KAFKA OPERATION

Đơn vị: Công ty CP Giáo dục và Công nghệ QNET

QNET JOINT STOCK COMPANY

Address: 14th Floor, VTC Online Tower
18 Tam Trinh Street. Hoang Mai District
Hanoi, Vietnam

Quality Network for Education and Technology

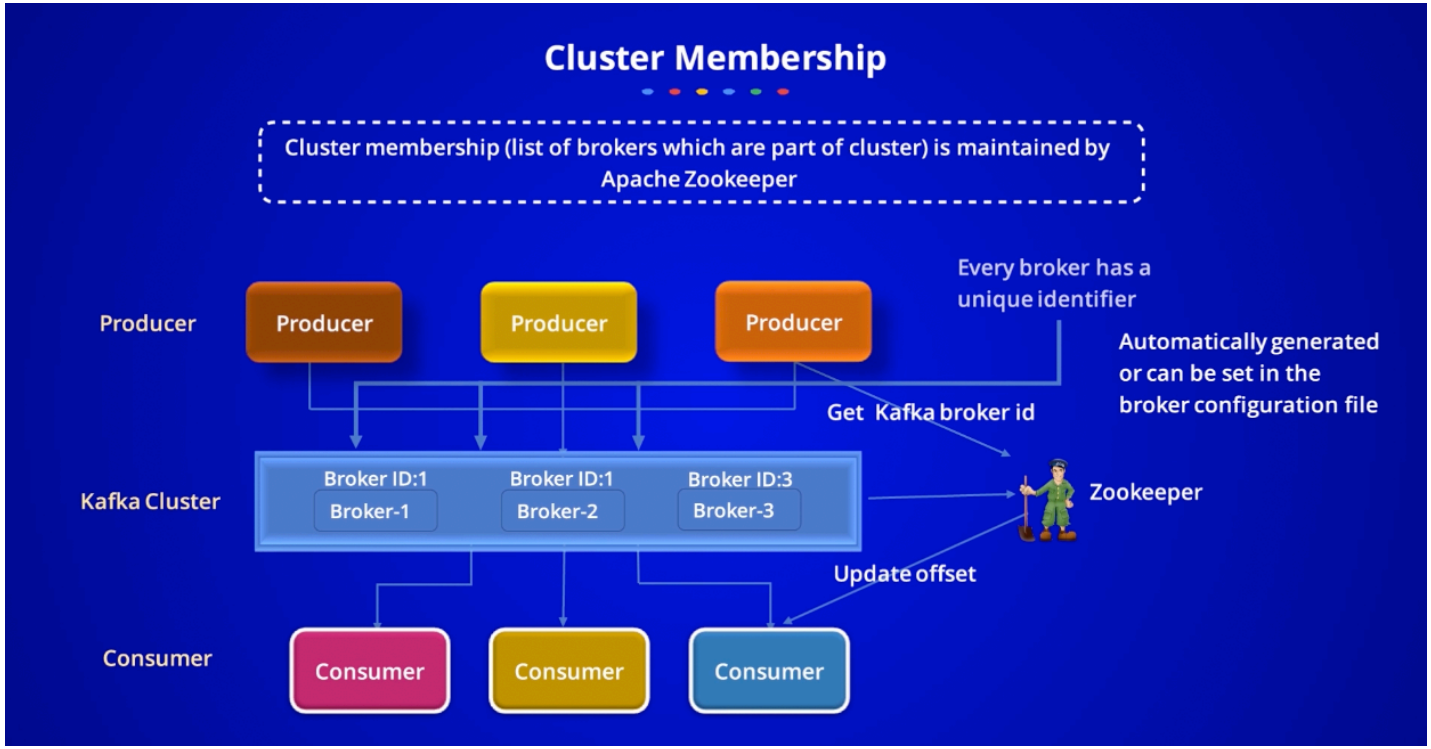
Learning Objective

- Kafka Internal Overview
- Replication-type, Request, Processing
- Storage-Partition allocation, Index, File, Compaction
- Reliability-configuration, Validation

