

Confluent KAFKA Administration

Rajesh Pasham

What is Confluent Platform?

- ▶ Confluent Platform is a full-scale data streaming platform that enables you to easily access, store, and manage data as continuous, real-time streams.
- ▶ Built by the original creators of Apache Kafka®, Confluent expands the benefits of Kafka with enterprise-grade features while removing the burden of Kafka management or monitoring.
- ▶ Today, over 80% of the Fortune 100 are powered by data streaming technology – and the majority of those leverage Confluent.

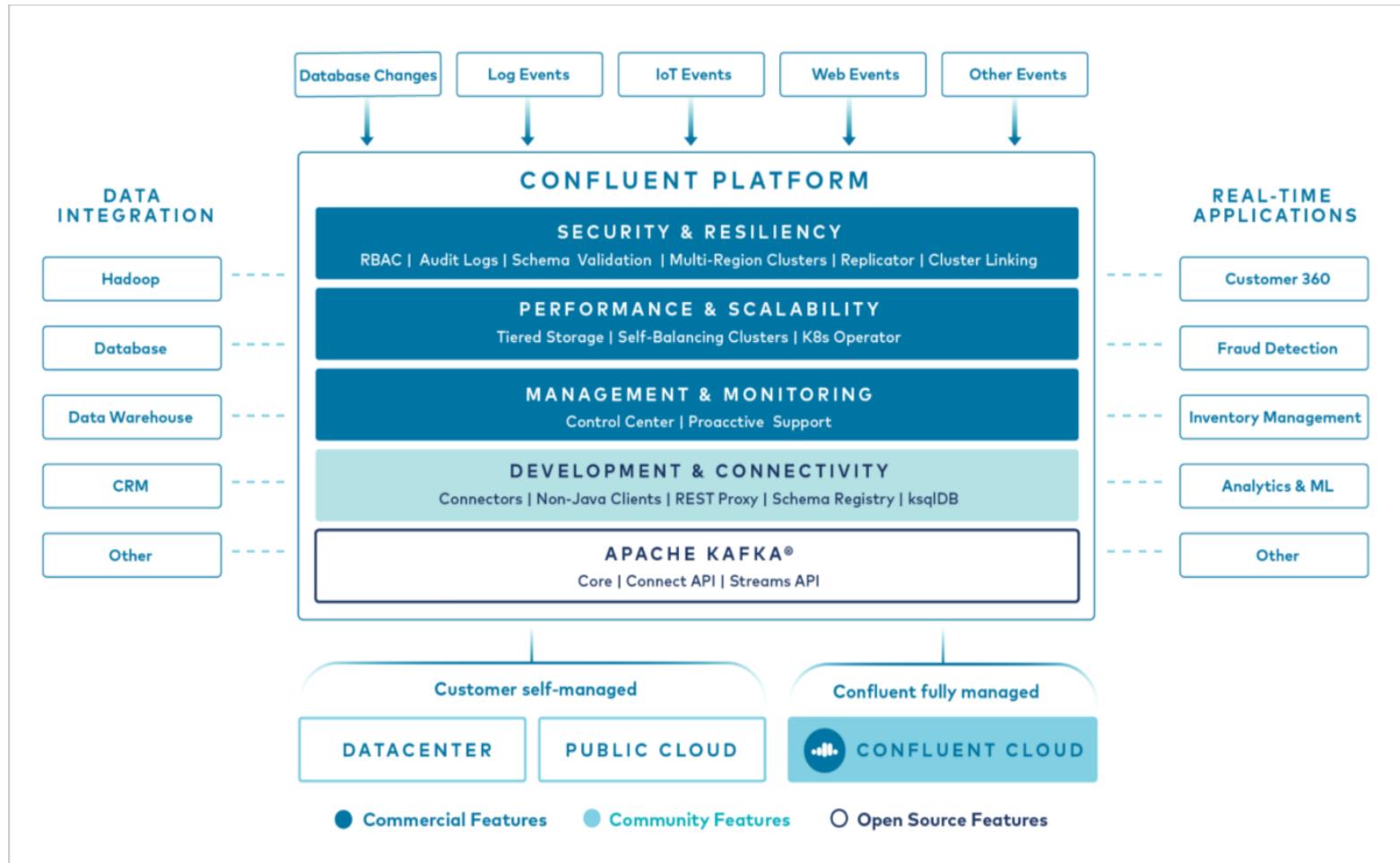
Why Confluent?

- ▶ By integrating historical and real-time data into a single, central source of truth, Confluent makes it easy to build an entirely new category of modern, event-driven applications, gain a universal data pipeline, and unlock powerful new use cases with full scalability, performance, and reliability.

What is Confluent Used For?

- ▶ Confluent Platform lets you focus on how to derive business value from your data rather than worrying about the underlying mechanics, such as how data is being transported or integrated between disparate systems.
- ▶ Specifically, Confluent Platform simplifies connecting data sources to Kafka, building streaming applications, as well as securing, monitoring, and managing your Kafka infrastructure.
- ▶ Today, Confluent Platform is used for a wide array of use cases across numerous industries, from financial services, omnichannel retail, and autonomous cars, to fraud detection, microservices, and IoT.

What is Confluent Used For?



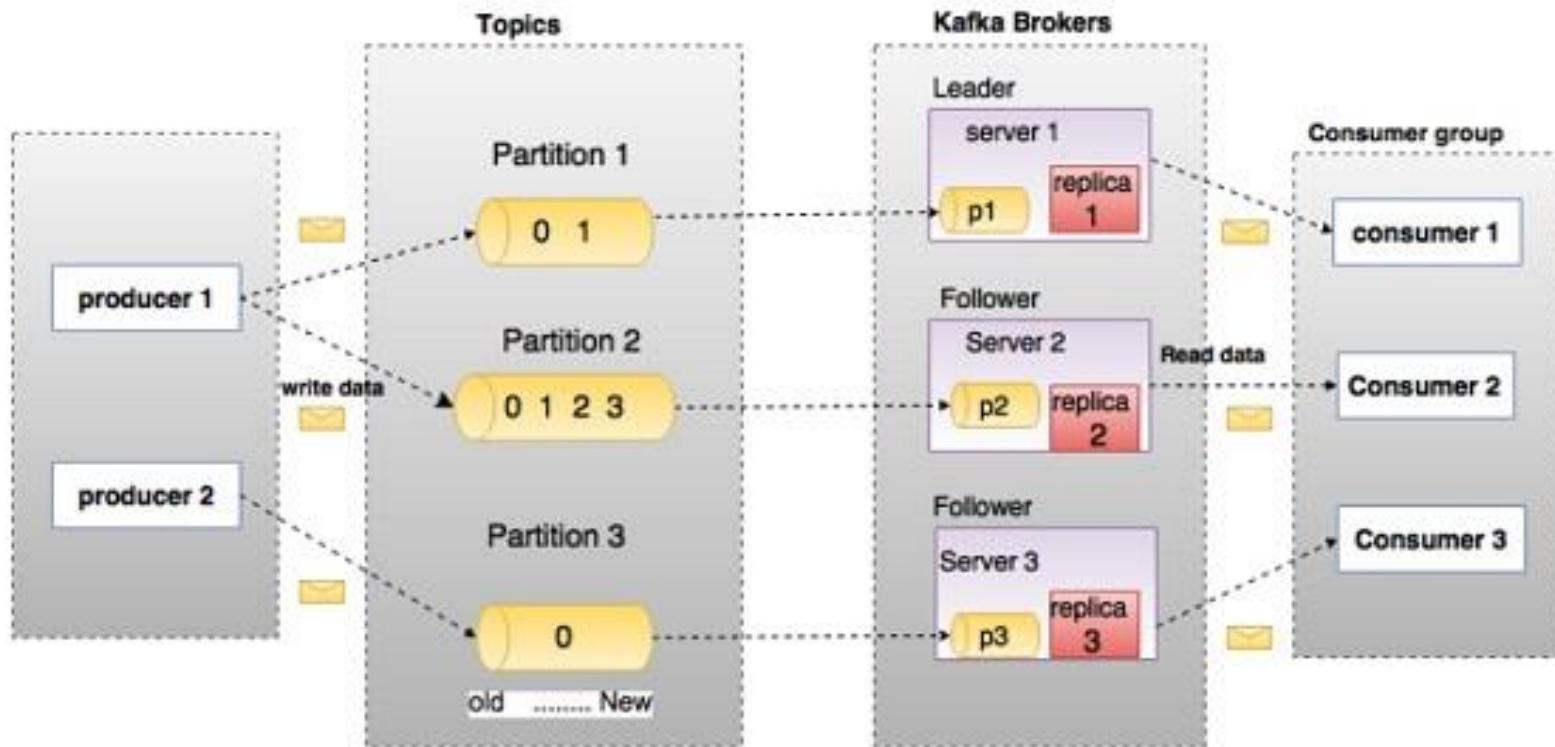
Confluent Platform Components

Overview of Confluent's Event Streaming Technology

- ▶ At the core of Confluent Platform is Apache Kafka, the most popular open source distributed streaming platform.
- ▶ The key capabilities of Kafka are:
 - Publish and subscribe to streams of records
 - Store streams of records in a fault tolerant way
 - Process streams of records
- ▶ Out of the box, Confluent Platform also includes Schema Registry, REST Proxy, a total of 100+ pre-built Kafka connectors, and ksqlDB.

Apache Kafka - Fundamentals

- The following diagram illustrates the main terminologies



Event

↓ Internet of Things

Event

Internet of Things
Business process change

Event

- ↓ Internet of Things
- ↓ Business process change
- ↓ User Interaction

Event

- ↓ Internet of Things
- ↓ Business process change
- ↓ User Interaction
- ↓ Microservice output

Notification

+

State

Key

Value



CONFLUENT

Developer

Topics

Topics



Topics

Named container for similar events

Topics

- ⬇️ Named container for similar events
- System contains lots of topics
- Can duplicate data between topics

Topics

- ⬇️ Named container for similar events
- System contains lots of topics
- Can duplicate data between topics
- ⬇️ Durable logs of events

Topics

- ↳ Named container for similar events
- System contains lots of topics
- Can duplicate data between topics
- ↳ Durable logs of events
- Append only

Topics

- ↳ Named container for similar events
- System contains lots of topics
- Can duplicate data between topics

- ↳ Durable logs of events
- Append only
- Can only seek by offset, not indexed

Topics

- ⬇️ Named container for similar events
- System contains lots of topics
- Can duplicate data between topics

