medium.com

A Practical Introduction to Kafka Storage Internals

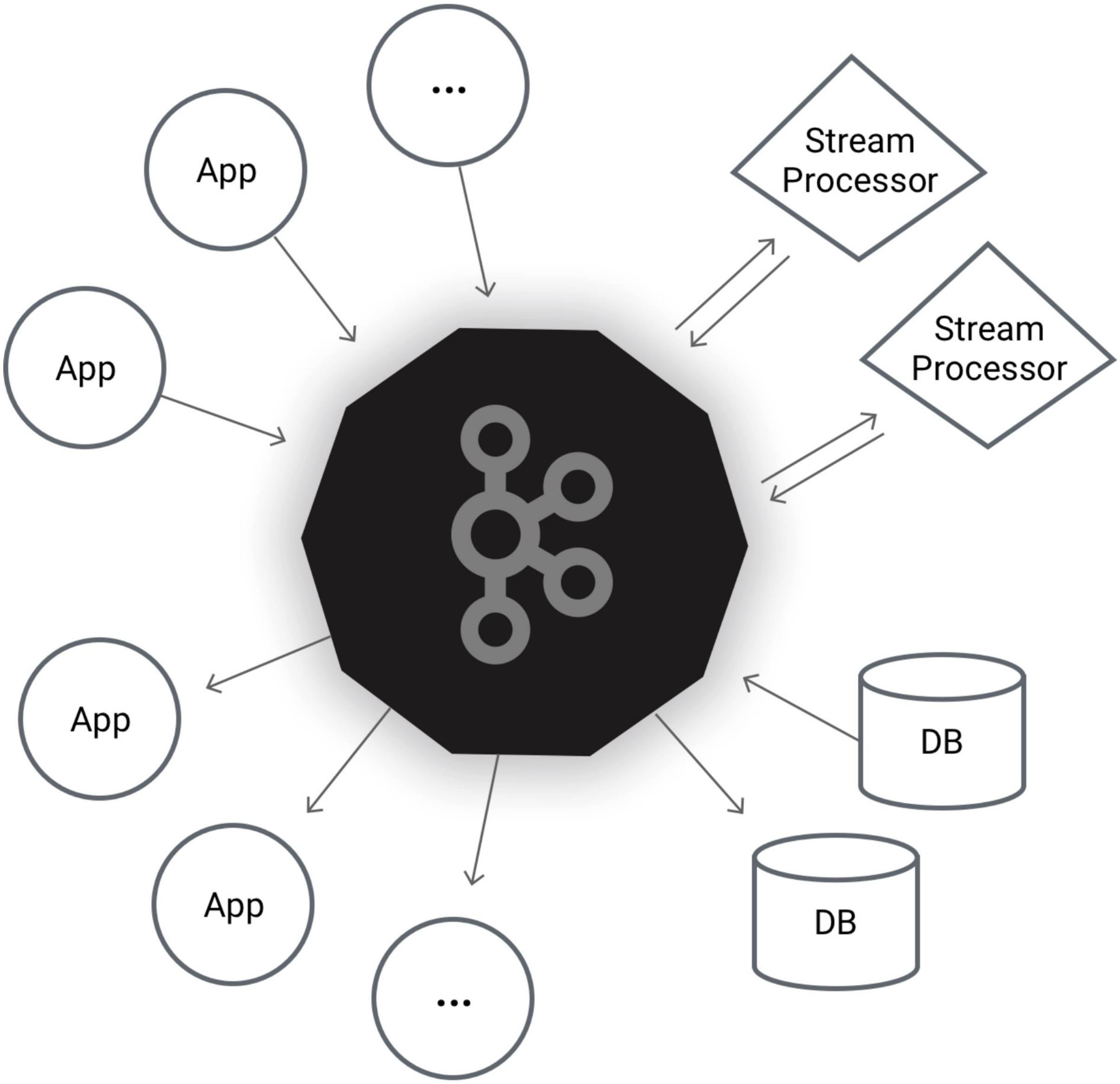
Durga Swaroop Perla

15-19 minutes

A Practical Introduction to Kafka Storage Internals

Kafka is everywhere these days. With the advent of Microservices and distributed computing, Kafka has become a regular occurrence in architecture's of every product. In this article, I'll try to explain how Kafka's internal storage mechanism works.

Since this is going to be a deep dive into Kafka's internals, I would expect you to have some understanding about Kafka. Although I've tried to keep the entry level for this article pretty low, you might not be able to understand everything if you're not familiar with the general workings of it. Proceed further with that in mind.



Kafka is usually referred to as a *Distributed, Replicated Messaging*

Queue, which although technically true, usually lead to some confusion depending on your definition of what a *messaging queue* is. Instead, I prefer the definition **Distributed, Replicated Commit Log**. This I think clearly represents what Kafka does as all of us understand how logs are written to disk. And in this case, it is the messages pushed into Kafka that are stored to disk.