Cluster membership

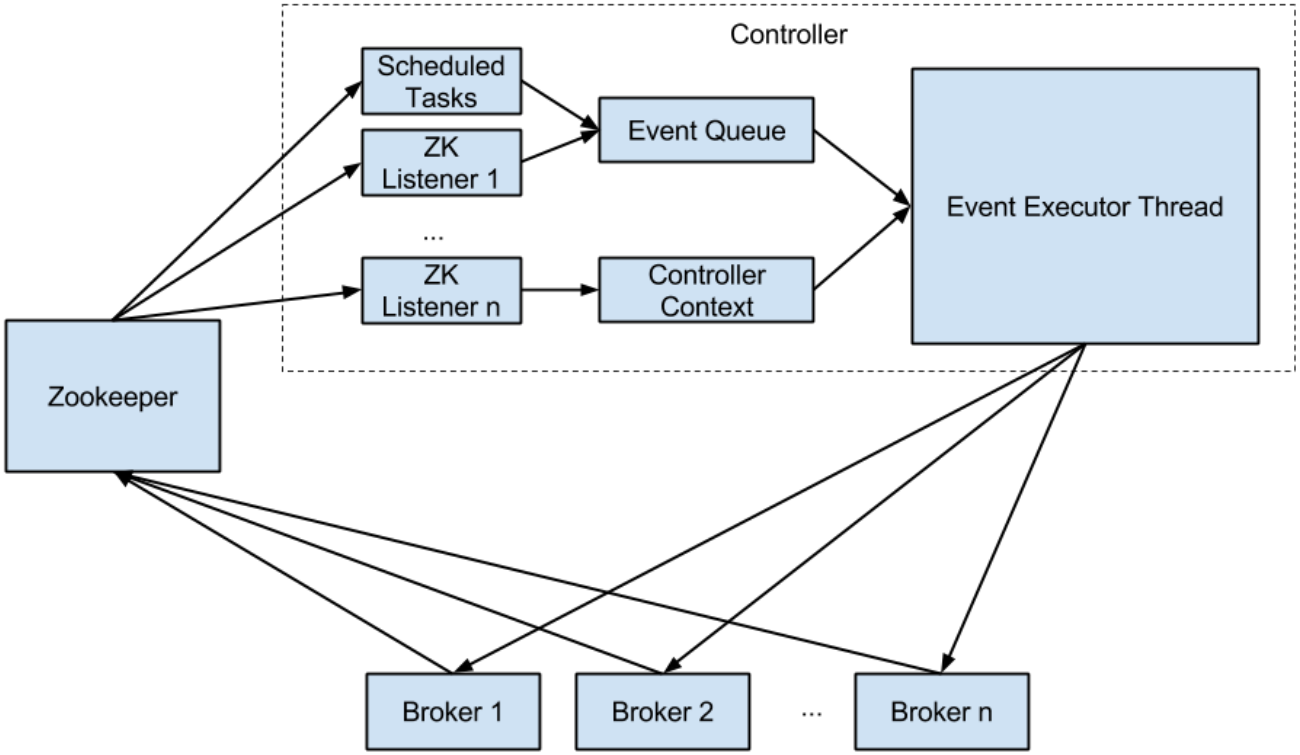
- Kafka use Zookeeper to maintain the list of brokers that are currently member of a cluster
- Every broker has a unique identifier that is either set in the broker configuration file or automatically generated
- Every time a broker process starts, it registers itself with its ID in Zookeeper by creating an *ephemeral node*.
- Kafka brokers, the controller, and some of the ecosystem tools subscribe to the /brokers/ids path in Zookeeper where brokers are registered so that they get notified when brokers are added or removed
- When a broker loses connectivity to Zookeeper, the ephemeral node that the broker created when starting will be automatically removed from Zookeeper. Kafka components that are watching the list of brokers will be notified that the broker is gone

Cluster membership



Kafka Controller

Architecture



Kafka Controller

Functions

- Kafka controller is one of the Kafka brokers that, in addtion to the usual broker functionality, is responsible for electing partition leaders
- The first broker that starts in the cluster becomes the controller by creating an ephemeral node in Zookeeper called /controller
- When other brokers start, they also try to create this node but receive a “node already exists” exception, which causes them to “realize” that the controller node already exists and that the cluster already has a controller
- The brokers create a Zookeeper watch on the controller node so they get notified of changes to this node. This way, we guarantee that the cluster will only have one controller at a time

