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 a **message** or a **message record**.
   2. Ultimately, it is a small to medium sized piece of data.
   3. The message record may have a different meaning or **schema or record structure** for us but for Kafka, it 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** not 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
   3. The consumer keeps asking for new msg and the server keeps giving new records & this goes in a circle as long as new msgs are arriving in the Kafka Server.
   4. Now it is up to the **consumer app** on how to use the data.
      1. It might do some aggregates.
      2. It might produce some alerts.
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 related to **Smart Electricity Meter**.  
      Diagram

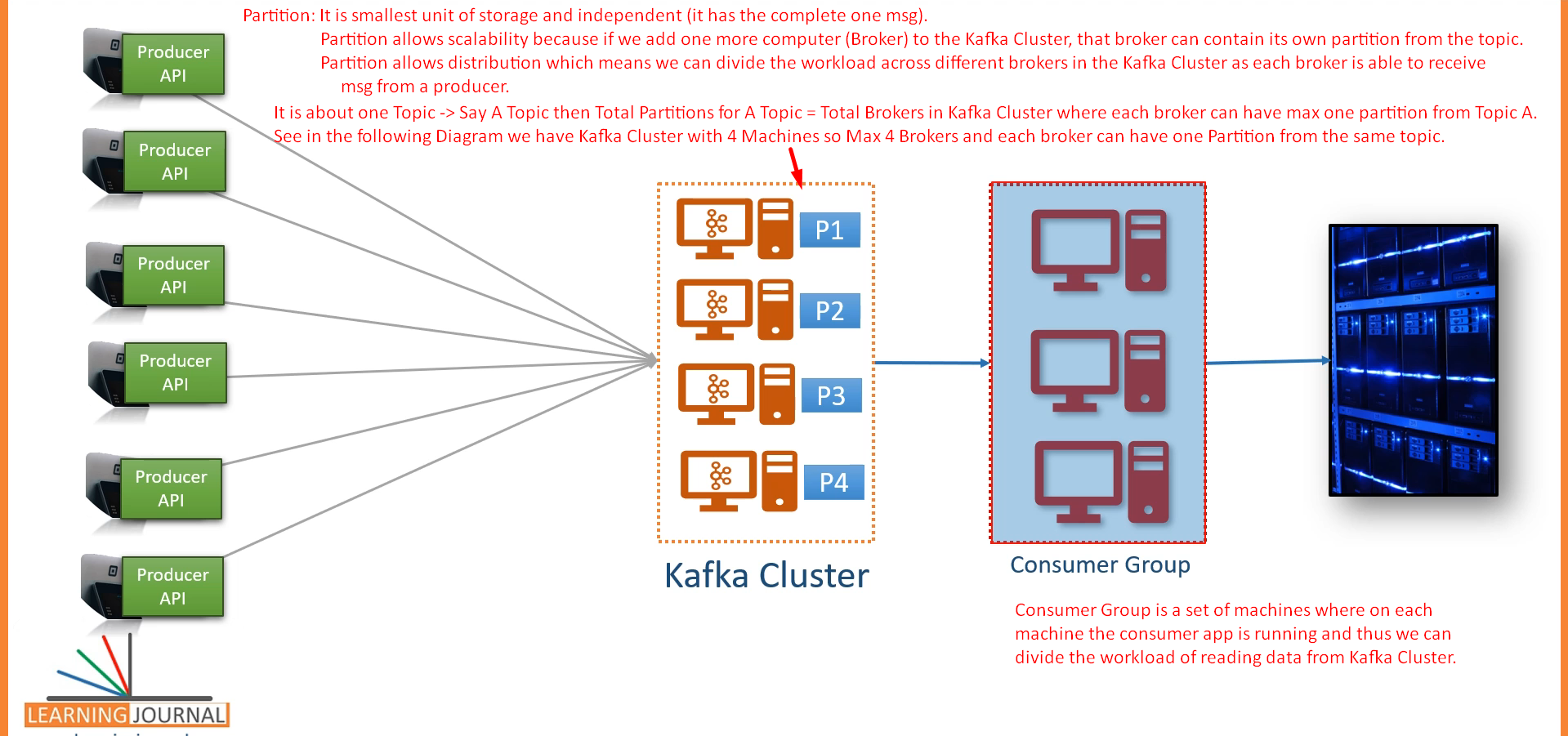
      Description automatically generated with low confidence  
      You can consider topic as Table in DB World.   
