Apache Kafka from zero to semi hero

Fincons Tech Hub
Java Spring Time

Francesco Lacriola

Software Eng. – Senior Analyst

Daniele Gallitelli

Software Eng. – Senior Analyst

Michele Caccia

Senior Software Eng. – Senior Consultant

What we’ll do today

- Introduction to Apache Kafka
- Build a real world, multi-service event-driven system
  - Simulates a fleet of trains sending real-time location data
  - Processes data to generate insights
  - Visualizes everything on a live dashboard
  - Uses multiple programming languages (Java & Python)
- Q/A

What is Apache Kafka

Distributed streaming platform for building real-time data pipelines

- Originally developed by LinkedIn, now open-source (Apache)
- **High-throughput message broker**, allowing systems to publish, store, and consume **streams of records** in a **fault-tolerant and scalable** way
- **Durable storage** — Data is persisted on brokers disk, allowing replay and recovery.

Key Features

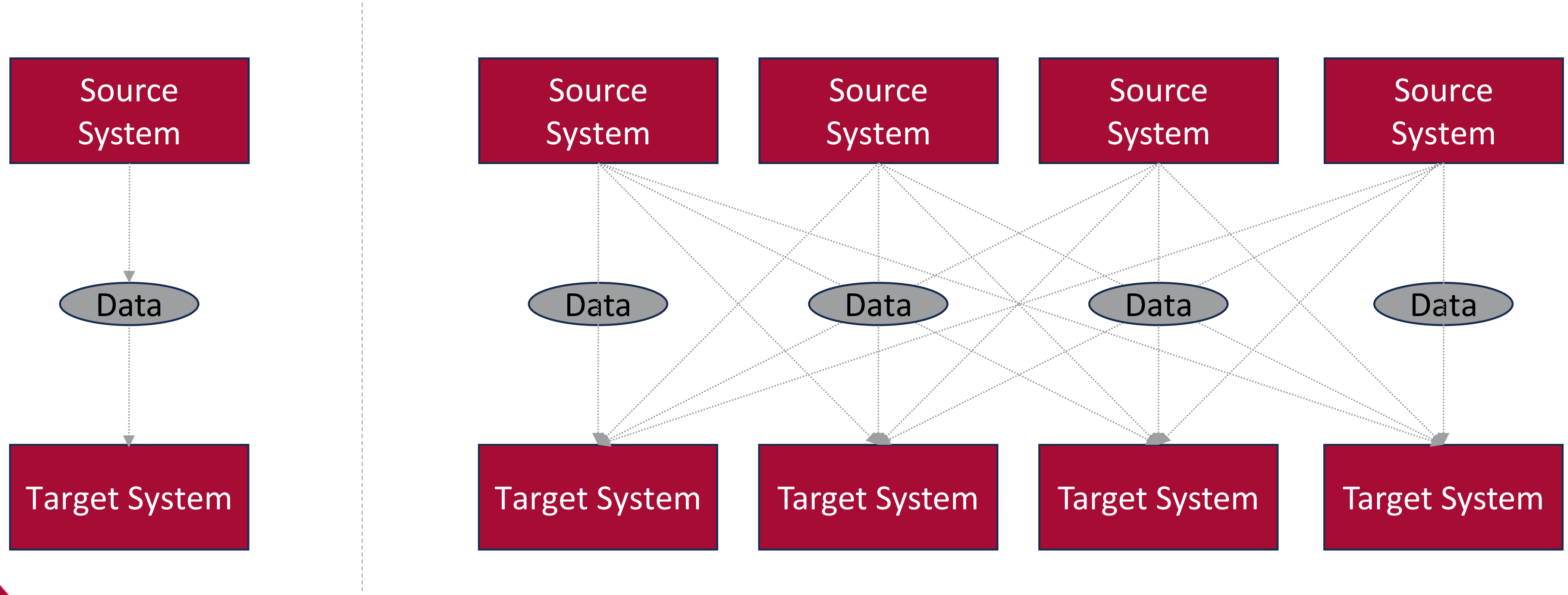
✓ High throughput & low latency

✓ Fault-tolerant & scalable

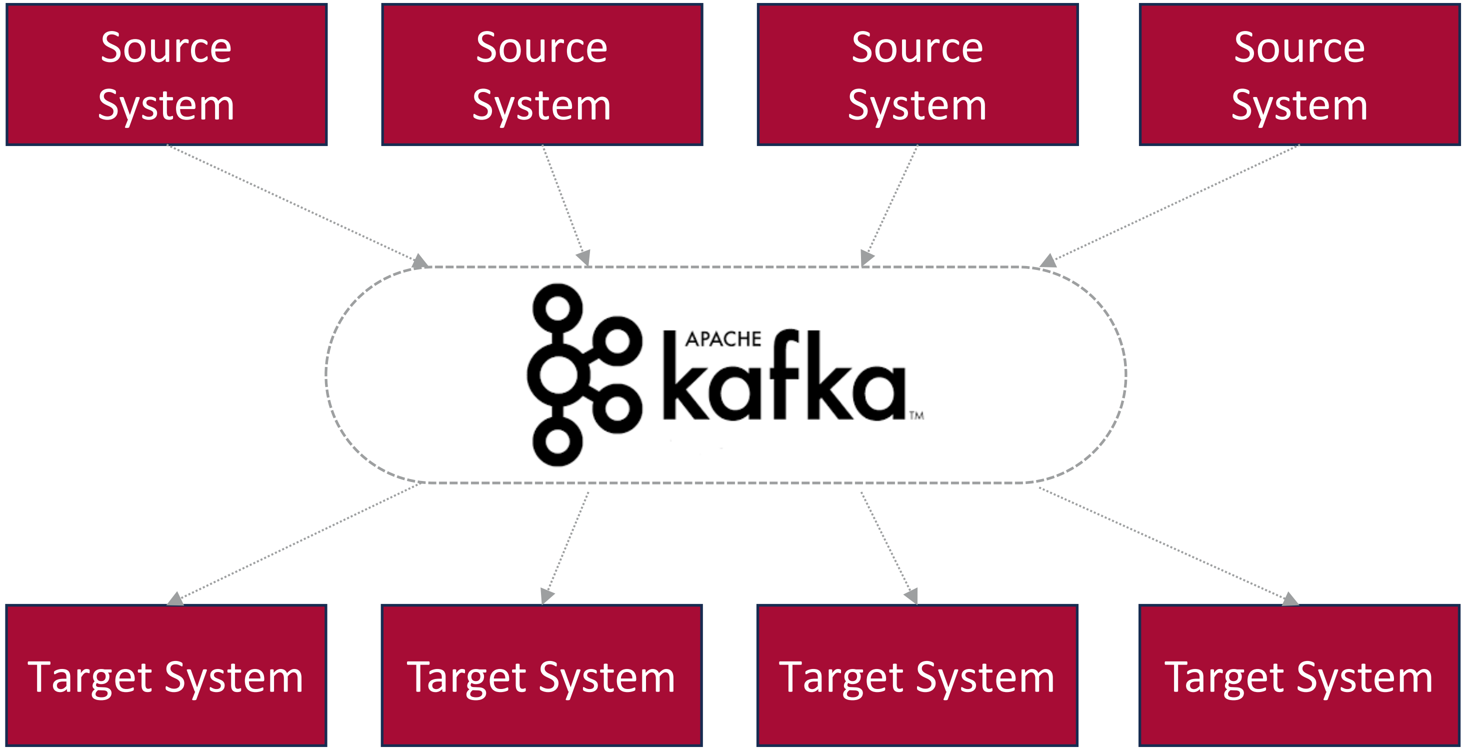
✓ Persistent storage

✓ Real-time processing

What is Apache Kafka



