# Big Data Infrastructure Backbone System

Big Data on Kubernetes – [Day 2]


LUAN MORENO

CEO & Data Architect
Data Engineer & MVP



MATEUS OLIVEIRA

Big Data Architect
Data In-Motion Specialist

# Cloud Managed Services

IaaS, PaaS and SaaS Software for Easy and Fast Development for Enterprises

Market Share by 2020

AWS = 52%

Microsoft Azure = 21%

Google Cloud Platform = 18%

Enterprise Support

24x7 Customer Support

IaaS, PaaS and SaaS Softwares for Easy and Fast Development

Managed Infrastructure

Managed by Cloud Provider

Pay as You Go

Most customers use resources on business hours and tears down after to reduce costs

Small Teams

Easy Collaboration using DevOps Principles Autonomous and Automatic Processes to Reduce Maintenance

Latest Versions

Latest Versions of the Products. Battle-Tested before going to GA

Business-Oriented

Focused on Delivery and Success of Your Business

# Open-Source Solutions

Best of Breed Big Data Technologies. Bleeding-Edge and Most Used in the Market

## Open-Source Jobs Report

- 455% Demand for Kubernetes
- 97% Hiring OS is Priority
- 50% Increase in Hiring

### Core Technologies

Open-Source, Worldwide Community, Feature Discussion, Constant Improvement, Vendor Neutral

### Cloud Services Engines

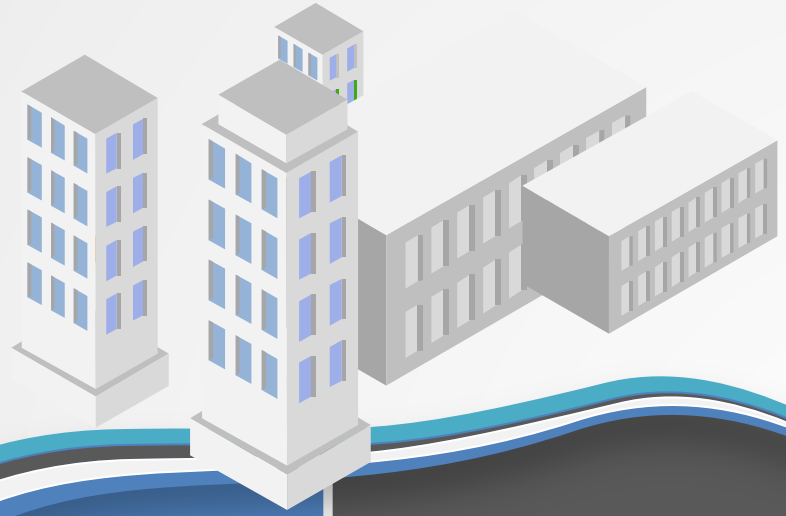
Used by Cloud Providers Encapsulated as Managed Services for Easy-to-Use Approach

### Costwise

Without License Fees, No Lock-In and, Almost No Code Change for Cloud Transition

### Create a Product

Create your Own Product, Without Lock-In and Low Tntry Point for Business



# Big Data Products Overview

Layers of a Big Data Architecture, and Open-Source Products Maintained by Cloud Providers and OSS Community

## Data Lake

store data coming from databases, internal & external APIs and also multiple services

- Azure Blob Storage
- AWS S3
- Google Cloud Storage
- MinIO

## Ingestion

ingest data from database and APIs with low latency (milliseconds spectrum)

- Azure Event Hubs
- AWS Kinesis
- Google Cloud Pub/Sub
- Apache Kafka
- Apache Pulsar

## Processing

process TB and PB of data, scale more efficiently, working on both batching or streaming pipelines

- Azure Synapse Analytics [Spark Pools]
- AWS Glue
- Google Cloud Dataflow
- Apache Spark
- KSQLDB

## Serving

store analytical and processed data into an analytical datastore

- Azure Synapse Analytics [SQL DW]
- AWS Redshift
- Google Cloud BigQuery
- Apache Pinot
- YugaByteDB

## Data as Product

deliver the pipeline for end users using visualization tools and self-service analytics

- Microsoft Power BI
- AWS Quick sight
- Google Cloud Data Studio
- Apache Superset
- Metabase

# Cost Comparison

Compare Cloud Provider's Costs against
Kubernetes Based Big Data Deployment

## Total Cost for Data Pipelines on Azure & AWS

- **Storage Layer =** R$ 2.414
- **Data Processing Layer =** R$ 6.556
- **Data Serving Layer =** R$ 14.580

- Total Monthly Cost – R$ 23.550

## Total Cost for Big Data Stack

- **Layers =** Ingestion, Processing, Serving, Visualization, Integration, Exploration, Orchestration, Logging, Monitoring, CI
- **Products =** Apache Kafka, Apache Spark, KSQLDB, MinIO, Apache Pinot, YugaByteDB, Trino, Zeppelin, Metabase, Superset, Apache Airflow, ELK, Prometheus, Grafana & ArgoCD
- **Category =** General Purpose
- **Series =** DSV4
- **Instance =** 4 vCPUs & 16 GB of RAM - [24 vCPUs & 96 GB of RAM]
- **VMs =** 6
- **Hours =** 730 Hs

