Apache Kafka

Tushar B. Kute,
http://tusharkute.com

Apache Kafka

- Apache Kafka is a software platform which is based on a distributed streaming process.
- 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.
- 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.

Apache Kafka – 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.

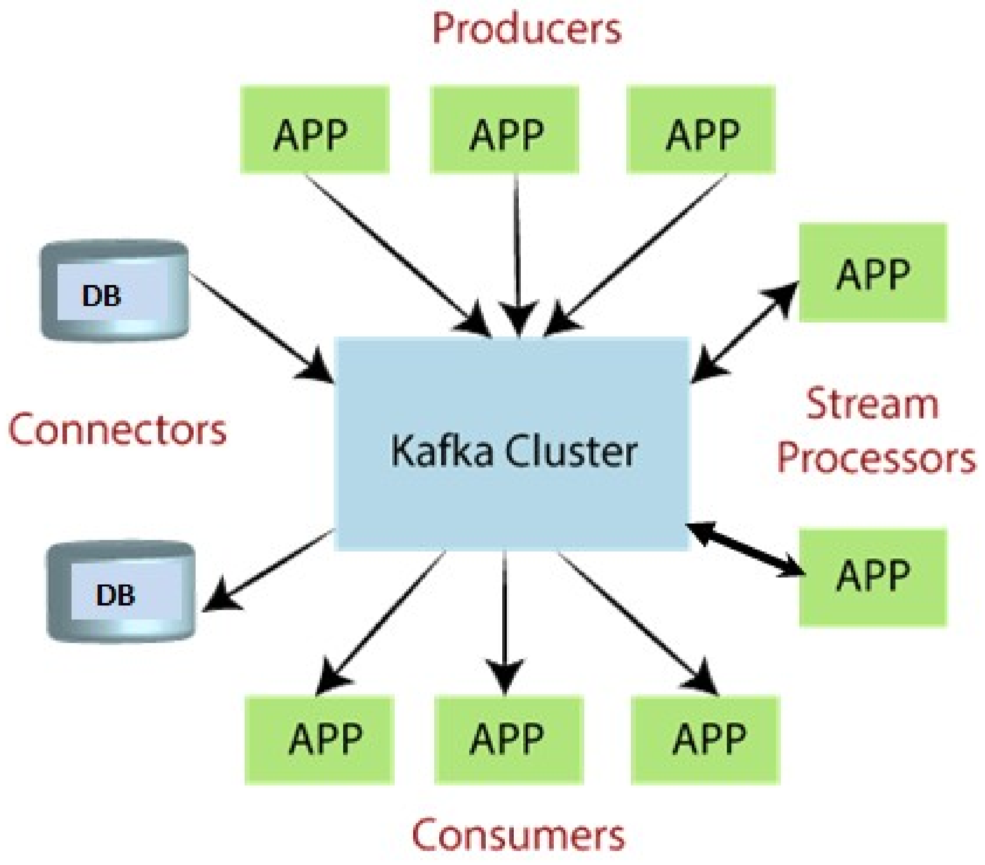
Apache Kafka – Streaming Process

- 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.

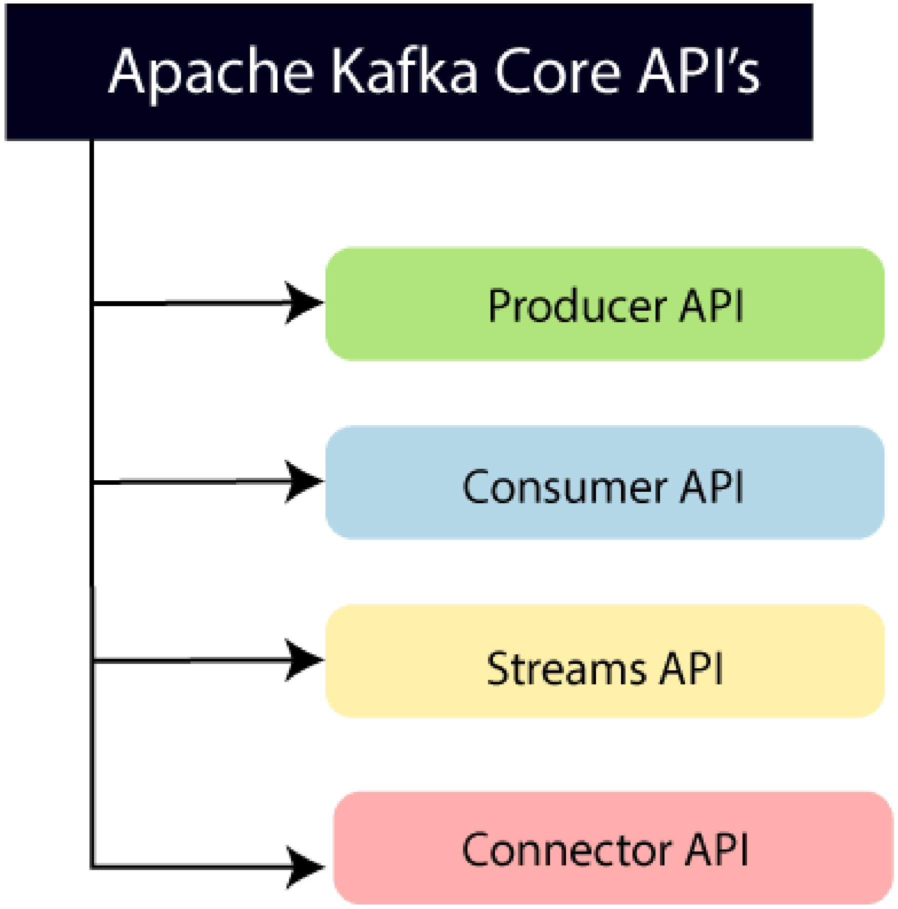
Apache Kafka – Streaming Process

- 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.