With reference to storage in Kafka, you'll always hear two terms, Partition and Topic. ***Partitions*** are the units of storage in Kafka for messages. And ***Topic*** can be thought of as being a container in which these partitions lie.

With the basic stuff out of our way, let's understand these concepts better by working with Kafka.

I am going to start by creating a topic in Kafka with three partitions. If you want to follow along, the command looks like this for a local Kafka setup on windows.

```
kafka-topics.bat --create --topic freblogg
--partitions 3 --replication-factor 1 --zookeeper
localhost:2181
```

If I go into Kafka's log directory, I see three directories created as follows.

```
$ tree freblogg*
freblogg-0
|-- 00000000000000000000.index
|-- 00000000000000000000.log
|-- 00000000000000000000.timeindex
`-- leader-epoch-checkpoint
freblogg-1
|-- 00000000000000000000.index
|-- 00000000000000000000.log
|-- 00000000000000000000.timeindex
`-- leader-epoch-checkpoint
freblogg-2
|-- 00000000000000000000.index
|-- 00000000000000000000.log
|-- 00000000000000000000.timeindex
`-- leader-epoch-checkpoint
```

We have three directories created because we've given three partitions for our topic, which means that each partition gets a directory on the file system. You also see some files like *index, log etc*. We'll get to them shortly.

One more thing that you should be able to see from here is that in Kafka, the **topic** is more of a logical grouping than anything else and that the **Partition is the actual unit of storage in Kafka**. That is what is physically stored on the disk. Let's understand partitions in some more detail.

Partitions

A partition, in theory, can be described as an immutable collection (or sequence) of messages. We can only append messages to a partition but cannot delete from it. And by "We", I am talking about the Kafka producer. A producer can't delete the messages in the topic.

Now we'll send some messages into the topic. But before that, I want you to see the sizes of files in our partition folders.

```
$ ls -lh freblogg-0
total 20M
```

```
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121   0 Aug  5 08:26
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121   0 Aug  5 08:26 leader-epoch-
checkpoint
```

You see the index files combined are about 20M in size while the log file is completely empty. This is the same case with `freblogg-1` and `freblogg-2` folders.

Now let us send a couple of messages and see what happens. To send the messages I'm using the console producer as follows:

```
kafka-console-producer.bat --topic freblogg
--broker-list localhost:9092
```

I have sent two messages, first a customary "hello world" and then I pressed the Enter key, which becomes the second message. Now if I print the sizes again:

```
$ ls -lh freblogg*
freblogg-0:
total 20M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121   0 Aug  5 08:26
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121   0 Aug  5 08:26 leader-epoch-
```

```
checkpoint
freblogg-1:
total 21M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121  68 Aug  5 10:15
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121  11 Aug  5 10:15 leader-epoch-
checkpoint

freblogg-2:
total 21M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121  79 Aug  5 09:59
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121  11 Aug  5 09:59 leader-epoch-
checkpoint
```