- ⬇️ Durable logs of events
- Append only
- Can only seek by offset, not indexed

- ⬇️ Events are immutable

Topics are durable

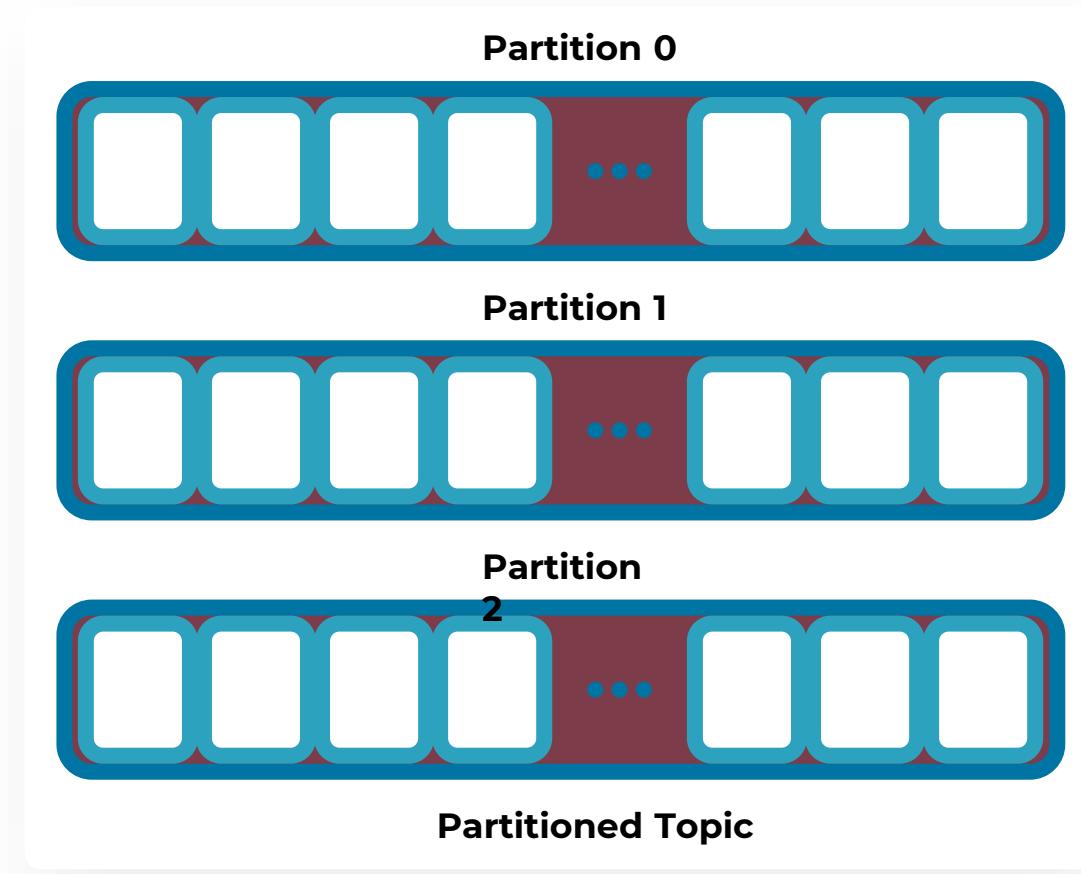
Retention period is configurable

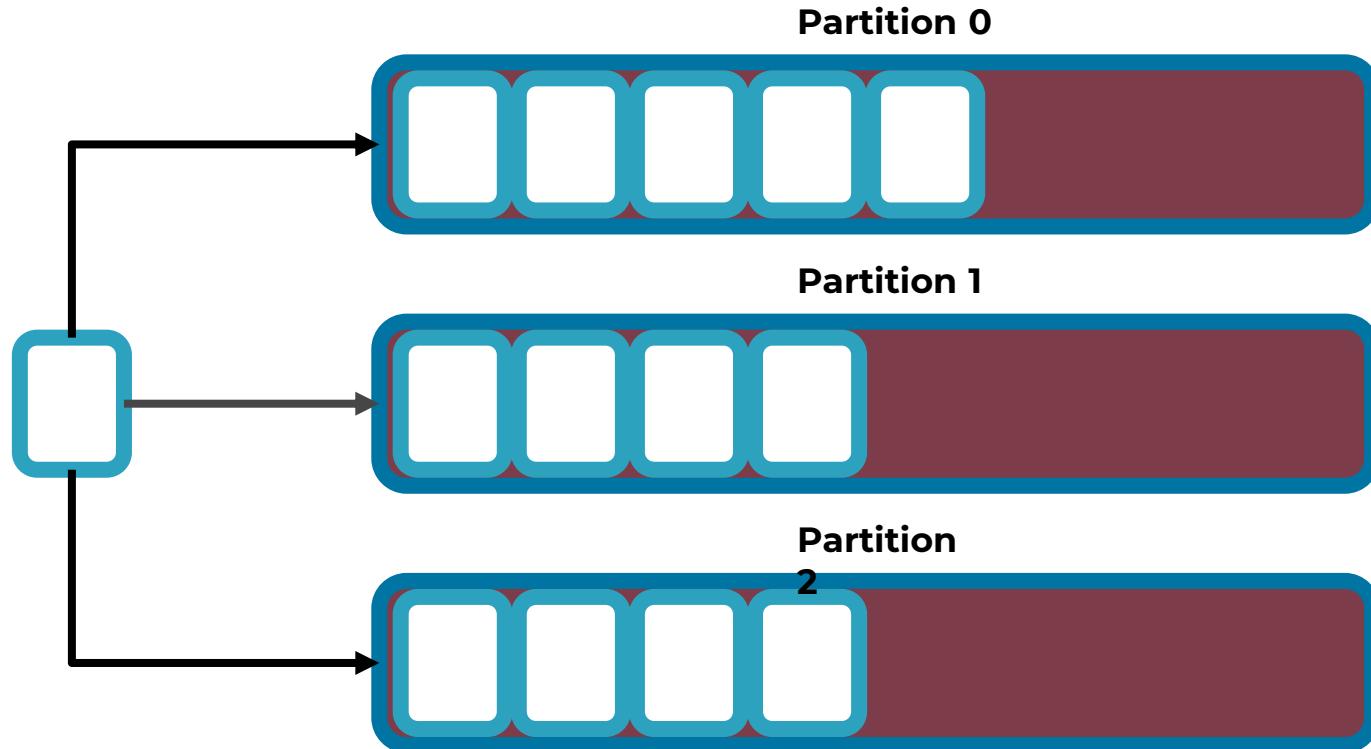


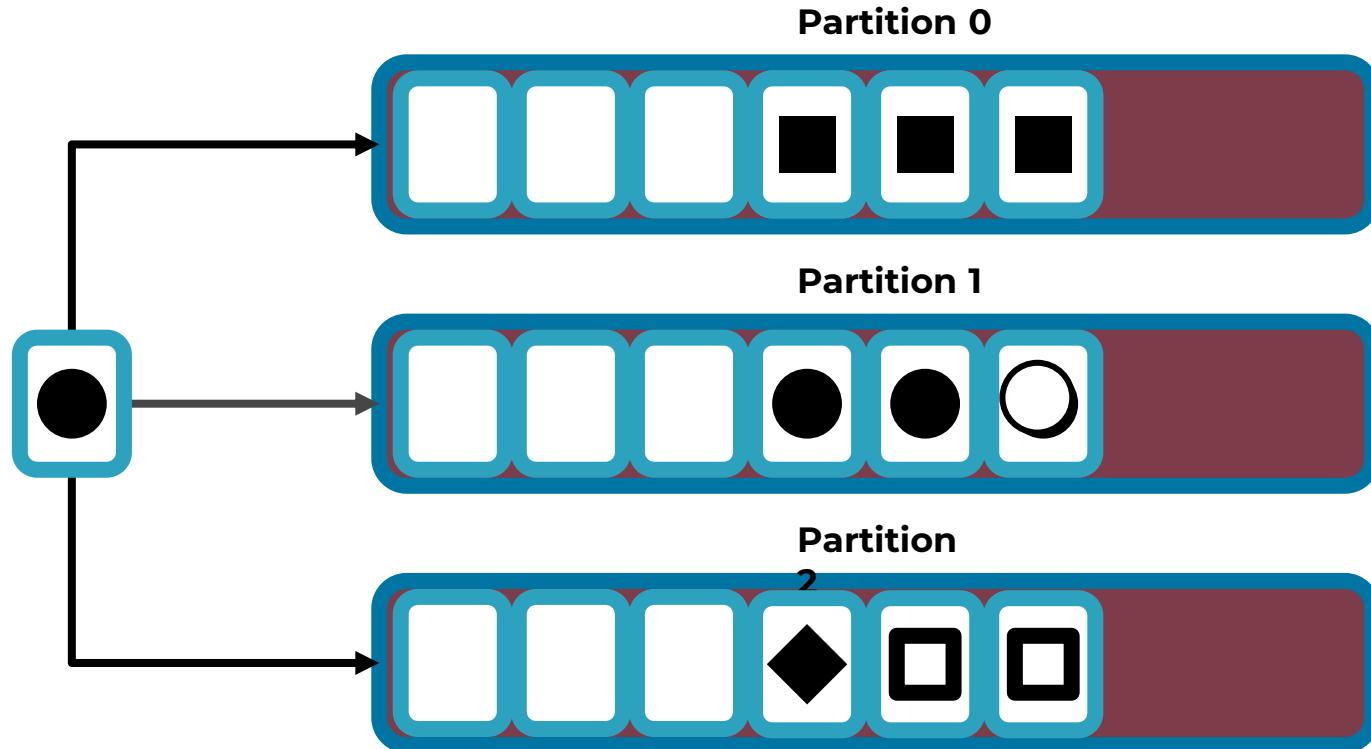
CONFLUENT

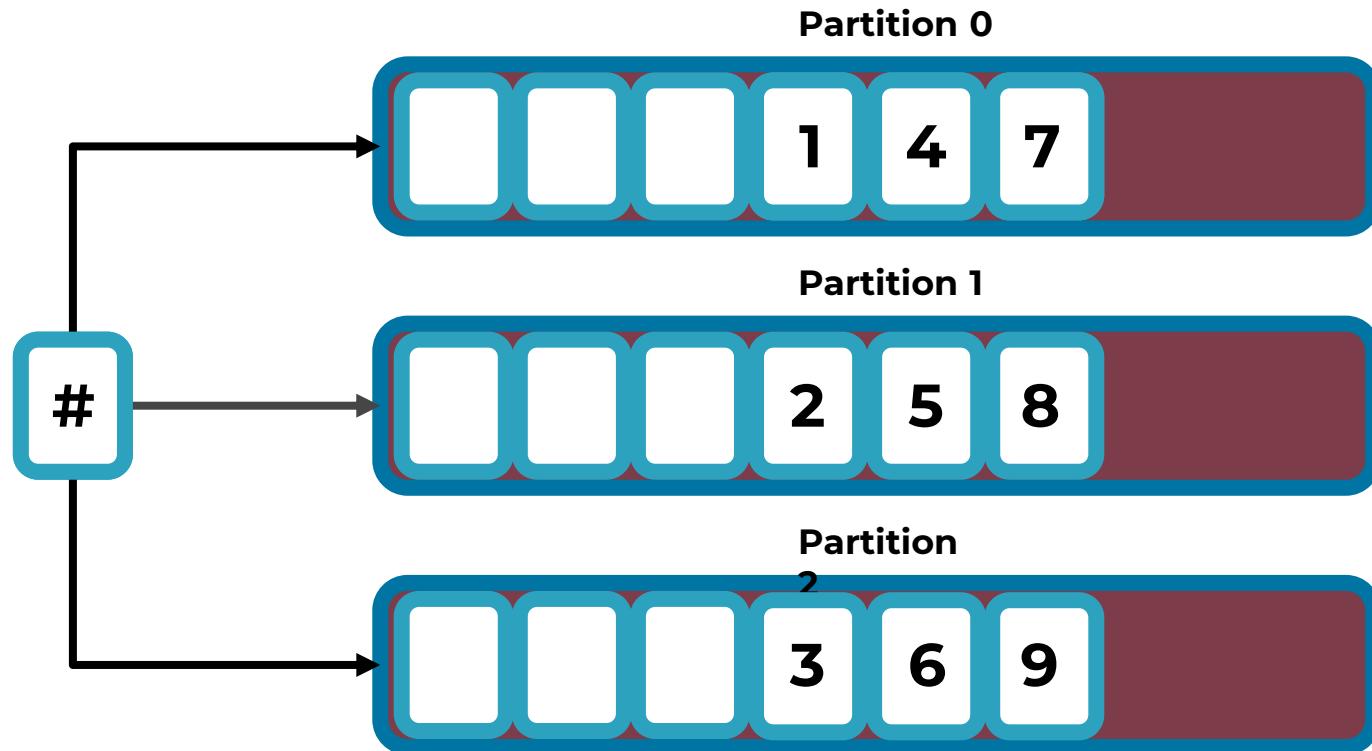
Developer

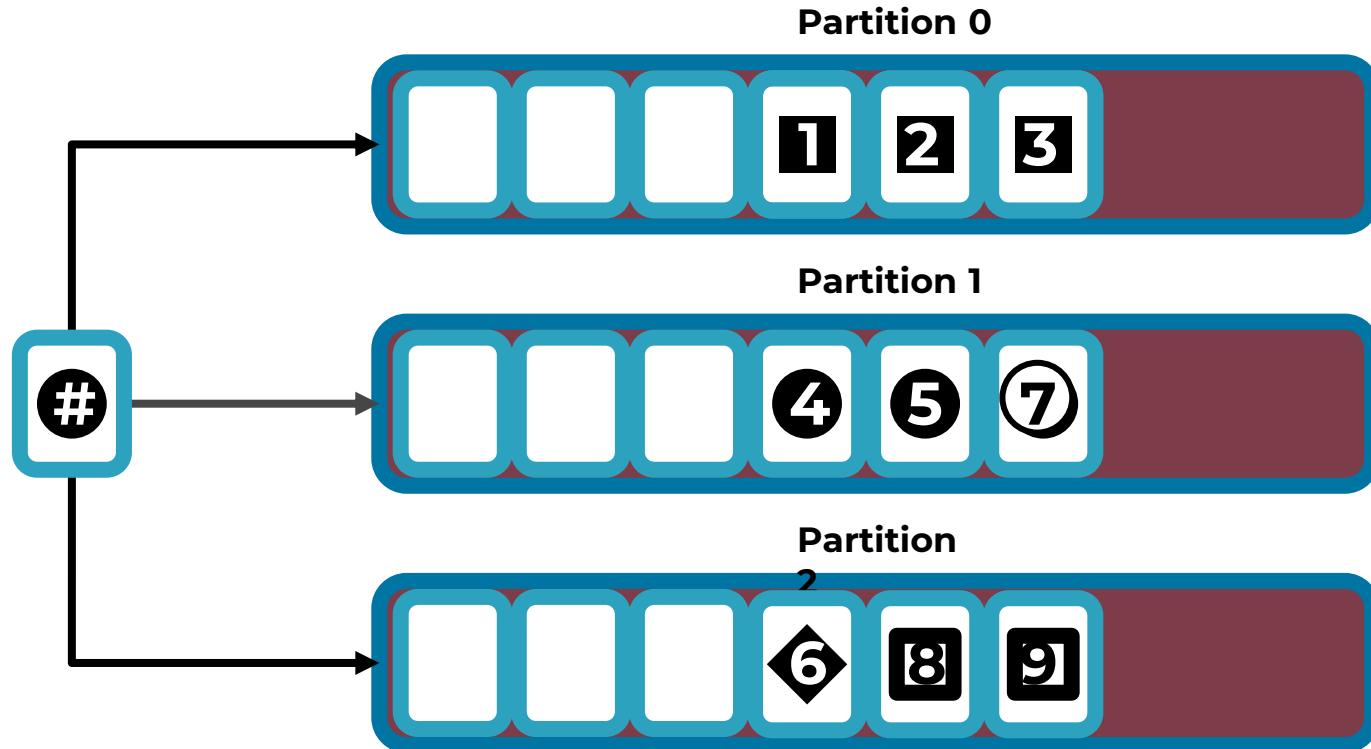
Partitioning

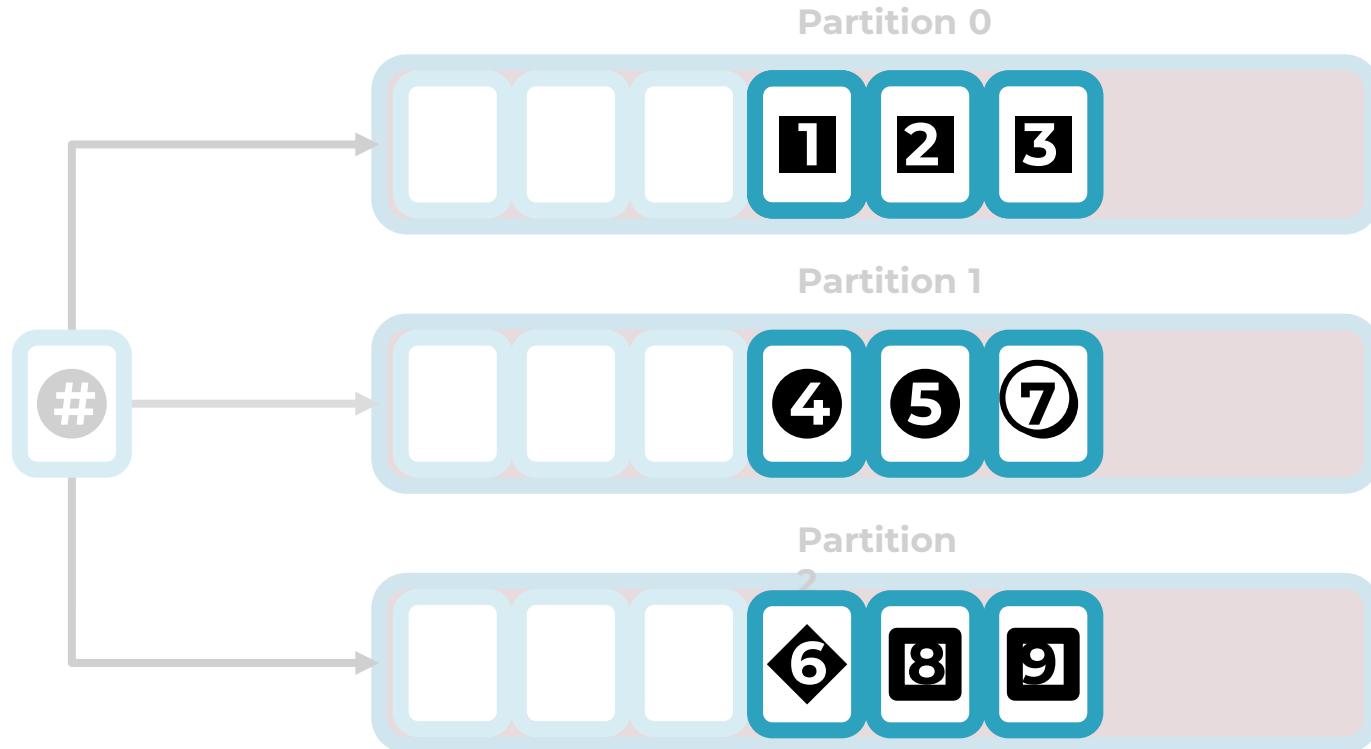














CONFLUENT
Developer

Brokers

Brokers



An computer, instance, or container running the Kafka process

Brokers

- ↓ An computer, instance, or container running the Kafka process
- ↓ Manage partitions
- ↓ Handle write and read requests

Brokers

- ↓ An computer, instance, or container running the Kafka process
- ↓ Manage partitions
- ↓ Handle write and read requests