Apache Kafka – Streaming Process



Apache Kafka APIs



Kafka APIs

- 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 one or more topics and process the stream of records produced to them.

Kafka APIs

- 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.

Why Kafka?

- Apache Kafka is capable of handling millions of data or messages per second.
- Apache Kafka works as a mediator between the source system and the target system.
- Thus, the source system (producer) data is sent to the Apache Kafka, where it decouples the data, and the target system (consumer) consumes the data from Kafka.
- Apache Kafka is having extremely high performance, i.e., it has really low latency value less than 10ms which proves it as a well-versed software.

Why Kafka?

- Apache Kafka has a resilient architecture which has resolved unusual complications in data sharing.
- Organizations such as NETFLIX, UBER, Walmart, etc. and over thousands of such firms make use of Apache Kafka.
- 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 presence of a bug in the consumer code.

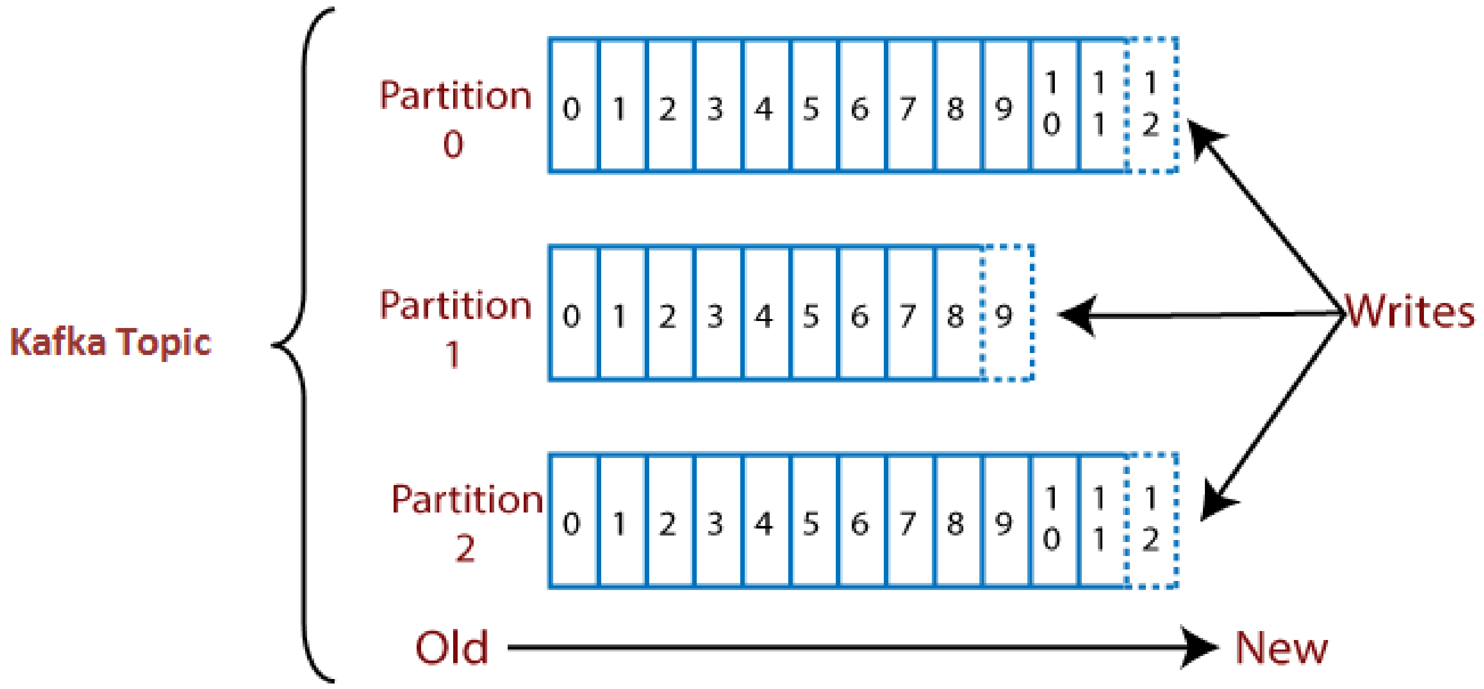
Kafka Topic

- Generally, a topic refers to a particular heading or a name given to some specific inter-related ideas. 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.

Kafka 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.

