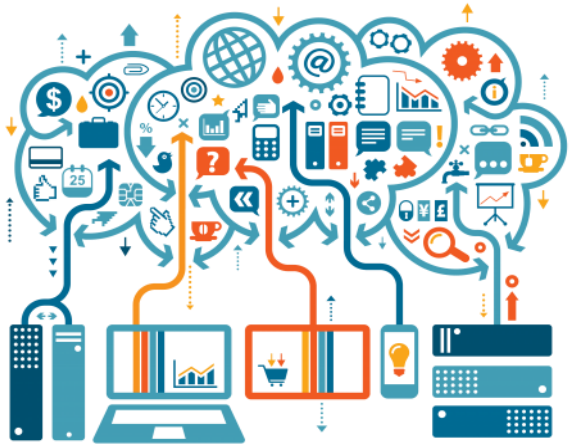
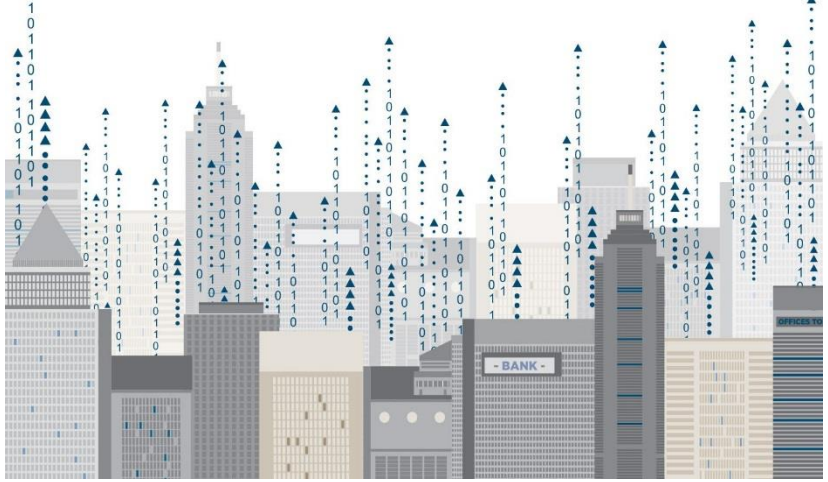




