Apache Kafka

TEAM MEMBERS

Achyuth Mukund - 22011102005
Harini NS - 22011102021
Balasubramanian I - 22011102025

Haneef Ahmad - 22011102019
Haytham RA - 22011102024
Divya M - 22011102047
Nikhil Krishnan S - 22011102063

What is Kafka?

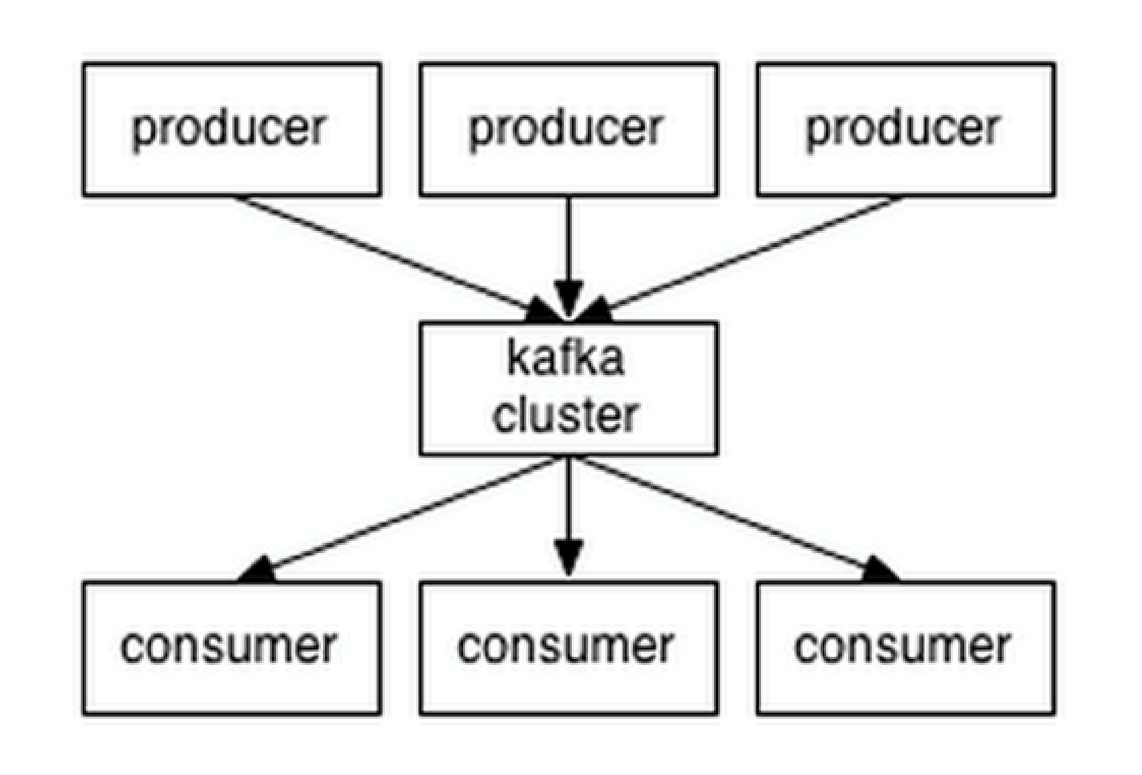
- Apache Kafka is a free, open-source platform for storing and processing streams of data in real time. It is a high-performance distributed platform designed for the efficient and reliable exchange of data between applications, microservices, and systems.
- It enables organizations to handle large volumes of real-time data by functioning as a messaging system, where one component produces data and another consumes it.

Kafka is a "**publish-subscribe messaging rethought as a distributed commit log".**

- **Publish-subscribe messaging:** A system where messages are sent (published) by one component and received (subscribed to) by another.
- **Distributed commit log:** A way of storing data where messages are written sequentially and never modified.

