Apache Kafka

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Apache Kafka was originated at LinkedIn and later became an open sourced Apache project in 2011, then First-class Apache project in 2012. Kafka is written in Scala and Java. Apache Kafka is publish-subscribe based fault tolerant messaging system. It is fast, scalable and distributed by design.

This tutorial will explore the principles of Kafka, installation, operations and then it will walk you through with the deployment of Kafka cluster. Finally, we will conclude with real-time applications and integration with Big Data Technologies.

Audience

This tutorial has been prepared for professionals aspiring to make a career in Big Data Analytics using Apache Kafka messaging system. It will give you enough understanding on how to use Kafka clusters.

Prerequisites

Before proceeding with this tutorial, you must have a good understanding of Java, Scala, Distributed messaging system, and Linux environment.

Copyright and Disclaimer

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

1. Kafka – Introduction

In Big Data, an enormous volume of data is used. Regarding data, we have two main challenges. The first challenge is how to collect large volume of data and the second challenge is to analyze the collected data. To overcome those challenges, you must need a messaging system.

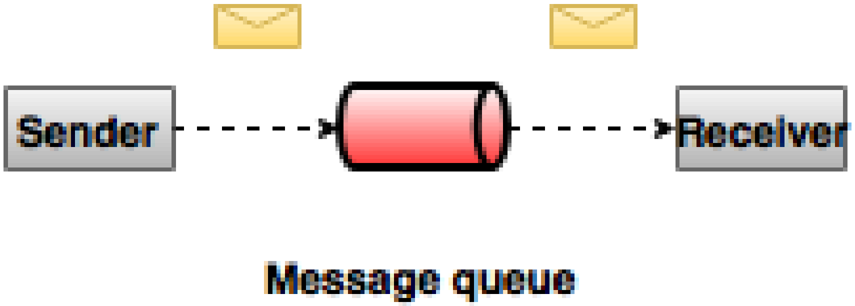
Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.

What is a Messaging System?

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system. Two types of messaging patterns are available – one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow **pub-sub**.

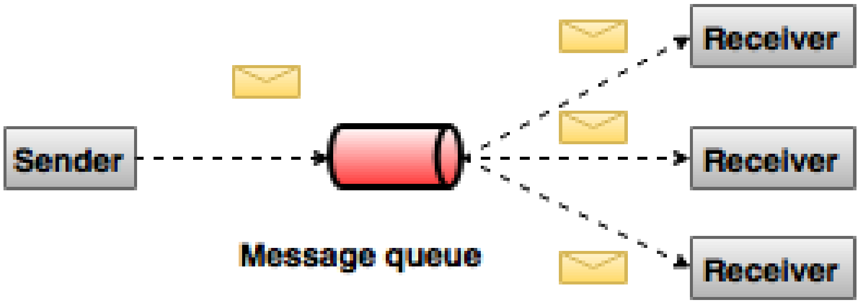
Point to Point Messaging System

In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.



Publish-Subscribe Messaging System

In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. 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., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.



What is Kafka?

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

Benefits

Following are a few benefits of Kafka:

- **Reliability** - Kafka is distributed, partitioned, replicated and fault tolerance.
- **Scalability** - Kafka messaging system scales easily without down time.
- **Durability** - Kafka uses "Distributed commit log" which means messages persists on disk as fast as possible, hence it is durable.
- **Performance** - Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

Use Cases

Kafka can be used in many Use Cases. Some of them are listed below:

- **Metrics -** Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.

- **Log Aggregation Solution -** Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.

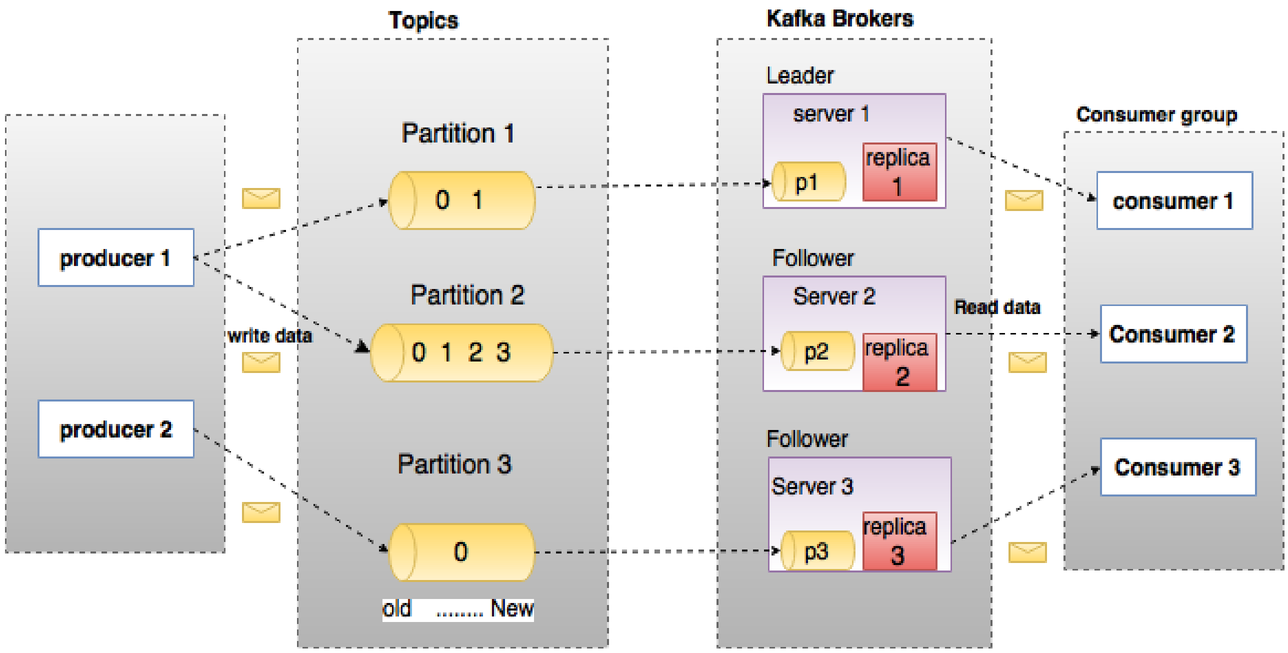
- **Stream Processing -** Popular frameworks such as Storm and Spark Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications. Kafka's strong durability is also very useful in the context of stream processing.

