Event Driven Software

Apache Kafka basics

Presenter

Timot Tarjani

Software Architect

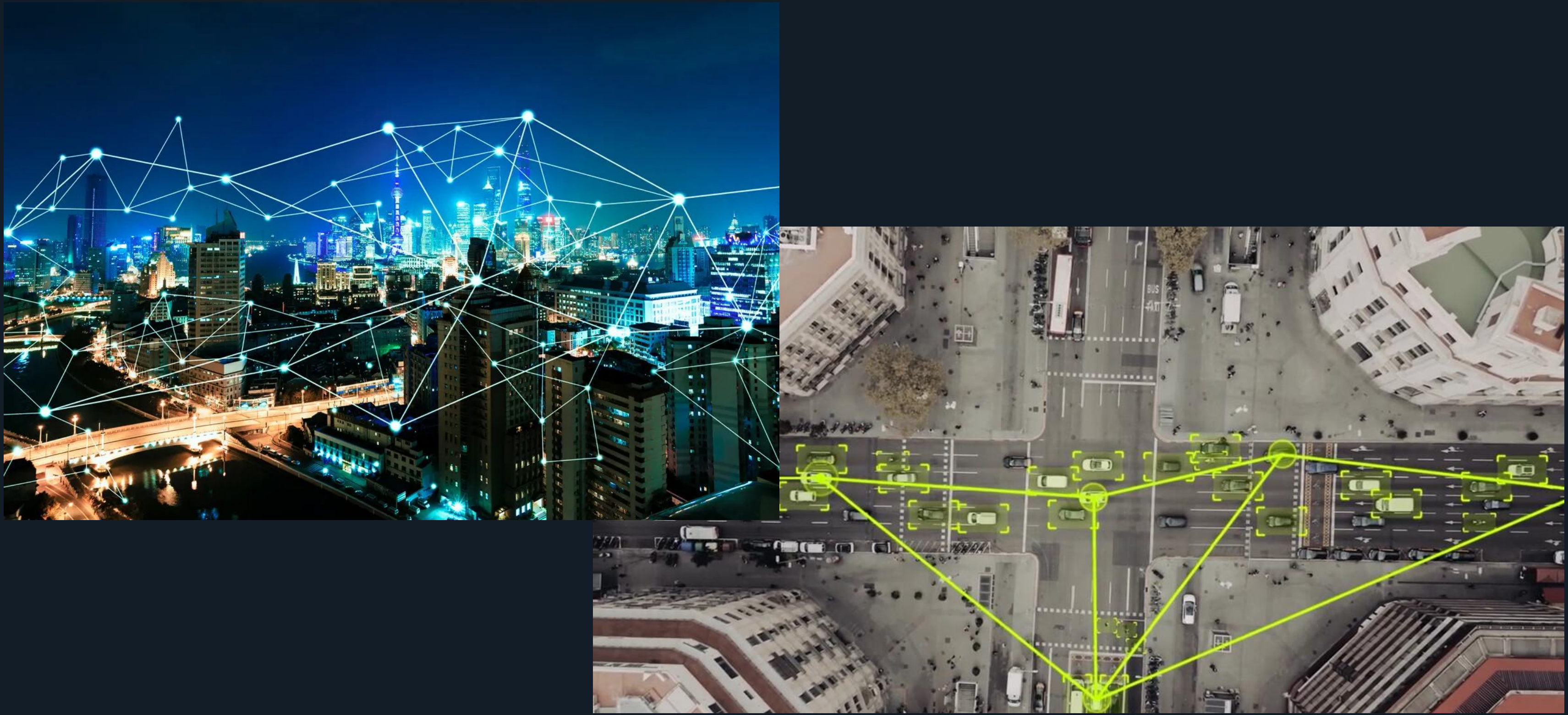
commsignia

Agenda

1. Events
2. Architectures
3. Apache Kafka
4. Code in Java

commsignia

Real world events



Events in computing context

- An event, in a computing context, is an action or occurrence that can be identified by a program and has significance for system hardware or software.
- Events can be user-generated, such as keystrokes and mouse clicks, or system-generated, such as program loading, running out of memory and errors, or information collected by sensors.
- In conclusion an event is any state change that indicates that something has happened

