1. Text

   Description automatically generated  
   
2. We already learnt about the Producer serialization.  
   The next thing is partitioner.
3. **Agenda**:
   1. Partitioner.
4. 
5. Before Proceeding further, just take a look at the following snapshot.   
   Chart, waterfall chart

   Description automatically generated
6. Every ProducerRecord includes the mandatory value Topic name as the destination address of the data.
7. However, the Kafka Topic is partitioned.
8. So, the Kafka Producer should decide on which Kafka Partition, the msg should be sent.
9. There are two approaches to specify the target partition# for the msg.
10. There are three ways actually to define the partition#.
    1. Setting the partition# in the ProducerRecord itself.  
       Diagram

       Description automatically generated with medium confidence
    2. You can supply your own partitioner that implements your desired partitioning strategy and assigns **partition# to each msg at runtime**.  
         
       This Java Object Properties is passed to the KafkaProducer when we instantiate it.
    3. However, even creating the custom partitioner is often not needed. Why?  
       The KafkaProducer comes with a **default** **partitioner** which is the most commonly used partitioner.  
       The default partitioner takes one of the two approaches to determine the destination topic partition.
       1. **Hash Key Partitioning**:
          1. This approach is based on the msg key in ProducerRecord object.
          2. When the msg key exists, the default partitioner will use the **hashing** algorithm on the key to determine the partitioner#.
          3. It is as simple as hashing the key to convert into a numeric value. Then use the hash number that we get from hashing to deduce the target partition#.
          4. The hashing ensures that all the msgs with the same key goes to the same partition#.
          5. This algo also takes the total number of partitions as input for module purpose with the hash# we get from above step.  
             So, if you increase the number of partitions in the topic, the default partitioner will start giving a different output.  
             That means if the partition is based on the key, then you should create a topic with enough partitions and should not increase it at the later stage.
          6. You can easily overprovision the number of partitions in the topic.   
             I mean if you need hundred partitions, you can easily overprovision it by 25% and create 125 partitions.  
             There is not no harm in taking this approach.  
             But remember, if you do increase the number of partitions later, you may have to redistribute the existing msgs.
          7. When the msg key is null, the default partitioner will take the approach of round-robin to achieve an equal distribution among the available partitions.
       2. **Round Robin Partitioning**:
          1. When the msg key is null, the default partitioner will take the approach of round-robin to achieve an equal distribution among the available partitions.
          2. That means the 1st msg goes to the 1st partition and 2nd to another partition and the partitioner repeats it in a loop.
11. The default partitioner is the most commonly used partitioner in most of the use cases.
12. However, Kafka allows us to implement our own partition strategy by implementing a **CustomPartitioner** class.
13. But you may not need a custom partitioner in most of the cases.  
    Chart, waterfall chart

    Description automatically generated