ClickHouse

CONFLUENT

Meetup

WORQ TTDI

Kuala Lumpur, Malaysia

December 11, 2024 at 5:30 PM MYT

ClickHouse + Kafka Integration

Derek Chia

Principal Support Engineer @ ClickHouse

derekchia

derekchia

derekchia

- Favorite Comedian - Dr. Jason Leong 😁
- Born in Malaysia!

Karthikayan Muthuramalingam

Solutions Engineer, Confluent

karthikayan-muthuramalingam

Cruzkn

- A tech adventure, from code to commerce and beyond.
- A techie by day, a foodie and gamer by night.

Agenda

01 Introduction

02 Apache Kafka 101

03 Integrating Kafka with ClickHouse

04 Recent improvements to Kafka Support

05 Demo

06 Questions

©2024 CLICKHOUSE INC., CONFIDENTIAL & PROPRIETARY

Introduction : ClickHouse

What is ClickHouse?

Open source

Developed since
2009 - OSS 2016
38k+ Github stars
1.6k+ contributors
600+ releases

column-oriented

Best for aggregations
Files per column
Sorting and indexing
Background merges

distributed

Replication
Sharding
Multi-master
Cross-region

OLAP database

Analytics use cases
Aggregations
Visualizations
Mostly immutable data

Key Features

Some of the cool things ClickHouse can do

1 Speaks SQL

Most SQL-compatible UIs, editors, applications, frameworks will just work!

2 Lots of writes

Up to several million writes per second - per server.

3 Distributed

Replicated and sharded, largest known cluster is 4000 servers.

4 Highly efficient storage

Lots of encoding and compression options - e.g. 20x from uncompressed CSV.

5 Very fast queries

Scan and process even billions of rows per second and use vectorized query execution.

6 Joins and lookups

Allows separating fact and dimension tables in a star schema.

What problem does ClickHouse solve?

Real-time Dashboards

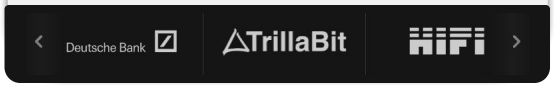
Powering interactive applications that analyze and aggregate large amounts of data on the fly. Often, these are customer facing.

Real-time Analytics

Managing continuous streams of events where real-time analysis is key.

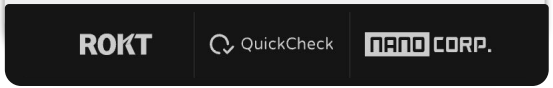
Business Intelligence

Interactively slicing and dicing data for analysis, reporting, and building internal applications.



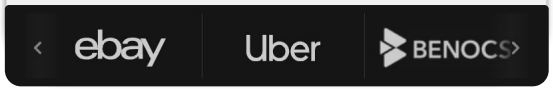
Data Warehouse Speed Layer

Offloading analytical workloads from a Data Warehouse to maximize resource efficiency and optimize for blazing speed.



Logging & Metrics

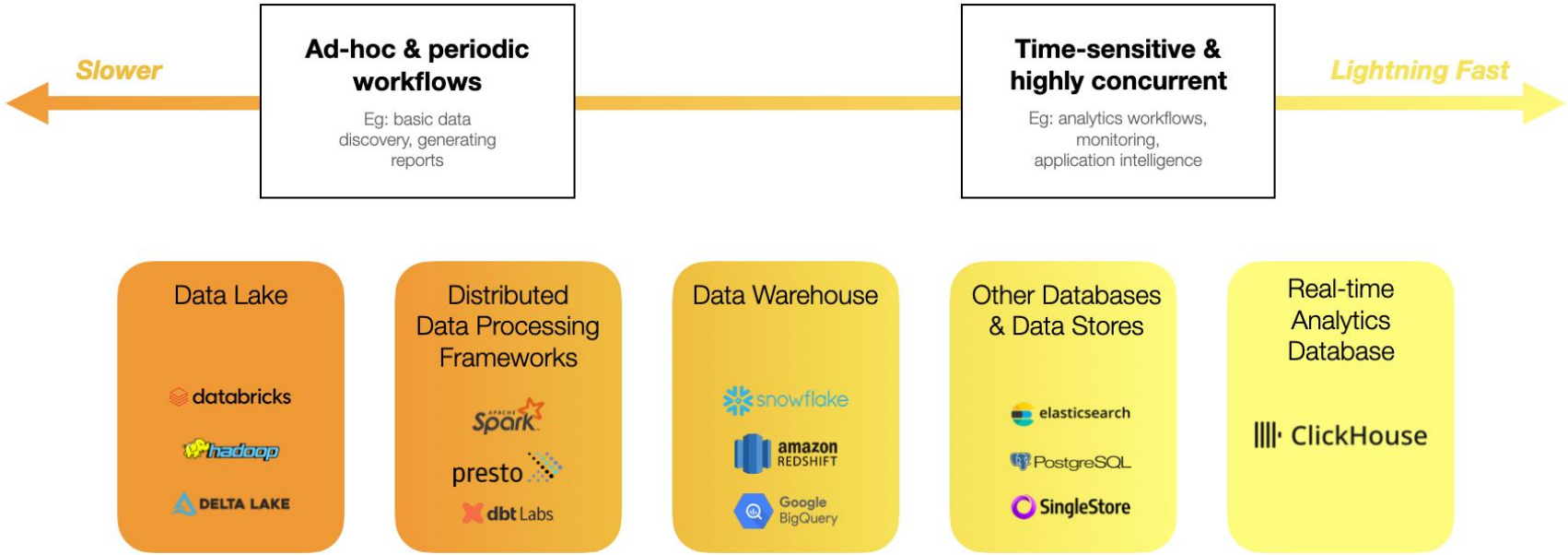
Used to monitor logs, metrics, and traces, and to detect anomalies, fraud, network or infrastructure issues, and more.



ML & Data Science

Leveraged to train models over massive data sets, and for fast and efficient vector queries.

What makes ClickHouse unique?



What makes ClickHouse unique?

1 Lightning fast queries

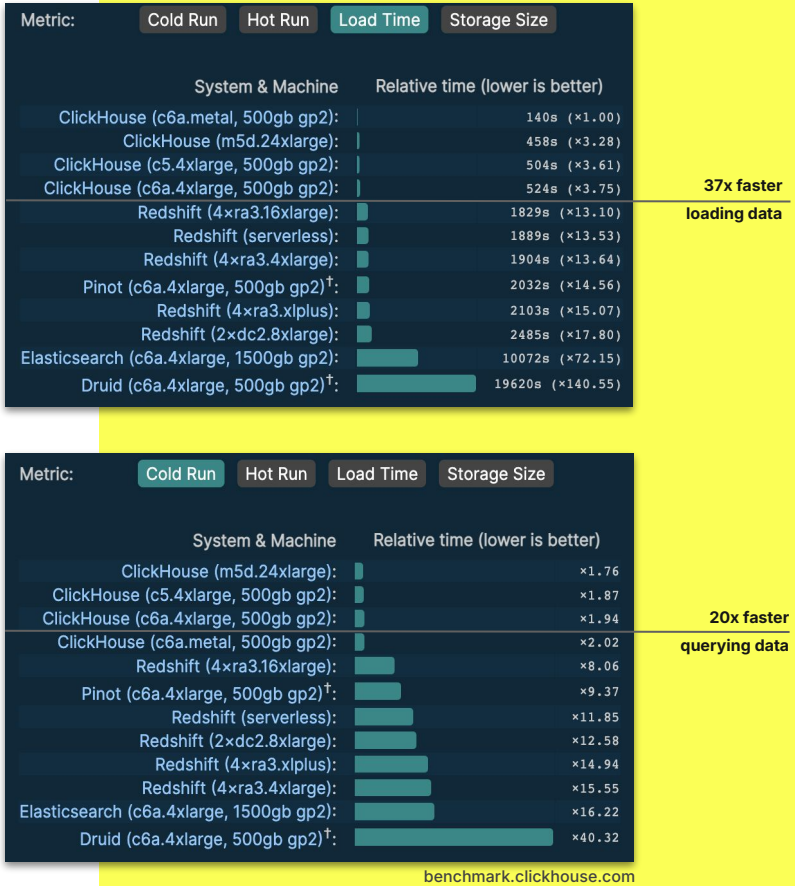
Analytical queries, aggregations, and computations over large datasets is lightning fast

2 Highly resource efficient

Industry-leading data compression - 10-100x storage efficiency over alternatives

3 Ease of use

Self-service data onboarding from a wide range of sources (e.g. S3, Delta Lake, Iceberg, Hudi), analyst-friendly standard SQL syntax.



Use cases

Real-time Analytics

Power interactive applications and dashboards that analyze and aggregate large amounts of data on the fly. Run complex internal analytics in ms, not mins or hrs.

Disney+ · CLOUDFLARE · Microsoft · lyft · Contentsquare · highlight.io

Logs, events, traces

Monitor with confidence your logs, events, and traces. Detect anomalies, fraud, network or infrastructure issues, and more.

Darwinium · ebay · BENOCS · resmo · zomato · runreveal

Business intelligence

Interactively slice and dice your data for analysis, reporting, and building internal applications. Evaluate user behaviors, ad and media perf, market dynamics, and more.

ROKT · Deutsche Bank · QuickCheck · NANO CORP. · TrillaBit · HIFI

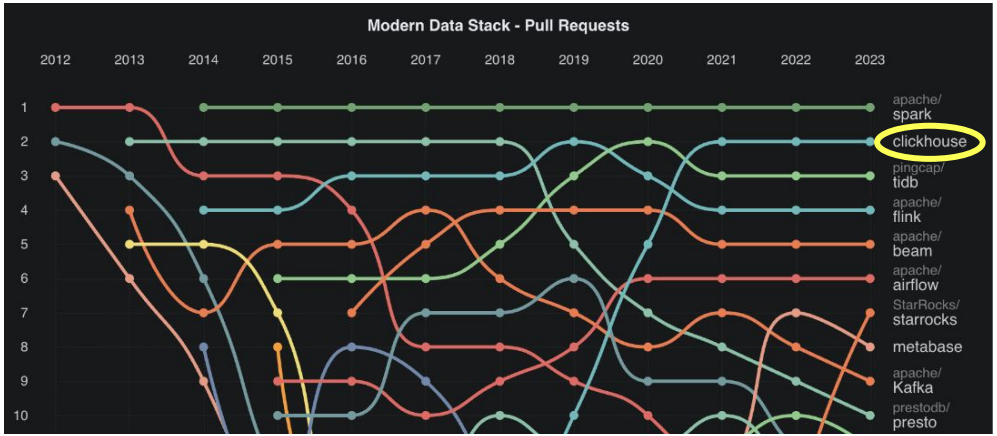
ML and Gen AI

Execute fast and efficient vector search. Plug-and-play Generative AI models from any provider. Use lightning-fast aggregations to power model training at petabyte scale.

denic · DeepL · ADMIXER · ensemble

ClickHouse Open Source

- ✔ 38k Stars
- ✔ 7k Forks
- ✔ 1.6k Contributors
- ✔ 160k+ Commits
- ✔ 200k+ Active community members



ClickHouse Cloud

aws

- ✔ Fast, scalable, and reliable
- ✔ Flexible, feature-rich, and easy to use
- ✔ Process billions of queries daily

NETFLIX vimeo Spotify Microsoft Deutsche Bank NGINX CLOUDFLARE

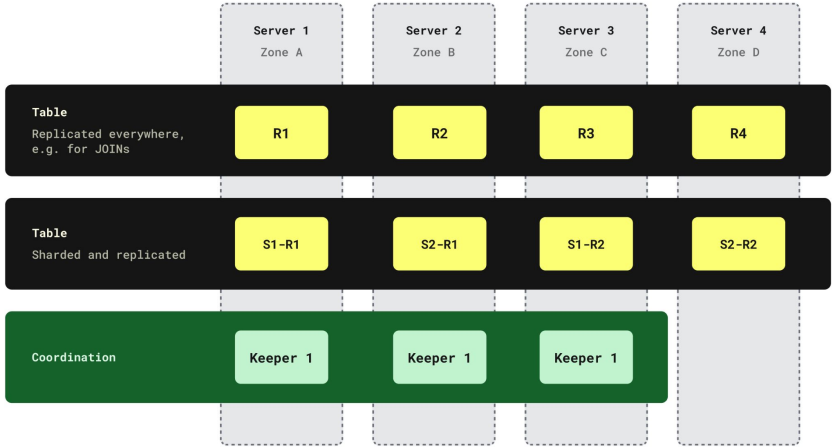
cisco COMCAST Walmart ROKT GitLab ebay IBM Contentsquare

