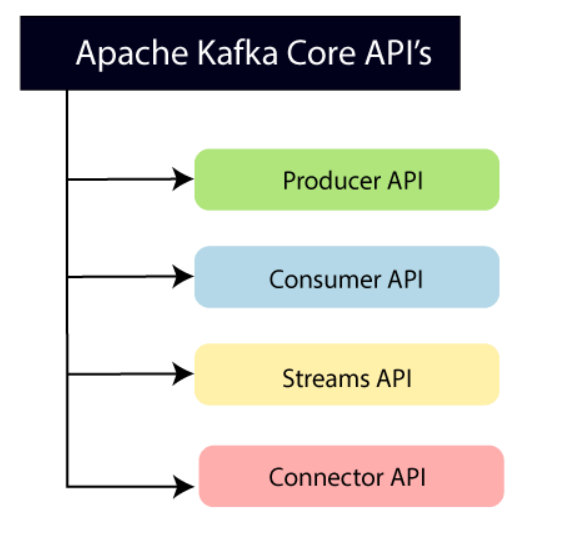
**Apache Kafka**

It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by **LinkedIn**, and later it was donated to the Apache Software Foundation. Apache Kafka is an open-source stream-processing software platform which is used to handle the real-time data storage. It works as a broker between two parties, i.e., a sender and a receiver. It can handle about trillions of data events in a day.

**Messaging System**

A messaging system is a simple exchange of messages between two or more persons, devices, etc. A publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message. In Apache Kafka, a sender is known as a **producer** who publishes messages, and a receiver is known as a **consumer** who consumes that message by subscribing it.



**Producer API:** This API allows/permits an application to publish streams of records to one or more topics. (discussed in later section)

**Consumer API:** This API allows an application to subscribe one or more topics and process the stream of records produced to them.

**Streams API:** This API allows an application to effectively transform the input streams to the output streams. It permits an application to act as a stream processor which consumes an input stream from one or more topics, and produce an output stream to one or more output topics.

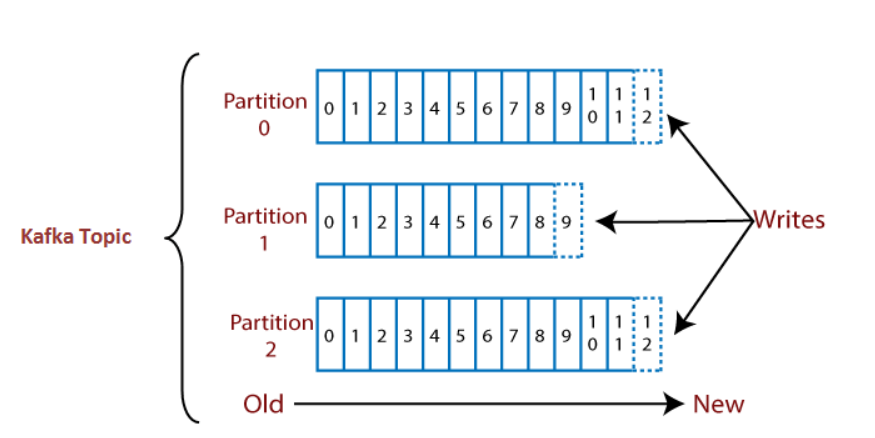
**Connector API:** This API executes the reusable producer and consumer APIs with the existing data systems or applications.

**Topics**

In Kafka, the word topic refers to a category or a common name used to store and publish a particular stream of data. Basically, topics in Kafka are similar to tables in the database, but not containing all constraints. In Kafka, we can create n number of topics as we want. It is identified by its name, which depends on the user's choice. A producer publishes data to the topics, and a consumer reads that data from the topic by subscribing it.

**Partitions**

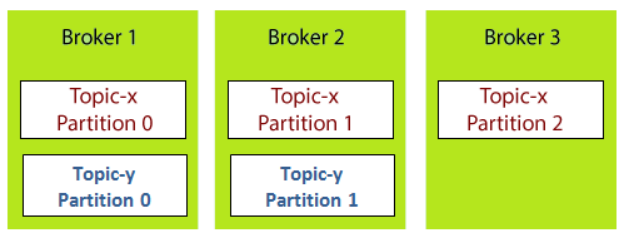
A topic is split into several parts which are known as the partitions of the topic. These partitions are separated in an order. The data content gets stored in the partitions within the topic. Therefore, while creating a topic, we need to specify the number of partitions(the number is arbitrary and can be changed later). Each message gets stored into partitions with an incremental id known as its Offset value. The order of the **offset value** is guaranteed within the partition only and not across the partition. The offsets for a partition are infinite.



