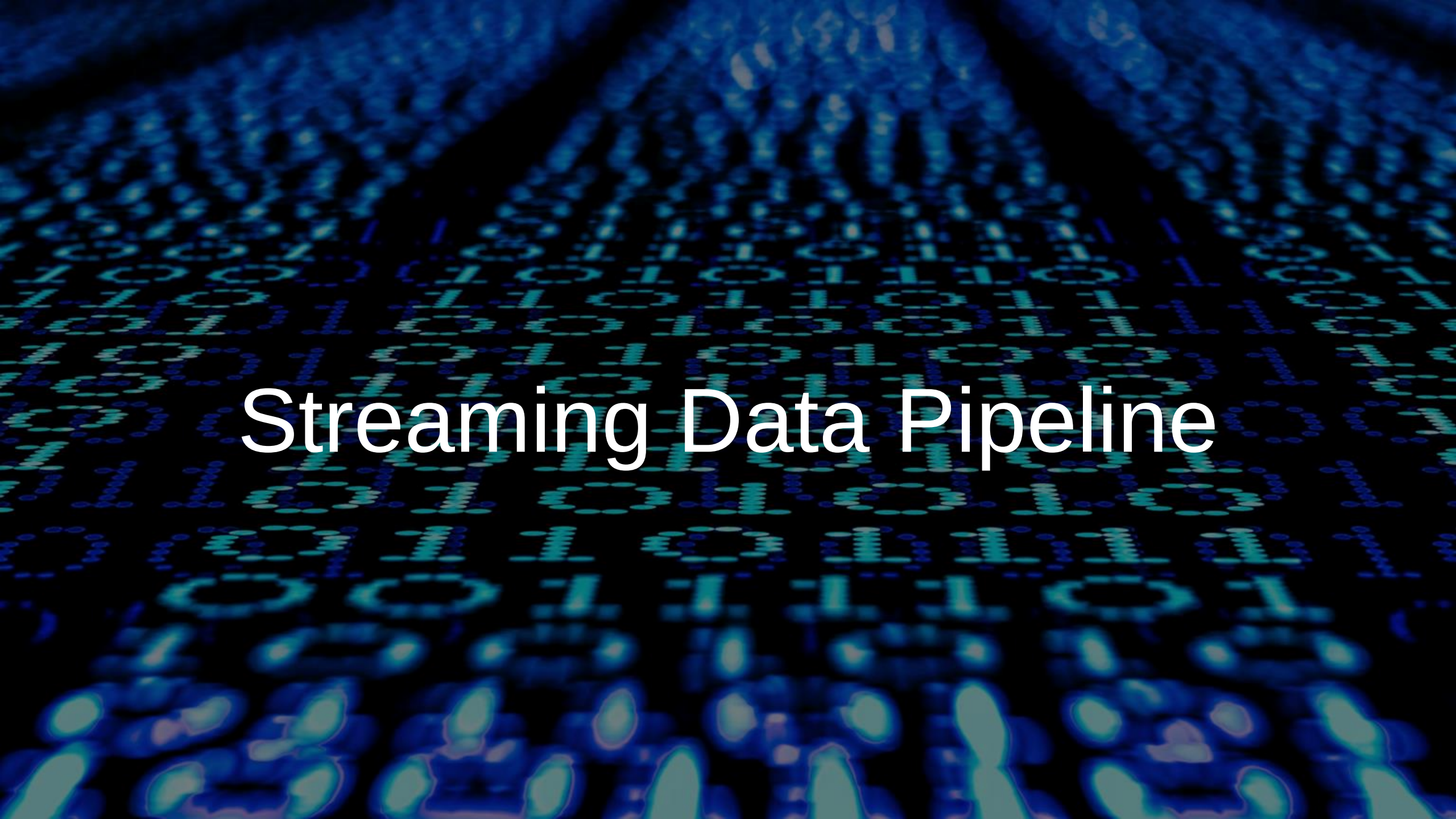
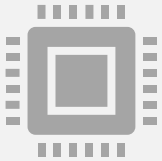

Streaming Data Pipeline

# Introduction to Streaming Data Pipelines

Streaming data pipelines are systems designed to ingest, process, and deliver data in real time.

