Member-only story

The World of Event Streaming — Day 1

Welcome to Kafka!

Arvind Kumar Following 9 min read · Dec 22, 2025

70 4

Imagine you're at a busy restaurant. The old way (message queues) is like a waiter taking your order, delivering it to the kitchen, waiting for the food, and bringing it back. One order at a time, slow and sequential.

Kafka is like a **conveyor belt system** in a modern restaurant. Orders come in continuously, get placed on the belt, and multiple chefs can grab orders, customers can see what's available, and managers can track everything in real-time. Everyone works at their own pace, and nothing gets lost.

That's event streaming — data flows continuously, and multiple systems can consume it independently.

> *Full story for non-members* | *Microservices E-Book* | *Spring Boot E-Book* | *Join Whatsapp Group for Daily Tech Bytes* | *Youtube* | *LinkedIn*

Why Kafka? Evolution from Message Queues to Event Logs

The Old Way: Message Queues

Traditional message queues (like RabbitMQ, ActiveMQ) work like a **postal system:**

- You send a letter (message) to a recipient
- Once delivered and acknowledged, the letter is typically removed
- Messages are “consumed” — after reading, they’re gone

- RabbitMQ CAN send to multiple consumers (via fanout/topic exchanges), but messages are usually deleted after acknowledgment

Key Characteristics:

- Messages are delivered and removed (or moved to dead-letter queue)
- No built-in replay capability — once consumed, you can't re-read it
- Designed for task distribution and request/response patterns
- Great for: RPC, task queues, request routing

The New Way: Event Logs (Kafka)

Kafka works like a **newspaper archive**:

- Events are published (like articles) and stored permanently
- Multiple readers can subscribe and read independently
- You can read today's paper, yesterday's, or even last week's
- New readers can join anytime and catch up on history
- Events are retained for a configurable period (days, weeks, or forever)

Key Differences:

- **Message Queue:** "Deliver this message" → consumed → deleted (postal system)
- **Event Log:** "Publish this event" → stored → replayable by anyone (newspaper archive)

Why This Matters:

- **Replay capability:** Re-process events if your consumer crashes or needs to catch up
- **Multiple independent consumers:** Each consumer reads at their own pace from their own position
- **Event sourcing:** Build state by replaying all events from the beginning
- **Audit trail:** All events are preserved for compliance and debugging

Real-World Example

Think of an e-commerce system:

- **Old way:** Order service sends message to Payment service → waits → sends to Inventory service → waits
- **Kafka way:** Order service publishes "OrderPlaced" event → Payment, Inventory, Email, Analytics all consume it simultaneously, at their own pace

Core Concepts: Building Blocks of Kafka

Let's understand Kafka's core concepts with simple analogies:

1. Topic

A topic is like a **newspaper section** (Sports, Business, Technology).

- All related events go into one topic

- Example: `orders`, `payments`, `user-events`

2. Partition

A partition is like a **page in that newspaper section.**

- Topics are split into partitions for parallel processing
- Each partition is an ordered sequence of events
- Think of it as multiple lanes on a highway — more lanes = more throughput

3. Broker

A broker is like a **newspaper printing press.**

- Kafka cluster = multiple printing presses (brokers)
- Each broker stores some partitions
- If one press breaks, others keep printing

4. Producer

A producer is like a **reporter** writing articles.

- Applications that publish events to topics
- Example: Order Service producing “OrderPlaced” events

5. Consumer

A consumer is like a **reader** subscribing to the newspaper.

- Applications that read events from topics
- Example: Payment Service consuming “OrderPlaced” events

