

ClickHouse



CONFLUENT

meetup

WORQ TTDI

Kuala Lumpur, Malaysia

December 11, 2024 at 5:30 PM MYT



ClickHouse + Kafka Integration

Derek Chia

Principal Support Engineer @



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



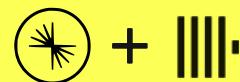
Karthikayan Muthuramalingam

Solutions Engineer, Confluent

 [karthikayan-muthuramalingam](https://www.linkedin.com/in/karthikayan-muthuramalingam)

 [Cruzkn](https://github.com/Cruzkn)

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



Agenda

01

Introduction

04

Recent improvements to
Kafka Support

02

Apache Kafka 101

05

Demo

03

Integrating Kafka with
ClickHouse

06

Questions

Introduction : ClickHouse



What is ClickHouse?

Open source	column-oriented	distributed	OLAP database
Developed since 2009 - OSS 2016 38k+ Github stars 1.6k+ contributors 600+ releases	Best for aggregations Files per column Sorting and indexing Background merges	Replication Sharding Multi-master Cross-region	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?



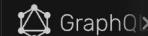
Real-time Dashboards

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



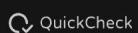
Real-time Analytics

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



Data Warehouse Speed Layer

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



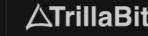
Logging & Metrics

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



Business Intelligence

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

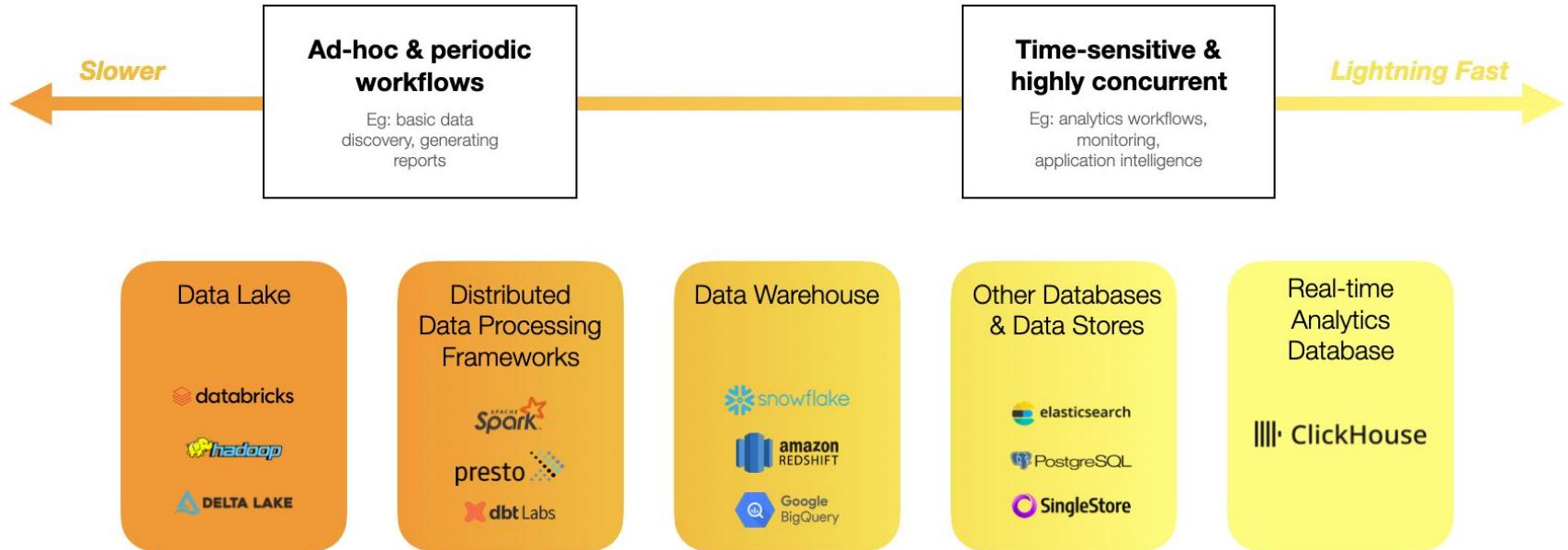


ML & Data Science

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

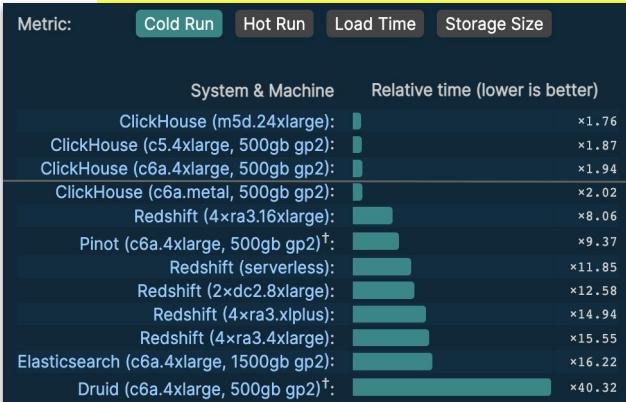
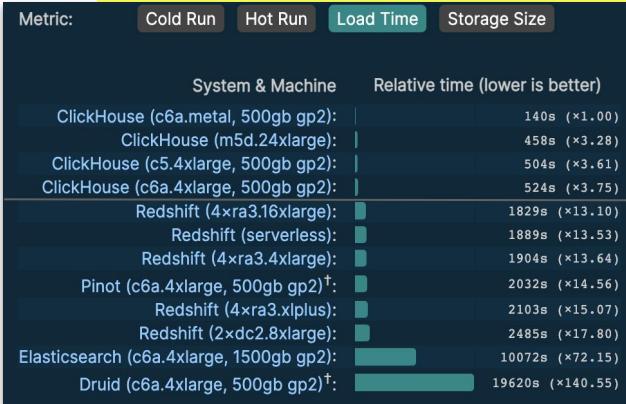
Highly resource efficient

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.



benchmark.clickhouse.com



Use cases



Logs, events, traces

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



zomato

ebay

resmo

runreveal



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.



Microsoft

Contentsquare



lyft

highlight.io



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

QuickCheck

TrillaBit

Deutsche Bank

NANO CORP.

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

***ADMIXER**

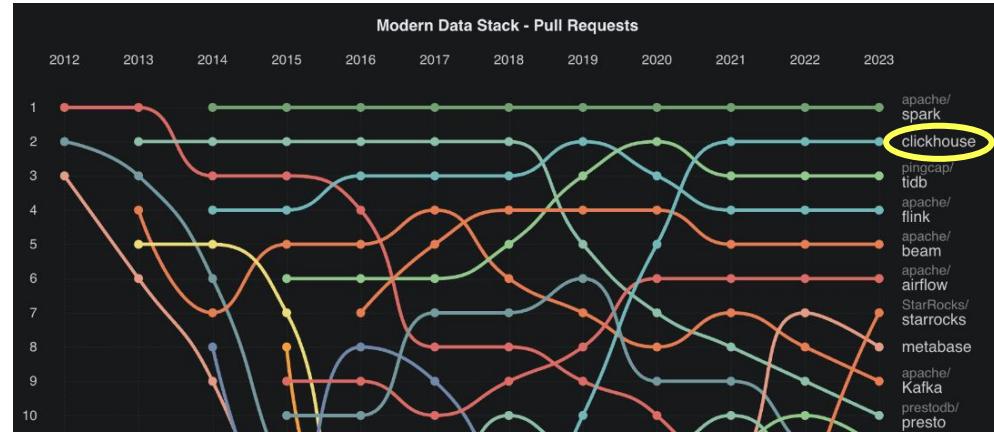
ensemble

DeepL



ClickHouse Open Source

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



ClickHouse Cloud



- ✓ 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

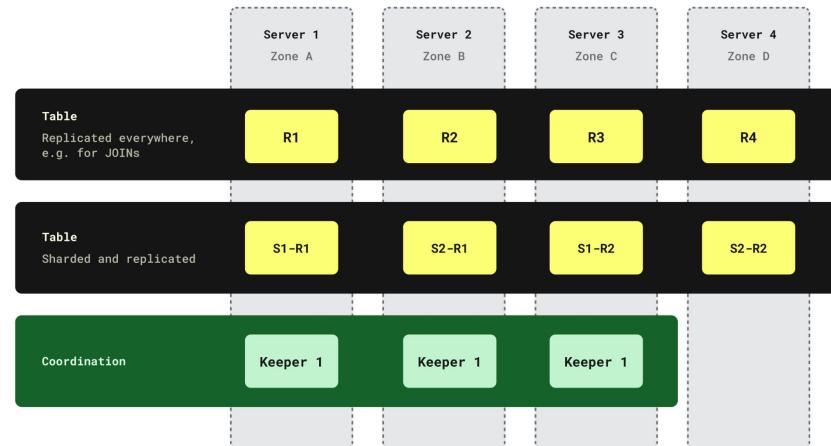


||| ClickHouse

Self-managed

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

Sample self-managed architecture

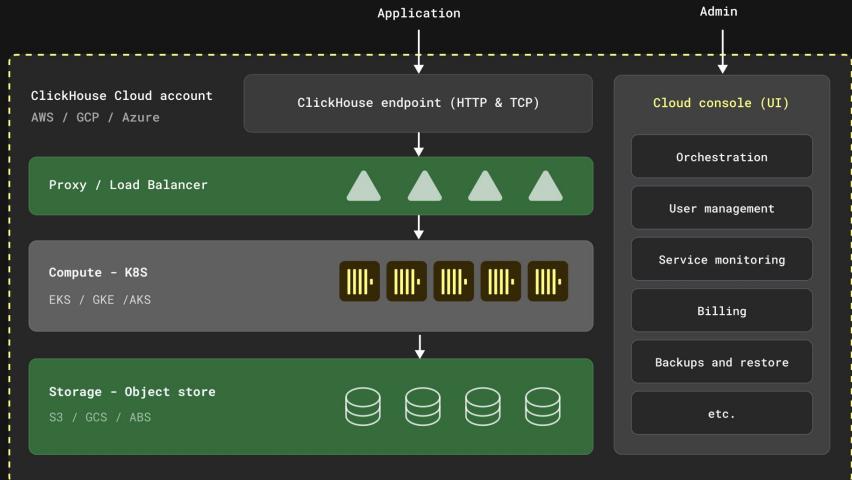


||| ClickHouse

Cloud

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

ClickHouse Cloud architecture



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

Integrations

If there's an integration that you would like to see added to this list, please let us know.