Need for Kafka

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It has the ability to handle a large number of diverse consumers. Kafka is very fast, performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

2. Kafka – Fundamentals

Before moving deep into the Kafka, you must aware of the main terminologies such as topics, brokers, producers and consumers. The following diagram illustrates the main terminologies and the table describes the diagram components in detail.



In the above diagram, a topic is configured into three partitions. Partition 1 has two offset factors 0 and 1. Partition 2 has four offset factors 0, 1, 2, and 3. Partition 3 has one offset factor 0. The id of the replica is same as the id of the server that hosts it.

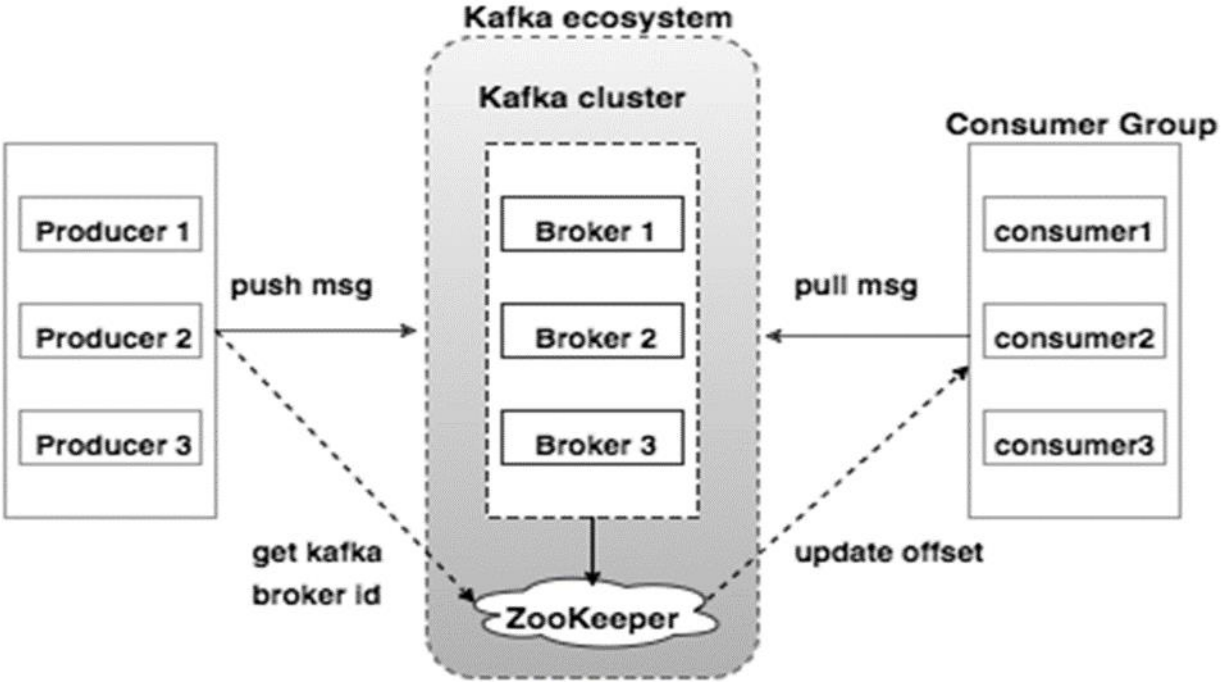
Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations. To balance a load in cluster, each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

Components	Description
Topics	A stream of messages belonging to a particular category is called a topic. Data is stored in topics.
Partition	Topics are split into partitions. For each topic, Kafka keeps a minimum 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. 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.
Brokers	i) Brokers are simple system responsible for maintaining the published data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition. ii) Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic. iii) Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distribution among the broker.
Kafka Cluster	Kafka's having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.

Producers	Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.
Consumers	Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.
Leader	"Leader" is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.
Follower	Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and updates its own data store.

3. Kafka – Cluster Architecture

Take a look at the following illustration. It shows the cluster diagram of Kafka.


Kafka ecosystem
Kafka cluster
Consumer Group
Producer 1
Producer 2
Producer 3
push msg
Broker 1
Broker 2
Broker 3
pull msg
consumer1
consumer2
consumer3
get kafka broker id
ZooKeeper
update offset