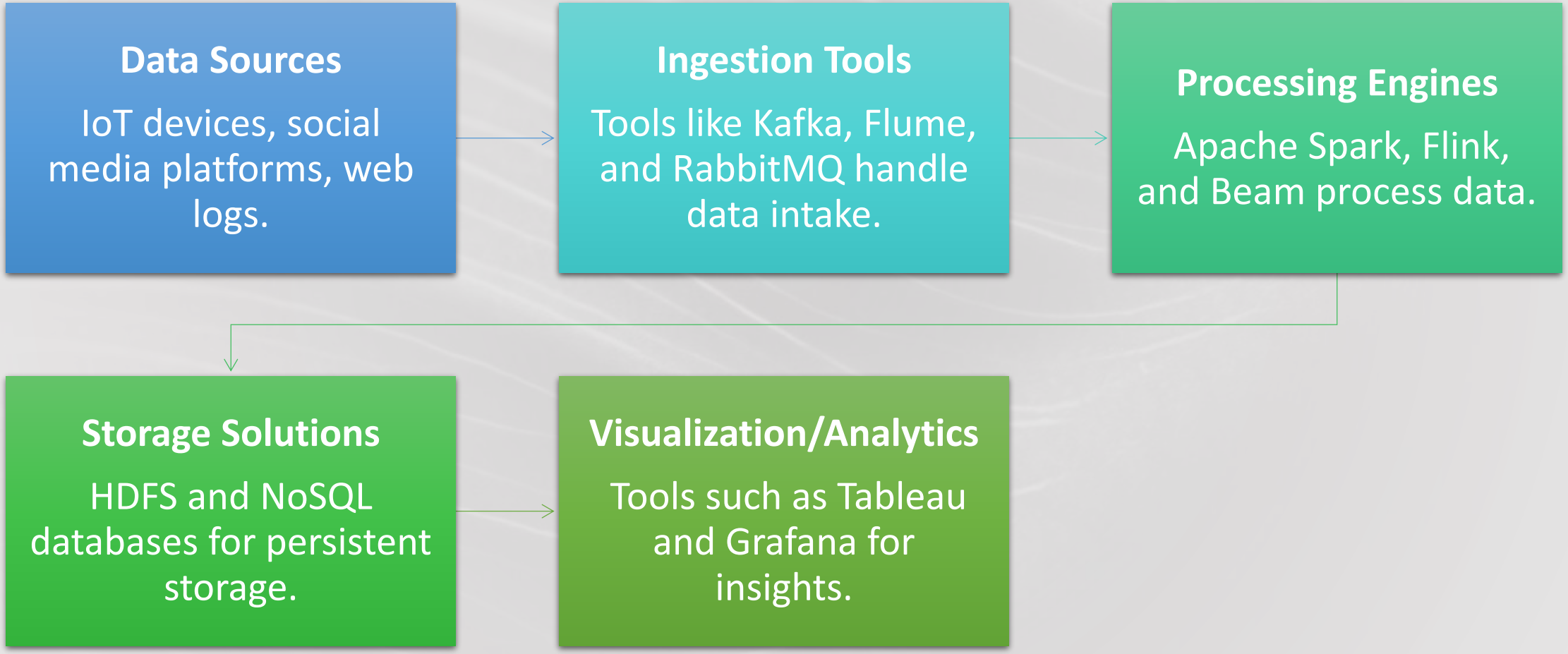


Essential for applications like financial transactions, IoT monitoring, and video streaming.

**Examples**

Fraud detection, live analytics, predictive maintenance.

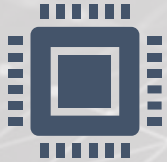
# Key Components of Streaming Data Pipelines



# Benefits of Streaming Data Pipelines

Enable real-time decision-making.

Boost operational efficiency.



Provide scalable and fault-tolerant architectures.

Improve user experiences through fast insights.

# Challenges in Building Streaming Pipelines

Managing high data throughput with low latency.

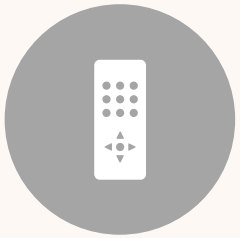
Ensuring fault tolerance and consistent data.

Integrating with diverse systems.

Handling schema changes and compatibility issues.

# Real-World Applications

**FINANCIAL SERVICES** FRAUD DETECTION, STOCK MARKET ANALYTICS.



**HEALTHCARE**

REMOTE PATIENT MONITORING AND DIAGNOSTICS.

**E-COMMERCE**

TAILORED PRODUCT RECOMMENDATIONS.

**MEDIA**

REAL-TIME VIDEO AND AUDIO STREAMING.

**IOT**

APPLICATIONS IN SMART CITIES AND CONNECTED VEHICLES.

# Open-Source Tools for Streaming Data Pipelines

**Apache Kafka**

High-throughput distributed event streaming.

**Apache Flink**

Real-time stream processing framework.

**Apache Beam**

Unified batch and streaming data processing.

**Spark Streaming**

Micro-batch processing for real-time data.

**RabbitMQ**

Lightweight message queuing.

**Pulsar**

Scalable distributed pub-sub messaging.

**Airflow**

Workflow orchestration and scheduling.

# Cloud-Based Streaming Data Pipeline Tools

**AWS Kinesis**

Managed service for real-time stream processing.

**Google Cloud Dataflow**

Unified stream and batch processing.

**Azure Stream Analytics**

Scalable analytics in real time.

**Databricks**

Unified analytics with real-time processing capabilities.

**Snowflake**

Handles continuous data ingestion and analytics.

# Monitoring and Debugging in Streaming Pipelines

**Why It Matters**

Ensures system reliability and performance.

**Key Metrics**

Latency, throughput, error rates.

**Tools**

Prometheus, Grafana, ELK Stack, Kafka Monitoring APIs.

# Trends in Streaming Pipelines

**Event-Driven Architectures**

Reactive systems for real-time operations.

**AI/ML Integration**

Enhanced analytics and predictive modeling.

**Edge Computing**

Processing closer to data sources in IoT.

**Serverless Solutions**

Reduced overhead for streaming services.

# Data Sources and Technologies for Business Insights

Businesses gather data from **internal sources** (e.g., CRM, ERP systems), **external sources** (e.g., social media platforms), and **third-party services** (e.g., Google Analytics).

Various technologies are employed to capture data, including **web scraping tools** and **browser fingerprinting technologies**.

The diversity of sources enables businesses to collect a wide range of structured and unstructured data for analysis.

# Data Pipeline

A data pipeline is a set of tools and processes used to automate the movement and transformation of data between a source system and a target repository.

Data pipeline is the process of moving data from one data source to a target repository.

Businesses can use data pipeline tools to extract data from various sources and load it into a destination such as a data warehouse or data lake.

Along the way, data pipeline architecture ingests and organizes extracted data, allowing business analysts and other users to analyze and gain insights.

# Data Pipeline

- A **data pipeline** is a set of actions that ingest raw data from disparate sources and move the data to a destination for storage and analysis.

- A pipeline also may include filtering and features that provide resiliency against failure.

- It seems as if every business these days is seeking ways to integrate data from multiple sources to gain business insights for competitive advantage.

Source: stitchdata

# How does a data pipeline work?

**1.Data collection:** Data can be collected in batches (e.g., a traditional database) or in real-time (e.g., sensors or sales transactions).

**2.Data ingestion:** Stream or batch ingestion is used to extract data. For example, if you want to extract data from an IoT system, you must use the stream ingestion method. However, if data is extracted at regular intervals. Then you must process and store the data in batches.