Search Integrations Request Integration

Categories

- View all
- Data Ingestion
- Data Visualization
- SQL Client
- Language Client

Data Ingestion

 Amazon S3	 Kafka	 Airbyte <small>New</small>	 dbt	 Vector	 PostgreSQL
Import from, export to, and transform S3 data in flight with ClickHouse built-in S3 functions.	Integration with Apache Kafka, the open-source distributed event streaming platform.	Use Airbyte to create ELT data pipelines with more than 140 connectors to load and sync your data into ClickHouse.	Use dbt data build tool to transform data in ClickHouse by simply writing SQL statements.	A lightweight, ultra-fast tool for building observability pipelines with built-in compatibility with ClickHouse.	Perform SELECT and INSERT operations on data stored on a remote PostgreSQL server directly from ClickHouse.
View Integration →	Community View Integration →	Community View Integration →	Community View Integration →	Partner View Integration →	Core View Integration →

 MySQL	 Google Cloud Storage (GCS)	 PrQL <small>New</small>	 Confluent Cloud
Perform SELECT and INSERT operations on data stored on a remote MySQL server directly from ClickHouse.	Import from, export to, and transform GCS data in flight with ClickHouse built-in functions.	Connect your ClickHouse instance to PrQL to sync data to and from your users' data warehouses, databases, and object storage.	Integration with Confluent Cloud, the Cloud-Native, Complete Solution For Apache Kafka.
View Integration →	Core View Integration →	Partner View Integration →	Core View Integration →

Data Visualization

 Grafana	 Superset	 Deepnote	 HEX	 Metabase <small>New</small>	 Tableau
With Grafana you can create, explore and share all of your data through dashboards.	Explore and visualize your ClickHouse data with Apache Superset.	Deepnote is a collaborative, Jupyter-compatible data notebook built for teams to discover and share insights.	Hex is a modern, collaborative platform with notebooks, data apps, SQL, Python, no-code, R, and so much more.	Metabase is an easy-to-use, open source UI tool for asking questions about your data.	Tableau is a popular visual analytics platform for business intelligence.
Partner View Integration →	Core View Integration →	Partner View Integration →	Partner View Integration →	Core View Integration →	Community View Integration →

SQL Client

 DBVisualizer	 ClickHouse Client	 DataGrip
DBVisualizer Pro is a comprehensive database management and administration tool with an easy connection to ClickHouse Cloud.	ClickHouse Client is the native command-line client for ClickHouse.	DataGrip is a powerful database IDE with dedicated support for ClickHouse.
Partner View Integration →	Core View Integration →	Partner View Integration →

Language Client

 Java	 Go	 Python	 Node.js <small>New</small>	 C# <small>New</small>
The Java client is an async, lightweight, and low-overhead library for ClickHouse.	The Go client uses the native interface for a performant, low-overhead means of connecting to ClickHouse.	A suite of Python packages for connecting Python to ClickHouse.	The official client for connecting Node.js applications to ClickHouse.	ClickHouse.Client is a feature-rich ADO.NET client implementation for ClickHouse.
Core View Integration →	Core View Integration →	Core View Integration →	Core View Integration →	Community View Integration →

Introduction : Confluent





Introduction to Confluent



The Rise of Event Streaming

Apache Kafka
originally created at
LinkedIn by
Confluent founders

2010

 CONFLUENT

2014

80%

Fortune 100
Companies
trust and use
Apache Kafka

2020

BILLBOARD
ENTERTAINMENT

YAHOO!



Tencent 腾讯



Goldman
Sachs

NETFLIX



PayPal



Dropbox



slack

LinkedIn

WIKIPEDIA
The Free Encyclopedia

Pinterest

airbnb

GoPro

stripe

intuit

Square



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



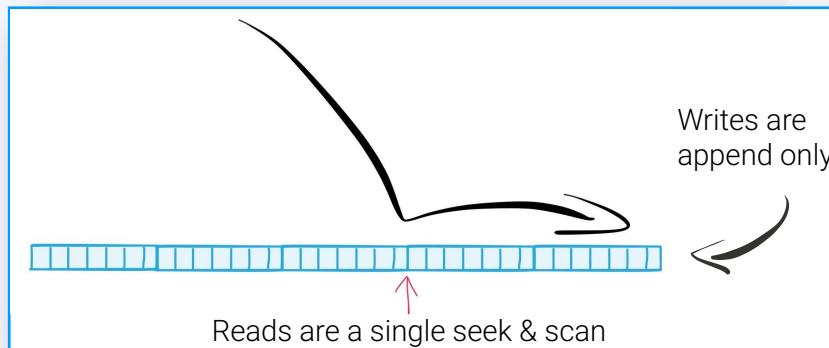
Introducing Apache Kafka®



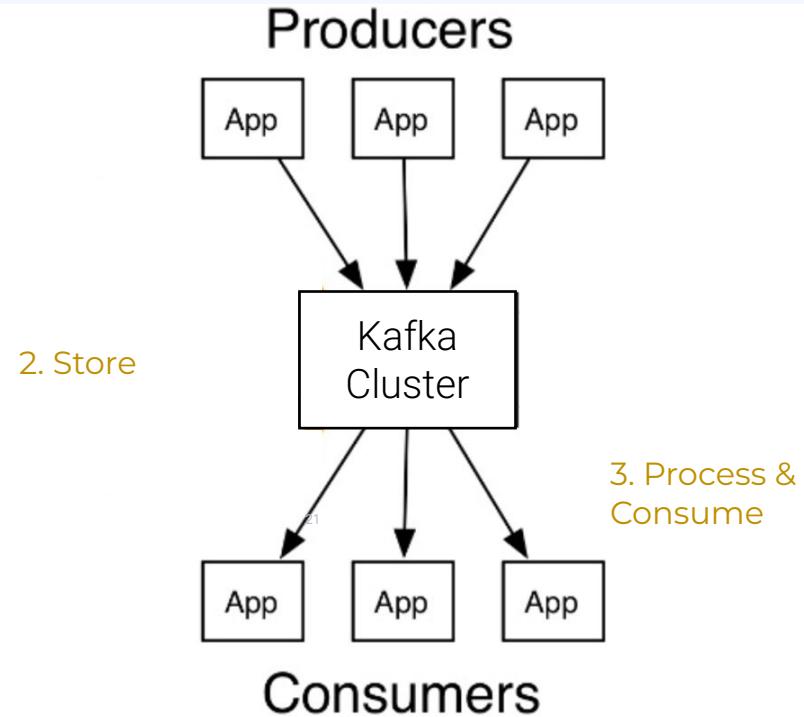
Apache Kafka - Publish & Subscribe Reimagined

