**Apache Kafka Tutorial**

**Topics**

1. Need of Messaging system

2. What is Kafka?

3. Kafka Features

4. Kafka Components

5. Kafka architecture

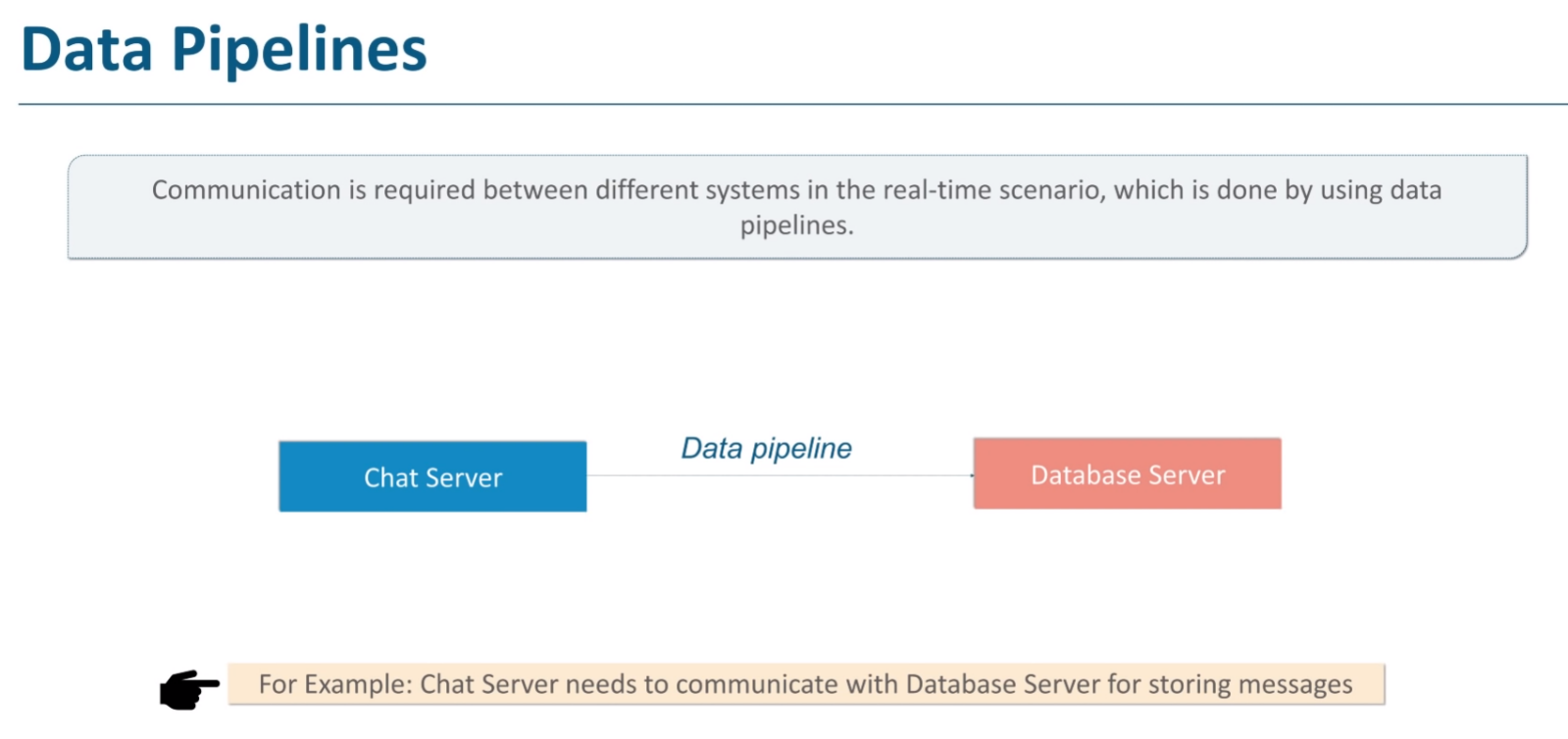
6. Installing Kafka

7. Working with Single Node Single Broker Cluster

**1. Need of Messaging system**

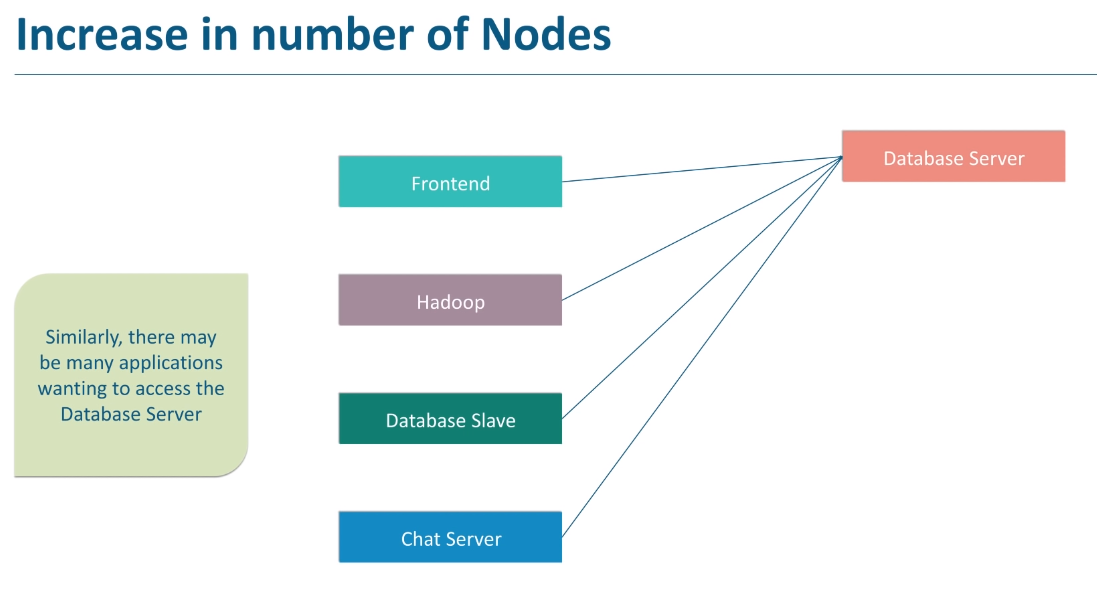
First we need to understand what is Data Pipelines