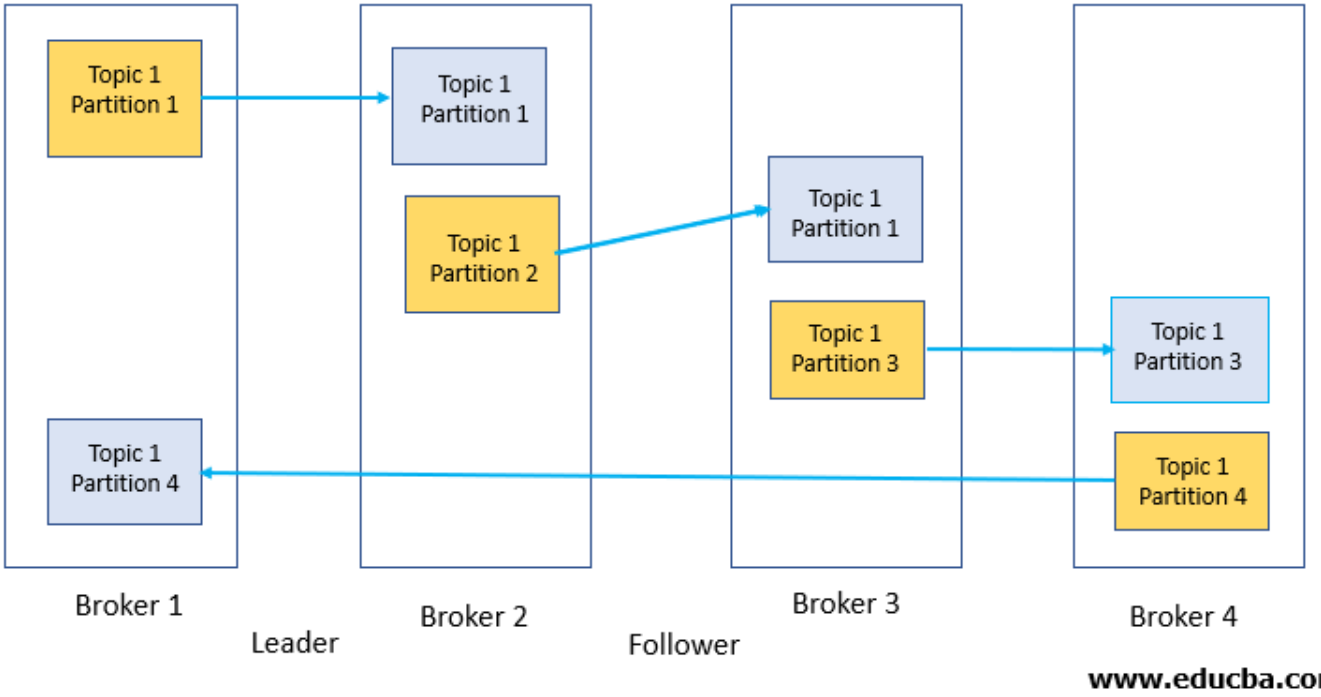
Kafka Controller

Functions

- When the controller broker is stopped or losses connectivity to Zookeeper, the ephemeral node will disappear. This includes any scenario in which the Zookeeper client used by the controller stops sending heartbeats to Zookeeper for longer than *zookeeper.session.timeout.ms*. When the ephemeral node disappears, other brokers in clusters will be notified through the Zookeeper watch that the controller is gone and will attempt to create the controller in Zookeeper themselves.
- The first node to create the new controller in Zookeeper becomes the next controller, while the other nodes will receive a “node already exists” exeception and re-create the watch on the new controller node. Each time a controller is elected, it receives a new, higher controller epoch number through a Zookeeper conditional increment operation. The brokers know the current controller epoch and if they receive a message from a controller with an older number, they knows to ignore it.

Kafka Replication

Flow



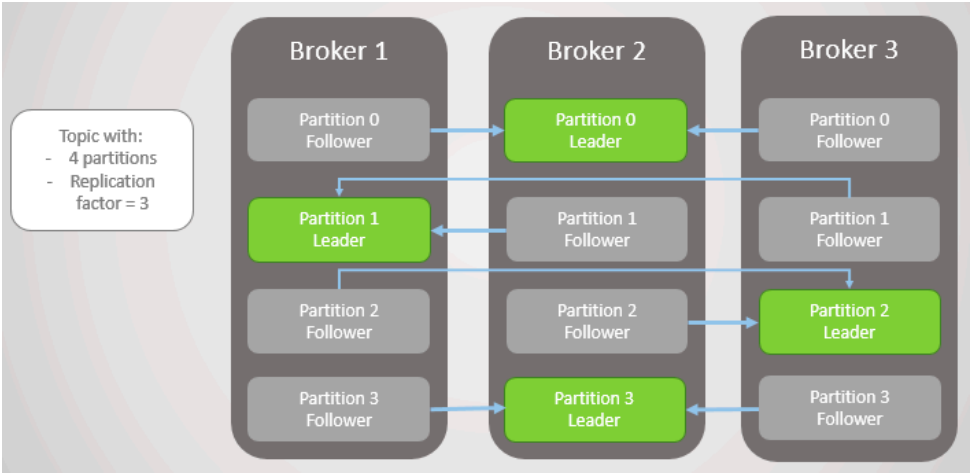
www.educba.com

Kafka Replication

- A replication factor of N allows to reliably read/write data to a topic even if you lost N-1 brokers
- A higher replication factor leads to higher availability, higher reliability and fewer disasters
- To make sure that all replicas are in-sync a daemon thread is required
- Daemon thread is needed because it's all background process or should be part of software itself.