- Total Monthly Cost – R$ 7.150 ~ 3X of Savings for ALL Layers

# Decision Points

Better Strategy to choose the right
Infrastructure for Big Data

## Moment of business

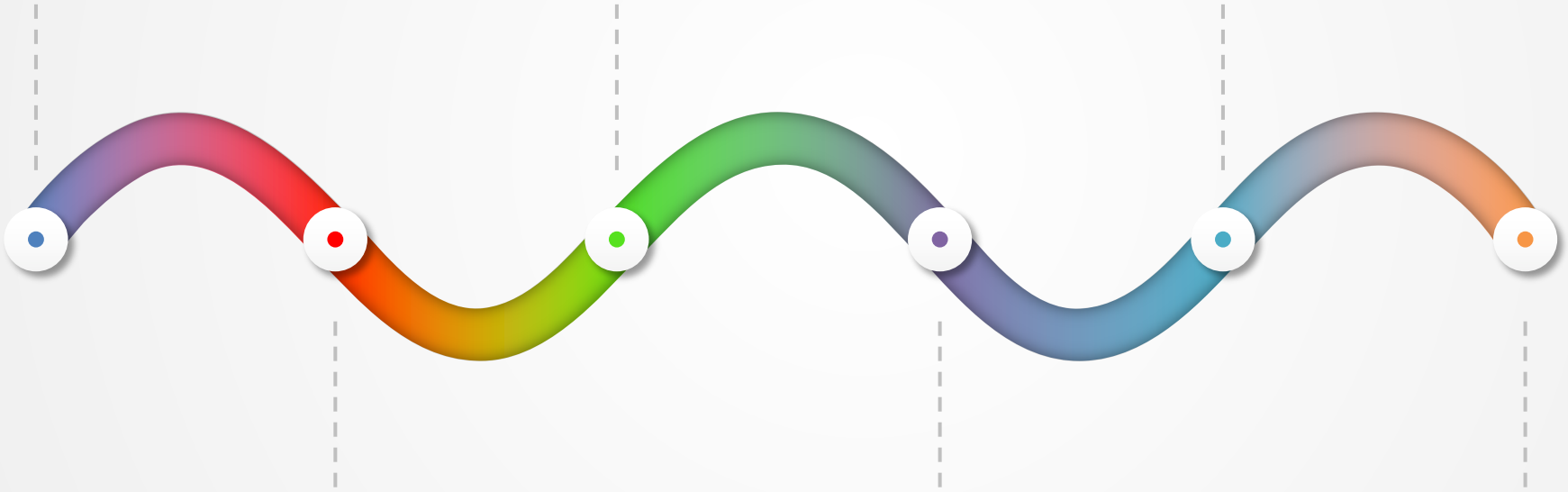
Evaluable the end goal of your pipelines and what are the main partnerships. Understand how to position the solution offered

## Budget

Not only understand, how much is going to cost but how much will cost to hire or train people to operate the solution

## Sight of the future

Need to know where you want to be, and which technologies can deliver the best insights, and fast to migrate if the business need



## Teamwise

Your team must be aligned with all the drawbacks in this new approach for big data architecture

## Automate

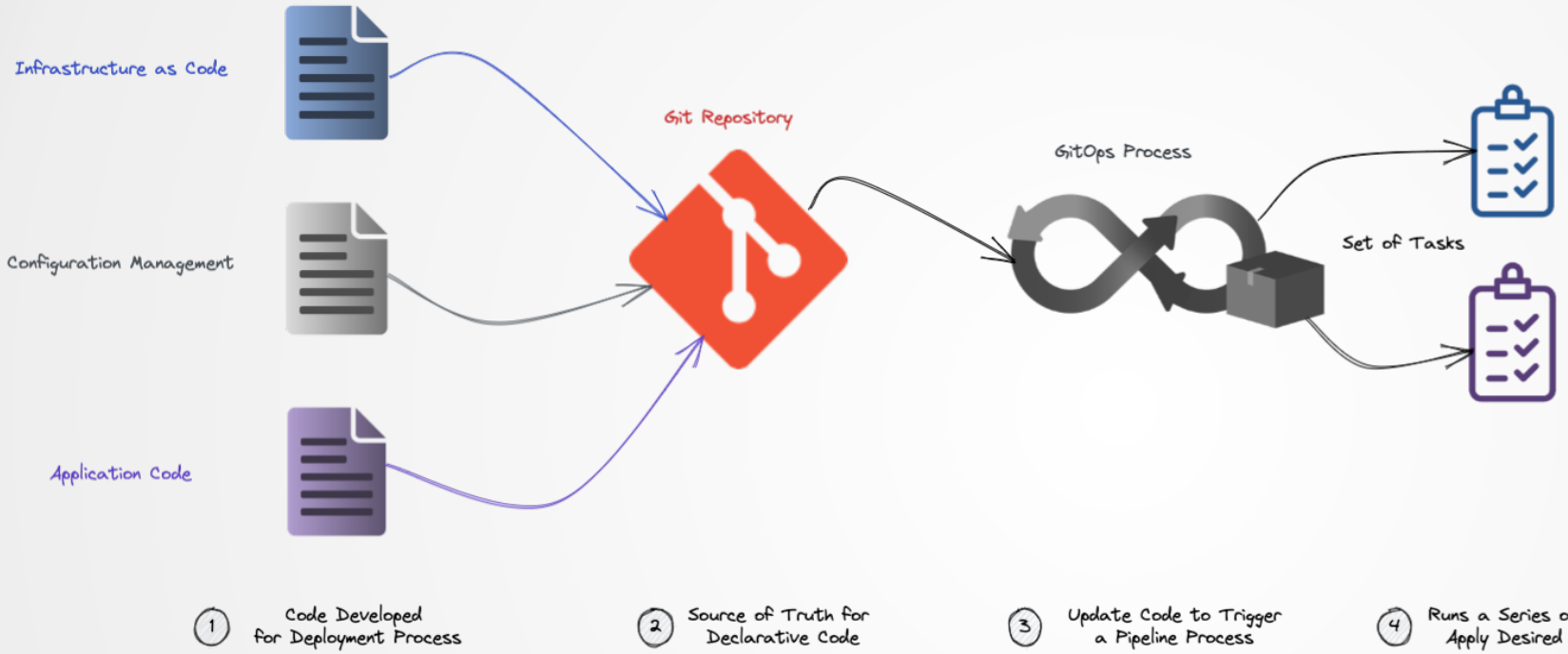
Understand your data and the level you can automate to deliver pipelines that reduces overall complexity

