Big Data

Big data refers to extremely large and diverse collections of structured, unstructured, and semi-structured data that continues to grow exponentially over time. These datasets are so huge and complex in volume, velocity, and variety, that traditional data management systems cannot store, process, and analyze them.

Types of Big Data

Big data can be classified into structured, semi-structured, and unstructured data. Structured data is highly organized and fits neatly into traditional databases. Semi-structured data, like JSON or XML, is partially organized, while unstructured data, such as text or multimedia, lacks a predefined structure.

Characteristics of Big Data

Big data is characterized by the "5 Vs": Volume (the sheer amount of data), Velocity (the speed at which data is generated and processed), Variety (the diversity of data types), Veracity (the accuracy and trustworthiness of data), and Value (the insights derived from the data).

List of 20 Big Data tools

1. Hadoop
2. Spark
3. NoSQL databases (MongoDB, Cassandra)
4. SQL databases (MySQL, PostgreSQL)
5. Hive
6. Pig
7. Flink
8. Kafka
9. HBase
10. Presto
11. Elasticsearch
12. Splunk
13. Tableau
14. Power BI
15. Talend
16. Apache NiFi
17. TensorFlow
18. RapidMiner
19. KNIME
20. DataRobot

Apache Kafka:

Apache Kafka is a distributed streaming platform used for high-throughput, fault-tolerant, and real-time data streaming, often used for data pipelines, streaming analytics, and data integration.

Apache Kafka is a free and open tool we use for streaming events. It helps us build real-time data pipelines and streaming apps. Kafka allows organizations to publish, subscribe to, store, and process streams of records in a way that is safe from errors.

Apache Kafka is a **publish-subscribe messaging system**. A messaging system lets you send messages between processes, applications, and servers. Broadly Speaking, Apache Kafka is software where topics (a topic might be a category) can be defined and further processed. Applications may connect to this system and transfer a message onto the topic. A message can include any kind of information from any event on your blog or can be a very simple text message that would trigger any other event.

**Kafka** is an **open-source messaging system** that was created by LinkedIn and later donated to the Apache Software Foundation. It’s built to handle large amounts of data in real time, making it perfect for creating systems that respond to events as they happen.

Kafka organizes data into categories called “topics.” Producers (apps that send data) put messages into these topics, and consumers (apps that read data) receive them. Kafka ensures that the system is reliable and can keep working even if some parts fail.

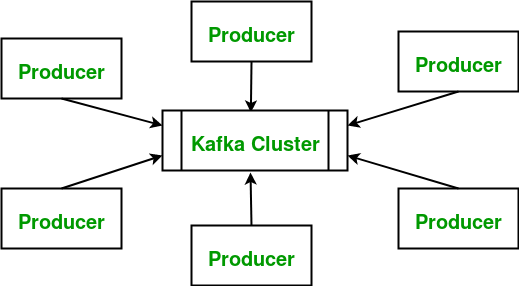
**Core Components of Apache Kafka**

**1. Kafka Broker**

A Kafka broker is a server that runs Kafka and stores data. Typically, a Kafka cluster consists of multiple brokers that work together to provide scalability, fault tolerance, and high availability. Each broker is responsible for storing and serving data related to topics.

**2. Producers**

A producer is an application or service that sends messages to a Kafka topic. These processes **push data** into the Kafka system. Producers decide which topic the message should go to, and Kafka efficiently handles it based on the partitioning strategy.



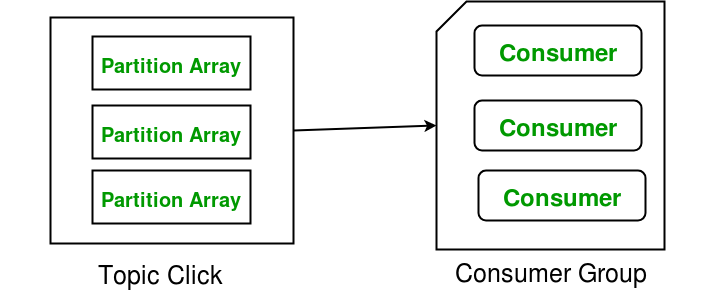
**3. Kafka Topic**

A topic in Kafka is a category or feed name to which messages are published. Kafka messages are always associated with topics, and when you want to send a message, you send it to a specific topic. Topics are divided into partitions, which allow Kafka to scale horizontally and handle large volumes of data.

**4. Consumers and Consumer Groups**

A Consumer is an application that reads messages from Kafka topics. Kafka allows consumer groups, where multiple consumers can read from the same topic, but Kafka ensures that each message is processed by only one consumer in the group. This helps with load balancing and allows consumers to read messages starting from any offset.