Kafka

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



1. Publish Stream Events



Topics

Clicks



Orders



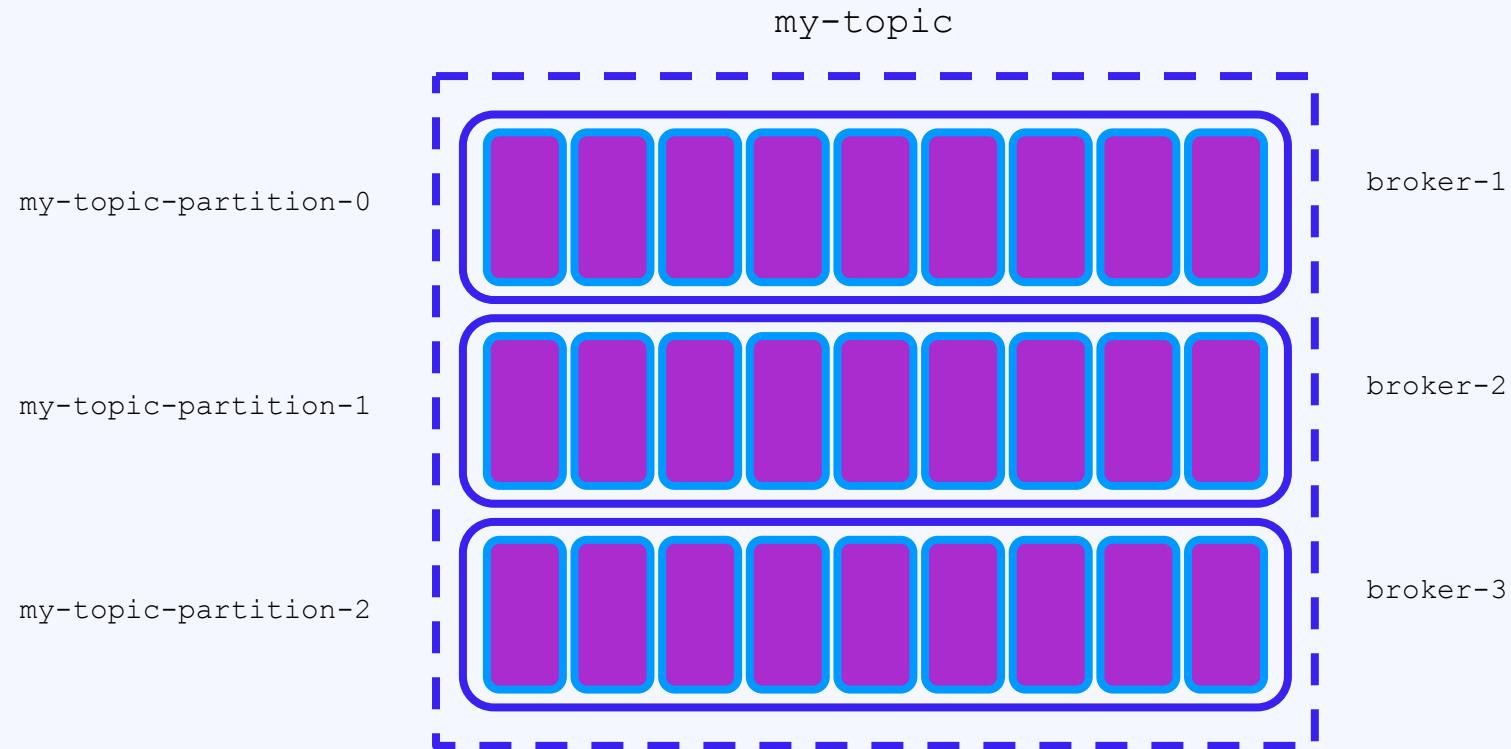
Customers



Topics are similar in concept
to tables in a database

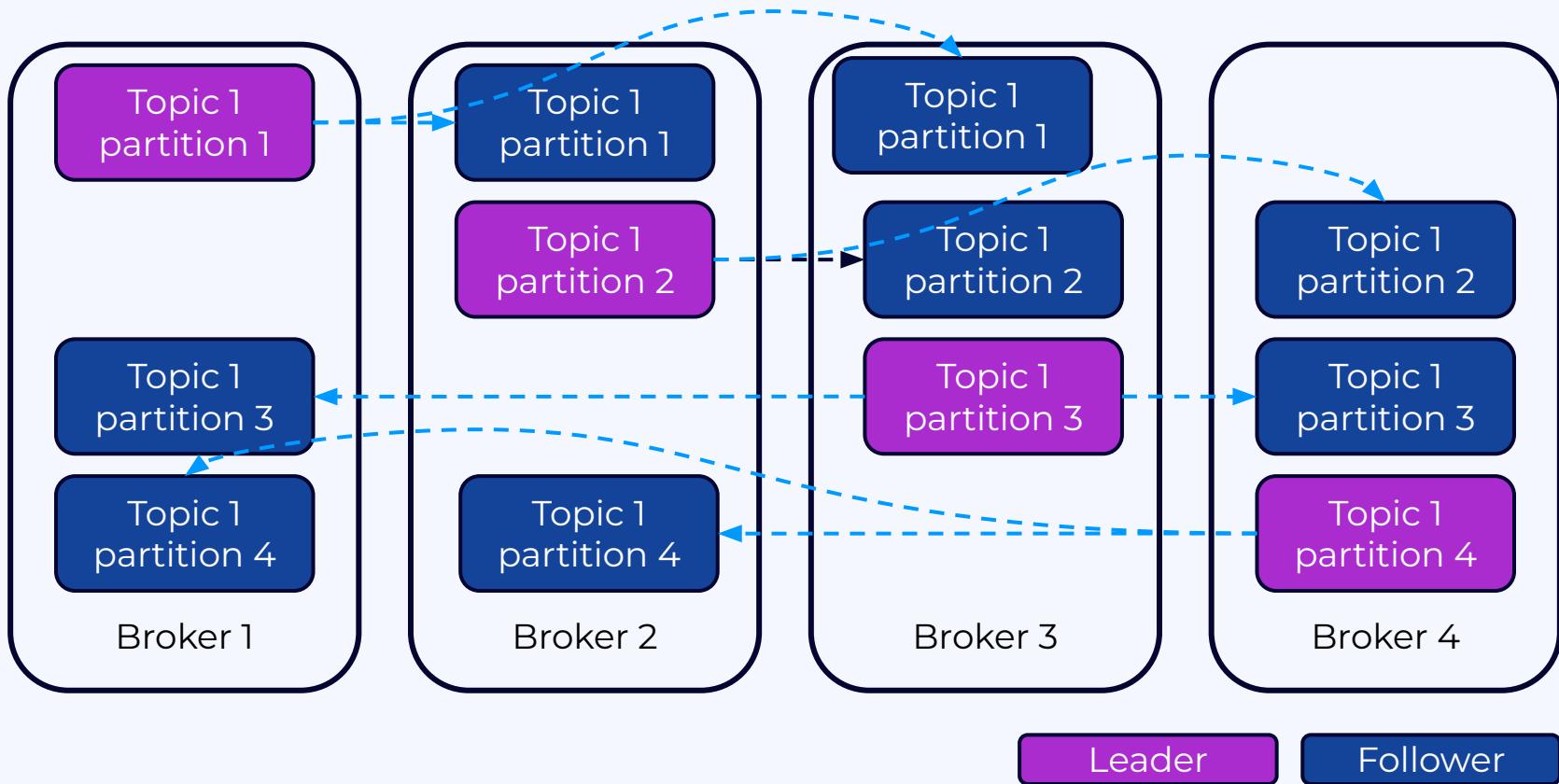


Kafka Topics





How Does Kafka Provide Resilience?

