Apache Kafka Basics

David Veres

Motivation

- Event-driven systems are becoming more popular
- Focus on data as events
- Producers and consumers are decoupled
- Process events, react to them
- Needs a single platform
  - Connects the participants
  - Real-time
  - Stores the events

Apache Kafka

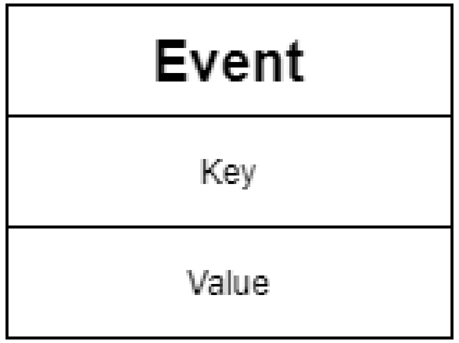
kafka

PayPal Hotels.com airbnb NETFLIX

Fundamentals

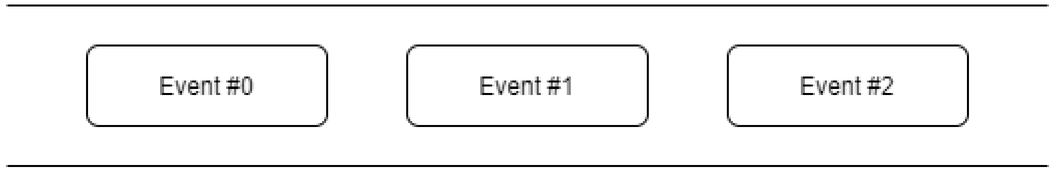
Events

- „A thing that has happened”
- It can be any kind of thing
  - A user clicks on a button
  - A microservice finishes a computation and submits the result
  - An IoT device sends data
- An event has a state
  - Describes what happened
  - Stored in structured format (like JSON)
- In Kafka it is a key-value pair



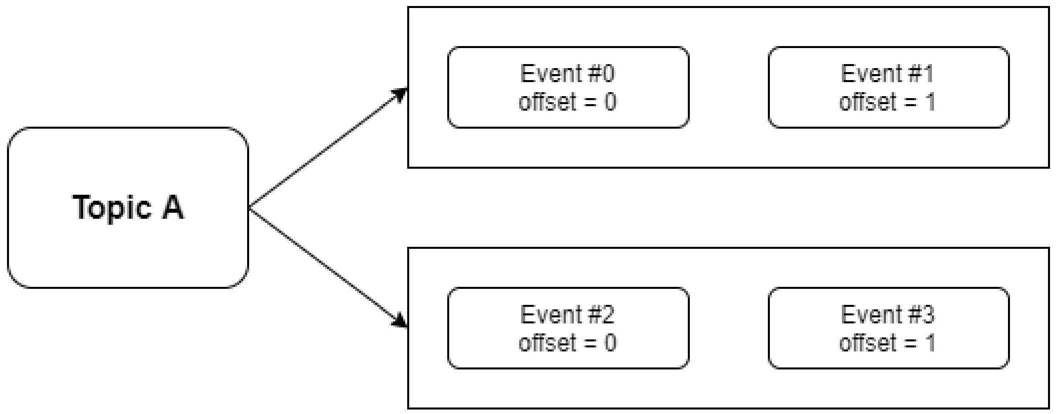
Topics

- The way of categorizing events
- A collection of similar events
- We can have multiple topics
- The same event can appear in different topics
- Topic = log of events
  - Append only
  - Events are immutable



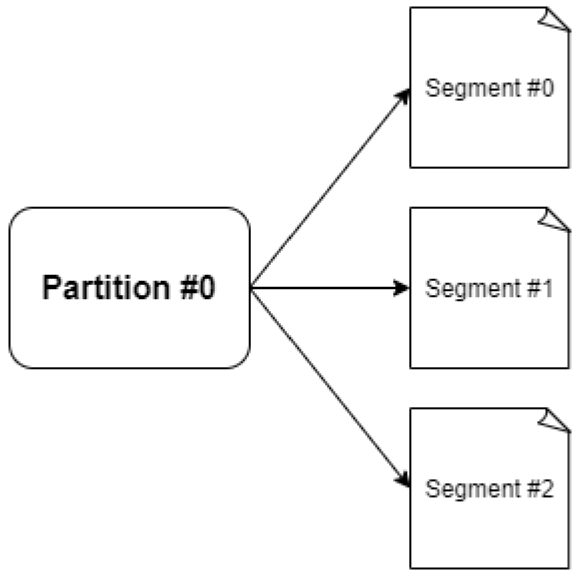
Partitions

- A topic can be split into multiple partitions
- „Real log of events”
  - The order of events is strict
  - This might not true for topics
- Every partition has it’s own offset space



