A

2022

APACHE KAFKA

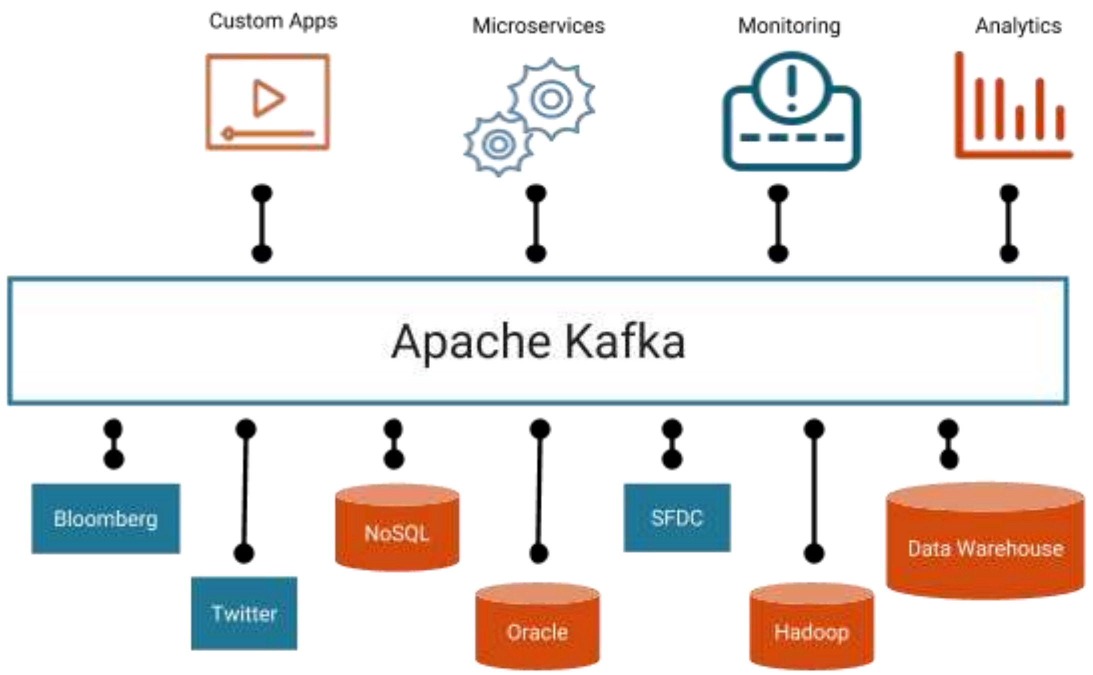
Arvind Singh

[APACHE KAFKA]

(ARVIND SINGH)

cloudwatch

Apache Kafka Architecture - Getting Started with Apache Kafka



In the '90s CLIENT-SERVER architecture was popular. There was a Monolith structure here. In ***Monolith*** structure; all structures are included in a single application. Access to the internet increased over time. And version updates have become a constant need. For this reason, the number of people who prefer this structure has decreased over time.

Then ***Microservices*** came to the fore. But, because there are too many applications in microservices, the issue of how applications will communicate with each other has been a big problem.

