

# Application Delivery Fundamentals :

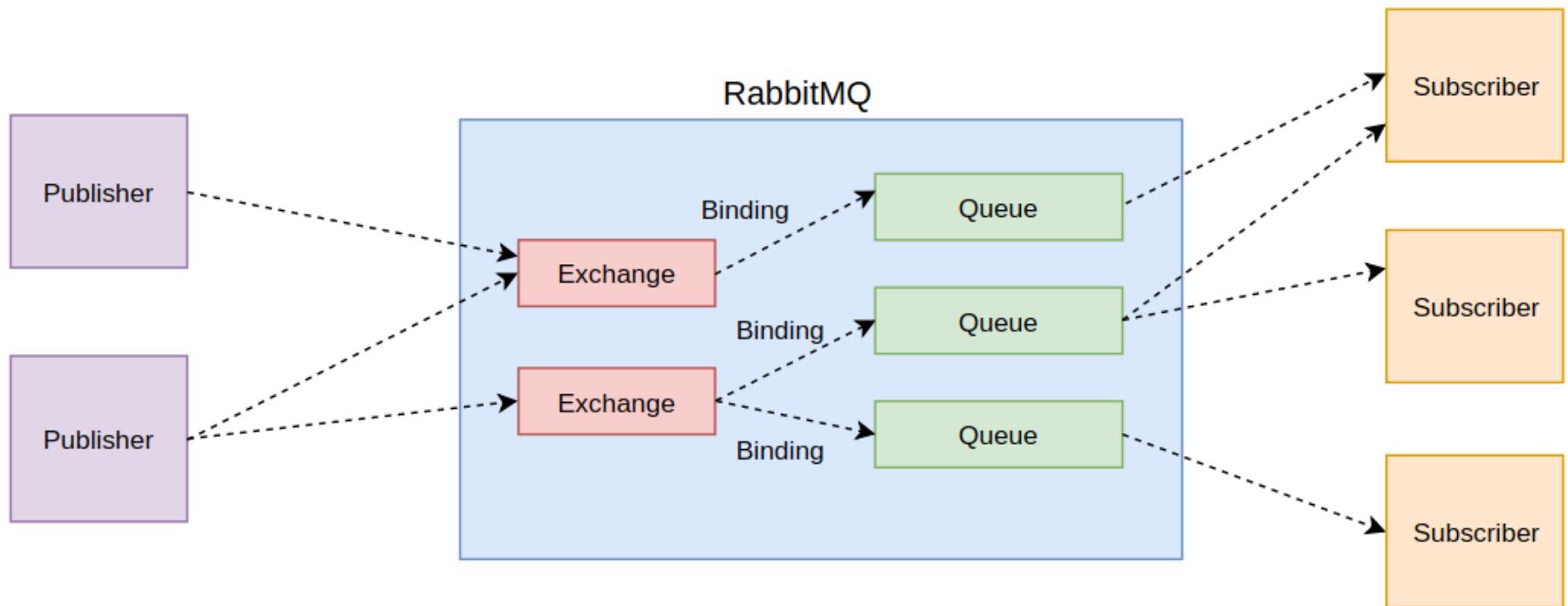
kafka



High performance. Delivered.

consulting | technology | outsourcing

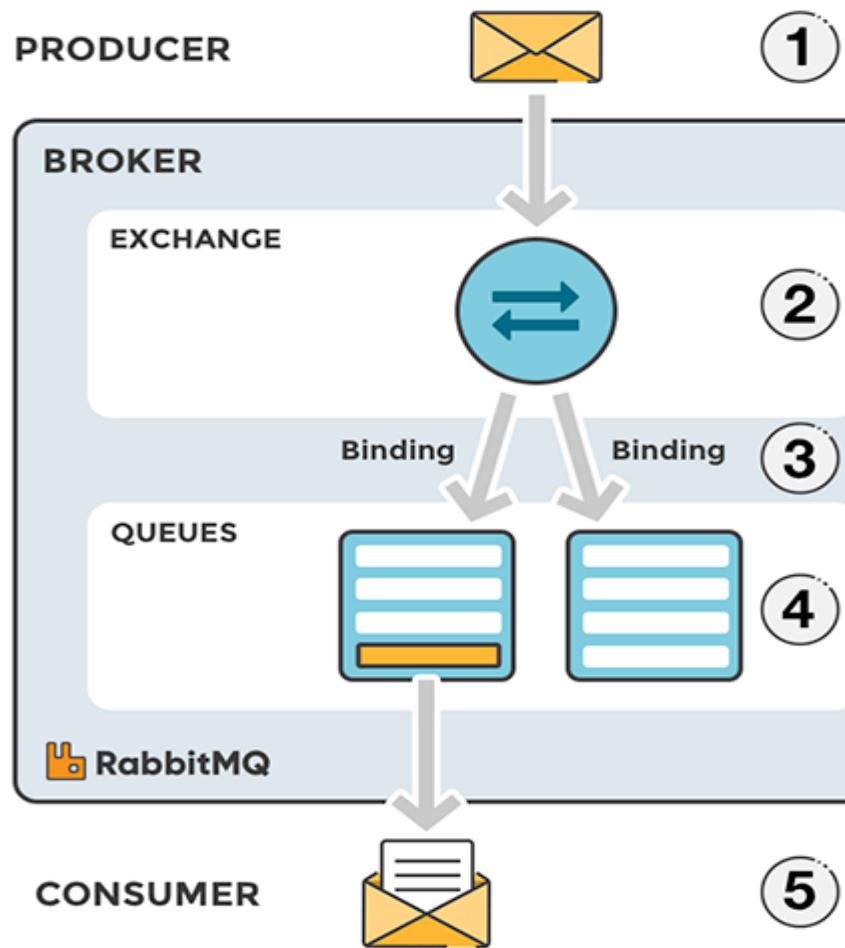
# Rabbitmq Architectural Pattern



## Rabbitmq Standard Exchange

- The producer publishes a message to the exchange.
- The exchange receives the message and is now responsible for the routing of the message.
- Binding must be set up between the queue and the exchange. In this case, we have bindings to two different queues from the exchange. The exchange routes the message into the queues.
- The messages stay in the queue until they are handled by a consumer.
- The consumer handles the message.

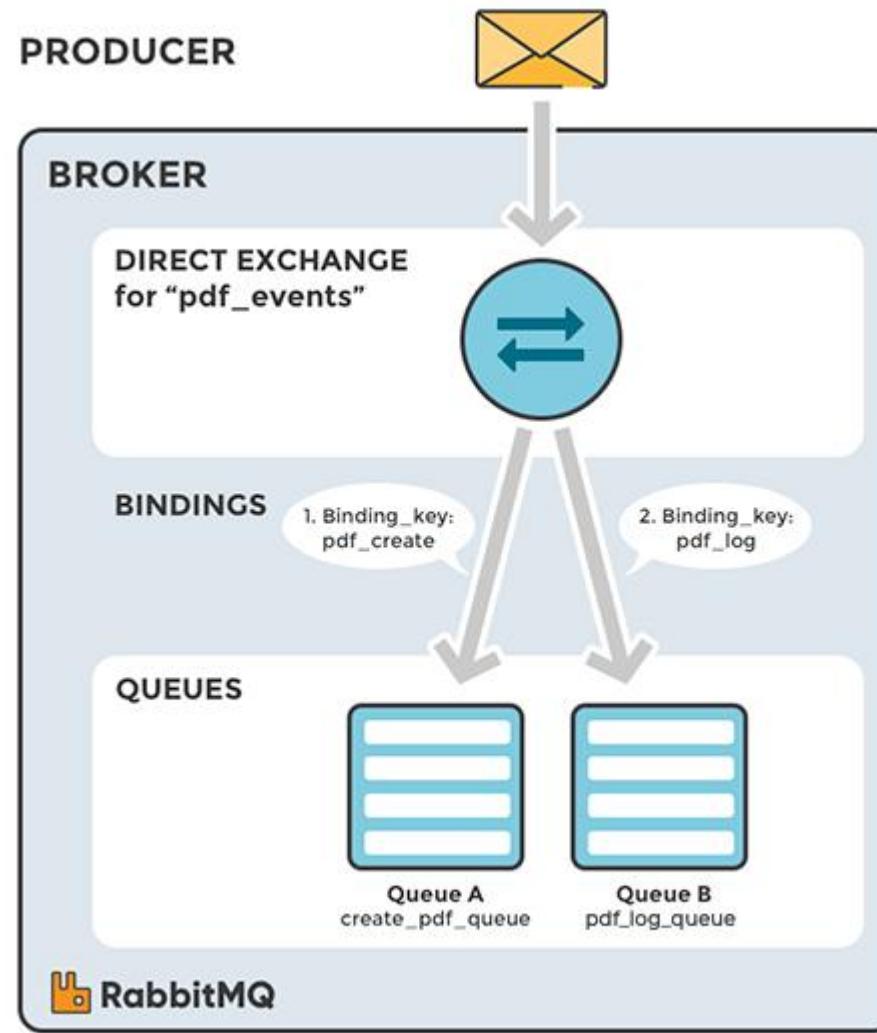
# Rabbitmq Standard Exchange



## Exchange types - Direct Exchange

- A direct exchange delivers messages to queues based on a message routing key.
- The routing key is a message attribute added to the message header by the producer.
- Think of the routing key as an "address" that the exchange is using to decide how to route the message.
- A message goes to the queue(s) with the binding key that exactly matches the routing key of the message.

# Exchange types - Direct Exchange



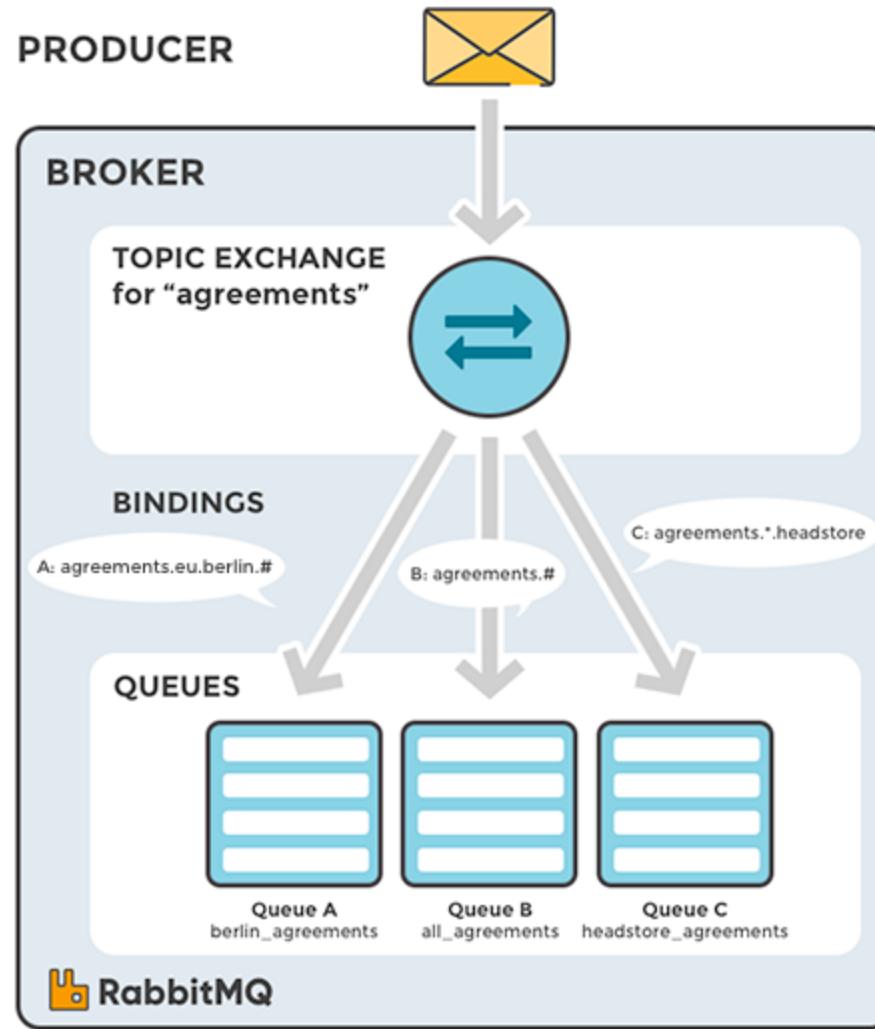
## Exchange types - Default exchange

- The default exchange is a pre-declared direct exchange with no name, usually referred by an empty string.
- When you use default exchange, your message is delivered to the queue with a name equal to the routing key of the message.
- Every queue is automatically bound to the default exchange with a routing key which is the same as the queue name.

## Exchange types - Topic Exchange

- Topic exchanges route messages to queues based on wildcard matches between the routing key and the routing pattern, which is specified by the queue binding.
- Messages are routed to one or many queues based on a matching between a message routing key and this pattern.

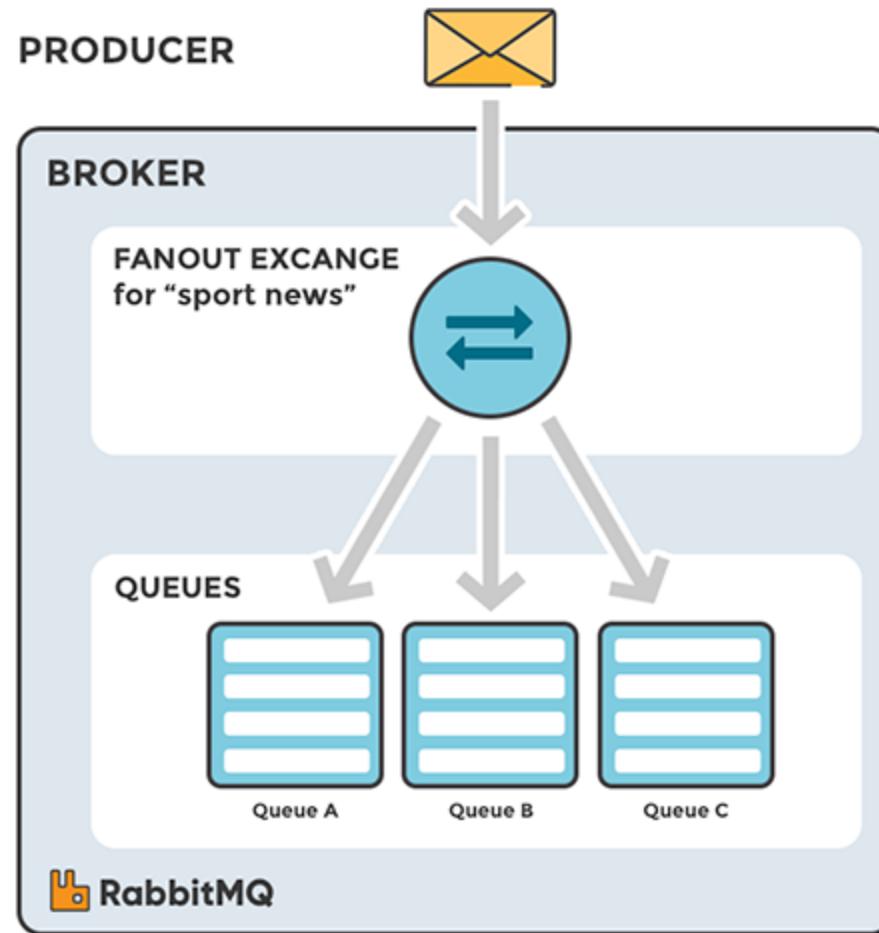
# Exchange types - Topic Exchange



## Exchange Types - Fanout Exchange

- A fanout exchange copies and routes a received message to all queues that are bound to it regardless of routing keys or pattern matching as with direct and topic exchanges. The keys provided will simply be ignored.
- Fanout exchanges can be useful when the same message needs to be sent to one or more queues with consumers who may process the same message in different ways.

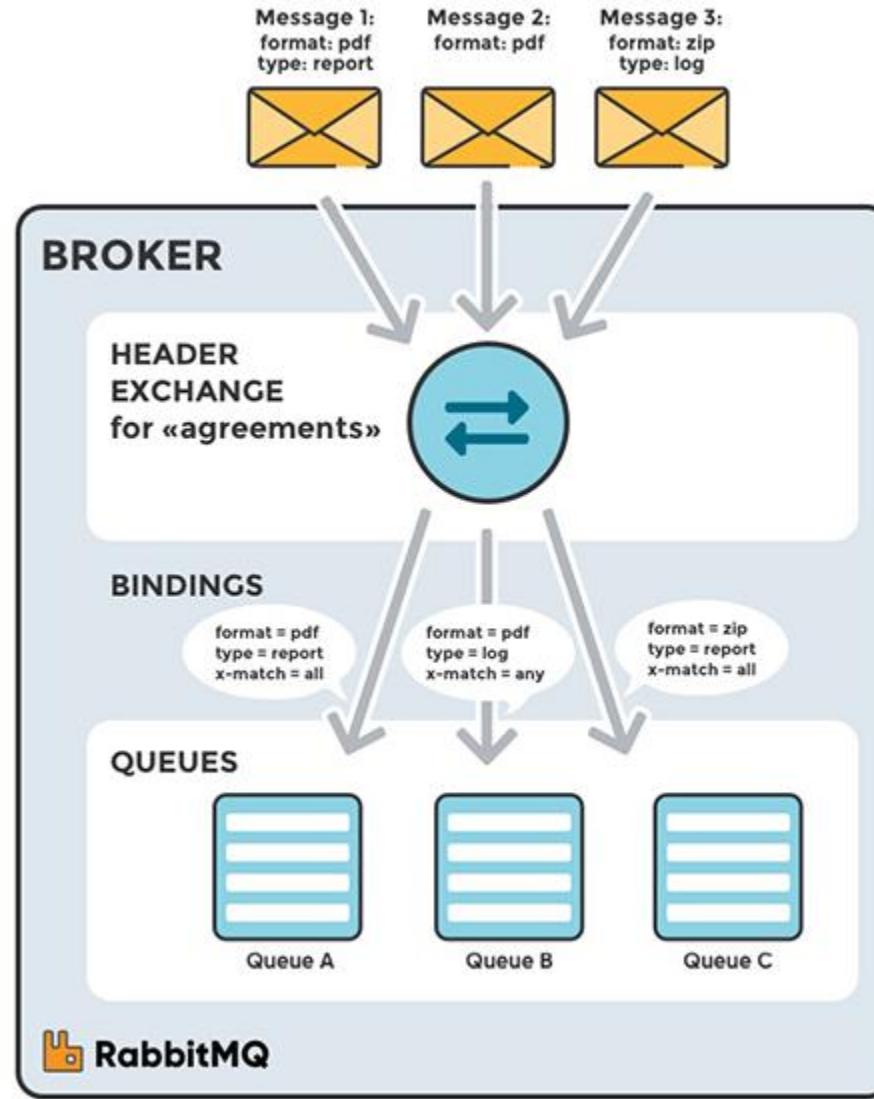
# Exchange Types - Fanout Exchange



## Exchange Types Headers Exchange

- A headers exchange routes messages based on arguments containing headers and optional values.
- Headers exchanges are very similar to topic exchanges, but route messages based on header values instead of routing keys.
- A message matches if the value of the header equals the value specified upon binding.

# Rabbitmq – Headers Exchange



## Exchange Type - DLQ

---

- If no matching queue can be found for the message, the message is silently dropped. RabbitMQ provides an AMQP extension known as the "Dead Letter Exchange", which provides the functionality to capture messages that are not deliverable.

# RabbitMQ

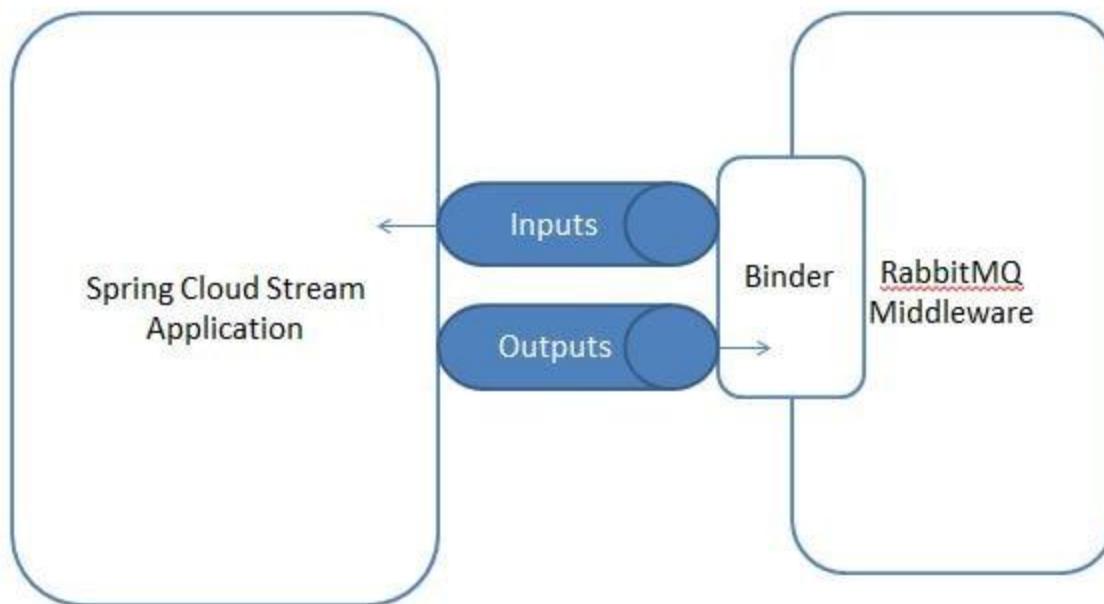
---

- The super simplified overview:
  - Publishers send messages to exchanges
  - Exchanges route messages to queues and other exchanges
  - RabbitMQ sends acknowledgements to publishers on message receipt
  - Consumers maintain persistent TCP connections with RabbitMQ and declare which queue(s) they consume
  - RabbitMQ pushes messages to consumers
  - Consumers send acknowledgements of success/failure
  - Messages are removed from queues once consumed successfully

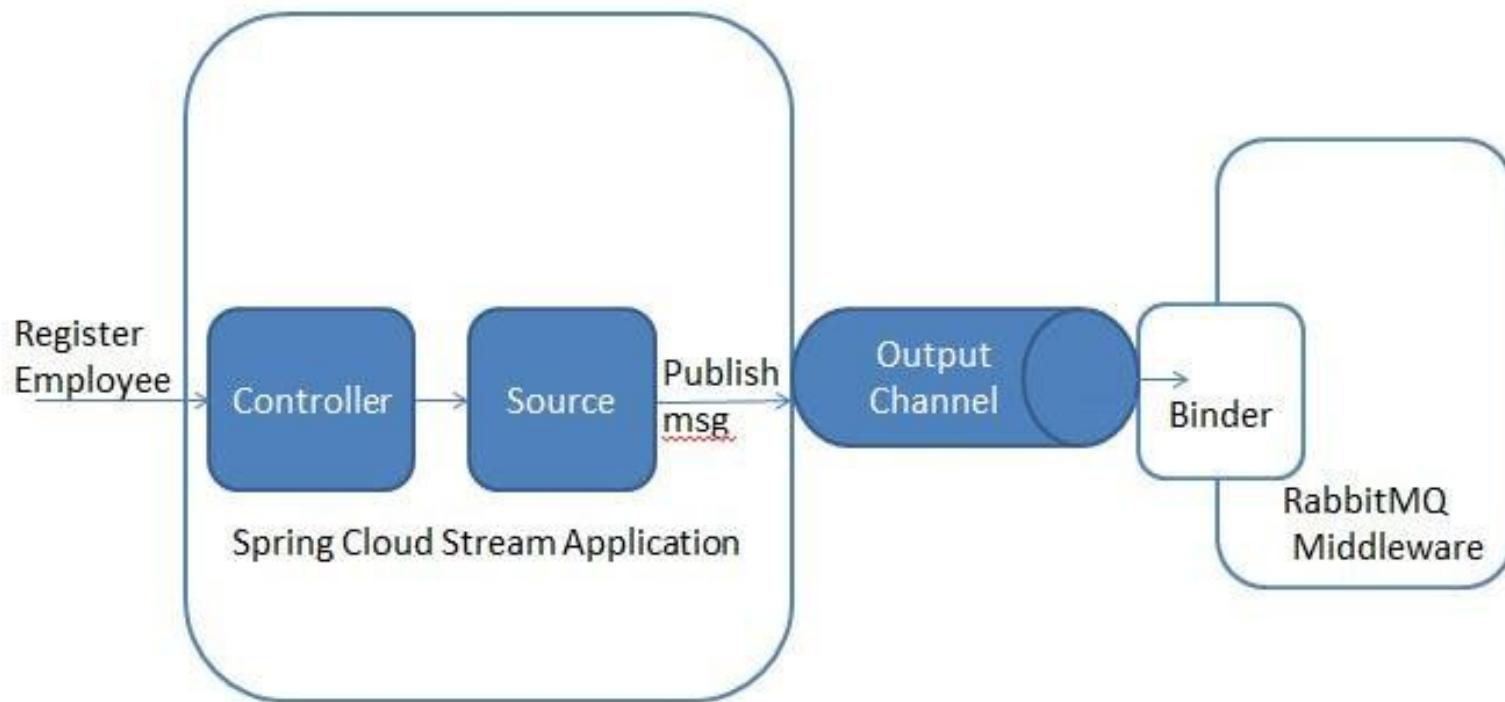
# Core Components of Cloud Stream

- The core building blocks of Spring Cloud Stream are:
- Destination Binders: Components responsible to provide integration with the external messaging systems.
- Destination Bindings: Bridge between the external messaging systems and application provided Producers and Consumers of messages (created by the Destination Binders).
- Message: The canonical data structure used by producers and consumers to communicate with Destination Binders (and thus other applications via external messaging systems).

