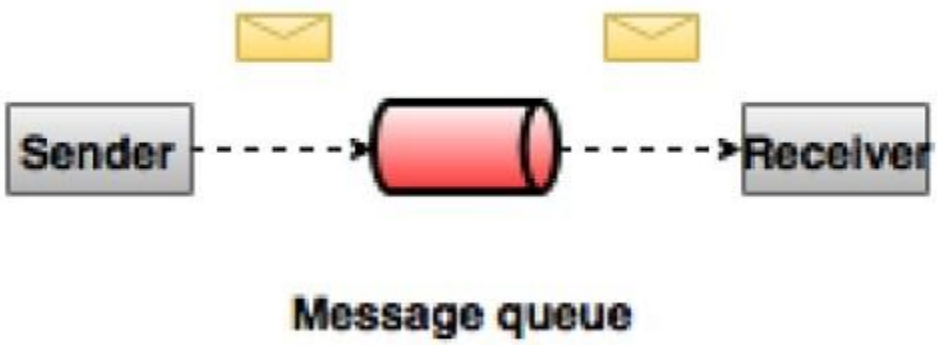
Messaging System

1. A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it
2. Messages are queued asynchronously between client applications and messaging system.
3. Two types of messaging patterns are available − one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow pub-sub.

Point to Point Messaging System

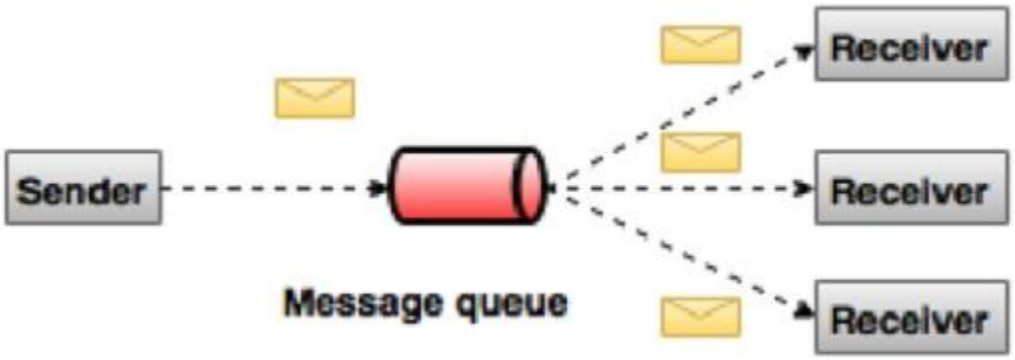
1. In a point-to-point system, messages are persisted in a queue.
2. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue.
3. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.



Publish-Subscribe Messaging System

1. In the publish-subscribe system, messages are persisted in a topic.
2. Unlike a point-to-point system, consumers can subscribe to one or more topics and consume all the messages in that topic.

3. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers.
4. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

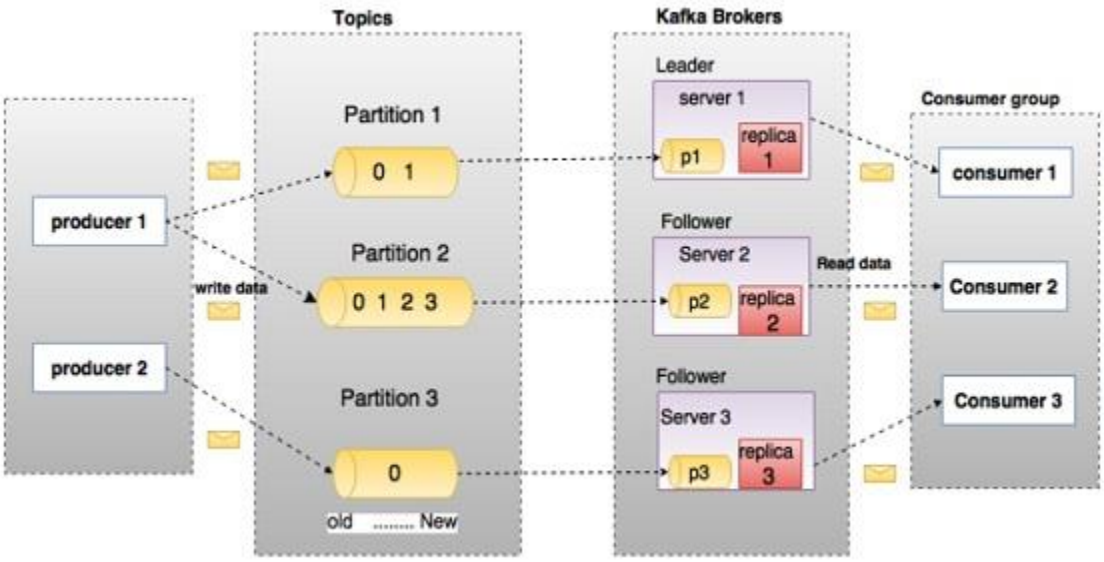


Kafka provides both pub-sub and queue based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner. In both cases, producers simply send the message to a topic and consumer can choose any one type of messaging system depending on their need.