What is Apache Kafka



Source System

Source System

Source System

Source System

APACHE kafka

Target System

Target System

Target System

Target System

Where is Kafka used

Used by 2000+ firms, 80% of Fortune 100

- **SBB:** Yes — we use Kafka too! (Master Data propagation, connected train telemetry and real-time services)
- **Netflix:** Real-time recommendations & monitoring
- **Uber:** Real-time pricing, trip tracking
- **LinkedIn:** Activity streams, operational metrics
- **PayPal:** Risk detection, fraud prevention
- **Banking:** Transaction processing, fraud detection
- **E-commerce:** Inventory management, order processing

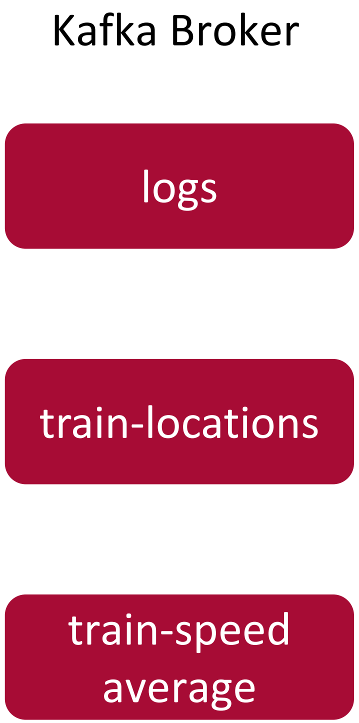
Topics: Your Data Channels

Topic = A particular stream of data published in your Kafka Cluster

- Like a table in a database or a folder in a filesystem
- Topics are **multi-subscriber** (many applications can read the same topic)
- A topic is identified by its name
- It is impossible to query topics. Use Kafka Producers to send data and Kafka Consumers to read the data

Examples in our workshop:

- train-locations - raw position data from trains
- train-speed-averages - processed speed analytics