      Producer sends the msgs to a table hence topic.   
      Consumer reads the msgs from the table hence topic.
9. **Partitions**: What are topic partitions?
   1. By now, you learnt that 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 second 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. We already learnt that Kafka is distributed system & it runs on a cluster of computers.
      2. One obvious solution is to break the **Topic** into smaller parts and distribute it over multiple computers in **Kafka Cluster**.  
         We already learnt 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 parts called **Partitions** and store those partitions on different machines in **Kafka Cluster**.  
         This approach can solve the storage capacity problem and that is where the **Topic Partitions** mean **🡺 “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.  
         So, the question is how Kafka will know that **Current-Load Topic** needs 100 partitions whereas **Consumed-Units** needs just 20 partitions.  
         **Answer:** The number of partitions for a topic is a **Design Decision** by an architect not by **Kafka** itself.  
         At the time of topic creation, we need to specify the numbers of partitions and Kafka Broker will produce it.  
         **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 automatically 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.  
       So, if you have three partitions in a topic, within each partition, the offset is going to start from zero & increases by one.  
       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. **Purpose**: Multiple consumers form a group to share a **workload**.
   3. Suppose you have a massive work and you want to divide it and assign the parts among multiple consumers.  
      So, you’re going to create a group of consumers and the members of the 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 thing 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**.  
         **Just think of the scale. 😊**  
         We have 500 producers producing a lot of data into a single topic and just one consumer.  
         So, how do you handle **volume (input msgs) & velocity**?
      7. But you learnt Kafka exceptionally well.   
         So, you decided to create a large Kafka Cluster & partition your topic.   
         Now, your **topic is partitioned & distributed** across the **cluster**.  
         Now, every **broker** in the cluster has a topic partition & it can take data from the producer and store in its partition.   
         On the Data Source side, you have hundreds of Producers and a bunch of Brokers to share the **workload**.  
         So, **Topic Partitions have two advantages**:
         1. It helps to increase the storage capacity.
         2. It also helps to reduce the workload across the group of Brokers (basically Kafka Cluster).
      8. Kafka Topic Partition is the core idea of making Kafka as a Distributed and a Scalable System.
         1. **Distributed**: As the msgs can be distributed across the partitions on different machines.
         2. **Scalable**: As Topic after partitions can reside on different machines.   
            By adding more brokers, we get more storage and producers can send msgs to more brokers and a consumer can read msgs from more brokers.  
            **NOTE**: Partition resides inside Broker and Broker resides on a machine & Kafka Cluster may have more than one machines so more than one Broker.   
            Actually, each machine is a Broker in the Kafka Cluster.
      9. Till now we studied from the Left Side (Left Side = Producers + Brokers (Kafka Cluster))
      10. Let’s talk about the Right Side = Consumer.
          1. See, we have a lot of incoming data.   
             How is a consumer going to handle that amount of data?
          2. Here comes the **Consumer Group**.
          3. Start multiple copies of Consumer application in the same group & let them divide the workload.
          4. How is workload divided among the consumers?
             1. Let’s say we have 500 topic partitions.
             2. And we’re starting 100 consumers in a group.
             3. So, each consumer will take 5 partitions & all together they will be processing 500 partitions.
             4. The maximum number of consumers is limited by the number of partitions.  
                Kafka doesn’t allow more than one consumer from the consumer group to read and process data from the same partition simultaneously.  
                This restriction is necessary to avoid the double reading of a record.  
                NOTE: Following diagram says that a broker can have one partition from a topic but actually, a broker can have any number of partitions from a topic. No issue.  
                **NOTE**: If two partitions, still more than two consumers in a consumer group are allowed but one consumer will sit idle because Kafka assigns one partition to only one consumer.
      11.   
          Diagram

          Description automatically generated with medium confidence