Public

zühlke
empowering ideas

Kafka
Modern messaging

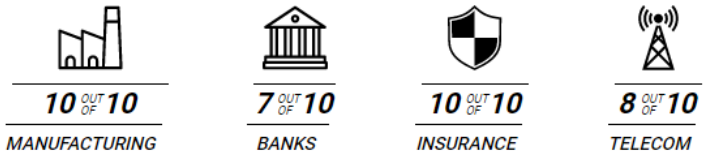
Zürich, 16.09.2024

zühlke
empowering ideas

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka 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.



SEE FULL LIST

Above is a snapshot of the number of top-ten largest companies using Kafka, per-industry.

Source: https://kafka.apache.org/

Kafka Training

Course Overview

Day 1

- Kafka Architecture & Core Concepts
- Lab: Cluster Behavior, Cluster Management
- Kafka Streams, Messaging & Schema Registry
- Lab: IoT Case, Kafka Streams

Day 2

- Architecture Concepts, Kafka application configurations
- Lab: IoT Case, Kafka Streams (cont.)
- Kafka Connect
- Lab: Witness Protection Case, Kafka Connect

Day 3

- Kafka Security, Monitoring, Cluster Design
- Lab: Witness Protection Case, Kafka Connect (cont.)
- Extended Kafka Ecosystem (KSQL, REST Proxy, …)
- Wrap-Up and Outlook

Your Name & expectations:

- Your Kafka know-how (1..4)
- Your role in your company
- The training is over: What should have been changed for you?

Kafka Training

Agenda Day 1

09:00 – 09:30 Welcome & Introduction

09:30 – 10:30 Kafka Architecture & Concepts

10:30 – 10:50 Break

10:50 – 11:45 Kafka Architecture & Concepts

11:45 – 12:45 Lunch Break

12:45 – 15:00 Kafka IoT Case

15:00 – 15:20 Break

15:20 – 16:20 Kafka IoT Case

16:20 – 17:00 Recap and Feedback

17:00 … Open work

Parking lot for questions / W-LAN

A location to collect your questions during the training

Feel free to add your questions to the parking lot. We will come back to them during the training.

=> Post-It’s

WLAN: use ‘z-Local’ on your device (access code on your batch)

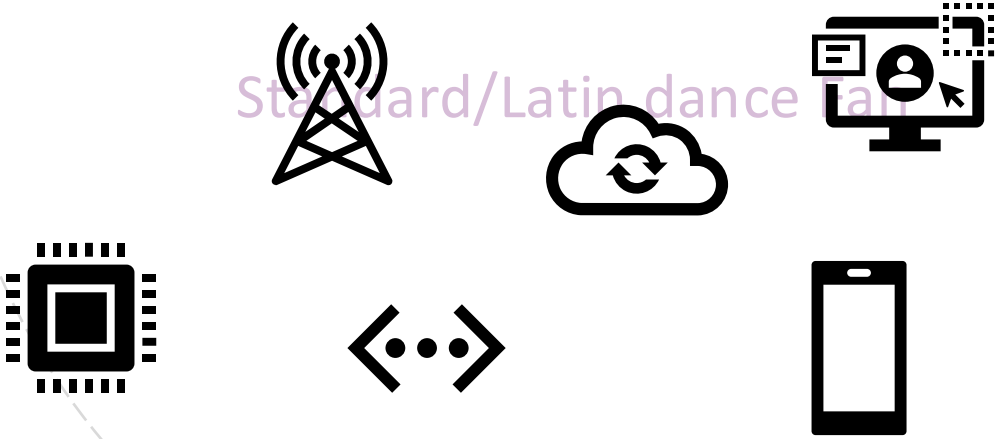
Principal Consultant

Severin Stöckli

severin.stoeckli@zuehlke.com

IoT Enthusiast &

Standard/Latin dance fan



Expert Software Engineer

3+ years of Apache Kafka hands-on experience

Andrea Mathis

Andrea.mathis@zuehlke.com

Kafka Architecture & Concepts Part 1

What is Kafka?

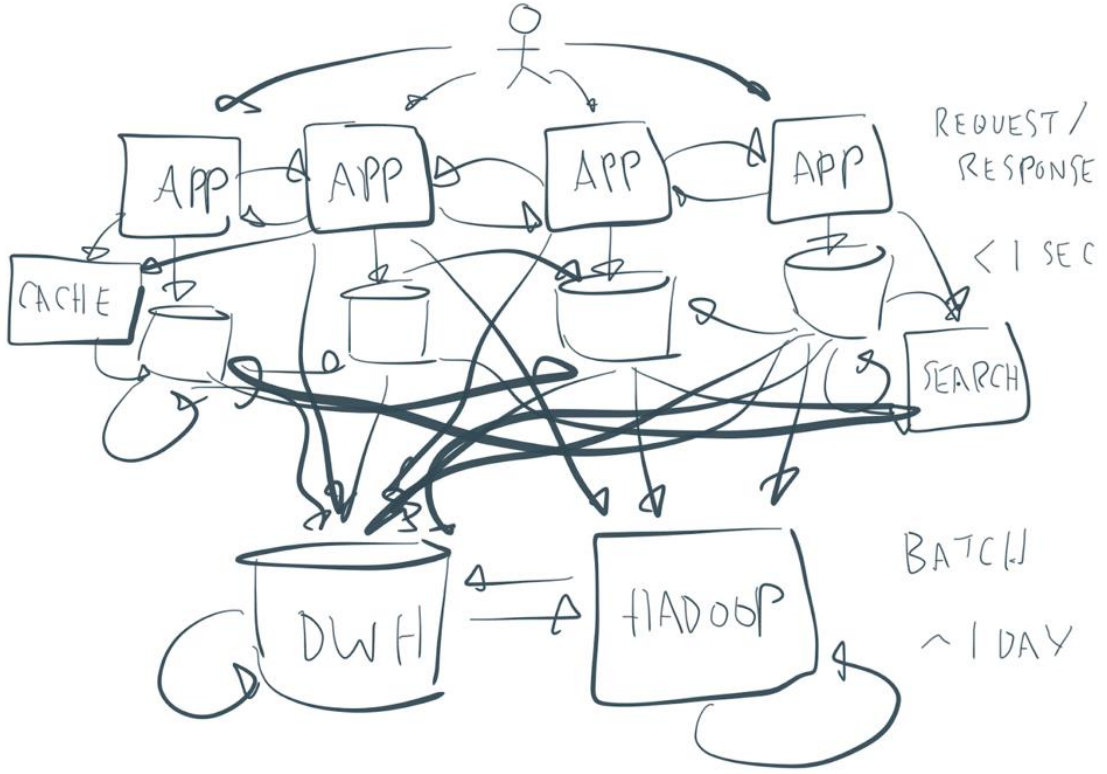
Question to the audience

- A Message Bus?
- A Database?
- A persistent Store?

Why Kafka?

The Architect’s pain

Franz Kafka: The Trail

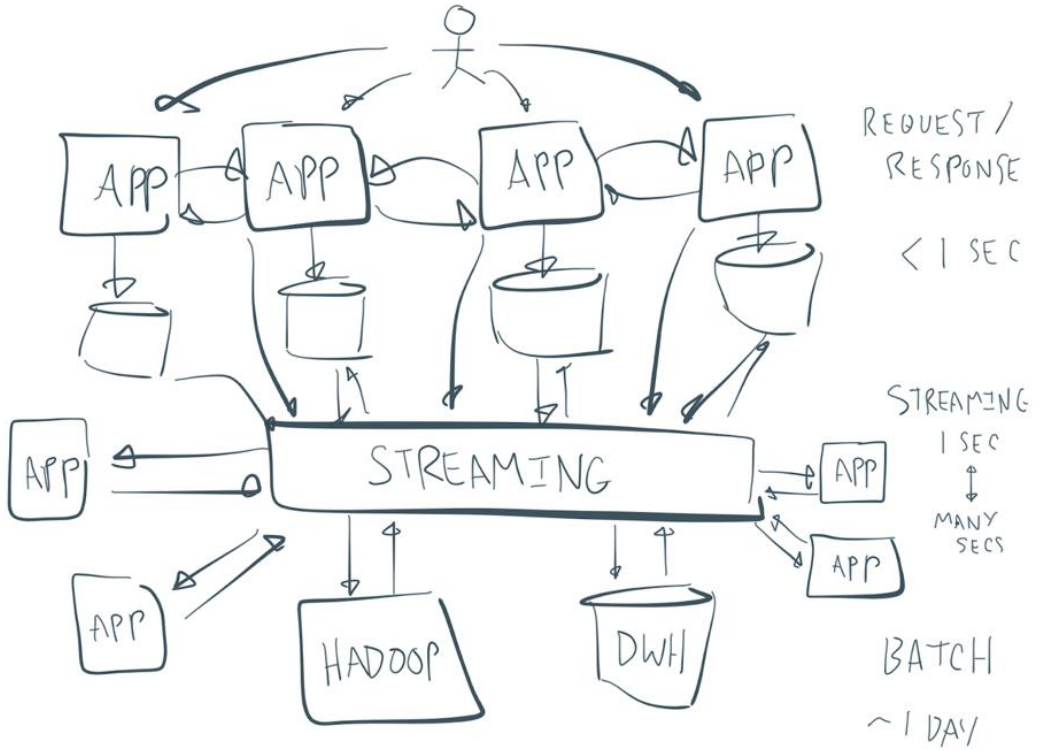


Source: https://www.confluent.io/blog/apache-kafka-vs-enterprise-service-bus-esb-friends-enemies-or-frenemies/

Why Kafka?

The Architect’s pain

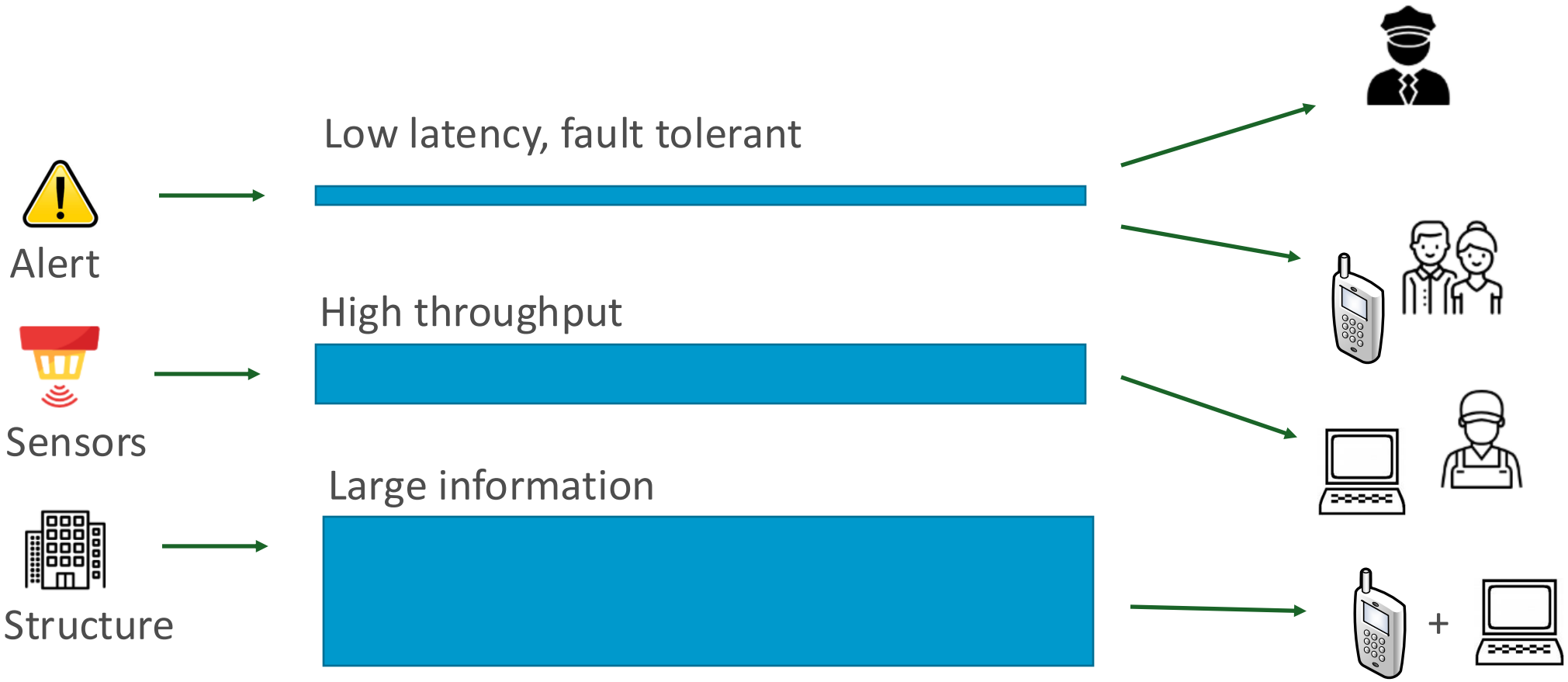
Apache Kafka before Franz Kafka



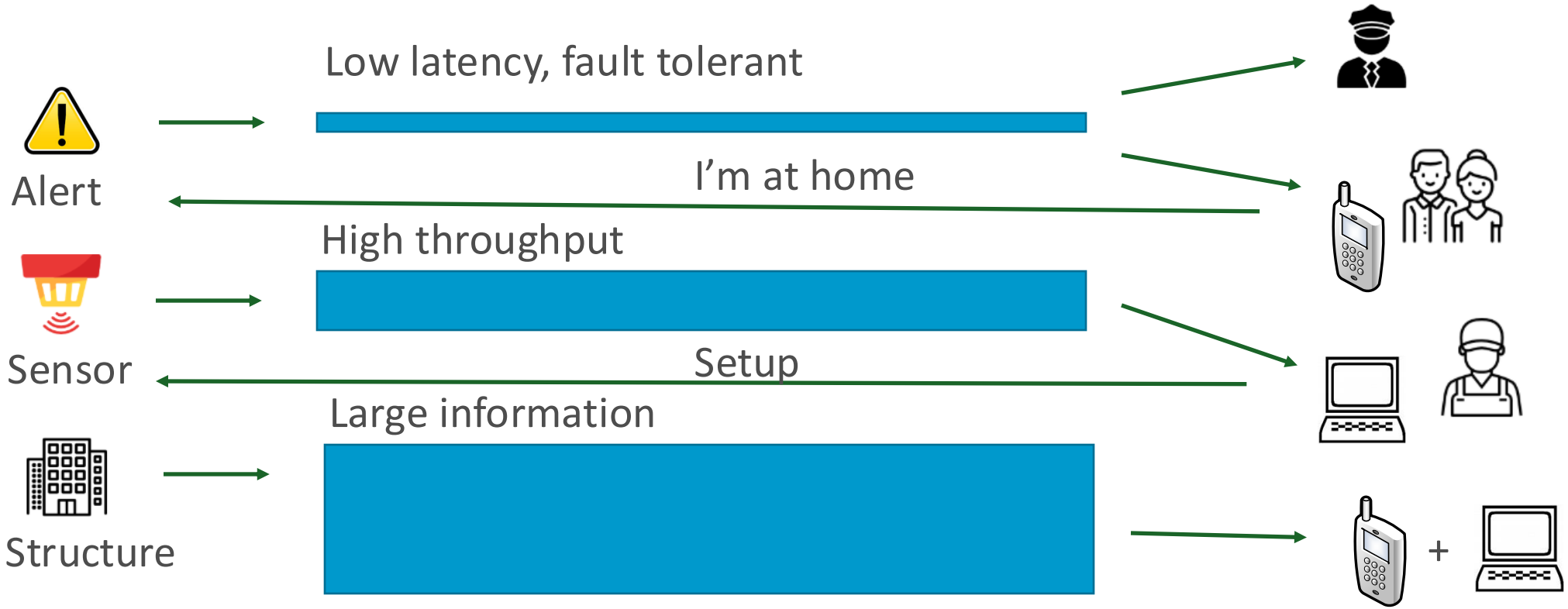
Source: https://kafka.apache.org/

Source: https://www.confluent.io/blog/apache-kafka-vs-enterprise-service-bus-esb-friends-enemies-or-frenemies/

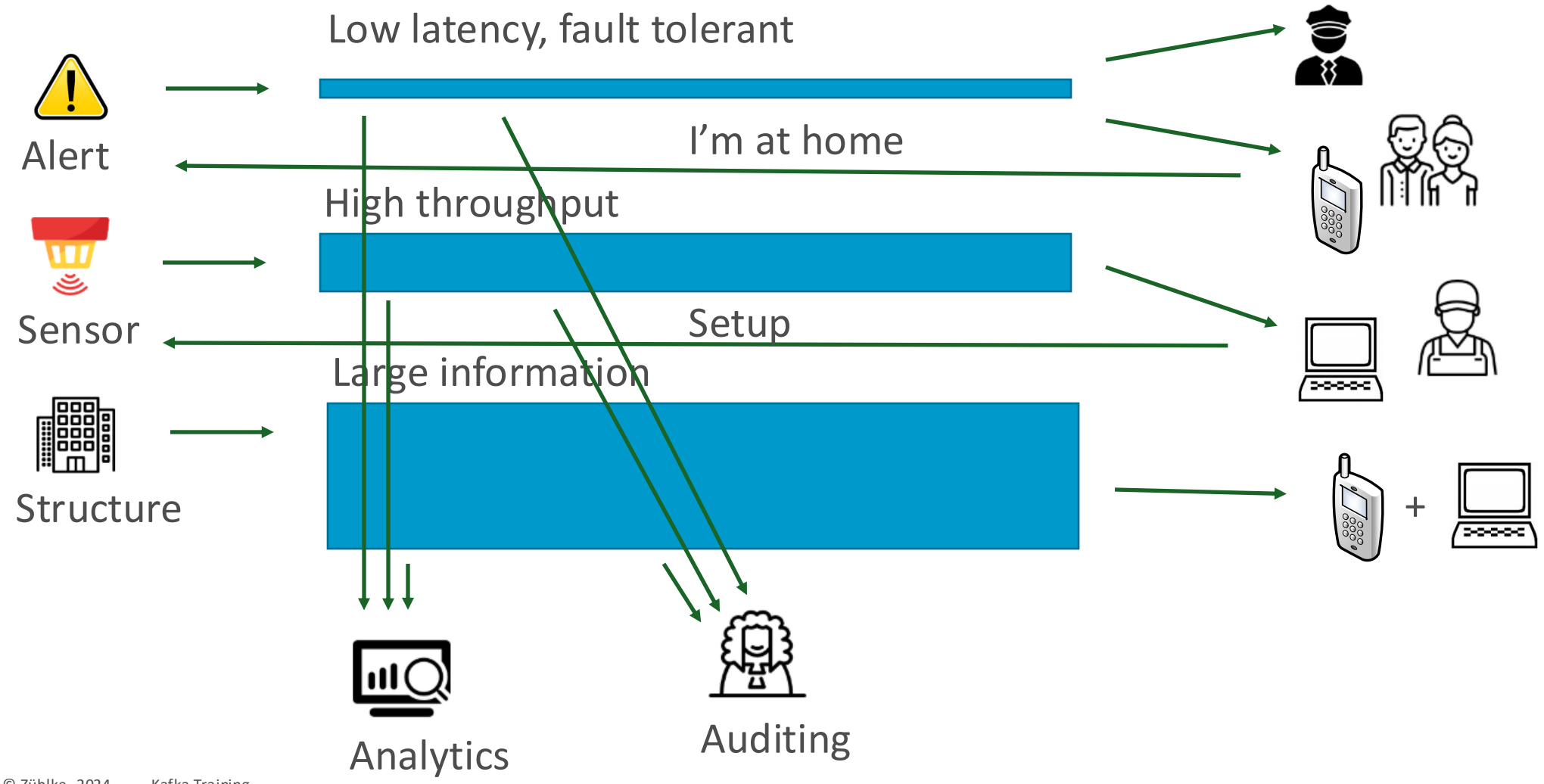
Integration Patterns (Context: Messaging)



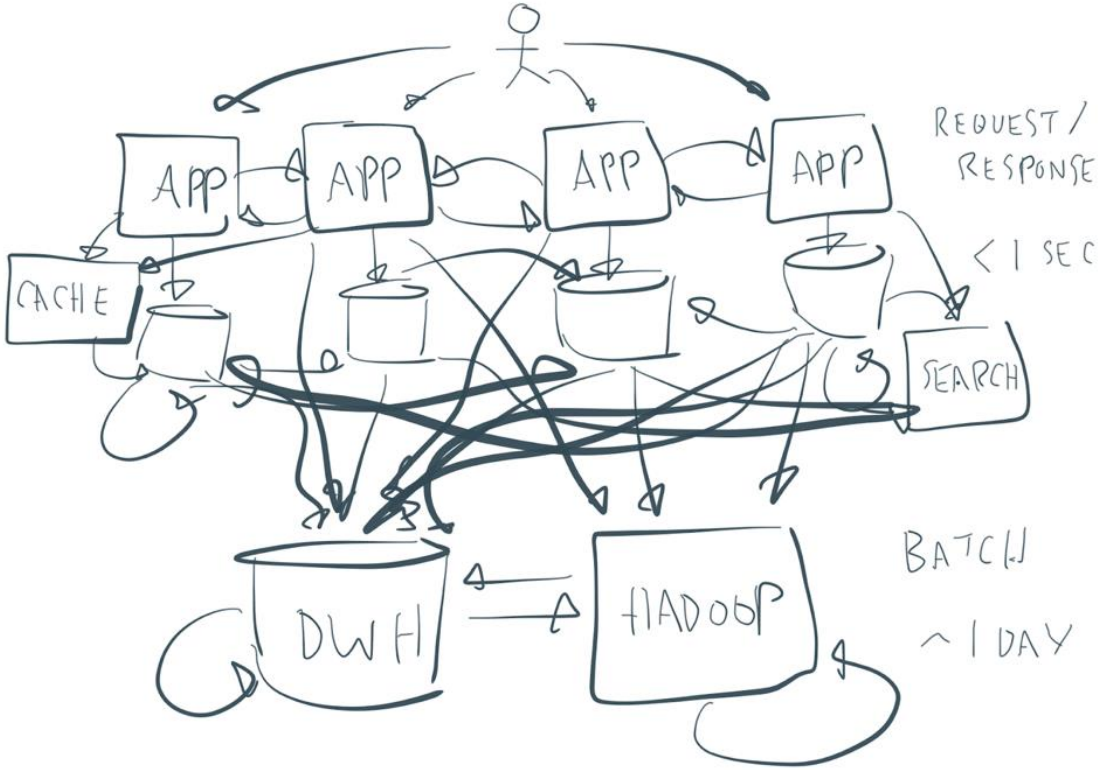
Integration Patterns (Context: Messaging)



Integration Patterns (Context: Messaging)



We want to avoid



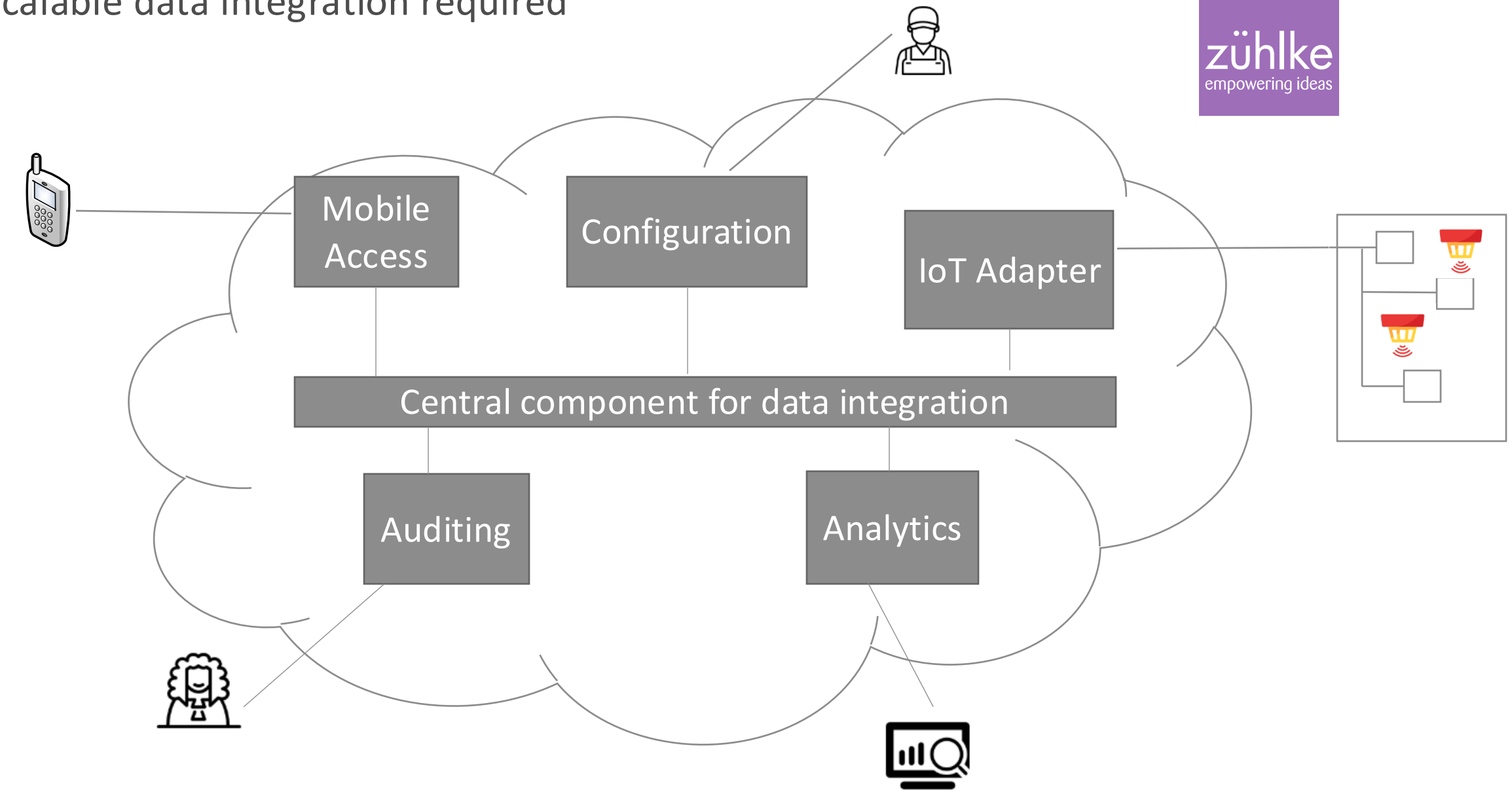
Operational Complexity (Point-to-Point Architecture), **high efforts in maintaining** and **scaling all pipelines between the systems**.

Lot of synchronous communication & jobs, unreliable data and inconsistency due system failures (prevents stream processing).

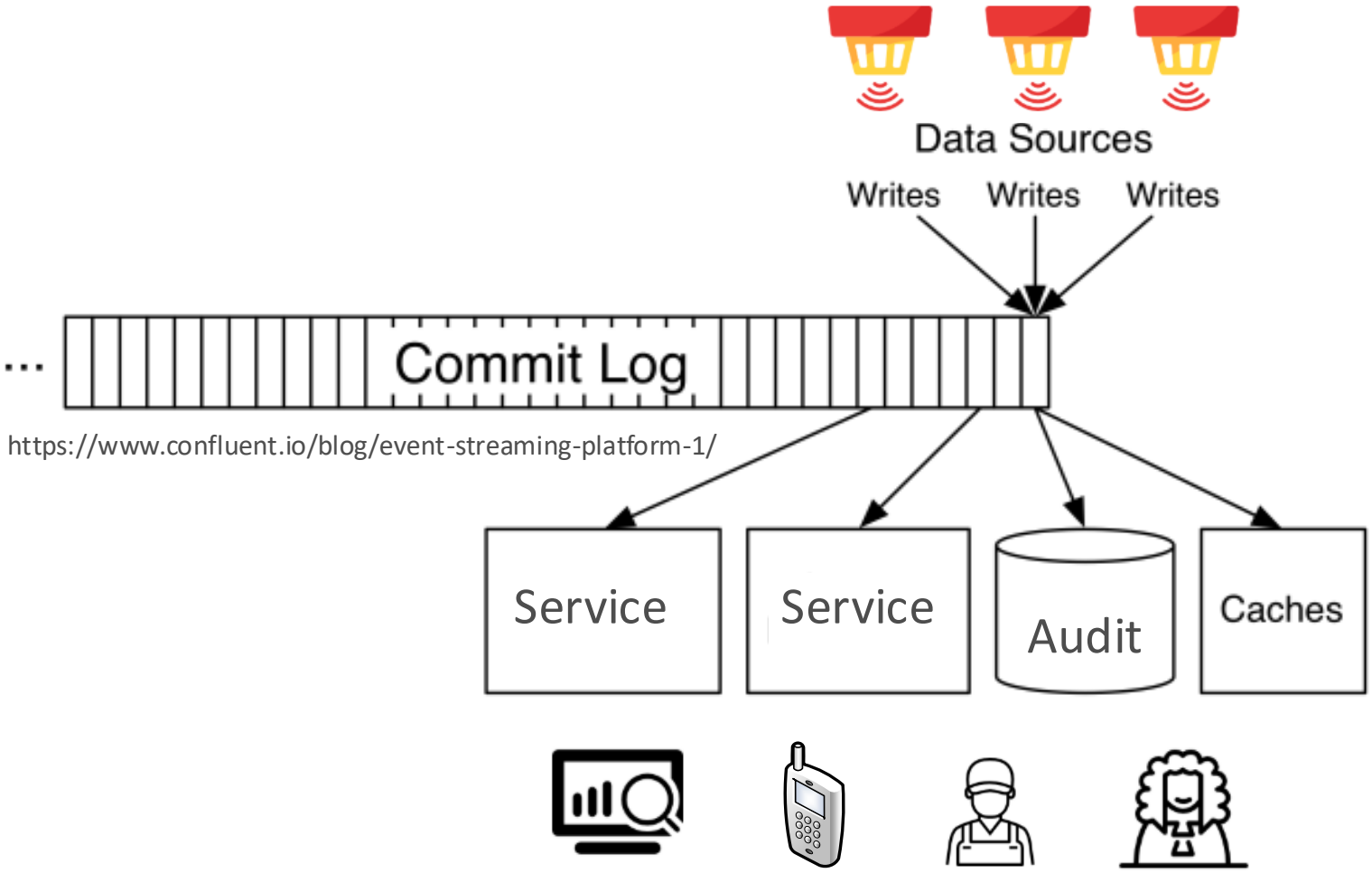
Source: https://www.confluent.io/blog/apache-kafka-vs-enterprise-service-bus-esb-friends-enemies-or-frenemies/

“Any suggestion to avoid Josef K.'s ordeal?”

Scalable data integration required



Commit Log as a central event store

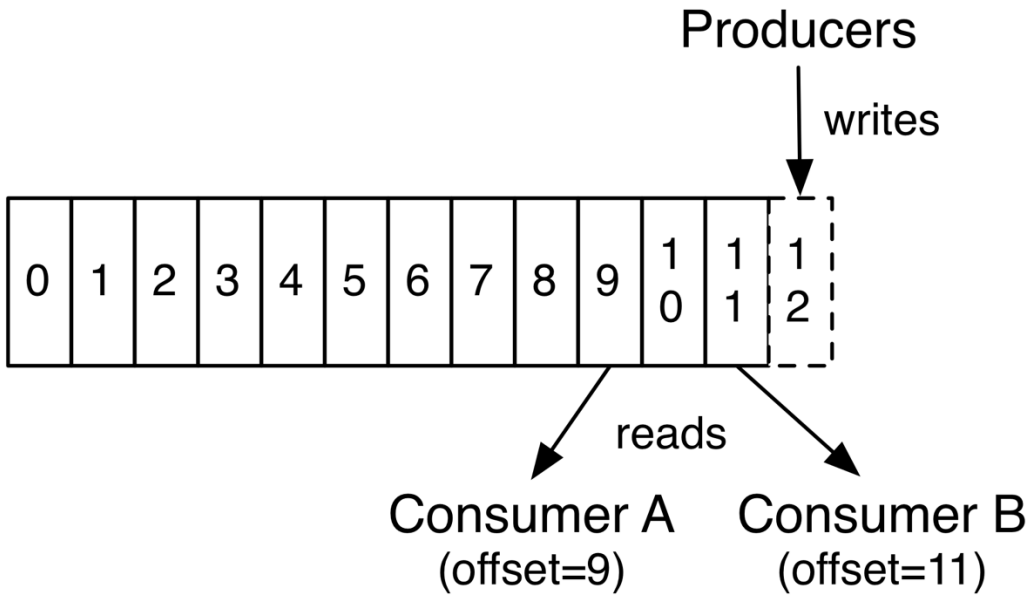


Source: https://www.confluent.io/blog/event-streaming-platform-1/

Why a commit Log?

Simple, append-only data structure

- Timely ordered sequence
- Durable event store that can be reprocessed at any time
- Every consumer controls its position of the log on its own
- Has high potential for optimizing read / write performance



Source: https://kafka.apache.org/23/documentation.html

Apache Kafka: A distributed, scalable commit log

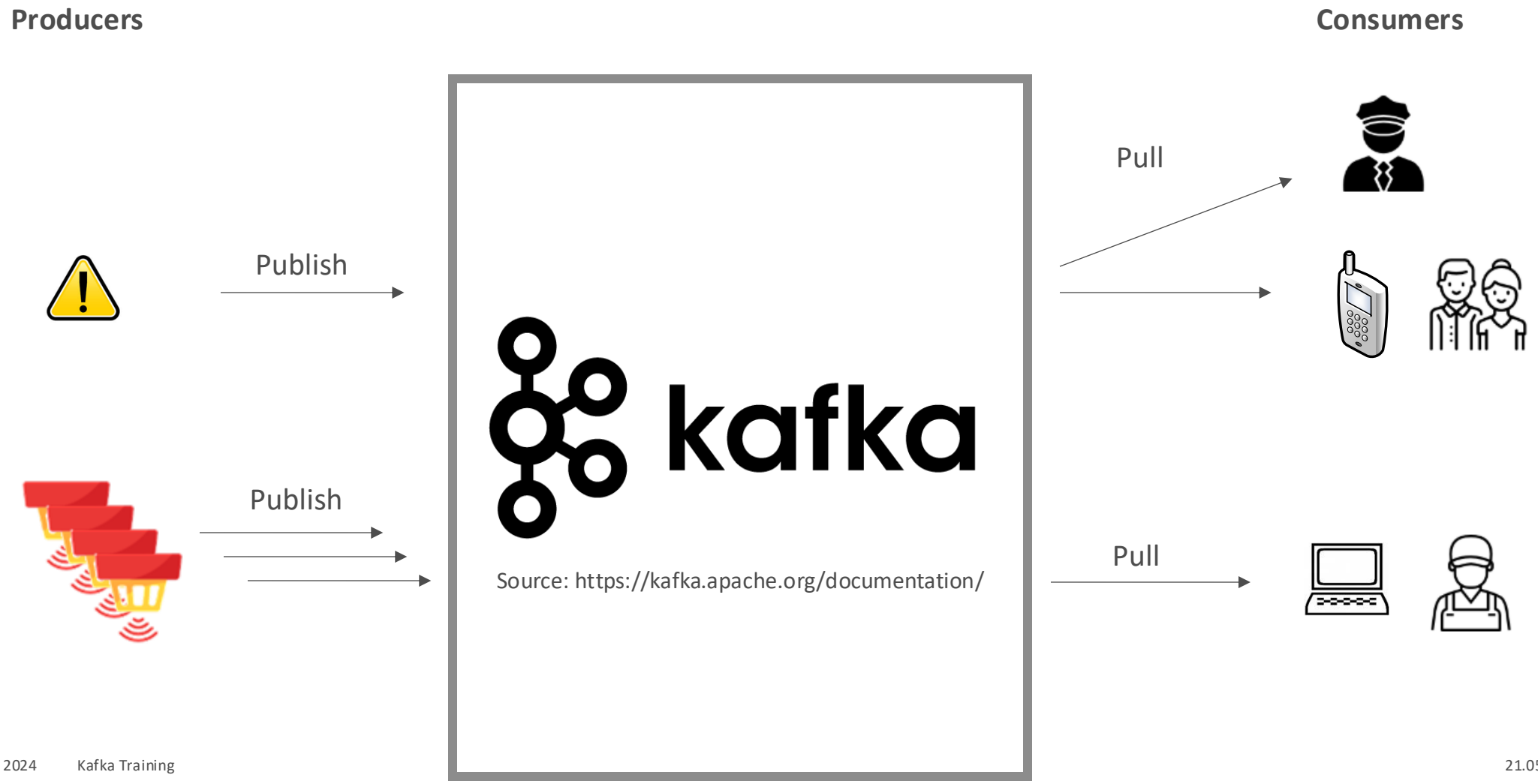
Horizontally Scalable

Low latency

Fault-Tolerant

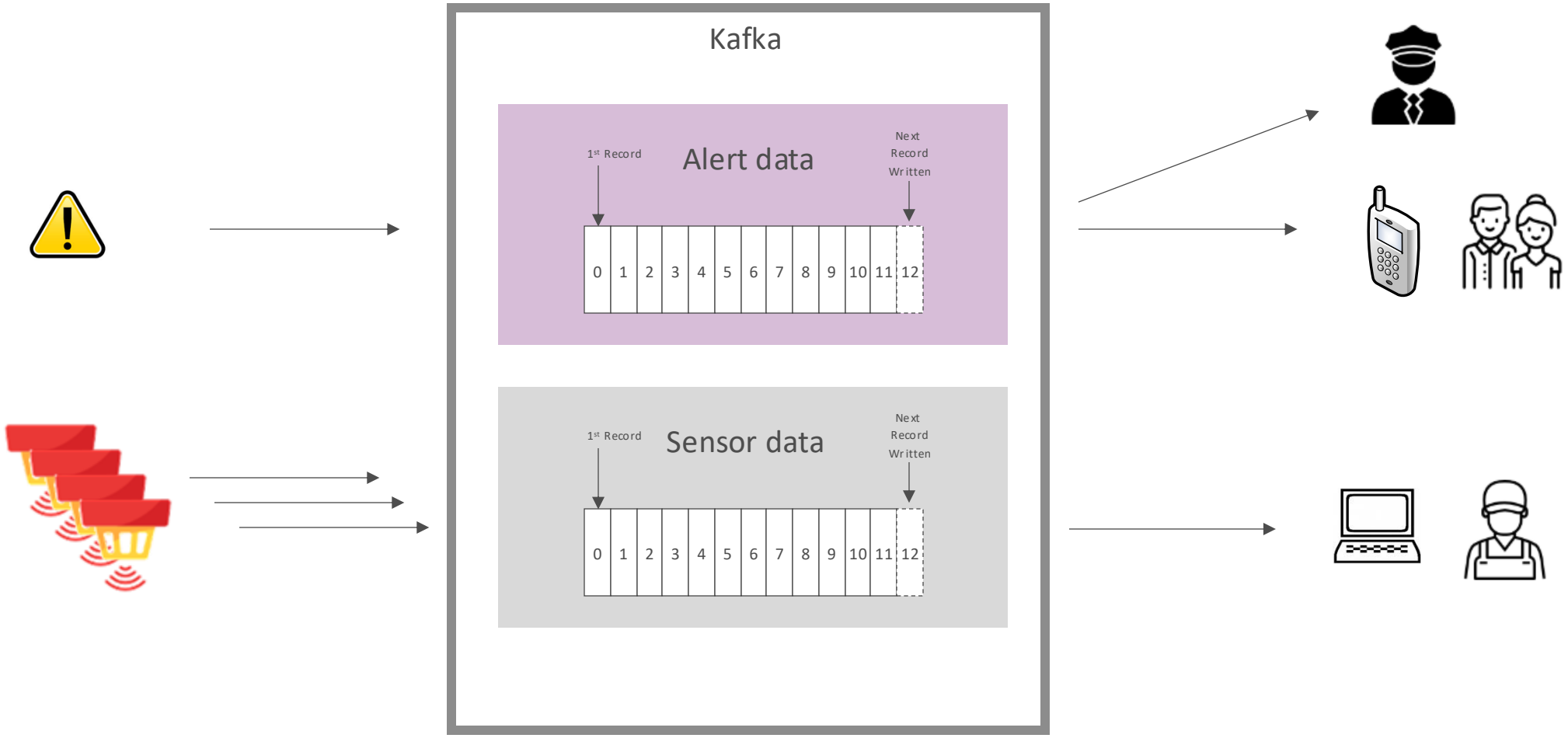
Source: https://kafka.apache.org/documentation/

