



# Kafka: Deep Dive

As of now, we have discussed the core concepts of Kafka. Let us now throw some light on the workflow of Kafka.

We'll cover the following ^

- Topic partitions
  - Leader
  - Follower
  - In-sync replicas
- High-water mark

Kafka is simply a **collection of topics**. As topics can get quite big, they are **split into partitions** of a smaller size for better performance and scalability.

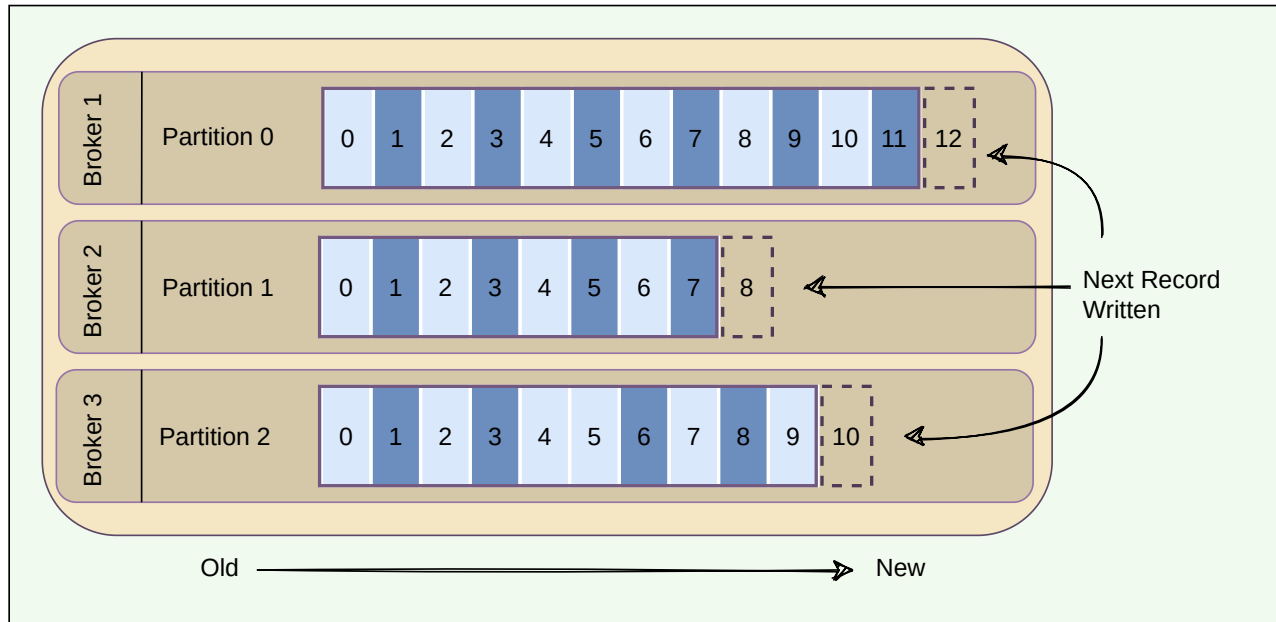
## Topic partitions#

Kafka topics are partitioned, meaning a topic is spread over a number of 'fragments'. Each partition can be placed on a separate Kafka broker.

When a new message is published on a topic, it gets appended to one of the topic's partitions. The producer controls which partition it publishes messages to based on the data. For example, a producer can decide that all messages related to a particular 'city' go to the same partition.


Essentially, a partition is an ordered sequence of messages. Producers continually append new messages to partitions. Kafka guarantees that all messages inside a partition are stored in the sequence they came in.

**Ordering of messages is maintained at the partition level, not across the topic.**



A topic having three partitions residing on three brokers

- A unique sequence ID called an **offset** gets assigned to every message that enters a partition. These numerical offsets are used to identify every message's sequential position within a topic's partition.
- Offset sequences are unique only to each partition. This means, to locate a specific message, we need to know the Topic, Partition, and Offset number.
- Producers can choose to publish a message to any partition. If ordering within a partition is not needed, a round-robin partition strategy can be used, so records get distributed evenly across partitions.
- Placing each partition on separate Kafka brokers enables multiple consumers to read from a topic in parallel. That means, different consumers can concurrently read different partitions present on separate brokers.

- Placing each partition of a topic on a separate broker also enables a topic to hold more data than the capacity of one server. 
- Messages once written to partitions are **immutable** and cannot be updated.
- A producer can add a '**key**' to any message it publishes. Kafka guarantees that messages with the same key are written to the same partition.
- Each broker manages a set of partitions belonging to different topics.

Kafka follows the principle of a **dumb broker** and **smart consumer**. This means that Kafka does not keep track of what records are read by the consumer. Instead, consumers, themselves, poll Kafka for new messages and say what records they want to read. This allows them to increment/decrement the offset they are at as they wish, thus being able to replay and reprocess messages. Consumers can read messages starting from a specific offset and are allowed to read from any offset they choose. This also enables consumers to join the cluster at any point in time.