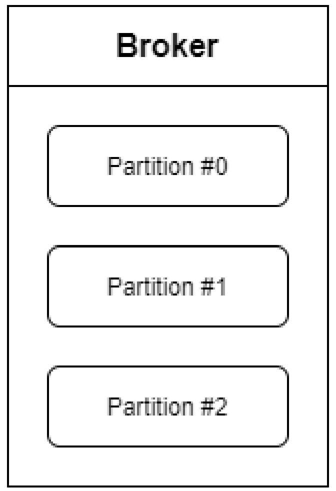
Segments

- A partition is broken into multiple segments
- These are individual files
- Events are stored on the disk in files



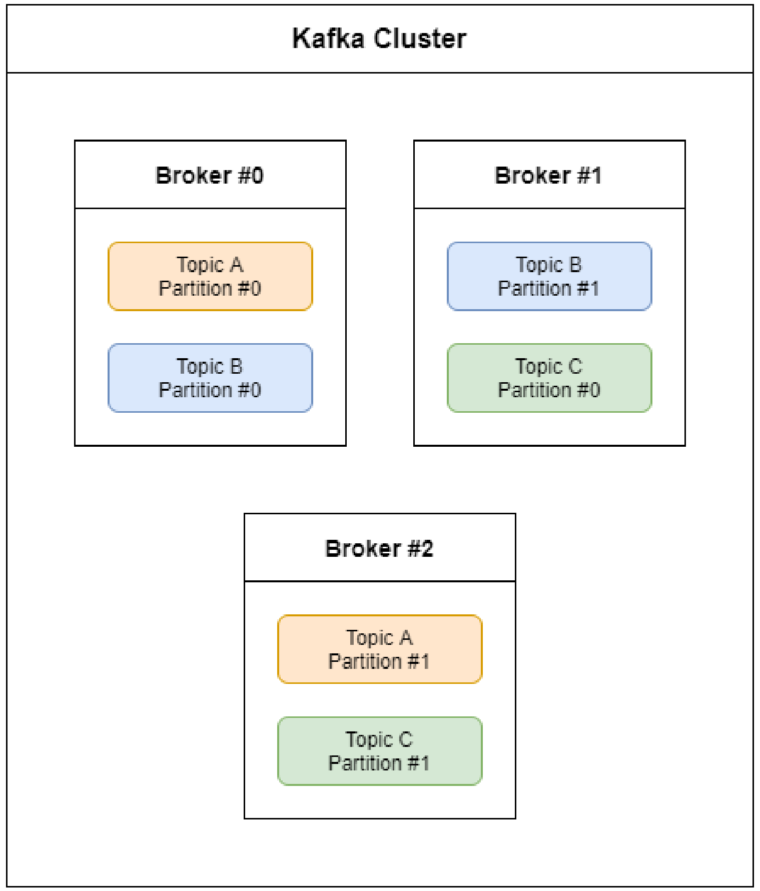
Brokers

- Brokers can be
  - Physical servers
  - Containers
  - Cloud instances
- Manage partitions
- Communicate with other brokers
- They are simple
  - Understand easily
  - Scale easily
  - Extend easily



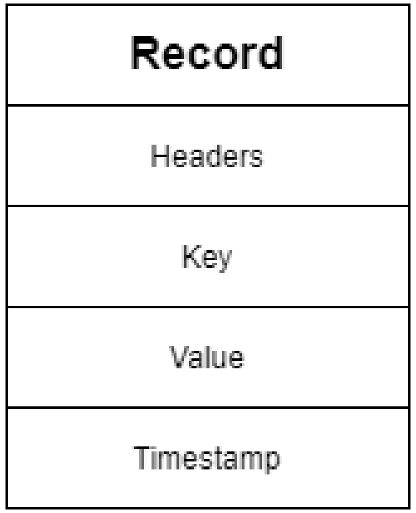
Kafka cluster

- Kafka is a distributed system
- Good scalability
- Many brokers form a Kafka cluster
- From outside, we see it as „one” Kafka