Big Data Analytics

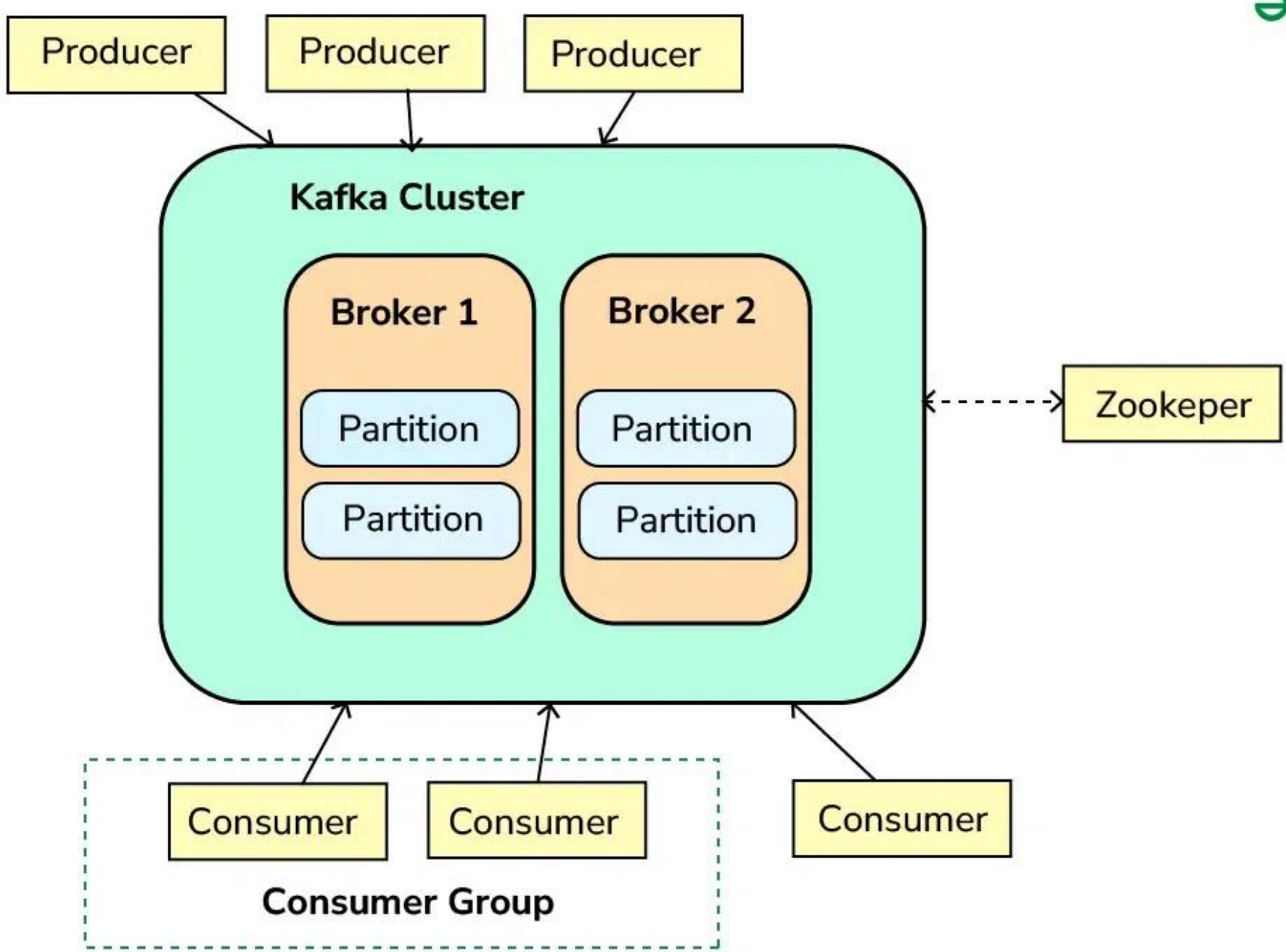
Fall 2025

Lecture 11

Definition

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

- It was originally developed at LinkedIn and later donated to the Apache Software Foundation.


Producer
Producer
Producer
Kafka Cluster
Broker 1
Partition
Partition
Broker 2
Partition
Partition
Zookeper
Consumer
Consumer
Consumer
Consumer Group

Architecture Overview

- A topic is a named stream of records — similar to a database table name or queue name. Example: user_logs, clickstream, orders

- Each topic is split into partitions (numbered 0, 1, 2, …).
- Each partition is:
  - Ordered: messages in a partition have a strict sequence
  - Immutable: once written, data is never modified
  - Parallelizable: partitions allow multiple consumers to read in parallel
- Every record in a partition has a unique numeric offset — like a position index.
- Consumers use offsets to remember their progress.

Architecture Overview

- A broker is a Kafka server process that stores and serves messages. A Kafka cluster has multiple brokers
- Each broker can host multiple partitions.

- Producer: An application that writes messages to Kafka topics.

- Consumer: An application that reads messages from Kafka topics.

Architecture Overview

- Consumer Group: Consumers can work as a group
- Kafka divides topic partitions among them for load balancing
- Each message is delivered to one consumer in the group (not all).

- Replication: Each partition has a leader replica (handles read/write) and follower replicas (copies data).
- If the leader fails → one follower becomes the new leader.

