1. Text

   Description automatically generated with medium confidence
2. 
3. If you look at the official definition:   
   **What does it mean?**

It means two things.

A picture containing text

Description automatically generated

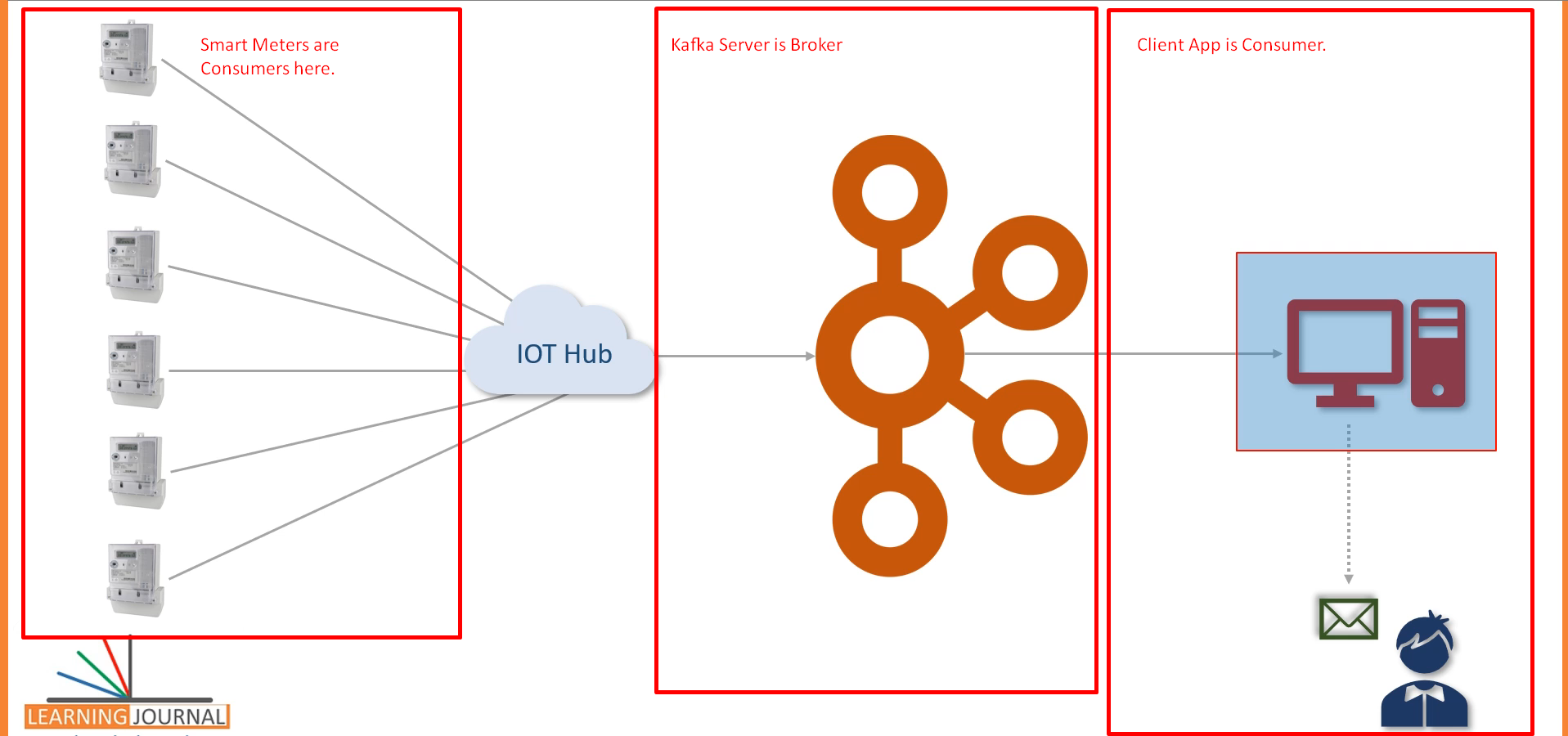
1. **Real Life Example**:  
   Text

   Description automatically generated with low confidence
   1. Suppose Each house has a **smart electricity meter** which is **generating load related data every minute**.  
      This data goes to **Kafka Server** somehow.  
      So, Kafka Server continuously receives this data every minute from each smart electricity meter **resulting in continuous stream of data**. This is what we call **continuous stream of data**.  
      Bringing data from your smart meter to Kafka server is what we call **creating real time Streams**.  
      You have also created an **app** to read and process this data.   
      App can be doing a lot of things.   
      Such as computing and monitoring the overall electricity load for every house.  
      As soon as the charge (load) goes above a predefined threshold, (Let’s say 2KVA), you are sending an SMS to the house owner to alert him, but such kind of alert doesn’t make any sense after an hour. You want to send such alerts in seconds or at least in 2-3 minutes.  
      So, your app is **not accumulating** **data and then processing** it to compute if the threshold is broken.   
      It must be continuously listening to the data and processing it as soon as it arrives at the Kafka’s Server. That is what we call **real-time stream processing**.  
      So, after studying all this, the following is the definition of Kafka.  
      Graphical user interface, text, application

      Description automatically generated
2. 
3. **How does it work?**
   1. Kafka adopted **Pub-Sub Messaging System Architecture** and it works as an **Enterprise Messaging System**.
   2. A typical messaging system has three components.
      1. Producer.
      2. Consumer.
      3. Broker.