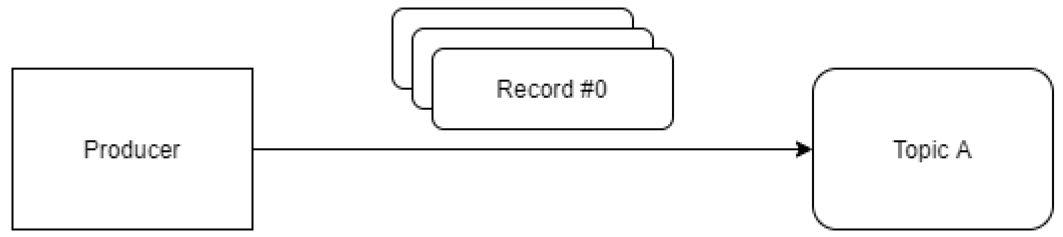
Kafka records

- Events represented in Kafka are called records
- Structure of a record
  - Key
  - Value
  - Headers
  - Timestamp



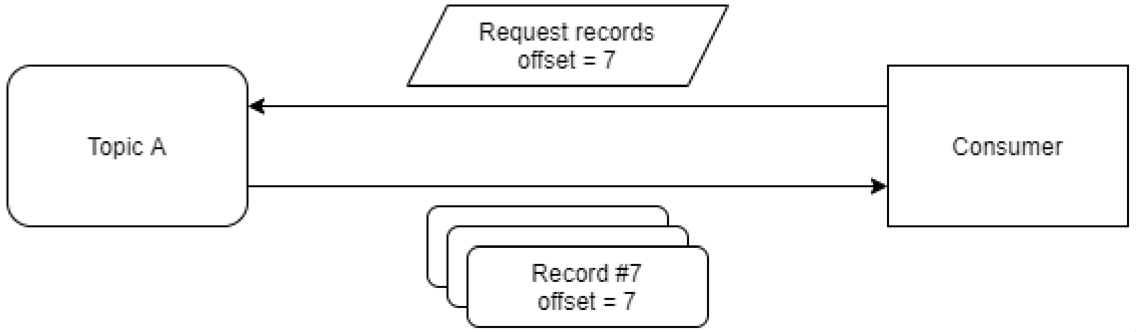
Producers

- Applications that we write
- Send records to topics (partitions)
- Producer API
  - Connection pooling
  - Networking
  - Partitioning