Kafka Replication

Replica types



- Leader replica: each partition has a single replica designed as the leader. All producer requests go through the leader to guarantee consistency. Clients can consume from either the lead replica or its followers
- Follower replica: All replicas for a partition that are not leaders are called followers. Unless configured otherwise, followers don't server client requests; their main jobs is to replicate messages from the leader and stay up-to-date with the most recent messages the leaders has. If a leader replica for a partition crashes, one of the followers replicas will be promoted to become the new leader for the partition

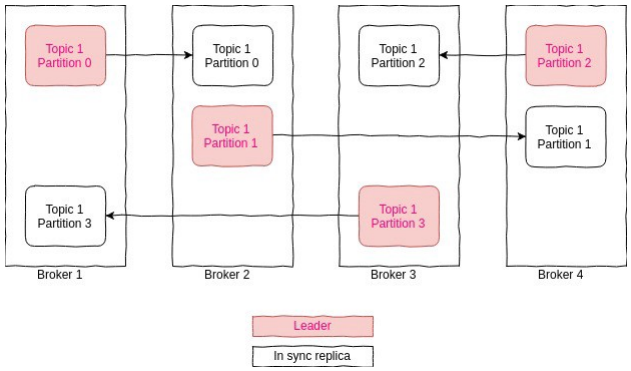
Kafka Replication

Leader Replica

- Responsibility:
  - Receive messages from producer to guarantee consistency
  - Known which of the follower replicas is up-to-date with the leader
- Process to update follower replicas status:
  - Replicas send the leader Fetch requests that contain offset of the message that replica want to receive next
  - Leader sends the messages to the replicas as response to those requests
  - If a replica hasn't requested a message in more than 10 seconds. Or if it has requested messages but hasn't caught up to the most recent message in more than 10 seconds (default), the replica is considered *out of sync*

Kafka Replication

Preferred leader



- Preferred leader is the replica which was leader at the time of topic creation
- It is preferred because when partitions are first created, the leaders are balanced among the brokers. As a result, we expect that when the preferred leader is indeed the leader for all partitions in the cluster, load will be evenly balanced between brokers.

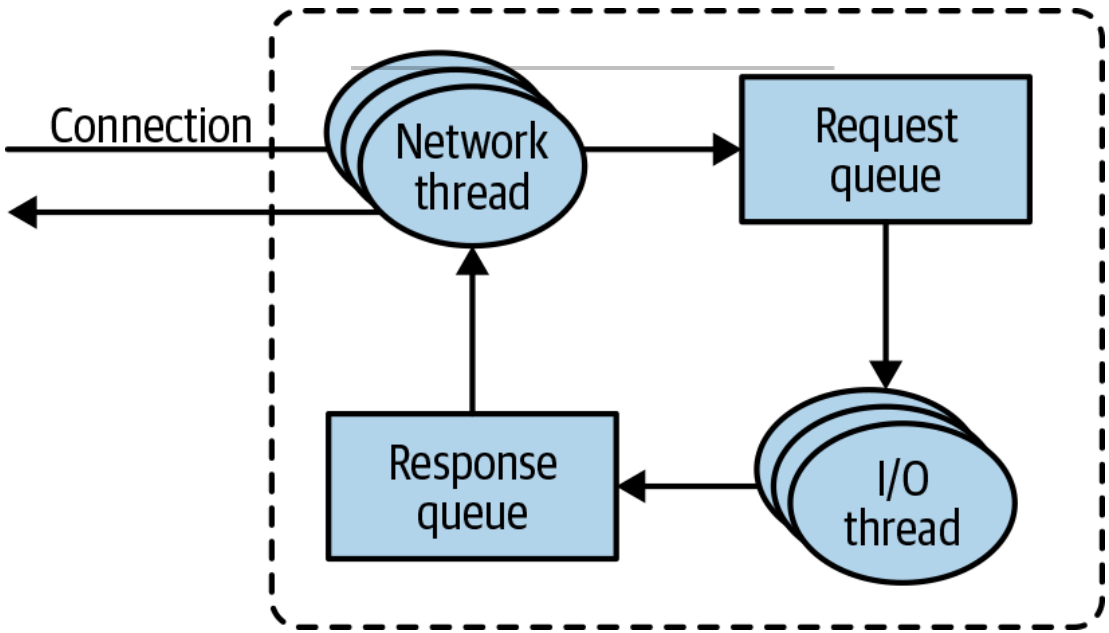
Kafka Request and Request Processing

Request headers

- All requests have a standard header that includes:
  - Request type (also called API key)
  - Request version (so the brokers can handle clients of different versions and respond accordingly)
  - Correlation ID: a number that uniquely identifies the request and also appears in the response and in the error logs
  - ClientID: used to identify the application that sent the request