> In real time scenario we have different systems or services which will be communicating with each other and the data pipelines are the one’s which are establishing the connection b/w 2 systems or services.

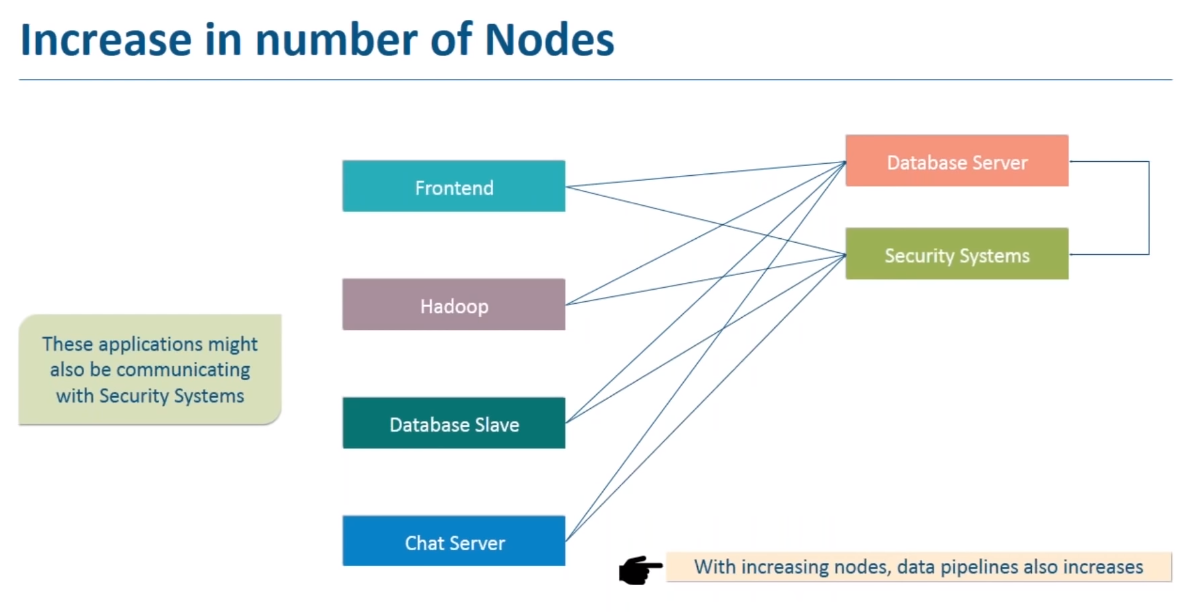


Now you can have different types of servers instead of Chat server or Database server like Web server, Application server, FTP server etc.