ClickHouse

Self-managed

- ✔ Open-source
- ✔ Flexible architecture
- ✔ Efficient and robust
- ✔ Support contracts available

Sample self-managed architecture

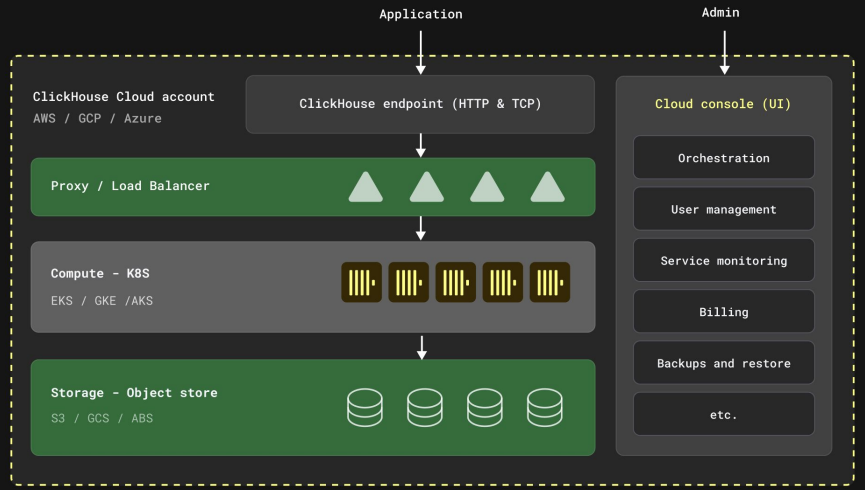


ClickHouse

Cloud

✔ Easy to use	Managed for you
✔ Feature-rich	Cloud-first features & tooling
✔ Fast	Automatically maximizes performance/efficiency
✔ Scalable	Scale seamlessly
✔ Reliable	Ensure reliability
✔ PAYG	SaaS usage and capacity based pricing

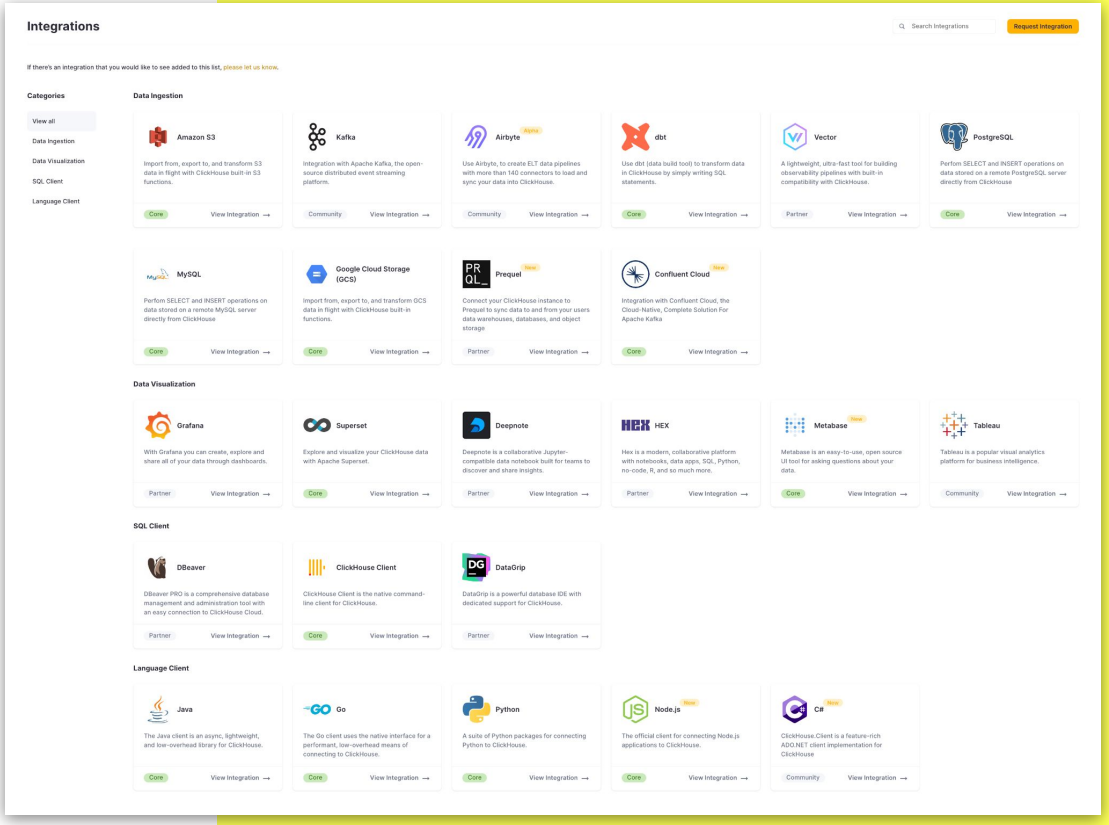
ClickHouse Cloud architecture



ClickHouse

Integrations made easy

- Leverage an ever-growing, curated list of core, partner, and community integrations vetted to work with ClickHouse Cloud
- Add **data sources**, data **visualization**, and other ecosystem tools with a few clicks
- Enterprise grade security enforced on all integrations under our control

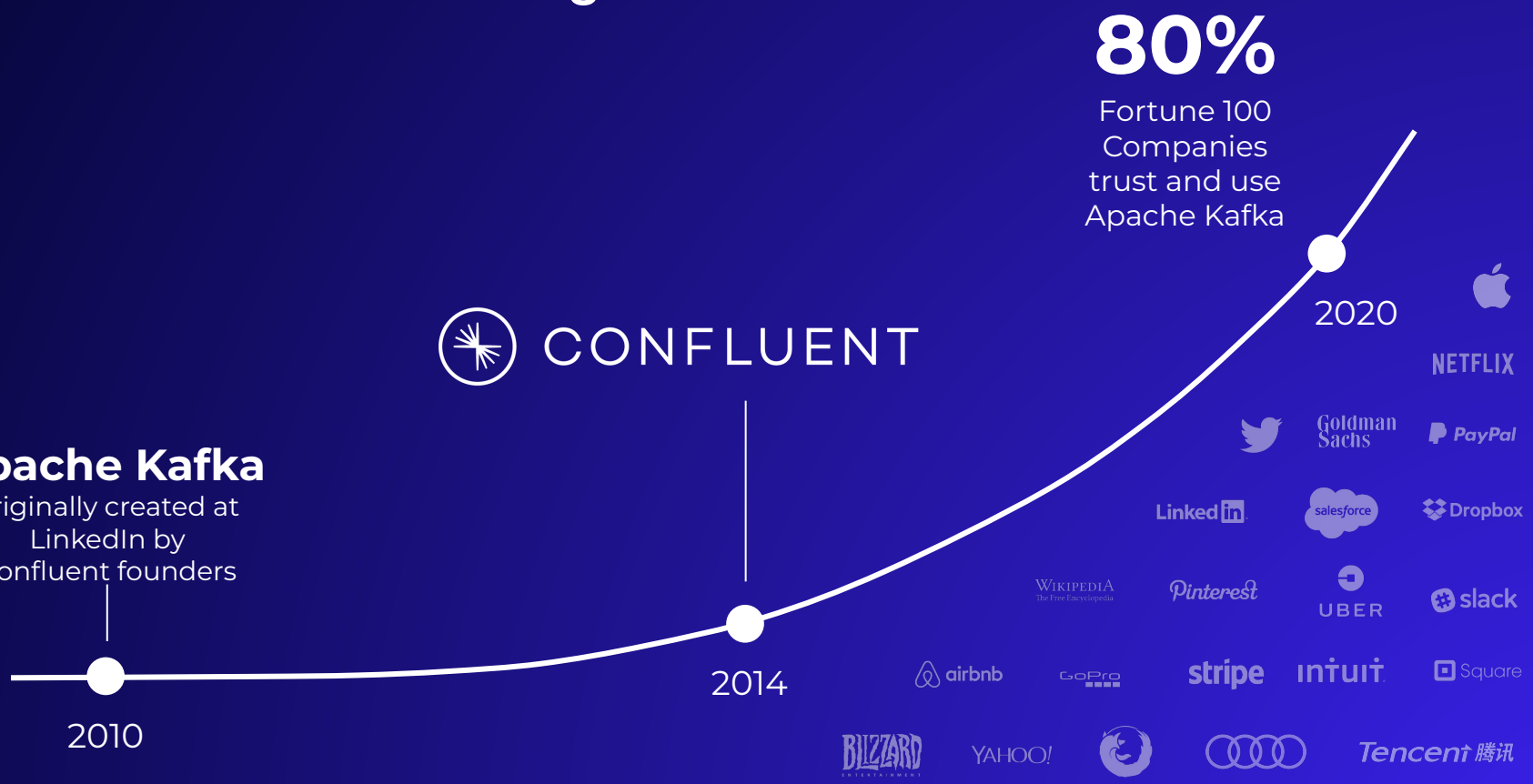


ClickHouse

Introduction : Confluent

Introduction to Confluent

The Rise of Event Streaming



Confluent Overview

Jay Kreps
CEO

Neha Narkhede

Jun Rao
VP Engineering

Founded by the original creators of **Apache Kafka**

Founded **September 2014**

Technology developed while **at LinkedIn**

BENCHMARK

Index Ventures

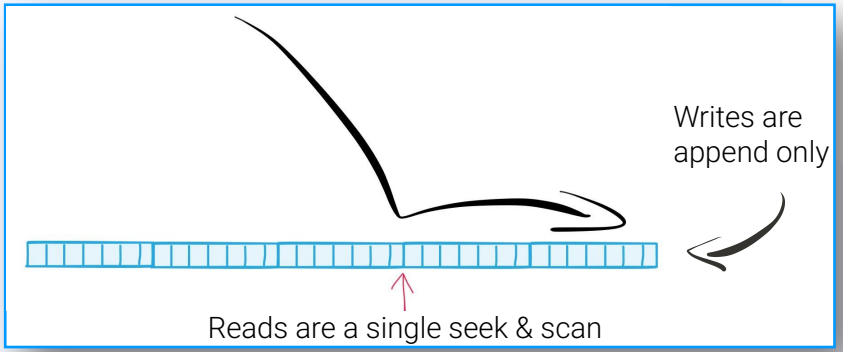
SEQUOIA

Introducing Apache Kafka®

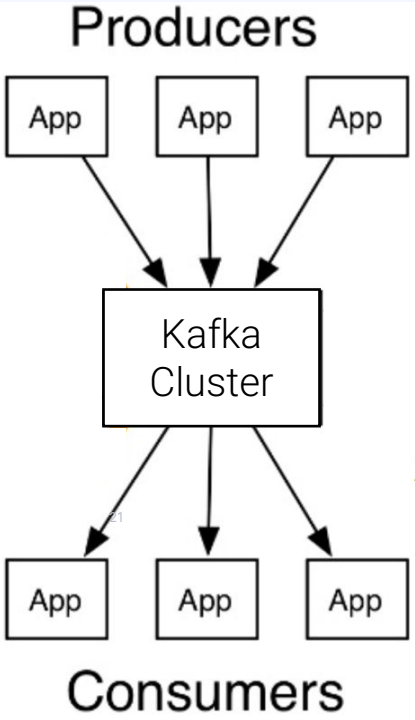
Apache Kafka - Publish & Subscribe Reimagined

Kafka

A **Distributed Commit Log**. Publish and subscribe to streams of records. Highly scalable, high throughput. Supports transactions. Persisted data.



