Kafka

Kafka is a publish-subscribe messaging system which works Spark and Storm for real-time analysis and data streaming. In this course you will learn about the cluster architecture, basic operations, workflow, integration with spark and Kafka application.

##### **How Kafka Came to be ..**

During the initial days of big data, the focus was mainly on batch processing. In batch processing, applications would be run daily or weekly to load and analyze data from big data stores.

More recently, businesses have an increased need for handling **real-time data feeds**, i.e. analyzing and processing data and events as they happen.

To meet these demands, engineers at **LinkedIn** developed and open sourced **Kafka**, a stream processing platform which scales on commodity hardware.

* Kafka is a publish-subscribe messaging system, written in **Scala** and **Java**, that is fast, distributed and durable.
* Kafka is fault-tolerant and enables you to build distributed applications that scale on commodity hardware.

##### **What is a Messaging System?**

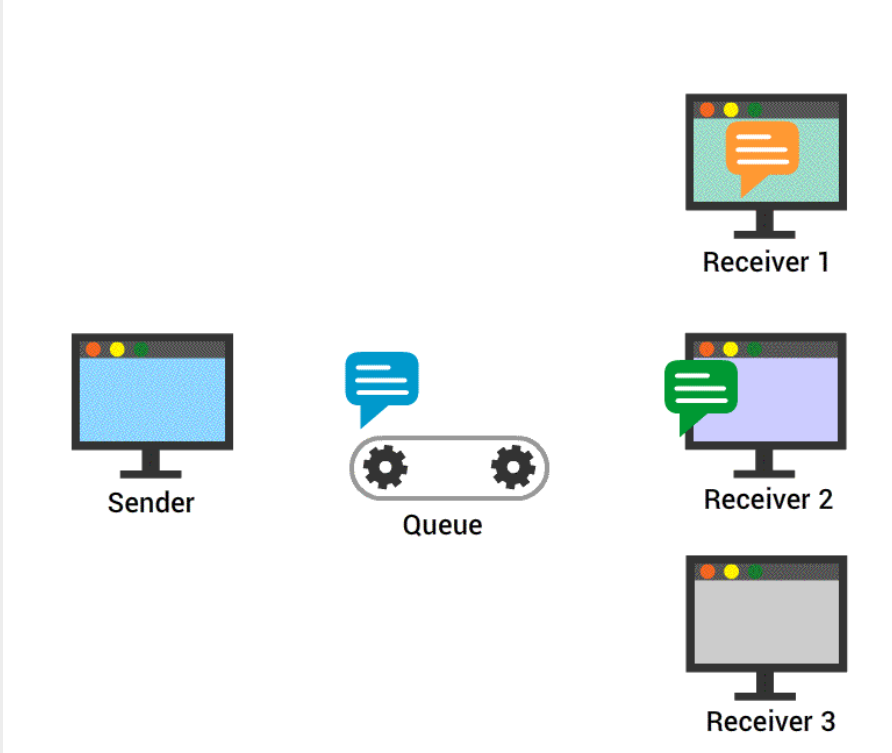
Before we explore Kafka, let's understand the types of messaging systems.

A **messaging system** is a medium that allows data transfer from one application to another so that the applications can focus on data without worrying about how to share it.

The two types of messaging patterns are:

* **Point to Point** Messaging System
* **Publish-Subscribe** Messaging System.