Read and write data like a pull-based messaging system

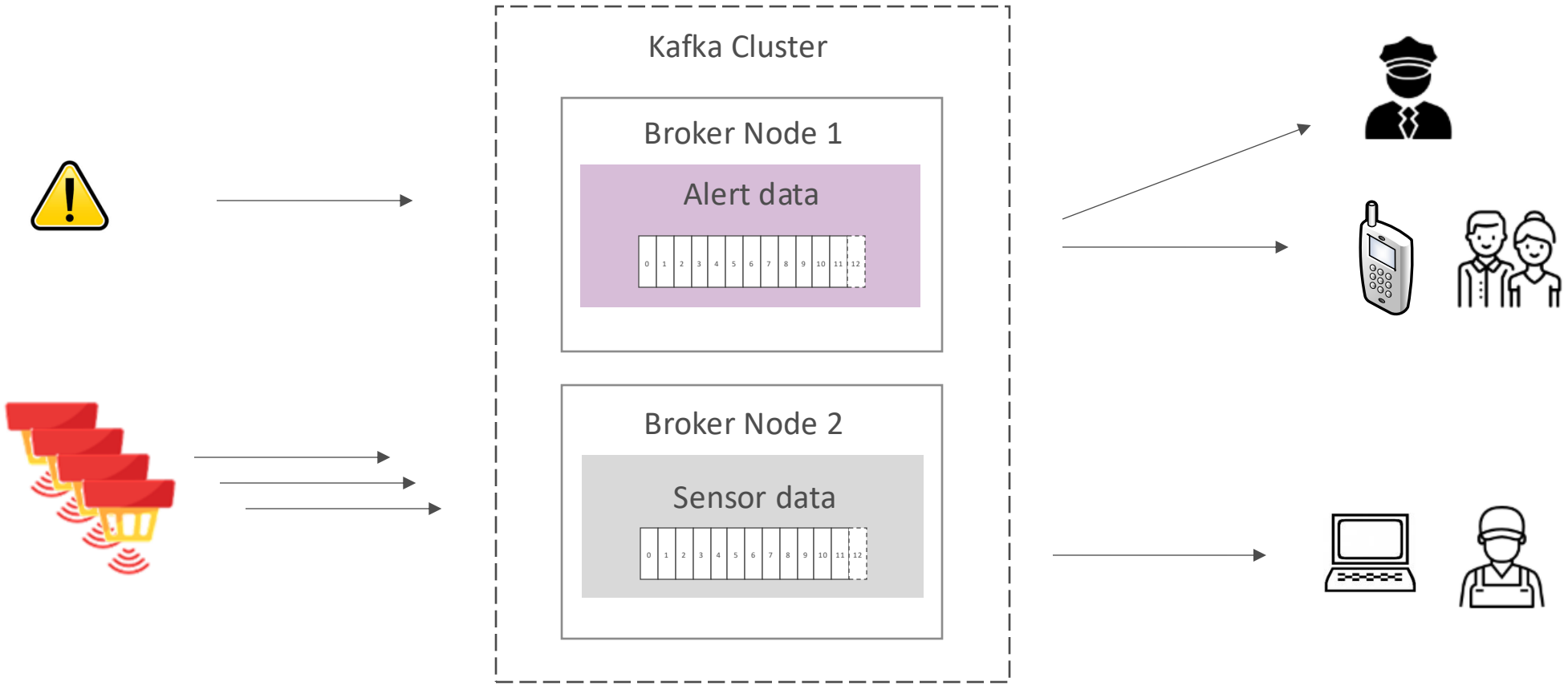


Events can be categorized into topics

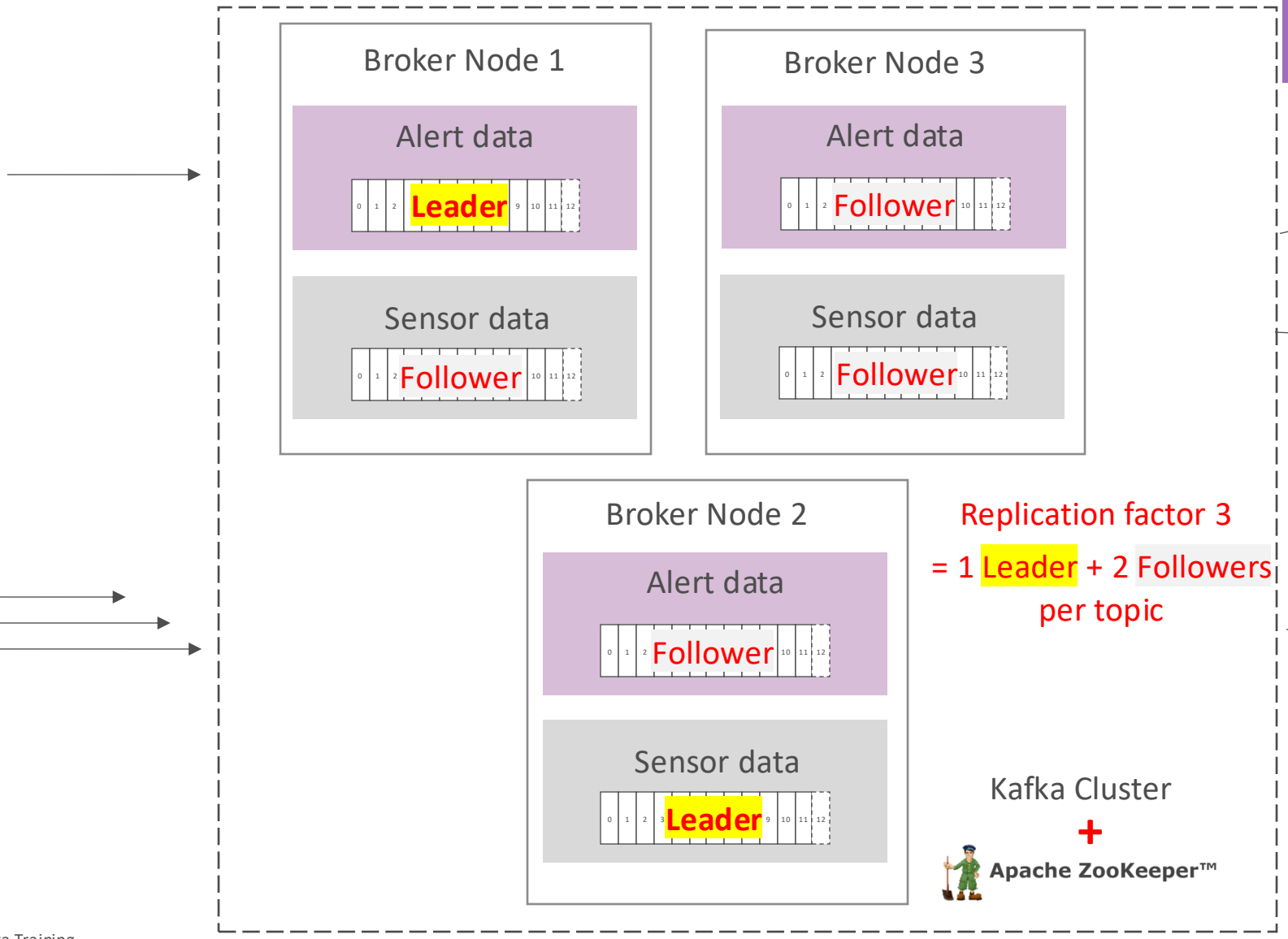
Each topic abstracts a separate log

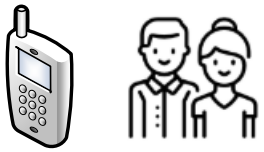


Kafka is designed for multi-broker setup

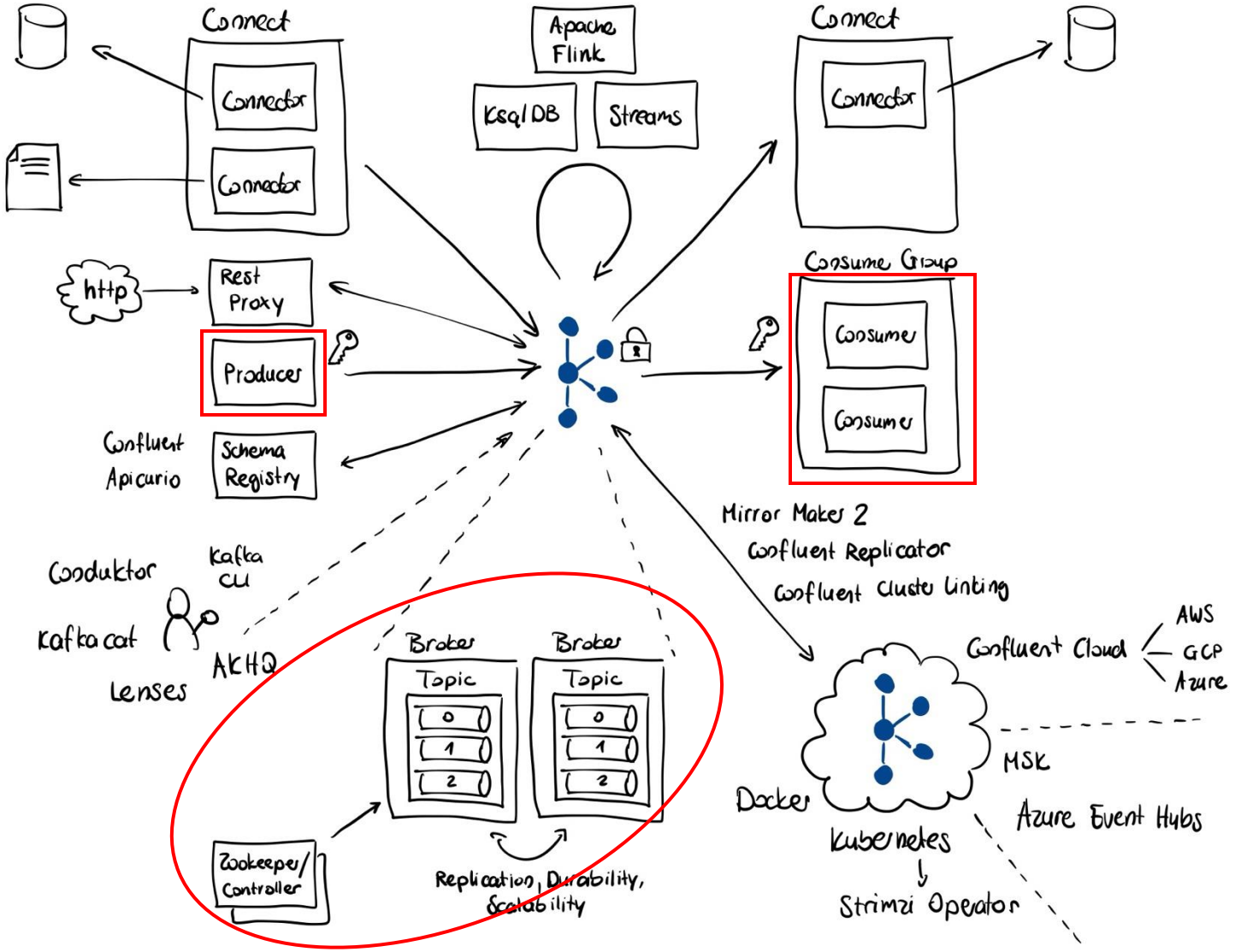


Topics can be replicated for fault tolerance





Overview


Connect
Connector
Connector
Apache Flink
KsqlDB
Streams
Connect
Connector
http
Rest Proxy
Producer
Confluent
Apicurio
Schema Registry
Consume Group
Consumer
Consumer
Mirror Maker 2
Confluent Replicator
Confluent cluster linting
Conduktor
kafka cli
kafkacat
AKHQ
Lenses
Broker
Topic
0
1
2
Broker
Topic
0
1
2
Zookeeper/ Controller
Replication, Durability, Scalability
Confluent Cloud
AWS
GCP
Azure
MSK
Azure Event Hubs
Docker
Kubernetes
Strimzi Operator

Summary

Most important terms

- **Broker:** A single **Kafka server** is called a broker. The broker **receives messages** from producers, assigns offsets to them, and writes the messages to **storage on disk**. It also services consumers, responding to fetch requests for partitions and responding with the messages that have been published. Depending“
- **Offset:** The offset is a simple integer number that is used by Kafka to maintain the current position of a consumer.
- **Topic**: Kafka topics are the **categories** used to organize messages. Each topic has a name that is unique across the entire Kafka cluster.
- **Leader/Follower:** Kafka has the notion of leader and follower brokers. In Kafka, for each topic partition, **one broker is chosen as the leader for the other brokers** (the followers). One of the chief duties of the leader is to assign replication of topic partitions to the follower brokers.
- **Replication:** 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. (fault tolerance)
- **Zookeeper:** Apache Kafka uses Apache Zookeeper to store **metadata about the Kafka cluster**, as well as consumer client details. Zookeeper is a **centralized service for maintaining** configuration information, naming, providing distributed synchronization, and providing group services. Zookeeper is deprecated and will be replaced by KRaft Controllers.
- **Partitioning**: Is what **enables messages to be split in parallel across several brokers in the cluster**. Using this method of parallelism, Kafka scales to support multiple consumers and producers simultaneously. This method of partitioning allows linear scaling for both consumers as well as producers.

Lab Kafka Cluster Part 1

Lab preparation

Finish

local setup: https://github.com/Zuehlke/kafka-streaming-technology/blob/main/training-environment.md

OR

cloud-setup: https://github.com/Zuehlke/kafka-streaming-technology/blob/main/training-env-cloud.md

Lab

Exercise 1: Cluster and topics, failover

Follow “Start the Cluster” and “Exercise 1” from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/cluster.md

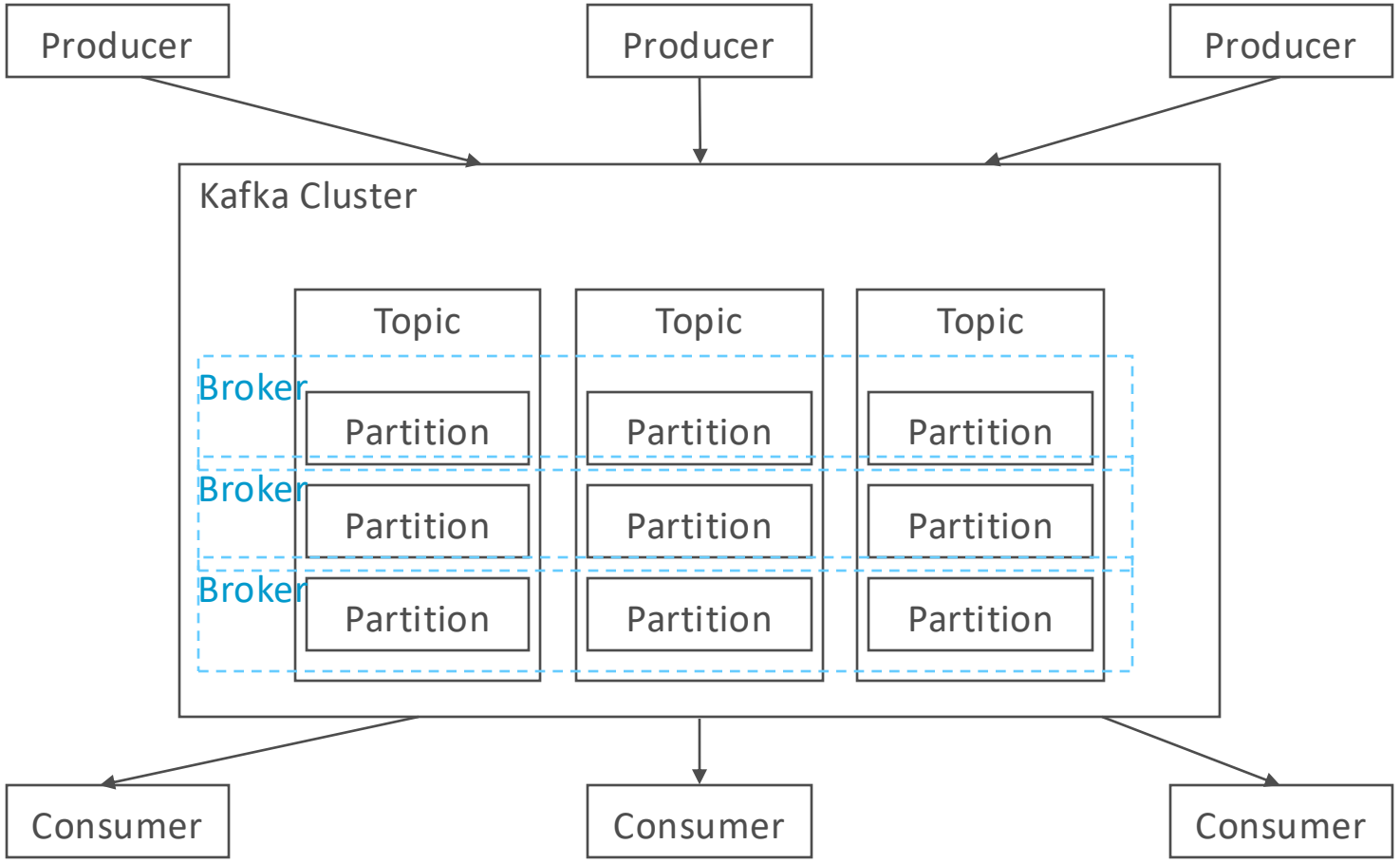
Lab

Exercise 1 – Discuss in the classroom

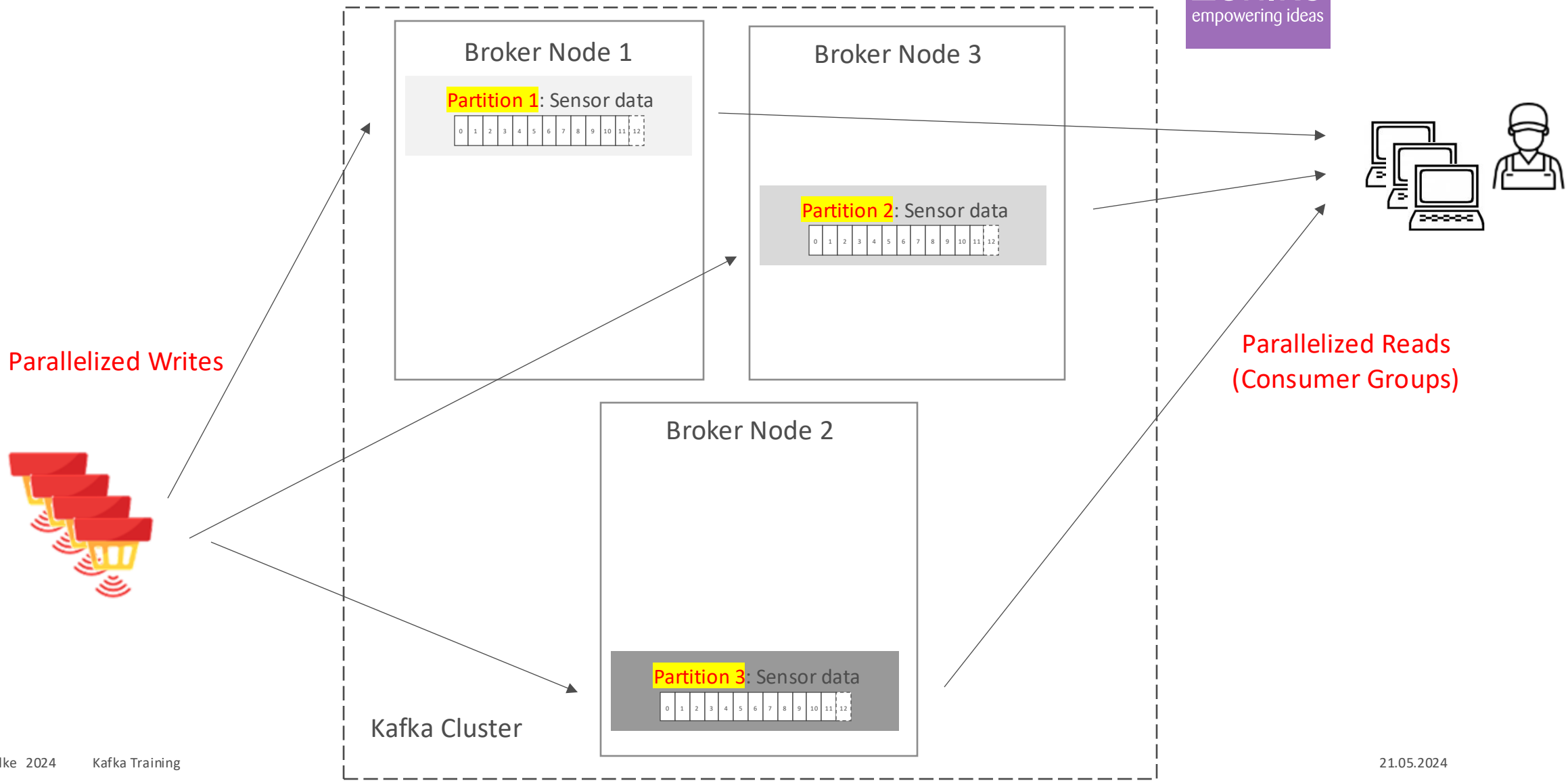
- What happens to the replicas if you create data with only one broker alive? Is this behavior desired?

Kafka Architecture & Concepts Part 2

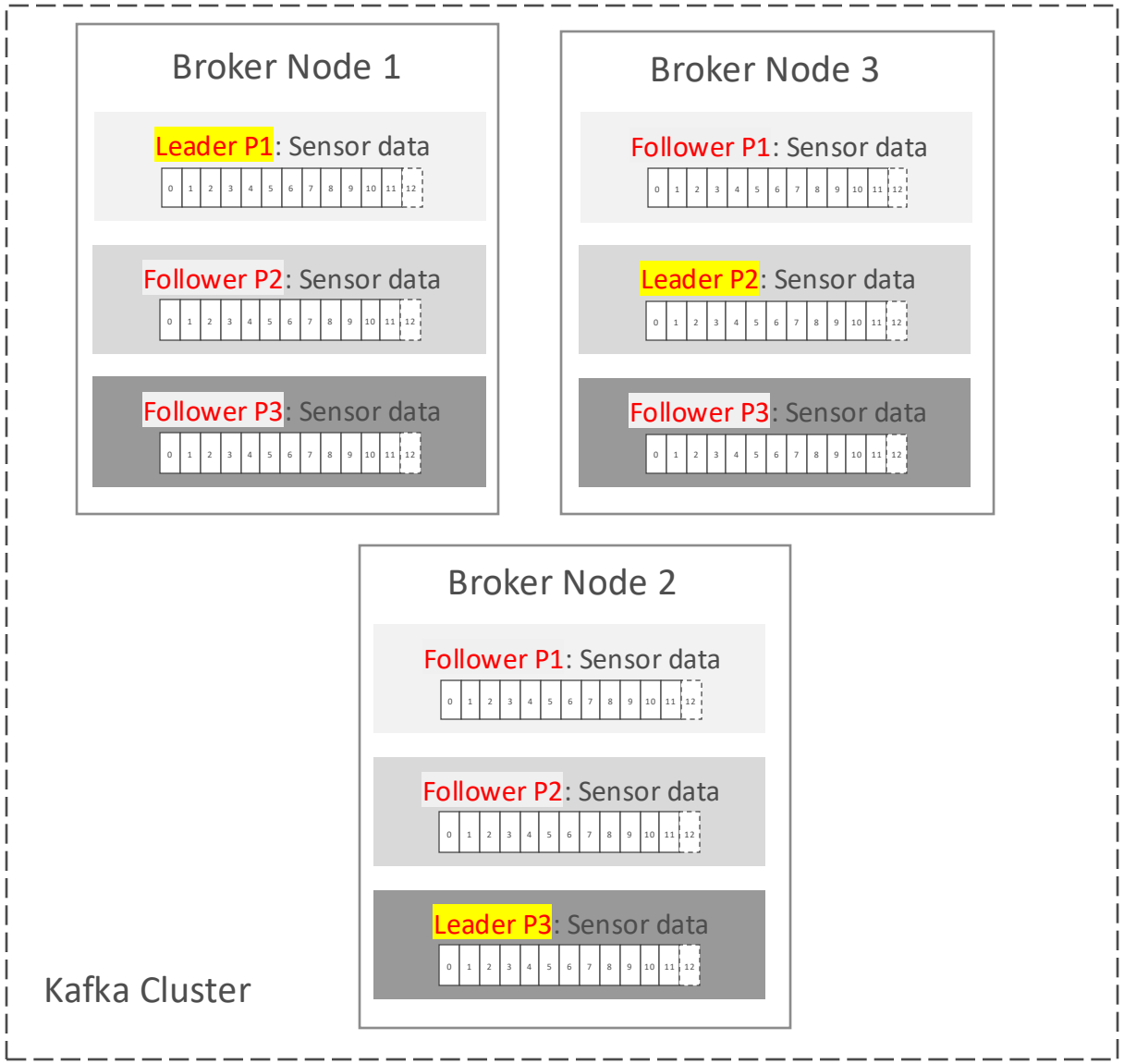
Scale out a topic through partitioning

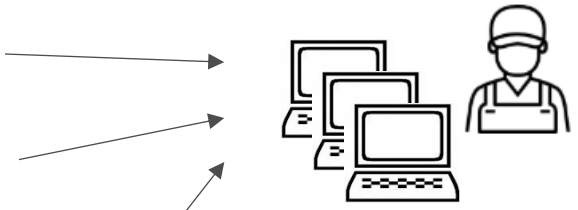


Topics can be split into partitions to optimize throughput



Replication applies on partition level





Replication factor 3
= 1 Leader + 2 Followers
per partition

Zookeeper vs. KRaft Controller

With Confluent Community release 7.5 (Apache Kafka release 3.5) Zookeeper is marked as deprecated.

- In your cluster configuration, you can replace Zookeeper-Instances through Broker-Instances in *controller* mode.
- For all learnings regarding the cluster-management in this course you can replace a Zookeeper-Instance through a Broker-Instance that runs in *controller* mode.
- NOTE: Apache Kafka also introduces a combined mode, where a Kafka node acts as a *broker* and as a *Kraft controller.* This mode is currently not supported for production workloads.

Lab Kafka Cluster Part 2

Lab

Exercise 2: Partitioning and Partition Keys, Consumer Groups

Follow “Exercise 2” from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/cluster.md

Lab

Exercise 2 – discuss in the classroom

- What did you observe?
- What are the key aspects you will consider when you design your Kafka-application?

Kafka Architecture & Concepts
Cluster design

Message Keys and Consumer Groups

Consumer Groups

- With consumer groups, we learnt that messages of a topic can be read parallel.
- At the same time, Kafka guarantees that messages have the same order as they are written to the log.
- How is this possible if they are distributed to different partitions?

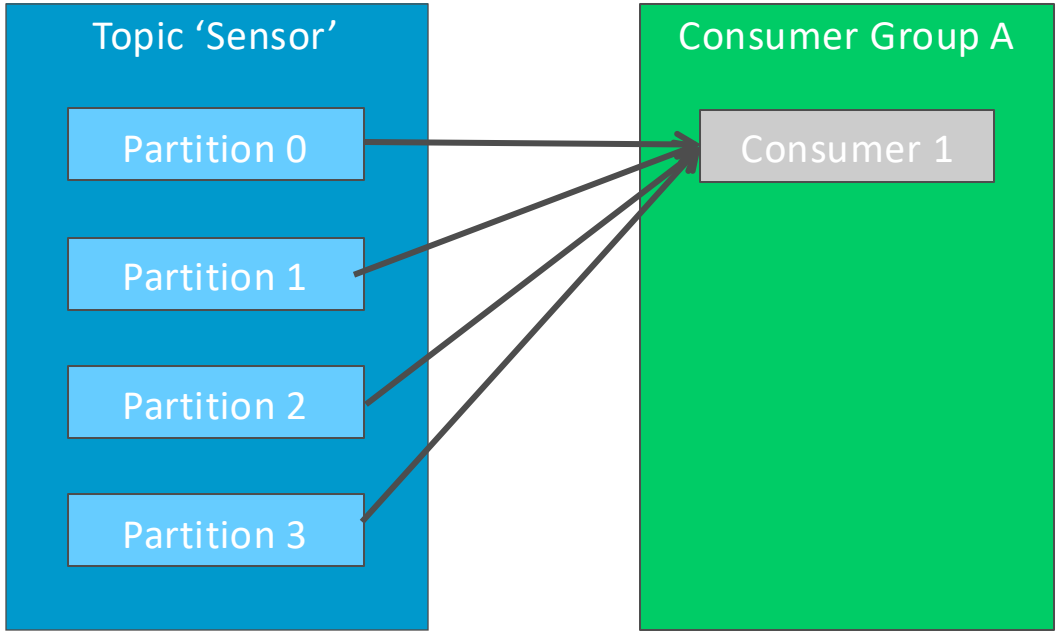
Rules:

Having a Topic ‘Sensor’ with multiple Partitions, a consumer of a consumer group reads:

- From different partitions
- Not from the same partitions than other consumers read

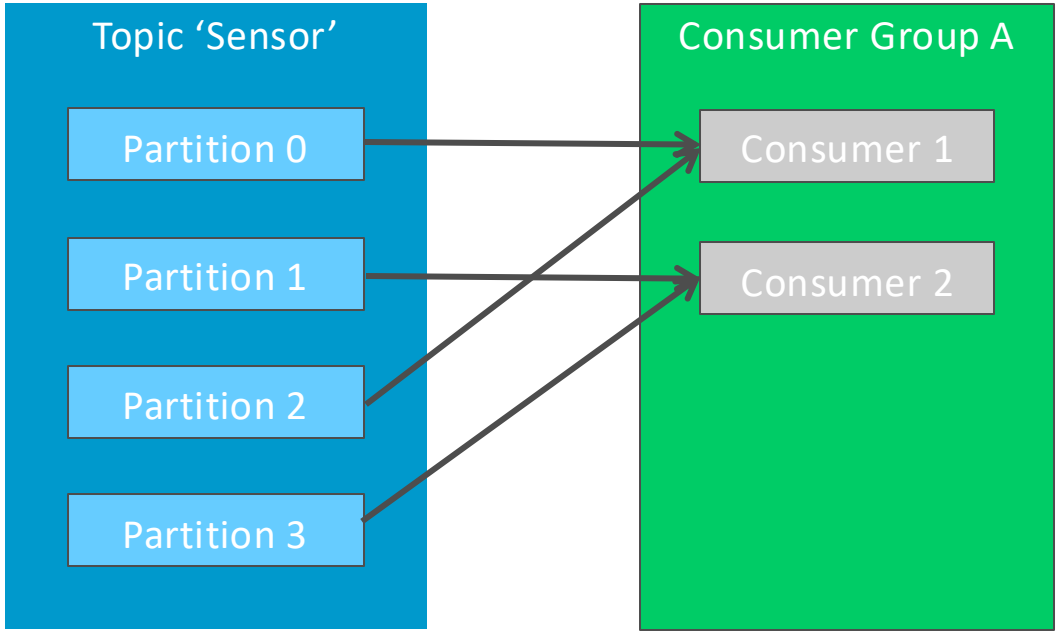
Message Keys and Consumer Groups

Consumer Groups: 1st consumer processes all partitions



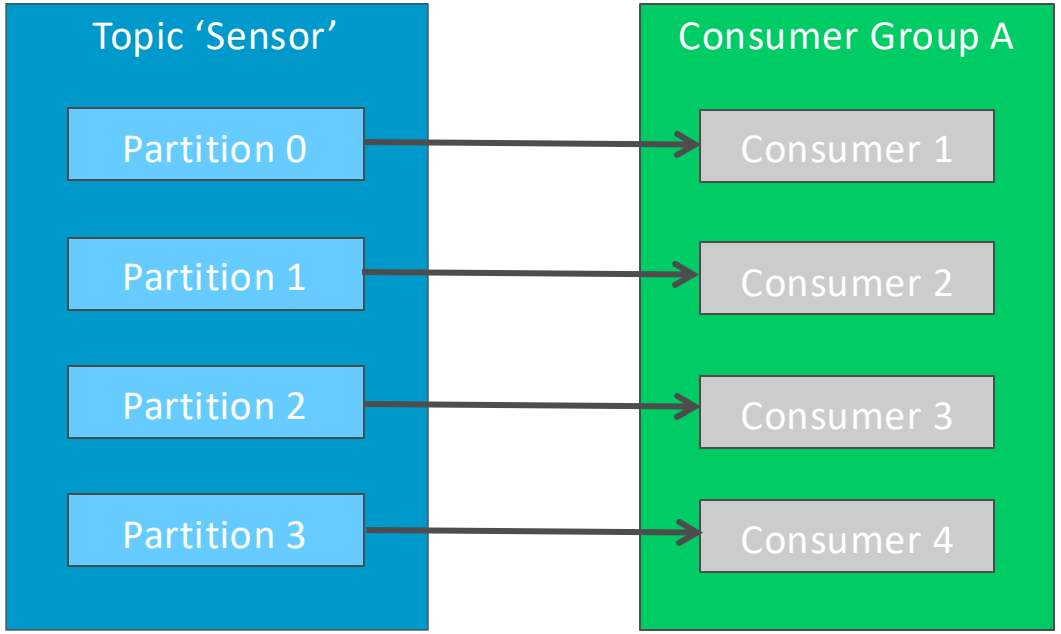
Message Keys and Consumer Groups

Consumer Groups: 2nd consumer joins the consumer group



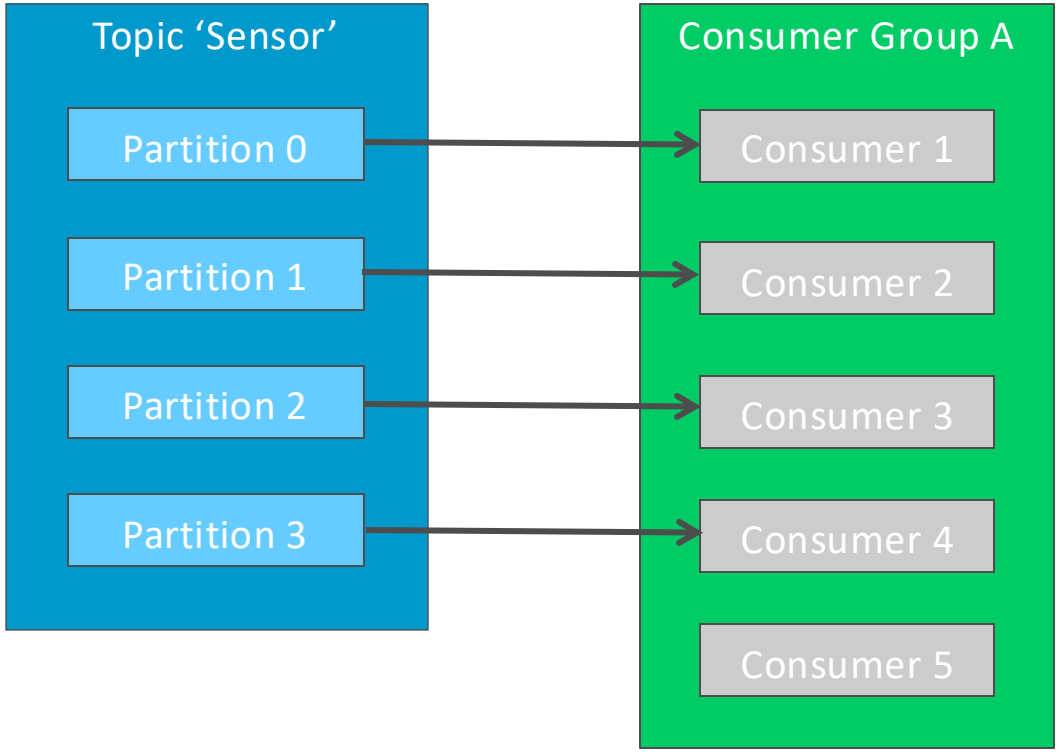
Message Keys and Consumer Groups

Consumer Groups: Same number of partitions and consumers



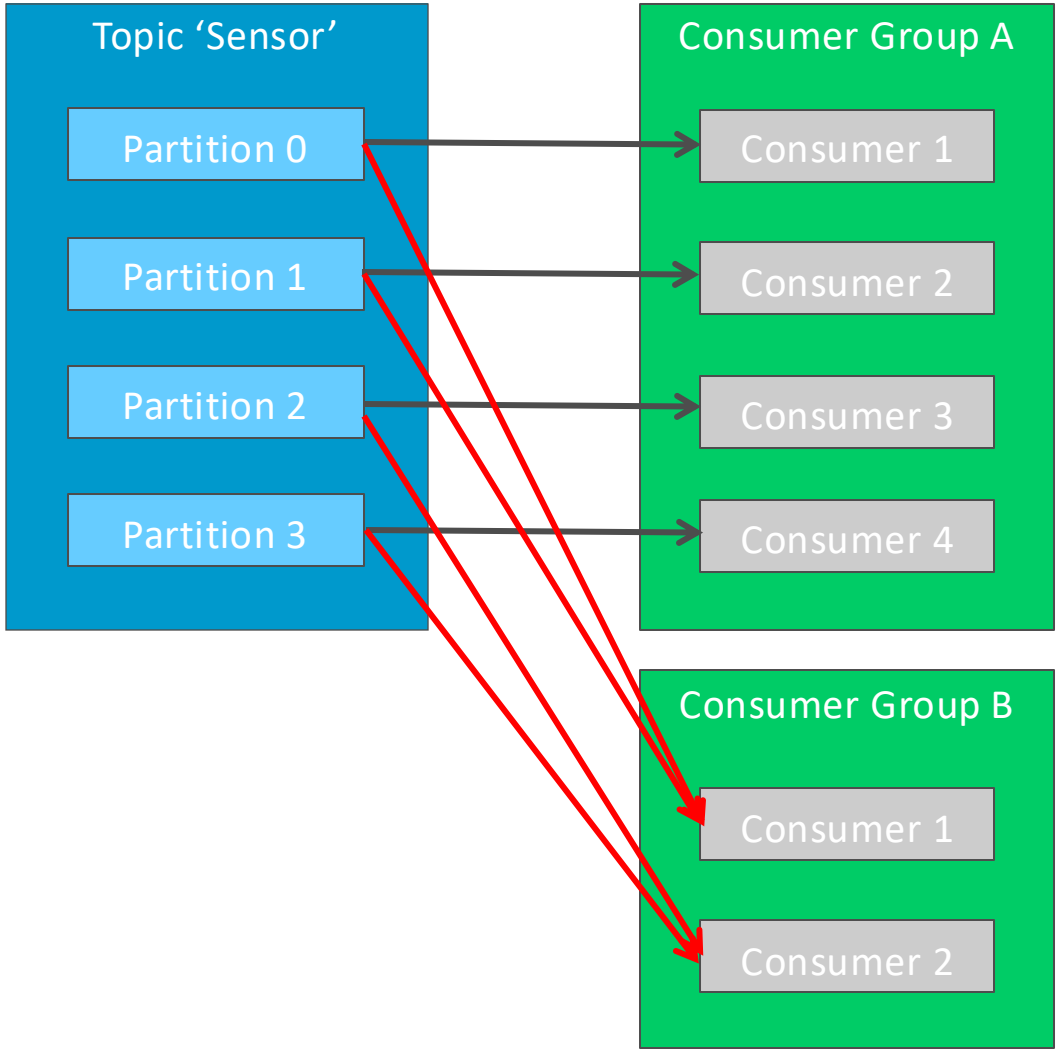
Message Keys and Consumer Groups

Consumer Groups: Number of partitions defines the consumers



Message Keys and Consumer Groups

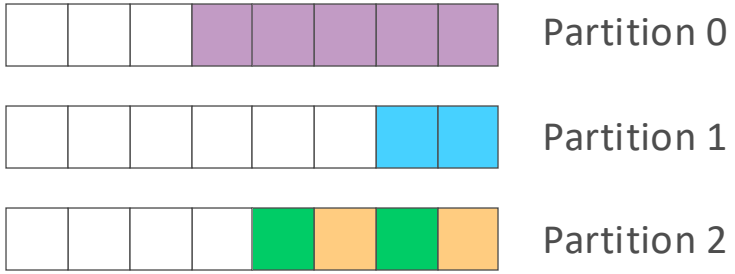
Consumer Groups: Adding another consumer group



Message Keys and Consumer Groups

Message Keys

- When sending a message (log entry) to Kafka, the producer can also specify a message key. This key defines to which partition a message is sent. The Colors in the following picture represent different messages with the same key:



- In our sensor topic, the key of a Kafka message could be the sensor id. That means, that the time series of a given sensor will be consumed in the same order as the producer (sensor) sent them to Kafka, since only one consumer reads from the same topic at the same time.
- It is a good practice to use the aggregate id of an object as Kafka Key while publishing domain events to Kafka.