- ↓ Manage replication of partitions

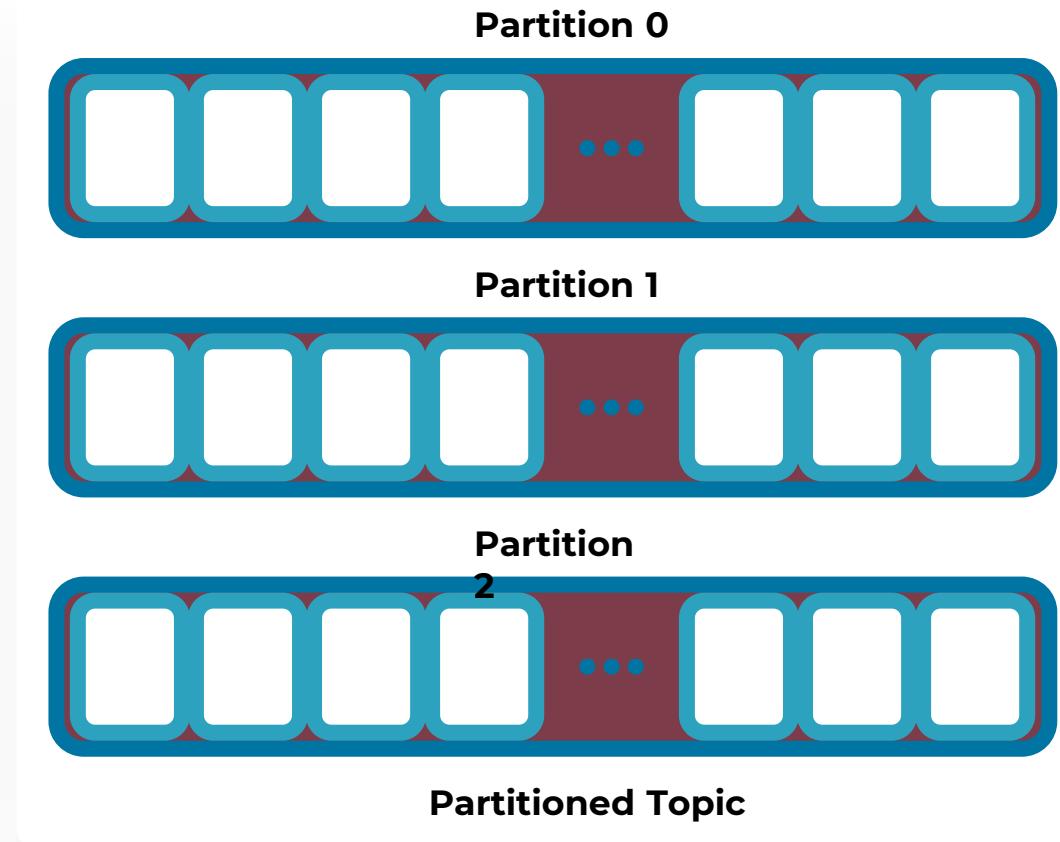
Brokers

- ↓ An computer, instance, or container running the Kafka process
- ↓ Manage partitions
- ↓ Handle write and read requests
- ↓ Manage replication of partitions
- ↓ Intentionally very simple



CONFLUENT
Developer

Replication



Replication



Copies of data for fault tolerance

Replication



Copies of data for fault tolerance

One lead partition and N-1 followers

Replication

- ↓ Copies of data for fault tolerance
- ↓ One lead partition and N-1 followers
- ↓ In general, writes and reads happen to the leader

Replication

- ↓ Copies of data for fault tolerance
- ↓ One lead partition and N-1 followers
- ↓ In general, writes and reads happen to the leader
- ↓ An invisible process to most developers