Architecture Overview

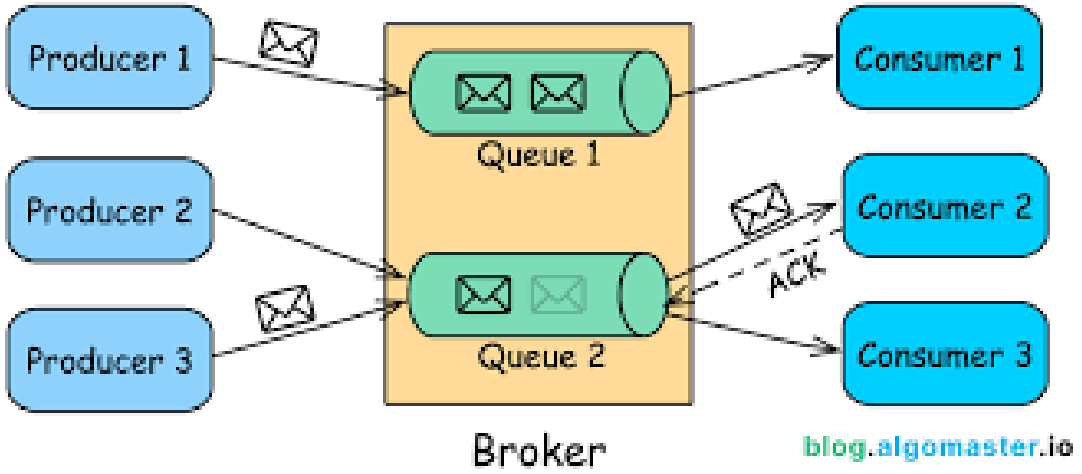
- ISR (In-Sync Replicas): Set of replicas that are fully caught up with the leader.
- Kafka writes are “committed” when all ISR replicas confirm receipt.

Architecture Overview

- Producers: Send messages to topic partitions.
- Kafka Brokers: Store messages and coordinate replication.
- Zookeeper (Legacy) or KRaft (New): Manage metadata and controller elections.
- Consumers: Read data from partitions, commit offsets.
- Connectors / Streams / ksqlDB: Handle integrations and processing.

Capabilities of Kafka

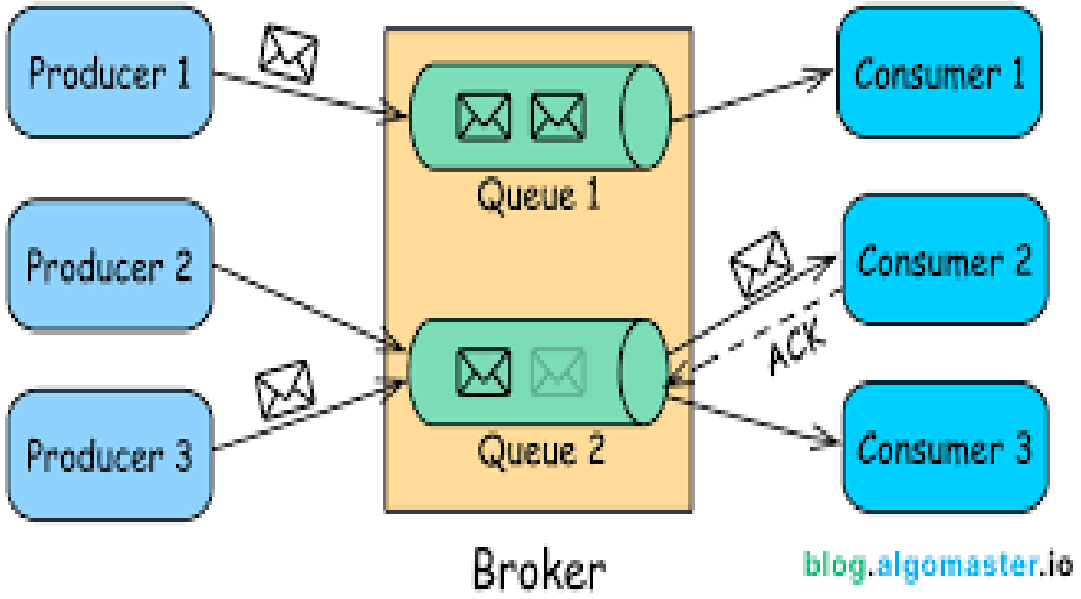
- Publish and subscribe to data streams (like a message queue).
- Store streams durably and fault-tolerantly.
- Process streams in real-time (via Kafka Streams / ksqlDB)
- Message queue:



Message Queue

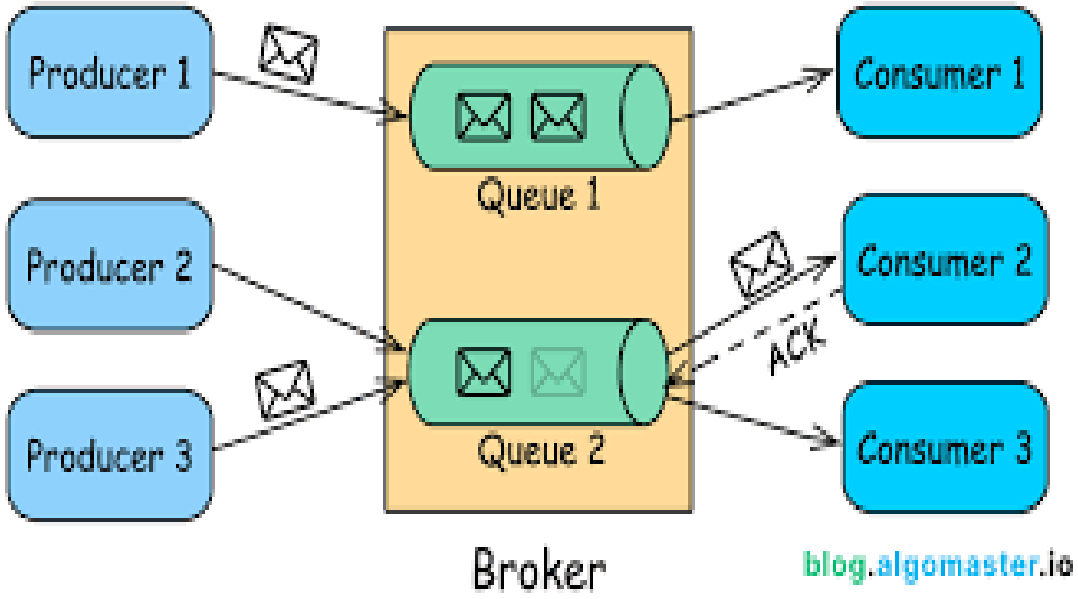
- Temporarily stores messages (data packets, tasks, or events) sent from one component (*producer*) until they are retrieved and processed by another (*consumer*).

- This ensures reliable delivery, scalability, and fault tolerance — even if one component is slow or temporarily offline.



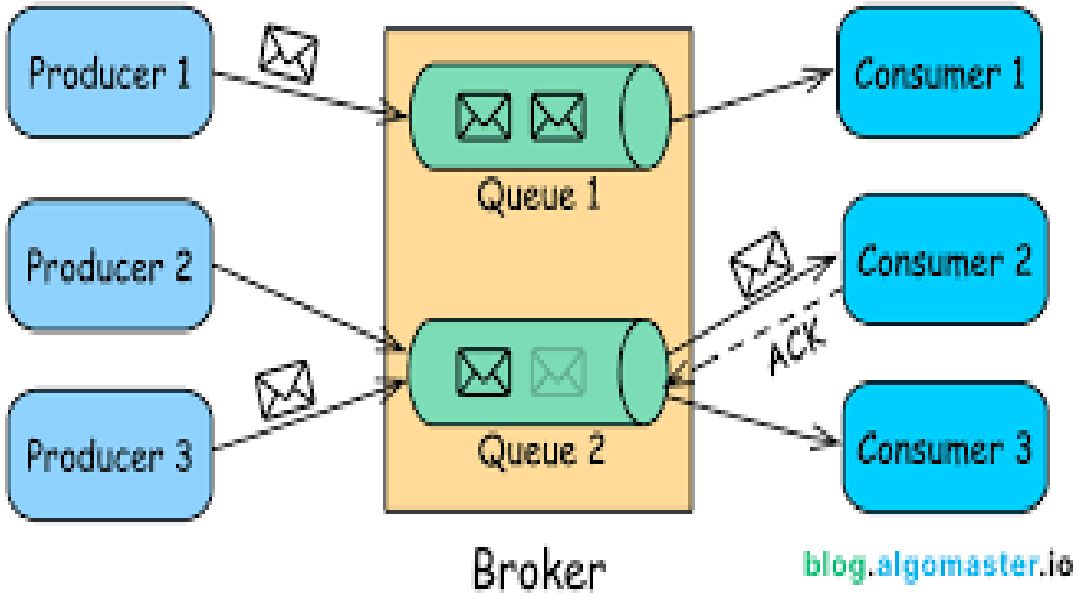
Message Queue

- Producer sends a message to the queue.
- The message is stored in the queue (usually persisted until acknowledged).
- Consumer retrieves messages from the queue, processes them, and acknowledges receipt.
- Once acknowledged, the message is deleted from the queue.