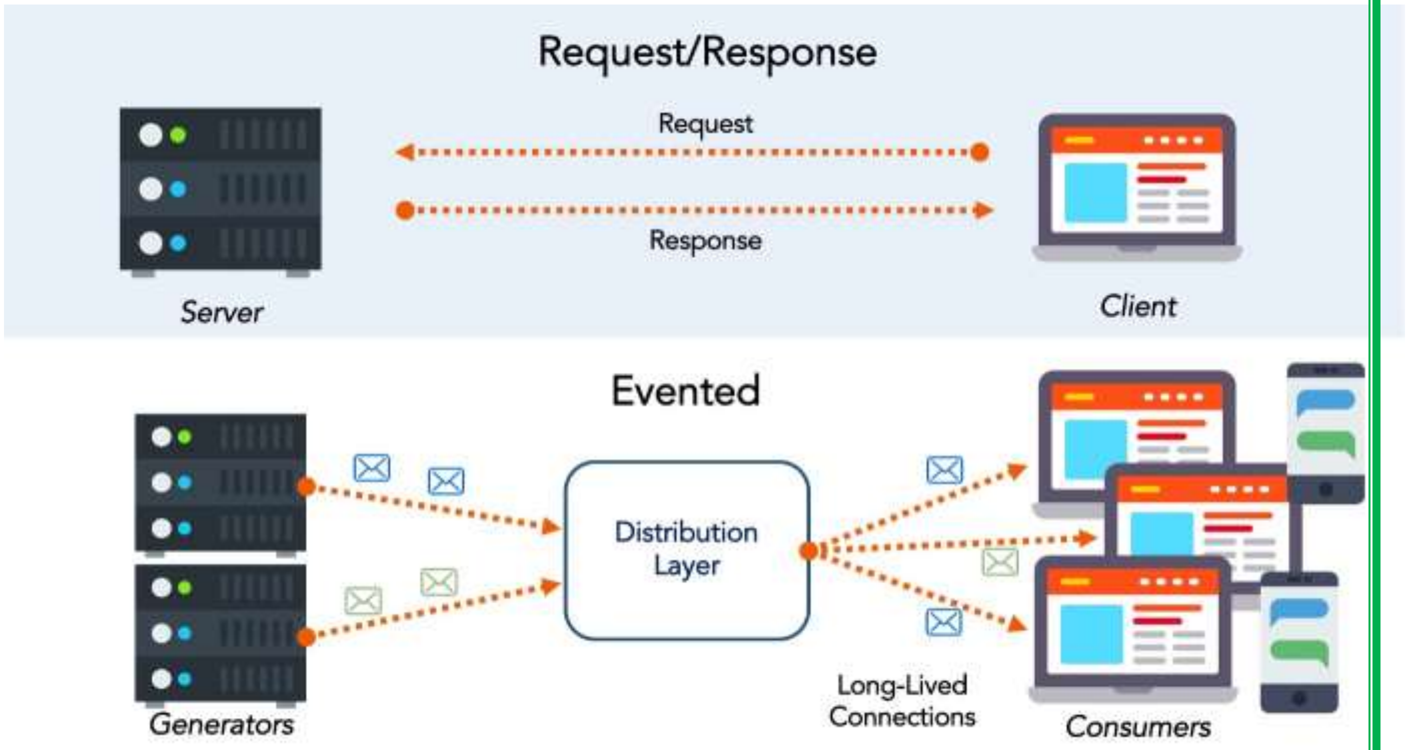
As time passed, new components started to emerge. Each newly released component led to the emergence of many new connections in the application network.

In the following times, two different architectures started to be used.

`Request/Response Architecture`; In this architecture, one piece was calling the other and waiting for that to answer. The network was very large and complex. Working with this architecture became a bit difficult as the data sizes grew.

The other is `Event-Driven Architecture`; ***The Message Bus*** made possible the implementation of this architecture. Now, communication with `Event`s has started between services. No service was directly contacting the other.

Each piece does things asynchronously, without being linked to each other. There are many other applications that use `Message Bus`. Apache Kafka is one of them and we can say the most popular.



So how does Apache Kafka do this messaging?

In Kafka, messages are sent to places called `Topic`. Each Topic has a name. Also, others can read from these Topics.

What is Apache Kafka?

Apache Kafka, in its most general definition, is a distributed messaging system. Some of Kafka's features are;

