**Apache Kafka**

https://kafka.apache.org/documentation.html

<https://sookocheff.com/post/kafka/kafka-in-a-nutshell/>

https://medium.com/javarevisited/robust-kafka-consumer-error-handling-on-a-spring-boot-3-application-6fc95e92c956

**Key features of Apache Kafka:**

1. **Publish-Subscribe Model:** Kafka uses a publish-subscribe model where producers publish messages to topics, and consumers subscribe to those topics to receive the messages.
2. **Topics and Partitions:** Messages in Kafka are organized into topics, which can be divided into multiple partitions. Each partition is ordered and immutable, and it can be distributed across different brokers in the cluster.
3. **Scalability:** Kafka is highly scalable due to its distributed nature. It can handle large volumes of data and high throughput by distributing the load across multiple brokers and partitions.
4. **Fault Tolerance:** Kafka provides fault tolerance by replicating each partition across multiple brokers. This ensures that data remains available even if some brokers or nodes fail.
5. **Retention**: Kafka allows data to be retained for a specified period or size, enabling replay of data and data archiving.
6. **Connectivity**: Kafka has a rich ecosystem of connectors, making it easy to integrate with various data sources and sinks, including databases, Hadoop, Elasticsearch, etc.
7. **Streaming APIs**: Kafka offers native streaming APIs to process and transform data in real-time, making it suitable for building streaming applications.

**The lifecycle of Apache Kafka can be summarized in several stages:**

1. **Topic Creation:** In the first stage, administrators or applications create Kafka topics. Topics represent a specific category or feed where messages are published.
2. **Producers:** Once topics are created, producers start publishing messages to these topics. Producers can be various applications or systems that generate data and send it to Kafka.
3. **Brokers and Partitions:** Kafka brokers receive and store the messages sent by producers. Each topic is divided into multiple partitions, and brokers handle the distribution and storage of these partitions.
4. **Consumers:** Consumers subscribe to topics and start reading messages from partitions. They can be applications, microservices, or data processing systems that process the data in real-time.
5. **Retention and Cleanup:** Kafka retains messages in topics for a specified period or size. Older messages may be deleted or archived based on the configured retention policy.
6. **Scalability and Replication:** As the data volume grows or to ensure fault tolerance, the Kafka cluster can be scaled by adding more brokers. Partitions are replicated across brokers to provide redundancy and high availability.
7. **Monitoring and Management:** Throughout the lifecycle, Kafka clusters are monitored and managed to ensure performance, stability, and reliability. Various tools and monitoring frameworks are used to manage Kafka clusters effectively.

The lifecycle of Kafka continues as new data is published, consumed, and processed in real-time, providing a continuous stream of data flow within the distributed Kafka system.

**About Cluster Broker Topic Partition and Replication**

1. **Kafka Cluster:** A Kafka cluster is a group of multiple Kafka brokers working together. It forms the distributed backbone of the Kafka messaging system. The cluster is responsible for managing the replication, distribution, and storage of messages across brokers.
2. **Broker:** A broker is an individual node/server in the Kafka cluster. It is responsible for receiving, storing, and serving messages for Kafka topics. Each broker can handle multiple partitions from various topics. Kafka brokers work together to form a highly available and fault-tolerant messaging platform.

A broker in Kafka refers to a single node/server in the Kafka cluster. Brokers are responsible for receiving, storing, and serving messages for Kafka topics. Each broker manages one or more partitions of one or more topics. A Kafka cluster consists of multiple brokers working together to form a distributed and fault-tolerant message storage system.

When you set up a Kafka cluster, you deploy multiple brokers that collaborate to manage the overall message flow and data storage.