Message Queue

- Asynchronous processing — producer doesn't wait for consumer.
- Load leveling — helps smooth spikes in workload.
- Fault tolerance — messages aren't lost if consumers crash.
- Scalability — multiple consumers can process messages in parallel.
- Loose coupling — services don't depend on each other's availability.



Message Queue

Tool	Description
RabbitMQ	Reliable, feature-rich message broker supporting AMQP protocol.
Apache Kafka	Distributed streaming platform, optimized for high throughput.
ActiveMQ	Open-source broker supporting many protocols (JMS, AMQP, MQTT).
Amazon SQS	Managed message queue service on AWS.
Redis Streams	Lightweight message queuing via Redis data structures.
ZeroMQ	Fast, brokerless messaging library for distributed apps.

The Data Flow!

- Producer sends data to Kafka broker (topic partition).
- Broker writes to disk and replicates to followers.
- Consumer reads from the leader partition and tracks offset.
- Consumer commits offset → ensures at-least-once delivery.

The Guarantees

Guarantee	Description	Risk
At-most-once	Messages are delivered at most once; some may be lost.	Data loss possible.
At-least-once	Messages delivered one or more times.	Duplicates possible.
Exactly-once	Messages delivered once and only once.	Requires configuration

The Components

Component	Purpose
Kafka Connect	Framework for integrating Kafka with databases, S3, files, etc.
Kafka Streams	Java library for stream processing on Kafka topics.
KSQL / ksqlDB	SQL-like interface for querying streams in Kafka.
Schema Registry	Stores Avro/JSON/Protobuf schemas for message compatibility.

Producer sends 6 messages

Message	Key	Partition (hash(key))
M1	userA	P0
M2	userB	P1
M3	userC	P2
M4	userA	P0
M5	userB	P1
M6	userC	P2

P0: M1 → M4
P1: M2 → M5
P2: M3 → M6

Msgs from same key always go to the same partition

Fault Tolerance and Replication

- Each partition is replicated across N brokers.
- One broker is the leader, others are followers.
- Followers continuously pull data from the leader.
- If leader fails → one follower becomes new leader.

A Major Problem – Consumer Lag

- The difference between the latest message written to a Kafka partition and the latest message that a consumer has processed (committed).
- It measures how far behind a consumer is from the head (latest offset) of the topic.
- Consumer Lag = Latest Offset (log end) – Last Committed Offset
- Cluster Lag = sum of lag across all partitions.

Topic “transactions” with 1 partition

Time	Message	Partition Offset
10:00	M1	0
10:01	M2	1
10:02	M3	2
10:03	M4	3
10:04	M5	4

- Kafka has written 5 messages (offsets 0–4)
- A consumer has only processed up to M3 (offset = 2)
- Consumer Lag=4−2=2 : 2 messages waiting to be consumed (M4 and M5).

Topic “transactions” with 1 partition

Time	Message	Partition Offset
10:00	M1	0
10:01	M2	1
10:02	M3	2
10:03	M4	3
10:04	M5	4

- Low Lag (0–few messages): consumer keeping up with prod. speed
- High Lag (hundreds/thousands): consumer is falling behind→ may cause delays, timeouts, or data freshness issues.

Why lag?

- Producer rate too high → consumer can’t keep up
- Consumer app slow → heavy processing per message
- Consumer failure → temporary disconnection
- Rebalancing → lag spikes during partition reassignment
- Network or disk issues → delay reading offsets or messages

Example

🎯 Scenario: "Order Processing System"

We have:

- **Producer:** an *Order Service* that sends order events.
- **Topic:** `orders` (where events are stored).
- **Consumer Group:** *Billing Service* and *Analytics Service* that process those orders.

Example

🧱 Step 1 — The Topic and Partitions

Kafka topic: `orders`

It has **3 partitions** (to allow parallelism):

Partition	Stored messages (offsets start from 0)
P0	M0, M1, M2, M3
P1	M0, M1, M2
P2	M0, M1, M2, M3, M4

Each message in a partition has a **unique offset number** that increases **within** that partition only.