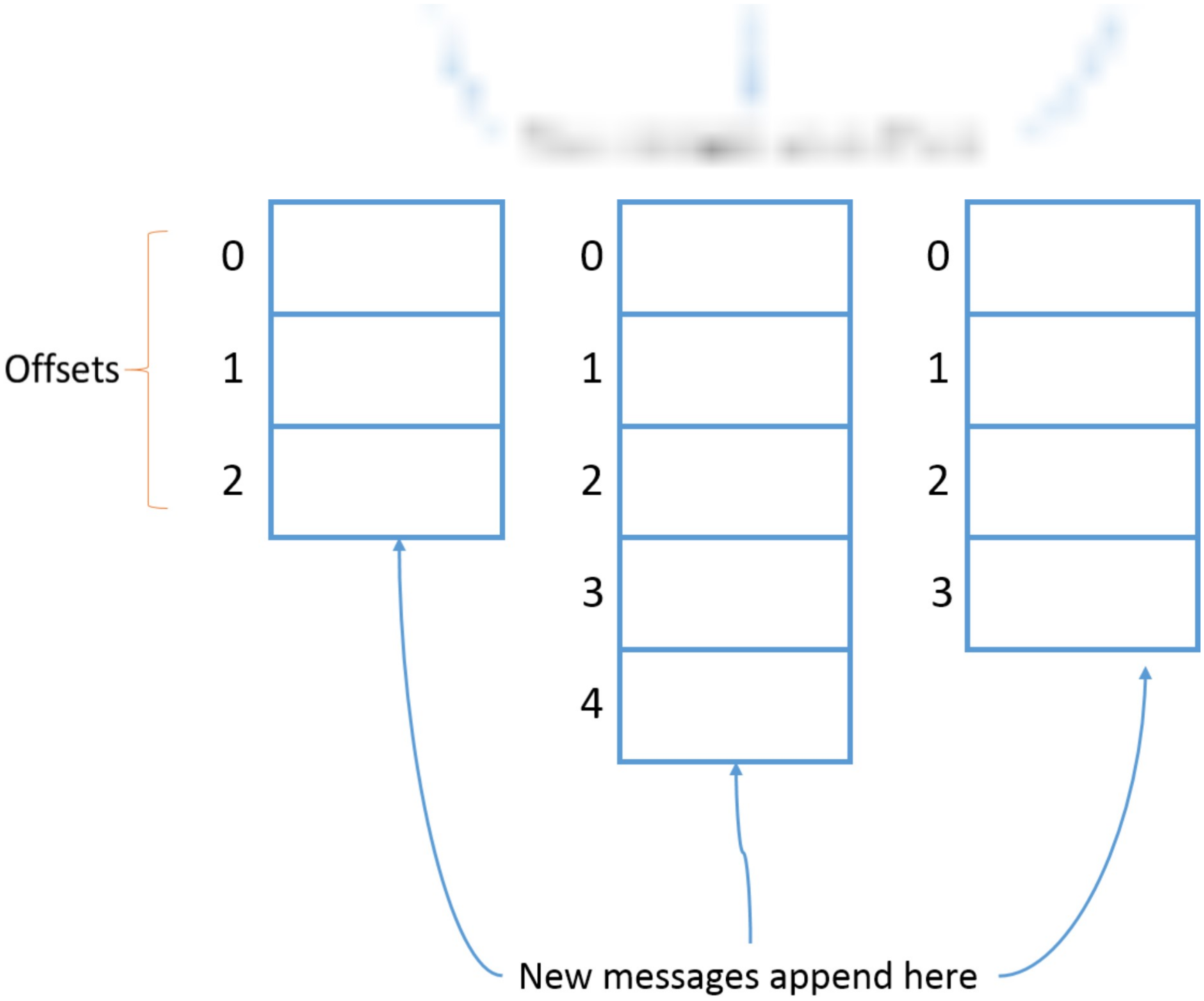
Our two messages went into two of the partitions where you can see that the log files have a non zero size. This is because **the messages in the partition are stored in the 'xxxx.log' file**. To confirm that the messages are indeed stored in the log file, we can just see what's inside that log file.

```
$ cat freblogg-2/*.log
@^@^BÂ°Â£Ã¦Ãƒ^@^K^XÃ¿Ã¿Ã¿Ã¿Ã¿Ã¿^@^@^@^A"^@^@^A^VHello
World^@
```

The file format of the ‘log’ file is not one that is conducive for textual representation but nevertheless, you should see the ‘Hello World’ at the end indicating that this file got updated when we have sent the message into the topic. The second message we have sent went into the other partition.

Notice that the first message we sent, went into the third partition (freblogg-2) and the second message went into the second partition (freblogg-1). This is because Kafka arbitrarily picks the partition for the first message and then distributes the messages to partitions in a round robin fashion. If a third message comes now, it would go into freblogg-0 and this order of partition continues for any new message that comes in. We can also make Kafka choose the same partition for our messages by adding a key to the message. Kafka stores all the messages with the same key into a single partition.

Each new message in the partition gets an Id which is one more than the previous Id number. This Id number is also called as the *Offset*. So, the first message is at ‘offset’ 0, the second message is at offset 1 and so on. These offset Id’s are always incremented from the previous value.



<Quick detour>

We can understand those random characters in the log file, using a Kafka tool. Those extra characters might not seem useful to us, but they are useful for Kafka as they are the metadata for each message in the queue. If I run,

```
kafka-run-class.bat kafka.tools.DumpLogSegments
--deep-iteration --print-data-log --files
logs\freblogg-2\00000000000000000000.log
```

This gives the output

```
Dumping logs\freblogg-2\00000000000000000000.log
Starting offset: 0
```