**Partitions allow you to parallelize a topic by splitting the data in a particular topic across multiple brokers.**



**5. Zookeeper**

Kafka uses**Apache ZooKeeper** to manage metadata, control access to Kafka resources, and handle **leader election** and **broker coordination.** ZooKeeper ensures high availability by making sure the Kafka cluster remains functional even if a broker fails.

**Important Concepts of Apache Kafka**

* **Topic partition**: Kafka topics are divided into a number of partitions, which allows you to split data across multiple brokers.
* **Consumer Group**: A consumer group includes the set of consumer processes that are subscribing to a specific topic.
* **Node**: A node is a single computer in the Apache Kafka cluster.
* **Replicas:** A replica of a partition is a “backup” of a partition. Replicas never read or write data. They are used to prevent data loss.
* **Producer**: Application that sends the messages.
* **Consumer**: Application that receives the messages.

**Why is Apache Kafka Needed?**

With businesses collecting **massive volumes of data** in real time, there is a need for tools that can handle this data efficiently. Kafka solves several key problems:

1. **Real-Time Processing**: Kafka is optimized for handling real-time data streams, allowing businesses to process and act on data as it happens.
2. **Fault-Tolerant**: Kafka ensures that even if parts of the system fail, data won’t be lost, making it a highly reliable messaging system.
3. **Scalable**: Kafka scales horizontally by adding more brokers, allowing it to handle growing data loads and increasing numbers of producers and consumers.
4. **Event-Driven Architecture**: Kafka powers **event-driven architectures**, enabling systems to respond to events in real-time without having to constantly poll for changes.

**How Apache Kafka Works**

Apache Kafka moves data from one place to another in a smooth and reliable way. Here’s how it works in simple terms:

**Step 1: Producers Send Data**

* Producers are applications that **create data** and send it to Kafka.
* This data can be anything—logs, transactions, user activities, or events.
* Kafka **splits** the data into smaller parts called **partitions**, making it easier to handle large amounts of information.

**Step 2: Kafka Stores the Data**

* Kafka **organizes** the data into **topics**, where it is saved for a certain period.
* Even if a consumer reads the data, Kafka **doesn’t delete it immediately**.
* To prevent data loss, Kafka **makes copies** of the data and stores them on different servers.

**Step 3: Consumers Read the Data**

* Consumers are applications that **subscribe to topics** and read messages.
* To manage the load, consumers are divided into **consumer groups**, so no message is processed twice.
* Consumers can **choose where to start reading**, whether from the newest message or an earlier point.

**Step 4: Kafka Balances the Load**

* **ZooKeeper** helps Kafka manage which server is in charge of storing and distributing data.
* If a server goes down, Kafka automatically **redirects the data** to another server.

**Step 5: Data is Processed and Used**

* Once consumers receive the data, they can **store it in a database, analyze it, or trigger other events**.
* Kafka can work with tools like **Apache Spark, Flink, and Hadoop** for deeper analysis.

**How Kafka Integrates Different Data Processing Models**

Apache Kafka is highly versatile and can seamlessly integrate various data processing models, including **event streaming**, **message queuing**, and **batch processing**.

**1. Event Streaming (Publish-Subscribe Model)**

Kafka’s primary function is **event streaming**, where:

* **Producers** (applications sending data) publish messages to Kafka topics.
* **Consumers** (applications reading data) subscribe to topics and receive messages as soon as they arrive.
* Multiple consumers can read the same message, allowing for real-time data distribution.

***Example****: A stock trading platform can use Kafka to stream live market data to multiple dashboards.*

**2. Message Queue (Point-to-Point Processing)**

Kafka can also act like a **message queue** by using **consumer groups**:

* When multiple consumers are in the same group, Kafka **distributes messages among them**, ensuring each message is processed only once.
* This setup helps in **load balancing**, making sure no single consumer is overwhelmed.

***Example****: A ride-hailing app like Uber can use Kafka to assign incoming ride requests to available drivers efficiently.*

**3. Batch Processing**

Even though Kafka is designed for real-time data, it can also handle **batch processing**:

* Messages can be stored in Kafka topics and processed later.
* Tools like **Apache Spark or Hadoop** can read data from Kafka in batches and perform analytics.

***Example****: An e-commerce company can collect website visitor data in Kafka and analyze it later to improve product recommendations.*

**4. Hybrid Model (Real-Time + Batch Processing)**

Kafka is flexible enough to support a **mix of real-time and batch processing**:

* It can send data immediately for real-time analytics while also storing it for batch processing later.
* This is often done using **Kafka Streams, Spark Streaming, or Flink**.