Every topic can be replicated to multiple Kafka brokers to make the data fault-tolerant and highly available. Each topic partition has one leader broker and multiple replica (follower) brokers.

## Leader#

A leader is the node responsible for all reads and writes for the given partition. Every partition has one Kafka broker acting as a leader.

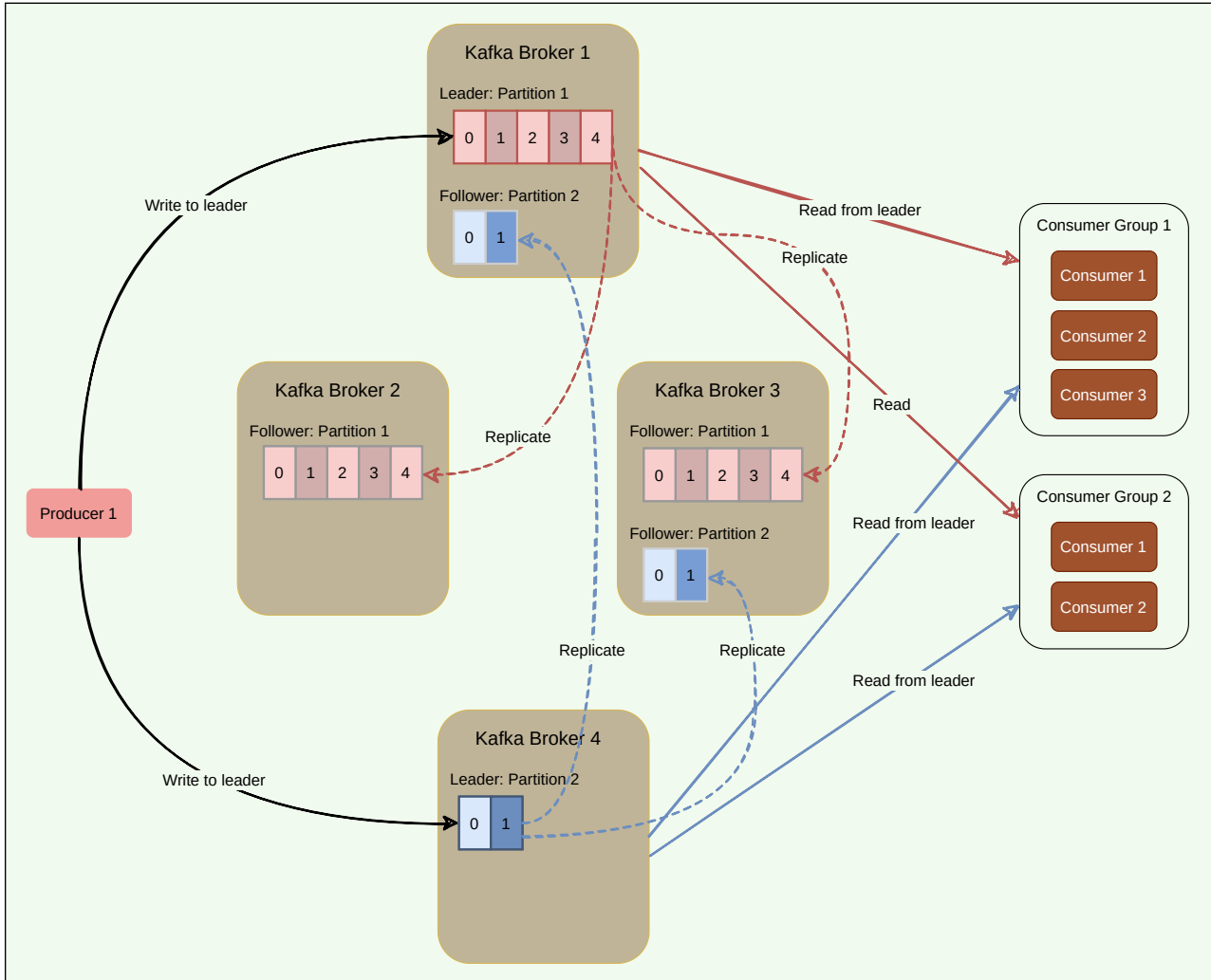
## Follower#

To handle single point of failure, Kafka can replicate partitions and distribute them across multiple broker servers called followers. Each follower's responsibility is to replicate the leader's data to serve as a 'backup' partition. This also means that any follower can take over the leadership if the leader goes down.

In the following diagram, we have two partitions and four brokers.

Broker 1 is the leader of Partition 1 and follower of Partition 2.

Consumers work together in groups to process messages efficiently. More details on consumer groups later.



Leader and followers of partitions

Kafka stores the location of the leader of each partition in ZooKeeper. As all writes/reads happen at/from the leader, producers and consumers directly talk to ZooKeeper to find a partition leader.

