Apache Kafka – Complete Interview Notes

1. What is Apache Kafka?

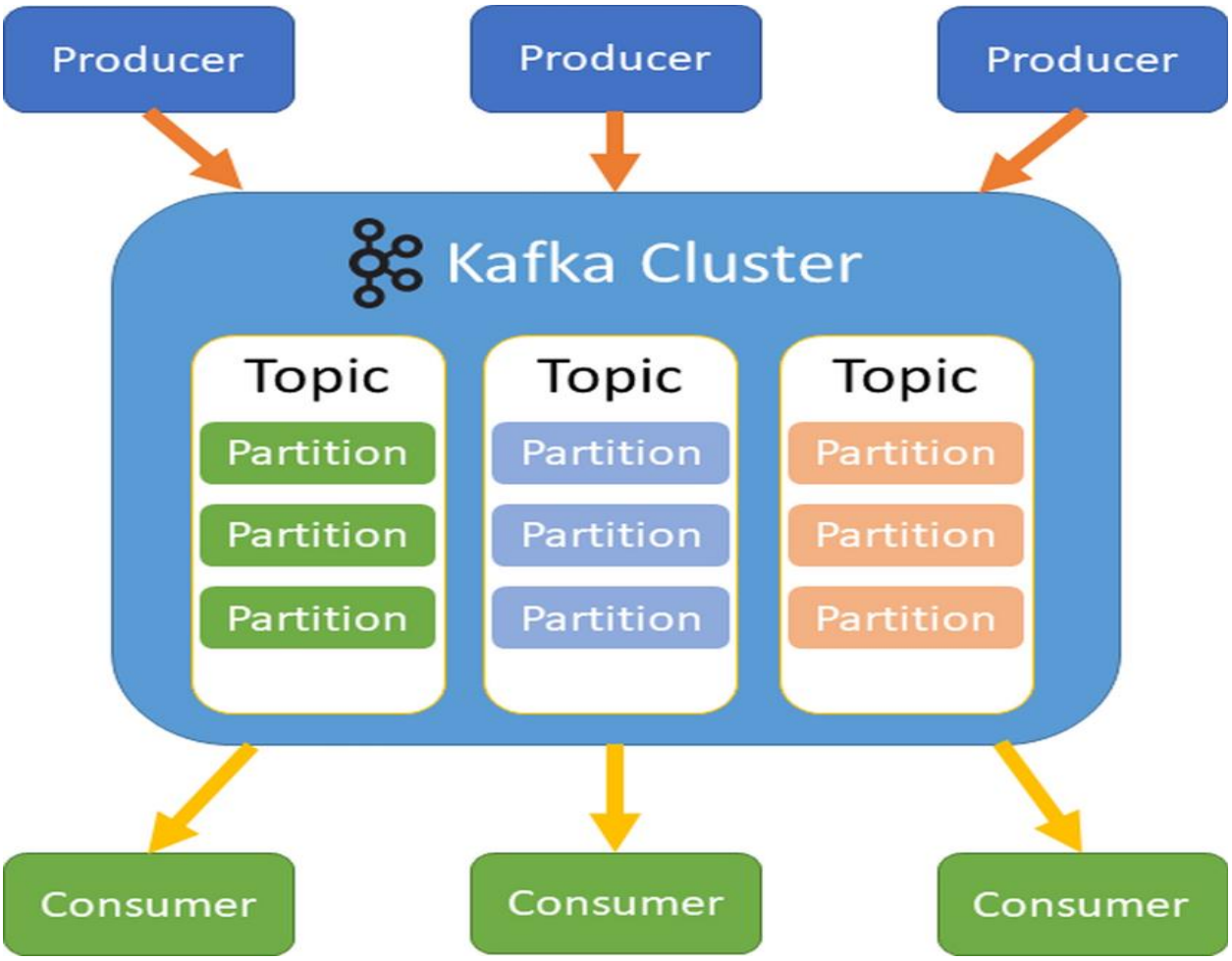
Apache Kafka is a distributed event-streaming platform designed to handle high throughput, low-latency, real-time data streams. It acts as a messaging system where producers send data to topics and consumers read data from topics.