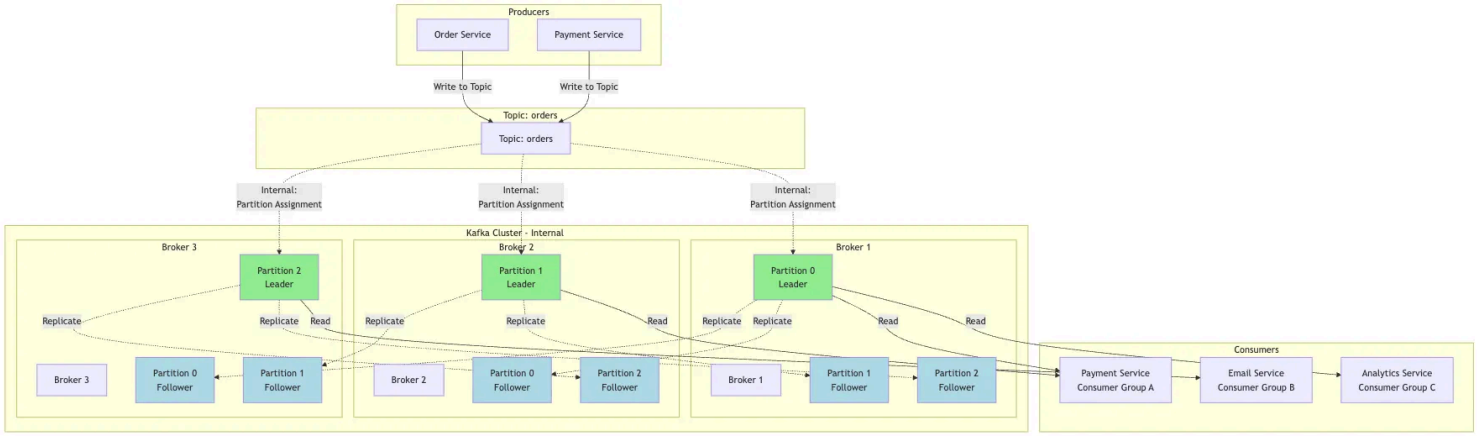
6. Consumer Group

A consumer group is like a **family sharing one newspaper subscription.**

- Multiple consumers in a group share the work
- Each partition is read by only one consumer in the group
- If one consumer is slow, others pick up the slack

Kafka Architecture Overview

Visual Architecture Diagram



Legend:

- **Green (Leader):** Handles all read/write requests

- **Light Blue (Follower):** Replicates data from leader
- **Solid arrows:** Producer/Consumer operations (external)
- **Dashed arrows:** Internal Kafka operations (partition assignment & replication)

Leader-Follower Replication

Imagine a **master chef and assistant chefs:**

- **Leader:** The master chef (handles all requests)
- **Followers:** Assistant chefs (copy everything the master does)
- If the master chef gets sick, an assistant chef becomes the new master

In Kafka:

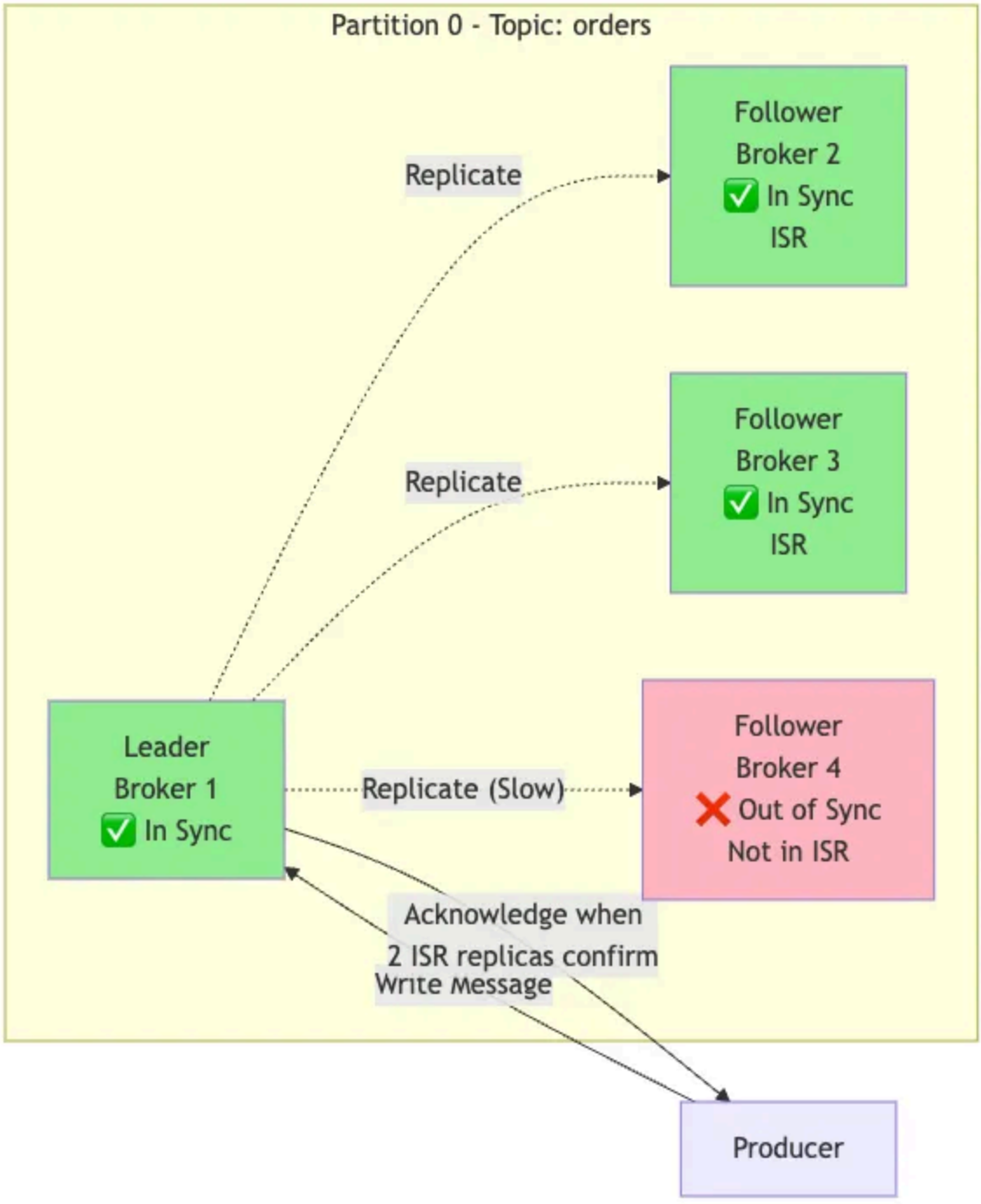
- Each partition has one **leader** (handles reads/writes)
- Multiple **followers** (replicate data)
- If leader fails, a follower becomes the new leader automatically

