**Kafka**

**Vedio link :** [**https://www.youtube.com/watch?v=clouToNoxGM&list=PLVz2XdJiJQxwoGuQb8lR-sTV26dz1SoXo**](https://www.youtube.com/watch?v=clouToNoxGM&list=PLVz2XdJiJQxwoGuQb8lR-sTV26dz1SoXo)

[**https://github.com/Java-Techie-jt/kafka-publisher**](https://github.com/Java-Techie-jt/kafka-publisher)

**The Saga design pattern is a way to manage data consistency across microservices in distributed transaction scenarios. A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step.**

**Q. What is Kafka?**

**Ans.** Kafka is an opensource distributed event streaming platform.

**================================================**

**Q: Why use Replication in Kafka?**

**=========================================================**

**Q: can we run producer consumer without zookeeper in kafka?**

Yes, you can run Kafka producers and consumers without ZooKeeper using Kafka Raft Metadata mode (KRaft). KRaft is an experimental feature that's not recommended for production use.

**How it works**

* **In KRaft mode, brokers use a consensus protocol called Raft to communicate with each other.**
* **Raft allows brokers to elect a leader and replicate metadata across the cluster.**
* **The quorum controller stores the state of the cluster using an event-sourced storage model.**
* **This ensures that the internal state machines can be recreated accurately.**

**Benefits**

* **KRaft mode simplifies Kafka management and deployment.**
* **It enhances scalability and fault tolerance.**
* **It can accommodate more partitions per cluster.**

**Considerations**

* **KRaft mode is experimental and should not be used in production.**
* **Some features, like ACLs, security features, transactions, partition reassignment, and JBOD, are not available in KRaft mode.**
* **You need to carefully monitor the broker's health and performance.**

**===============================================================**

**Q. What is ZooKeeper?**

**Ans. ZooKeeper:** ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

**Kafka Broker:** Just a server. It is just an intermediate entity that helps in message exchanges between consumers and producers.

**Kafka cluster:** There can be one or more brokers in kafka cluster.

**Topic:** It specifies the category of messages or classification of messages. Listeners can then just respond to the messages that belong to the topics they are listening to.

**Kafka download Link**

<https://kafka.apache.org/downloads>

https://www.tutorialspoint.com/apache\_kafka/apache\_kafka\_basic\_operations.htm

Start zookeeper.start

D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows>zookeeper-server-start.bat D:\Software\java setup\kafka\_2.12-1.1.0\config\zookeeper.properties

Start Kafka server

D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows>kafka-server-start.bat

D:\Software\java setup\kafka\_2.12-1.1.0\config>server.properties

**Create topics:**

D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 –partitions 1 -topic <topic\_name>

**Create topic syntax:**

--create --zookeeper localhost:2181 --replication-factor 1 –partitions 1 -topic <topic\_name>

**Test:**

**Produce Message in kafka:**

D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows> kafka-console-producer.bat --broker-list localhost:9092 -topic <topic\_name>

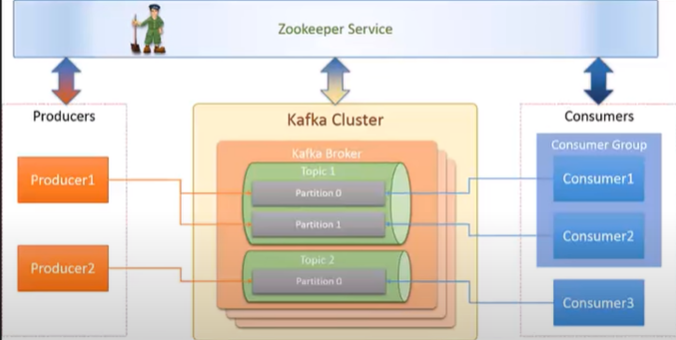
**Consume a message in Kafka:**

D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows> kafka-console-consumer.bat --zookeeper localhost:2181 -topic <topic\_name>