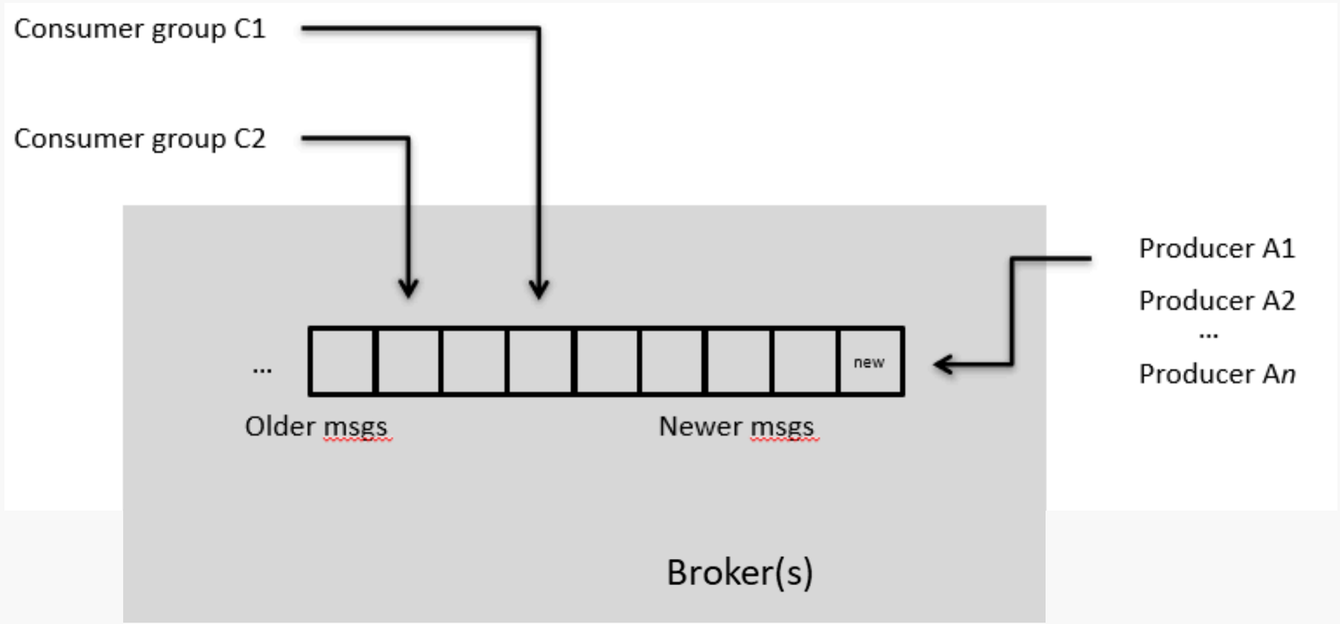
A First Look

- **Producers** – Components that send (write) data into Kafka.
- **Brokers** (Kafka Cluster) – Servers that store and distribute data.
- **Consumers** – Components that read (consume) data from Kafka.



- Data is stored in topics.
- In Apache Kafka, a topic serves as a category or logical channel where messages are stored.
- Partition = ordered sequence of messages that can be added to but not changed.
- The number of partitions controls how many consumers can process the data in parallel.

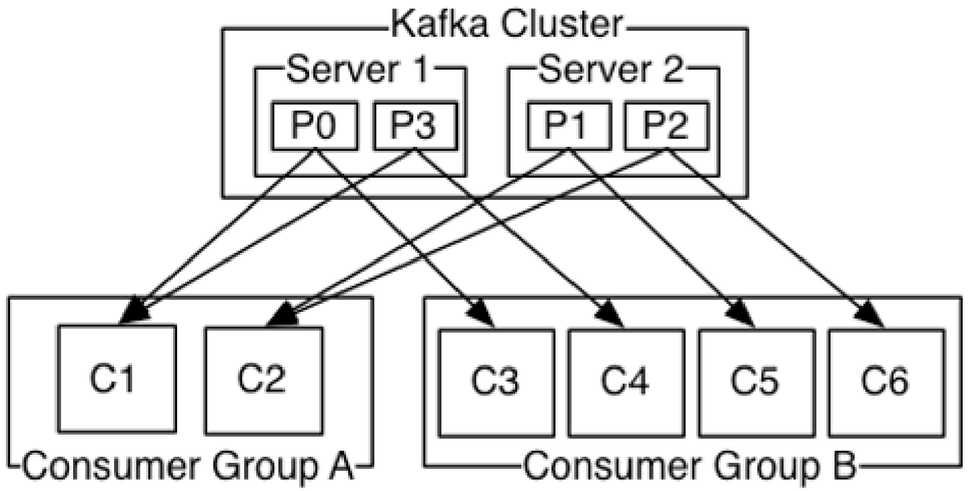
Topics



- When a producer sends a message to a topic, it is appended to the end of a partition in a sequential manner.
- Kafka also provides message retention policies, allowing it to prune (delete) messages based on factors such as age (time-based expiration), maximum storage size, or specific keys, ensuring efficient data management.
- Consumers use an “offset pointer” to track/control their read progress.