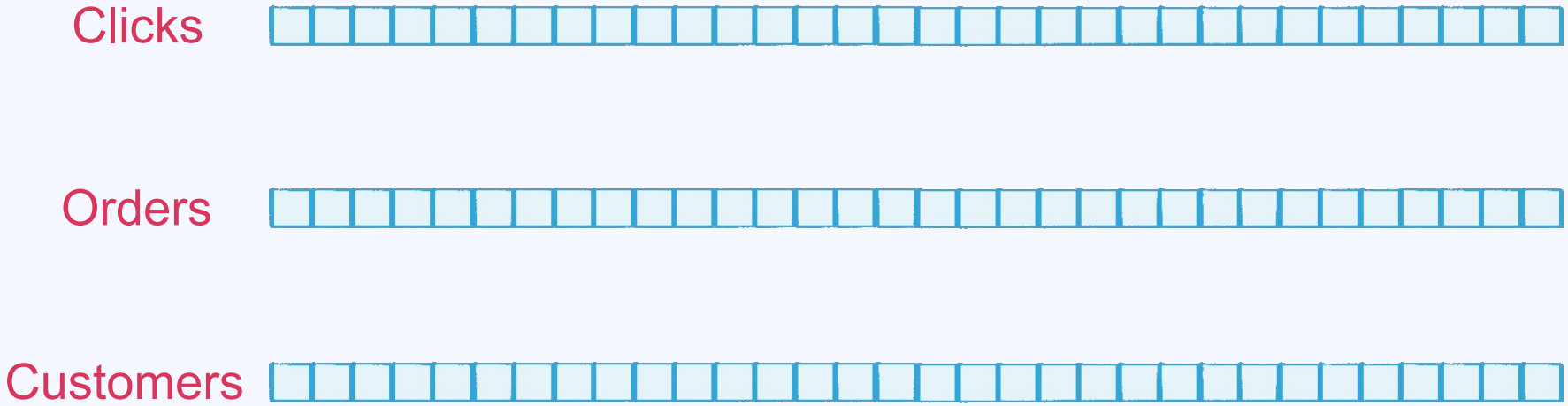
1. Publish Stream Events



2. Store

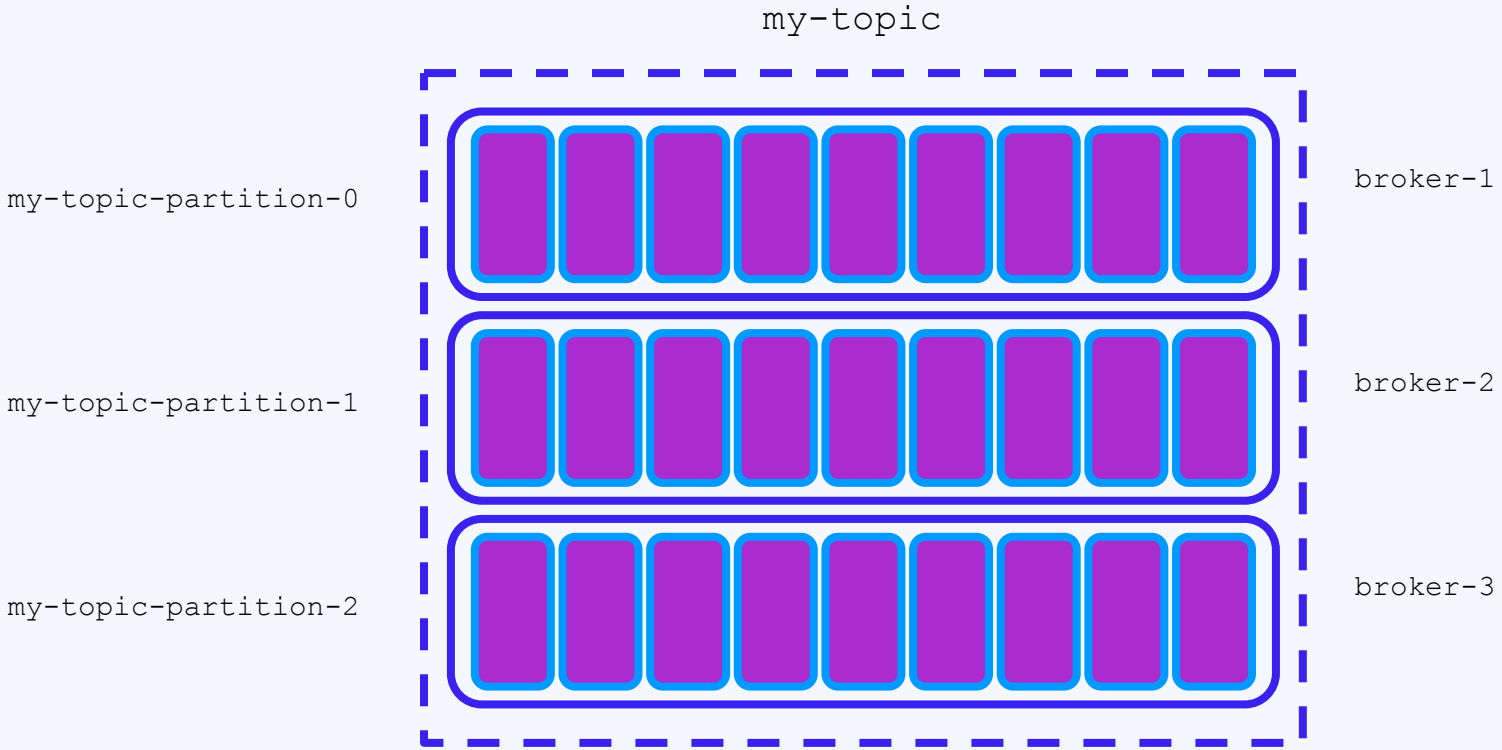
3. Process & Consume

Topics

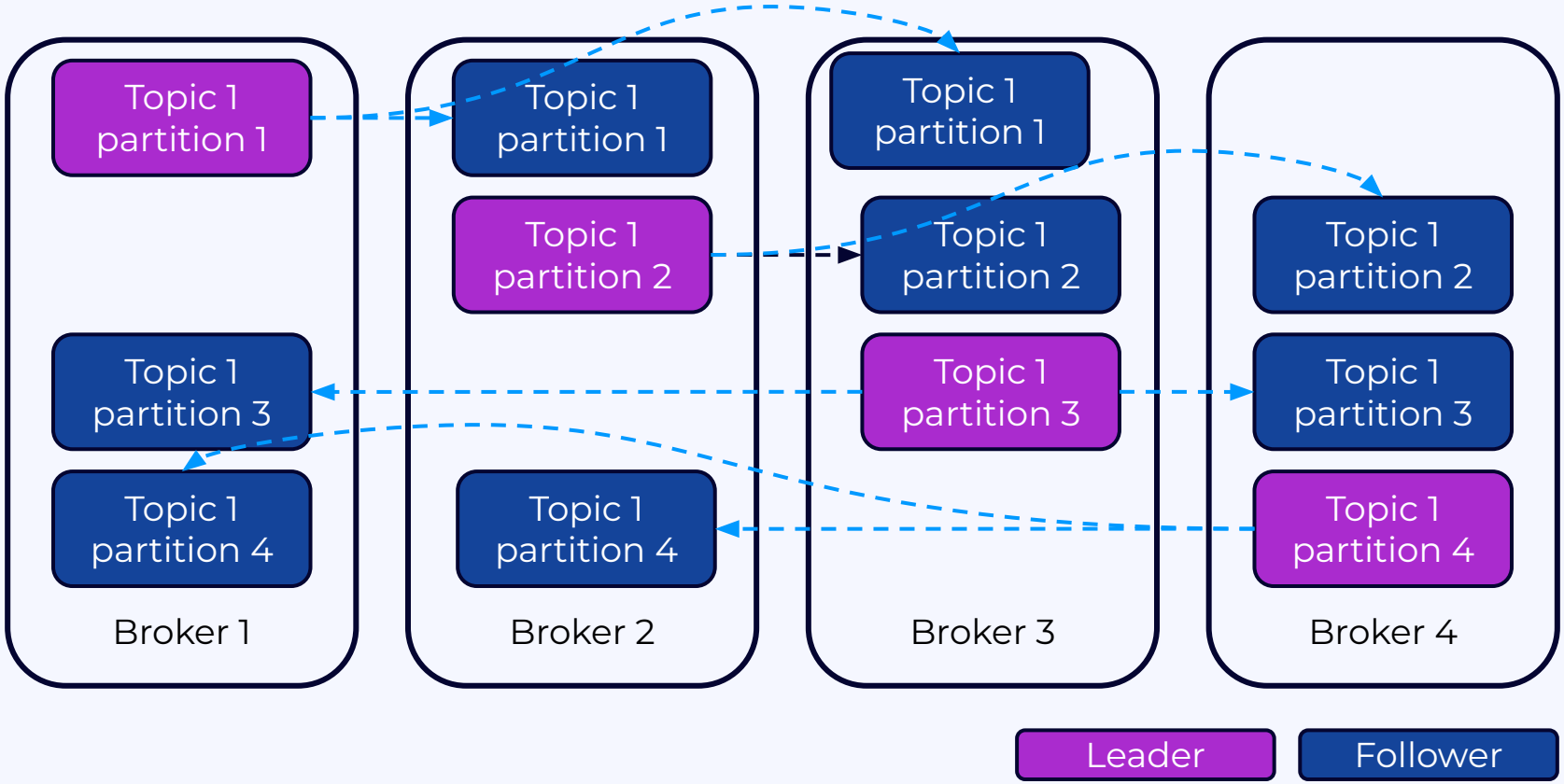


Topics are similar in concept
to tables in a database

Kafka Topics

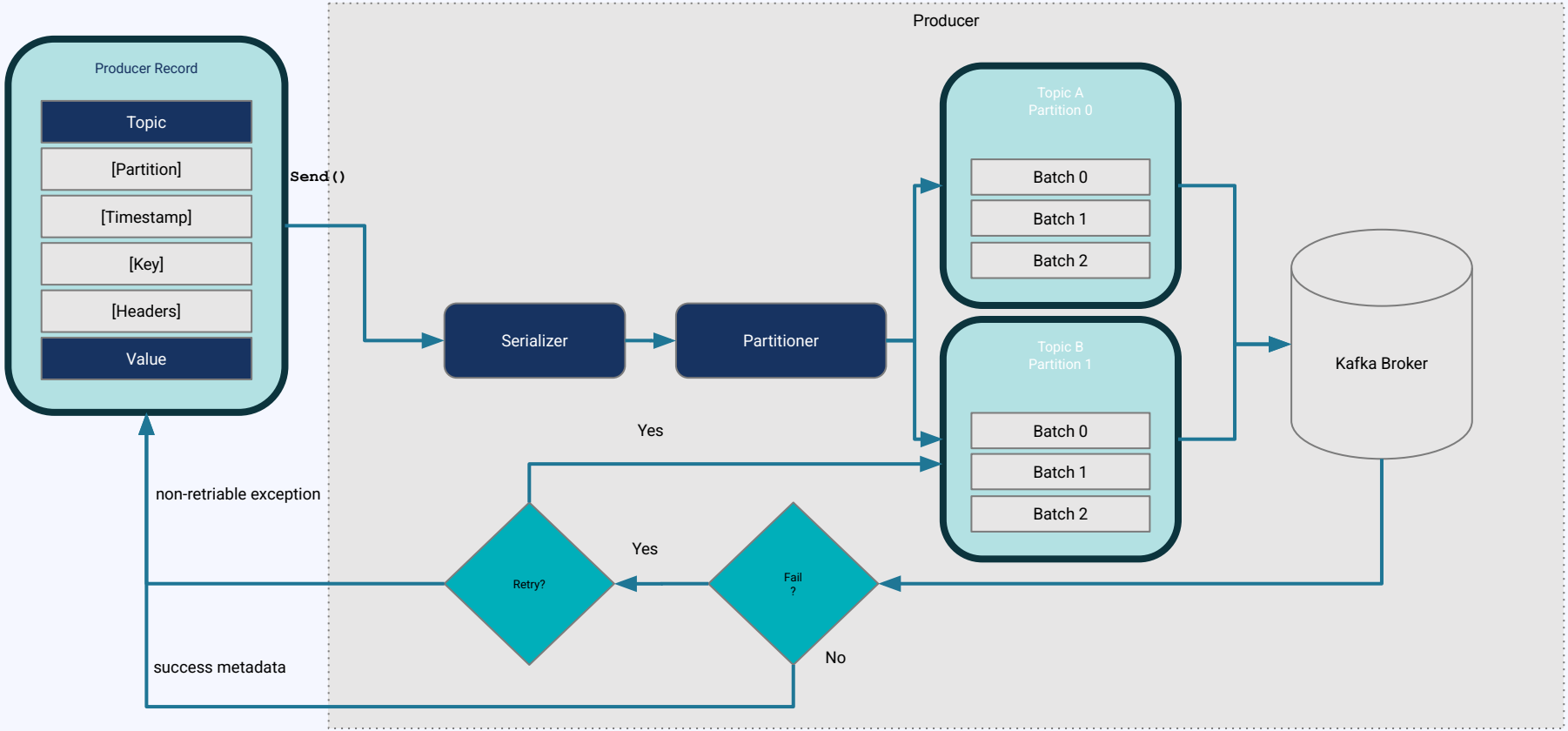


How Does Kafka Provide Resilience?