Kafka Internal

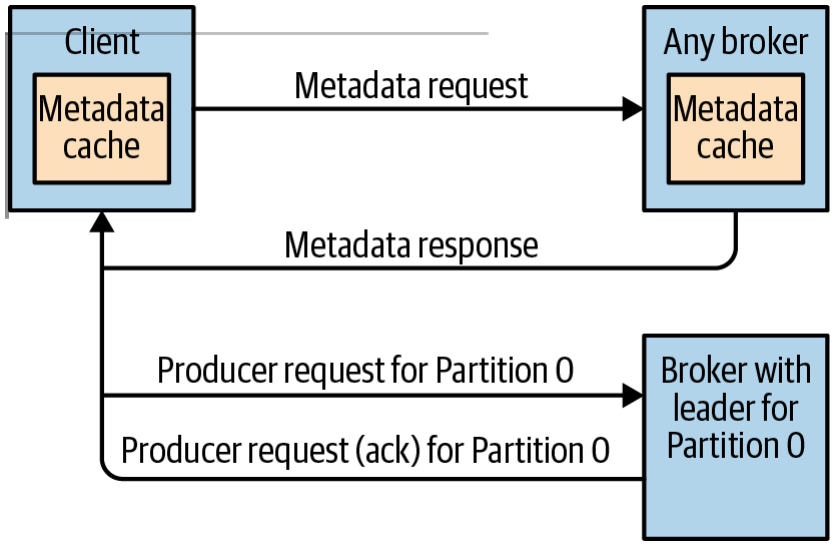
Request Processing inside Kafka



1. The broker runs a acceptor thread that creates a connection and hands it over to a processor thread for handling
2. The number of processor threads (also called network threads) is configurable. The network threads are responsible for taking requests from client connections, placing them in a request queue
3. Once request are placed on the request queue, I/O threads (also called request handler threads) are responsible for picking them up and processing them
4. After messages are processed by I/O thread, responses send to *response queue* and send back to the clients

Kafka Internal

Metadata request processing inside Apache Kafka



- Kafka client uses a *metadata request* to know where to send requests to get data.
- *Metadata request* includes a list of topics the client is interested in
- Broker responses specifies which partition exist in the topics, the replicas for each partition and which replica is the leader
- *Metadata requests* can be sent to any broker because all brokers have a metadata cache that contains this information

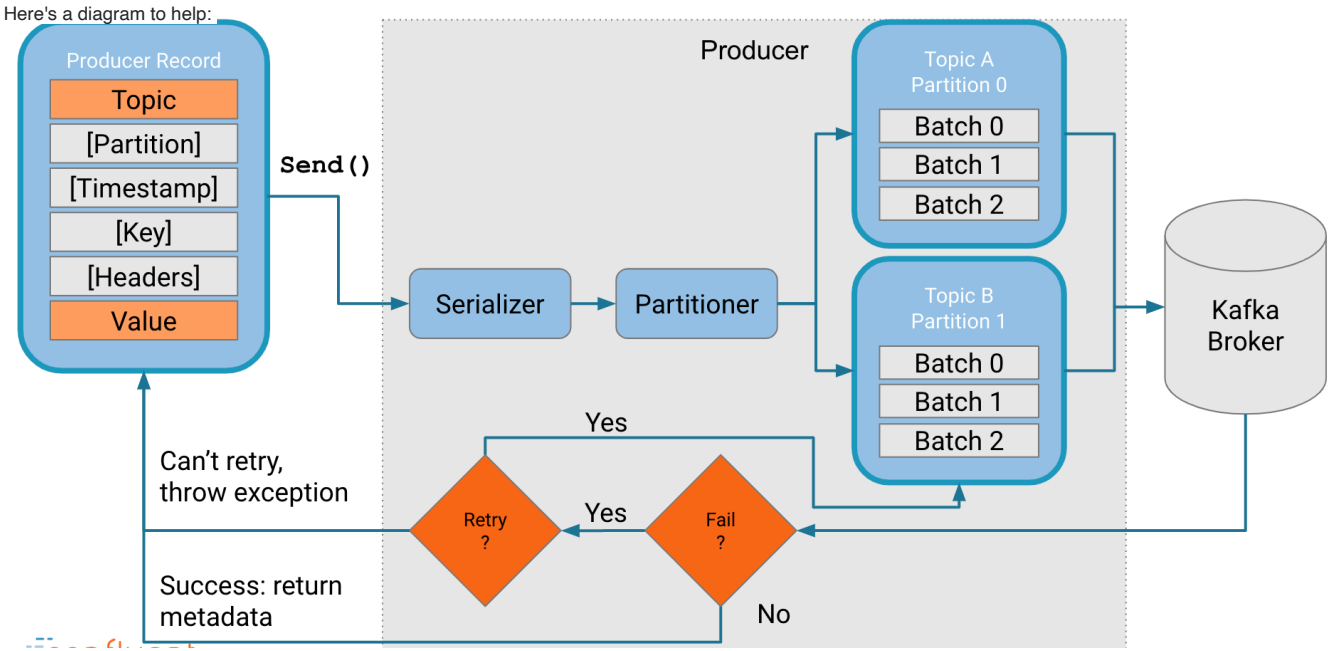
Kafka Internal

Common types of client requests

- Produce requests: send by producers and contain messages the clients write to Kafka brokers
- Fetch requests: send by consumers and follower replicas when they read messages from Kafka brokers
- Admin requests: Sent by admin clients when performing metadata operations such as creating and deleting topics

