**What is Apache Kafka?**

**Apache Kafka** is a distributed, high-throughput, fault-tolerant messaging system designed for handling real-time data streams. Initially developed by **LinkedIn**, it was open-sourced and is now part of the **Apache Software Foundation**. Kafka is widely used for building real-time streaming data pipelines, applications that process real-time event streams, and for log aggregation.

Kafka provides:

* **Message queueing** and **publish-subscribe** functionality.

**Core Concepts in Apache Kafka**

1. **Topics**: Kafka organizes messages into categories called **topics**. A topic is a logical channel to which producers send messages, and consumers read from. Kafka topics are partitioned, meaning they can be split into multiple partitions for parallel processing.
2. **Producers**: **Producers** are the applications or services that publish (write) messages to Kafka topics. Producers push data to a specific topic.
3. **Consumers**: **Consumers** are applications or services that subscribe to topics and process messages. A consumer reads messages from Kafka topics in a **pull-based** model (consumers fetch messages when they are ready).
4. **Brokers**: A **Kafka broker** is a server that runs the Kafka software. Brokers store messages in topic partitions and serve client requests. Multiple brokers form a Kafka cluster, which ensures load distribution and fault tolerance.
5. **Partitions**: Each topic in Kafka is split into multiple **partitions**. This allows parallelism as each partition can be consumed by a different consumer. Partitions also provide ordering guarantees—within a partition, messages are ordered, but across partitions, the order is not guaranteed.
6. **Replication**: Kafka supports **replication** to ensure fault tolerance. Each partition has a set of replicas, and at least one of the replicas is designated as the leader. The leader handles all read and write requests for that partition, while the other replicas act as backups.
7. **Offset**: Messages in Kafka are identified by a unique **offset**, which is a sequential number assigned to messages in a partition. Consumers keep track of the offset of the last message they read, allowing them to continue reading from where they left off.
8. **Consumer Groups**: Kafka allows multiple consumers to form a **consumer group**. Each message in a topic partition is delivered to only one consumer within a group, enabling load balancing among consumers. This ensures that messages are processed only once (message delivery is split among the consumers in the group).
9. **Zookeeper**: **Zookeeper** is used by Kafka for managing distributed brokers, maintaining metadata, and providing leader election among brokers. However, newer versions of Kafka (starting from 2.8) are transitioning away from Zookeeper with an internal "KRaft" (Kafka Raft) architecture for metadata management.

**How Apache Kafka Works**

1. **Producers Send Messages**: Producers write messages to Kafka topics. A producer specifies the topic and may also specify which partition to send the message to. Kafka brokers receive the messages and store them in the appropriate partition.
2. **Kafka Stores Messages**: Kafka stores messages in a log format, where each partition is an ordered sequence of messages. These messages are persisted on disk and replicated across multiple brokers for durability and fault tolerance.
3. **Consumers Read Messages**: Consumers subscribe to topics and request messages from Kafka brokers. Consumers can read messages at their own pace (asynchronously). Kafka keeps track of the **offset**, allowing consumers to resume reading from the last consumed message.
4. **Message Retention**: Kafka is not just a message queue but also a storage system. It retains messages for a configurable period (e.g., 7 days). Even after a consumer reads a message, it is not removed from Kafka until it exceeds the retention period. This makes Kafka ideal for log processing and auditing.
5. **Consumer Group Mechanism**: If multiple consumers form a group, Kafka assigns partitions to consumers within the group. This ensures parallelism and fault tolerance. For example, if one consumer fails, another consumer can take over the partitions of the failed consumer and resume processing.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Kafka Consumer (Java)**: Here's an example of a Kafka consumer:

A screen shot of a computer program

Description automatically generated