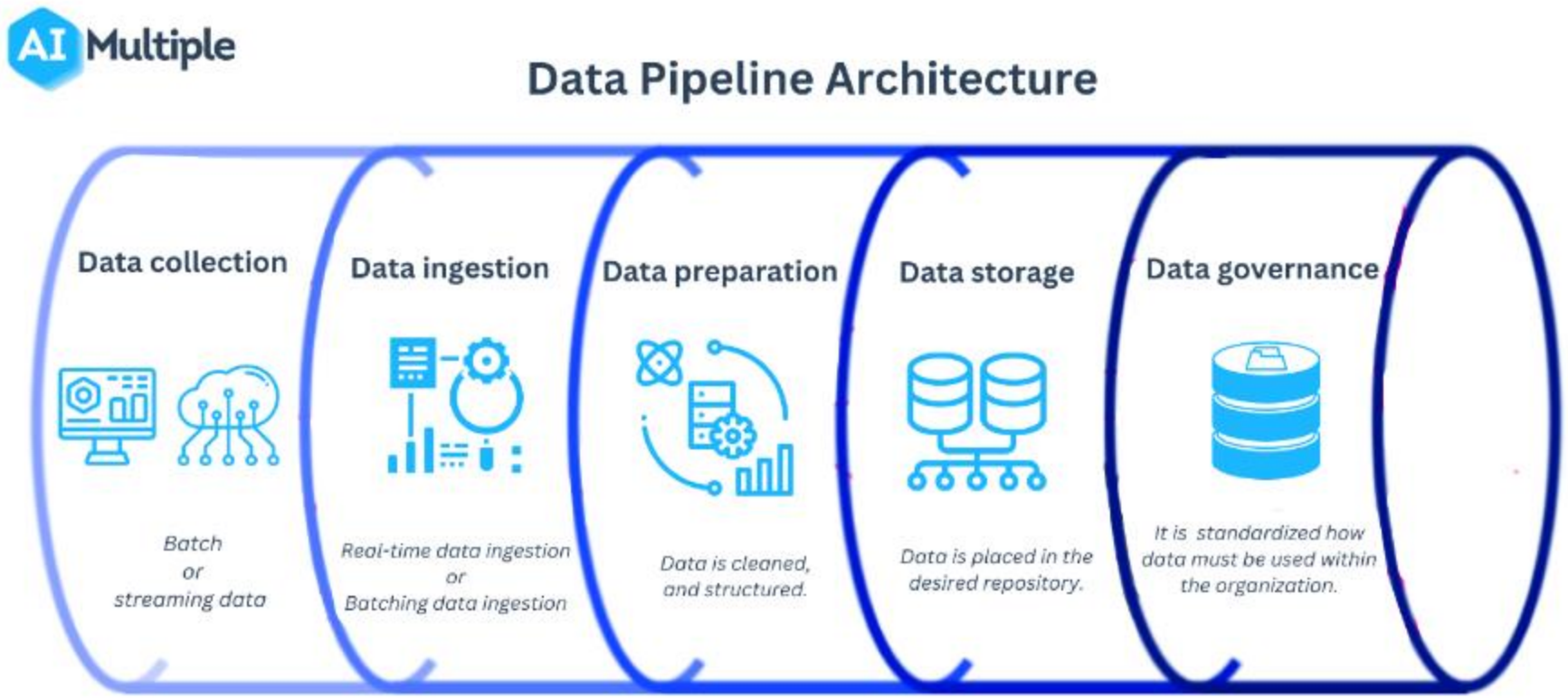
**3.Data preparation:** Data is cleaned and organized.

**4.Data storage:** Data is placed in the target repository.

**5.Data governance:** It is the process of identifying and standardizing how various departments within an organization should use data.

Source: aimultiple

# Data pipeline Architecture



# Why Do We Use Data Pipeline?

Some of the use cases of Data Pipeline are listed below

Delivering the Sales and Marketing data to CRM platforms to enhance customer service.

Streaming the data from sensors to the applications for monitoring the performance and status.

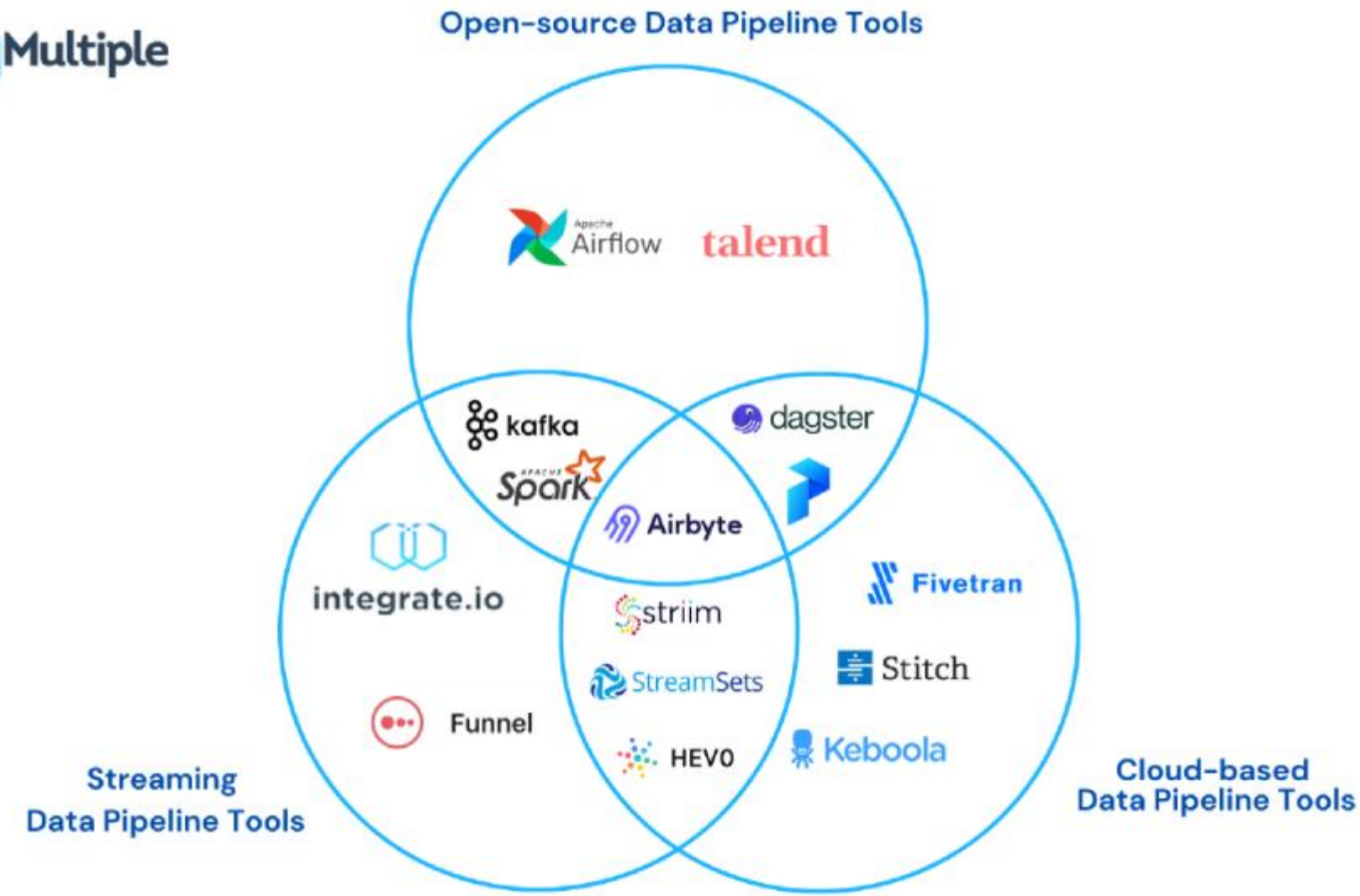
Unifying the data together so that it can speed up the development of new products.