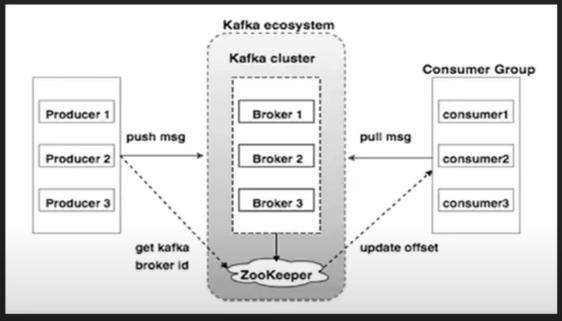
**New for consumer:**

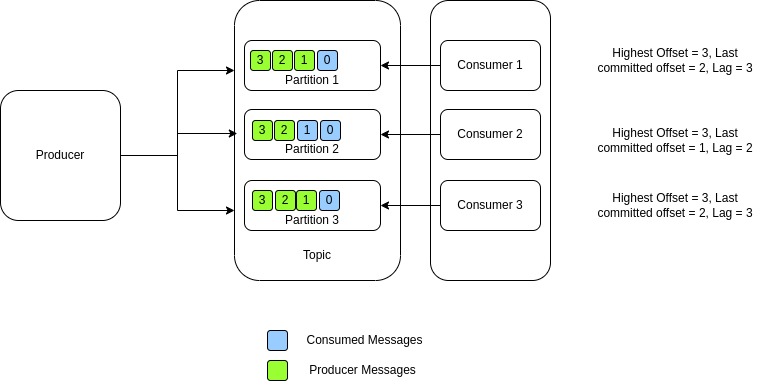
D:\Software\java setup\kafka\_2.12-1.1.0\bin\windows>kafka-console-consumer.bat -bootstrap-server localhost:9092 --topic <topic\_name>

Note: Bootstrap server is the Kafka server, having port number=9092.



**Replication: It is nothing shadow instance of your streaming data.**

****



What is Apache Kafka? https://www.javatpoint.com/apache-kafka

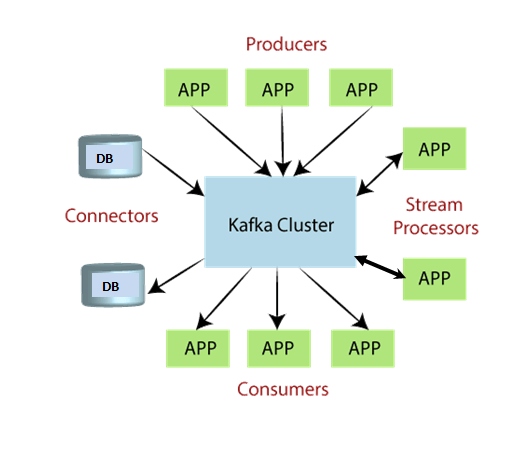
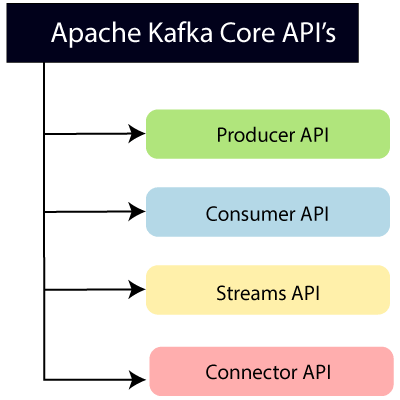
Apache Kafka is an open-source stream-processing software platform that is used to handle real-time data storage. It works as a broker between two parties, i.e., a sender and a receiver. It can handle trillions of data events in a day.

Apache Kafka is a software platform that is based on a distributed streaming process. It is a publish-subscribe messaging system that lets 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. Currently, it is maintained by **Confluent** under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.

## What is the Streaming process?

A streaming process is the processing of data in parallelly connected systems. This process allows different applications to limit the parallel execution of the data, where one record executes without waiting for the output of the previous record. Therefore, a distributed streaming platform enables the user to simplify the task of the streaming process and parallel execution. Therefore, a streaming platform in Kafka has the following key capabilities:

* As soon as the streams of records occur, it processes it.
* It works similar to an enterprise messaging system where it publishes and subscribes streams of records.
* It stores the streams of records in a fault-tolerant durable way.

 ****

**Producer API:** This API allows/permits an application to publish streams of records to one or more topics.

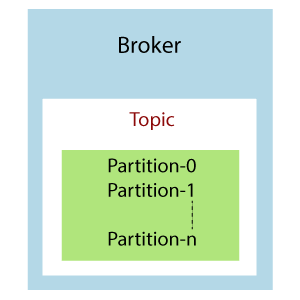
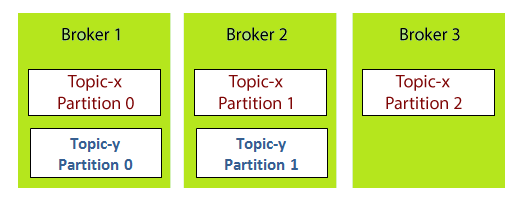
**Consumer API:** This API allows an application to subscribe to one or more topics and process the stream of records produced to them.