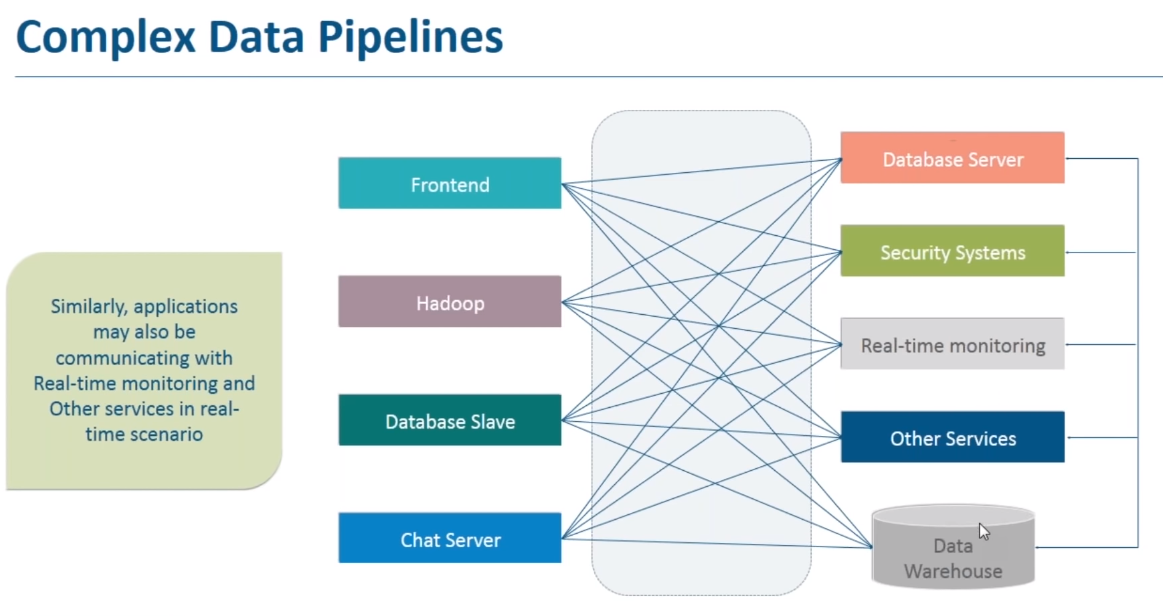
> If it would have been 20th or early 21st century, your organization can go ahead with 2 or 3 servers & fulfill their requirements. But now today, the organization can’t survive with 2 or 3 servers…. Moving ahead with time, server also started increasing.



For e.g. In Ecommerce scenario, where it can have multiple servers in front end like Application server, Hadoop server, Chat server, Payment server etc. Now all these servers want to communicate with Database Server so will have multiple data pipelines connecting all of them to Database server.



Similarly, organizations can have several servers at backend which will be receiving messages from different servers based on the requirements. They can have security systems for user authentication or authorization, Real time monitoring system which will gather data from various servers in real time & then show predictions for users, Data Warehouse where they can be dumping all their data for further analysis using various ETL or BI tools like Informatica etc.



> As you can see the data pipelines are getting complex with the increase in number of systems. So, adding a new system or server requires more data pipelines which will again make the system flow more complicated.

> Managing these data pipelines also become very difficult as each data pipeline has its own set of requirements.

For e.g., Data pipeline which handles Transactions should be Fault Tolerant & Robust. Organizations can’t miss any transaction on the other hand Clickstream data pipelines can be more fragile, then again data pipelines from a Catalog server to a Web server needs to be more reliable.

> Adding some pipeline or removing some pipeline becomes more difficult from the complex system.

This was the problem due to which Messaging system was originated.

> Messaging System reduces the complexity of these data pipelines & makes the communication b/w systems simpler & manageable.

> So, using Messaging Systems, you can easily establish remote communication & send your data easily across a network.