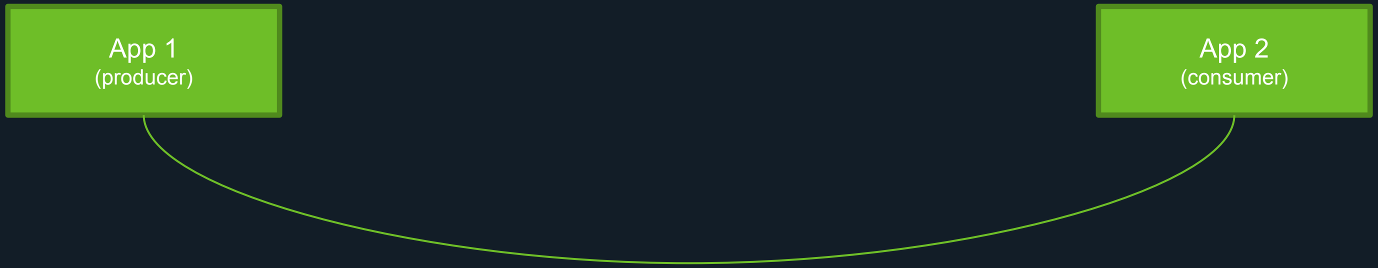
Producer & Consumer

- **Producer** - Actors who producing and publishing events

- **Consumer** - Actors who subscribing to events they are interested in

commsignia

Producer & Consumer services

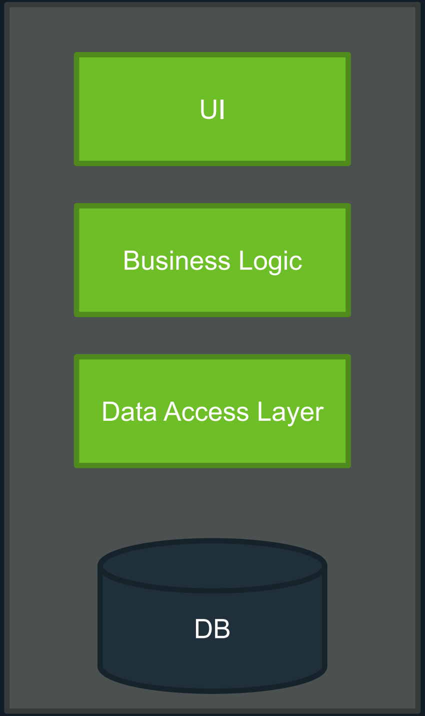


service 1 needs to know about
service 2 to be able to notify

Monolithic architecture

Strengths

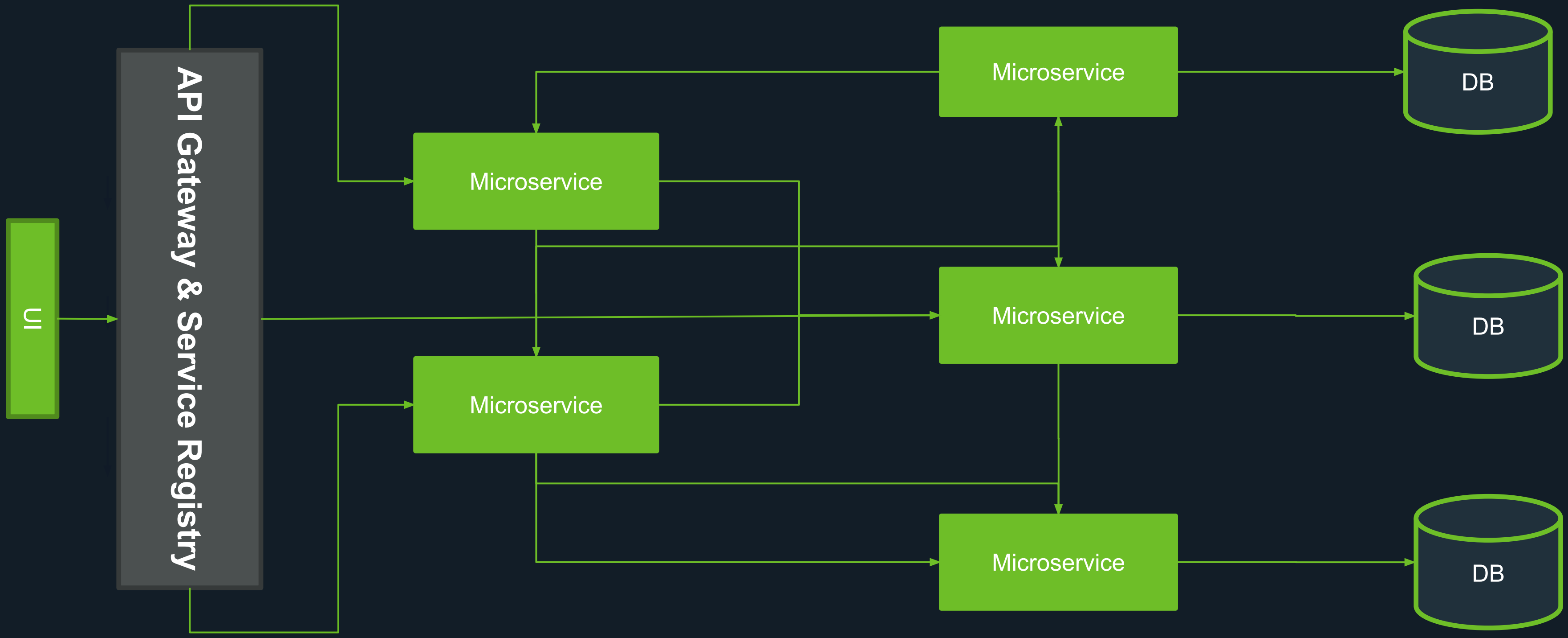
- less cross-cutting concerns
- easier debugging and testing
- simple deploy process
- simple to develop



