kafka

what is a kafka ?

Apache Kafka is a distributed messaging service.

Apache Kafka was developed by LinkedIn and joined the Apache Software Foundation as an open-source project in 2011. Kafka is written in Scala and Java.

What is a messaging system ?

A messaging system is responsible for transferring data from one program to another.

Distributed messaging is built on a concept known as reliable message queuing. In this model, messages are consistently placed in queues that act as intermediaries between client applications and the messaging system. There are two primary messaging patterns: point-to-point and publish-subscribe (pub-sub) systems.

What is a point-to-point message system?

Messages are placed in queues, where one or more consumers can consume them. However, each specific message can only be consumed by a single consumer. Once a consumer reads a message from the queue, the message can be removed from it. A practical example of such systems is order processing, where each order is processed sequentially by a single process, while multiple order processes can work simultaneously to handle different orders.

What is a Publish-subscribe message system ?

In this system, messages are organized based on topics, unlike the point-to-point system. In a topic-based system, messages remain in one or more topics, and consumers can access all messages within the topics they are subscribed to.

Advantages of Apache Kafka

1.Fault Tolerance:

Kafka supports fault tolerance by replicating and partitioning data across a distributed system.

2.Scalability:

Kafka’s messaging system is fully scalable and operates with zero downtime.

3.Durability:

Kafka uses a distributed commit log to store messages on disk, ensuring data durability and persistence.

4.High Performance:

Kafka excels in both publishing and subscribing to messages, maintaining high efficiency and performance.

It operates at high speed, preventing downtime and data loss.

Use Cases of Kafka

1.Kafka Metrics:

Kafka is often used for operational data monitoring, collecting data from distributed applications and centralizing it into a single source of operational data.

2.Kafka Log Aggregation Solution:

Kafka can aggregate logs from various services across an organization and present them in a standardized format.

3.Stream Processing:

Frameworks such as Spark Streaming and Storm read data from a topic, process it, and write the results to a new topic, making it available for users and applications.

Key Features of Kafka

1.Unified Platform:

Kafka is a unified and integrated platform used for managing real-time data feeds.

2.Low Latency:

Kafka supports minimal delay in message delivery.

3.Fault Tolerance:

Kafka ensures data integrity even in the event of server failures.

4.Support for Multiple Consumers:

Kafka can handle a large number of consumers efficiently.

5.High Speed:

Kafka is extremely fast, capable of handling 2 million writes per second.

6.Disk Storage:

In Kafka, all data is first written to a cache, then transferred to the network socket, and finally persisted to disk. This process enhances both durability and performance.

Kafka terminology

1.Topic

A Topic is a logical category or a name for a stream of messages. Data in Kafka is stored in Topics.
Topics are divided into Partitions, ensuring scalability and parallel processing.
Each Topic in Kafka has at least one Partition, and messages within a Partition are ordered and immutable.
A Partition is essentially a set of segmented files of equal size.

2.Partition

A Partition is a subset of a Topic that enables Kafka to handle large amounts of data efficiently.
Each message in a Partition has a unique Offset, which acts as an identifier indicating the message's position.
Create a partition for the data disk (optional)
A. `fdisk <device-name>`

```
[root@localhost ~]# fdisk /dev/sdc

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write them.
```

```
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x9379b641.

Command (m for help): g
Created a new GPT disklabel (GUID: 83B4445D-DE99-3B4A-8870-C9A4C73AAFF4).

Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-20971486, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-20971486, default 20971486):

Created a new partition 1 of type 'Linux filesystem' and of size 10 GiB.

Command (m for help): p
Disk /dev/sdc: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 83B4445D-DE99-3B4A-8870-C9A4C73AAFF4

Device     Start      End  Sectors Size Type
/dev/sdc1   2048 20971486 20969439  10G Linux filesystem

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

B.Creating a file system

```
mkfs.xfs /dev/sdc1
```

C.create a data directory and mount the data disk to it