Kafka Partitions



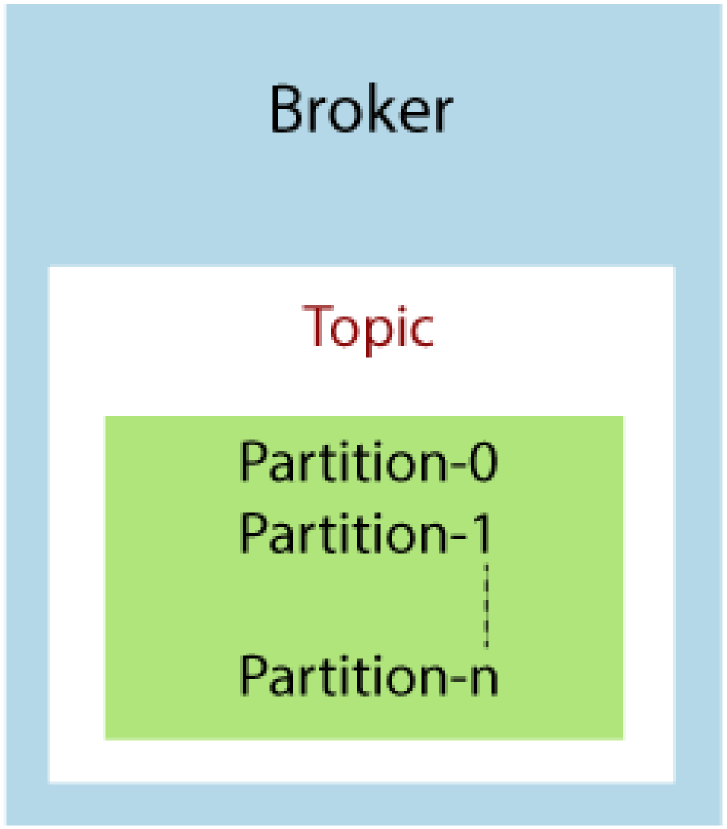
Kafka Partitions

- 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.

Broker

- A Kafka cluster is comprised of one or more servers which are known as brokers or Kafka 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.
- Kafka brokers are also known as Bootstrap brokers because connection with any one broker means connection with the entire cluster.
- Although a broker does not contain whole data, but each broker in the cluster knows about all other brokers, partitions as well as topics.

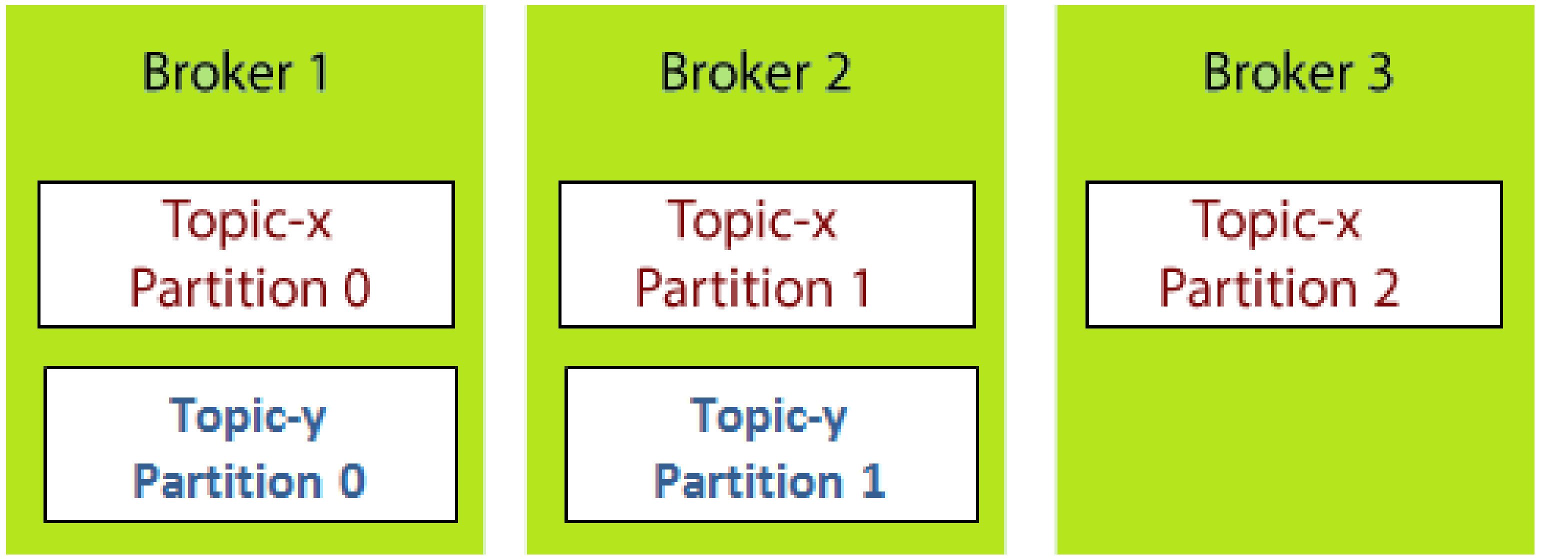
Broker



Broker and Topics

- Suppose, a Kafka cluster consisting of three brokers, namely Broker 1, Broker 2, and Broker 3.
- 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 (depends on the quantity).
- Broker 1 and Broker 2 contains 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.

