Introduction to 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.

What is a messaging system

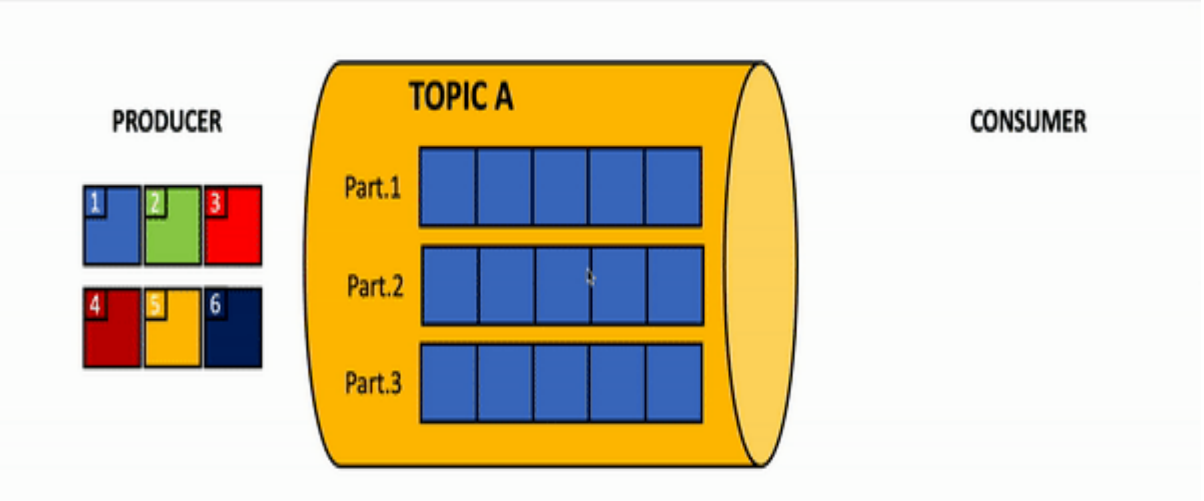
A messaging system is a simple exchange of messages between two or more persons, devices, etc.

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.

What is 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:

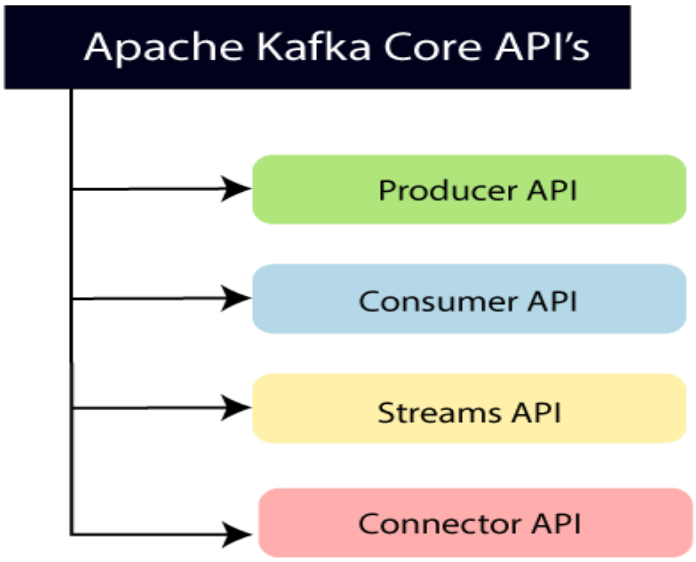


Apache Kafka core APIs :

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.

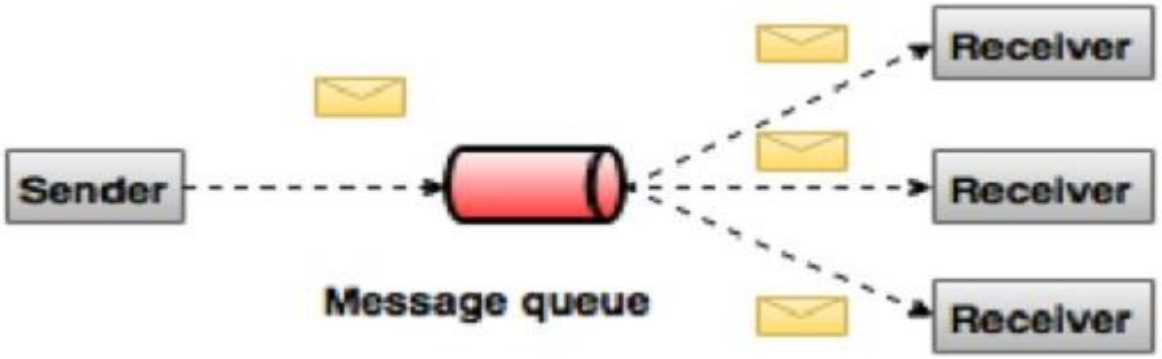
Why Apache Kafka

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 the 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 really low latency value less than 10ms which proves it as a well-versed software.

