International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 08– Issue 12 || December 2023 || PP. 40-44

Architecture Kafka with and without ZooKeeper

Filisov Denis Alexandrovich
Senior Software Engineer,
https://career.luxoft.com/
Belgrade, Serbia

Abstract: Kafka has an extensive structure, within the Kafka architecture, Zookeeper serves as a centralized controller managing all metadata information related to Kafka producers, brokers and consumers. It is possible to install and run Kafka without Zookeeper, where all Kafka configuration data will be stored in a special section inside Kafka itself, and not in Zookeeper.

This article will cover the architecture of this system, API, Kafka brokers, consumers, Zookeeper and manufacturers. Some basic Kafka concepts will also be covered. The purpose of the work is to consider Architecture Kafka with and without ZooKeeper. The methodological foundations were scientific articles, books, and various expert opinions.

Keywords: Kafka, ZooKeeper, programming, IT, modern technologies, Kafka architecture.

Introduction

The producer, consumer, thread, and connector APIs in Apache Kafka provide four key services: permanent storage of large amounts of data, a message bus with a bandwidth of millions of messages per second, excessive storage of huge amounts of data and parallel processing of huge data streams. The essence of the Apache Kafka architecture is simple, which is due to a special purpose: eliminating the typical complexities associated with messaging architectures. The purpose of this architecture is to provide a more understandable method of messaging between applications than most other alternatives. Kafka is renowned for its fault-tolerant and scalable log structure with a simple data design.

Kafka provides a permanent ordered data structure in which an entry cannot be deleted or modified - it can only be added to the log. The Kafka cluster keeps track of the order of the items in the logs and guarantees equal priority individual commits. Messages are stored in the order in which they are received, ensuring the integrity and order of records. Each record has a unique sequential identifier, known as an offset, used to extract data, which provides a unique start and end point of the log.

By providing set ordering and deterministic processing, Kafka solves typical problems of distributed systems. In addition to sequential reads from disk, Kafka benefits from the orderly storage of message data on disk, since data is stored on disk in an orderly manner. Disk search is expensive in terms of resource loss, and the Kafka process, consisting of the first read and write at a constant pace, followed by simultaneous reads and writes, reduces resource loss [1,2].

1. General characteristics of Kafka

Apache Kafka is an open source software platform for streaming data processing. Developed by LinkedIn and donated to the Apache Software Foundation, it is written in Scala and Java. The main goal of the project is to provide high throughput and low latency for real-time data processing. Its key architectural concept is an immutable message log organized into sections for use by multiple users or applications. This log, stored in a file system or database, keeps permanent records of messages, which allows Kafka to replay them to maintain a consistent system state.

Kafka securely stores data in serialized form and distributes it across a cluster of nodes, providing scalable performance and fault tolerance.

The main features of Kafka:

- **High scalability:** Using the partitioned log model, Kafka distributes data across multiple servers, allowing them to surpass the capabilities of a single server.
- **Low latency:** The separation of data streams in Kafka provides low latency and high throughput.
- **Fault tolerance and durability:** Data is written to disk, partitions are distributed and replicated on various servers, which provides protection against failures and makes data stable and reliable even in case of server failures. The Kafka cluster can eliminate failures in the main server and database, capable of restoring the server's functionality automatically.

- **Extensibility:** Kafka connectors have been integrated into many other applications in recent years, making it easy to add additional features, such as integration with various systems, within seconds. For example, you can learn how to integrate Kafka with Amazon Redshift and Salesforce.
- **Metrics and Monitoring:** Kafka has become a popular tool for tracking operational data. It allows you to collect data from different applications and combine them into centralized channels with metrics. For a deeper understanding of Kafka data analysis, real-time reports are available using Kafka Analytics.

The four key APIs in the Apache Kafka architecture include the producer API, the consumer API, the Streams API, and the Connector API. Let's look at each of them in more detail:
Producer API: This API allows the application to publish a continuous stream of entries to a specific Kafka topic.

Consumer API: An application using the Consumer API can subscribe to one or more topics and process the stream of entries coming into these topics.