***Example****: A fraud detection system can process transactions in real time to flag suspicious activity while also running deeper batch analysis at the end of the day.*

**Common Use Cases of Apache Kafka**

Apache Kafka is widely used across various industries. Some popular use cases include:

* **Real-time Analytics:** Processing data streams for live analytics, like monitoring user activities or stock prices.
* **Event-Driven Applications:** Kafka powers event-driven architectures, ensuring that systems react in real time to events like user actions, transactions, or sensor data.
* **Log Aggregation**: Collecting logs from multiple systems into a centralized logging system for better analysis and monitoring.
* **Stream Processing:** Kafka, along with tools like Apache Flink or Apache Spark, is used to process streams of data in real-time.
* **Data Integration:** Kafka integrates data between different systems, such as moving data between different microservices or syncing databases.

**Companies using Apache Kafka**

The following shows the list of companies using Apache Kafka:

| **Company** | **Use Case** |
| --- | --- |
| **LinkedIn** | Uses Kafka to manage real-time activity streams, news feeds, and operational metrics. |
| **Netflix** | Streams real-time data for monitoring, analytics, and recommendations |
| **Twitter** | Processes live tweets, trends, and analytics using Kafka. |
| **Uber** | Tracks real-time ride locations and processes event-driven data. |
| **Airbnb** | Manages real-time booking, pricing, and user analytics. |
| **Spotify** | Analyzes music streaming data and user behavior in real time. |
| **Pinterest** | Handles event logging and recommendation systems. |
| **Walmart** | Uses Kafka for inventory tracking and fraud detection. |
| **Box** | Implements Kafka for real-time monitoring and analytics. |
| **Goldman Sachs** | Uses Kafka for financial data streaming and trading analysis. |

**Benefits of Apache Kafka**

The following are some of the benefits of using Apache Kafka:

**1. Handles Large Data Easily**

Kafka is designed to handle large volumes of data, making it ideal for businesses with massive data streams.

**2. Reliable & Fault-Tolerant**

Even if some servers fail, Kafka keeps data safe by making copies.

**3. Real-Time Data Processing**

Perfect for applications that need instant data updates.

**4. Easy System Integration**

Producers and consumers work independently, making it flexible.

**5. Works with Any Data Type**

Can handle structured, semi-structured, and unstructured data.

**6. Strong Community Support**

With many companies using Kafka, there is a large and active community supporting it, along with integrations with tools like Apache Spark and Flink.

**Limitations of Apache Kafka**

The following are some of the limitations you have to face while using Apache Kafka:

**1. Difficult to Set Up**

Requires technical knowledge to install and manage.

**2. Storage Can Be Expensive**

Since it saves messages for some time, costs may rise.

**3. Message Order Issues**

Guarantees order only within a single partition, not across multiple ones.

**4. No Built-in Processing**

Needs extra tools for transforming or analyzing data.

**5. Needs High Resources**

Uses a lot of CPU, memory and network bandwidth.

**6. Not Ideal for Small Messages**

Better for large data streams; smaller tasks may have unnecessary overhead.

**Features of Apache Kafka**

Many companies rely on Apache Kafka because it helps them process large amounts of data in real time. Here’s why it’s so popular:

**1. Scalability**

Kafka can handle massive amounts of data by breaking it into smaller pieces (partitions) and distributing them across multiple servers. This means it can grow as a business’s data needs increase.

**2. Fault Tolerance**

Even if some servers fail, Kafka keeps running smoothly because it makes copies of data (replication). This ensures that no important information is lost.

**3. Flexibility**

Kafka can work with any type of data since it stores information as byte arrays. Whether it’s logs, events, or structured records, Kafka can handle it all.

**4. Offset Management**

Consumers (applications that read data) don’t have to start from the beginning every time—they can pick up exactly where they left off. This makes it easier to process data without interruptions.

**Other Apache Technologies often used with Kafka**

Apache Kafka works well with several **Apache technologies** that help improve data management, processing, and integration. Here’s how they work together:

**1. Apache ZooKeeper**

Kafka relies on **ZooKeeper** to manage cluster information, such as keeping track of active brokers and handling leader elections. It ensures the system runs smoothly.

**2. Apache Avro**

Kafka often uses **Avro** for data serialization. It makes storing and sharing structured data more efficient while allowing schema changes without breaking compatibility.

