What is Kafka?

In event streaming terminology, Kafka is a well-matured solution. Kafka acts as the nervous system for streaming your data for different use cases. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premise as well as cloud environments. Kafka is a distributed, highly scalable, elastic, fault-tolerant, and secure solution for data streaming.

Key capabilities of Kafka

Kafka combines three key core capabilities to achieve our use cases. Which includes:

- To publish (write) and subscribe to (read) streams of events, including continuous import/export of your data from other systems.
- To store streams of events durably and reliably for as long as you want.
- To process streams of events.

Kafka core APIs

- Producer API
- Consumer API
- Streams API
- Connector API

Read more from here: https://docs.confluent.io/platform/current/kafka/introduction.html

Kafka core components

Brokers

- A Kafka server or Kafka Node is known as a Kafka broker.
- We can set up a Kafka cluster with one or many brokers.
- In real life, a broker is a third person who stands in between a buyer and a seller. In the case of Kafka modelling, the broker plays the same role but between Producer and Consumer. A Kafka broker receives messages from producers and stores them on a disk keyed by a unique offset.
- Kafka broker hosts topics with events in it. Topics can have one or more partitions.
- A Kafka broker allows consumers to fetch messages by topic, partition and offset.
- Kafka brokers can create a Kafka cluster by sharing information with each other directly or indirectly using Zookeeper.

Topics

- The topic is a channel where publishers publish data and where subscribers (consumers) receive data.
- It's a stream of a particular type/classification of data.
- Messages are structured or organised into topics or a particular type of messages are published to a particular topic.
- Kafka topics are identified by their name and the name is unique in the cluster. The name is our choice.
- You can create as many topics as you want.
- The data retention (how long) is configurable. The default is one week.
- Data in a topic is immutable. The data in the offset will remain immutable.
- As a comparison to RDBMS topics are similar to tables (but not containing all constraints).

Partitions

- Kafka topics are further split into partitions.
- These partitions are replicated across the brokers in a Kafka cluster.
- Data is written to partitions randomly unless a key is added to it.
- No limitations to partitioning the topic.
- Messages are stored in a sequenced fashion in one partition.

- Each message in a partition is assigned an incremental ID, also called offset.
- We can define the replication factor to increase the availability. Partitions can have copies to increase durability and availability and enable Kafka to failover to a broker with a replica of the partition if the broker with the leader partition fails.
- Read more from: https://www.instaclustr.com/the-power-of-kafka-partitions-how-to-get-the-most-out-of-your-kafka-cluster/

Topic Replication Factor

- We already pointed out the replication factor above. This is for the high availability or durability of our data/events stored in a partition.
- The replication factor 0 means we do not have any copy of our data. If we enable the replica, Kafka will create a copy of the partition and it store it to any other brokers in the cluster. So in case of any broker/node failure, we have a copy/replica on the other node and that will become the leader. So we do not need to struggle with data in case of failures.
- At a time only 1 broker can be a leader for a given partition and other brokers will have an in-sync replica. That is known as ISR.
- You can't have the number of replication factor more than the number of available brokers.

Producers

- Ones who write data to a Topic.
- Producers need to specify the Topic name and one broker details to connect to the cluster. Kafka will take of the rest. Kafka will automatically take care of sending the data to the right partition of the right broker.
- Producers have the provision to receive back the acknowledgement of the data it writes. There are three ways/types:
    - Ack = 0 [Fastest way of writing. Producers do not wait for Ack. It will write and move on.]
    - Ack = 1 [Producers wait for the Ack from the primary partition.]
    - Ack = all [Producers wait for the Ack from the primary as well as the replica partitions. Slow writes.]
- If a producer sends a key along with the message, Kafka guarantees that messages with the same key will appear in the same partition.

Consumers

- Ones who read data from the topics.
- Consumers read data from partitions in a topic.
- Consumers need to mention the topic name and a broker while connecting. Kafka will ensure the consumer is connected to the entire cluster whenever a consumer connects to a single broker as its distributed architecture.
- Consumers read data from a partition in order

- However, the order is not guaranteed across partitions. Kafka consumers read from across partitions in parallel.