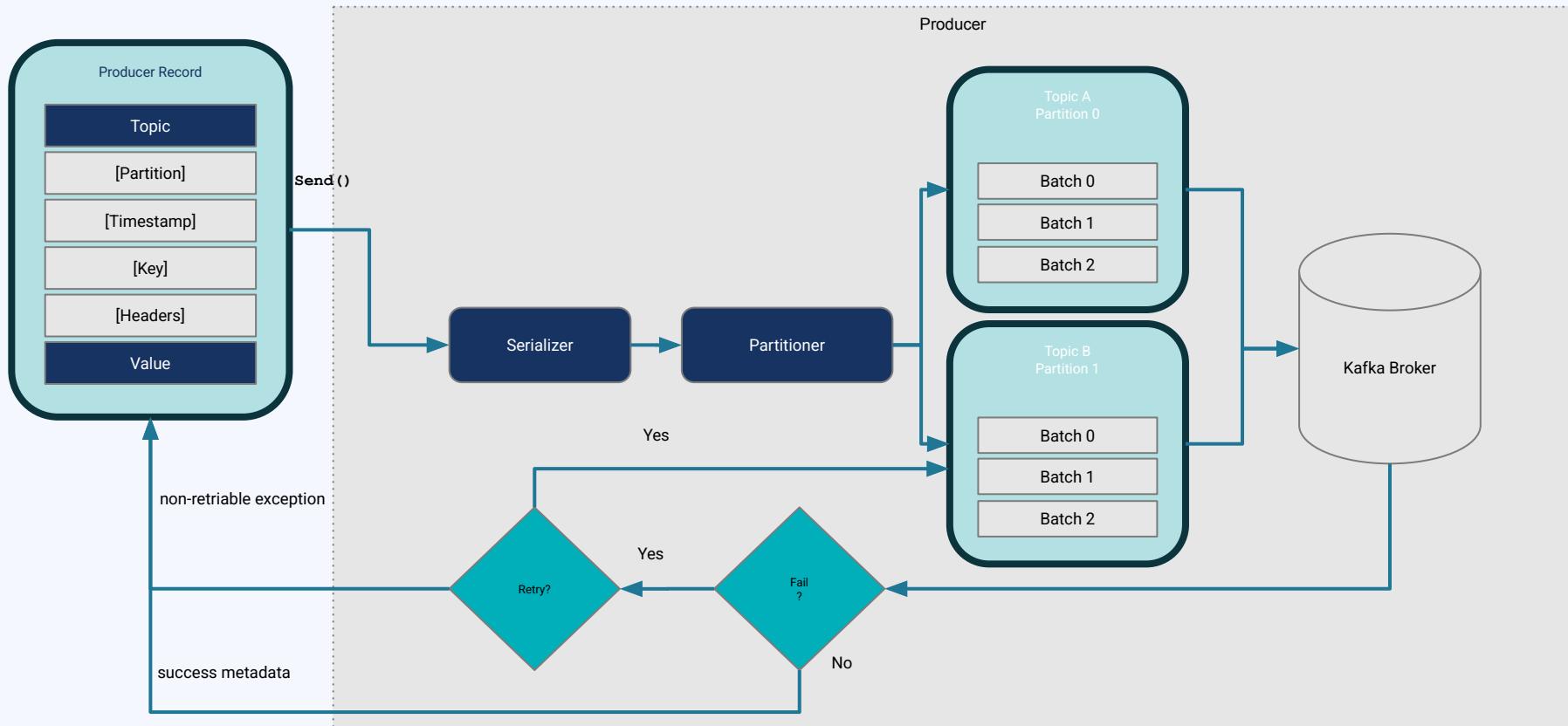




Producing to Kafka



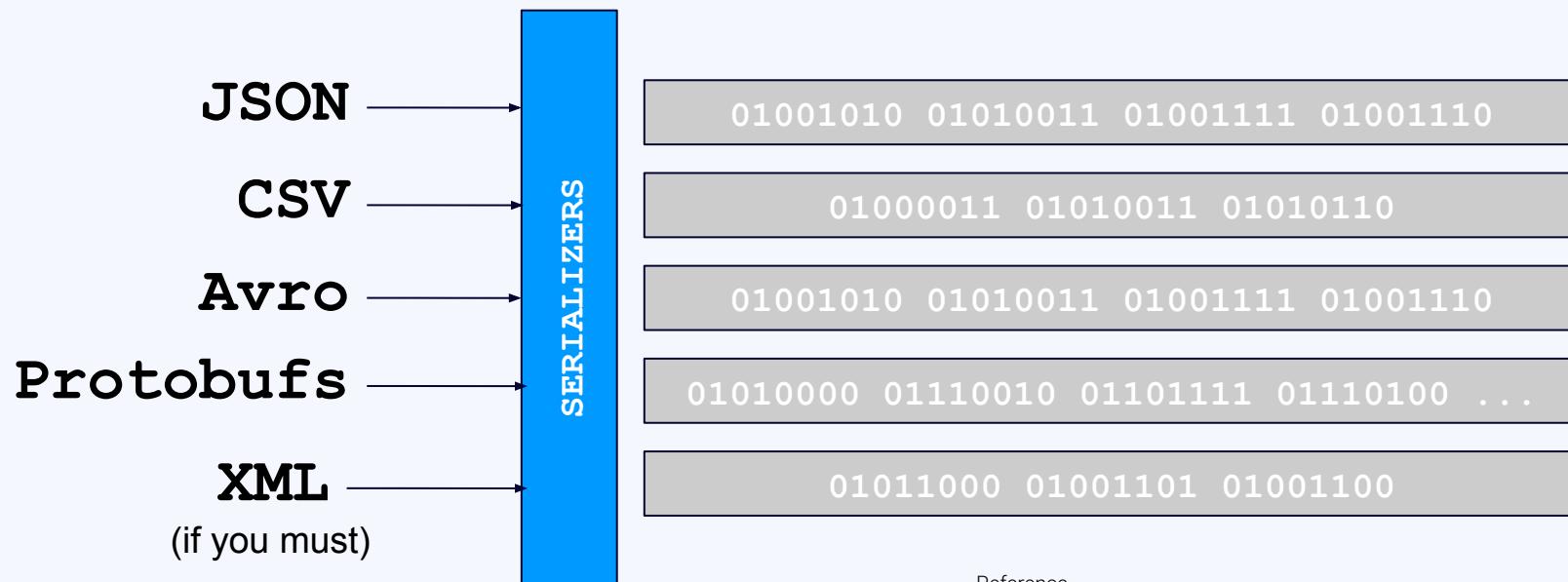
What happens inside a producer?