# Core Components of Cloud Stream



# Core Components of Cloud Stream



# Asynchronous communication in Microservices



- Kafka
- RabbitMQ
- Google Pub/Sub
- Amazon Services
- ActiveMQ
- Azure Services

# What is ZooKeeper

- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- All of these kinds of services are used in some form or another by distributed applications.
- Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.
- Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage.
- Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

# What is ZooKeeper?



- Apache ZooKeeper plays the very important role in system architecture as it works in the shadow of more exposed Big Data tools, as Apache Spark or Apache Kafka.
- Originally, the ZooKeeper framework was built at “Yahoo!”. Because it helps to access their applications in an easy manner.

## Why Apache ZooKeeper ?



1  
Naming Service

2  
Configuration Management

3  
Synchronization Service

4  
Group Services

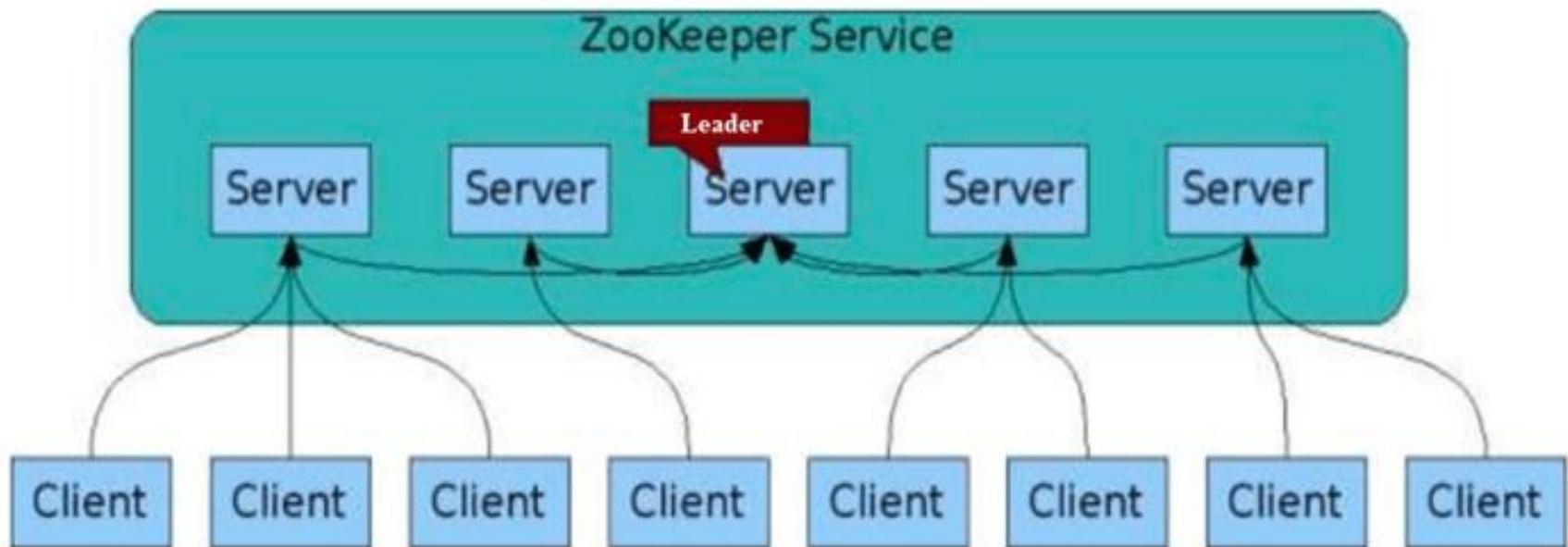
# Why Zookeeper

- It allows for mutual exclusion and cooperation between server processes.
- It ensures that your application runs consistently.
- The transaction process is never completed partially. It is either given the status of Success or failure. The distributed state can be held up, but it's never wrong.
- Irrespective of the server that it connects to, a client will be able to see the same view of the service
- Helps you to encode the data as per the specific set of rules

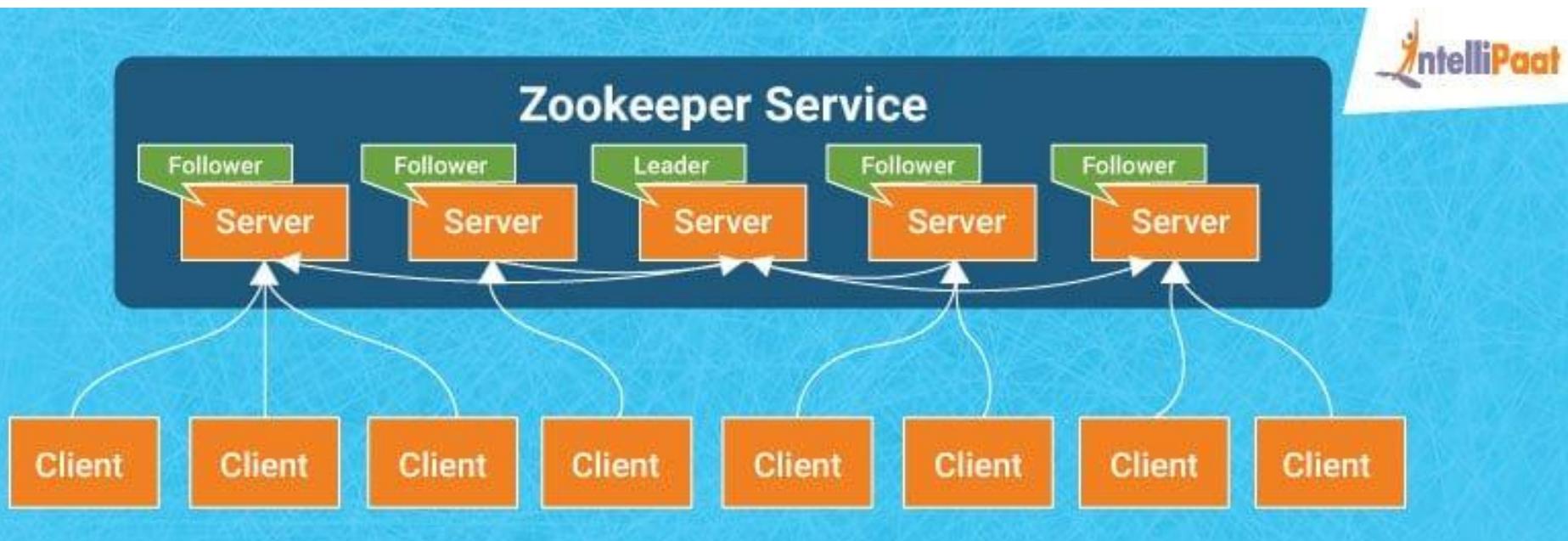
# Why Zookeeper

- It helps to maintain a standard hierarchical namespace similar to files and directories.
- Computers, which run as a single system which can be locally or geographically connected.
- It allows to Join/leave node in a cluster and node status at the real time
- You can increase performance by deploying more machines.
- It allows you to elect a node as a leader for better coordination.
- ZooKeeper works fast with workloads where reads to the data are more common than writes.

# ZooKeeper Architecture: How it works?



# ZooKeeper Architecture: How it works?



# ZooKeeper Architecture

- Server: The server sends an acknowledge when any client connects. In the case when there is no response from the connected server, the client automatically redirects the message to another server.
- Client: Client is one of the nodes in the distributed application cluster. It helps you to accesses information from the server. Every client sends a message to the server at regular intervals that helps the server to know that the client is alive.

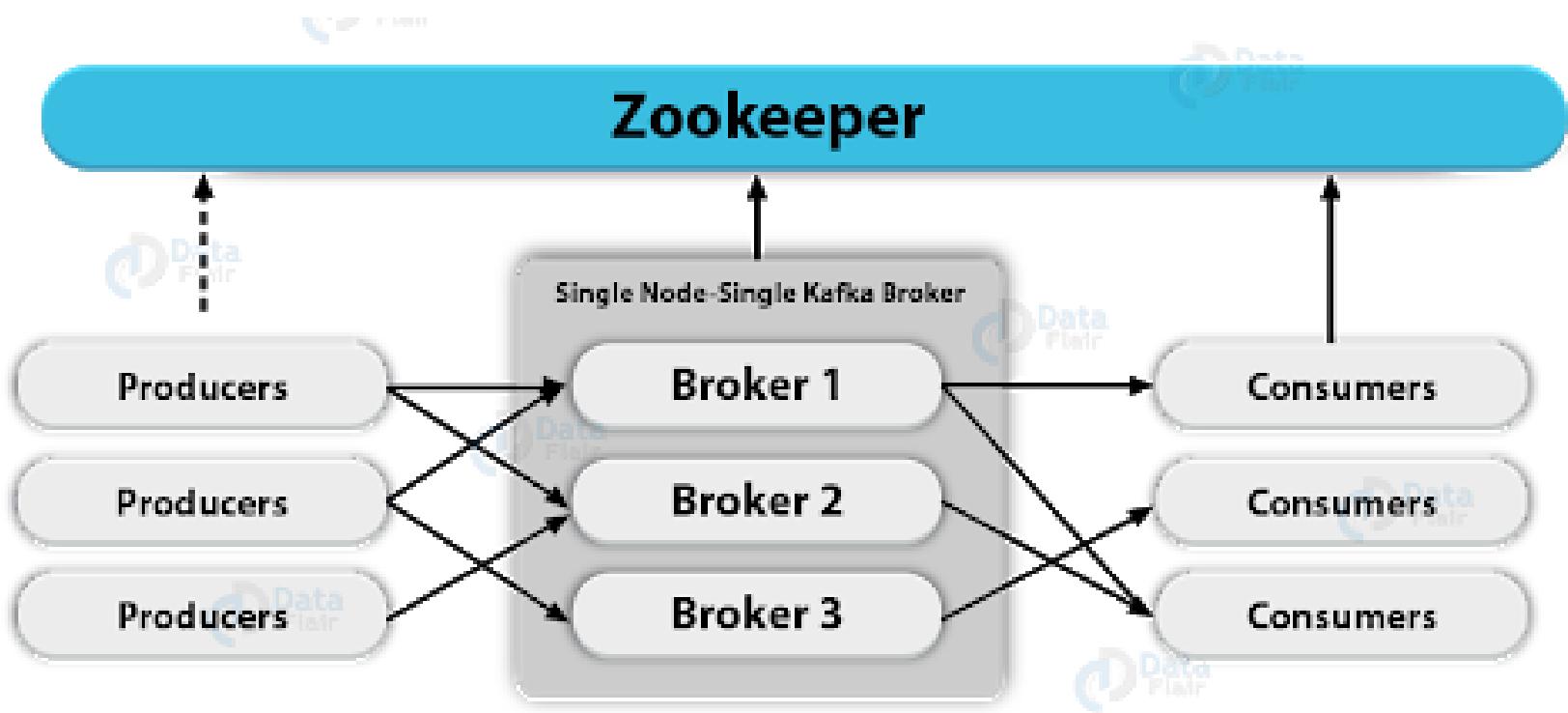
# ZooKeeper Architecture

- Leader: One of the servers is designated a Leader. It gives all the information to the clients as well as an acknowledgment that the server is alive. It would performs automatic recovery if any of the connected nodes failed.
- Follower: Server node which follows leader instruction is called a follower.
- Client read requests are handled by the correspondingly connected Zookeeper server
- The client writes requests are handled by the Zookeeper leader.

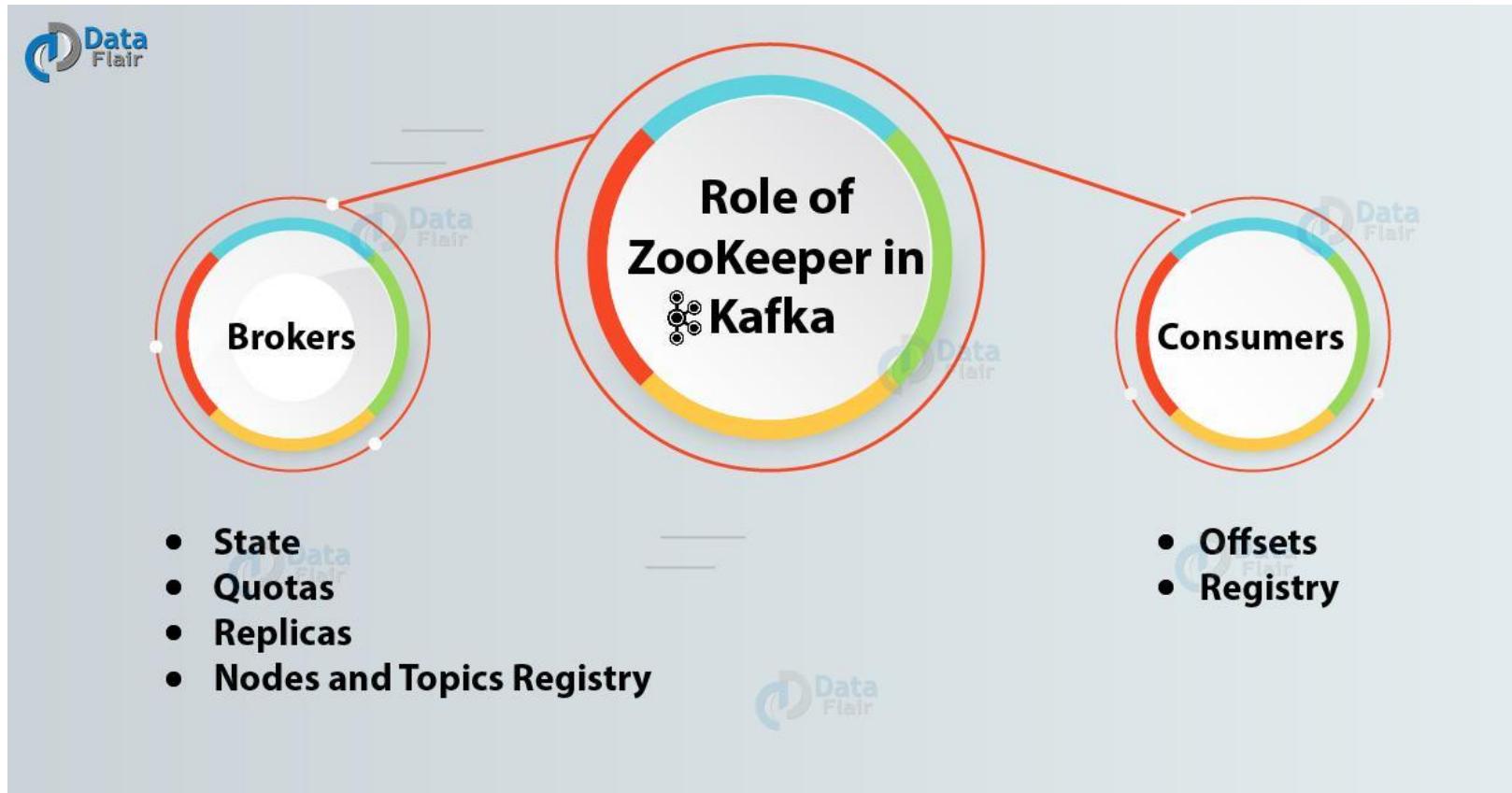
# ZooKeeper Architecture

- Ensemble/Cluster: Group of Zookeeper servers which is called ensemble or a Cluster. You can use ZooKeeper infrastructure in the cluster mode to have the system at the optimal value when you are running the Apache.
- ZooKeeper WebUI: If you want to work with ZooKeeper resource management, then you need to use WebUI. It allows working with ZooKeeper using the web user interface, instead of using the command line. It offers fast and effective communication with the ZooKeeper application.

# What is ZooKeeper?



# ZooKeeper in Kafka



# Companies using Zookeeper



- 
- Yahoo
  - Facebook
  - eBay
  - Twitter
  - Netflix
  - Zynga
  - Nutanix

# What is Kafka

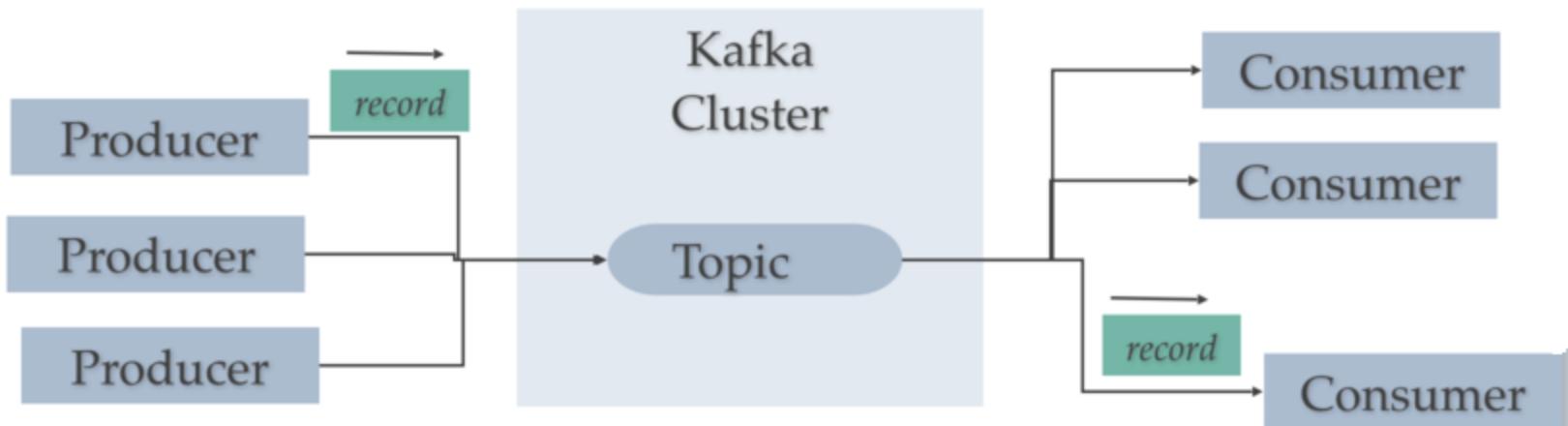
- Kafka consists of Records, Topics, Consumers, Producers, Brokers, Logs, Partitions, and Clusters. Records can have key (optional), value and timestamp.
- Kafka Records are immutable.
- A Kafka Topic is a stream of records ("/orders", "/user-signups").
- You can think of a Topic as a feed name.
- A topic has a Log which is the topic's storage on disk.
- A Topic Log is broken up into partitions and segments.



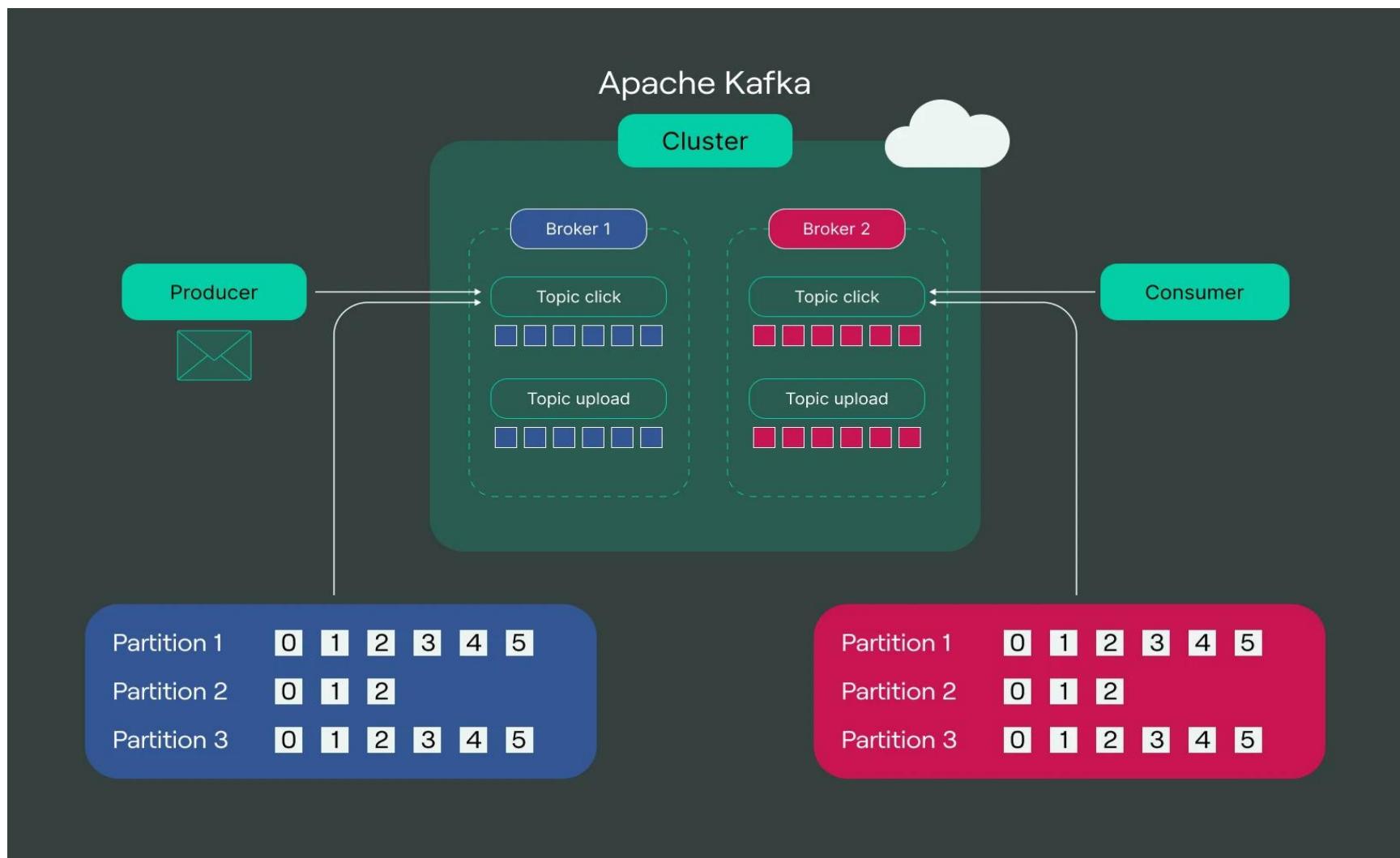
# What is Kafka

- The Kafka Producer API is used to produce streams of data records.
- The Kafka Consumer API is used to consume a stream of records from Kafka.
- A Broker is a Kafka server that runs in a Kafka Cluster.
- Kafka Brokers form a cluster.
- The Kafka Cluster consists of many Kafka Brokers on many servers.
- Broker sometimes refer to more of a logical system or as Kafka as a whole.

