Apache Kafka: Core Concepts & Use Cases

These days, considering data as streams is a well-known methodology. By and large, it allows the structuring of data engineering architecture in a more productive manner than when considering data as a state. Yet, to aid the streaming data model, additional technologies need to be utilized. In this regard, one of the most well-known tools utilized today for data streaming is Apache Kafka. In this blog let us talk about the core concepts of Kafka and the best situations for deploying the same.

What is Apache Kafka?

Apache Kafka was initially built at LinkedIn as a messaging queue, yet today is significantly much more than that. It is an incredible asset for working with data streams and can be utilized in several use cases.

Kafka is distributed, which implies that it very well may be scaled up when required. It simply requires adding new nodes or servers to the Kafka cluster.

Apache Kafka is an open-source streaming platform that has been written in Java and Scala, and is consistent with numerous well-known programming languages. It has the capacity to deal with a huge load of data per unit of time and likewise has low a dormancy ratio, which aids in the processing of data in real-time.

Kafka is not the same as conventional message queues and holds the message after it was consumed for a while (by default seven days), while most messaging queues eliminate messages at once, following the clients' affirmation. Additionally, such messaging queues push messages to clients and monitor their load. It chooses the number of messages that ought to be in processing by every single client (there are settings available for this conduct), whereas Kafka upholds fetching messages by clients (pulling). As intended, Kafka was prepared to scale horizontally, by attaching additional nodes. Traditional messaging queues however, hope to scale vertically, by increasing the capacity. These are the main contrasts among Kafka and traditional messaging frameworks.

What is Apache Kafka?

The initial point that each and every individual who works with streaming applications ought to comprehend is the concept, which is a diminutive piece of data. For instance, when a user registers within the system, an event is created. You can likewise ponder on an event like a message with data, which can be processed and saved at a certain place, if at all required. This event is the message wherein the data regarding details such as the user's name, email, password, and so forth can be added. This highlights that Kafka is the platform that works well when it comes to streaming of events.

Events are continually composed by producers. They are called producers since they compose events or data to Kafka. There are numerous sorts of producers. Instances of clients include web servers, parts of applications, whole applications, IoT gadgets, monitoring specialists, and so on. A new user registration event can be produced by the component of the site that is liable for client

registrations. A climate sensor (IoT device) can deliver hourly "climate" events with data regarding temperature, wind speed, humidity, etc. Therefore, the producer is whatever creates information.

Customers are elements that utilize information (events). At the end of the day, they can get data written by producers and utilize this data. There are a ton of instances of data consumers. It is additionally a fact that similar entities (components of applications, entire applications, monitoring frameworks, and so forth) can go about as the two producers and consumers. Everything relies upon the specific architecture and design of the system. Be that as it may, as a rule, entities like databases, data lakes, and data analytics applications go about as data consumers on the grounds that the produced or generated data typically should be stored away some place.

Kafka is the agent between applications that produce data and applications that burn-through data. The Kafka framework is known as the Kafka cluster since it can comprises of numerous elements which are called nodes and the software components that run on a node is called a Broker. Also, this is the reason Kafka is classified as a distributed framework. Information in the Kafka cluster is circulated among more than a few brokers. There are manifold duplicates of the same information in the Kafka cluster, which are called replicas. This makes Kafka more steady, fault-tolerant, and dependable. In the event that an error takes place with one broker, the data won't be lost, and another broker will begin to play out the functions of the broken part.

Producers distribute events to Kafka topics. Consumers can subscribe to subjects to access the data they require. Kafka topics are a permanent log of events or sequences. Every topic can serve information to numerous consumers, which is why producers are in some cases referred to as publishers, and consumers are called subscribers.

Partitions serve to repeat information across brokers. Every Kafka topic is isolated into segments and each partition can be set on a different node.

Best Apache Kafka Use Cases

Here are some common use cases of Apache Kafka.

Real-time data processing

Numerous advanced system frameworks expect data to be prepared and processed when it is available. For instance, in the finance vertical, it is imperative to block fraudulent exchanges the moment they happen. In predictive support, the models ought to continually analyse streams of metrics from the working equipment and trigger alarms immediately post deviations are recognized. IoT devices are frequently pointless without real-time data handling and processing capacity and Kafka can be a helpful source here as it has the ability to send data from producers to data handlers as well as data storages.

Real-time data processing

This is the use case Kafka was initially created for, to be utilized in LinkedIn. Every event that happens in the application can be distributed to the dedicated Kafka topic. User clicks, enlistments, likes, time spent on specific pages by clients, orders, and so forth – this load of events can be sent to Kafka's topics. Then, at that point, different applications or customers can subscribe in to topics and cycle the received data for various purposes including observing, analysis, reports, newsfeeds, user personalization, etc.

Logging and monitoring framework

Apache Kafka can be utilized for logging or monitoring as it is feasible to distribute and publish logs into

Kafka topics. The logs can be put away in a Kafka cluster for quite a while. There, they can be accumulated or processed. It is feasible to construct pipelines that comprise of a few producers where the logs are changed with a particular goal in mind. Eventually, logs can be saved in a conventional log-storage solution.

When it comes to monitoring, assume that you have an extraordinary component of the framework that is devoted to observing and cautioning. This component i.e. monitoring application can peruse data from Kafka topics. This makes Kafka helpful for observing purposes, particularly in case it is real-time monitoring.

When To Not To Use Kafka?

- Kafka is pointless when you need to deal with just a modest quantity of messages each day, generally up to a few thousand. Kafka is intended to adapt to a heavy load and in case you have relatively little data it is recommended to go for traditional messaging queues.

- Kafka is an incredible answer for conveying messages. However, notwithstanding the way that Kafka has a Stream API, it isn't not difficult to perform data changes on the fly. You need to construct an intricate pipeline of associations and interactions between producers and consumers and after wards maintain the whole framework. This requires a ton of work and therefore, you should try not to utilize Kafka for ETL jobs, particularly where real-time processing is required.

- At the point when you need to utilize a straightforward task queue you should utilize proper instru ments. Kafka isn't intended to be a task queue. There are different tools that are better suited for such use cases.

- On the off chance that you need a database, it is recommended to utilize a database and not Kafka. The reason being, Kafka isn't suitable for long term storage. It upholds saving data during a specified retention period, however for the most part, it ought not to be extremely long. Kafka likewise stores excess duplicates of data, which can skyrocket storage expenses. Databases are improved and optimized for the storage of new data. They have additionally flexible query languages and support productive data inserting and retrieving. In the event that relational databases are not what you need for your use case, attempt to search for a non-relational solution instead of Kafka.

Final Thoughts

In this article, we portrayed Apache Kafka and the most appropriate use cases for deploying this tool. It ought to be not difficult to perceive any reason why Kafka is an incredible streaming platform. Kafka is a significant tool in situations requiring real-time data processing and application activity tracking, or monitoring purposes. Simultaneously, Kafka shouldn't be utilized for information changes on the fly, data storing, and when all you need is a basic task queue.

CIGNEX comes with an Apache Kafka specialization and is a certified Confluent Partner, and helps enterprises to build Big Data and IoT applications using Apache Kafka & Confluent for real-time data streaming and analysis. The aim is to provide enterprises with deep insights into their Kafka pipelines through enhanced metrics provided by the Confluent Platform. The experienced Apache Kafka consultants come with proven capabilities in establishing streamlined data pipelines & building streaming applications and provide services such as Apache Kafka Application Development, Apache Kafka Integration, Apache Kafka Implementation, Apache Kafka Support and Managed Services, as well as Professional Services.

The article is authored by **Rebecca Sampson***, Senior Marketer,* **CIGNEX**