What is an Apache Kafka Topic?

A Kafka Topic is a logical collection of messages that are produced and consumed by Kafka applications. A message is a piece of data that contains some information, such as an event, a record, or a notification. For example, a message could be a product created event, which contains the details of a new product that was added to an online store.

A Kafka Topic has a unique name that identifies it in a Kafka cluster. A Kafka cluster is a group of servers, called brokers, that store and manage the topics and messages. A Kafka application can connect to a Kafka cluster and interact with the topics and messages.

What is Topic Partition?

A Kafka Topic is divided into one or more partitions, each of which stores a subset of messages in an ordered sequence. A partition is a physical unit of storage and processing in a Kafka broker. Each partition has a unique identifier, called a partition ID, that is assigned by the broker.

The number of partitions for a topic is determined when the topic is created, and it can be changed later. The number of partitions affects the

scalability, parallelism, fault-tolerance, and ordering guarantees of a topic.

- **Scalability**: The more partitions a topic has, the more messages it can store and handle. A topic with many partitions can be distributed across multiple brokers, which increases the storage capacity and performance of the topic.
- **Parallelism**: The more partitions a topic has, the more producers and consumers can interact with it concurrently. A topic with many partitions can have multiple producers sending messages to different partitions, and multiple consumers receiving messages from different partitions. This increases the throughput and efficiency of the topic.
- **Fault-tolerance**: The more partitions a topic has, the more resilient it is to failures. A topic with many partitions can have replicas, which are copies of the partitions stored on different brokers. If a broker fails, the replicas can take over and continue serving the messages. This increases the availability and reliability of the topic.
- **Ordering guarantees**: The more partitions a topic has, the less strict the ordering guarantees are. A topic with many partitions can only guarantee the order of messages within each partition, but not across partitions. This means that messages from different partitions may be delivered out of order to the consumers. This may or may not be acceptable, depending on the use case of the topic.

How to create topic?

To create a topic in a Kafka cluster, we can use the `kafka-topics.sh` command line tool, which is provided by Kafka. This tool allows us to

perform various operations on topics, such as creating, deleting, listing, and describing topics.

To create a topic, we need to specify the following parameters:

- `--bootstrap-server`: The address of one or more brokers in the Kafka cluster that we want to connect to. For example, `localhost:9092`.
- `--create`: The flag that indicates that we want to create a topic.
- `--topic`: The name of the topic that we want to create. For example, `product-created-events-topic`.
- `--partitions`: The number of partitions that we want to create for the topic. For example, `3`.
- `--replication-factor`: The number of replicas that we want to create for each partition of the topic. For example, `2`.

For example, the following command creates a topic named `product-created-events-topic` with 3 partitions and 2 replicas in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --create --topic
product-created-events-topic --partitions 3 --replication-factor 2
```

If the topic is created successfully, the tool will print a confirmation message, such as:

```
Created topic product-created-events-topic.
```

If the topic already exists, or if there is an error, the tool will print an error message, such as:

```
Topic 'product-created-events-topic' already exists.
```

How to create topic with partitions?

As I explained in the previous section, a Kafka Topic is divided into one or more partitions, each of which stores a subset of messages in an ordered sequence. The number of partitions for a topic affects its scalability, parallelism, fault-tolerance, and ordering guarantees.

To create a topic with partitions, you can use the `kafka-topics.sh` command line tool, which I introduced in the previous section. This tool allows you to perform various operations on topics, such as creating, deleting, listing, and describing topics.

To create a topic with partitions, you need to specify the following parameters:

- `--bootstrap-server`: The address of one or more brokers in the Kafka cluster that you want to connect to. For example, `localhost:9092`.
- `--create`: The flag that indicates that you want to create a topic.
- `--topic`: The name of the topic that you want to create. For example, `product-created-events-topic`.
- `--partitions`: The number of partitions that you want to create for the topic. For example, `3`.

For example, the following command creates a topic named `product-created-events-topic` with 3 partitions in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --create --topic
product-created-events-topic --partitions 3
```

If the topic is created successfully, the tool will print a confirmation message, such as:

```
Created topic product-created-events-topic.
```

If the topic already exists, or if there is an error, the tool will print an error message, such as:

```
Topic 'product-created-events-topic' already exists.
```

How to update topic?

Sometimes, you may want to change the configuration and properties of an existing topic, such as changing the number of partitions, the replication factor, the retention policy, and other parameters. You can do this by using the `kafka-topics.sh` command line tool, which I introduced in the previous sections. This tool allows you to perform various operations on topics, such as creating, deleting, listing, and describing topics.

To update a topic, you need to specify the following parameters:

- `--bootstrap-server`: The address of one or more brokers in the Kafka cluster that you want to connect to. For example, `localhost:9092`.
- `--alter`: The flag that indicates that you want to update a topic.
- `--topic`: The name of the topic that you want to update. For example, `product-created-events-topic`.

- `--config`: A comma-separated list of `key=value` pairs, where `key` is the name of the parameter that you want to update, and `value` is the new value that you want to assign to it. For example, `retention.ms=43200000`.

