Apache KAFKA (theory + imagination)

what is Apache Kafka ?

Imagine : Your system is not one app. Its many services talking to each other -: orders, payments, emails, analytics, notifications etc

If they call each other directly, the system becomes slow, tightly coupled and fragile.

Kafka Solves this :- It is a high speed data pipeline that lets system send and receive data in real time without depending on each other. Imagine it as -:

App A ⟶ Kafka ⟶ App B

Apps dont talk directly, they drop a message in Kafka, others pick them up when ready.

Imagine kafka like a post office for Data.

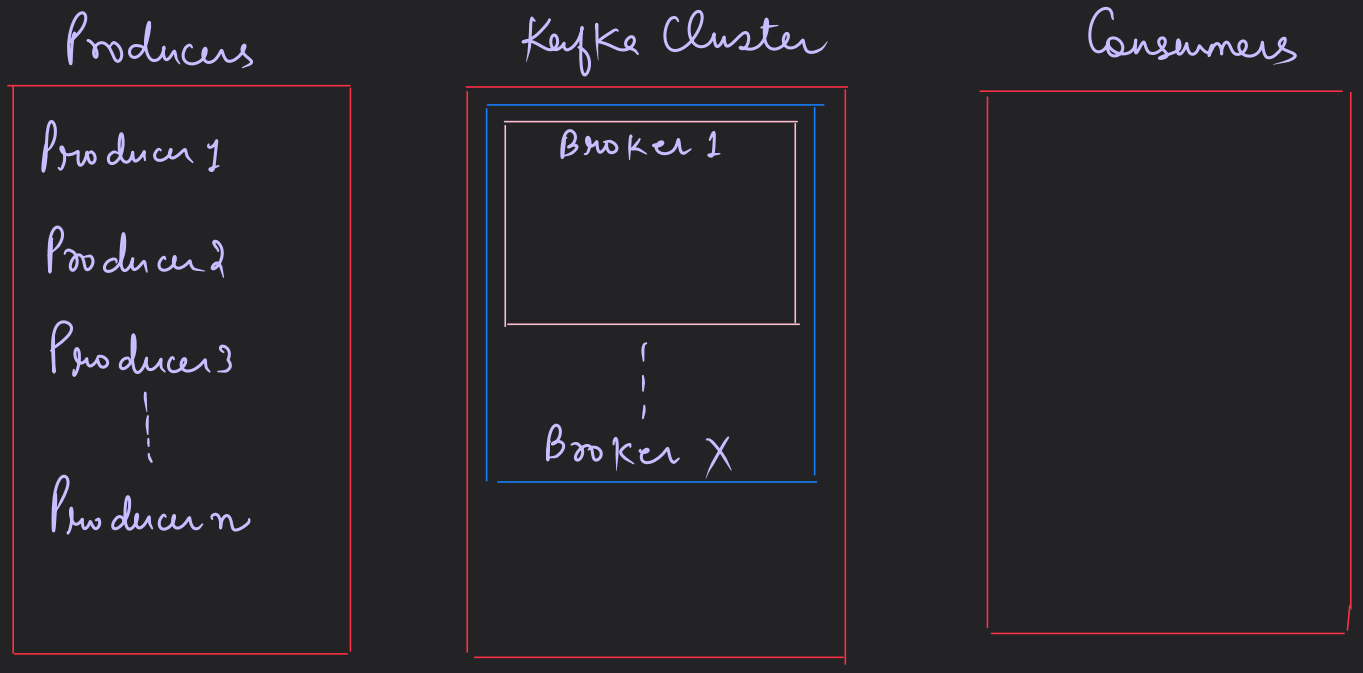
Producer ⟶ Person sending letter

Kafka ⟶ Post office

Topic ⟶ mailbox category

Consumer ⟶ Person Receiving letter

Kafka Architecture



think of Kafka as a huge warehouse of boxes. Apps dont talk to each other. They drop boxes in warehouse. other apps come & pick boxes.

 # Kafka Cluster :- Imagine it as a group of machines (servers) working together. Each machine is called a Broker.

Kafka Cluster = many Brokers (servers)

 # Topic :- Inside a warehouse we dont mix everything. We create separate sections Eg.

Section name	what goes inside?
Orders	order messages
Payments	Payment messages
Emails	Email events

Topic ≈ Category of message.