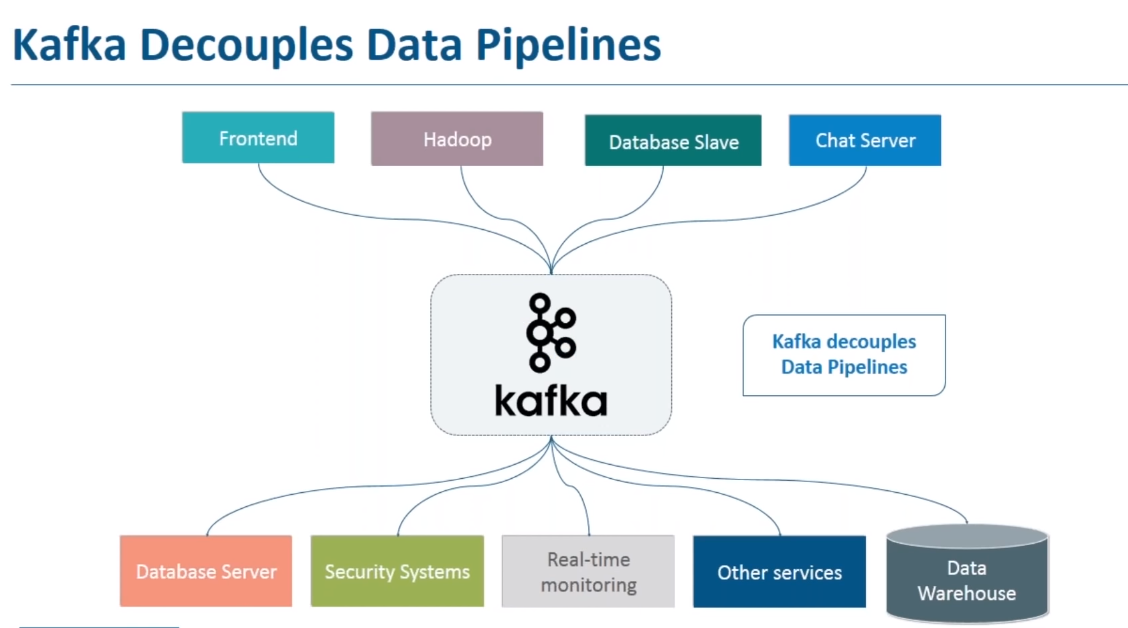
> Different systems use different platforms & languages but Messaging system provides you a common paradigm independent of those platforms & languages.

> You can also establish an asynchronous communication & send messages so that the sender doesn’t have to wait for the receiver to process the messages.

> You can also ensure the reliable communication even when the receiver or network is not working properly, your message should not get lost.

**2. What is Kafka?**

Let’s see How Kafka solves the Problem



> Kafka decouples the data pipeline & solves the complexity problem.

> The applications which are producing messages to Kafka like Frontend server, Hadoop server, Database server, chat server are called **Producers**.

> The applications which are consuming those messages from Kafka like Database server, Security systems, Real – time monitoring etc.. are called **Consumers**.

> So, **Producer** sends the data to Kafka, Kafka stores those messages & **Consumers** who want those messages can subscribe & receive them.

> You can have multiple subscribers to a single category of messages.

For e.g. The record generated by Frontend Server may be required by your Database server, Security systems etc. So

using Kafka, it becomes very easy for the consumer to consume the messages generated from a single producer.

> Also adding a new Consumer is also very easy, just subscribe to the message categories that’s required.

Note: Let’s understand it in simpler analogy – Kafka works similar to Radio i.e., different radio stations broadcast their messages, user can go ahead & listen to any channel based on his preferences. Multiple people can listen to the same channel.

Here Radio stations are the Producers, broadcasting towers are the Kafka clusters & subscriber are the consumer.