- Stream Processing Platform
- Open Source Software
- Distributed System

- Fully Scalable
- High Performance - Low Latency

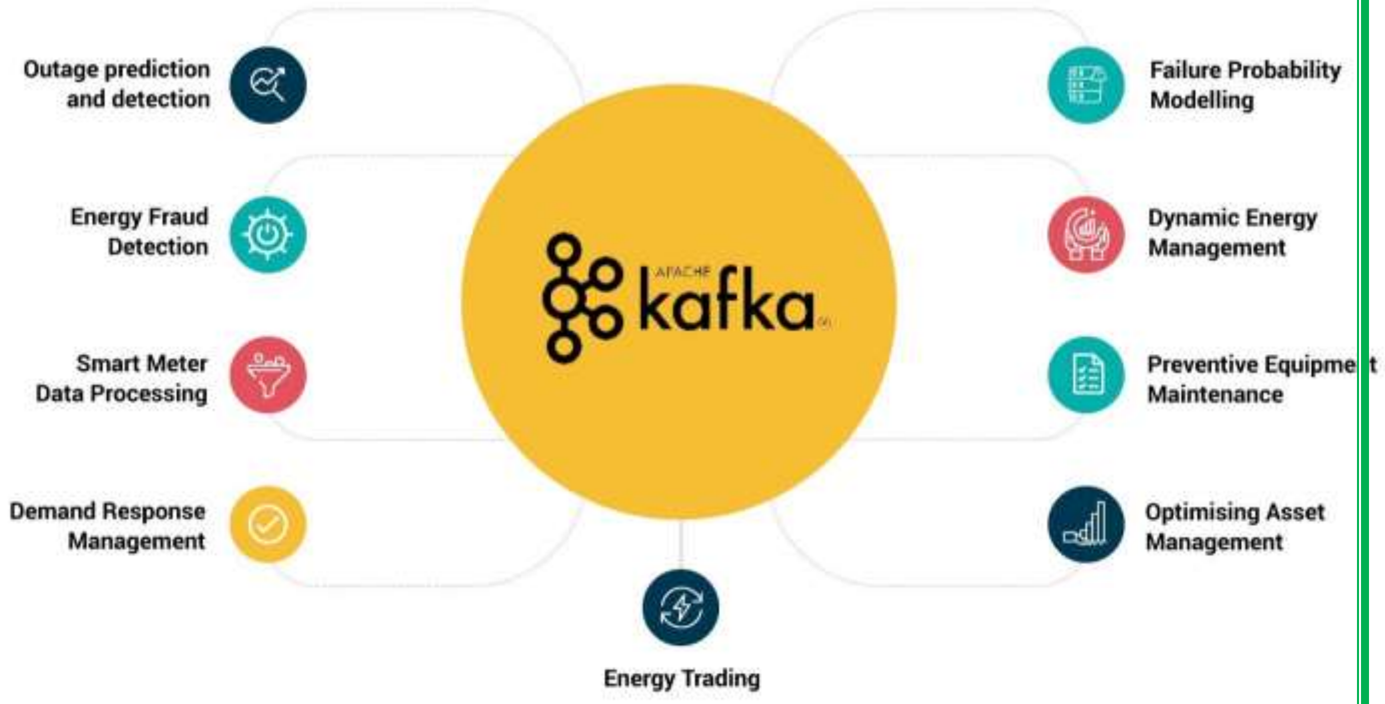
Apache Kafka; is used by many major companies such as Netflix, Apple, Uber, Spotify, and LinkedIn. Now, real-time stream processing has become very important for companies.

WHO USES APACHE KAFKA?
A few small companies you might have heard of...
UBER
CISCO
DATADOG
NETFLIX
PayPal
Spotify
Walmart
Microsoft
YAHOO!
Linked in
BUILDING EVENT-DRIVEN SYSTEMS WITH APACHE KAFKA

What Can We Do With Apache Kafka?

- Fraud and anomaly detection
- Recommendation engine

- Monitoring / Metrics
- Activity Tracking
- Integrate systems
- Real-time stream processing