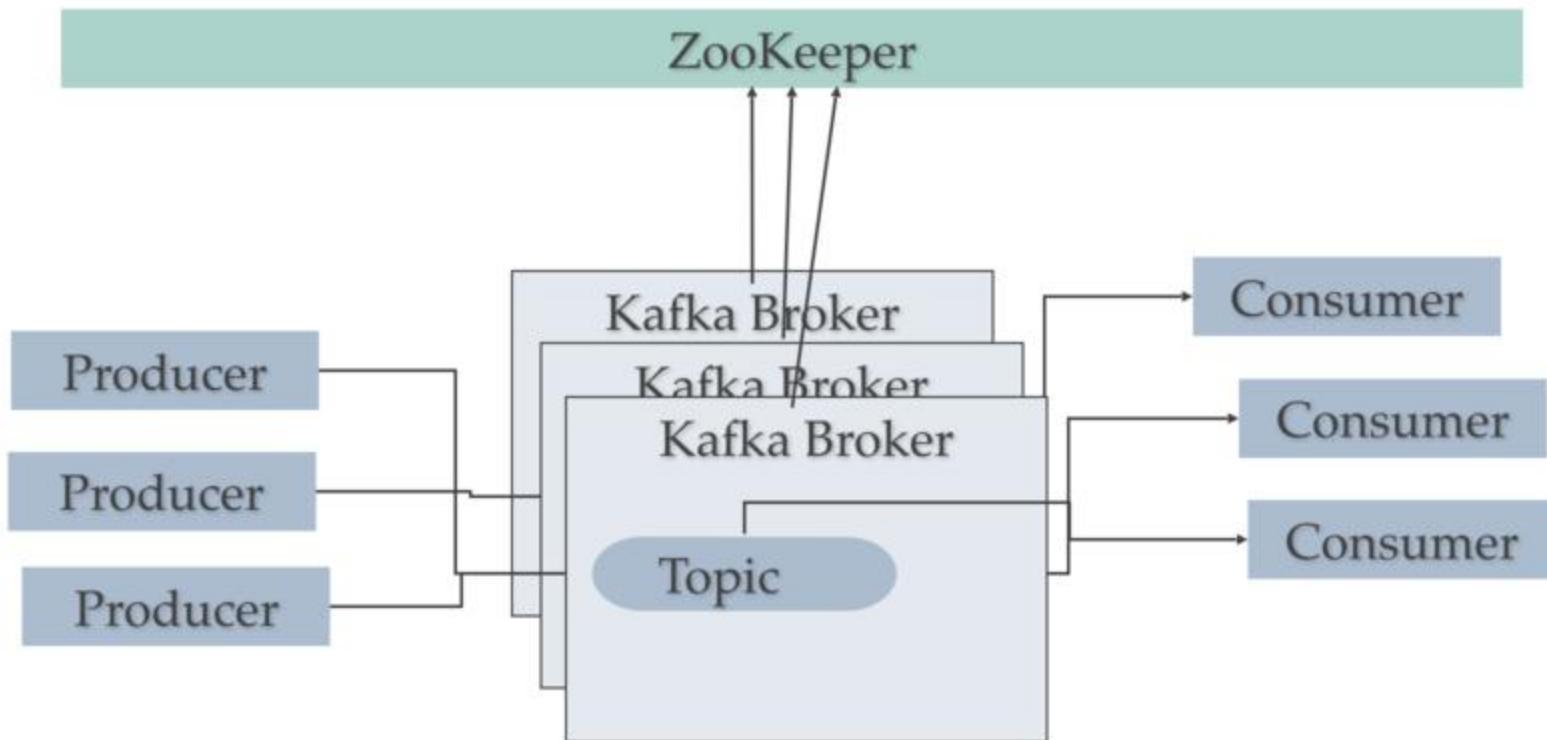
## Kafka: Topics, Producers, and Consumers



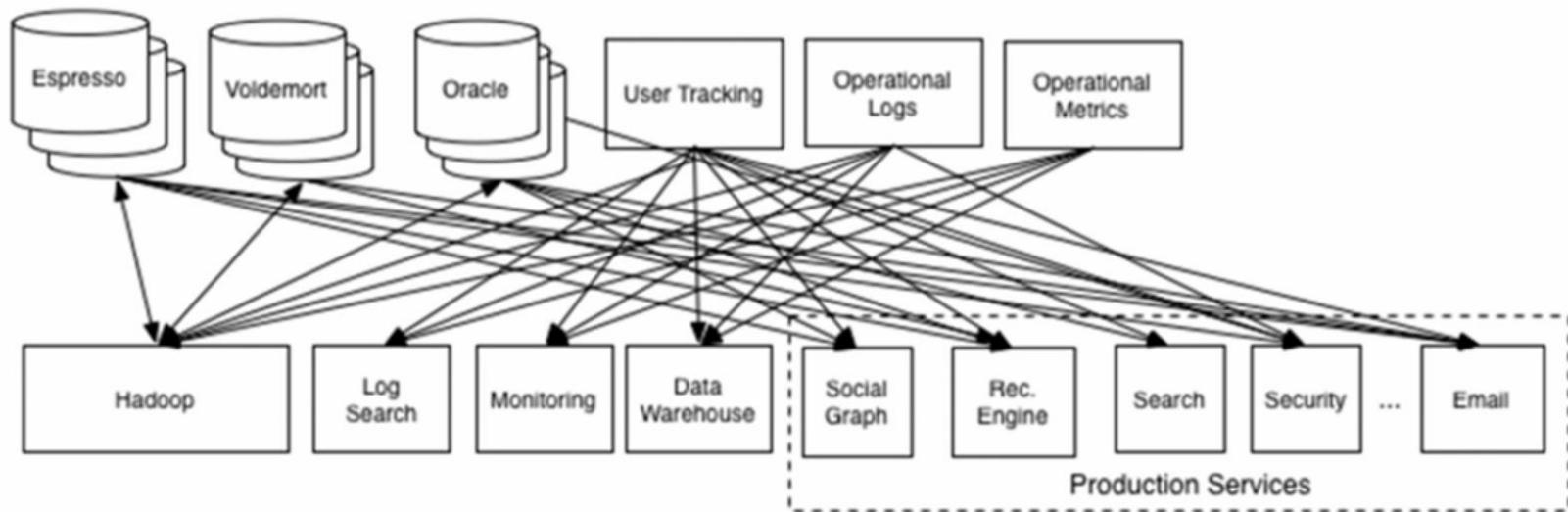
# Apache Kafka



ZooKeeper does coordination for Kafka Cluster



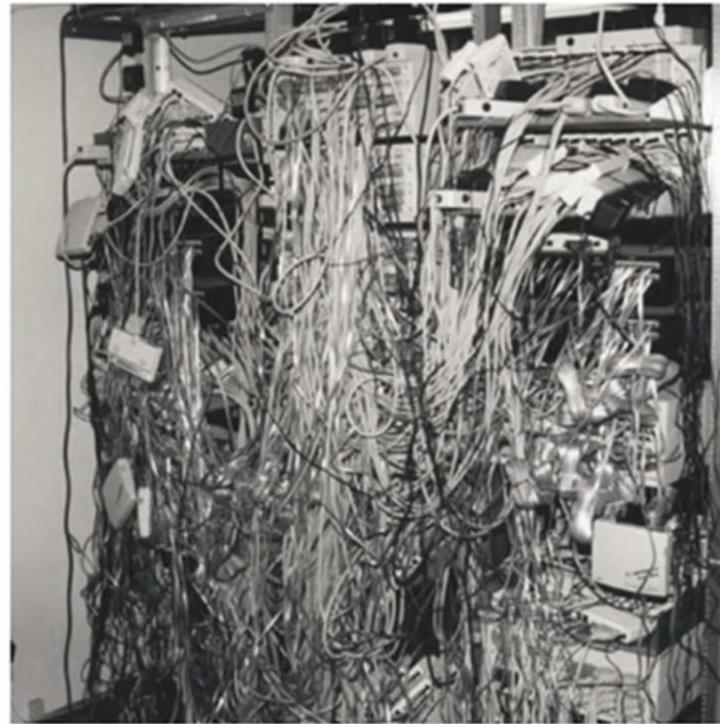
# Why Kafka



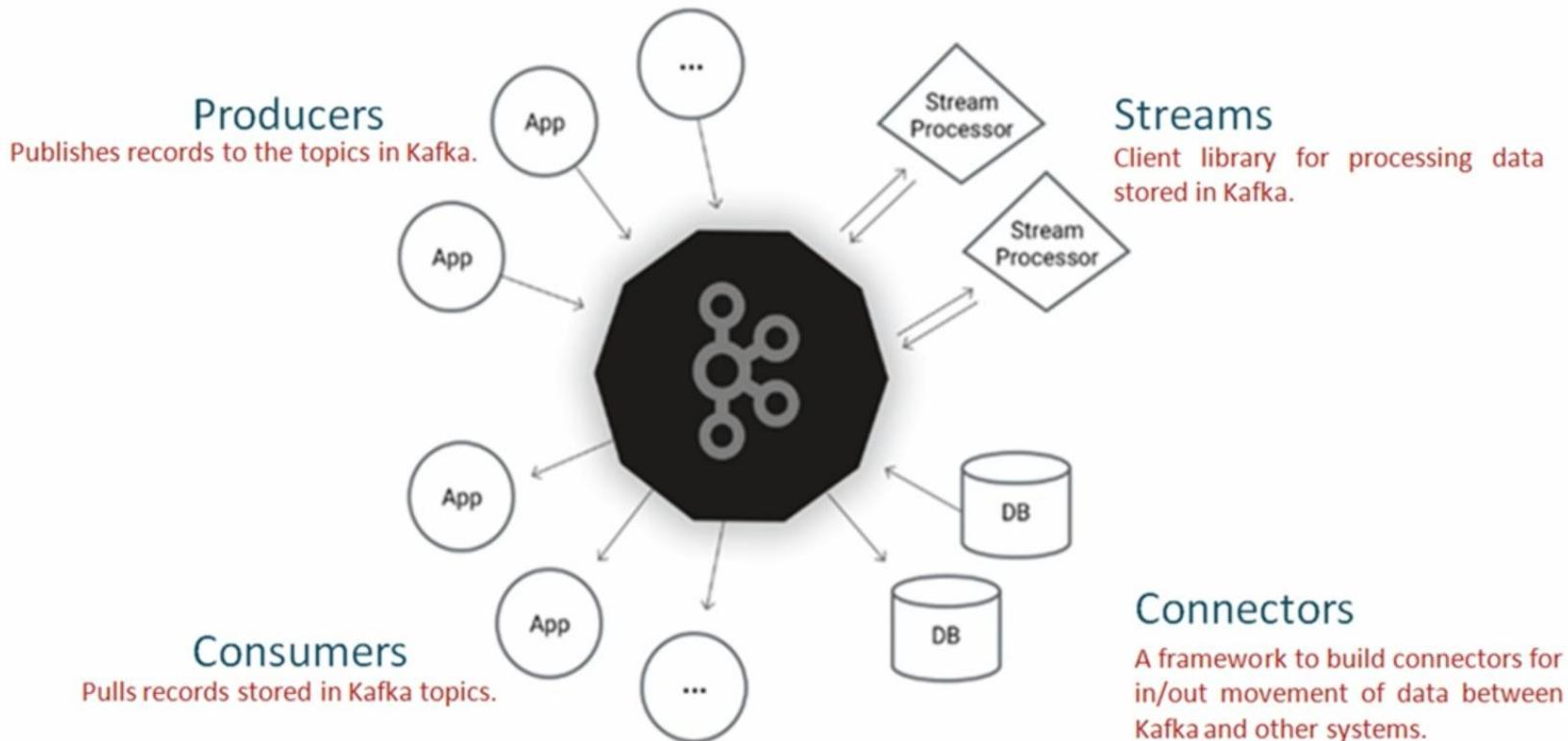


# Why Kafka

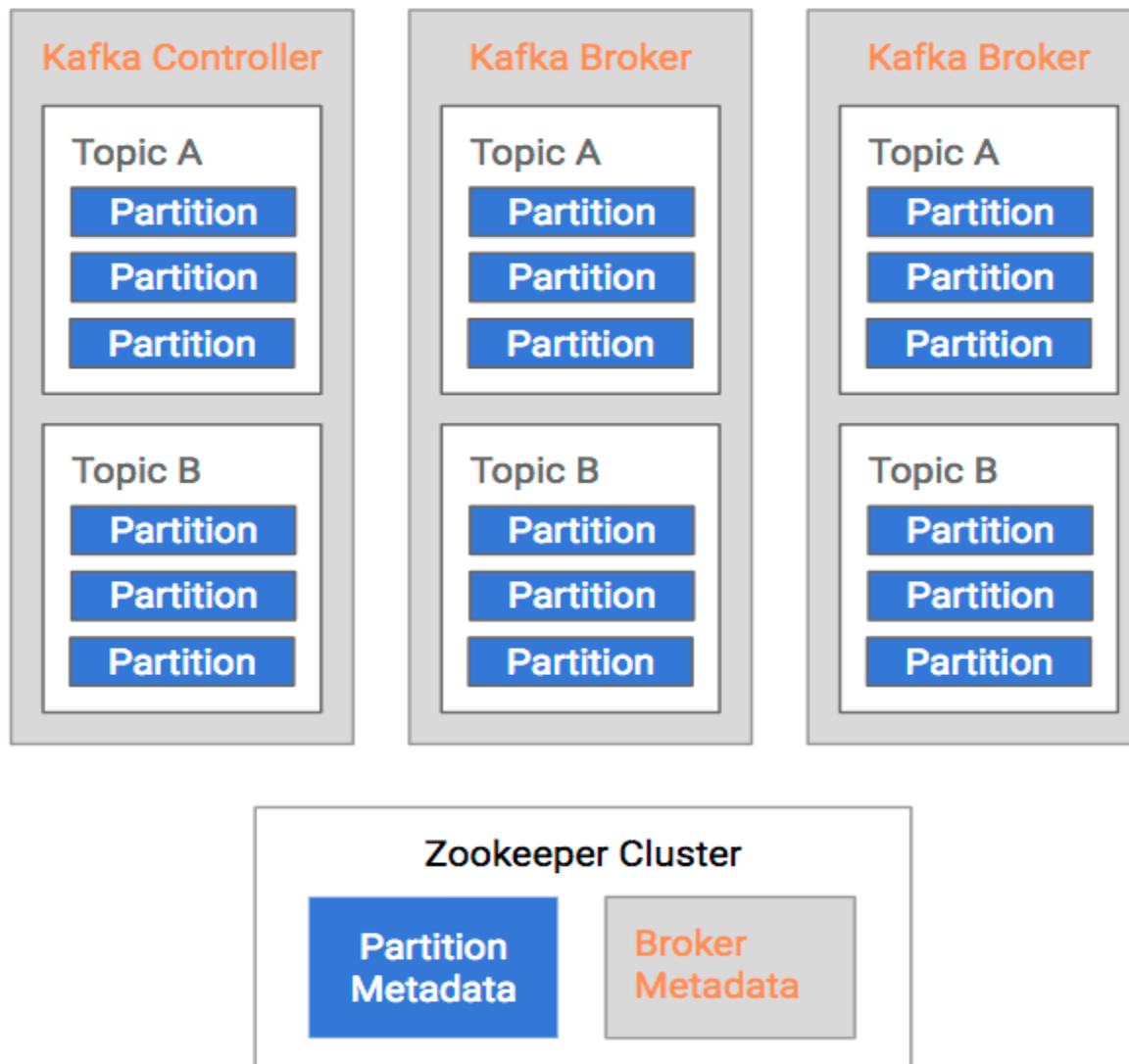
---



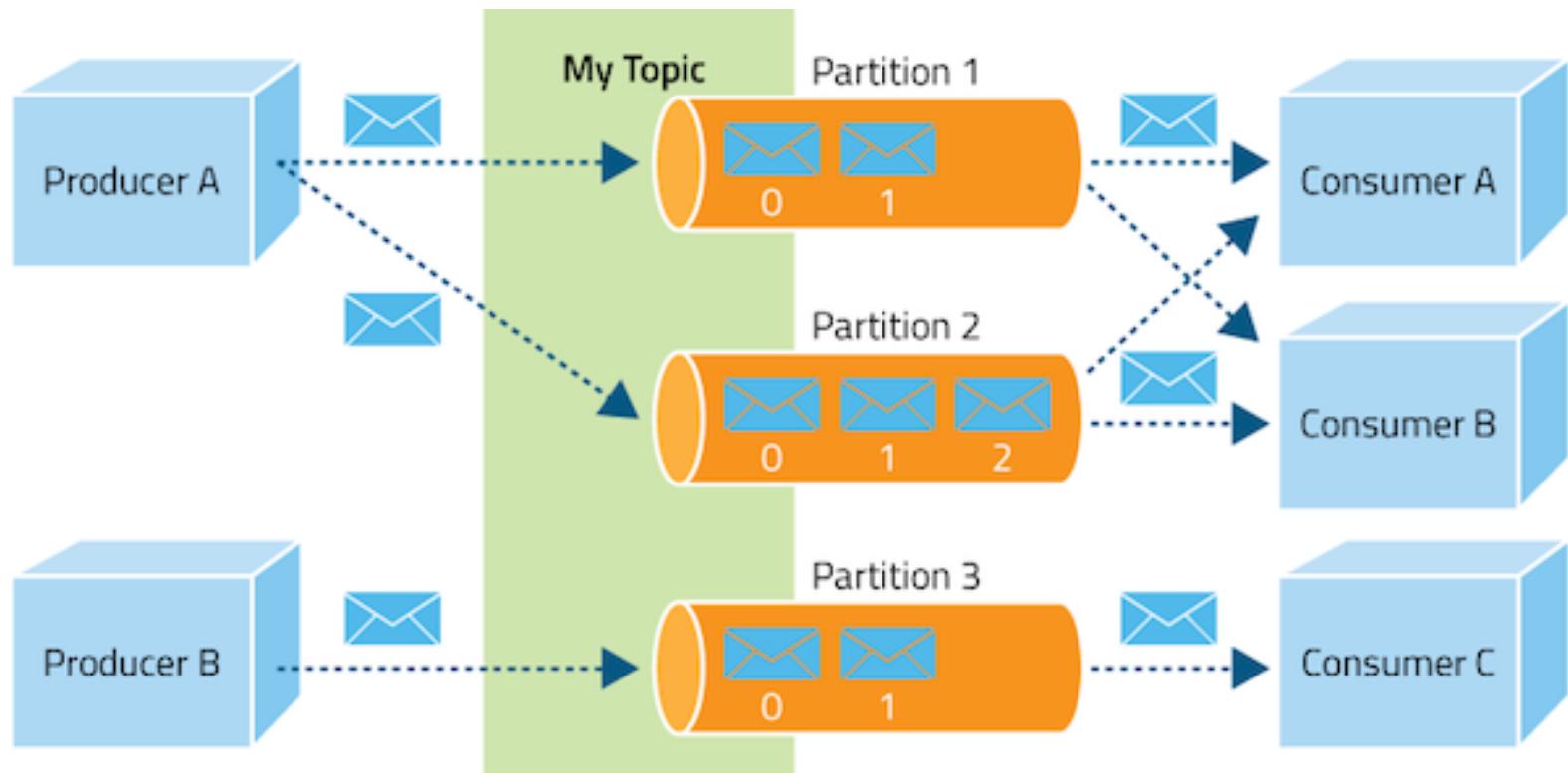
# Why Kafka



# Kafka brokers and Zookeeper



# Kafka brokers and Zookeeper



# Kafka brokers, Topics and Partitions

## Kafka Cluster

The first event  
is offset 0



partition 0  

0	1
---	---

 account-deposits



partition 1  

0	1	2	3
---	---	---	---

 account-deposits



partition 2  

0	1	2
---	---	---

 account-deposits  
partition 4  

0	1	2	3
---	---	---	---

 account-deposits



partition 3  

0	1	2	3	4
---	---	---	---	---

 account-deposits

Offsets increase  
monotonically

Confluent Tiered Storage  
(AWS S3, Google Cloud Storage)

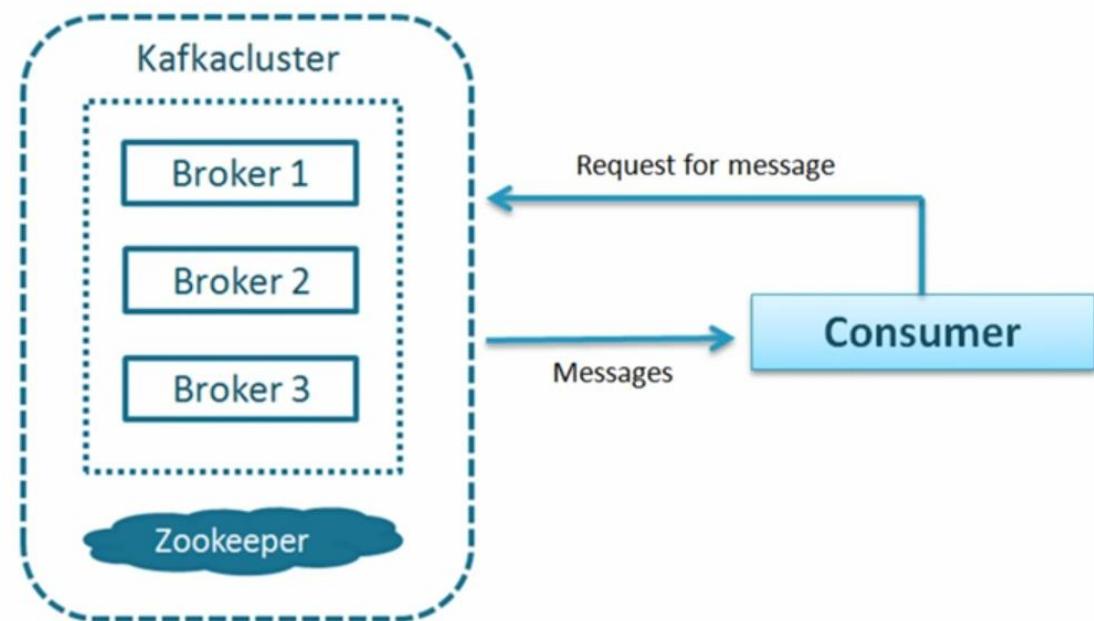
# Kafka brokers and Zookeeper

- 1. Producer
- 2. Consumer
- 3. Broker
- 4. Cluster
- 5. Topic
- 6. Partitions
- 7. Offset
- 8. Consumer groups

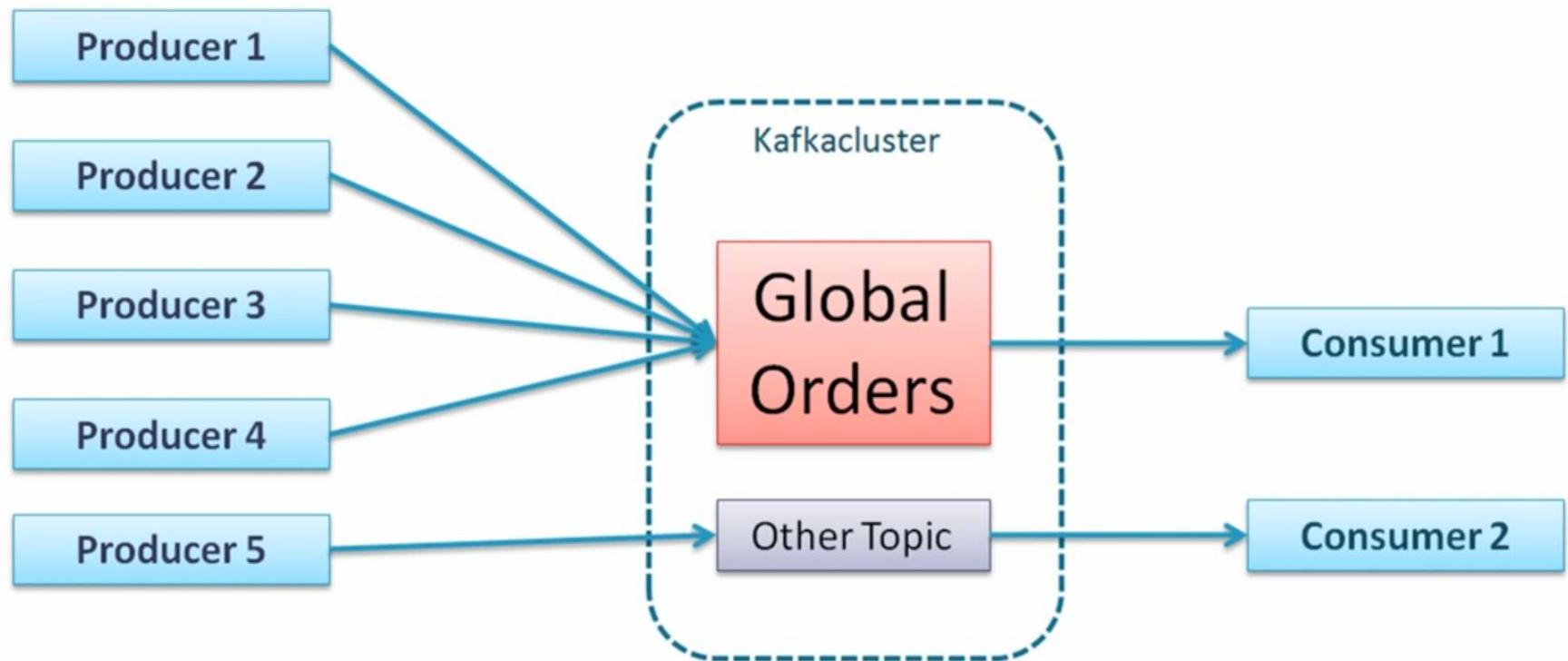


Producer

Send message

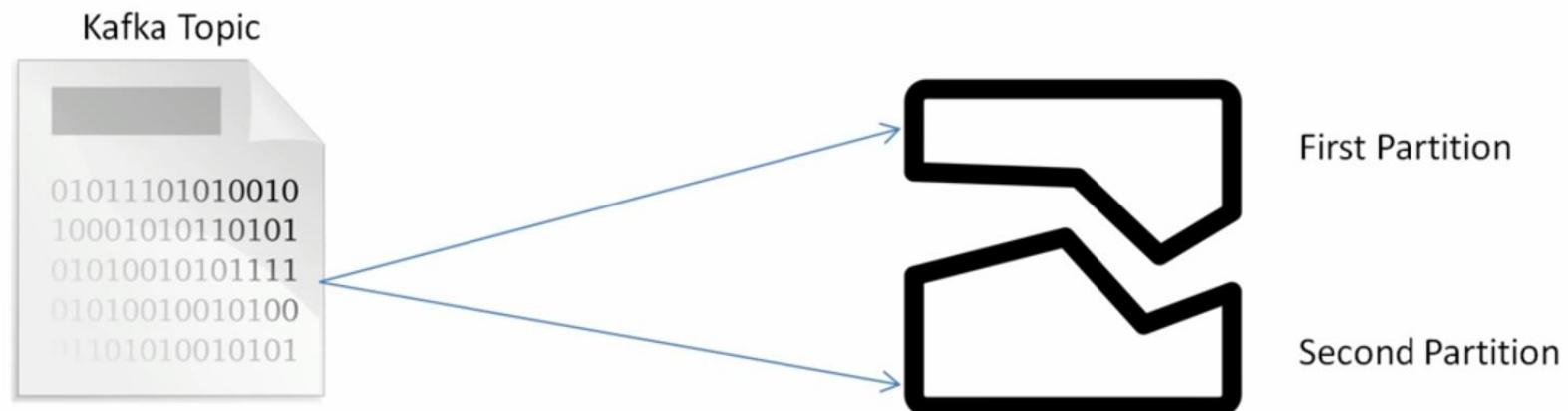


# Kafka brokers and Zookeeper

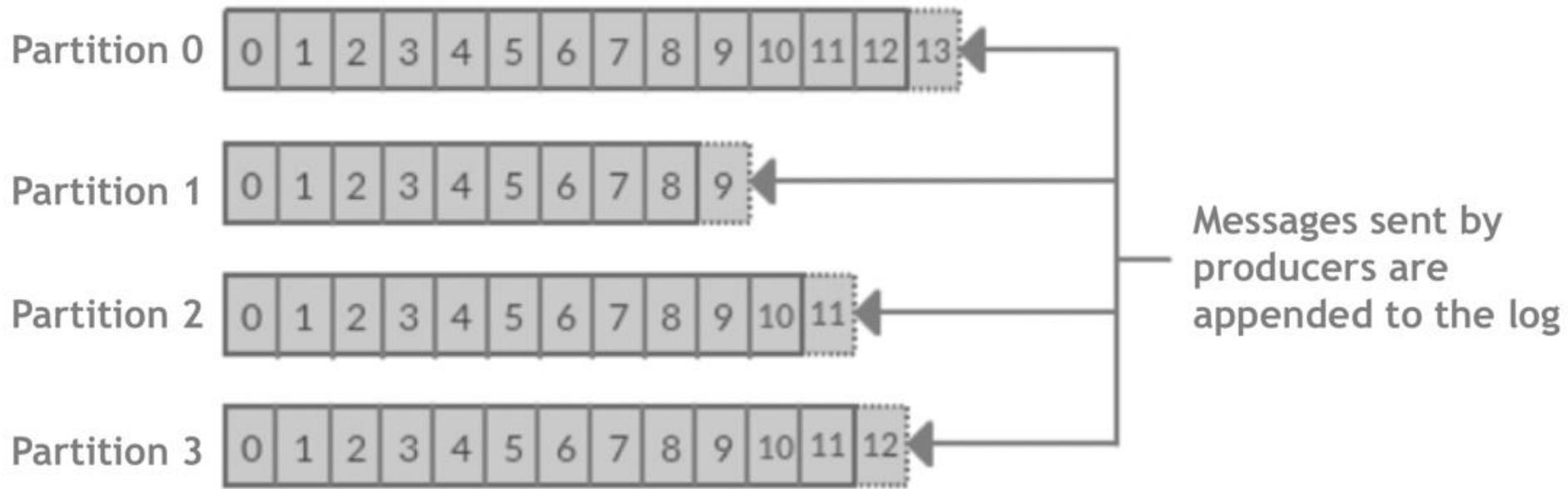


# Kafka brokers and Zookeeper

## What is a Partition?

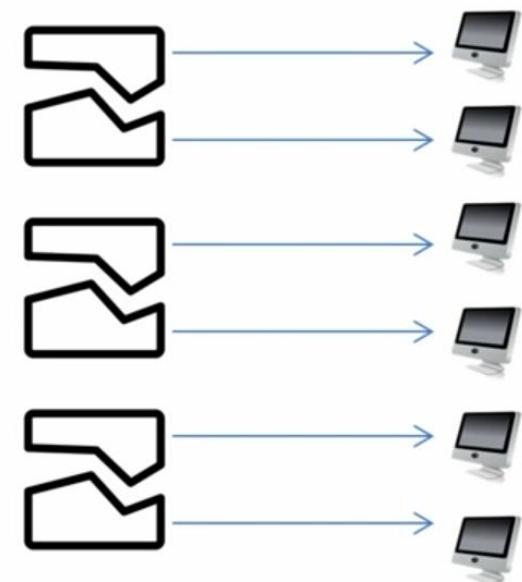


# Kafka brokers and Zookeeper



# Kafka brokers and Zookeeper

How many partitions?