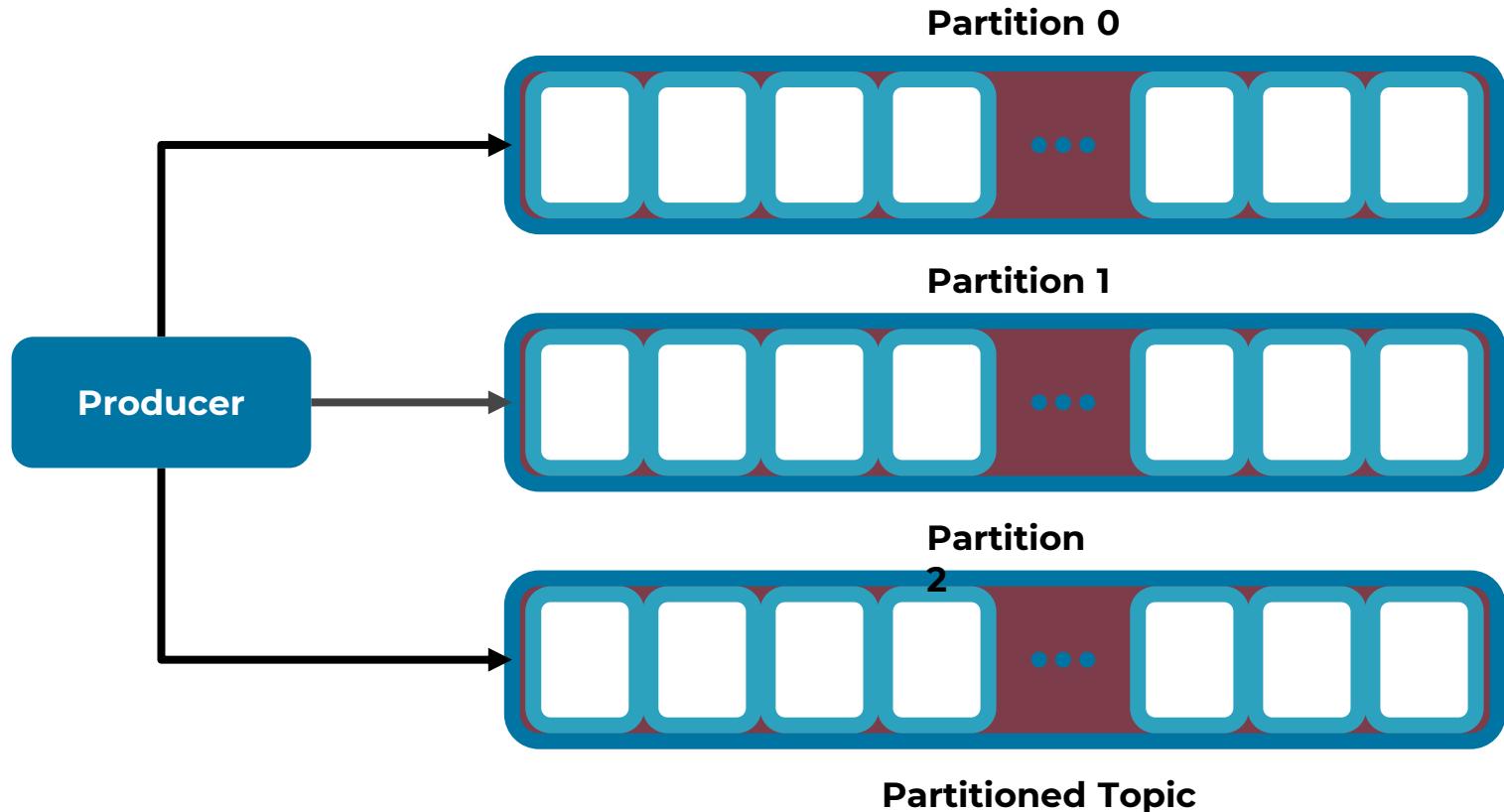
Replication

- ↓ Copies of data for fault tolerance
- ↓ One lead partition and N-1 followers
- ↓ In general, writes and reads happen to the leader
- ↓ An invisible process to most developers
- ↓ Tunable in the producer



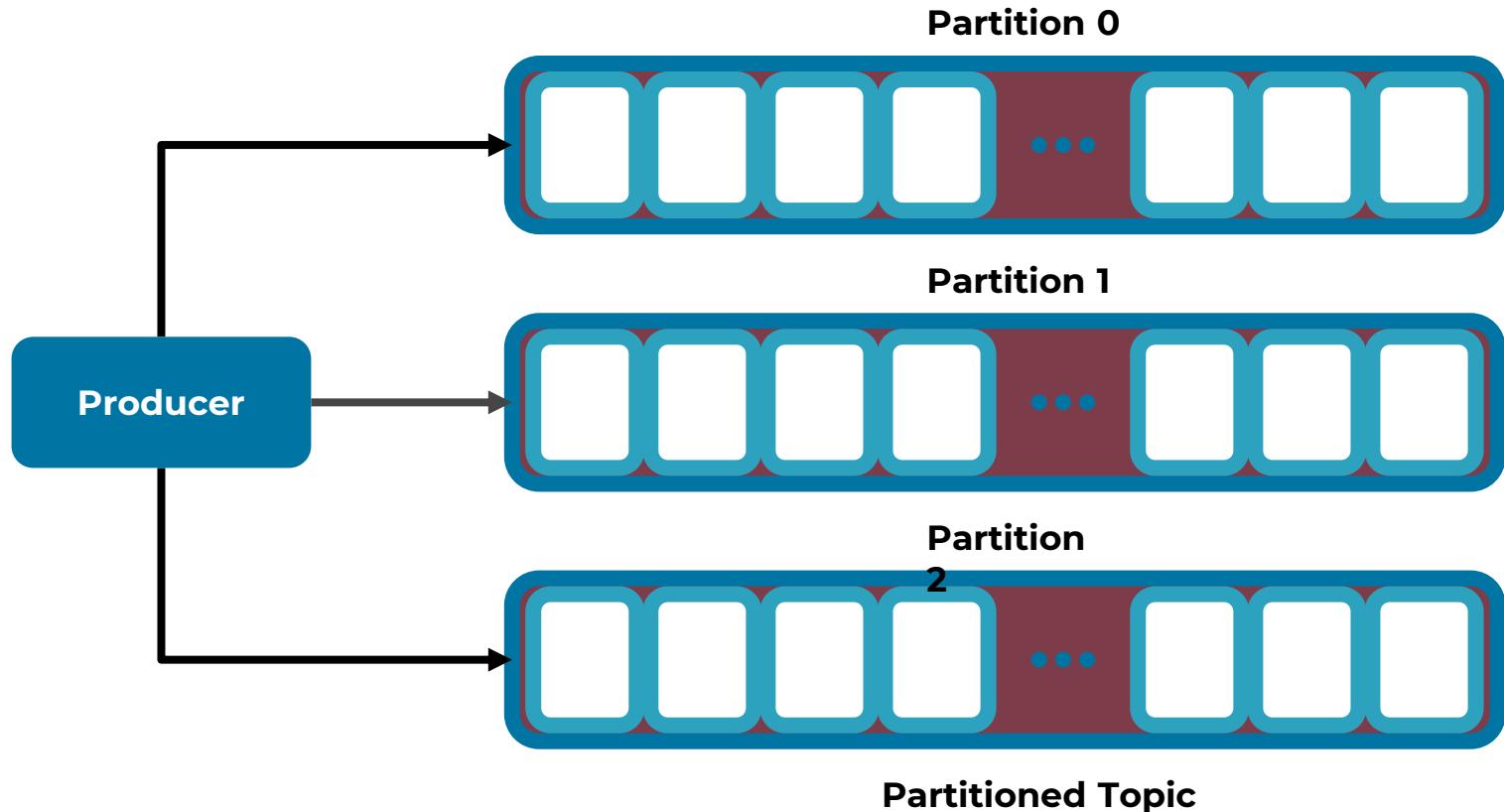
CONFLUENT
Developer

Producers



Producers

- Client application
- Puts messages into topics
- Connection pooling
- Network buffering
- Partitioning