What is Kafka?

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

Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.



S.No	Components and Description
1	**Topics** A stream of messages belonging to a particular category is called a topic. Data is stored in topics. Topics are split into partitions. For each topic, Kafka keeps a minimum of one partition. Each such partition contains messages in an immutable ordered sequence.

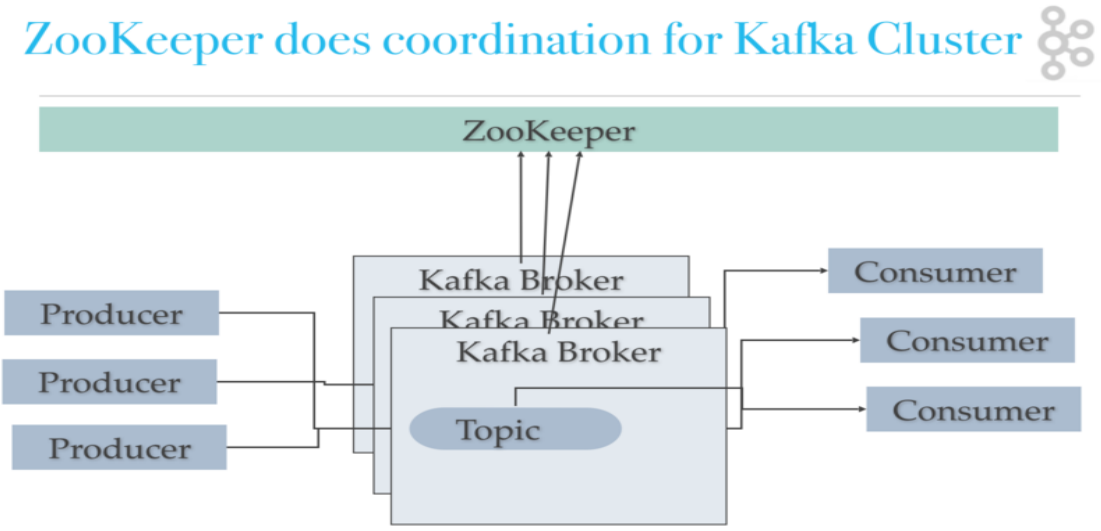
2	**Partition** Topics may have many partitions, so it can handle an arbitrary amount of data.
3	**Partition offset** Each partitioned message has a unique sequence id called as offset.
4	**Replicas of partition** Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss.
5	**Brokers** ● Brokers are simple system responsible for maintaining the published data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition. ● Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic. ● Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distri-bution among the broker.

6	**Kafka Cluster** Kafka's having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.
7	**Producers** Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to a partition. Producers can also send messages to a partition of their choice.
8	**Consumers** Consumers read data from brokers. Consumers subscribe to one or more topics and consume published messages by pulling data from the brokers.
9	**Leader** Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.