Partitions

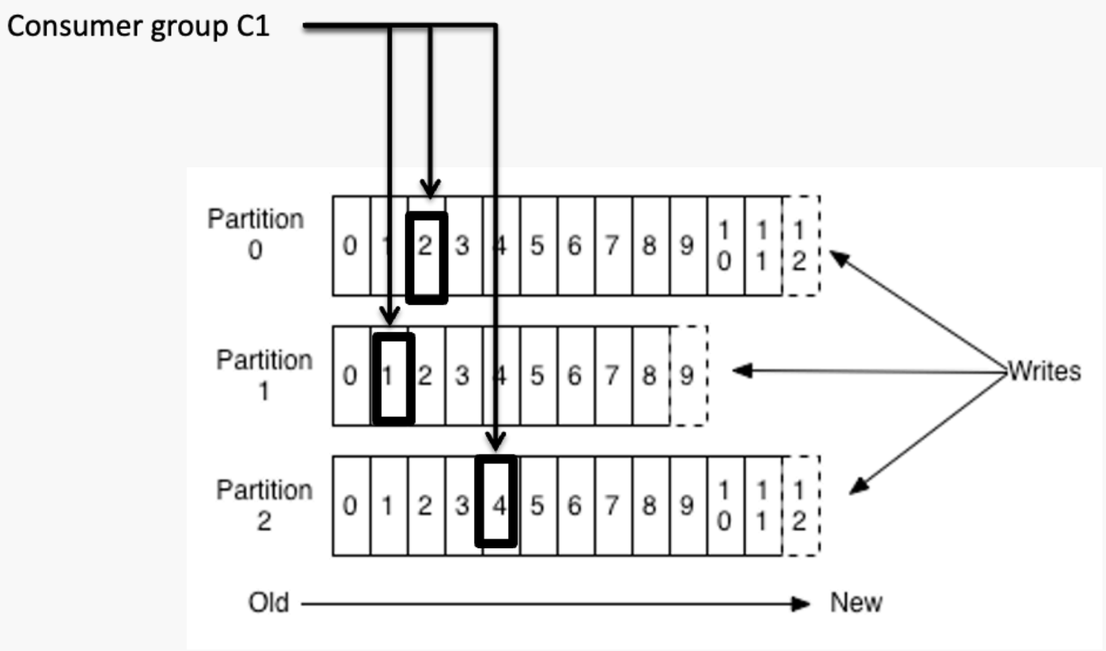
- Partitions are ordered immutable sequence of messages.
- Each message in a partition has a unique offset (ID). Consumers keep track of the last message they read using (offset, partition, topic).
- This allows consumers to continue from where they left off.
- The image shows how Kafka consumer groups read data from a Kafka cluster with multiple partitions



– Consumer group A, with 2 consumers, reads from a 4-partition topic
– Consumer group B, with 4 consumers, reads from the same topic

Partition offsets

A partition offset is a unique identifier assigned to each message within a Kafka partition. It represents the position of the message in the partition's log.