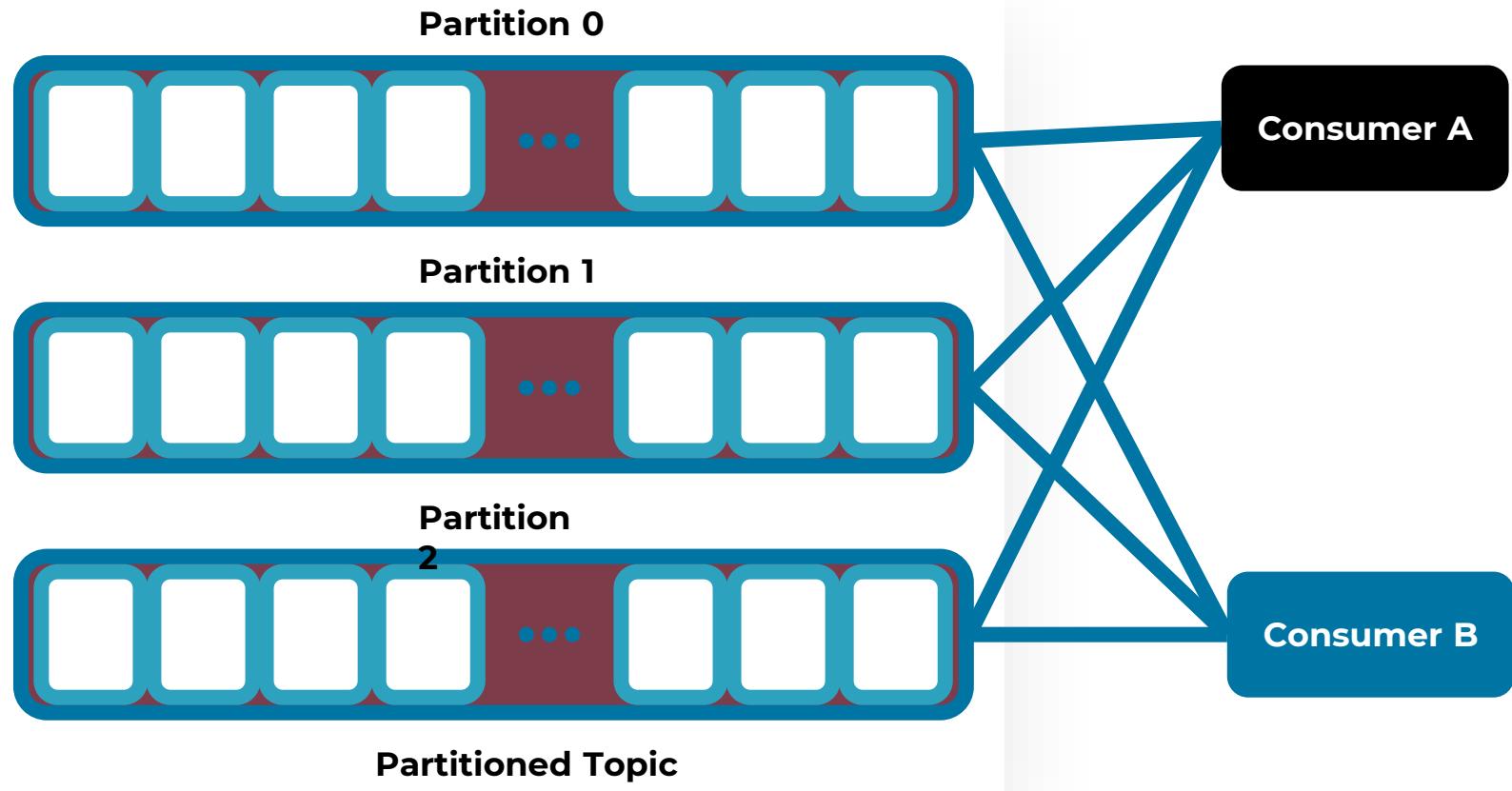


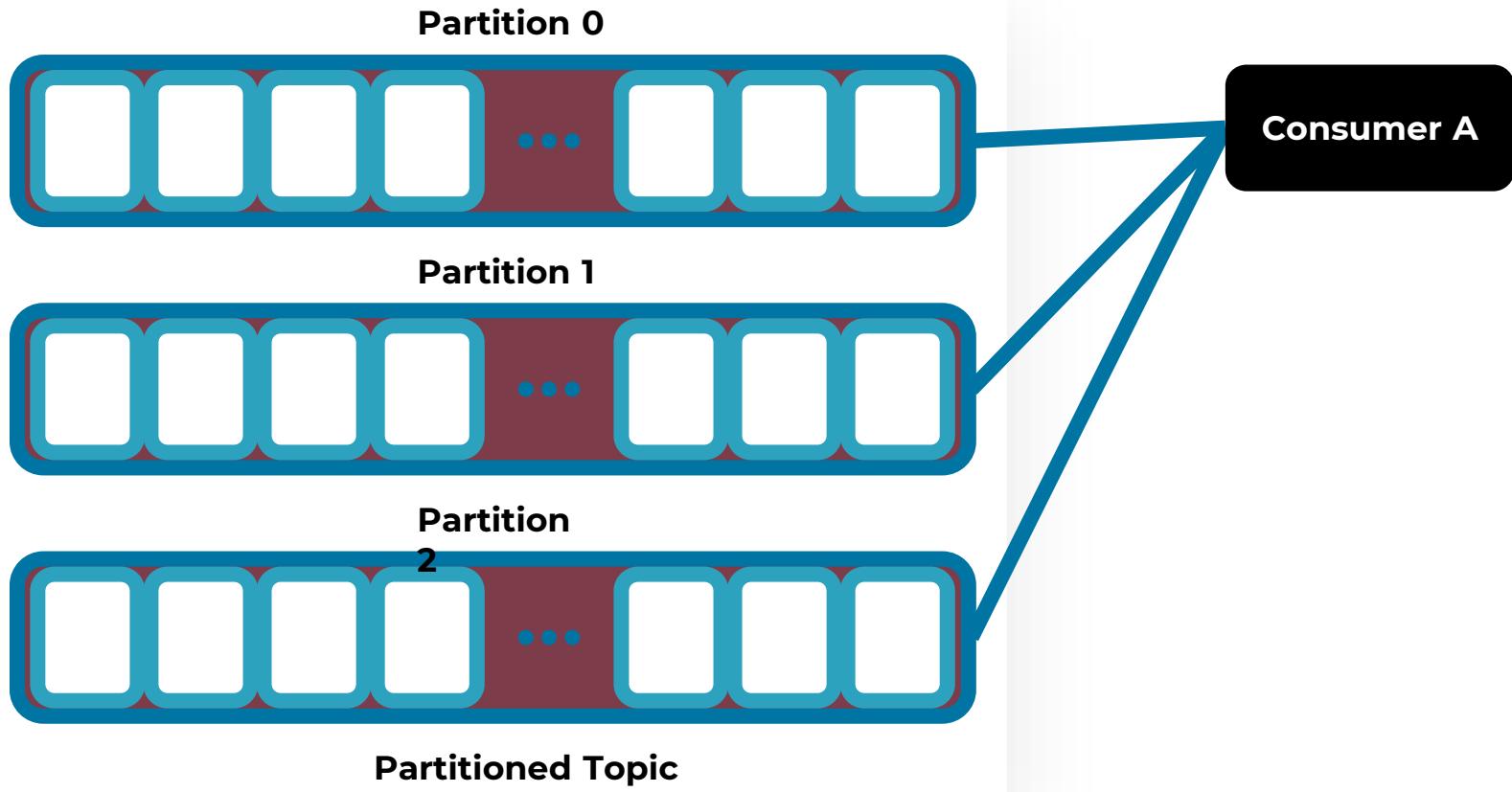
CONFLUENT
Developer

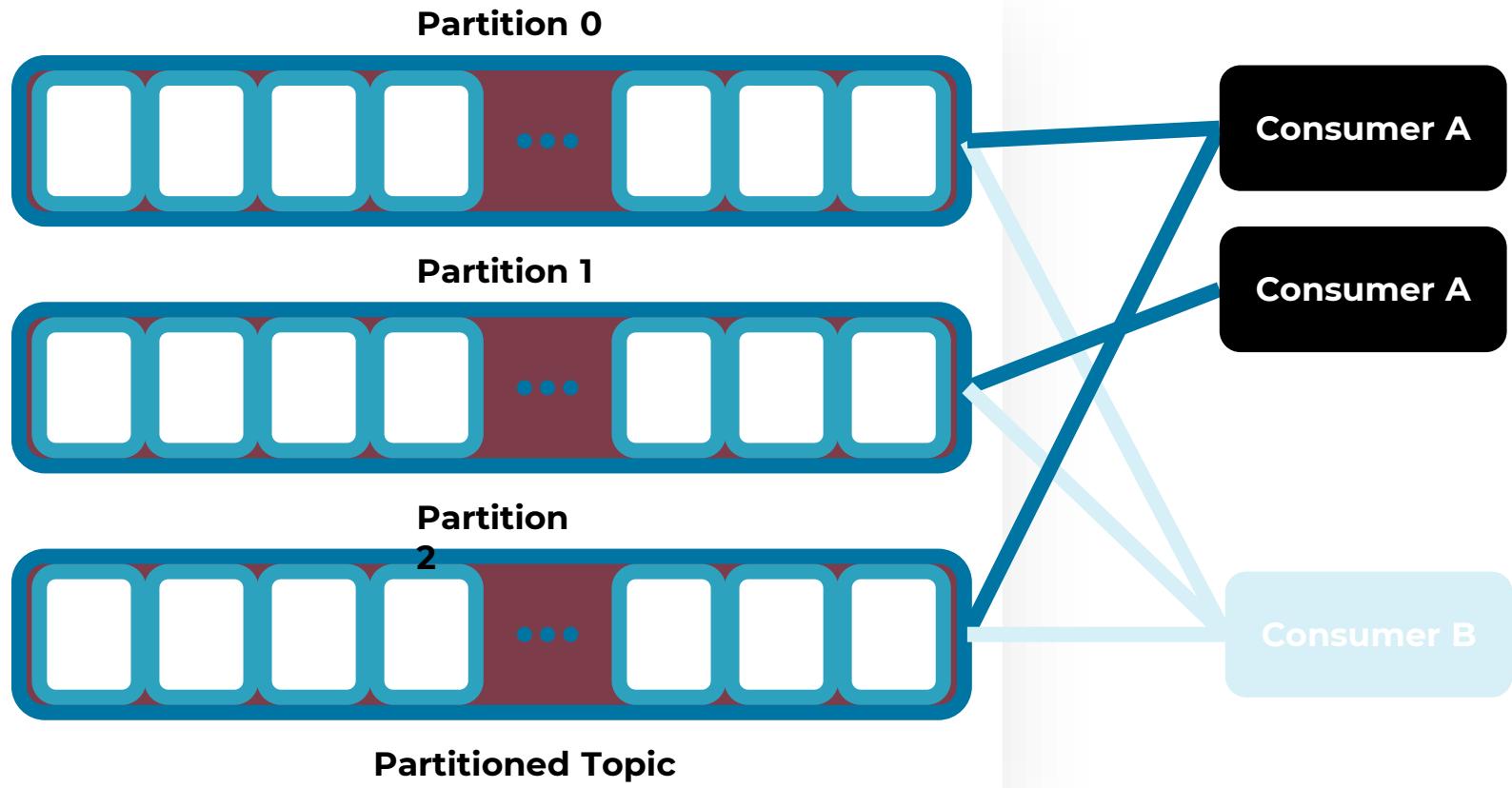
Consumers

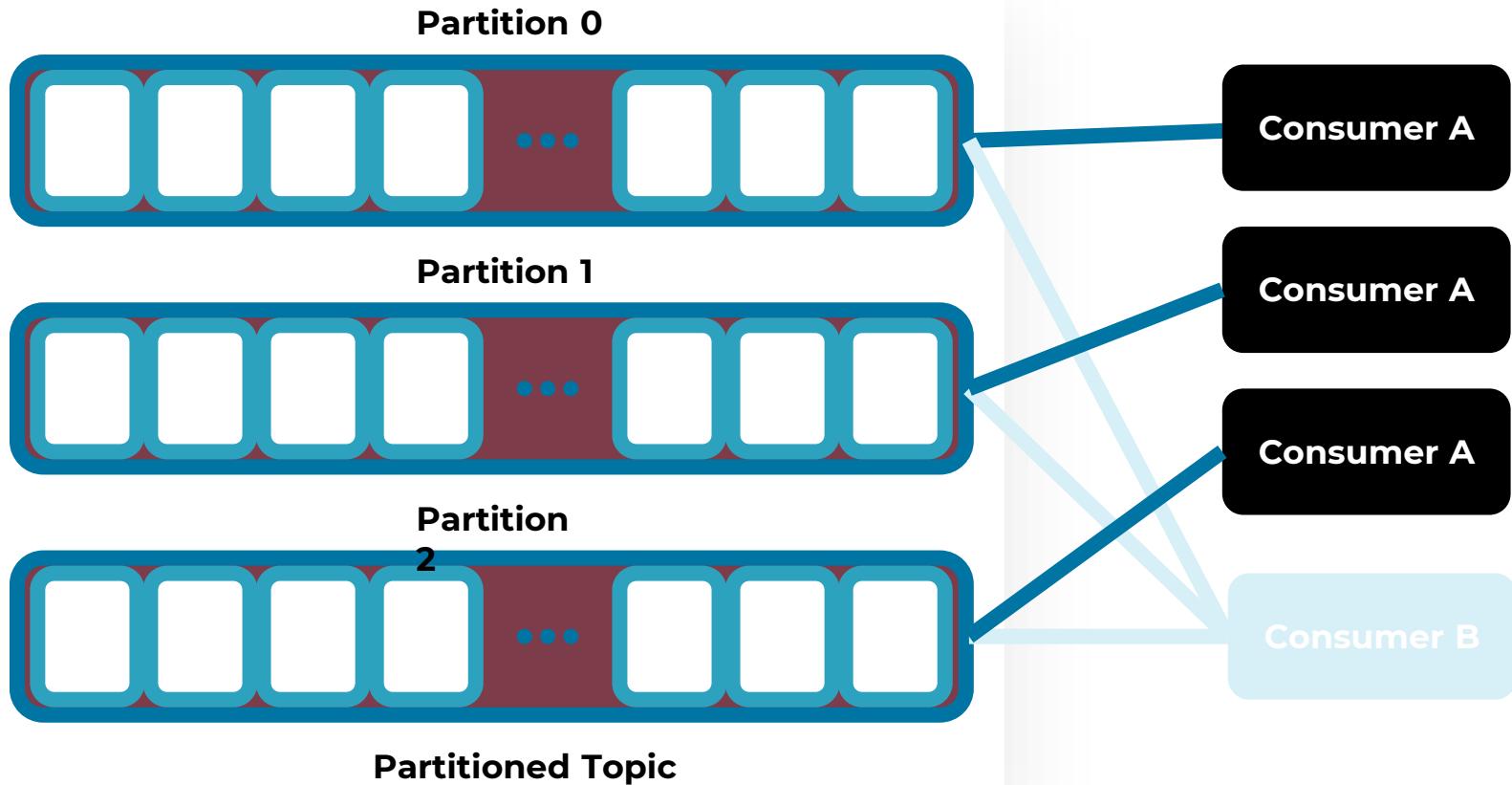
Consumers

- Client application
- Reads messages from topics
- Connection pooling
- Network protocol
- Horizontally and elastically scalable
- Maintains ordering within partitions at scale

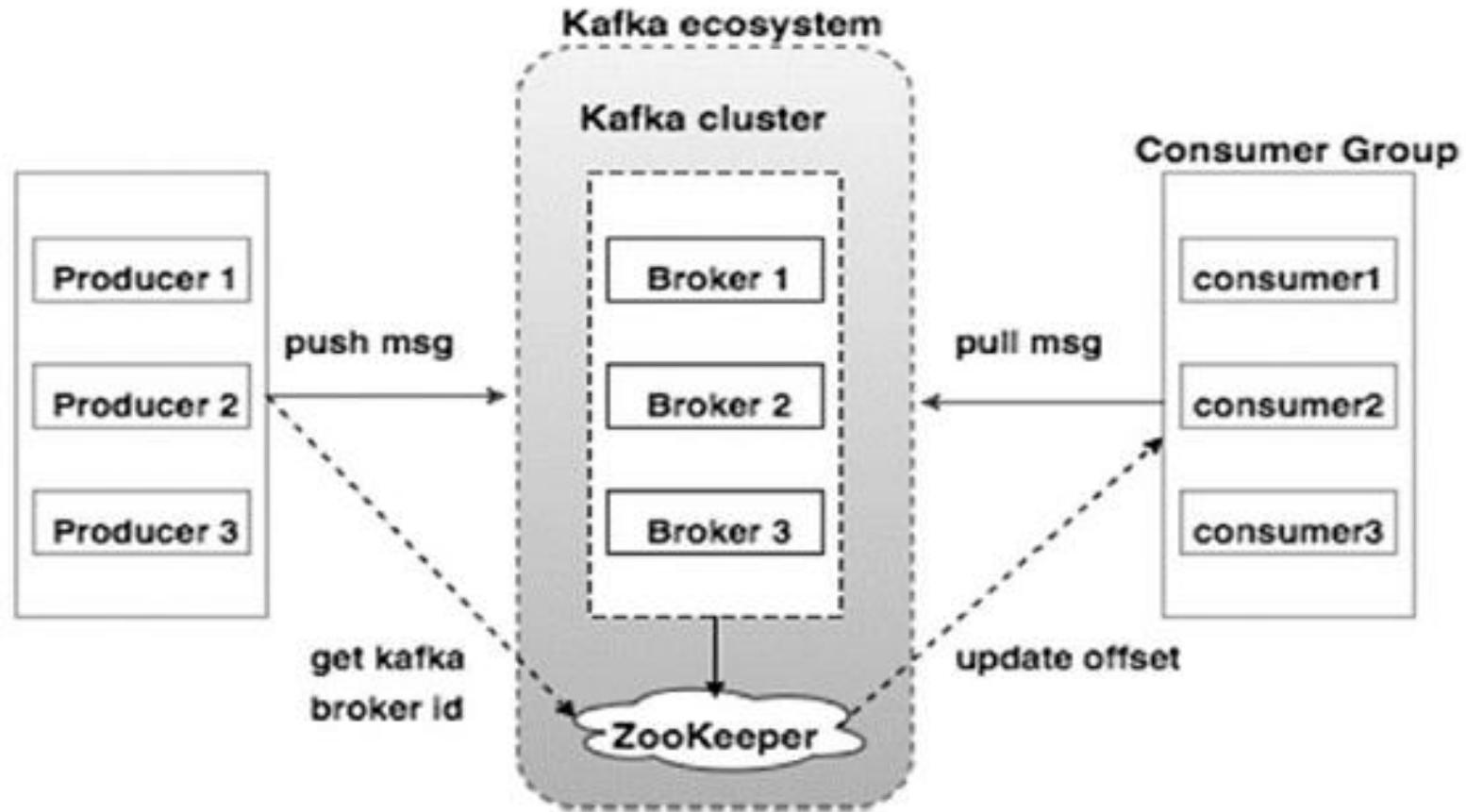








Apache Kafka - Cluster Architecture



Apache Kafka - Cluster Architecture

Broker

- ▶ Kafka cluster typically consists of multiple brokers to maintain load balance.
- ▶ Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state.
- ▶ One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact.
- ▶ Kafka broker leader election can be done by ZooKeeper.