Broker and Topics



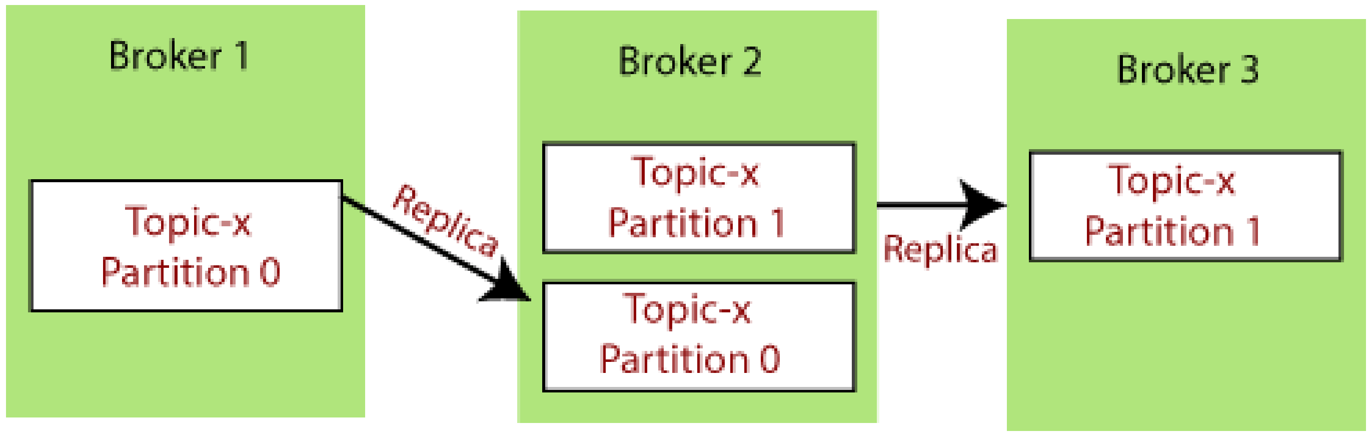
Topic Replication

- Apache Kafka is a distributed software system in the Big Data world.
- Thus, for such a system, there is a requirement to have copies of the stored data. 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.

Topic Replication

- 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.

Topic Replication



Topic Replication Factor

Topic Replication

- Thus, we can see that Partition 0 of Topic-x is having its replicas in Broker 1 and Broker 2.
- Also, Partition1 of Topic-x is having its replication in Broker 2 and Broker 3.
- It is obvious to have confusion when both the actual data and its replicas are present.
- The cluster may get confuse that which broker should serve the client request.

Topic Replication

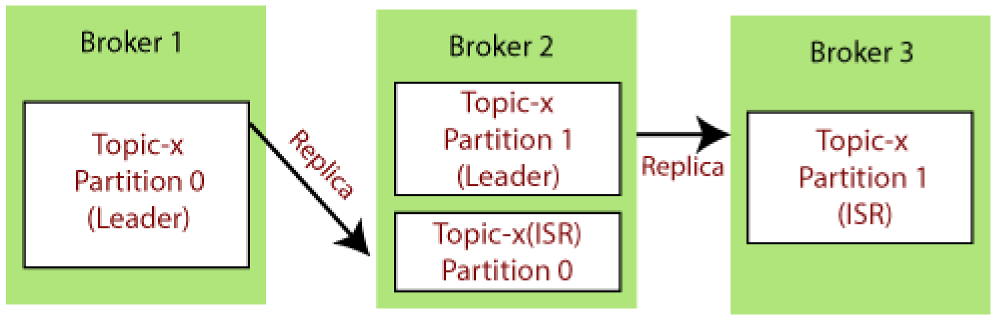
- To remove such confusion, the following task is done by Kafka:
  - It chooses one of the broker's partition as a leader, and the rest of them becomes its followers.
  - The followers(brokers) will be allowed to synchronize the data. But, in the presence of a leader, none of the followers is allowed to serve the client's request.
  - These replicas are known as ISR(in-sync-replica). So, Apache Kafka offers multiple ISR(in-sync-replica) for the data.

Topic Replication

- Therefore, only the leader is allowed to serve the client request.
- The leader handles all the read and writes operations of data for the partitions. The leader and its followers are determined by the zookeeper.
- If the broker holding the leader for the partition fails to serve the data due to any failure, one of its respective ISR replicas will takeover the leadership.
- Afterward, if the previous leader returns back, it tries to acquire its leadership again.

Topic Replication

- Let's see an example to understand the concept of leader and its followers.
- Suppose, a cluster with the following three brokers 1,2, and 3. A topic x is present having two partitions and with replication factor=2.


Broker 1
Topic-x Partition 0 (Leader)
Replica
Broker 2
Topic-x Partition 1 (Leader)
Topic-x(ISR) Partition 0
Replica
Broker 3
Topic-x Partition 1 (ISR)

Topic Replication

- So, to remove the confusion, Partition-0 under Broker 1 is provided with the leadership.
- Thus, it is the leader and Partition 0 under Broker 2 will become its replica or ISR. Similarly, Partition 1 under Broker 2 is the leader and Partition 1 under Broker 3 is its replica or ISR.
- In case, Broker 1 fails to serve, Broker 2 with Partition 0 replica will become the leader.