Partitions Imagine the order section is huge and just one shelf is not enough. So we split it into multiple shelves. these shelves ≃ Partitions

Topic : Orders

Shelf 0 (Partition 0)

Shelf 1 (Partition 1)

Shelf 2 (Partition 2)

⇒ topics are big category and partitions are smaller storage unit inside it.

Kafka Cluster

- Broker1
- Broker2
- Broker3

Topics : Exists at cluster level

- Orders
- Payments
- Emails

Each topic → split into partitions

Partitions → Stored across brokers.

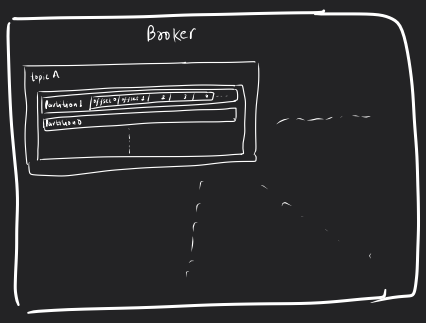
Broker1 → orders partition 0

Broker2 → orders partition 1

Broker3 → orders partition 2

⋮

Brokers donot own any topic. they store some partitions of many topics.



Remember A topic is split into partitions and those partitions are stored across different brokers in a kafka cluster.

A partition number is unique inside a topic. but copies (replica) of that partition can live on multiple brokers.

If you see

Broker 1 → Topic A Partition 0

Broker 2 → Topic A → Partition 1

this is correct, but one is leader and other is Replica

Replication — Kafka doesn't keep only 1 copy of data (dangerous)

It keeps Backups. This is called Replication factor.

If Topic A

Partitions = 2 (0 and 1)

then replication factor = 2.

means Each partition has 2 copies on different brokers.

Topic A (2 partitions, RF=2)

Partition	Leader Broker	Replica Broker
0	Broker 1	Broker 2
1	Broker 2	Broker 1

So Physically,

Broker 1 stores :-

Topic A → Partition 0 (leader)

Topic A → Partition 1 (Replica)

Broker 2 stores :-

Topic A → Partition 1 (leader)

Topic A → Partition 0 (Replica)

leader :- handles reads & writes

Replica :- just keeps copy of data

When and how is Replica Actually get used?

(1) ⇒ Suppose Broker 1 (leader) crashes.
→ Kafka automatically
⇒ detects leader is dead
⇒ chooses one replica (say broker 2)
⇒ makes it a new leader.
⇒ This is called leader election
⇒ Replica becomes active only during failure

↳ Fault Tolerance in Kafka

(2) ⇒ Case 2: Continuous Data Backup
⇒ Even when leader is alive,
→ Replicas continuously copy new message.
→ They stay in sync.
→ this is called ISR (In-Sync-Replicas)
→ these replicas are fully up to date.

(3) ⇒ Case 3: Prevention of Data loss
→ If Replication factor is 3,
You can loose 2 brokers and still not loose data (if atleast 1 ISR survives)

think it of →
leader → main driver
Replica → backup driver in car
the backup dont drive until leader faints

** Kafka replicas are passive followers that keep copying the data from the leader and become active only if the leader fails.
⇒ Kafka only picks in-sync replicas to become leader. If Replica was slow/outdated ⇒ not chosen

How does Kafka Know a particular Replica is Outdated?

Kafka Compares message positions (offset) between leader & replica.

offset's — Every message in a partition gets a number.

message 1 → offset 0
message 2 → offset 1
message 3 → offset 2

} offset = message position number.

→ leader partition is like a main book. Whenever producer sends data, leader writes message in book & tells replicas, to copy the same message!.

→ Replica keeps copying messages from leader in order. It also keeps track: my last copied offset is 105.

→ Kafka checks if Replica is healthy by checking offset lag, or heartbeat / fetch requests.

offset lag ⇓ how far the replica is

heartbeat / fetch requests ⇓ is replica actively syncing?

