Apache Kafka:

Powering Real-Time Data Pipelines and Event-Driven Architectures

Agenda

01. **Introduction**

02. **Historical Background**

03. **Development and need for Kafka**

04. **Architecture**

05. **Demo**

Kafka

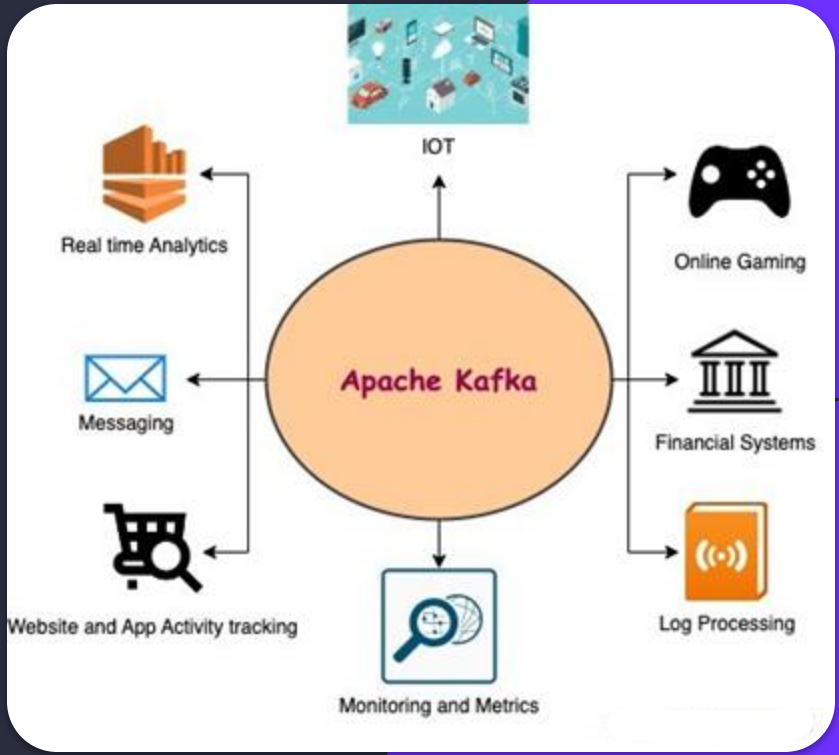
The topic of our term paper is Apache Kafka. It is an open-source distributed event streaming

platform used by thousands of companies for high-performance data pipelines, streaming

analytics, data integration, and mission-critical applications

Examples:

- Ingesting clickstream data from websites into storage (e.g., S3, Snowflake, BigQuery).
- Streaming IoT sensor data into analytics systems.
- Sending data from microservices to analytics dashboards.

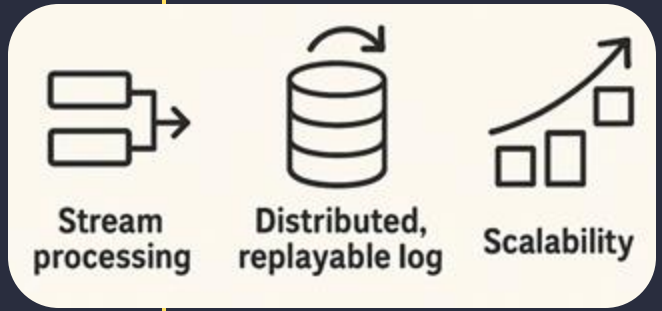


Kafka: Who, When and Why?

Kafka was originally **created at LinkedIn in 2010** by engineers Jay Kreps, Neha Narkhede, and Jun Rao, who were frustrated by the limitations of traditional messaging systems like ActiveMQ and RabbitMQ in handling the massive scale of LinkedIn's data pipelines.

ActiveMQ and RabbitMQ could handle thousands of messages per second, but LinkedIn needed to handle millions. These systems stored messages in memory or relational databases, causing severe slowdowns under high load. Kafka, in contrast, writes messages sequentially to disk and batches network I/O, allowing near–file-system-level performance.