2. Why Kafka?

- - High throughput
- - Horizontal scalability
- - Fault tolerance
- - Distributed architecture
- - Persisted logs
- - Used heavily in real-time analytics and microservices

3. Kafka Architecture

Kafka follows a distributed, publish-subscribe messaging model.



1. Brokers

- A **broker** is a Kafka server that stores data and serves client requests.
- A Kafka cluster is made up of **multiple brokers** to ensure fault tolerance and scalability.
- Each broker is identified by a unique **broker ID**.
- Responsibilities:
    - Store and serve messages.
    - Handle **produce** (write) and **consume** (read) requests.
    - Manage **leader/follower roles** for partitions.
    - Replicate data to other brokers for fault tolerance.
- Example: In a 3-broker cluster, topic data may be replicated across all brokers so that if one broker fails, the data is still available.

2. Topics

- A **topic** is a logical channel to which data is sent.
- Messages are published to a topic, and consumers subscribe to topics to receive messages.
- Each topic can have **multiple partitions**.

- Topics provide **decoupling** between producers and consumers.

3. Partitions

- A **partition** is a log inside a topic. It is the unit of parallelism and scalability in Kafka.
- Features:
    - Each partition is **ordered** and immutable.
    - Messages in a partition have a unique **offset** (like a sequence number).
    - Partitions can be **replicated** across multiple brokers for fault tolerance.
- Partitioning allows Kafka to **scale horizontally**:
    - Multiple consumers can read from different partitions simultaneously.
    - Load is distributed across brokers.

4. ZooKeeper (before Kafka 3.3)

- ZooKeeper was used for **cluster management and metadata storage**.
- Responsibilities:
    - Keeping track of broker nodes and their status.
    - Storing **topic and partition metadata**.
    - Managing **controller election**.
- In **Kafka 3.3+**, ZooKeeper is optional with **KRaft mode** (Kafka Raft Metadata mode), which replaces ZooKeeper.

5. Controllers

The **controller** is a special broker in the Kafka cluster responsible for **managing the cluster state**. There is **only one active controller at a time**, selected among brokers.

Responsibilities of the Controller:

1. **Partition Leader Election**
    a. For each partition, one broker acts as a **leader** and handles all reads/writes.
    b. If a broker fails, the controller s**elects a new leader** for the affected partitions.
2. **Cluster Metadata Management**
    a. Keeps track of which broker hosts which partition and its replicas.
    b. Updates cluster metadata for all brokers and clients.
3. **Handling Broker Failures**
    a. Detects when brokers go down (via ZooKeeper or KRaft).

    b. Triggers leader reassignment and replica synchronization.
4. **Topic Management**
    a. Creates or deletes topics and partitions.
    b. Ensures replication factors are respected.
5. **Replication Management**
    a. Ensures followers replicate data from the partition leaders.
6. **Administrative Tasks**
    a. Orchestrates reassignments of partitions for balancing.
    b. Handles throttling and recovery in case of broker failures.

Important Kafka Concepts

1. **Topic**
    a. A **named stream of data** in Kafka.
    b. Producers write messages to topics; consumers read from topics.
    c. Example: A topic named `orders` could store all customer orders.
2. **Partition**
    a. A **subset of a topic** that allows parallel processing and scalability.
    b. Each partition is **ordered** and immutable.
    c. Example: `orders` topic might have 3 partitions to distribute load among brokers.
3. **Offset**
    a. The **unique position of a message** within a partition.
    b. Helps consumers track which messages have been read.
    c. Example: The first message in partition 0 has offset `0`, the next is `1`, and so on.
4. **Broker**
    a. A **Kafka server** that stores partitions and handles client requests.
    b. A cluster has **multiple brokers** for fault tolerance.
    c. Example: Broker 1 might store partitions 0 and 1 of `orders`.
5. **Cluster**
    a. A **group of brokers** working together.
    b. Provides reliability and scalability.
    c. Example: A 3-broker cluster can replicate partitions so data isn't lost if one broker fails.
6. **Leader Partition**
    a. The **main replica of a partition** that handles all read and write operations.
    b. Each partition has **one leader** at any time.
    c. Example: Partition 0 of `orders` might have Broker 2 as its leader.
7. **Follower Partition**
    a. **Replicas of the leader partition** that stay in sync.
    b. Do **not serve client requests**, only replicate data for fault tolerance.
    c. Example: Partition 0 on Broker 1 and Broker 3 could be followers of leader on Broker 2.
8. **Consumer Group**
    a. A **group of consumers** that divide the partitions among themselves.

b. Ensures **load balancing**: each partition is read by only one consumer in the group.
c. Example: `group-A` has 3 consumers reading 3 partitions; each consumer gets one partition.