Streams API: The Streams API allows applications to transform incoming data streams into output streams. This is achieved by processing the input stream from one or more topics and creating an appropriate output stream. This API also allows you to perform data streaming by processing the input stream and creating an output stream for one or more topics.

Connector API: The Connector API is used to integrate Kafka partitions with existing applications or data systems. For example, a connector associated with a relational database is able to track every change in a table and transfer it to Kafka for further processing [3].

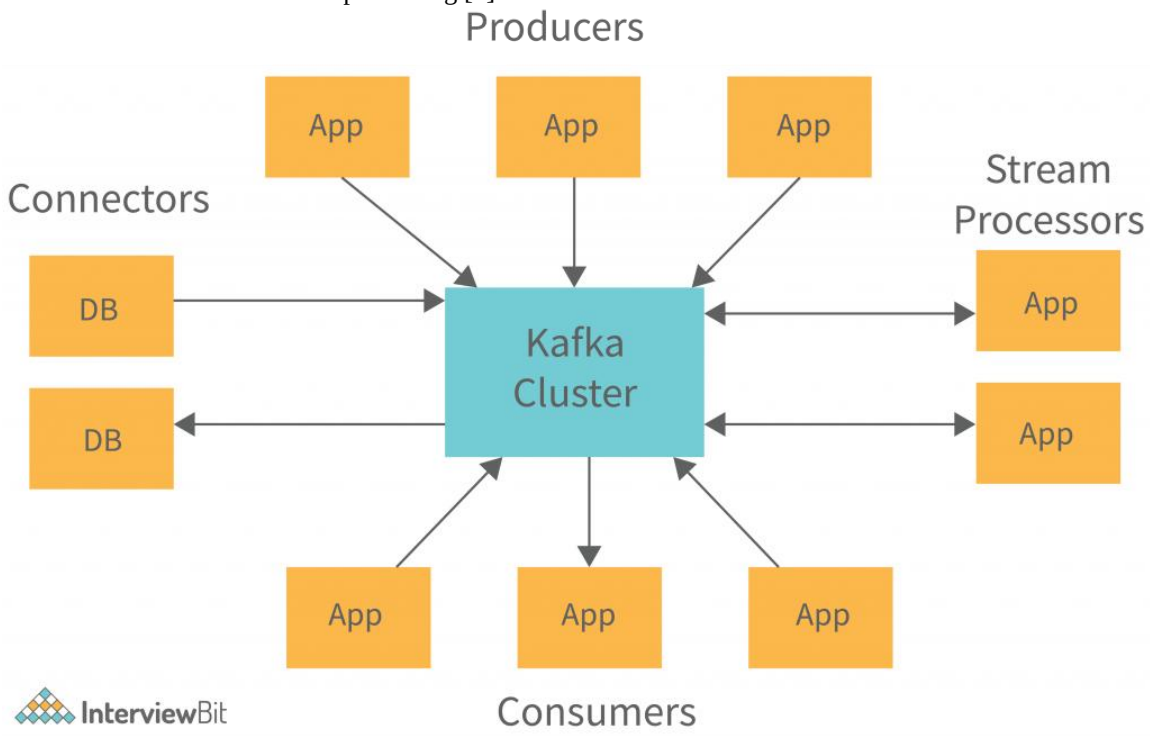

Fig.1. Kafka structure

2. General characteristics of ZooKeeper

Zookeeper is an open management system designed to coordinate various distributed applications. The Apache Kafka distributed data transfer platform relies on Zookeeper to store and control settings related to Kafka partitions, servers, data producers and consumers. In other words, Zookeeper, used by Apache Kafka, acts as a centralized reconciliation system that stores and manages information about Kafka clusters and their shared metadata, including data about Kafka brokers or servers[4,5].