Consumer Group

- A consumer group can have multiple consumer processes running.
- One consumer group will have one unique group id.
- While reading, data from one partition is read by exactly one consumer instance in one consumer group.
- If there are more than one consumer group, one instance from each of these groups can read from one single partition.
- Read more: https://docs.confluent.io/platform/current/clients/consumer.html

Zookeeper

- Zookeeper is a required component in the Kafka ecosystem.
- Zookeeper helps in managing Kafka brokers
- It also helps in the leader election of partitions.
- It helps in maintaining the cluster membership. For example, when a new broker is added, a broker is removed, a new topic is added or a topic is deleted when a broker goes down or comes up etc, Zookeeper manages such situations, and informs Kafka.
- It also manages topic configurations like the number of partitions a topic has, and the leader of the partitions for a topic.

Jump to the cheat sheet

If you are new to Kafka, there is no point in jumping into the cheat sheet without mentioning the key components. That's why I added those details above. I hope, now you guys have a brief knowledge of Kafka internals. I suggest going through the official documentation to get a clear picture of its internals. Here we go!!

Kafka services

Sl No.	Service	URL → Port
1	Zookeeper	confluent-zookeeper.service → 2181
2	Kafka	confluent-kafka.service → 9092
3	Schema Registry	confluent-schema-registry.service
4	Kafka REST	confluent-kafka-rest.service → 8082
5	Connect	confluent-kafka-connect.service → 8083
6	KSQL Server	confluent-ksql.service → 8088
7	Control Center	confluent-control-center.service

Manage services

Manage all services together

```
sudo systemctl start confluent-kafka-rest.service confluent-kafka.service confluent-control-center.service confluent-kafka-connect.service confluent-ksql.service confluent-schema-registry.service

sudo systemctl stop confluent-kafka-rest.service confluent-kafka.service confluent-control-center.service confluent-kafka-connect.service confluent-ksql.service confluent-schema-registry.service

sudo systemctl status confluent-kafka-rest.service confluent-kafka.service confluent-control-center.service confluent-kafka-connect.service confluent-ksql.service confluent-schema-registry.service
```

To restart the connect service [one service]

```
sudo systemctl stop confluent-kafka-connect.service

sudo systemctl start confluent-kafka-connect.service

sudo systemctl status confluent-kafka-connect.service
```

Like above you can manage services separately.

Connectors

List all configured connectors

```
sudo curl localhost:8083/connectors
```

List all installed connector plugins

```
sudo curl localhost:8083/connector-plugins
```

Fetch a connector configuration details

```
sudo curl localhost:8083/connectors/<connector-name> | jq .
```

Check connector status

```
sudo curl localhost:8083/connectors/<connector-name>/status
```

Restart a connector

```
sudo curl -XPOST localhost:8083/connectors/<connector-name>/restart
```

Delete a connector

```
sudo curl -X DELETE http://localhost:8083/connectors/<connector-name>
```

Tasks

List tasks

```
sudo curl localhost:8083/connectors/<connector-name>/tasks | jq
```

Restart a task

```
sudo curl -XPOST localhost:8083/connectors//tasks/<task-id>/restart
```

Topics

Create a topic

```
sudo kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic new-topic
```

List topic

```
sudo kafka-topics --list --zookeeper localhost:2181
```

List topic with details (describe)

```
sudo kafka-topics --zookeeper localhost:2181 --describe --topic <topic-name>
```

This will show the “ReplicationFactor”, “PartitionCount” and more details about a topic.

Describe all topics

```
sudo kafka-topics --describe --zookeeper localhost:2181
```

Alter / to add more partitions

```
sudo kafka-topics --zookeeper localhost:2181 --alter --topic <topic-name> --partitions 16
```

Delete a topic

```
sudo kafka-topics --zookeeper localhost:2181 --delete --topic <topic-name>
```

Also refer https://sparkbyexamples.com/kafka/kafka-delete-topic/

Details about under replicated partition

```
sudo kafka-topics --zookeeper localhost:2181/kafka-cluster --describe --under-replicated-partitions
```