4. Apache Kafka has a resilient architecture which 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 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.

Publish-Subscribe Messaging System

In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc.



Topics: A stream of messages belonging to a particular category is called a topic. Data is stored in topics. Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.

Partition: Topics may have many partitions, so it can handle an arbitrary amount of data.

Partition offset: Each partitioned message has a unique sequence id called as offset.

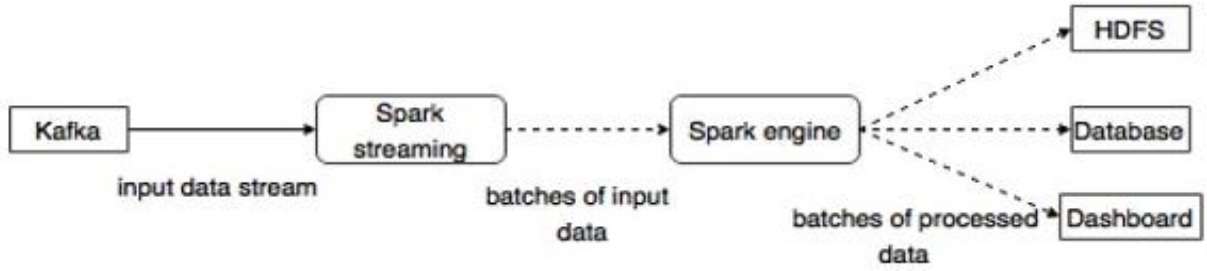
Replicas of partition: Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss.

Apache Kafka - Cluster Architecture

S.No	Components and Description
1	**Broker** Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use Zookeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each bro-ker can handle TB of messages without performance impact. Kafka broker leader election can be done by Zookeeper.
2	**ZooKeeper** ZooKeeper is used for managing and coordinating Kafka broker. ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker then pro-ducer and consumer takes decision and starts coordinating their task with some other broker.
3	**Producers** Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.
4	**Consumers** Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.

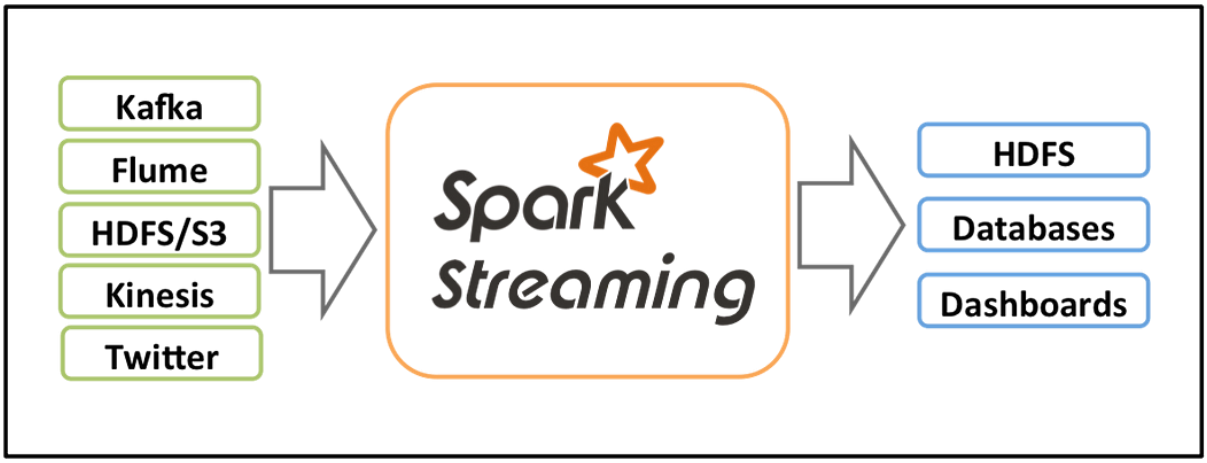
Integration with Spark

Kafka is a potential messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. Once the data is processed, Spark Streaming could be publishing results into yet another Kafka topic or store in HDFS, databases or dashboards. The following diagram depicts the conceptual flow.



WHAT IS SPARK STREAMING?