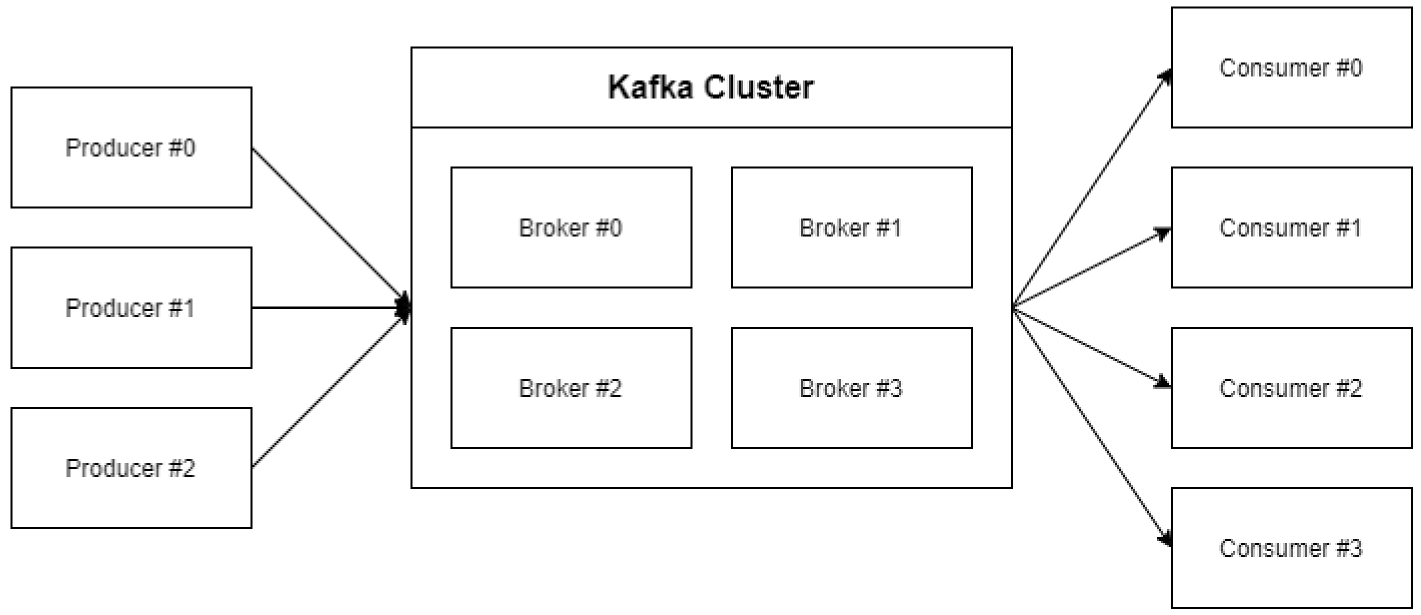


Consumers

- Applications that we write
- Read (pull) records from topics
  - Read by offset
- Consumer API
  - Connection pooling
  - Networking
- A record doesn’t get destroyed after read



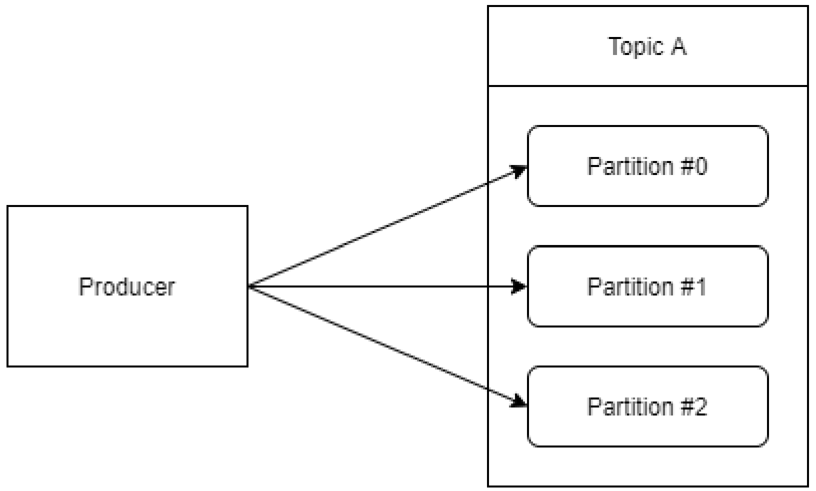
Fundamental parts of Kafka



Let’s take a closer look!

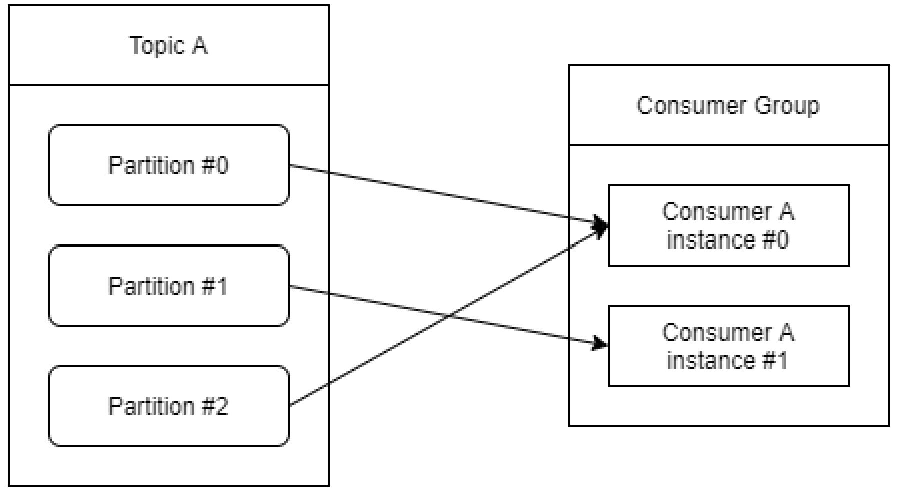
We produce to partitions

- Records are sent to partitions
- Purpose
  - Load balancing
  - Semantic partitioning
- Record without key
  - Round-robin strategy
- Record with key
  - `hash(key) % num_of_partitions`
  - Records with same key to same partition
- Custom partitioner logic