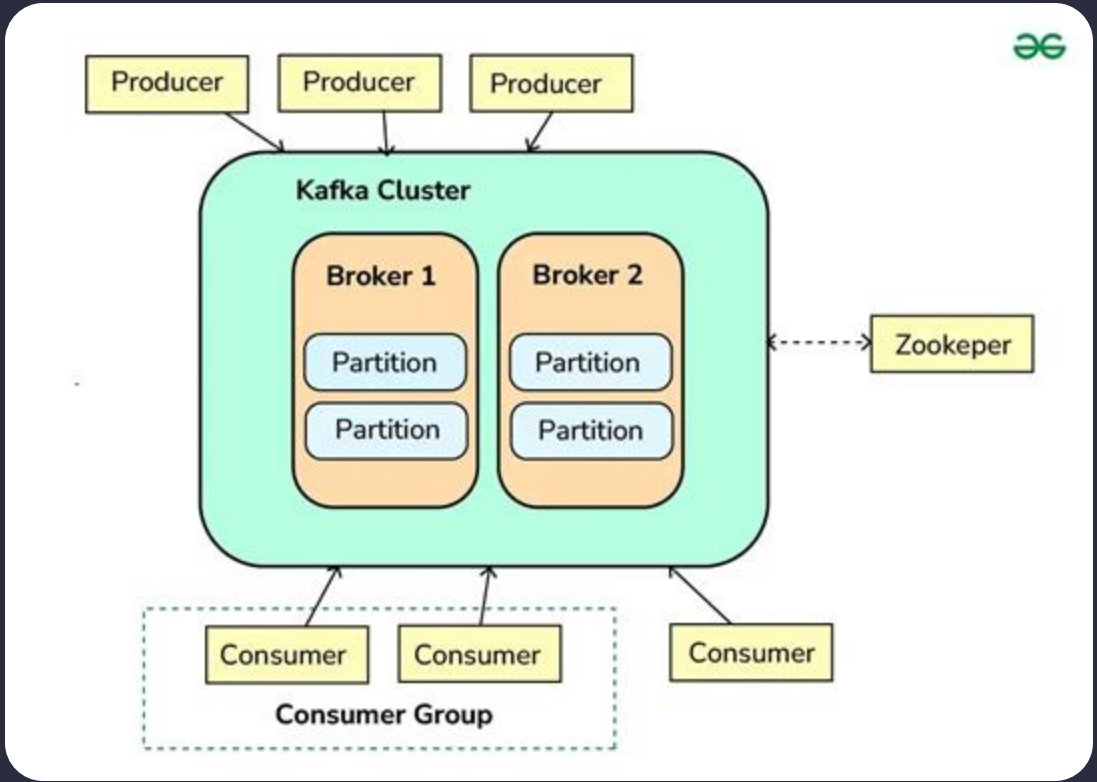
Their goal was to design a platform capable of ingesting billions of events per day, ranging from user activity logs and clickstream data to recommendation engine signals, while providing strong guarantees around scalability, durability, and fault tolerance. **This internal project was open-sourced under the Apache Software Foundation in 2011.**



Kafka: Who, When and Why?

Feature	Traditional Messaging	Kafka
Scalability	Single broker or small cluster	Horizontally scalable across servers
Storage	Deletes after consumption	Persistent, replayable log
Speed	Thousands/sec	Millions/sec
Use cases	Notifications, transactions	Real-time data pipelines, analytics, ML
Integration	One-to-one	One-to-many