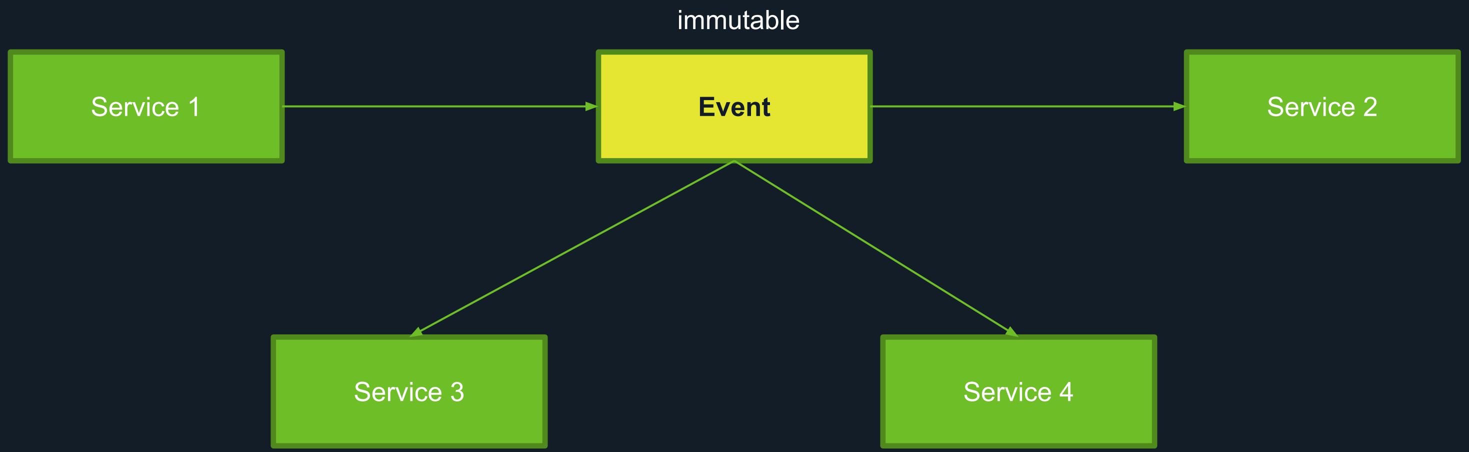
Weaknesses

- understanding
- making changes
- scalability
- new technologies

Microservice architecture?



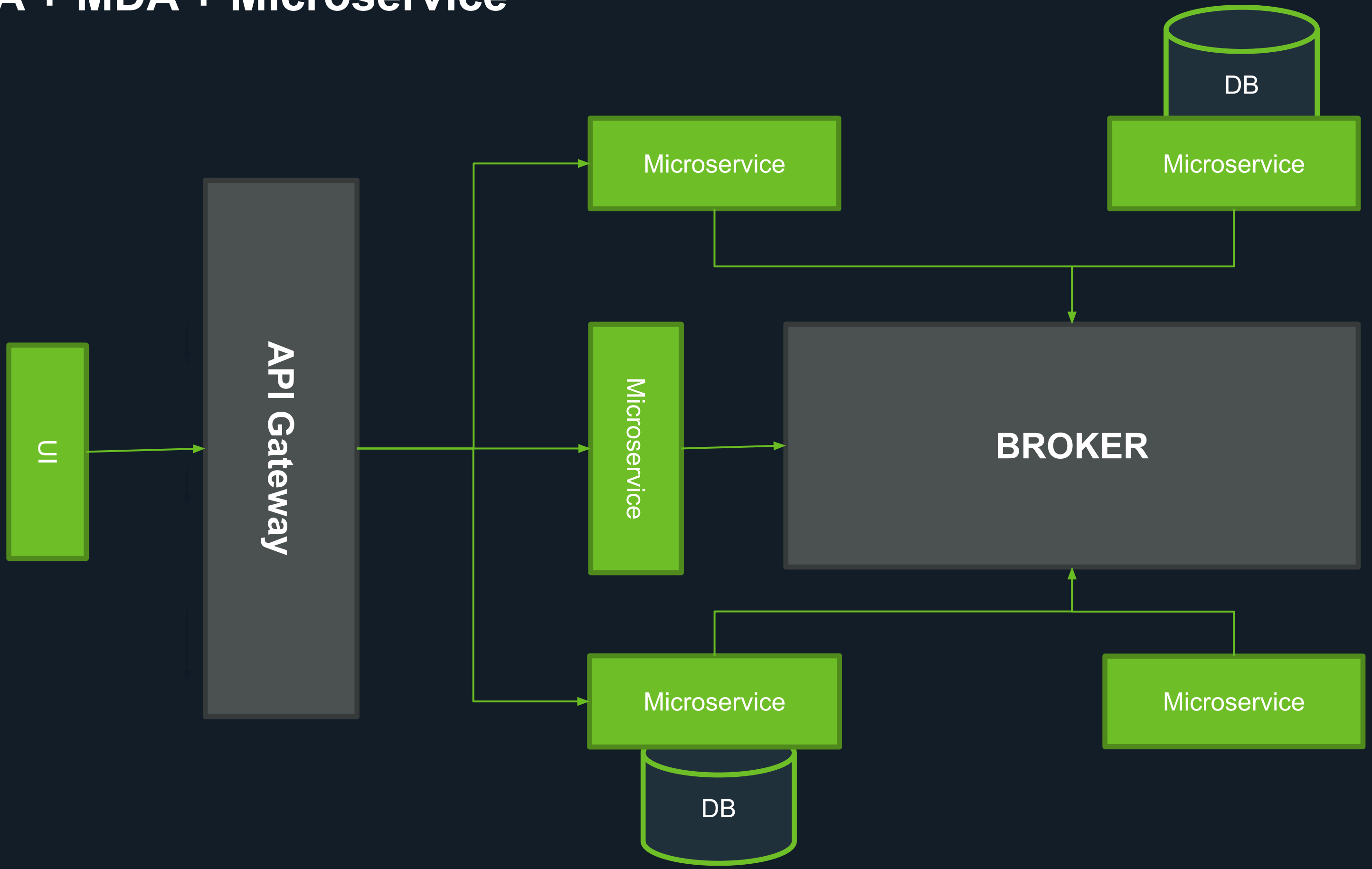
Event driven



Event driven architecture

- The event driven architecture (EDA) based on the detection, publishing, transmitting of events and receiving, processing and reacting to them
- Events triggering the listening services
  - two types: events and commands
- EDA is mostly based on message driven architecture (MDA)
  - Publish, Subscribe
  - Message Queue systems
- Complex event processing - message flows, data pipelines
- Events are immutable
- EDA and MDA fits well into Microservice architecture