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/documentationstreams/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);
```

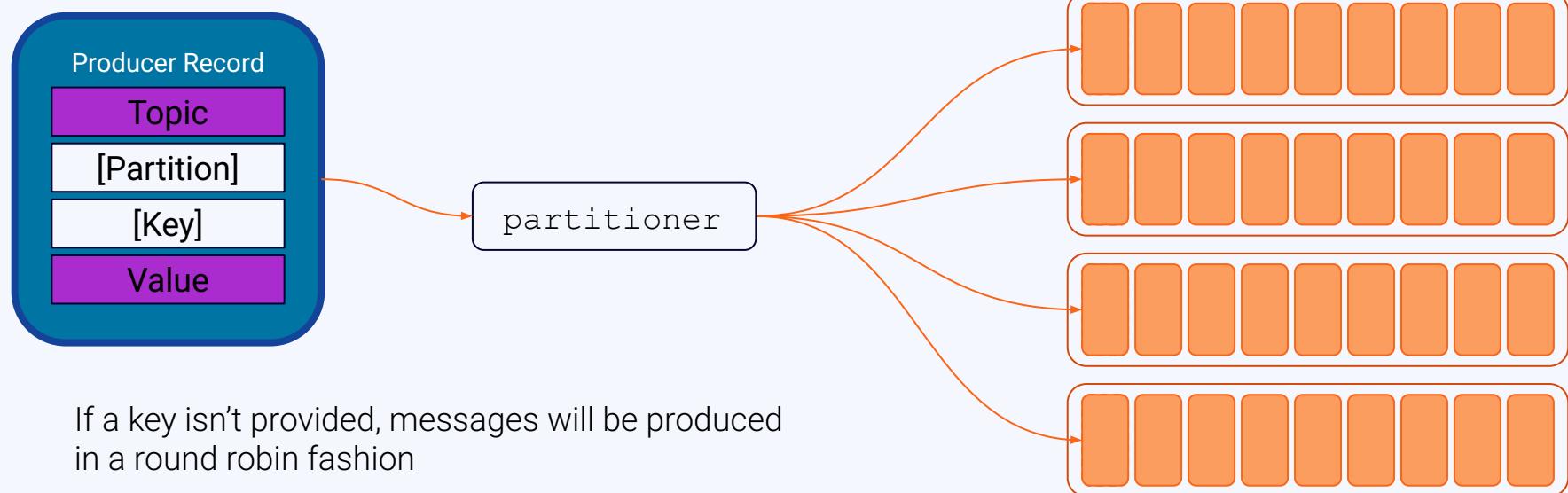
Reference

<https://kafka.apache.org/10/documentationstreams/developer-guide/datatypes.html>



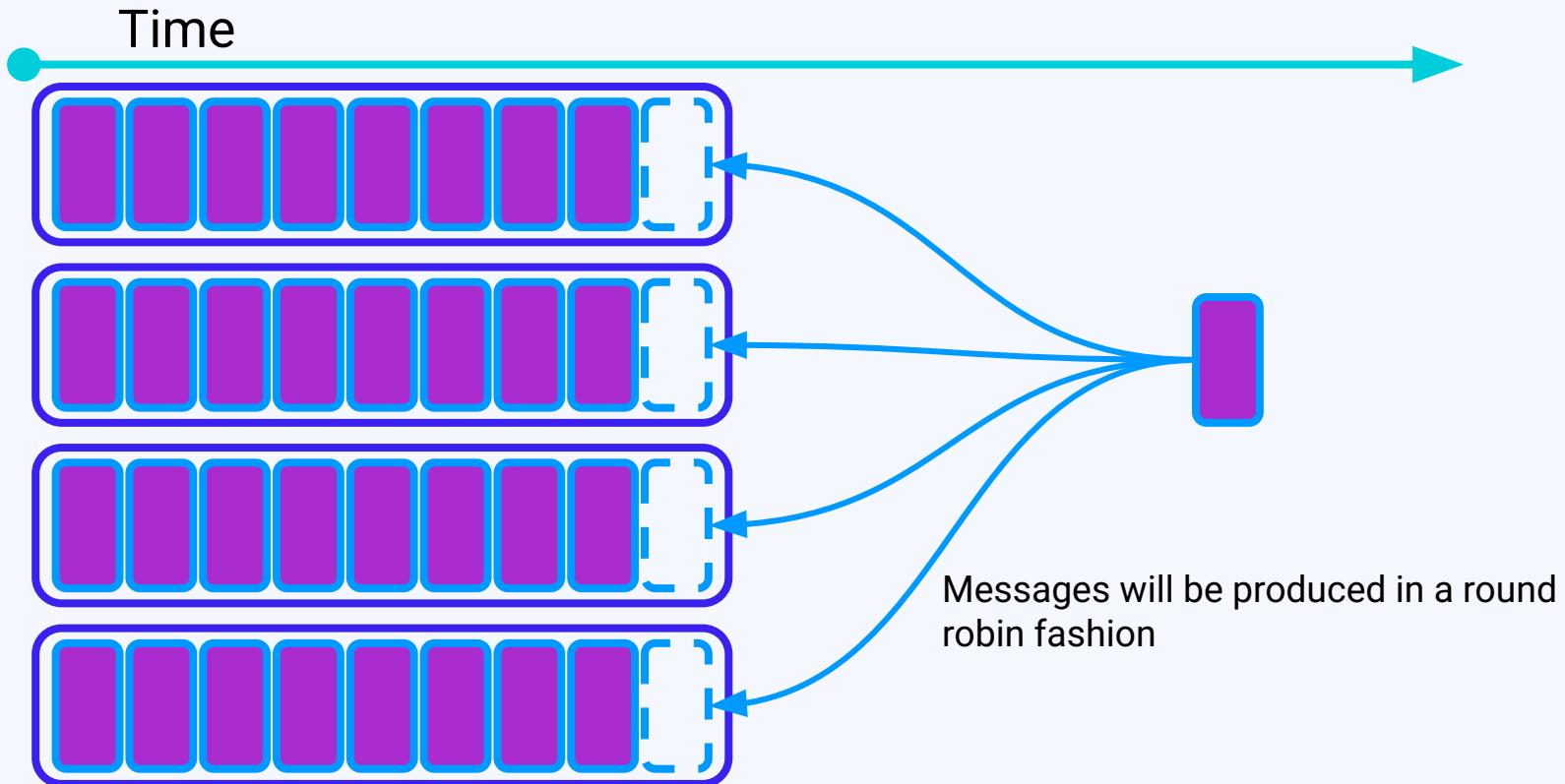
Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner



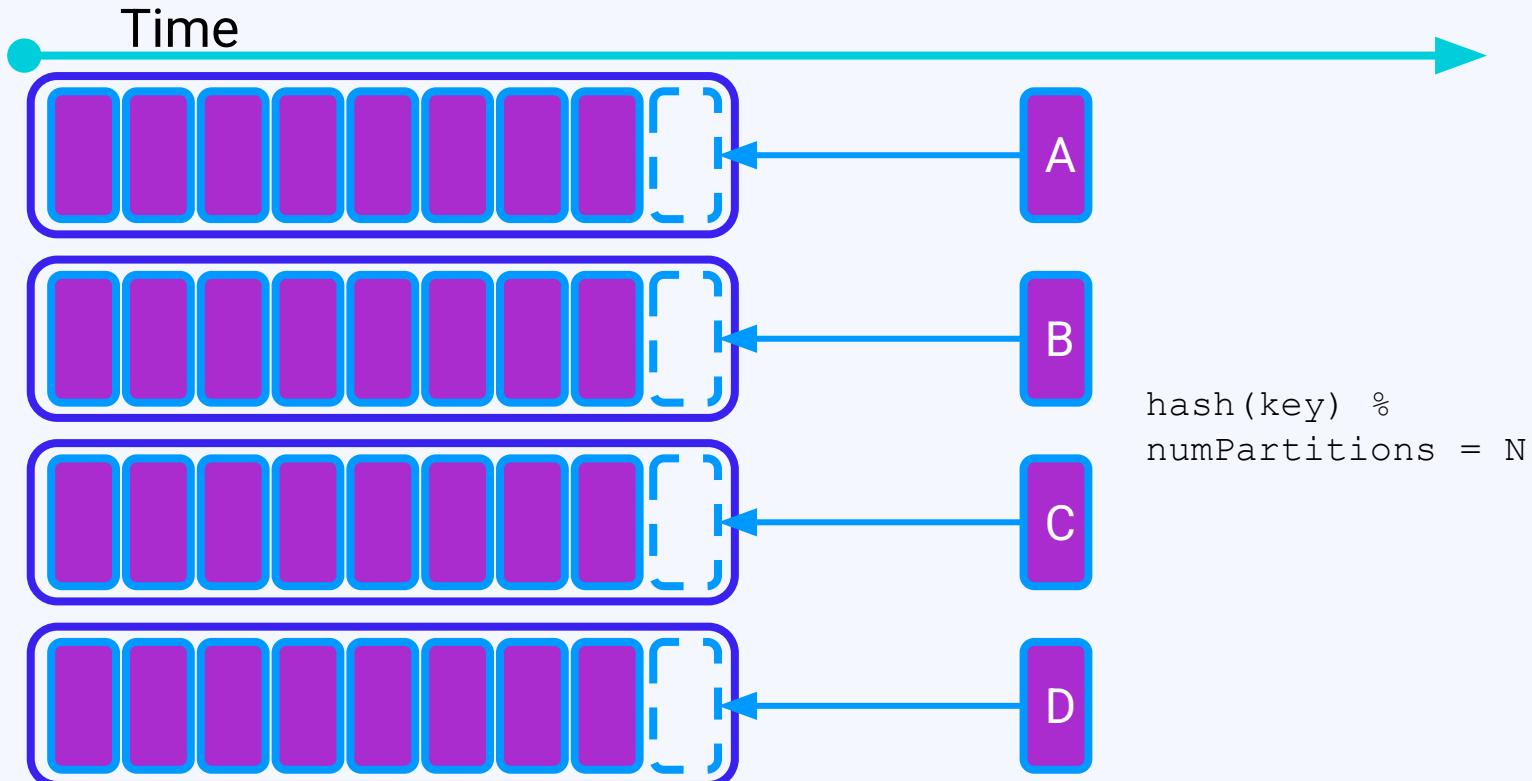


Producing to Kafka - No Key





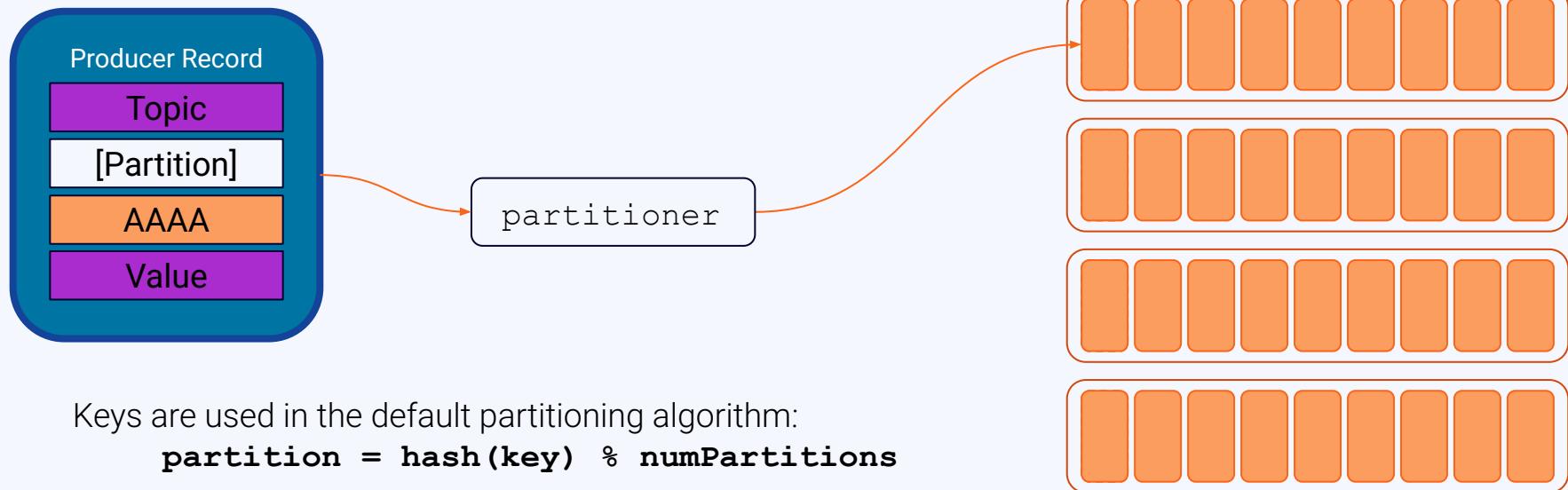
Producing to Kafka - With Key





Record Keys and why they're important - Ordering

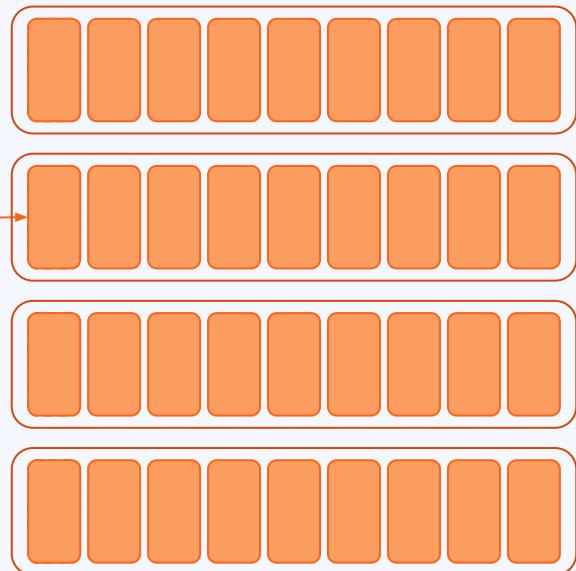
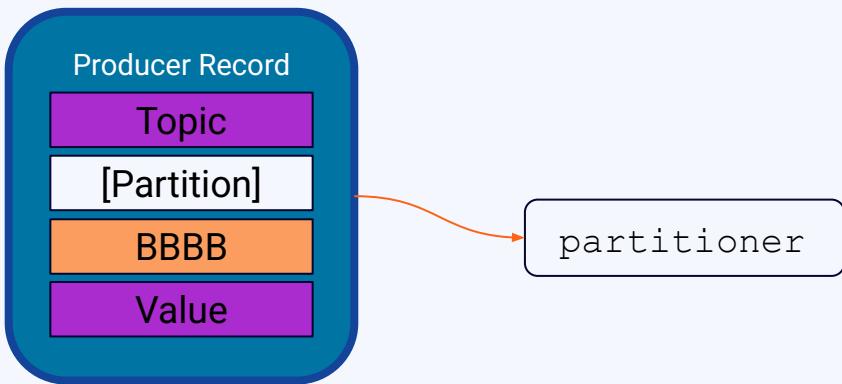
Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key



Record Keys and why they're important - Ordering



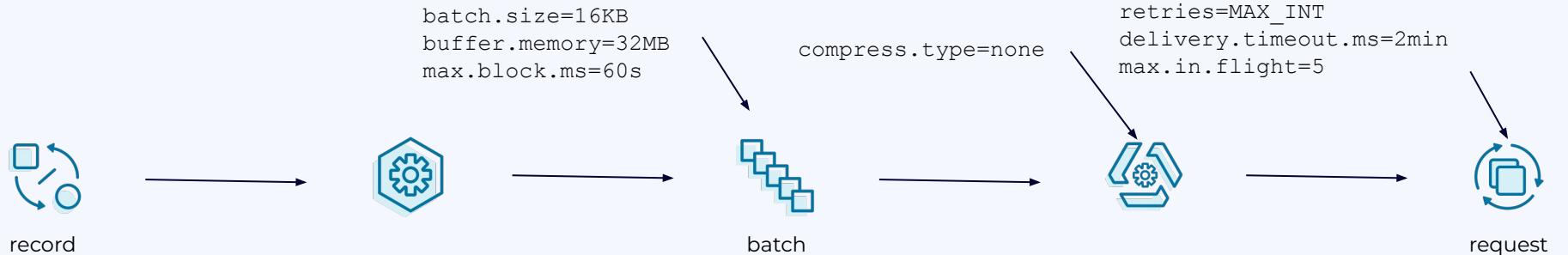
Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key



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

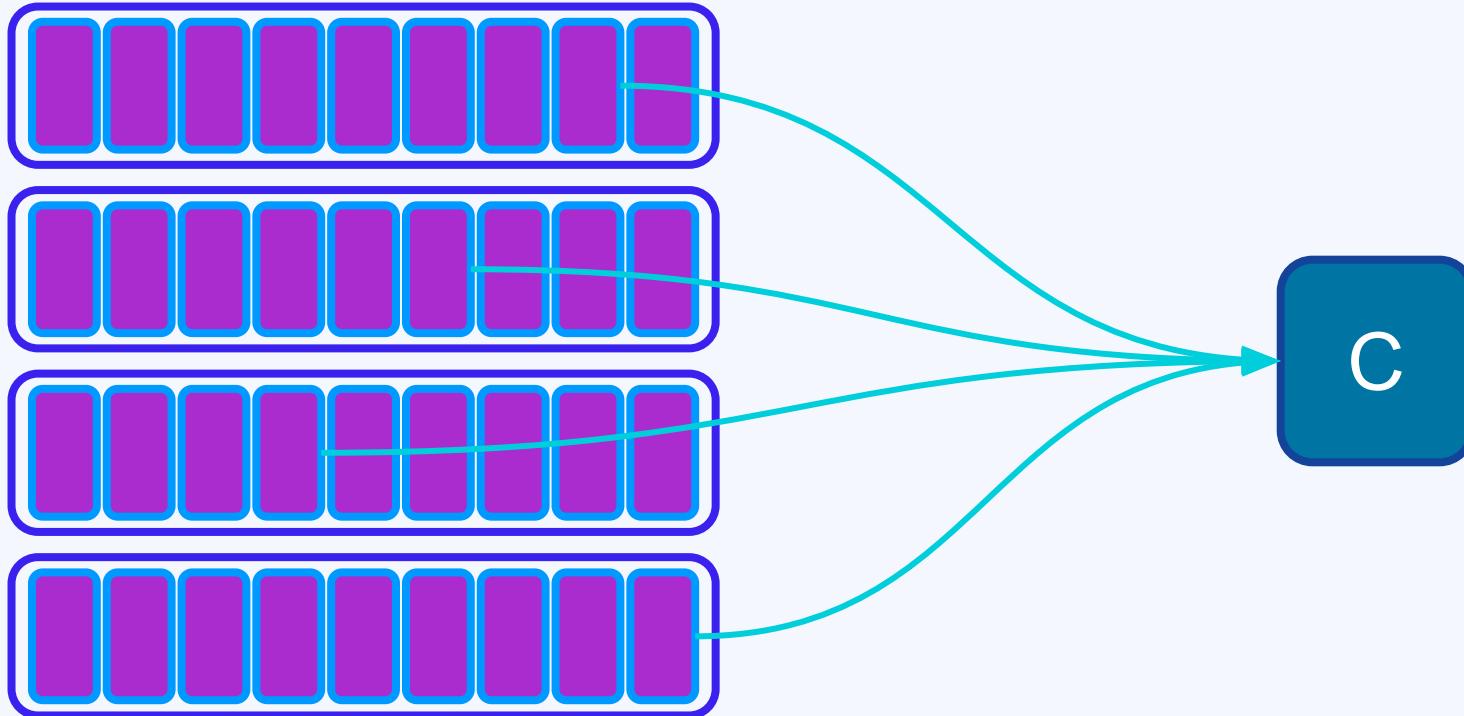


A basic Java Consumer

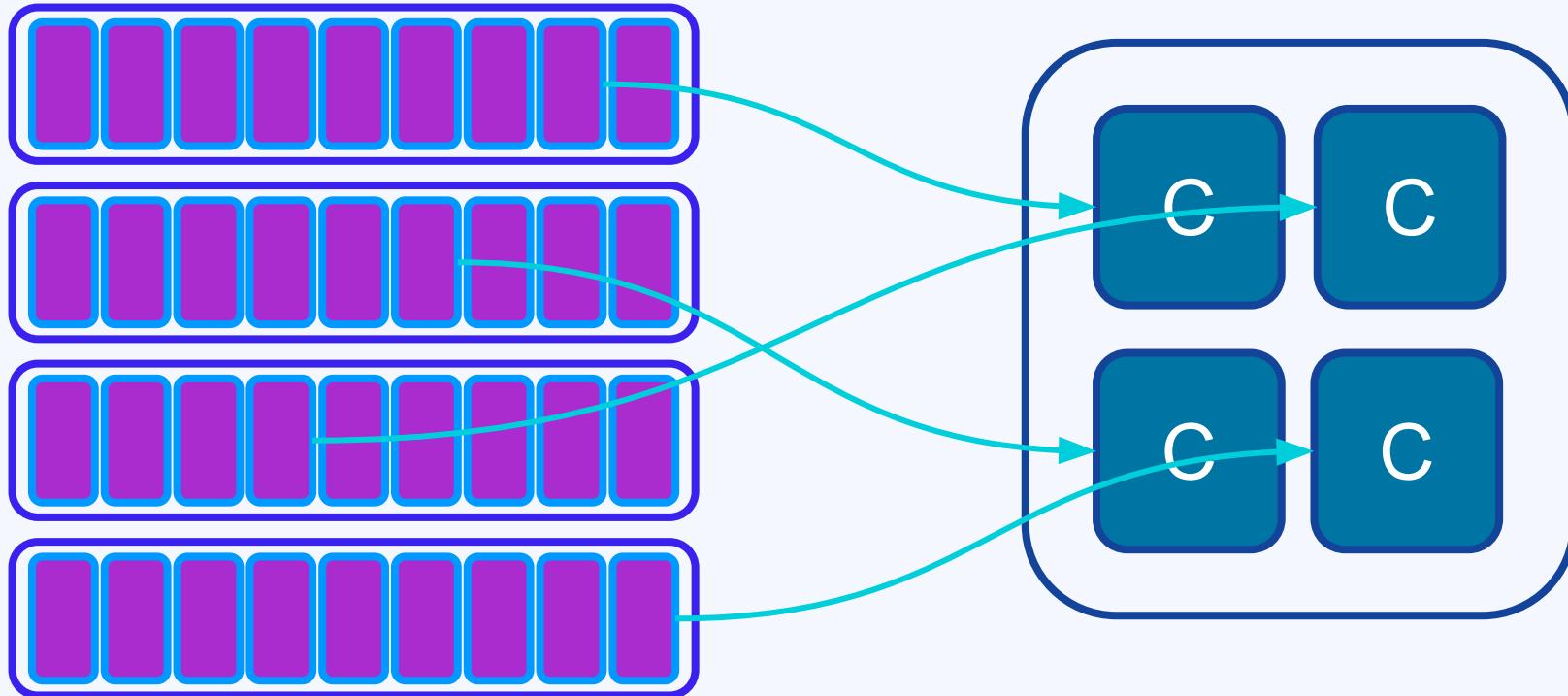
```
final Consumer<String, String> consumer = new KafkaConsumer<String, String>(props);
consumer.subscribe(Arrays.asList(topic));
try {
    while (true) {
        ConsumerRecords<String, String> records = consumer.poll(100);
        for (ConsumerRecord<String, String> record : records) {
           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



