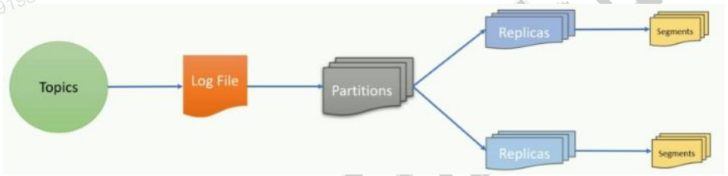
Apache Kafka

Kafka Storage Architecture

In the Kafka Storage Architecture, we will understand the core concepts of Kafka such as:

- ✓ Topics
- ✓ Logs
- ✓ Partitions
- ✓ Replication Factor
- ✓ Segments
- ✓ Offset and
- ✓ Offset index

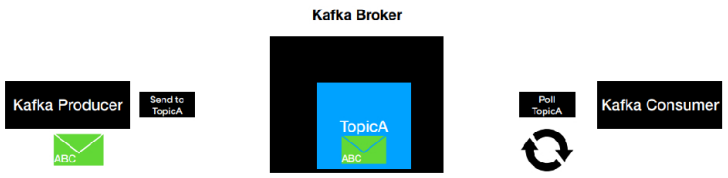


Topic:

A Topic is a logical name to group the messages, like in a database, we must create the table to store the data in records, in kafka we must create a topic to store the messages.

- ✓ The Topic is the mechanism to categorize the messages. If we are coming from the database world, we **can think of the topic as a table name.**
- ✓ The producer always writes the message to a Topic, and the consumer reads it from the Topic.
- ✓ The broker creates a log file (a partitioned and replicated log) for each topic.
- ✓ When a producer sends a message, it specifies the topic name for the message and accordingly the broker persists the message in the corresponding log file.
- ✓ When a consumer wants to read the message, they consume messages from the specified topic.

Apache Kafka



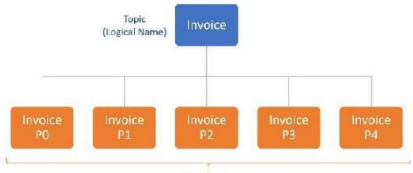
When we create a topic in Apache Kafka, we must specify two configurations:

1. Number of Partitions
2. Replication Factor

If we do not provide these parameters, Kafka assumes the default values as Number Of Partitions as 1 and Replication Factor as also 1, but every topic must have some value for these two parameters.

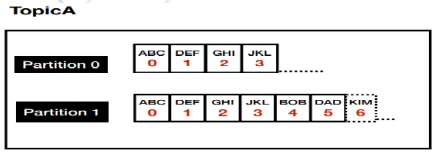
Partitions

A single topic may store millions of messages, and hence, it is not practical to keep all those messages in a single file. The topic partitions are a mechanism to break the topic further into smaller parts. For **Apache Kafka, a partition is nothing but a physical directory**



Apache Kafka creates a separate directory for each topic partition. If we create a topic for invoices and specify five partitions, that means, Kafka will create five directories for the invoices. The actual log files are stored within these directories as segment files.

Example:



✓ Each Partition is an ordered , immutable sequence of records

Apache Kafka

✓ Each record is assigned a sequential number called offset

✓ Each partition is independent of each other

✓ **Ordering is guaranteed only at the partition level**

✓ Partition continuously grows as new records are produced

✓ All the records are persisted in a commit log in the file system where Kafka is installed

Replication Factor

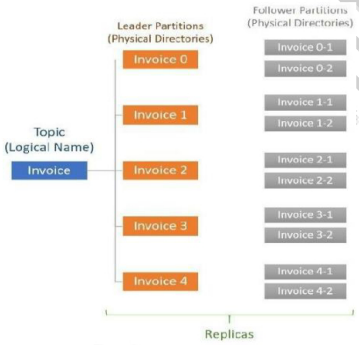
The replication factor specifies how many copies do we want to maintain for each partition. The purpose of keeping multiple copies of the partitions is to provide **fault tolerance.**

The replication factor multiplies to the number of partitions. For example, we created one topic for Invoices. While creating the topic, we specified

➢ the number of partitions as 5 and

➢ a replication factor as 3.

In this case, Kafka should create fifteen directories. The calculation is simple. We wanted to produce five partitions and maintain three copies of each partition, and that makes fifteen directories. We term these directories as a **partition replica.** As shown in the below diagram, we have fifteen partition replicas. They are nothing but directories.



We can classify those fifteen directories into two categories:

1. Leaders

2. Followers

While creating the topic, we specified the number of partitions as 5 and Kafka creates five directories. These five directories are called the Leader Partitions. So, the leaders are created first.

Apache Kafka

Then we also specified the replication factor as 3. That means Kafka should ensure three copies for each of these five partitions. One is already there (the leader), hence Kafka creates two more directories for each broker, and we call these copies as Followers.

Remember, followers are duplicate copies of the leader partitions. So the leader partitions are followed by the several followers. The number of follower partition depends on the replication factor.

Note:

1. Since the partition replica is a directory, it cannot grow beyond the capacity of the disk or the mount point. **So, the maximum size of a partition is limited by the space available on a single mount point**.