EDA + MDA + Microservice



Apache Kafka Platform

Apache Kafka is a community distributed event streaming platform capable of handling trillions of events a day. Initially conceived as a messaging queue,

Kafka is based on an abstraction of a distributed commit log.

Since being created and open sourced by LinkedIn in 2011, Kafka has quickly evolved from messaging queue to a full-fledged event streaming platform.

Written in Java & Scala.

kafka.apache.org

Who uses?

Pinterest
shopify
tinder
salesforce
twitter
NETFLIX
LinkedIn
Spotify®
PayPal
ORACLE
skyscanner

What is Kafka for?

“All of your data is a stream of events”

Publish/Subscribe

Read and write stream of events like a traditional messaging system

Processing

Support scalable stream processing applications that react to events in real time

Store

Store streams of data safely in a distributed, replicated fault-tolerant cluster

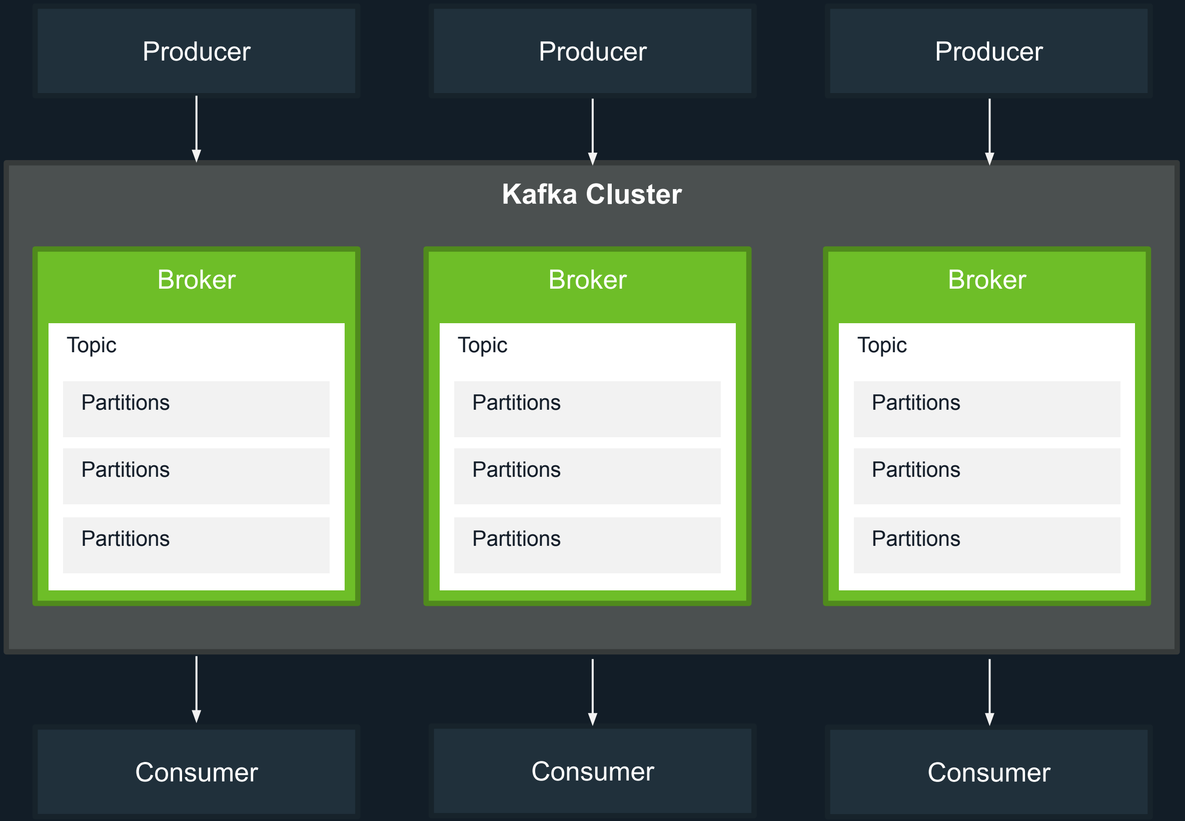
Event streams - real time

Event stream processing (ESP) is the practice of taking action on a series of data points that originate from a system that continuously creates data. The term “event” refers to each data point in the system, and “stream” refers to the ongoing delivery of those events. A series of events can also be referred to as “streaming data” or “data streams.” Actions that are taken on those events include aggregations (e.g., calculations such as sum, mean, standard deviation), analytics (e.g., predicting a future event based on patterns in the data), transformations (e.g., changing a number into a date format), enrichment (e.g., combining the data point with other data sources to create more context and meaning), and ingestion (e.g., inserting the data into a database).