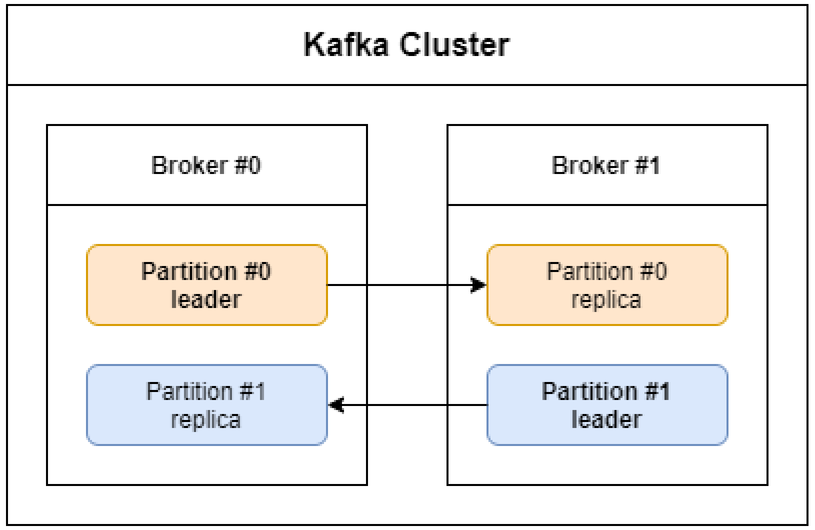
Consumer groups

- A consumer reads from partition(s)
- We might want to scale a consumer application
- Every consumer is part of a consumer group
- Kafka handles rebalancing automatically



Replication

- Brokers store records (in partitions, segments)
- Brokers might go down
- Replication factor
  - 1 leader partition
  - N-1 follower partition
- Synchronization is automatic
- Produce only to leaders
- We can configure to read from followers

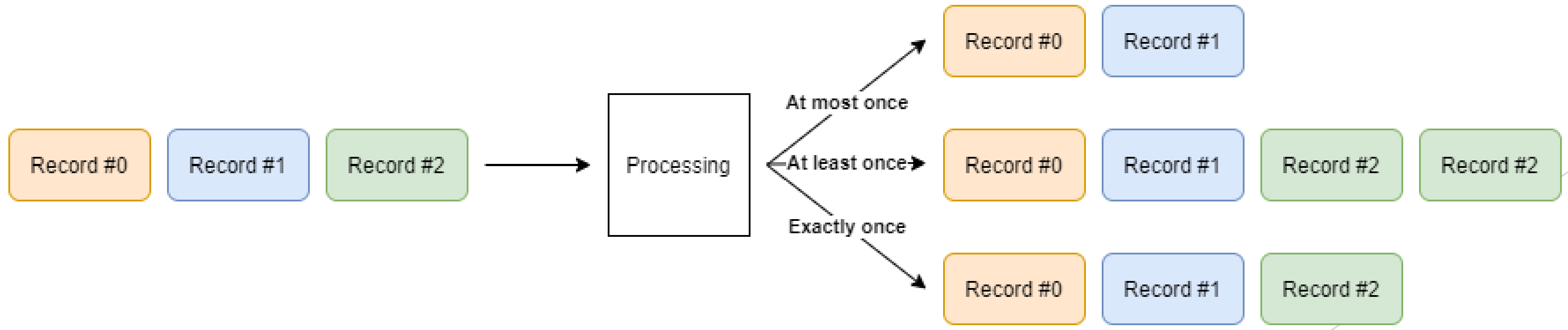


Producer guarantees

- We can wait for an acknowledgement from the broker(s)
  - Is the record successfully written to disk?
- This is a producer configuration
- We have 3 options
  - **NONE**
    - Fast, but might lose data
  - **LEADER**
    - Default, data is written for sure
  - **ALL**
    - Slowest option