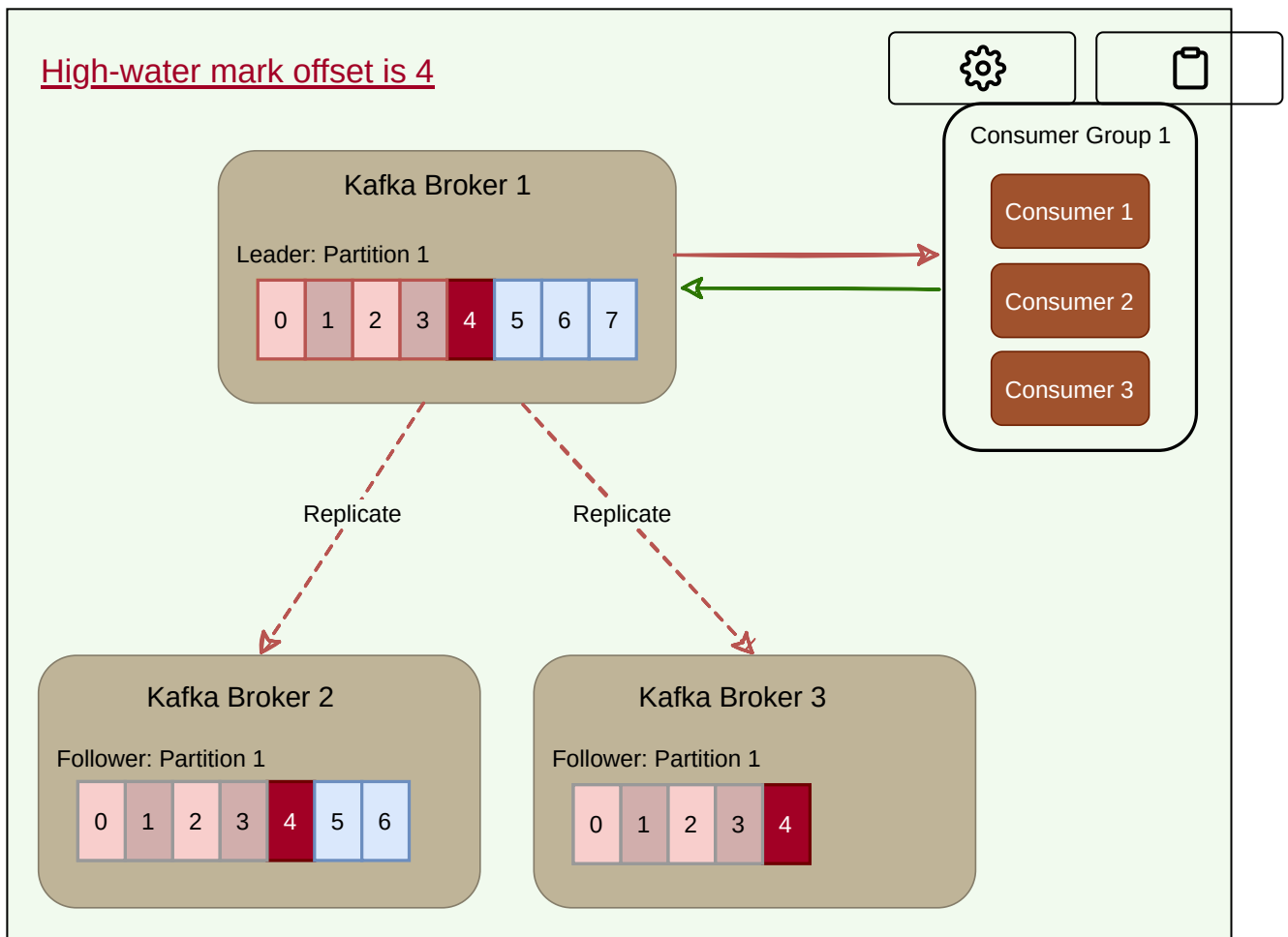
## In-sync replicas#

An in-sync replica (ISR) is a broker that has the latest data for a given partition. A leader is always an in-sync replica. A follower is an in-sync replica only if it has fully caught up to the partition it is following. In other words, ISRs cannot be behind on the latest records for a given partition. **Only ISRs are eligible to become partition leaders.** Kafka can choose the minimum number of ISRs required before the data becomes available for consumers to read.

## High-water mark#

To ensure data consistency, the leader broker never returns (or exposes) messages which have not been replicated to a minimum set of ISRs. For this, brokers keep track of the high-water mark, which is the highest offset that all ISRs of a particular partition share. The leader exposes data only up to the high-water mark offset and propagates the high-water mark offset to all followers. Let's understand this with an example.

In the figure below, the leader does not return messages greater than offset '4', as it is the highest offset message that has been replicated to all follower brokers.



High-water mark offset

If a consumer reads the record with offset '7' from the leader (*Broker 1*), and later, if the current leader fails, and one of the followers becomes the leader before the record is replicated to the followers, the consumer will not be able to find that message on the new leader. The client, in this case, will experience a non-repeatable read. Because of this possibility, Kafka brokers only return records up to the high-water mark.

[← Back](#)[High-level Architecture](#)[Next →](#)[Consumer Groups](#)☒ Mark as Completed[Report an Issue](#)