Event stream processing is often viewed as complementary to batch processing. Batch processing is about taking action on a large set of static data (“data at rest”), while event stream processing is about taking action on a constant flow of data (“data in motion”). Event stream processing is necessary for situations where action needs to be taken as soon as possible. This is why event stream processing environments are often described as “real-time processing.”

commsignia

Kafka Architecture



Producer

Producer

Producer

Kafka Cluster

Broker

Topic

Partitions

Partitions

Partitions

Broker

Topic

Partitions

Partitions

Partitions

Broker

Topic

Partitions

Partitions

Partitions

Consumer

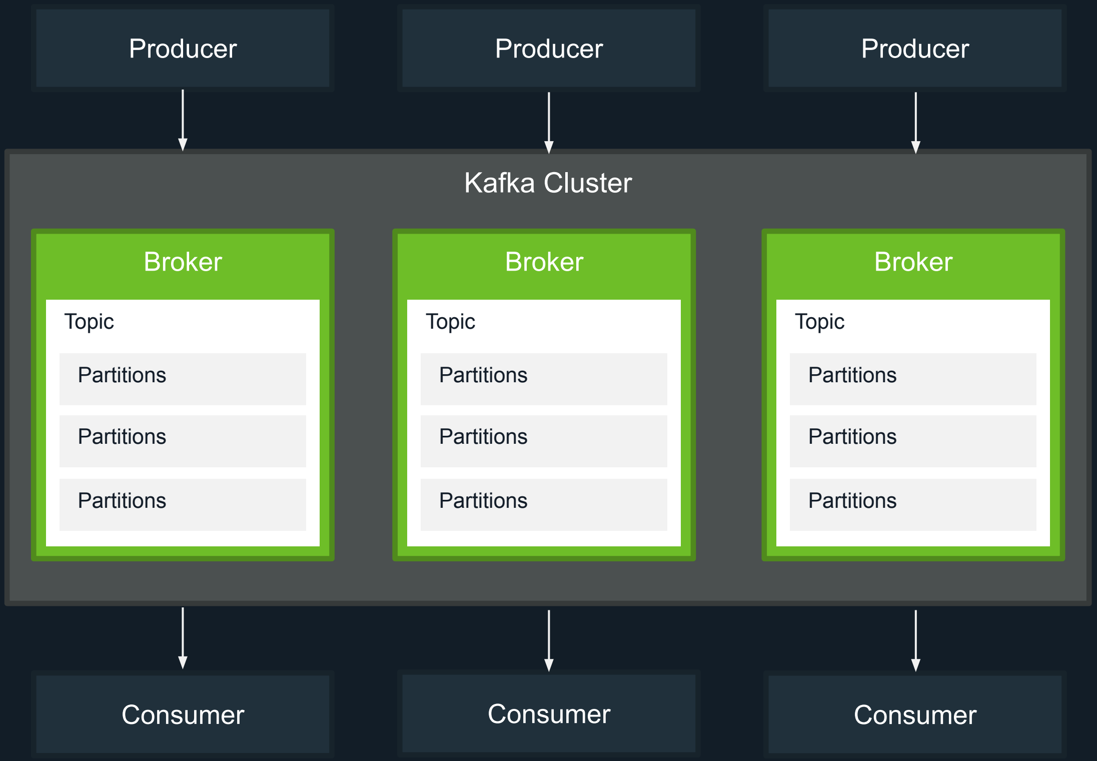
Consumer

Consumer

Topic

Records are organized into topics. Producer applications write data to topics and consumer applications read from topics.

Kafka retains records in the **log**, making the consumers responsible for tracking the position in the log, known as the “**offset**”. Typically, a consumer advances the offset in a linear manner as messages are read. However, the position is actually controlled by the consumer, which can consume messages in any order.



Anatomy of a Topic

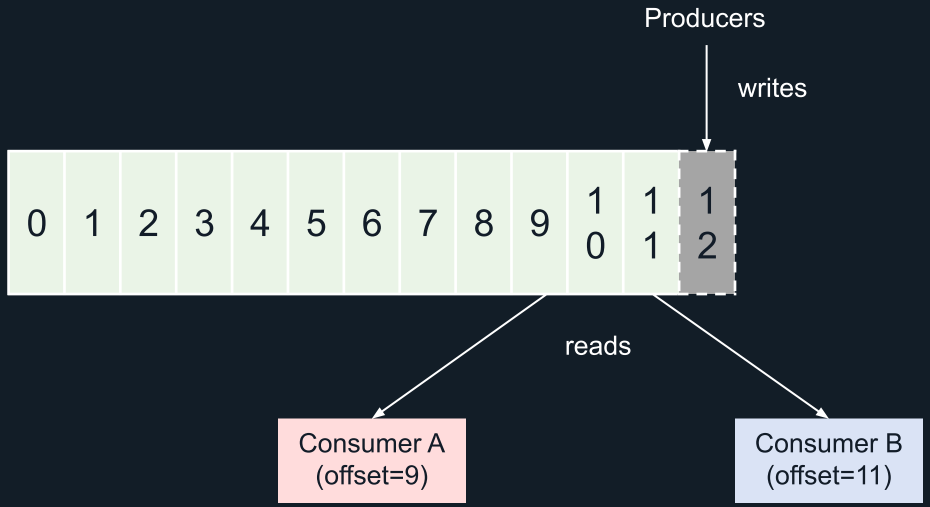
Record

- key
- value *
  - AVRO
  - JSON
  - Protobuf
  - XML
  - Plain text
- timestamp
- headers