Delivery guarantees

- Message processing guarantee categories
  - At most once
  - At least once
  - Exactly once
- Kafka provides a transactional API to reach exactly once
- Configurable for producer and consumer



Retention policy

- For how long a record should be stored?
- Decision factor
  - Business
  - Cost
- Default: 1 week
- Configurable
  - Globally
  - Per topic
- Data purge happens per segment

Security

- Kafka supports data encryption in transit
- Supports authentication and authorization
- Data stored in disk by Kafka is not encrypted
  - We can encrypt the disk itslef
  - We can write a wrapper to encrypt/decrypt a record’s value

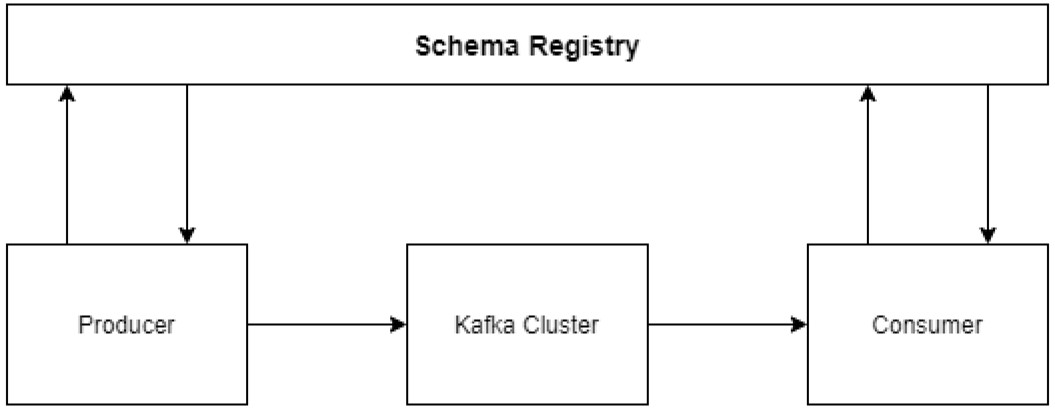
The Kafka ecosystem

ZooKeeper

- Provides multiple features for distributed applications
  - Distributed configuration management
  - Election
  - Key-value store
- Kafka uses ZooKeeper
  - Leader election
  - Topic configuration
  - Access control
- KIP-500, Apache Kafka 2.8
  - We can leave ZooKeeper
  - Not production ready yet

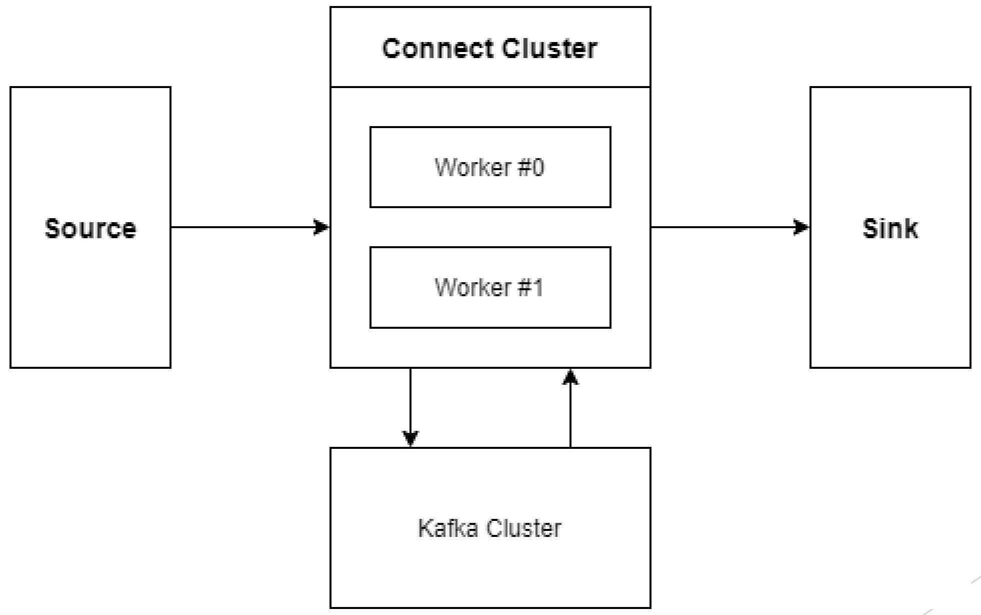
Schema Registry

- Schema of or records might change by time
- A schema registry is a database of schemas
- Supported formats: AVRO, JSON schema, Protobuf, custom
- Producers and consumers validate the records
  - They call the schema registry
  - Check the compatibility



