What is Kafka?

We use Apache Kafka when it comes to enabling communication between producers and consumers using message-based topics. Apache Kafka is a fast, scalable, fault-tolerant, publish-subscribe messaging system. Basically, it designs a platform for high-end new generation distributed applications. Also, it allows a large number of permanent or ad-hoc consumers. One of the best features of Kafka is, it is highly available and resilient to node failures and supports automatic recovery. This feature makes Apache Kafka ideal for communication and integration between components of large-scale data systems in real-world data systems.

Moreover, this technology replaces the conventional message brokers, with the ability to give higher throughput, reliability, and replication like JMS, AMQP and many more. In addition, core abstraction Kafka offers a **Kafka broker**, a **Kafka Producer**, and a **Kafka Consumer**. Kafka broker is a node

on the Kafka cluster, its use is to persist and replicate the data. A Kafka Producer pushes the message into the message container called the Kafka Topic. Whereas a Kafka Consumer pulls the message from the Kafka Topic.

**a. Messaging System in Kafka**

When we transfer data from one application to another, we use the Messaging System. It results as, without worrying about how to share data, applications can focus on data only. On the concept of reliable message queuing, distributed messaging is based. Although, messages are asynchronously queued between client applications and messaging system. There are two types of messaging patterns available, i.e. point to point and publish-subscribe (pub-sub) messaging system. However, most of the messaging patterns follow pub-sub.

• **Point to Point Messaging System(QUEUE)**

Here, messages are persisted in a queue. Although, a particular message can be consumed by a maximum of one consumer

only, even if one or more consumers can consume the messages in the queue. Also, it makes sure that as soon as a consumer reads a message in the queue, it disappears from that queue.

• **Publish-Subscribe Messaging System**

Here, messages are persisted in a topic. In this system, Kafka Consumers can subscribe to one or more topic and consume all the messages in that topic. Moreover, message producers refer publishers and message consumers are subscribers here.

History of Apache Kafka

Previously, LinkedIn was facing the issue of low latency ingestion of huge amount of data from the website into a lambda architecture which could be able to process real-time events. As a solution, Apache Kafka was developed in the year 2010, since none of the solutions was available to deal with this drawback, before.

However, there were technologies available for batch processing, but the deployment details of those technologies were shared with the downstream users. Hence, while it comes to Real-time Processing, those technologies were not enough suitable. Then, in the year 2011 Kafka was made public.

Why Should we use Apache Kafka Cluster?

As we all know, there is an enormous volume of data in **Big Data**. And, when it comes to big data, there are two main challenges. One is to collect the large volume of data, while another one is to analyze the collected data. Hence, in order to overcome those challenges, we need a messaging system. Then Apache Kafka has proved its utility. There are numerous **benefits of Apache Kafka** such as:

• Tracking web activities by storing/sending the events for real-time processes.

• Alerting and reporting the operational metrics.

• Transforming data into the standard format.

• Continuous processing of streaming data to the topics.

Therefore, this technology is giving a tough competition to some of the most popular applications like ActiveMQ, RabbitMQ, AWS etc. because of its wide use.

Kafka Architecture

Four core APIs in this Apache Kafka :

**a. Kafka Producer API**

This Kafka Producer API permits an application to publish a stream of records to one or more Kafka topics.

**b. Kafka Consumer API**

To subscribe to one or more topics and process the stream of records produced to them in an application, we use this Kafka Consumer API.

**c. Kafka Streams API**

In order to act as a stream processor consuming an input stream from one or more topics and producing an output stream to one or more output topics and also effectively transforming the input streams to output streams, this Kafka Streams API gives permission to an application.