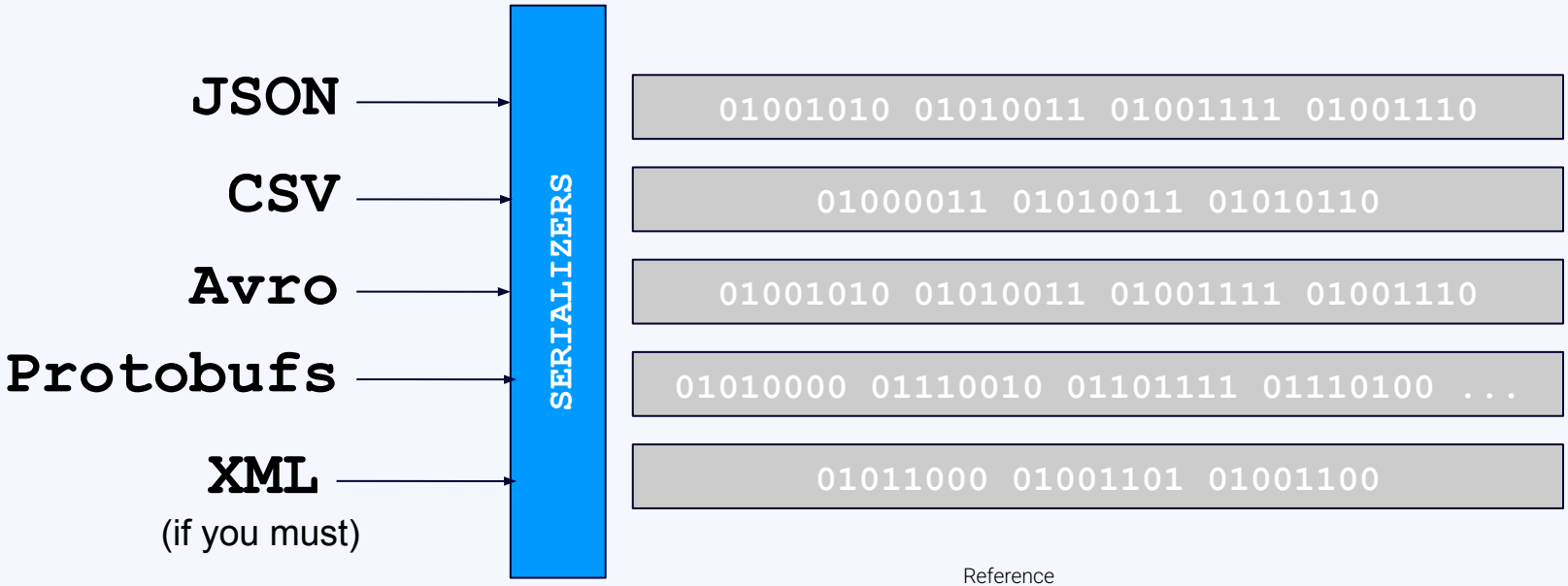
Producing to Kafka

What happens inside a producer?



The Serializer

Kafka doesn't care about what you send to it as long as it's been converted to a byte stream beforehand.



Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

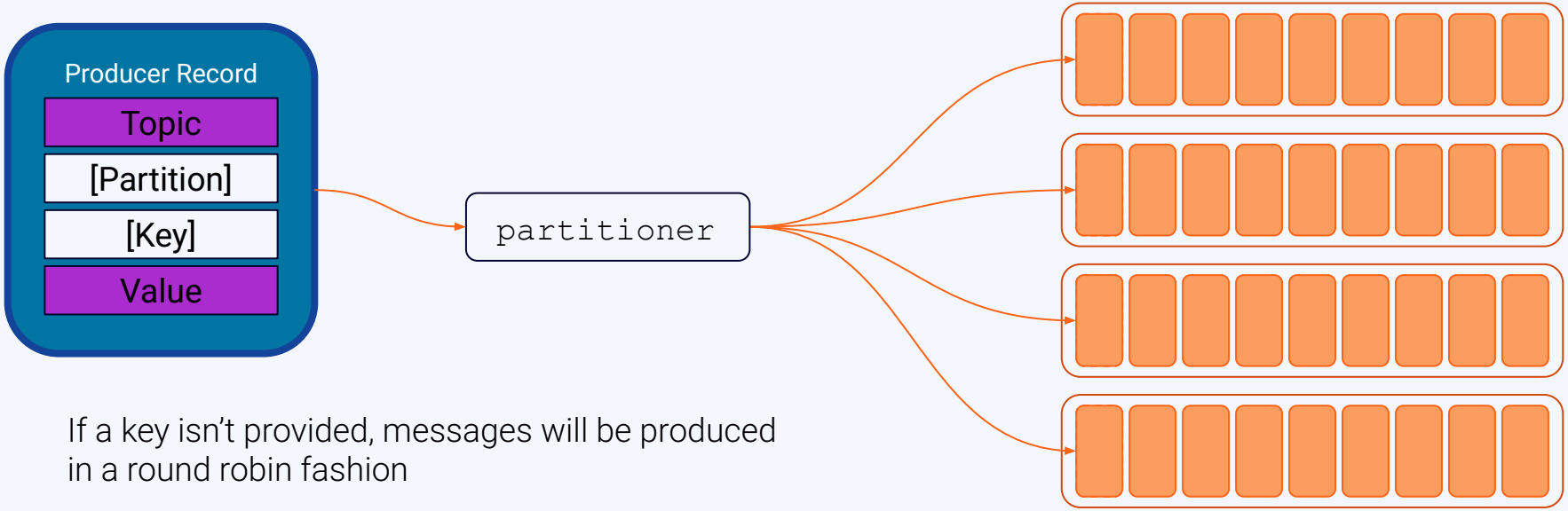
The Serializer

```
private Properties kafkaProps = new Properties();

kafkaProps.put("bootstrap.servers", "broker1:9092,broker2:9092");
kafkaProps.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
kafkaProps.put("schema.registry.url", "https://schema-registry:8083");

producer = new KafkaProducer<String, SpecificRecord>(kafkaProps);
```

Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

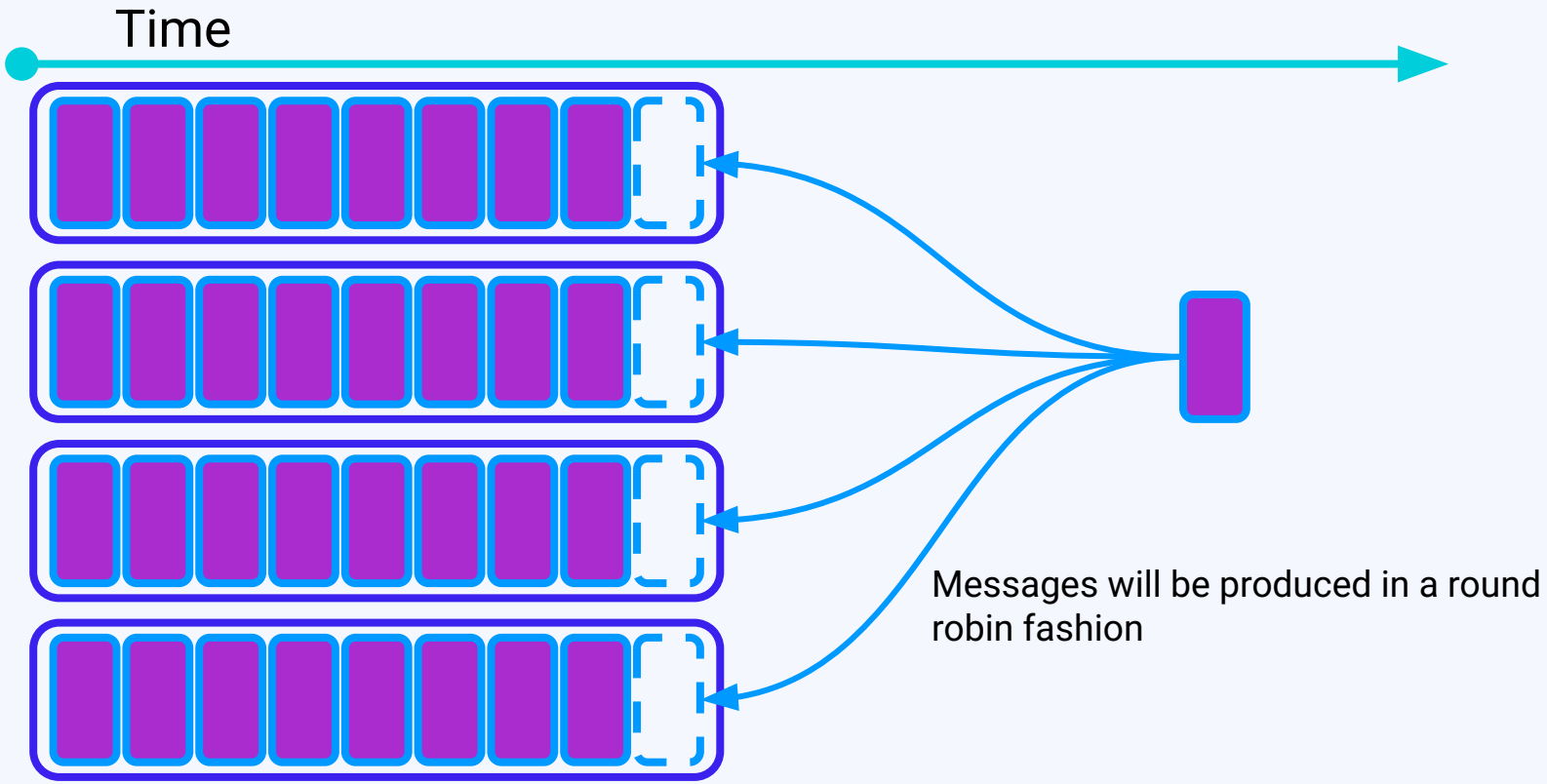
Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner

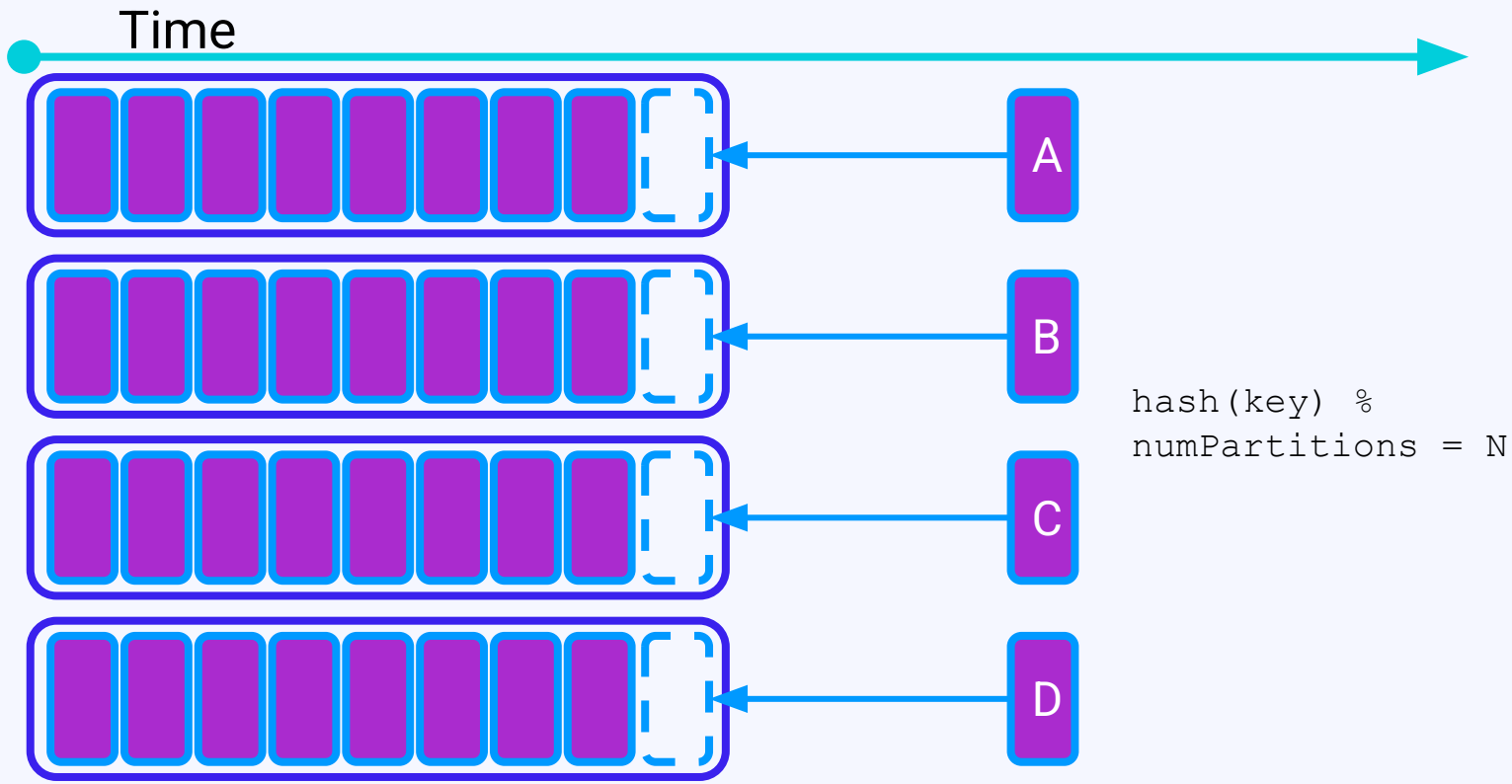


If a key isn't provided, messages will be produced in a round robin fashion

Producing to Kafka - No Key

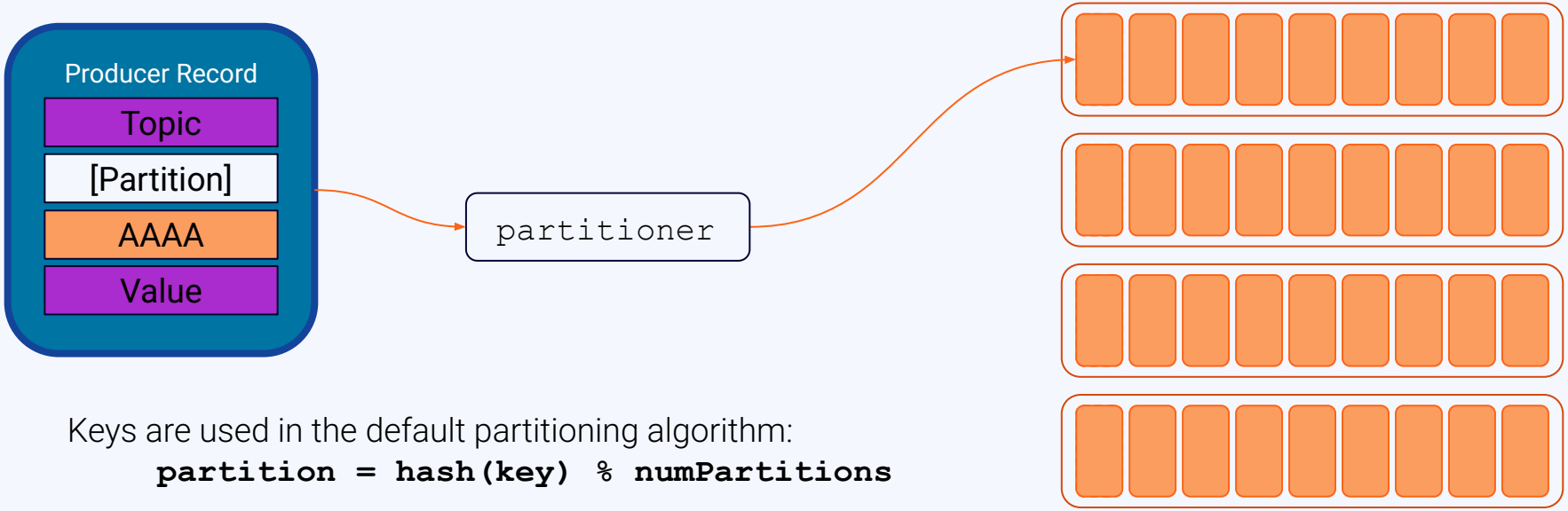


Producing to Kafka - With Key



Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key

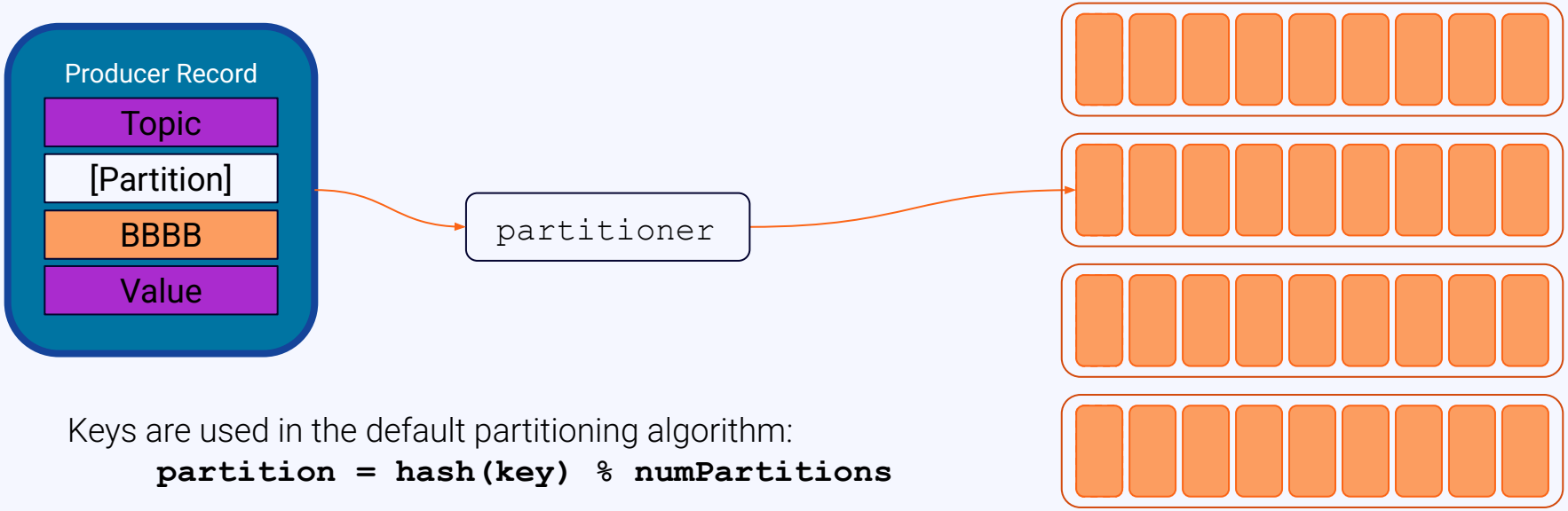


Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Ordering

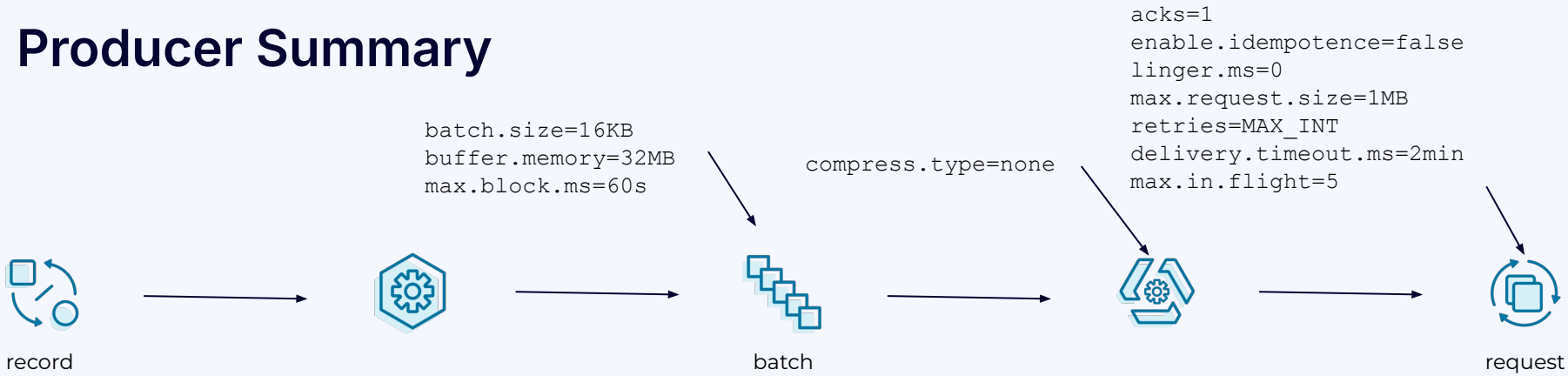
Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key



Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Producer Summary



Serializer

- Retrieves and caches schemas from Schema Registry

Partitioner

- Java client uses murmur2 for hashing
- If key not provided performs round robin
- If keys unbalanced it will overload one leader

Record accumulator

- Buffer per partition, seldom used partitions may not achieve high batching due to this
- If many producers are in the same JVM, memory and GC could become important
- Sticky partitioner could be used to increase batches in the case of round robin (KIP-408)

Compression

- Allows faster transfer to the producer
- Reduces the inter broker replication load
- Uses less page cache and disk space at broker
- Compress the batch not the record
- Gzip is more CPU intensive, Snappy is lighter, LZ4/ZStd are a good balance*

Sender thread

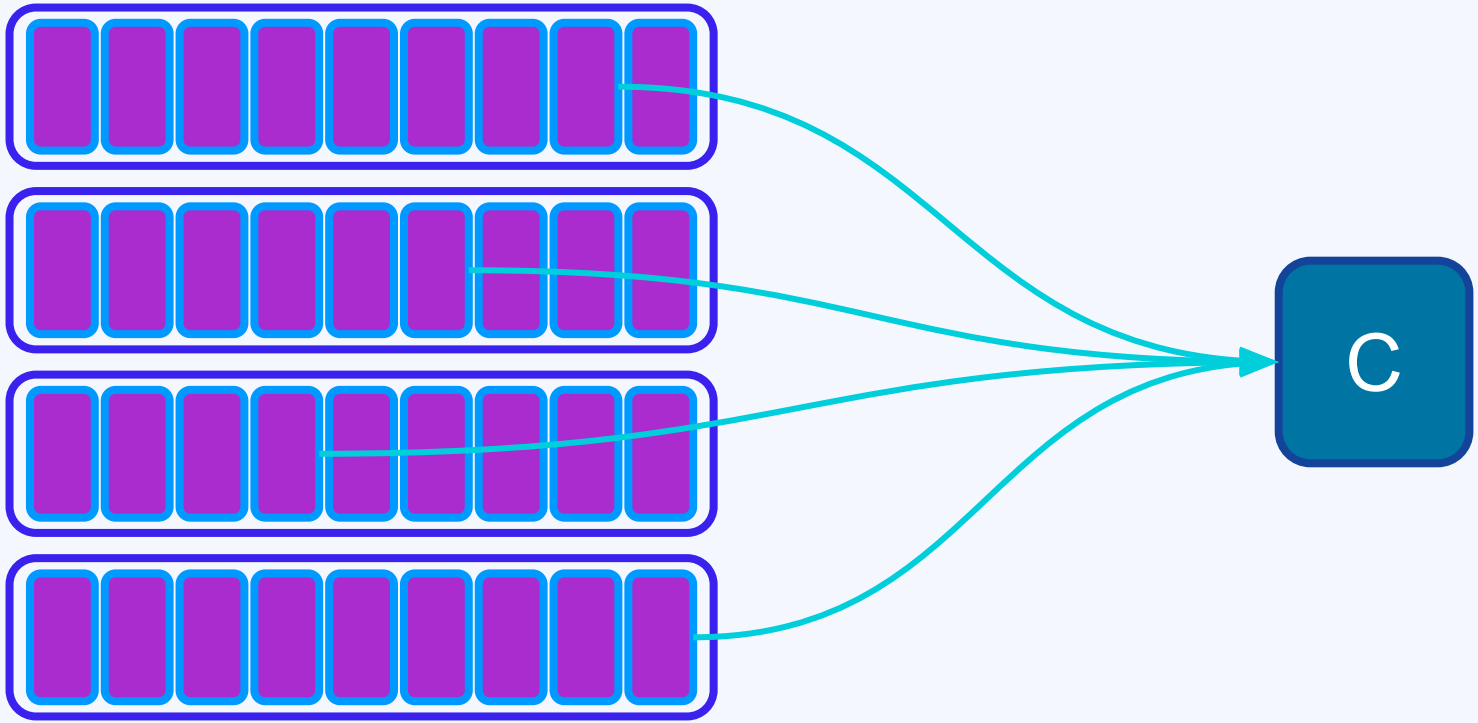
- Batches grouped in a producer request and sent by broker
- Multiple batches to different partitions could be in the same producer request