Kafka Broker

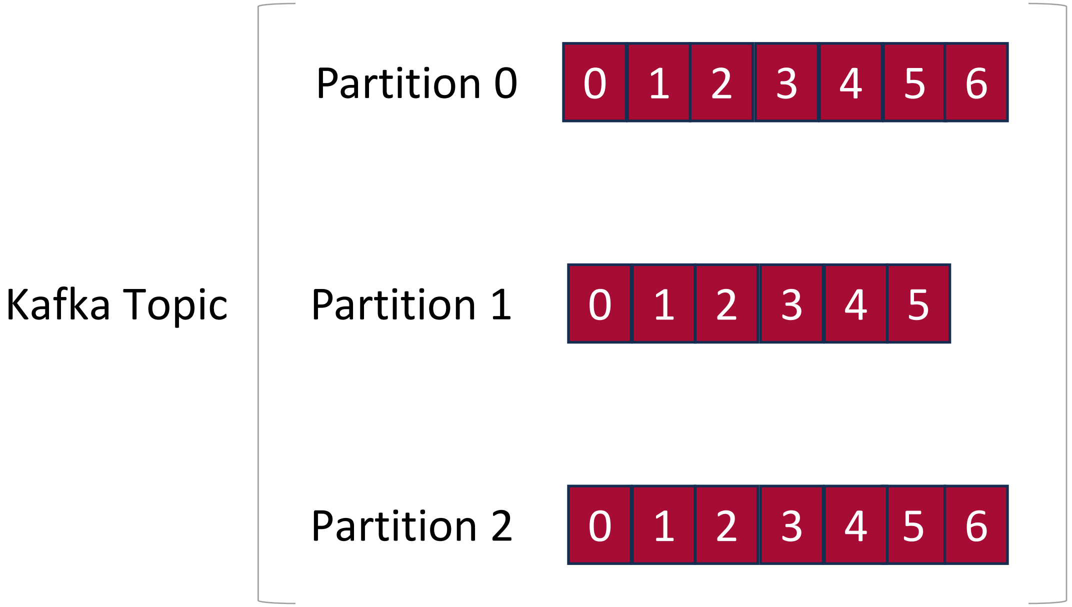
logs

train-locations

train-speed average

Partitions and offsets

- Each topic is divided into **partitions**
- Partitions allow parallel processing
- Messages in a partition can be **ordered**
- Each message has an **offset** (sequential ID)
- Kafka topics are **immutable**
- Our train-locations topic has **3 partitions**



Kafka Topic

Partition 0: 0 1 2 3 4 5 6

Partition 1: 0 1 2 3 4 5

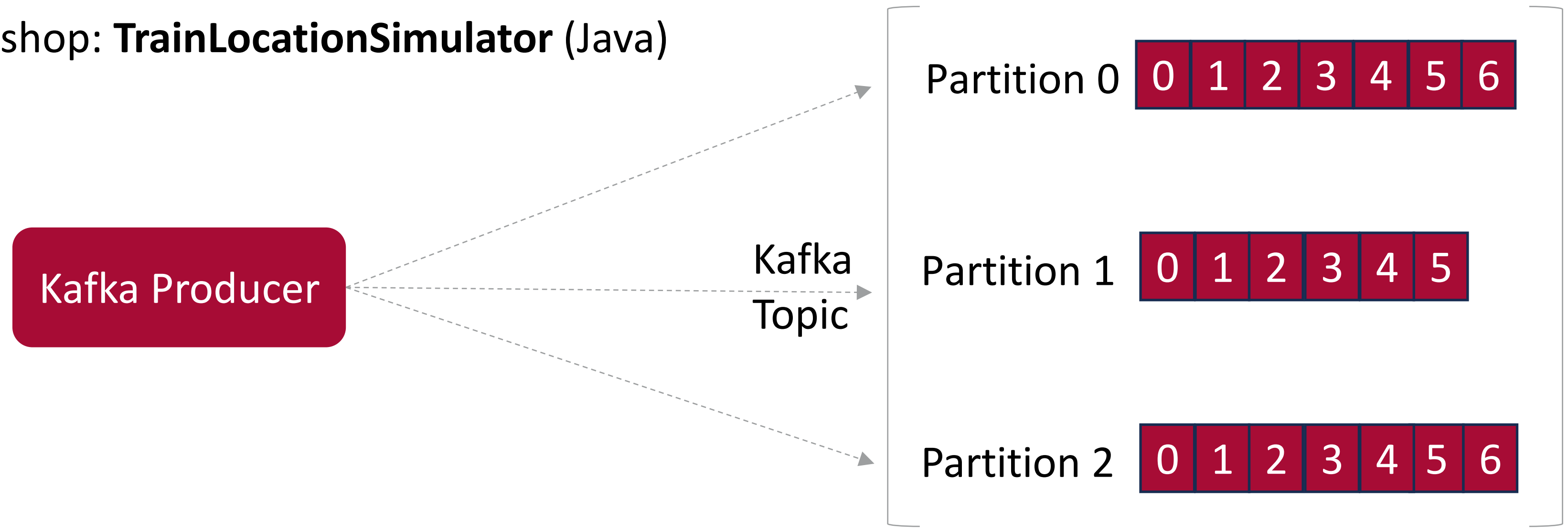
Partition 2: 0 1 2 3 4 5 6

```
        Topic: train-locations
├── Partition 0: [msg0, msg3, msg6, ...]
├── Partition 1: [msg1, msg4, msg7, ...]
└── Partition 2: [msg2, msg5, msg8, ...]
```

Producers: Publishing Data

Applications that **publish** (write) records to topics

- Specify which partition to send to (and which Kafka broker has it)
- Can use **keys** for consistent routing
- In our workshop: **TrainLocationSimulator** (Java)



Example:

```
Train T-123 → partition based on train ID → always same partition
```

This ensures all messages from the same train stay in order!