##### **Point to Point Messaging System**

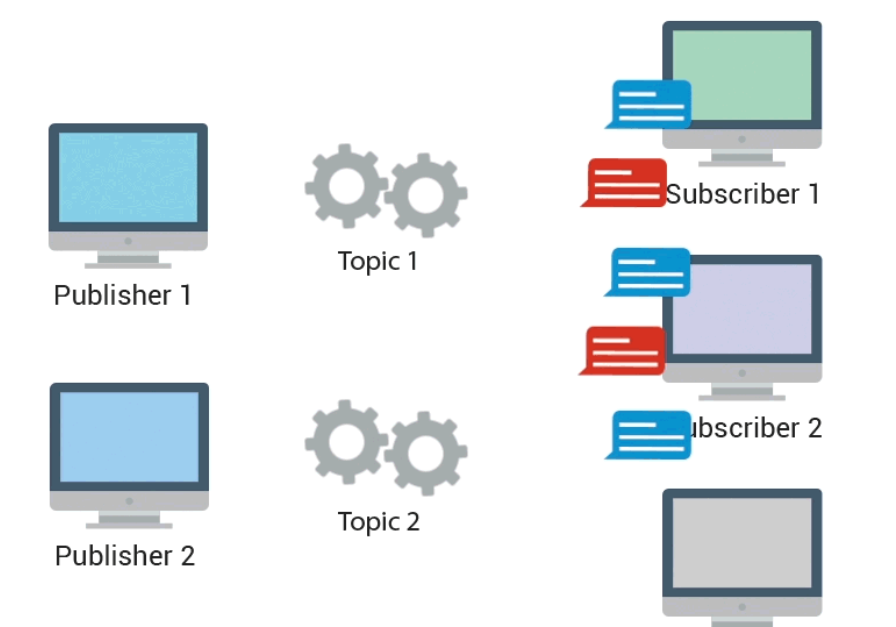


In **Point to Point** messaging system, senders send messages to a queue and receivers consume messages from the queue.

But there is a restriction that a particular message can be consumed by a maximum of only one receiver.

The message disappears from the queue, once the message is consumed by the receiver.

##### **Publish-Subscribe Messaging System**



In **Publish-Subscribe** messaging system, senders, also known as **publishers**, classifies messages and publish them to a **topic**. Receivers, or **subscribers**, can receive messages only on subscribing to that **topic**.

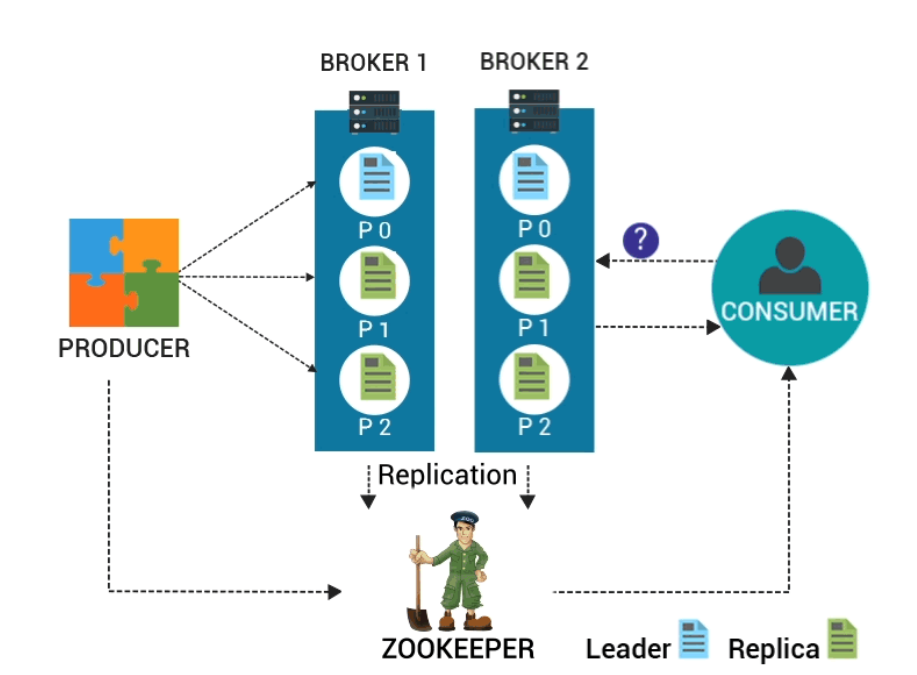
Unlike point to point, in Publish-Subscribe messaging system,

* A message on a topic is broadcast to all subscribed consumers.
* Consumers can subscribe to multiple topics to receive messages.

##### **What is Kafka?**

**Apache Kafka** is a distributed **publish-subscribe** messaging system used for collecting and delivering high volumes of data with low latency, similar to a traditional message broker.

##### **How Kafka Works**



* Kafka stores messages coming from multiple applications called **producers**.
* The messages get partitioned into **partitions** and based on some classification written to different **topics**.
* The messages in a partition are indexed and stored with a **timestamp**.
* Multiple applications called **consumers** polls messages from partitions.
* Kafka runs as a **cluster of servers**.
* Each topic partition is replicated over the nodes of the Kafka Cluster.
* **TCP protocol** is used for communication between clients and Kafka nodes.