Offset

- current
- committed
  - rebalancing

event log



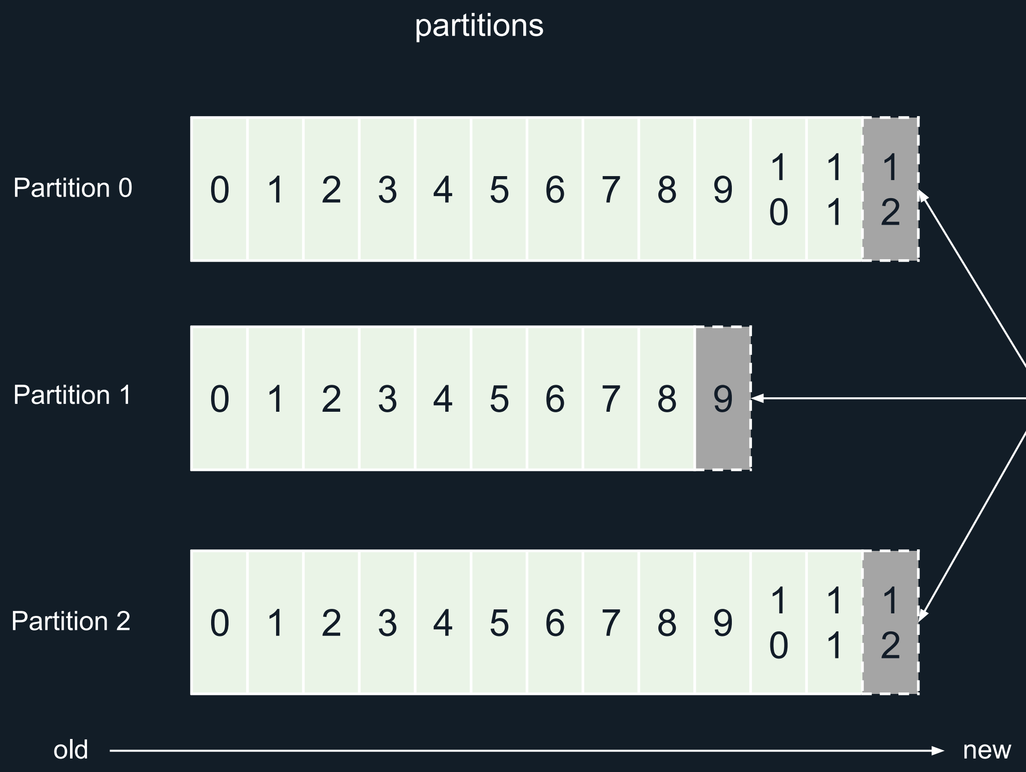
Anatomy of a Topic

Partitioning:

- by key
- default: round-rubin

Consumer Lag (combined lag)

Consumer lag is simply the **delta between the consumer's last committed offset and the producer's end offset** in the log.



partitions

Partition 0 0 1 2 3 4 5 6 7 8 9 10 11 12

Partition 1 0 1 2 3 4 5 6 7 8 9

Partition 2 0 1 2 3 4 5 6 7 8 9 10 11 12

writes

old → new

Retention

Time (default: 7 days)
This configuration controls the maximum time we will retain a log before we will discard old log segments to free up space if we are using the "delete" retention policy. This represents an SLA on how soon consumers must read their data. If set to -1, no time limit is applied.

Size
This configuration controls the maximum size a partition (which consists of log segments) can grow to before we will discard old log segments to free up space if we are using the "delete" retention policy. By default there is no size limit only a time limit. Since this limit is enforced at the partition level, multiply it by the number of partitions to compute the topic retention in bytes.

