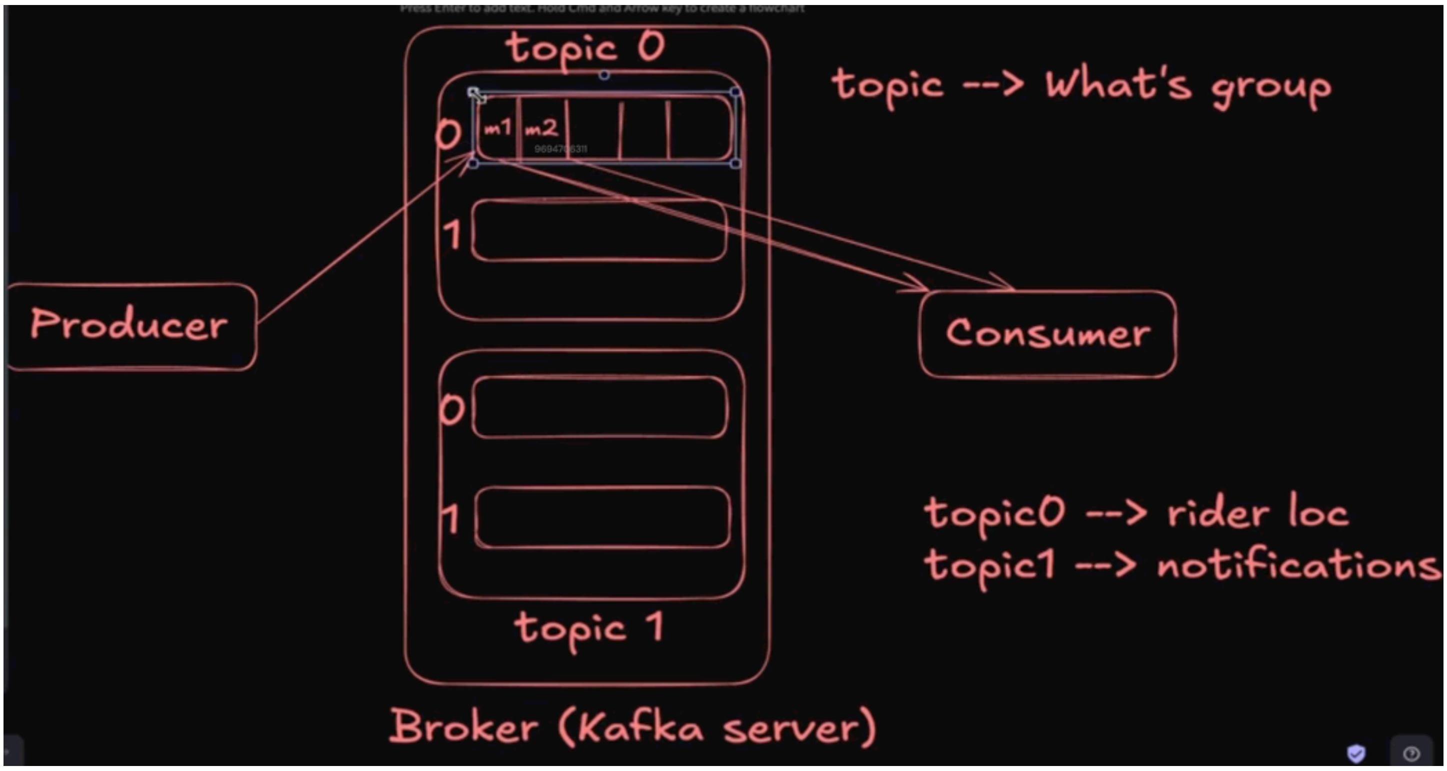
Kafka and Rabbit MQ

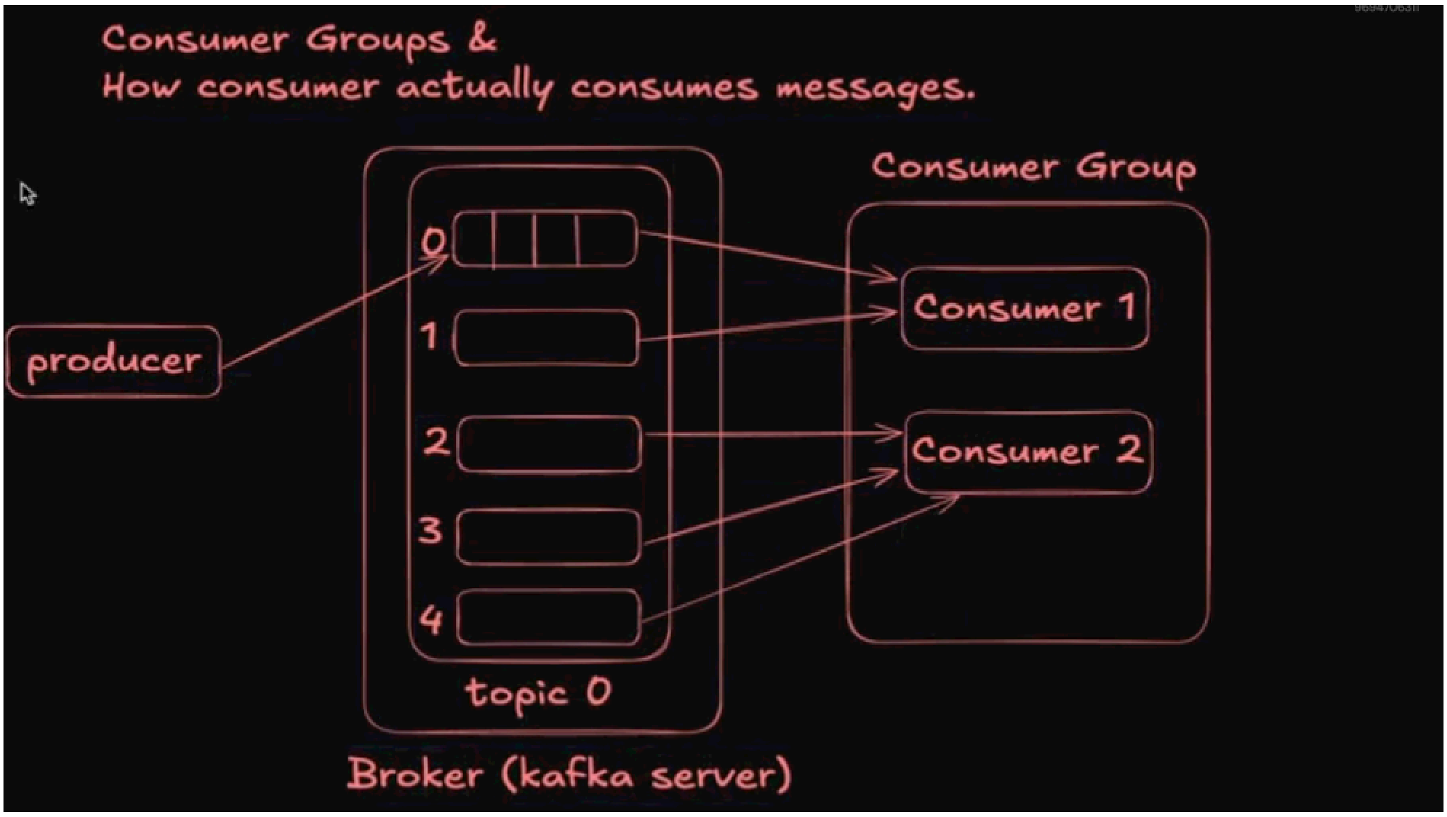


1.What is kafka ?

->Kafka is an open-source messaging system used to send, receive, and process large amounts of data in real time.

->Kafka organizes data into categories called "topics." Producers (apps that send data) put messages into these topics, and consumers (apps that read data) receive them. Kafka ensures that the system is reliable and can keep working even if some parts fail.

2.Core Components of Apache Kafka



1.What is a Kafka server(Broker)?

A Kafka broker is a server that receives, stores, and sends data in Kafka.

It works like a middleman between:

- Producers (send data)
- Consumers (read data)

2.What does a Kafka Broker do?

A Kafka broker:

- Receives messages from producers
- Stores messages safely
- Sends messages to consumers
- Keeps data safe even if something fails

3.Kafka Cluster

A Kafka cluster is a group of multiple brokers working together.

