Kafka

Kafka is an open-source **message broker** project developed by the Apache Software Foundation written in Scala and is a distributed **publish-subscribe messaging system**. Kafka's design pattern is mainly based on the **transactional logs** design.

Feature	**Description**
High Throughput	Support for millions of messages with modest hardware
Scalability	Highly scalable distributed systems with no downtime
Replication	Messages are replicated across the cluster to provide support for multiple subscribers and balances the consumers in case of failures
Durability	Provides support for persistence of message to disk
Stream Processing	Used with real-time streaming applications like Apache Spark & Storm
Data Loss	Kafka with proper configurations can ensure zero data loss

List the various components in Kafka:

- Topic – a stream of messages belonging to the same type
- Producer – that can publish messages to a topic
- Brokers – a set of servers where the publishes messages are stored
- Consumer – that subscribes to various topics and pulls data from the brokers.

What can you do with Kafka?

- In order to transmit data between two systems, we can build a real-time stream of data pipelines with it.
- Also, we can build a real-time streaming platform with Kafka, that can actually react to the data.

What are the types of traditional method of message transfer?

- **Queuing**: It is a method in which a pool of consumers may read a message from the server and each message goes to one of them.
- **Publish-Subscribe**: Whereas in Publish-Subscribe, messages are broadcasted to all consumers.

Why should we use Apache Kafka Cluster?

In order to overcome the challenges of collecting the large volume of data, and analyzing the collected data we need a messaging system. Hence Apache Kafka came into the story. Its benefits are:

- It is possible to track web activities just by storing/sending the events for real-time processes.

- Through this, we can Alert as well as report the operational metrics.
- Also, we can transform data into the standard format.
- Moreover, it allows continuous processing of streaming data to the topics.

Topic VS Queue

Is Kafka a Topic or a Queue?

https://abhishek1987.medium.com/kafka-is-it-a-topic-or-a-queue-30c85386afd6

Kafka is both a Topic and a Queue. Queue based systems are typically designed in a way that there are multiple consumers processing data from a queue and the work gets distributed such that **each consumer gets a different set of items to process**. Hence there is no overlap, allowing the workload to be shared and enables horizontally scalable architectures.

A Kafka topic is sub-divided into units called partitions for fault tolerance and scalability. **Consumer Groups allow Kafka to behave like a Queue**, since each consumer instance in a group processes data from a non-overlapping set of partitions (within a Kafka topic).

Zookeeper role

Kafka uses **Zookeeper** to store offsets of messages consumed for a specific topic and partition by a specific Consumer Group. (Message offset role: messages contained in the partitions are assigned a unique ID number that is called the offset. The role of the offset is to uniquely identify every message within the partition)

It is not possible to bypass Zookeeper and connect directly to the Kafka server. If, for some reason, ZooKeeper is down, you cannot service any client request.

Apache ZooKeeper is an open-source server for highly reliable distributed coordination of cloud applications. It is a project of the Apache Software Foundation.

ZooKeeper is essentially a service for distributed systems offering a hierarchical **key-value store**, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed systems. ZooKeeper was a sub-project of Hadoop but is now a top-level Apache project in its own right.

Typical use cases for ZooKeeper are:

- Naming service
- Configuration management
- Data Synchronization
- Leader election
- Message queue
- Notification system

Leader-Follower

Every partition in Kafka has one server which plays the role of a Leader, and none or more servers that act as Followers. The Leader performs the task of all read and write requests for the partition, while the role of

the Followers is to passively replicate the leader. In the event of the Leader failing, one of the Followers will take on the role of the Leader. This ensures load balancing of the server.

Replicas

Replicas are essentially a list of nodes that **replicate the log for a particular partition** irrespective of whether they play the role of the Leader. On the other hand, ISR stands for In-Sync Replicas. It is essentially a set of message replicas that are synced to the leaders.

Replication ensures that published messages are not lost and can be consumed in the event of any machine error, program error or frequent software upgrades.