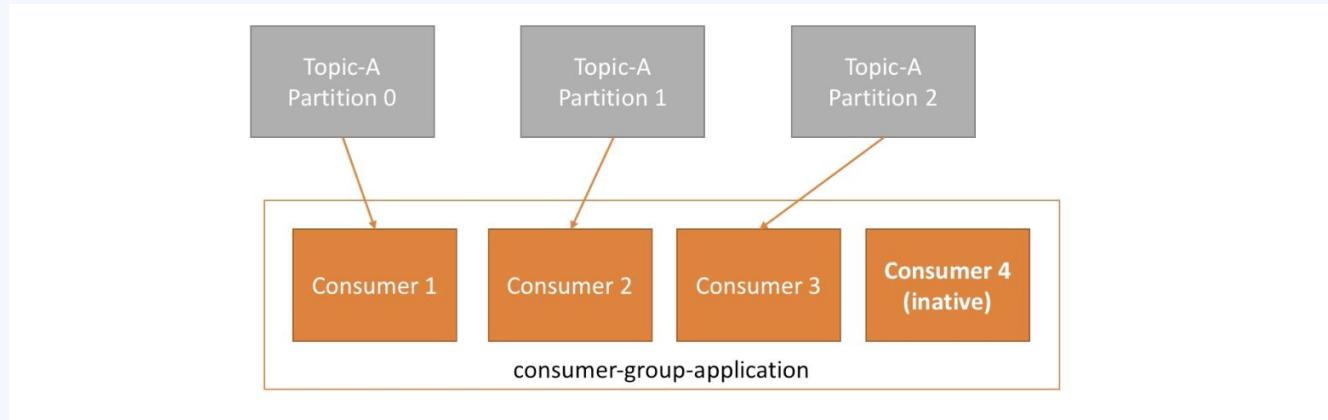
Consuming From Kafka - Single Consumer



Consuming From Kafka - Grouped Consumers

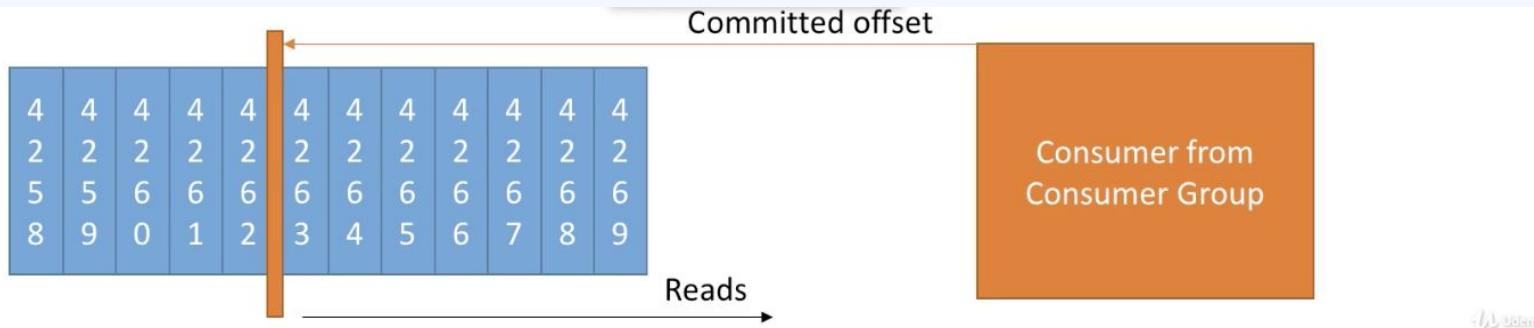


What if you have more Consumers than partitions?



