Summary: Kafka

Here is a quick summary of Kafka for you!

We'll cover the following

Summary#

1. Kafka provides low-latency, high-throughput, fault-tolerant **publish and subscribe pipelines** and can process huge continuous streams of events.
2. Kafka can function both as a **message queue** and a **publisher-subscriber** system.
3. At a high level, Kafka works as a **distributed commit log**.
4. Kafka server is also called a **broker**. A Kafka cluster can have one or more brokers.
5. A Kafka **topic** is a logical aggregation of messages.
6. Kafka solves the scaling problem of a messaging system by splitting a topic into multiple **partitions**.

7. Every topic partition is replicated for fault tolerance and redundancy.
8. A partition has one leader replica and zero or more follower replicas.
9. Partition leader is responsible for all reads and writes. Each follower's responsibility is to replicate the leader's data to serve as a 'backup' partition.
10. Message ordering is preserved only on a per-partition basis (not across partitions of a topic).
11. Every partition replica needs to fit on a broker, and a partition cannot be divided over multiple brokers.
12. Every broker can have one or more leaders, covering different partitions and topics.
13. Kafka supports a single queue model with multiple readers by enabling consumer groups.
14. Kafka supports a publish-subscribe model by allowing consumers to subscribe to topics for which they want to receive messages.
15. ZooKeeper functions as a centralized configuration management service.

System design patterns#

Here is a summary of system design patterns used in Kafka.

High-water mark#

To deal with non-repeatable reads and ensure data consistency, brokers keep track of the high-water mark, which is the largest offset that all ISRs of a particular partition share. Consumers can see messages only until the high watermark.

Leader and follower#

Each Kafka partition has a designated leader responsible for all reads and writes for that partition. Each follower's responsibility is to replicate the leader's data to serve as a 'backup' partition.

Split-brain#

To handle split-brain (where we have multiple active controller brokers), Kafka uses 'epoch number,' which is simply a monotonically increasing number to indicate a server's generation. This means if the old Controller had an epoch number of '1', the new one would have '2'. This epoch is included in every request that is sent from the Controller to other brokers. This way, brokers can easily differentiate the real Controller by simply trusting the Controller with the highest number. This epoch number is stored in ZooKeeper.

Segmented log#

Kafka uses log segmentation to implement storage for its partitions. As Kafka regularly needs to find messages on disk for purging, a single long file could be a performance bottleneck and error-prone. For easier management and better performance, the partition is split into segments.

References and further reading#

- Confluent Docs (https://docs.confluent.io/current/kafka/design.html#ak-design)
- New York Times use case (https://www.confluent.io/blog/publishing-apache-kafka-new-york-times/)
- Kafka Summit (https://www.confluent.io/resources/kafka-summit-san-francisco-2019/)
- Kafka Acks Explained (https://medium.com/better-programming/kafka-acks-explained-c0515b3b707e)
- Kafka as distributed log (https://www.youtube.com/watch?

v=ElilYxUOjOQ)

- Minimizing Kafka Latency (https://www.confluent.io/blog/configure-kafka-to-minimize-latency/)
- Kafka internal storage (https://thehoard.blog/how-kafkas-storage-internals-work-3a29b02e026)
- Exactly-Once semantics (https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/)
- Split-brain (https://techthoughts.typepad.com/managing_computers/2007/10/split-brain-quo.html)

← Back
Kafka Characteristics

Next →
Quiz: Kafka

✓ Mark as Completed

Report an Issue