Kafka Architecture



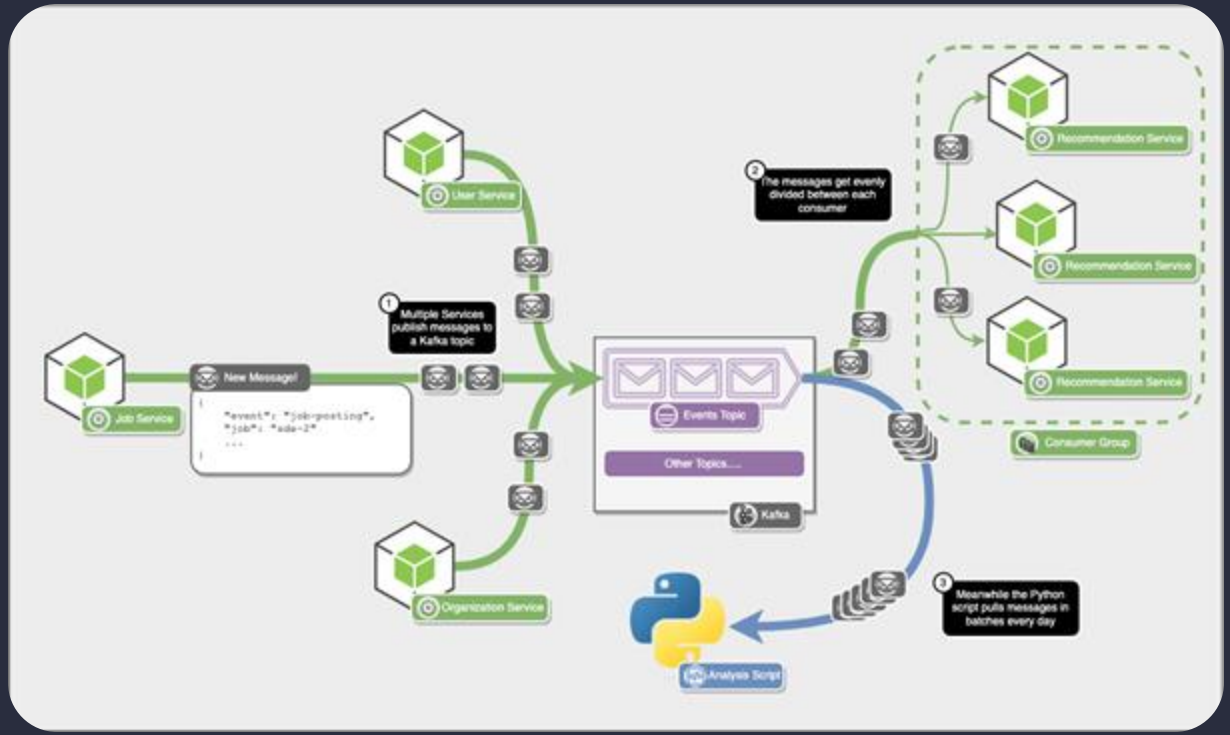
- **Producers** send data (events) into the **Kafka Cluster**.
- The **cluster** is made up of multiple **brokers** (servers) that store data.
- Each broker holds **partitions**, which keep messages in order for fast access.
- **Consumers** read data from these partitions, often as part of a **consumer group** to share the load.
- **ZooKeeper** (or **KRaft** in newer versions) manages coordination and health of brokers.

Together, this design makes Kafka **scalable, fault-tolerant, and real-time.**

Multiple services like the Job, User, and Organization services generate events (such as job postings) and **publish them to a Kafka topic**. Kafka acts as a **central event hub**, storing and streaming these messages in real time.