Message Keys and Consumer Groups

Message Key Examples

- Which field would you use as message key in the following domain examples:

User
ID
Name
First Name
E-Mail
…

Sensor
ID
SerialNo
Type
Value
Unit
…

(Hint: User logs in with his e-mail)

Cluster Design

How to choose the number of partitions in a Kafka topic

Scalability View:

- If you have **one partition**, you have **max 1 machine** serving a topic (what perhaps is already enough).
  - **More partitions** lead to **higher throughput**
  - The throughput per partition depends on the message size, compression, topic- and consumer structure…

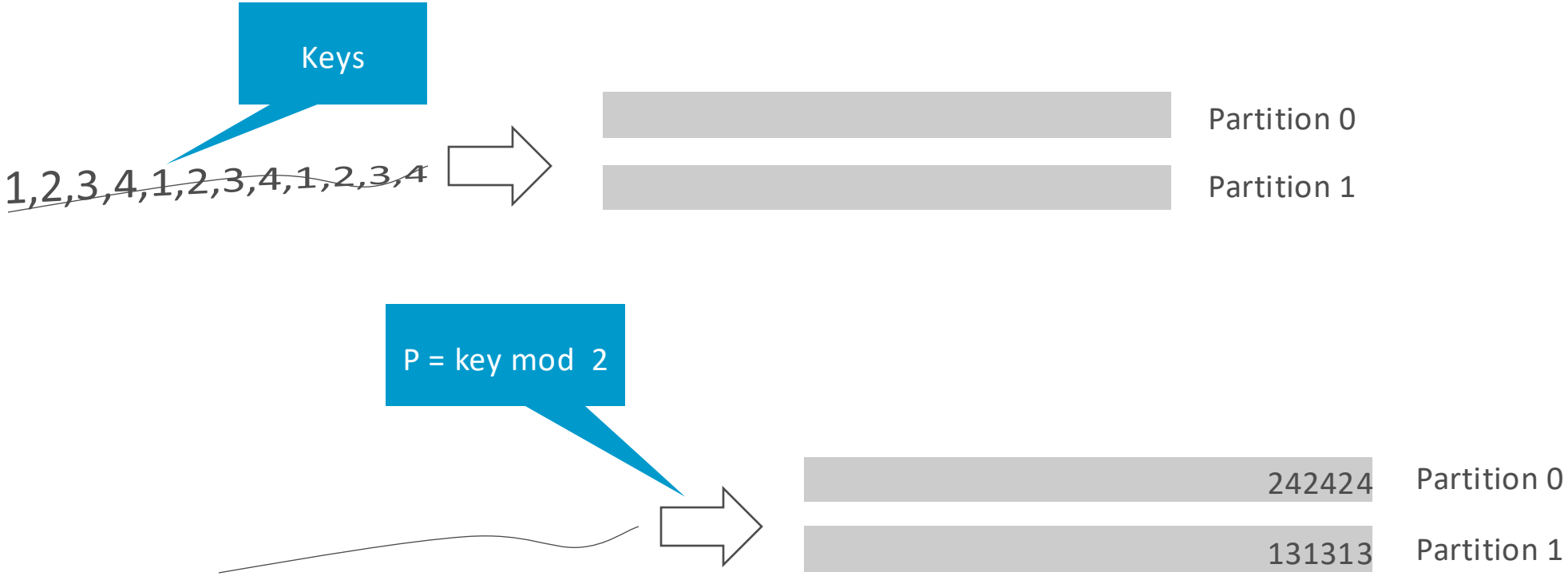
- Theoretically, we could start to configure the # of partitions in our cluster with 1, and increase as needed, but:

Repartitioning lead to data inconsistency => *the best is to get the parameters right at the first time!*

Why?

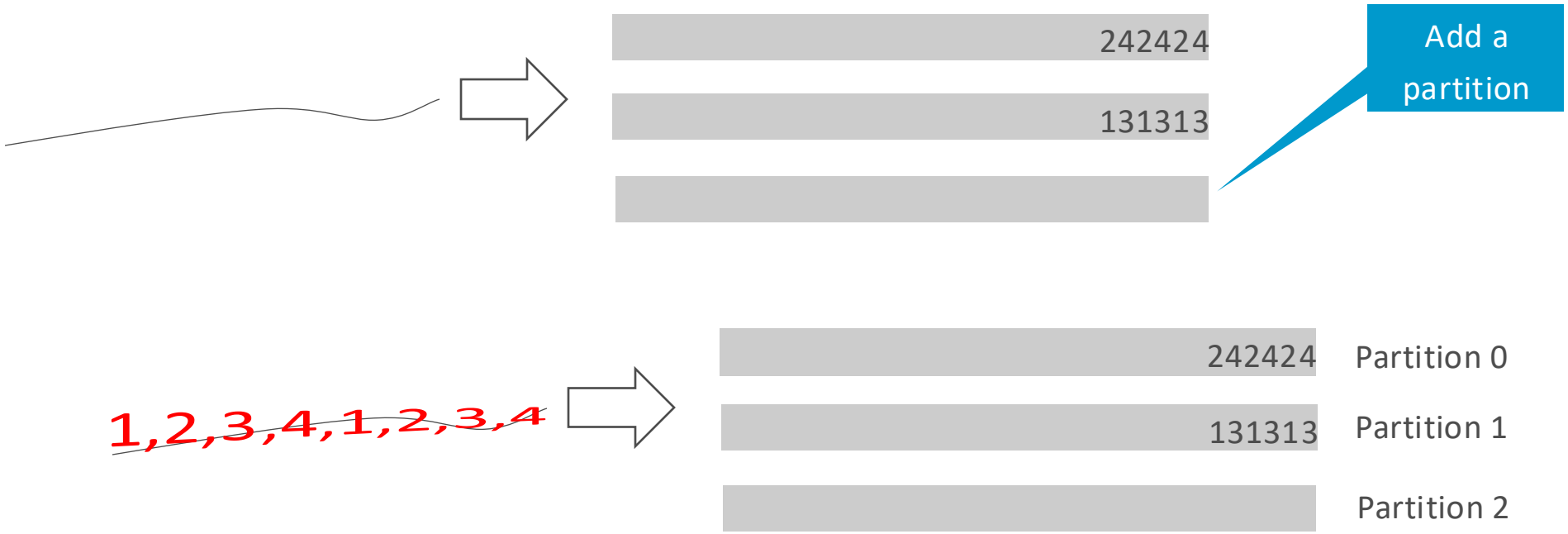
Cluster Design

Late partitioning problem



Cluster Design

Late partitioning problem



Cluster Design

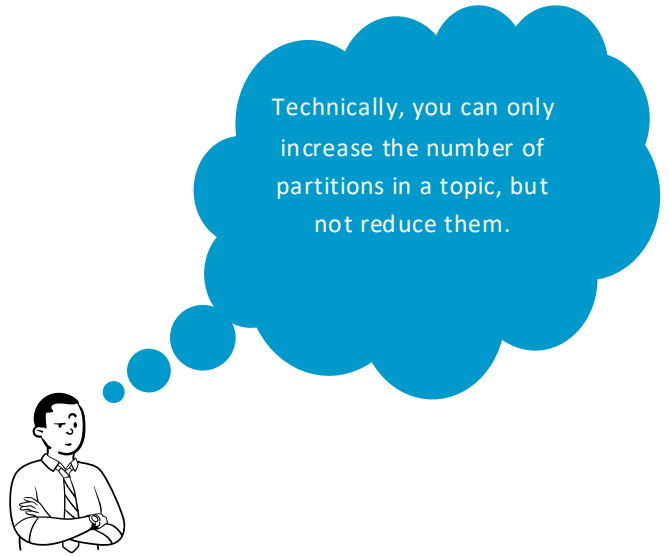
Late partitioning problem

Cluster Design

How to choose the number of partitions in a Kafka topic

Scalability View: What is the upper limit of partitions?

Can’t we just set 1000 partitions per topic?



- More partitions require more open file handles.
- More partitions may increase unavailability.
  - longer leader election process on broker failure, because one broker has more partitions.
- More partitions may require more memory in the client.
  - The producer accumulates messages before committing them to the broker (depending on your settings). More partitions => more messages are accumulated.
- Stream applications create automatically temporary topics. The effects above are then multiplicated.

Cluster Design

Conclusion

Out of our project experience

- In a **typical message-based** system you might set the number of partitions to **~10**. Particularly if you use streaming extensively.
- If you know in advance that you have **high load**, you might also start **with 20 partitions** for the given topics.

Additionally

- In a typical **HA-Environment** you a good number of brokers is 6. Depending on the load, the number of brokers might be up to 10.

Data cleanup management

Options

What about disk cleanup for old messages?

Kafka has different options on how messages will be removed if they are no longer needed.

1. Never
2. After a given time (retention.ms)
3. When a given amount of disk space is used (retention.bytes)

In practice, the option (2) is mostly used and is also the default setting. Further, there is a semantically interesting option called ‘compaction’.

4. Only the newest message with the same key is kept.

Data cleanup management

Cleanup old messages

Kafka brokers splits each partition into segments. Each segment is stored **in a single data file** on the disk attached to the broker. By default, each segment contains either **1 GB of data or a week of data**, whichever limit is attained first. When the Kafka broker receives data for a partition, as the segment limit is reached, it will close the file and start a new one:

Removing data from the broker means always to remove the whole segment. With other words, an active segment will never be removed.

Data cleanup management

Cleanup old messages – «retention.ms»

Option 1: After a given time

retention.ms

- 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.
  - -1 => never delete
  - Default: 604800000 (in ms => 7 days)

Data cleanup management

Cleanup old messages – «retention.bytes»

Option 2: When a given amount of disk space is used

retention.bytes

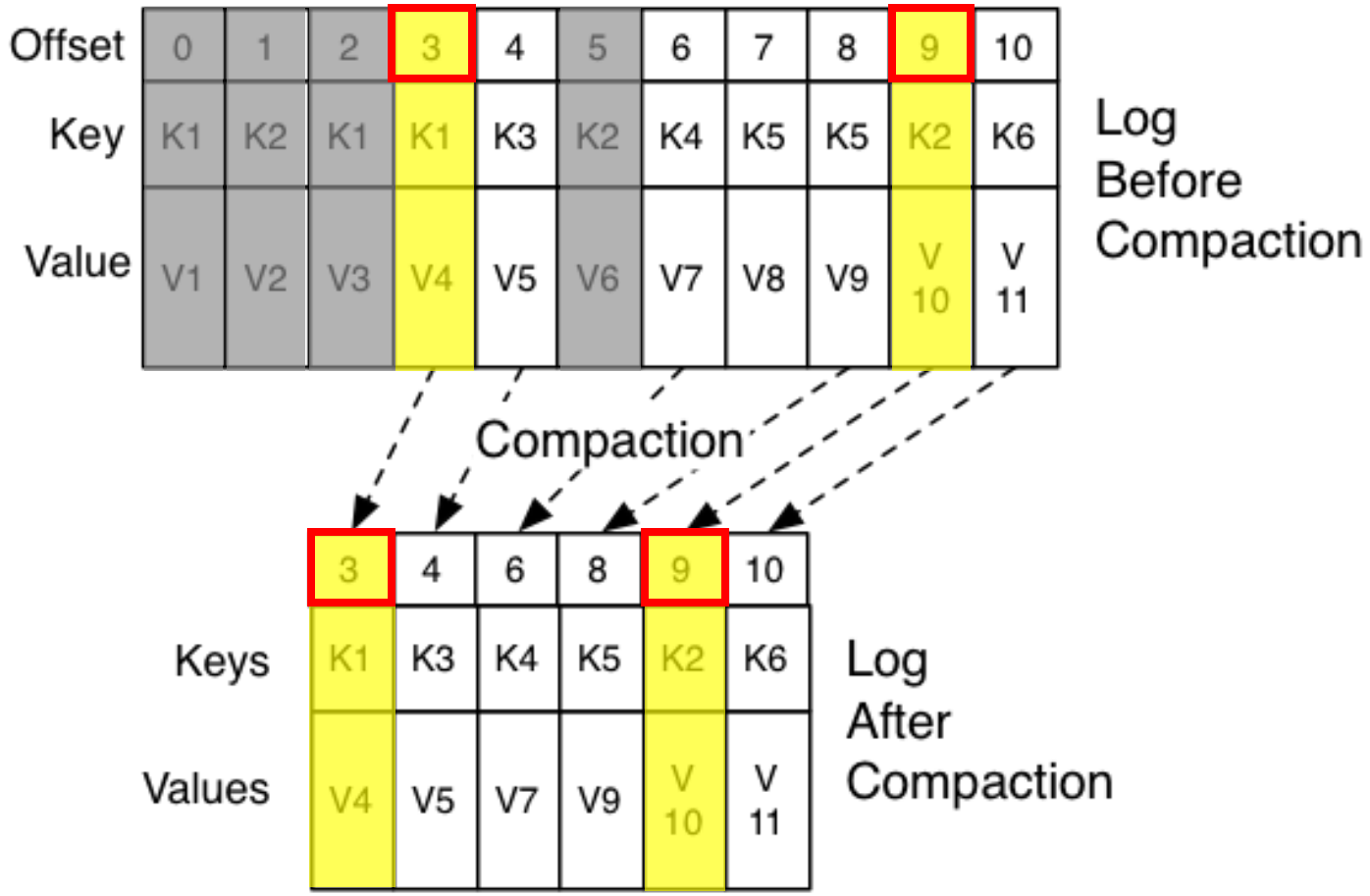
- This configuration controls the maximum **size a partition** can grow to before we will discard old log segments if we are using the "delete" retention policy.
  - 10’073’741’824 => 10 GB **per partition**
  - Default: -1 (never)

Example: topic with 10 partitions, segment size 1 GB (default), no replication:

10 GB per partition x 10 partitions + 10 x 1GB ‘active segments’ = 110 GB maximum disk space needed

Data cleanup management

Cleanup old messages – «compact» retention policy



Source: https://kafka.apache.org/documentation/#compaction

Data cleanup management

Cleanup old messages – «compact» retention policy

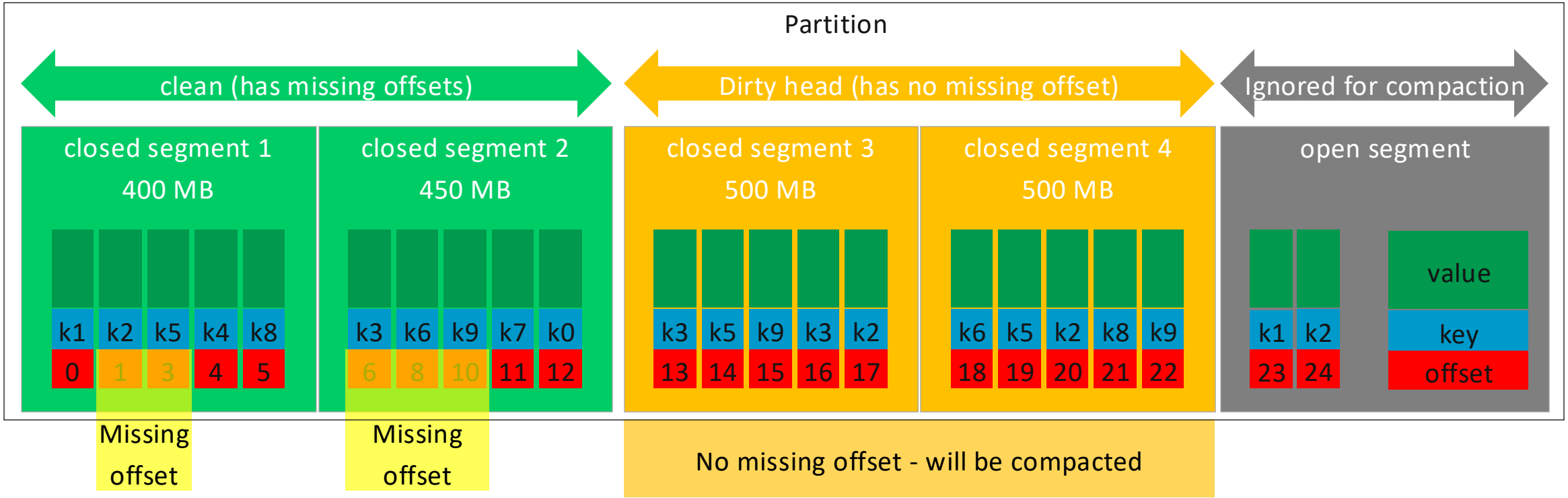
With this cleanup policy, the Kafka message key becomes more semantics. We already discussed that the key is related to an aggregate (business object). Cleanup removes old messages with the same key from a topic. With other words, if you use an aggregate id as key, old messages of an aggregate are removed.

Some Rules:

- The **active segment** is not taken into consideration during compaction. => we can always have several messages with the same key
- In the ‘old’ segments, the compaction process copies the newest message of each key to a new segment.
  - The index of the message is maintained.
  - After compaction, old segments are replaced by the new segments (that contain the copy of the newest messages). Old segments are deleted from the disk.
- **Compaction is triggered** when a new message arrives and
  - A given (configurable) time is elapsed or
  - A given (configurable) dirty ratio is reached

Compaction

Clean and dirty records



By default, compaction starts if the partition's dirty ratio is > 0.5

Which means that compaction will be triggered if more than 50% of the total segment file size is dirty.

(500 MB + 500 MB) / (400 MB + 450 MB + 500 MB + 500 MB) = 0.54

Lab Kafka Cluster

Part 3

Lab

Exercise 3: Log compaction

Follow “Exercise 3” from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/cluster.md

Recap

What did we learn so far:

Kafka building blocks:

- Distributed commit log
- Topics
- Replication
- Partitioning
- Consumer Groups
- Keys
- Log Compaction

Kafka applied:

- # of replicas and partitions
- # of consumers
- Keys and Aggregates
- When to use log compaction

HowTo

Skip a (poisoned) message

Check the Lag of your consumer group:

```
 ./kafka-consumer-groups --bootstrap-server localhost:19092 --describe --group myGroup
GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
myGroup         sensor2         0          20              40              20              -               -               -
myGroup         sensor2         1          22              44              22              -               -               -
```

Skip a bad message in one specific partition (partition 0)

```
./kafka-consumer-groups --bootstrap-server localhost:19092 --reset-offsets --topic sensor2:0 --shift-by 1 --group myGroup --execute
GROUP                          TOPIC                          PARTITION  NEW-OFFSET
myGroup                        sensor2                        0          21
```

Skip a bad message of all your partitions

```
./kafka-consumer-groups --bootstrap-server localhost:19092 --reset-offsets --topic sensor2 --shift-by 1 --group myGroup -execute
GROUP                          TOPIC                          PARTITION  NEW-OFFSET
myGroup                        sensor2                        0          22
myGroup                        sensor2                        1          23
```

Case study: IoT

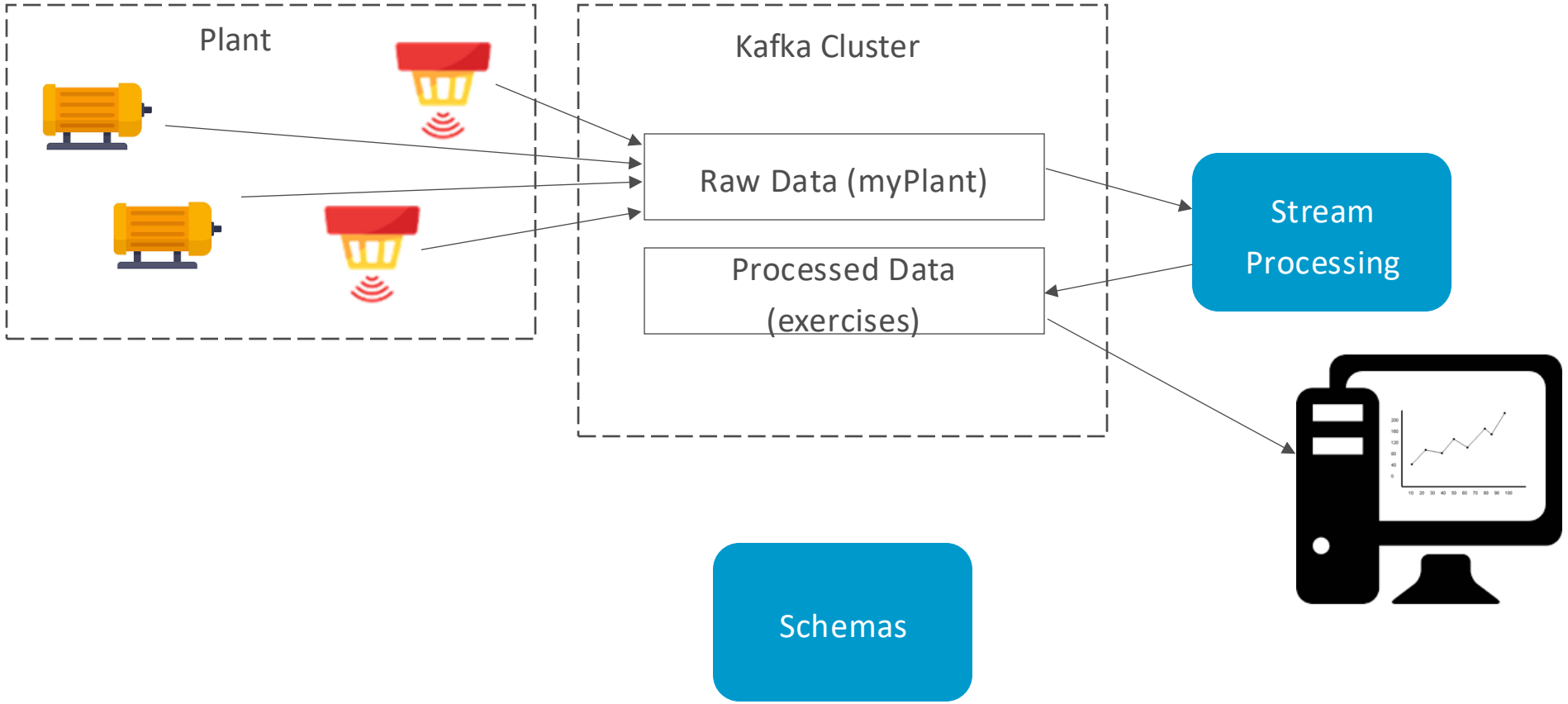
IoT Use Case

In this Use Case, you as a software engineer **must support a company** that is specialized in **generating energy from waste** (e.g. using the heat produced by burning litter, but also extracting bio-gas from organic waste). **The company decided to collect the data from the different plants by Apache Kafka and hired you to help them to manage and transform the data to support their operation processes.**

Typically, you should transform time series **signals** (temperature, pressure, etc.) but also signals from **processing** units (motors…) according to needs of the process engineers responsible for plant operations.

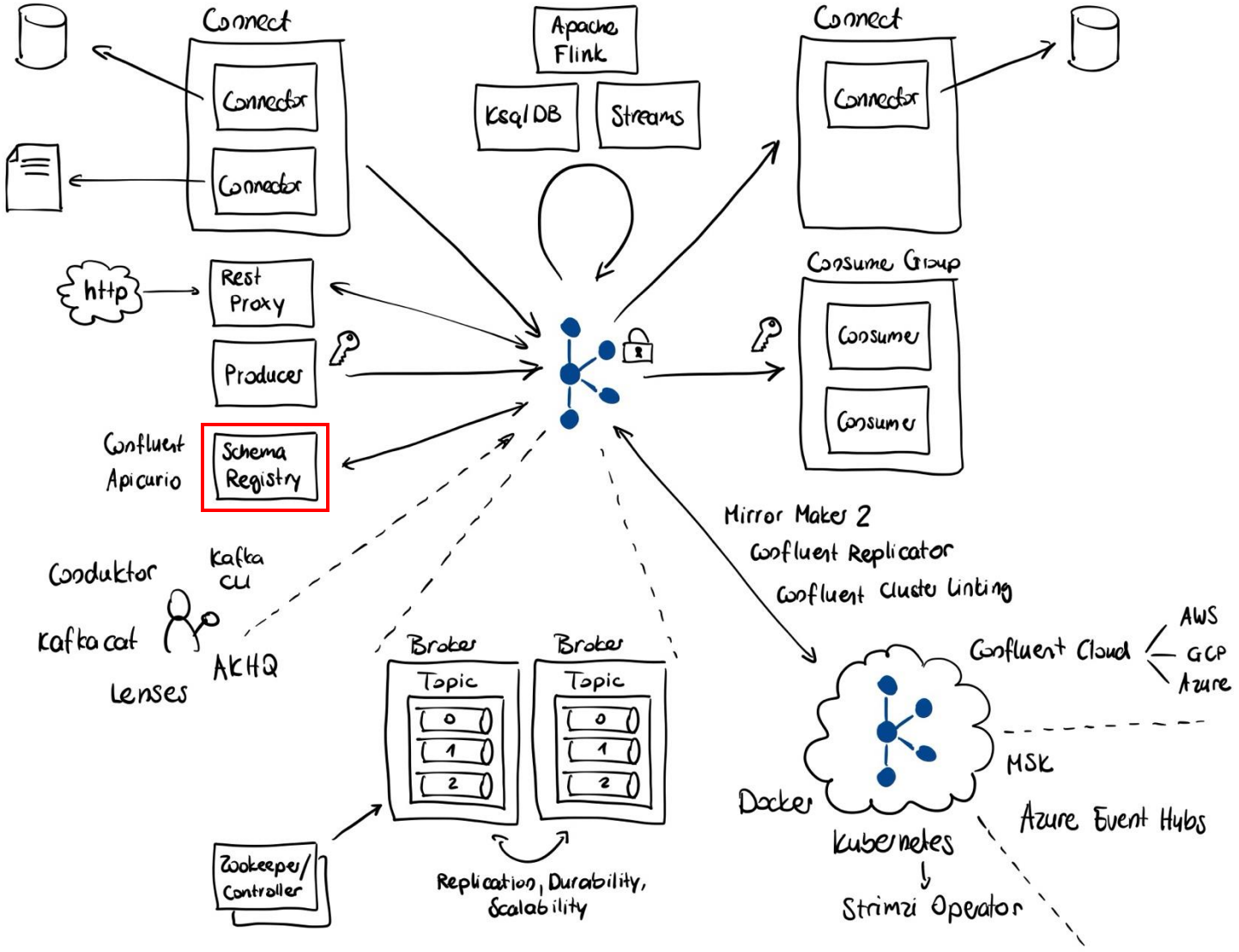
IoT Use Case

Exercise



Schema Registry

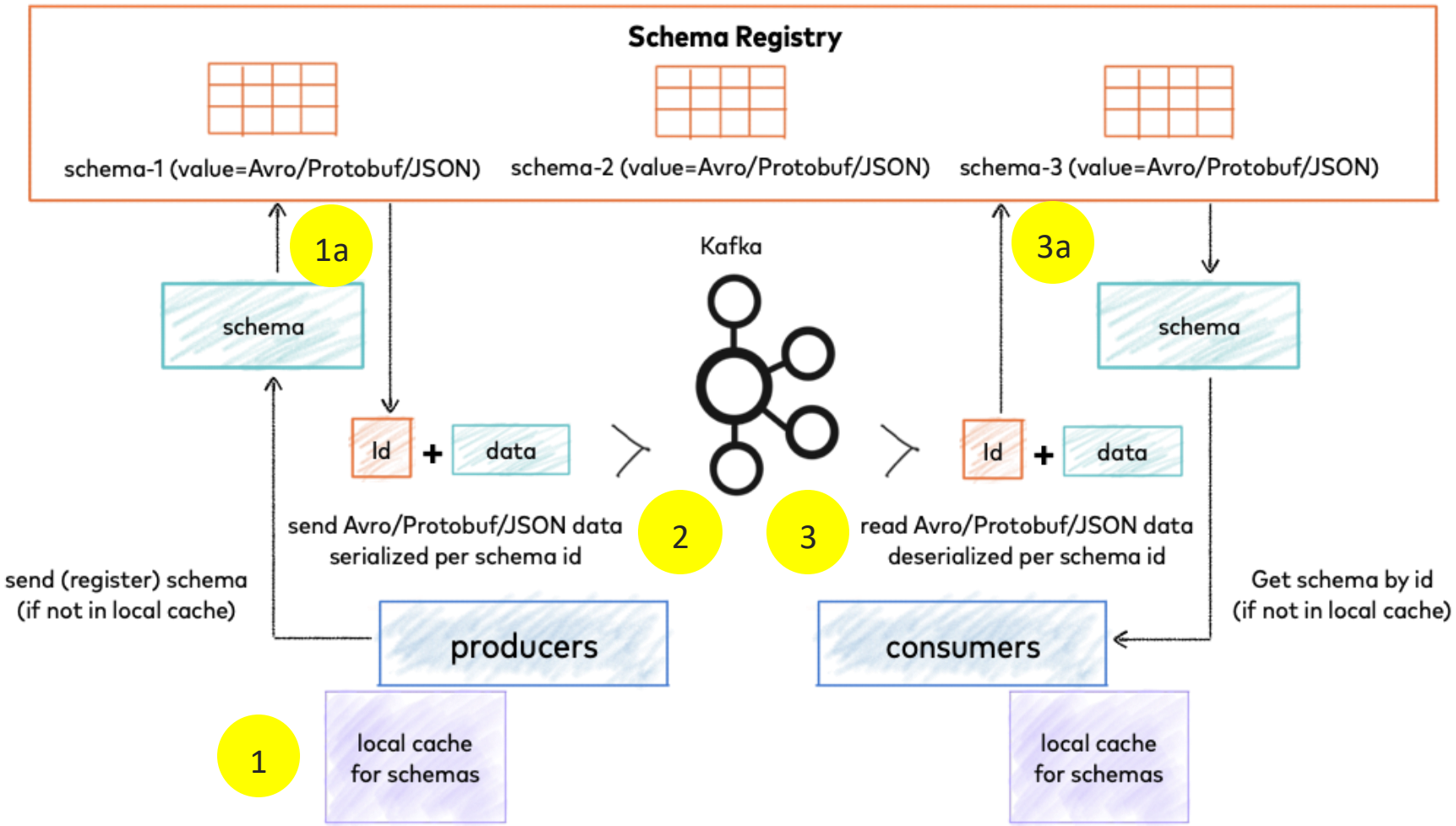
Overview



Advantages of schema registry

1. **Schema Evolution:** A Schema Registry helps in managing schema evolution. Formats **change over time**, you can version and evolve schemas without breaking compatibility with existing consumers. Newer consumers can still understand and process older message versions.
2. **Data Validation:** The Schema Registry enforces **schema validation** on incoming and outgoing messages. This ensures that only messages adhering to a specified schema are produced or consumed, preventing data quality issues.
3. **Compatibility Checks:** When a new **schema version is introduced**, the Schema Registry can check for compatibility with existing schemas. This helps avoid issues that might arise from incompatible changes and ensures a smooth transition.
4. **Centralized Schema Storage:** Storing schemas **centrally in a registry** simplifies schema management. It's easier to maintain and govern schemas when they are stored in one place, making it accessible to different producers and consumers.
5. **Consumer Flexibility:** Consumers can **dynamically fetch the schema from the registry**, allowing them to adapt to schema changes without needing code modifications. This is particularly useful in microservices architectures and data streaming applications.
6. **Schema Documentation:** A Schema Registry often **includes documentation for schemas**, making it easier for developers to understand the structure of messages and how to interact with them.

Confluent Schema Registry



Source: https://docs.confluent.io/platform/current/schema-registry/index.html

Schema evolution in Kafka

Summary

Evolution	Original Schema	New Schema	Update order
Add a mandatory field	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "name" : "value" ]`	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "name" : "value", "name" : "type" ]`	First: Producer Then: Consumers
Remove a mandatory field	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "name" : "value" ]`	`"fields" : [ "name" : "sensor_id", "name" : "value" ]`	First: Consumers Then: Producer
Add an optional field	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "name" : "value" ]`	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "name" : "value", "name" : "type", "default": "cm" ]`	Doesn’t matter
Remove an optional field	`"fields" : [ "name" : "sensor_id", "name" : "datetime", "default": 0 "name" : "value" ]`	`"fields" : [ "name" : "sensor_id", "name" : "value“ ]`	Doesn’t matter

Schema Evolution

The Schema Registry supports automatic schema evolution

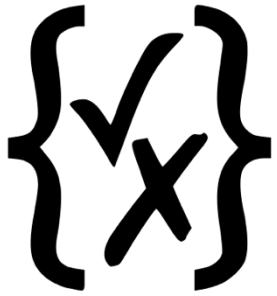
Compatibility Type	Changes Allowed	Update first	Description
BACKWARD (default)	• Delete Fields • Add optional Fields	Consumer	"consumers using the new schema can read data produced with the last schema"
FORWARD	• Add Fields • Delete optional Fields	Producer	"data produced with a new schema can be read by consumers using the last schema"
FULL	• Add optional Fields • Delete optional Fields	Any Order	"old data can be read with the new schema, and new data can also be read with the last schema"
NONE	• All Changes Allowed	Depends (no compatibility checks)	

Schemas

Contract between Producer and Consumer

No Schema

vs



Avro Schema Definitions

Optional Fields example:

```
{
    "name" : ”age",
    “alias": "value_with_null",
    "type" : [ “null”, "long“ ],
    "doc"  : "…"
    "default": 25
}
```

The field has additionally this name

Optional Field in Avro

Defines what is used if there is ‘nothing’:

- ~~Produced by default~~
- Consumed if missing

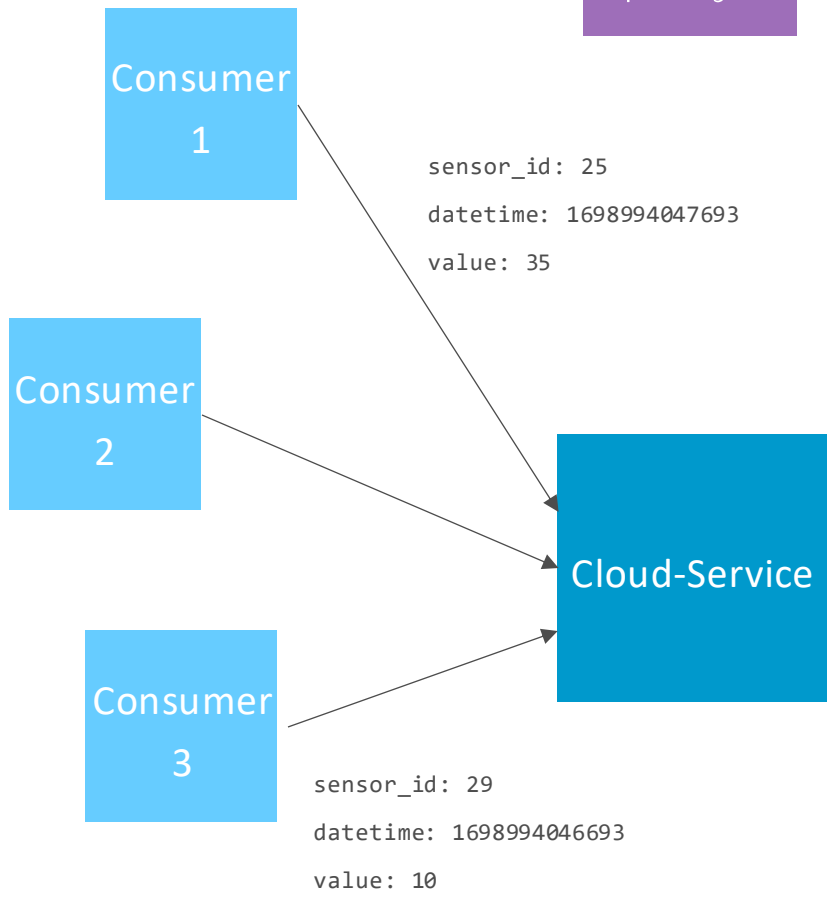
Explicit versioning example:

```
{
  "namespace": "com.zuehlke.training.kafka.iot.v1",
  "type": "record",
```

Might be an option for ‘totally different’ versions. Consumers / Producer must be managed explicitly.

