1. Graphical user interface, text

   Description automatically generated
2. In the earlier lectures, we learnt to create Kafka Producer and now we’re comfortable to meet most of the basics requirements of Streaming events to the Kafka Cluster.  
   However, some specific and intricate scenarios require some extra attention.
3. **Agenda**:
   1. Some Advanced Producer Concepts.
4. 
5. Apache Kafka provides msg durability guarantee by committing the msg at the partition log.
6. Durability means once the msg is persisted in the leader partition by the leader Broker, we can’t lose the msg till the Leader Broker is alive.

However, if the leader Broker goes down, we can lose the msg.  
To protect the loss of the msg due to leader Broker failure, the Kafka implements the replication right we know that.  
Kafka implements replications using followers.  
Followers can copy the msg from the Leader and provide the fault-tolerance in case of Leader Broker failure.

In other words, when a msg is persisted in the Leader as well as the followers in the ISR list, we consider the msg to be fully committed.  
Once the msg is fully committed, we can’t lose the msg until the Leader & all the follower replicas are lost which is unlikely case.

1. But in all this, we still have the possibility of committing duplicate msg due to the producer retry mechanism.
2. As we learned in the earlier section, if the Producer I/O thread fails to get a successful acknowledgement from the Leader Broker, it will try to send the same msg again.
3. Assume that the I/O thread transmits a msg to the Leader Broker.  
   The Leader receives the msg and stores it into the partition log.  
   Then the Leader Broker sends an acknowledge for the success but the response doesn’t reach back to the I/O thread due to a network error.  
   In that case, the I/O thread waits for an acknowledgement and ultimately resend the same msg again assuming some failure.  
   The Leader Broker receives the msg but it doesn’t have any mechanism to identify that the msg is a duplicate of an earlier message.  
   Hence, the Leader Broker saves the duplicate msg causing a duplication problem.  
   This implementation is known as **At Least Once Semantics** where we can’t lose msg because we’re retrying until we get a success acknowledgement.  
   However we may have duplicate msgs as per this implementation as we don’t have any mechanism to identify the duplicate msgs.  
   For that reason, Kafka is said to provide **At Least Once Semantics**.
4. Kafka also allows you to implement **At Most Once Semantics**. How?  
   A picture containing chart

   Description automatically generated

We learnt that Kafka is by default **At Least Once System** but we can configure it to get **At Most Once**.