Kafka Producer

- 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.
- How does the producer write data to the cluster?
- A producer uses following strategie//s to write data to the cluster:
  - Message Keys
  - Acknowledgment

Message Keys

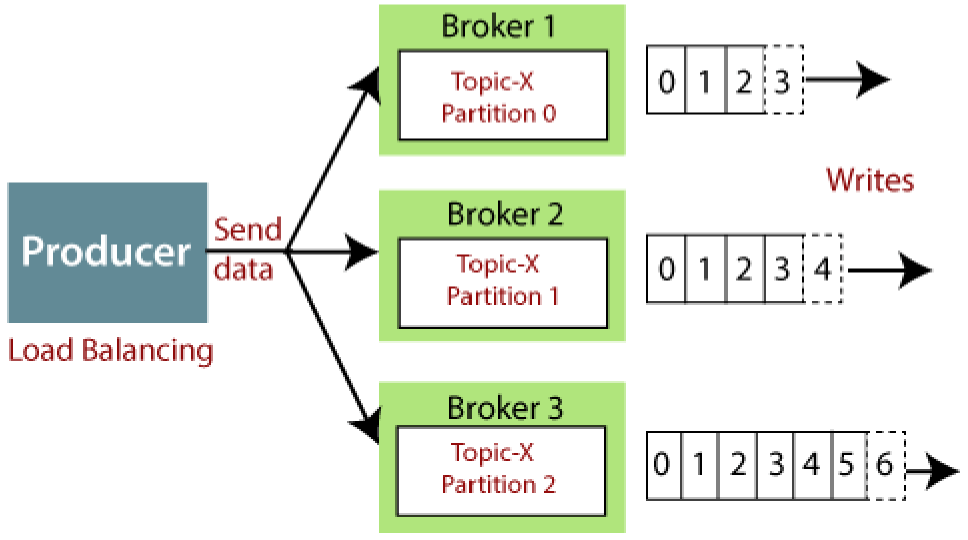
- Apache Kafka enables the concept of the key to send the messages in a specific order.
- The key enables the producer with two choices, i.e., either to send data to each partition (automatically) or send data to a specific partition only.
- Sending data to some specific partitions is possible with the 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.

Message Keys

- In Kafka, load balancing is done when the producer writes data to the Kafka topic without specifying any key, Kafka distributes little-little bit data to each partition.
- Therefore, a message key can be a string, number, or anything as we wish.
- There are two ways to know that the data is sent with or without a key:
  - If the value of key=NULL, it means that the data is sent without a key. Thus, it will be distributed in a round-robin manner (i.e., distributed to each partition).
  - If the value of the key!=NULL, it means the key is attached with the data, and thus all messages will always be delivered to the same partition.

Example:

- Consider a scenario where a producer writes data to the Kafka cluster, and the data is written without specifying the key.
- So, the data gets distributed among each partition of Topic-T under each broker, i.e., Broker 1, Broker2, and Broker 3.

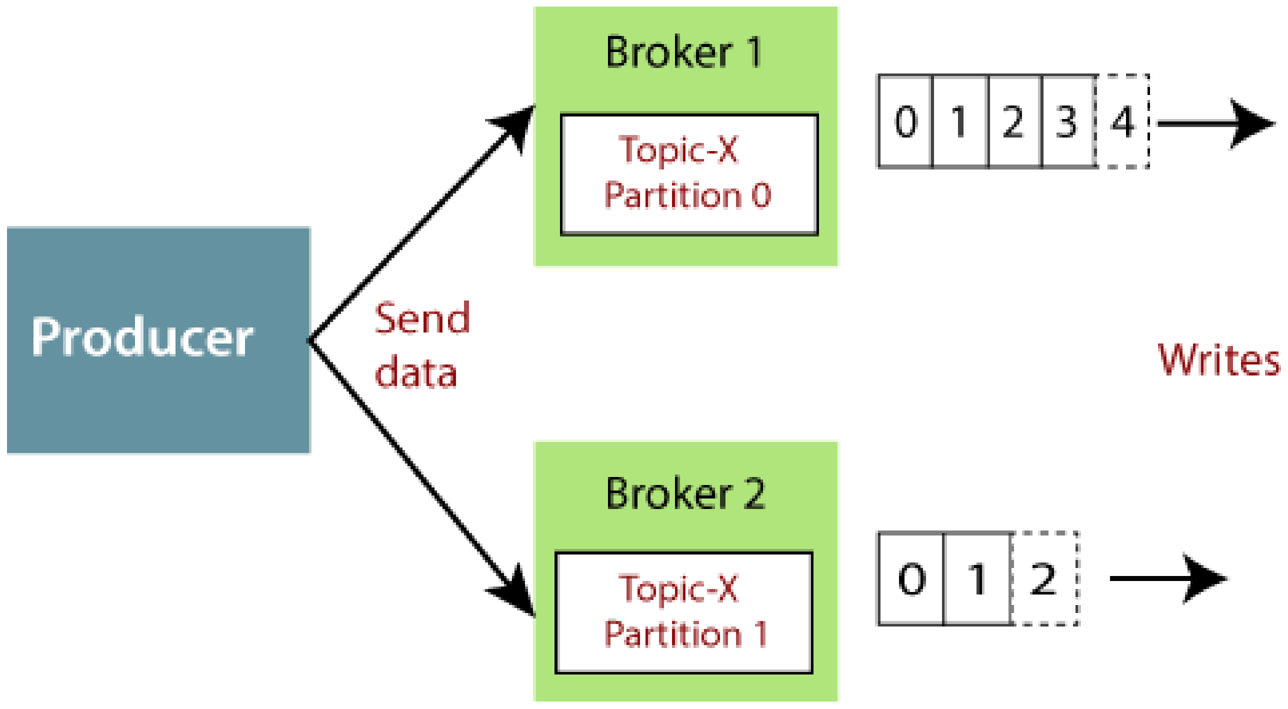


Sending Data Without Key

Example:

- Consider another scenario where a producer specifies a key as Prod_id.
- So, data of Prod_id_1(say) will always be sent to partition 0 under Broker 1, and data of Prod_id_2 will always be in partition 1 under Broker 2.
- Thus, the data will not be distributed to each partition after applying the key (as saw in the above scenario).