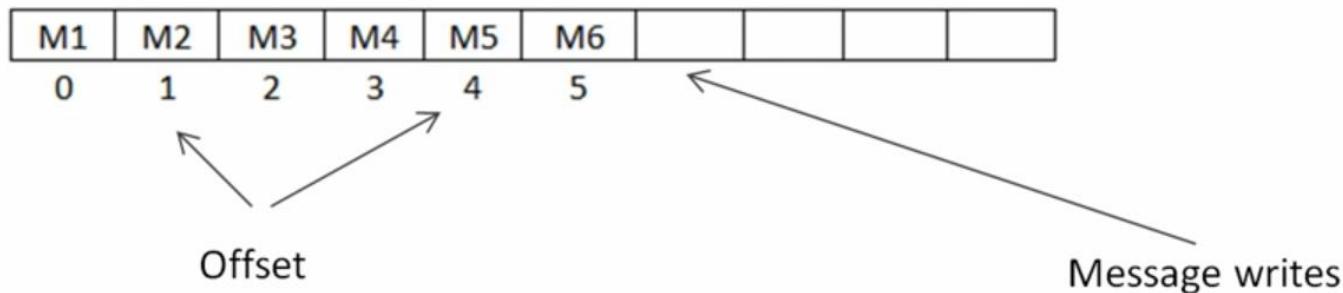


# Kafka brokers and Zookeeper

## What is offset?

A sequence id given to messages as they arrive in a partition.

A Partition



# Kafka brokers and Zookeeper

Gosh! We need to have some identification mechanism.



Consumer



Broker

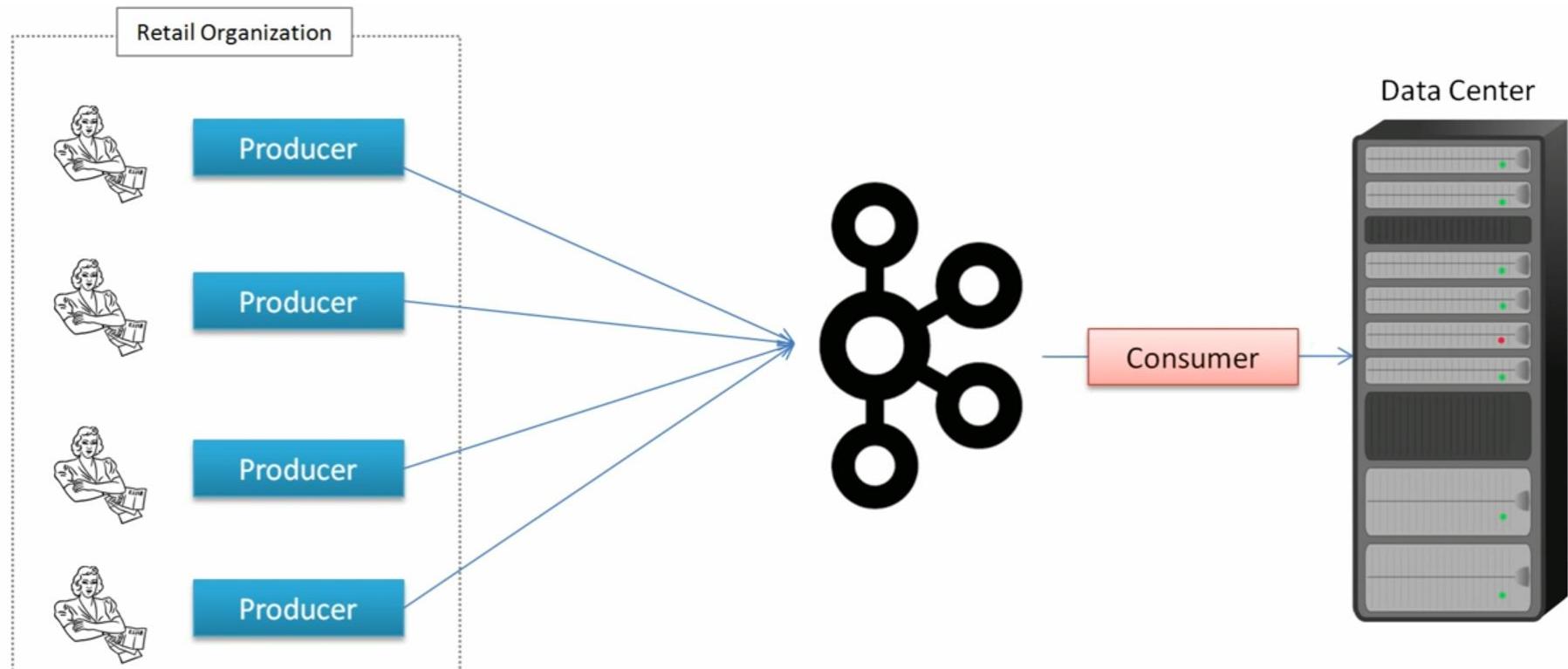
# Kafka brokers and Zookeeper

---

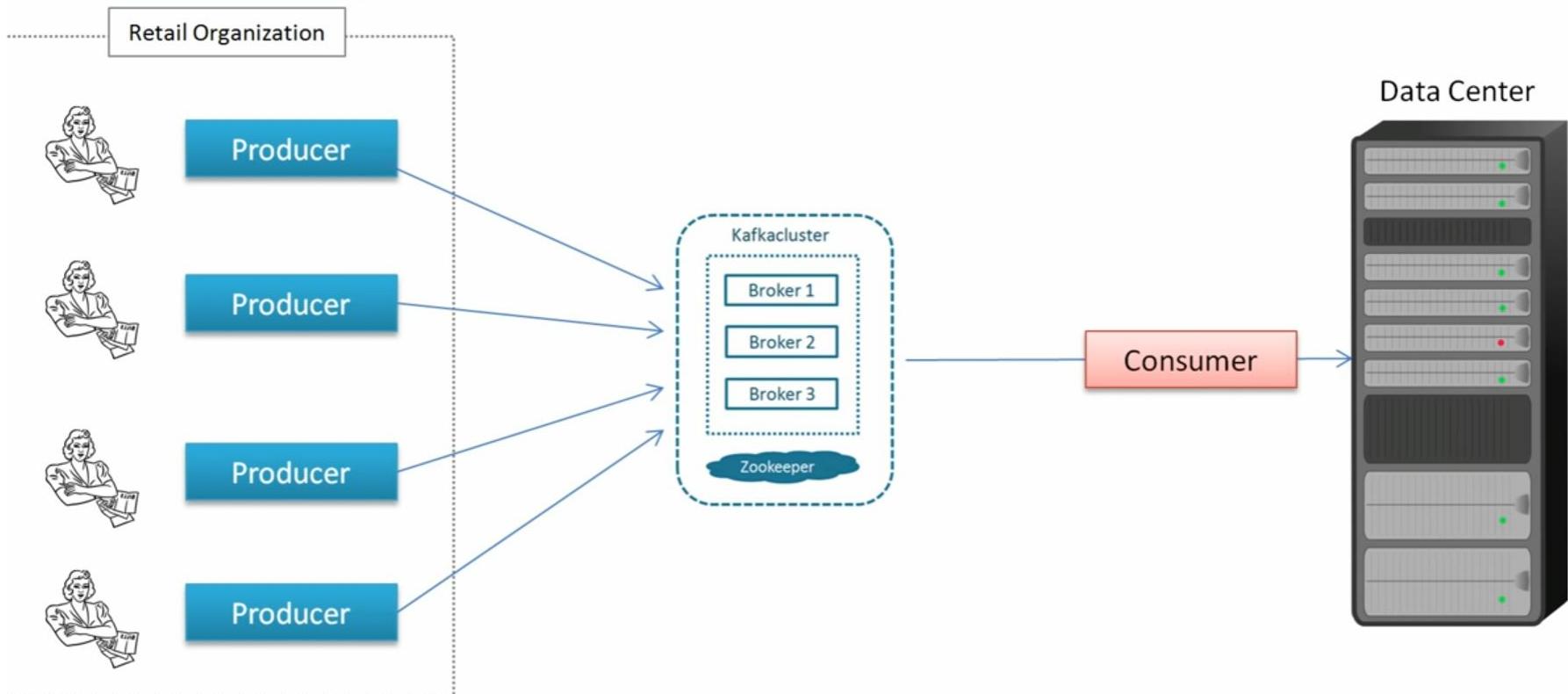
Global unique identifier of a message?

Topic Name → Partition Number → Offset

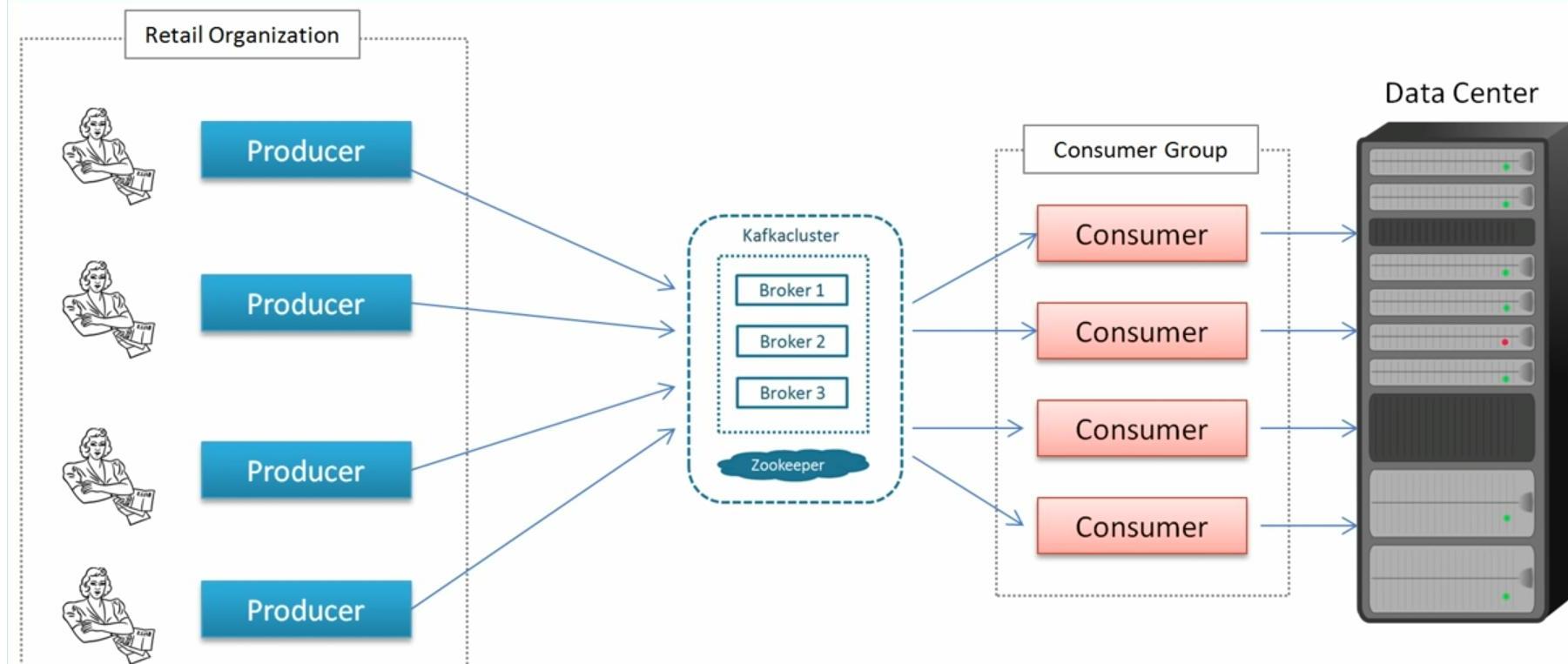
# Kafka brokers and Zookeeper



# Kafka brokers and Zookeeper

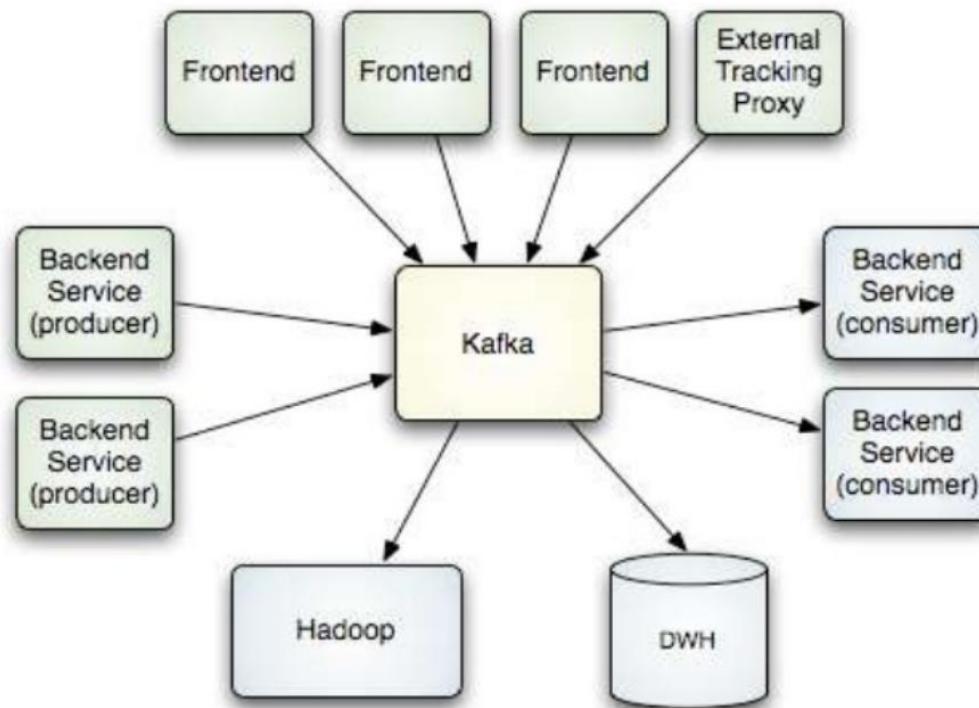


# Kafka brokers and Zookeeper



# Kafka Work Flow

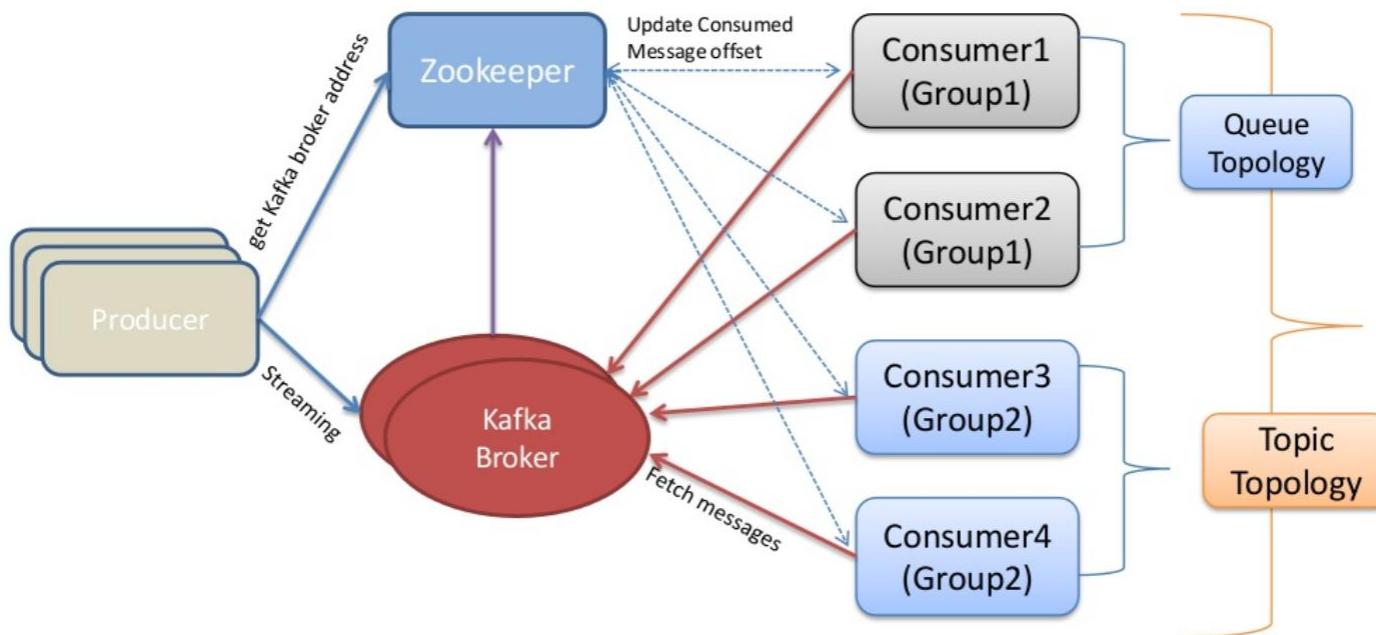
## How it works



Credit : <http://kafka.apache.org/design.html>

# Real Time Transfer

## Real time transfer



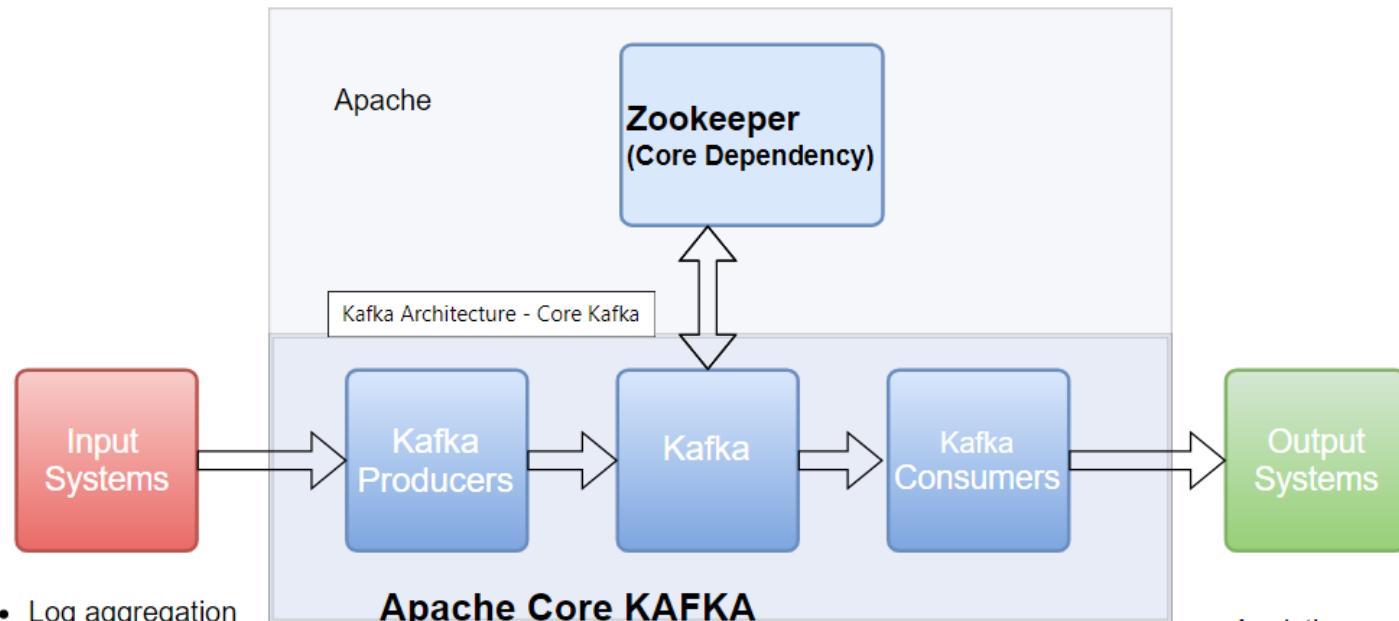
# ZooKeeper in Kafka



- Kafka needs ZooKeeper
- Kafka uses Zookeeper to do leadership election of Kafka Broker and Topic Partition pairs.
- Kafka uses Zookeeper to manage service discovery for Kafka Brokers that form the cluster.
- Zookeeper sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joined, a Broker died, a topic was removed or a topic was added, etc.
- Zookeeper provides an in-sync view of Kafka Cluster configuration.