For example, the following command updates the retention policy of the topic named `product-created-events-topic` to 12 hours in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic
product-created-events-topic --config retention.ms=43200000
```

There are many parameters that you can update for a topic, such as the number of partitions, the replication factor, the cleanup policy, the compression type, and more. You can find the full list of topic configurations and their descriptions in the Kafka topic configuration reference.

In the following subsections, I will show you how to update some of the common parameters for a topic.

How to update topic partitions?

The number of partitions for a topic determines how many subsets of messages the topic is divided into. The number of partitions affects the scalability, parallelism, fault-tolerance, and ordering guarantees of a topic.

To update the number of partitions for a topic, you can use the `--partitions` option in the `kafka-topics.sh` command line tool, and specify the new number of partitions that you want to assign to the topic. For example, the following command increases the number of

partitions for the topic named `product-created-events-topic` to 6 in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic
product-created-events-topic --partitions 6
```

Note that you can only increase the number of partitions for a topic, but not decrease it. This is because reducing the number of partitions would cause data loss and inconsistency. If you want to reduce the number of partitions for a topic, you have to delete the topic and recreate it with the desired number of partitions.

How to update topic replication factor?

The replication factor for a topic determines how many copies of each partition are stored on different brokers. The replication factor affects the fault-tolerance and availability of a topic.

To update the replication factor for a topic, you can use the `--replication-factor` option in the `kafka-topics.sh` command line tool, and specify the new replication factor that you want to assign to the topic. For example, the following command increases the replication factor for the topic named `product-created-events-topic` to 3 in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic
product-created-events-topic --replication-factor 3
```

Note that you can only increase the replication factor for a topic, but not decrease it. This is because reducing the replication factor would cause data loss and inconsistency. If you want to reduce the replication factor

for a topic, you have to delete the topic and recreate it with the desired replication factor.

How to update retention policy?

The retention policy for a topic determines how long the messages in the topic are retained before they are deleted. The retention policy affects the storage capacity and performance of a topic.

To update the retention policy for a topic, you can use the `--config` option in the `kafka-topics.sh` command line tool, and specify the new retention policy that you want to assign to the topic. There are two ways to specify the retention policy for a topic: by time or by size.

- To specify the retention policy by time, you can use the `retention.ms` parameter, and specify the maximum time in milliseconds that a message can remain in the topic before it is deleted. For example, the following command sets the retention policy for the topic named `product-created-events-topic` to 12 hours in a Kafka cluster with a broker at `localhost:9092`:
- `kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic product-created-events-topic --config retention.ms=43200000`
- To specify the retention policy by size, you can use the `retention.bytes` parameter, and specify the maximum size in bytes that the topic can occupy before the oldest messages are deleted. For example, the following command sets the retention policy for the topic named `product-created-events-topic` to 1 GB in a Kafka cluster with a broker at `localhost:9092`:
- `kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic product-created-events-topic --config retention.bytes=1073741824`

You can also use both parameters together, and the topic will be deleted when either of the conditions is met.

How to delete topic?

Sometimes, you may want to delete a topic from a Kafka cluster, either because you no longer need it, or because you want to recreate it with different settings. You can do this by using the `kafka-topics.sh` command line tool, which I introduced in the previous sections. This tool allows you to perform various operations on topics, such as creating, updating, listing, and describing topics.

To delete a topic, you need to specify the following parameters:

- `--bootstrap-server`: The address of one or more brokers in the Kafka cluster that you want to connect to. For example, `localhost:9092`.
- `--delete`: The flag that indicates that you want to delete a topic.
- `--topic`: The name of the topic that you want to delete. For example, `product-created-events-topic`.

For example, the following command deletes the topic named `product-created-events-topic` from a Kafka cluster with a broker at `localhost:9092`:

```
kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic
product-created-events-topic
```

If the topic is deleted successfully, the tool will print a confirmation message, such as:

```
Topic product-created-events-topic is marked for deletion.
```

If the topic does not exist, or if there is an error, the tool will print an error message, such as:

```
Topic 'product-created-events-topic' does not exist.
```

How to read messages from the topic?

A Kafka Consumer is an application that receives messages from one or more topics in a Kafka cluster. A consumer can subscribe to one or more topics, and consume messages from them in a streaming or batch manner. A consumer can also specify the offset, or the position, from which it wants to start consuming messages from a topic. For example, a consumer can start from the beginning, the end, or a specific offset of a topic.

To read messages from a topic, you can use the `kafka-console-consumer.sh` command line tool, which I introduced in the previous sections. This tool allows you to consume messages from a topic and print them to the console (standard output). By default, it outputs the raw bytes in the message with no formatting (using the Default Formatter).

To read messages from a topic, you need to specify the following parameters:

- `--bootstrap-server`: The address of one or more brokers in the Kafka cluster that you want to connect to. For example, `localhost:9092`.
- `--topic`: The name of the topic that you want to read messages from. For example, `product-created-events-topic`.
- `--from-beginning`: The flag that indicates that you want to read messages from the beginning of the topic. If you omit this flag, the

tool will read messages from the end of the topic, or the latest offset.

- `--property`: A comma-separated list of `key=value` pairs, where `key` is the name of the property that you want to set, and `value` is the value that you want to assign to it. For example, `print.key=true,key.separator=-`. You can use this option to customize the output format of the messages, such as showing the key and the value, or changing the separator between them.

For example, the following command reads messages from the topic named `product-created-events-topic` from the beginning of the topic, and shows the key and the value of each message, separated by a dash, in a Kafka cluster with a broker at `localhost:9092`:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
product-created-events-topic --from-beginning --property
print.key=true,key.separator=-
```

If the tool successfully connects to the topic and starts consuming messages, it will print them to the console.