A **consumer group** (the Recommendation Service instances) subscribes to this topic, and Kafka **evenly distributes messages** across them for parallel processing.

Meanwhile, other consumers, like a **Python analysis script**, can independently **pull and process the same data later** — showing Kafka's ability to support both real-time and batch processing simultaneously.



Kafka Techniques & Approaches

1. Distributed Publish–Subscribe System
Kafka enables multiple producers and consumers to exchange data in real time through topics — allowing systems to remain loosely coupled and highly scalable.

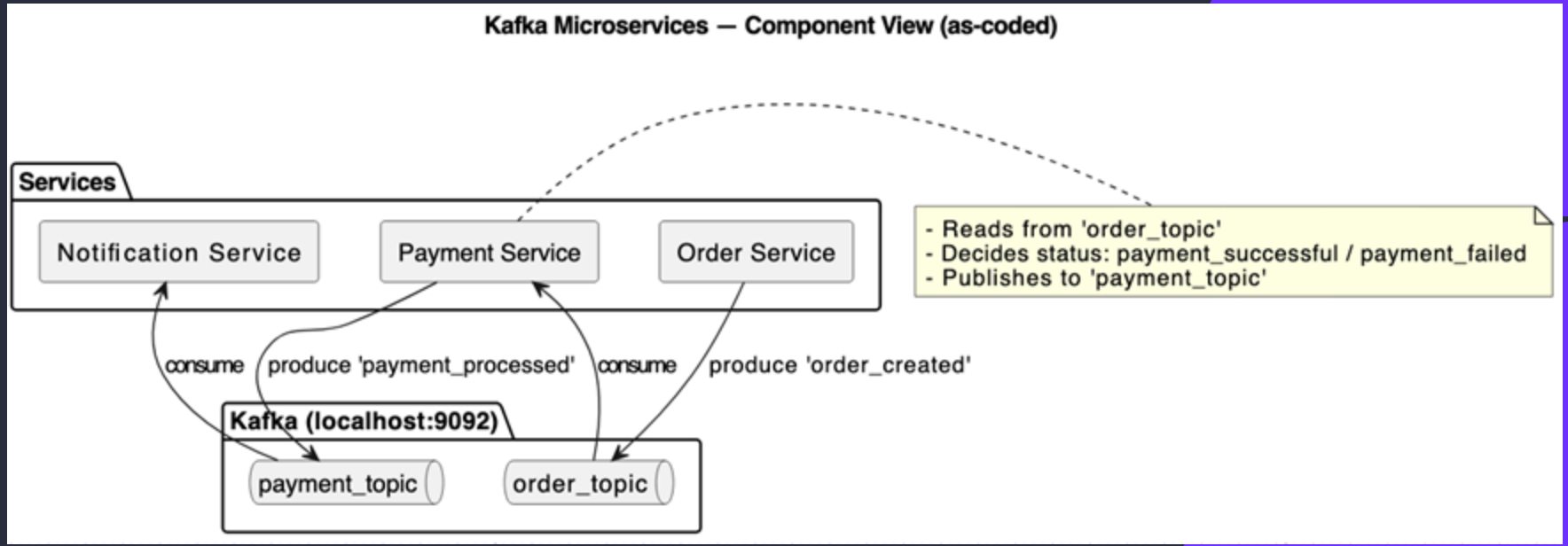
2. Partitioning for Parallelism
Topics are divided into partitions, letting multiple brokers handle data simultaneously. This ensures **high throughput** and **load balancing** across servers.