Source: hevodata

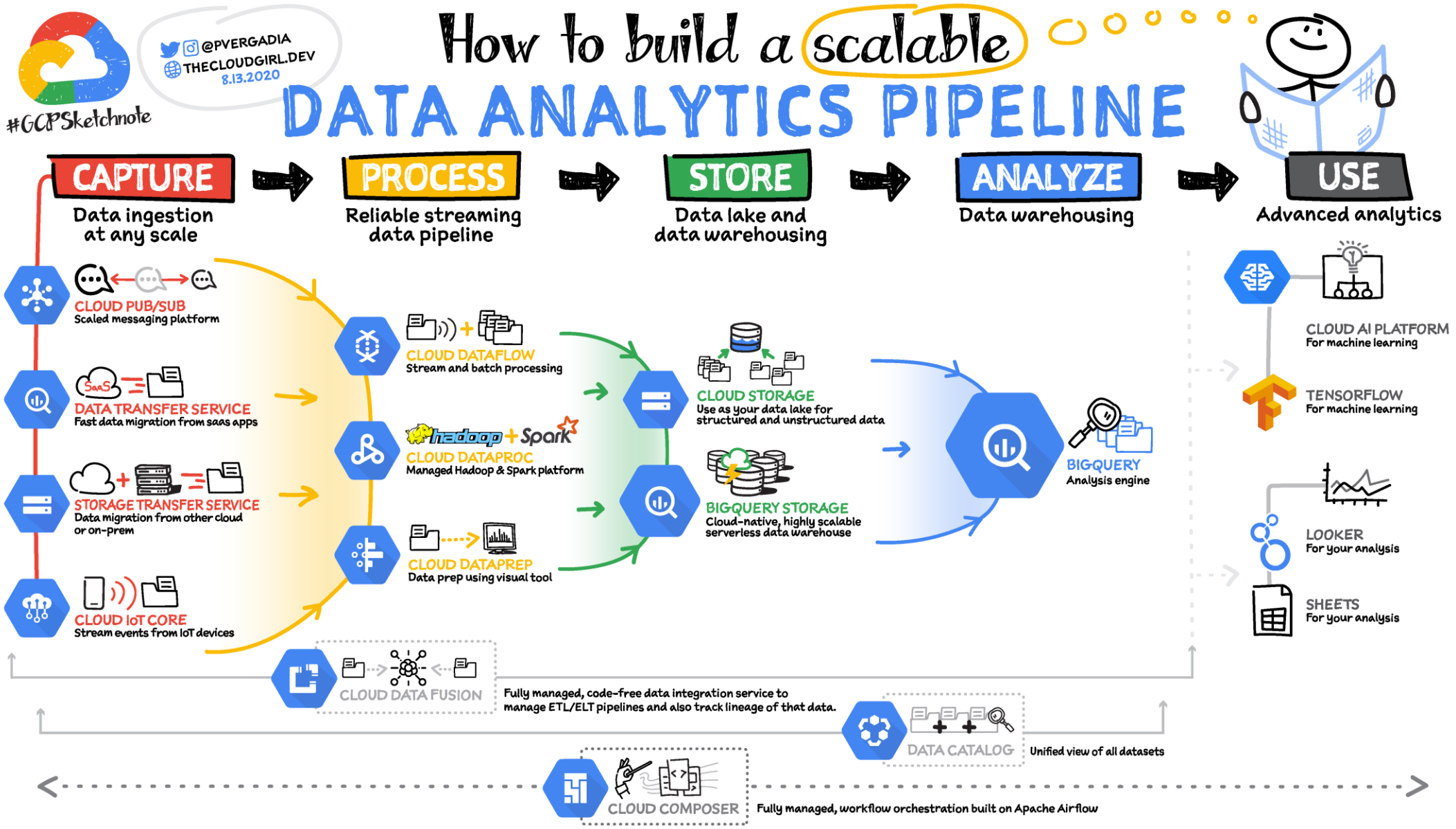
# Data Pipeline vs ETL

| Aspect | Data Pipeline | ETL (Extract, Transform, Load) |
| --- | --- | --- |
| **Definition** | A framework or workflow to move and process data between systems. | A specific process to extract, transform, and load data. |
| **Scope** | Broader, includes real-time streaming, batch processing, and other tasks. | Narrower, focused on transforming and loading structured data. |
| **Data Types** | Handles structured, semi-structured, and unstructured data. | Primarily designed for structured or semi-structured data. |
| **Processing Mode** | Supports both real-time (streaming) and batch processing. | Generally batch processing, with real-time extensions in modern ETL tools. |
| **Use Cases** | Data integration, machine learning pipelines, and event-driven workflows. | Data warehouse population, reporting, and analytics. |

AI Multiple


Open-source Data Pipeline Tools
Apache Airflow
talend
kafka
dagster
Spark
Airbyte
integrate.io
striim
Fivetran
StreamSets
Stitch
Funnel
HEVO
Keboola
Streaming Data Pipeline Tools
Cloud-based Data Pipeline Tools

# GCP-Data pipeline process




Source: Priyanka Vergadia

# Summary

Streaming pipelines drive real-time data handling and analytics.