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

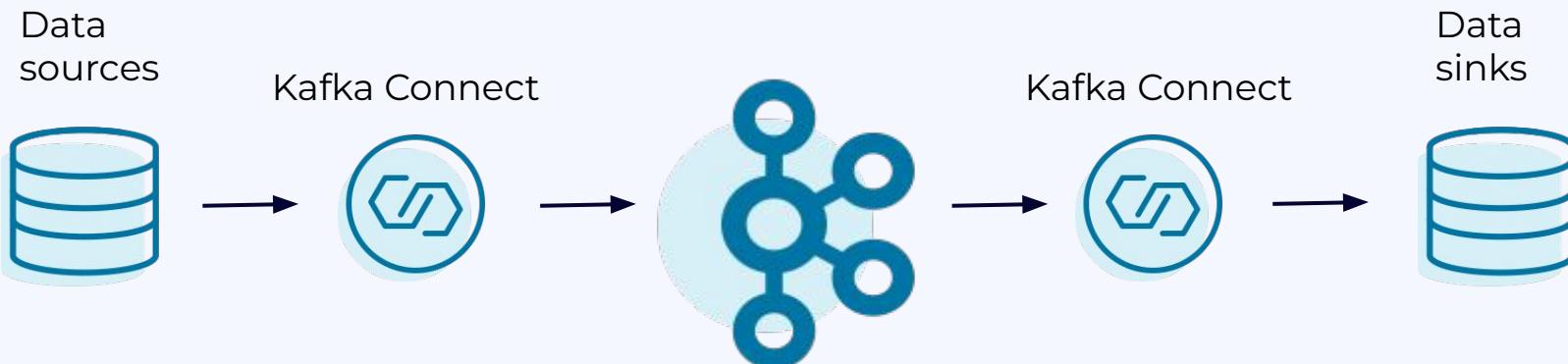
No/Low Code connectivity to many systems



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

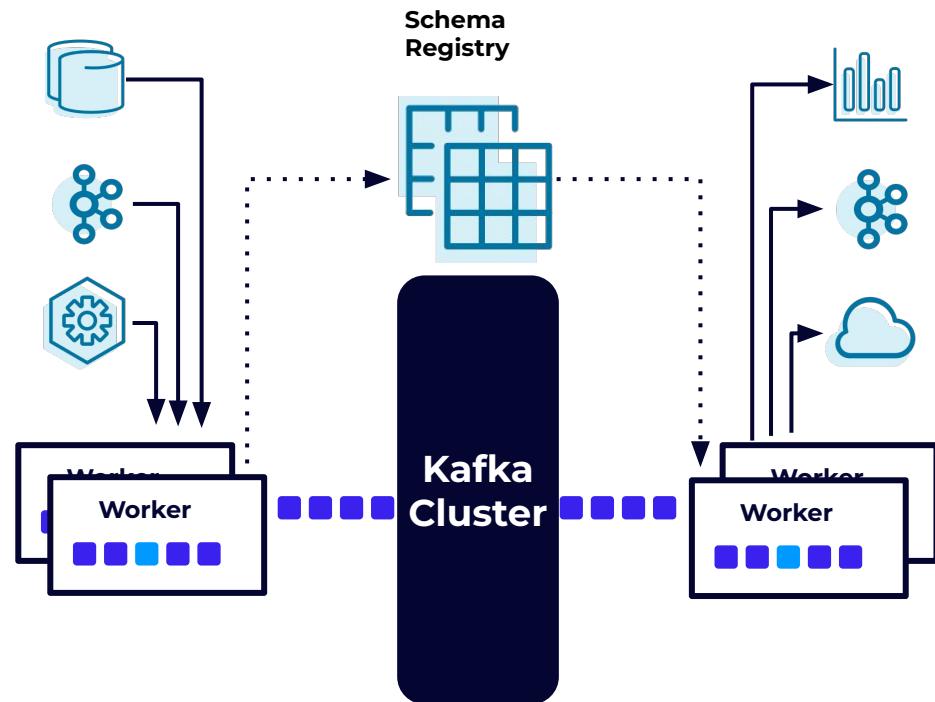


Kafka Connect Durable Data Pipelines

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

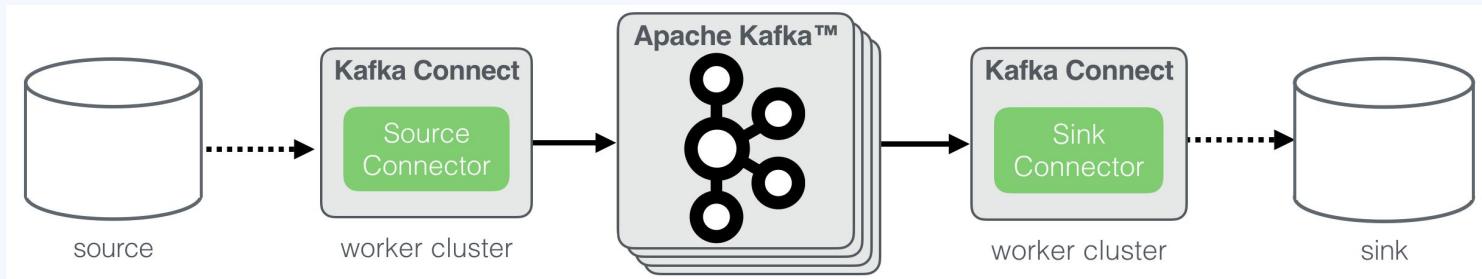
Kafka Connect



Kafka Connect



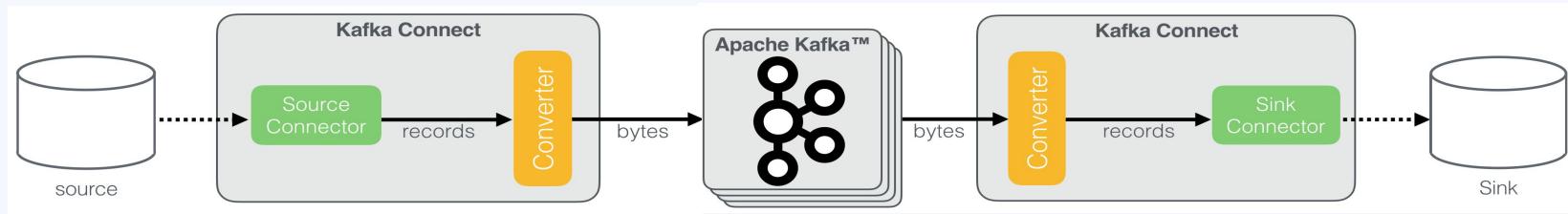
Connectors are reusable components that know how to talk to specific **sources and sinks**.



Kafka Connect



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

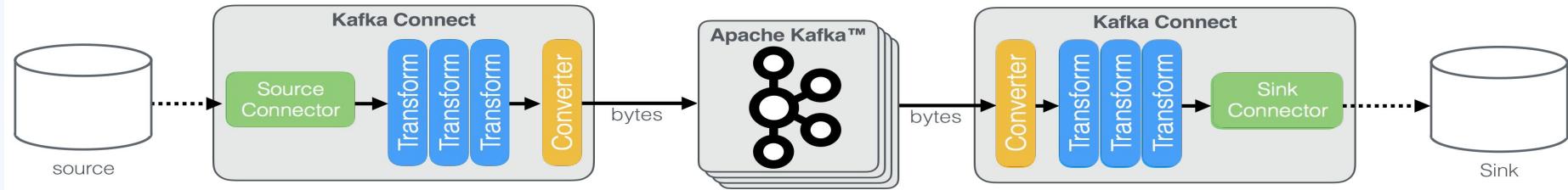


JSON, Avro, and others

Kafka Connect



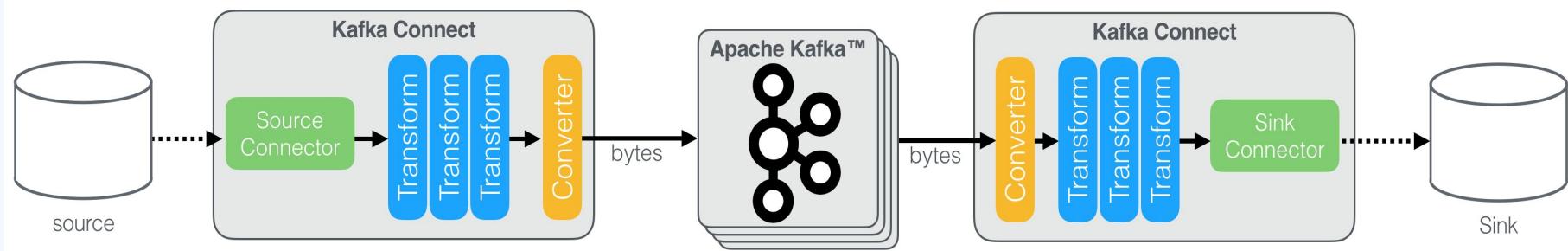
Connectors, transforms, and converters
are **pluggable components**.