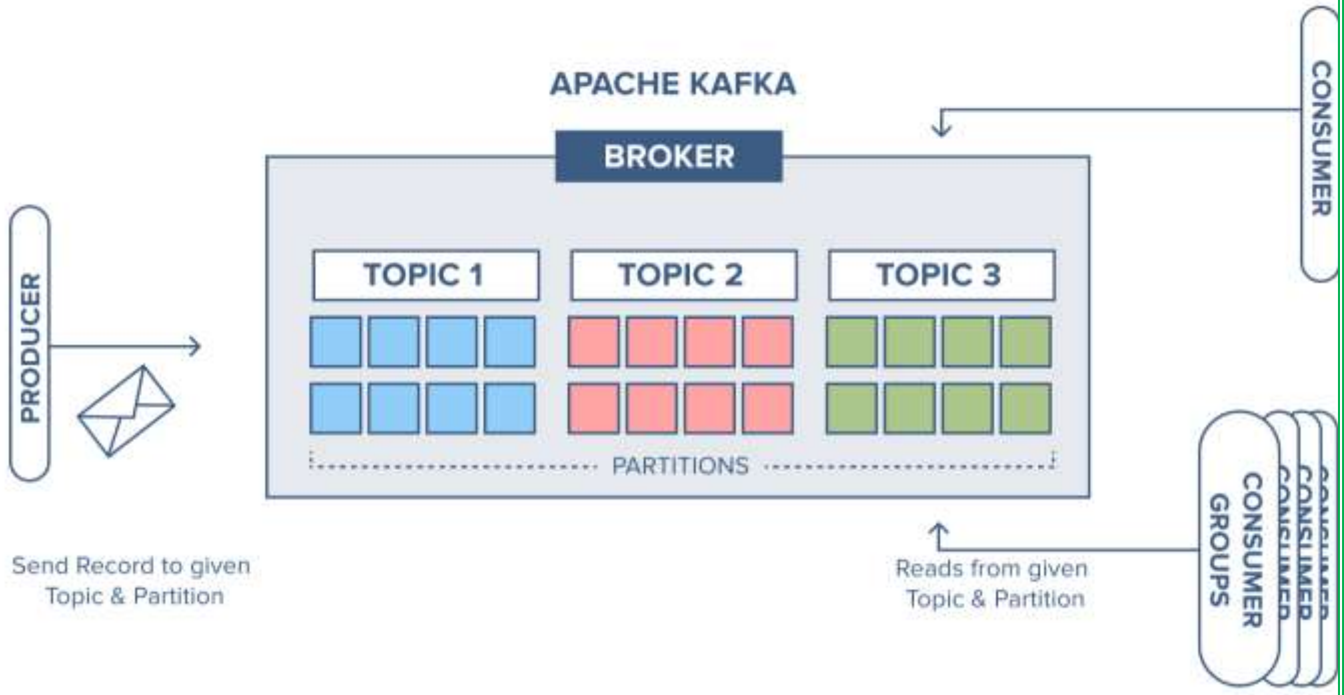

Outage prediction and detection
Energy Fraud Detection
Smart Meter Data Processing
Demand Response Management
APACHE kafka
Failure Probability Modelling
Dynamic Energy Management
Preventive Equipme t Maintenance
Optimising Asset Management
Energy Trading

Architecture of Apache Kafka

Kafka architecture has 4 actors. These are;

(1) Broker (2) Zoo keeper

(3) Producer (4) Consumer



Each Kafka cluster consists of one or more `Brokers`. Each Broker works interdependently. Each Broker has its own unique identification number. Messages sent to Kafkaya in brokers are stored and processed. Computer memory is not used for data storage. Data is stored on the hard drive.

`Zookeeper` is open-source software. Kafka uses Zookeeper to manage all Brokers. The data sent is never stored here. Zookeeper's responsibilities are:

- Coordinating brokers.
- Choosing the Leader Partition.
- To ensure that brokers get to know each other.

- Discovering new or deleted Brokers or newly added, changed Topics.

`Producer`; writes data to Kafka. `Consumer`reads data from Kafka.

In real life, the number of Brokers and Zookeepers should consist of odd numbers and should be a minimum of 3. The reason for this is to ensure that the replication process can be performed properly and to prevent problems such as `Split-Brain`.

Replication is the reproduction of data for the desired number of times. In other words, a copy of the data is created and reproduced. What should be the number of replications? We can prove this with a simple formula.

```
Number of Brokers > = Number of Replications
```

How to Write Data to Apache Kafka?

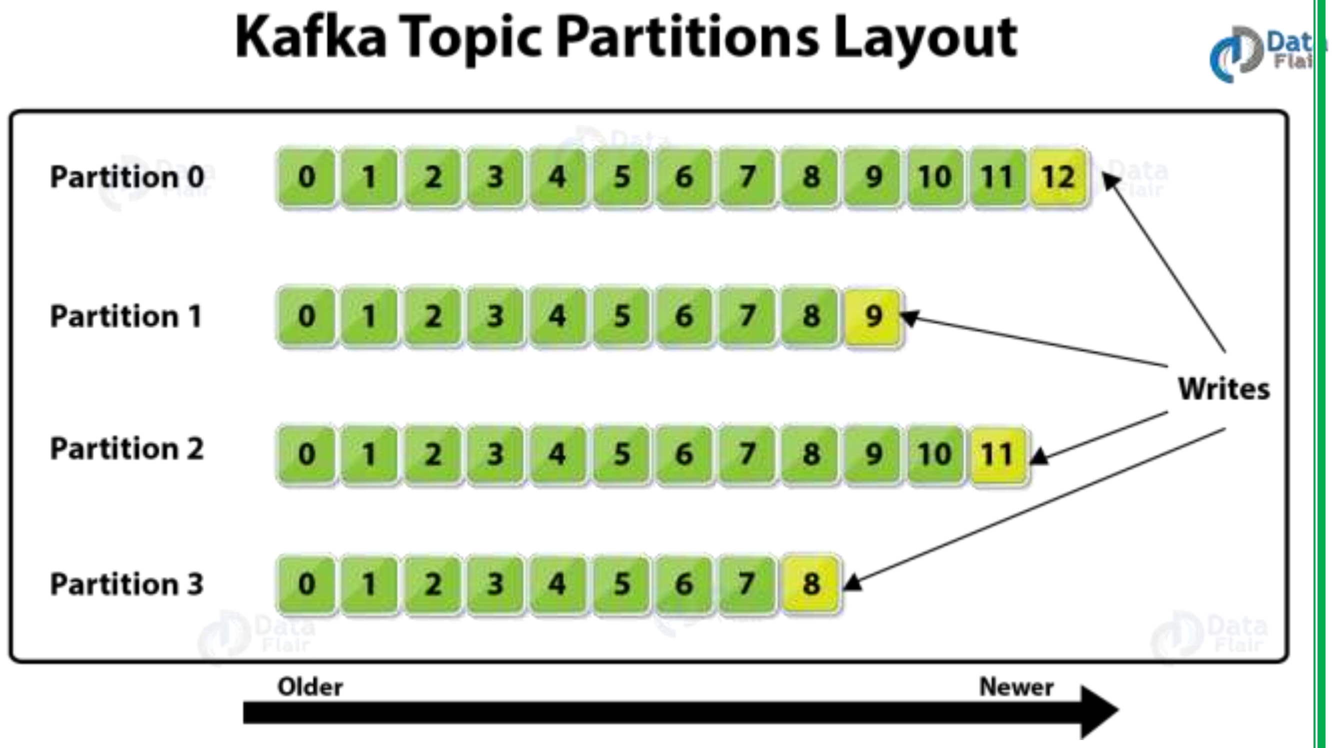
Apache Kafka stores data using Topic. Each Topic has its own name. Topics are stored on Brokers.