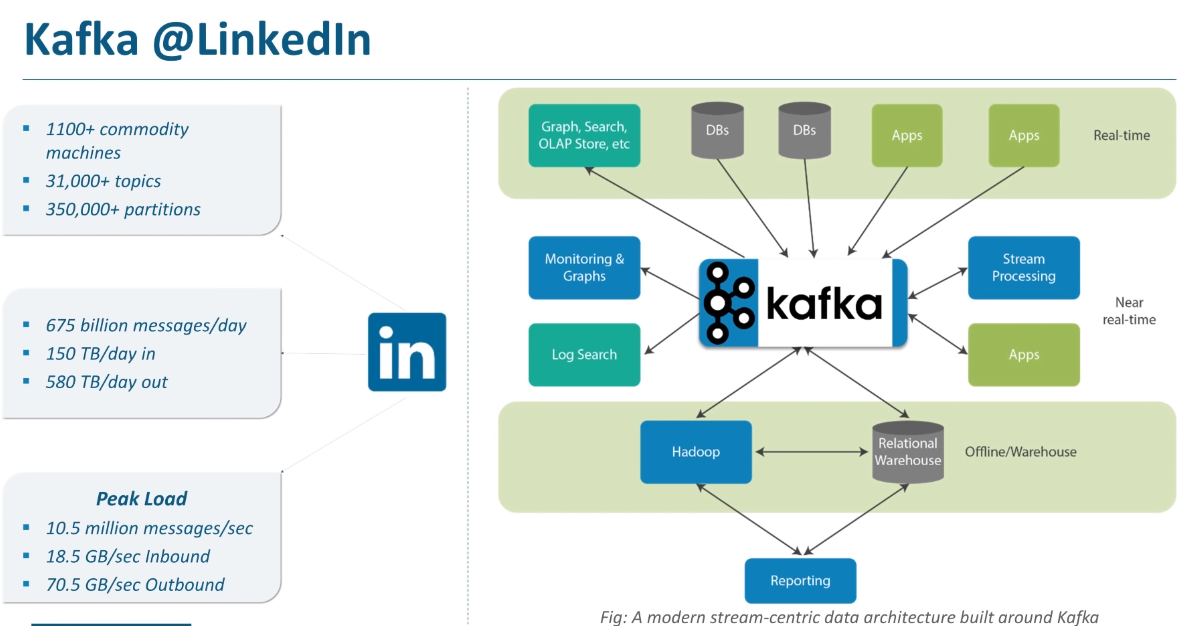


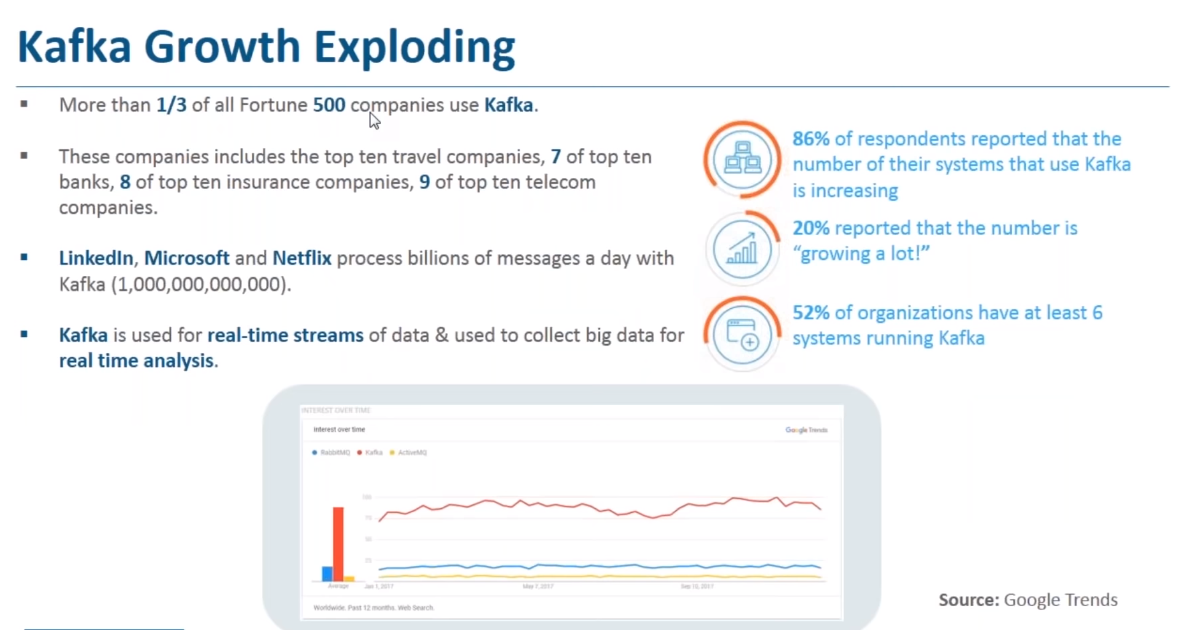
> Messaging traditionally has 2 models: - Queuing & Publish – Subscribe (pub/sub).

