Kafka

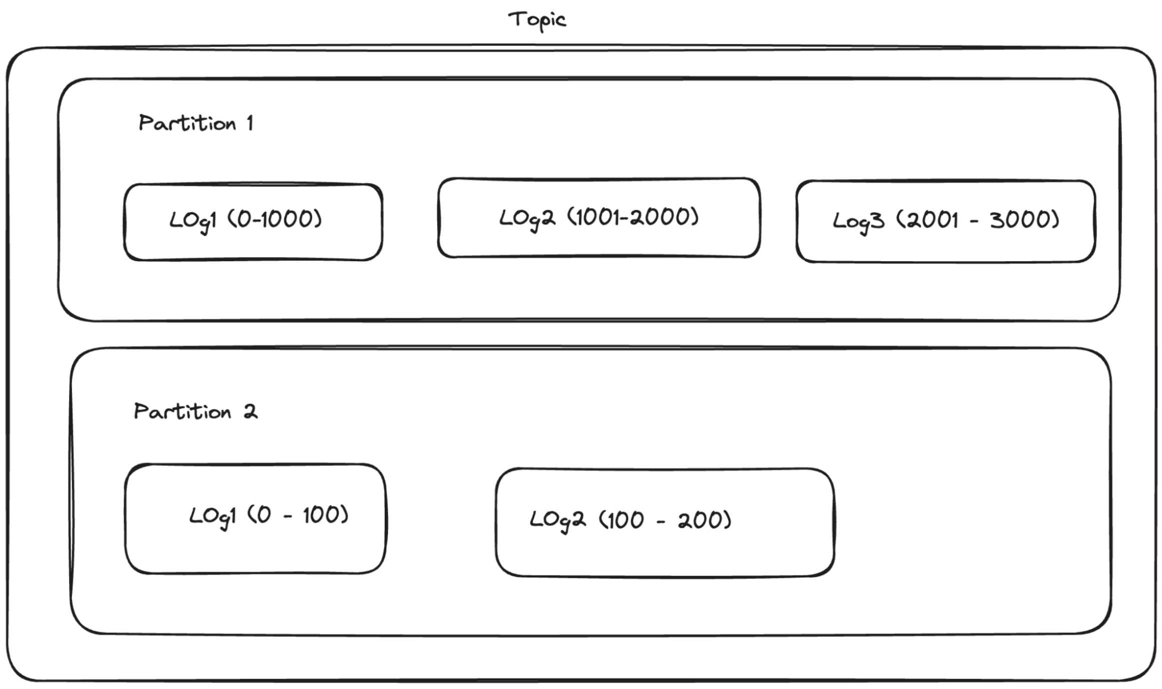
Apache Kafka is a distributed pub / sub platform used for building real-time data pipelines and streaming applications. Its architecture is designed for high throughput, fault tolerance, and scalability.

Concepts:

Topic: Topics are **the categories used to organize messages**. Each topic has a name that is unique across the entire Kafka cluster.

Partitions

- Each topic is divided into log segments called partitions. As the name suggests a log is append only file
- Partitions are the **primary unit of parallelism** in Kafka and allow for data to be distributed across a cluster.
- For each partition a directory is created which contains logs. Each log further contains two files `index and the data file`



Producers

- Producers send messages to Kafka topics. The target partition for each message is determined either by a key (using a partitioner) or in a round-robin manner if no key is provided.

Consumer group

- Consumers read messages from Kafka topics. They can be grouped to form a consumer group.

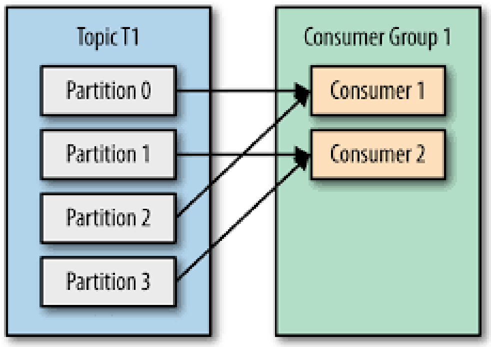
- A consumer in a consumer group can consume messages from single or multiple partitions depends upon the count of partitions and consumers in the group