## Goal

More that know the future is to now the present goal, what can be better for the business today and think about cost of any nature.

# The Principles of GitOps Model

Set of Practices that Empowers Developers to Perform Tasks for CI/CD Lifecycle Process


## DevOps

- Set of Practices to Combine Dev & IT Ops
- Shorten System Development Life Cycle
- Provide CI & CD Approaches
- Usually, Use of Agile Methodology

## Continuous Integration (CI)

- Software Development Strategy
- Increase Speed of Development
- Ensure Quality of Code
- Continually Commit Code in Small Increments
- Automatically Built and Tested Before Merged
- Use of a Git Repository

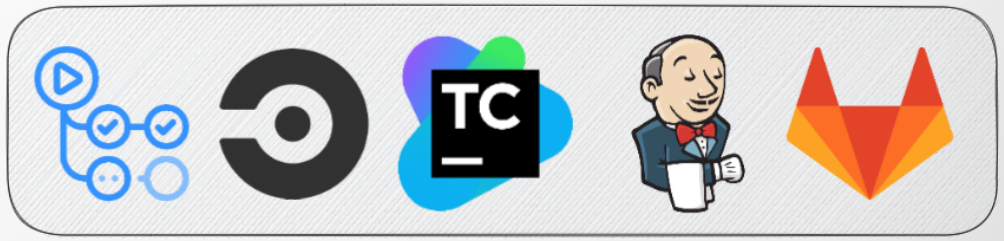
## Continuous Delivery (CD)

- Software Delivered to Production at Any Time
- Produce Software in Short Cycle
- Reliably Release at Any Time
- Automatic Process
- Build, Test and Release

## GitOps Workflow Process

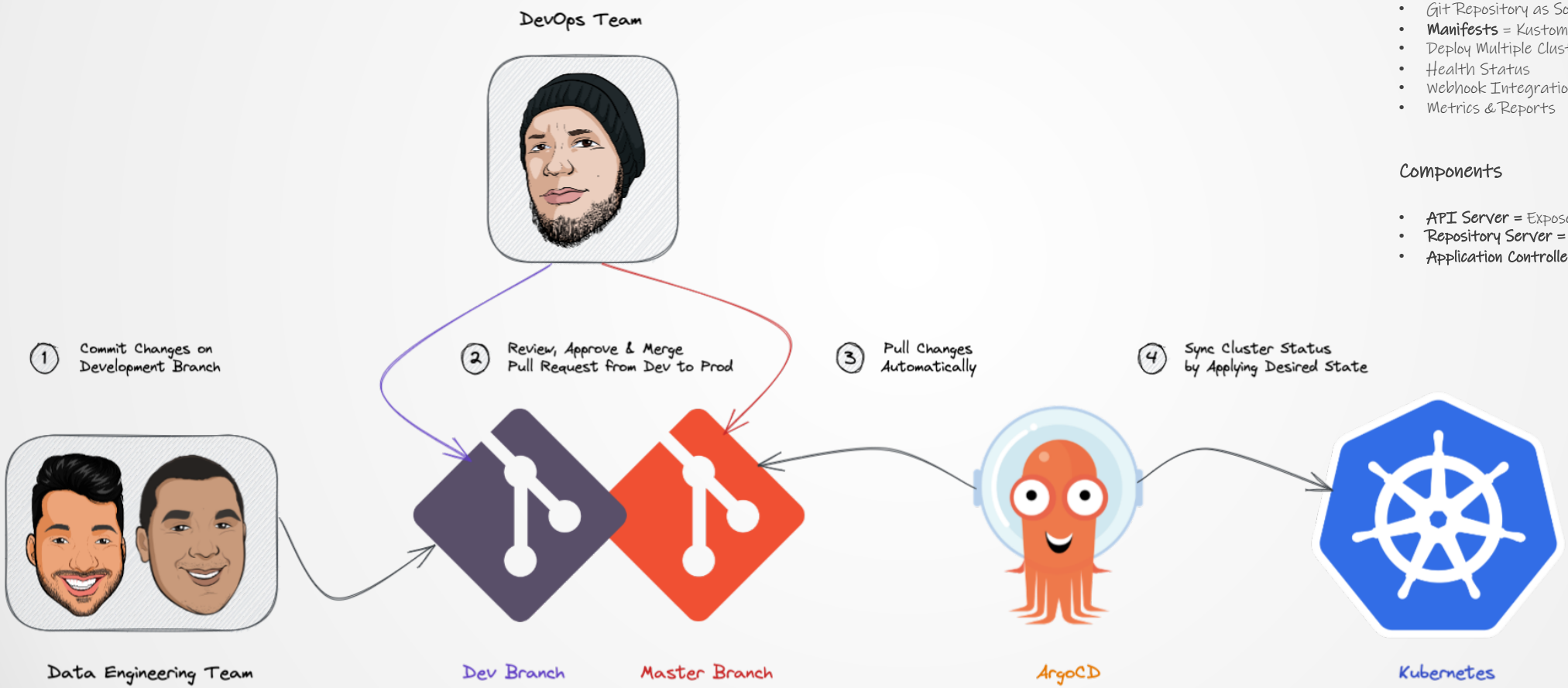
- Set of Practices to Manage Infrastructure and Application
- Use Git as a Single Source of Truth for Declarative Process
- Git Pull Requests to Automatically Manage Deployments
- Apply Desired State of System
- Enable CD for Kubernetes