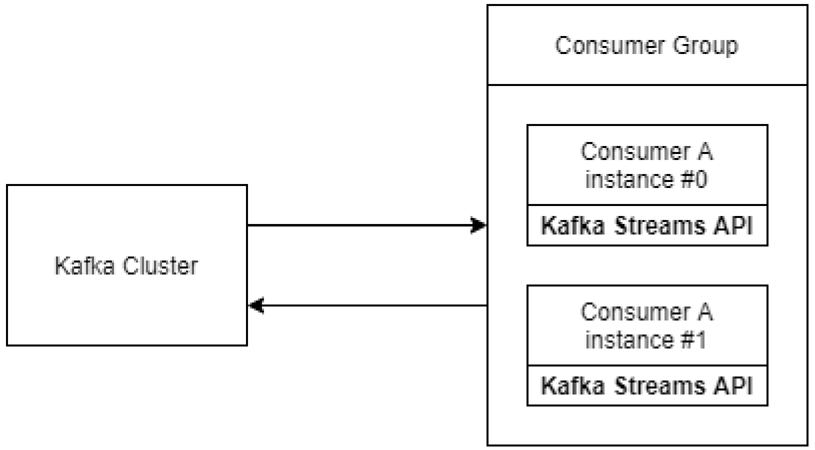
Kafka Connect

- We might want to transfer data
  - From Kafka to another system
  - Into Kafka from another system
- Many existing connectors
- Easy to configure



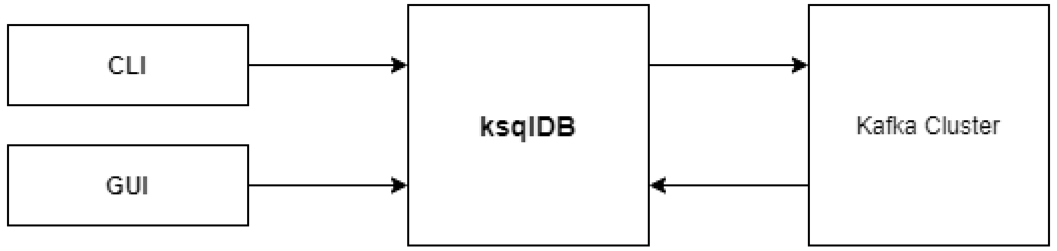
Kafka Streams

- The consumer API is very simple
- Kafka Streams provides more complex operations
  - Filtering
  - Mapping
  - Aggregating
- Fluent functional API
- Output is stored in Kafka
- Scaleable, fault-tolerant



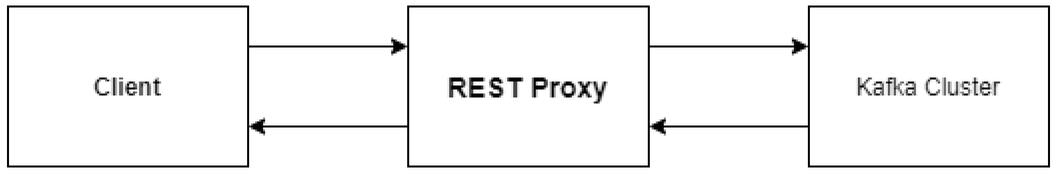
ksqlDB

- Another stream processing option
- Without programming
- Stream processing in an SQL format
- Communicate with ksqlDB
  - CLI
  - GUI



REST Proxy

- Java is the native language for Kafka
- Lot of other supported languages
- We can use the REST Proxy instead of libraries
  - HTTP calls for producing and consuming
  - Useful if our chosen language is not supported



Sources and result

- **Sources**
  - Confluent - Apache Kafka® Tutorials | Kafka 101
  - Confluent - Course | Apache Kafka® Fundamentals
  - https://kafka.apache.org/
- **Result**
  - https://github.com/veresdavid/apache-kafka-basics

Thank you for your attention!