1. A picture containing diagram

   Description automatically generated
2. Text

   Description automatically generated
3. Text

   Description automatically generated with medium confidence
4. **Producer**:
   1. A producer is an app that sends **data** also called **message** or **message record**.
   2. Ultimately, it is a small to medium sized piece of data.  
      The message record may have a different meaning and **schema or record structure** for us but for Kafka is simply an **array of bytes**.  
      **For example**:   
      A picture containing text

      Description automatically generated  
      A picture containing chart

      Description automatically generated A picture containing diagram

      Description automatically generated   
      So, basically, you need to create an appropriate producer app as per your requirement.   
      You may find a producer app out of box which fits your requirement.
5. **Consumer**:
   1. A consumer is an app that receives data. So basically, it is a recipient but producer sends data to **Kafka Server** and directly to recipient and any consumer interested in the data, can come forward and consume it.
   2. Consumer can ask for data sent by any producer provided it has read permission.   
      Icon

      Description automatically generated
6. **Broker**:
   1. It is Kafka Server.
   2. This name makes sense as Kafka Server is acting as message broker b/w producer & consumer.  
      It means producer & consumer don’t interact directly but they use Kafka Server as an agent or a broker to exchange messages.
7. **Cluster**:  
   Graphical user interface, application

   Description automatically generated
   1. If you have any background in distributed system, you already know a cluster is **a group of computers** **acting together for a common purpose**.
   2. Kafka is **distributed system** so cluster means the same thing for Kafka.
   3. So, **Kafka cluster is a group of computers, each running one instance of the Kafka Broker**.
8. **Kafka Topic**:
   1. We already studied that producer will be sending data to the broker and the consumer will be asking for data from the broker but what data or which data?
   2. Let’s see the following conversation.   
      **NOTE**: Apart from Smart Meter Producers, there may be other kinds of producers too.Diagram

      Description automatically generated with medium confidence  
      Diagram

      Description automatically generated with medium confidence  
      Diagram

      Description automatically generated with low confidence Diagram

      Description automatically generatedGosh!!! That is too much confusion. We must have some standard methods to cut down these back & forth questioning & that is where we have the notion of topic.
   3. So, a topic is an **arbitrary name** given to a **data set**.   
      You better say, it is **a unique name for a data stream**.  
      You can think of topic as a Database Table.  
      Creating a topic is a **Design Time Decision.**So, when you’re designing your app, you as an architect are responsible for creating one or more topics.  
      Once topic is there, the producer and consumer are going to send and receive data by the topic.  
      Now see the simplified communication b/w producer and broker and b/w broker and consumer.  
      Diagram

      Description automatically generated with low confidence
9. **Partitions**: What are topic partitions?
   1. By now, you learnt that data will be stored in a topic and topics are just like DB Table.
   2. This data could be **massive** so it could be larger than the storage capacity of a single computer.  
      Think about hundreds or thousands of smart meters sending data every seconds and within a few hours, you’re going to collect several terabytes of data.  
      So, broker is going to face a storage capacity challenge.
   3. **Solutions**:  
      Diagram

      Description automatically generated
      1. One obvious solution is to break the **Topic** into smaller parts and distribute it over multiple computers in **Kafka Cluster**.  
         We already learn that Kafka is distributed system, and it runs on a cluster of computers.  
         So, it is evident that Kafka can break a topic into smaller partitions and store those partitions on different machines.  
         This approach can solve the storage capacity problem and that is where the **Topic Partitions** means **🡺 “A Small and Independent portion”** of the topic.  
         **Question**: How can Kafka decide the number of partitions of a topic?  
         **Answer**: Some **partitions** can be reasonably large and others can be comparatively small.   
         **For example**: Current Load topic may be receiving data every minute so large topic & consume-unit topic may get data every hour so smaller topic.  
         The number of partitions for a topic is a **Design Decision** by an architect not by **Kafka** itself.  
         **Partition is the smallest unit** which can’t be broken further so sitting in a single machine unlike a topic. So, estimation for the number of partitions for a topic is by you.
10. **Partition Offset**:
    1. **Def**: A unique **Sequence ID** of a message in a partition.
    2. Assigned by **Broker** to every **message record** as it arrives in the partition 🡺 **Immutable**.
    3. So, when a first message is stored in a partition, it gets an offset ID as Zero & the next one gets one and so on.  
       This sequencing means that the Kafka stores messages in the partition in the order of arrival.
    4. So, offset ID is clearly an **arrival order number**.
    5. **NOTE**: These offsets are local within the partitions.  
       There is no global ordering in the topic across the partitions.  
       See the diagram below, in each partition related to the same topic, the first msg starts from zero.  
       So, if you want **to locate a specific message**, you must need **three things**
       1. Topic Name.
       2. Partition Number.
       3. Offset Number.

A picture containing diagram

Description automatically generated

1. **Consumer Group**:
   1. A group of consumers.
   2. Multiple consumers form a group to share a workload.
   3. Suppose, you have a massive work and you want to divide it and distribute the parts among multiple consumers.  
      So, you’re going to create a group of consumers and the members of the same group are going to share the burden and accomplish the bigger task together.
   4. **Example**: From retail industry
      1. A picture containing icon

         Description automatically generated
      2. So basically, you have multiple billing counters and you want to bring all invoices from each billing counter to your data center.
      3. Since, you know Kafka so you know Kafka is excellent solution to transport data from your billing locations to your data center and you have decided to implement it.
      4. The first think you may want to do is **to create a producer** at each billing location and the producer is going to send the invoices as messages to the Kafka Topic.
      5. Next thing you want to do is **to create a consumer** which is going to read data from Kafka topic and write it to the Data Center.
      6. **Note**: This looks like a **perfect solution** but there is a **small problem**.  
         We have 500 producers producing a lot of data and just one consumer.  
         Diagram

         Description automatically generated with medium confidence