> Offset = "position" of message in that partition (like line number in a log file).

Example

📓 Step 2 — Producer Behavior

The **Order Service (Producer)** sends order events like:

Order ID	Partition Chosen	Message Key	Offset Assigned
101	P0	Lahore	0
102	P2	Karachi	0
103	P0	Lahore	1
104	P1	Islamabad	0
105	P2	Karachi	1
106	P0	Lahore	2

Example

📓 Step 2 — Producer Behavior

The **Order Service (Producer)** sends order events like:

Order ID	Partition Chosen	Message Key	Offset Assigned
101	P0	Lahore	0
102	P2	Karachi	0
103	P0	Lahore	1
104	P1	Islamabad	0
105	P2	Karachi	1
106	P0	Lahore	2

Topic: orders

P0: [0:101], [1:103], [2:106], [3:109]

P1: [0:104], [1:107], [2:110]

P2: [0:102], [1:105], [2:108], [3:111], [4:112]

Example

Topic: orders

P0: [0:101], [1:103], [2:106], [3:109]

P1: [0:104], [1:107], [2:110]

P2: [0:102], [1:105], [2:108], [3:111], [4:112]

🧍 Step 3 — Consumers

Now, there's a **consumer group** named `billing-group` with **two consumers**:

- **Consumer-A**
- **Consumer-B**

Kafka will **balance** the partitions between consumers:

Partition	Assigned Consumer
P0	Consumer-A
P1	Consumer-B
P2	Consumer-B

Example

🧮 Step 4 — Offsets and Reading Progress

Each consumer maintains a **committed offset** — i.e., "how far I've read."

Partition	Latest Offset (head)	Consumer Committed Offset	Lag
P0	3	2	1
P1	2	2	0
P2	4	2	2

📊 **Lag** = `Latest Offset - Committed Offset`

So:

- Consumer-A (P0) is **1 message behind**.
- Consumer-B (P2) is **2 messages behind**, but up-to-date on P1.

Total **group lag = 3 messages**.

Topic: orders

P0: [0:101], [1:103], [2:106], [3:109]

P1: [0:104], [1:107], [2:110]

P2: [0:102], [1:105], [2:108], [3:111], [4:112]

Example

🕒 Step 5 — Message Flow Timeline

Time	Action	Explanation
10:00	Producer sends Order #101 (P0 offset=0)	stored in Kafka
10:01	Consumer-A reads offset 0 → commits	up-to-date
10:02	Producer sends Order #103, #106	new offsets 1,2
10:03	Consumer-A still busy → lag increases (2 messages behind)	
10:04	Consumer-A processes offset=1, commits	lag = 1
10:05	Producer sends Order #109 (P0 offset=3)	new message, lag again = 2

Topic: orders

P0: [0:101], [1:103], [2:106], [3:109]

P1: [0:104], [1:107], [2:110]

P2: [0:102], [1:105], [2:108], [3:111], [4:112]

What to Monitor

- Broker health
- Consumer lag
- Disk usage
- Network I/O
- GC activity

Tools

- Prometheus + JMX Exporter
- Grafana dashboards
- Confluent Control Center
- Burrow (for consumer lag)

```
version: '2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:8.1.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
  kafka:
    image: confluentinc/cp-kafka:8.1.0
    depends_on: [zookeeper]
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT:
'zookeeper:2181'
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://localhost:9092

KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

create topic
docker exec -it <kafka_container> kafka-topics --create --topic test --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

produce
docker exec -it <kafka_container> kafka-console-producer --topic test --bootstrap-server localhost:9092
type messages

consume
docker exec -it <kafka_container> kafka-console-consumer --topic test --bootstrap-server localhost:9092 --from-beginning

```
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
for i in range(10):
    producer.send('test', key=b'key', value=f'msg-{i}'.encode())
producer.flush()
```

```
from kafka import KafkaConsumer
consumer = KafkaConsumer('test',
bootstrap_servers='localhost:9092',
auto_offset_reset='earliest', group_id='g1')
for msg in consumer:
    print(msg.offset, msg.key, msg.value)
```