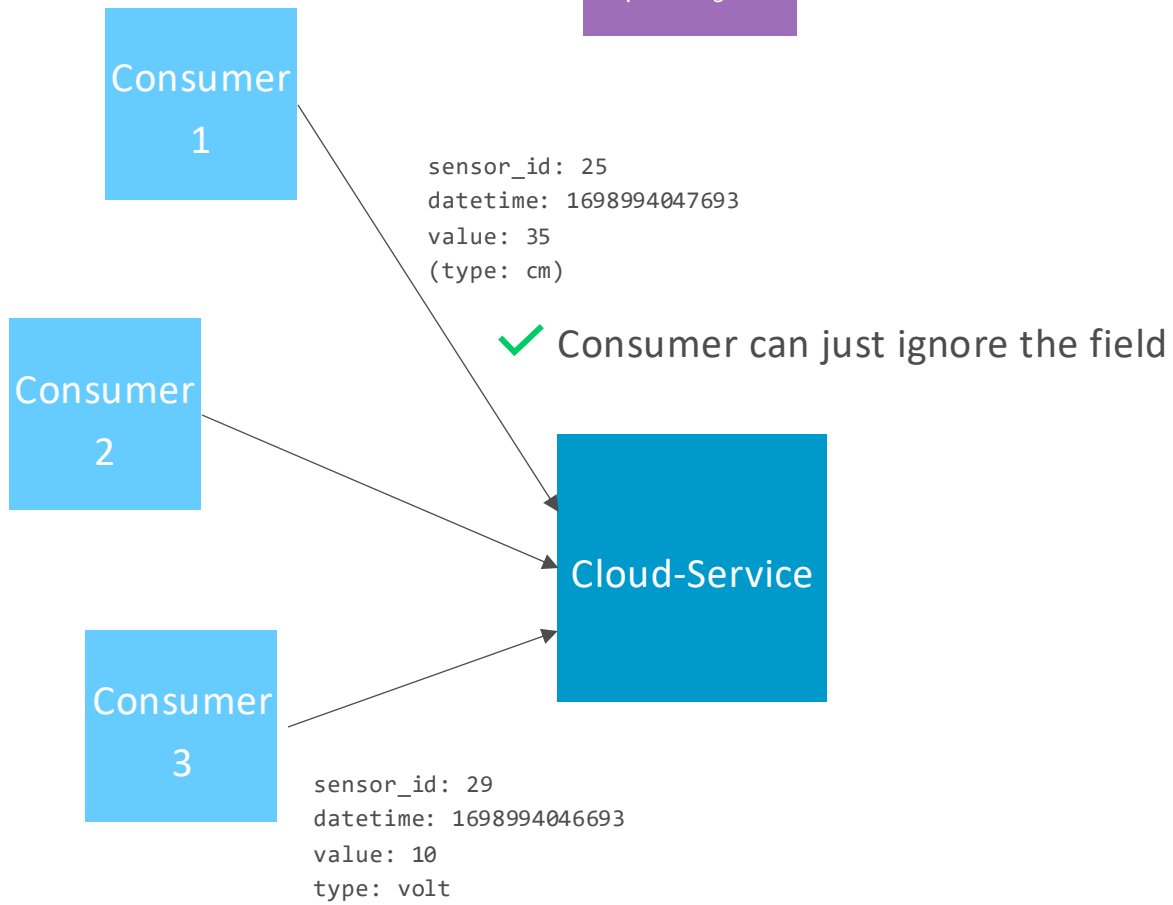
Contract in the synchronous world

```
…
  "fields" : [
    {
      "name" : "sensor_id",
      "type" : "string",
      "doc"  : "ID which uniquely identifies the sensor across the plant"
    },
    {
      "name" : "datetime",
      "type" : "long",
      "doc"  : "UTC timestamp of the sensor measurement"
    },
    {
      "name" : "value",
      "type" :  "long",
      "doc"  : "The value of the sensor measurement"
    }
  ]
…
```



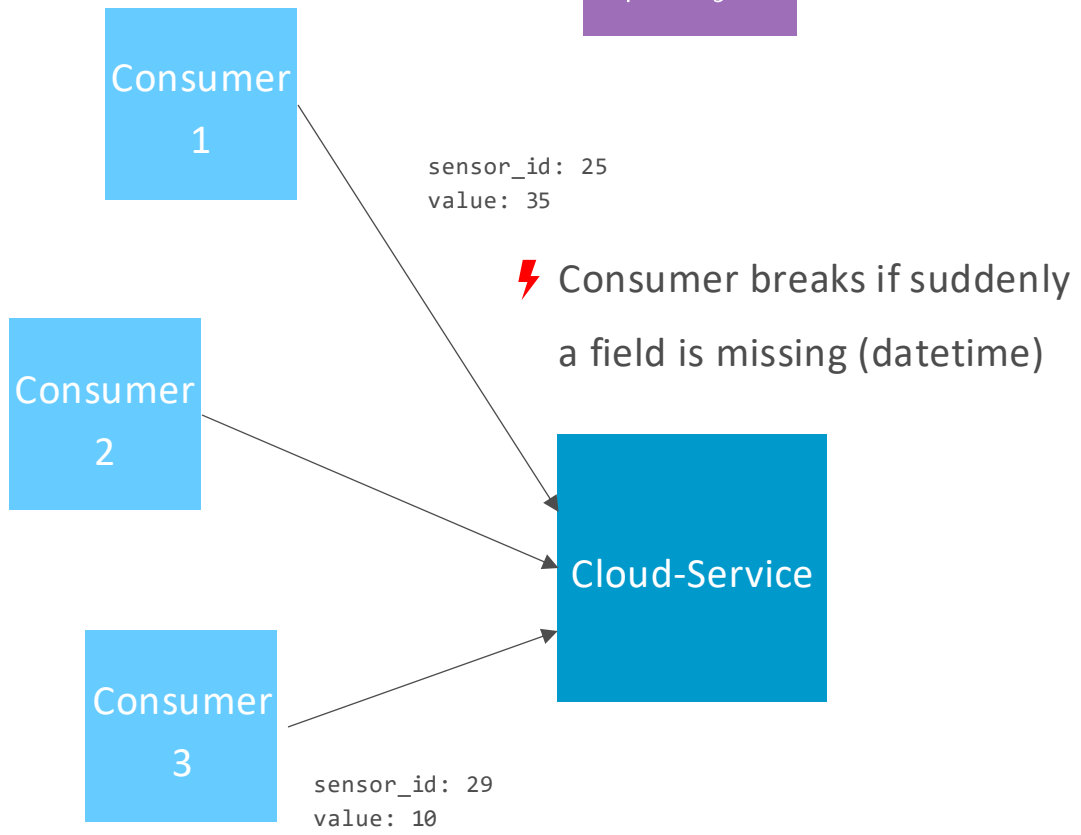
Change the service contract: add a field

```
...
  "fields" : [
    {
      "name" : "sensor_id",
      "type" : "string",
      "doc"  : "ID which uniquely identifies the sensor across the plant"
    },
    {
      "name" : "datetime",
      "type" : "long",
      "doc"  : "UTC timestamp of the sensor measurement"
    },
    {
      "name" : "value",
      "type" :  "long",
      "doc"  : "The value of the sensor measurement"
    },
    {
      "name" : "type",
      "type" : "string",
      "doc"  : "The type of the measurement"
    }
  ]
...
```



Change the service contract: remove a field

```
…
  "fields" : [
    {
      "name" : "sensor_id",
      "type" : "string",
      "doc"  : "ID which uniquely identifies the sensor across the plant"
    },
    {
      "name" : "value",
      "type" :  "long",
      "doc"  : "The value of the sensor measurement"
    }
  ]
…
```



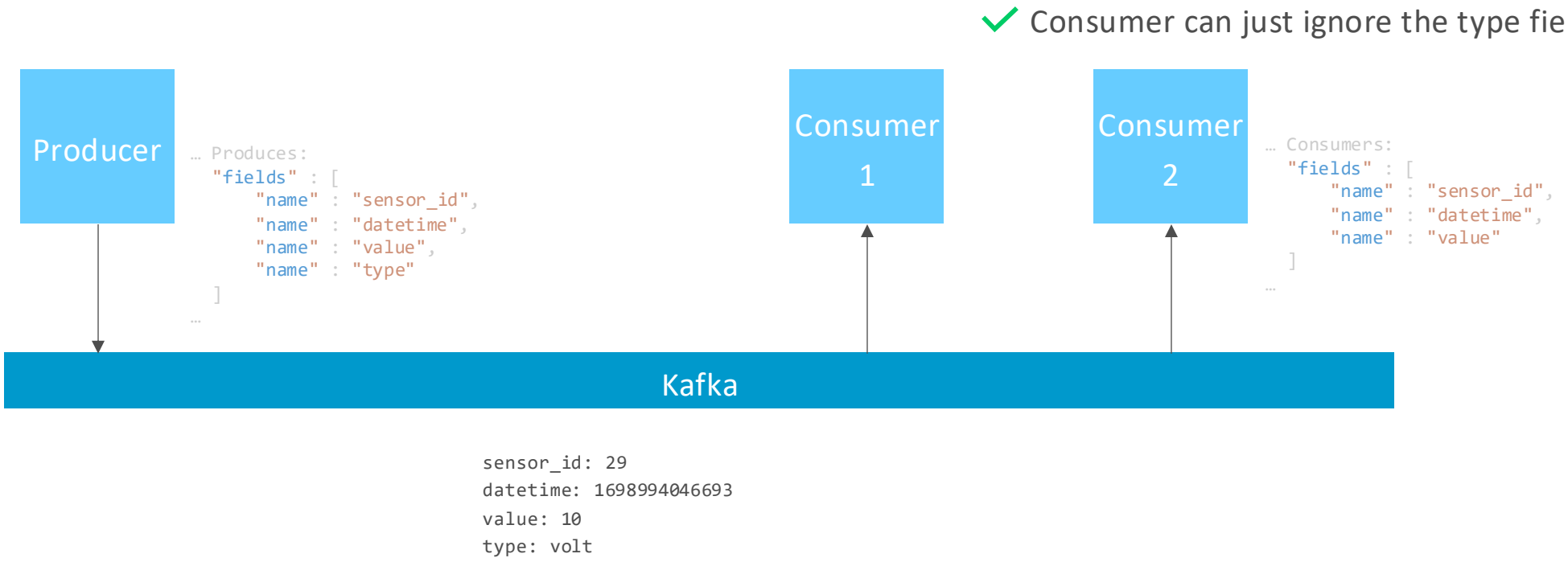
Consumer breaks if suddenly a field is missing (datetime)

Solution: Consumer defines a default

Consequence: Update all consumers first

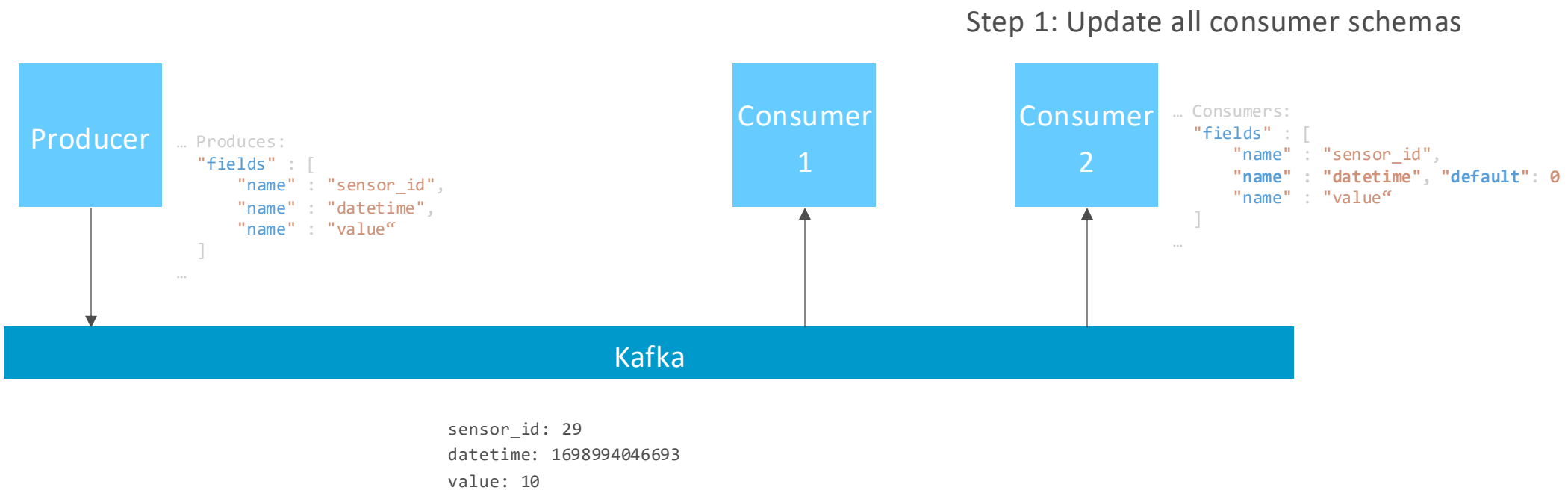
Your consumers gets the Kafka Consumers

Add fields



Your consumers gets the Kafka Consumers

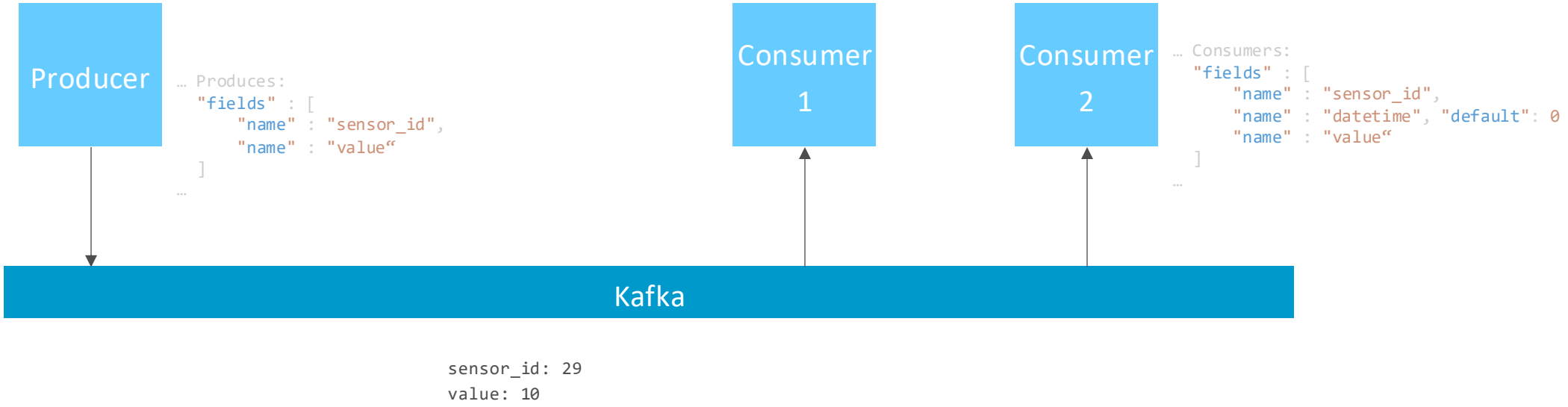
Remove fields



Your consumers gets the Kafka Consumers

Remove fields

Step 2: Producer can remove the corresponding field



Lab IoT Case Part 1

Lab

Explore Schema Registry

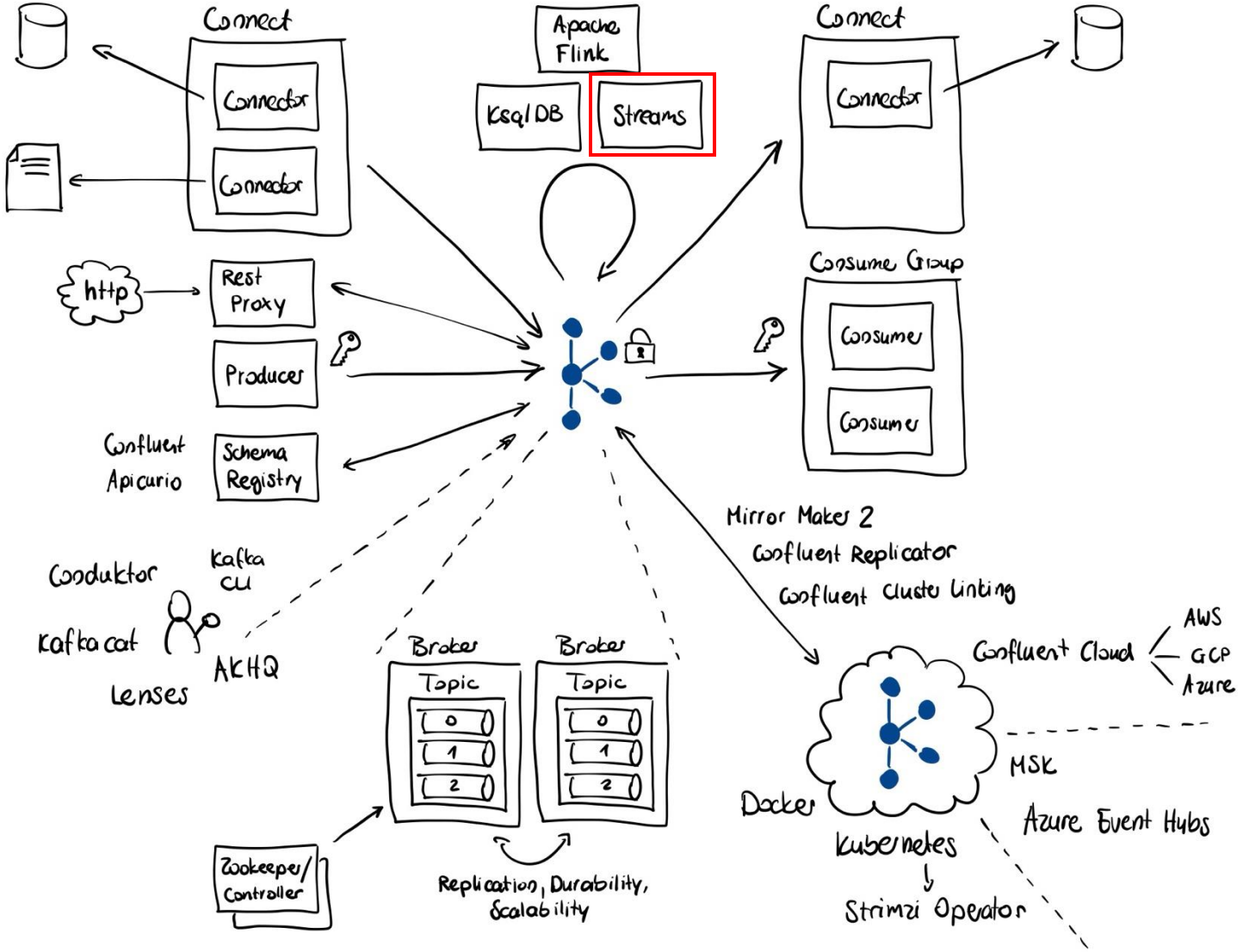
Follow Exercises from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/schema-registry.md

- Produce Messages with Avro Schema
- Consume Messages with Avro Schema
- Perform a schema evolution and learn about its limitations

Kafka Streams

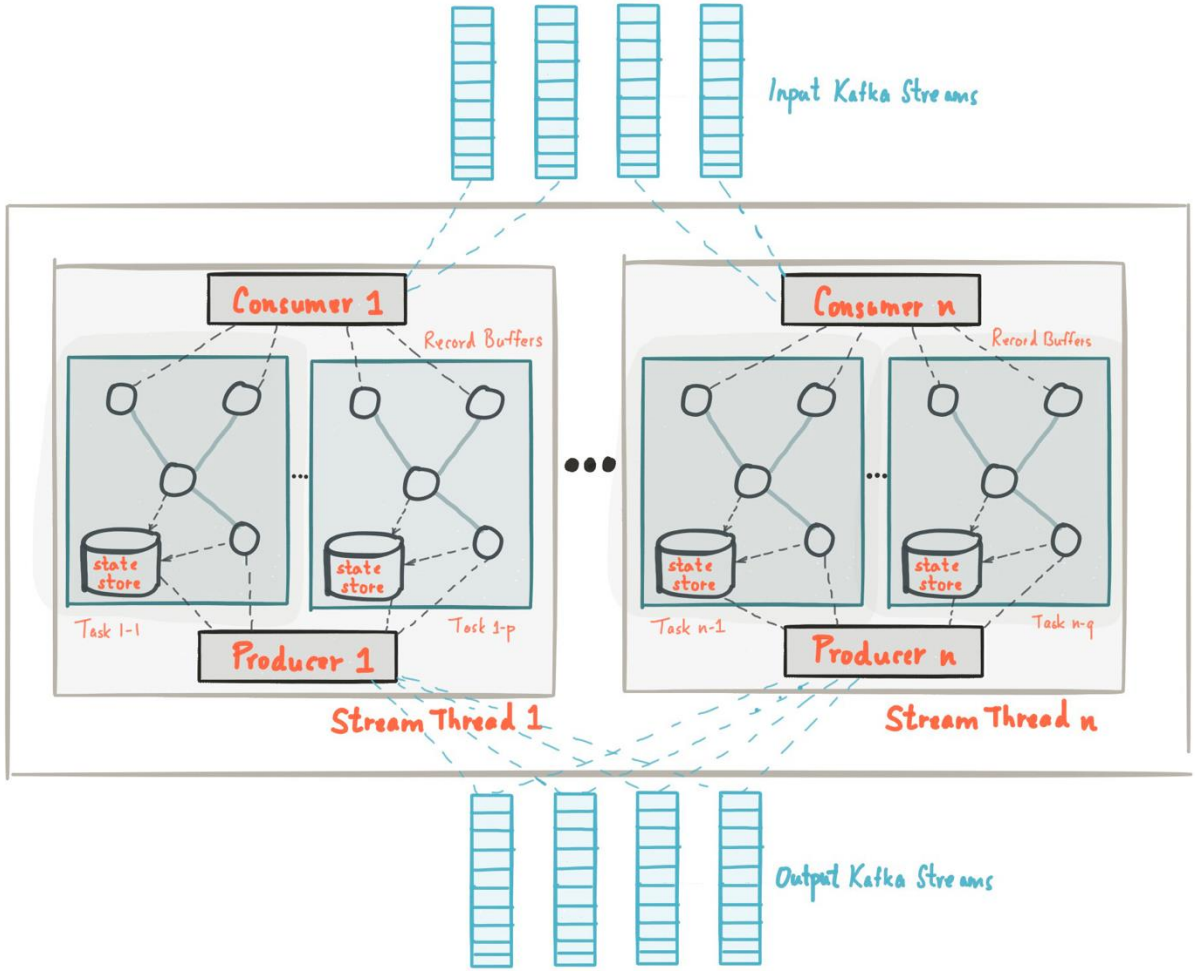
Overview


Connect
Connector
Connector
Apache Flink
KsqlDB
Streams
Connect
Connector
http
Rest Proxy
Producer
Confluent
Apicurio
Schema Registry
Consume Group
Consumer
Consumer
Mirror Maker 2
Confluent Replicator
Confluent cluster linting
Conduktor
kafka cli
kafkacat
AKHQ
Lenses
Broker
Topic
0
1
2
Broker
Topic
0
1
2
Zookeeper/ Controller
Replication, Durability, Scalability
Confluent Cloud
AWS
GCP
Azure
MSK
Azure Event Hubs
Docker
Kubernetes
Strimzi Operator

Stream Processing with Kafka Streams

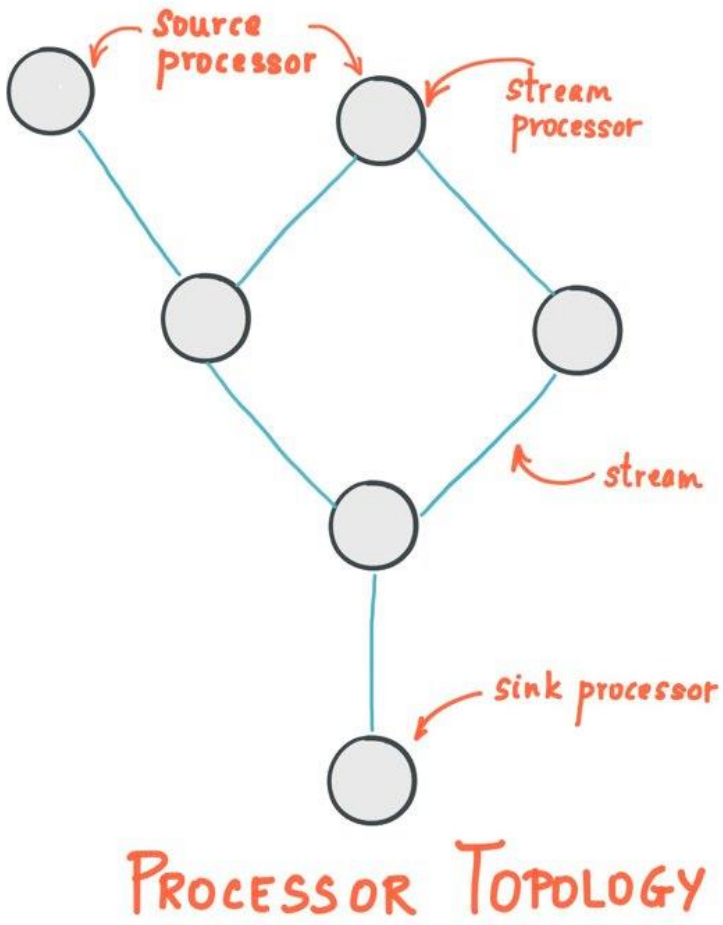
Kafka Streams is a

(Java / Scala) client library for building applications and microservices, where the input and output data are stored in an Apache Kafka cluster.



Source: https://kafka.apache.org/32/documentation/streams/architecture

Kafka Streams Topology




```
dataStream
   .transform(…)
   .filter(…)
   .mapValues(…)
   .to(…)
```

Source: https://kafka.apache.org/32/documentation/streams/core-concepts

Kafka Streams Types

KStream

- Record Stream (INSERT)

Key	Value
Sensor1	30 °
Sensor2	50 cm
Sensor3	2000 rpm
Sensor1	29 °
Sensor3	1500 rpm

KTable

- Changelog Stream (UPSERT)
- Log Compaction
- Special Case: GlobalKTable

Key	Value
Sensor2	50 cm
Sensor1	29 °
Sensor3	1500 rpm

Kafka Streams Operations

Streams DSL vs Processor API

Stateless Operations

- Branch / Merge
- Filter / FilterNot
- Map / MapValues / FlatMap / FlatMapValues
- Foreach
- GroupBy / GroupByKey / Cogroup
- Peek / Print
- SelectKey
- ToStream / ToTable

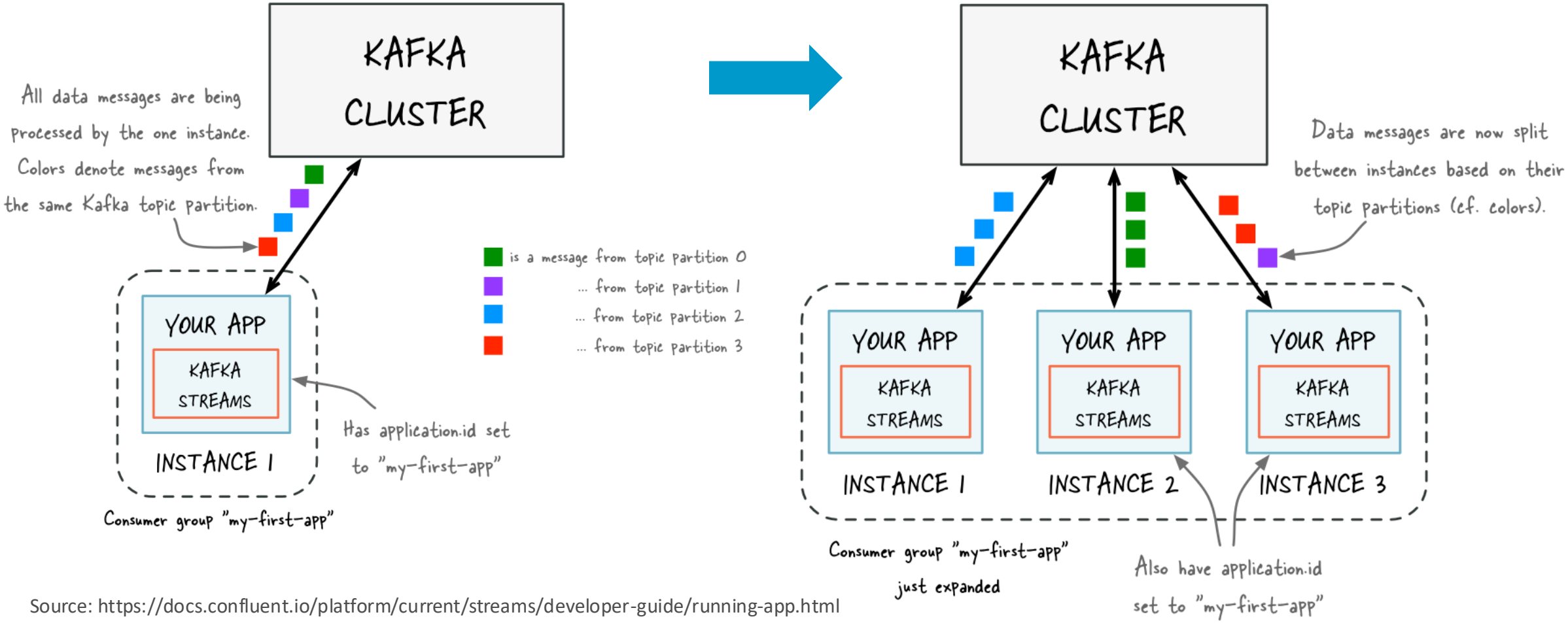
Stateful Operations

- Aggregating (count, reduce, …)
- Joining
- Windowing
- Custom Transformations ★

A example use case for split and branch : https://developer.confluent.io/tutorials/split-a-stream-of-events-into-substreams/confluent.html

Kafka Streams Scalability

Using Partitions and Consumer Groups



Source: https://docs.confluent.io/platform/current/streams/developer-guide/running-app.html

Lab IoT Case Part 2

Lab

Implement Stream Processing

Follow Exercises from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/iot.md

- Run a Kafka Streams application
- Write your own Kafka Streams application with stateless and stateful Operations

Day 2

Kafka Training

Agenda Day 2

09:00 – 09:30 Recap Day 1

09:30 – 10:30 Kafka IoT Case

10:30 – 10:50 Break

10:50 – 12:00 Delivery Guarantees, Architecture Concepts

12:00 – 13:00 Lunch Break

13:00 – 15:00 Kafka Connect

15:00 – 15:20 Break

15:20 – 16:20 Kafka Witness Protection Case

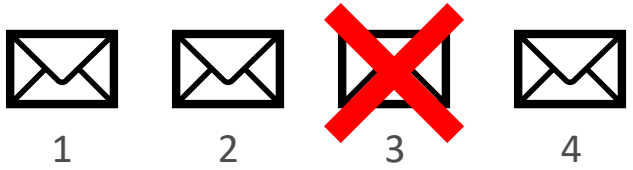
16:20 – 17:00 Recap and Feedback

17:00 … Open work

Delivery Guarantees

Delivery Guarantees

- At most once
  - Messages may be lost but are never redelivered



- At least once
  - Messages are never lost but may be redelivered

- Exactly once
  - Each messages is delivered once and only once

Delivery Guarantees

At least once

- Best option for most use cases
  - Ensure transformations are idempotent

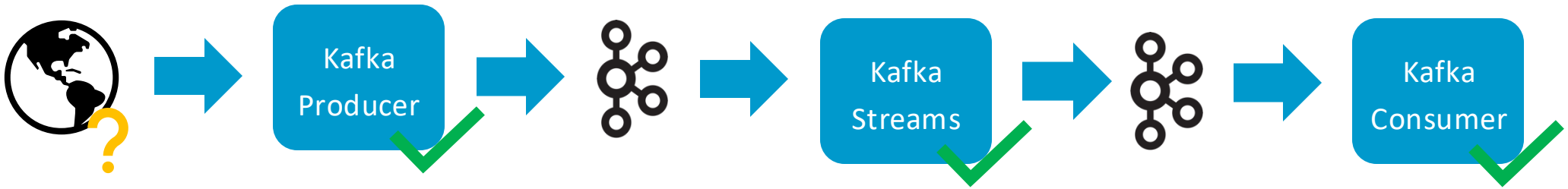
- Configure the kafka cluster to accept a message only if more than one broker has written it to the disk
  - Broker config: **min.insync.replicas = 2**
  - Producer config: **acks = all**

- Configure the consumer to commit the message only after the processing
  - Use manual commit (commitSync / commitAsync)
  - Commit after the message has been sucessfully processed

Delivery Guarantees

Exactly Once

- End-to-End Guarantee over Kafka components
  - **Kafka Transactions**



- If one component does not guarantee “exactly-once” anywhere in the data pipeline, the whole pipeline will no longer be “exactly-once”
  - This includes source and sink systems (MQTT, …)

Lab Delivery Guarantees

Lab

Configure Delivery Guarantees

Follow Exercises from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/delivery-guarantees.md
- Update the Kafka Streams application to guarantee an "exactly once" processing of messages

Architecture Concepts

Event Sourcing & Event Streaming

Event sourcing and event streaming are **both architectures** used to build **real-time data processing systems**, but they have different characteristics and use cases.

Event Sourcing:

- Stores **every change to the application state** as a sequence of events
- Used for historical analysis and to maintain an audit trail of changes

Event Streaming:

- Involves **continuously processing and analyzing streams** of events in real-time
- Useful for building real-time analytics, monitoring, and alerting systems

They can be used together in some cases. Event sourcing can be used to maintain the current state of an object (e.g. a shopping card), while event streaming can be used to analyze (real-time) the user's behavior during the shopping activity.

Event Streaming

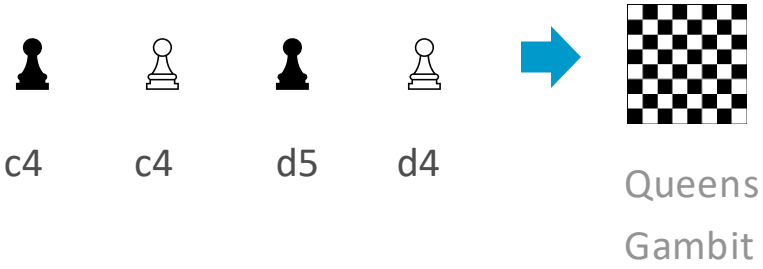
- Unbounded / Infinite stream of events
- Any type of events
- Object in scope might be mutable

Example: time series of a temperature

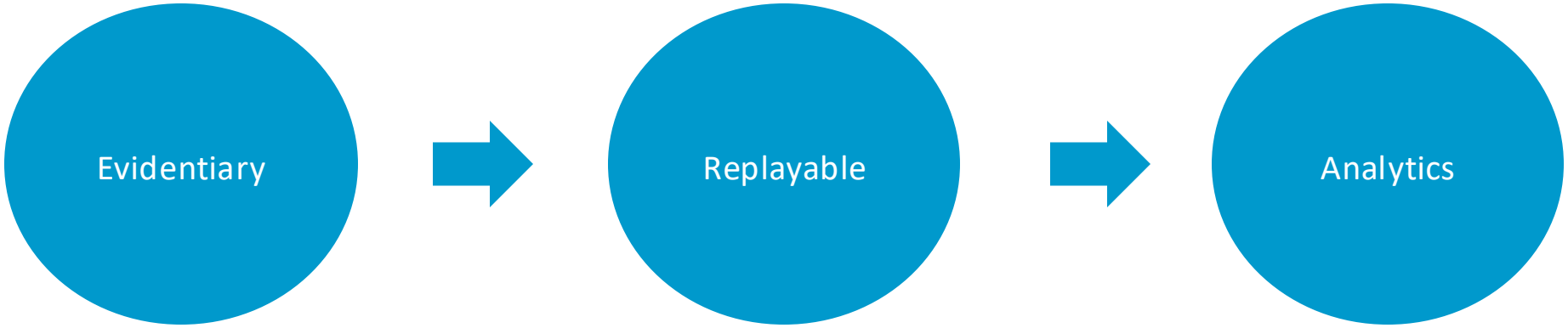
Event Sourcing

- Every state change is represented in an event
- Timeline of activity
- Object in scope is immutable
- Source of truth

Example: Banking Account



Event Sourcing



- Immutable event log
- go back in time

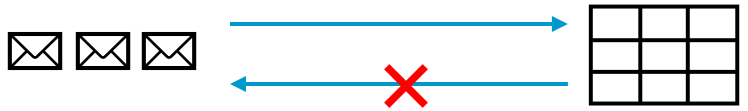
- Replay computation
- fix, rewind, replay

- Analyse journey of user
- apply ML

Source: https://developer.confluent.io/learn-kafka/event-sourcing/why-store-events/

Event Sourcing

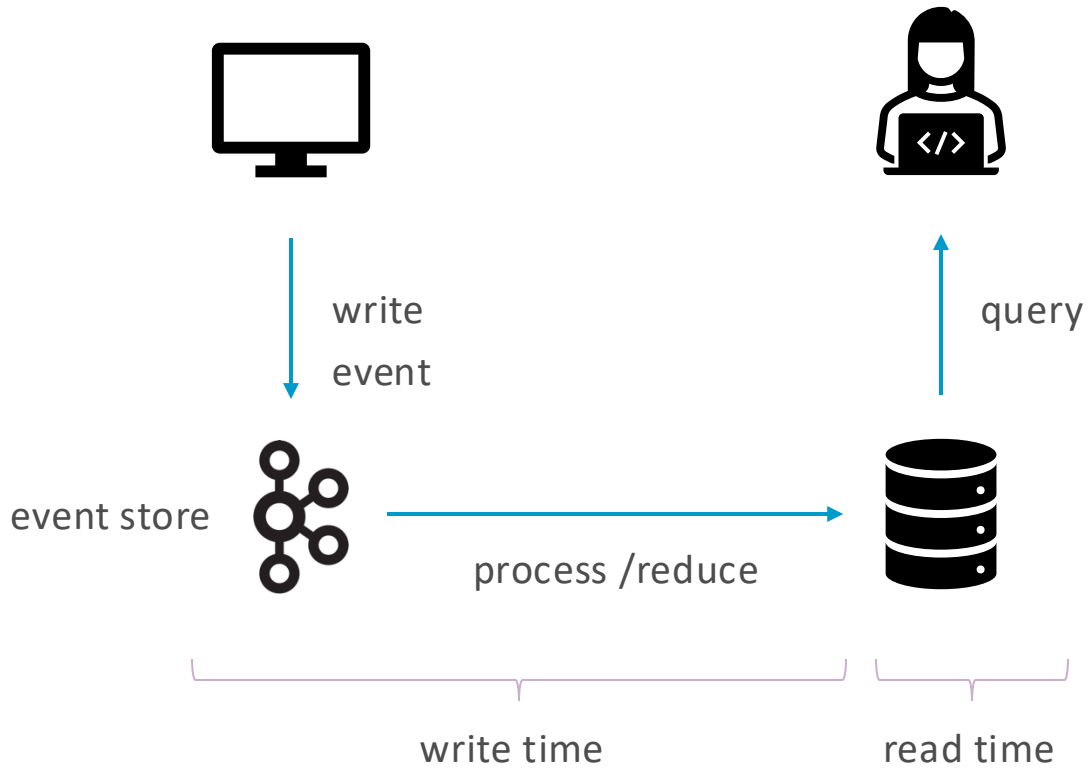
- How do we get from events to the current state?
  - Read all Events
  - Perform a chronological reduce
  - Only one way possible
  - Schema Evolution is important



- What if we have a lot of events? Wouldn't this take a long time every time we read the data?