There are Partitions in Topics. So Topic is a structure consisting of Partitions. Data is actually written to any partition in Topics. We can decide to determine the number of Partitions for each Topic.

Partition actually uses the Log principle. That is, data is constantly added to the back. We cannot add a beginning or middle of the partition. Data is written in the order sent to Partition. It is in order from old to new. A written data cannot be changed again. The data in the partition is not stored here forever. Kafka has two storage configurations. These are:

- Time-based retention (e.g. 7 days)
- Data size based storage (e.g. 100 GB)

An Offset is defined for each message in the partition. Data is read using this offset. Actually, Offsets are for determining the position of data.



Why Do We Create Partition?

- To `Aggregation` according to the features specified in the Producer data
- Keeping data in order (`Sorting — Event-Sourcing`)
- Reading data faster by increasing `Parallelism`
- Storing data more efficiently (`Efficiency`)

How is data written to Partition?

We can give a key to the messages we send to Kafka. Kafka writes the same key values to the same Partition. Suppose we give no Key. In this case, Kafka applies the Round-Robin method. In other words, it writes the incoming messages by dividing them into Partitions. For example, if we give the Category Key-Value, Kafka writes the data of the same category into the same Partition.

Some things to know about Partition: For example, if there was only one Partition, the performance to read and write data would be very low. Parallelism would not exist.Only one Consumer with the same ID can read in a partition at a time.Using partition, we can route large messages the way we want.

Leader Partition Election

One of the replication copies of each Partition is assigned as the Leader Partition. Data is always written to these Leader Partitions. Then these Leader Partitions provide synchronization by sending the data to other copies. If a Leader Partition crashes, one of the duplicates is assigned as Leader by Kafka.

Producer Acknowledgement

acks =0 → The fastest but most risky method. Data is sent to Kafka, it continues without waiting for an answer.

acks = 1 → Medium fast, considered safe. The data is sent to Kafka, it waits until the data is written to the Leader Partition, then continues.

acks = 2 → The slowest, safest method. Data is sent to Kafka, data is written to Leader Partitions after Leader Partition writes to other copies, the system continues.

How to Read Data from Apache Kafka?

The first thing Kafka determines when reading data is from which position to read the data. This position is called Offset. If a reader has already read or discontinued, it is clear where he last stayed in Zookeeper. And Kafka continues to read where that left off.

Information Reading Strategies from Kafka

Data can be read with 3 different strategies.

- **At most once:** There is a high risk of messages being lost. Read, Commit, Process, and Save to Database steps are applied respectively.
- **At least once:** It is the most preferred strategy. We make sure that we process the message without errors. Read, Process, Save to Database and Commit steps are applied respectively.
- **Exactly once:** The incoming message is held in a Transaction. The message will not disappear even if there is any break in the process. But this method has a huge impact on performance. It should be used only when necessary. Read, Transaction, and Commit steps are applied respectively.