3. Replication for Reliability
Each partition has replicas across brokers to prevent data loss and support **fault-tolerant recovery** in case of hardware or network failures.

4. Stream Processing Capability
Kafka Streams API supports filtering, aggregating, and transforming messages in real time, turning raw events into actionable insights.

5. Seamless Integration
Kafka connects easily with **Spark, Flink, and Hadoop ecosystems**, making it central to **real-time analytics and event-driven architectures**.

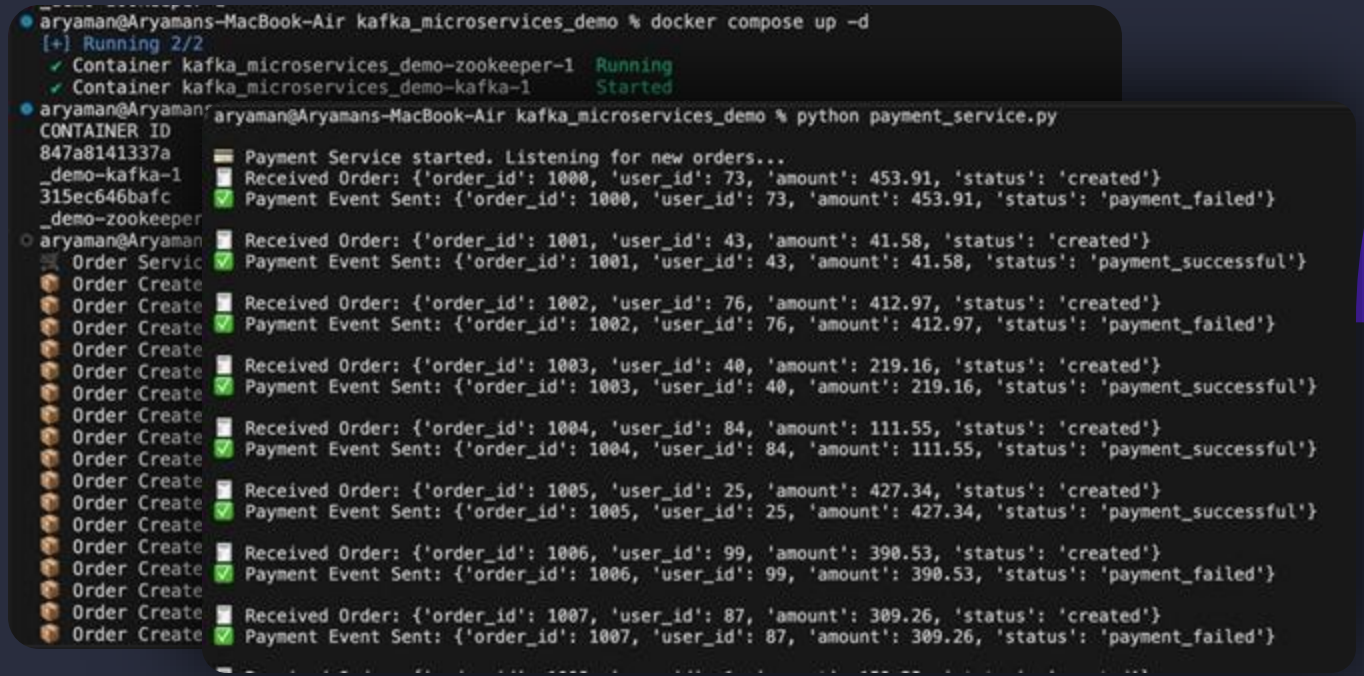
Demo Architecture



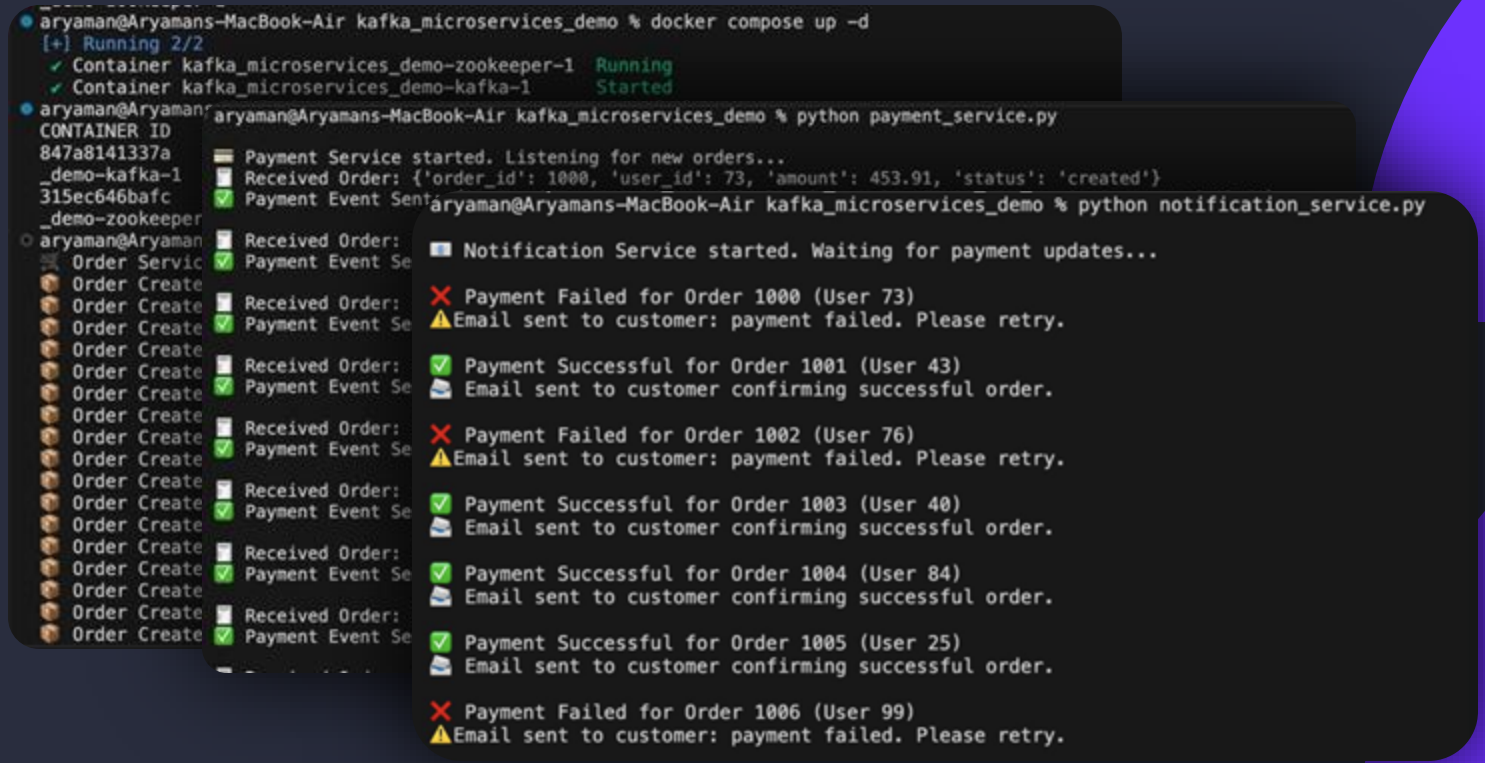
Sample Output

```
aryaman@Aryamans-MacBook-Air kafka_microservices_demo % docker compose up -d
[+] Running 2/2
 ✔ Container kafka_microservices_demo-zookeeper-1  Running
 ✔ Container kafka_microservices_demo-kafka-1      Started
aryaman@Aryamans-MacBook-Air kafka_microservices_demo % docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED      STATUS
847a8141337a   confluentinc/cp-kafka:7.6.1       "/etc/confluent/dock…"   8 days ago   Up 6 seconds
_demo-kafka-1
315ec646bafc   confluentinc/cp-zookeeper:7.6.1   "/etc/confluent/dock…"   8 days ago   Up 3 minutes
_demo-zookeeper-1
aryaman@Aryamans-MacBook-Air kafka_microservices_demo % python order_service.py
🛒 Order Service started. Generating new orders...
📦 Order Created: {'order_id': 1000, 'user_id': 73, 'amount': 453.91, 'status': 'created'}