ISR (In-Sync Replicas)

ISR is like a **quality control team:**

- Only followers that are “caught up” with the leader are in ISR
- If a follower falls behind (network issue, slow disk), it’s removed from ISR
- Leader only considers writes successful when ISR replicas confirm

Why this matters: Ensures data durability. Even if a broker crashes, your data is safe on other brokers.



What this shows:

- **Leader** and **2 Followers** are in ISR (In-Sync Replicas) — they're caught up

- **1 Follower** is out of sync (maybe network issue or slow disk) — removed from ISR
- Producer gets acknowledgment when **ISR replicas** confirm (not the out-of-sync one)
- If leader crashes, only ISR followers can become the new leader

KRaft Mode (No More Zookeeper!)

Modern Kafka (4.1.0+) uses KRaft mode:

- **Old way:** Kafka relied on Zookeeper for cluster coordination and metadata storage
- **New way:** Kafka uses its own **KRaft** (Kafka Raft) protocol for cluster management
- **Benefits:** Simpler setup, better performance, easier scaling, no separate Zookeeper cluster needed

Think of it like a company that used to hire an external manager (Zookeeper) but now manages itself internally (KRaft). Everything runs smoother and faster!

Hands-on: Setting Up Kafka Locally

Let's get Kafka running on your machine. We'll use Docker (the easiest way) with **official Apache Kafka 4.1.0** (open source).

> ***Good news!*** *Modern Kafka (4.1.0+) no longer needs Zookeeper! It uses* ***KRaft mode*** *(Kafka Raft) for cluster management, making setup simpler.*

Prerequisites

- Docker Desktop installed and running
- Basic terminal/command line knowledge

> *If you want to run kafka without Docker follow this guide — link*

Step 1: Create Docker Compose File

Create a file named `docker-compose.yml` in a new folder. We're using the latest official Apache Kafka with KRaft mode (no Zookeeper needed):

```
services:
  kafka:
    image: apache/kafka:4.1.0
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_NODE_ID: 1
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka:9093

      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092,PLAINTEXT_INTERNAL://0.0.0.0:290
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL:
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: CONTROLLER:PLAINTEXT,PLAINTEXT:PLAIN
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER

      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1

      KAFKA_LOG_DIRS: /var/lib/kafka/data
      CLUSTER_ID: MkU3OEVBNTcwNTJENDM2Qk

    volumes:
```

```
      - kafka_data:/var/lib/kafka/data

  kafdrop:
    image: obsidiandynamics/kafdrop:latest
    container_name: kafdrop
    ports:
      - "9000:9000"
    environment:
      KAFKA_BROKERCONNECT: kafka:29092
    depends_on:
      - kafka

volumes:
  kafka_data:
```

What's different?

- **No Zookeeper!** Kafka manages itself using KRaft
- `KAFKA_PROCESS_ROLES: broker,controller` - Single node acts as both broker and controller
- `KAFKA_CONTROLLER_QUORUM_VOTERS` - Defines the controller quorum (just one node for local dev)

Step 2: Start Kafka

Open your terminal in the folder where you created `docker-compose.yml` and run:

```
docker-compose up -d
```

```
codefarm@Arvinds-MacBook-Pro notification-service % docker-compose up -d
[+] Running 3/3
 ✔ Network notification-service_default  Created
 ✔ Container kafka                      Started
 ✔ Container kafdrop                    Started

codefarm@Arvinds-MacBook-Pro notification-service %
```

This starts Kafka in KRaft mode. Wait about 30–45 seconds for it to fully start (Kafka needs to format the storage on first run).