Graphical user interface, application

Description automatically generated

* 1. 
  2. This is 5000feet view of how Kafka works but it answers your question.
  3. Kafka works as **Pub-Sub Messaging System** where we create a **Producer App** to create and send data **as a stream**.   
     We install and configure **Kafka Server** to act as **a message Broker**.  
     Finally, we create **consumer app** to process the **data stream** in **realtime**.

1. 
   1. Kafka was initially conceptualized and developed at **Linkedin** and later on **open sourced** in 2011.
   2. However, in the beginning, Kafka was designed to handle the **data integration problem**.
   3. Let’s try to understand the **LinkedIn problem**.
   4. The below diagram was borrowed from **Jay Krep’s blog** who was leading Kafka at **LinkedIn**.

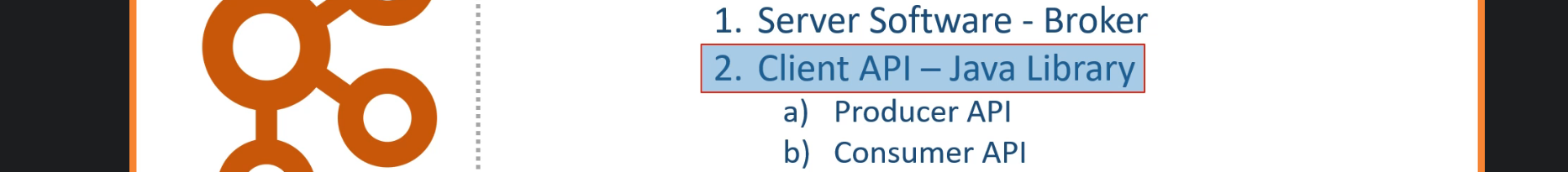
Now, he is CEO of **Confluent** Inc.  
A picture containing text, appliance, screenshot, stove

Description automatically generated

* 1. So, this diagram explains the **Data Integration problem** in large enterprise.
  2. All the small boxes (rectangular or Cylindric) in this diagram are **backend app** which generate and store some data.
  3. However, data generated by one app is needed by another app.  
     So, you have requirement to send data from one app to another app.
  4. The lines are showing data movement **from source app to destination app**.
  5. In growing organization, the number of source and destination systems keeps growing.  
     So, you will end up in a situation like this:  
     A picture containing text

     Description automatically generated
  6. Some parts of these **pipelines** will keep breaking every day and situation becomes like a big mess to maintain.
  7. LinkedIn tried to solve this problem Using **Pub/Sub Messaging Architecture** and the result looked like this.  
     Diagram

     Description automatically generated
  8. They put a broker in the middle and other apps started **working as producer and consumer**.  
     This architecture offered a bunch of advantages and simplified the solution. We will see later on.  
     But you got the answer to your question where did it come from?

1. **How Kafka evolved?**  
   
   1. How has it evolved from a Data Integration Solution to a Streaming Platform?
   2. Initially, Kafka started with two things.  
      
      1. **Server Software – Broker**: Server Software which we install and configure to work as a message broker.
      2. **Client API** – **Java Library**: A Java-Based Client API Library to help with the following
         1. To create Kafka Producer App.
         2. To create Kafka Consumer App.
   3. Later on, Kafka aspired to become a full-fledged real-time streaming platform and to achieve that objective, they augmented Kafka with three more components.
      1. **Kafka Connect.**
      2. **Kafka Streams.**
      3. **KSQL.**

Text

Description automatically generated **Kafka Connect & Kafka Streams** are open source and available with Apache 2.0 License  
however **KSQL** is available with some **licensing restrictions** and offered by confluent as a commercial tool.

* 1. To answer your question, from 2011 to 2019, Kafka evolved as a set of five components.
     1. **Kafka Broker**: Central Server System.
     2. **Kafka Client API**: Producer & Consumer API Library.
     3. **Kafka Connect**: Which addresses initial data integration problem for which Kafka was initially designed.
     4. **Kafka Stream**: This is another library to create **real-time stream processing app**.
     5. **KSQL**: With KSQL, Kafka is trying to become **a real-time Database and capture** some market share in DB and **DW/BI Space**.   
        Text

        Description automatically generated

1.   
   Timeline

   Description automatically generated
   1. By adopting Pub/Sub semantics, Apache Kafka is becoming the **circulatory system of your data ecosystem**.
   2. In this case, just like your circulatory system carries blood, Kafka brings data to various members of the infrastructure.
   3. Kafka occupies a central place in your real-time data integration infrastructure.
   4. The **data producers** can send data like messages to broker as quickly as the business event occurs &   
      **data** **consumers** consume the data from the broker as soon as it arrives.
   5. With careful design, the messages can reach from producer to consumer in milliseconds.
   6. The producers and consumers are completely decoupled, and they don’t need any kind of coupling or direct connections.
   7. They always communicate with the Kafka Broker using a **consistent interface**.  
      Producer is not worried about who and how many consumers are reading data.
   8. Producers & consumers can be added, removed, and modified as the business case evolves.