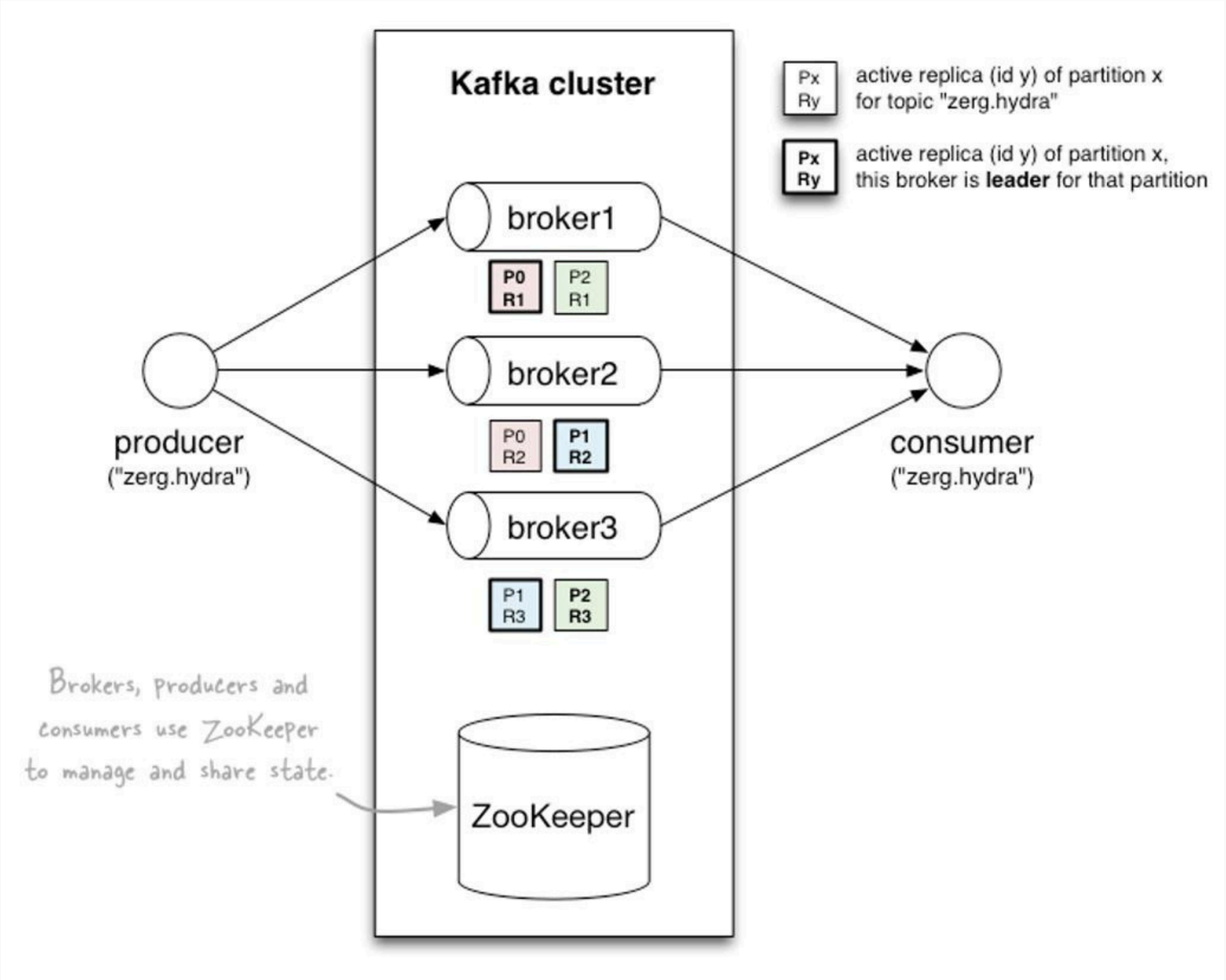


- Unique & Sequential – Offsets are monotonically increasing within a partition.
- Partition-Specific – The same offset number can exist in different partitions of a topic.
- Used for Consumer Tracking – Consumers track offsets to resume reading from where they left off.
- Not Reused – Even if a message is deleted (due to retention policies), its offset remains unique.
- Stored in Kafka – Consumer offsets are recorded in the __consumer_offsets topic.

Replicas of a partition

Replicas: “backups” of a partition

- A replica is an exact copy of a partition stored on a different Kafka broker.
- Its primary function is to prevent data loss in case a broker fails.
- Unlike leader partitions, replica partitions do not handle producer or consumer requests.
- This means they do not improve performance or parallelism—their only job is to provide fault tolerance.
- Kafka tolerates failures based on the number of replicas
- Kafka can handle (numReplicas - 1) broker failures before data loss occurs.
- Example:
    - If numReplicas = 2, at most 1 broker can fail without losing data.
    - If numReplicas = 3, then 2 brokers can fail before data loss occurs.



ZooKeeper manages the state of the Kafka cluster. It keeps track of broker metadata, leader election, and consumer offsets to ensure smooth coordination.

Key features:

- Fast – Can handle millions of messages per second.
- Scalable – Can easily grow by adding more machines.
- Durable – Ensures data is not lost.
- Distributed – Works across multiple computers.

Use Cases:

- LinkedIn: activity streams, operational metrics, data bus
  - –400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s), May 2014
- Netflix: real-time monitoring and event processing
- Twitter: as part of their Storm real-time data pipelines
- Spotify: log delivery (from 4h down to 10s), Hadoop