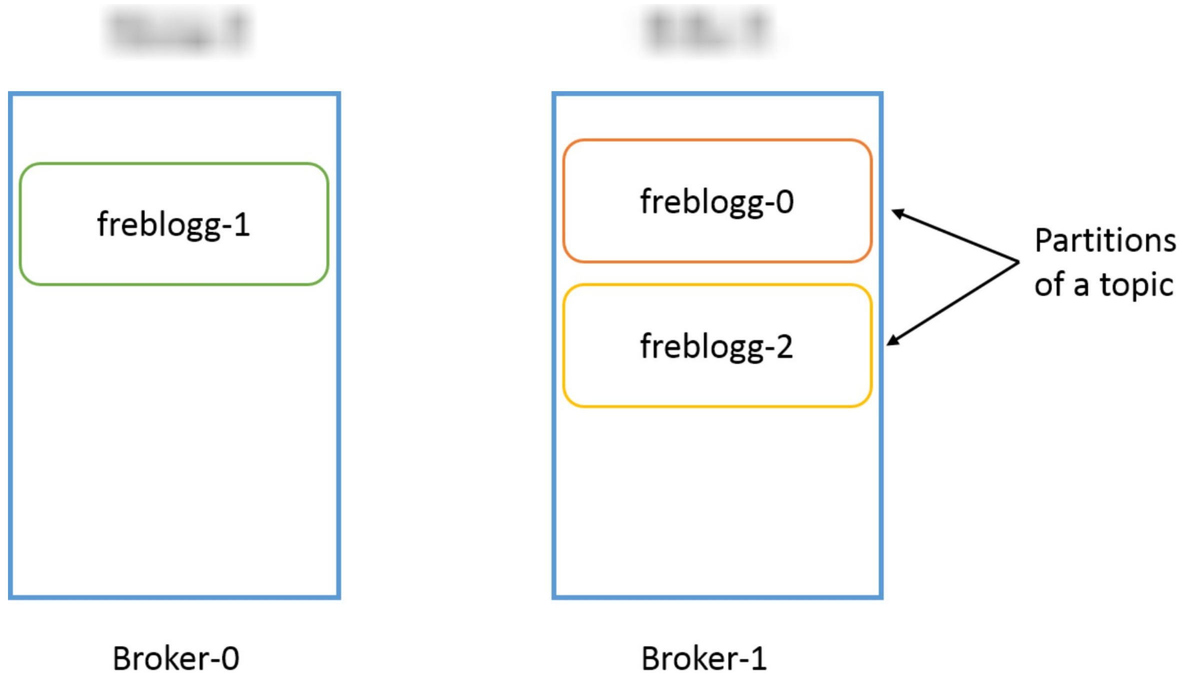
```
offset: 0 position: 0 CreateTime: 1533443377944
isvalid: true keysize: -1 valuesize: 11
producerId: -1 headerKeys: [] payload: Hello World

offset: 1 position: 79 CreateTime: 1533462689974
isvalid: true keysize: -1 valuesize: 6 producerId:
-1 headerKeys: [] payload: amazon
```

(I've removed a couple of things from this output which are not necessary for this discussion.)

You can see that it stores information of the **offset**, **time of creation**, **key and value sizes** etc along with the actual message payload in the log file.

</Quick detour>

It is also important to note that **a partition is tied to a broker**. In other words, If we have three brokers and if the folder `freblogg-0` exists on broker-1, you can be sure that it will not appear in any of the other brokers. Partitions of a topic can be spread out to multiple brokers but a partition is always present on one single Kafka broker (When the replication factor has its default value, which is 1. Replication is mentioned further below).



Segments

We’ll finally talk about those index and log files we’ve seen in the partition directory. Partition might be the standard unit of storage in Kafka, but it is not the lowest level of abstraction provided. Each partition is sub-divided into **segments**.

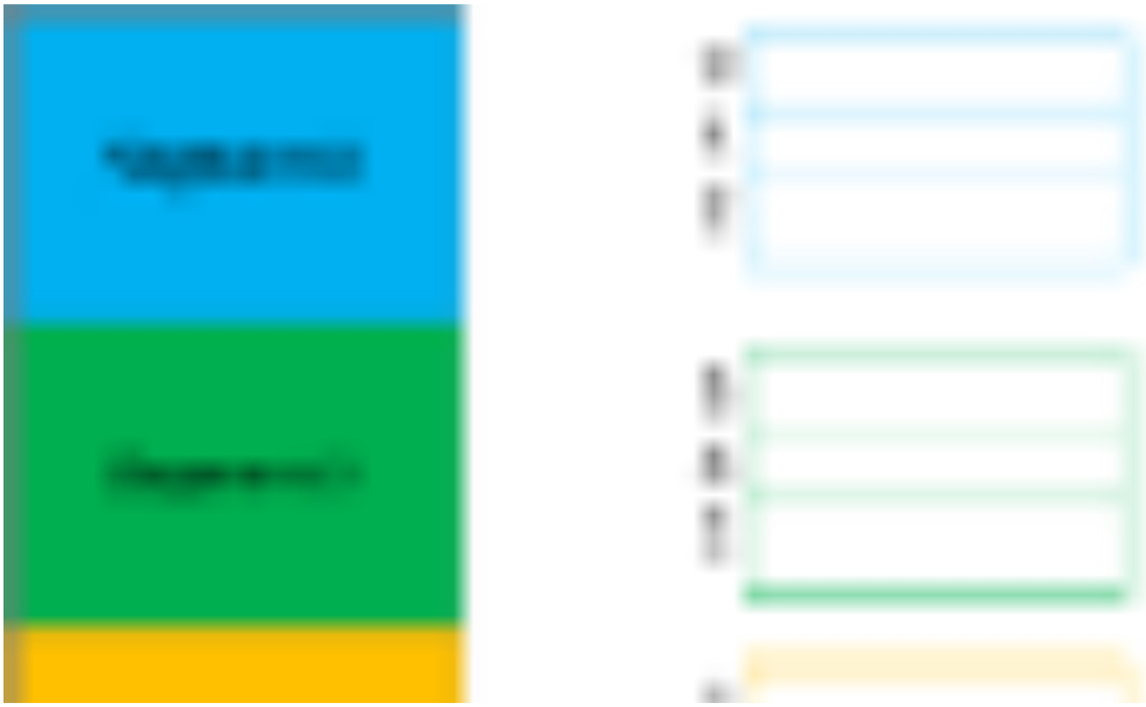
A segment is simply a collection of messages of a partition. Instead of storing all the messages of a partition in a single file (think of the log file analogy again), Kafka splits them into chunks called segments. Doing this provides several advantages. Divide and Conquer FTW!