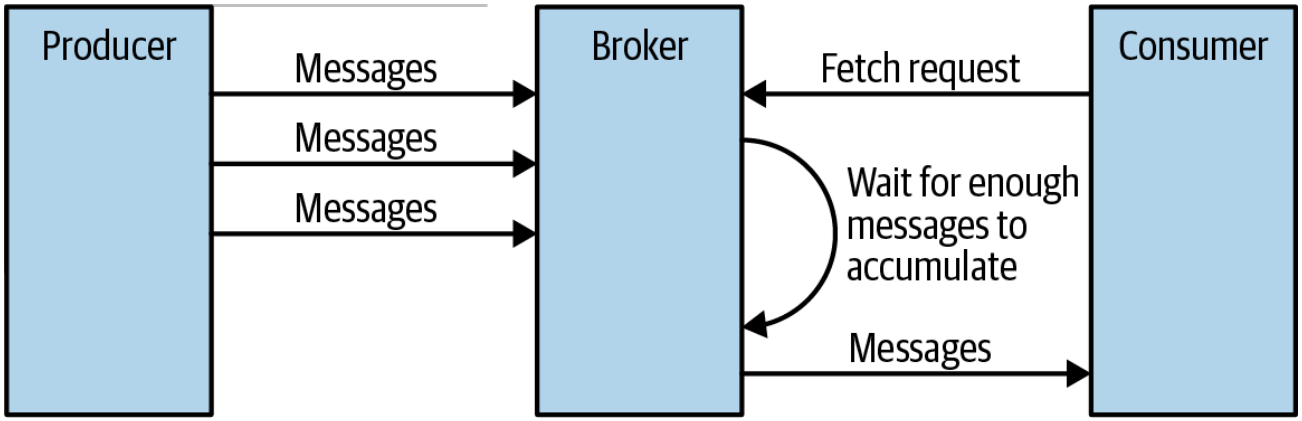
Kafka Internal

Producer request



Kafka Internal

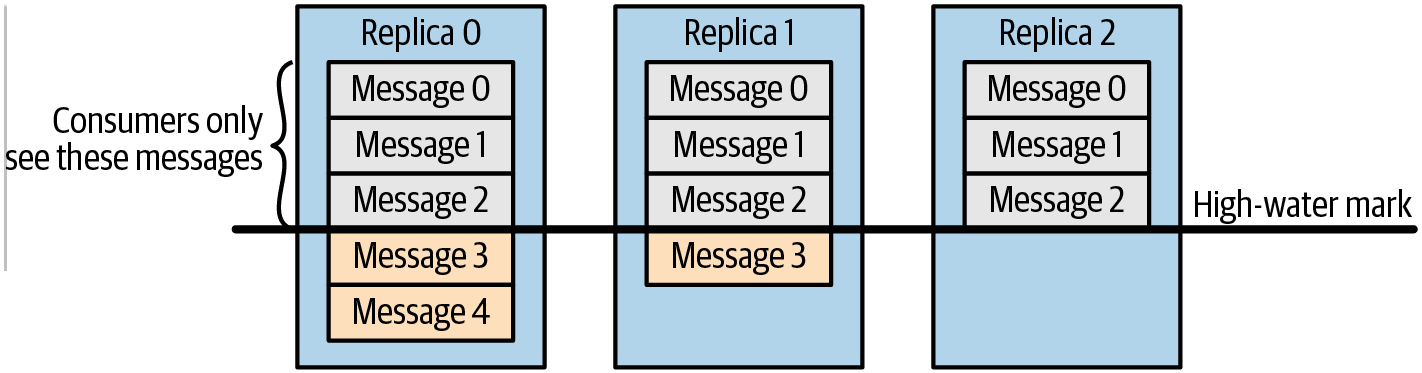
Fetch request



Broker delaying response until enough data accumulated

Kafka Internal

Fetch request



Consumers only see messages that were replicated to in-sync replicas

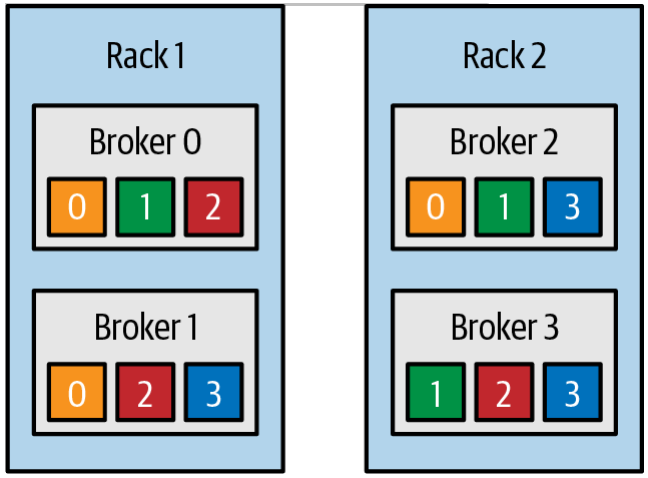
Kafka Internal

Other Requests

- Kafka protocol current handles more than 61 different request types.
- Consumer alone use 15 request types to form group, coordinate consumption, and allow developers to manage the consumer groups.