Example:



Sending data with key = Prod_id

Acknowledgement

- In order to write data to the Kafka cluster, the producer has another choice of acknowledgment.
- It means the producer can get a confirmation of its data writes by receiving the following acknowledgments:
- acks=0: This means that the producer sends the data to the broker but does not wait for the acknowledgement. This leads to possible data loss because without confirming that the data is successfully sent to the broker or may be the broker is down, it sends another one.

Acknowledgement

- acks=1: This means that the producer will wait for the leader's acknowledgement. The leader asks the broker whether it successfully received the data, and then returns feedback to the producer. In such case, there is limited data loss only.
- acks=all: Here, the acknowledgment is done by both the leader and its followers. When they successfully acknowledge the data, it means the data is successfully received. In this case, there is no data loss.

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.

Kafka Use Cases

- 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.

Kafka Use Cases

- 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.
  - Kafka also ensures that data is successfully sent and received by both parties.

Kafka Use Cases

- 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 Use Cases

- Commit Logs
  - Apache Kafka is used for data replication between the nodes and to restore data on failed nodes.
  - Kafka can also act as a pseudo commit-log. For example, suppose if a user is tracking device data for IoT sensors.
  - He finds an issue with the database that it is not storing all data, then the user can replay the data for replacing the missing or unstored information in the database.

Kafka Use Cases

- Log Aggregation
  - Several organizations make use of Apache Kafka to collect logs from various services and make them available to their multiple customers in a standard format.

Kafka Use Cases

- 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.

Kafka Applications

- LinkedIn
  - In 2010, LinkedIn developed Apache Kafka. As Kafka is a publish-subscriber messaging system, thus various LinkedIn products such as LinkedIn Today and LinkedIn Newsfeed use it for message consumption.

Kafka Applications

- Uber
  - Uber use Kafka as a message bus to connect different parts of the ecosystem. Kafka helps both the passengers and drivers to meet to their correct matches.
  - It collects information from the rider's app as well as from the driver's app, then makes that information available to a variety of downstream consumers.

Kafka Applications

- Twitter
  - Because Kafka has fulfilled the requirements of data replication and durability, twitter has become one of the best applications/users of Apache Kafka.
  - Adopting Kafka led twitter to a vast resource saving, upto 75%, i.e., a good cost reduction.

Kafka Applications

- Netflix
  - Netflix uses Kafka under Keystone Pipeline. A Keystone is a unified collection, event publishing, and routing infrastructure used for stream and batch processing.
  - The Keystone Pipeline uses two sets of Kafka cluster, i.e., Fronting Kafka and Consumer Kafka.
  - Fronting Kafka gets the message from the producers. Consumer Kafka contains topics subsets which are routed by Samza (an Apache framework) for the real-time consumers.
  - Thus, Kafka has maintained the cost by providing a lossless delivery of the data pipeline.

Kafka Applications

- Oracle
  - Apache Kafka has supported Oracle Database as a Kafka Consumer. It has also supported Oracle for publishing events to Kafka.
  - Apache Kafka provides reliable and scalable data streaming. The oracle user can easily retrieve data from a Kafka topic.
  - Oracle developers are now more capable of implementing staged data pipelines through OSB(Oracle Service Bus).

Kafka Applications

- Mozilla
  - Mozilla Firefox is an open-source and free web browser to all. It supports Windows, Linux, macOS, and many other operating systems. Mozilla uses Kafka for backing up the data, i.e., used as a backing data store.
  - Soon, Kafka is going to replace Mozilla's current production system for collecting performance and usage data from the end users for Telemetry, Test Pilot like projects.

Kafka Advantages

- 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.
- 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.
- Fault tolerance: Kafka has an essential feature to provide resistant to node/machine failure within the cluster.

Kafka Advantages

- Durability: Kafka offers the replication feature, which makes data or messages to persist more on the cluster over a disk. This makes it durable.
- 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.
- Easily accessible: As all our data gets stored in Kafka, it becomes easily accessible to anyone.

Kafka Advantages

- Distributed System: Apache Kafka contains a distributed architecture which makes it scalable. Partitioning and replication are the two capabilities under the distributed system.
- Real-Time handling: Apache Kafka is able to handle real-time data pipeline. Building a real-time data pipeline includes processors, analytics, storage, etc.
- Batch approach: Kafka uses batch-like use cases. It can also work like an ETL tool because of its data persistence capability.
- Scalability: The quality of Kafka to handle large amount of messages simultaneously make it a scalable software product.

Kafka Disadvantages

- 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.
- Message tweaking 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.

Kafka Disadvantages

- Do not support wildcard topic selection: Apache Kafka does not support wildcard topic selection. Instead, it matches only the exact topic name. It is because selecting wildcard topics make it incapable to address certain use cases.
- 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.