1. **Topic:** A topic is a category or feed name to which messages are published. It represents a stream of records in Kafka. Topics are divided into one or more partitions to enable scalability and parallel processing. Producers publish messages to topics, and consumers subscribe to topics to consume those messages.
2. **Partition:** A partition is a logical unit that represents a portion of a Kafka topic. Each topic can be divided into one or more partitions. Partitions allow data to be distributed and processed in parallel across multiple brokers in the cluster. Messages within a partition are ordered and immutable, ensuring that data with the same key goes to the same partition and preserving the order of messages.

A partition is a fundamental unit of parallelism and scalability in Kafka. Topics in Kafka are divided into one or more partitions, and each partition is hosted on a specific broker. Partitions allow you to split the data of a topic across multiple servers (brokers) to handle large message streams and provide parallel processing and read/write capabilities. Each message published to a topic is assigned to one of the partitions based on the message's key or using a round-robin mechanism if no key is provided.

**each partition can be placed on a separate machine to allow for multiple consumers to read from a topic in parallel. In Kafka**, the **placement of partitions on separate machines is handled automatically** by the Kafka brokers when you create topics and start producing data. The distribution of partitions across brokers allows for parallel processing and load balancing among consumers.

In summary, brokers are the individual nodes that make up a Kafka cluster, while partitions are the subdivisions of a topic that allow for scalable and distributed data storage and processing across multiple brokers. Each partition is hosted by a single broker, but a broker can handle multiple partitions from different topics.

1. **Replication:** For fault tolerance and high availability, Kafka allows you to replicate each partition across multiple brokers. Each partition has a configurable replication factor that determines how many copies (replicas) of that partition are maintained in the cluster. Replicas ensure that data is still available even if some brokers fail.

**To summarize the flow:**

**Cluster is a collection of multiple Kafka brokers working together.**

**Each broker is a single node/server in the Kafka cluster.**

**Topics are logical categories or feeds to which messages are published.**

**Partitions are subdivisions of a topic that allow for scalable and distributed data storage and processing across multiple brokers.**

In practical terms, when you produce a message to a topic, the message will be assigned to one of the partitions for that topic. The Kafka cluster and brokers manage the distribution of messages across the partitions and ensure that consumers can read the messages in parallel from the partitions.

**scenarios that can cause for a Kafka cluster down**

There are much possible thins that for down a Kafka cluster, Downtime can occur due to various reasons, including hardware failures, network issues, software bugs, misconfigurations, or maintenance activities. Even though Kafka is designed to be highly available and fault-tolerant, unexpected failures or planned maintenance events can lead to cluster downtime.

**Here are some scenarios that can cause Kafka cluster downtime:**

1. **Broker Failure:** If one or more Kafka brokers fail, the partitions hosted on those brokers may become temporarily unavailable until they are either recovered or reassigned to healthy brokers.
2. **Network Partition**: Network issues can lead to communication problems between brokers, causing the cluster to split into separate partitions. In such cases, the partitions may not be able to replicate data to each other until the network issue is resolved.
3. **Zookeeper Failure**: Kafka uses Apache **ZooKeeper** for coordination and maintaining metadata. If **ZooKeeper** becomes unavailable, the Kafka cluster may not be able to function correctly.
4. **Disk Full**: If the disk space on Kafka brokers gets exhausted, it can prevent Kafka from writing new messages or replicating data.
5. **Software Bugs**: Software bugs or configuration errors can cause unexpected behavior in Kafka, leading to performance issues or cluster instability.
6. **Planned Maintenance**: Scheduled maintenance activities, such as software upgrades or hardware replacements, may require planned downtime to ensure a smooth and controlled process.

To minimize the impact of potential downtime, Kafka provides mechanisms for replication, data redundancy, and fault tolerance. Kafka's design allows you to set up multiple brokers, replicate data across brokers, and have multiple replicas of each partition, ensuring that data remains available even during broker failures. Additionally, careful cluster management, monitoring, and disaster recovery planning can help reduce the risk and impact of potential downtime.

If you are using Kafka in a critical production environment, it is essential to implement best practices for high availability and ensure you have contingency plans in place to handle potential downtime scenarios.