Verify it's running:

```
docker-compose ps
```

```
codefarm@Arvinds-MacBook-Pro notification-service % docker-compose ps
NAME      IMAGE                          COMMAND                  SERVICE   CREATED          STATUS          PORTS
kafdrop   obsidiandynamics/kafdrop:latest   "/kafdrop.sh"            kafdrop   15 seconds ago   Up 15 seconds   0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp
kafka     apache/kafka:4.1.0               "/__cacert_entrypoin…"   kafka     15 seconds ago   Up 15 seconds   0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp
```

You should see the `kafka` service running. Check the logs to ensure it started successfully:

```
docker-compose logs kafka
```

```
kafka  | [2025-12-28 10:19:26,791] INFO authorizerStart completed for endpoint PLAINTEXT. Endpoint is now READY. (org.
kafka  | [2025-12-28 10:19:26,791] INFO authorizerStart completed for endpoint PLAINTEXT_INTERNAL. Endpoint is now REA
kafka  | [2025-12-28 10:19:26,791] INFO [SocketServer listenerType=BROKER, nodeId=1] Enabling request processing. (kafka.network.SocketServer)
kafka  | [2025-12-28 10:19:26,791] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
kafka  | [2025-12-28 10:19:26,792] INFO Awaiting socket connections on 0.0.0.0:29092. (kafka.network.DataPlaneAcceptor)
kafka  | [2025-12-28 10:19:26,795] INFO [BrokerServer id=1] Waiting for all of the authorizer futures to be completed (kafka.server.BrokerServer)
kafka  | [2025-12-28 10:19:26,795] INFO [BrokerServer id=1] Finished waiting for all of the authorizer futures to be completed (kafka.server.BrokerServer)
kafka  | [2025-12-28 10:19:26,795] INFO [BrokerServer id=1] Waiting for all of the SocketServer Acceptors to be started (kafka.server.BrokerServer)
kafka  | [2025-12-28 10:19:26,795] INFO [BrokerServer id=1] Finished waiting for all of the SocketServer Acceptors to be started (kafka.server.BrokerServer)
kafka  | [2025-12-28 10:19:26,795] INFO [BrokerServer id=1] Transition from STARTING to STARTED (kafka.server.BrokerServer)
kafka  | [2025-12-28 10:19:26,795] INFO Kafka version: 4.1.0 (org.apache.kafka.common.utils.AppInfoParser)
kafka  | [2025-12-28 10:19:26,797] INFO Kafka commitId: 13f70256db3c994c (org.apache.kafka.common.utils.AppInfoParser)
kafka  | [2025-12-28 10:19:26,797] INFO Kafka startTimeMs: 1766917166795 (org.apache.kafka.common.utils.AppInfoParser)
kafka  | [2025-12-28 10:19:26,797] INFO [KafkaRaftServer nodeId=1] Kafka Server started (kafka.server.KafkaRaftServer)
kafka  | [2025-12-28 10:19:26,880] INFO [GroupCoordinator id=1] Scheduling loading of metadata from __consumer_offsets-13 with epoch 4 (org.apache.kafka.coor
.CoordinatorRuntime)
kafka  | [2025-12-28 10:19:26,880] INFO [GroupCoordinator id=1] Scheduling loading of metadata from __consumer_offsets-46 with epoch 4 (org.apache.kafka.coor
.CoordinatorRuntime)
```

Look for messages like “Kafka Server started” — this means Kafka is ready!

Step 3: Access Kafdrop (Kafka Web UI)