# Kafka Architecture

## Kafka Architecture: Core Kafka



- Log aggregation
- Metrics
- KPIs
- Batch imports
- Audit trail
- User activity logs
- Web logs

*Not part of core*

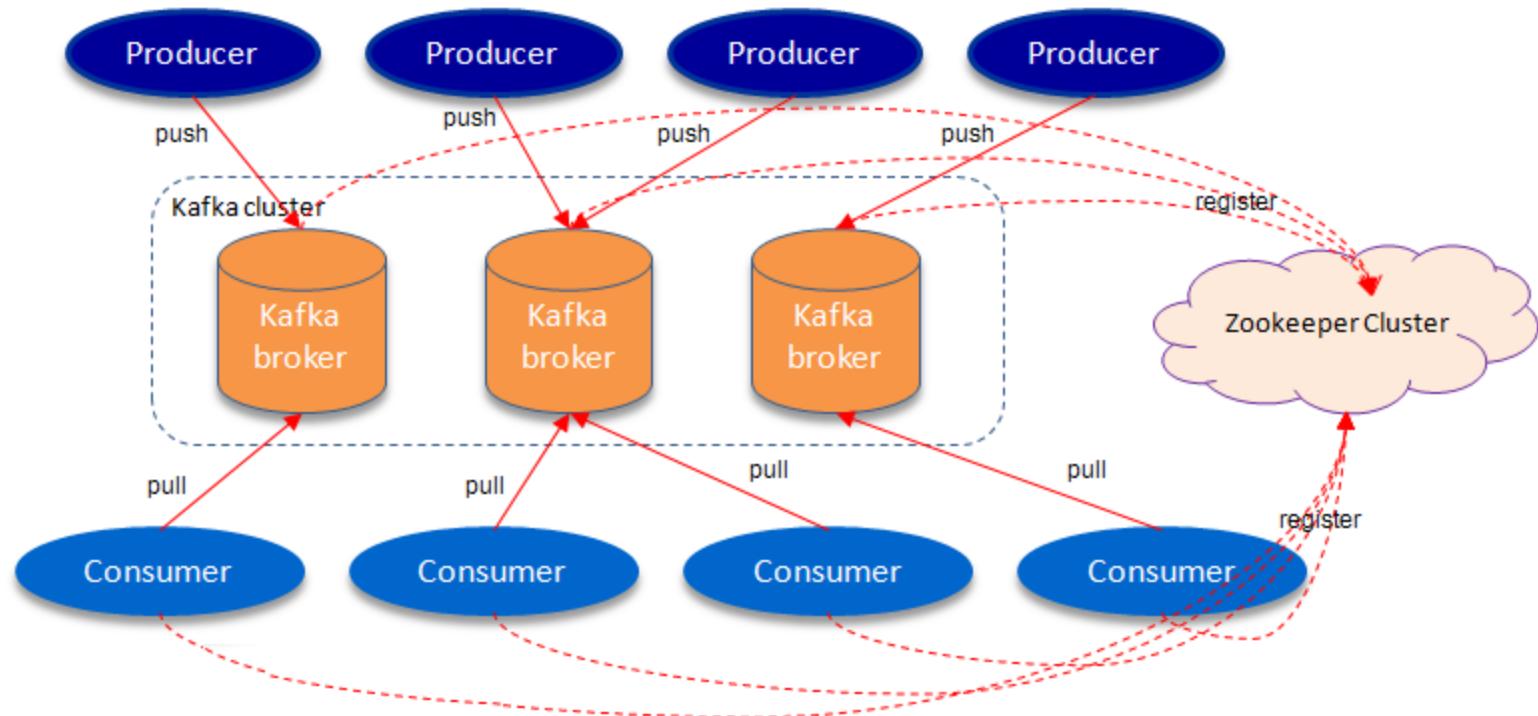
- Schema Registry
- Avro
- Kafka REST Proxy
- Kafka Connect
- Kafka Streams

Apache Kafka Core

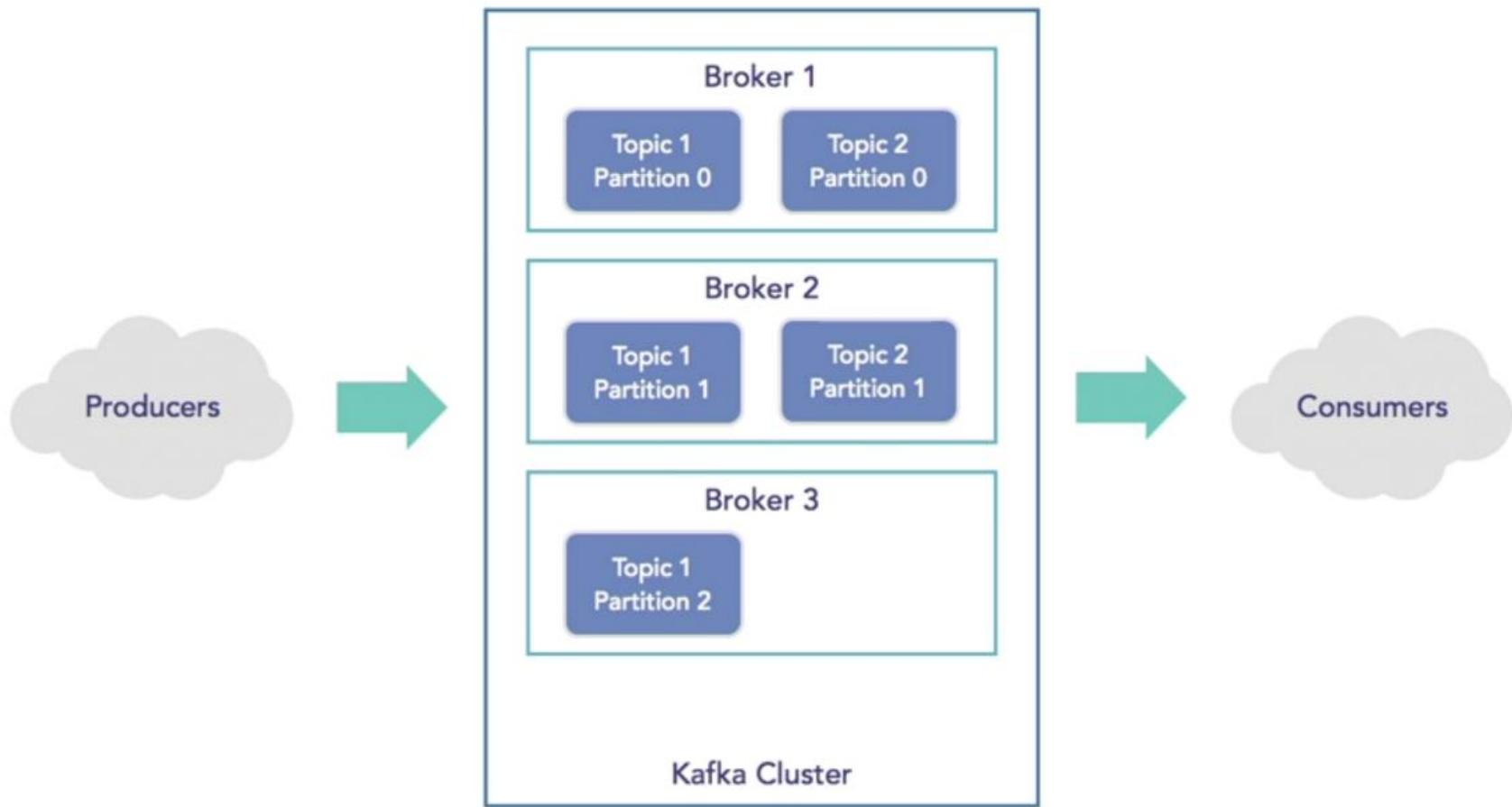
- Server/Broker
- Scripts to start libs
- Script to start up Zookeeper
- Utils to create topics
- Utils to monitor stats

- Analytics
- Databases
- Machine Learning
- Dashboards
- Indexed for Search
- Business Intelligene

# Kafka architecture



# Kafka architecture



Go to

E:\software\A08\file\apache-zookeeper-3.5.6-bin.tar\apache-zookeeper-3.5.6-bin\apache-zookeeper-3.5.6-bin\conf

Zkserver from command prompt

```
2019-11-03 11:48:12,127 [myid:] - WARN  [NIOWorkerThread-1:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:12,992 [myid:] - WARN  [NIOWorkerThread-2:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:13,906 [myid:] - WARN  [NIOWorkerThread-4:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:15,023 [myid:] - WARN  [NIOWorkerThread-6:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:16,051 [myid:] - WARN  [NIOWorkerThread-9:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:17,642 [myid:] - WARN  [NIOWorkerThread-13:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:18,605 [myid:] - WARN  [NIOWorkerThread-12:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:19,724 [myid:] - WARN  [NIOWorkerThread-16:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:20,792 [myid:] - WARN  [NIOWorkerThread-1:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:21,807 [myid:] - WARN  [NIOWorkerThread-5:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
```





```
C:\Windows\System32\cmd.exe -.\bin\windows\kafka-server-start.bat .\config\server.properties
tention.ms -> 86400000, cleanup.policy -> [delete], flush.ms -> 9223372036854775807, segm
ent.ms -> 604800000, segment.bytes -> 1073741824, retention.ms -> 604800000, message.time
stamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760, flush.me
ssages -> 9223372036854775807}. (kafka.log.LogManager)
[2019-11-03 11:43:35,072] INFO [Partition test11-0 broker=0] No checkpointer highwatermar
k is found for partition test11-0 (kafka.cluster.Partition)
[2019-11-03 11:43:35,073] INFO Replica loaded for partition test11-0 with initial high wa
termark 0 (kafka.cluster.Replica)
[2019-11-03 11:43:35,074] INFO [Partition test11-0 broker=0] test11-0 starts at Leader Ep
och 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)
[2019-11-03 11:48:44,857] INFO [GroupCoordinator 0]: Preparing to rebalance group console
-consumer-8213 in state PreparingRebalance with old generation 0 (__consumer_offsets-33)
(reason: Adding new member consumer-1-217673c0-2a77-44ea-b5dc-5b486eefded2) (kafka.coordin
ator.group.GroupCoordinator)
[2019-11-03 11:48:44,868] INFO [GroupCoordinator 0]: Stabilized group console-consumer-82
13 generation 1 (__consumer_offsets-33) (kafka.coordinator.group.GroupCoordinator)
[2019-11-03 11:48:44,882] INFO [GroupCoordinator 0]: Assignment received from leader for
group console-consumer-8213 for generation 1 (kafka.coordinator.group.GroupCoordinator)
[2019-11-03 11:51:33,455] INFO [GroupMetadataManager brokerId=0] Removed 0 expired offset
s in 3 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```



Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-producer.bat --broker-list localhost:9092 --topic test

[2019-11-03 11:43:19,657] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'test' already exists.  
(kafka.admin.TopicCommand\$)

E:\software\A08\file\kafka\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test11  
Created topic test11.

E:\software\A08\file\kafka\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topic test  
>hi  
>how are you  
>enjoy  
>





Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-consumer.bat --bootstrap-server localhost9092 --topic test --from-beginning  
consumption.

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test  
Processed a total of 0 messages  
Terminate batch job (Y/N)? y

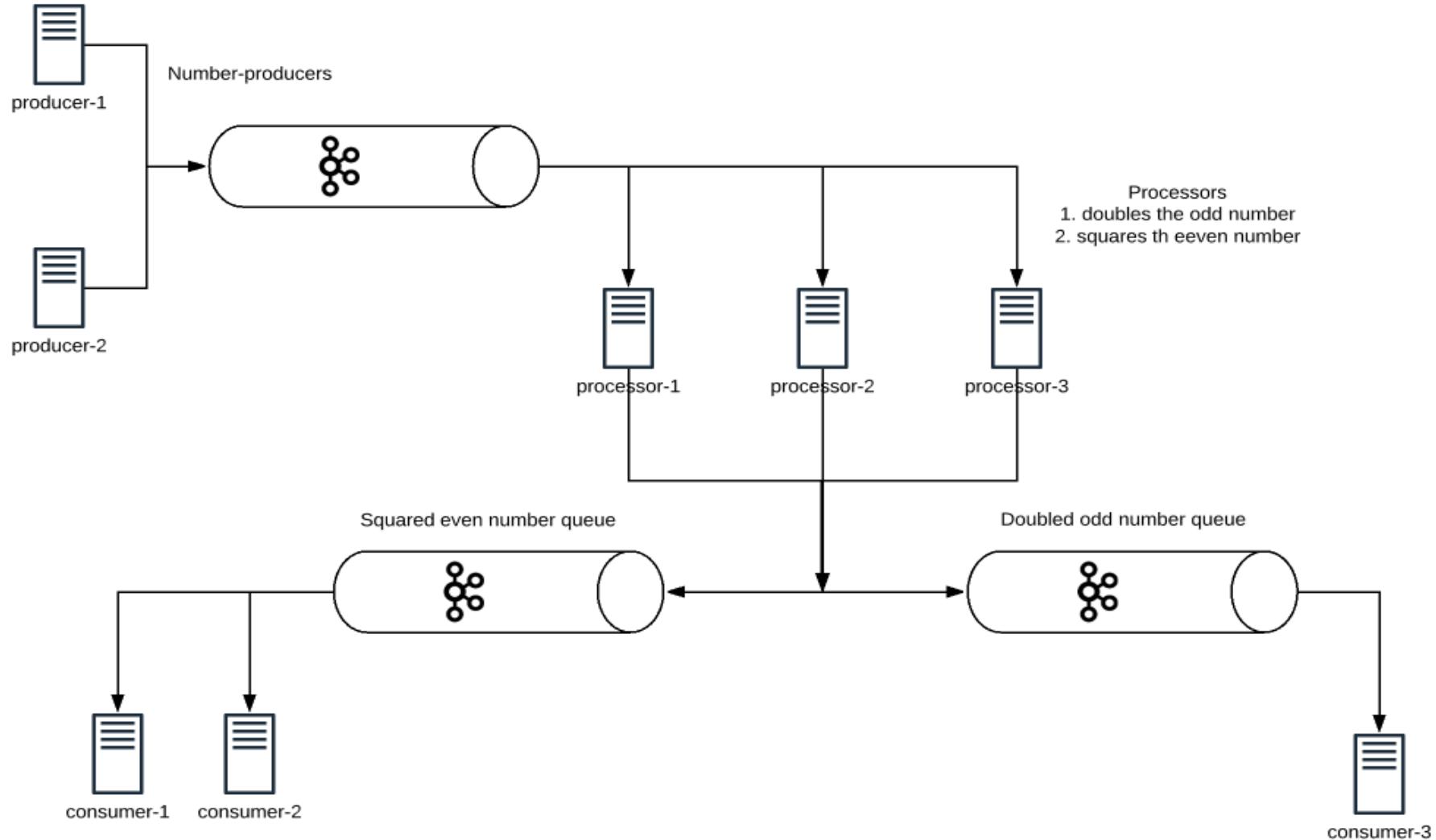
E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test  
Processed a total of 0 messages  
Terminate batch job (Y/N)? y

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning  
hi  
how are you  
enjoy



# Process the incoming messages, filter and split into multiple topics based on certain conditions.

Apache Kafka





# Kafka vs RabbitMQ and Others

Category	Kafka	RabbitMQ	Amazon SQS	Google Pub/Sub
Message Rate (msgs/sec)	100k+	20k+	Varies (can be scaled)	10k+
Message Acknowledgements	No	Yes	Yes	Yes
Built in UI	No	Yes	Yes	Yes
Protocol	Kafka	AMQP	HTTP REST	HTTP REST
Additional features	Apache Storm, etc	Several plugins available to add more features	In-flight messages	-
Use cases	High ingestion platforms where speed, scale and efficiency is the prime concern	Robust systems where features like acknowledgements, deeper management are important	Useful when deploying applications on AWS EC2, Elastic Beanstalk to facilitate low-latency communication	Works well with applications deployed on GCE

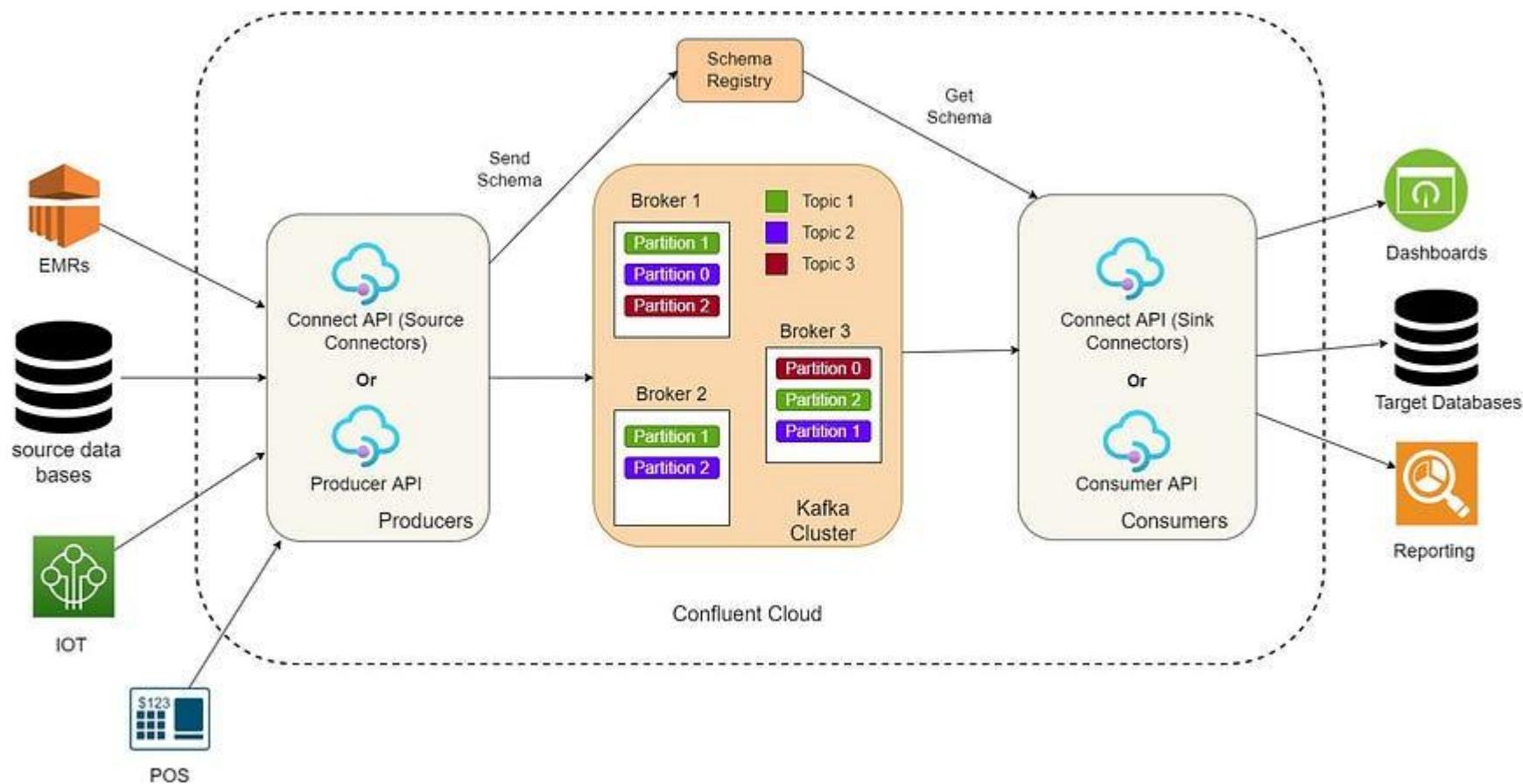


## Confluent Kafka

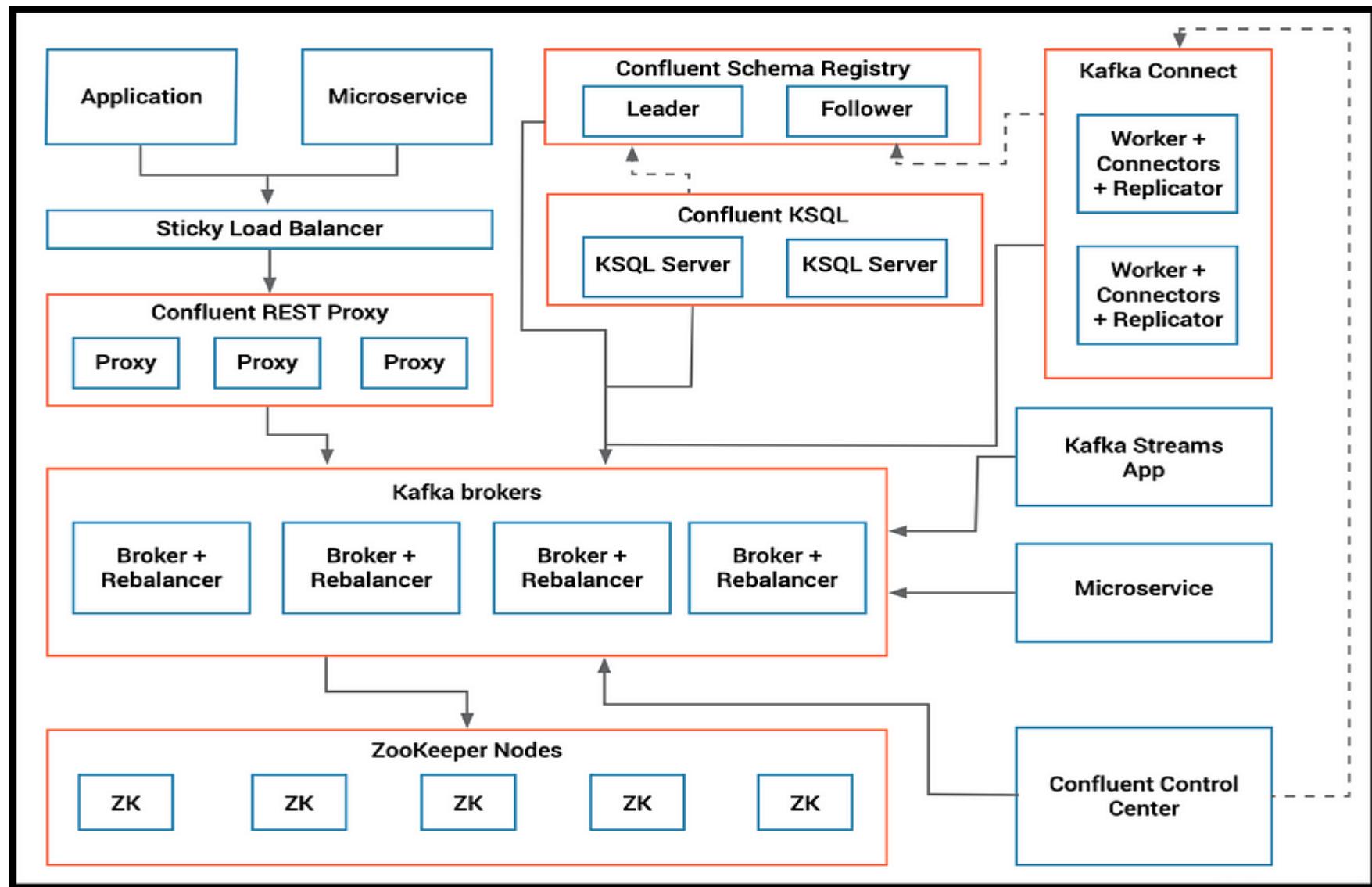
---

- A commercial distribution of Apache Kafka developed by Confluent Inc. (the company founded by the original creators of Kafka).
- Confluent Kafka extends the core Kafka functionality with additional enterprise-grade features and managed services.

# Confluent Kafka Architecture



# Confluent Kafka Architecture



# 1. ZooKeeper:



- Manages distributed processes, cluster memberships, and elects cluster controllers.
- High availability requires 5 Zookeeper nodes (2 for failure node)
- Number of ZooKeeper nodes must be odd.

## 2. Kafka Brokers:

---

- Main storage, messaging components, and maintains streams of ordered logs of messages in Topics.
- Contains clusters that maintain streams of messages called Topics.
- Topics are shared into partitions of ordered immutable logs of messages.
- Partitions are replicated and distributed for high availability.
- Need 4 Kafka Brokers in a cluster each replicating partition on a separate server/node.
- Highly loaded Brokers should have their own nodes otherwise they can share Zookeepers but separate disks/controllers for Zookeepers.



### 3. Kafka Connect Workers:

- Stateless Workers integrate external systems to pull/push data on source/target.
- Pluggable JDBC Connectors in Avro or JSON format to Azure Data Lake Storage Gen2 Sink.
- The Kafka Connect Snowflake Sink connector maps and persists events from Topics directly exposing the data to services for querying, enrichment, and analytics.

### 3. Kafka Connect Workers:

- ***Standalone Mode*** — File or Spooling Directory Connector on the Producer machine to read the file and send events to Kafka.
- ***Cluster Mode*** — Deployment on multiple machines discover each other and Brokers serving as synchronization layer automatically load-balance and failover work between them.
- Manage Connectors on cluster using Connect Node (aka. “Worker”) uses REST API to configure, start, stop, pause and resume. Once connector starts regardless of node, it will load balance parallel tasks to the least loaded available worker node.
- Because they are stateless, they can be deployed in containers.

## 4. Kafka Clients



- Java client JARs interfaces applications to Kafka.
- • Are installed along Kafka Brokers but deployed with application client libraries.
- • APIs and semantics in variety programming languages (C, C++, Python, and Go).

## 5. Kafka Streams APIs



- Application embedded library for building distributed stream processing applications for late-arriving data.
- Benefits from higher cpu/core count.
- Deploying multiple instances of application on multiple servers Kafka Streams will auto load-balance and fail-over.

## 6. ksqlDB Server



- SQL engine that queries continuously against Apache Kafka in real-time.
- Command Line Interface allows interactive queries to ksqlDB server from any machine.
- ksqlDB server includes data processing to the target Kafka cluster.
- ksqlDB is deployed on a set of servers/nodes that form a cluster determined by processing capacity required (concurrent queries/complexity) handling auto load-balancing/fail-over.
- Benefits from higher CPU counts, networking throughput and SSD storage.

## 7. Confluent REST Proxy

---

- HTTP server that provides RESTful interface to Kafka cluster.
- Mandatory component when using RESTful HTTP protocol on separate machine/node.
- If application uses Native Kafka Client then no need to deploy REST Proxy.
- Deploy multiple REST Proxies behind a sticky load-balancer for the same Consumer.



## 8. Confluent Schema Registry

- Metadata layer using RESTful interface for storing and retrieving Avro schemas.
- Stores versioned history of all schemas and allows compatibility settings.
- Includes Message Serializers that plug into Kafka clients
- Handles Schema Storage and retrieval of Kafka messages from Avro format.
- Installed on its own server/node. But small installations can be alongside REST Proxy and Connect Workers.

## 8. Confluent Schema Registry

---

- High availability requires multiple Schema Registry servers/nodes.
- Multi Schema Registry uses a leader-followers architecture with at most 1 node being Leader any given time.
- Only Leader can publish writes to Kafka Log where all nodes can process read requests.
- Follower nodes forward write requests to Leader.
- Schema Registry stores its schemas in Kafka and do not require storage and can be deployed in containers.

## 9. Confluent Replicator

---

- Manages multi-cluster deployments of Kafka.
- Provides centralized configuration of cross-cluster replication.
- Replicates Topic configuration in addition to Topic Messages.
- Integrated with Kafka Connect framework and be installed in Connect Nodes in destination cluster. For Multiple Connect Workers the Replicator should be installed in all of the Connect Nodes.
- Large number of Replicator Nodes will have high availability with built-in fail-over.



## 10. Confluent Auto Data Balancing

- Optimizes resource utilization to scale Kafka Clusters.
- Installed on any machine in the Confluent Platform cluster to communicate to Brokers and ZooKeeper but recommended install on the Brokers or on Control Center Node.
- Collects load metrics and sends instructions to move partitions.

## 11. Confluent Control Center

---

- Web based tool for managing and monitoring data pipelines and streaming applications on Apache Kafka.
- Data Stream Monitoring and Alerting using drill-down from Producer to Consumer.
- Multi-cluster monitoring and management data replication between clusters.
- Kafka Connect Configuration to add new sources to load external data systems.
- Runs on a single dedicated machine.



# Confluent Kafka

Feature	Description
Confluent Control Center	GUI for <b>monitoring, managing, and troubleshooting</b> Kafka clusters.
Schema Registry	Helps enforce <b>schema evolution</b> and compatibility for message formats (e.g., Avro, JSON, Protobuf).
Kafka Connect	A library for <b>connecting Kafka to external systems</b> using pre-built connectors (e.g., databases, S3).
ksqlDB	Allows real-time <b>stream processing using SQL-like syntax</b> without writing Java code.
Enhanced Security	Includes <b>RBAC (Role-Based Access Control), OAuth, encryption, and audit logging</b> .
Multi-Region Clustering	Enables Kafka replication and failover across geographic locations.
Managed Services (Cloud Offering)	Fully managed Kafka clusters on <b>AWS, Azure, or GCP</b> via Confluent Cloud.
Tiered Storage	Moves cold data to <b>cost-effective cloud storage</b> , reducing storage costs.

# Key Differences

Aspect	Apache Kafka	Confluent Kafka
<b>Open-source License</b>	Apache 2.0 License	Open-core: Apache Kafka + proprietary features under Confluent Community License
<b>Ease of Use</b>	Requires manual installation, scaling, and config	Confluent simplifies deployment, management, and scaling (especially for enterprise).
<b>Schema Management</b>	No native schema enforcement	Built-in Schema Registry for enforcing data format consistency.
<b>Connectors</b>	Limited (need custom or open-source plugins)	Large set of pre-built connectors for popular data sources and sinks.
<b>Monitoring &amp; Management</b>	Limited (needs custom tools like Prometheus)	Confluent provides Control Center for full visibility.
<b>Security</b>	Basic (SASL, SSL, ACL)	Enterprise-grade security features such as RBAC, audit logs, and OAuth integration.
<b>Cloud-Native/Managed Option</b>	No official support (but cloud vendors offer it)	Fully managed Confluent Cloud on AWS, GCP, and Azure.
<b>Cost</b>	Free (self-managed)	Paid plans for enterprise features (on-prem and cloud).



# Key Differences

Use Case	Recommended Option
Small-scale, on-prem, or non-critical workloads	Apache Kafka
Enterprise use cases needing built-in security, monitoring, and connectors	Confluent Kafka
Want to minimize infrastructure management	Confluent Cloud
Simple development/testing environments	Apache Kafka
Real-time stream processing without coding	Confluent Kafka (ksqlDB)



# Confluent Kafka

- Refer F:\Local disk\ Docker\kafkahelm\helm

# Confluent Kafka



localhost:9021/clusters/b4b84cc9-ab20-4a83-ac0/overview

Click to go forward, hold to see history New Tab How to use Assertio... Browser Automatio... node.js - How can I... Freelancer-dev-810... Courses New Tab Airtel 4G Hotspot All Bookmarks

CONFLUENT Enterprise trial ends in 29 days

HOME > CONTROLCENTER.CLUSTER >

**Overview**

**Brokers**

3	38.79K	32.13K
Total	Production (bytes / second)	Consumption (bytes / second)

**Topics**

60	--	--	--
Total	Partitions	Under replicated partitions	Out-of-sync replicas

New

# Quick Summary of Options

Option	When to Use
New consumer group	If you want a fresh start without affecting previous offsets.
Seek to the beginning	If you want to force consumption of all messages without changing the group ID.
Delete committed offsets	If you want the current consumer group to reprocess all messages.
Manual partition assignment	If you want complete control over message consumption and offset management.

# Error Handling Scenarios

Scenario	Solution
Transient network errors	Use Kafka's built-in retry mechanism with <code>Retries</code> and <code>RetryBackoffMs</code> .
Message processing errors	Implement manual retry logic as shown in the consumer example.
Brokers unavailable	Set <code>MessageTimeoutMs</code> and handle retries in the producer.
Duplicate messages	Use <code>EnableIdempotence</code> on the producer to avoid sending duplicates during retries.
Poison messages	Log unprocessable messages to a dead-letter queue (or external system) after retries fail.

# Recommended Producer and Consumer Configurations



Parameter	Recommended Value	Description
Acks	All	Wait for acknowledgment from all brokers.
Retries	5 or higher	Number of retries on transient errors.
EnableIdempotence	true	Ensure exactly-once delivery by preventing duplicate messages.
RetryBackoffMs	1000 or higher	Backoff time between retries.

## Consumer Configuration

Parameter	Recommended Value	Description
EnableAutoCommit	false	Manually commit offsets to handle retries properly.
AutoOffsetReset	Earliest	Start reading from the beginning when no offset is committed.
MaxPollIntervalMs	Adjust based on workload	Maximum time between polls to prevent consumer rebalancing during processing.

# Transactional Messages

- Transactional messaging in Kafka ensures that messages are atomically produced to multiple topics or partitions and consumed exactly once.
- This is crucial when you have scenarios involving:
  - Chained processing where data needs to be produced/consumed consistently across multiple topics.
  - Distributed transactions requiring atomicity.
  - Exactly-once processing semantics where messages must not be duplicated or lost.

# Key Concepts

- Transactional Producer: Guarantees atomic writes of a group of messages to multiple Kafka partitions.
- Transaction ID: The producer uses this ID to identify its ongoing transaction and recover after failures.
- Exactly-once Semantics (EOS): The combination of transactional producers and idempotent writes guarantees that each message is processed exactly once.



# Confluent Kafka Flink Query

CONFLUENT

Home > Environments > default >

**Navigator** Workspaces

Catalogs located in gcp | us-east1

- examples (read-only)
- marketplace
- Tables
- default
- cluster\_0

**workspace-2025-02-10-211441**

PARAMESWARI E GCP | us-east1  
User Account GCP.us-east1.env-j9xv08.3da1 | Running

Use default ▾ Use cluster\_0 ▾

1 SELECT \* FROM `default`.`cluster\_0`.`patient-topic` LIMIT 10;

START TIME: 2025-02-10T21:14:54.490Z STATEMENT STATUS: Running STATEMENT NAME: workspace-2025-02-10...

Stop

# Confluent CLI under Cluster Settings (scroll down)

Home > Environments > default > cluster\_0 >

**Cluster**  
cluster\_0

Cluster Overview

- Networking
- API Keys
- Cluster Settings
- Stream Lineage
- Stream Designer
- Topics
- ksqlDB
- Connectors
- Clients

## CLI and tools

Confluent CLI   Confluent Platform Components   Kafka Connect

### Try it out!

Now that you have a cluster up and running in Confluent Cloud, you can administer using the [Confluent CLI](#).

#### 1. Install / Update the Confluent CLI

Run this command to install the Confluent CLI:

```
$ curl -sL --http1.1 https://cnfl.io/cli | sh -s -- latest
```

[Copy](#)

This script will install the CLI in `./bin` by default. If you want to install it somewhere else, add the path to the end of the command and to your \$PATH variable.

**Note:** On Windows, you might need to install an appropriate Linux environment to have the `curl` and `sh` commands available, such as the [Windows Subsystem for Linux](#). You can also download and install the [raw binaries](#).

If already installed, update to the latest version with:

# Confluent CLI using Ubuntu

- curl -sL --http1.1 https://cnfl.io/cli | sh -s -- latest

```
eswaribala@DESKTOP-B08BAAN:~ System information as of Tue Feb 11 02:58:04 IST 2025
System load: 1.95 Processes: 40
Usage of /: 0.2% of 1006.85GB Users logged in: 2
Memory usage: 12% IPv4 address for eth0: 172.19.49.37
Swap usage: 0%
* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s just raised the bar for easy, resilient and secure K8s cluster deployment.
https://ubuntu.com/engage/secure-kubernetes-at-the-edge
This message is shown once a day. To disable it please create the
/home/eswaribala/.hushlogin file.
eswaribala@DESKTOP-B08BAAN:~$ curl -sL --http1.1 https://cnfl.io/cli | sh -s -- latest
confluentinc/cli info checking S3 for tag 'latest'
confluentinc/cli info found version: latest for latest/linux/amd64
confluentinc/cli info NOTICE: see licenses located in /tmp/tmp.146a8weCJk/confluent
confluentinc/cli info installed ./bin/confluent
confluentinc/cli info please ensure ./bin is in your PATH
eswaribala@DESKTOP-B08BAAN:~$
```



# Confluent CLI using Ubuntu

```
eswaribala@DESKTOP-B08BAAN:~/bin  
eswaribala@DESKTOP-B08BAAN:~$ ls  
bin  bulkdata  filebeat.yml  perltraining  readme.txt  test.txt  training  trainingbackup.tar  trainingrestore  
eswaribala@DESKTOP-B08BAAN:~$ cd bin  
eswaribala@DESKTOP-B08BAAN:~/bin$ ls  
confluent  
eswaribala@DESKTOP-B08BAAN:~/bin$ █
```



# Confluent CLI using Windows

- <https://github.com/confluentinc/cli/releases/tag/v4.17.0>
- Download zip file and env variable

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.3037]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>confluent
^C
C:\Windows\System32>confluent login --save
Enter your Confluent Cloud credentials:
Email: parameswaribala@gmail.com
Logged in as "parameswaribala@gmail.com" for organization "454f3e6a-55c9-40e7-9259-f947fe31b0b8" ("VEB dental").

C:\Windows\System32>
```



# Confluent CLI using Windows

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.3037]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>confluent
^C
C:\Windows\System32>confluent login --save
Enter your Confluent Cloud credentials:
Email: parameswaribala@gmail.com
Logged in as "parameswaribala@gmail.com" for organization "454f3e6a-55c9-40e7-9259-f947fe31b0b8" ("VEB dental").

C:\Windows\System32>confluent environment use env-j9xv08
Using environment env-j9xv08.

C:\Windows\System32>confluent kafka cluster use lkc-61dgc6
Set Kafka cluster lkc-61dgc6 as the active cluster for environment env-j9xv08.
```



# Confluent CLI using Windows

```
Administrator: Command Prompt
-v, --verbose count    Increase verbosity (-v for warn, -vv for info, -vvv for debug, -vvvv for trace).

C:\Windows\System32>confluent api-key store RMINMSPHVA25JTUF WlkUBrN62WUyOXQri5TwZz1Z10lzp0/oLojeT9a1/0CKh74Ccm0pIwd5B
qizafrG
Stored secret for API key "RMINMSPHVA25JTUF".

C:\Windows\System32>
```

# Confluent CLI using Windows

Administrator: Command Prompt

```
C:\Windows\System32>confluent kafka topic create test-topic  
Created topic "test-topic".
```

```
C:\Windows\System32>confluent kafka topic list  
Name      | Internal | Replication Factor | Partition Count  
-----+-----+-----+-----  
patient-topic | false    |            3 |          6  
test-topic    | false    |            3 |          6  
vodaorder     | false    |            3 |          6
```

```
C:\Windows\System32>
```



## Confluent CLI using Windows

---

- 3. Select your environment
- Run this command to set your environment. To view the available environments, use the `confluent environment list` command.
- `confluent environment use env-j9xv08`



## Confluent CLI using Windows

---

- 4. Select your cluster
- Run this command to set your cluster. To view the available clusters, use the confluent kafka cluster list command.
- confluent kafka cluster use lkc-61dgz6

# Confluent CLI using Windows

```
Administrator: Command Prompt
stream-share          Manage stream shares.
update               Update the Confluent CLI.
version              Show version of the Confluent CLI.

Flags:
--version            Show version of the Confluent CLI.
-h, --help            Show help for this command.
--unsafe-trace       Equivalent to -vvvv, but also log HTTP requests and responses which might contain plaintext se
crets.
-v, --verbose count Increase verbosity (-v for warn, -vv for info, -vvv for debug, -vvvv for trace).

Use "confluent [command] --help" for more information about a command.
```

```
C:\Windows\System32>confluent login --save
Enter your Confluent Cloud credentials:
Email: parameswaribala@gmail.com
Logged in as "parameswaribala@gmail.com" for organization "454f3e6a-55c9-40e7-9259-f947fe31b0b8" ("VEB dental").
```

```
C:\Windows\System32>confluent environment list
Current | ID      | Name    | Stream Governance Package
-----+-----+-----+-----+
*     | env-j9xv08 | default |
```

```
C:\Windows\System32>confluent environment use env-j9xv08
Using environment env-j9xv08.
```

```
C:\Windows\System32>
```



# Confluent CLI using Windows

```
Administrator: Command Prompt
Logged in as "parameswaribala@gmail.com" for organization "454f3e6a-55c9-40e7-9259-f947fe31b0b8" ("VEB dental").

C:\Windows\System32>confluent environment list
Current | ID | Name | Stream Governance Package
-----+---+---+---+
* | env-j9xv08 | default | 

C:\Windows\System32>confluent environment use env-j9xv08
Using environment env-j9xv08.

C:\Windows\System32>confluent schema-registry cluster describe
+-----+-----+
| Name | Stream Governance Package |
| Cluster | lsrc-3m270m |
| Endpoint URL | https://psrc-wrp99.us-central1.gcp.confluent.cloud |
| Catalog Endpoint URL | https://psrc-wrp99.us-central1.gcp.confluent.cloud |
| Used Schemas | 3 |
| Available Schemas | 0 |
| Free Schemas Limit | 0 |
| Global Compatibility | BACKWARD |
| Mode | READWRITE |
| Cloud | GCP |
| Region | us-central1 |
| Package | ESSENTIALS |
+-----+-----+

C:\Windows\System32>
```

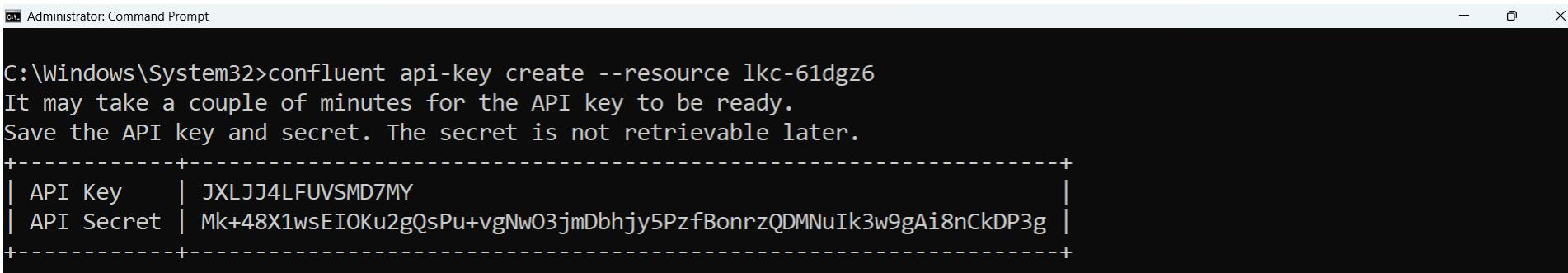
# Confluent CLI using Windows

```
confluent api-key create --resource lsrc-3m270m
```

```
C:\Windows\System32>confluent api-key create --resource lsrc-3m270m
It may take a couple of minutes for the API key to be ready.
Save the API key and secret. The secret is not retrievable later.
+-----+
| API Key      | 3BFMWQFZ50XHUDLQ
| API Secret   | 6D3IyPRg4kA2SmFJRR/CFqVkJ3fd6CMVAVvTS8w7EK33nvYLJ/Uc9MJIpWhuBDC
+-----+
C:\Windows\System32>
```

# Confluent CLI using Windows

```
confluent api-key create --resource lkc-61dgz6
```



A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the command "confluent api-key create --resource lkc-61dgz6" being run. The output indicates that it may take a couple of minutes for the API key to be ready and advises saving the API key and secret. It then displays a table with two rows: "API Key" and "API Secret", each with their respective values.

API Key	JXLJJ4LFUVSMD7MY
API Secret	Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBonrzQDMNuIk3w9gAi8nCkDP3g



# Confluent CLI using Windows

Cluster

cluster\_0

## Cluster Overview

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

⚠️ Here are more ways to get data moving through your cluster!

**Set up connector**  
Integrate your cluster to the most popular data sources within the Kafka ecosystem.

[Get started](#)



**Set up client**  
Write to Kafka in the programming language of your choice

[Get started](#)



**Produce sample data**  
Set up the DataGen Kafka Connector to produce sample events

[Get started](#)



## Tags

Add tags to this cluster

+ Add business metadata

## Cluster ID

lkc-61dqz6

## Cluster type

Standard

## Date created

Jan. 20 2025 5:21 PM

## Date modified

Feb. 11 2025 2:43 AM

## Cloud provider

GCP

## Cloud region





# Confluent CLI using Windows

```
Administrator: Command Prompt

C:\Windows\System32>confluent kafka topic create transactions-avro --cluster lkc-61dgz6
Created topic "transactions-avro".

C:\Windows\System32>
```

# AVRO Message

- **Apache Avro** is a data serialization framework that helps serialize structured data efficiently. When producing messages to Kafka, **Avro messages** provide:
  1. **Compact and efficient serialization:** Avro uses binary encoding, making messages small and efficient for transport.
  2. **Schema-based serialization:** Avro relies on schemas defined in **Avro IDL** or **JSON format** to ensure that data can be deserialized correctly by consumers.
  3. **Forward and backward compatibility:** Because schemas are versioned, producers and consumers can evolve independently, making schema evolution easy to manage.

# Schema Evolution and Compatibility

- One of Avro's strengths is its support for **schema evolution**. You can add or modify fields in the schema while maintaining compatibility between producers and consumers.
- **Types of Compatibility**
- **Backward compatibility:** Consumers using the old schema can still read messages produced with the new schema.
- **Forward compatibility:** Consumers using the new schema can read messages produced with the old schema.
- **Full compatibility:** Ensures both forward and backward compatibility.

# Schema Evolution and Compatibility

Initial schema:

```
json

{
  "type": "record",
  "name": "Transaction",
  "fields": [
    { "name": "id", "type": "string" },
    { "name": "amount", "type": "double" }
  ]
}
```

# Schema Evolution and Compatibility

New schema (adding a new optional field):

```
json

{
  "type": "record",
  "name": "Transaction",
  "fields": [
    { "name": "id", "type": "string" },
    { "name": "amount", "type": "double" },
    { "name": "description", "type": ["null", "string"], "default": null }
  ]
}
```

## Why Use Avro in Kafka?

- Compact binary format: Smaller messages reduce network overhead and storage costs.
- Schema evolution: Producers and consumers can have different schema versions without breaking compatibility.
- Schema registry integration: Confluent Kafka integrates Avro with its Schema Registry, where schemas are stored, validated, and version-controlled.

# Avro Message Workflow in Kafka

- 1. Define an Avro schema:** Define the structure of the message using the Avro schema language.
- 2. Serialize the message using the schema:** The producer serializes the message using the Avro schema before sending it to Kafka.
- 3. Store schema in Schema Registry:** The producer sends the schema to the **Confluent Schema Registry**, where it is assigned a schema ID.
- 4. Send the message to Kafka:** The producer sends the message (along with the schema ID) to the Kafka topic.
- 5. Consumer deserialization:** The consumer fetches the schema from the **Schema Registry** using the schema ID and deserializes the Avro message.

# Avro Message Structure

- An Avro message typically consists of:
  - 1. Schema ID:** Stored in the header of the Kafka message, this identifies the schema version used to serialize the message.
  - 2. Serialized message data:** Binary-encoded data representing the Avro message fields.

# Example: Avro Schema Definition

Schema for a `Transaction` message:

json

```
[  
  {"type": "record",  
   "name": "Transaction",  
   "namespace": "com.example",  
   "fields": [  
     { "name": "id", "type": "string" },  
     { "name": "amount", "type": "double" }  
   ]  
}
```

# Avro Serialization Flow

- **Avro Serialization Flow**

## 1. Producer Side:

1. When a producer sends an Avro message, the **AvroSerializer**:
  1. Serializes the message according to the schema.
  2. Stores the schema in the **Schema Registry** (if it's not already there).
  3. Adds the **schema ID** to the Kafka message headers.

## 2. Kafka Broker:

1. The Kafka broker simply stores the Avro message as binary data without being aware of its structure.

## 3. Consumer Side:

1. When a consumer reads an Avro message:
  1. It fetches the **schema ID** from the message header.
  2. Queries the **Schema Registry** to get the corresponding schema.
  3. Deserializes the binary Avro data back into the original object using the schema.

# Confluent CLI using Windows (consumer)

- confluent kafka topic produce transactions-avro --cluster lkc-61dgz6 --schema "F:\Local disk\ Docker\kafkahelm\helm\schema.txt" --schema-registry-endpoint "https://psrc-wrp99.us-central1.gcp.confluent.cloud" --schema-registry-api-key "3BFMWQFZ5OXHDLQ" --schema-registry-api-secret "6D3lyPRg4kA2SmFJRR/CFqVkJ3fd6CMVAVvTS8w7EK33nvYLJ/Uc9MJlptWhuBDC" --api-key "JXLJJ4LFUVSMD7MY" --api-secret "Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBo nrzQDMNulk3w9gAi8nCkDP3g" --value-format "avro"



# Confluent CLI using Windows (producer)

```
C:\Windows\System32>confluent api-key create --resource lkc-61dgz6
It may take a couple of minutes for the API key to be ready.
Save the API key and secret. The secret is not retrievable later.
+-----+
| API Key      | JXLJJ4LFUVSMD7MY |
| API Secret   | Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBonrzQDMNuIk3w9gAi8nCkDP3g |
+-----+
C:\Windows\System32>confluent kafka topic produce transactions-avro --cluster lkc-61dgz6 --schema "F:\Local disk\Docke
r\kafkahelm\helm\schema.txt" --schema-registry-endpoint "https://psrc-wrp99.us-central1.gcp.confluent.cloud" --schema-
registry-api-key "3BFMWFQFZ5OXHUDLQ" --schema-registry-api-secret "6D3IyPRg4kA2SmFJRR/CFqVkJFd6CMVAvTS8w7EK33nvYLJ/Uc
9MJIpWhuBDC" --api-key "JXLJJ4LFUVSMD7MY" --api-secret "Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBonrzQDMNuIk3w9gAi8nCkD
P3g" --value-format "avro"
Successfully registered schema with ID "100004".
%Starting Kafka Producer. Use Ctrl-C to exit.
6|1739287716.680|GETSUBSCRIPTIONS|Confluent-CLI_v4.17.0#producer-1| [thrd:main]: Telemetry client instance id changed
from AAAAAAAAAAAAAAAAAAAAAAA to YogWmqPATaygACKYwZtc2g
{ "id": "1000", "amount": 500 }
```

## Confluent CLI using Windows (consumer)

- ```
kafka topic consume transactions-avro --cluster lkc-61dgz6 --schema-registry-endpoint "https://psrc-wrp99.us-central1.gcp.confluent.cloud" --schema-registry-api-key "3BFMWQFZ5OXHUDLQ" --schema-registry-api-secret "6D3lyPRg4kA2SmFJRR/CFqVkf3fd6CMVAVvTS8w7EK33nvYLJ/Uc9MJlptWhuBDC" --api-key "JXLJJ4LFUVSMD7MY" --api-secret "Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBonrzQDMNulk3w9gAi8nCkDP3g"
```



# Confluent CLI using Windows (consumer)

```
Administrator: Command Prompt - confluent kafka topic consume transactions-avro --cluster lkc-61dgz6 --schema-registry-endpoint "https://psrc-wrp99.us-central1.gcp.confluent.cloud" --schema-registry-api-key "3BFWQFZ5OXHDLQ" --schema-registry-api-secret "6D3IyPRg4kA2SmFJRR/CFqVkJF3fd6CMVAvvTS8w7EK33nvYLJ/Uc9MJIptWhuBDC" --api-key "JXLJJ4LFUVSMD7MY" --api-secret "Mk+48X1wsEIOKu2gQsPu+vgNwO3jmDbhjy5PzfBonrzQDMNuIk3w9gAi8nCkDP3g"  
%6|1739288501.554|GETSUBSCRIPTIONS|Confluent-CLI_v4.17.0#consumer-1| [thrd:main]: Telemetry client instance id changed  
from AAAAAAAAAAAAAAAA to MuwqCr0zR/ezMjyZzGyPJQ  
Starting Kafka Consumer. Use Ctrl-C to exit.  
1001p@
```



# Confluent CLI using Windows (consumer)

Cluster Overview

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

Overview    **Messages**    Data contracts New    Configuration

Production in last hour    Consumption in last hour    Total messages    Retention time

2 messages    1 messages    2    1 week

Filter by timestamp, offset, key or value    All partitions    Latest    Max 50 results

2 messages shown    Auto-refresh on    CSV    JSON

| Timestamp     | Offset | Partition | Key | Value                          |
|---------------|--------|-----------|-----|--------------------------------|
| 1739288527638 | 1      | 5         | ""  | {"id": "1001", "amount": 1500} |
| 1739287813234 | 0      | 5         | ""  | {"id": "1000", "amount": 500}  |

## AVRO c#

---

- dotnet tool install Chr.Avro.Cli –global
- dotnet avro generate < transaction.avsc >  
Transaction.cs



# AVRO c#

```
Administrator: Developer Command Prompt for VS 2022
F:\Local disk\ Docker\kafkahelm\helm>dotnet avro generate < transaction.avsc > Transaction.cs