2. The replication factor number should be less than or equals to the number of brokers otherwise the replication wont be happened and will end up with the following error:

```
D:\personal\my_trainigs\7.apache-kafka\kafka-multi-node-cluster\commands>kafka-topics --bootst
rap-server localhost:9092 --create --topic stock-ticks --partitions=5 --replication-factor=5
Error while executing topic command : Replication factor: 5 larger than available brokers: 3.
[2023-05-07 09:01:38,072] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorExcepti
on: Replication factor: 5 larger than available brokers: 3.
 (kafka.admin.TopicCommand$)

D:\personal\my_trainigs\7.apache-kafka\kafka-multi-node-cluster\commands>
```

3. The best industry practice is to have the **replication factor number= no of available brokers**

Segment

Finally, the messages are stored within the directories in the log files. However, instead of creating one large file in the partition directory, Kafka creates several smaller files. That means the ***Kafka log file is split into smaller files known as segments.*** So, if we look inside any of the partition directories, we will see a bunch of segment files in each directory as shown below:

Let's try to understand how this split happens:

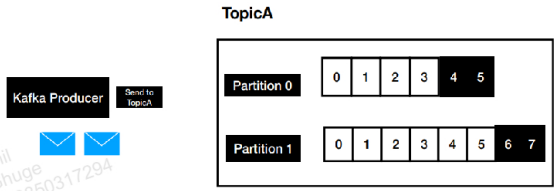
When the partition receives its first message, it creates a segment file and stores the message in the segment.

The next message also goes in the same segment, and the segment file continues to grow until the maximum segment limit is reached. **The default maximum segment size is either 1 GB of data or a week of data, whichever is smaller**.

As the Kafka broker is writing to a partition, if the segment limit is reached, they close the file and start a new one.

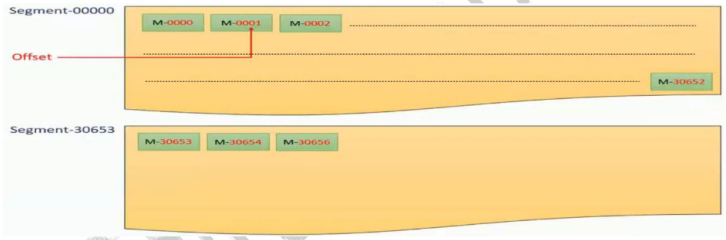
That's how we get multiple segment files in each partition directory. The format of the segment file is a sequence of messages. The internal structure of the message may be changing over the Kafka versions, and those changes may not impact our application because we will be using standard APIs to read/write these messages.

However, at a high level, a message contains a variable length header, a variable length key byte array, and a variable length value byte array. Each Kafka message in the partition is uniquely identified by a 64-bit integer offset.



Offset

Every Kafka message within a single partition is uniquely identified by the offset. For example, the offset for the first message in the partition would be 0000, the offset for the second message would be 0001 and so on.



This numbering also continues across the segments to keep the offset unique within the partition.

Let's consider that the offset of the last message in the first segment is **M-30652**. Assume that the maximum segment limit is reached, so the Kafka should close this segment and create a new segment file for the next message.

The offset of the first message in the new segment continues from the earlier segment and hence the offset of the first message in the second segment would be **M-30653**. For an easy identification, segment file name is also suffixed by the first offset in that segment.

Note:

The offset is unique within the partition. If we look across the partitions, the offset starts from zero in each partition. Hence the offset is a 64-bit integer giving a unique ID of the message in a given partition.

Since the offsets are not unique across the topic, if we want to locate a specific message, we must know at least three things:

- ✓ Topic name,
- ✓ Partition number and then
- ✓ the Offset number

- the Offset number.

Offset Index

We now understand that each message in Kafka can be uniquely identified by providing three things.

- ✓ Topic Name
- ✓ Partition Number
- ✓ Segments and
- ✓ offsets

If we think in database terms to locate a pertucular record we will writes the queries like:

Select * from invoices where customer-name='John'

this unique identification does not make a good sense for locating the messages in case of Kafka.

In Kafka, The topic name is like a table name in the database. But partition and offset are just numbers. In database applications, we may want to access data on some critical columns such as invoice_number, customer_name or customer_id. However, we cannot do that in Kafka because messages are not structured into column names.

However, this may not be a problem for a real-time stream processing application. In a stream processing application, the requirement is different. A stream processing application wants to read all messages in a sequence.

Let's try to understand it with a simple example. Let's assume that we have a stream processing application that computes loyalty points for the customer in real-time. The application should read each invoice and calculate loyalty points. While computing loyalty points, we need customer_id and the amount, but we must read all the events. How does it happen?

1. The application connects to the broker and asks for the messages starting from the offset 0000.

2. Let's assume the broker sends ten messages to the application.

3. The application takes a few milliseconds to compute loyalty points on those ten invoices.

4. Now it is again ready to process a few more messages. So, it goes back to the broker and requests for more messages starting from offset 0010.

5. The broker provides another batch of fifteen messages starting from offset 0010 to 0024.

6. Next time, the application requests for another set of messages starting from the offset 0025. This process continues for the life of the application.

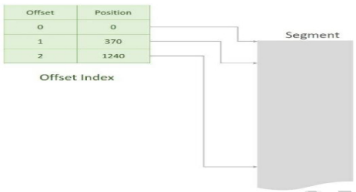
The process explained in this example is the typical pattern of how a stream processing application would work.

In this process, we must have noticed, that the **consumer application is requesting messages based on the offset. Kafka allows consumers to start fetching messages from a given offset number.**

This means, if a consumer demands for messages beginning at offset 100, the broker must be able to locate the message for offset 100.

To help brokers rapidly find the message for a given offset, Kafka maintains an index of offsets. The following diagram shows a logical view of the index.

The index files are also segmented for easy management, and they are also stored in the partition directory along with the log file segments.



Time Index

Kafka allows the consumers to start searching messages based on the offset number.However, many use cases might need to seek messages based on timestamp.

In Such requirements as we want to read all the events that are created after a specific timestamp, Kafka maintains a timestamp for each message and builds a time index to quickly seek the first message that arrived after the given timestamp.

The time index is like the offset index, and it is also segmented and stored in the partition directory along with the offset index and log file segment.