**KAFKA**

It is a distributed event-streaming platform that is used for building real-time data pipeline and streaming applications.

One of the tools that opened way for microservice architecture

**Event:**

Consists of key, value, timestamp, metadata

Basically events will be fired and forgot by the producers and they no need to wait for response.

**Topic:**

Grouping same type of events.

**Partitions:**

we decide based on schema design

Eg. Orders based on area – US, UK, Asia

**Consumer-groups:**

Instance of consumers -> each consumer in the same group will be assigned each partition enabling parallel consumption.

Group-id is same for all the consumers in the same group

**Kafka-broker:**

Server that stores data(messages) in topics, manages message distribution to consumer.

**Fault-tolerance:**

Topic’s partitions are distributed across multiple brokers.

Each partition has a reader broker and multiple replicas

**KAFKA’s uniqueness when compared to the other message brokers:**

Other message brokers will remove the message once consumed by consumer. In kafka we can set retention period. (how long we need to store the messages)

In real time, Kafka can be compared to Netflix and Rabbitmq/Active MQ to normal television.

**Cluster management:**

Maintaining a registry of all active brokers in cluster.

**Leader election:**

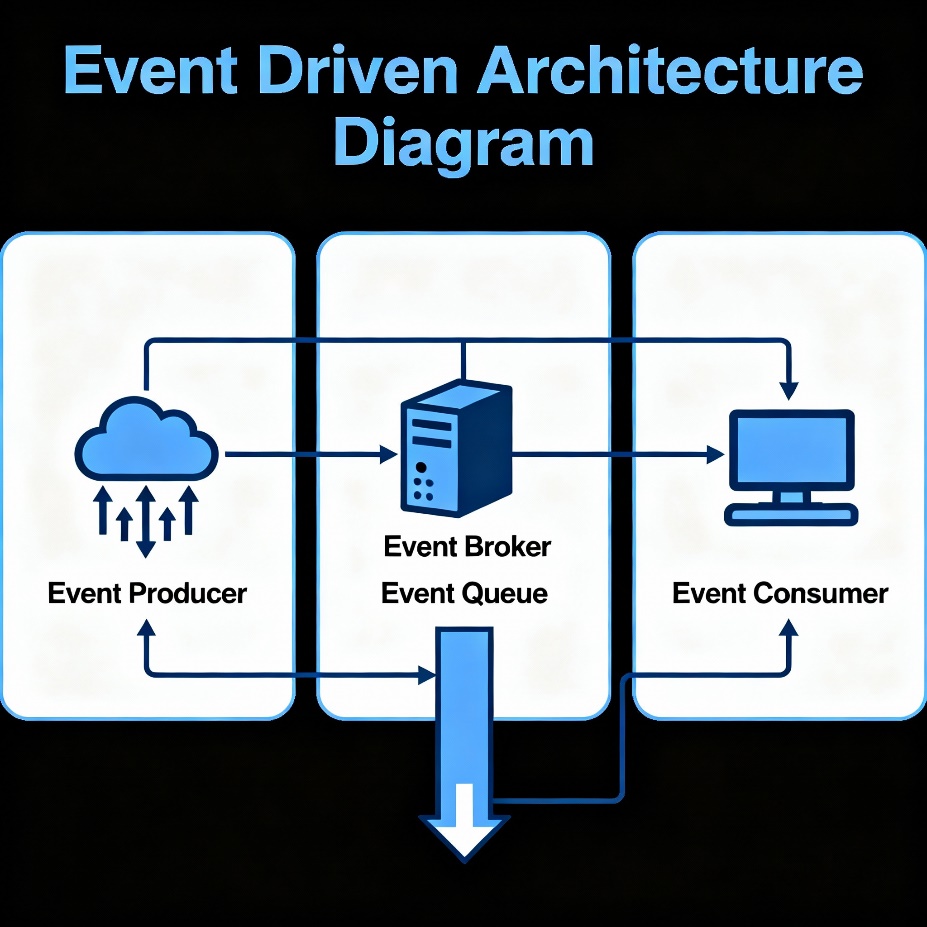
Each kafka partition has a leader broker. Zookeeper facilitates election of a leader.

Zookeeper-> centralized service for managing metdata and co-ordination tasks for distributed systems. It is actually an external dependency of kafka.

**But from kafka version 3.0 -> removed dependency on zookeeper.**

Metadata is now managed natively with kafka-broker.

**Event-driver Architecture:**



Event driven architecture pitfalls before kafka or before any broker services

1. Tight coupling
2. Reduced scalability – struggling to handle growing data and uses demands due to interdependencies.
3. Single point of failure – No backup or redundant instances.
4. No message persistence – Loss of data
5. Limited functionality

**Kafka – Advantages:**

* High throughput – kafka handles massive event streams with ease, ensuring no messages gets left behind.
* Fault tolerance – Guarantees resilience, high availability, even with failures.
* Scalability
* Real-time processing of data

In Kafka, the **foundation for message management, cluster node management, scalability, fault tolerance, and dynamic membership** is the core Kafka cluster architecture, primarily built around the following components:

**1. Broker**

* A broker is a single Kafka server responsible for storing and managing messages.
* Multiple brokers form a Kafka cluster, distributing partitions of topics among themselves.
* The broker is responsible for message management, persistence, and serving client (producer/consumer) requests.

**2. Cluster**

* A Kafka cluster consists of multiple brokers working together.
* This setup allows horizontal scaling (adding more brokers increases capacity) and built-in fault tolerance because data (topic partitions) can be replicated across brokers.
* There is always one broker acting as the controller (or leader) that manages cluster metadata and orchestrates dynamic membership (adding/removing brokers/partitions).

**3. Topics and Partitions**

* Topics provide logical separation of message streams.
* Each topic is divided into partitions, which are distributed across brokers for scalability and parallelism.
* Partitions are the foundation for both scalability (allowing parallel reads/writes) and fault tolerance (through replication).

**4. Replication**

* Each partition can be replicated across multiple brokers.
* One broker serves as the leader for that partition; others act as followers.
* This ensures messages are available even if a broker fails, providing fault tolerance.

**5. Controller (metadata management)**

* The controller (which now may be handled by KRaft in new versions, or by Zookeeper in legacy Kafka) elect leaders for partitions, tracks cluster changes, and manages dynamic membership.
* The controller coordinates operations like adding/deleting brokers and partitions for automatic rebalancing and resilience.

**Advantages of topic:**

* Message categorization – based on key, if same key -> same partitions
* Immutable logs – message in a topic are stored sequentially
* Multi-consumer access – Multiple consumers can read from same topic
* Decouple communication – Producer and consumers don’t interact directly but through topics
* Replication for higher availability

**Single Broker limitation:**

Storing an entire topic across one broker restricts its size and availability to the capacity of the single machine.

**Partitioning for scalability:**

Kafka partitions distribute a topic across multiple brokers, enabling horizontal scalability, fault tolerance and higher throughput.

**Advantages of partitioning of the topic:**

* Distribution
* Parallelism – parallel processing of messages
* Scalability
* Fault tolerance
* Ordering – messages within the partition are strictly ordered (FCFS)