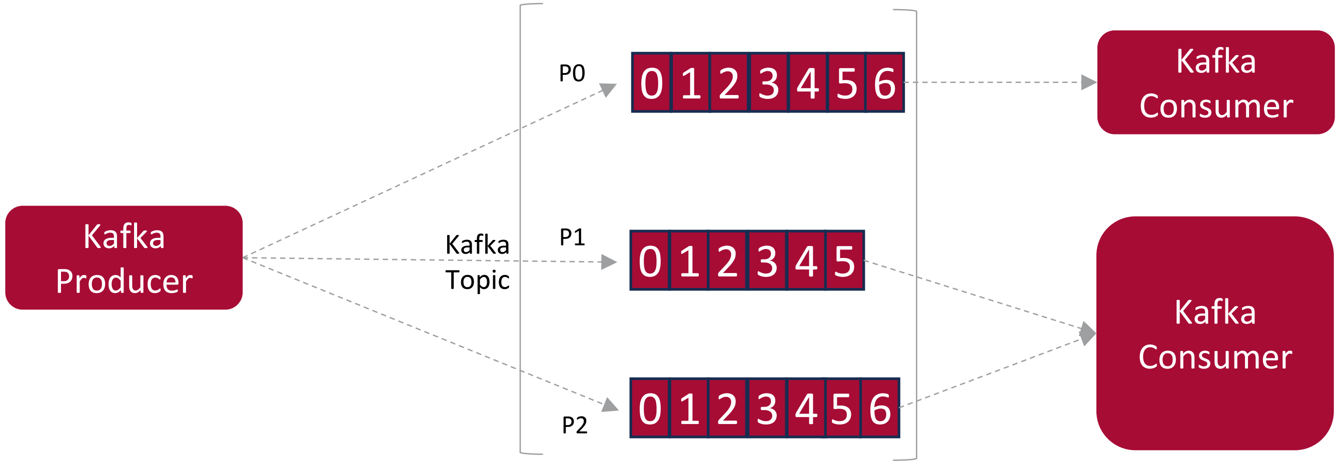
Consumers: Reading Data

Applications that **subscribe** to topics and process records (pull model)

- Maintain an **offset** to track what they've read -> Consumers know which broker to read from
- Can rewind and replay messages

In our workshop:

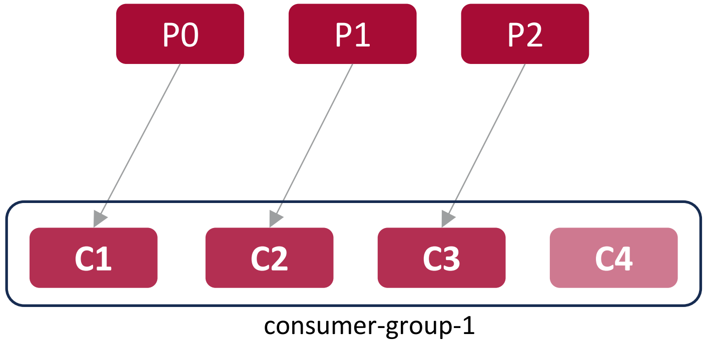
- **SpeedAnalysisStream** (Kafka Streams)
- **DashboardWebApp** (Java)
- **MaintenanceAlerter** (Python)



Consumer Groups: Teamwork!

Multiple consumers working together as a **group**

- Each partition assigned to **only one** consumer in the group
- Enables **parallel processing** and **load balancing**
- If a consumer fails, its partitions are reassigned → **fault tolerance**

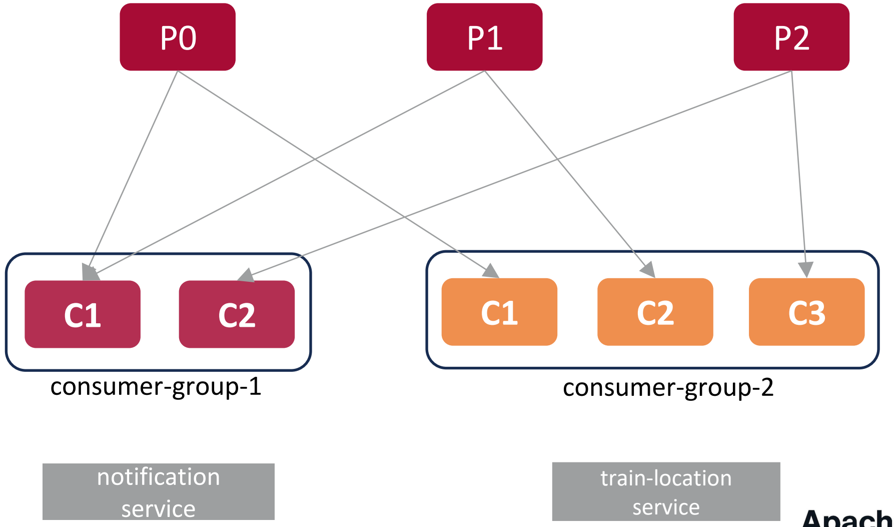


Benefits:

✓ Same group = share the workload

✓ Different groups = each gets all messages

✓ Zero message loss

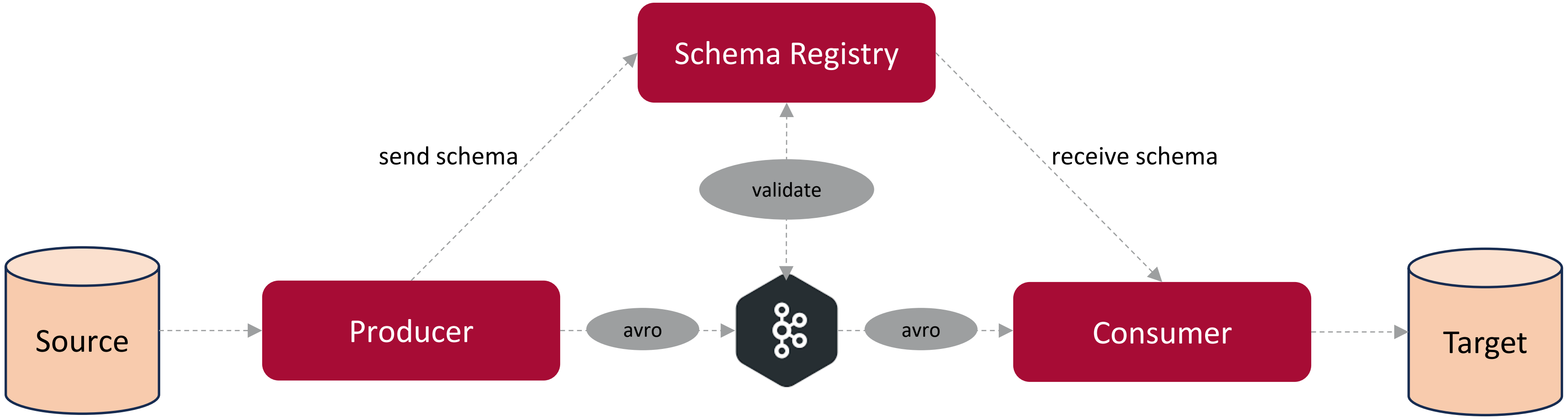


Schema Registry

Apache Kafka doesn’t validate data streams

- What if a producer sends bad data?
- What if a field gets renamed?
- What if the data format changes from one day to another?

→ **consumers breaks!**



Apache Avro

Data Serialization Framework

efficient, compact, binary format

Schema Evolution

backward & forward compatibility

Readability

schema readable (JSON)
Data is not (binary)

Schema-based

JSON-defined schemas & ensures data consistency

Kafka vs RabbitMQ/ActiveMQ 🥊

Kafka Advantages:

✅ Higher throughput (millions of msgs/sec)
✅ Persistent storage (replay capability)
✅ Horizontal scalability
✅ Built for streaming & big data

Traditional Messaging:

❎ Complex routing (exchanges, bindings)
❎ Lower latency for small messages
❎ Message priority queues
❎ Simpler request/reply patterns

When to use Kafka:
📌 High volume event streaming
📌 Log aggregation
📌 Real-time analytics
📌 Event sourcing