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

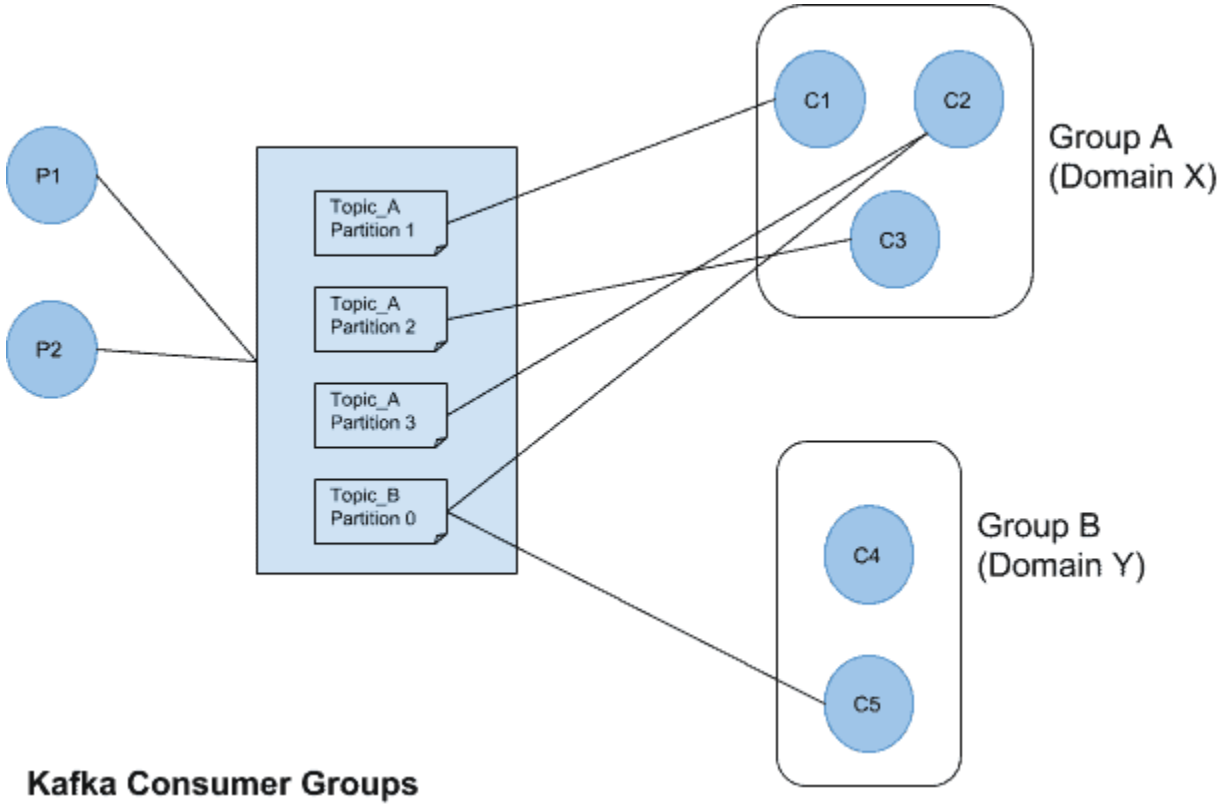
Role of ZooKeeper

1. Kafka brokers are stateless. All states are maintained by the zookeeper.
2. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets and so on.
3. The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.



Consumer Groups

1. Consumer groups give Kafka the flexibility to have the advantages of both message queuing and publish-subscribe models. Kafka consumers belonging to the same consumer group share a group id. The consumers in a group then divide the topic partitions as fairly amongst themselves as possible by establishing that each partition is only consumed by a single consumer from the group.



Kafka Consumer Groups

2. If all consumers are from the same group, the Kafka model functions as a traditional message queue. All the records and processing is then load balanced Each message would be consumed by one consumer of the group only. Each partition is connected to at most one consumer from a group.
3. When multiple consumer groups exist, the flow of the data consumption model aligns with the traditional publish-subscribe model. The messages are broadcast to all consumer groups.
4. you create a new consumer group for each application that needs all the messages from one or more topics. You add consumers to an existing consumer group to scale the reading and processing of messages from the topics, so each additional consumer in a group will only get a subset of the messages.

Workflow of Pub-Sub Messaging

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

- Producers send messages 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 the 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.

Benefits of Kafka

Following are a few benefits of Kafka −

1. **Reliability** − Kafka is distributed, partitioned, replicated and fault tolerance.
2. **Scalability** − Kafka messaging system scales easily without down time..
3. **Durability** − persists data on multiple replicas, hence it is durable..
4. **Performance** − Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even when many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

Use Cases

- **Metrics** − Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
- **Log Aggregation Solution** − Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.

Commands :

Start Zookeeper :

bin/zookeeper-server-start.sh config/zookeeper.properties &

Start Kafka :

bin/kafka-server-start.sh config/server.properties &

Create Kafka Topic :

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2 --partitions 3 --topic test4

Describe topic :

bin/kafka-topics.sh --describe --topic topicName --bootstrap-server localhost:9092

List Kafka Topics :

bin/kafka-topics.sh --list --zookeeper localhost:2181

List messages from kafka topic :

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic txn_main_details --from-beginning

Create Consumer with groupId :

1. create a new `consumer.properties` file, say `consumer1.properties`.
2. change `group.id=<Give a new group name>` in the `consumer1.properties`.
3. `bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic group_topic --from-beginning --consumer.config config/consumer1.properties`

Start producer :

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

Sources :

https://www.tutorialspoint.com/apache_kafka/apache_kafka_workflow.htm
https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html
https://stackoverflow.com/questions/45175424/how-multiple-consumer-group-consumers-work-across-partition-on-the-same-topic-in
https://blog.cloudera.com/scalability-of-kafka-messaging-using-consumer-groups/
https://kafka.apache.org/quickstart
https://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/