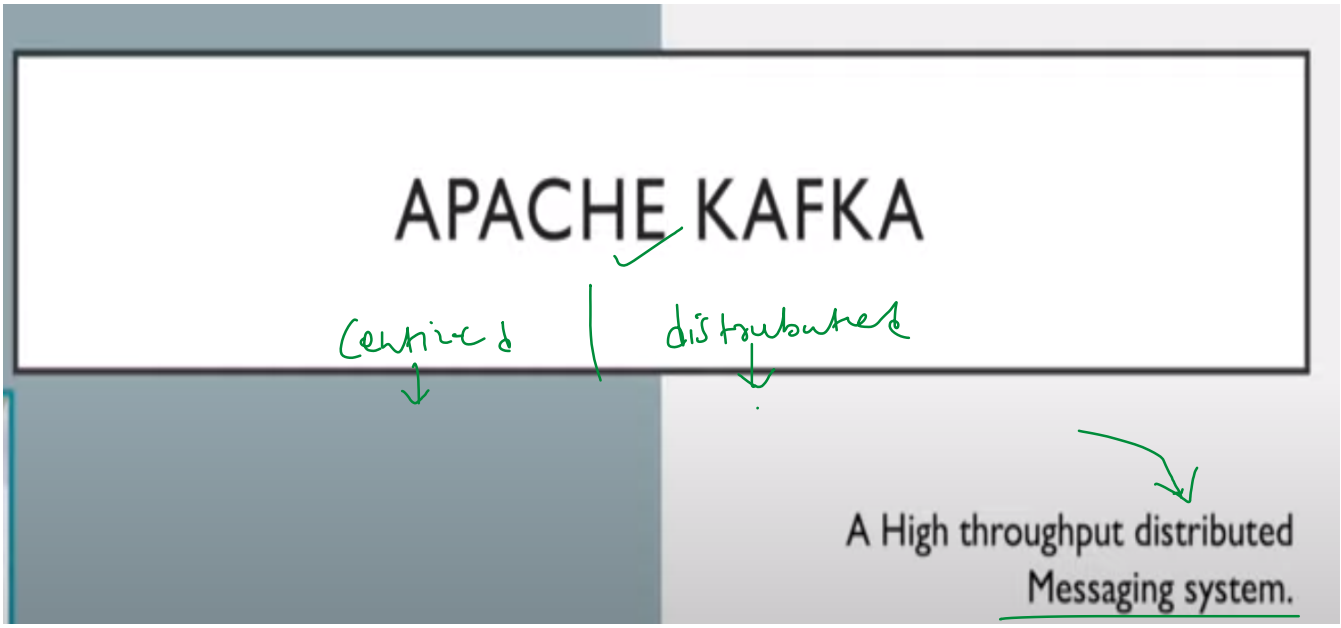
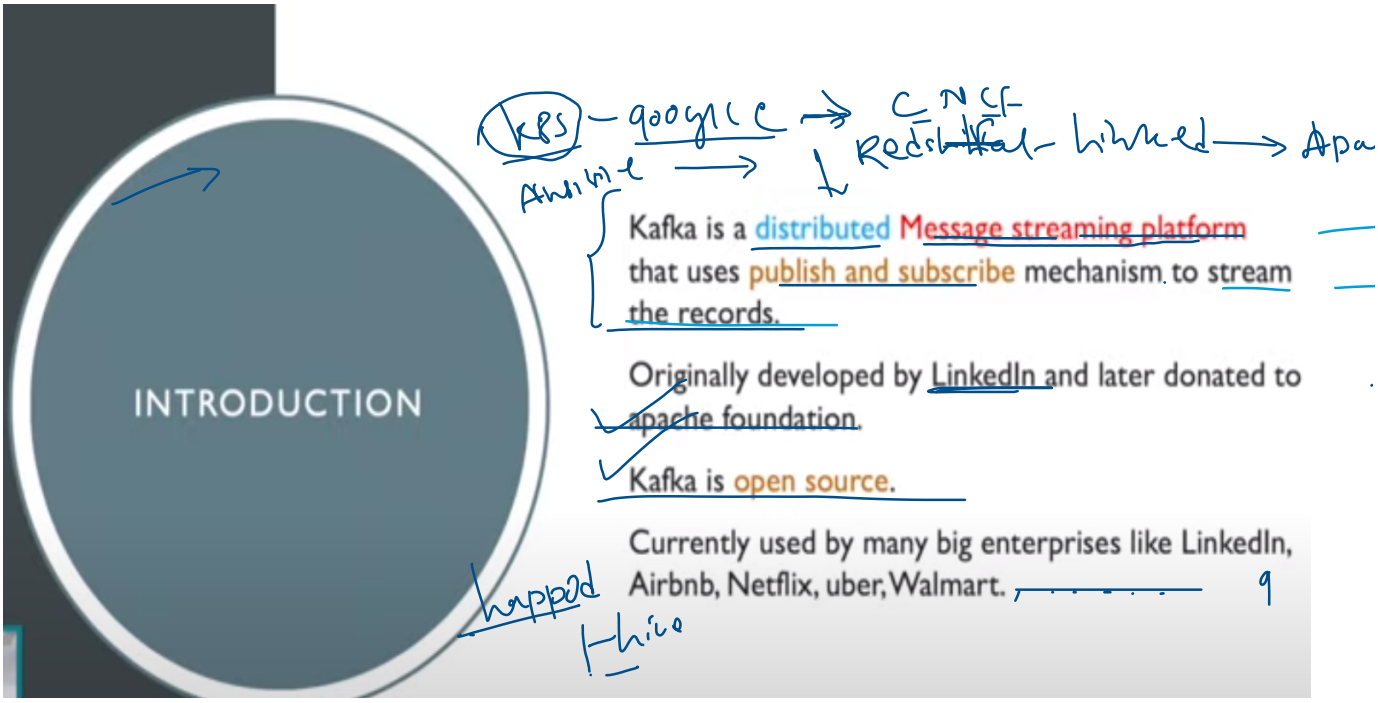