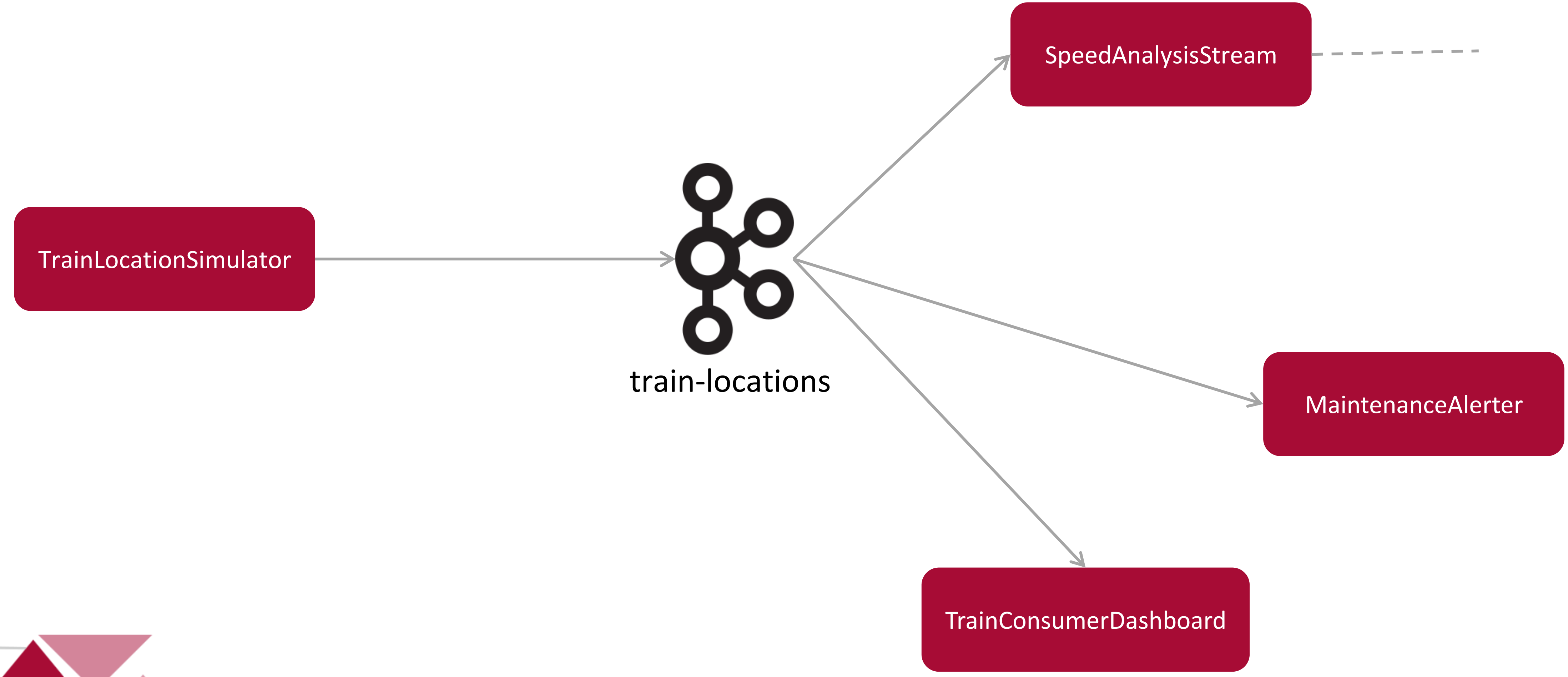
Our Workshop Architecture 🏗️

Services:

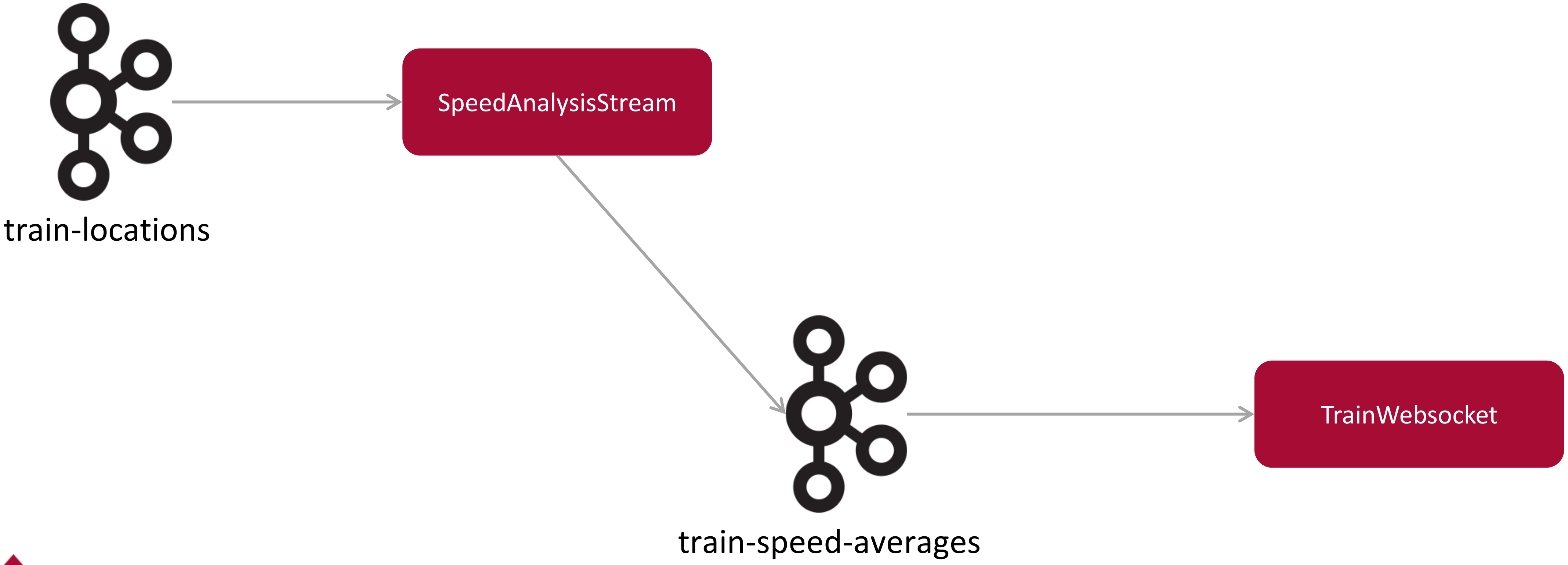
1. **Producer:** TrainLocationSimulator (Java) - generates train positions
2. **Maintenance Alerter:** MaintenanceAlerter (Python) - slow train alerts
3. **Dashboard Consumer:** TrainConsumerDashboard (Java) - Demonstrates partition rebalancing
4. **Stream Processor:** SpeedAnalysisStream (Java, Kafka Streams) - calculates averages
5. **WebSocket Dashboard:** TrainWebsocket (Java, WebSocket) - real-time visualization

All connected through Kafka topics! 🔄

Our Workshop Architecture: topic train-locations 🏗️



Our Workshop Architecture: topic train-speed-averages 🏗️



train-locations

SpeedAnalysisStream

train-speed-averages

TrainWebsocket

Kafka is Language Agnostic 🌐

Any language can produce/consume from Kafka:

☕ Java (most common, best support)
🐍 Python
🟧 JavaScript/Node.js
🔵 Go
🟣 C#/.NET
And many more...