Case 1 Replica is healthy

leader offset = 200
Replica offset = 200

} Replica is in-sync ⇒ It will be part of ISR (in Sync Replica)

Case 2 Replica is slow

leader offset = 200
Replica offset = 180

} Replica is 20 msg behind. Kafka waits for a configured time (replica.lag.time.max.ms)
If it doesn't catchup → Remove from ISR

Case 3 Replica Stops talking

If Replica
→ stops sending fetch request
→ Or network breaks
→ leader says:- this guy is dead or too slow to be a leader
→ Removed from ISR.

What happens when a producer sends a message — inside KAFKA?

Lets follow 1 message from Producer → Kafka Cluster

Inside kafka cluster we have -
* multiple brokers (servers)
* metadata manager → Zookeeper for older, KRAFT for new Kafka version
* Topics
* Partitions

Step 0 :- Topic Creation (happens before messages)
(we / system has to create topics)

→ topic : orders
partitions 3
Replication Factor: 2 → 1 leader, 1 replica.

what happens internally?
→ metadata system chooses brokers
→ assigns partitions

these are created during topic creation.

Eg.

Partition	Leader	Replica
0	Broker 1	Broker 2
1	Broker 2	Broker 3
2	Broker 3	Broker 1

⇒ this mapping is stored in cluster metadata.

→ Brokers donot decide at runtime where message goes.
Partition-to-broker mapping is already decided when topic is created.

NOW producer Sends Message

Step 1 :- Producer asks cluster for metadata.
Producer says, where is topic Orders?

Cluster replies →
Partition 0 → Broker 1 (leader)
1 → Broker 2 (leader)
2 → Broker 3 (leader)

Producer cache this.

Step 2 :- Producer Choses Partition based on (a) key Provided
⇓
hash(key) % partitions

(b) No Key.
↳ round Robin.

Eg. Key = Customer ID 55
hash → Partition 2

Step 3 Producer Sends Message to leader Broker
Producer directly contacts Broker 3 (leader of partition 2)
Kafka Cluster is not a load balancer. Producer knows exactly where to send.

Step 4 leader broker stores messages.
→ partition is just a log file on disk
leader → opens partition log files → appends message at end → assigns offset

Before write : last offset = 100

After write : last offset = 101

offset : Simply next number in file.

Step 5 Replication Happens

leader tells replicas —: fetch new data from me.

Replicas copy message & reach offset (101) → now they are ISR (in Sync Replica)

Step 6 Acknowledgement to Producer

depending on acks :

Setting	when producer gets Success
1	leader wrote message.
all	leader + all ISR wrote

How are offsets created ?

Offset is not random.

for each partition, offset = position of message in append-only log

Each partition has its own offset sequence.

Partition 0 offset 10 ≠ Partition 1 offset 10

How consumers read using offsets and consumer groups ?

Consumer reads message from a topic But kafka doesnot push messages. Consumers pull messages.

What is an offset for Consumers ?

As we already know producers create offset when writing

Eg. (Partition 0)

Offset	Message
0	Order A
1	Order B
2	Order C
3	Order D

Consumers keep track :

→ I have read until offset 1, so next read starts from offset 2.

→ Kafka stores consumer progress in an internal topic :-

- Consumer_offsets

So, kafka remembers reading position even if app starts.

Part 2 Consumer Group :- team of consumers working together

topic :- Orders (3 partitions)

Confluent Kloud
Bank's lonely child

Case 1 → 1 Consumer

Consumer A → Reads P_0 → Reads P_1 → Reads P_2

Case 2 → 3 Consumers (same group)

Consumer A → Partition 0
Consumer B → Partition 1
Consumer C → Partition 2

Kafka divides work :- One partition → One consumer at a time (within a group)

Case 3 → More Consumers than partitions.

topic partitions = 3

Consumers = 5

⇒ 2 consumers will sit ideal.

Part 3 How consumers read step by step.

(a) Consumer joins a group

↓

kafka assigns partitions

↓

Consumer sends fetch requests

↓

Broker returns messages starting from stored offset.

↓

Consumer processes message

↓

Consumer commits new offset

↳ Auto Commit

Manual Commit

Part 4 Rebalancing

when Consumer Joins / Consumer Leaves / Broker fails, kafka re-assigns partitions → this is automatic

Producer tracks where to write (partitions)

Consumer tracks where to start read (offset)