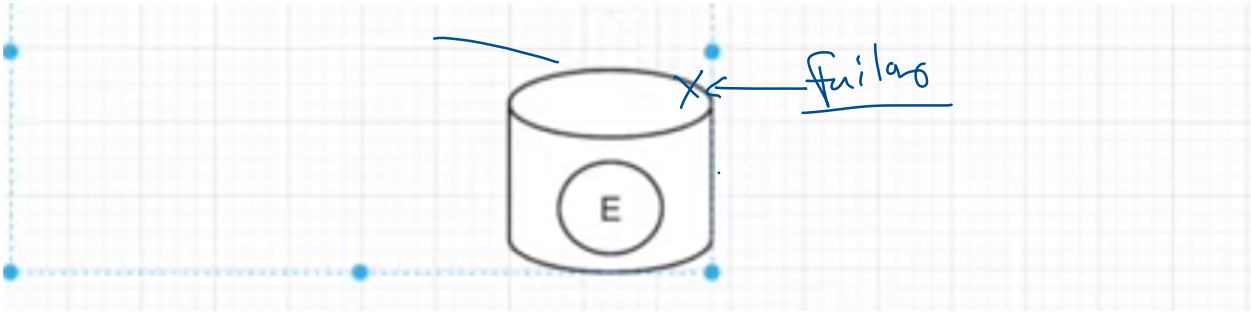
Kafka - Fundamentals