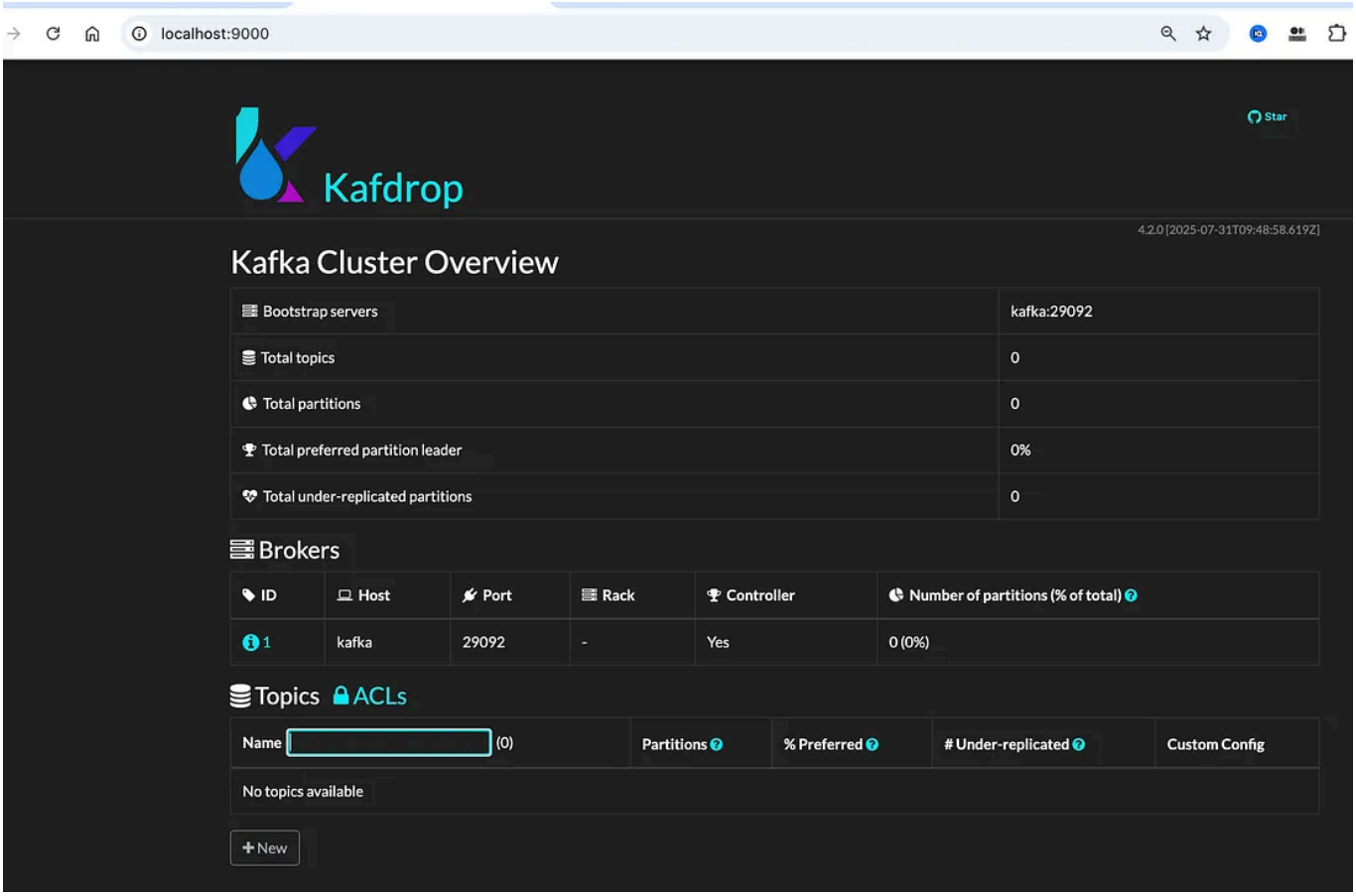
Kafdrop is a web-based UI for monitoring your Kafka cluster. It’s already included in the docker-compose file above!

Access Kafdrop:

1. Open your web browser
2. Navigate to: `http://localhost:9000`
3. You should see the Kafdrop interface

What you’ll see:

- **Topics list:** All topics in your cluster (empty for now)
- **Brokers:** Information about your Kafka brokers
- **Consumer groups:** Active consumer groups
- **Message browser:** View messages in topics (once you create some)



Kafdrop features:

- ✅ Browse topics and partitions
- ✅ View messages in real-time
- ✅ See consumer group lag
- ✅ Monitor broker health
- ✅ Create topics (though we'll use CLI for learning)

Note: If Kafdrop doesn't load immediately, wait 10–20 seconds for it to start and connect to Kafka.

Hands-on: Create Your First Topic

Now let's create a topic! Think of this as creating a new newspaper section.