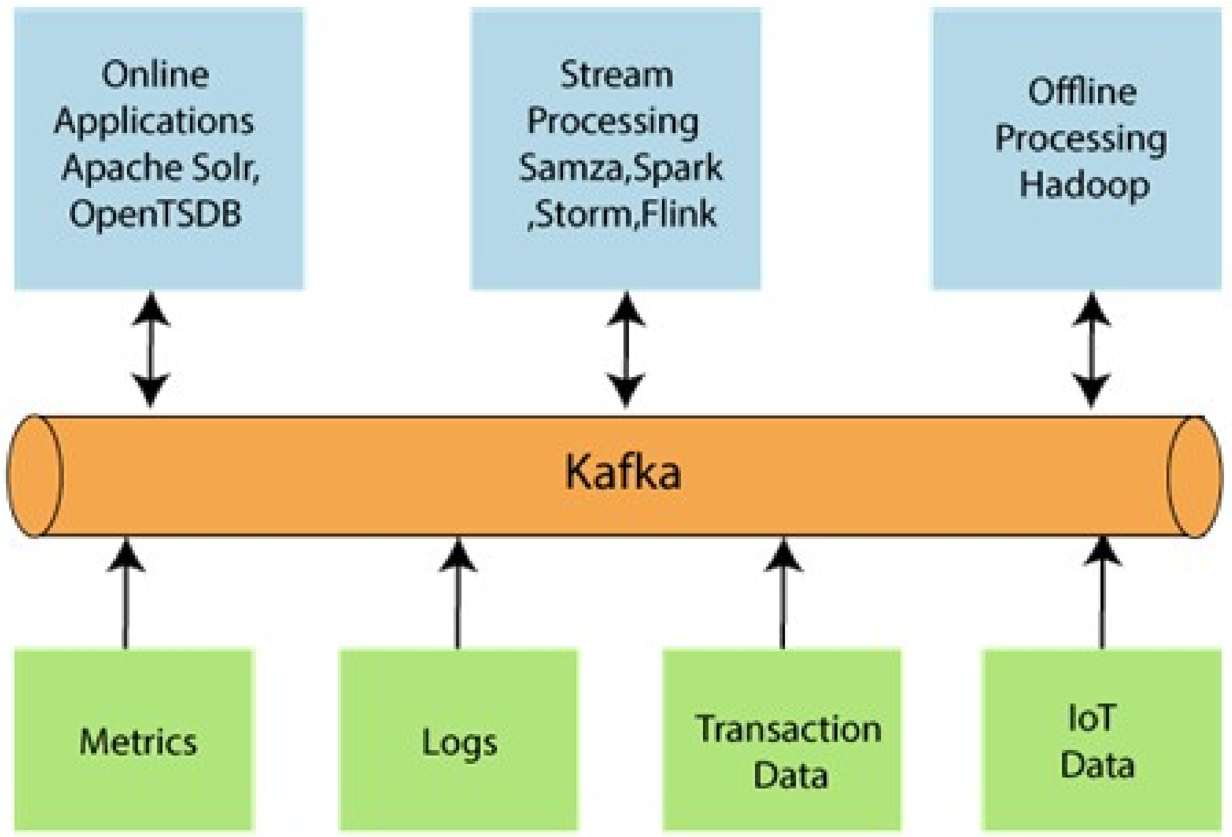
Kafka Disadvantages

- Clumsy Behaviour: Apache Kafka most often behaves a bit clumsy when the number of queues increases in the Kafka Cluster.
- Lack some message paradigms: Certain message paradigms such as point-to-point queues, request/reply, etc. are missing in Kafka for some use cases.

Architecture

- Data Ecosystem: Several applications that use Apache Kafka forms an ecosystem.
- This ecosystem is built for data processing. It takes inputs in the form of applications that create data, and outputs are defined in the form of metrics, reports, etc.
- The below diagram represents a circulatory data ecosystem for Kafka.