Suppose, a topic containing three partitions 0,1 and 2. Each partition has different offset numbers. The data is distributed among each offset in each partition where data in offset 1 of Partition 0 does not have any relation with the data in offset 1 of Partition1. But, data in offset 1of Partition 0 is inter-related with the data contained in offset 2 of Partition0.

**Brokers**

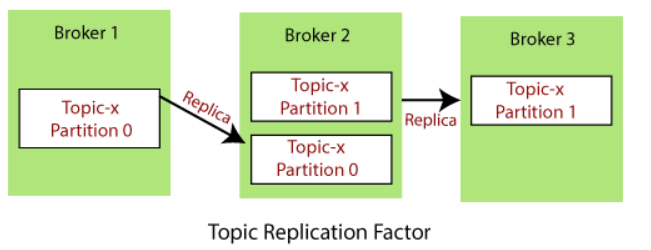
A broker is a container that holds several topics with their multiple partitions. The brokers in the cluster are identified by an integer id only. A producer and a consumer both need to get in touch with broker to publish or subscribe the message.



**Kafka Topic Replication**

In Kafka, each broker contains some sort of data. But, what if the broker or the machine fails down? The data will be lost. Precautionary, Apache Kafka enables a feature of replication to secure data loss even when a broker fails down. To do so, a **replication factor** is created for the topics contained in any particular broker. A replication factor is the number of copies of data over multiple brokers. The replication factor value should be greater than 1 always (between 2 or 3). This helps to store a replica of the data in another broker from where the user can access it.

For example, suppose we have a cluster containing three brokers say Broker 1, Broker 2, and Broker 3. A topic, namely Topic-X is split into Partition 0 and Partition 1 with a replication factor of 2.



**Kafka Producers**

A producer is the one which publishes or writes data to the topics within different partitions. Producers automatically know that, what data should be written to which partition and broker. The user does not require to specify the broker and the partition.

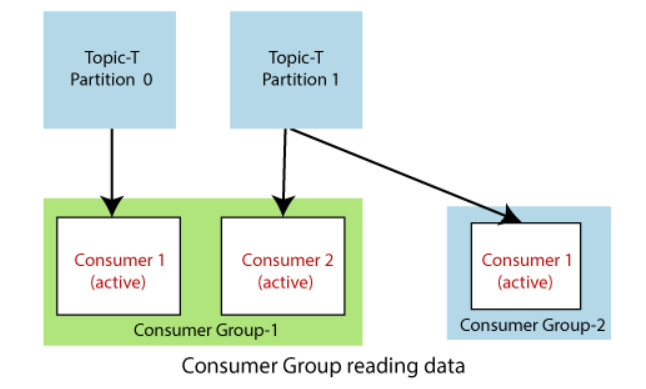
Apache Kafka enables the concept of the key to send the messages in a specific order. The key enables the producer with two choices, i.e., either to send data to each partition (automatically) or send data to a specific partition only. Sending data to some specific partitions is possible with the message keys. If the producers apply key over the data, that data will always be sent to the same partition always. But, if the producer does not apply the key while writing the data, it will be sent in a round-robin manner. This process is called **load balancing**.

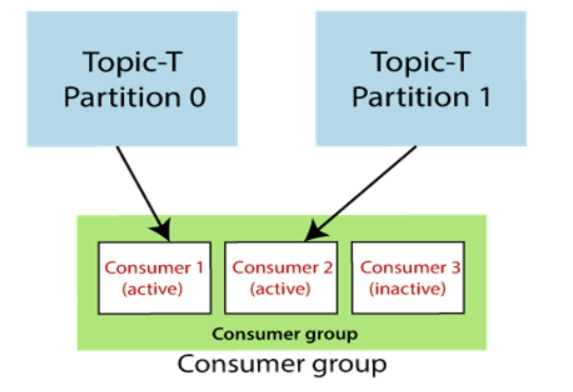
**Consumer**

A consumer is the one that consumes or reads data from the Kafka cluster via a topic. A consumer also knows that from which broker, it should read the data. The consumer reads the data within each partition in an orderly manner. It means that the consumer is not supposed to read data from offset 1 before reading from offset 0. Also, a consumer can easily read data from multiple brokers at the same time.