**Used for Kubernetes and Cloud Native Applications**



# Automate Continuous Delivery (CD) using ArgoCD Tool

Declarative GitOps Tool for Kubernetes. Application Deployment and Lifecycle Management




## ArgoCD

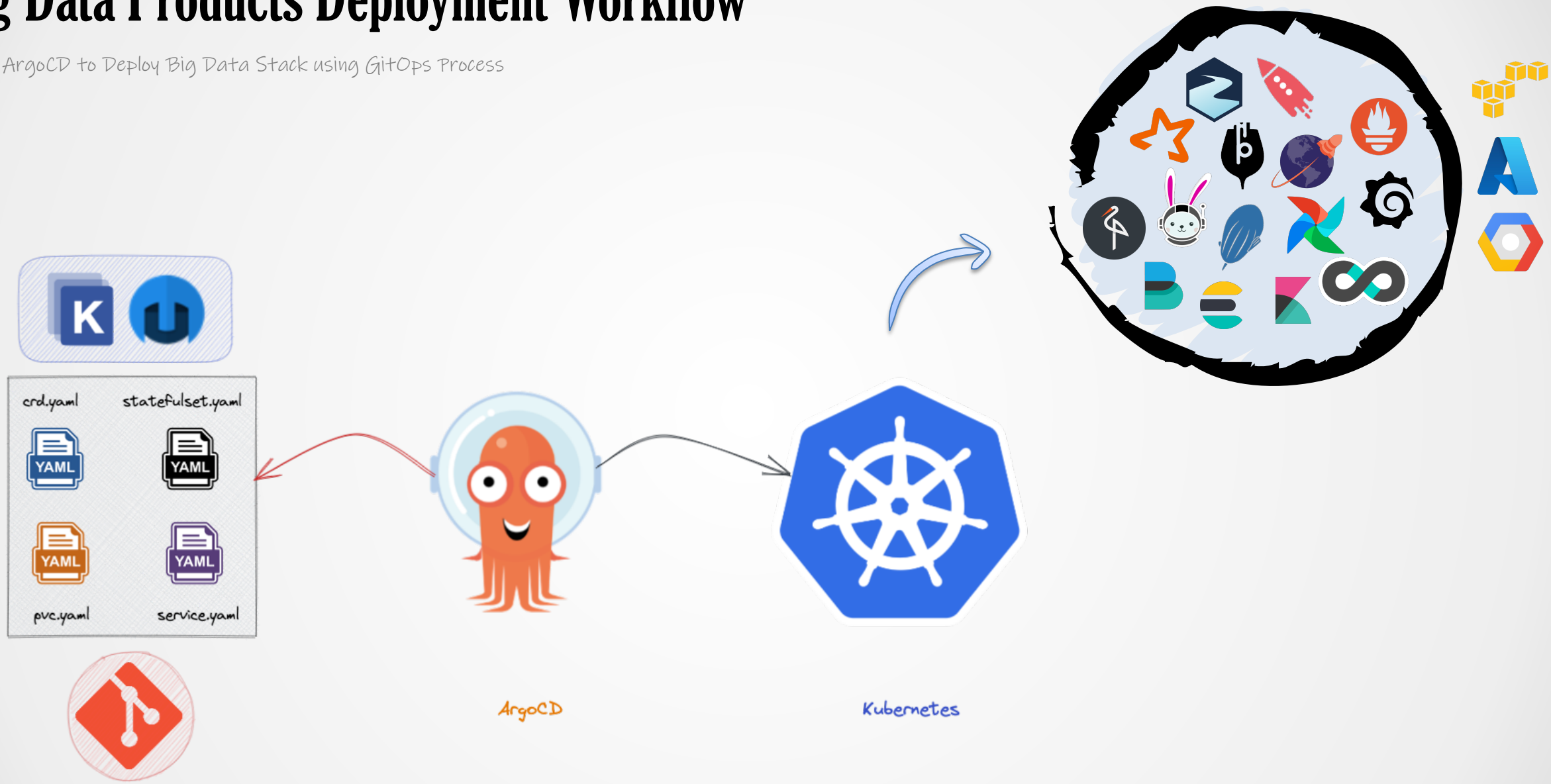
- Git Repository as Source of Truth
- **Manifests** = Kustomize & Helm
- Deploy Multiple Clusters
- Health Status
- Webhook Integrations
- Metrics & Reports