Apache Kafka - Cluster Architecture

ZooKeeper

- ▶ ZooKeeper is used for managing and coordinating Kafka broker.
- ▶ ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.
- ▶ As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.

Apache Kafka - Cluster Architecture

Producers

- ▶ Producers push data to brokers.
- ▶ When the new broker is started, all the producers search it and automatically sends a message to that new broker.
- ▶ Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

Apache Kafka - Cluster Architecture

Consumers

- ▶ Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset.
- ▶ If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages.
- ▶ The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume.

Apache Kafka - Cluster Architecture

Consumers

- ▶ The consumers can rewind or skip to any point in a partition simply by supplying an offset value.
- ▶ Consumer offset value is notified by ZooKeeper.

Role of ZooKeeper

- ▶ A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service.
- ▶ Zookeeper serves as the coordination interface between the Kafka brokers and consumers.
- ▶ The Kafka servers share information via a Zookeeper cluster.
- ▶ Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

Role of ZooKeeper

- ▶ Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster.
- ▶ Kafka will restore the state, once the Zookeeper restarts.
- ▶ This gives zero downtime for Kafka.
- ▶ The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.



CONFLUENT

Developer

Ecosystem



Kafka Connect

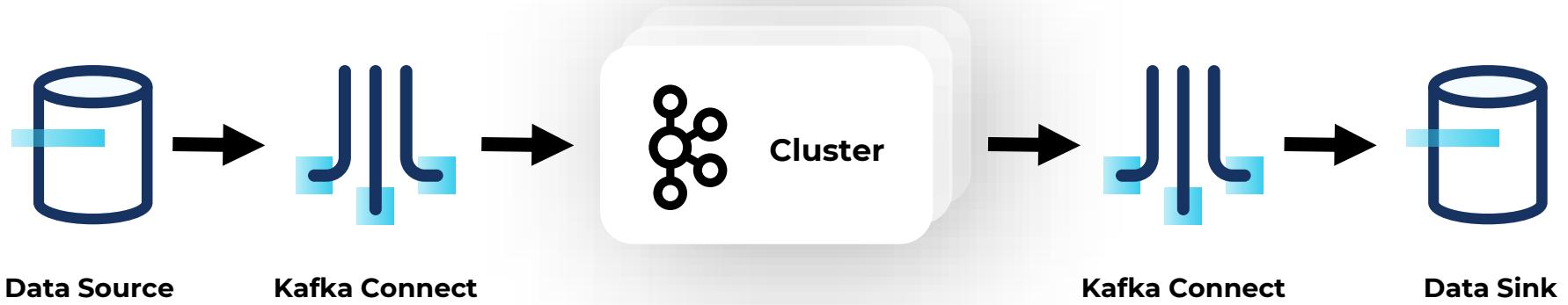
Kafka Connect



Data integration system and ecosystem

Because some other systems are not Kafka

External client process; does not run on brokers



Kafka Connect

- ↓ Data integration system and ecosystem
- ↓ Because some other systems are not Kafka
- ↓ External client process; does not run on brokers
- ↓ Horizontally scalable
- ↓ Fault tolerant

Kafka Connect

- ↓ Data integration system and ecosystem
- ↓ Because some other systems are not Kafka
- ↓ External client process; does not run on brokers
- ↓ Horizontally scalable
- ↓ Fault tolerant
- ↓ Declarative

Connectors



Pluggable software component

Connectors



Pluggable software component

Interfaces to external system and to Kafka

Connectors

- Pluggable software component
- Interfaces to external system and to Kafka
- Also exist as runtime entities

Connectors

- ↓ Pluggable software component
- ↓ Interfaces to external system and to Kafka
- ↓ Also exist as runtime entities
- ↓ Source connectors act as producers

Connectors

- ↓ Pluggable software component
- ↓ Interfaces to external system and to Kafka
- ↓ Also exist as runtime entities
- ↓ Source connectors act as producers
- ↓ Sink connectors act as consumers



Confluent Schema Registry

New consumers will emerge

Schemas evolve with the business

Schema Registry



Server process external to Kafka brokers

Schema Registry



Server process external to Kafka brokers

Maintains a database of schemas

Schema Registry

- ↓ Server process external to Kafka brokers
- ↓ Maintains a database of schemas
- ↓ HA deployment option available

Schema Registry

- Server process external to Kafka brokers
- Maintains a database of schemas
- HA deployment option available
- Consumer/Producer API component

Schema Registry



Defines schema compatibility rules per topic

Schema Registry



Defines schema compatibility rules per topic

Producer API prevents incompatible messages from being produced

Schema Registry



Defines schema compatibility rules per topic

Producer API prevents incompatible messages from being produced



Consumer API prevents incompatible messages from being consumer

Supported Formats

- ↓ JSON Schema
- ↓ Avro
- ↓ Protocol Buffers



Kafka Streams

Kafka Streams



Functional Java API

Kafka Streams



Functional Java API

Filtering, grouping, aggregating, joining, and more

Kafka Streams



Functional Java API

Filtering, grouping, aggregating, joining, and more

Scalable, fault-tolerant state management

Kafka Streams

- ↓ Functional Java API
- ↓ Filtering, grouping, aggregating, joining, and more
- ↓ Scalable, fault-tolerant state management
- ↓ Scalable computation based on consumer groups

Kafka Streams



Integrates within your services as a library

Kafka Streams



- Integrates within your services as a library
- Runs in the context of your application

Kafka Streams

- ↓ Integrates within your services as a library
- ↓ Runs in the context of your application
- ↓ Does not require special infrastructure



CONFLUENT

Developer

ksqldb

ksqlDB



A database optimized for stream processing

ksqlDB



- A database optimized for stream processing
- Runs on its own scalable, fault-tolerant cluster adjacent to the Kafka cluster

ksqlDB

- ↓ A database optimized for stream processing
- ↓ Runs on its own scalable, fault-tolerant cluster adjacent to the Kafka cluster
- ↓ Stream processing programs written in SQL

ksqlDB

- ↓ Command line interface
- ↓ REST API for application integration

ksqlDB

- ↓ Command line interface
- ↓ REST API for application integration
- ↓ Java library

ksqlDB

- ↓ Command line interface
- ↓ REST API for application integration
- ↓ Java library
- ↓ Kafka Connect integration

Confluent Platform vs Kafka – Key Differences

- ▶ Each release of Confluent Platform includes the latest release of Kafka and additional tools and services that make it easier to build and manage an Event Streaming Platform.
- ▶ Confluent Platform delivers both community and commercially licensed features that complement and enhance your Kafka deployment.

Overview of Confluent Platform's Enterprise Features

Confluent Control Center

- ▶ Confluent Control Center is a GUI-based system for managing and monitoring Kafka.
- ▶ It allows you to easily manage Kafka Connect, to create, edit, and manage connections to other systems.
- ▶ It also allows you to monitor data streams from producer to consumer, assuring that every message is delivered, and measuring how long it takes to deliver messages.

Overview of Confluent Platform's Enterprise Features

Confluent Control Center

- ▶ Using Control Center, you can build a production data pipeline based on Kafka without writing a line of code.
- ▶ Control Center also has the capability to define alerts on the latency and completeness statistics of data streams, which can be delivered by email or queried from a centralized alerting system.

Overview of Confluent Platform's Enterprise Features

Confluent for Kubernetes

- ▶ Confluent for Kubernetes is a *Kubernetes operator*.
- ▶ Kubernetes operators extend the orchestration capabilities of Kubernetes by providing the unique features and requirements for a specific platform application.
- ▶ For Confluent Platform, this includes greatly simplifying the deployment process of Kafka on Kubernetes and automating typical infrastructure lifecycle tasks.

Overview of Confluent Platform's Enterprise Features

Confluent Connectors to Kafka

- ▶ Connectors leverage the Kafka Connect API to connect Kafka to other systems such as databases, key-value stores, search indexes, and file systems.
- ▶ Confluent Hub has downloadable connectors for the most popular data sources and sinks.
- ▶ These include fully tested and supported versions of these connectors with Confluent Platform.
- ▶ Confluent provides both commercial and Community licensed connectors.

Overview of Confluent Platform's Enterprise Features

Self-Balancing Clusters

- ▶ Self-Balancing Clusters provides automated load balancing, failure detection and self-healing.
- ▶ It provides support for adding or decommissioning brokers as needed, with no manual tuning.
- ▶ Self-Balancing is the next iteration of Auto Data Balancer in that Self-Balancing auto-monitors clusters for imbalances, and automatically triggers rebalances based on your configurations.

Overview of Confluent Platform's Enterprise Features

Self-Balancing Clusters

- ▶ You can choose to auto-balance *Only when brokers are added or Anytime.*
- ▶ Partition reassignment plans and execution are taken care of for you.

Overview of Confluent Platform's Enterprise Features

Confluent Cluster Linking

- ▶ Cluster Linking directly connects clusters together and mirrors topics from one cluster to another over a link bridge.
- ▶ Cluster Linking simplifies setup of multi-datacenter, multi-cluster, and hybrid cloud deployments.

Overview of Confluent Platform's Enterprise Features

Confluent Auto Data Balancer

- ▶ As clusters grow, topics and partitions grow at different rates, brokers are added and removed and over time this leads to unbalanced workload across datacenter resources.
- ▶ Some brokers are not doing much at all, while others are heavily taxed with large or many partitions, slowing down message delivery.
- ▶ When executed, Confluent Auto Data Balancer monitors your cluster for number of brokers, size of partitions, number of partitions and number of leaders within the cluster.

Overview of Confluent Platform's Enterprise Features

Confluent Auto Data Balancer

- ▶ It allows you to shift data to create an even workload across your cluster, while throttling rebalance traffic to minimize impact on production workloads while rebalancing.

Overview of Confluent Platform's Enterprise Features

Confluent Replicator

- ▶ Replicator makes it easier than ever to maintain multiple Kafka clusters in multiple data centers.
- ▶ Managing replication of data and topic configuration between data centers enables use-cases such as:
 - Active-active geo-localized deployments: allows users to access a nearby data center to optimize their architecture for low latency and high performance
 - Centralized analytics: Aggregate data from multiple Kafka clusters into one location for organization-wide analytics

Overview of Confluent Platform's Enterprise Features

Confluent Replicator

- ▶ Managing replication of data and topic configuration between data centers enables use-cases such as:
 - Cloud migration: Use Kafka to synchronize data between on-prem applications and cloud deployments
- ▶ You can use Replicator to configure and manage replication for all these scenarios from either Confluent Control Center or command-line tools.