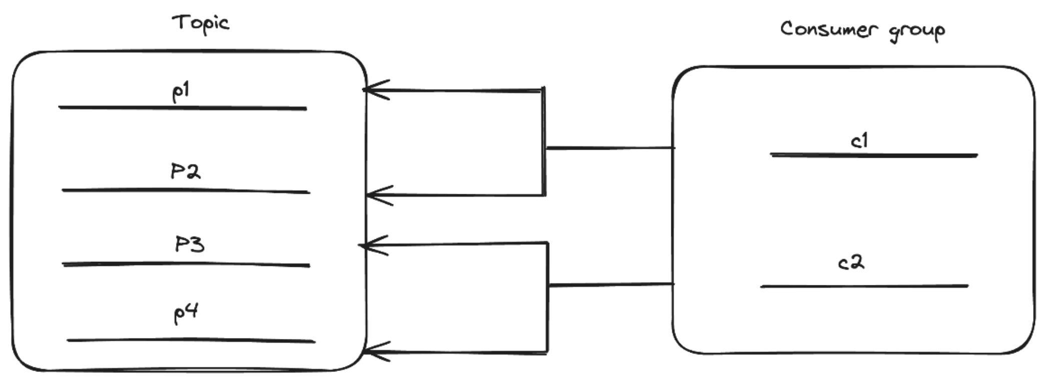
- Each partition in a topic can be consumed by only one consumer in a consumer group but a consumer can consume from multiple partitions.
  The latter is only possible when no of partitions are greater than no of consumer groups.
  On the other hand if no of consumers are greater than number of partitions then extra consumers will remain idle

  Some Examples:
  We have a consumer group having 2 consumers
  Scenario 1: Consuming from a topic having 4 partitions: In this case kafka will assign 2 Partitions to each consumer

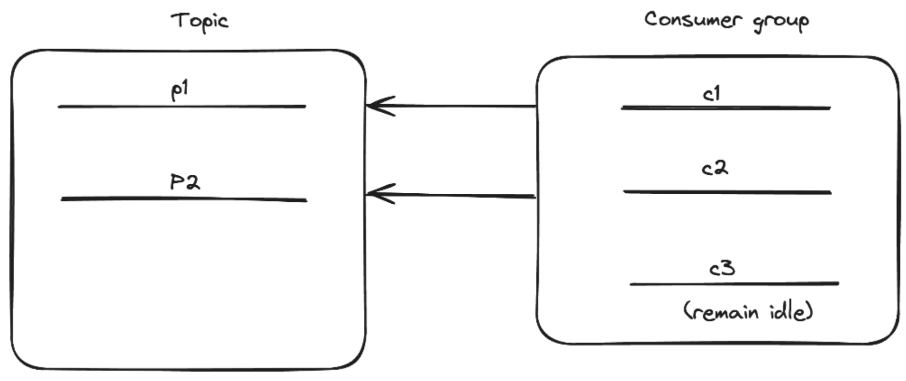
  Scenario 2: Consuming from a topic having 1 partition: Here kafka will assign one partition to consumer 1 and other one will remain in idle state

- Multiple consumer groups can be attached to same topic and kafka guarantees that a message will be delivered to all consumer groups attached to an topic once

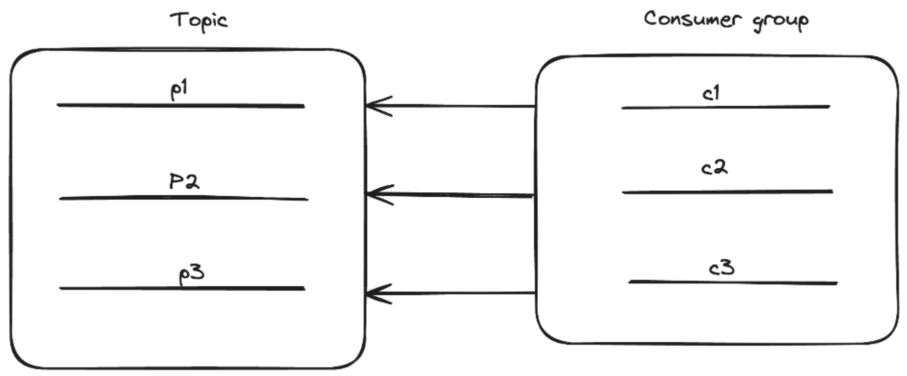




Case 1: No of partitions > no of consumers



Case 2: No of paritions < no of consumers



Case 3: No of paritions =. no of consumers in a cg

Replication and Leadership

- For durability and high availability, each partition can be replicated across multiple brokers.