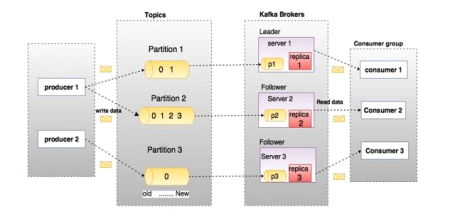
**d. Kafka Connector API**

This Kafka Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Kafka Components

Using the following components, Kafka achieves messaging: a. Kafka Topic

Basically, how Kafka stores and organizes messages across its system and essentially a collection of messages are Topics. In addition, we can replicate and partition Topics. Here, replicate refers to copies and partition refers to the division. Also, visualize them as logs wherein, Kafka stores messages. However, this ability to replicate and partitioning topics is one of the factors that enable Kafka’s fault tolerance and scalability.

b. Kafka Producer

It publishes messages to a Kafka topic.

c. Kafka Consumer

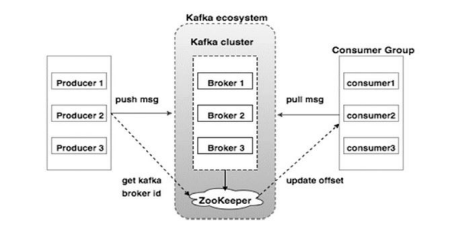
This component subscribes to a topic(s), reads and processes messages from the topic(s).

d. Kafka Broker

Kafka Broker manages the storage of messages in the topic(s). If Kafka has more than one broker, that is what we call a Kafka cluster.

e. Kafka Zookeeper

To offer the brokers with metadata about the processes running in the system and to facilitate health checking and broker leadership election, Kafka uses Kafka **zookeeper**.

**Kafka connect:**

**Apache Kafka** is a distributed streaming platform, and Kafka Connect is a framework for connecting Kafka to other systems such as databases, key-value stores, search indexes, and file systems through the use of Connectors.

Connectors are used to indicate the transformations to be made to the data as it passes between the source and sink systems and to define the source and sink systems.

**How Kafka Connect Works?**

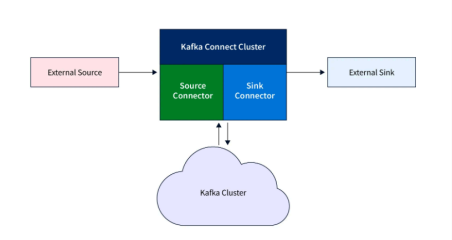
A feature of Apache Kafka called **Kafka Connect** makes it simple to import data from other systems into Kafka or to export Kafka data into other systems. Built on top of Kafka's distributed architecture, Kafka Connect makes use of Kafka's fault-tolerance and scalability features.

**Connectors** are used by Kafka Connect to transfer data between Kafka and other systems. Connectors are modular parts that specify how data should be exported from Kafka to an outside system. Connectors may be set up to function either independently or as a unit in a distributed Kafka Connect cluster.

One or more tasks are created when a connection is launched. The actual data transfer between Kafka and the external system is carried out through tasks. Each job uses a different client library to interface with Kafka and the external system while running in its thread.

**Architecture of Kafka Connect**

To make the data readily available for low-latency stream processing, Kafka Connect may ingest whole databases or gather metrics from all of your application servers into Kafka topics.

The data flow can be explained as follows:

● Various sources are connected to **Kafka Connect Cluster**. Kafka Connect Cluster pulls data from the sources.

● **Kafka Connect Cluster** consists of a set of worker processes that are containers that execute connectors, and tasks automatically coordinate with each other to distribute work and provide scalability and fault tolerance.

● Kafka Connect Cluster pushes data to Kafka Cluster.

● Kafka Cluster persists the data on the broker's local disk or Hadoop.

**Connectors and Tasks**

The Kafka Connect API has three main components: **connectors**, **tasks**, and **workers**. Connectors:

● **Connectors** are responsible for defining how data is read from or written to external systems. There are two types of connectors: **source connectors** and **sink connectors**.

● **Source connectors** are responsible for reading data from external systems and writing it to Kafka, while **sink connectors** are responsible for reading data from Kafka and writing it to external systems.

**Source and Sink Connectors**

● A **Source Connector** specifies the data systems from which data should be collected, which might be a database, real-time data streams, message brokers, or application metrics. ● The Source Connector, once defined, connects to the source data platform and makes the data available to Kafka topics for stream processing.

● A **Sink Connector**, on the other hand, specifies the destination data platform or the endpoint of the data.

● These endpoints might be any variety of data systems, including index engines, other databases, or file storage.

● While sink connections take data from Kafka topics and send it to external systems, source connectors draw data from external systems and write it to Kafka topics.

● Accordingly, sink connections are also unidirectional and can only receive data from Kafka, whereas source connectors are unidirectional and can only send data to Kafka.