Kafka Connect



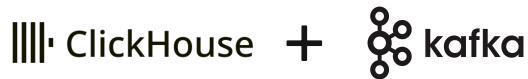
Modify the structure of keys and values, topics, and partition of source and sink record objects.





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

The screenshot shows the ClickHouse Kafka Connect Sink interface. It includes sections for the JDBC Connector (Source and Sink) and the HTTP Sink Connector. Both sections provide download links for Java JDBC Version 10.7.4 and Confluent Cloud versions 1.7.4 and 0.0.17 respectively. The JDBC connector also indicates it is available fully managed on Confluent Cloud. The HTTP sink connector indicates it is available fully managed on Confluent Cloud. The ClickHouse Kafka Connect Sink section describes it as the official Kafka Connect Sink connector for ClickHouse, provides installation instructions using the Confluent Hub CLI, and includes a download link for the ZIP file.

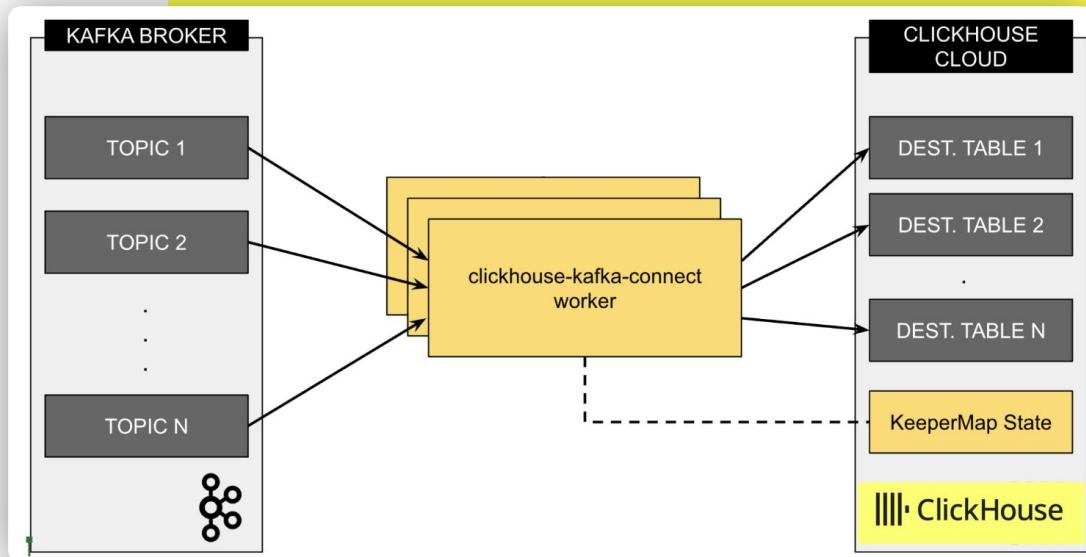
03 ClickPipes

The screenshot shows the ClickPipes integration engine interface. It features a central ClickPipes icon with a yellow bar chart icon. To the left is a grid of icons representing various data sources: Apache Kafka, Confluent Cloud, Amazon MSK, Amazon Kinesis, Redpanda, WarpStream, Azure Event Hubs, and Amazon S3. Below the grid, there is a banner for the ClickPipes connector for PostgreSQL CDC, which is now in Private Preview. The banner includes a "Get started today" button and a "View documentation" button. The overall background is dark with a grid pattern.

The screenshot shows the ClickPipes setup interface. It starts with a "Select the data source" step, listing various options like Apache Kafka, Confluent Cloud, Amazon MSK, Amazon Kinesis, Redpanda, WarpStream, Azure Event Hubs, and Amazon S3. Step 2: "Setup your ClickPipe Connection" shows a diagram of a ClickPipe with four stages: Incoming Data, Parse Information, Details and Settings, and a final stage. Step 3: "Incoming Data" and Step 4: "Parse Information" are shown below the diagram. Step 5: "Details and Settings" is at the bottom. A note at the top right says: "If you need any help to setup ClickPipes, you can access the documentation for more details."

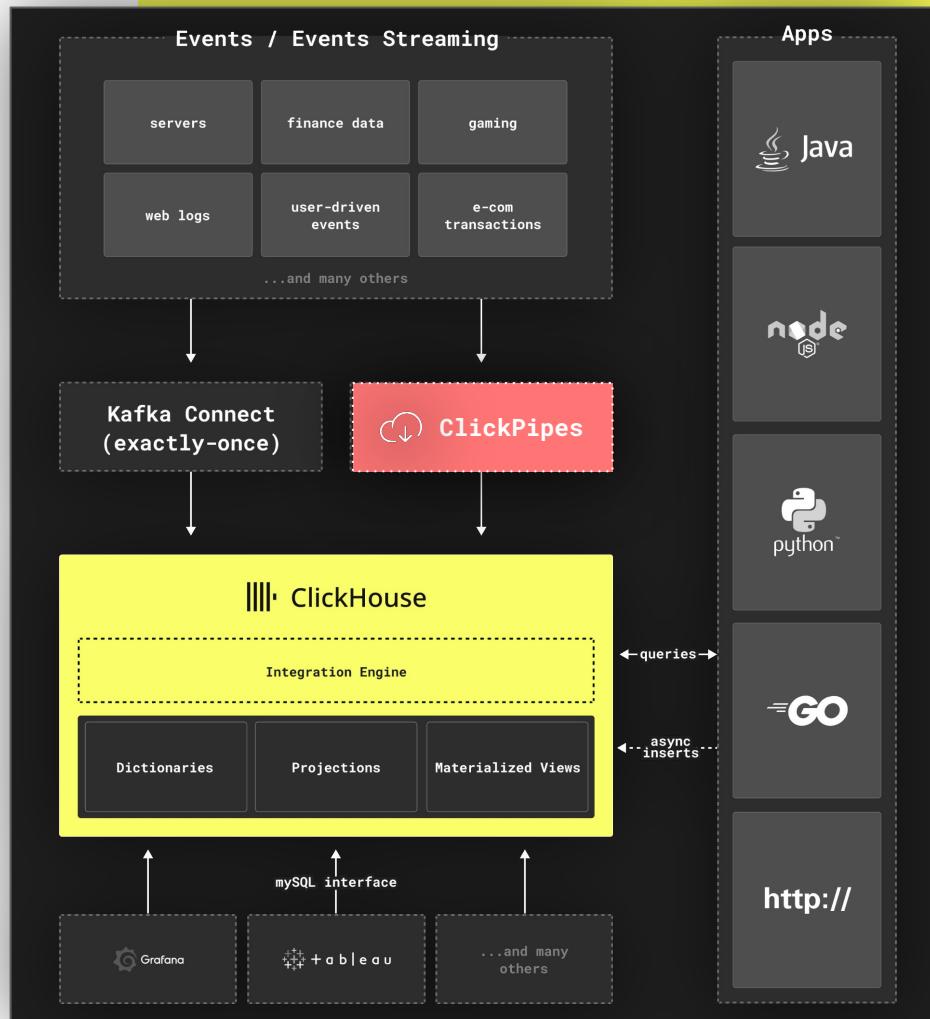
ClickHouse Kafka Connect Sink

- 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*
/klik paips/

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



ClickPipes for Confluent Cloud

ClickHouse RD

Data sources / ClickPipes / New

1 Select the data source

If you need any help to setup ClickPipes, you can [access the documentation](#) for more details.

Apache Kafka Confluent Cloud Amazon MSK Amazon Kinesis

Redpanda WarpStream Azure Event Hubs Amazon S3

Google Cloud Storage Postgres CDC

2 Setup your ClickPipe Connection

3 Incoming Data

4 Parse Information

5 Details and Settings

Organization

Integrations team >

Integrations Chat with support All systems operational

Select the data source

Setup your ClickPipe Connection

3 Incoming Data

Topic: topic_0 Offset selection (Beta): From beginning

We expect JSON format If a schema registry is not provided, we expect JSON format in

Sample Data

Sample displayed from partition: 4 Offset: 0 Timestamp: 1732881515572

```
{"_time": 231, "agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36", "bytes": "278", "ip": "223.145.8.144", "referrer": "-", "remote_user": "-", "request": "GET /site/user_status.html HTTP/1.1", "status": "404", "time": "231", "userid": 1}
```

Upload data to

New table Existing table

Database: default Name: topic_0

Sorting key:

Select an option

Column settings

Name	Type	Default value	Nullable
_time	Int64		✓
agent	String		✓
bytes	Int64		✓
ip	String		✓
referrer	String		✓
remote_user	String		✓
request	String		✓
status	Int64		✓
time	Int64		✓
userid	Int64		✓

Back Next: Parse Information

Parse Information

Details and Settings

4 Incoming Data

5 Parse Information

Read Docs

Read Docs

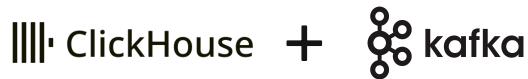
Back Next: Incoming Data

Incoming Data

Parse Information

Back Next: Details and Settings

Advanced settings



Improvement s

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

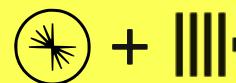
- [Improvement] Introduce separate consumer/producer tags for the **Kafka** configuration. This avoids warnings from **librdkafka** that consumer properties were specified for producer instances and vice versa (e.g. Configuration property `session.timeout.ms` is a consumer property and will be ignored 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 \(János Benjamin Antal\)](#).



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';
```

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





Questions