**Consumer Groups**

A consumer group is a group of multiple consumers which visions to an application basically. Each consumer present in a group reads data directly from the exclusive partitions. In case, the number of consumers are more than the number of partitions, some of the consumers will be in an inactive state. Somehow, if we lose any active consumer within the group then the inactive one can takeover and will come in an active state to read the data.





**Apace Kafka Use Cases**

* **Message Broker** 
  + Apache Kafka is one of the trending technology that is capable to handle a large amount of similar type of messages or data. This capability enables Kafka to give high throughput value. Also, Kafka is a publish-subscribe messaging system that makes users to read and write data more conveniently.
* **Metrics**
  + Apache Kafka is used to monitor operational data by producing centralized feeds of that data. Operational data means monitoring things from technology to security logs to supplier information, and so on.
* **Website Activity Tracking**
  + It is one of the widely used use cases of Kafka. It is because a website activity usually creates a huge amount of data, generating various messages for each particular page view and user's activity.
* **Event Sourcing**
  + Apache Kafka supports the collection of huge amounts of log data. Thus it becomes a crucial component for any Event Management System, which includes Security Information Event Management(SIEM). Handling large amounts of logs data make it an excellent backend for building an application.
* **Kafka Stream Processing**
  + We have various popular frameworks that read data from a topic, process it, and write that processed data over a new topic. This new topic containing the processed data becomes available to users and applications such as Spark Streaming, Storm, etc.

**Advantages**

1. **Low Latency:** Apache Kafka offers low latency value, i.e., upto 10 milliseconds. It is because it decouples the message which lets the consumer to consume that message anytime.
2. **High Throughput:** Due to low latency, Kafka is able to handle more number of messages of high volume and high velocity. Kafka can support thousands of messages in a second. Many companies such as Uber use Kafka to load a high volume of data.
3. **Fault tolerance:** Kafka has an essential feature to provide resistant to node/machine failure within the cluster.
4. **Durability:** Kafka offers the replication feature, which makes data or messages to persist more on the cluster over a disk. This makes it durable.
5. **Reduces the need for multiple integrations:** All the data that a producer writes go through Kafka. Therefore, we just need to create one integration with Kafka, which automatically integrates us with each producing and consuming system.
6. **Easily accessible:** As all our data gets stored in Kafka, it becomes easily accessible to anyone.
7. **Distributed System:** Apache Kafka contains a distributed architecture which makes it scalable. Partitioning and replication are the two capabilities under the distributed system.
8. **Real-Time handling:** Apache Kafka is able to handle real-time data pipeline. Building a real-time data pipeline includes processors, analytics, storage, etc.
9. **Batch approach:** Kafka uses batch-like use cases. It can also work like an ETL tool because of its data persistence capability.
10. **Scalability:** The quality of Kafka to handle large amount of messages simultaneously make it a scalable software product.

**Disadvantages**

1. **Do not have complete set of monitoring tools:** Apache Kafka does not contain a complete set of monitoring as well as managing tools. Thus, new startups or enterprises fear to work with Kafka.
2. **Reduces Performance:** Brokers and consumers reduce the performance of Kafka by compressing and decompressing the data flow. This not only affects its performance but also affects its throughput.
3. **Message tweaking(changing) issues:** The Kafka broker uses system calls to deliver messages to the consumer. In case, the message needs some tweaking, the performance of Kafka gets significantly reduced. So, it works well if the message does not need to change.

**ZooKeeper**

A ZooKeeper is used to store information about the Kafka cluster and details of the consumer clients. It manages brokers by maintaining a list of them. Also, a ZooKeeper is responsible for choosing a leader for the partitions. If any changes like a broker die, new topics, etc., occurs, the ZooKeeper sends notifications to Apache Kafka.

