What is Kafka? Architecture Deep Dive

A deep dive into Apache Kafka architecture and inner working

JAVINPAUL AND SOMA

MAY 03, 2024 • PAID

Hello guys, in last few years, **Apache Kafka** has emerged as a robust and scalable messaging platform in event driven architecture and Microservices.

With its ability to handle high volumes of data in real-time, Kafka has become a popular choice for building data pipelines, stream processing applications, and event-driven architectures.

Kafka is also quite important from interview point of view and that's why I shared **difference between Apache Kafka, and RabbitMQ** in my last article and in this article, we will deep dive into the Kafka architecture and inner workings of Apache Kafka, exploring how it stores data, manages partitions, transactions, and maintains data integrity.

What is Apache Kafka?

If you have worked in distributed system then you must have heard about Kafka, an open-source distributed event streaming platform developed by the Apache Software Foundation.

It is designed to **handle high-throughput, real-time data feeds**. Kafka is often used for building real-time data pipelines and streaming applications. **LinkedIn was one of the first few company** which used Kafka to implement their high volume messaging platform.

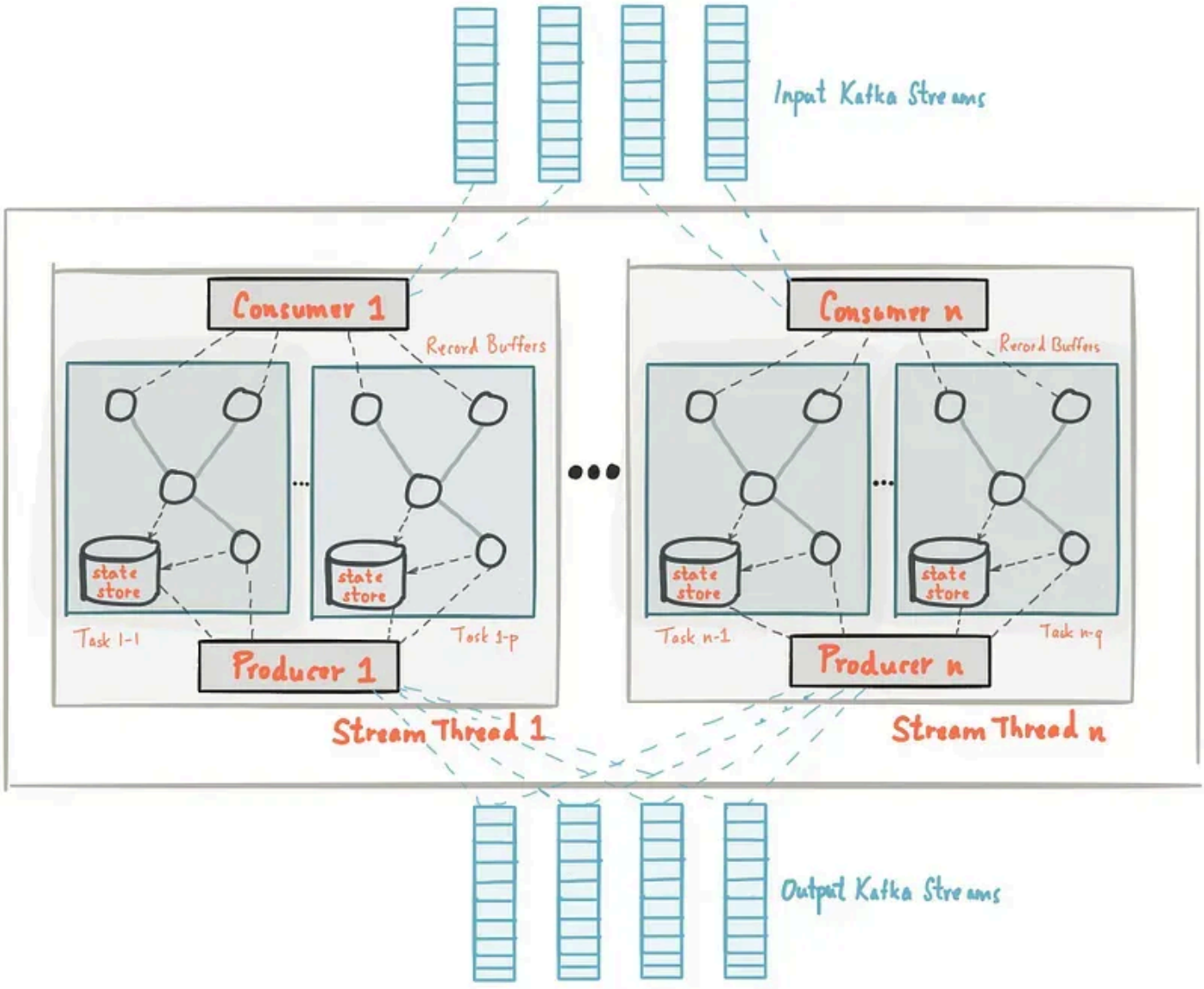
Here are some key concepts associated with Apache Kafka:

1. **Topics**: Data is organized and stored in Kafka topics, which are similar to a feed or a category for messages.
2. **Producers**: Producers are responsible for publishing data to Kafka topics.
3. **Consumers**: *Consumers subscribe to Kafka topics* and process the data published to those topics.
4. **Brokers**: Kafka runs as a cluster of servers, called **brokers**, which store the data and handle the distribution of data across topics.

5. **Partitions**: Topics are divided into partitions, which allow data to be distributed across multiple brokers in a Kafka cluster.
6. **Replication**: Kafka **provides fault tolerance through data replication**. Each partition can have one or more replicas distributed across different brokers.
7. **Connectors**: Kafka Connect is a framework for building and running reusable data import/export connectors to move data between Kafka and other systems.
8. **Streams**: Kafka Streams is a library for building stream processing applications that can process and analyze data in real-time.

Apache Kafka has gained popularity for its scalability, fault tolerance, and high throughput capabilities, making it a popular choice for building real-time data pipelines and streaming applications in various industries, including finance, retail, telecommunications, and more.

Here is a good example of Apache Kafka Cluster:

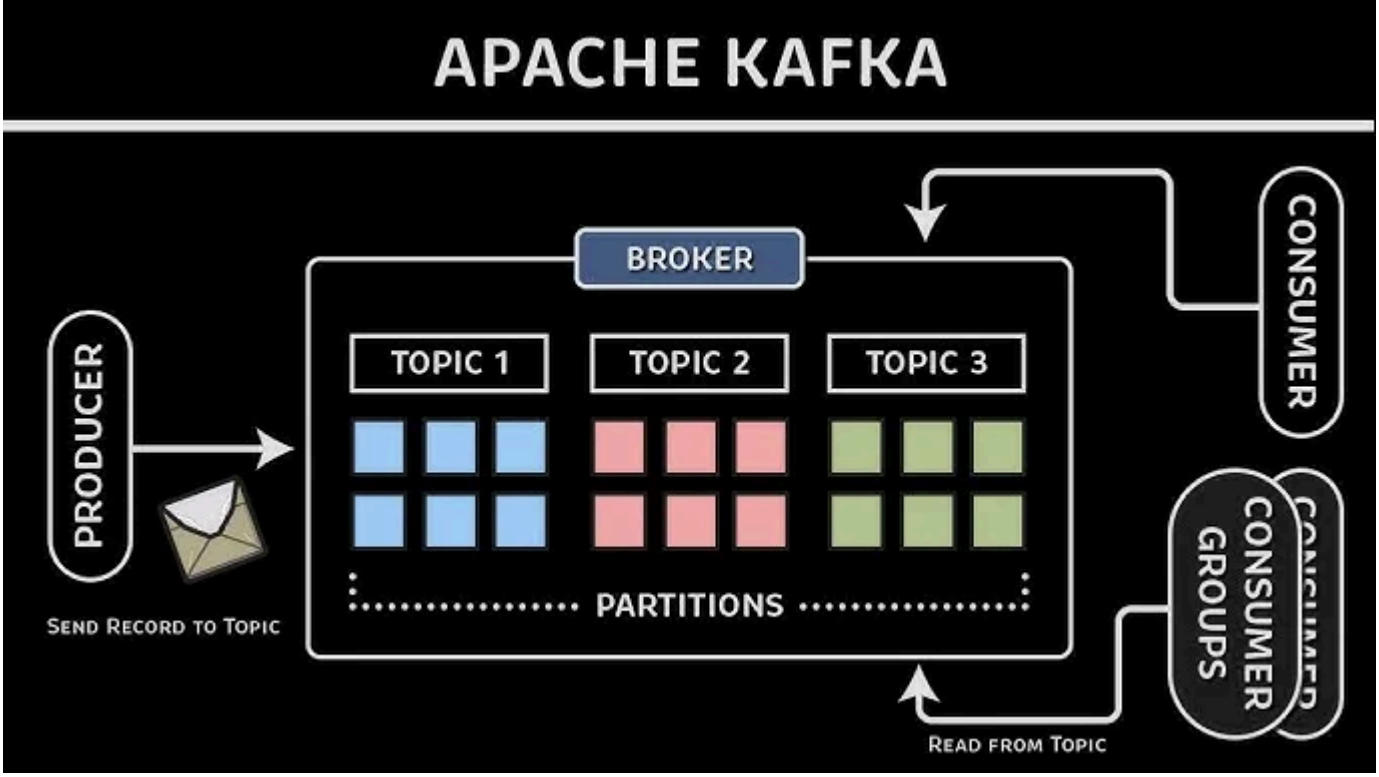


What are key components of Apache Kafka Architecture?

Apache Kafka has a **distributed architecture** designed to handle high-throughput, fault tolerance, and scalability. Here's an overview of the main components and how they interact:

1. **Brokers**: Kafka clusters consist of one or more servers called brokers. Each broker is responsible for handling data storage, replication, and serving consumer requests. Brokers are where Kafka topics are partitioned and replicated.
2. **Topics**: Topics are logical channels or feeds where data is published by producers and consumed by consumers. Topics are divided into partitions, and each partition is replicated across multiple brokers for fault tolerance and scalability.
3. **Partitions**: Each topic is divided into one or more partitions. Partitions are the basic unit of parallelism in Kafka. They allow data within a topic to be distributed and processed concurrently across multiple brokers. Each message within a partition is assigned a unique offset.
