1️⃣ First: WHY Kafka exists (the story)

Imagine this situation:

- A website is generating **thousands of events per second**
  - user clicks
  - orders
  - payments
  - logs

Now the problem:

- Producer (website) is **fast**
- Consumer (analytics / billing) is **slow**
- If consumer crashes → data lost
- If producer waits → website slows down

❌ Direct communication fails

Producer → Consumer directly = **tight coupling**, failures, data loss.

✅ Solution: Kafka in the middle

Kafka acts like a **buffer + log + pipeline**.

👉 **Producer throws data into Kafka and forgets**
👉 **Consumer reads whenever it is ready**

This is the core idea of Kafka.

2️⃣ Definition (WRITE THIS IN EXAM)

Apache Kafka is an open-source distributed event streaming platform designed for high-throughput, fault-tolerant, real-time data pipelines and stream processing.

Key words your teacher wants:

- distributed
- event streaming
- real-time
- fault tolerant

- high throughput

3️⃣ Kafka basic building blocks (VERY IMPORTANT)

(A) Topic

A **topic** is a **named stream of records**
Think of it like:

- a table name
- or a queue name

Examples:

- `orders`
- `clickstream`
- `user_logs`

👉 Producers write to topics
👉 Consumers read from topics

(B) Partitions (MOST IMPORTANT)

Each topic is split into **partitions**.

Each partition is:

- **Ordered** → messages have strict sequence
- **Immutable** → once written, never changed
- **Parallelizable** → multiple consumers can read in parallel

So:

Topic: orders
Partitions: P0, P1, P2

(C) Offset (EXAM FAVORITE)

Inside **each partition**, every message has an **offset**.

Offset = position of message inside a partition

- Offsets start from `0`
- Increase sequentially
- Unique **only within that partition**

📌 Very important:
Kafka **does NOT track consumption** — **consumer does** using offsets.

4️⃣ Producer (who sends data)

A **producer**:

- Sends messages to Kafka
- Chooses **which partition** to send to

How partition is chosen?

- If **message key is present** → `hash(key) % num_partitions`
- Same key → **same partition** (order preserved)

Example:

key = "Lahore" → always P0
key = "Karachi" → always P2

👉 This guarantees **ordering for same key**

5️⃣ Broker & Kafka Cluster

Broker

A **broker** is a **Kafka server**.

- Stores messages
- Serves producers & consumers
- A cluster has **multiple brokers**

Each broker can host:

- multiple partitions
- from multiple topics

6️⃣ Replication & Fault Tolerance (VERY IMPORTANT)

Each partition is **replicated**.

- **Leader replica** → handles read & write
- **Follower replicas** → copy data

If leader fails:
👉 one follower becomes **new leader** automatically

ISR (In-Sync Replicas)

ISR = replicas fully caught up with leader.

Kafka commits a write **only when ISR confirms**.

👉 This ensures **fault tolerance**

7️⃣ Consumer (who reads data)

A **consumer**:

- Reads messages from partitions
- Tracks **offsets**
- Commits offsets after processing

Kafka does **NOT push** data — consumer **pulls** data.

8️⃣ Consumer Group (EXAM GOLD)

Consumers work in **groups**.

Rules:

- Each partition → **only ONE consumer in a group**

- Consumers share load
- Kafka balances partitions automatically

Example:

Topic: orders (3 partitions)
Consumer group: billing-group
Consumers: C1, C2

P0 → C1
P1 → C2
P2 → C2

9️⃣ Offset Commit & Delivery Guarantees

Kafka supports:

At-most-once

- Message may be lost
- No duplicates

At-least-once

- Message may be processed twice
- No loss (default)

Exactly-once

- Processed once
- Needs special configuration

🔟 Consumer Lag (VERY VERY IMPORTANT)

Consumer Lag = Latest Offset − Committed Offset

It shows:

- how far consumer is behind producer

Low lag = healthy
High lag = problem

Causes:

- producer too fast
- consumer slow
- rebalancing
- crash/network issues

1️⃣1️⃣ Zookeeper (linked with Kafka – EXAM)

Kafka (legacy) uses **Zookeeper** for:

- metadata management
- broker coordination
- leader election

👉 You will study Zookeeper in **later lecture**, but here know:
Kafka depends on Zookeeper for **cluster coordination**.

1️⃣2️⃣ Kafka is like a Message Queue (but better)

Kafka provides:

- asynchronous communication
- load leveling
- fault tolerance
- scalability
- loose coupling

Difference:

- Kafka **does not delete messages immediately**
- Consumers track offsets themselves

1️⃣3️⃣ Where Kafka fits in Big Data Architecture

Kafka is used for:

- real-time ingestion
- streaming pipelines
- lambda architecture **speed layer**

🔑 EXAM-READY SHORT ANSWERS (3 points)

Q1: What is a Kafka topic?

- A named stream of records
- Divided into partitions
- Used by producers and consumers

Q2: Why partitions are important?

- Enable parallelism
- Preserve ordering within partition
- Improve scalability

Q3: What is offset?

- Position of message in partition
- Used by consumer to track progress
- Enables replay of data

Q4: Explain consumer group.

- Group of consumers
- Each partition assigned to one consumer
- Enables load balancing