The main features of Zookeeper include the following:

- **Naming system:** Each node in ZooKeeper has a unique identifier, similar to an individual genetic code, which makes it easier to recognize them.
- **Tracking the status of the node:** Zookeeper's ability to update the status of each node allows the IT department to keep up-to-date information about the status of each node in the cluster.
- **Cluster Management:** Zookeeper monitors the status of each node in real time, which reduces the possibility of errors and simplifies cluster management.
- **Automatic crash recovery:** When a database failure occurs, ZooKeeper locks the data, providing the ability to automatically restore the cluster.
- **Scalability:** Zookeeper's performance can be increased by using multiple computers. • Efficiency: Zookeeper provides high-speed operation under workloads where reading prevails.
- **Message Ordering:** Zookeeper tracks messages by assigning a sequence number to each update, which indicates their sequence.
- **Reliability:** Updates applied by Zookeeper remain constant until the client overwrites them.
- **Consistent consistency:** Changes made by the client are applied in the same sequence in which they were received.
- **Single system image:** The client gets the same view of the service, regardless of which Zookeeper server is connected to

ZooKeeper in the Kafka system performs the functions of coordinating, managing and reporting on the status of brokers. This system also notifies producers and consumers about the appearance of new brokers or the failure of existing ones. For Kafka users, the key advantage is the ability to track used messages through section offsets. Such systems can control the number of messages processed by monitoring the offset of sections, and making sure that all previous messages are used by confirming the offset of each of them.

A byte buffer is required so that the consumer can initiate a data retrieval request. For example, if the offset is 5, consumers can rewind or jump to a specific point in the partition by specifying 5 as the offset value. ZooKeeper informs consumers about the value of the offset for their work [6,7].

3. Comparison of Kafka architecture with Zookeeper and without Zookeeper

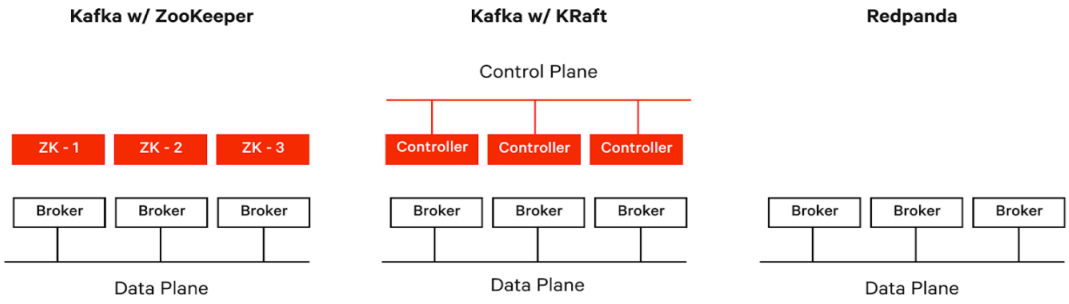

Figure 2. Comparison of Kafka architecture with Zookeeper (left), without Zookeeper (center) and Red panda (right).

The ZooKeeper installer has a separate ZooKeeper cluster, which is used by the Kafka cluster to discover Kafka cluster members, store a significant portion of configurations, and coordinate brokers and data about them. Everyone starts the controller process, and the first one who registers with ZooKeeper becomes the active controller, which gives the right to manage the cluster and transmit changes to all brokers. Other controllers continue to regularly check ZooKeeper in case the active controller suddenly disappears, and in this case the first one that detects the absence of an active controller will try to register and promote to the active cluster controller [8,9].

If you decide to launch Kafka using the new quorum controller, all metadata processing responsibilities previously performed by the Kafka controller and ZooKeeper will be combined into this new service running inside the Kafka cluster itself. The quorum controller can also run on dedicated hardware if you have a use case that requires it.

However, the internal mechanism becomes slightly different. Quorum controllers are implementing the new KRaft protocol to ensure accurate replication of metadata across the entire quorum. Although this protocol

resembles ZooKeeper and Raft's ZAB protocol in many ways, it still differs in several important aspects, including the use of an event-based architecture.

The quorum controller maintains its state by applying a storage model with an event source that ensures accurate recreation of internal state machines at any moment. The event log used to store this state (also known as the metadata section) is periodically shortened by snapshots, preventing its size from increasing indefinitely. The other controllers in the quorum monitor the active controller, tracking the events that it initiates and stores in its log. This means that if one of the nodes is temporarily suspended, for example, due to network separation, it quickly catches up based on the log when reconnecting. This approach significantly reduces the period of unavailability, increasing the system recovery time in extreme cases.