Critical for industries needing low-latency systems.

Supported by a wide range of tools and technologies.

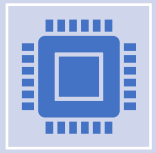
# Apache Kafka

# Introduction to Apache Kafka

Kafka is an open-source platform for distributed event streaming.

**Core Use Case**

Facilitates publish-subscribe messaging, data storage, and stream processing.



**Key Features**

Scalability, high throughput, fault tolerance.

# Kafka Architecture Overview

- **Core Components**
  - **Producers:** Send messages to specific topics.
  - **Brokers:** Servers that manage messages within Kafka.
  - **Consumers:** Retrieve messages from topics.
  - **ZooKeeper:** Manages configurations and cluster state (being replaced by KRaft).
- **Topics**
  - Logical groupings of related messages.

# Apache Kafka

- Kafka also provides a robust set of features including:-
  - Fault-tolerance through automatic replication of data across multiple nodes.
  - Scalability through the ability to add more nodes to handle increasing data volume.
  - Durability through the ability to persist data to disk.
  - Low latency through the use of a distributed architecture.

# Messaging System

There are two types of Messaging System:-

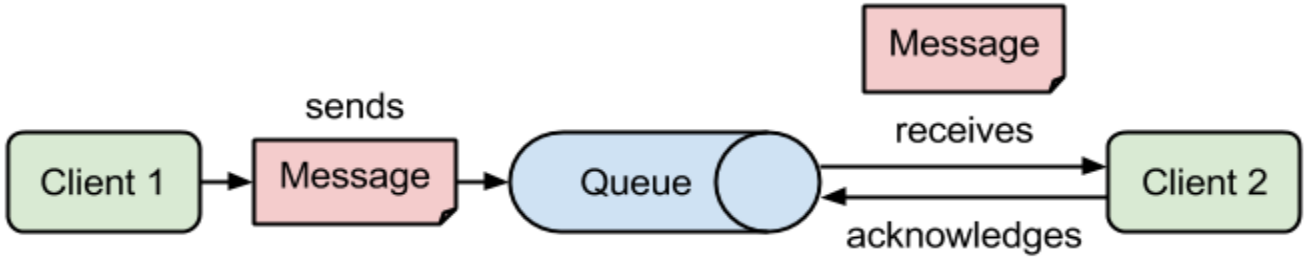
1. **Point to Point System**
2. **Publish-Subscribe System**

**1. Point to Point System**

Messages are persisted in a queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue.

The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time.

The following diagram depicts the structure.