4. **Producers**: Producers are responsible for publishing data to Kafka topics. They write messages to specific topics, and Kafka brokers then append these messages to the appropriate partition within the topic. Producers can choose to send messages to specific partitions or let Kafka decide based on a partitioning strategy.
5. **Consumers**: Consumers subscribe to Kafka topics and read messages from partitions. They can consume messages in real-time as they are published or rewind to consume past messages. Consumers can be part of a consumer group, which allows multiple consumers to work together to process messages from the same topic in parallel.
6. **Consumer Groups**: Consumer groups are a way to scale consumers horizontally. Each consumer within a group reads messages from one or more partitions within a topic. Kafka ensures that each message is delivered to only one consumer within the same group, providing load balancing and fault tolerance.
7. **ZooKeeper**: Kafka uses Apache *ZooKeeper for managing and coordinating cluster* nodes, leader election, and storing metadata. ZooKeeper keeps track of broker and partition information, consumer group offsets, and other cluster-related metadata.
8. **Connectors**: Kafka Connect is a framework for building and running reusable data import/export connectors to move data between Kafka and other systems. Connectors are used for integrating Kafka with external data sources and sinks.
9. **Streams Processing**: Kafka Streams is a library for building real-time stream processing applications directly within Kafka. It allows developers to process and analyze data streams in real-time, enabling tasks like data transformation, aggregation, and event-driven microservices.

This architecture provides flexibility, scalability, and fault tolerance, making Kafka suitable for building real-time data pipelines, event-driven applications, and stream processing systems.



How Apache Kafka works?

Now, let's **deep dive into Apache Kafka architecture** and try to understand how does it work? How it can handle trillions of messages and still ensures data integrity and speed. We will deep dive into *how Apache Kafka stores its data*, how it manages transaction, how does it work with partitions and most importantly how it ensures data integrity.