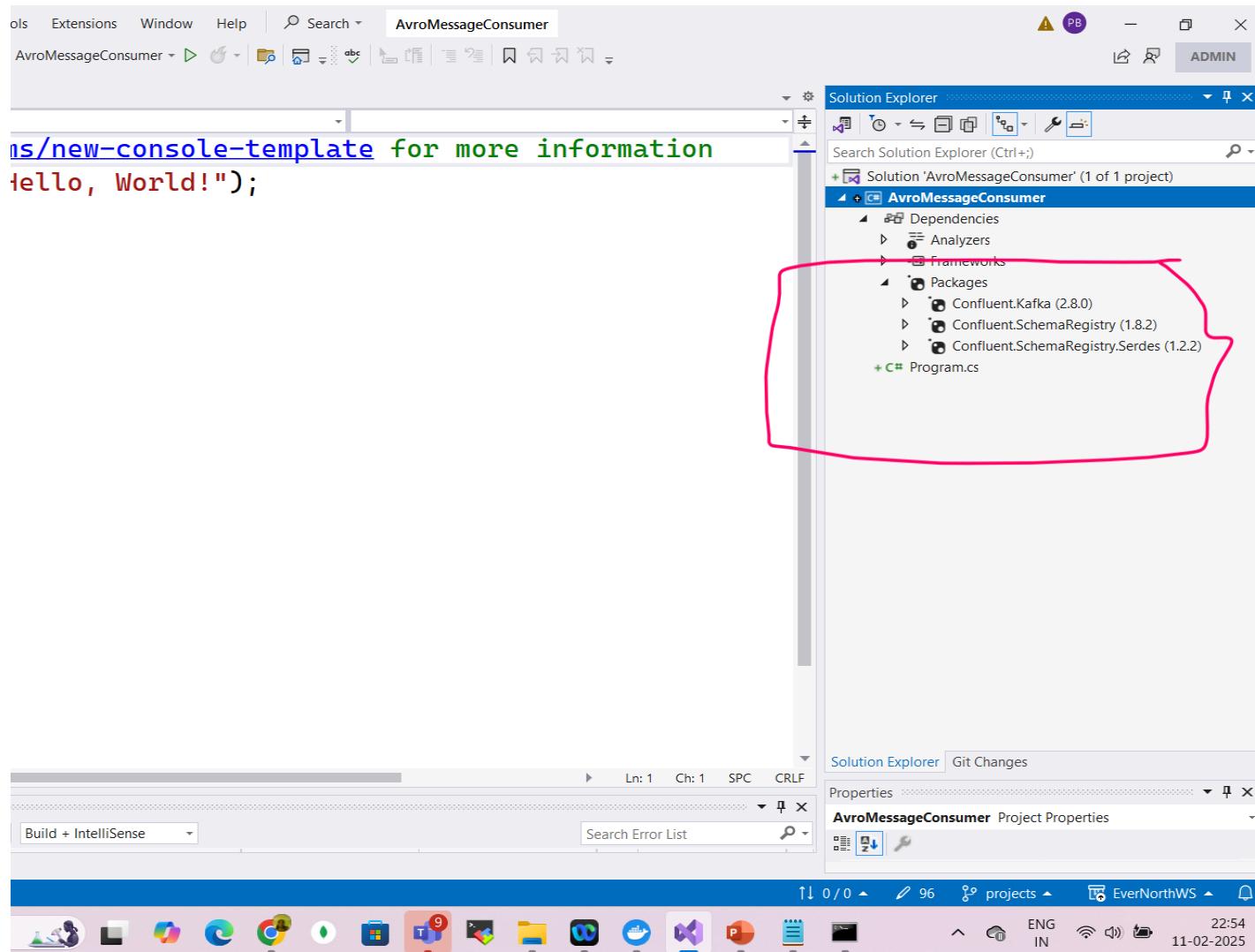
F:\Local disk\ Docker\kafkahelm\helm>dir
Volume in drive F is New Volume
Volume Serial Number is 8E55-7759

Directory of F:\Local disk\ Docker\kafkahelm\helm

11-02-2025  22:12      <DIR>          .
11-02-2025  03:34      <DIR>          ..
11-02-2025  09:09            160 api-key-JHTTEFVOEV60ZT5T.txt
09-02-2025  18:08            222 api-key-RMINMSPHVA25JTUF.txt
11-02-2025  21:27        651,391 avro-1.11.4.jar
11-02-2025  21:32        57,470,557 avro-tools-1.11.0.jar
11-02-2025  21:33        54,810,036 avro-tools-1.11.1.jar
11-02-2025  21:58            2,533 avroschemaregistry.txt
10-02-2025  21:18            2,039 helminstructions.txt
11-02-2025  18:18            716 javaconfig.txt
11-02-2025  19:41            428 kafkaconduktor.txt
11-02-2025  20:48            163 schema.txt
11-02-2025  21:29            163 transaction.avsc
11-02-2025  22:16            108 Transaction.cs
10-02-2025  11:16            688 values.yml
                           13 File(s)   112,939,204 bytes
                           2 Dir(s)  23,157,321,728 bytes free

F:\Local disk\ Docker\kafkahelm\helm>
```

# AVRO c#



```
ns/new-console-template for more information
Hello, World!);
```

Solution Explorer

- + Solution 'AvroMessageConsumer' (1 of 1 project)
  - AvroMessageConsumer
    - Dependencies
    - Analyzers
    - Frameworks
    - Packages
      - Confluent.Kafka (2.8.0)
      - Confluent.SchemaRegistry (1.8.2)
      - Confluent.SchemaRegistry.Serdes (1.2.2)
  - + C# Program.cs

Properties

AvroMessageConsumer Project Properties

Build + IntelliSense

Search Error List

LN: 1 Ch: 1 SPC CRLF

11-02-2025 22:54 ENG IN

# ProtoBuf

- **Protocol Buffers (ProtoBuf)** is a **language-neutral, platform-neutral** mechanism for serializing structured data.
- Developed by **Google**, it's widely used for **data serialization**, offering compact and efficient encoding of data compared to formats like **JSON** or **XML**.
- **Compact**: Smaller message sizes due to binary encoding.
- **Fast**: Faster serialization/deserialization.
- **Schema evolution-friendly**: Designed to handle changes to message schemas over time.

# Why Use ProtoBuf Instead of JSON or XML?

| Feature          | ProtoBuf                                        | JSON                               | XML                                |
|------------------|-------------------------------------------------|------------------------------------|------------------------------------|
| Efficiency       | Compact binary format (smaller size)            | Human-readable text format         | Human-readable text format         |
| Speed            | Faster serialization/deserialization            | Slower compared to ProtoBuf        | Slowest among the three            |
| Schema           | Strongly typed using <code>.proto</code> schema | No formal schema enforcement       | Optional schema (DTD/XSD)          |
| Schema Evolution | Handles forward and backward compatibility      | Schema changes require extra logic | Schema changes require extra logic |

# How ProtoBuf Works

- Define a schema using the .proto file format.
- Compile the schema using the ProtoBuf compiler (protoc) to generate code for your programming language.
- Use the generated code to serialize and deserialize messages.

## Use Cases of ProtoBuf

- Inter-service communication (gRPC): ProtoBuf is the underlying serialization mechanism for gRPC.
- Efficient storage: Storing large amounts of structured data efficiently.
- Configuration files: Machine-readable configuration files for high-performance applications.

# Create ProtoBuf Schema

Cluster

cluster\_0

Value

Key

Compatibility mode: Backward ↗ Associated with topics: --

Cluster Overview

Networking

API Keys

Cluster Settings

Stream Lineage

Stream Designer

Topics

ksqlDB

Connectors

Clients

Version: 1 (current) ▾ Schema ID: 100006 Type: Protobuf

Evolve

⋮

Schema

References

Metadata

Rules

&lt;/&gt;

Edit

```
1 syntax = "proto3";
2 package com.verb.kafkaproto;
3
4 message User {
5     int32 age = 1;
6     string name = 2;
7 }
8
```



# Install Protoc compiler

```
C:\ Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.3037]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>winget install protobuf
The `msstore` source requires that you view the following agreements before using.
Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction
The source requires the current machine's 2-letter geographic region to be sent to the backend service to function properly (ex. "US").

Do you agree to all the source agreements terms?
[Y] Yes [N] No: y
Found protobuf [Google.Protobuf] Version 29.1
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://github.com/protocolbuffers/protobuf/releases/download/v29.1/protoc-29.1-win64.zip
[██████████] 3.04 MB / 3.04 MB
Successfully verified installer hash
Extracting archive...
Successfully extracted archive
Starting package install...
Command line alias added: "protoc"
Path environment variable modified; restart your shell to use the new value.
Successfully installed

C:\Windows\System32>
```





# Generate c# class from proto file

```
F:\Local disk\Disk\kafkahelm\helm>protoc --csharp_out=. user.proto

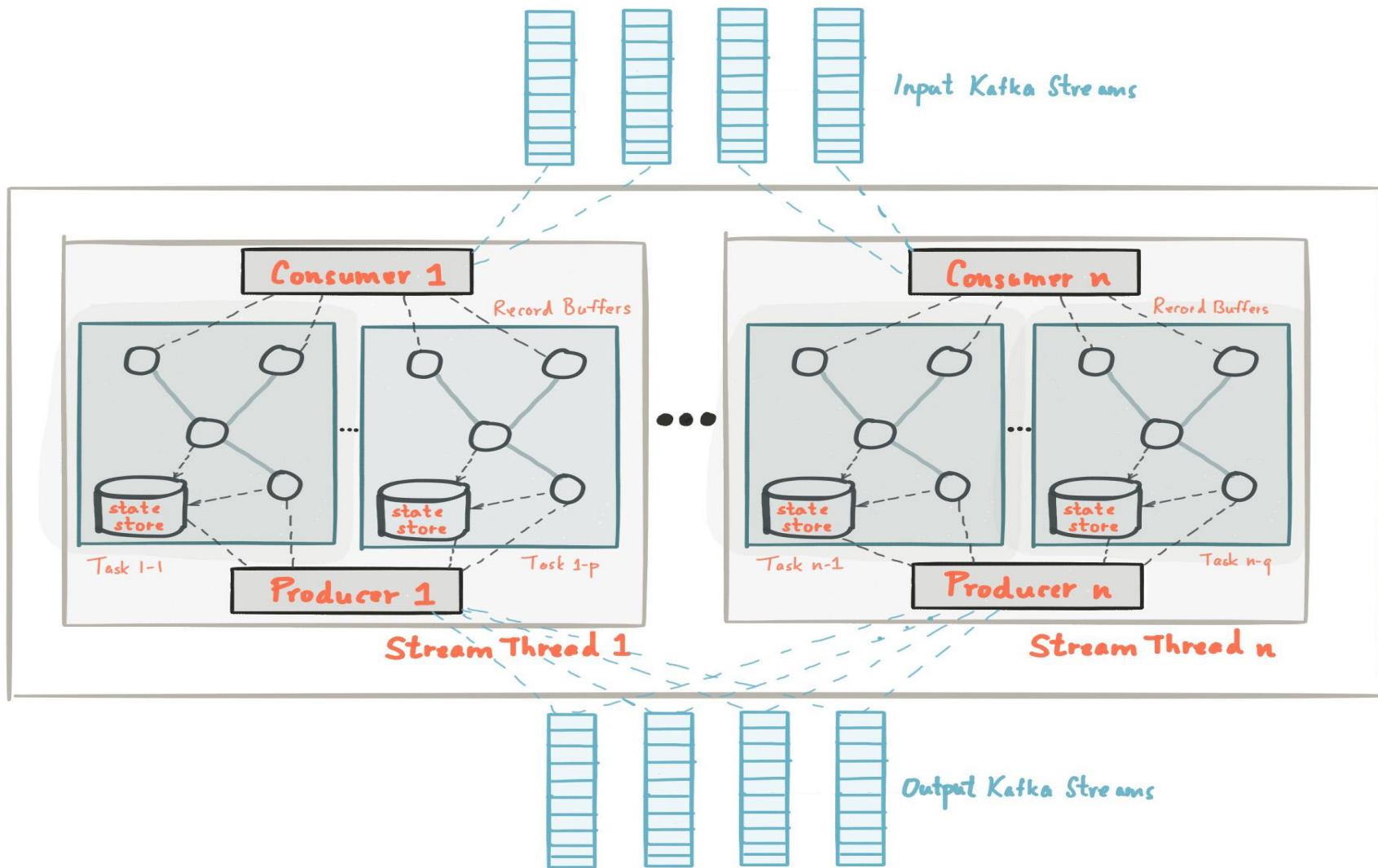
F:\Local disk\Disk\kafkahelm\helm>dir
Volume in drive F is New Volume
Volume Serial Number is 8E55-7759

Directory of F:\Local disk\Disk\kafkahelm\helm

12-02-2025  00:30    <DIR>          .
11-02-2025  03:34    <DIR>          ..
11-02-2025  09:09            160 api-key-JHTTEFVOEV60ZT5T.txt
09-02-2025  18:08            222 api-key-RMINMSPHVA25JTUF.txt
11-02-2025  21:27        651,391 avro-1.11.4.jar
11-02-2025  21:32      57,470,557 avro-tools-1.11.0.jar
11-02-2025  21:33      54,810,036 avro-tools-1.11.1.jar
11-02-2025  22:17        2,658 avroschemaregistry.txt
10-02-2025  21:18        2,039 helminstructions.txt
11-02-2025  18:18          716 javaconfig.txt
11-02-2025  19:41        428 kafkaconduktor.txt
11-02-2025  20:48          163 schema.txt
11-02-2025  21:29        163 transaction.avsc
11-02-2025  22:16          108 Transaction.cs
12-02-2025  00:30        9,095 User.cs
12-02-2025  00:20          101 user.proto
10-02-2025  11:16        688 values.yml
               15 File(s)   112,948,525 bytes