cleanup policy: delete

Latency, performance, scaling

Publish time: Sending and appending the record to the leader replica

Commit time: Replicating the record from leader to followers

Catch-up time: Catching up to the record's offset in the log

Fetch time: Fetching the record from the broker

Hundreds of configuration

- fetch.batch.size
- fetch.min.bytes
- fetch.max.wait.ms
- …

Number of partitions

- partition keys

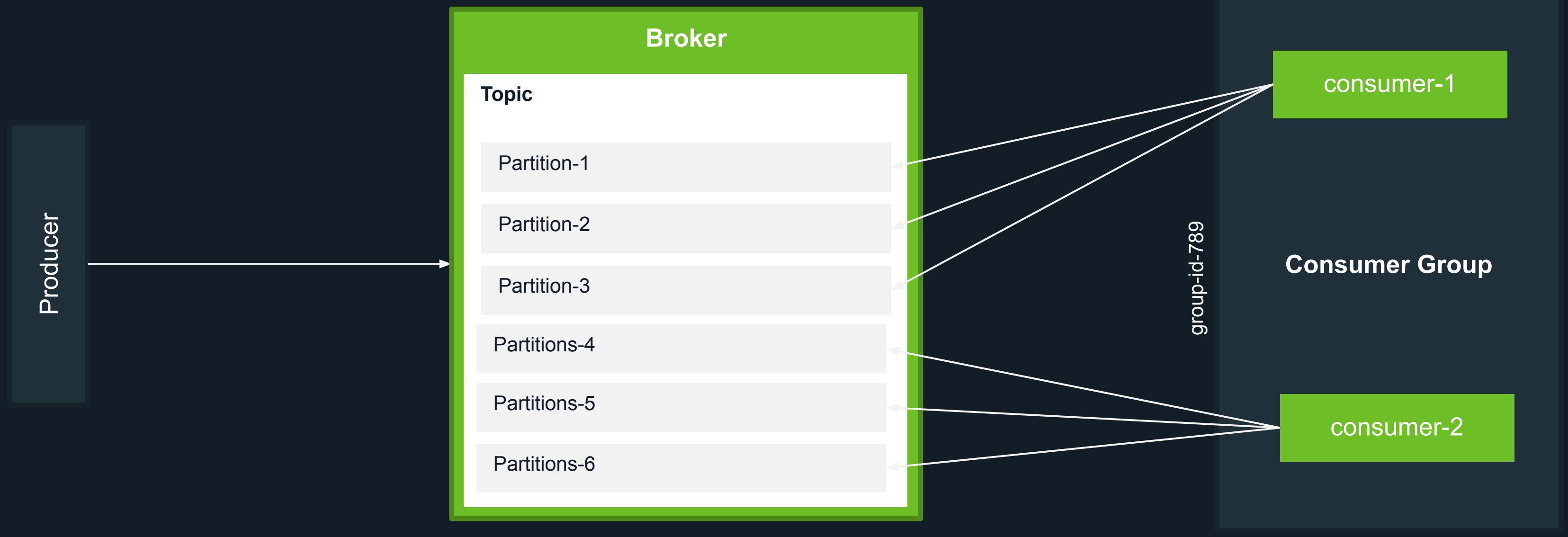
Horizontal scaling of

- Kafka
- producer service
- consumer service

Consumer groups

- group id

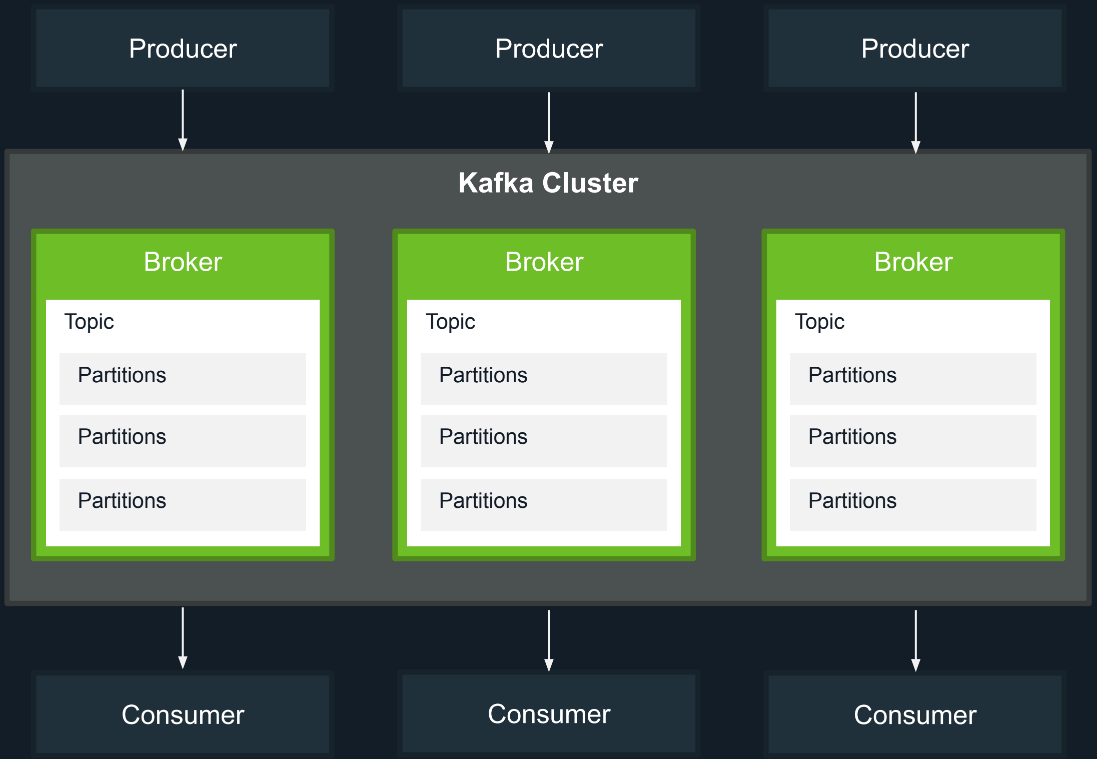
Consumer Groups



Cluster

A Kafka **cluster** consists of one or more servers (Kafka brokers) running Kafka. Producers are processes that push records into Kafka topics within the broker. A consumer pulls records off a Kafka topic.

Replication means having multiple copies of the data, spread across multiple servers/brokers. This helps in maintaining high availability in case one of the brokers goes down and is unavailable to serve the requests.

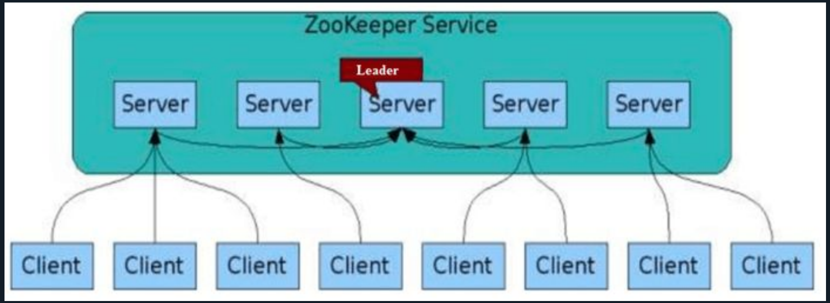


Distributed system

Zookeeper and Leader + Follower

Kafka Replication is allowed at the partition level, copies of a partition are maintained at multiple broker instances using the partition’s Write-Ahead Log. Amongst all the replicas of a partition, Kafka designates one of them as the “Leader” partition and all other partitions are followers or “in-sync” partitions.