Overview of Confluent Platform's Enterprise Features

Tiered Storage

- ▶ Tiered Storage provides options for storing large volumes of Kafka data using your favorite cloud provider, thereby reducing operational burden and cost.
- ▶ With Tiered Storage, you can keep data on cost-effective object storage, and scale brokers only when you need more compute resources.

Overview of Confluent Platform's Enterprise Features

Confluent JMS Client

- ▶ Confluent Platform includes a JMS-compatible client for Kafka.
- ▶ This Kafka client implements the JMS 1.1 standard API, using Kafka brokers as the backend.
- ▶ This is useful if you have legacy applications using JMS, and you would like to replace the existing JMS message broker with Kafka.
- ▶ By replacing the legacy JMS message broker with Kafka, existing applications can integrate with your modern streaming platform without a major rewrite of the application.

Overview of Confluent Platform's Enterprise Features

Confluent MQTT Proxy

- ▶ Provides a way to publish data directly Kafka from MQTT devices and gateways without the need for a MQTT Broker in the middle.

Overview of Confluent Platform's Enterprise Features

Confluent Security Plugins

- ▶ Confluent Security Plugins are used to add security capabilities to various Confluent Platform tools and products.
- ▶ Currently, there is a plugin available for Confluent REST Proxy which helps in authenticating the incoming requests and propagating the authenticated principal to requests to Kafka.
- ▶ This enables Confluent REST Proxy clients to utilize the multi-tenant security features of the Kafka broker.

Community Features

ksqlDB

- ▶ ksqlDB is the streaming SQL engine for Kafka.
- ▶ It provides an easy-to-use yet powerful interactive SQL interface for stream processing on Kafka, without the need to write code in a programming language such as Java or Python.
- ▶ ksqlDB is scalable, elastic, fault-tolerant, and real-time.
- ▶ It supports a wide range of streaming operations, including data filtering, transformations, aggregations, joins, windowing, and sessionization.

Community Features

ksqlDB

- ▶ ksqlDB supports these use cases:

Streaming ETL

- Kafka is a popular choice for powering data pipelines.
- ksqlDB makes it simple to transform data within the pipeline, readying messages to cleanly land in another system.

Materialized cache / views

- A materialized view is a query result that is precomputed (before a user or app actually runs the query) and stored for faster read access.
- ksqlDB supports the building of materialized views in Kafka as event streams for distributed materializations.
- Complexity is reduced by using Kafka for storage and ksqlDB for computation.

Community Features

ksqlDB

- ▶ ksqlDB supports these use cases:

Event-driven Microservices

- Provides support for modeling stateless, event-driven microservices in Kafka.
- Stateful stream processing is managed on a cluster of servers, while side-effects run inside your stateless microservice, which reads events from a Kafka topic and takes action as needed.

Community Features

Confluent Connectors to Kafka

- ▶ Connectors leverage the Kafka Connect API to connect Kafka to other systems such as databases, key-value stores, search indexes, and file systems.
- ▶ Confluent Hub has downloadable connectors for the most popular data sources and sinks.
- ▶ Confluent provides both commercial and Community licensed connectors.

Confluent Clients

C/C++ Client Library

- ▶ The library **librdkafka** is the C/C++ implementation of the Kafka protocol, containing both Producer and Consumer support.
- ▶ It was designed with message delivery, reliability and high performance in mind.
- ▶ Current benchmarking figures exceed 800,000 messages per second for the producer and 3 million messages per second for the consumer.

Confluent Clients

C/C++ Client Library

- ▶ This library includes support for many new features of Kafka 0.10, including message security.
- ▶ It also integrates easily with libserdes, our C/C++ library for Avro data serialization (supporting Schema Registry).

Confluent Clients

Python Client Library

- ▶ A high-performance client for Python.

Go Client Library

- ▶ Confluent Platform includes of a full-featured, high-performance client for Go.

.NET Client Library

- ▶ Confluent Platform bundles a full featured, high performance client for .NET.

Confluent Schema Registry

- ▶ One of the most difficult challenges with loosely coupled systems is ensuring compatibility of data and code as the system grows and evolves.
- ▶ With a messaging service like Kafka, services that interact with each other must agree on a common format, called a *schema*, for messages.
- ▶ In many systems, these formats are ad hoc, only implicitly defined by the code, and often are duplicated across each system that uses that message type.

Confluent Schema Registry

- ▶ As requirements change, it becomes necessary to evolve these formats.
- ▶ With only an ad-hoc definition, it is very difficult for developers to determine what the impact of their change might be.
- ▶ **Confluent Schema Registry** enables safe, zero downtime evolution of schemas by centralizing the schema management.
- ▶ It provides a RESTful interface for storing and retrieving Avro®, JSON Schema, and Protobuf schemas.

Confluent Schema Registry

- ▶ Schema Registry tracks all versions of schemas used for every topic in Kafka and only allows evolution of schemas according to user-defined compatibility settings.
- ▶ This gives developers confidence that they can safely modify schemas as necessary without worrying that doing so will break a different service they may not even be aware of.
- ▶ Support for schemas is a foundational component of a data governance solution.

Confluent Schema Registry

- ▶ Schema Registry also includes plugins for Kafka clients that handle schema storage and retrieval for Kafka messages that are sent in the Avro format.
- ▶ This integration is seamless – if you are already using Kafka with Avro data, using Schema Registry only requires including the serializers with your application and changing one setting.

Confluent REST Proxy

- ▶ The Confluent REST Proxy makes it easy to work with Kafka from any language by providing a RESTful HTTP service for interacting with Kafka clusters.
- ▶ The REST Proxy supports all the core functionality: sending messages to Kafka, reading messages, both individually and as part of a consumer group, and inspecting cluster metadata,
- ▶ such as the list of topics and their settings. You can get the full benefits of the high quality, officially maintained Java clients from any language.

Confluent REST Proxy

- ▶ The REST Proxy also integrates with Schema Registry.
- ▶ It can read and write Avro data, registering and looking up schemas in Schema Registry.
- ▶ Because it automatically translates JSON data to and from Avro, you can get all the benefits of centralized schema management from any language using only HTTP and JSON.

Docker images of Confluent Platform

- ▶ Docker images of Confluent Platform are available on Docker Hub.
- ▶ Some of these images contain proprietary components that require a Confluent commercial license.
- ▶ These are identified cp-enterprise-\$ {component_name}.

Confluent CLI and other Command Line Tools

- ▶ Confluent Platform ships a number of command line interface (CLI) tools, including the Confluent CLI.

Confluent Quick Start

Local install

- ▶ **Prerequisites:**
 - Internet connectivity.
 - Operating System currently supported by Confluent Platform.
 - A supported version of Java downloaded and installed.
 - Java 8 and Java 11 are supported in this version of Confluent Platform (Java 9 and 10 are not supported).

Confluent Quick Start

Step 1: Download and Start Confluent Platform

- ▶ Go to the [downloads](#) page.
- ▶ Scroll to the **Download Confluent Platform** section and provide the following:
 - Email: Your email address
 - Format: deb, rpm, tar, or zip
- ▶ Click **DOWNLOAD FREE**.
- ▶ Decompress the file. You should have the directories, such as bin and etc.
- ▶ Set the environment variable for the Confluent Platform directory.

Confluent Quick Start

Step 1: Download and Start Confluent Platform

- ▶ Set the environment variable for the Confluent Platform directory.
 - `export CONFLUENT_HOME=<path-to-confluent>`
- ▶ Add the Confluent Platform bin directory to your PATH.
 - `export PATH=$PATH:$CONFLUENT_HOME/bin`
- ▶ Install the Kafka Connect Datagen source connector using the Confluent Hub client.
 - This connector generates mock data for demonstration purposes and is not suitable for production.
 - Confluent Hub is an online library of pre-packaged and ready-to-install extensions or add-ons for Confluent Platform and Kafka.

Confluent Quick Start

Step 1: Download and Start Confluent Platform

- Confluent Hub is an online library of pre-packaged and ready-to-install extensions or add-ons for Confluent Platform and Kafka.

confluent-hub install --no-prompt confluentinc/kafka-connect-datagen:latest

- ▶ Start Confluent Platform using the Confluent CLI confluent local services start command.
- ▶ This command starts all of the Confluent Platform component, including Kafka, ZooKeeper, Schema Registry, HTTP REST Proxy for Kafka, Kafka Connect, ksqlDB, and Control Center.

confluent local services start

Confluent Quick Start

Step 1: Download and Start Confluent Platform

Your output should resemble:

Starting Zookeeper

Zookeeper is [UP]

Starting Kafka

Kafka is [UP]

Starting Schema Registry

Schema Registry is [UP]

Starting Kafka REST

Kafka REST is [UP]

Starting Connect

Connect is [UP]

Starting KSQL Server

KSQL Server is [UP]

Starting Control Center

Control Center is [UP]

Confluent Quick Start

Step 2: Create Kafka Topics

- ▶ In this step, you create Kafka topics using Confluent Control Center.
- ▶ Confluent Control Center provides the functionality for building and monitoring production data pipelines and event streaming applications.
- ▶ Navigate to the Control Center web interface at <http://localhost:9021>.
- ▶ If you installed Confluent Platform on a different host, replace localhost with the host name in the address.
- ▶ It may take a minute or two for Control Center to come online.

Confluent Quick Start

Step 2: Create Kafka Topics

- ▶ Click the **controlcenter.cluster** tile.

Home

1 Healthy clusters 0 Unhealthy clusters

Search cluster name or id

controlcenter.cluster

Running

Overview	
Brokers	1
Partitions	321
Topics	61
Production	17.53KB/s
Consumption	13.57KB/s
Connected services	
ksqlDB clusters	1
Connect clusters	1

Confluent Quick Start

Step 2: Create Kafka Topics

- In the navigation bar, click **Topics** to open the topics list, and then click **Add a topic**.

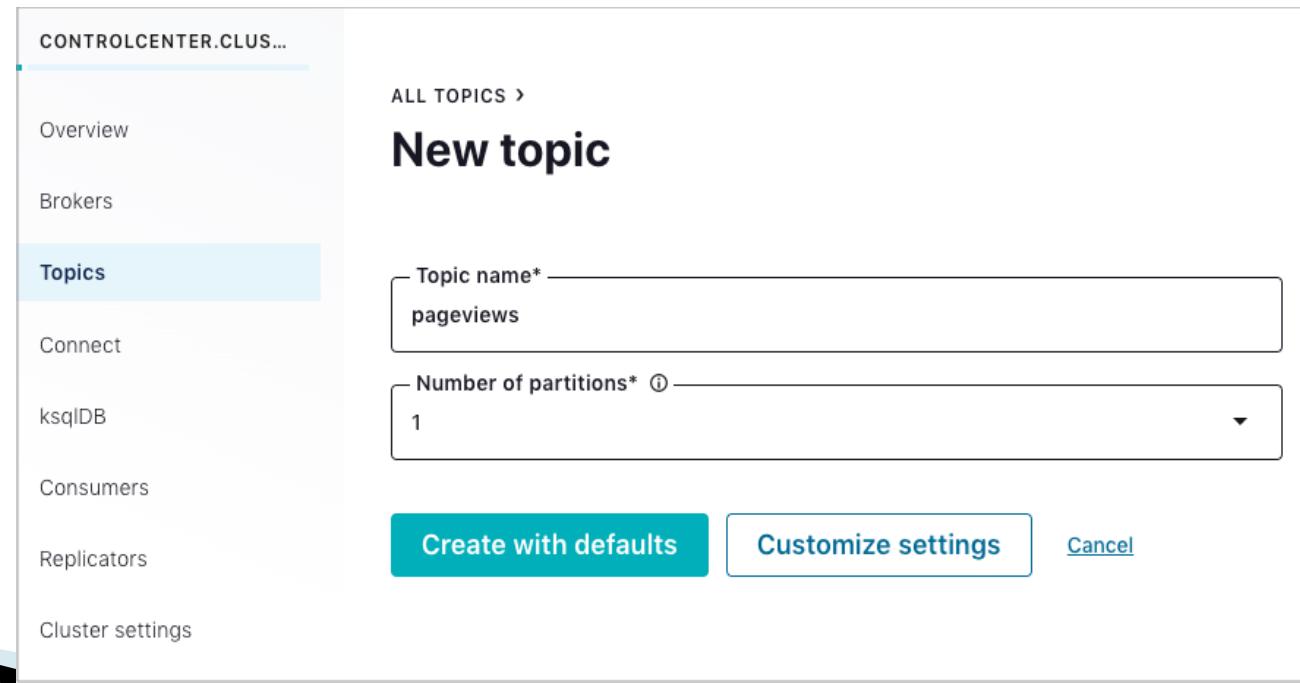
The screenshot shows the 'Topics' page in the Confluent Cloud UI. On the left, there's a navigation sidebar with links like 'Cluster overview', 'Brokers', 'Topics' (which is highlighted in blue), 'Connect', 'ksqlDB', 'Consumers', 'Replicators', and 'Cluster settings'. The main area has a title 'Topics' and a search bar. Below it is a table with columns: 'Topics' (Topic name), 'Availability' (Partitions, Under replicated, Followers, Out of sync), and 'Throughput' (Produced, Consumed). A single row shows a topic named 'default_ksql_processing_log' with 1 partition, 0 under replicated, 1 follower, 0 out of sync, 0 observers, and 0 out of sync. At the top right of the table is a button labeled '+ Add a topic' which is highlighted with a red box.