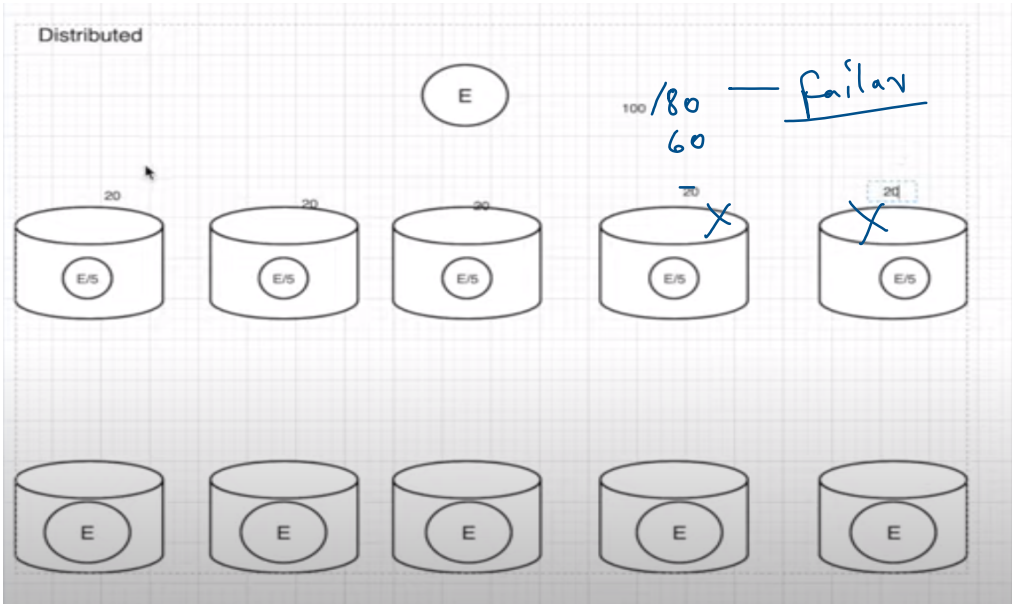
04:27 PM

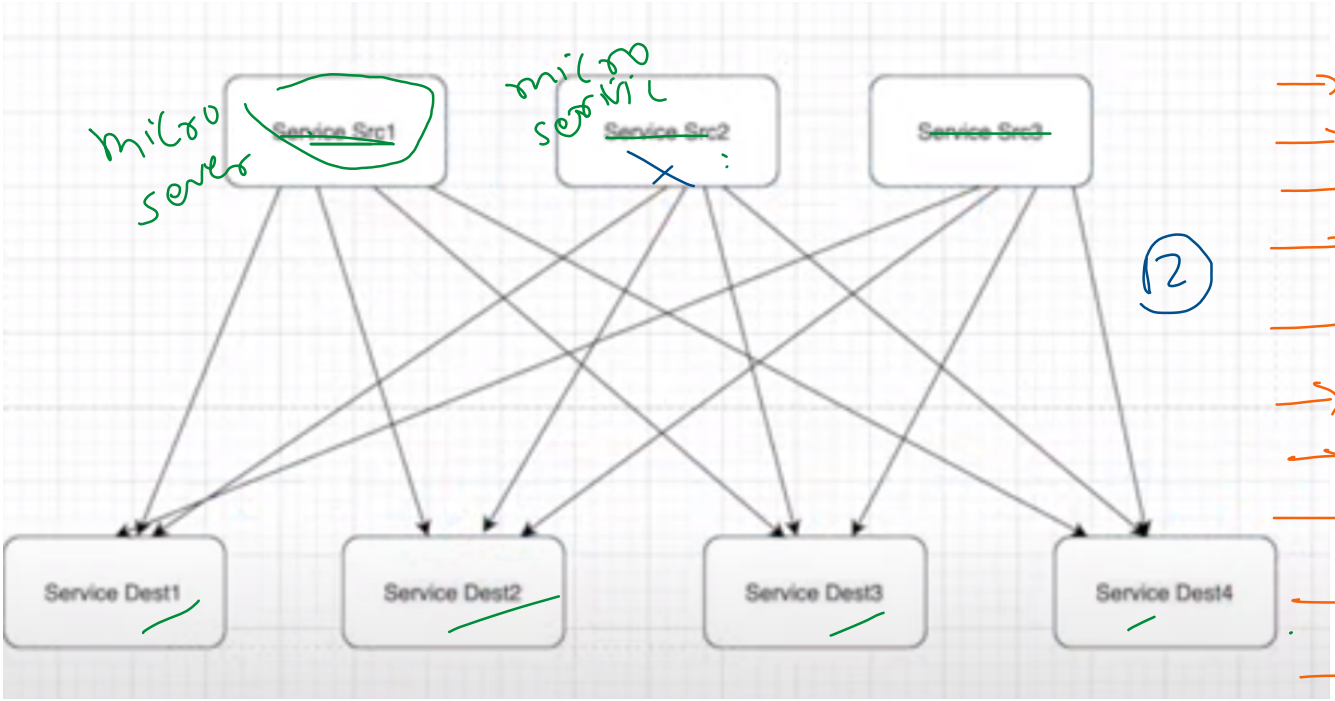
Apache Kafka



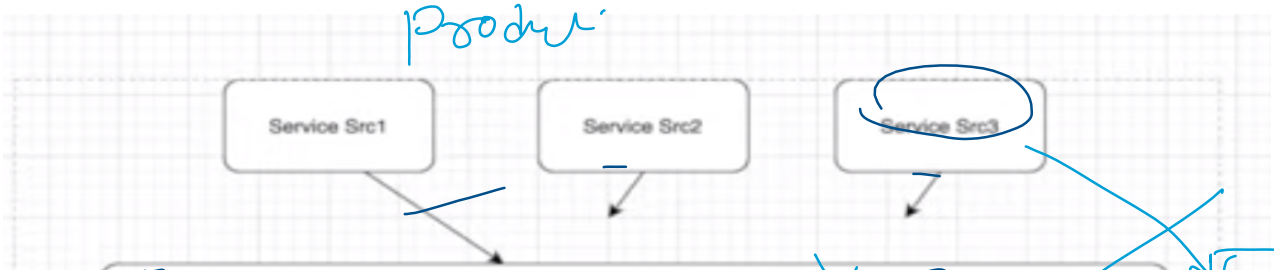


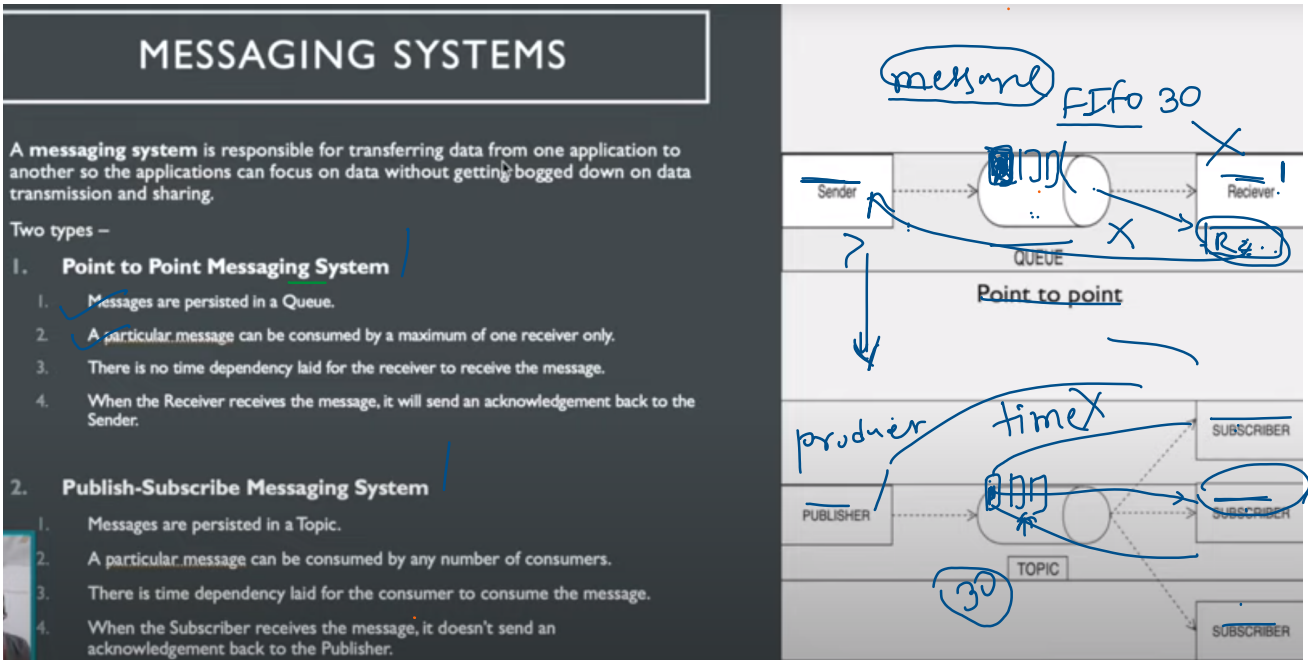






MESSAGEING SYSTEM →





Kafka is ideal for big data use cases that require the best throughput, while RabbitMQ is ideal for low latency message delivery, guarantees on a per-message basis, and complex routing.

Apache Kafka is a framework implementation of a software bus using stream-processing. It is an open-source software platform developed by the Apache Software Foundation written in Scala and Java.

The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

https://kafka.apache.org/powered-by

What is event streaming?

Technically speaking, event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events;

storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed.

Event streaming thus ensures a continuous flow and interpretation of data so that the right information is at the right place, at the right time.

What can I use event streaming for?

Event streaming is applied to a wide variety of use cases across a plethora of industries and organizations. Its many examples include:

- To process payments and financial transactions in real-time, such as in stock exchanges, banks, and insurances.
- To track and monitor cars, trucks, fleets, and shipments in real-time, such as in logistics and the automotive industry.
- To continuously capture and analyze sensor data from IoT devices or other equipment, such as in factories and wind parks.
- To collect and immediately react to customer interactions and orders, such as in retail, the hotel and travel industry, and mobile applications.
- To monitor patients in hospital care and predict changes in condition to ensure timely treatment in emergencies.
- To connect, store, and make available data produced by different divisions of a company.
- To serve as the foundation for data platforms, event-driven architectures, and microservices.

Apache Kafka® is an event streaming platform. What does that mean?