Kafka Interview Questions

Q1. What is Kafka?

- Apache Kafka is a **distributed streaming platform**.
- It allows **publishing, storing, and consuming streams of records** in real-time.
- Key features: **high throughput, fault tolerance, scalability, durability**.

Q2. Difference between Kafka and RabbitMQ

Feature	Kafka	RabbitMQ
Messaging model	Pub/Sub & Stream processing	Queue-based (traditional)
Message storage	Persistent log	In-memory / persistent queue
Scalability	Horizontal (partitions)	Limited by queues
Throughput	Very high	Moderate
Delivery semantics	At-least-once / exactly-once	At-most-once / at-least-once

Q3. What is a partition?

- A **partition is a subset of a topic**, allowing parallelism and scalability.
- Messages in a partition are **ordered** and identified by an **offset**.

Q4. How does Kafka achieve fault tolerance?

- **Replication:** Each partition has a **leader** and one or more **followers**.
- **Leader election:** If a leader fails, a follower becomes the new leader.
- **In-Sync Replicas (ISR):** Only followers in sync with leader can be elected as new leader.

Q5. Explain Kafka consumer groups

- A **consumer group** is a set of consumers that work together to read a topic.
- Each partition is consumed by **only one consumer per group**, ensuring **load balancing**.
- Multiple groups can read the same topic independently.

Q6. What is ISR (In-Sync Replicas)?

- ISR is the set of replicas that are **fully caught up** with the leader.
- Only replicas in the ISR can be **elected as leader** if the current leader fails.

Q7. What is Kafka offset?

- Offset is the **unique identifier of a message in a partition**.
- Consumers use it to **track read position**.
- Allows **replay** of messages by resetting the offset.

Q8. How to ensure exactly-once delivery?

- Kafka provides **exactly-once semantics (EOS)** using:
  - **Idempotent producers**: prevent duplicate writes.
  - **Transactional writes**: atomic writes to multiple partitions/topics.
  - **Consumer offsets committed within transactions**.

Q9. What is the role of ZooKeeper? (old Kafka)

- ZooKeeper was used to:
  - Manage **cluster metadata** (broker status, topic/partition info).
  - Elect the **controller broker**.
  - Monitor **broker health**.
- Note: Kafka 3.3+ can run **without ZooKeeper** using KRaft mode.

Q10. Difference between Kafka Streams and Kafka Connect

Feature	Kafka Streams	Kafka Connect
Purpose	Stream processing (real-time)	Data integration / ETL
API	Java library	Connector framework
Deployment	Embedded in app	Standalone / distributed
Transformation	Complex business logic	Simple ETL, source/sink only

12. Additional Notes for Kafka Interviews

- **Kafka is NOT a database**
  - It is a **streaming platform**, designed for **real-time data pipelines** and **event streaming**, not long-term storage or complex queries.
- **Kafka stores data on disk (not memory!)**
  - Data is **persisted to disk** for durability.
  - Kafka uses **sequential disk I/O**, making it extremely fast.
- **Messages are immutable**
  - Once a message is written to a topic, it **cannot be changed**.
  - Ensures reliability and consistency.
- **Kafka is pull-based**
  - Consumers **pull messages** from brokers at their own pace, unlike push-based systems.
  - Gives consumers **control over processing speed**.

- **Consumers maintain offsets**
  - Each consumer tracks the **last read message offset**.
  - Allows **message replay** and **fault-tolerant consumption**.
- **Horizontal scaling = add more partitions**
  - To increase throughput, **add more partitions** across brokers.
  - Partitions allow **parallel processing** by multiple consumers.
- **Perfect for microservices + event-driven architecture**
  - Kafka acts as an **event backbone**, enabling decoupled, asynchronous communication between services.
  -