- Among the replicas of a partition, one serves as the **leader**, while others act as **followers**.

- The leader handles all read and write requests for the partition. Followers replicate the data from the leader and can become leaders if the current leader fails.

Acknowledgments and Offsets

- Kafka uses an acknowledgment mechanism to ensure data integrity.
- The acknowledgment settings (`acks`) dictate how a producer knows if its message has been received and persisted by the broker(s).
- ack = 0: Producer doesn’t wait for acknowledgment
- ack = 1: Acknowledgment after the leader replica receives the message
- acks=all: Acknowledgment after all in-sync replicas receive the message

- Consumers commit their offsets back to Kafka, ensuring they can resume from where they left off after restarts or failures.

Throughput

As mentioned above partitions control the unit of parallelism or throughput in a kafka cluster. More partitions you have, the more consumers can consume parallelly which in turn increase the throughput

How many partitions can we create?

There are no hard limits on the number of partitions in Kafka clusters. But here are a few general rules: maximum 4000 partitions per broker (in total; distributed over many topics) maximum 200,000 partitions per Kafka cluster (in total; distributed over many topics) **in zookeeper mode.**

How to decide the ideal number of partitions?

A rough formula for picking the number of partitions is based on throughput. Let's say that your target throughput is **T** and per partition throughput is **C.** So you'll need at least **(T / C)** partitions.

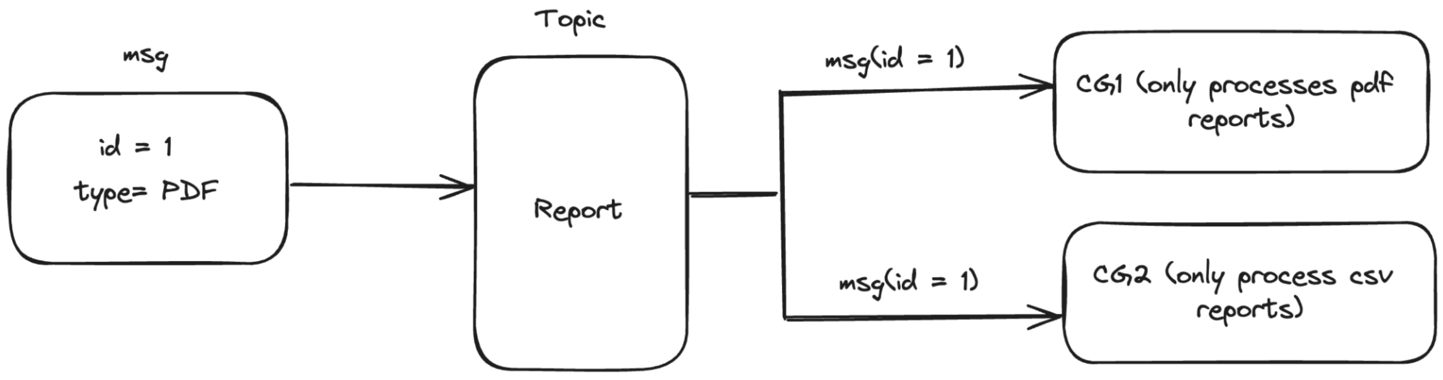
Things to consider while increasing the number of partitions:

Each partition maps to a directory in the file system in the broker. Within that log directory, there will be two files (one for the index and another for the actual data) per log segment. Currently, each broker opens a file handle of both the index and the data file of every log segment

More partitions may increase unavailability

When a broker is shut down the observed unavailability could be proportional to the number of partitions. Suppose that a broker has a total of 2000 partitions, each with 2 replicas. Roughly, this broker will be the leader for about 1000 partitions. When this broker fails uncleanly, all those 1000 partitions become unavailable at exactly the same time. Suppose that it takes 5 ms to elect a new leader for a single partition. It will take up to 5 seconds to elect the new leader for all 1000 partitions. So, for some partitions, their observed unavailability can be 5 seconds plus the time taken to detect the failure.

So is there any way we can increase the throughput without increasing partitions?
As we know that we can attach multiple consumer groups to a partition, all of them will receive the same message.


msg
id = 1
type= PDF
Topic
Report
msg(id = 1)
CG1 (only processes pdf reports)
msg(id = 1)
CG2 (only process csv reports)

Both CG1 and CG2 are attached to Topic Report

Now we have seen that we can attach multiple consumer groups to increase the throughput but can we further increase the throughput?