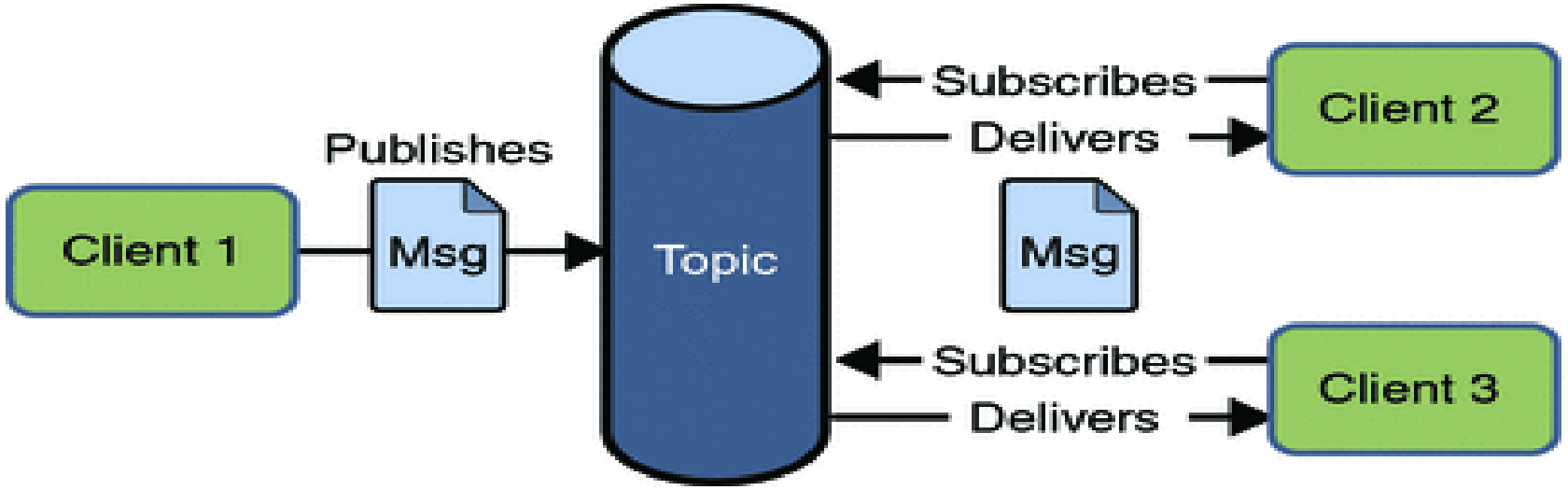


# Messaging System

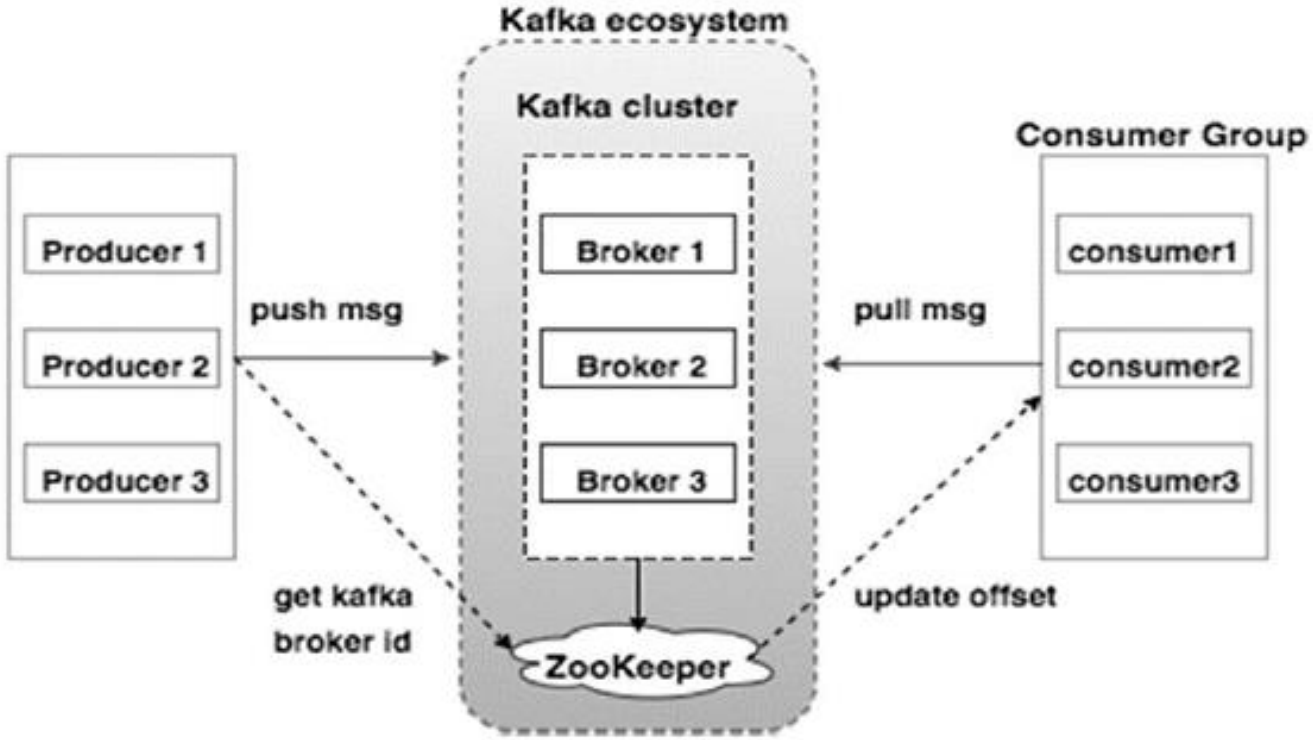
**2. Publish-Subscribe System**

Messages are persisted in a topic.
Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic.
In the Publish-Subscribe system, message producers are called **publishers** and message consumers are called **subscribers.**

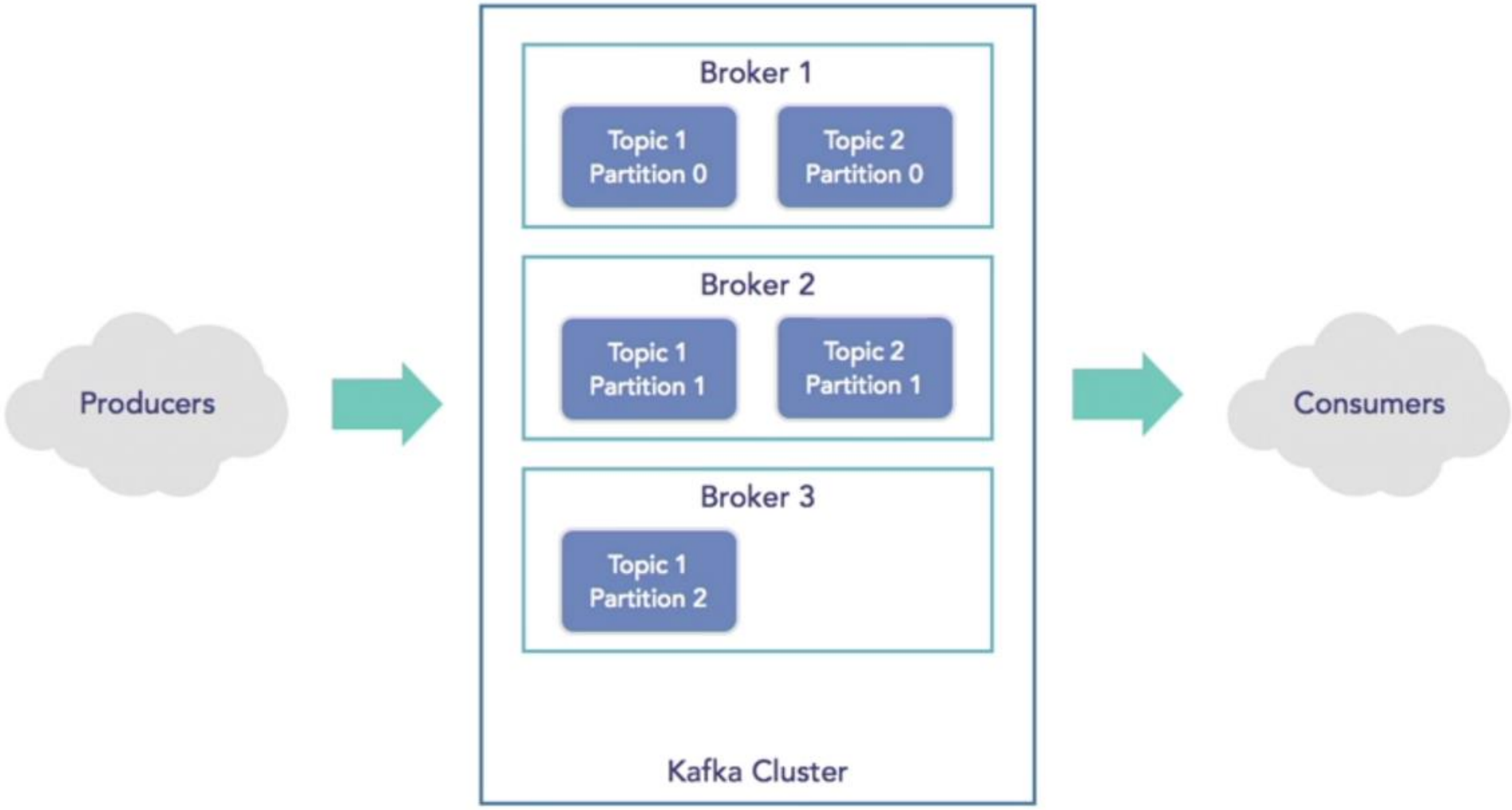
A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.