Example:

- Broker 1 (ID = 1)
- Broker 2 (ID = 2)
- Broker 3 (ID = 3)

Each broker has a unique ID.

4.Topic

A topic is like a category or box for messages.

Example:

- order-topic
- payment-topic

Producers send messages to topics.
Consumers read messages from topics.

5.Partitions

Each topic is divided into partitions.

Why partitions?

- Data is spread across brokers
- Faster processing
- Multiple consumers can work at the same time

One partition is stored on one broker.

6.Offset

Offset is a number given to each message in a partition.

It tells where the consumer is reading from.

Example:

Message 0 → offset 0

Message 1 → offset 1

Message 2 → offset 2

Kafka uses offset to:

- Remember last read message
- Continue from same place after restart

7.Consumer Group

A consumer group is a team of consumers working together to read data.

Rule:

- One partition → one consumer (inside a group)

Example:

- Topic has 3 partitions
- Consumer group has 3 consumers
  ➡ Each consumer reads one partition

Benefits:

- Faster processing
- Load balancing
- No duplicate work (inside same group)

8.ZooKeeper

ZooKeeper is a manager/helper for Kafka.

It:

- Keeps track of brokers
- Manages leader election
- Detects broker failure

Think of ZooKeeper as:
Kafka’s brain / coordinator

Note:

- Older Kafka used ZooKeeper
- New Kafka (KRaft mode) does not need ZooKeeper

What is KRaft? (Simple English)

KRaft stands for Kafka Raft Metadata mode.

It is a new way Kafka works without ZooKeeper.

Why KRaft was introduced?

Earlier:

- Kafka needed ZooKeeper to manage metadata
- This made Kafka complex to manage

Now (with KRaft):

- Kafka manages everything by itself
- ZooKeeper is not required

What does KRaft do?

KRaft handles:

- Broker information
- Topic and partition metadata
- Leader election
- Cluster management

(All work that ZooKeeper used to do)

3.How Kafka Broker Works

1. Producers send messages

Producers send data (logs, events, orders) to Kafka brokers.

2. Message storage

The broker stores messages safely on disk.

3. Replication (Data safety)

Kafka makes copies of data on different brokers.

So if one broker fails, data is not lost.

4. Leader and Follower

- One broker is Leader (handles read & write)
- Other brokers are Followers (copy data)

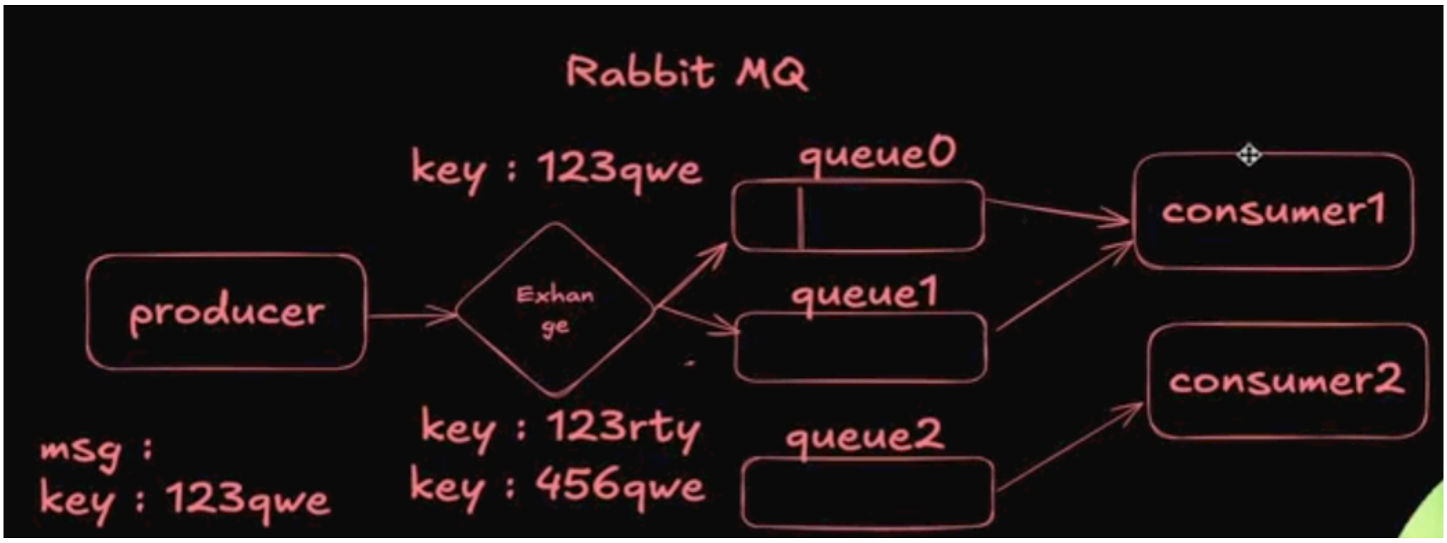
If leader fails → a follower becomes new leader automatically