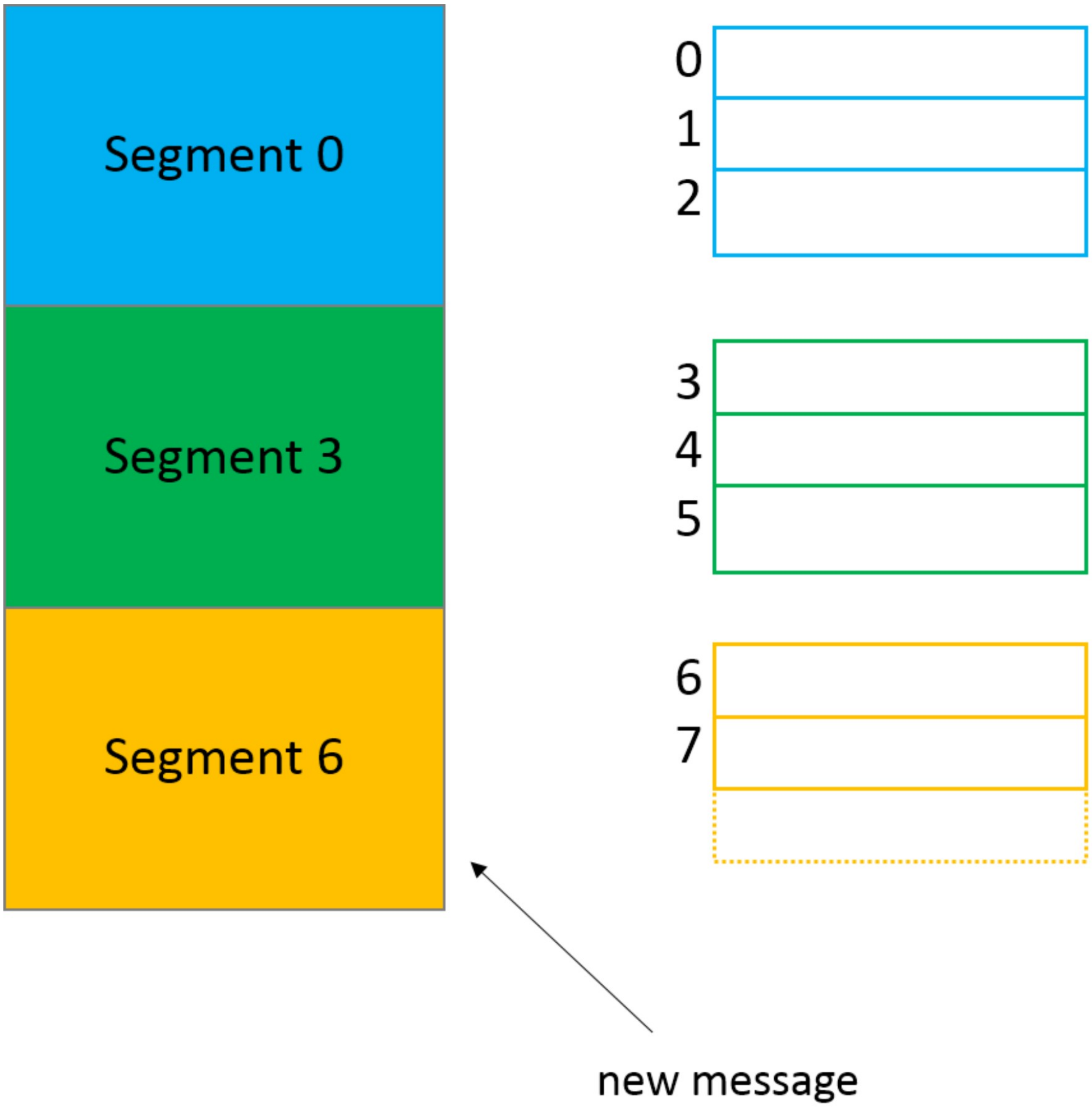
Most importantly, it makes purging data easy. As previously introduced partition is immutable from a consumer perspective. But Kafka can still remove the messages based on the “Retention policy” of the topic. Deleting segments is much simpler than deleting things from a single file, especially when a producer might be pushing data into it.

```
$ ls -lh freblogg-0
total 20M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121   0 Aug  5 08:26
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121   0 Aug  5 08:26 leader-epoch-
checkpoint
```

The `00000000000000000000` in front of the log and the index files in each partition folder, is the name of our segment. Each segment file has `segment.log`, `segment.index` and `segment.timeindex` files.

Kafka always writes the messages into these segment files under a partition. There is always an *active* segment to which Kafka writes to. Once the segment's size limit is reached, a new segment file is created and that becomes the newly active segment.