## Components

- **API Server** = Expose API for Application Management
- **Repository Server** = Maintains Local Cache of Git
- **Application Controller** = Continuously Monitors Apps

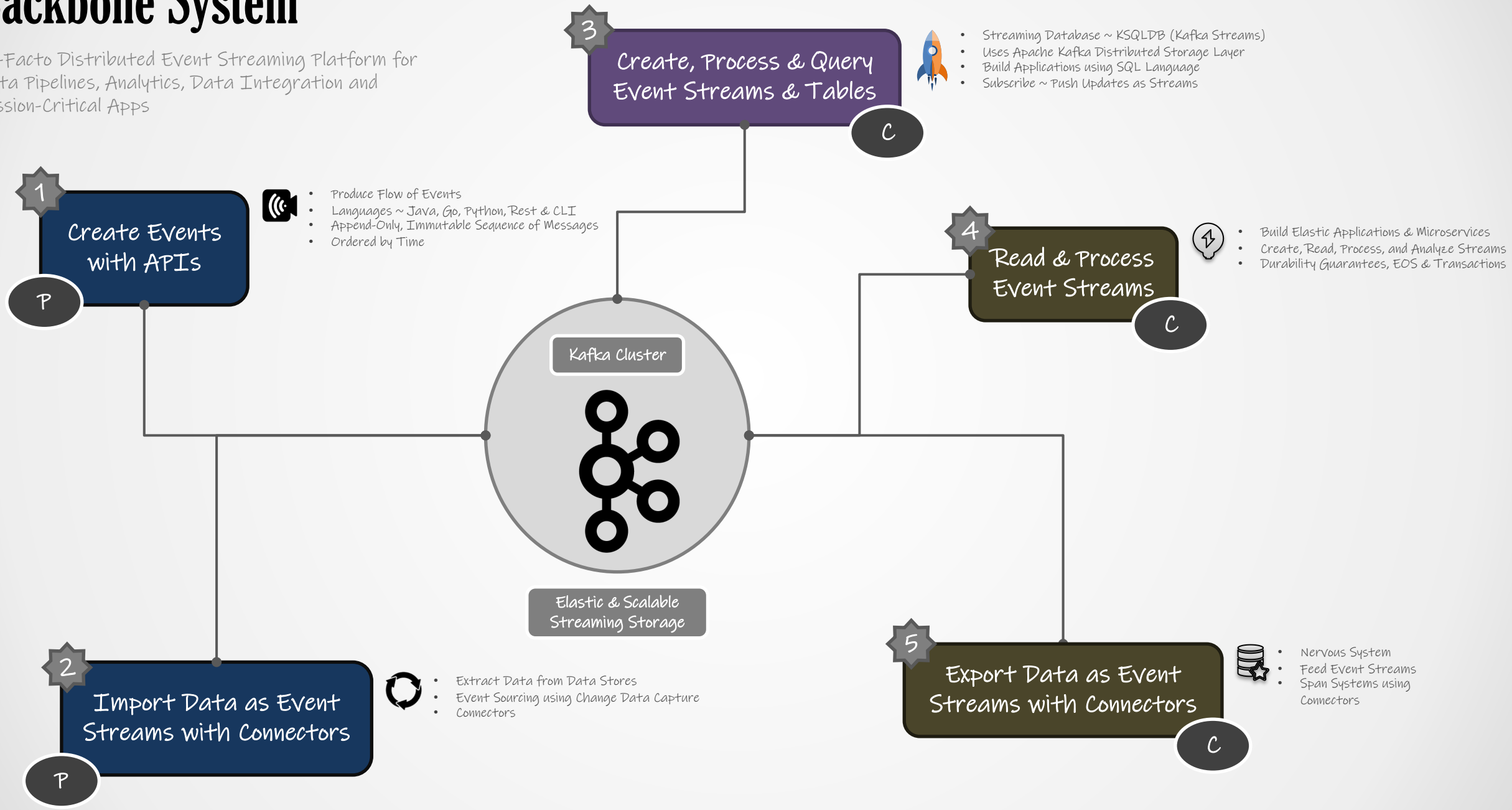
# Big Data Products Deployment Workflow

Using ArgoCD to Deploy Big Data Stack using GitOps Process



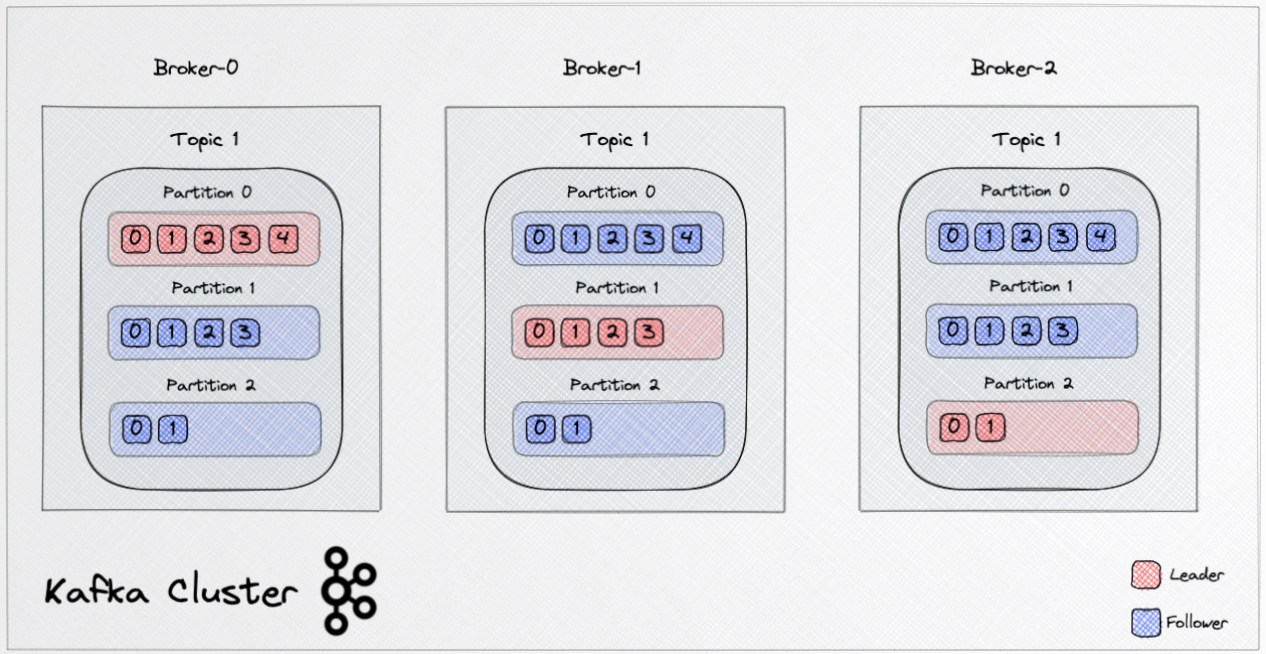