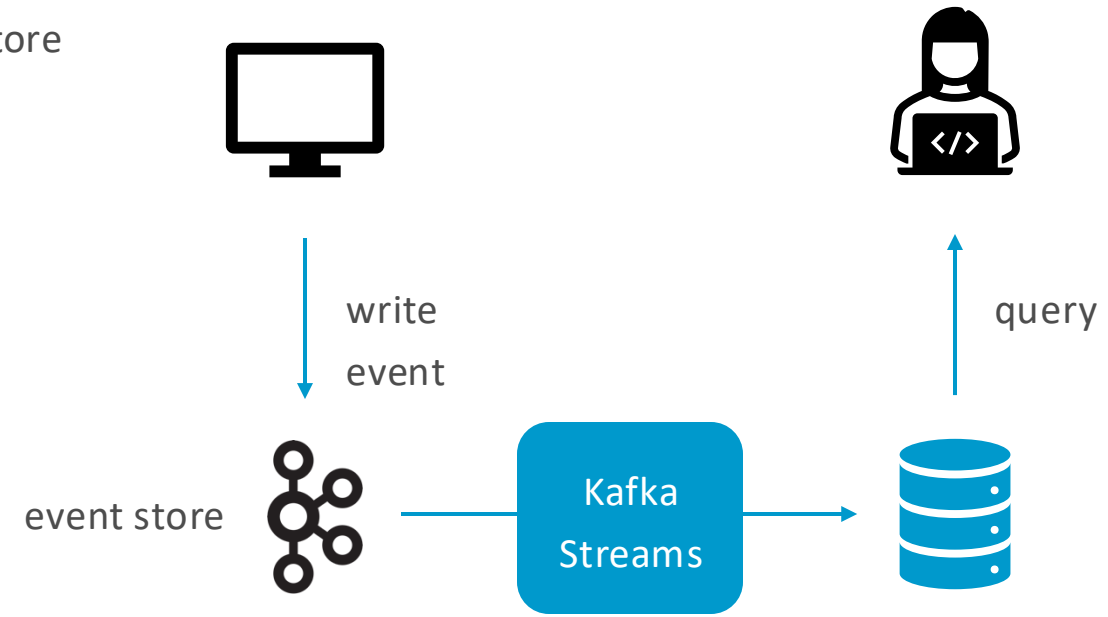
CQRS

- **C**ommand **Q**uery **R**esponsibility **S**egregation
  - Separate Reads from Writes
  - Compute data (only once) when it is written
  - Eventually consistent
    - No read-your-writes guarantee



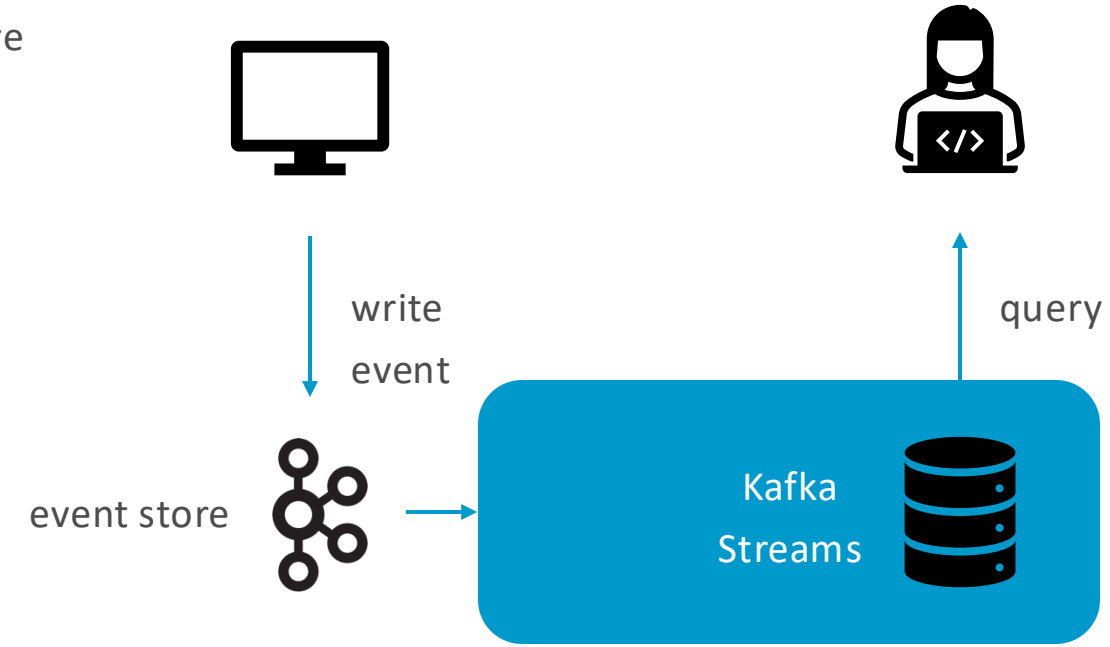
CQRS

- Process data using kafka streams
  - Application state is modelled as external datastore



CQRS

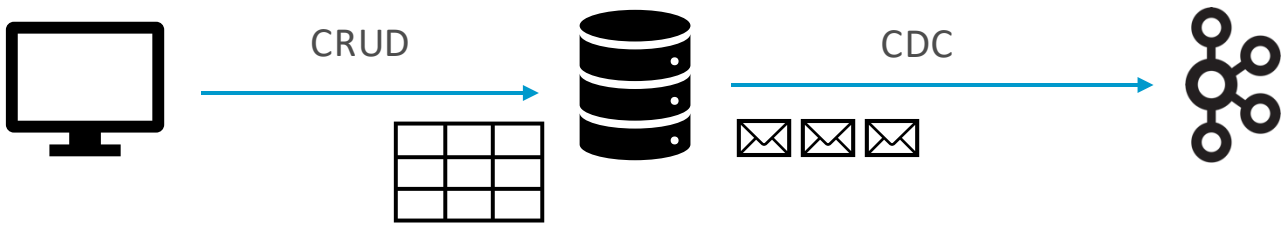
- Process data using kafka streams
  - Application state is modelled as local state store
  - Kafka Stream Interactive Queries
  - RocksDB



Source: https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

Change Data Capturing (CDC)

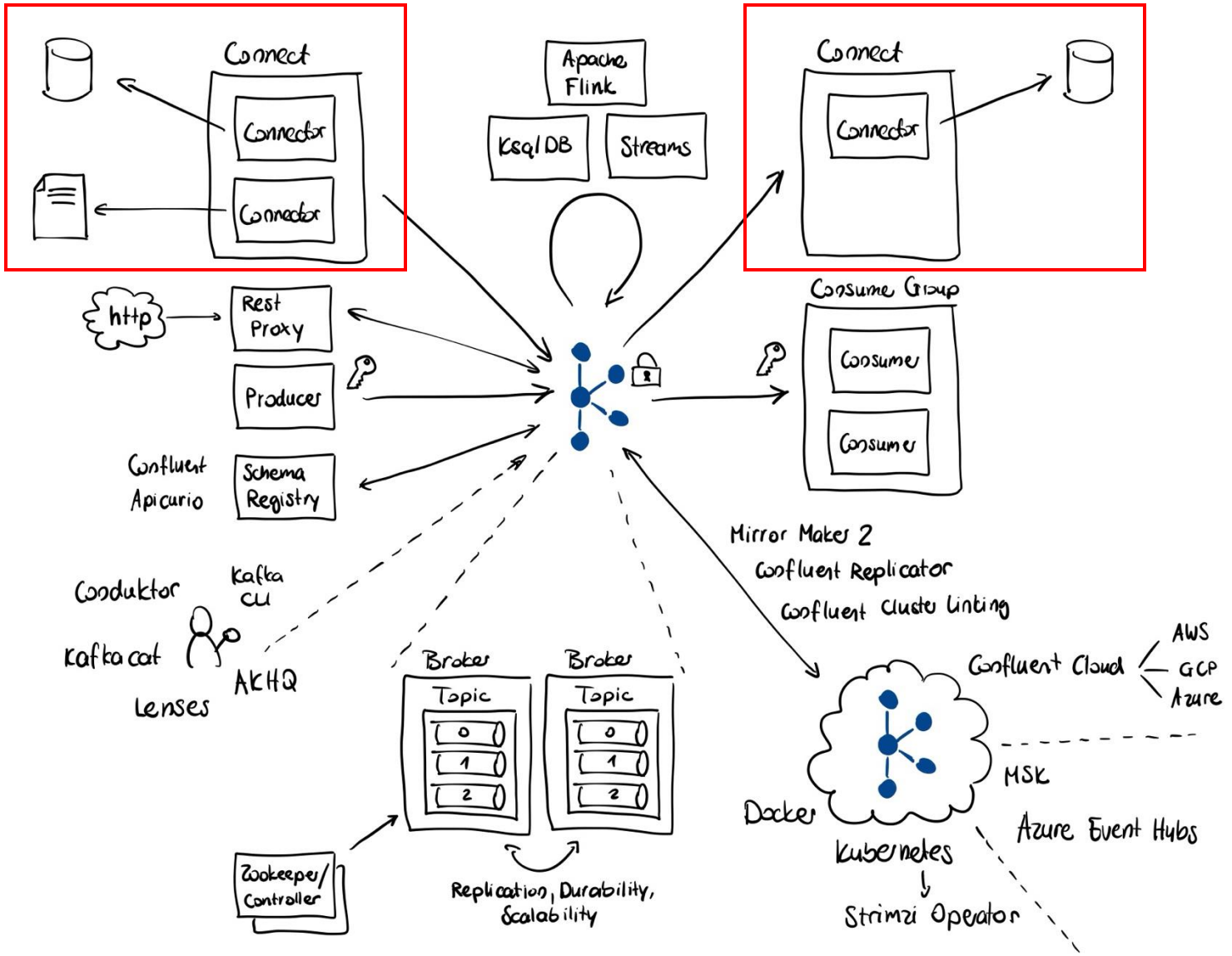
- Capture changes made to data in a DB
- Send Event for every Change



Kafka Connect & Compaction in action

Kafka Connect

Overview


Connect
Connector
Connector
Apache Flink
KsqlDB
Streams
Connect
Connector
http
Rest Proxy
Producer
Confluent
Apicurio
Schema Registry
Consume Group
Consumer
Consumer
Mirror Maker 2
Confluent Replicator
Confluent cluster linking
Conduktor
kafka cli
kafkacat
AKHQ
Lenses
Broker
Topic
0
1
2
Broker
Topic
0
1
2
Zookeeper/ Controller
Replication, Durability, Scalability
Confluent Cloud
AWS
GCP
Azure
MSK
Azure Event Hubs
Docker
Kubernetes
Strimzi Operator

Kafka Connect

- Framework for getting Data **in** and **out** of Kafka.
  - Source Connector => **in** Kafka
  - Sink Connector Kafka => **out** of Kafka

- Standardizes integration of other data systems with Kafka
  - simplifying connector development, deployment, and management
  - separated from the Kafka Brokers, but uses many Kafka concepts

- Introduced in **2015** with Kafka version 0.9x

Kafka Connect

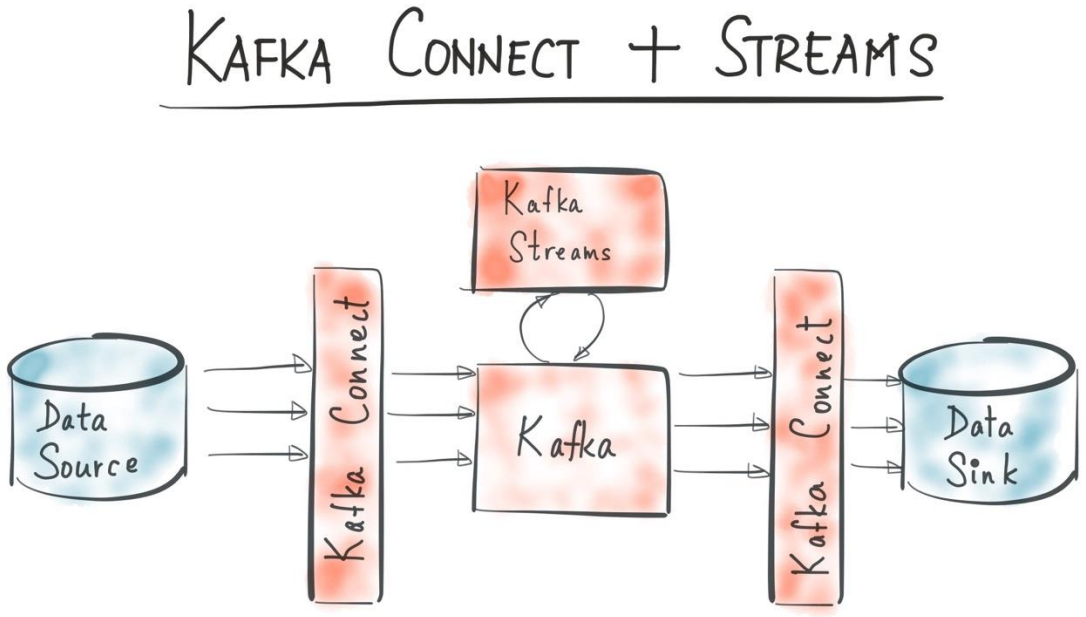
High Level Architecture

Source

Getting data

into Kafka

(pull)



Sink

Getting data

out of Kafka

(push)

Source: https://www.confluent.io/blog/hello-world-kafka-connect-kafka-streams/

Kafka Connect

Connectors - a re-usable piece of code (jar-Files)

- Connectors include...
  - **FileStream** Connectors (for Development and Testing)
  - ActiveMQ Source Connector
  - Amazon S3 Sink Connector
  - Elasticsearch Sink Connector
  - HDFS 2 Sink Connector
  - IBM MQ Source Connector
  - **JDBC Connector** Source and Sink Connector
  - JMS Source Connector

- **Further**: **MongoDB**, **Splunk**, DynamoDB, **Twitter, Prometheus** etc.

Kafka Connect

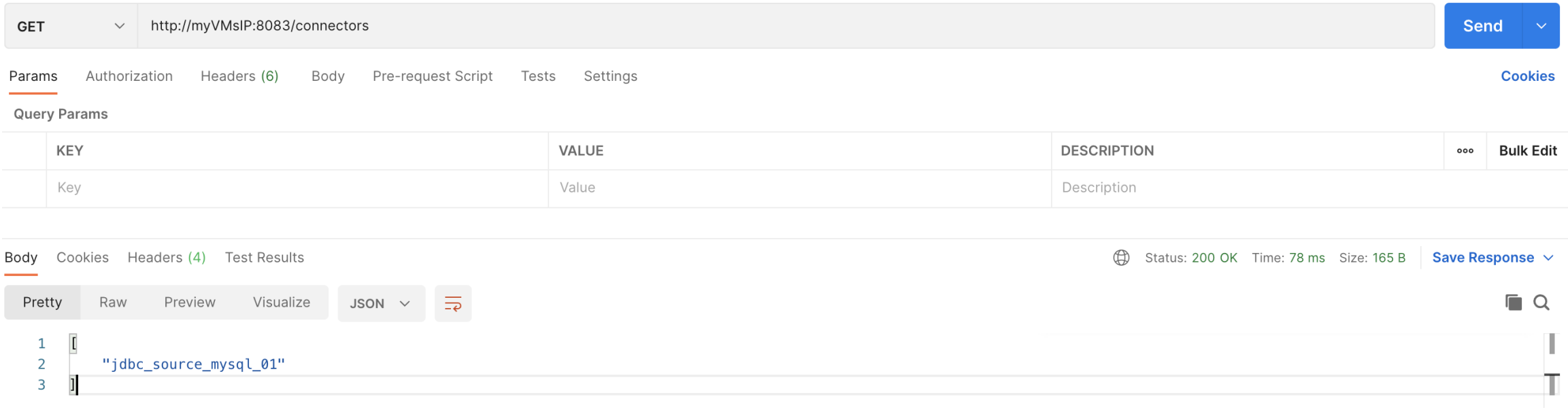
Connectors - a re-usable piece of code (jar-Files)

- Not all connectors are free of charge.
  - Some systems offer their own implementation of a Connector (e.g. Oracle)
- Confluent managed Connectors can be found here: https://www.confluent.io/hub/
- You can build your own connector: https://docs.confluent.io/platform/current/connect/devguide.html

Kafka Connect

REST Interface

- REST API for managing connectors
- Run as a **service** by default on port **8083**
- Documentation: https://docs.confluent.io/platform/current/connect/references/restapi.html#kconnect-rest-interface



Lab Kafka Connect

Lab

Exercise: Kafka Connect

Follow the Kafka Connect Exercise from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/kafka-connect.md

Kafka Connect: Architecture

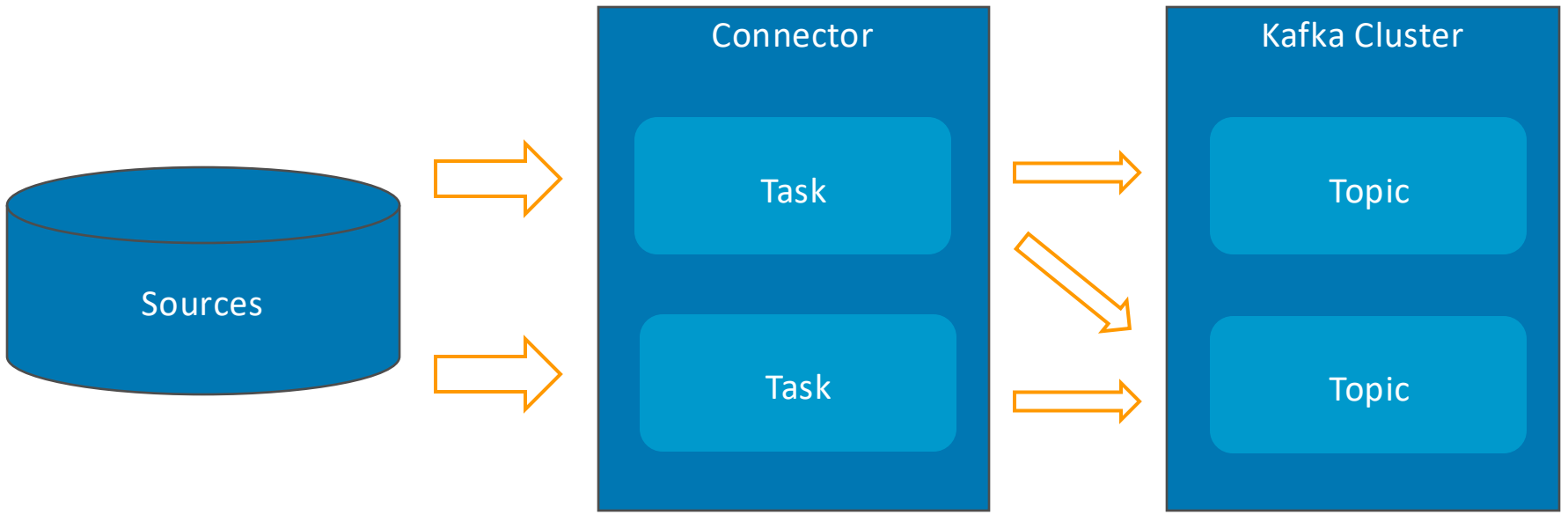
Kafka Connect

Concepts

- Connectors
  - Instance of a Connector plugin, that coordinates Tasks
- Tasks
  - Copy the data between Kafka and another System
  - State is stored in Kafka
- Workers
  - Processes that execute the Tasks
  - Can run in Standalone or Distributed mode
- Converters
  - Convert the data to a supported format (e.g. Avro, Protobuf, String)
- Transforms
  - Simple, lightweight transformations for single messages

Kafka Connect

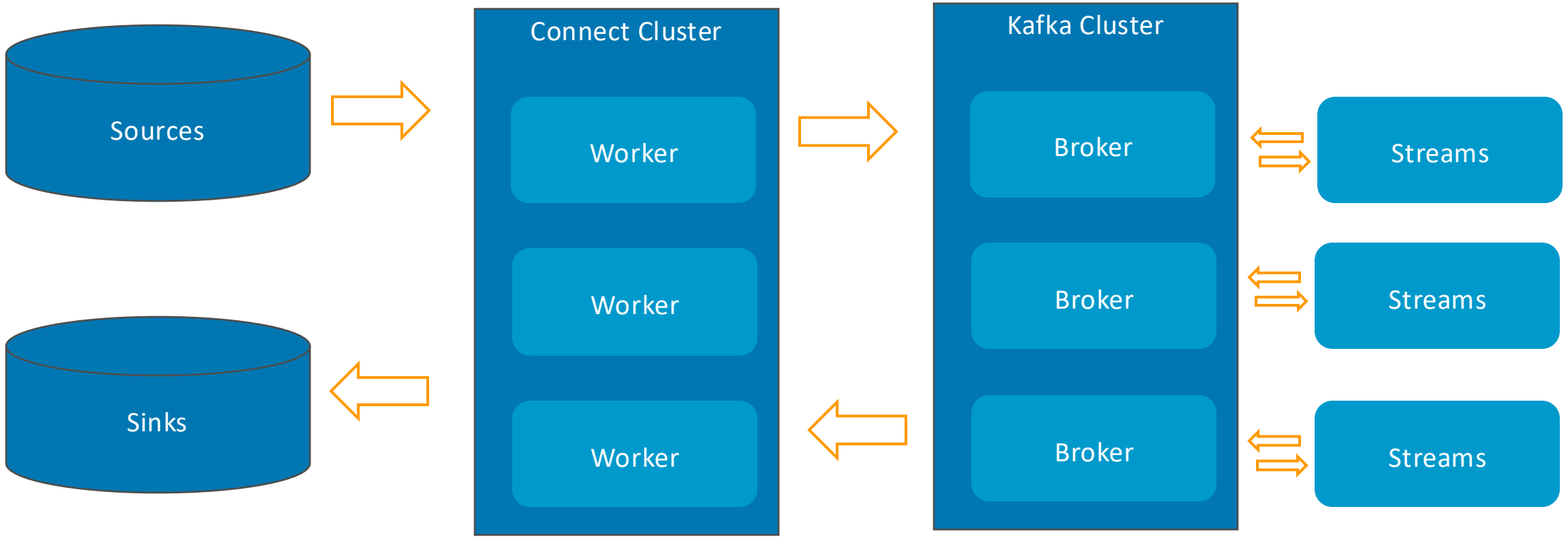
Tasks



Task: Connector (jar-File) + User Configuration; is linked to a connector configuration

Kafka Connect

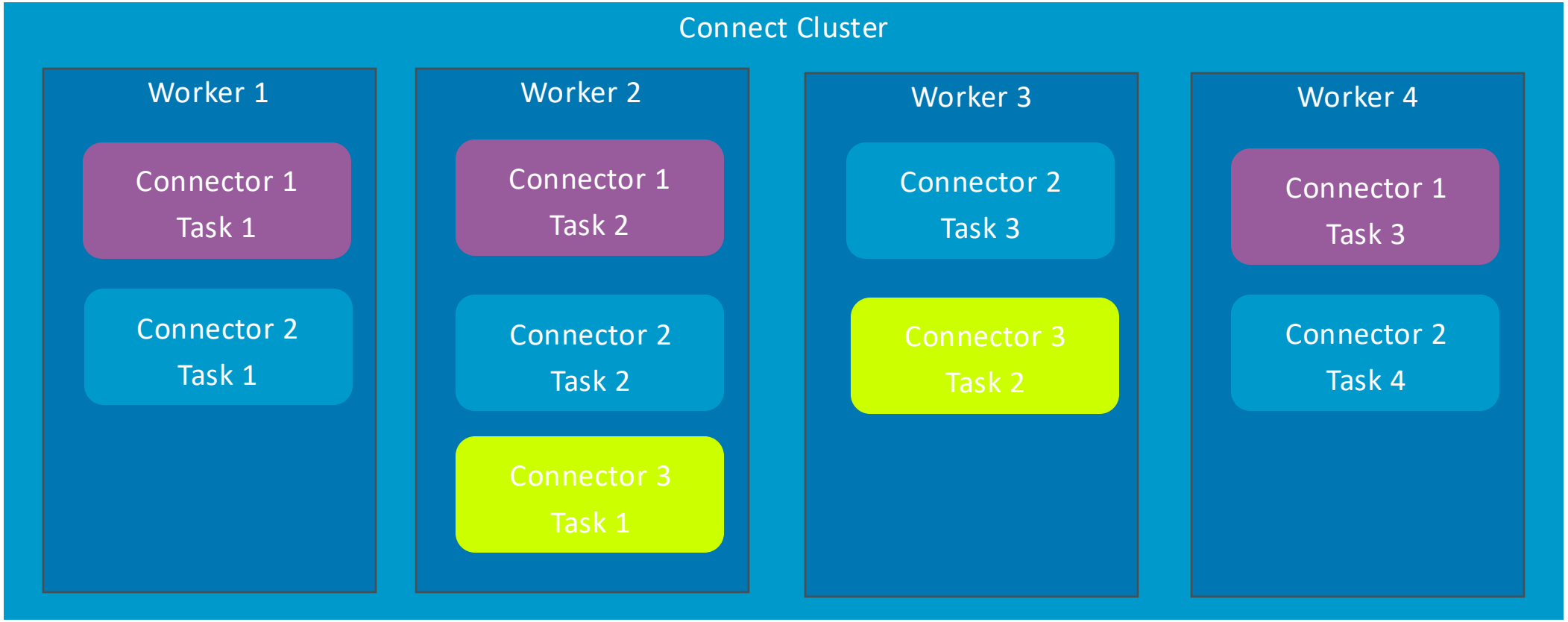
Workers



Worker: Executes Tasks, is a single Java Process and can be **standalone** or **distributed**

Kafka Connect

Connect Cluster

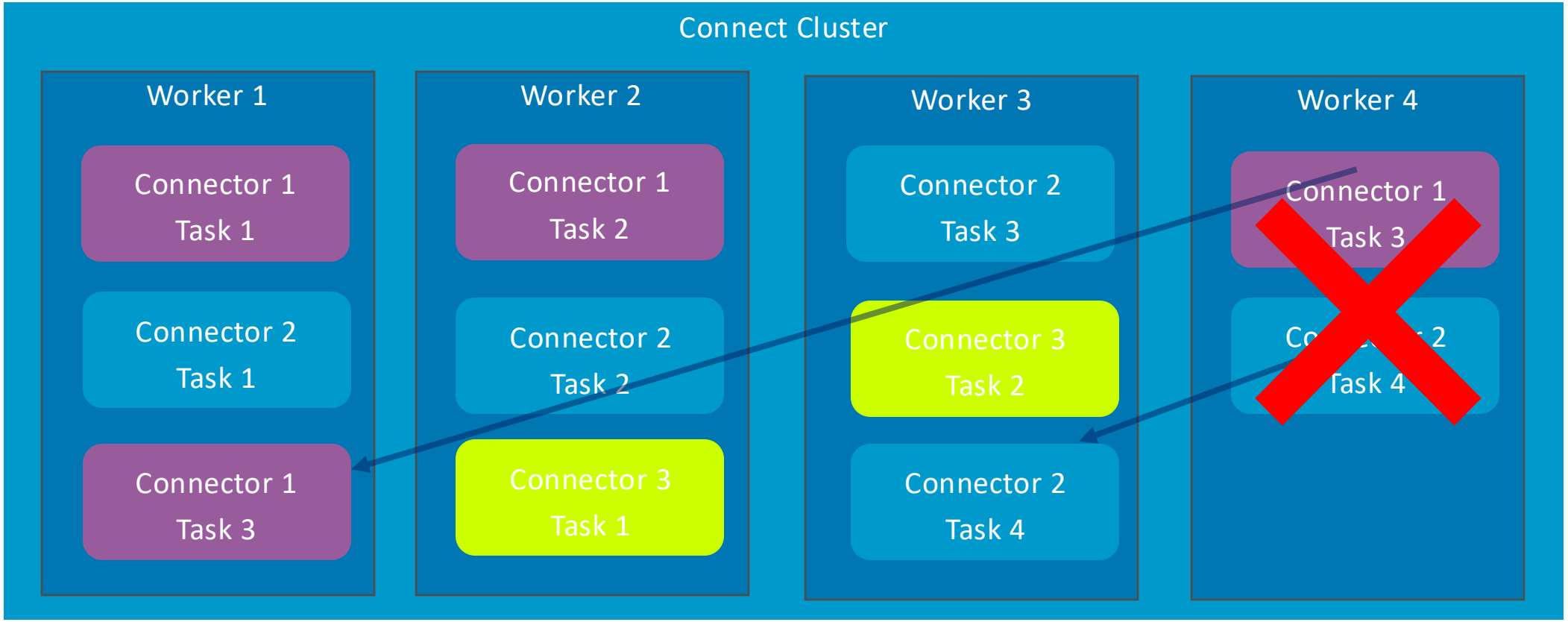


Standalone Mode (development & testing): Single process, not fault tolerant and scalable

Distributed (production): Multiple workers; scalable/fault tolerant

Kafka Connect

Connect Cluster



Fault Tolerance: If a server goes down everything will be **rebalanced**.

Kafka Connect

Transforms

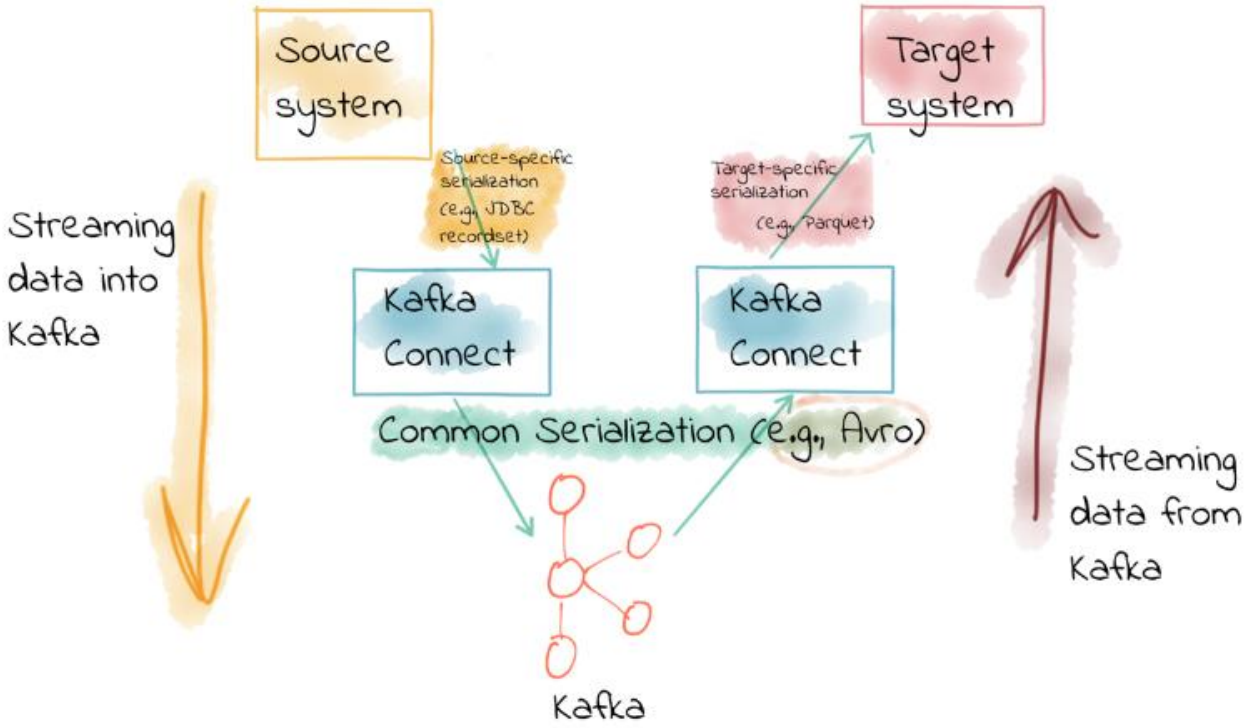
- Connectors can be configured with transformations to make modifications to messages.
  - Transformations are only possible on a **single** message (**Single Message Transformation** SMT).
  - Is stateless; for stateful transformations, use Kafka Streams.

- Some Transformations
  - *Cast*: Cast a key or value to a specific type
  - *Drop*: Remove the content of a key or value
  - *ExtractField*: Extract a single field from the value
  - *ValueToKey*: Replace the key with the value

- All possible transformations: https://docs.confluent.io/platform/current/connect/concepts.html#transforms
- Write your own: https://docs.confluent.io/platform/current/connect/transforms/custom.html#custom-transformations

Kafka Connect

Converter



https://www.confluent.io/blog/kafka-connect-deep-dive-converters-serialization-explained/

Serialization formats:

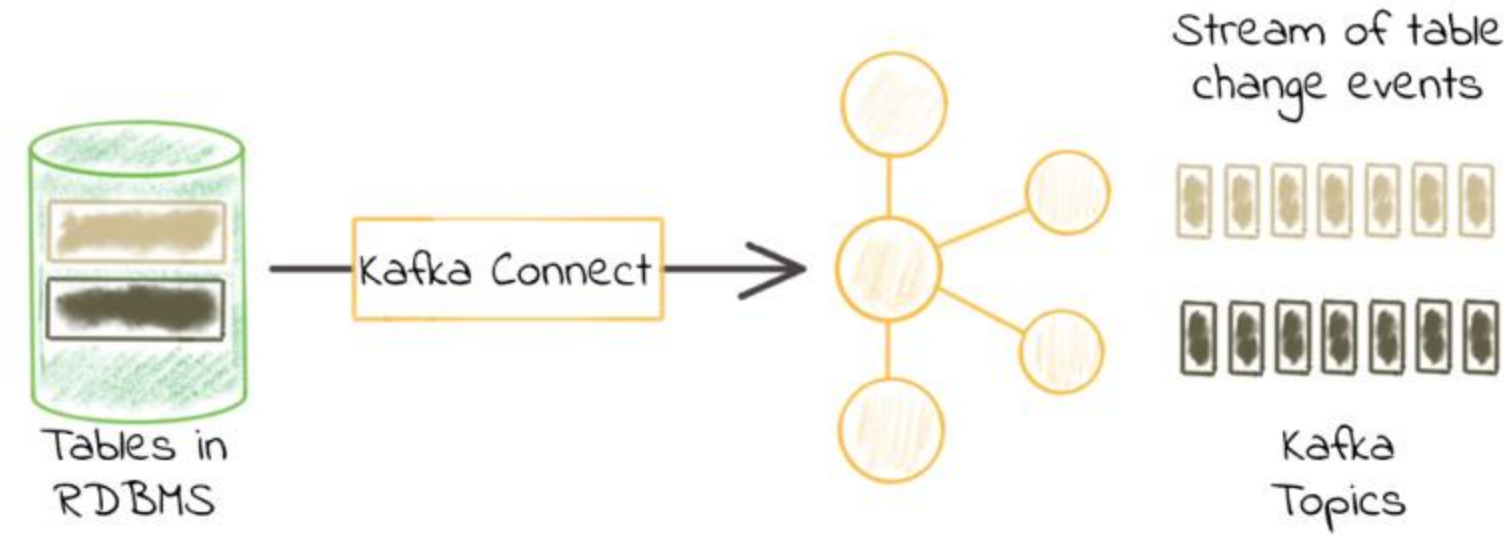
- JSON
- Avro
- Protobuf
- String delimited (e.g., CSV)

Converter need to be specified for both keys (*key.converter*) and value (*value.converter*).

Kafka Connect

JDBC connector for Kafka Connect

For many relational databases **JDBC driver** are available like **Microsoft SQL, MySQL or Postgres**. Feature **vary** between implementations and licenses – e.g., some driver do not support delete records for the database.



Source: https://www.confluent.io/blog/kafka-connect-deep-dive-jdbc-source-connector/

Kafka Connect

Further concepts

- Some Connectors can do **bulk import** and **incremental (ID or timestamp)** ingest
- Some Connectors support **CDC**

Compaction in Action

Compaction

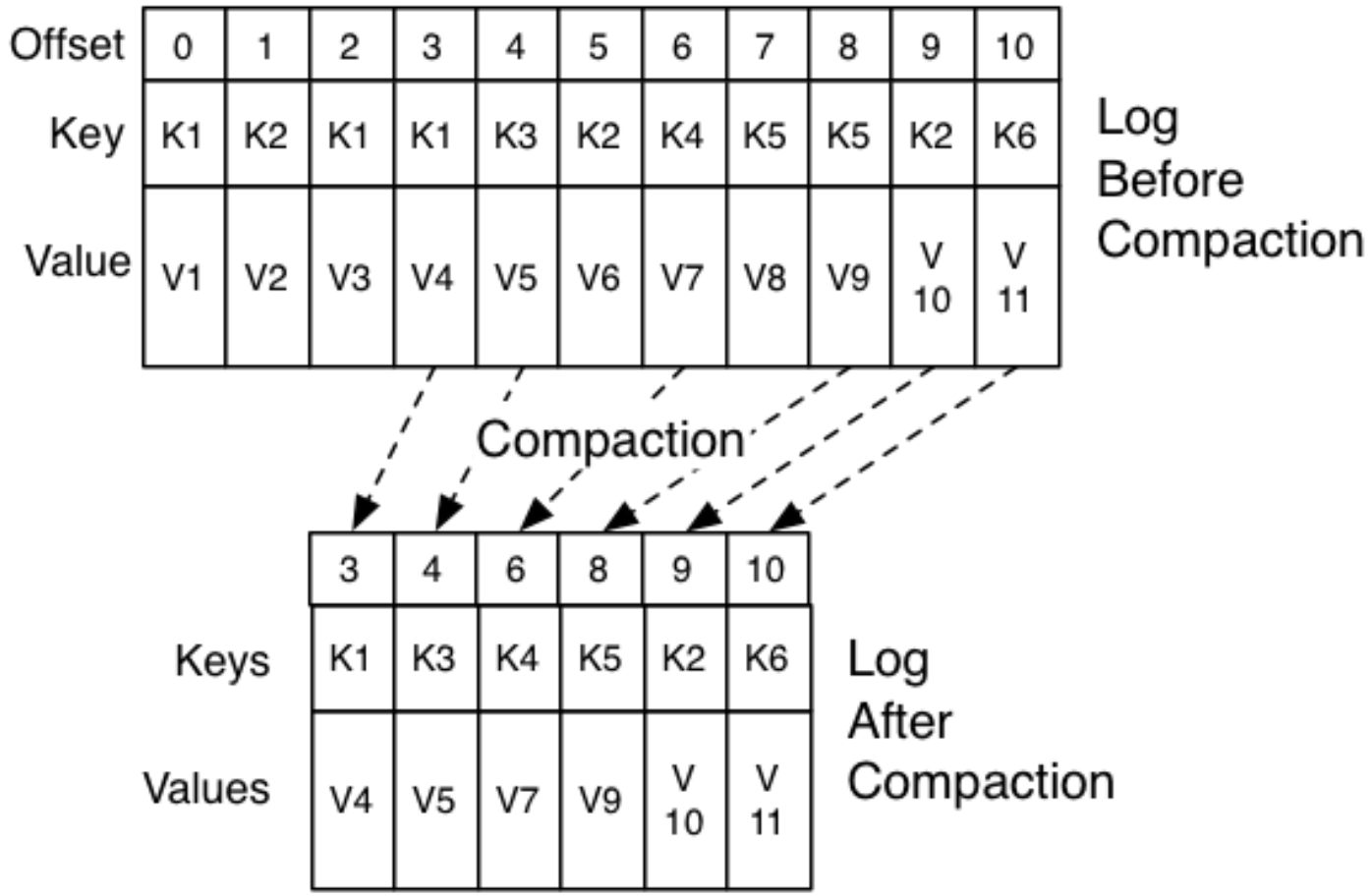
Recap Day1: Compaction Configuration

- **cleanup.policy:** A string that is either "**delete**" or "**compact**" or both. This string designates the retention policy to use on old log segments. The **default policy ("delete")** will **discard old segments** when their **retention time** or **size limit** has been reached. The "compact" setting will enable log compaction on the topic.
- **max.compaction.lag.ms: The maximum time** a message will remain ineligible for compaction in the log. Only applicable for logs that are being compacted.
- **min.compaction.lag.ms:** The **minimum** time a message will remain uncompacted in the log. Only applicable for logs that are being compacted.
- **min.cleanable.dirty.ratio:** This configuration controls **how frequently the log compactor will attempt to clean the log** (assuming log compaction is enabled). By default, we will avoid cleaning a log where more than 50% of the log has been compacted.
- **segment.ms:** This configuration controls the period of time after which Kafka will force the log to roll even if the segment file isn't full to **ensure that retention can delete** or compact old data.

Source: https://docs.confluent.io/platform/current/installation/configuration/topic-configs.html

Data cleanup management

Recap Day1: Cleanup old messages – «compact» retention policy



Source: https://kafka.apache.org/documentation/#compaction

Data cleanup management

Recap Day 1: Cleanup old messages – «compact» retention policy

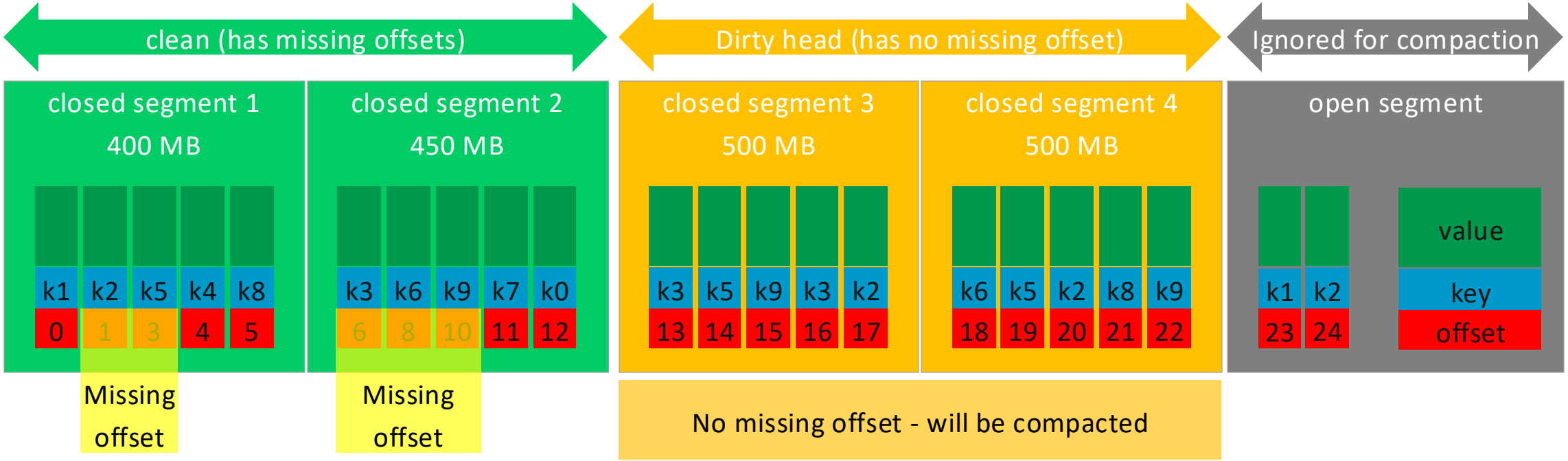
With this cleanup policy, the Kafka message key becomes more semantics. We already discussed that the key is related to an aggregate (business object). Cleanup removes old messages with the same key from a topic. With other words, if you use an aggregate id as key, old messages of an aggregate are removed.