```
mkdir /data
mount /dev/sdc1 /data
```

D.Disk persistence

```
vim /etc/fstab
```

```
/dev/sdc1  /data xfs defaults 0 0
```

E.Create Kafka directory

```
mkdir /data/kafka
chown -R cp-kafka:confluent /data/kafka
```

3.Replicas of a Partition

A Replica is a copy of a Partition stored on different Kafka brokers to provide fault tolerance.
Replicas are not involved in the read/write process; they serve as backups to prevent data loss.

4.Broker

A Broker is a Kafka server that stores and manages published data.

Each Broker can have multiple Partitions for a given Topic.
Kafka follows different scenarios based on the number of Brokers and Partitions:
If there are N Partitions and N Brokers, each Broker holds one Partition.
If there are N Partitions and more than N Brokers (N + M), M Brokers will not have any Partitions for that Topic.
If there are N Partitions and fewer than N Brokers (N - M), some Brokers will store multiple Partitions, leading to uneven load distribution (which is not recommended).

5.Kafka Cluster

A Kafka Cluster consists of multiple Brokers working together.

A Kafka Cluster can be expanded without downtime.
Clusters manage data replication and fault tolerance.

Producers send messages to Kafka Topics, and Brokers distribute them to appropriate Partitions.

6.Producer

A Producer is an application that publishes messages to Kafka Topics.

Producers send data to Kafka Brokers, which then assign messages to Partitions.
A Producer can either let Kafka decide the Partition or specify it manually.

7.Consumer

A Consumer is an application that reads data from Kafka Topics.

Consumers subscribe to one or more Topics and receive messages from Brokers.

8.Leader

A Leader is the main node responsible for handling read and write requests for a Partition.
Each Partition has one Leader Broker.

9.Follower

A Follower is a node that replicates data from the Leader.
If the Leader fails, a Follower automatically becomes the new Leader.
Followers act as Consumers, fetching data from the Leader and updating their own data store.

10.Cluster Architecture

A Kafka Cluster typically consists of multiple Brokers for load balancing.

Broker in the cluster :

Kafka Brokers are stateless, meaning they rely on Zookeeper to manage cluster metadata.

A single Broker can process thousands of messages per second.
Leader selection among Kafka Brokers is handled by Zookeeper.

Zookeeper in Kafka

Zookeeper is used to manage and coordinate Kafka Brokers. It plays a crucial role in detecting Producers, Consumers, and the presence or failure of Brokers in the Kafka system.

Zookeeper sends notifications about the availability or failure of Brokers.
Based on these notifications, Producers and Consumers adjust their behavior by switching to an available Broker.

What is a Producer?

Producers send data to Kafka Brokers.

When a new Broker starts, all Producers automatically detect and start sending messages to it.
Kafka Producers do not wait for message acknowledgment from Brokers; they send messages rapidly, and Brokers handle message storage.

What is a Consumer?

Since Kafka Brokers are stateless, Consumers must track the number of messages they have processed using Partition offsets.

If a Consumer acknowledges an offset, it indicates that all previous messages have been processed.
The Consumer then indirectly requests a byte buffer from the Broker for consumption.
Consumers can navigate to any position in the Partition using the offset value.
Zookeeper notifies Consumers of the current offset.

Kafka Workflow

Kafka consists of multiple Topics, each divided into one or more Partitions, forming a sequential log of messages.

Each message has a unique identifier called an offset.
All data in a Kafka cluster is stored in separate Partitions.
Incoming messages are appended to the end of a Partition and read by Consumers.
Workflow of Pub-Sub Messaging

The Pub-Sub messaging workflow follows these steps:

Producers send messages to Topics at regular intervals.
Kafka Brokers store all messages in a configured location for the corresponding Topic and distribute them evenly across Partitions.
If a Producer sends multiple messages and there are two Partitions, Kafka stores one message in the first Partition and the next in the second.
Consumers subscribe to specific Topics.
When a Consumer subscribes to a Topic, Kafka retrieves the current offset for that Consumer and stores it in Zookeeper.
The Consumer periodically requests new messages from Kafka.
Kafka receives messages from the Producer and forwards them to the Consumer.
The Consumer processes the messages.
Once processing is complete, the Consumer sends an acknowledgment to the Broker.
Kafka updates the offset in Zookeeper.
Since offsets are stored in Zookeeper, the Consumer can correctly retrieve subsequent messages.
The above process continues until the Consumer stops requesting messages.
Consumers can rewind to previous offsets to reread messages if needed.

Workflow of Queue Messaging (Consumer Groups)

Unlike a single Consumer, Consumer Groups allow multiple Consumers to work together.
A Consumer Group is identified by a unique Group ID.
Consumers within the same group share the messages from a Topic.

Steps of Queue Messaging:

A Producer sends messages to a Topic at regular intervals.
Kafka stores the messages in a configured location, similar to the Pub-Sub model.
A Consumer subscribes to a Topic, for example, Topic_01, with Group ID: Group_1.

Kafka follows the Pub-Sub model until another Consumer with the same Group ID (e.g., Group_1) subscribes to the same Topic.
Kafka then distributes messages among all Consumers in the group.
Distribution continues until all Partitions are assigned.
If the number of Consumers exceeds the number of Partitions, the extra Consumers will not receive any messages.
In this scenario, each Consumer must have at least one Partition.
When all Partitions are assigned, new Consumers must wait.
This behavior is known as a Consumer Group.

What is Zookeeper?

Zookeeper is a distributed coordination service and a core dependency of Apache Kafka.
It acts as an interface for managing Kafka Brokers and Consumers.
Kafka stores metadata in Zookeeper, including information about Topics, Brokers, Consumers, and Offsets.
Zookeeper also manages large clusters of hosts.
Managing a distributed system is complex, but Zookeeper simplifies it with its architecture.
It allows developers to focus on application logic without worrying about distributed system complexities.

install kafka

Kafka installation methods

1.Apache Kafka (Download from the official website)
2.Confluent Kafka
3.Docker (Using Kafka Docker image)
4.Kafka as a package (APT/YUM for Linux)
5.Homebrew (macOS)
6.Kubernetes (Kafka on Kubernetes)
7.Cloud services (AWS MSK, Confluent Cloud, etc.)

Recommended method for installing Kafka

confluent

1.update & upgrade