# Backbone System

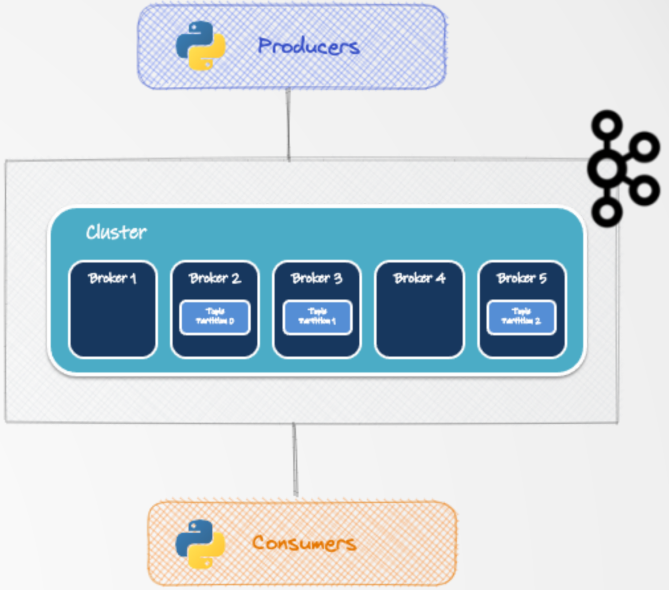
De-Facto Distributed Event Streaming Platform for Data Pipelines, Analytics, Data Integration and Mission-Critical Apps

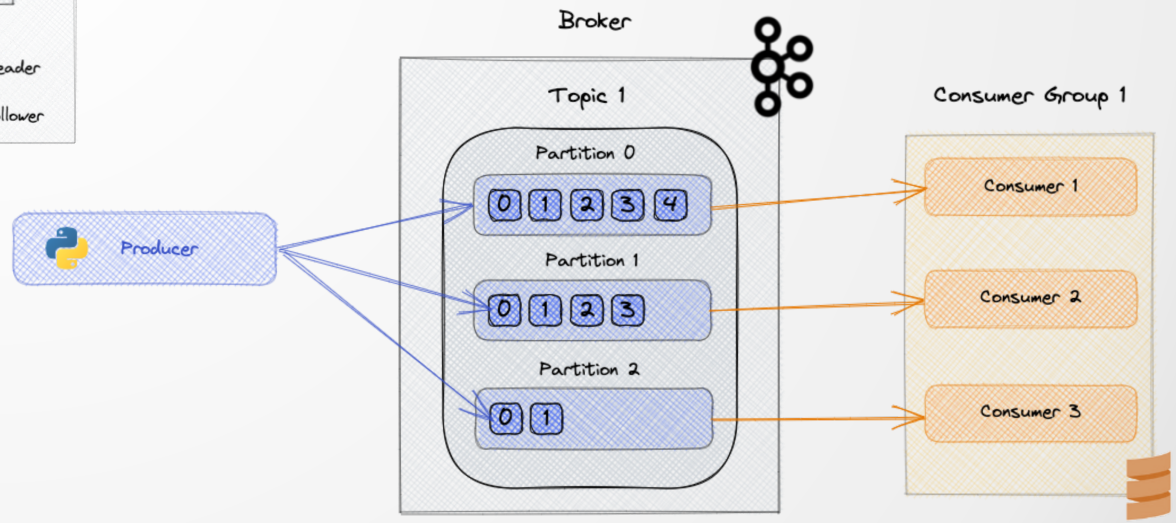


# Components & Architecture

Main Components and Architecture of Apache Kafka, Available Features and Characteristics