Apache Spark is an open-source, data processing framework designed for use with real-time data applications. It is relatively easy to scale and is useful for large-scale data processing, making it a popular framework for AI, ML and other big data applications. Spark can integrate with a variety of data sources and supports functional, declarative, and imperative programming styles.



Spark Streaming has 3 major components: input sources, streaming engine, and sink. Input sources generate data like Kafka, Flume, HDFS/S3, etc. Spark Streaming engine processes incoming data from various input sources. Sinks store processed data from Spark Streaming engine like HDFS, relational databases, or NoSQL datastores.

Output modes

After processing the streaming data, Spark needs to store it somewhere on persistent storage. Spark uses various output modes to store the streaming data.

- **Append Mode**: In this mode, Spark will output only newly processed rows since the last trigger.
- **Update Mode**: In this mode, Spark will output only updated rows since the last trigger. If we are not using aggregation on streaming data (meaning previous records can't be updated) then it will behave similarly to append mode.
- **Complete Mode**: In this mode, Spark will output all the rows it has processed so far.

Spark Streaming APIs

Apache Spark, a powerful open-source distributed computing system for big data processing. Spark Streaming enables the processing of real-time streaming data.

Key concept of Spark Streaming API:

1. **DStream (Discretized Stream):**
   - DStream is the basic abstraction provided by Spark Streaming. It represents a continuous stream of data.
   - DStreams can be created from various sources such as Kafka, Flume, HDFS, and more.
2. **Input DStreams:**
   - Input DStreams are DStreams that are created by ingesting live data from various sources.
   - Examples of Input DStreams include KafkaInputDStream, FlumeInputDStream, and others.
3. **Transformations:**
   - Transformations are operations applied to DStreams to produce a new DStream. Examples include map, filter, reduceByKey, and more.
   - These transformations are similar to those in batch processing with RDDs (Resilient Distributed Datasets).
4. **Output Operations:**
   - Output operations are applied to DStreams to write the processed data to an external system or print it. Examples include saveAsTextFiles, print, foreachRDD, etc.
5. **Windowed Operations:**
   - Windowed operations allow you to perform transformations and reduce functions over a sliding window of data in a DStream.
6. **Stateful Operations:**
   - Spark Streaming supports stateful operations, where you can maintain and update a state across batches.
7. **Checkpoints:**
   - Checkpoints are a way to recover from failures. Spark Streaming allows you to periodically checkpoint your DStream operations to HDFS.
8. **Receiver-based Approach:**
   - In the receiver-based approach, Spark Streaming uses a receiver to receive data from sources like Apache Kafka. The receiver stores the data and then Spark processes it.

9. **Direct Approach:**

- In the direct approach, Spark Streaming directly connects to the source (like Kafka) and pulls the data.

10. **Spark Streaming Context:**

- The entry point for Spark Streaming is the StreamingContext, which is the main entry point for Spark Streaming functionality.

11. **Integration with Spark Core:**

- Spark Streaming is tightly integrated with the Spark core API, which allows seamless transition between batch and streaming processing.

12. **Windowing and Sliding Interval:**

- Windowing allows you to perform operations on a specific window of data, and the sliding interval determines how frequently the window is updated.

Building Stream Processing Application with Spark

Step 1: Install and configure Apache Kafka

The first step is to install and configure Apache Kafka. You can download Apache Kafka from the official website and extract the files. Once you have installed Kafka, start the Zookeeper and Kafka services. You can create a topic using the Kafka command line tools.

Step 2: Install and configure Apache Spark

The second step is to install and configure Apache Spark. You can download Apache Spark from the official website and extract the files. Set up the environment variables SPARK_HOME and PATH, and start the Spark service.

Step 3: Write a producer to generate data and send it to Kafka

write a producer to generate data and send it to Kafka. You can use the Kafka Producer API to send data to Kafka. Define the Kafka topic to which the data will be sent.

Step 4: Write a Spark Streaming program to consume data from Kafka

write a Spark Streaming program to consume data from Kafka. You can use the Kafka Receiver API to consume data from Kafka. Define the Kafka topic from which the data will be consumed and define the batch interval, which specifies the frequency at which the data is processed.

Step 5: Process the data using Spark Streaming

Write a process of the data using Spark Streaming. You can use Spark Streaming operations to transform and analyse the data. You can query the data using Spark SQL and the Spark Streaming API to write the processed data to an external system.

Step 6: Monitor the streaming data pipeline

monitor the streaming data pipeline. You can use the Kafka consumer group command to monitor the Kafka topic and the Spark Web UI to monitor the Spark Streaming job.