5. Consumers read messages

Consumers read messages from brokers:

- In the same order
- In real time
- Using consumer groups for load balancing

What is RabbitMQ?



RabbitMQ is an open-source message broker that helps applications talk to each other using messages.

It uses queues to send and receive messages asynchronously.

1.Why RabbitMQ is used?

RabbitMQ is used to:

- Connect microservices
- Handle background tasks
- Process events
- Send messages reliably

Real-life idea

Think of RabbitMQ like a post office:

- Producer → sends letters
- Queue → stores letters
- Consumer → receives letters

2.Core Components

1.Producer

Sends messages to RabbitMQ.

2.Queue

Stores messages until a consumer reads them.

3.Consumer

Reads and processes messages from the queue.

4.Exchange

Decides which queue should get the message.

Types:

- Direct → exact match
- Fanout → send to all
- Topic → pattern match
- Headers → based on headers

5.Routing Key

A label used by exchange to route messages.

6.Binding

Connection between exchange and queue.

3.How RabbitMQ Works

1. Producer sends message
2. Exchange receives it
3. Exchange sends it to queue
4. Consumer reads message
5. Consumer sends ACK (done)

Messaging Patterns

- Point-to-Point → one sender, one receiver
- Publish / Subscribe → one sender, many receivers
- Request / Reply → two-way communication

Advantages of RabbitMQ

1.Reliable (ACK, persistence)
2.Scalable (handles many messages)
3.Flexible (many patterns & protocols)
4.Easy to use (UI + libraries)
5. Strong community support

Feature	Kafka	RabbitMQ
Type	Event streaming platform	Message broker
Data model	Log-based (streams)	Queue-based
Message storage	Stores messages for long time	Messages removed after consume
Performance	Very high (millions/sec)	Medium to high
Real-time streaming	Best choice	Not ideal
Message order	Maintained in partition	Maintained in queue
Replay messages	Yes (using offset)	No (by default)
Scalability	Very high	Good but limited
Use case	Logs, events, analytics	Tasks, background jobs
Protocol	Custom TCP protocol	AMQP, MQTT, STOMP
Consumer model	Pull-based	Push-based
Complexity	More complex	Easy to use

When to use Kafka?

Use Kafka when:

- Huge amount of data
- Real-time streaming
- Need message replay
- Event-driven systems

Example: Uber, Netflix, LinkedIn

When to use RabbitMQ?

Use RabbitMQ when:

- Task queues

- Background jobs
- Microservices communication
- Simple messaging

Example: Slack, Instagram, eBay

4.Kafka vs RabbitMQ (Pull / Push)

Kafka – Pull-based

- Consumers pull data from Kafka
- Consumer decides when and how much data to read
- Uses offset to track position

Benefits:

- Better control
- Handles high data volume
- Easy message replay

Example:
Consumer says → *“Give me messages from offset 100”*

RabbitMQ – Push-based

- RabbitMQ pushes messages to consumers
- Broker decides when to send messages
- Consumer just receives messages

Benefits:

- Simple
- Low latency
- Good for task queues

Example:
RabbitMQ says → *“Here is your next task”*

Feature	Kafka	RabbitMQ
Model	Pull-based	Push-based
Who controls flow	Consumer	Broker
Back-pressure	Easy to handle	Harder
Message replay	Yes	No
Best for	Streaming data	Background tasks

- Kafka uses pull-based messaging where consumers fetch data.
- RabbitMQ uses push-based messaging where the broker sends data.