1. Data Storage in Apache Kafka

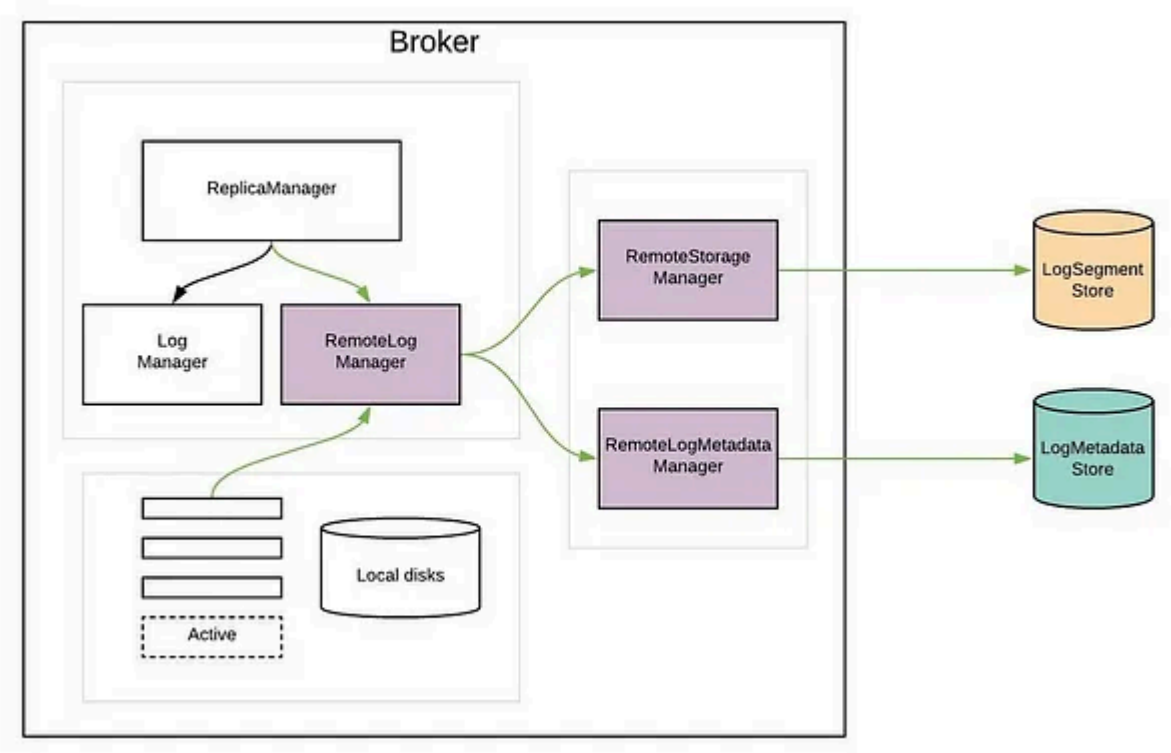
At its core, Apache Kafka is designed to persist and distribute streams of records, known as topics, across a cluster of servers. Kafka stores these records in a distributed, fault-tolerant, and append-only manner.

Instead of relying on a traditional file system, Kafka utilizes its own storage abstraction called "**log.**"

A log is an ordered sequence of records where each record represents a key-value pair, along with additional metadata.

The log is divided into partitions, which are the fundamental unit of parallelism in Kafka. Each partition is an ordered, immutable sequence of records.

Kafka ensures that the order of records within a partition is preserved, which allows for sequential processing and ensures data consistency.



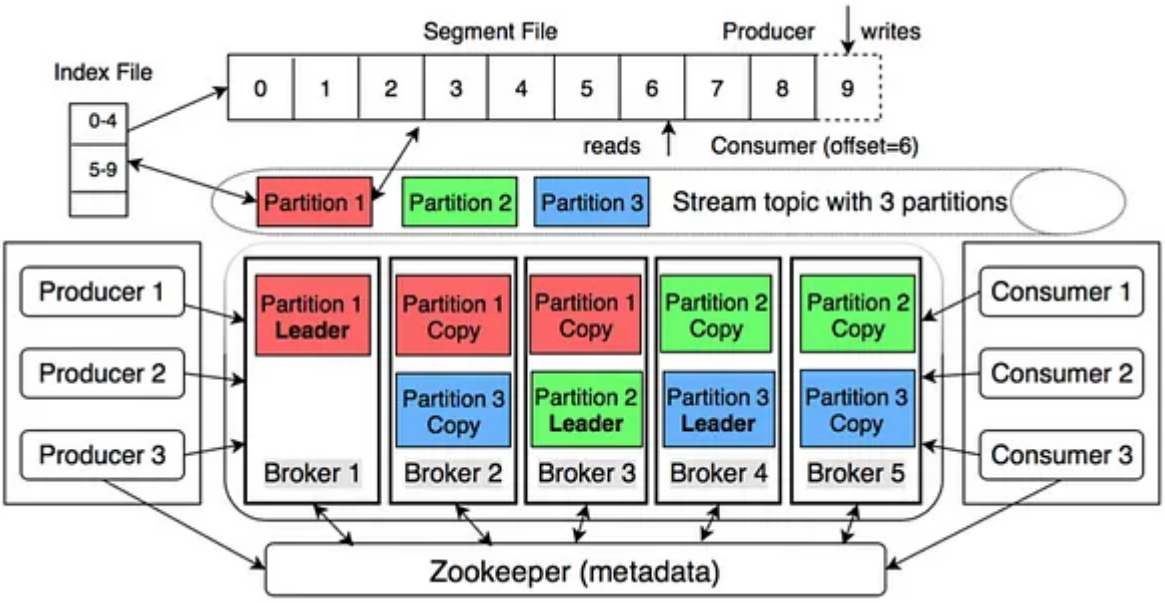
2. Partition Management

Partitions play a crucial role in Kafka's scalability and fault-tolerance. They allow Kafka to distribute the load across multiple brokers and enable data parallelism.