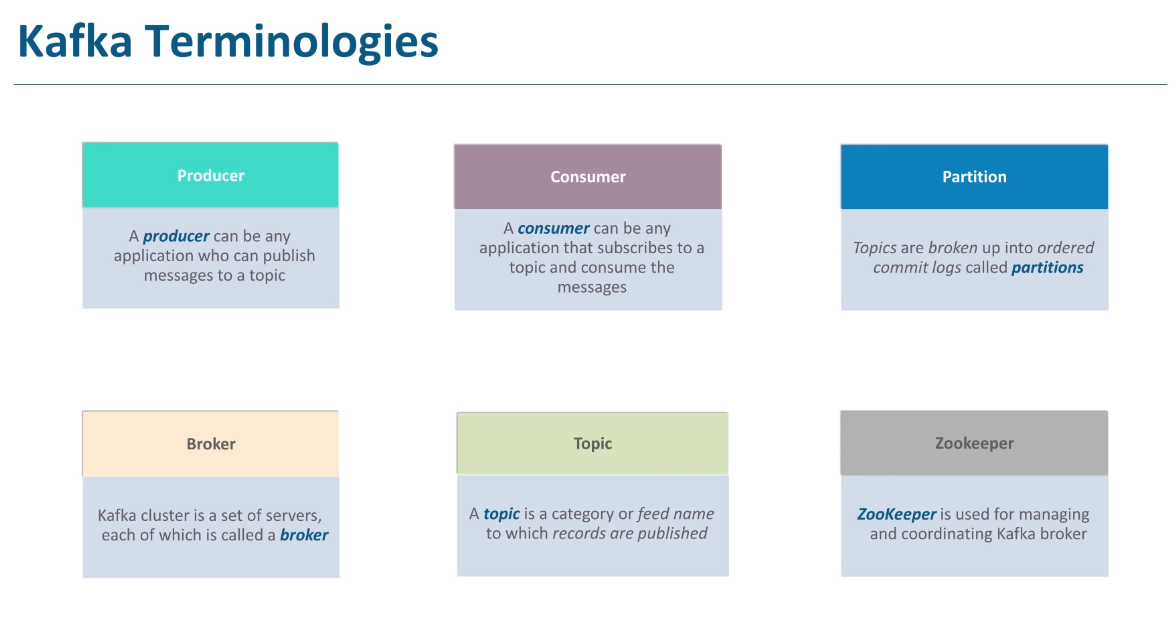
1. Queuing Messaging Model / Point – To – Point Model: - In a queue, a pool of customers may read from a single server & each record only goes to one of them.

2. Publish – Subscribe Model: - A record is broadcasted to all the consumers so multiple consumers can get the record.

> The Kafka cluster is distributed & have multiple machines running in parallel. This is the reason why Kafka is fast, scalable, durable & fault – tolerant.







**1. Topic: -** A topic is a category or feed name to which records are published. Topics in Kafka are always multi – subscriber i.e., 0, one or more consumers can subscribe to a topic & consume the data written to Kafka.

e.g., You can have sales records getting published to Sales topic, Product records getting published to Product topic & soon.

> This will actually segregate your messages & the consumer will only subscribe to the topic which they need.

> Kafka topics are divided into number of partitions

**2. Partition: -** Topics are broken up into ordered commit logs called partitions.

> Partitions allow you to parallelize a topic by splitting the data in a particular topic across multiple brokers. Each partition can be placed on a separate machine to allow multiple consumers to read from a topic parallelly.

> So, in case of Sales Topic, you can have 3 partitions from where 3 consumers can read data parallelly.

> **A Producer** can be any application who can publish messages to a topic.

> **Producers** publish data to Kafka (producer publishes the data to the topics of their own choice)

> **A Consumer** can be any application that subscribes to a topic & consume the messages.

> **Consumer** can subscribe to one topic & can consume data from that topic. You can have multiple consumers in a consumer group. So, consumers label themselves with a Consumer Group Name.

> Each record published to a topic is delivered to one consumer instance within that subscribing consumer group. But you can have multiple consumer group which can subscribe to a topic where one record can be consumed by multiple consumers i.e., one consumer from each **consumer group**.

> Consumer instances can be a separate processes or separate machines.