**3. Apache Flink**

Kafka and **Flink** work together to process real-time data streams. Flink helps analyze data as it arrives, making it useful for live monitoring, fraud detection, and event-driven applications.

**4. Apache Spark**

**Spark** can read data from Kafka for both real-time and batch processing. It is widely used for machine learning, ETL (Extract, Transform, Load) tasks, and big data analytics.

**5. Apache Hadoop**

Kafka streams large amounts of data, and **Hadoop** provides long-term storage for deep analysis. This combination is useful for businesses handling massive datasets.

**6. Apache Storm**

For real-time, low-latency processing, **Storm** works well with Kafka. It helps in applications like tracking live events, detecting unusual activities, or updating dashboards in real time.

**7. Apache Camel**

Kafka often integrates with different systems using **Camel**, which acts as a bridge between Kafka and various APIs, databases, or cloud services. It simplifies message routing and data transformation.

**8. Apache NiFi**

**NiFi** automates data flow between Kafka and other sources or destinations. It helps build scalable data pipelines without needing extensive coding.

These tools make Kafka more powerful, helping companies handle real-time data efficiently.

**What is Confluent Kafka?**

Confluent Kafka serves as a data streaming platform that includes almost all of Kafka's functionality and a few other things. Its primary goal is not just to provide a pub-sub platform, but also to provide data storage and processing services.

| **Apache Kafka** | **Confluent Kafka** |
| --- | --- |
| Apache Kafka provides you the flexibility and features to customize the code to your specifications. | Confluent Kafka helps organizations build real-time data applications. |
| Apache Kafka 2.0 license is free and open source | Confluent Kafka 2.0 License is also free and open-source. |
| Apache Kafka is easily accessible and provides real-time feedback. | Confluent Kafka is not easily accessible and can not provide real-time feedback. |
| In REST Proxy, Apache Kafka can not access from any network-connected application. | In REST Proxy, Confluent Kafka can directly access any network-connected application. |
| Apache Kafka has authorization controls for the cluster data. | Confluent Kafka also has authorization controls for the cluster data. |
| It is not serverless, as Apache Kafka is not fully managed and automated, with zero operations. | It is serverless, as Confluent Kafka is fully managed and automated, with zero operations. |
| There is a ZooKeeper management in Apache Kafka and metadata management is abstracted away. | There is no ZooKeeper management in Confluent Kafka. |
| Apache Kafka lacks some connectors and add-ons that can make implementation easier. | Users cannot provide some SaaS or other services that use platform-integrated technologies. |
| Apache Kafka does not offer 24/7 expert support, with over 1 million hours of experience. | Confluent Kafka offers all-time expert support, with over 1 million hours of experience. |
| Apache Kafka | Confluent Kafka |
| Apache Kafka provides you the flexibility and features to customize the code to your specifications. | Confluent Kafka helps organizations build real-time data applications. |
| Apache Kafka 2.0 license is free and open source | Confluent Kafka 2.0 License is also free and open-source. |
| Apache Kafka is easily accessible and provides real-time feedback. | Confluent Kafka is not easily accessible and can not provide real-time feedback. |
| In REST Proxy, Apache Kafka can not access from any network-connected application. | In REST Proxy, Confluent Kafka can directly access any network-connected application. |
| Apache Kafka has authorization controls for the cluster data. | Confluent Kafka also has authorization controls for the cluster data. |
| It is not serverless, as Apache Kafka is not fully managed and automated, with zero operations. | It is serverless, as Confluent Kafka is fully managed and automated, with zero operations. |
| There is a ZooKeeper management in Apache Kafka and metadata management is abstracted away. | There is no ZooKeeper management in Confluent Kafka. |
| Apache Kafka lacks some connectors and add-ons that can make implementation easier. | Users cannot provide some SaaS or other services that use platform-integrated technologies. |
| Apache Kafka does not offer 24/7 expert support, with over 1 million hours of experience. | Confluent Kafka offers all-time expert support, with over 1 million hours of experience. |

Spark and Kafka

Kafka and Spark are both stream-processing frameworks designed to process data in real time. They share many features, such as fault tolerance, scalability, high throughput/low latency message delivery, automatic offset management, and integration with multiple languages.

However, there are some key differences between them. Kafka focuses on messaging (publishing/subscribing), while Spark focuses more on data processing with support for batch processing and SQL queries. Kafka is designed to process data from multiple sources, whereas Spark is designed to process data from only one source.

Hadoop, on the other hand, is a distributed framework that can store and process large amounts of data across clusters of commodity hardware. It provides support for batch processing and SQL queries but lacks the real-time processing capabilities provided by Kafka and Spark.