Some Rules:

- **The active segment is not taken into consideration during compaction.** → we can always have several messages with the same key
- In the 'old' segments, the compaction process copies the newest message of each key to a new segment.
  - The index of the message is maintained.
  - After compaction, old segments are replaced by the new segments (that contain the copy of the newest messages). Old segments are deleted from the disk.
- **Compaction is triggered when a new message arrives** and
  - A given (configurable) time is elapsed or
  - A given (configurable) dirty ratio is reached

Compaction

Recap day 1: Clean and dirty records



By default, compaction starts if the partition's dirty ratio is > 0.5

Which means that compaction will be triggered if more than 50% of the total segment file size is dirty.

(500 MB + 500 MB) / (400 MB + 450 MB + 500 MB + 500 MB) = 0.54

Compaction

Tombstone Message

- A message that has a **key** and a **null** value.
- Will do a normal compaction and **retain only** the message with **null** value.
- After a configured period, the tombstone itself is **deleted**
  - **Can be used to enforce GDPR rules**
- Topic configuration “delete.retention.ms":
  - The amount of time to retain delete tombstone markers for log compacted topics.
  - ***Why do think it makes sense to keep this bigger than 0?***

Compaction

Tombstone Message

- A message with null value cannot be sent by Kafka **console producer**
  - Use kafkacat
    - Example for message key *Key*: `echo "key:" | kafkacat -b localhost -t mysql-01-events  -Z -K:`
  - Use AKHQ
    - Select the "Tombstone" Checkbox when producing a message

Case study:
Witness protection program

Case study

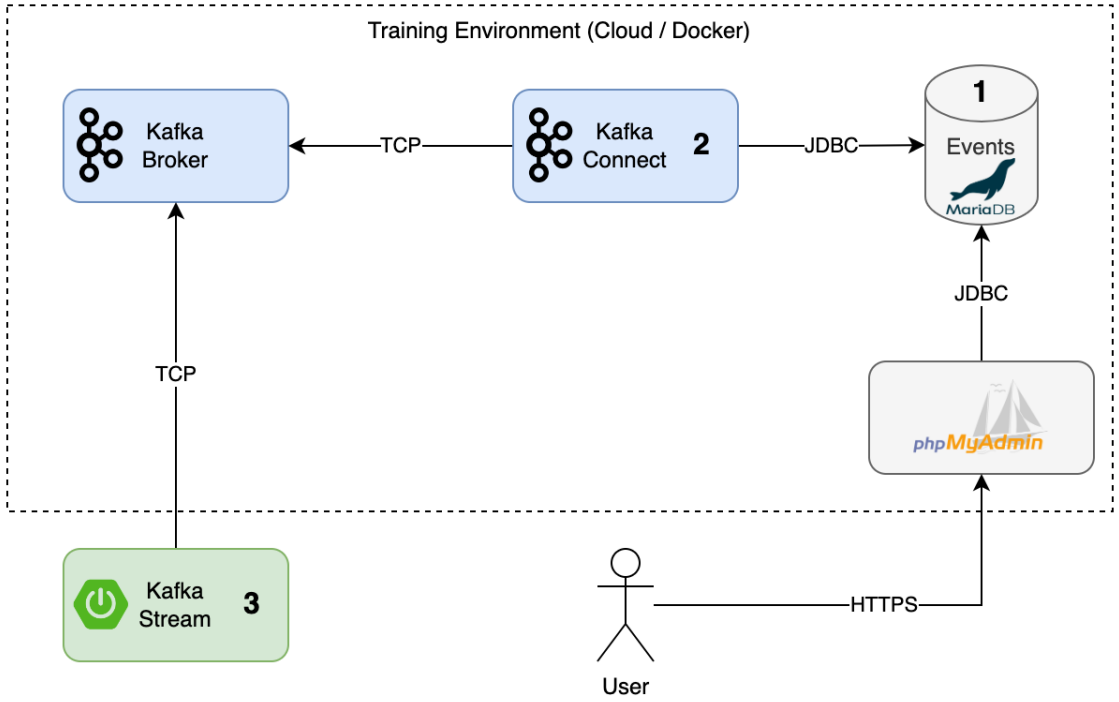
Overall information

This fictional use case is **inspired** by a **real-life case.**

- A double agent moves to canton Zurich to collect evidence of a criminal organization.
- On his journey he is generating events which are stored in a traditional database using the **Swiss ech20 standard** defined in **XML**. An event can be moving-in or moving-out.
- You work for the canton Zurich as an IT professional and your task is to **ingest** and **process** the events with **Kafka**.
- As soon as the double agent is **done**, we need to ensure that **no events** are stored in Kafka anymore.

Case study

Setup



1: MariaDB

- contains all events. It is already prepared with test data of simplified events. JDBC URL: jdbc:mysql://mariadb:3306/events

2: Kafka Connect

- runs in a Docker container with the name **kafka-connect-01.** It will be used to ingest the events via JDBC connector.

3: Kafka Streams

- used to write the data to other topics.

The folder **uc-witness-protection** contains the material to support the exercises.

Lab Witness protection program

Lab

Case study: Witness protection program

Follow the lab exercise from:

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/witness-protection.md

Day 3

Kafka Training

Agenda Day 3

09:00 – 09:30 Recap Day 2

09:30 – 10:30 Kafka Security

10:30 – 10:50 Break

10:50 – 12:00 Logging & Monitoring / Cluster management

12:00 – 13:00 Lunch Break

13:00 – 14:30 KSQL & Rest Proxy

14:30 – 15:00 Break

15:00 – 15:30 Best Practices & Outlook

15:30 – 17:00 Recap and Feedback / Discussion

Security
TLS Encryption, Client Authentication & ACLs

Security

Kafka Security

- Data Protection
  - TLS (Transport Layer Security)
- Authentication
  - SASL (Simple Authentication Security Layer)
  - Mutual TLS (Two way authentication with TLS)
- Authorization
  - ACL (Access Control List, Kafka)
  - RBAC (Role-based access control, on top of ACL)

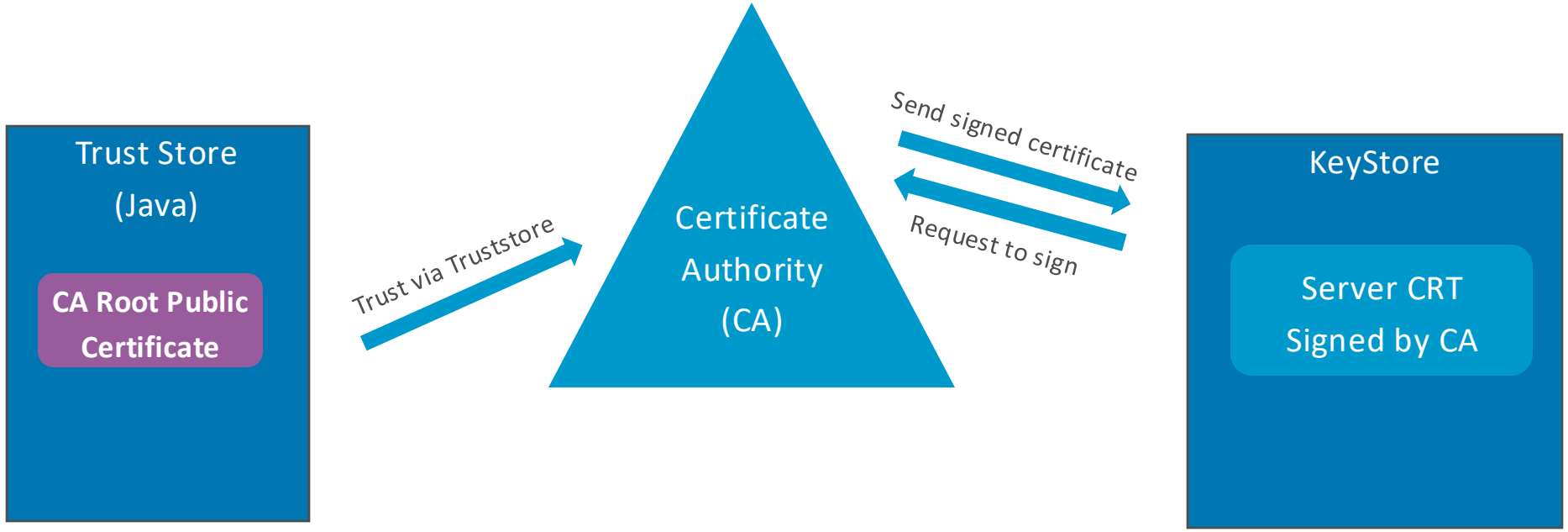
Security

Recap on terms

- **Certificate authority:** In cryptography, a certificate authority or certification authority (CA) is an entity that stores, signs, and issues digital certificates. (Big vendors like **GeoTrust** or **Verisign**)
- A **certificate signing request** (CSR) is one of the first steps towards getting your own SSL/TLS certificate. Generated on the same server you plan to install the certificate on, the CSR contains information (e.g. **common name**, organization, country) the Certificate Authority (CA) will use to create your certificate.
- **Client authentication** provides additional authentication and access control by checking client certificates at the server. This support prevents a client from obtaining a connection without an installation approved certificate.
- **Server Hostname Verification:** Hostname of server stored in the server certificate matches the host the client is connecting to.
- **Java keystores**: A Java keystore stores **private key entries**, certificates with public keys, or just secret keys that we may use for various cryptographic purposes. It stores each by an alias for ease of lookup.
- **A truststore** is the opposite. While a keystore typically holds onto certificates that identify us, a truststore holds onto certificates that identify others.

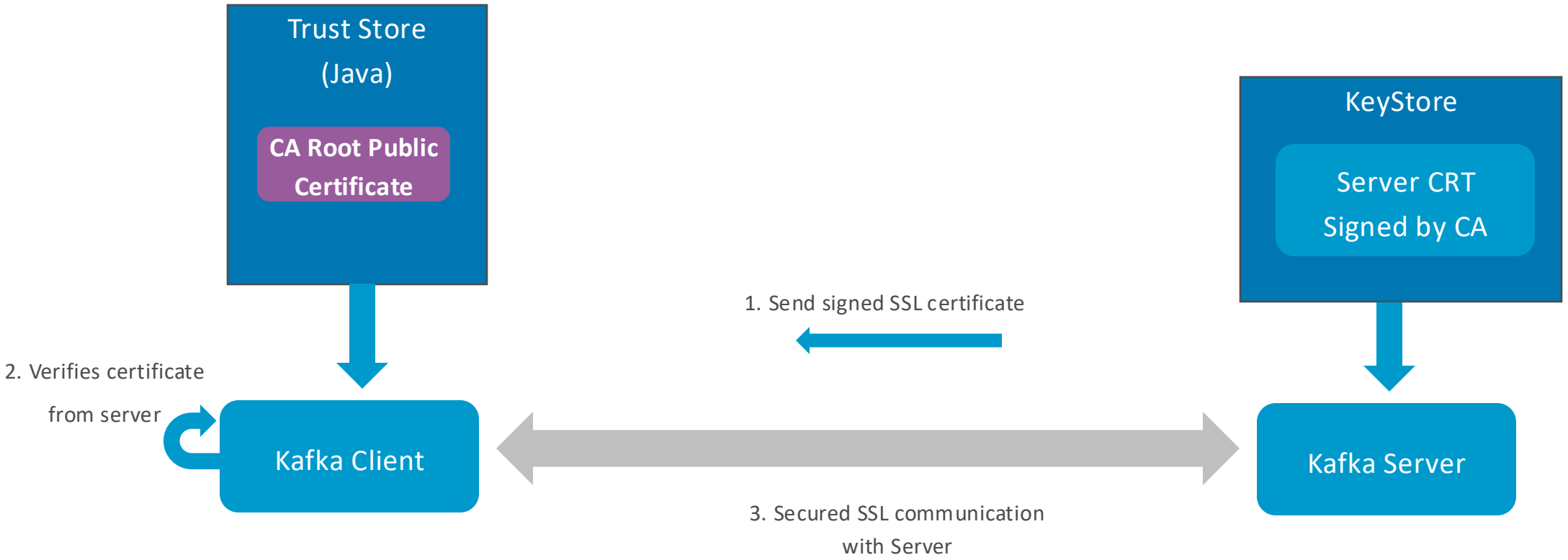
Kafka Security

Overview of certification: Setup



Kafka Security

SSL Handshake: Simplified for use case



Security

Authentication

- **PLAINTEXT:** No authentication. Only Suitable within private networks and no sensitive data.
- **SSL/TLS:** Transport layer encryption with optional **client authentication (mTLS)**
- **SASL** (Simple Authentication Security Layer) is a framework that provides developers of applications and shared libraries with mechanisms for authentication, data integrity-checking, and encryption.
  - **SASL/GSSAPI** uses your Kerberos or Active Directory server for authentication.
  - **SASL/OAUTHBEARER** uses Oauth2
  - **SASL/PLAIN** uses username/password
  - **SASL/SCRAM** uses Zookeeper stored username/password
  - **LDAP**, and more...

- Further Information: See here
- SASL explained: See here

Security

Client Authentication (two-way authentication)

KAFKA_SSL_**KEYSTORE**_FILENAME: kafka.broker.keystore.jks

KAFKA_SSL_**KEYSTORE**_CREDENTIALS: broker_keystore_creds

KAFKA_SSL_KEY_CREDENTIALS: broker_sslkey_creds

KAFKA_SSL_**TRUSTSTORE**_FILENAME: kafka.broker.truststore.jks

KAFKA_SSL_**TRUSTSTORE**_CREDENTIALS: broker_truststore_creds

KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:29092,PLAINTEXT_HOST://localhost:9092,SSL://**broker**:9093

KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT,**SSL**:SSL

KAFKA_SSL_CLIENT_AUTH: 'required'

Note: In the training environment these variables are written to "/etc/kafka/kafka.properties" on the broker.

Security

Authorization with ACL (Access Control Lists)

- ACLs are stored in Zookeeper or KRaft metadata log

- ACLs define access to
  - **Topics**: Which consumer/producer can read/write data
  - **Consumer Groups**: which client can use a specific consumer group
  - **Cluster**: which client can create / delete topics or apply settings

- ACLS can **Allow** or **Deny** access

- Managing with tool command line tool “kafka-acls”
- There is a super user (*super.users* configuration) who can do everything

- Further information: See here

Security

ACL Authorization components

- **Authorizer**: An authorizer is a server plugin used by Kafka to authorize operations
  - Kafka ships with a default **AclAuthorizer** that has to be enabled

- **Principal**: An entity (user) that can be authenticated, depends on the security protocol
  - For SSL client authentication taken from the **certificate**

- **Operations**: Types of operations that are possible for a certain resource (read, write, alter, …)

- **Resource**: Type of resource (topics, clusters, groups)
  - Resource Patterns can be defined (e.g. prefix)

ACLs applied

Typical access matrix in a microservice architecture

Topics / Service	Account Management Service	Authorization Management Service	Device Management Service	Audit Trail Service	…
usermanagement.accountmanagement.avro.v1	rw	r	r	r	…
authorisationmanagement.usergroup.avro.v1		w		r	…
authorisationmanagement.devicegroup.avro.v1		w		r	…
authorisationmanagement.permissionassignment.avro.v1		w		r	…
devicemanagement.device.avro.v1		r	w	r	…
devicemanagement.deviceconfiguration.avro.v1				r	…
…	…	…	…	…	…

Security

ACL Authorization example

- In the following ACL, the plain text principals (User:alice, User:fred) are identified as Kafka users who are allowed to run specific operations (read, write) from either of the specified hosts (host-1, host-2) on a specific resource (finance-topic):