Consuming from Kafka

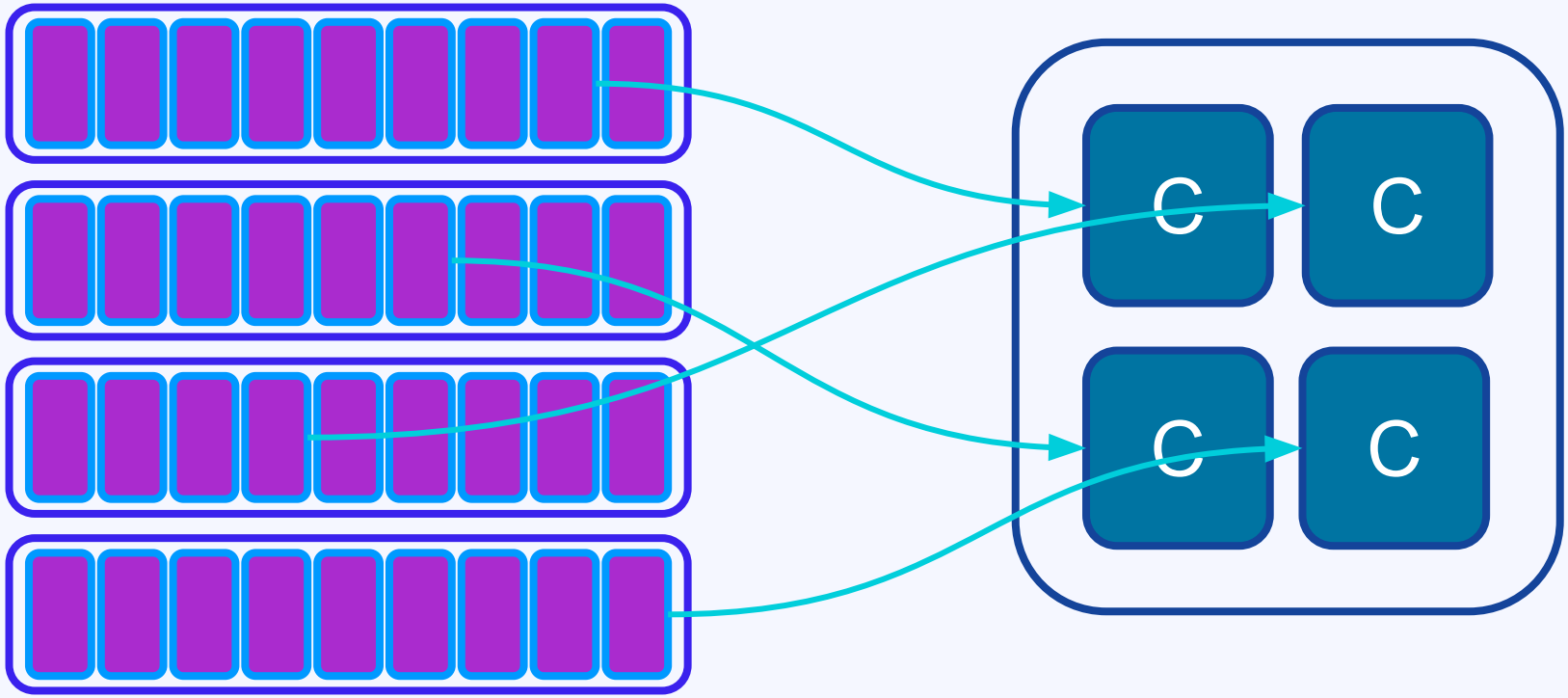
A basic Java Consumer

```
  final Consumer<String, String> consumer = new KafkaConsumer<String,
String>(props);
  consumer.subscribe(Arrays.asList(topic));
  try {
    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records) {
        -- Do Some Work --
      }
    }
  } finally {
    consumer.close();
  }
}
```

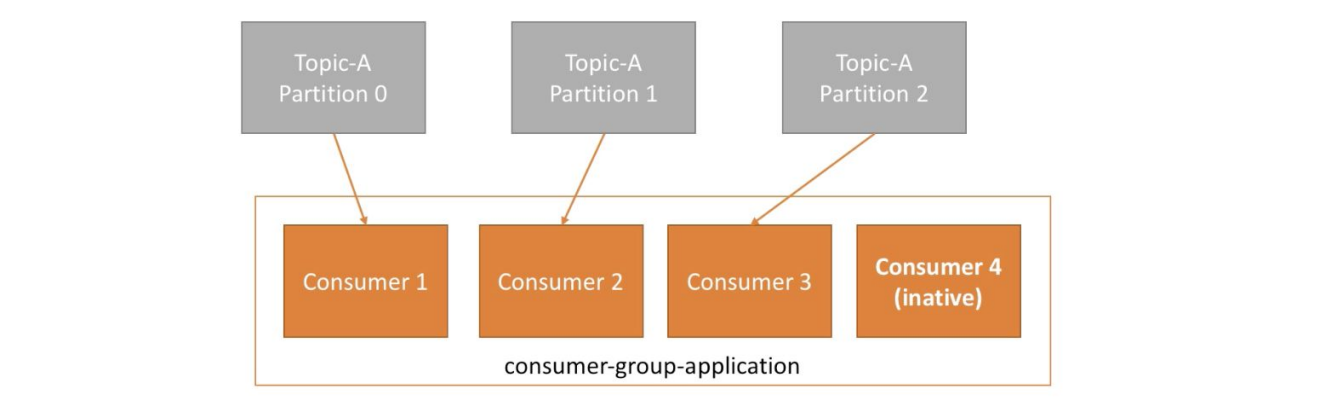
Consuming From Kafka - Single Consumer



Consuming From Kafka - Grouped Consumers

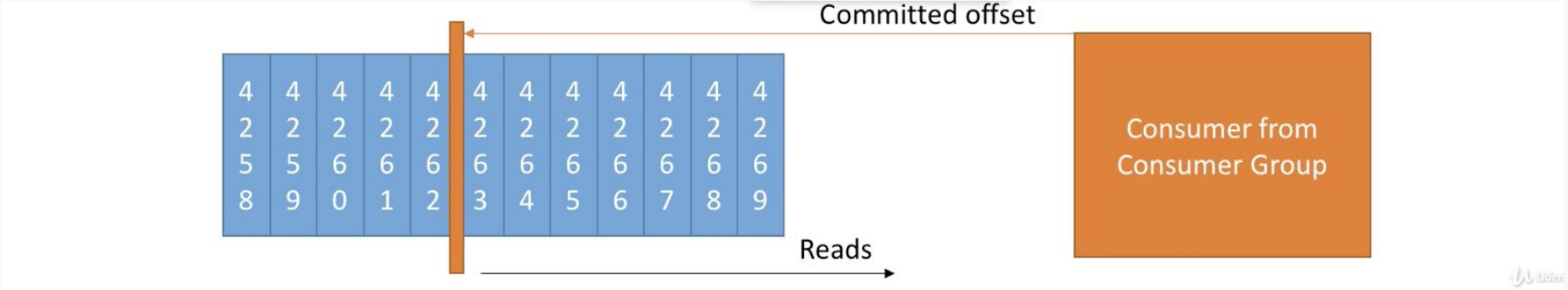


What if you have more Consumers than partitions?



Consumer Offsets

- Kafka stores the offsets at which a consumer group has been reading
- Offsets are committed in an internal **topic** called **_consumer_offsets**
- When a consumer in a group has processed data received from Kafka it should be committing the offsets
- If a consumer dies Kafka will be able to tell it where it was so it can start reading again
- Working example



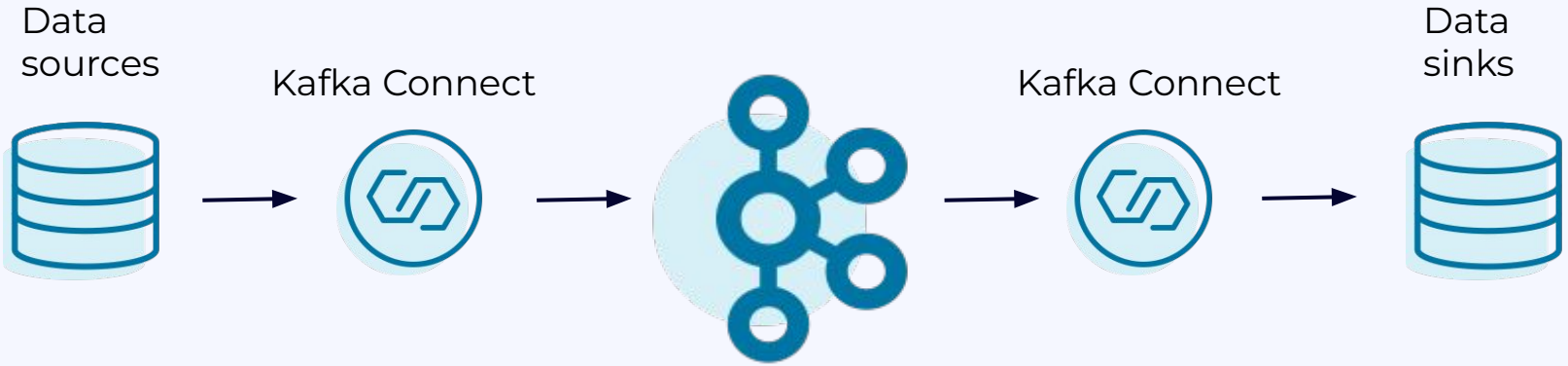
Kafka Connect

No/Low Code connectivity to many systems

Kafka Connect

No-Code way of connecting known systems (databases, object storage, queues, etc) to Apache Kafka

Some code can be written to do custom transforms and data conversions though maybe out of the box Single Message Transforms and Converters exist