Characteristics

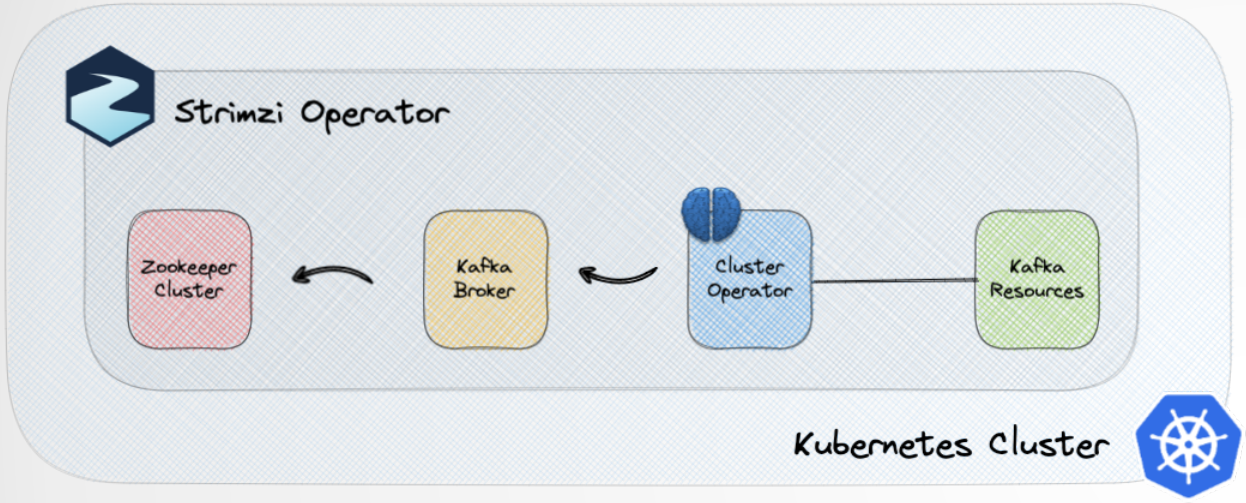
- **Kafka Broker** = Server or Node, Orchestrates Storage
- **Cluster** = Group of Broker Instances

- **Topic** = Main Entry, Split into One or More Partitions
- **Partition** = Horizontal Partitioning for Writes

- **Partition Leader** = Handles ALL Producer Requests
- **Partition Follower** = Replicates Partitions from Leader

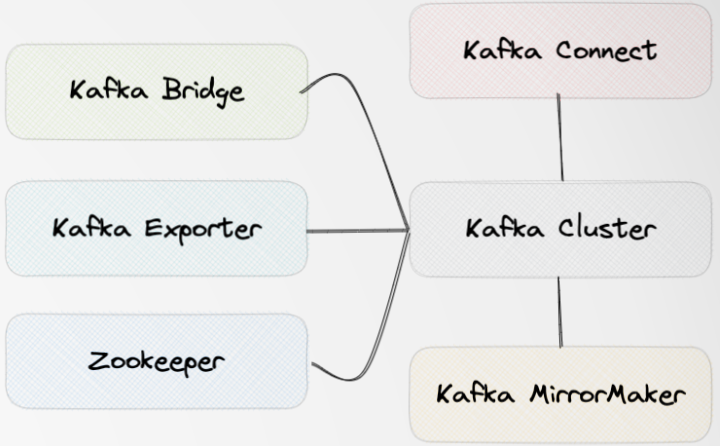
# Strimzi - Kafka on Kubernetes

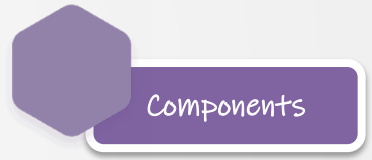
The Best Way to Run Kafka on Kubernetes in Various Deployment Flavors, From Development to Production Effortlessly




## Operators

- **Cluster** = Deploys and Manage Resources
- **Entity** = Topic & User Operator
- **Topic** = Manages Kafka Topics
- **User** = Manager Kafka Users



## Components

- **Kafka Broker** = Cluster of Broker Nodes
- **Zookeeper** = Replicated Instances
- **Kafka Connect** = External Data Connections
- **Kafka MirrorMaker** = Mirror for Secondary Cluster
- **Kafka Exporter** = Extract Metris for Monitoring
- **Kafka-Bridge** = HTTP-Based Request Component
- **Cruise Control** = Optimize Rebalance and Anomalies

## Storage Types

Apache Kafka & Zookeeper Store Data on Disks

* Ephemeral
* Persistent
* JBOD

## Listener Types

How Client Connects to a Kafka Cluster

* Route
* Load Balancer
* Node Port
* Ingress

## Security Options

Perform a Secure Deployment of Apache Kafka on Kubernetes

* In-Transit mTLS
* SALS SCRAM-SHA-512
* OAuth 2.0 Token Based
* ACLs

# Features

Kubernetes Native Multi-Cloud Open-Source Object Storage Available on Public, Hybrid and Private Cloud


## Information

- **Interfaces** = Client, Admin, Server, Gateway and Console
- **Notifications** = ElasticSearch, Kafka, Postgres, Redis
- **Gateway** = NAS, Azure, S3, HDFS, GCS