When a topic is created, the user specifies the number of partitions, which determines the level of parallelism for consuming and producing data.

Each partition is hosted by a single broker within the Kafka cluster. The broker acts as the leader for that partition and handles all read and write requests for that partition.

Additionally, Kafka replicates each partition across multiple brokers to provide fault-tolerance. These replicas, known as followers, passively replicate data from the leader and can take over as the leader if the current leader fails.



image_credit—

https://www.researchgate.net

3. Transactions in Kafka

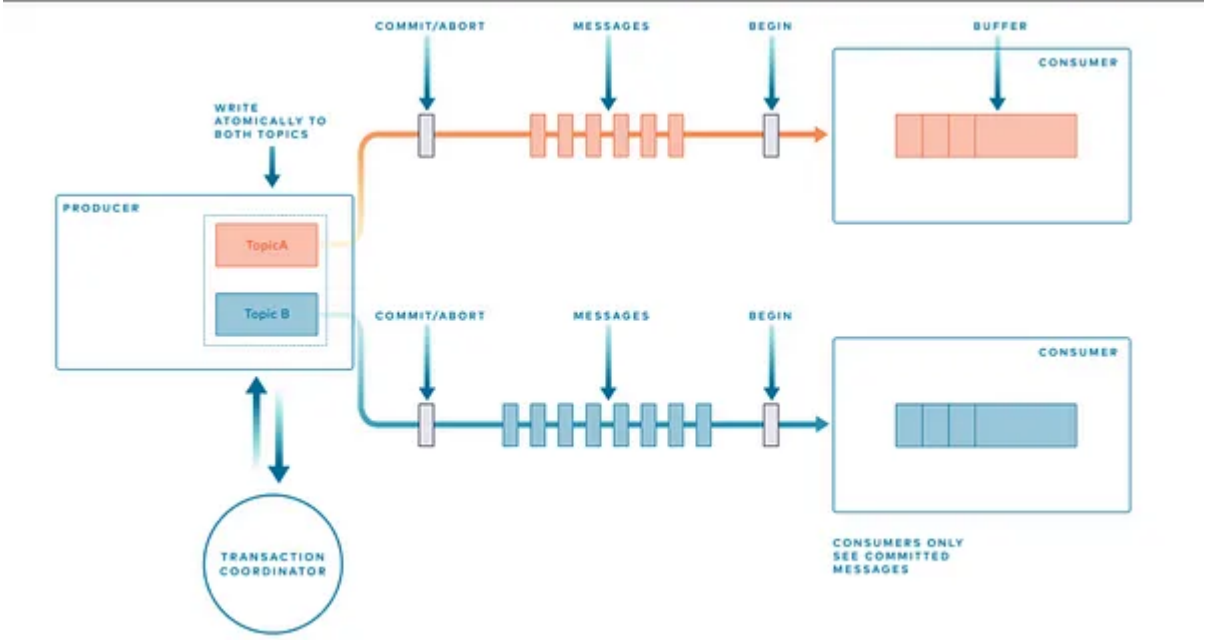
Apache Kafka introduced transactional support in version 0.11, providing atomicity and isolation guarantees for groups of records.

Transactions allow producers to write multiple records to multiple partitions atomically, ensuring that either all the records are written successfully, or none of them are.

This is particularly useful when maintaining data integrity across different systems or when processing events with dependencies.

To enable transactions, Kafka uses a combination of a transaction log and transaction markers within the log. Producers interact with the transaction coordinator to initiate and commit transactions.

The transaction coordinator assigns transactional IDs to producers and ensures that the transactional guarantees are maintained.