# Apache Kafka as a Messaging System



# Apache Kafka Architecture



# Apache Kafka Work Flow

Following is the step wise workflow of the Pub-Sub Messaging :-

- **Producers send message to a topic at regular intervals.**
  - Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions.
  - If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
- **Consumer subscribes to a specific topic.**
  - Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper.
  - Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.
  - Once Kafka receives the messages from producers, it forwards these messages to the consumers.
  - Consumer will receive the message and process it.
  - Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
  - Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper.
  - This above flow will repeat until the consumer stops the request.
  - Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages

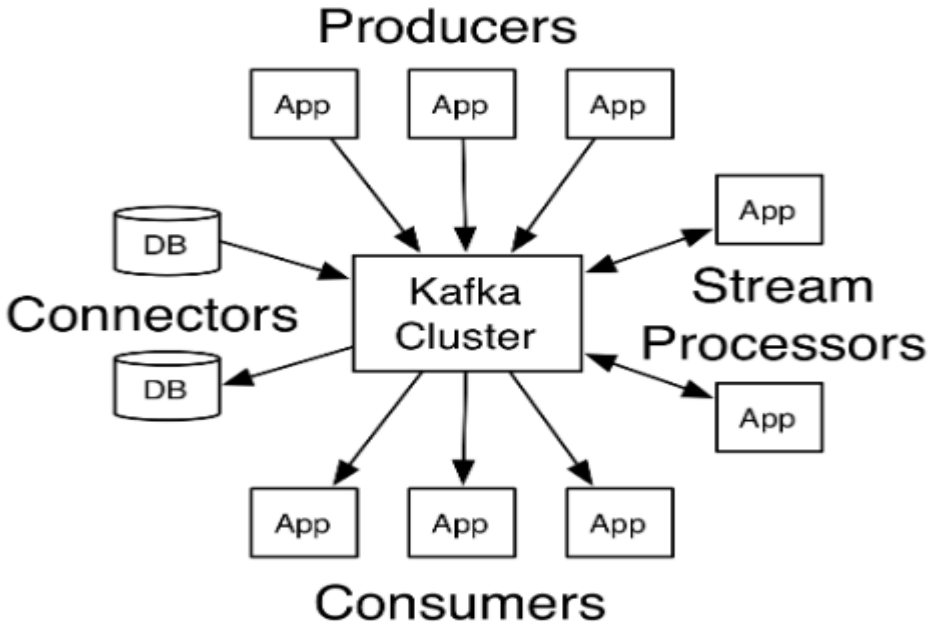
# Apache Kafka Core API

The Producer API allows an application to publish a stream of records to one or more Kafka topics.

The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.

The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.

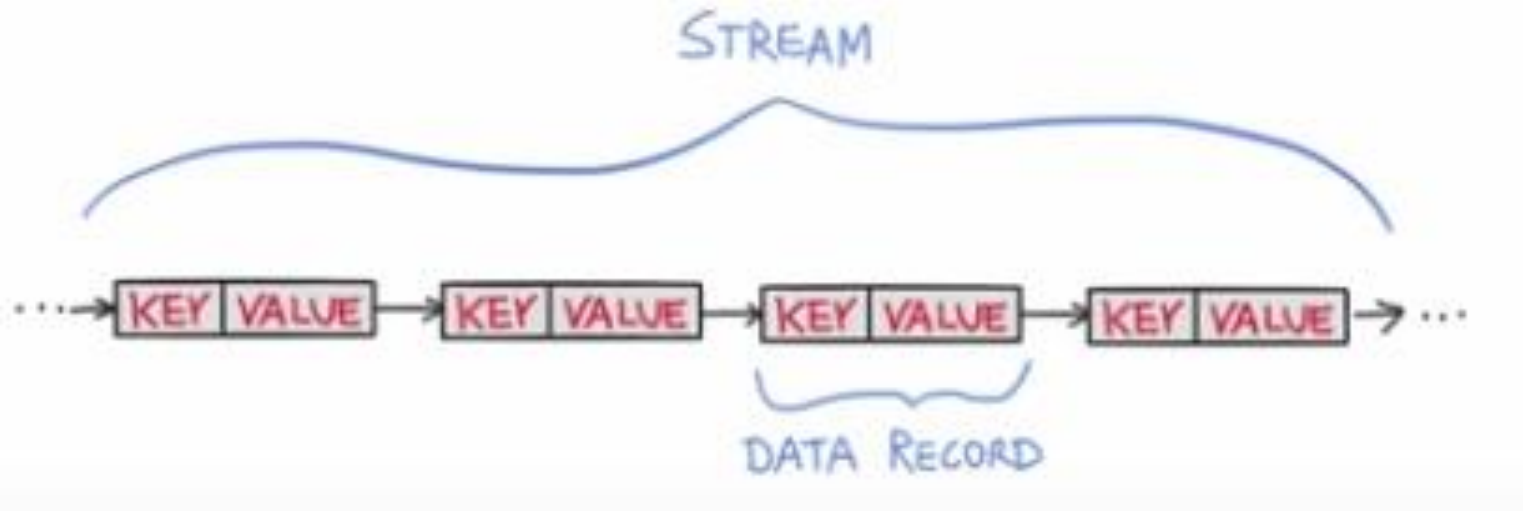
The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.