Step 1: Enter the Kafka Container

```
docker exec -it <kafka-container-name> bash
```

```
codefarm@Arvinds-MacBook-Pro notification-service % docker exec -it kafka bash
cc57d0c5d04b:/$
```

To find the container name:

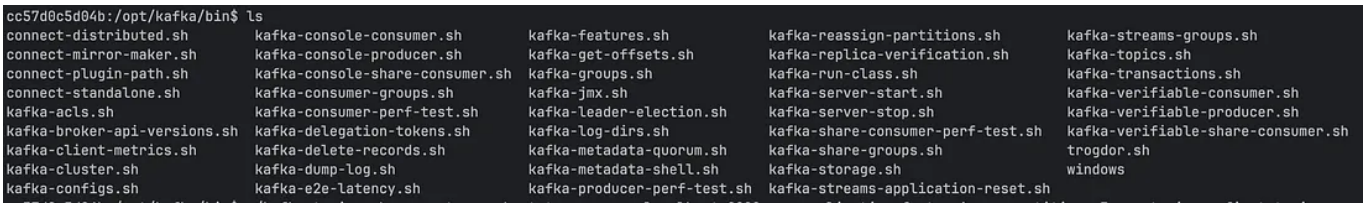
```
docker ps
```

Look for the container with `apache/kafka` image. Copy its name (it should be `kafka` if you used the docker-compose above, or something like.

Step 2: Create a Topic

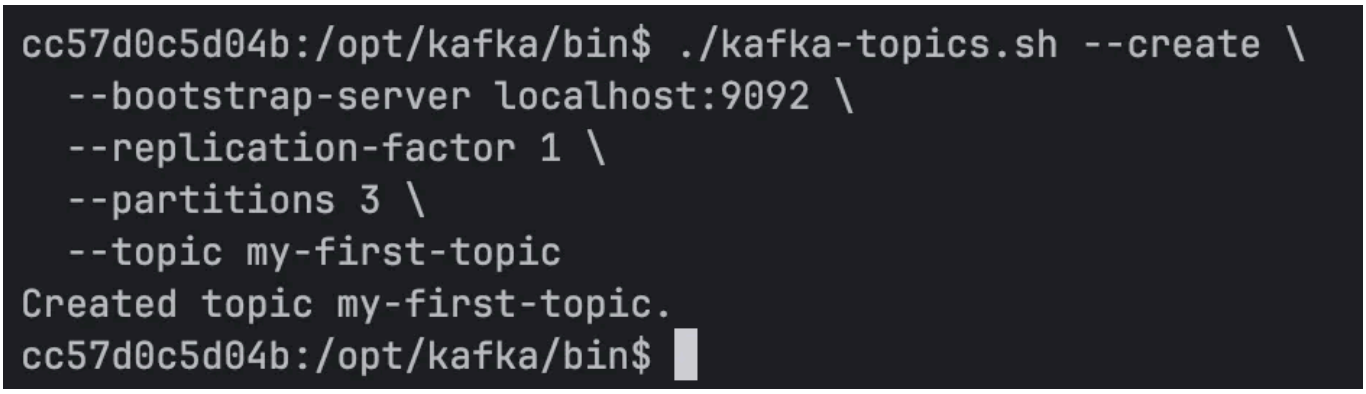
Inside the container, run:

Move to bin directory *:/opt/kafka/bi* and check all commands available



Then command to create the topic

```
./kafka-topics.sh --create \
  --bootstrap-server localhost:9092 \
  --replication-factor 1 \
  --partitions 3 \
  --topic my-first-topic
```



What this means:

- `--create` : We're creating a new topic
- `--bootstrap-server localhost:9092` : Where Kafka is running
- `--replication-factor 1` : How many copies of data (1 = no replication, fine for local dev)
- `--partitions 3` : Split the topic into 3 partitions (like 3 lanes on a highway)

- `--topic my-first-topic` : The name of our topic

You should see: `Created topic my-first-topic.`

Step 3: Verify Your Topic

Let's list all topics to confirm it was created:

```
./kafka-topics.sh --list --bootstrap-server localhost:9092
```

You should see `my-first-topic` in the list.

```
cc57d0c5d04b:/opt/kafka/bin$ ./kafka-topics.sh --list --bootstrap-server localhost:9092
my-first-topic
cc57d0c5d04b:/opt/kafka/bin$
```

Step 4: Describe Your Topic (Optional but Useful)

See detailed information about your topic:

```
./kafka-topics.sh --describe \
  --bootstrap-server localhost:9092 \
  --topic my-first-topic
```

This shows:

- Partition count (3)