##### **What is Kafka Used For?**

Kafka is used for :

* Building real-time **streaming pipelines** that move data between different applications.
* Building real-time **streaming applications** that are capable of processing streams of data.
* Building a fault tolerant **storage system** that stores streams of records.

We will be discussing the above points in detail in the upcoming topics.

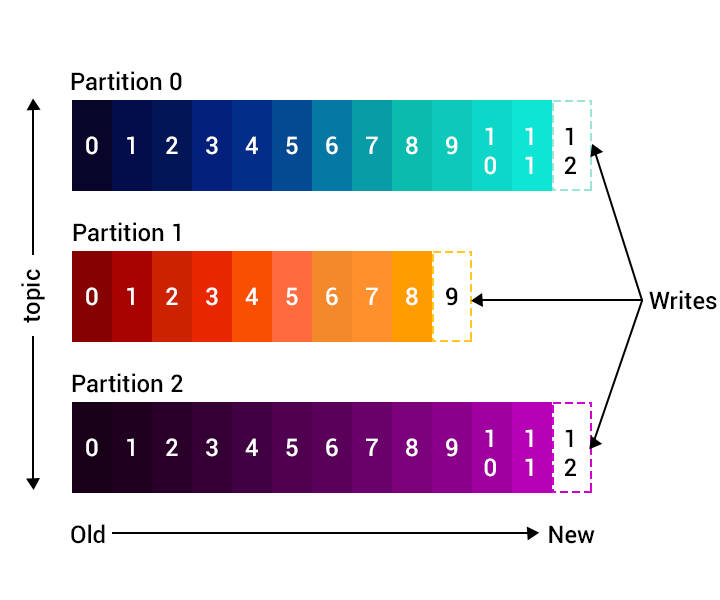
##### **Benefits of Kafka**

* **Reliability**: Kafka's distributed design, topic partitioning, and data replication over servers make it reliable.
* **Scalability**: Kafka system exists as a cluster of brokers. The number of brokers can grow over time when more data comes. Any failure of an individual broker in a cluster is handled by the system providing uninterrupted service.
* **Durability**: Disk-based data retention makes Kafka durable. Messages remain on the disk based on the retention rule configured on a per-topic basis. Even if a consumer falls backs due to any reason, the data continue to reside in the Broker till the retention period and is not lost.
* **High-Performance**: All the above features make Kafka a High-Performance messaging system.

*Kafka subscribes data from a topic, process, and writes the processed data to another new topic which is then made available to users.*

##### **Fundamental Components**

##### **Topic**



* A **Kafka topic** is a category or feed name under which messages are stored.
* A **Kafka producer** publishes messages to a topic, which may be subscribed by zero or more **consumers**.
* As shown in the figure, the Kafka cluster maintains a **partitioned log** for each **topic**. Each of the partitions contains messages or records in an immutable ordered sequence.

##### **Partitions**

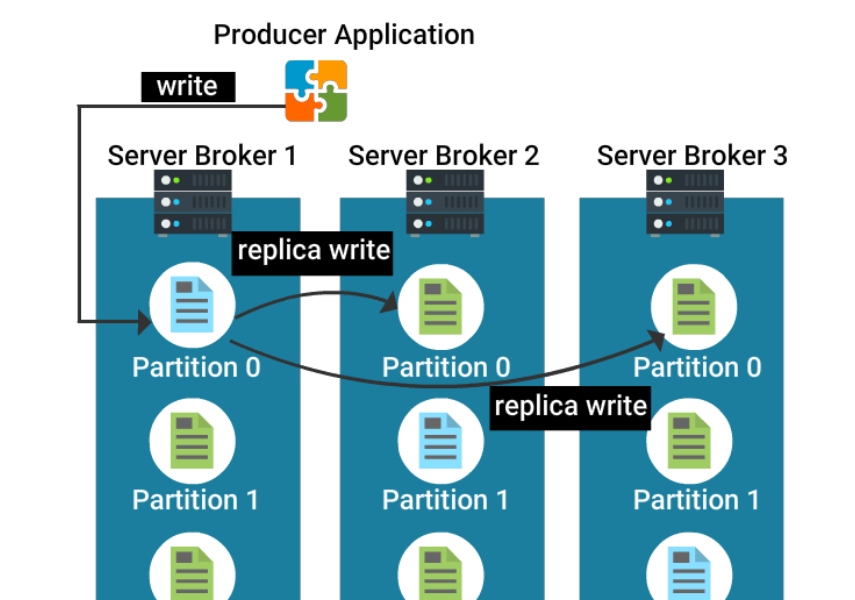
A **topic partition** is a structured commit log to which the records are continually appended. For each topic, Kafka keeps a minimum of one partition.