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-configs.conf \
  --add --allow-principal User:alice --allow-principal User:fred --allow-host host-1 \
  --allow-host host-2 --operation read --operation write --topic finance-topic
```

Lab Security

Security

Exercise: Kafka Security – what we are going to do

Lab

Exercise: Kafka Security

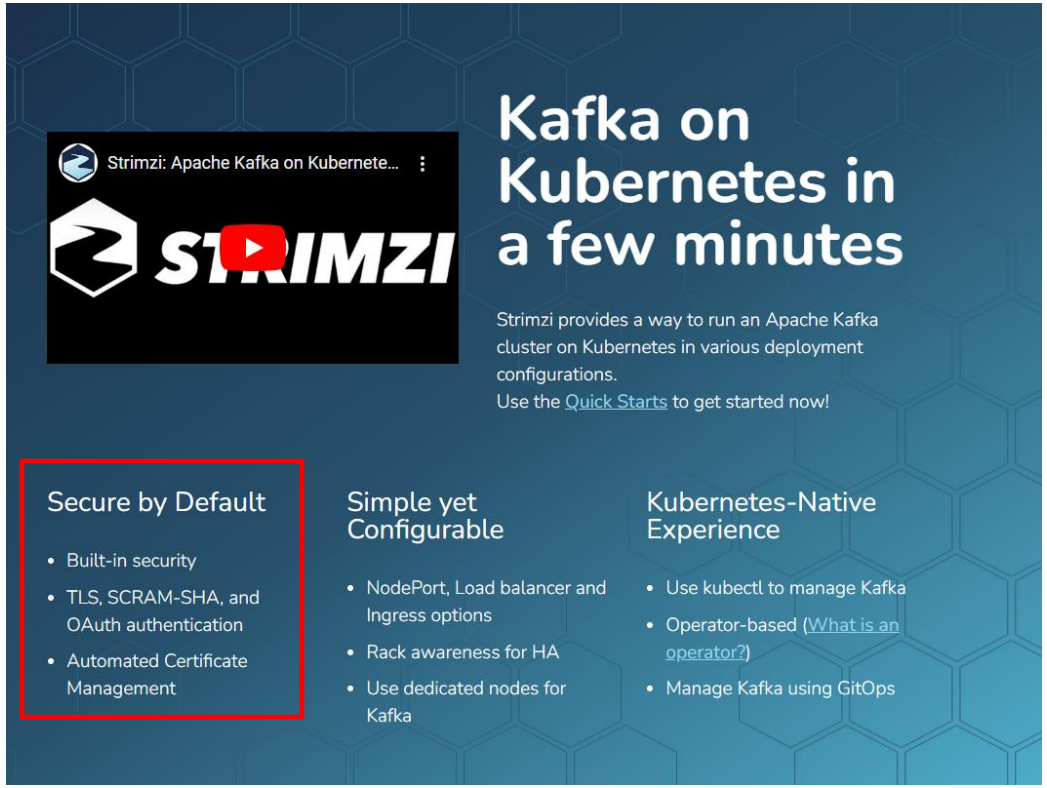
Follow the Kafka Security Exercise from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/kafka-security.md

Security

Strimzi Kubernetes Operator

- Deploying and running Kafka clusters
- Deploying and running Kafka components
- Configuring access to Kafka
- **Securing access to Kafka**
- Upgrading Kafka
- Managing brokers
- Creating and managing topics
- Creating and managing users



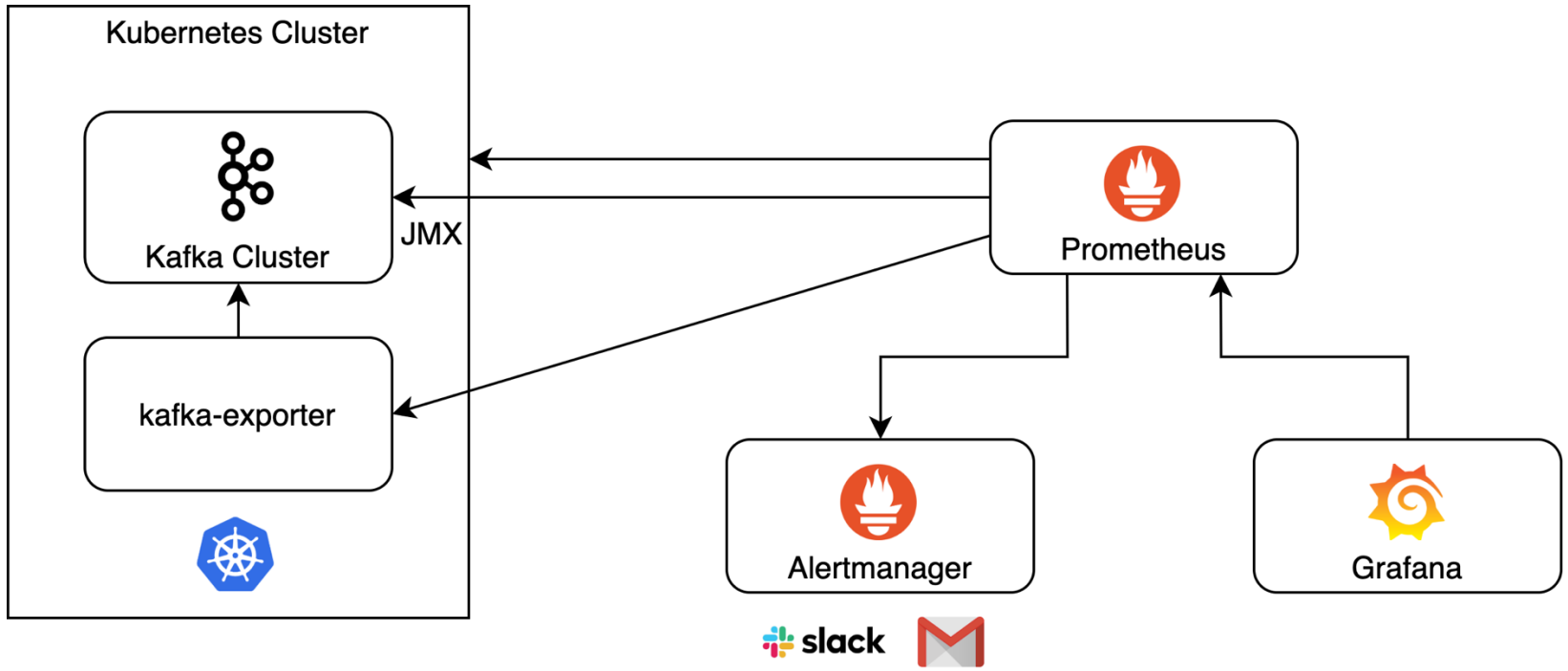
https://strimzi.io/docs/operators/latest/overview#key-features-product_str

https://strimzi.io/

Monitoring and Logging

Monitoring

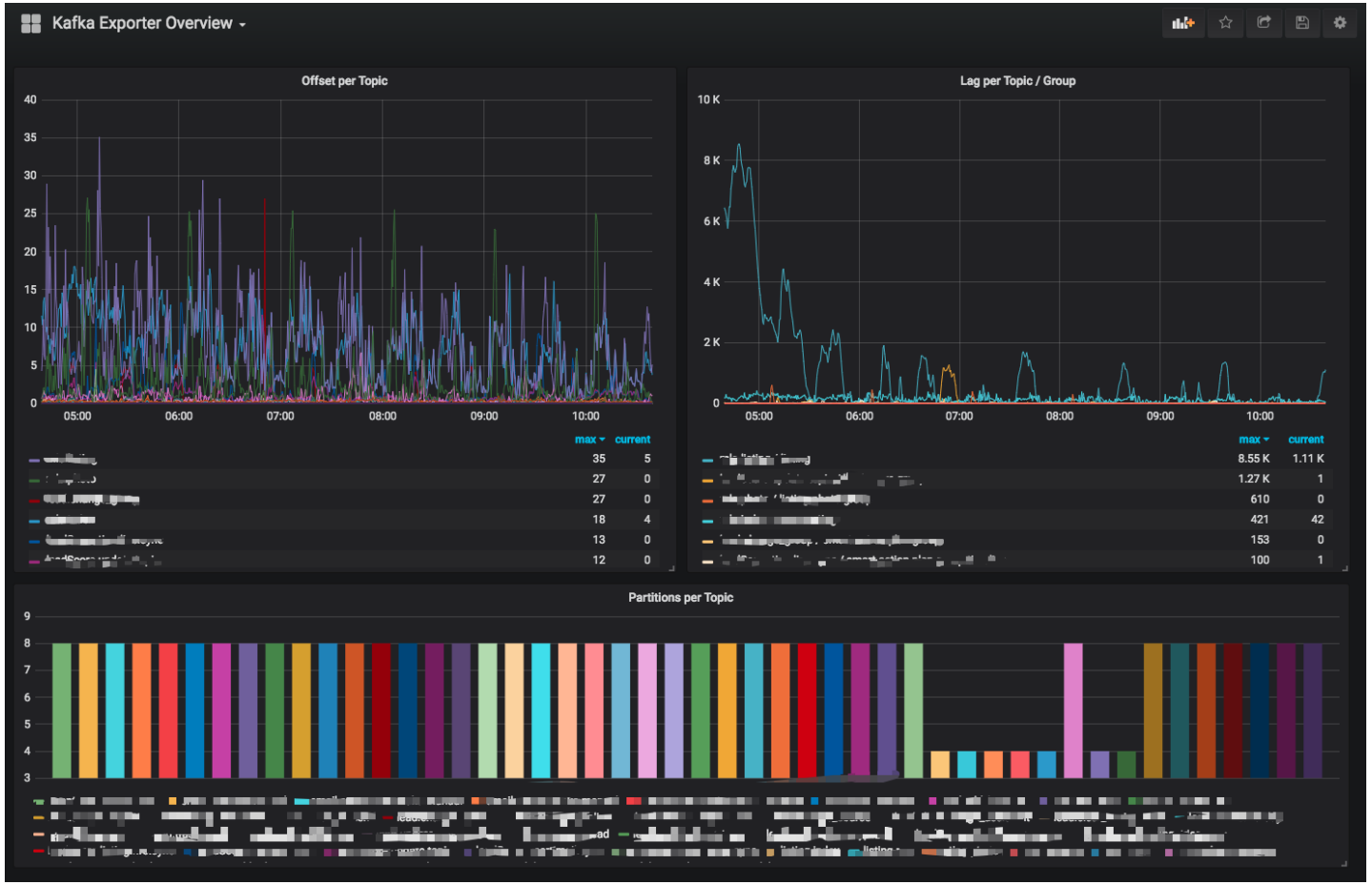
- Prometheus scraping metrics
  - Kafka / Connect via JMX
    - Cluster health
  - Kafka-exporter
    - Topics Offset
    - Consumer Lag
  - Kubernetes / Openshift Cluster
    - Pod restarts
    - PV Space
    - CPU / Memory Usage
  - Consumer / Producer / Streams via JMX
    - Client health



Source: https://github.com/danielqsj/kafka_exporter

Monitoring

- Prometheus scraping metrics
  - Kafka / Connect via JMX
    - Cluster health
  - Kafka-exporter
    - Topics Offset
    - Consumer Lag
  - Kubernetes / Openshift Cluster
    - Pod restarts
    - PV Space
    - CPU / Memory Usage
  - Consumer / Producer / Streams via JMX
    - Client health

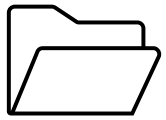


Source: https://grafana.com/grafana/dashboards/7589-kafka-exporter-overview/

Logging

- Apache Log4j
  - Zookeeper
  - Kafka Brokers
  - Kafka Connect

- Consumer / Producers
- Streams

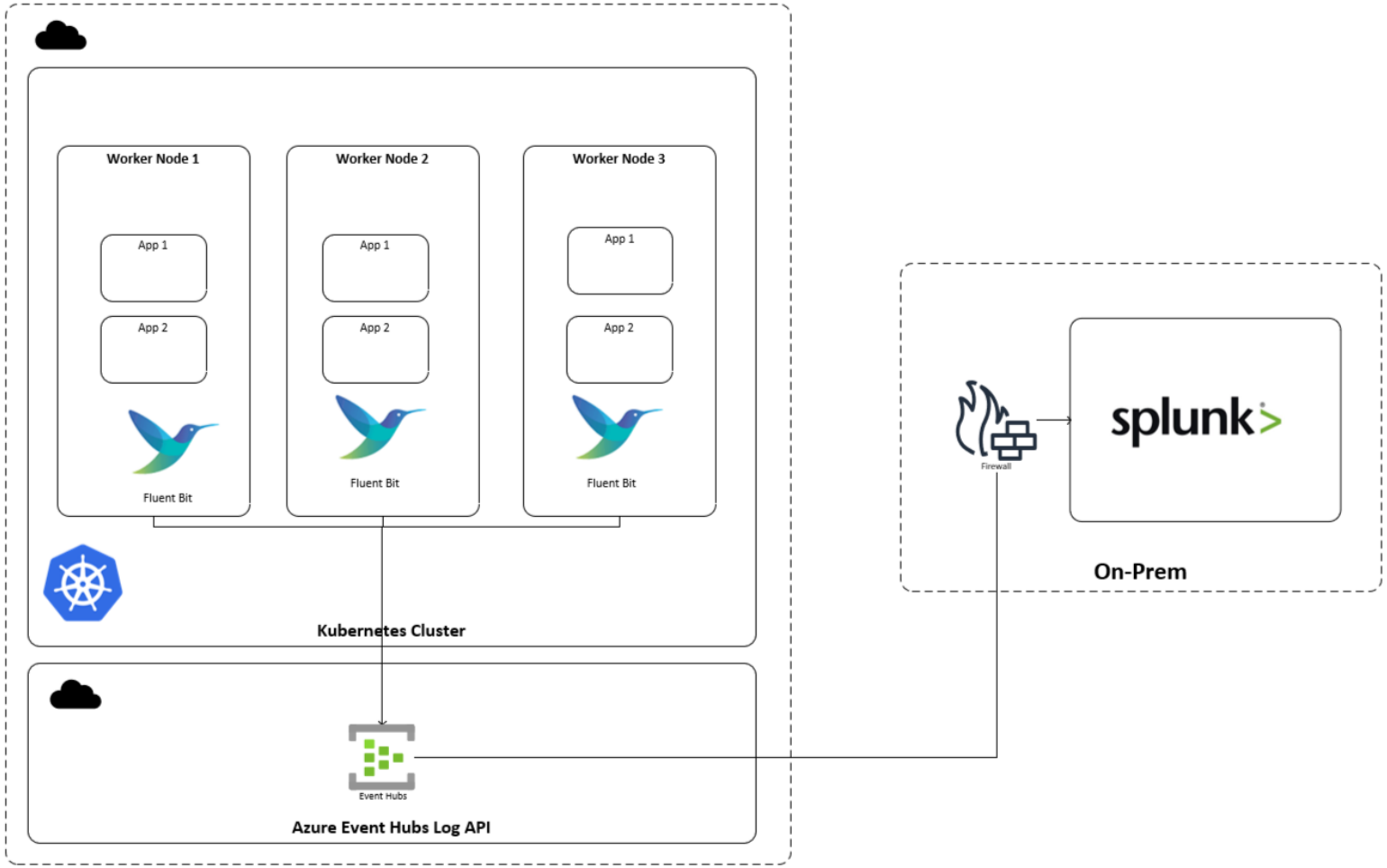


By default, log are written to stdout

Modify conf/log4j.properties to change settings

Logging

Client Case: Kafka for Logging



Kafka Connect

Monitoring, Logging and Debugging for Kafka Connect

Standard monitoring and logging integration:

- Logging: Configure Log4j for Kafka Connect: https://docs.confluent.io/platform/current/connect/logging.html

- Monitoring: Connect to REST and JMX endpoints: https://docs.confluent.io/platform/current/connect/monitoring.html
  - Example: Connector Status – running, paused or stopped

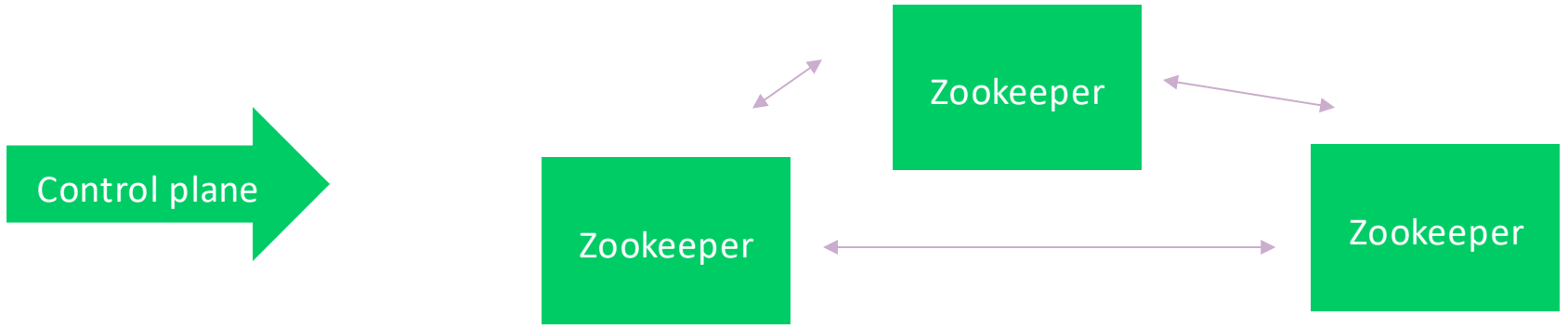
During development time, enable debugging and fast recovery for a specific connector:
https://docs.confluent.io/platform/current/installation/configuration/connect/source-connect-configs.html

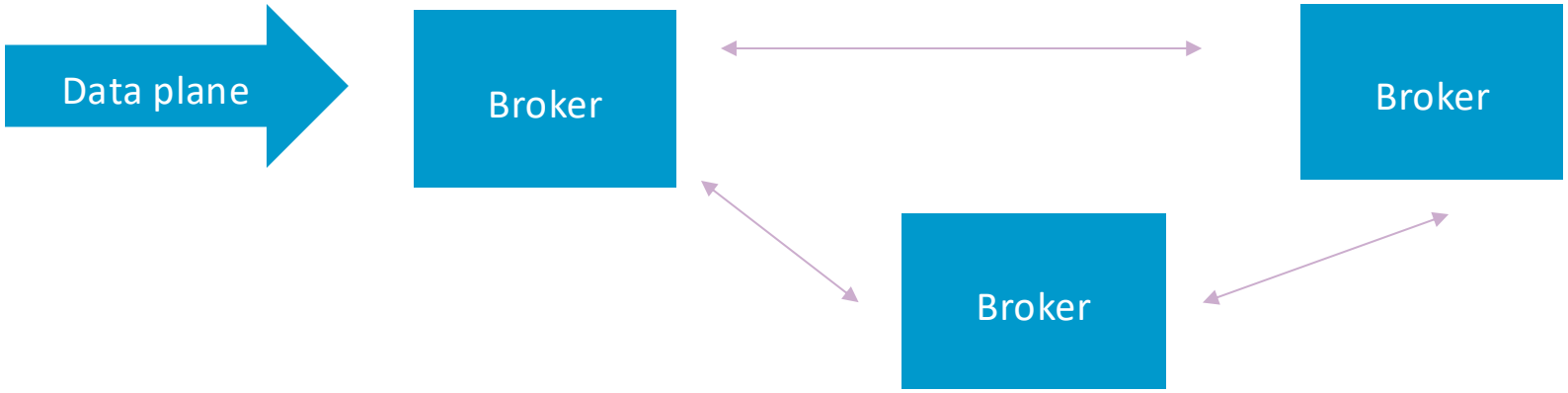
- errors.log.enable: true (default false) => write details of the failed operation to the application log
- errors.log.include.messages: (default false) => log record that resulted in the failure
- errors.tolerance: all (default none) => skip problematic records and continue

Clusterdesign

Cluster Design

We have to look at two different topics

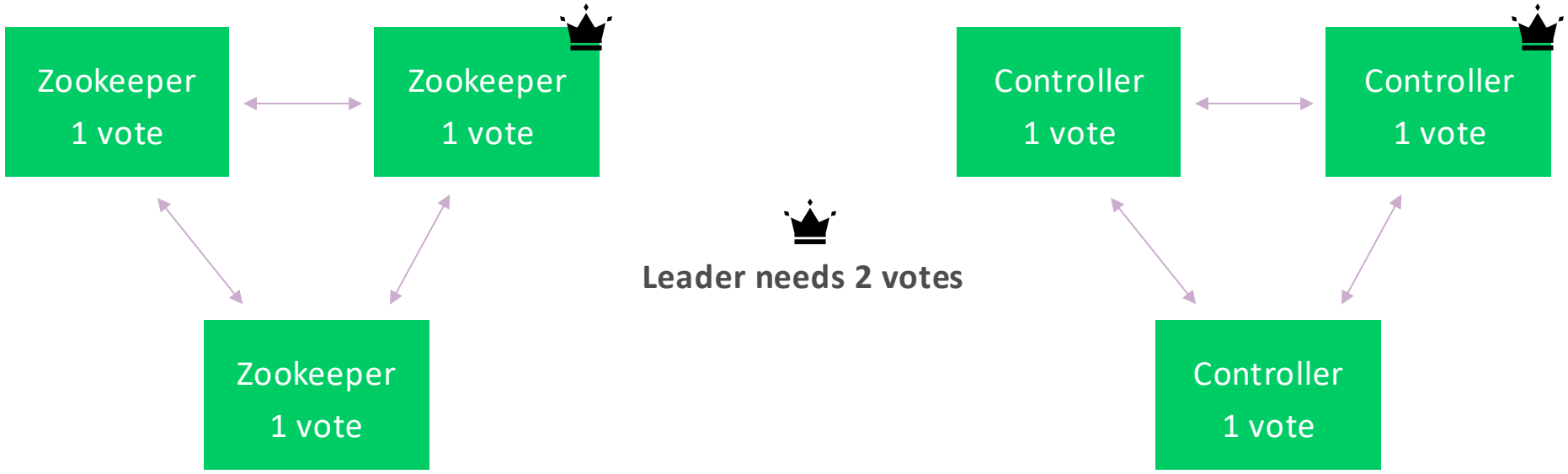




Quorum

Leader election in Kafka

- Kafka consists of a distributed set of nodes. Communication between nodes can be lost at any time (e.g. network outage)
- To keep the cluster running, a consensus among the “control plane” (Zookeepr or KRaft Controller) needs to be achieved
- **Majority rule or Quorum**: Minimum Number of Servers required to guarantee consistency
- Quorum = (Total Number of Servers + 1) / 2



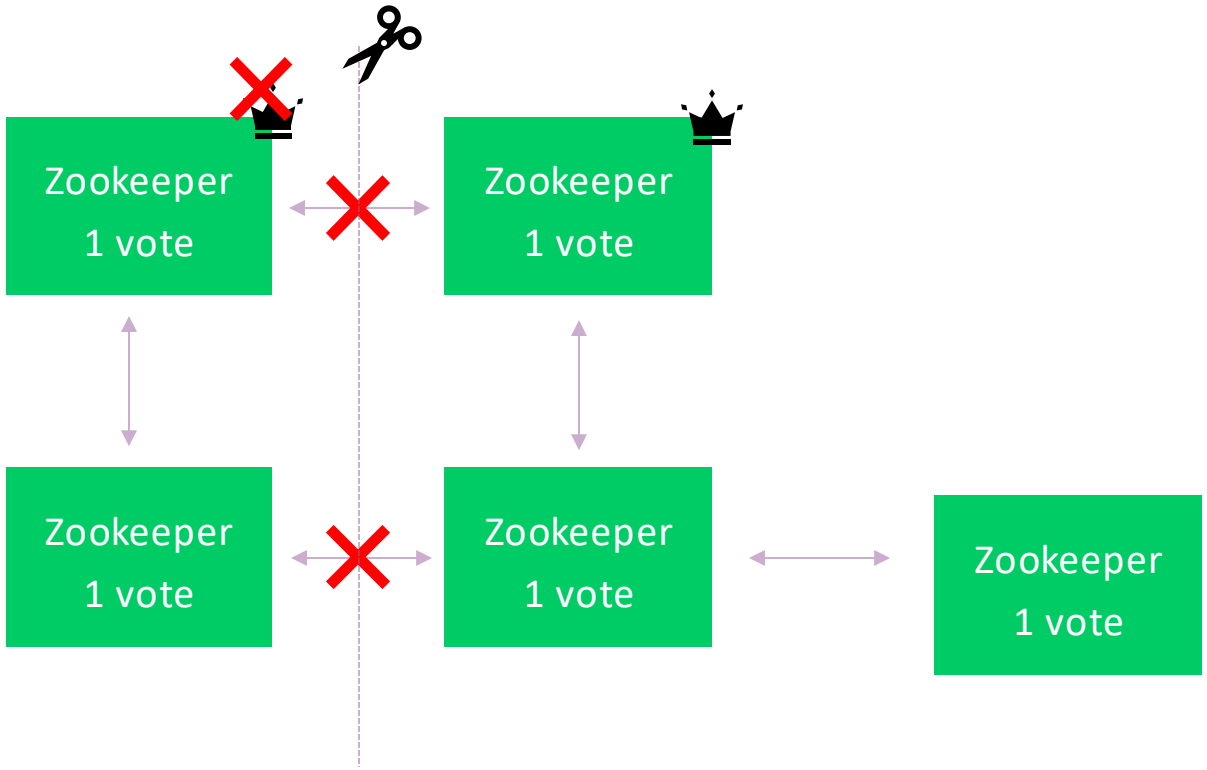
Kafka with Zookeeper

Kafka with Kraft > 3.5

Quorum

“Split Brain”

- If majority rule is not followed and a network outage happens between two equally sized groups of Zookeepers / Controllers
- Both sides think they have achieved consensus and build their own cluster, drifting apart

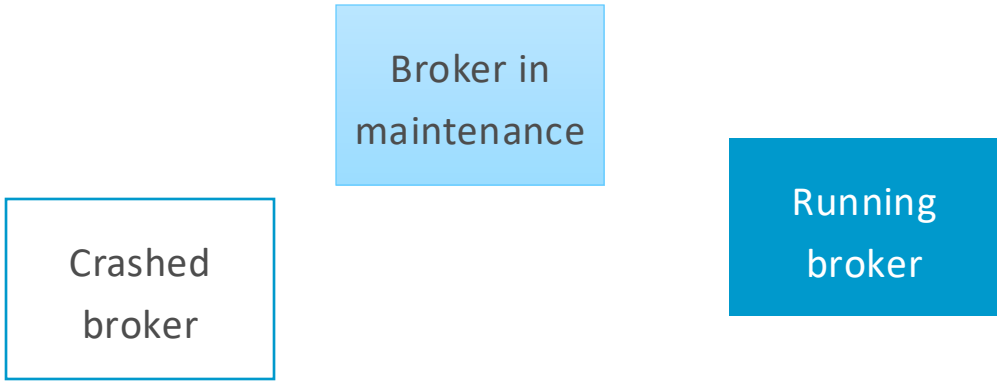


Cluster Design

How many brokers run in your cluster?

Availability view:

- You might have one broker, if availability of your Kafka cluster is not a requirement.
- If availability* of your Kafka cluster is a requirement, and you run in one data center, **you should have at least 3 broker**.
  - One broker is in maintenance mode (e.g., upgrade)
  - One broker fails
  - Still, from the replication perspective, you might have a problem



* as an example, clients have **7/24 service**

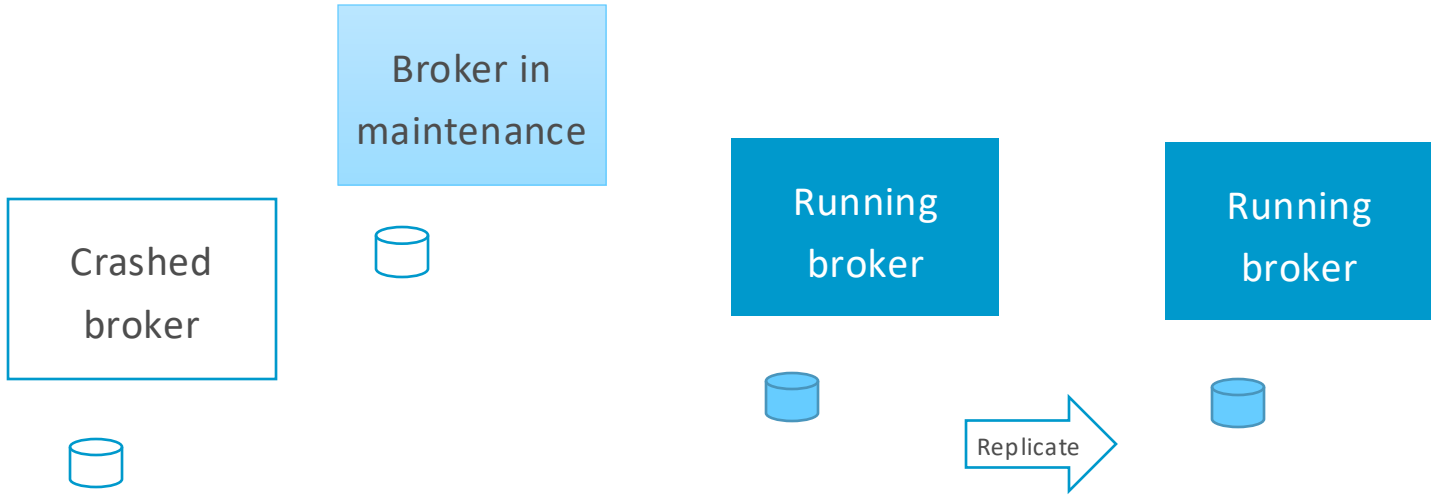
Cluster Design

How to choose the replication factor in a Kafka topic

(Data-)availability view:

Assume that our client asks for '**minimum replication factor = 2**' to assure at **least one copy is there in the case of a disk crash**.

- Continuing with the previous scenario, Kafka would not be available for our client, as the running broker cannot fulfill the contract anymore:



- The **replication factor** would **be 4**, so that you can at any time upgrade one broker, and still one broker can crash at the same time.

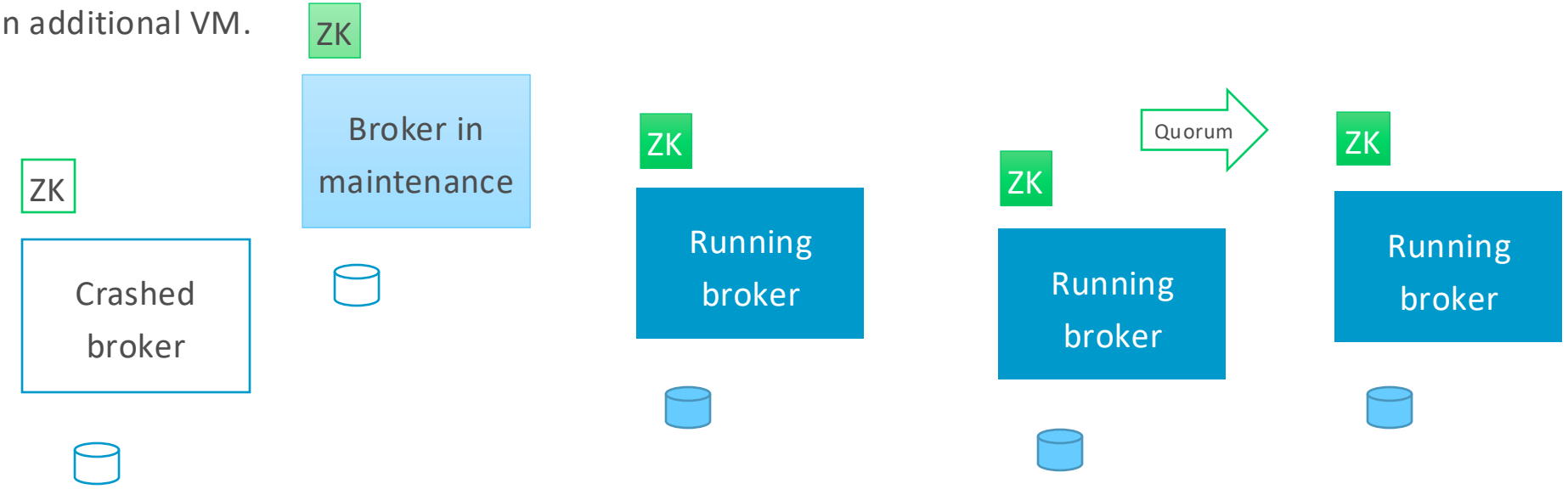
Cluster Design

How many Zookeepers do you need?

Availability view:

Assume that we run our cluster on VM's (or hardware). **Zookeepers would also run on the same VM.** As Zookeeper need Quorum to fulfill their job, we must have an **impair** number of Zookeepers to run.

- With 3 Zookeepers, we would probably have the wrong machines down in the 'maintenance and failure' scenario
- Even with 5 Zookeepers (two on the same machine) the problem is not solved. The wrong VM could crash.
- We need an additional VM.

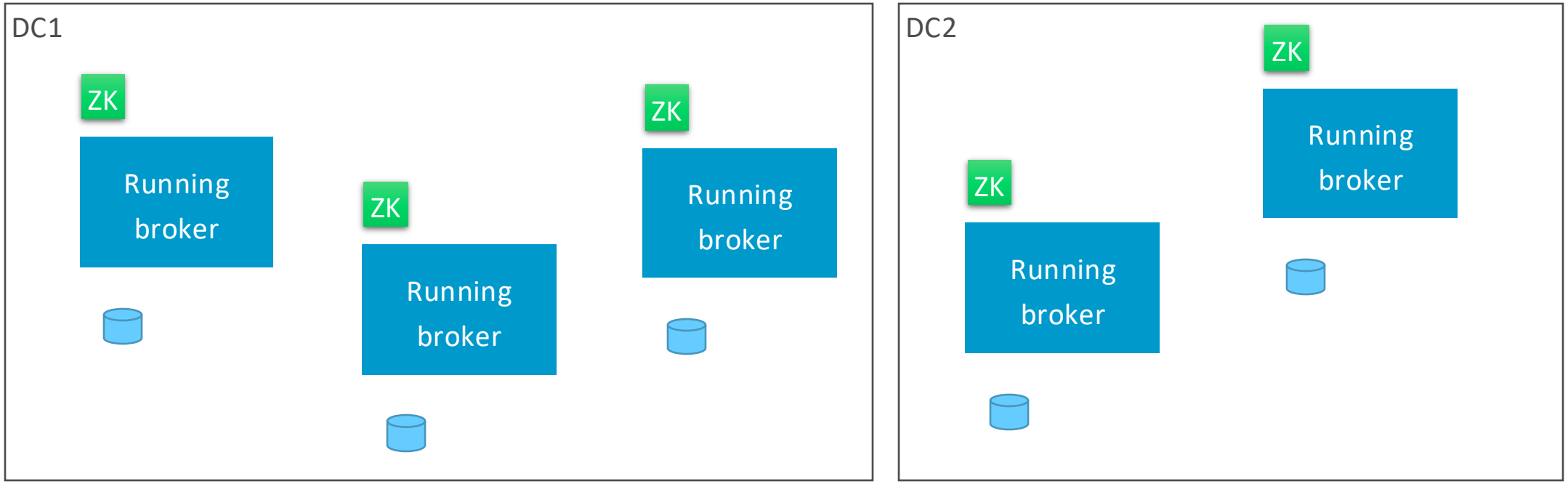


Cluster Design

How to setup Kafka in a multi-location datacenter

Typically, companies with HA-Services locate their datacenter **in at least two different locations**. The services are with **hot-** or **cold standby co-located**. We assume in this scenario that we have two datacenters in different locations. **They are connected with a low-latency network**, so that the services can be provided as if they were in the same DC.

- Our brokers run as one Kafka cluster. What happens if one DC 'burns down'? How can we address this problem?

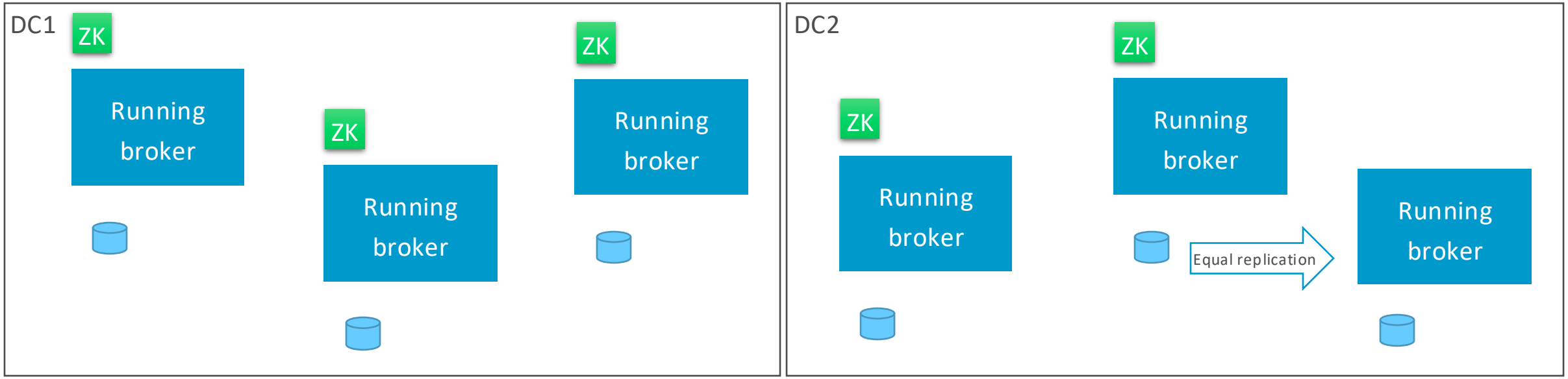


Cluster Design

How to setup Kafka in a multi-location datacenter

In a **hot-standby scenario**

- 'No data loss' scenario: With our setting '**minimum replication factor=2**' we cannot guarantee that, at a given point in time, the data is distributed among both datacenters. But you can tell the cluster which broker runs in which datacenter. If possible, Kafka will then distribute the replicas among the datacenter. (**broker.rack='DC1').** It might be a good idea to start an additional broker so that the partitions are equally distributed in the normal operation mode.

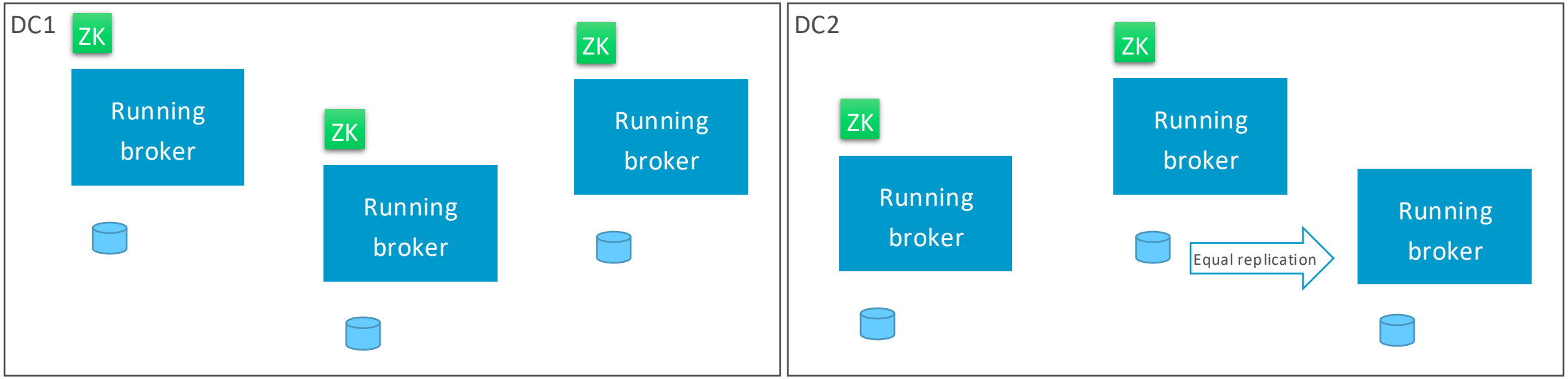


Cluster Design

How to setup Kafka in a multi-location datacenter

In a hot-standby scenario

- You **cannot resolve the Quorum Problem with only 2 datacenters**. You have always a **minority** in one of the datacenters.
- A human could decide if the DC burned down or there is a Split Brain and start the necessary missing ZK instance on the remaining DC.

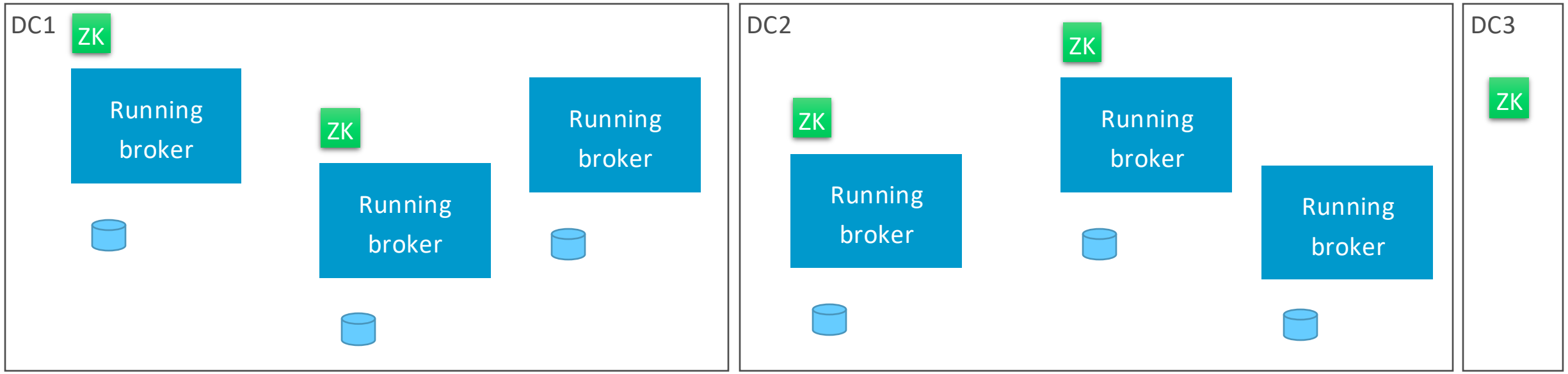


Cluster Design

How to setup Kafka in a multi-location datacenter

If a third Data Center is available

- Host an equal number of Zookeeper nodes in DC1 and DC2
- Host a single Zookeeper node in DC3
- This way, if one region goes down, there is still a majority of Zookeepers to build the Quorum



Cluster Design

Multi-Location datacenter setup – Why?

- Disaster Recovery:
  - Prevent data loss if one data center fails
  - Prevent interruption of the services if one data center fails
- Minimized Latency
  - Guarantee low latency for users in different regions by having a cluster closer to the user location
- Data governance
  - Comply with laws that require user data to be stored in certain region

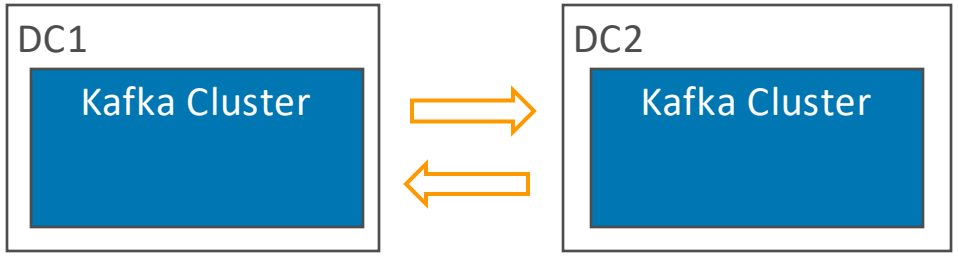
https://docs.confluent.io/platform/current/multi-dc-deployments/multi-region-architectures.html

Cluster Design

Multi-Location datacenter setup - Architectures

- Stretched Multi Region Cluster
  - Works best if the regions are relatively close or connected with highspeed networks

- Multiple Clusters in different Regions with async data replication → **RTO and RPO is > 0!**
  - Mirror Maker 2 (based on Kafka Connect) or Confluent Replicator
  - Confluent Cluster Linking



https://developer.confluent.io/courses/architecture/geo-replication/

Best Practice Configuration

Kafka Config

No automatic creation of topics

- Broker:
  - auto.create.topics.enable = **false**
    - Disable automatic creation of topics in the production environment
    - Create topics manually with custom configs
    - No effect on internal topics used by Kafka Streams

Kafka Config

Cluster durability

- Topic
  - replication.factor **> 1**
    - Replicate message to multiple partitions

- Broker:
  - min.insync.replicas = **2**
    - Minimum replicas a message has to be written to before it will be acknowledged

- Producer:
  - acks = **all**
    - Wait for all in-sync replicas to acknowledge the message being received

Kafka Config

Cluster durability – exactly once

- Stream
  - PROCESSING_GUARANTEE = **EXACTLY_ONCE**
- Consumer:
  - enable.auto.commit = **false**
    - Wait with commit until message has been pocessed
  - isolation.level = **read_commited**
    - Read only from stable offset
- Producer:
  - enable.idempotence = **true**
    - Prevent duplication of messages (default is true since Kafka 3.0)
  - max.in.flight.requests.per.connection = **1**
    - Prevent out-of-order messages

Kafka Configs

Optimization

- What do you need in your Use Case?

Availability

- session.timeout.ms

Throughput / Latency

- **batch.size**
- **linger.ms**
- compression.type
- buffer.memory
- fetch.min.bytes
- fetch.max.wait.ms

Durability

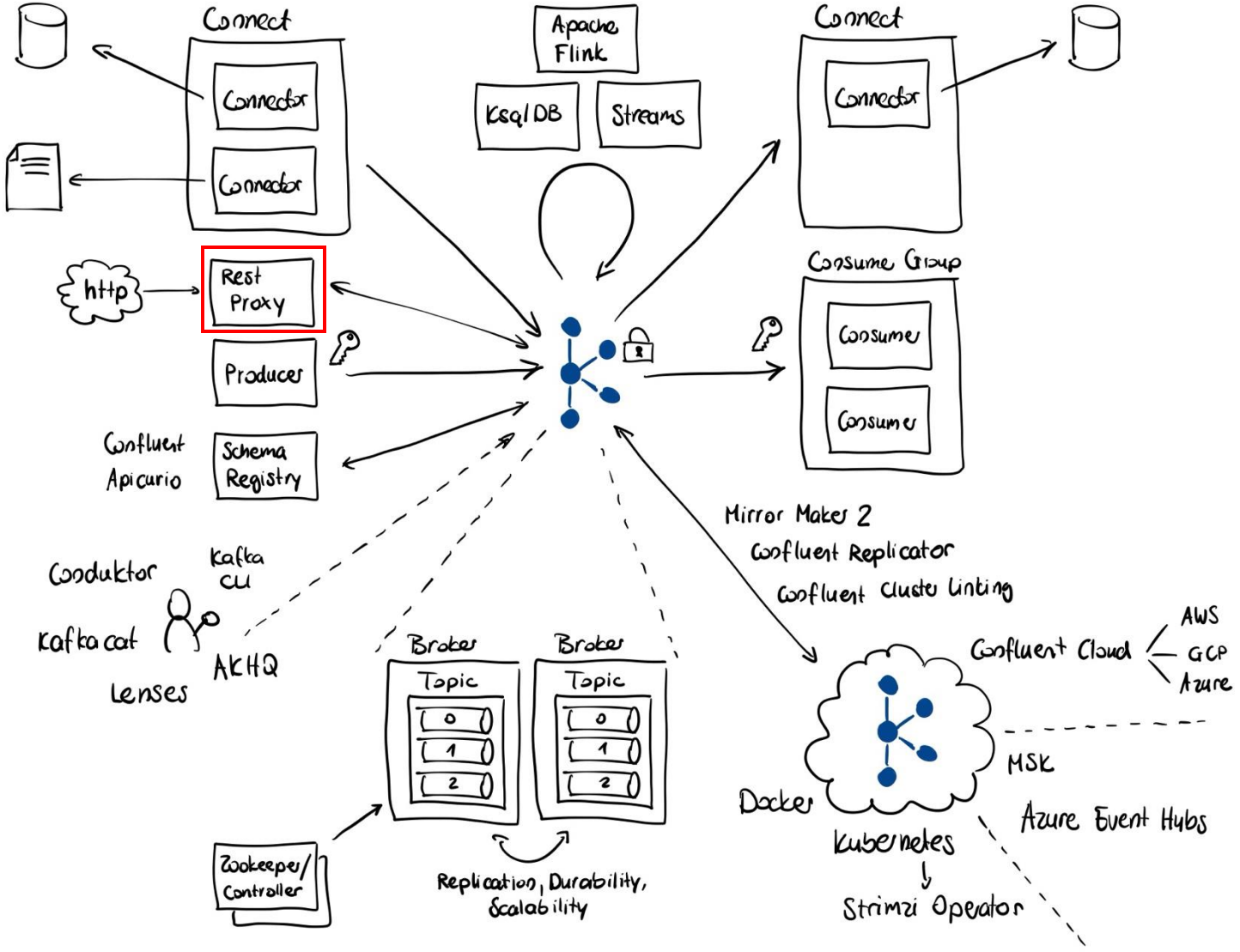
- replication.factor
- acks
- enable.idempotence
- max.in.flight.requests.per.connection
- enable.auto.commit
- Isolation.level

Source: https://docs.confluent.io/cloud/current/client-apps/optimizing/index.html

Extended Kafka Ecosystem

REST Proxy

Overview



REST Proxy

Confluent Extension to Kafka

The Kafka REST Proxy provides **a RESTful interface** to a Kafka cluster. It makes it easy to produce and consume messages, view the state of the cluster, and perform administrative actions without using the native Kafka protocol or clients.

Rest Proxy supports the following formats: json, binary, avro, protobuf and jsonschema

You find the different API calls here: https://docs.confluent.io/platform/current/kafka-rest/api.html

- For example, you can check the topics of your lab environment as follows: http://localhost:8082/topics or http://myvmsip:8082/topics
- You can check the settings of your plant-topics here: http://localhost:8082/topics/myPlant or http://myvmsip:8082/v3/clusters
- With the REST Proxy API v3, you can examine your cluster / broker / topic settings: http://localhost:8082/v3/clusters or http://myvmsip:8082/v3/clusters

Rest Proxy

When should you use it?

- If you want to write to / read from Kafka using a Framework / Language that does not support Kafka
- If you want to Script administrative actions
- If you don’t want to use a UI Tool to debug Kafka (e.g. AKHQ)

- Make sure to include Rest Proxy Security!

Limitations

- Delivery Guarantees
- Throughput

Lab Rest Proxy

Lab

Exercise: Rest-Proxy

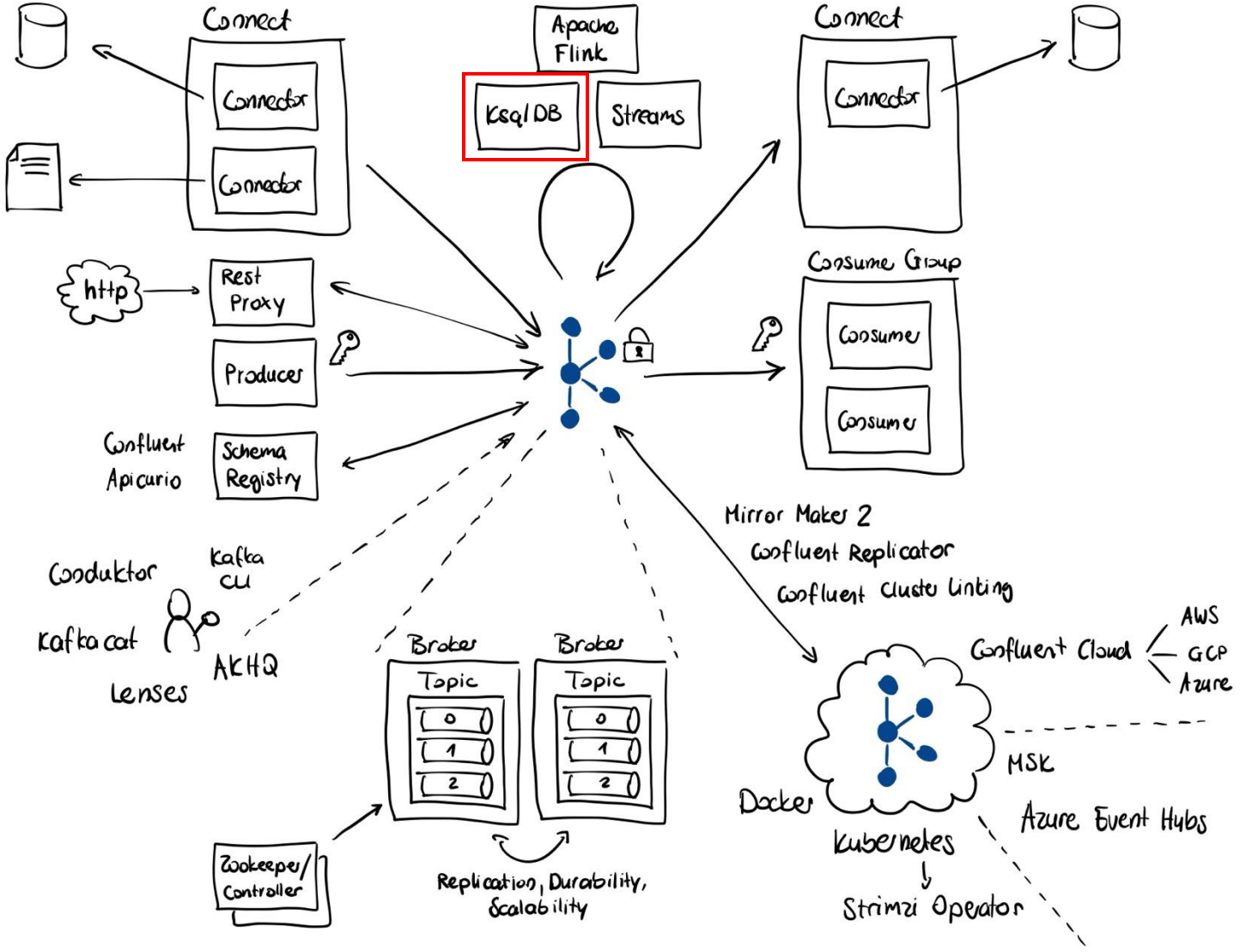
Follow the Rest-Proxy Exercise from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/rest-proxy.md

Extended Kafka Ecosystem

KSQL

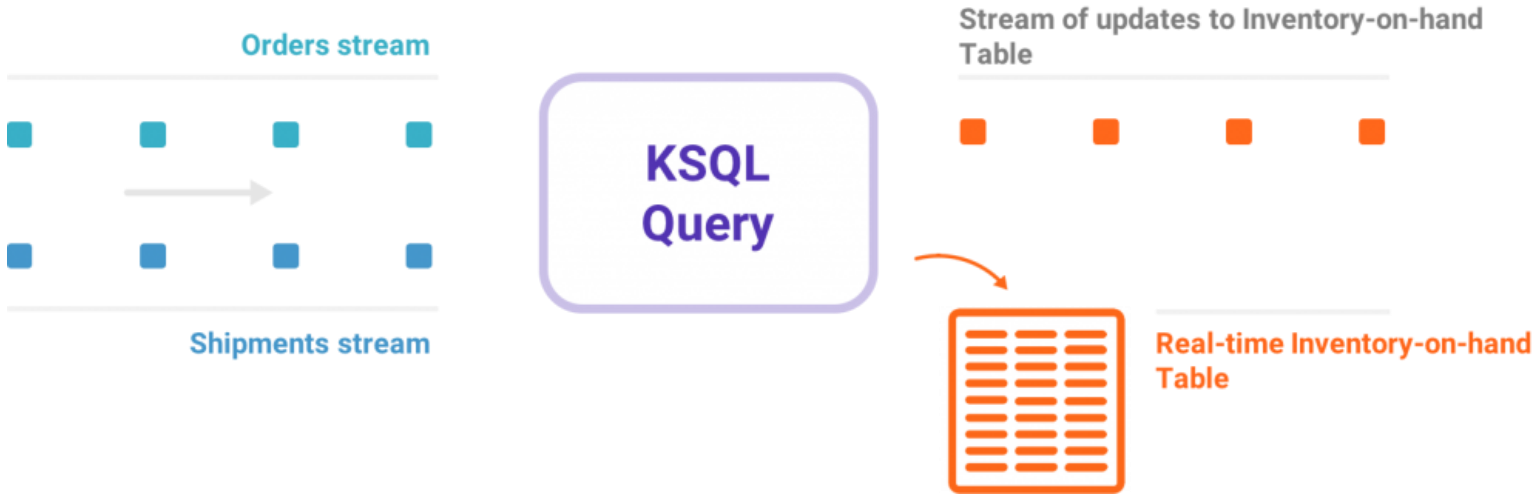
Overview



KSQL

Confluent Extension to Kafka

KSQL is a streaming SQL engine based on Kafka Streams. KSQL lowers the entry bar to the world of stream processing, providing a simple and completely interactive SQL interface for processing data in Kafka:



Source: https://www.confluent.io/en-gb/blog/ksql-streaming-sql-for-apache-kafka/

KSQL

Streams and Tables

You can create KSQL streams or KSQL tables from a topic. In the background, they are transformed topics.

- KSQL Stream: default topic
- KSQL Table: compacted topic

Once you have created a **stream/table**, you can emit SQL-Like commands to the stream/table:

```
select * from myPlant
    where sensor_id='mySensor'
    emit changes;
```

The result can be populated to a stream/table, and you can use the result in your applications. You have a powerful tool to create stream application without the need to learn a programming language (if you are familiar with SQL).

KSQL

Query Language

Beside filtering (WHERE…) you can also join data (JOIN…):

```
SELECT p.datetime, p.value, m.unit
       FROM myPlant as p
       LEFT JOIN metadata AS m
       ON s.sensor_id = m.sensor_id
EMIT CHANGES;
```

Lab KSQL

Lab

Exercise: KSQL

Follow the KSQL Exercise from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/ksql.md

KSQL

Windowing

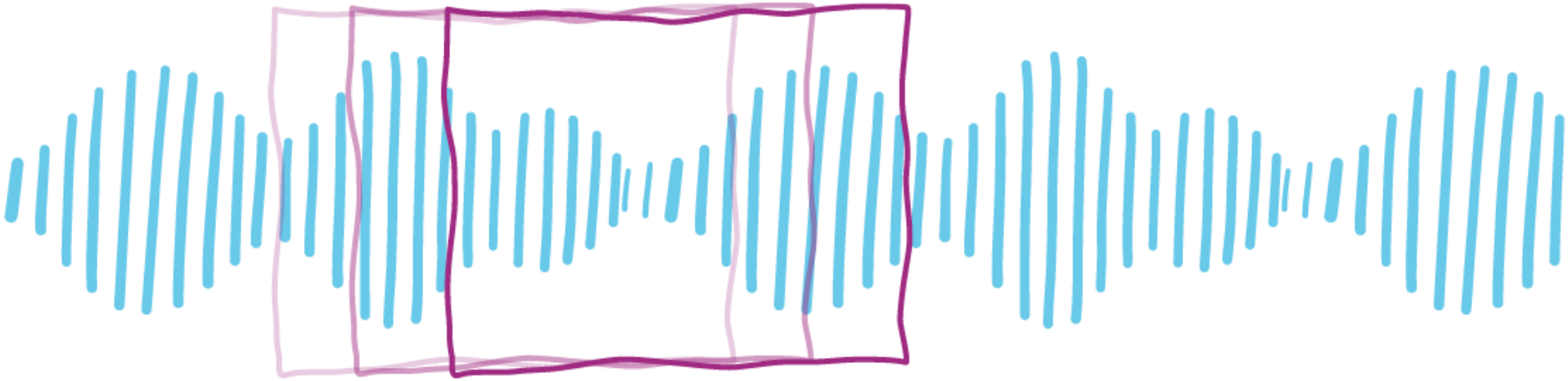
When building applications with **Kafka Streams** or **ksqlDB** involving aggregations, windowing becomes essential as it sets limits on the **accumulation of data over time**.

There are **four different ways** to define a time window:

- **Hopping** (Fixed-duration, have a hop interval, overlaps)
- **Tumbling** (Fixed-duration, no overlap)
- **Session** (Not fixed duration, based on duration of activity, separated by gaps of activity)

KSQL

Hopping Window



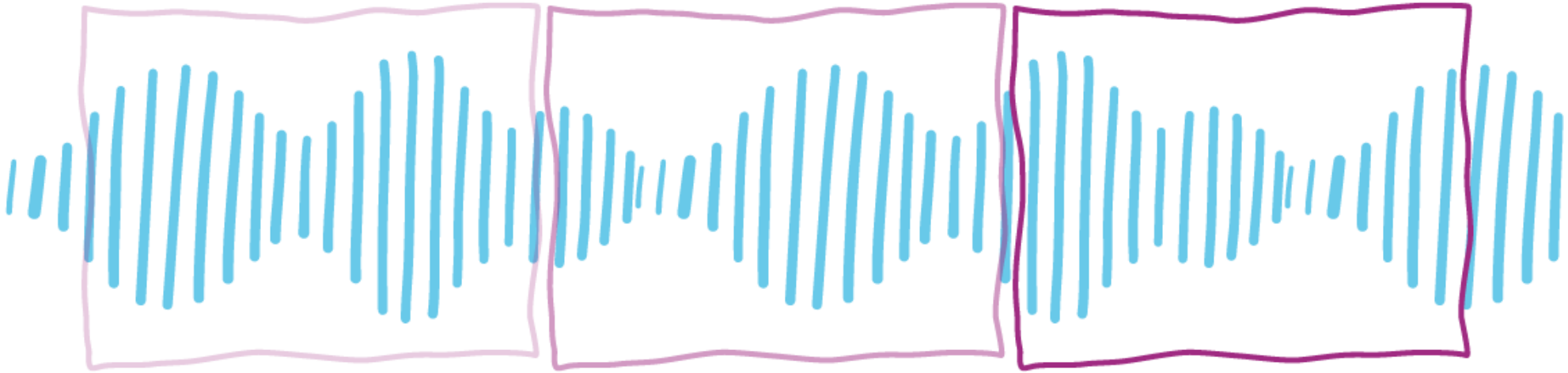
Fixed-duration, have a hop interval: window size + a time block at which it advances, overlaps

Example: Dashboard displays the moving average of clicks on a specific e-commerce page, calculated over 2-minute windows within the past 24 hours.

Source: https://www.confluent.io/blog/windowing-in-kafka-streams/

KSQL

Tumbling Window



Fixed-duration, no overlap, subtype of a hopping window with “windowSize” and “advanceSize” the same

Example: You might want to find the average temperature recorded by multiple sensors in a region over a given time.

Source: https://www.confluent.io/blog/windowing-in-kafka-streams/

KSQL

Session Window

No fixed duration, based on duration of activity, triggered by events, separated by gaps of activity

Example: Ttracking user sessions on a website during a given time.

Source: https://www.confluent.io/blog/windowing-in-kafka-streams/

KSQL

Useful addition information

- Two deployment option for KSQL
  - Interactive mode (CLI, Rest API)
  - Headless mode (for production, only scripted with a file, CLI disabled)
- You can use Kafka Connect in KSQL (see here)
- ksql-datagen is a tool to generate synthetic test data (see here)
- Scalar function to transform, manipulate, or derive values within KSQL queries. (here) – e.g., to calculate the geo distance

KSQL

When should you use it?

- If you only need to write simple stream processings
- If your Developers don’t know Java but are familiar with SQL
- If you want to explore on Stream-Data
- (If you are using Confluent Cloud)

→ Otherwise we still suggest using Kafka Streams

Lab KSQL Advanced

Lab

Exercise: KSQL Advanced

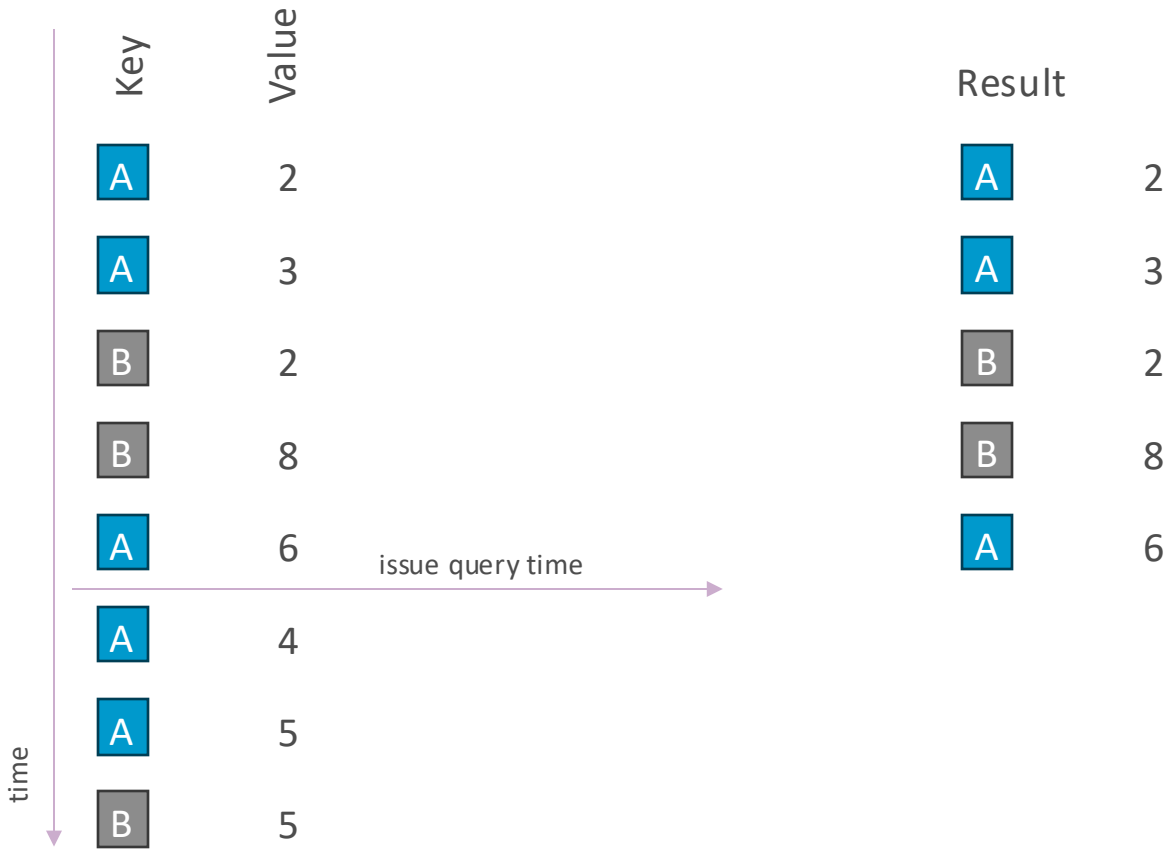
Follow the KSQL Exercise from

- https://github.com/Zuehlke/kafka-streaming-technology/blob/main/ksql-advanced.md

KSQL Reference

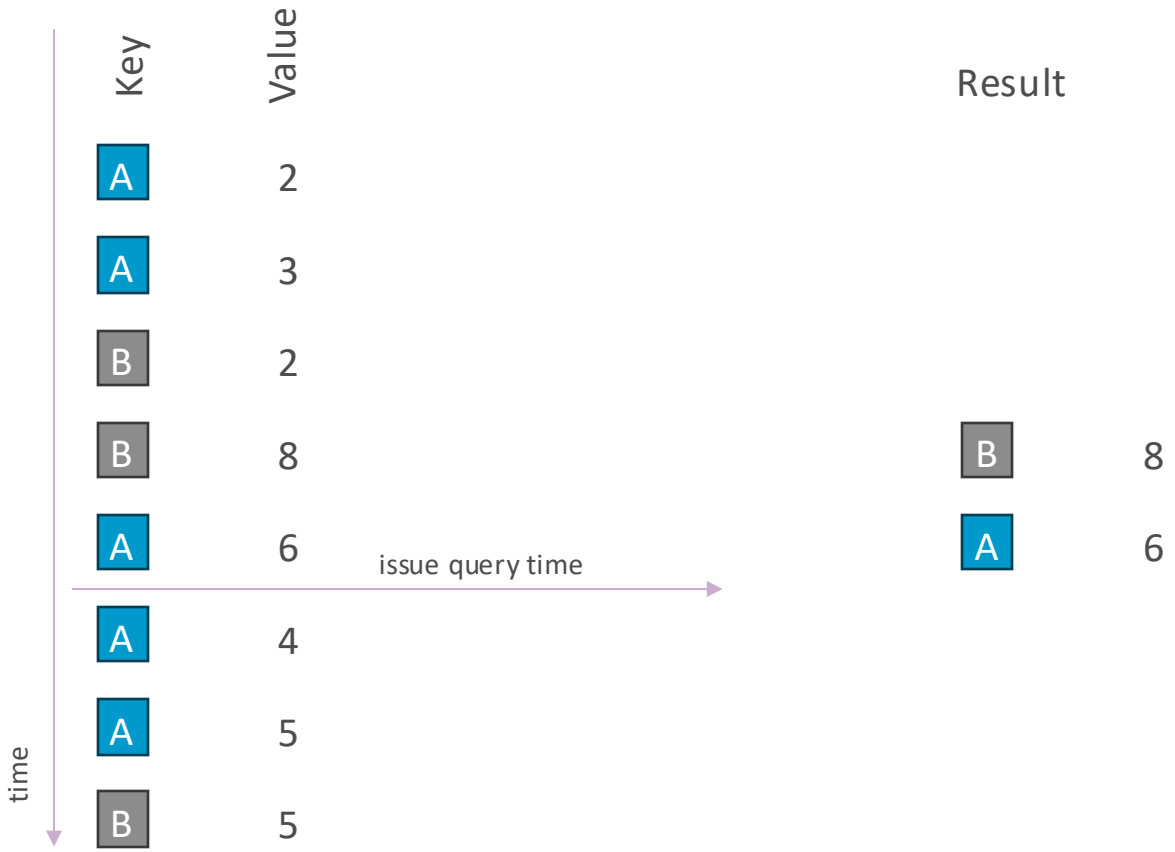
Streams: Pull Query