- Replication factor (1)
- Which broker is the leader for each partition

```
cc57d0c5d04b:/opt/kafka/bin$ ./kafka-topics.sh --describe \
  --bootstrap-server localhost:9092 \
  --topic my-first-topic
Topic: my-first-topic   TopicId: Lz3aK68jTfSt4jv2bV0EOA PartitionCount: 3       ReplicationFactor: 1    Configs: min.insync.replicas=1
        Topic: my-first-topic   Partition: 0    Leader: 1       Replicas: 1     Isr: 1  Elr:    LastKnownElr:
        Topic: my-first-topic   Partition: 1    Leader: 1       Replicas: 1     Isr: 1  Elr:    LastKnownElr:
        Topic: my-first-topic   Partition: 2    Leader: 1       Replicas: 1     Isr: 1  Elr:    LastKnownElr:
cc57d0c5d04b:/opt/kafka/bin$
```

Exit the container:

```
exit
```

```
cc57d0c5d04b:/opt/kafka/bin$ exit
exit

codefarm@Arvinds-MacBook-Pro notification-service %
```

What You've Learned Today

✅ **Why Kafka?** — Event logs vs message queues (newspaper archive vs postal system)
✅ **Core Concepts** — Topics, partitions, brokers, producers, consumers
✅ **Architecture** — Leader-follower replication and ISR
✅ **Hands-on** — Set up Kafka locally and created your first topic

Quick Check: Do You Understand?

Before moving to Day 2, make sure you can answer:

1. What's the difference between a message queue and an event log?
2. Why do we split topics into partitions?
3. What happens if a Kafka broker (leader) crashes?
4. What does ISR stand for and why is it important?

What's Next?

Tomorrow (Day 2), you'll:

- Use CLI tools to produce and consume messages
- Explore topic metadata and offsets
- Simulate broker failures
- Set up Kafdrop for visual monitoring

Congratulations! You've completed Day 1. You now have Kafka running locally and understand the fundamentals. Take a break, and we'll dive deeper tomorrow!

Troubleshooting

Kafka won't start?

- Make sure Docker Desktop is running
- Check if ports 9092 (broker) or 9093 (controller) are already in use
- Try: `docker-compose down` then `docker-compose up -d`
- On first run, Kafka needs to format storage — wait 30–45 seconds

Can't find the container?

- Run `docker ps -a` to see all containers (including stopped ones)
- Container name should be `kafka` (or `<folder-name>-kafka-1` if using default naming)

Topic creation fails?

- Wait 30–60 seconds after starting Kafka (longer on first run)
- Check logs: `docker-compose logs kafka`
- Look for "Kafka Server started" in the logs to confirm it's ready

KRaft mode issues?

- If you see errors about CLUSTER_ID, make sure the environment variable is set correctly
- For a fresh start, remove the container and volumes: `docker-compose down -v`

=======

Below is a collection of all the stories in one place

List: Kafka Bootcamp | Master Kafka with Codefarm | Curated by Arvind Kumar | Medium

Kafka Bootcamp | Master Kafka with Codefarm · Kafka Bootcamp related stories · 1 stories on Medium

codefarm0.medium.com

Kafka | Event Driven Architecture | Apache Kafka | Kafka Consumer | Kafka Producer

Written by Arvind Kumar

3.2K followers · 110 following

Following

Staff Engineer | System Design, Microservices, Java, SpringBoot, Kafka, DBs, AWS, GenAI | Teaching concepts via stories & characters | linkedin.com/in/codefarm0

Responses (4)

Pradeep Kasula

What are your thoughts?

Priyanshi
Dec 28, 2025

very well explained.

1 1 reply Reply

San Rawat
Dec 27, 2025

why localhost is not working rather replace with kafka in yml file make it working.

1 1 reply Reply

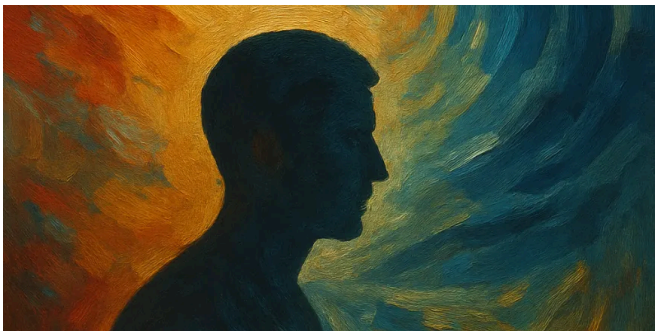
Kritikakhanna she/her
Dec 24, 2025

I loved how you managed to create a simple analogy like newspaper to describe components of Kafka. It is easy to follow and remember.