📦 Order Created: {'order_id': 1001, 'user_id': 43, 'amount': 41.58, 'status': 'created'}
📦 Order Created: {'order_id': 1002, 'user_id': 76, 'amount': 412.97, 'status': 'created'}
📦 Order Created: {'order_id': 1003, 'user_id': 40, 'amount': 219.16, 'status': 'created'}
📦 Order Created: {'order_id': 1004, 'user_id': 84, 'amount': 111.55, 'status': 'created'}
📦 Order Created: {'order_id': 1005, 'user_id': 25, 'amount': 427.34, 'status': 'created'}
📦 Order Created: {'order_id': 1006, 'user_id': 99, 'amount': 390.53, 'status': 'created'}
📦 Order Created: {'order_id': 1007, 'user_id': 87, 'amount': 309.26, 'status': 'created'}
📦 Order Created: {'order_id': 1008, 'user_id': 1, 'amount': 159.22, 'status': 'created'}
📦 Order Created: {'order_id': 1009, 'user_id': 61, 'amount': 17.69, 'status': 'created'}
📦 Order Created: {'order_id': 1010, 'user_id': 95, 'amount': 50.73, 'status': 'created'}
📦 Order Created: {'order_id': 1011, 'user_id': 86, 'amount': 295.61, 'status': 'created'}
📦 Order Created: {'order_id': 1012, 'user_id': 6, 'amount': 398.75, 'status': 'created'}
📦 Order Created: {'order_id': 1013, 'user_id': 81, 'amount': 115.58, 'status': 'created'}
📦 Order Created: {'order_id': 1014, 'user_id': 82, 'amount': 322.48, 'status': 'created'}
📦 Order Created: {'order_id': 1015, 'user_id': 81, 'amount': 22.5, 'status': 'created'}
📦 Order Created: {'order_id': 1016, 'user_id': 63, 'amount': 459.19, 'status': 'created'}
```