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

## Why Apache Kafka

1. Apache Kafka is capable of handling millions of data or messages per second.
2. Apache Kafka works as a mediator between the source system and the target system. Thus, the source system (producer) data is sent to Apache Kafka, where it decouples the data, and the target system (consumer) consumes the data from Kafka.
3. Apache Kafka is having extremely high performance, i.e., it has a really low latency value of less than 10ms which proves it as a well-versed software.
4. Apache Kafka has a resilient architecture that has resolved unusual complications in data sharing.
5. Organizations such as NETFLIX, UBER, Walmart, etc., and over thousands of such firms make use of Apache Kafka.
6. Apache Kafka is able to maintain the fault-tolerance. Fault tolerance means that sometimes a consumer successfully consumes the message that was delivered by the producer. But, the consumer fails to process the message back due to backend database failure, or due to the presence of a bug in the consumer code. In such a situation, the consumer is unable to consume the message again. Consequently, Apache Kafka has resolved the problem by reprocessing the data.
7. Learning Kafka is a good source of income. So, those who wish to raise their income in the future in the IT sector can learn.

Each broker is holding a topic, namely Topic-x with three partitions 0,1 and 2. Remember, all partitions do not belong to one broker only, it is always distributed among each broker (depending on the quantity). Broker 1 and Broker 2 contain another ‘topic-y’ having two partitions 0 and 1. Thus, Broker 3 does not hold any data from ‘topic-y’. It is also concluded that no relationship ever exists between the broker number and the partition number.

# **Kafka Topic Replication**

## When to Use RabbitMQ vs Kafka

To summarize, if you’re looking for a message broker to handle high throughput and provide access to stream history, Kafka is the likely the better choice. If you have complex routing needs and want a built-in GUI to monitor the broker, then RabbitMQ might be best for your application.

Redis is another message broker option. Being in-memory only, Redis is faster than even Kafka. It works best for customers whose destination can receive data far faster than the data can be generated.

**Differences between RabbitMQ and Apache Kakka?**

RabbitMQ employs the smart broker/dumb consumer model. The broker consistently delivers messages to consumers and keeps track of their status. Kafka uses the dumb broker/smart consumer model. Kafka doesn't monitor the messages each user has read.

**Differences between RabbitMQ and Apache Kakka in a tabular form:**

| **Parameter** | **RabbitMQ** | **Kafka** |
| --- | --- | --- |
| Performance | Up to 10K messages per second | Up to 1 million messages per second |
| Data Type | Transactional | Operational |
| Synchronicity of messages | Can be synchronous/asynchronous | Durable message store that can replay messages |
| Topology | Exchange type: Direct, Fan out, Topic, Header-based | Publish/subscribe based |
| Payload Size | No constraints | Default 1MB limit |
| Usage Cases | Simple use cases | Massive data/high throughput cases |
| Data Flow | Distinct bounded data packets in the form of messages | Unbounded continuous data in the form of key-value pairs. |
| Data Unit | Message | Continuous stream |
| Data Tracking | Broker/Publisher keeps track of message status (read/unread) | Broker/Publisher keeps only unread messages; it doesn’t retain sent messages. |
| Broker/Publisher Type | Smart | Dumb |
| Consumer Type | Dumb | Smart |
| Routing messages | Complex routing is possible based on event types | Complex routing is not possible; however, we can subscribe to individual topics. |
| Topology | Exchange queue topology | publish/subscribe topology |
| Message delivery system | Message pushed to specific queues | Pull-based model; consumer pulls messages as required |
| Message management | Prioritize messages | Order/Retain/Guarantee messages |
| Message Retention | Acknowledgment based | Policy-based (e.g., ten days) |
| Event storage structure | Queue | Logs |
| Consumer Queues | Decoupled Consumer queues | Coupled consumer partition/groups |

**Q: Kafka consumer lag?** [**https://www.redpanda.com/guides/kafka-performance-kafka-consumer-lag**](https://www.redpanda.com/guides/kafka-performance-kafka-consumer-lag)

Ans: Kafka consumer lag is the difference between the last message produced by the producer and the offset committed by the consumer group. It represents the consumer processing delay.

|  |  |
| --- | --- |
| Kafka consumer lag concept | Description |
| What is Kafka consumer lag? | Kafka consumer lag is the difference between the last offset stored by the broker and the last committed offset for that partition. |
| Reasons for Kafka consumer lag | Four common reasons for consumer lag are (1) Incoming traffic surges, (2) Data skew in partitions, (3) Slow processing jobs, and (4) Errors in code and pipeline components |
| Monitoring Kafka consumer lag | Teams can monitor Kafka consumer lag with the consumer group script, Burrow (a Kafka monitoring companion), or generic monitoring tools and exporters. |
| Strategies for addressing Kafka consumer lag | The best strategy to address Kafka consumer lag will vary depending on the underlying cause. Processing logic optimizations, partition count modifications, rate limiting, and configuration tuning are four common strategies to address consumer lag. |