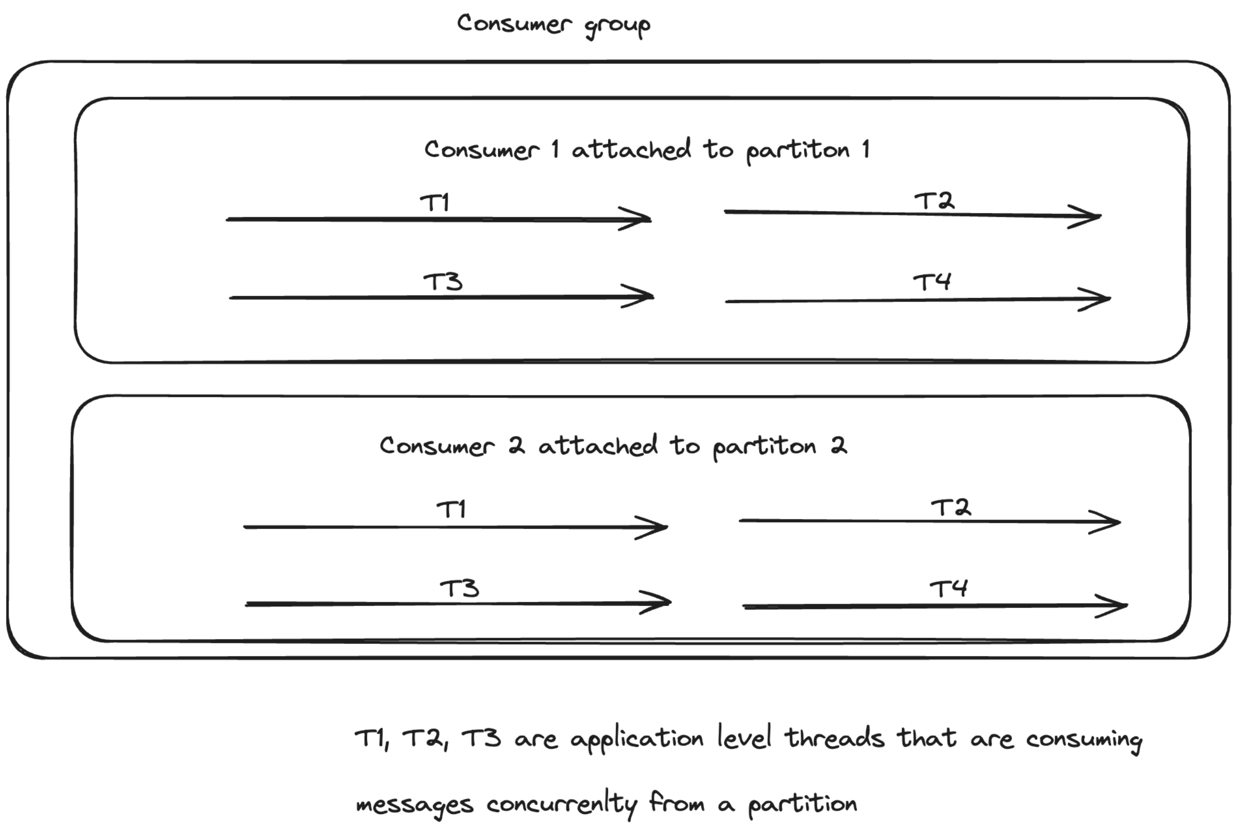
The **answer is yes**. We can make use of the application concurrency. By default client libraries create one thread / virtual thread that consumes messages from a partition. We can take this further by creating a thread pool for each partition that concurrently consumes messages.

For example: Topic with 2 partitions. Consumer group cg1 with 2 consumers attached to the topic.

Without application level concurrency: we have 2 consumers attached to one partition each

With application level concurrency (Thread pool = 2): From **kafka pov** we still have 2 consumers but instead of one thread, a thread pool is consuming messages from each partition.

How is it beneficial? If our per partition throughput is 10 msg / sec. With a thread pool of 2 we have increased throughput from 10 - 20 (approx). Actual number depends on a lot of other factors.


Consumer group
Consumer 1 attached to partiton 1
T1
T2
T3
T4
Consumer 2 attached to partiton 2
T1
T2
T3
T4
T1, T2, T3 are application level threads that are consuming messages concurenlty from a partition

Combination of both (thread pool and multiple consumer group) we can easily scale the overall throughput for a topic in kafka