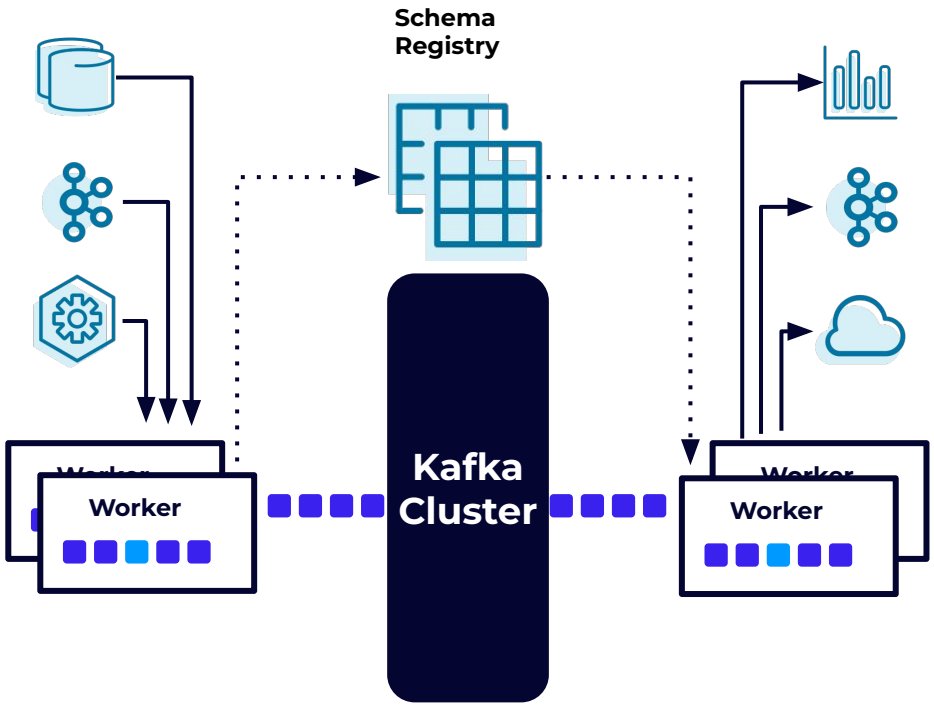


Kafka Connect Durable Data Pipelines

Integrate upstream and downstream systems with Apache Kafka®

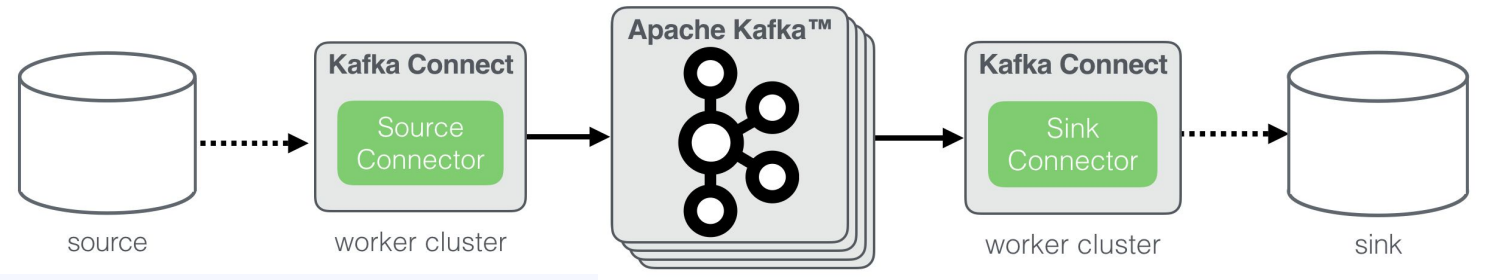
- **Capture** schema from sources, use schema to inform data sinks
- **Highly Available** workers ensure data pipelines aren't interrupted
- **Extensible** framework API for building custom connectors

Kafka Connect



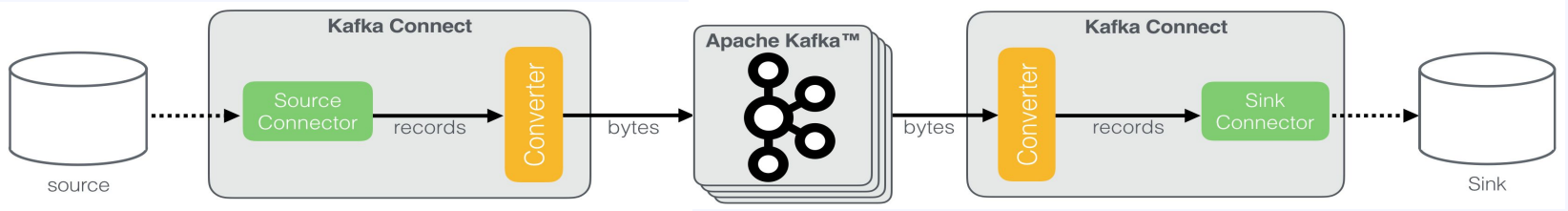
Kafka Connect

Connectors are reusable components that know how to talk to specific **sources and sinks**.



Kafka Connect

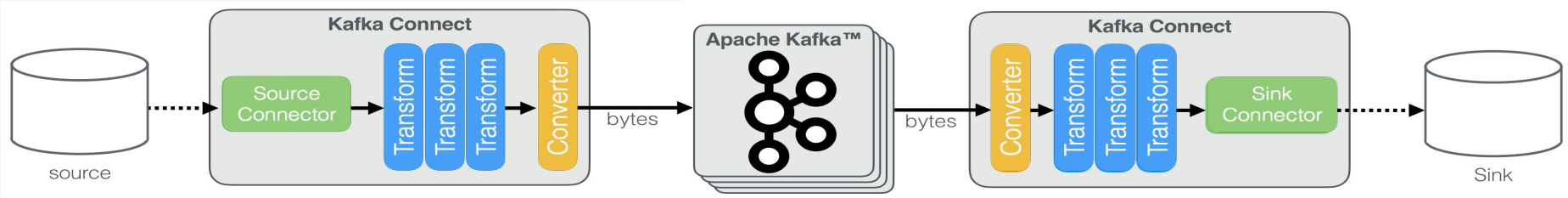
Convert between the **source and sink record** objects and the **binary format** used to persist them in Kafka.



JSON, Avro, and others

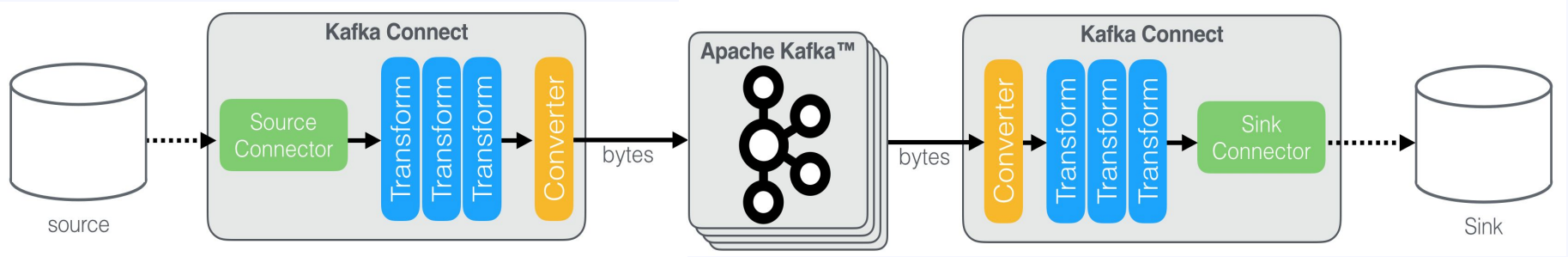
Kafka Connect

Connectors, transforms, and converters
are **pluggable components**.



Kafka Connect

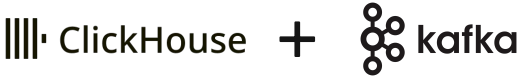
Modify the structure of keys and values, topics, and partition of **source and sink record objects**.



CONFLUENT
Developer

Your Apache Kafka journey begins here

developer.confluent.io



Integrating Kafka with ClickHouse

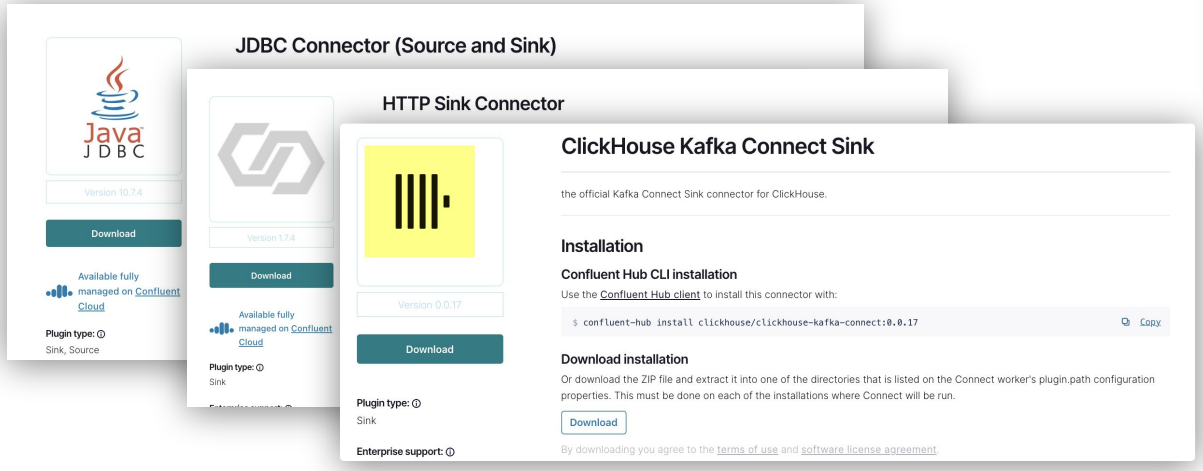
$(\text{real-time})^2$

Existing Kafka Integration Options

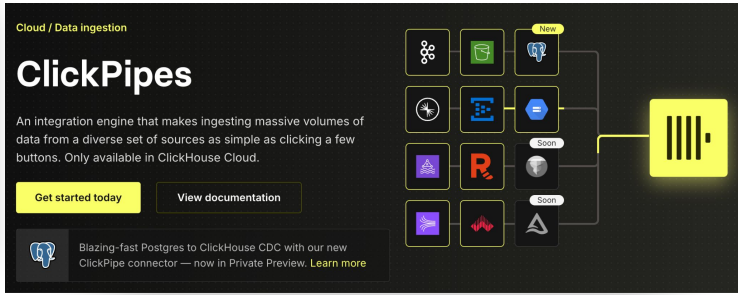
01 Kafka Table Engine

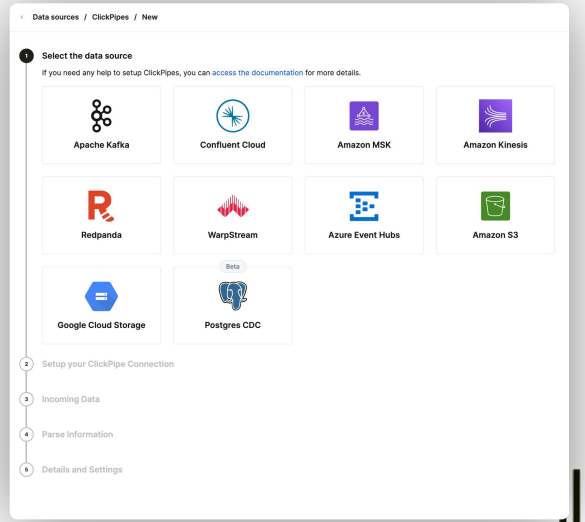
```
CREATE TABLE queue (
        timestamp UInt64,
        level String,
        message String
    )
ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');
```