Each segment file is created with the offset of the first message as its file name. So, In the above picture, segment 0 has messages from offset 0 to offset 2, segment 3 has messages from offset 3 to 5 and so on. Segment 6 which is the last segment is the active segment.

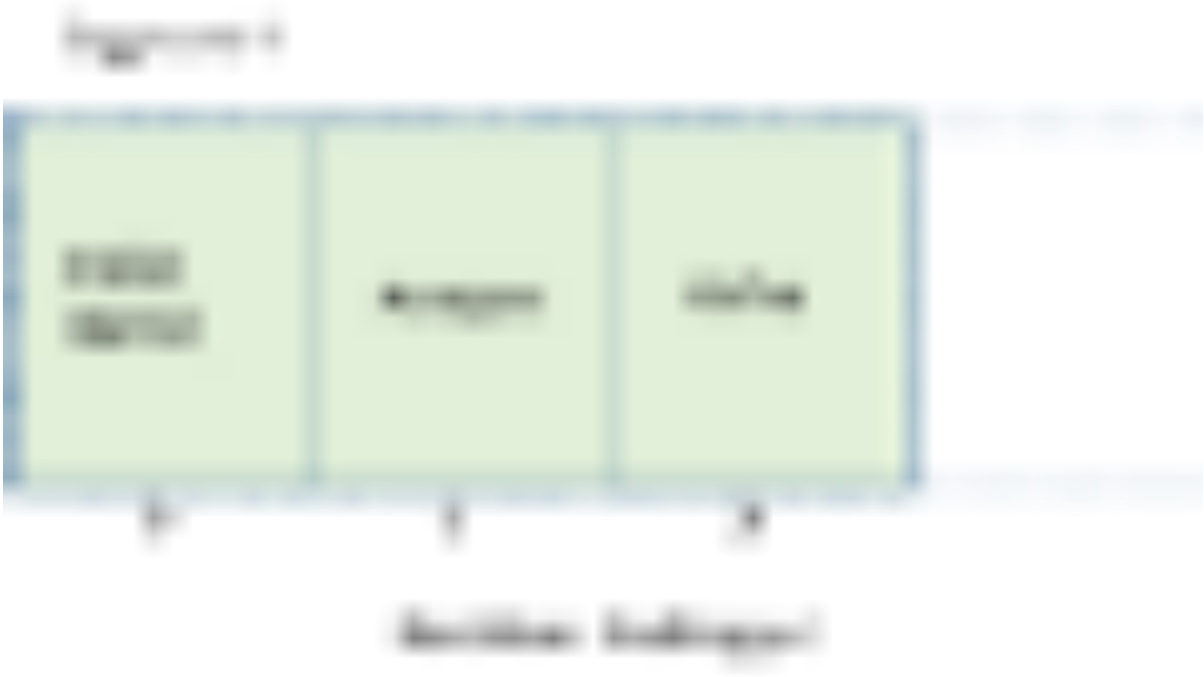
```
$ ls -lh freblogg*
freblogg-0:
total 20M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121   0 Aug  5 08:26
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121   0 Aug  5 08:26 leader-epoch-
checkpoint
freblogg-1:
total 21M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121  68 Aug  5 10:15
00000000000000000000.log
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121  11 Aug  5 10:15 leader-epoch-
checkpoint

freblogg-2:
total 21M
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.index
- freblogg 197121  79 Aug  5 09:59
00000000000000000000.log
```

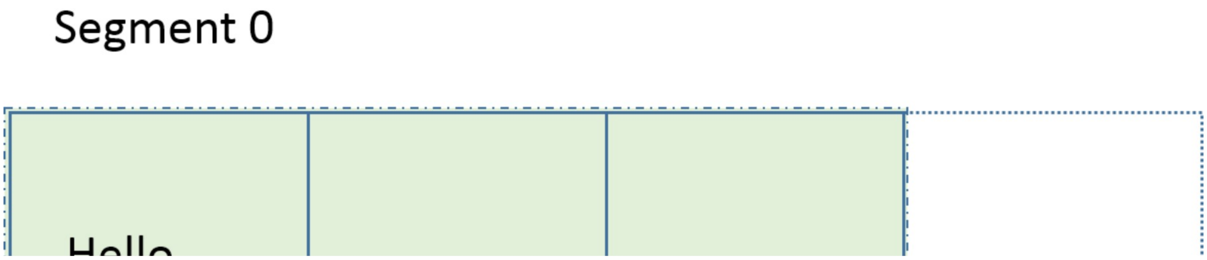
```
- freblogg 197121 10M Aug  5 08:26
00000000000000000000.timeindex
- freblogg 197121  11 Aug  5 09:59 leader-epoch-
checkpoint
```