**3. Broker:** - Kafka cluster is a set of servers, each of which is called a broker.

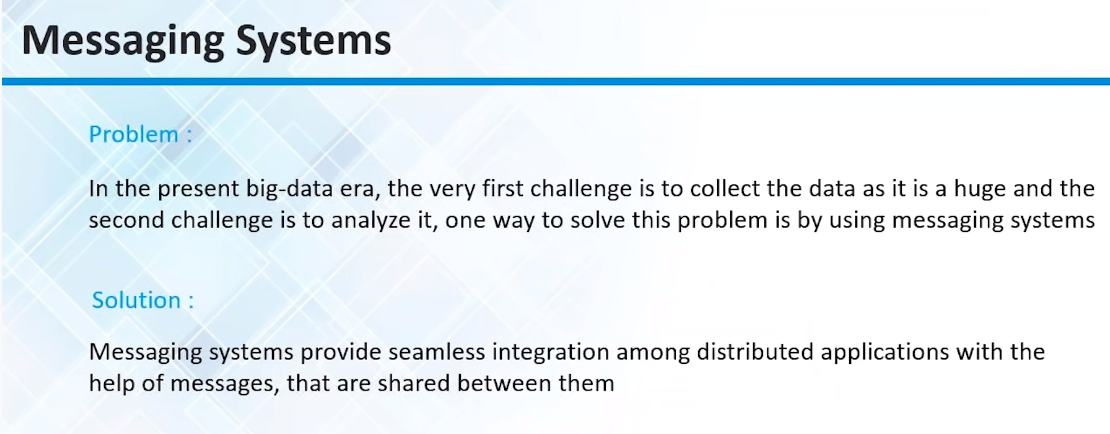
> Brokers are single machine in a Kafka cluster.

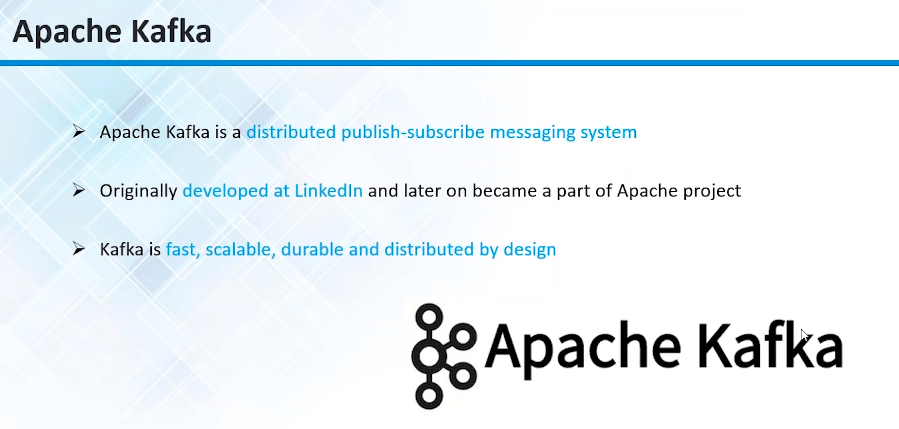
**4. Zookeeper:** - Zookeeper is used for managing & coordinating Kafka broker.

**Data : The Ingredient**

> Data is the main ingredient of Internet applications & typically includes the following:

* Page visits & clicks
* User activities
* Events corresponding to logins
* Social networking activities such as likes, shares, & comments
* Application specific metrics (e.g., logs, page load time, performance etc.)





> Apache Kafka is an open source also called Message Broker or Publish subscriber system.

> The Project intention of Apache Kafka development was to have a unified distributed messaging platform which has a very high throughput (say millions of or billions of messages per day) & a very low latency for all publishing or consumption & it should really really handle the real – time activity data stream from different systems.

> Basically, Developers picked up a design approach which was heavily relied or heavily dependent on the concept of transaction logs (commit logs or right ahead logs). This was a concept which is very efficient because of all the disk effectively used techniques so the transaction logs are primarily used in most of the transaction or OLTP based system to record all the changes happening in the OLTP databases.

> So this was the technique which most of the databases or NoSQL use it to make their data consistent so same technique has been adapted by the Apache Kafka designers to make it consistent & make it persistent in the data which the messaging platform supports so by doing that they also focused on distributed aspect of Apache Kafka which should be not just run in one single machine but potentially it should run on multiple machines with a cluster centric design where if you run on distributed machines then eventually the partitioning challenges comes into the play so the partition is if you have a huge data set in your cluster you can’t keep that entire data set cluster in one machine or in one node so you have to