F:\Local disk\Disk\kafkahelm\helm>
```

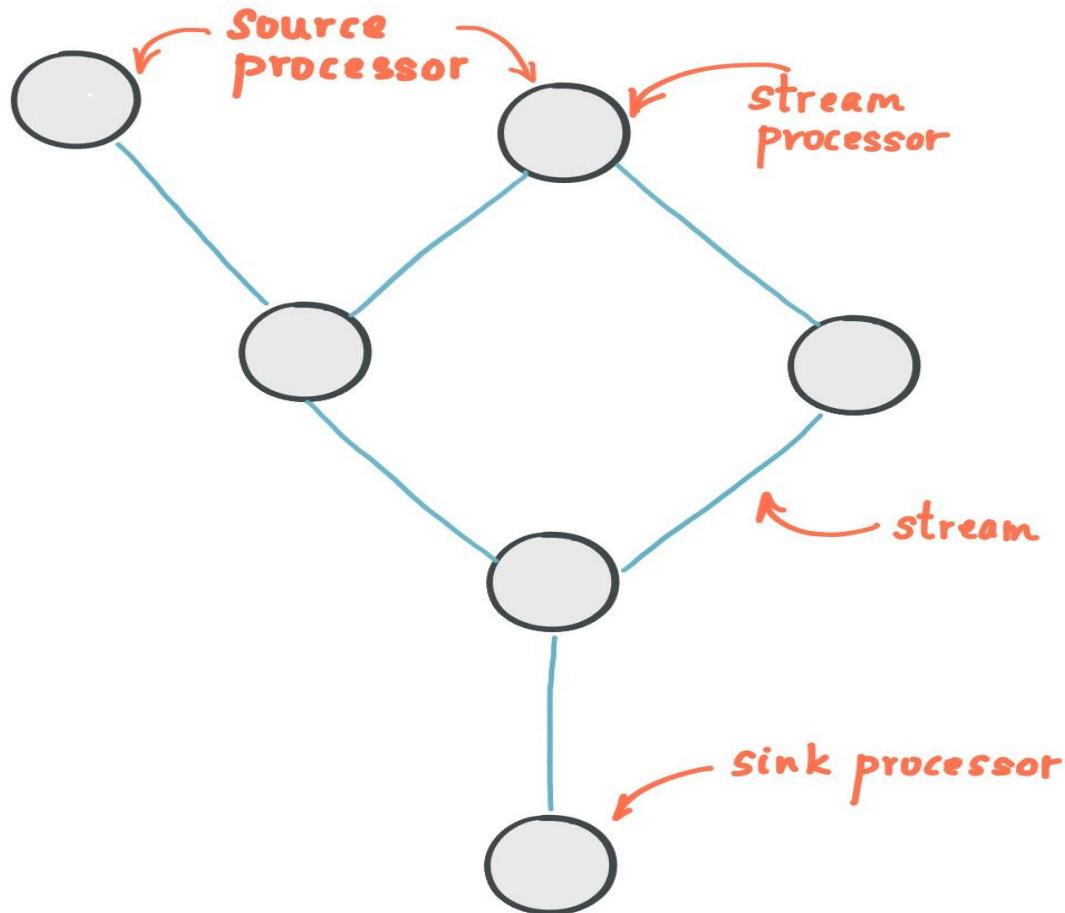
# Key Concepts of Kafka Streams



# Key Concepts of Kafka Streams

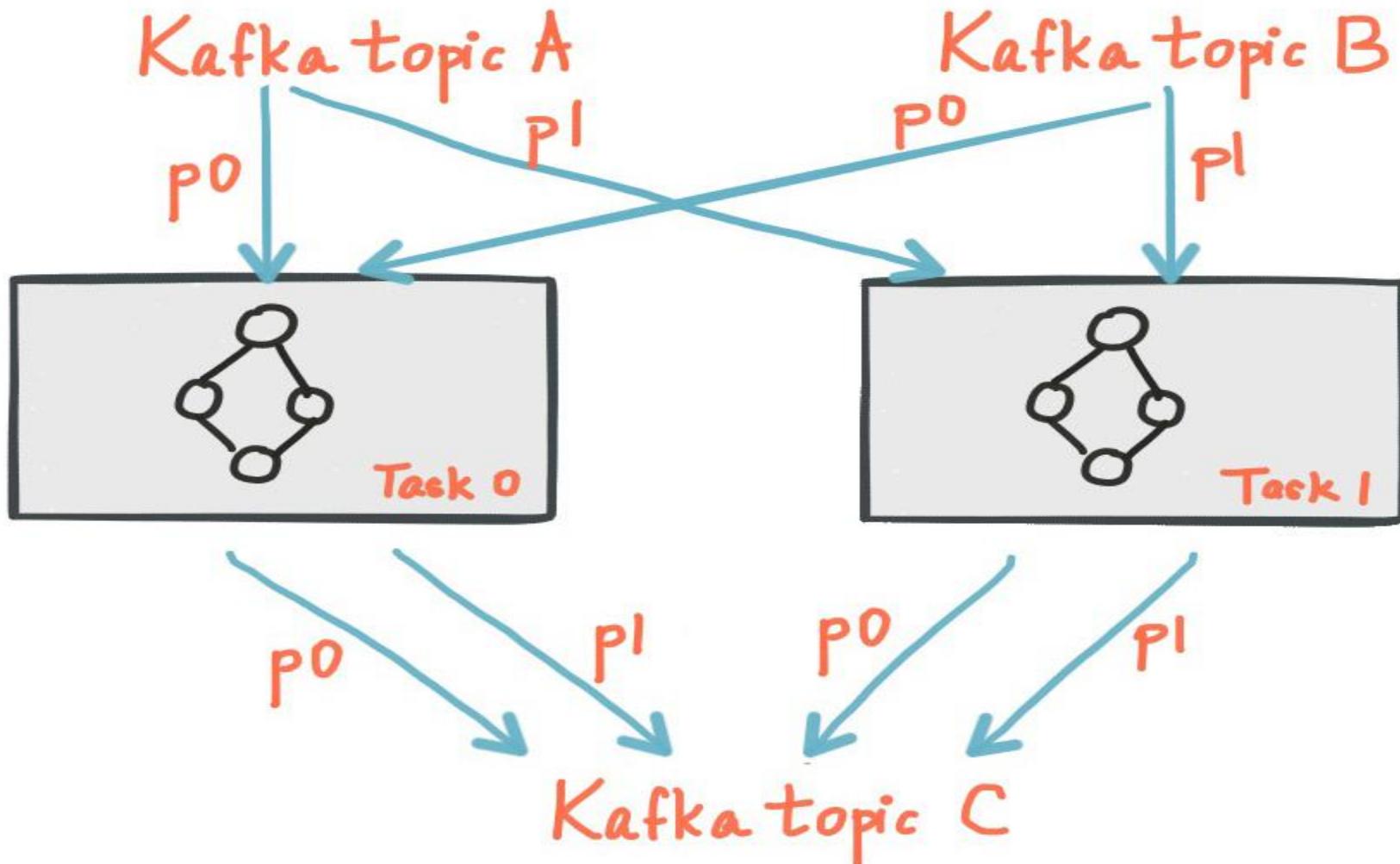
1. **Stream**: A continuous flow of data records.
2. **Topic**: Kafka stores streams in topics.
3. **KStream & KTable**: Core abstractions for processing records.
  1. **KStream**: Represents a stream of records (immutable, append-only).
  2. **KTable**: Represents a changelog stream (stateful, latest value per key).
4. **Stateful Processing**: Supports operations like windowing, aggregations, and joins.
5. **Topology**: Defines how streams are processed in a DAG (Directed Acyclic Graph).
6. **Processor API**: Allows custom processing logic beyond the DSL.

# Processor topology



PROCESSOR TOPOLOGY

# Processor topology



# Stream Processing Model

- Kafka Streams operates on Kafka topics and continuously processes data as it arrives.
- The processing model follows the **event-driven** and **distributed computing** paradigms.
- **Record (Event)**: The fundamental unit of data in Kafka Streams.
- **Streams**: A continuous flow of records.
- **Tables (Stateful Data)**: A key-value abstraction for tracking state over time.
- **Topology**: A directed graph that defines transformations.

# Key Abstractions

| Abstraction   | Description                                                       |
|---------------|-------------------------------------------------------------------|
| KStream       | Represents a stream of records (immutable, append-only)           |
| KTable        | Represents stateful data (latest value per key)                   |
| GlobalKTable  | Similar to KTable but globally available on all instances         |
| State Stores  | Persist intermediate results for aggregations, joins, and queries |
| Processor API | Low-level API for custom transformations                          |

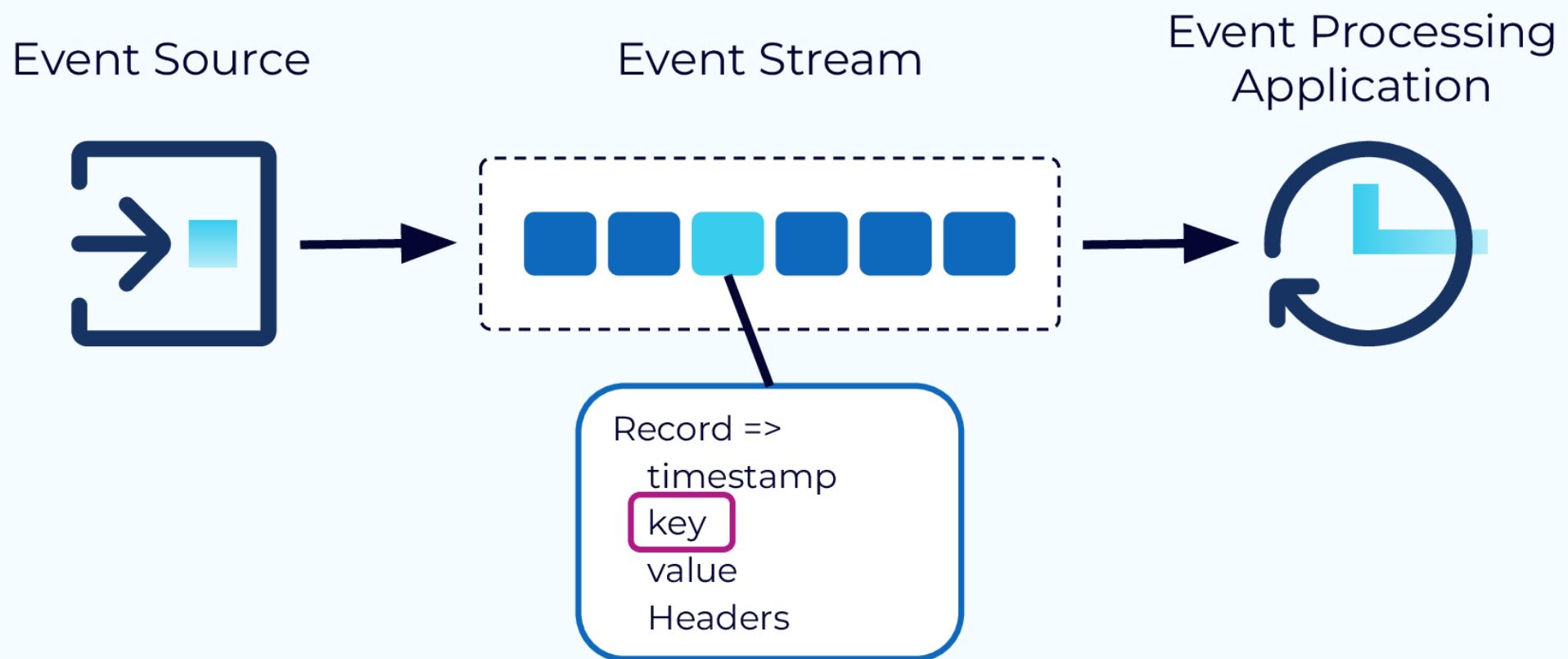
# Stream Processing Operations

- Kafka Streams provides various transformation operations:
- **2.1 Stateless Transformations**
- **map()** – Transforms key/value pairs.
- **filter()** – Filters records based on conditions.
- **flatMap()** – Produces multiple output records from one input record.
- **peek()** – Performs a side effect (e.g., logging) without modifying the stream.

# Stream Processing Operations

- **2.2 Stateful Transformations**
- Stateful transformations require maintaining intermediate results, often stored in **RocksDB-backed state stores**.
- **groupBy()** – Reorganizes data for aggregation.
- **count()** – Counts occurrences per key.
- **reduce()** – Combines records with a specified logic.
- **aggregate()** – Builds complex aggregations.

# Event Processing



# Event Stream

Record =>  
 timestamp  
key  
value  
 Headers

| key/<br>value<br>Bytes | Area       | Description                                                                |
|------------------------|------------|----------------------------------------------------------------------------|
| 0                      | Magic Byte | Confluent serialization format version number; currently always <b>0</b> . |
| 1-4                    | Schema ID  | 4-byte schema ID as returned by Schema Registry.                           |
| 5...                   | Data       | Serialized data for the specified schema format.                           |



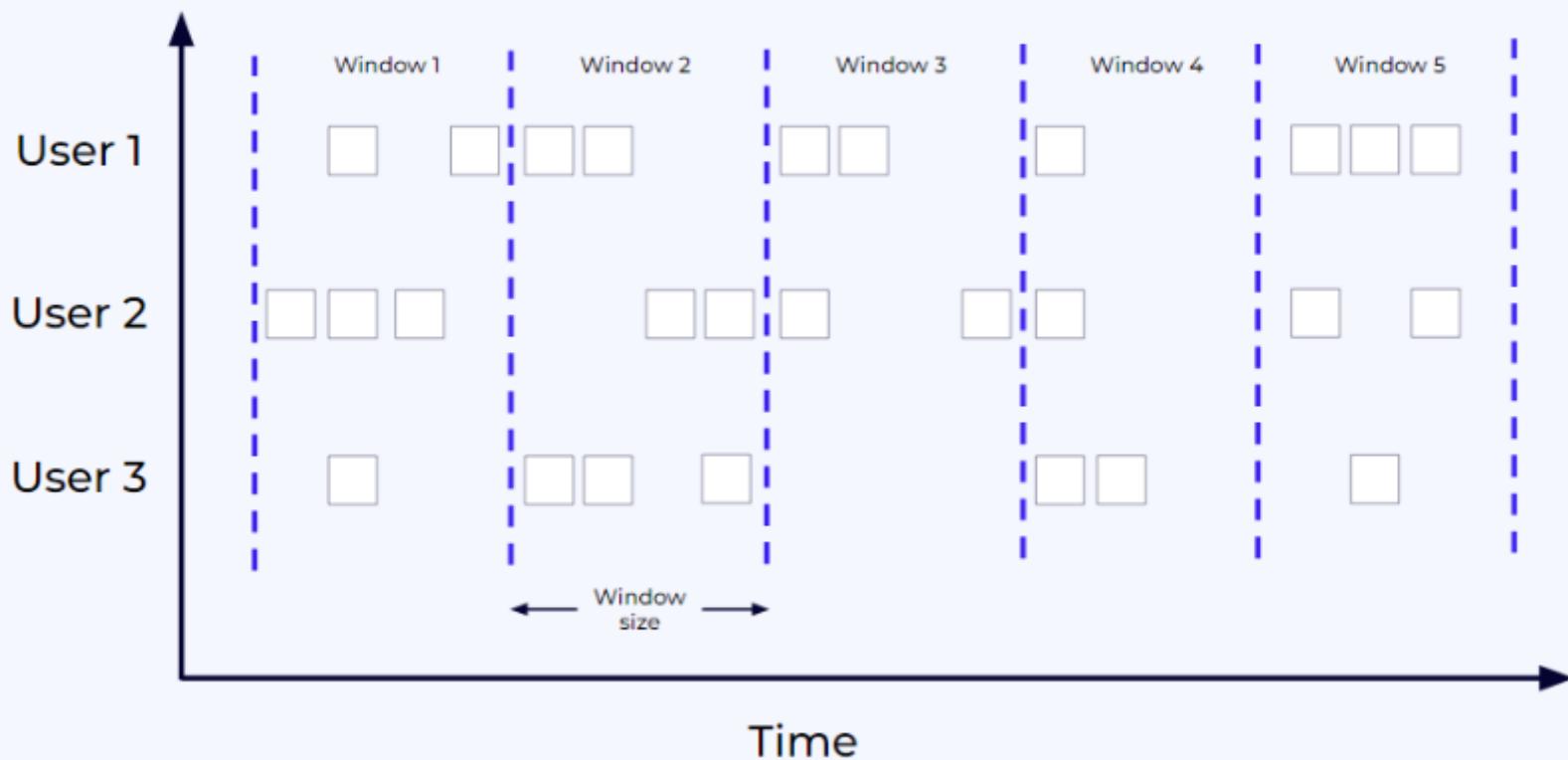
Event Stream

# Windowing

- Windowing is used for time-based aggregations in Kafka Streams. Windows define **how long** records should be grouped together.
- **Types of Windows**
  1. **Tumbling Window:** Fixed-size, non-overlapping time windows.
  2. **Hopping Window:** Overlapping windows (fixed interval + hop size).
  3. **Sliding Window:** Overlapping windows where each event can belong to multiple windows.
  4. **Session Window:** Based on user activity (closes when no new events appear).

# Tumbling Windows

## Tumble Window Function

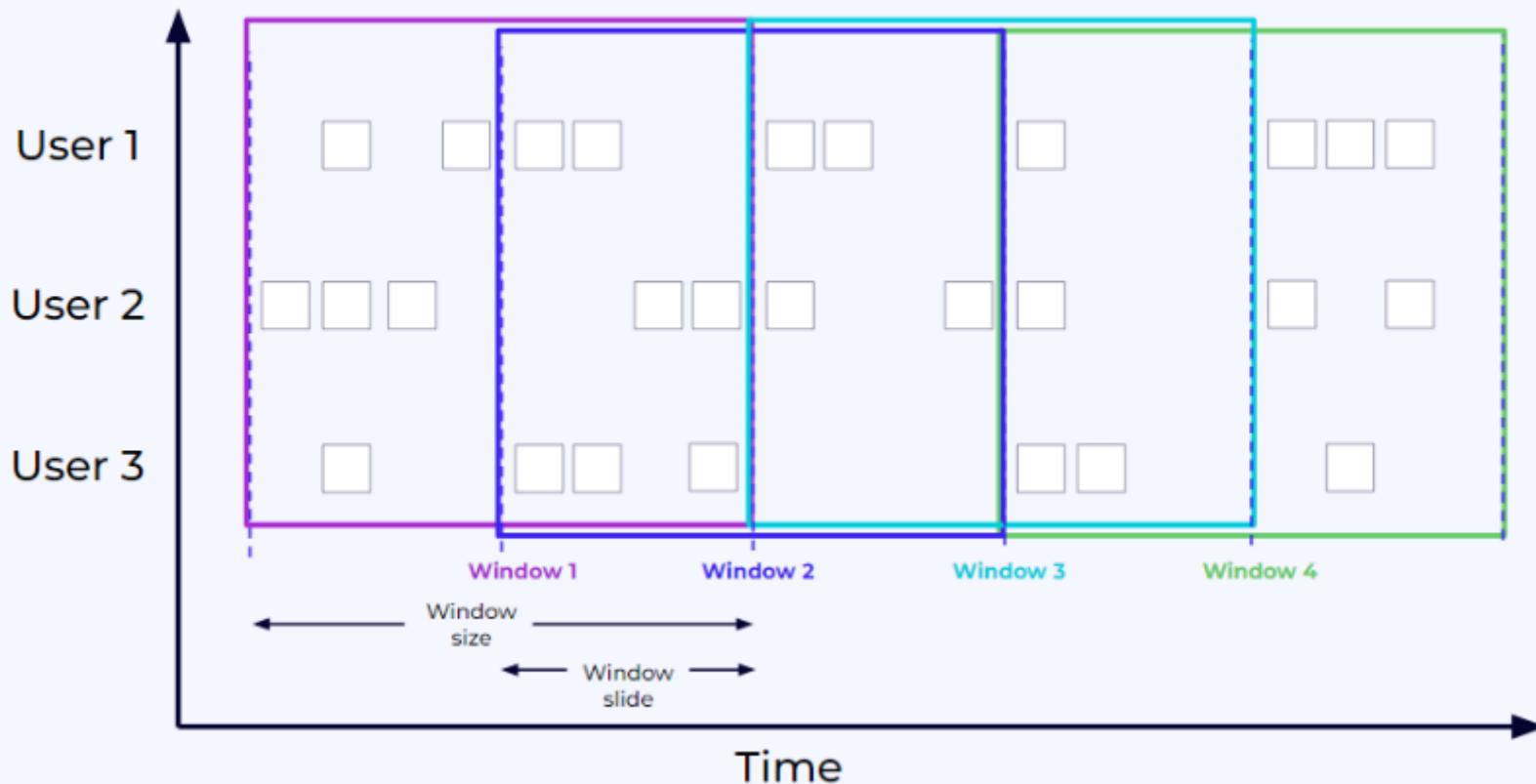


# Tumbling Windows

- `SELECT * FROM TABLE(`
- `TUMBLE(TABLE `examples`.`marketplace`.`orders`, DESCRIPTOR($rowtime), INTERVAL '10' MINUTES)`
- -- or with the named params
- -- note: the DATA param must be the first
- `SELECT * FROM TABLE(`
- `TUMBLE(`
- `DATA => TABLE `examples`.`marketplace`.`orders`,`
- `TIMECOL => DESCRIPTOR($rowtime),`
- `SIZE => INTERVAL '10' MINUTES));`

# Hopping Windows

Hop Window Function



# Hopping Windows

- `SELECT * FROM TABLE(`
- `HOP(TABLE `examples`.`marketplace`.`orders`, DESCRIPTOR($rowtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES)`
- -- or with the named params
- -- note: the DATA param must be the first
- `SELECT * FROM TABLE(`
- `HOP(`
- `DATA => TABLE `examples`.`marketplace`.`orders`,`
- `TIMECOL => DESCRIPTOR($rowtime),`
- `SLIDE => INTERVAL '5' MINUTES,`
- `SIZE => INTERVAL '10' MINUTES));`

# Session Windows

- `SELECT * FROM TABLE(`
- `SESSION(TABLE `examples`.`marketplace`.`orders` PARTITION BY product_id, DESCRIPTOR($rowtime), INTERVAL '1' MINUTES);`
- -- or with the named params
- -- note: the DATA param must be the first
- `SELECT * FROM TABLE(`
- `SESSION(`
- `DATA => TABLE `examples`.`marketplace`.`orders` PARTITION BY product_id,`
- `TIMECOL => DESCRIPTOR($rowtime),`
- `GAP => INTERVAL '1' MINUTES));`

# Windowing

Hopefully, this clarifies the concepts of windowing in Kafka Streams. Below

| Feature/Type        | Tumbling Window                                                                                         | Hopping Window                                                                                                                     | Session Window                                                                                                        | Sliding Window                                                                                                                            |
|---------------------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Definition          | A type of window that has a fixed size and does not overlap. Each record belongs to exactly one window. | Similar to tumbling windows but can overlap. It is defined by two parameters: the window's size and its advance interval (or hop). | Dynamic windows that group events by activity sessions.<br>Windows are defined by periods of activity and inactivity. | A type of window that allows for overlapping windows where each window is defined by a fixed duration and can slide over the data stream. |
| Window Size         | Fixed                                                                                                   | Fixed                                                                                                                              | Dynamic, determined by inactivity gap                                                                                 | Fixed                                                                                                                                     |
| Overlap             | No                                                                                                      | Yes, if the advance interval is smaller than the window size                                                                       | No, but windows are not fixed and can vary in length                                                                  | Yes, windows can overlap due to the sliding nature                                                                                        |
| Use Cases           | Aggregations over a uniform time interval, such as hourly metrics.                                      | Aggregations where more frequent updates are needed within a fixed period, allowing overlap.                                       | Aggregations where the activity is bursty or irregular, such as user sessions in web analytics.                       | Fine-grained aggregations where updates are needed for every event or at short intervals, capturing trends over time.                     |
| Key Characteristics | Simple and non-overlapping, making it suitable for clear-cut time intervals.                            | Provides flexibility with overlapping windows, useful for more nuanced time-based aggregations.                                    | Ideal for handling irregular or event-driven data, adapting to the data's inherent patterns.                          | Offers the most granular control over windowing, allowing for continuous updates within overlapping periods.                              |

# Joins

## 4.1 Types of Joins

| Join Type            | Description                                       |
|----------------------|---------------------------------------------------|
| KStream-KStream Join | Joins two KStreams on a key within a time window. |
| KStream-KTable Join  | Enriches a stream with data from a table.         |
| KTable-KTable Join   | Merges two tables on matching keys.               |



## Confluent Kafka using Local

- Refer F:\Local disk\ Docker\kafkahelm\docker
- <https://developer.confluent.io/tutorials/change-topic-partitions-replicas/ksql.html>

# Scaling Kafka Streams

- Kafka Streams applications scale **horizontally** across multiple instances.
- Each **Kafka Streams instance** runs multiple **tasks**.
- Each task processes a subset of **Kafka partitions**.
- More instances = More parallelism.
- To scale, just deploy multiple instances of the application.

# Kafka Streams vs Other Stream Processing Frameworks



| Feature             | Kafka Streams                 | Apache Flink         | Apache Spark Streaming |
|---------------------|-------------------------------|----------------------|------------------------|
| Processing Model    | Event-driven                  | Batch + Event-driven | Micro-batch            |
| Stateful Processing | Yes (RocksDB)                 | Yes                  | Yes                    |
| Scalability         | High (Parallel per partition) | High                 | Moderate               |
| Deployment          | Embedded (microservices)      | Clustered            | Clustered              |
| Latency             | Low (ms-level)                | Low (ms-level)       | High (sec-level)       |

# Kafka Streaming vs Traditional Messaging (Message Queues)



| Feature          | Kafka Streaming                          | Traditional Messaging (MQ)                          |
|------------------|------------------------------------------|-----------------------------------------------------|
| Processing Model | Continuous stream processing (real-time) | Message-based (discrete events)                     |
| Data Flow        | Continuous & event-driven                | Request-response, point-to-point                    |
| Message Storage  | Persistent log-based storage (Kafka)     | Queue-based, messages are removed after consumption |
| Scaling          | Distributed, parallel processing         | Limited to broker-defined consumers                 |
| State Management | Supports stateful processing via RocksDB | Stateless (or requires an external DB)              |
| Fault Tolerance  | Built-in state recovery and replay       | Message loss possible if not handled properly       |
| Replayability    | Supports replaying past messages         | Messages are removed after consumption              |
| Latency          | Low latency (ms-level)                   | Moderate latency (depends on broker performance)    |



# Key Considerations for Choosing

| Criteria            | Kafka Streaming                   | Traditional Message Queue          |
|---------------------|-----------------------------------|------------------------------------|
| Use Case            | Real-time stream processing       | Point-to-point messaging           |
| Data Persistence    | Stored in Kafka for replayability | Typically removed after processing |
| Ordering Guarantees | Partition-level ordering          | FIFO (depending on broker)         |
| Throughput          | High                              | Moderate                           |
| Scaling             | Horizontally scalable             | Limited scaling based on queues    |

# Kafka elastic Connector

-  Cluster
-  **cluster\_0**
-  Cluster Overview
-  Networking
-  API Keys
-  Cluster Settings
-  Stream Lineage
-  Stream Designer
-  Topics
-  ksqldb
-  **Connectors**
-  Clients

## Connectors

Search by connector or plugin name

 **ElasticsearchSinkConnector\_0**  
Running

| Tasks        | Bytes/sec       |
|--------------|-----------------|
| 1            | --              |
| Messages/sec | Messages behind |
| --           | --              |

### Overview

| Category    | Sink                    |
|-------------|-------------------------|
| ID          | lcc-r1vmpk              |
| Plugin name | Elasticsearch Servic... |



# Kafka Connector sends data to elastic index

Insert title here Empire New Tab How to use Assertio... Browser Automatio... node.js - How can I... Freelancer-dev-810... Courses New Tab Airtel 4G Hotspot

| health | status | index                                                              | uuid                   | pri | rep | docs.count | docs.deleted | store.size | pri.store.size | dataset.size |
|--------|--------|--------------------------------------------------------------------|------------------------|-----|-----|------------|--------------|------------|----------------|--------------|
| green  | open   | .internal.alerts-transform.health.alerts-default-000001            | fbk0rj8bR0Kjo1iBCe6J3A | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-observability.logs.alerts-default-000001          | xwu1_nuLRzmlv19VvMTBCQ | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .ds-logs-enterprise_search.audit-default-2025.02.12-000001         | -tuXVmXkRC52kviiGm1Sg  | 1   | 1   | 9          | 0            | 159.4kb    | 79.6kb         | 79.6kb       |
| green  | open   | .ds-logs-enterprise_search.api-default-2025.02.12-000001           | PSI3DEe5TVGEud760eAo9Q | 1   | 1   | 2          | 0            | 62.8kb     | 31.4kb         | 31.4kb       |
| green  | open   | .internal.alerts-observability.uptime.alerts-default-000001        | fbhmjnC3RtSnzKxp3a-3pw | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-ml.anomaly-detection.alerts-default-000001        | j13byq0lRCe7_6v0JY_HSw | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-observability.slo.alerts-default-000001           | 75VABm0WR3GVCCxtBbKTiw | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .ds-metrics-fleet_server.agent.status-default-2025.02.12-000001    | LznAEXkaSw6jbKT6yPAwCw | 1   | 1   | 63         | 0            | 857.1kb    | 428.5kb        | 428.5kb      |
| green  | open   | .internal.alerts-default.alerts-default-000001                     | boo0z9RTQeTjzmpOpxTzw  | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-observability.apm.alerts-default-000001           | nZXXAa7VRq0Wt-1AbrZDQw | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .elastic-connectors-v1                                             | pX2Q6RMDsSK56RuGylPGg  | 1   | 1   | 0          | 0            | 3.8kb      | 247b           | 247b         |
| green  | open   | .internal.alerts-observability.metrics.alerts-default-000001       | LdM-gqdrRmqgFEQ0giWGg  | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | ordersv2                                                           | EVfwxB7zRM-A770MSX-ItQ | 1   | 1   | 6          | 0            | 24.5kb     | 12.2kb         | 12.2kb       |
| green  | open   | .ds-metrics-fleet_server.agent.versions-default-2025.02.12-000001  | eX4hR9WURa6zwjMLgcWYSg | 1   | 1   | 63         | 0            | 604.6kb    | 302.3kb        | 302.3kb      |
| green  | open   | .internal.alerts_ml.anomaly-detection-health.alerts-default-000001 | a659hX6RQLKbfLNOJyfrw  | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-observability.threshold.alerts-default-000001     | z6Z0m-kTS1qCtJefoMWGBw | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .elastic-connectors-sync-jobs-v1                                   | fjMmjY00SECbARBQEe-xrA | 1   | 1   | 0          | 0            | 3.8kb      | 247b           | 247b         |
| green  | open   | .internal.alerts-security.alerts-default-000001                    | V5DyZT4YQt2VIg2Wcv0jg  | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |
| green  | open   | .internal.alerts-stack.alerts-default-000001                       | Tgis9mUPTLyynBaJUn6s7Q | 1   | 1   | 0          | 0            | 498b       | 249b           | 249b         |

# Kafka Connector elastic keeps in dlq (error data)

Cluster  
cluster\_0

## dlq-lcc-r1vmpk

[Query with Flink](#) [Actions](#)

Overview [Messages](#) Data contracts [New](#) Configuration

|                                        |                                         |                      |                          |
|----------------------------------------|-----------------------------------------|----------------------|--------------------------|
| Production in last hour<br>88 messages | Consumption in last hour<br>-- messages | Total messages<br>88 | Retention time<br>1 week |
|----------------------------------------|-----------------------------------------|----------------------|--------------------------|

Filter by timestamp, offset, key or value All partitions Latest Max 50 results

50 messages shown  Auto-refresh on [CSV](#) [JSON](#)

| Timestamp     | Offset | Partition | Key                | Value                      |
|---------------|--------|-----------|--------------------|----------------------------|
| 1739371522965 | 55     | 0         | "c7c40bd9-0b2a-... | "Public Health Department" |
| 1739371522965 | 78     | 0         | "c7c40bd9-0b2a-... | "Public Health Department" |



# Confluent Kafka using Local

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.3037]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>docker container ls -a
CONTAINER ID   IMAGE                               COMMAND                  CREATED        STATUS
PORTS
20ed9f747e0f   confluentinc/cp-ksqldb-cli:6.2.15   "/bin/sh"               2 minutes ago  Up 2 minute
s
3b1d98ac5655   confluentinc/ksqldb-examples:6.2.15  "bash -c 'echo Waiti..."  2 minutes ago  Exited (1)
53 seconds ago
90538056ad2f   confluentinc/cp-enterprise-control-center:6.2.15  "0.0.0.0:9021->9021/tcp"  2 minutes ago  Up 2 minute
s
1e67f164dc46   confluentinc/cp-ksqldb-server:6.2.15    "0.0.0.0:8088->8088/tcp"  2 minutes ago  Up 2 minute
s
16b67a789ba2   confluentinc/cp-kafka-rest:6.2.15     "0.0.0.0:8082->8082/tcp"  2 minutes ago  Up 2 minute
s
0f04a221f07    cnfldemos/cp-server-connect-datagen:0.5.0-6.2.0  "0.0.0.0:8083->8083/tcp, 9092/tcp"  2 minutes ago  Up 2 minute
s
27bdc2960da0   confluentinc/cp-schema-registry:6.2.15   "0.0.0.0:8081->8081/tcp"  2 minutes ago  Up 2 minute
s
3adf2ea6d9a6   confluentinc/cp-server:6.2.15       "0.0.0.0:9092->9092/tcp, 0.0.0.0:9101->9101/tcp"  2 minutes ago  Up 2 minute
s
598dd079dc1e   confluentinc/cp-zookeeper:6.2.15    "2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp"  2 minutes ago  Up 2 minute
s

C:\Windows\System32>docker exec -it broker kafka-topics --bootstrap-server broker:29092 --topic topic1 --create --replication-factor 1 --partitions 1
```





# Confluent Kafka using Local

- docker exec -it broker kafka-topics --bootstrap-server broker:29092 --topic topic1 --create --replication-factor 1 --partitions 1

```
Administrator: Command Prompt
7bdc2960da0  confluentinc/cp-schema-registry:6.2.15          "/etc/confluent/dock..."    2 minutes ago   Up 2 minute
               0.0.0.0:8081->8081/tcp
adf2ea6d9a6   confluentinc/cp-server:6.2.15                 "/etc/confluent/dock..."    2 minutes ago   Up 2 minute
               0.0.0.0:9092->9092/tcp, 0.0.0.0:9101->9101/tcp
98dd079dc1e   confluentinc/cp-zookeeper:6.2.15            "/etc/confluent/dock..."    2 minutes ago   Up 2 minute
               2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp
zookeeper

:\Windows\System32>docker exec -it broker kafka-topics --bootstrap-server broker:29092 --topic topic1 --create --replication-factor 1 --partitions 1
LF4J: Class path contains multiple SLF4J bindings.
LF4J: Found binding in [jar:file:/usr/share/java/kafka/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: Found binding in [jar:file:/usr/share/java/kafka-rest-lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLogge
Binder.class]
LF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
LF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
topic created topic topic1.

:\Windows\System32>
```

# Confluent Kafka using Local

- docker exec -t broker kafka-topics --bootstrap-server broker:29092 --topic topic1 --describe

```
Administrator: Command Prompt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-rest-lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
Created topic topic1.

C:\Windows\System32>docker exec -t broker kafka-topics --bootstrap-server broker:29092 --topic topic1 --describe
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-rest-lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
Topic: topic1    TopicId: TXqKSFvRSOC_wiDLOUnayg PartitionCount: 1      ReplicationFactor: 1      Configs:
          Topic: topic1    Partition: 0      Leader: 1      Replicas: 1      Isr: 1      Offline:

C:\Windows\System32>
```

# KSQLDB



- docker exec -it ksqlDb-cli ksql http://ksqldb-server:8088

Copyright 2017-2021 Confluent Inc.

CLI v6.2.15, Server v6.2.15 located at <http://ksqldb-server:8088>

Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

# KSQLDB

- CREATE STREAM S1 (COLUMN0 VARCHAR KEY, COLUMN1 VARCHAR) WITH (KAFKA\_TOPIC = 'topic1', VALUE\_FORMAT = 'JSON');

```
Administrator: Command Prompt - docker exec -it ksqlDb-cli ksql http://ksqldb-server:8088
ksql> CREATE STREAM S1 (COLUMN0 VARCHAR KEY, COLUMN1 VARCHAR) WITH (KAFKA_TOPIC = 'topic1', VALUE_FORMAT = 'JSON');

Message
-----
Stream created

-----
ksql> CREATE STREAM S2 WITH (KAFKA_TOPIC = 'topic2', VALUE_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 2) AS SELECT *
FROM S1;
Failed to guarantee existence of topic topic2
Caused by: Replication factor: 2 larger than available brokers: 1.
ksql> CREATE STREAM S2 WITH (KAFKA_TOPIC = 'topic2', VALUE_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 1) AS SELECT *
FROM S1;

Message
-----
Created query with ID CSAS_S2_3

-----
ksql>
```

# KSQLDB



- CREATE STREAM S2 WITH (KAFKA\_TOPIC = 'topic2', VALUE\_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 2) AS SELECT \*
- FROM S1;

```
Administrator: Command Prompt - docker exec -it ksqldb-cli ksql http://ksqldb-server:8088
ksql> CREATE STREAM S1 (COLUMN0 VARCHAR KEY, COLUMN1 VARCHAR) WITH (KAFKA_TOPIC = 'topic1', VALUE_FORMAT = 'JSON');

Message
-----
Stream created
-----

ksql> CREATE STREAM S2 WITH (KAFKA_TOPIC = 'topic2', VALUE_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 2) AS SELECT *
FROM S1;
Failed to guarantee existence of topic topic2
Caused by: Replication factor: 2 larger than available brokers: 1.
ksql> CREATE STREAM S2 WITH (KAFKA_TOPIC = 'topic2', VALUE_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 1) AS SELECT *
FROM S1;

Message
-----
Created query with ID CSAS_S2_3
-----

ksql>
```

# Confluent Kafka using Local

- docker exec -t broker kafka-topics --bootstrap-server broker:29092 --topic topic2 --describe

```
Administrator: Command Prompt
FROM S1;
Failed to guarantee existence of topic topic2
Caused by: Replication factor: 2 larger than available brokers: 1.
ksql> CREATE STREAM S2 WITH (KAFKA_TOPIC = 'topic2', VALUE_FORMAT = 'JSON', PARTITIONS = 2, REPLICAS = 1) AS SELECT *
FROM S1;

Message
-----
Created query with ID CSAS_S2_3
-----
ksql> exit
Exiting ksqlDB.

C:\Windows\System32>docker exec -t broker kafka-topics --bootstrap-server broker:29092 --topic topic2 --describe
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/kafka/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-rest-lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
Topic: topic2    TopicId: nto1C1P0R3i-EfuPug-Sgg PartitionCount: 2      ReplicationFactor: 1      Configs: cleanup.policy=delete
          Topic: topic2    Partition: 0      Leader: 1      Replicas: 1      Isr: 1  Offline:
          Topic: topic2    Partition: 1      Leader: 1      Replicas: 1      Isr: 1  Offline:

C:\Windows\System32>
```



# Questions



# Module Summary

- What is Grafana
- How Grafana Works
- Grafana Installation
- DataSources
- Dashboards