The Leader is responsible for receiving as well as sending data, for that partition. The total number of replicas including the leader constitute the Replication factor. To maintain these clusters and the topics/partitions within, Kafka has a centralized service called the Zookeeper.

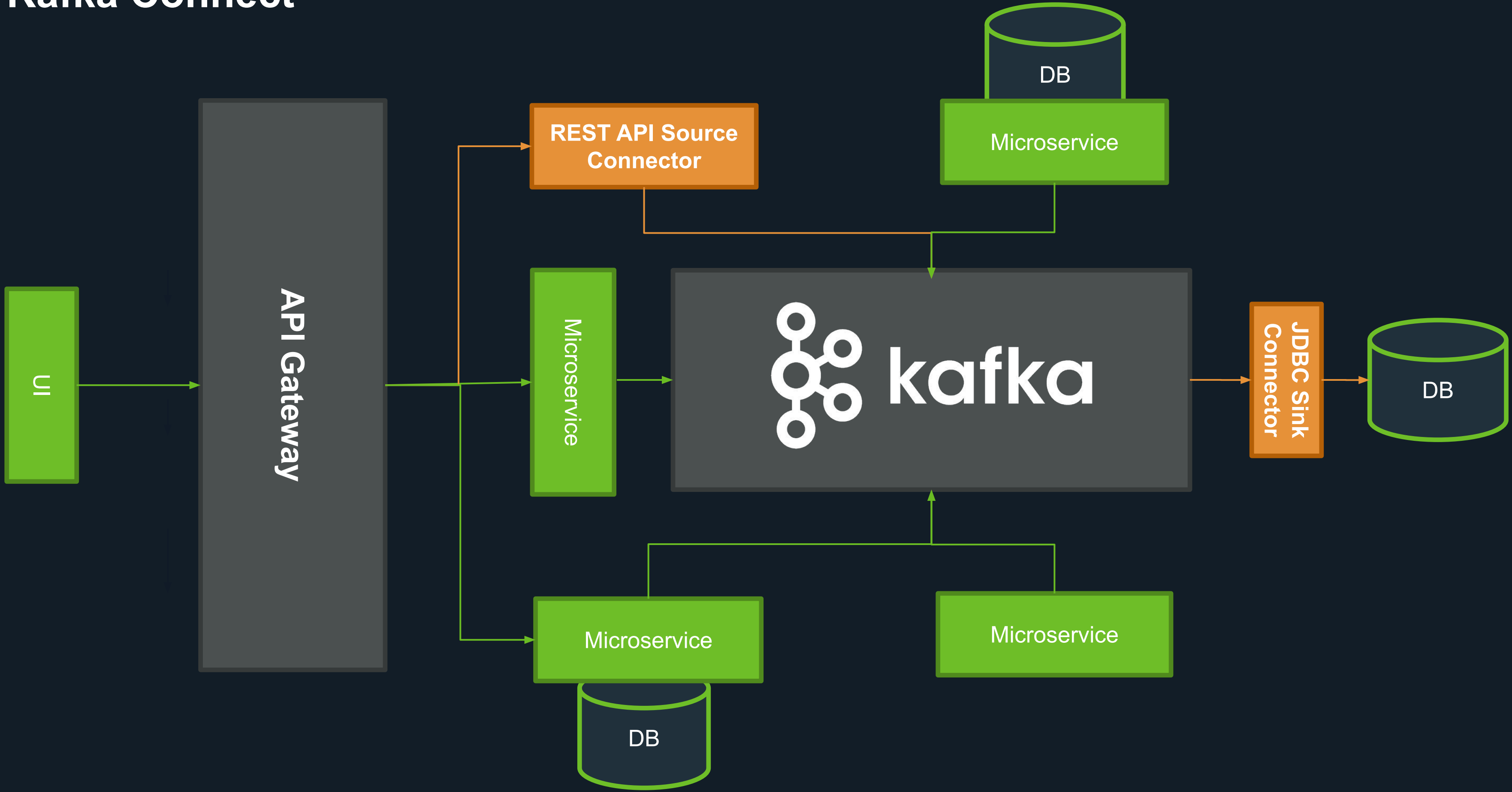


Kafka Connect

- A Kafka Connect is a framework for connecting Kafka with external system
  - databases
  - search indexes
  - file systems
- Kafka Connectors are ready-to-use components
- **Source** connector - Import data to topics
- **Sink** connector - Export data to external systems
- custom connectors can be developed
- built on Confluent platform

commsignia

Kafka Connect



Show me the code!

- Docker
- Java - Spring Boot project setup
  - spring-kafka
- Classes, Beans, Services
- Consumer
  - Message filtering

commsignia

Want to learn more?

- Kafka Connect
- Spring kafka-streams library
- Integration Testing Kafka based services
- KSQL
- https://kafka.apache.org/quickstart
- https://github.com/ttimot24/webinar-kafka-demo

Questions & Answers

commsignia