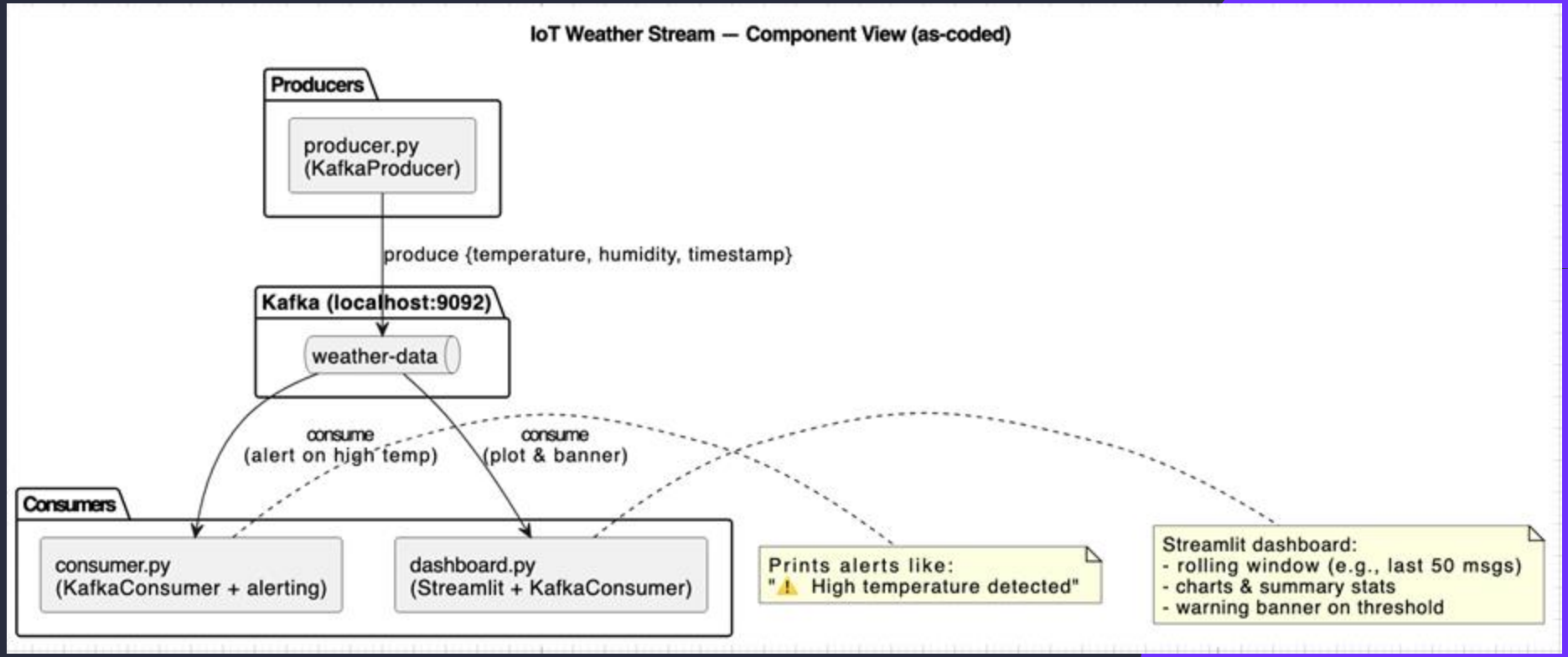
Sample Output



Sample Output



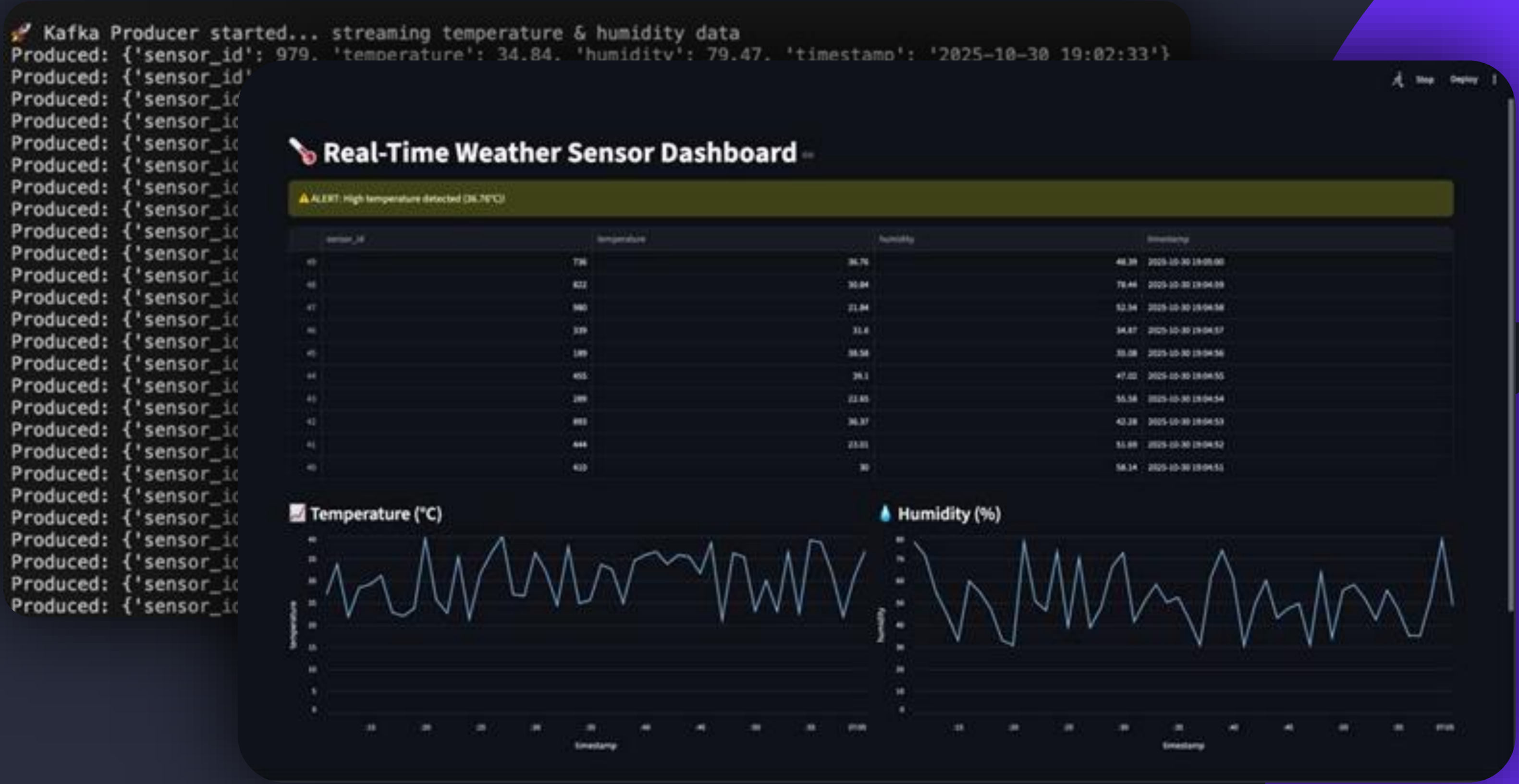
Demo Architecture



Sample Output

```
🚀 Kafka Producer started... streaming temperature & humidity data
Produced: {'sensor_id': 979, 'temperature': 34.84, 'humidity': 79.47, 'timestamp': '2025-10-30 19:02:33'}
Produced: {'sensor_id': 387, 'temperature': 31.33, 'humidity': 72.8, 'timestamp': '2025-10-30 19:02:34'}
Produced: {'sensor_id': 198, 'temperature': 35.67, 'humidity': 66.24, 'timestamp': '2025-10-30 19:02:35'}
Produced: {'sensor_id': 616, 'temperature': 32.38, 'humidity': 60.81, 'timestamp': '2025-10-30 19:02:36'}
Produced: {'sensor_id': 228, 'temperature': 28.62, 'humidity': 62.56, 'timestamp': '2025-10-30 19:02:37'}
Produced: {'sensor_id': 243, 'temperature': 23.51, 'humidity': 41.97, 'timestamp': '2025-10-30 19:02:38'}
Produced: {'sensor_id': 228, 'temperature': 21.27, 'humidity': 75.02, 'timestamp': '2025-10-30 19:02:39'}
Produced: {'sensor_id': 436, 'temperature': 23.1, 'humidity': 50.03, 'timestamp': '2025-10-30 19:02:40'}
Produced: {'sensor_id': 492, 'temperature': 37.15, 'humidity': 36.14, 'timestamp': '2025-10-30 19:02:41'}
Produced: {'sensor_id': 224, 'temperature': 32.47, 'humidity': 49.07, 'timestamp': '2025-10-30 19:02:42'}
Produced: {'sensor_id': 540, 'temperature': 38.71, 'humidity': 31.43, 'timestamp': '2025-10-30 19:02:43'}
Produced: {'sensor_id': 635, 'temperature': 25.99, 'humidity': 39.88, 'timestamp': '2025-10-30 19:02:44'}