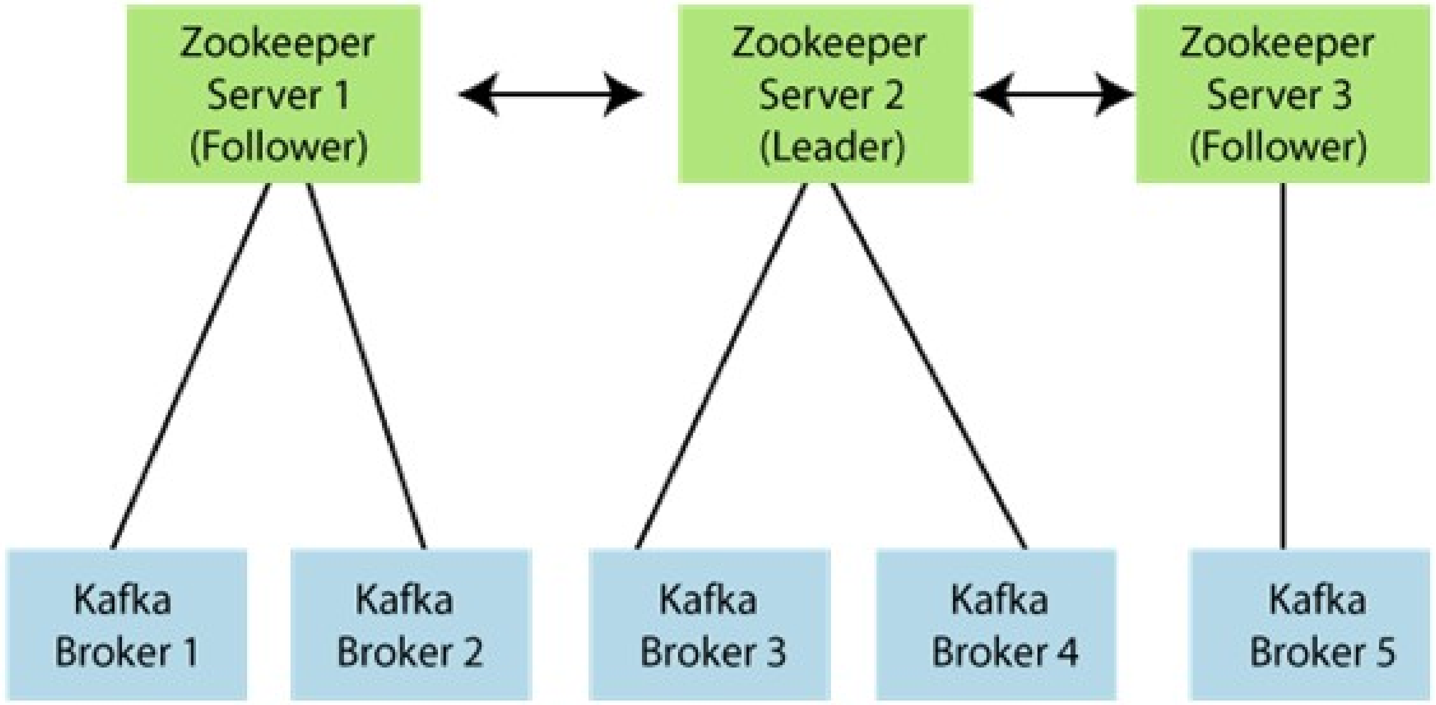
Architecture


Online Applications Apache Solr, OpenTSDB
Stream Processing Samza,Spark ,Storm,Flink
Offline Processing Hadoop
Kafka
Metrics
Logs
Transaction Data
IoT Data

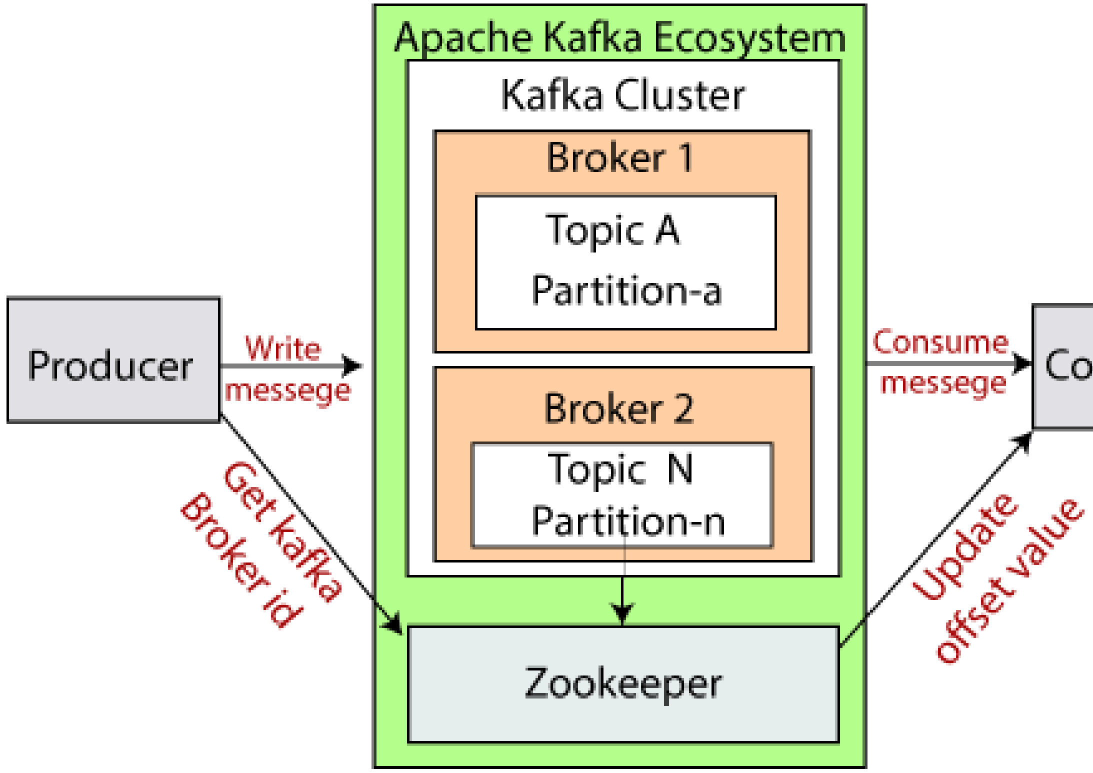
Zookeeper

- 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.
- A ZooKeeper is designed to operate with an odd number of Kafka servers. Zookeeper has a leader server that handles all the writes, and rest of the servers are the followers who handle all the reads.
- However, a user does not directly interact with the Zookeeper, but via brokers. No Kafka server can run without a zookeeper server. It is mandatory to run the zookeeper server.

Zookeeper


Zookeeper Server 1 (Follower)
Zookeeper Server 2 (Leader)
Zookeeper Server 3 (Follower)
Kafka Broker 1
Kafka Broker 2
Kafka Broker 3
Kafka Broker 4
Kafka Broker 5

Summary



Apache Kafka Architecture

Thank you

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com