image_credit—

https://developer.confluent.io

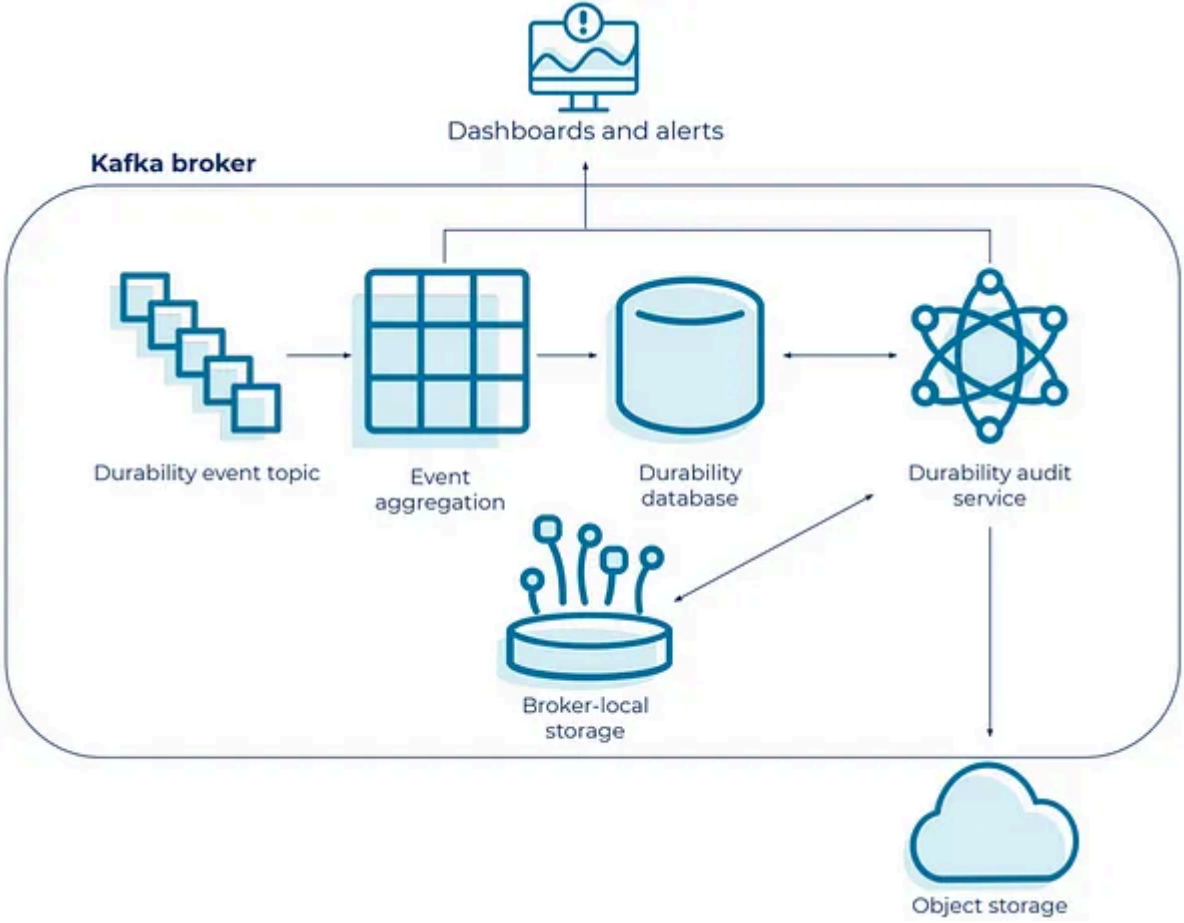
4. Data Integrity

Data integrity is a critical aspect of any messaging system, and Kafka employs various mechanisms to ensure the reliability of data. Firstly, Kafka replicates data across multiple brokers to provide fault-tolerance. This replication factor can be configured to tolerate a specific number of broker failures.

Additionally, Kafka allows producers to specify the durability level for each record. Producers can choose to write records with different durability requirements, ranging from writing to disk on the leader broker to replicating to multiple followers.

By configuring durability levels, Kafka enables developers to make trade-offs between performance and data safety based on their specific use cases.

In Confluent, more than **8 trillion message are audited daily** which means it ensures **data integrity issues on well over 8 trillion Kafka messages per day in Confluent Cloud**.



Conclusion

That's all about **What is Apache Kafka, Its architecture and key components**, and **how Apache Kafka works internally.** Apache Kafka's architecture and design make it a powerful and reliable messaging platform for handling real-time data streams.

By understanding how Kafka stores data, manages partitions, handles transactions, and maintains data integrity, developers can leverage its capabilities effectively.

Whether it's building scalable data pipelines or developing event-driven applications, Apache Kafka provides a robust foundation for managing and processing streams of data in distributed environments.

Kafka is also very important topic for interviews. If you are preparing for interviews then you can also prepare questions like *difference between API Gateway and Load Balancer* and horizontal vs vertical scalability, they are quite popular on interviews.

← Previous

Next →

A guest post by

Soma

Java and React Developer

Subscribe to Soma

Comments

Write a comment...