Produced: {'sensor_id': 592, 'temperature': 21.73, 'humidity': 47.49, 'timestamp': '2025-10-30 19:02:45'}
Produced: {'sensor_id': 847, 'temperature': 23.04, 'humidity': 66.43, 'timestamp': '2025-10-30 19:02:46'}
Produced: {'sensor_id': 597, 'temperature': 30.83, 'humidity': 65.59, 'timestamp': '2025-10-30 19:02:47'}
Produced: {'sensor_id': 593, 'temperature': 34.57, 'humidity': 54.17, 'timestamp': '2025-10-30 19:02:48'}
Produced: {'sensor_id': 785, 'temperature': 39.26, 'humidity': 51.04, 'timestamp': '2025-10-30 19:02:49'}
Produced: {'sensor_id': 362, 'temperature': 39.89, 'humidity': 31.26, 'timestamp': '2025-10-30 19:02:50'}
Produced: {'sensor_id': 541, 'temperature': 22.99, 'humidity': 74.48, 'timestamp': '2025-10-30 19:02:51'}
Produced: {'sensor_id': 495, 'temperature': 39.63, 'humidity': 36.27, 'timestamp': '2025-10-30 19:02:52'}
Produced: {'sensor_id': 469, 'temperature': 37.1, 'humidity': 71.31, 'timestamp': '2025-10-30 19:02:53'}
Produced: {'sensor_id': 566, 'temperature': 39.72, 'humidity': 36.07, 'timestamp': '2025-10-30 19:02:54'}
Produced: {'sensor_id': 439, 'temperature': 27.7, 'humidity': 61.46, 'timestamp': '2025-10-30 19:02:55'}
Produced: {'sensor_id': 809, 'temperature': 38.51, 'humidity': 77.44, 'timestamp': '2025-10-30 19:02:56'}
Produced: {'sensor_id': 400, 'temperature': 35.45, 'humidity': 57.46, 'timestamp': '2025-10-30 19:02:57'}
Produced: {'sensor_id': 678, 'temperature': 31.51, 'humidity': 60.09, 'timestamp': '2025-10-30 19:02:58'}
```

Sample Output



Thank you!