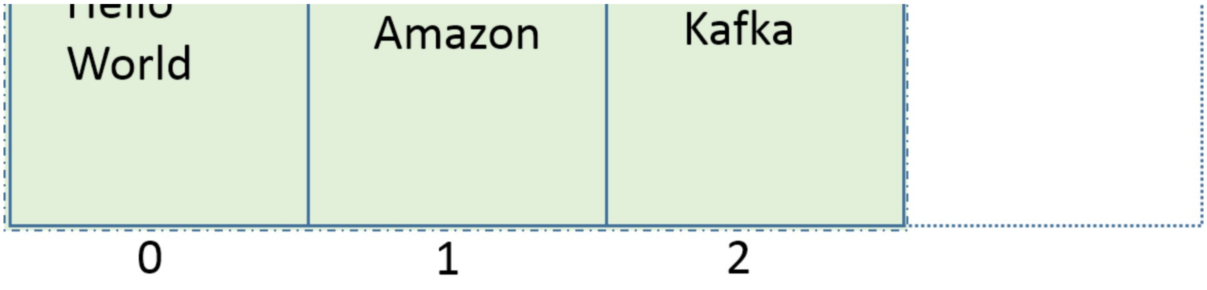
In our case, we only had one segment in each of our partitions which is `00000000000000000000`. Since we don't see another segment file present, it means that `00000000000000000000` is the active segment in each of those partitions.

The default value for segment size is a high value (1 GB) but let’s say we’ve tweaked Kafka configuration so that each segment can hold only three messages. Let’s see how that would play out.

Say this is the current state of the `freblogg-2` partition. We've three messages pushed into it.





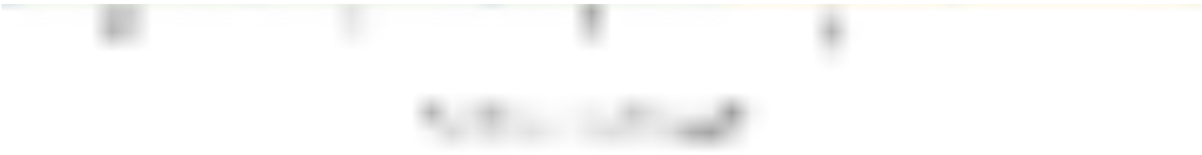


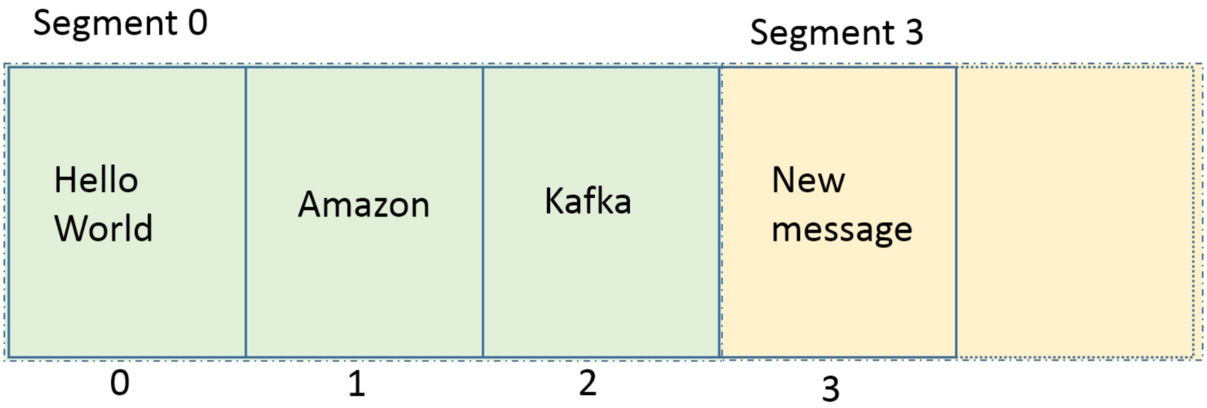
Partition: freblogg-2

Since ‘three messages’ is the limit we’ve set, If a new message comes into this partition, Kafka will automatically close the current segment, create a new segment, make that the active segment and store that new message in the new segment’s log file.

```
(I'm not showing the preceding zeroes to make it
easy on the eyes)
freblogg-2
|-- 00.index
|-- 00.log
|-- 00.timeindex
|-- 03.index
|-- 03.log
|-- 03.timeindex
`--
```

You should’ve noted that the name of the newer segment is not `01`. Instead, you see `03.index, 03.log`. So, what is going on?