* Each record in the partition is assigned a sequential id called as the **offset**, which uniquely identifies each of them within the partition.
* The partitions enable the topic to scale beyond a single server and act as the unit of parallelism.

##### **What is a Producer ?**

* Kafka producers **publish messages** to one or more Kafka topics.
* Every time a producer sends a message to a broker, the broker **appends them** to the corresponding topic’s partition. Producers can also send messages to a partition of their choice.
* Producers write to a single leader so that each write is served by a separate broker which helps in **load balancing**.

##### **How does a Producer Work ?**



The image shows a producer writing to partition 0 of a topic, present in broker 1.

Being a leader partition 0 replicates that write to the available followers - broker 2 & broker 3.

When each replica acknowledges that it has received the message, the system is in sync.

##### **What is a Consumer ?**

* A consumer subscribes to a topic and consumes published messages by pulling data from the brokers.
* Consumers read from a single partition so that you can scale the throughput of message consumption similar to message production.
* If the number of consumers is more than the number of partitions then some consumers will remain idle as they have no partitions to read from.
* If the number of partitions is greater than the number of consumers, then each consumer will receive messages from multiple partitions.
* If the number of consumers is equal to the number of partitions, then each consumer reads messages in order from exactly one partition.

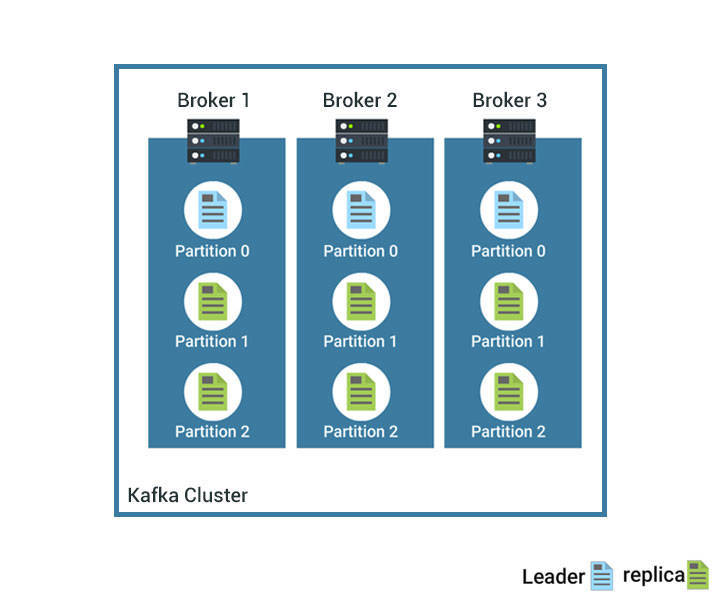
##### **Consumers & Consumer Groups**

Consumers can be organized into **consumer groups** for a given topic.

Each message published on a topic will be delivered to one consumer instance within each subscribed consumer group. These consumer instances may either be in separate processes or on separate machines.

* If all the consumer instances are within the same consumer group, then the records will be load balanced over the instances.
* If all the consumer instances are within different consumer groups, then each record will be broadcast to all the consumer processes.
* The **Offset or position of a consumer** in the partition log is the only metadata retained for that consumer.
* The consumer controls the offset.
* When a consumer reads records, the offset advances linearly along the partition log.
* The consumer can read data from any position in the partition log - it can move back to an older offset to re-read older data or jump ahead to the latest record and start consuming from there.

##### **Kafka Broker**



* Being a **distributed system**, Kafka runs in a **cluster** of machines, where each node in the cluster is called a Kafka **broker**.
* A Kafka cluster is a Kafka distribution with more than one broker.
* A Kafka cluster will expand without downtime.
* Each broker may hold zero or more partitions of a topic. For example, if you have a topic with 24 partitions and a cluster with 3 Kafka brokers, each one will hold 8 partitions of the topic.
* Kafka and Zookeeper will handle the load distributions among these partitions and redistribute them correctly when any broker goes down.