```
sudo dnf update -y
```

2.essential items for installing kafka

```
sudo dnf install -y java-17-openjdk java-17-openjdk-devel
```

3.create dir kafka

```
mkdir /data/kafka
```

```
chown –R cp-kafka:confluent /data/kafka
```

4.install kafka

```
rpm –import https://packages.confluent.io/rpm/5.5/archive.key
```

5.add repo confluent

```
vim /etc/yum.repos.d/confluent.repo
```

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.5/8
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.5/archive.key
enabled=1

[Confluent]
```

```
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.5
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.5/archive.key
enabled=1
```

6.install kafka

`yum update –y && yum install -y confluent-kafka`

7.config kafka

`vim /etc/kafka/server.properties`

```
log.dirs=/data/kafka
```

8.enable service

`systemctl enable --now confluent-zookeeper.service`

`systemctl enable --now confluent-kafka.service`

9.start service

`systemctl start confluent-zookeeper.service`

`systemctl start confluent-kafka.service`

Commonly used codes in Kafka

1.Craet topic

`kafka-topics --bootstrap-server host 1:9092 --create --topic my-topic`

2.Producer topic

```
kafka-console-prooducer --topic my-topic --bootstrap-server host 1 :9092
```

3.Consumer topic

```
kafka-console-consumer --topic my-topic --bootstrap-server host 1 :9092
```

4.View list of topics

```
kafka-topics.sh --list --bootstrap-server <broker-address>
```

5.Describe Topic

```
kafka-topics.sh --describe --topic <topic-name> --bootstrap-server <broker-address>
```

6.Delete topic

```
kafka-topics.sh --delete --topic <topic-name> --bootstrap-server <broker-address>
```

7.Consumer group status monitor

```
kafka-consumer-groups.sh --describe --group <group-name> --bootstrap-server <broker-
address>
```

8.View list of consumer groups

```
kafka-consumer-groups.sh --list --bootstrap-server <broker-address>
```

9.Reset Offsets

```
kafka-consumer-groups.sh --reset-offsets --group <group-name> --topic <topic-name> --to-
earliest --bootstrap-server <broker-address> --execute
```

10.Kafka Broker health check

`kafka-broker-api-versions.sh --bootstrap-server <broker-address>`

11.View messages with a specific filter

`kafka-console-consumer.sh --topic <topic-name> --bootstrap-server <broker-address> --property print.key=true --property key.separator="|"`

12.Check Latency Thread

`kafka-run-class.sh kafka.tools.GetOffsetShell --topic <topic-name> --bootstrap-server <broker-address>`

final task

Configuring and deploying a Kafka cluster

zookeeper config

1.creat myid

```
# In the srv1 server
echo "1" > /var/lib/zookeeper/myid
# In the srv2 server
echo "2" > /var/lib/zookeeper/myid
# In the srv3 server
echo "3" > /var/lib/zookeeper/myid
```

2.Zookeeper config file

`vim /etc/kafka/zookeeper.properties`

```
maxClientCnxns=60
admin.enableServer=true
admin.serverPort=8080
tickTime=2000
initLimit=10
syncLimit=5
server.0=kafka1.local:2888:3888
server.1=kafka2.local:2888:3888
server.2=kafka3.local:2888:3888
```

kafka config

1.kafka config file

`vim /etc/kafka/server.properties`

```
broker.id=0             >> change by other server
advertised.listeners=PLAINTEXT://kafka1.local:9092
log.dirs=/data/kafka
offsets.topic.replication.factor=3
transaction.state.log.replication.factor=3
transaction.state.log.min.isr=2
zookeeper.connect=kafka1.local:2181,kafka2.local:2181,kafka3.local:2181
```

2. restart service

`systemctl restart confluent-kafka.service`

`systemctl restart confluent-zookeeper.service`

Provded by Hamed Taromi.