Kafka combines three key capabilities so you can implement your use cases for event streaming end-to-end with a single battle-tested solution:

1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
2. To **store** streams of events durably and reliably for as long as you want.
3. To **process** streams of events as they occur or retrospectively.

And all this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner. Kafka can be deployed on bare-metal hardware, virtual machines, and containers, and on-premises as well as in the cloud. You can choose between self-managing your Kafka environments and using fully managed services offered by a variety of vendors.

How does Kafka work in a nutshell?

Kafka is a distributed system consisting of **servers** and **clients** that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premise as well as cloud environments.

Servers: Kafka is run as a cluster of one or more servers that can span multiple datacenters or cloud regions. Some of these servers form the storage layer, called the brokers. Other servers run Kafka Connect to continuously import and export data as event streams to integrate Kafka with your existing systems such as relational databases as well as other Kafka clusters.

To let you implement mission-critical use cases, a Kafka cluster is highly scalable and fault-tolerant: if any of its servers fails, the other servers will take over their work to ensure continuous operations without any data loss.

Clients: They allow you to write distributed applications and microservices that read, write, and process streams of events in parallel, at scale, and in a fault-tolerant manner even in the case of network problems or machine failures.
Kafka ships with some such clients included, which are augmented by dozens of clients provided by the Kafka community: clients are available for Java and Scala including the higher-level Kafka Streams library, for Go, Python, C/C++, and many other programming languages as well as REST APIs.

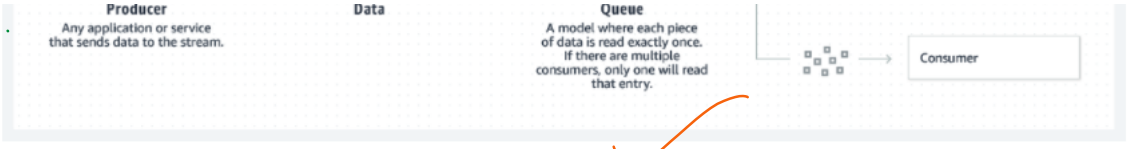
https://kafka.apache.org/uses

Kafka provides three main functions to its users:

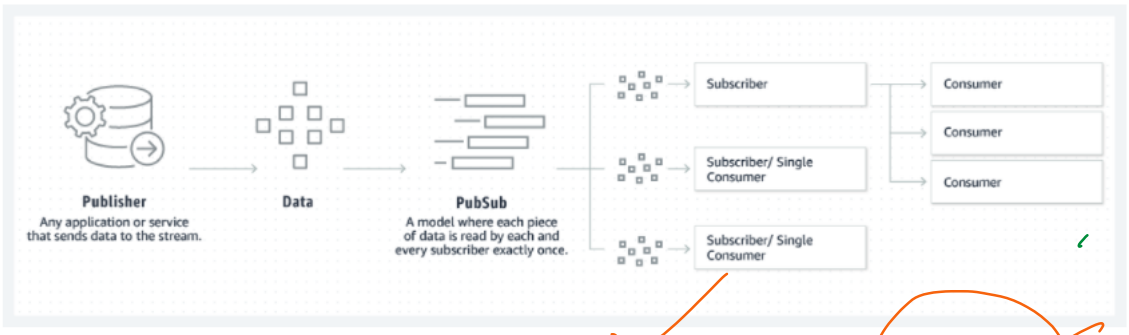
- Publish and subscribe to streams of records
- Effectively store streams of records in the order in which records were generated
- Process streams of records in real time

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

Queuing



Publish-Subscribe



CHARACTERISTICS	APACHE KAFKA	RABBITMQ
Architecture	Kafka uses a partitioned log model, which combines messaging queue and publish subscribe approaches.	RabbitMQ uses a messaging queue.
Scalability	Kafka provides scalability by allowing partitions to be distributed across different servers.	Increase the number of consumers to the queue to scale out processing across those competing consumers.
Message retention	Policy based, for example messages may be stored for one day. The user can configure this retention window.	Acknowledgement based, meaning messages are deleted as they are consumed.
Multiple consumers	Multiple consumers can subscribe to the same topic because Kafka allows the same message to be replayed for a given window of time.	Multiple consumers cannot all receive the same message, because messages are removed as they are consumed.
Replication	Topics are automatically replicated, but the user can manually configure topics to not be replicated.	Messages are not automatically replicated, but the user can manually configure them to be replicated.
Message ordering	Each consumer receives information in order because of the partitioned log architecture.	Messages are delivered to consumers in the order of their arrival to the queue. If there are competing consumers, each consumer will process a subset of that message.