A Quick Introduction to Kafka

Chintan MistryAug 31, 2020 · 4 min read ★

Kafka logo: https://kafka.apache.org/

This article will teach you the basics of a fast-growing and reliable streaming platform that makes data processing and storage a breeze!

What is Kafka?

Kafka is a publish/subscribe (pub/sub) messaging system that provides data streaming capabilities while also taking advantage of distributed computing.

What is a pub/sub messaging system? A pub/sub messaging system contains two components that relay some form of data or information between each other. One component publishes data while the other component subscribes to the publisher to receive the published data.

Kafka follows this pattern with its own set of components and features.

Producers

The first component in a pub/sub messaging system is the publisher which is referred to as a *Producer* in Kafka. The producer is a data source that publishes or produces a message into Kafka. One of the great features of Kafka is that it is data type independent. This means that Kafka does not care about what type of data is being produced, whether it's the GPS signal of a car, application metrics from front-end servers, or even images!

Consumers

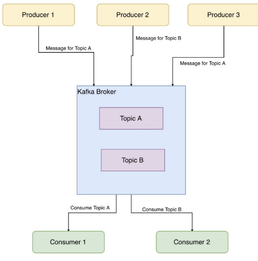
The second component in a puh/sub messaging system is the subscriber, which is referred to as a *Consumer* in Kafka. The consumer can subscribe or listen to a data stream and consume messages from that stream while having no relationship or knowledge about the producers.

Consumers can subscribe to multiple streams of data regardless of the type of data being consumed. In other words, you can have a single application that takes in data from as many different sources as you'd like. Kafka makes it easy to access the data you need while leaving the processing steps entirely in your control.

. . .

High-level Architecture

Now that you know where messages come from (producers) and how messages can be retrieved (consumers), let's discuss what happens in between.

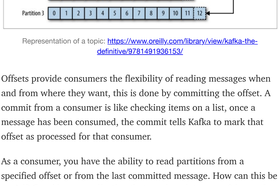


Simple Kafka flow

The above image illustrates a simple Kafka flow with three producers and two consumers. Each producer must specify a destination for the message and each consumer must specify from where it needs to consume. This middle ground between the producer and consumer, where the Kafka message is stored, is called a *Topic*.

Topics, Partitions, and Offsets

Topics can be thought of as a table in a database, where producers can write to and where consumers can read from. Each topic contains *Partitions* which are essentially logs that *commit* and append Kafka messages as they arrive. To identify messages, partitions use an auto-incrementing integer called an *Offset*, which is unique within partitions.



Representation of a topic: https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/

Offsets provide consumers the flexibility of reading messages when and from where they want, this is done by committing the offset. A commit from a consumer is like checking items on a list, once a message has been consumed, the commit tells Kafka to mark that offset as processed for that consumer.

As a consumer, you have the ability to read partitions from a specified offset or from the last committed message. How can this be useful? Consider an application that receives data every 2 hours. In this case, having the application continuously running and waiting for messages can be very expensive. By reading from the last committed message, you could have the application go live every 8 hours to simply consume all new messages and commit the offset of the latest message. This can reduce costs and usage of resources significantly.

Brokers and Clusters

Now, you know about producers, consumers, and how Kafka messages flow within Kafka, but one of the most important components remain, the Kafka *Broker*. The broker is what ties the whole system together; it is the Kafka server that is responsible for dealing with all communications involving producers, consumers, and even other brokers. Producers rely on the broker to correctly accept and store the incoming Kafka message to its appropriate topic. Consumers rely on the broker to handle their fetch and commit requests while consuming from topics.

A group of brokers is called a Kafka *Cluster*. One of the biggest perks of using Kafka is its use of distributed computing. A distributed system shares its workload among many other computers called nodes. These nodes all work together and communicate to complete the work rather than having all the work assigned to a single node. When we have multiple Kafka brokers and clusters dealing with large amounts of data, distributed computing saves resources and increases overall performance; making Kafka a desirable choice for big data applications.

Summary

Kafka is a great tool when it comes to handling and processing data especially with big data applications. It's a reliable platform that provides low-latency and high throughput with its data-streaming capabilities and gives an ample amount of helpful features and services to make your application better.

This has been a high-level overview of what Kafka is and how it works, but there is still much more to Kafka that makes it the great tool that it is. I recommend reading Kafka: The Definitive Guide, it provides in great detail the structure of Kafka and easy to follow steps on its use.