02 ClickHouse Kafka Connect Sink



03 ClickPipes

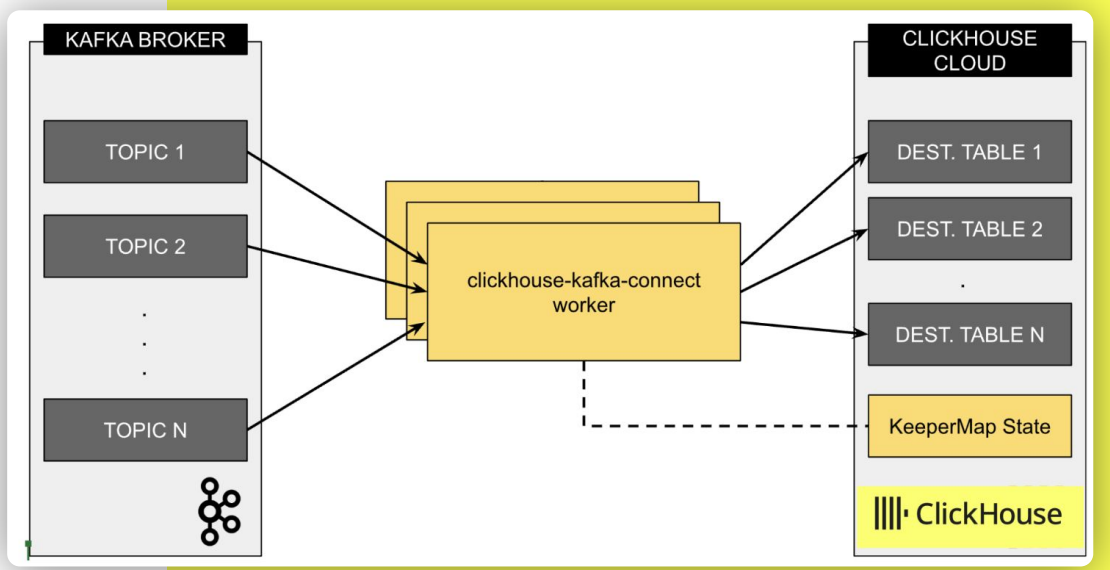




ClickHouse

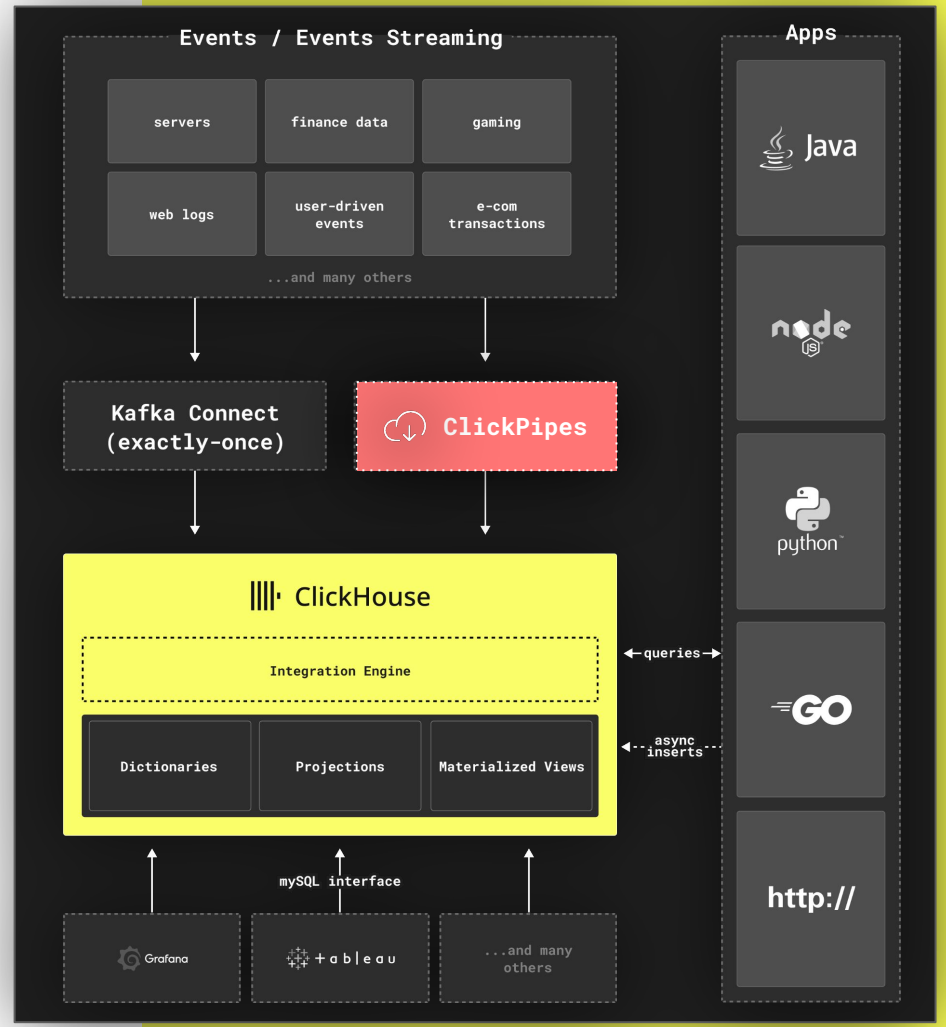
ClickHouse Kafka Connect Sink

- Shipped with out-of-the-box **exactly-once semantics** powered by ClickHouse core feature named **KeeperMap** (table engine used as a state store by the connector) and allows for minimalistic architecture
- **Core integration**: Built, maintained, and **supported by ClickHouse**
- Tested **continuously** against ClickHouse Cloud.
- Data inserts with a **declared schema** (schema-based data, e.g. Avro, Protobuf, etc.) and **schemaless** (e.g. JSON)
- Support for all data types of ClickHouse

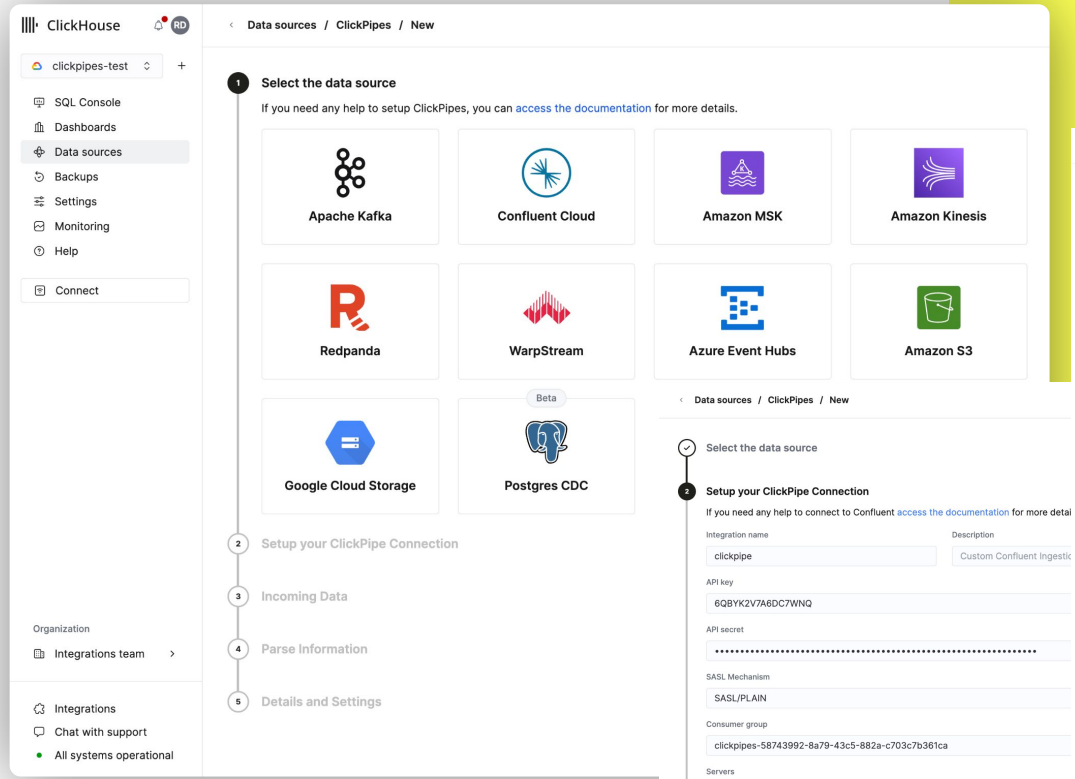


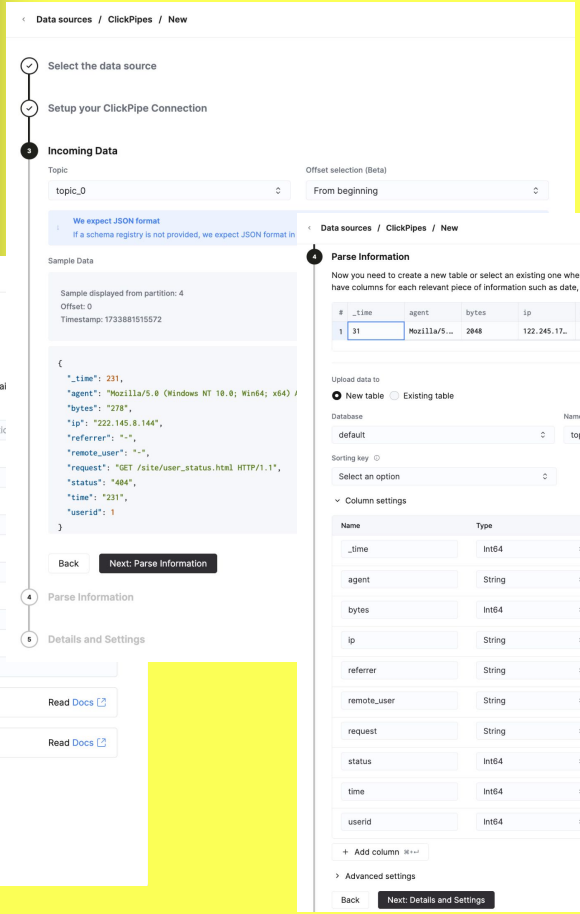
ClickPipes *noun*
/klɪk paɪps/

Cloud-Native experience to ingest data from remote data sources to ClickHouse Cloud



ClickPipes for Confluent Cloud





ClickHouse + kafka

Improvements

Recent improvements related to Kafka

24.8

- [Experimental] Add new experimental **Kafka** storage engine to **store offsets in Keeper** instead of relying on committing them to Kafka. It makes the commit to ClickHouse tables atomic with regard to consumption from the queue. #57625 (János Benjamin Antal).
- [Bug Fix] Fix inserting into stream like engines (**Kafka**, RabbitMQ, NATS) through HTTP interface. #67554 (János Benjamin Antal).

24.4

- [Improvement] Introduce separate consumer/producer tags for the **Kafka** configuration. This avoids warnings from **librdkafka** that consumer properties were specified for producer instances and vice versa (e.g. `Configuration property session.timeout.ms is a consumer property and will be ignored by this producer instance`). Closes: #58983. #58956 (Aleksandr Musorin).
- [Improvement] Add **librdkafka**'s client identifier to log messages to be able to differentiate log messages from different consumers of a single table. #62813 (János Benjamin Antal).

24.2

- [Improvement] Unify XML and SQL created named collection behaviour in **Kafka** storage. #59710 (Pervakov Grigorii).

24.1

- [Improvement] Create consumers for **Kafka** tables on the fly (but keep them for some period - **`kafka_consumers_pool_ttl_ms`**, since last used), this should fix problem with statistics for **`system.kafka_consumers`** (that does not consumed when nobody reads from Kafka table, which leads to live memory leak and slow table detach) and also this PR enables stats for **`system.kafka_consumers`** by default again. #58310 (Azat Khuzhin).
- [Bug Fix] Delay reading from **StorageKafka** to allow multiple reads in materialized views #58477 (János Benjamin Antal).

Blog / Engineering

ClickHouse Release 24.8 LTS

The ClickHouse Team
Sep 3, 2024 - 20 minutes read

Another month goes by, which means it's time for another release!

ClickHouse version 24.8 contains **19 new features** 🎁 **18 performance optimisations** 🚗 **65 bug fixes** 🐛

This release is an LTS (Long Term Support) one, which means it will be supported for 12 months after release. To learn more about Stable and LTS releases, see the documentation.

In this release, we've got the newly revamped JSON type, a table engine for time-series data, exactly-once processing of Kafka messages, and of course, join improvements!

The New Kafka Engine 🧪

The Kafka engine exists in ClickHouse since 2017
— it implements streaming consumption and data pipelines from Kafka.

Its downside: non-atomic commit to Kafka and to ClickHouse, leading to the possibility of duplicates in the case of retries.

Now there is an option to manage the offsets in Keeper:

```
SET allow_experimental_kafka_offsets_storage_in_keeper = 1;

CREATE TABLE ... ENGINE = Kafka(
    'localhost:19092', 'topic', 'consumer', 'JSONEachRow')
SETTINGS
    kafka_keeper_path = '/clickhouse/{database}/kafka',
    kafka_replica_name = 'r1';
```

Developer: János Benjamin Antal.

The New Kafka Engine 🧪

```
CREATE TABLE ... ENGINE = Kafka(
    'localhost:19092', 'topic', 'consumer', 'JSONEachRow')
SETTINGS
    kafka_keeper_path = '/clickhouse/{database}/kafka',
    kafka_replica_name = 'r1';
```

With the new option it does not rely on Kafka to track the offsets,
and does it by itself with ClickHouse Keeper.

If an insertion attempt fails, it will take exactly the same chunk of data and repeat the insertion, regardless of network or server failures.

This enables deduplication and makes the consumption exactly-once.

Developer: János Benjamin Antal.

ClickHouse + kafka

Demo

KEEP IN TOUCH!

ClickHouse Malaysia Meetup Group

clickhouse.com/slack

@clickhousedb

@clickhouseinc #clickhouseDB

clickhouse

ClickHouse

FUNDAMENTALS WORKSHOP

January 15, 2025 - 11:30 AM MYT

ClickHouse + CONFLUENT

Questions