# What is Streams

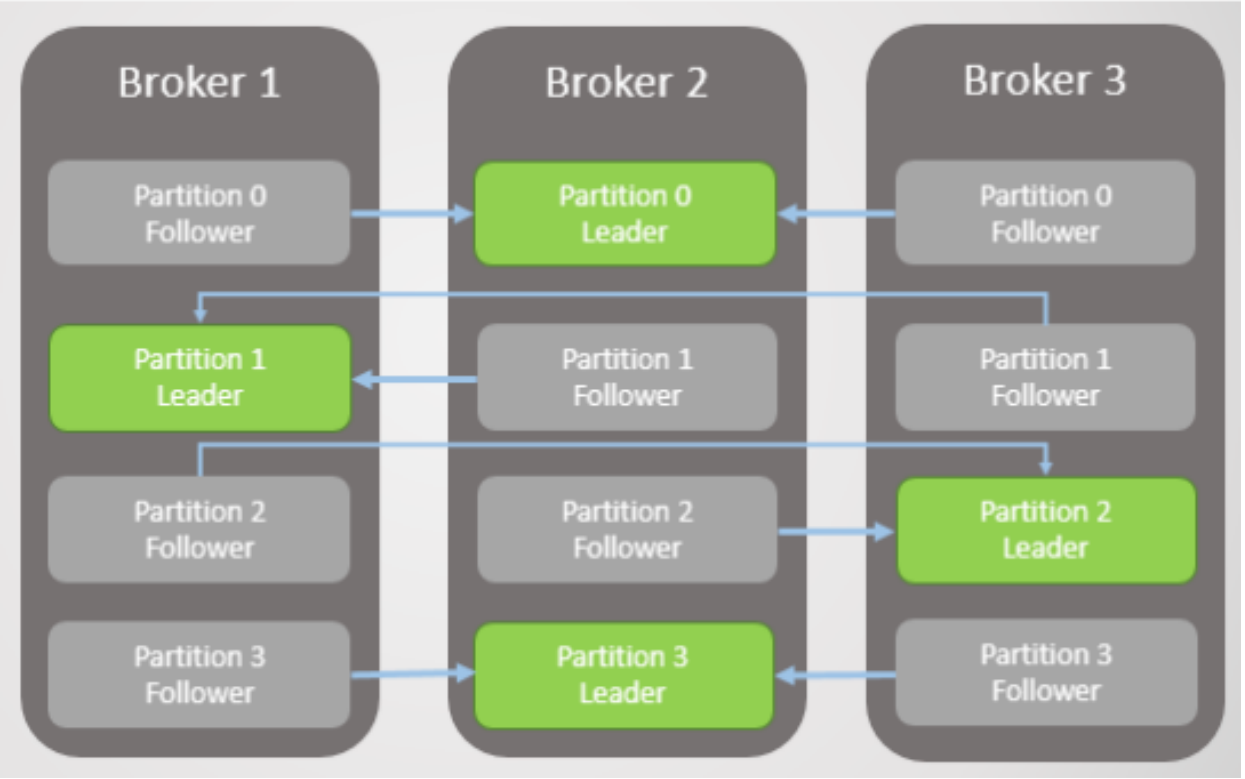
Think of s Stream as an infinite. Continuous real-time flow of data.
data are a key-value pairs.

In **Kafka stream API** transform and increases data.
- Support per-record stream processing with milisecond.



# Kafka Components

- Partitions for the same topic are distributed across multiple brokers in the cluster.
- Partitions are replicated across multiple servers; number of replicas is a configurable parameter.
- Each Partition has one server as a *leader* and a number of servers as *followers.*
- Each Server acts a leader for some of its partitions and as a follower for some other.
- The Producers are responsible for choosing which message to assign to which partition within the topic based on key assigned to message.



# Kafka Cluster

**Scalability**

Broker clusters distribute workloads.

**Partitioning**

Divides topics for parallel processing.

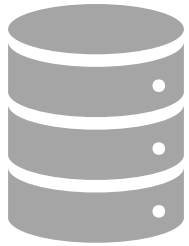
**Replication**

Ensures data availability across nodes.

# Producers and Consumers

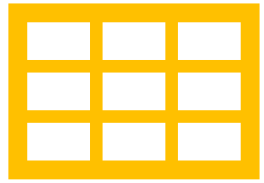
**Producers**

Publish data to designated topics.



**Consumers**

Retrieve data, subscribing to specific topics or partitions.



**Consumer Groups**

Enable parallel data consumption.

# Kafka Topics and Partitions

**Topics**

Organize messages logically.

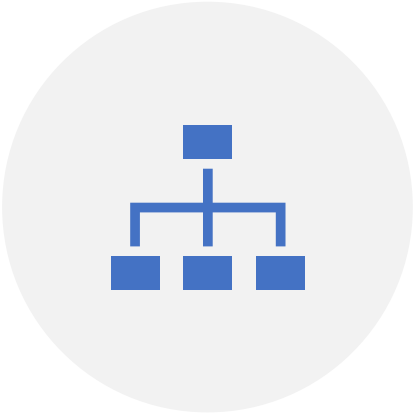
**Partitions**

Allow parallel data handling.

**Replication**

Each partition has a leader and backup replicas.

# Kafka with Zookeeper



**ROLE**

COORDINATES BROKERS AND MANAGES METADATA.

**FUNCTIONS**

LEADER ELECTION AND FAULT RECOVERY.

**FUTURE**

KRAFT WILL ELIMINATE DEPENDENCY ON ZOOKEEPER.

# Kafka Ecosystem

**Kafka Connect**

Simplifies integration with external systems.

**Kafka Streams**

API for stream processing.

**KSQL**

Enables SQL-like queries on Kafka topics.

**MirrorMaker**

Replicates data across clusters.

# Kafka in Action-Companies

**LinkedIn**

Powers real-time user activity tracking.

**Netflix**

Handles streaming analytics and recommendation engines.

**Uber**

Supports dynamic pricing and geolocation services.

**Spotify**

Enables personalized user recommendations.

**Goldman Sachs**

Processes high-frequency financial transactions.

# Kafka Integration with Other Tools

**Hadoop/Spark**

For big data batch and stream processing.

**Flink**

Real-time analytics and stream processing.

**ElasticSearch**

For log analysis and visualization.

**MongoDB**

Stores processed event data for querying.

# Kafka Use Cases

**Log Aggregation**

Centralizing application logs.

**Event Sourcing**

Maintaining event history.

**Metrics Collection**

Monitoring system performance.

**Real-Time Analytics**

Insights as events happen.

**Messaging Backbone**

Distributed communication across systems.

# Advantages of Apache Kafka

**High Throughput**

Handles millions of events per second.

**Scalability**

Add brokers to increase capacity.

**Fault Tolerance**

Replication ensures data reliability.

**Ecosystem**

Broad compatibility with external tools.

# Summary

# Summary