Kafka Internal

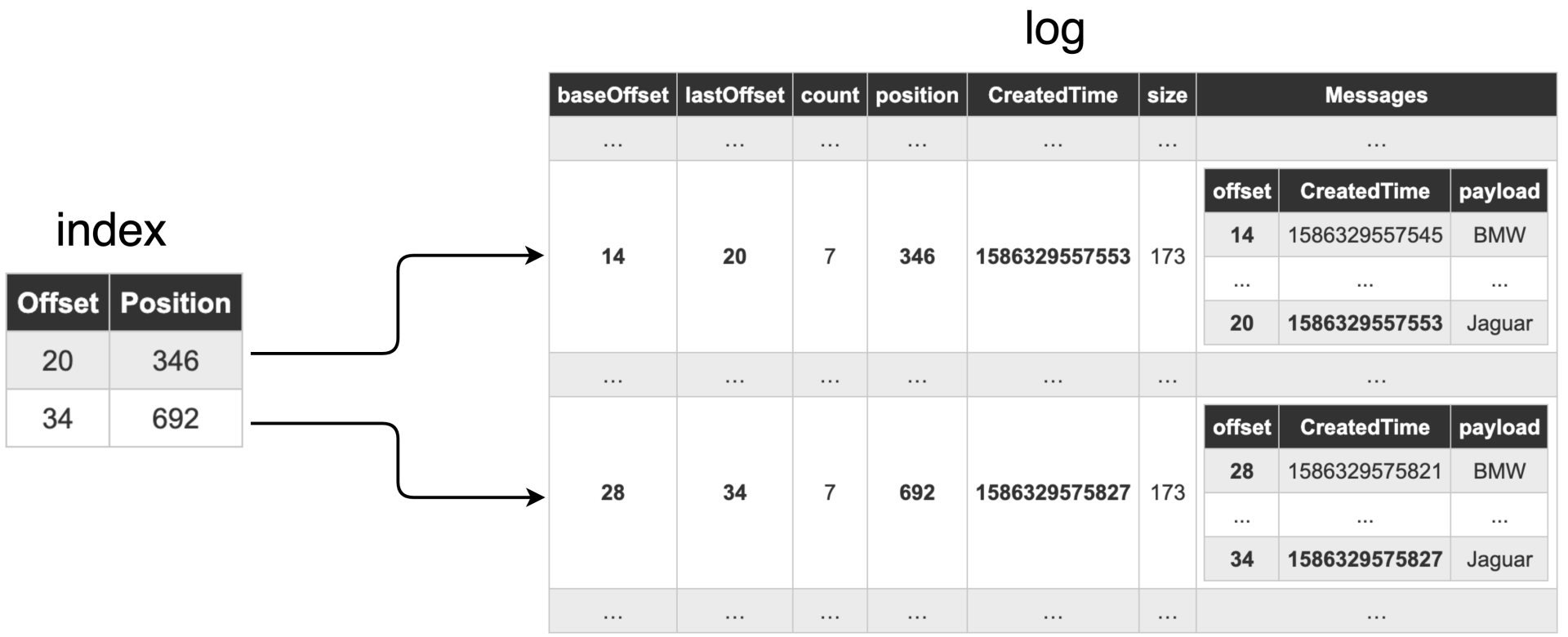
Partition allocation



- Kafka decides how to allocate the partitions between brokers when creating a topic.
- Partition allocation works to get the goals:
  - To spread replicas evenly among brokers
  - To make sure that for each partition, each replica is on a different broker
  - If the brokers have rack information, then assign the replicas for each partition to different racks if possible.