The event-driven nature of the KRaft protocol means that, unlike a ZooKeeper-based controller, the quorum controller does not need to load state from ZooKeeper before the base becomes active. When you change the manual, the new active controller already has all the recorded metadata entries in memory. Moreover, the same event-driven mechanism used in the KRaft protocol is used to track metadata throughout the cluster. A task that was previously performed using RPC now benefits from the fact that it is event-driven and also uses the actual log for communication. A pleasant consequence of these changes — and those that were largely embedded in the original design — is that Kafka can now support many more partitions than before.

Table 1 Comparison with and without a ZooKeeper-based controller

	With a ZooKeeper-based controller	With a quorum controller
Controlled shutdown time (2 million partitions)	135 seconds	32 seconds
Recovery after an uncontrolled shutdown (2 million partitions)	503 seconds	37 seconds

Both measures for controlled and uncontrolled shutdown are important. Controlled shutdowns affect common operational scenarios such as continuous restart: a standard procedure for deploying software changes while maintaining constant availability.

Recovery from uncontrolled outages is perhaps more important because it sets a target recovery time (RTO) for the system, say, after an unexpected failure such as a VM or pod failure or data center unavailability. Although these indicators are only indicators of the overall performance of the system, they directly measure the well-known bottleneck that occurs when using ZooKeeper [10].

Conclusion

The Kafka architecture with and without ZooKeeper represents two different approaches to organizing and managing data in a distributed system. Using ZooKeeper, Kafka uses it to store and manage metadata, coordinate tasks, select a controller, and support data separation and access control functionality. This ensures reliability and consistency of operations in the cluster, but at the same time requires the installation and support of an additional component (ZooKeeper).

At the same time, starting with version 2.8.0, Kafka offers the ability to work without ZooKeeper. This update provides new perspectives, reducing dependence on additional service and simplifying the architecture. However, this approach is still in development and is not ready for full use in a production environment.

Both approaches have their advantages and disadvantages. The architecture with ZooKeeper provides reliability and extensive management capabilities, while the version without ZooKeeper can offer a simplified architecture, but requires further improvements for stable operation in a production environment.

References

[1]. Real Time API in the context of Apache Kafka. [Electronic resource] Access mode: https://habr.com/ru/companies/otus/articles/532954 /.– (accessed 09.12.2023).
[2]. Kafka APIs. [Electronic resource] Access mode: https://docs.confluent.io/kafka/kafka-apis.html .– (accessed 09.12.2023).
[3]. Apache Kafka Cluster Architecture: the basics of Big Data for beginners. [Electronic resource] Access mode: https://kafka-school.ru/blogs/kafka-cluster-structure /.– (accessed 09.12.2023).
[4]. What is the benefit of ZooKeeper for admins and developers. A seminar at Yandex . [Electronic resource] Access mode: https://habr.com/ru/companies/yandex/articles/234335 /.– (accessed 09.12.2023).
[5]. Apache ZooKeeper™. [Electronic resource] Access mode: https://zookeeper.apache.org /.– (accessed 09.12.2023).
[6]. Kafka Architecture – Detailed Explanation. [Electronic resource] Access mode:

https://www.interviewbit.com/blog/kafka-architecture /.– (accessed 09.12.2023).

[7]. Architecture Kafka. [Electronic resource] Access mode: https://hevodata.com/learn/kafka-without-zookeeper/#:~:text=In%20Kafka%20architecture%2C%20Zookeeper%20serves,separate%20partition%20within%20Kafka%20itself. – (accessed 09.12.2023).

[8]. Kafka without ZooKeeper. [Electronic resource] Access mode: https://redpanda.com/guides/kafka-tutorial/kafka-without-zookeeper .– (accessed 09.12.2023).

[9]. Kafka without Zookeeper. [Electronic resource] Access mode: https://habr.com/ru/companies/otus/articles/670440 /.– (accessed 09.12.2023).

[10]. Apache Kafka Made Simple: A First Glimpse of a Kafka Without ZooKeeper. [Electronic resource] Access mode: https://www.confluent.io/blog/kafka-without-zookeeper-a-sneak-peek /.– (accessed 09.12.2023).