```
SELECT * FROM MY_STREAM;
```



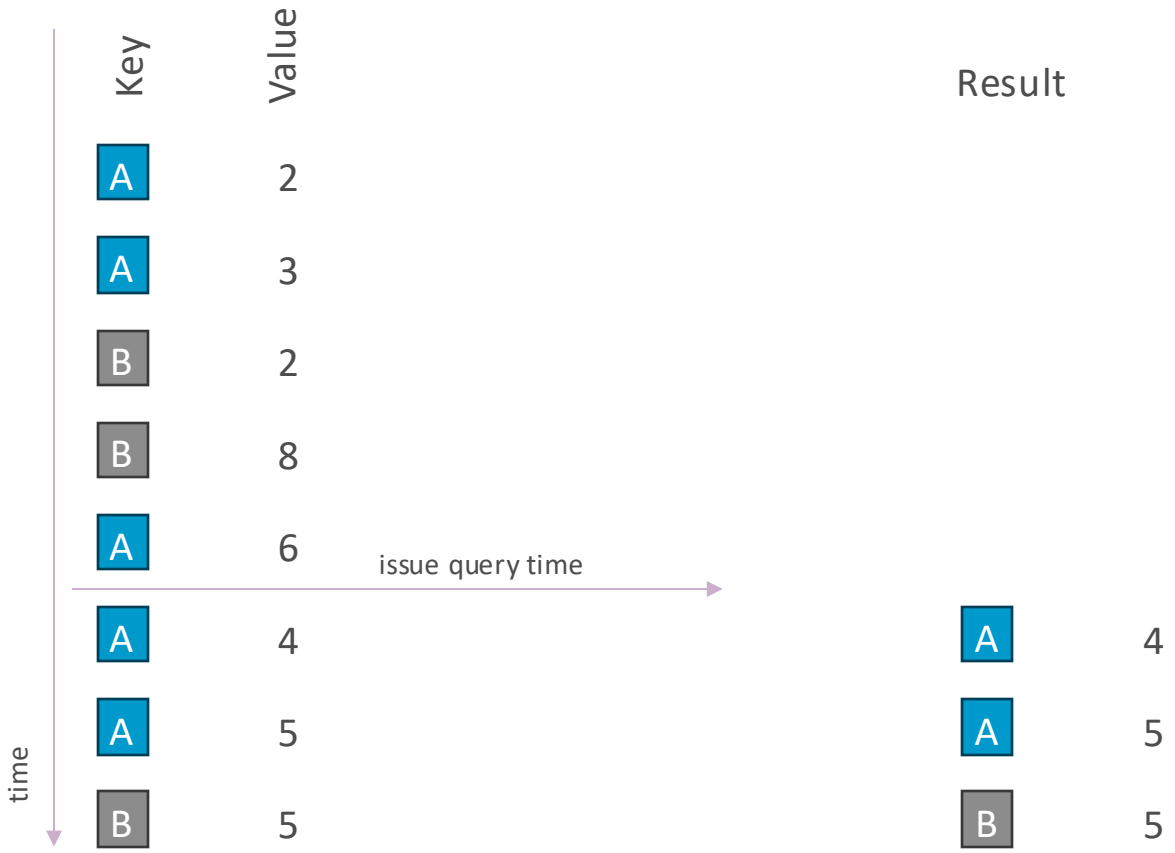
Tables: Pull Query

```
SELECT * FROM MY_TABLE;
```



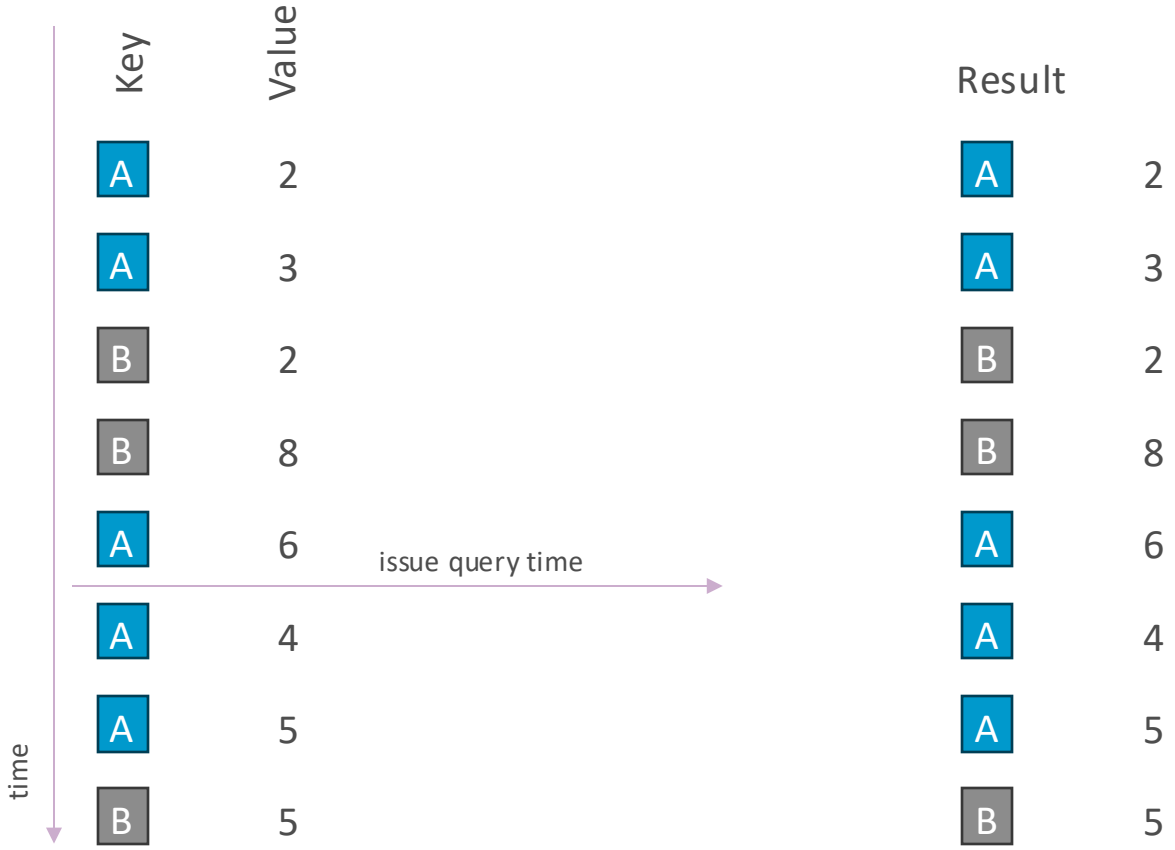
Streams: Push Query

```
SELECT * FROM MY_STREAM emit changes;
```



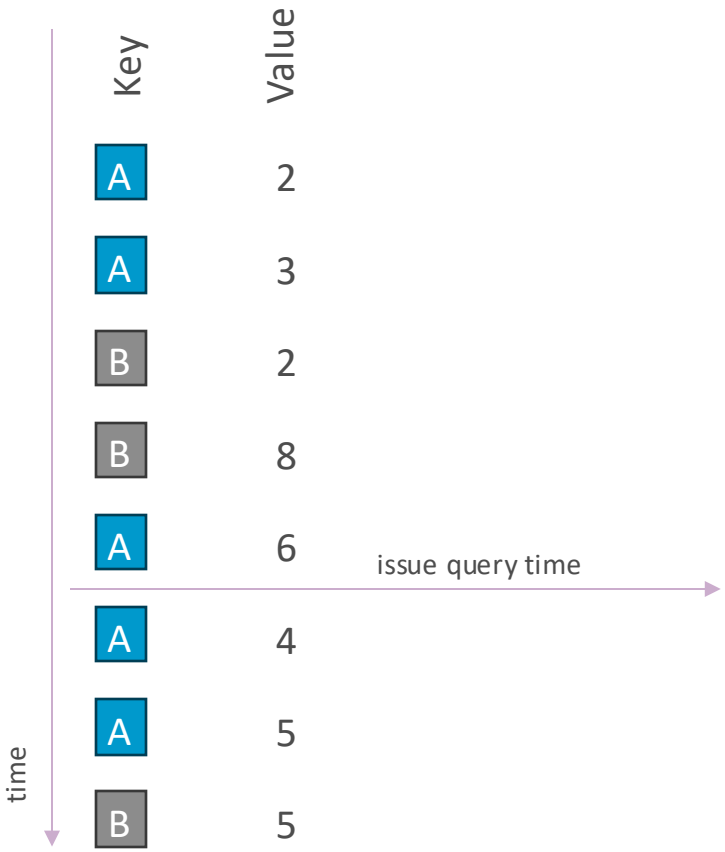
Tables: Push Query

```
SELECT * FROM MY_TABLE emit changes;
```



Streams: Push Query

```
SELECT * FROM MY_TABLE emit changes;
```



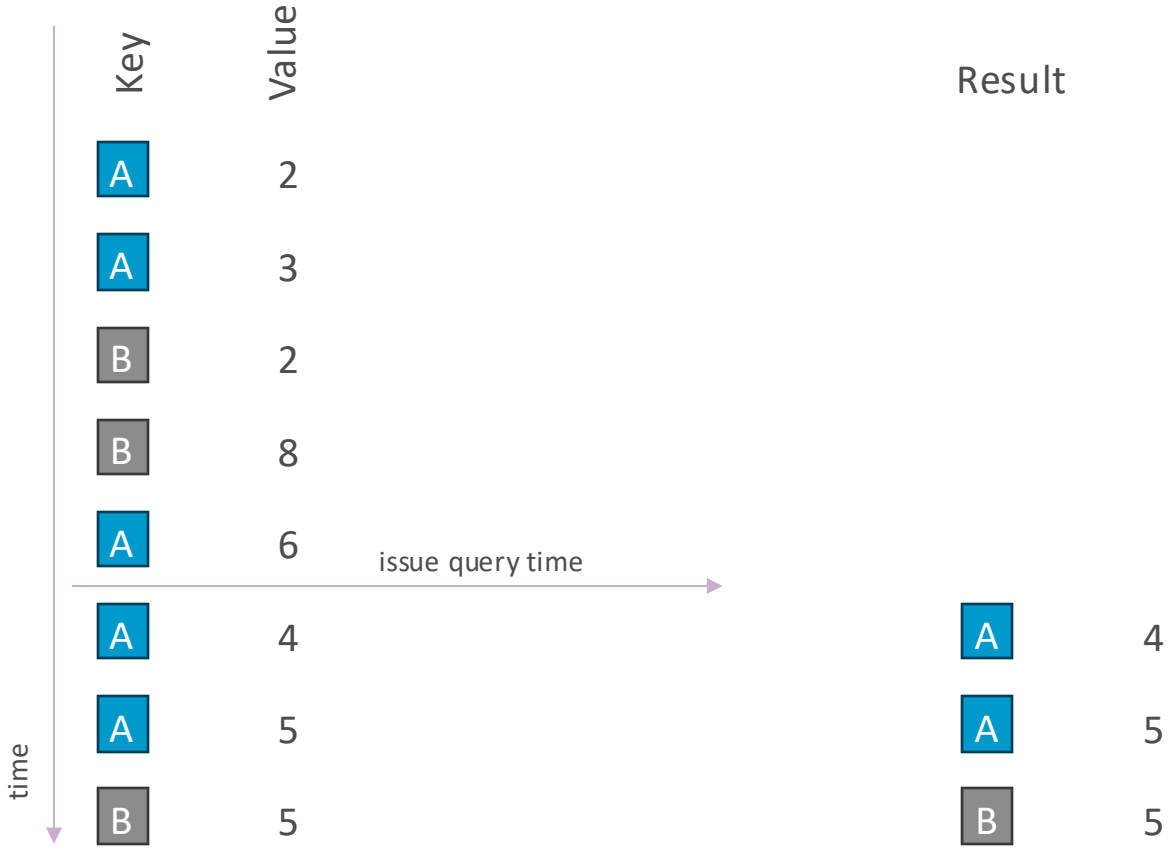
Result

A	2
A	3
B	2
B	8
A	6
A	4
A	5
B	5

```
SET 'auto.offset.reset' =
'earliest';
```

Tables: Push Query

```
SELECT * FROM MY_TABLE emit changes;
```




```
SET 'auto.offset.reset' =
'latest';
```

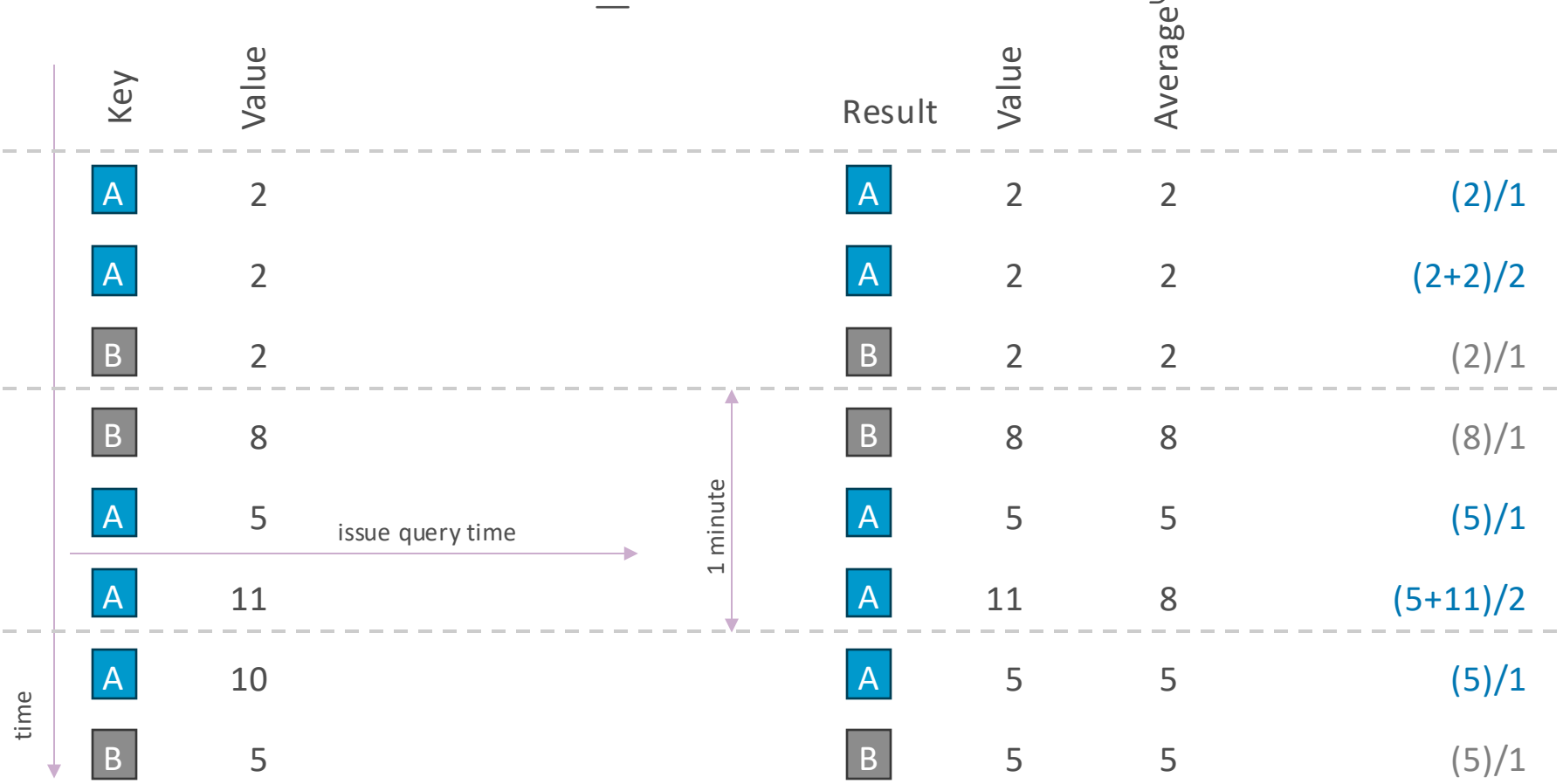
Tables as materialized stream

```
SELECT * FROM MY_TABLE emit changes;
```


```
CREATE TABLE MY_TABLE
  AS SELECT KEY,
  LATEST_BY_OFFSET
   (VALUE) AS VALUE,
  AVG(VALUE) AS
   AVERAGE
FROM MY_STREAM
GROUP BY KEY;
```

Tables as materialized stream, windowed

```
SELECT * FROM MY_TABLE emit changes;
```




```
CREATE TABLE MY_TABLE
  AS SELECT KEY,
  LATEST_BY_OFFSET
   (VALUE) AS VALUE,
  AVG(VALUE) AS
   AVERAGE
FROM MY_STREAM

WINDOW TUMBLING
(SIZE 1 MINUTE)

GROUP BY KEY;
```

Outlook

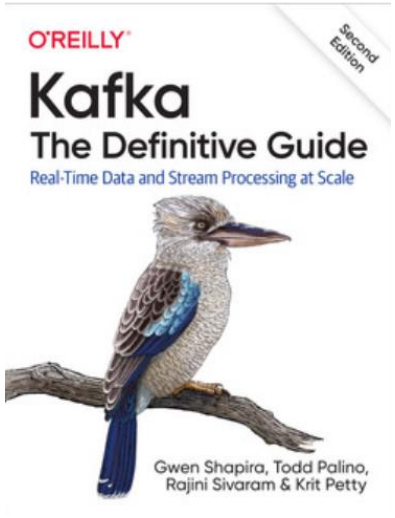
Kafka Releases

- New Release (approximately) every 4 months
- Community Driven Kafka Improvements Proposals (KIP)

- Newest Release 3.8 from July 29, 2024
  - JBOD (Just a bunch of disks) is not anymore considered as early access feature
    - hovever, Zookeeper remains
  - configurable level of compression
  - docker image for GraalVM based Native Broker
  - https://www.confluent.io/blog/introducing-apache-kafka-3-8/
  - https://downloads.apache.org/kafka/3.8.0/RELEASE_NOTES.html

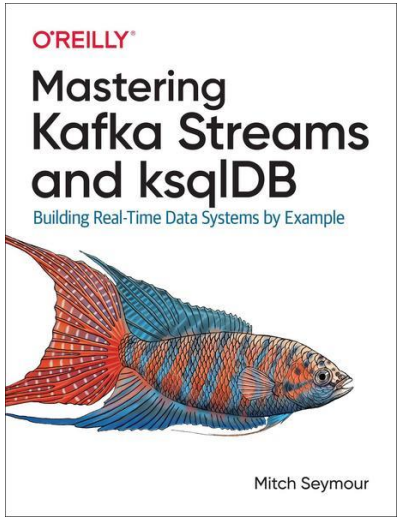
Recommendations

Books

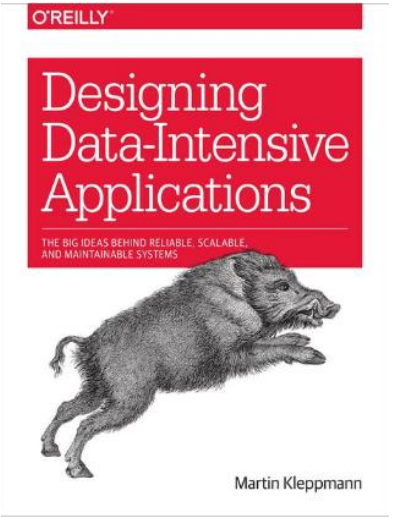


Overall Kafka book

Note: Please choose the 2nd edition



Dig deeper in streams



Overall book about data-intensive application

Note: You can click on the books

Recommendations

Trainings & Events

- Confluent Developer Resources: https://developer.confluent.io/

- Kafka Summit 2024: https://www.kafka-summit.org/
  - Austin Convention Center, yesterday and today

Recommendations

Certification

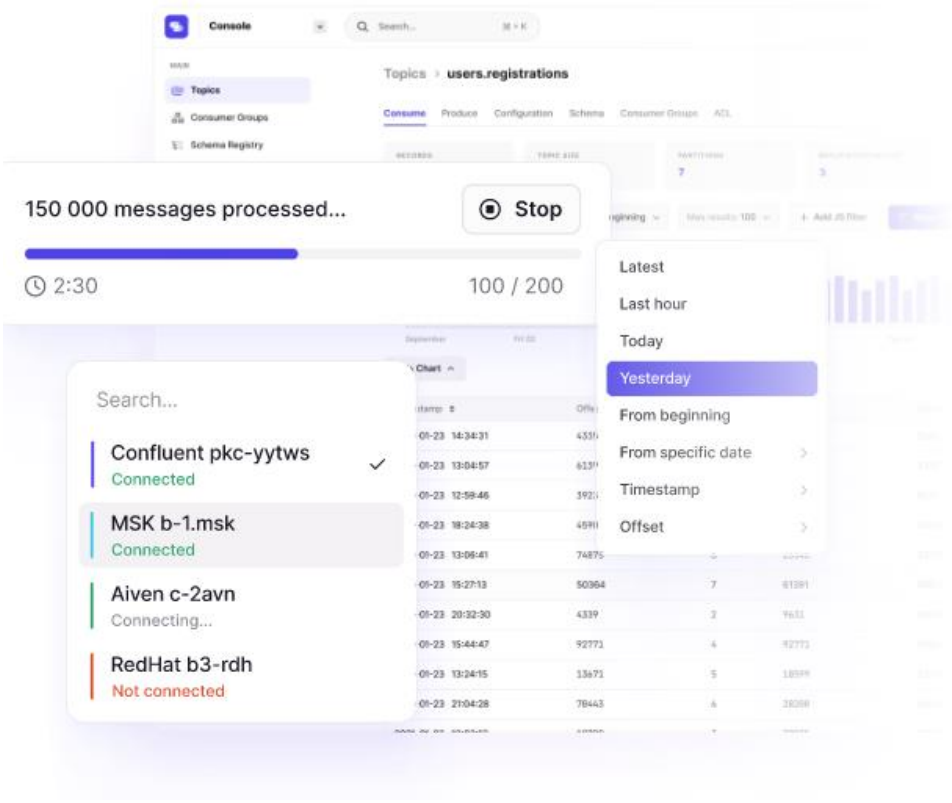
- Confluent Certified Developer for Apache Kafka (CCDAK)
  - Developer and Architects
  - Confluent / Apache Kafka with focus on platform and developing applications
  - 55 multiple choice questions in **90 minutes**

- Confluent Certified Administrator for Apache Kafka (CCAAK)
  - Cluster Administrators
  - Confluent / Apache Kafka with focus on platform and managing clusters (configuration, deployment, monitoring)
  - 40 multiple choice questions in **90 minutes**

https://www.confluent.io/certification/#get-certified

Recommendations

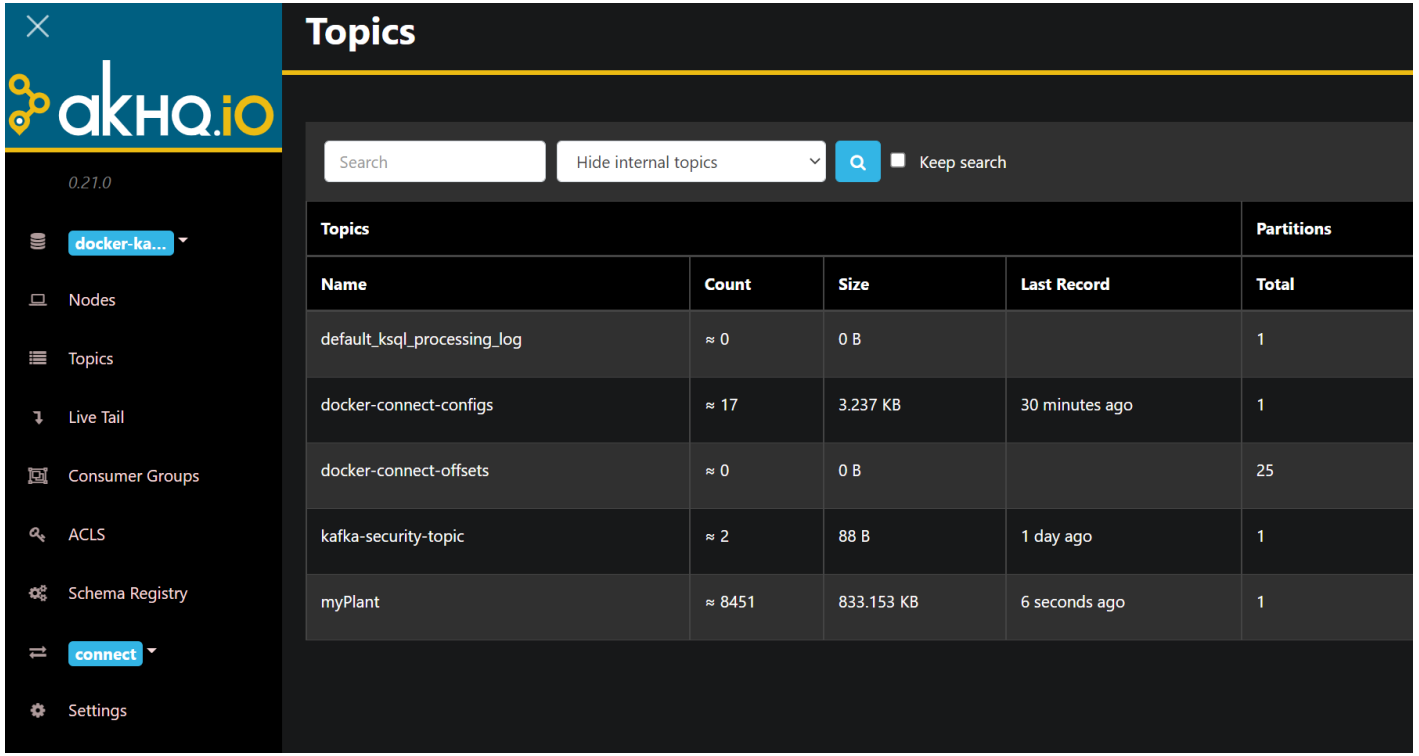
Tools: Conduktor



https://www.conduktor.io

Recommendations

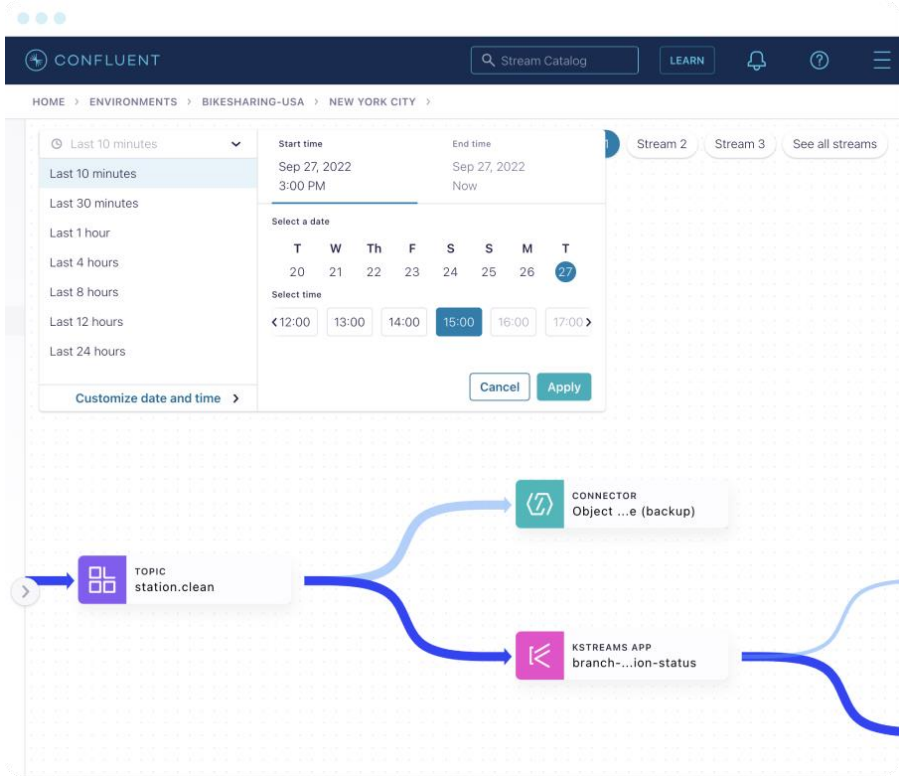
Tools: akHQ



https://akhq.io/

Confluent Cloud

- Fully Managed Kafka Services
- Multi Cloud / Hybrid:
  - AWS
  - Azure
  - GCP
- Confluent Additions (not available in MSK)
  - ksqlDb
  - Stream Designer
  - Cluster Linking
  - Encryption
- https://confluent.cloud/



Example Producer and Consumer setup (for your reference)

Producer & Consumer

Spring Boot Kafka Producer

```
@Service
@Slf4j
@RequiredArgsConstructor
public class MotorProducer {

    private final MotorConfig motorConfig;
    private final KafkaTemplate<String, SensorMeasurement> producer;

    @Scheduled(fixedDelayString = "${random.int(${motor.max-interval-ms})}")
    public void produceMeasurement() {
        String topic = motorConfig.getPlantId();
        String key = motorConfig.getMotorId();
        SensorMeasurement value = createMeasurement();
        log.info("Sending record to topic '{}' with key '{}' and value: {}", topic, key, value);
        producer.send(topic, key, value);
    }

    private SensorMeasurement createMeasurement() {
        return new SensorMeasurement(motorConfig.getMotorId(), System.currentTimeMillis(), motorConfig.getRandomMotorState());
    }
}
```

Producer & Consumer

Spring Boot Kafka Consumer

```
@Service
@Slf4j
public class MotorConsumer {

    private final String consumerGroup = "${spring.application.name}";
    private final String topic = "plant01";

    @KafkaListener(groupId = consumerGroup, topics = topic)
    public void consumeMeasurement(ConsumerRecord<String, SensorMeasurement> record) {
        log.info("Received record '{}'", record);
        doCrazyStuff(record);
    }

    private void doCrazyStuff(ConsumerRecord<String, SensorMeasurement> record) {
//      var key = record.key();
//      var measurement = record.value();
//      calculate stuff, update database - maybe in idempotent manner
    }
}
```

Producer & Consumer

Spring Boot Kafka Configuration

```
spring:
  application:
    name: motor
  kafka:
    bootstrap-servers: localhost:9092
    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer
      # your non-default producer configuration here
    consumer:
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: io.confluent.kafka.serializers.KafkaAvroDeserializer
      # your non-default consumer configuration here
    properties:
      schema.registry.url: http://localhost:8081
      specific.avro.reader: true
```

Producer & Consumer

More Spring Boot Configuration

Producer:

https://kafka.apache.org/documentation/#producerconfigs

```
spring.kafka.producer.acks (Number of acknowledgments the producer
spring.kafka.producer.batch-size (Default batch size)
spring.kafka.producer.bootstrap-servers (Comma-delimited list of h
spring.kafka.producer.buffer-memory (Total memory size the produce
spring.kafka.producer.client-id (ID to pass to the server when mak
spring.kafka.producer.compression-type (Compression type for all d
spring.kafka.producer.properties (Additional producer-specific pro
spring.kafka.producer.retries (When greater than zero, enables ret
spring.kafka.producer.security.protocol (Security protocol used to
spring.kafka.producer.ssl.key-password (Password of the private ke
spring.kafka.producer.ssl.key-store-certificate-chain (Certificate
spring.kafka.producer.ssl.key-store-key (Private key in PEM format
spring.kafka.producer.ssl.key-store-location (Location of the key
spring.kafka.producer.ssl.key-store-password (Store password for t
spring.kafka.producer.ssl.key-store-type (Type of the key store)
spring.kafka.producer.ssl.protocol (SSL protocol to use)
spring.kafka.producer.ssl.trust-store-certificates (Trusted certif
spring.kafka.producer.ssl.trust-store-location (Location of the tr
spring.kafka.producer.ssl.trust-store-password (Store password for
spring.kafka.producer.ssl.trust-store-type (Type of the trust stor
spring.kafka.producer.transaction-id-prefix (When non empty, enabl
```

Consumer:

https://kafka.apache.org/documentation/#consumerconfigs

```
spring.kafka.consumer.group-id (Unique string that identifies the consumer grou
spring.kafka.consumer.auto-commit-interval (Frequency with which the consumer o
spring.kafka.consumer.auto-offset-reset (What to do when there is no initial of
spring.kafka.consumer.bootstrap-servers (Comma-delimited list of host:port pair
spring.kafka.consumer.client-id (ID to pass to the server when making requests)
spring.kafka.consumer.enable-auto-commit (Whether the consumer's offset is peri
spring.kafka.consumer.fetch-max-wait (Maximum amount of time the server blocks
spring.kafka.consumer.fetch-min-size (Minimum amount of data the server should
spring.kafka.consumer.heartbeat-interval (Expected time between heartbeats to t
spring.kafka.consumer.isolation-level-read-uncommitted (Isolation level for rea
spring.kafka.consumer.max-poll-records (Maximum number of records returned in a
spring.kafka.consumer.properties (Additional consumer-specific properties used
spring.kafka.consumer.security.protocol (Security protocol used to communicate
spring.kafka.consumer.ssl.key-password (Password of the private key in either k
spring.kafka.consumer.ssl.key-store-certificate-chain (Certificate chain in PEM
spring.kafka.consumer.ssl.key-store-key (Private key in PEM format with PKCS#8
spring.kafka.consumer.ssl.key-store-location (Location of the key store file)
spring.kafka.consumer.ssl.key-store-password (Store password for the key store
spring.kafka.consumer.ssl.key-store-type (Type of the key store)
spring.kafka.consumer.ssl.protocol (SSL protocol to use)
spring.kafka.consumer.ssl.trust-store-certificates (Trusted certificates in PEM
spring.kafka.consumer.ssl.trust-store-location (Location of the trust store fil
spring.kafka.consumer.ssl.trust-store-password (Store password for the trust st
spring.kafka.consumer.ssl.trust-store-type (Type of the trust store)
```