Workshop Demonstration:

✅ Java producers and consumers

✅ Python consumer (maintenance alerter)

✅ Both read from the **same topic** seamlessly

Kafka Streams 🔄

Library for building stream processing applications

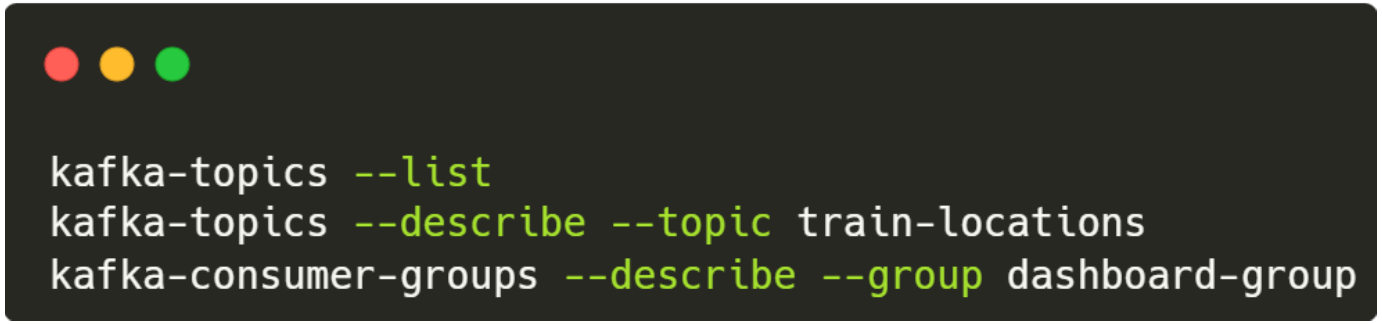
- Processes data **as it arrives** (real-time)
- Supports transformations, aggregations, joins, windowing
- No separate cluster needed

Our Use Case:

1. Read from `train-locations`
2. Calculate average speed per train (10-second window)
3. Write to `train-speed-averages`

Kafka Monitoring Tools 📊

Command Line Tools:



```
kafka-topics --list
kafka-topics --describe --topic train-locations
kafka-consumer-groups --describe --group dashboard-group
```

Web Interface:

- **Kafka UI** (localhost:8099, shipped with our containers 🎉)
  - Monitor topics, partitions, consumer groups
  - View messages in real-time
  - Track consumer lag
- **RedPanda**
- … and many more!

Common Patterns

Event Sourcing

🧾 Store every change to application state as an immutable sequence of events.

🧩 Kafka acts as the **event log** — a single source of truth.

⏳ Enables **time-travel debugging**, replay, and rebuilding state at any point.

CQRS (Command Query Responsibility Segregation)

✍️ Separate **commands (writes)** from **queries (reads)** for scalability and clarity.

💡 Kafka Streams can manage the **write-side events**, while **materialized views** serve reads.

⚖️ Perfect for systems with **uneven read/write loads**.

Common Patterns

Data Pipelines

🔗 Kafka connects **producers and consumers** in real-time data flow.

⚙️ Simplifies ETL: **ingest → process → distribute**, no batch delays.

📊 Powers **streaming analytics** and long-term data storage.

Microservices Communication

📨 Kafka decouples services with **asynchronous, event-driven** messaging.

🧠 Reduces reliance on REST calls, improving **resilience** and **fault tolerance**.

🌱 Promotes **loose coupling** and **scalable** architecture.

Production Best Practices 🎯

Design:

📦 Use shared libraries for data models (avoid duplication)

🗄️ Schema Registry for data contracts (Avro, JSON Schema)

📌 Check avro services in the repository for more!

🔄 Version your message formats

🎯 Use message keys for partitioning logic (same key = same partition)

⚡ Configure appropriate retention policies

Execution:

📊 Monitor consumer lag

⚖️ Right-size partition count

🔒 Enable authentication & encryption (production)

💾 Configure replication factor

🧪 Test rebalancing scenarios

Hands-On Time! 🛠️

What You'll Do:

1. ✅ Set up Kafka infrastructure (Docker)
2. ✅ Start the train location producer
3. ✅ Run the stream processor
4. ✅ Launch the WebSocket dashboard
5. ✅ Add Python maintenance alerter
6. ✅ Test partition rebalancing with scaling

Prerequisites check:

