1. Graphical user interface, text

   Description automatically generated
2. 
3. We learnt that Kafka is by default **At Least Once System** but we can configure it to get **At Most Once**.
4. However, some use cases want to implement **Exactly Once Semantics**.  
   It means we don’t lose anything and at the same time we don’t want any duplicate msgs.
5. 
6. To meet **Exactly Once Requirement**, Kafka offers an **idempotent producer configuration**.
7. All you need to do is to enable **idempotent** and Kafka cares of implementing **Exactly-Once**.  
   **enable.idempotence=true**
8. Once we configure this, the behavior of Producer will change.  
   There are many things happen internally.  
   But at high-level, Producer API will do the two things.  
   A picture containing circle

   Description automatically generated
9. **Internal ID for Producer Instance:**
   1. It will perform an initial handshake with the Leader Broker and ask for Unique Producer ID.
   2. At the Broker side, it dynamically assigns a unique ID to each Producer.
   3. The next thing that happens is the msg sequencing.  
      The Producer API will start assigning a sequence number to each msg.  
      This sequence number starts from zero and monotonically increments per partition.
   4. Now when the I/O thread sends a msg to the Leader, the msg is uniquely identified by the Producer ID and Sequence Number.  
      A picture containing timeline

      Description automatically generated
   5. Now the Leader Broker knows that the last committed msg sequence from a particular producer is X then next msg seq# should be X+1.
   6. This allows the Leader Broker to identify the duplicates as well as missing seq numbers.
   7. So, setting enable.idempotent=true will help you ensure that the msgs are neither duplicated nor lost.
   8. How exactly that happens is not relevant and we leave that on Producer API and Broker.
   9. However, you must always remember one thing.  
      If you’re sending duplicate msgs at app level, this configuration can’t protect you from duplicates.  
      That should be considered as a bug in your app even if two different threads or two different Producer Apps are sending duplicate msgs.  
      That is app design problem.  
      The idempotent is only guaranteed for the Producer retries and you should not try to send the same msg at the app level.  
      The idempotent is not guaranteed for the app level msg resends or duplicates.