1 1 reply Reply

See all responses

More from Arvind Kumar

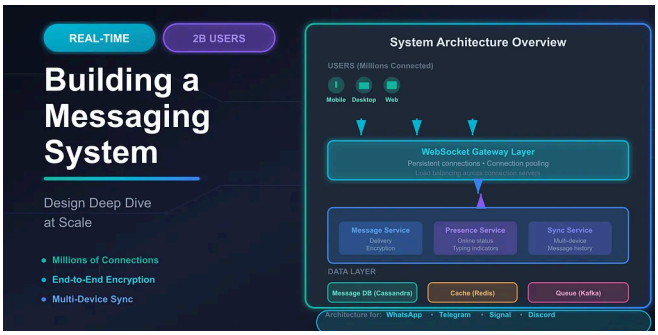


In FAUN.dev() 🐾 by Arvind Kumar

The Silent Earthquake in Tech: Why Every Engineer Must Rethink Thei...

Sharing my personal experience, thought process, and happenings around me.

Dec 13, 2025 482 20

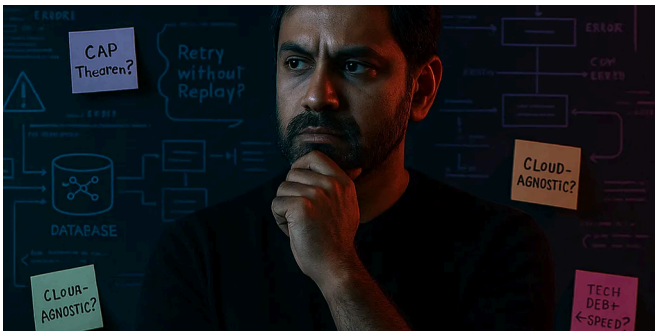


Arvind Kumar

Building a Real-Time Messaging System: Design Deep Dive

How do you deliver messages to 2 billion users in real-time? Let's design a messaging...

Dec 7, 2025 131 2



Arvind Kumar

12 Interview Questions That Separate Real Architects from...

"Staff Engineer or Just Staff Meeting Engineer?"

Jul 3, 2025 121 1

Arvind Kumar

Forward Proxy vs Reverse Proxy: The Deep Dive Every Engineer Mu...

There are few networking components as critical—and as misunderstood—as forwar...

Dec 4, 2025 188 2

See all from Arvind Kumar

Recommended from Medium

Suchirreddy

🎟️ Designing a Concurrency-Safe Ticket Booking System from...

🧩 Defining the Problem

Oct 20, 2025

Arvind Kumar

Common Tricky Questions in the Kafka Ecosystem

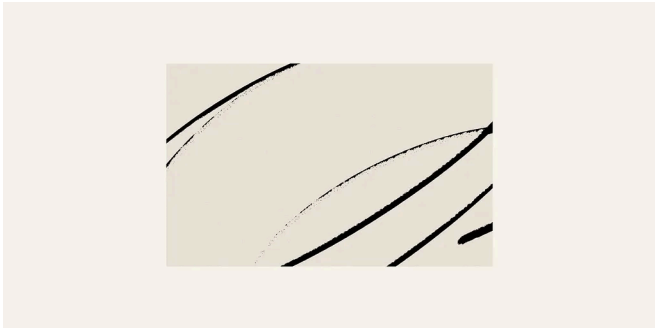
These common questions are part of the Kafka Bootcamp learning series—link

✦ 4d ago 25

Gaddam.Naveen

If You Know These 12 Microservice Design Patterns, You're Interview-...

if you are not a medium member then Click here to read free



Ram

Part 2: Master Kubernetes Interviews in 2026: Top 10 Essenti...

Kubernetes has evolved far beyond basic container orchestration and, by 2026, it has...

Dec 20, 2025 14 2

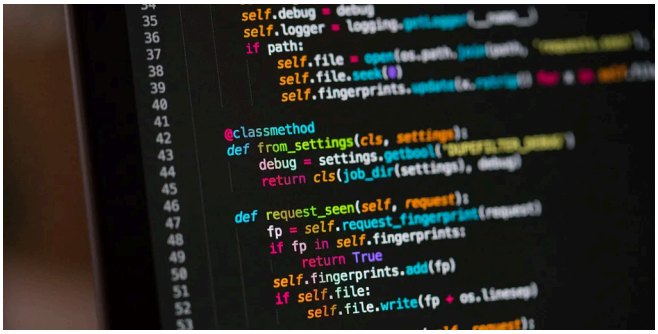
CoVaib DeepLearn

Day 10 : System Design Concept : Event Driven Architecture

What Is Event-Driven Architecture?

Jul 26, 2025

6d ago 9



In CodeX by Riya Sharma

2 Ways to Make Spring Boot Fully Serverless in 2026

A practical guide to running Spring Boot without servers, ops headaches, or idle costs

5d ago 83

See more recommendations