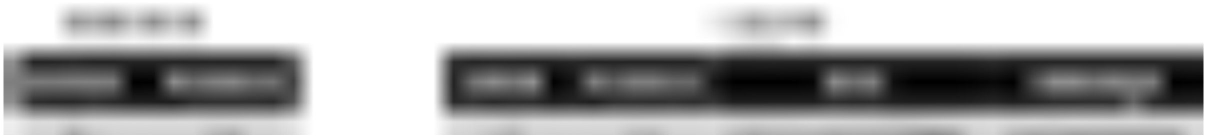


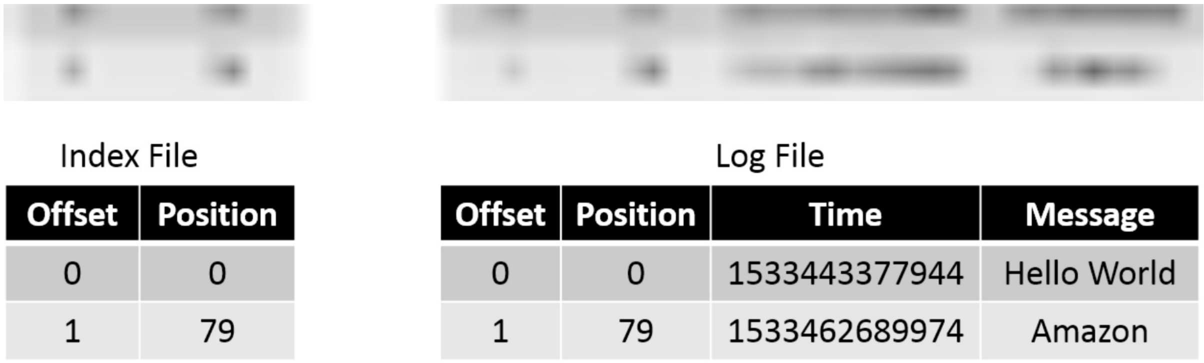
Partition: freblogg-2

This is because Kafka makes the lowest offset in the segment as its name. Since the new message that came into the partition has the offset `3`, that is the name Kafka gives for the new segment. It also means that since we have `00` and `03` as our segments, we can be sure that the messages with offsets 0,1 and 2 are indeed present in the `00` segment. New messages coming into `freblogg-2` partition with offsets 3,4 and 5 will be stored in the segment `03`.

One of the common operations in Kafka is to read the message at a particular offset. For this, if it has to go to the log file to find the offset, it becomes an expensive task especially because the log file can grow to huge sizes (Default — 1G). This is where the `.index` file becomes useful. **Index file stores the offsets and physical position of the message in the log file**.

An index file for the log file I've showed in the 'Quick detour' above would look something like this:





Index File

Offset	Position
0	0
1	79

Log File

Offset	Position	Time	Message
0	0	1533443377944	Hello World
1	79	1533462689974	Amazon

If you need to read the message at offset 1, you first search for it in the index file and figure out that the message is in position `79`. Then you directly go to position 79 in the log file and start reading. This makes it quite effective as you can use binary search to quickly get to the correct offset in the already sorted index file.

Parallelism with Partitions

To guarantee the order of reading messages from a partition, Kafka restricts to having only consumer (from a consumer group) per partition. So, if a partition gets messages a,f and k, the consumer will also read them in the order a,f and k. This is an important thing to make a note of as **order of message consumption is not guaranteed at a topic level** when you have multiple partitions.

Just increasing the number of consumers won't increase the parallelism. You need to scale your partitions accordingly. To read data from a topic in parallel with two consumers, you create two partitions so that each consumer can read from its own partition. Also since partitions of a topic can be on different brokers, two consumers of a topic can read the data from two different brokers.

Topics

We've finally come to what a topic is. We've covered a lot of things about topics already. The most important thing to know is that **a Topic is merely a logical grouping of several partitions**.

A topic can be distributed across multiple brokers. This is done using the partitions. But a partition still needs to be on a single broker. Each topic will have its unique name and the partitions will be named from that.

Replication

Let's talk about replication. Whenever we're creating a topic in Kafka, we need to specify what the `replication factor` we need for that topic. Let's say we've two brokers and so we've given the `replication-factor` as 2. What this means is that Kafka will try to always ensure that each partition of this topic has a backup/replica. The way Kafka distributes the partitions is quite similar to how HDFS distributes its data blocks across nodes.

Say for the `freblogg` topic that we've been using so far, we've given the replication factor as 2. The resulting distribution of its three partitions will look something like this.

Kafka Partition distribution

Even when you have a replicated partition on a different broker, Kafka wouldn’t let you read from it because in each replicated set of partitions, there is a `LEADER` and the rest of them are just mere `FOLLOWERS` serving as backup. The followers keep on syncing the data from the leader partition periodically, waiting for their chance to shine. When the leader goes down, one of the `in-sync` follower partitions is chosen as the new leader and now you can consume data from this partition.

A Leader and a Follower of a single partition are never in a single broker. It should be quite obvious why that is so.

Finally, this long article ends. Congratulations on making this far. You now know most of what there is to know about Kafka’s data storage. To ensure that you retain this information let’s do a quick recap.

Recap

- Data in Kafka is stored in topics
- Topics are partitioned
- Each partition is further divided into segments
- Each segment has a log file to store the actual message and an index file to store the position of the messages in the log file
- Various partitions of a topic can be on different brokers but a

partition is always tied to a single broker

- Replicated partitions are passive. You can consume messages from them only when the leader is down

That ought to cover everything we've talked about. Thanks for reading. See you again in the next one.