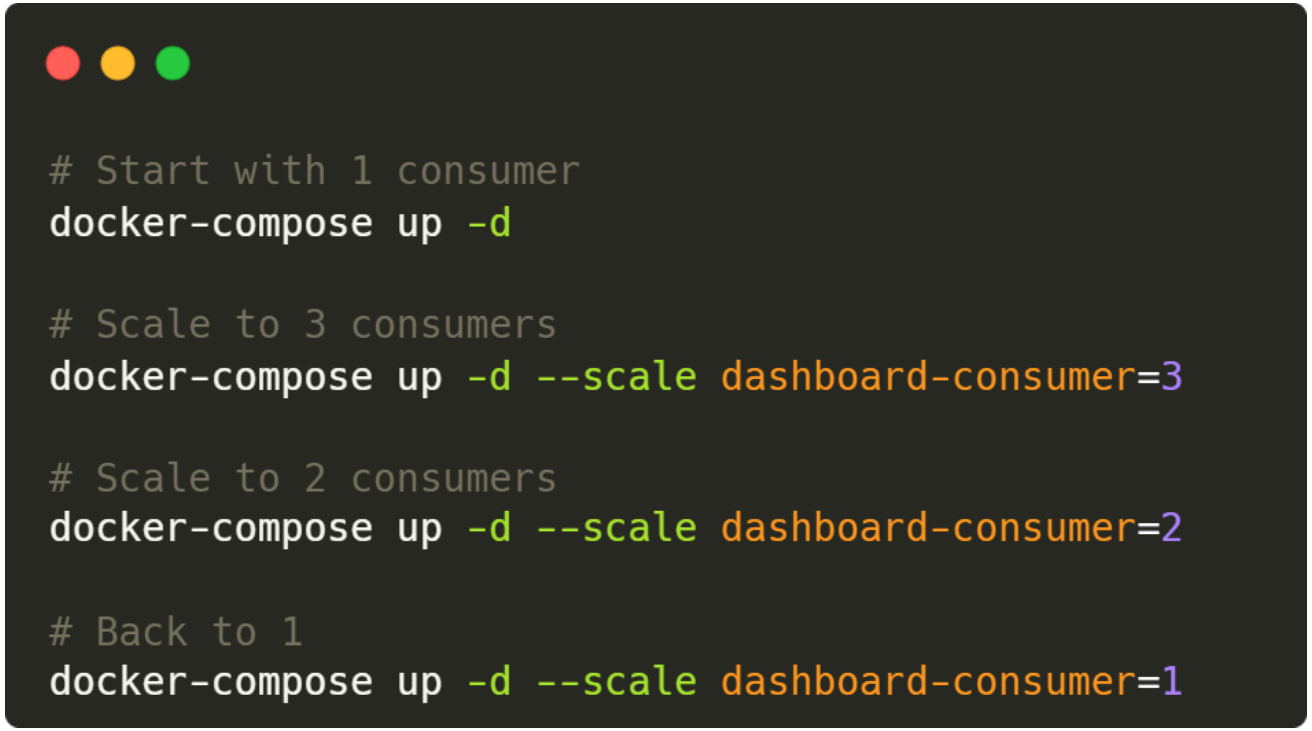
🐳 Docker & Docker Compose

If you want to build and run it yourself:

☕ Java JDK 17+ • 🔨 Maven • 🐍 Python 3.8+

Partition Rebalancing in Action ⚖️

Live Scaling Demo:



```
# Start with 1 consumer
docker-compose up -d

# Scale to 3 consumers
docker-compose up -d --scale dashboard-consumer=3

# Scale to 2 consumers
docker-compose up -d --scale dashboard-consumer=2

# Back to 1
docker-compose up -d --scale dashboard-consumer=1
```

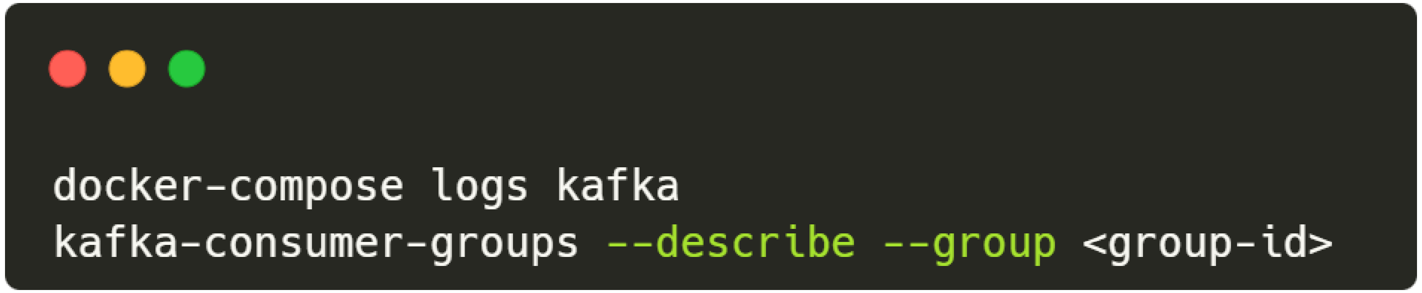
Watch: Partition reassignment • Rebalancing events • Zero message loss

Avoiding Common Mistakes ⚠️

Top Issues:

1. 🔌 **Connection refused:** Kafka not running → `docker ps`
2. 🔄 **Deserialization errors:** Mismatched serializers
3. 📈 **Consumer lag:** Too few consumers for partition count
4. 🔑 **Wrong consumer group:** Competing consumers
5. ⏱️ **Offset reset issues:** Check `auto.offset.reset` config

Debug Commands:



```
docker-compose logs kafka
kafka-consumer-groups --describe --group <group-id>
```

Continue Your Kafka Journey 🚀

Deep Dive Topics:

- 📊 Advanced Kafka Streams (joins, state stores)
- 🔒 Security (SSL, SASL, ACLs)
- 🌍 Multi-datacenter replication
- ⚡ Performance tuning
- 🗄️ Schema evolution strategies

Learning Resources:

- 📚 Confluent Documentation
- 📖 "Kafka: The Definitive Guide" (O'Reilly)
- 🎓 Confluent Developer courses
- 🐙 Apache Kafka official docs

Q/A

Thank You!

Workshop repository: GitHub

Call to Action:
Star the repository
Try building your own use case
Share your experience

Happy Streaming!