Kafka Internal

Indexes



Kafka Internal

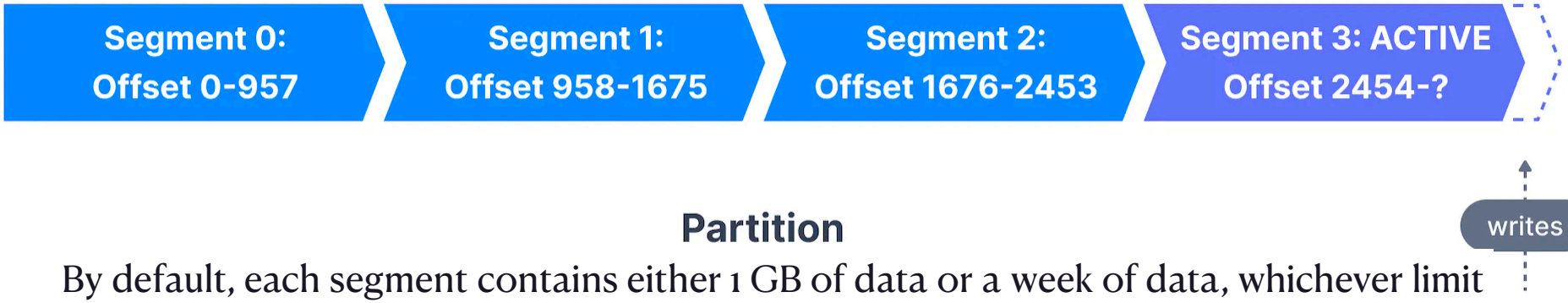
File Management

- Kafka does not keep data forever, nor does it wait for all consumers to read a message before deleting it.

- The Kafka administrator configures a retention period for each topic—either the amount of time to store messages before deleting them or how much data to store before older mes- sages are purged.

Kafka Internal

Segment



Partition

By default, each segment contains either 1 GB of data or a week of data, whichever limit is attained first. When the Kafka broker receives data for a partition, as the segment limit is reached, it will close the file and start a new one.

The segment we are currently writing to is called an active segment.

Kafka Internal

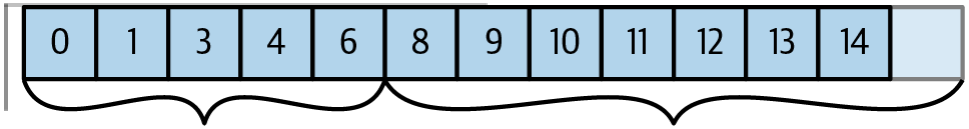
Compaction

Compaction is used when there is requirement to save messages older than the retention period by compacting the message

Use Cases of	Store shipping addresses for your customers applic
How does it	By only storing the most recent value for each key in the topic
Impact	Compaction can impact the read/ write performance on a topic

Kafka Internal

How Compaction Works



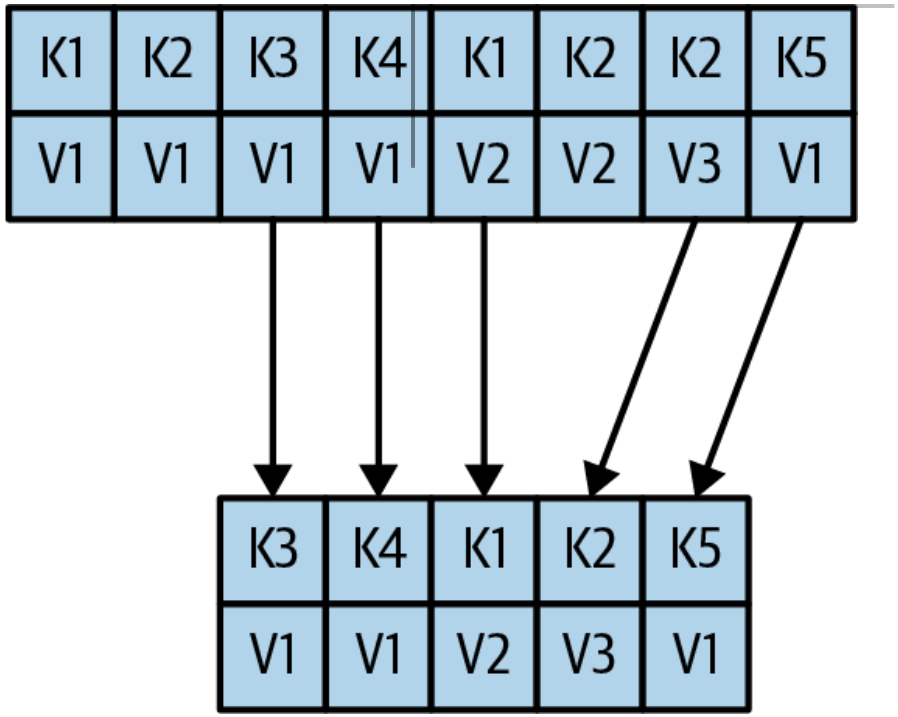
This section is now clean. Note how some offsets are missing. Those are messages that were removed by compaction.

This section is the dirty head of the log. It will get compacted later.

- Each log is viewed as split into two portions:
  - Clean: Message that have been compacted before. This section contains only one value for each key.
  - Dirty: Partition with clean and dirty portions

Kafka Internal

How Compaction Works



DISCUSSION

Quality Network for Education and Technology

XIN CHÂN THÀNH CẢM ƠN!