Topics	Availability						Throughput	
	Topic name	Partitions	Under replicated	Followers	Out of sync	Observers	Out of sync	Produced
default_ksql_processing_log	1	0	1	0	0	0	--	--

Confluent Quick Start

Step 2: Create Kafka Topics

- ▶ In the Topic name field, specify pageviews and click **Create with defaults**.
- ▶ Note that topic names are case-sensitive.



Confluent Quick Start

Step 2: Create Kafka Topics

- ▶ In the navigation bar, click **Topics** to open the topics list, and then click **Add a topic**.
- ▶ In the Topic name field, specify users and click **Create with defaults**.

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

- ▶ In this step, you use Kafka Connect to run a demo source connector called kafka-connect-datagen that creates sample data for the Kafka topics pageviews and users.
- ▶ The Kafka Connect Datagen connector was installed manually in Step 1: Download and Start Confluent Platform.

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

- ▶ Run the first instance of the Kafka Connect Datagen connector to produce Kafka data to the pageviews topic in AVRO format.
 - In the navigation bar, click **Connect**.
 - Click the connect-default cluster in the **Connect Clusters** list.
 - Click **Add connector**.
 - Select the DatagenConnector tile.
 - In the **Name** field, enter datagen-pageviews as the name of the connector.

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

- ▶ Run the first instance of the Kafka Connect Datagen connector to produce Kafka data to the pageviews topic in AVRO format.
 - Enter the following configuration values:
 - **Key converter**
class: org.apache.kafka.connect.storage.StringConverter.
 - **kafka.topic:** pageviews.
 - **max.interval:** 100.
 - **quickstart:** pageviews.
 - Click **Next**.
 - Review the connector configuration and click **Launch**.

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

Add Connector

1. Setup connection —— 2. Test and verify

```
{  
  "name": "datagen-pageviews",  
  "config": {  
    "name": "datagen-pageviews",  
    "connector.class": "io.confluent.kafka.connect.datagen.DataGenConnector",  
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",  
    "kafka.topic": "pageviews",  
    "max.interval": "100",  
    "quickstart": "pageviews"  
  }  
}
```

[Launch](#)

[Back](#)

[Download connector config file](#)

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

- ▶ Run the second instance of the Kafka Connect Datagen connector to produce Kafka data to the users topic in AVRO format.
 - Click **Add connector**.
 - Select the DatagenConnector tile.
 - In the **Name** field, enter datagen-users as the name of the connector.

Confluent Quick Start

Step 3: Install a Kafka Connector and Generate Sample Data

- Enter the following configuration values:
 - **Key converter**
`class: org.apache.kafka.connect.storage.StringConverter`
 - **kafka.topic:** users
 - **max.interval:** 1000
 - **quickstart:** users
- Click **Next**.
- ▶ Review the connector configuration and click **Launch**.

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Create Streams and Tables

In this step, you use ksqlDB to create a stream for the pageviews topic and a table for the users topic.

- ▶ In the navigation bar, click **ksqlDB**.
- ▶ Select the ksqlDB application.
- ▶ Copy the following code into the editor window and click **Run query** to create the pageviews stream. Stream names are not case-sensitive.

```
CREATE STREAM pageviews WITH (KAFKA_TOPIC='pageviews', VALUE_FORMAT='AVRO');
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Create Streams and Tables

- ▶ Copy the following code into the editor window and click **Run query** to create the users table. Table names are not case-sensitive.

```
CREATE TABLE users (id VARCHAR PRIMARY KEY) WITH (KAFKA_TOPIC='users', VALUE_FORMAT='AVRO');
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Write Queries

- ▶ In this step, you create ksqlDB queries against the stream and the table you created above.
 - In the **Editor** tab, click **Add query properties** to add a custom query property.
 - Set the auto.offset.reset parameter to Earliest.
 - The setting instructs ksqlDB queries to read all available topic data from the beginning. This configuration is used for each subsequent query.

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Write Queries

- ▶ In this step, you create ksqlDB queries against the stream and the table you created above.
 - Create the following queries.
 - Click **Stop** to stop the current running query.
 - Create a non-persistent query that returns data from a stream with the results limited to a maximum of three rows:
 - Enter the following query in the editor:
`SELECT pageid FROM pageviews EMIT CHANGES LIMIT 3;`

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Write Queries

- In this step, you create ksqlDB queries against the stream and the table you created above.
 - Click **Run query**. Your output should resemble:

The screenshot shows the ksqlDB interface. On the left, a sidebar displays metrics for a stream: Data structure (STREAM), Total messages (---), Messages/sec (---), Total message bytes (---), and Message fields (PAGEID). The main area shows a table with a single column labeled PAGEID, containing three rows: Page_71, Page_13, and Page_52. At the top right, there are icons for Run, Filter by keyword, and View (Card or Table).

- Click the Card view or Table view icon to change the output layout.

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Write Queries

- ▶ In this step, you create ksqlDB queries against the stream and the table you created above.
 - Create a persistent query (as a stream) that filters the PAGEVIEWS stream for female users. The results from this query are written to the Kafka PAGEVIEWS_FEMALE topic:
 - Enter the following query in the editor:

```
CREATE STREAM pageviews_female
    AS SELECT users.id AS userid, pageid, regionid
        FROM pageviews LEFT JOIN users ON pageviews.userid = users.id
        WHERE gender = 'FEMALE'
        EMIT CHANGES;
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Write Queries

- ▶ In this step, you create ksqlDB queries against the stream and the table you created above.
- ▶ Click **Run query**. Your output should resemble:

```
0  {
1    "@type": "currentStatus",
2    "statementText": "CREATE STREAM PAGEVIEWS_FEMALE WITH (KAFKA_TOPIC='PAGEVIEWS_",
3    "commandId": "stream`/PAGEVIEWS_FEMALE`/create",
4    "commandStatus": {
5      "status": "SUCCESS",
6      "message": "Created query with ID CSAS_PAGEVIEWS_FEMALE_0"
7    },
8    "commandSequenceNumber": 6,
9    "warnings": []
10 }
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Create a persistent query where REGIONID ends with 8 or 9. Results from this query are written to the Kafka topic named pageviews_enriched_r8_r9 as explicitly specified in the query:
- Enter the following query in the editor:

```
CREATE STREAM pageviews_female_like_89
  WITH (KAFKA_TOPIC='pageviews_enriched_r8_r9', VALUE_FORMAT='AVRO')
  AS SELECT * FROM pageviews_female
  WHERE regionid LIKE '%_8' OR regionid LIKE '%_9'
  EMIT CHANGES;
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Click **Run query**. Your output should resemble:

```
0  {
1    "@type": "currentStatus",
2    "statementText": "CREATE STREAM PAGEVIEWS_FEMALE_LIKE_89 WITH (KAFKA_TOPIC='pag",
3    "commandId": "stream/`PAGEVIEWS_FEMALE_LIKE_89`/create",
4    "commandStatus": {
5      "status": "SUCCESS",
6      "message": "Created query with ID CSAS_PAGEVIEWS_FEMALE_LIKE_89_7"
7    },
8    "commandSequenceNumber": 8,
9    "warnings": []
10 }
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Create a persistent query that counts the PAGEVIEWS for each REGION and GENDER combination in a tumbling window of 30 seconds when the count is greater than 1. Because the procedure is grouping and counting, the result is now a table, rather than a stream. Results from this query are written to a Kafka topic called PAGEVIEWS_REGIONS:
- Enter the following query in the editor:

```
CREATE TABLE pageviews_regions WITH (KEY_FORMAT='JSON')
    AS SELECT gender, regionid, COUNT(*) AS numusers
        FROM pageviews LEFT JOIN users ON pageviews.userid = users.id
        WINDOW TUMBLING (SIZE 30 SECOND)
        GROUP BY gender, regionid
        HAVING COUNT(*) > 1
        EMIT CHANGES;
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Click **Run query**. Your output should resemble:

```
0  {
1      "@type": "currentStatus",
2      "statementText": "CREATE TABLE PAGEVIEWS_REGIONS WITH (KAFKA_TOPIC='PAGEVIEWS_F",
3      "commandId": "table/`PAGEVIEWS_REGIONS`/create",
4      "commandStatus": {
5          "status": "SUCCESS",
6          "message": "Created query with ID CTAS_PAGEVIEWS_REGIONS_9"
7      },
8      "commandSequenceNumber": 10,
9      "warnings": []
10 }
```

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Click the **Persistent queries** tab. You should see the following persisted queries:
 - PAGEVIEWS_FEMALE
 - PAGEVIEWS_FEMALE_LIKE_89
 - PAGEVIEWS_REGIONS

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- Click the **Editor** tab. The **All available streams and tables** pane shows all of the streams and tables that you can access.

The screenshot shows a search bar at the top with the placeholder "Search". Below it is a list titled "All available streams and tables". The list contains the following items:

- St** KSQL_PROCESSING_LOG
- St** PAGEVIEWS
- St** PAGEVIEWS_FEMALE
- St** PAGEVIEWS_FEMALE_LIKE_89
- Tb** PAGEVIEWS_REGIONS
- Tb** USERS

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

- In the **All available streams and tables** section, click **KSQ_L_PROCESSING_LOG** to view the stream's schema, including nested data structures.

Confluent Quick Start

Step 4: Create and Write to a Stream and Table using ksqlDB

Run Queries

- ▶ In this step, you run the ksqlDB queries you save as streams and tables above in the previous section.
- ▶ In the **Streams** tab, select the `PAGEVIEWS_FEMALE` stream.
- ▶ Click **Query stream**.
- ▶ The editor opens, and streaming output of the query displays.
- ▶ Click **Stop** to stop the output generation.
- ▶ In the **Tables** tab, select `PAGEVIEWS_REGIONS` table.
- ▶ Click **Query table**.
- ▶ The editor opens, and streaming output of the query displays.
- ▶ Click **Stop** to stop the output generation.

Confluent Quick Start

Step 5: Monitor Consumer Lag

- ▶ In the navigation bar, click **Consumers** to view the consumers created by ksqlDB.
- ▶ Click the consumer group ID to view details for the _confluent-ksql-default_query_CSAS_PAGEVIEWS_FEMALE_5 consumer group.
- ▶ From the page, you can see the consumer lag and consumption values for your streaming query.

Confluent Quick Start

Step 6: Stop Confluent Platform

When you are done working with the local install, you can stop Confluent Platform.

- ▶ Stop Confluent Platform using the Confluent CLI confluent local services connect stop command.
`confluent local services stop`
- ▶ Destroy the data in the Confluent Platform instance with the confluent local destroy command.
`confluent local destroy`