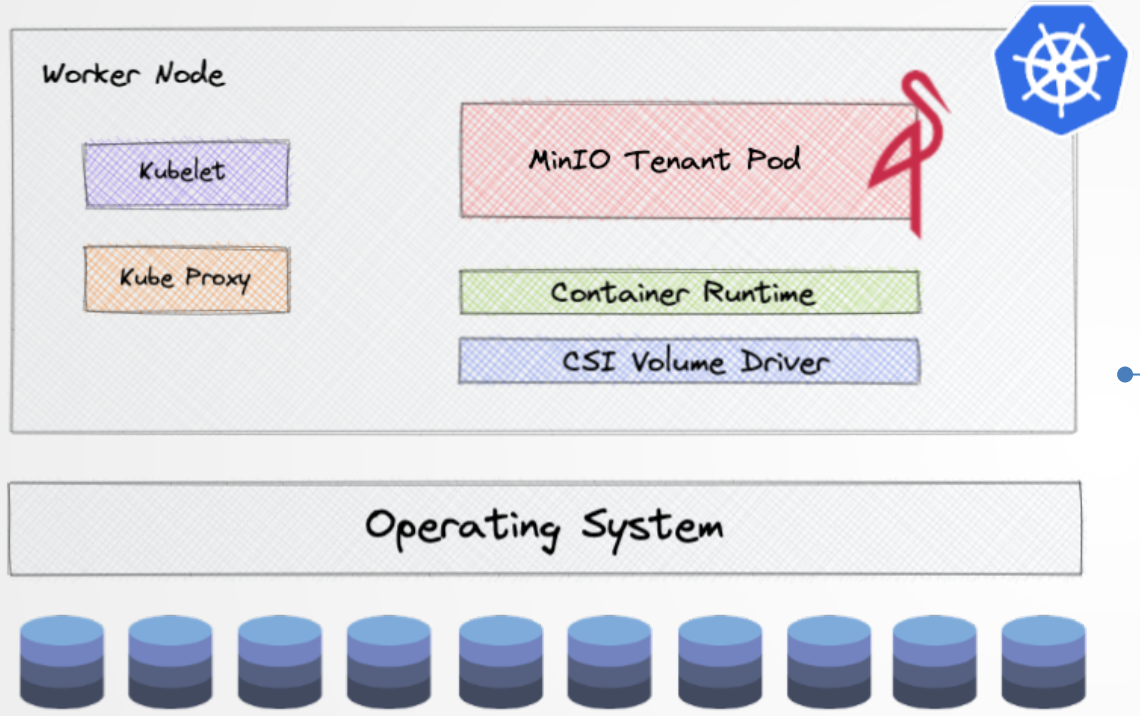
## High Performance

World's Fastest Object Storage Server

Read & Write of **183 GiB/s** and **171 GiB/s** on Standard Hardware

# MinIO and Kubernetes

Deployment Spec for MinIO on Kubernetes,
Architecture Diagram and Killer Features




### Volume Type

Directory Determined
by a Volume Type

* AWS EBS
* Azure Disk
* Google Persistent Disk
* Container Storage Interface

## Features

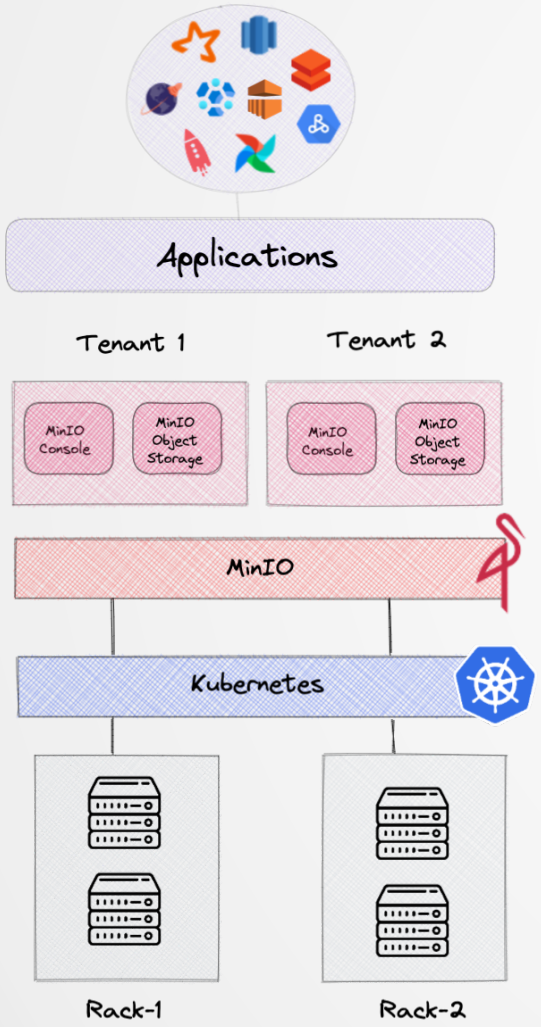
- Active-Active Replication
- Identify & Access Management
- Encryption
- Bucket & Object Immutability
- **Bucket & Object Versioning**
- **Data Lifecycle Management and Tiering**
- Automated Data Management Interfaces
- Monitoring and Scalability

### Container Storage Interface (CSI)

Standard for Exposing Arbitrary Block and File Storage Systems
for Container Orchestration Systems (COs)

# MinIO Operator Architecture

Kubernetes Friendly Object-Storage System for Big Data Workloads with High-Speed Response Time




## Information

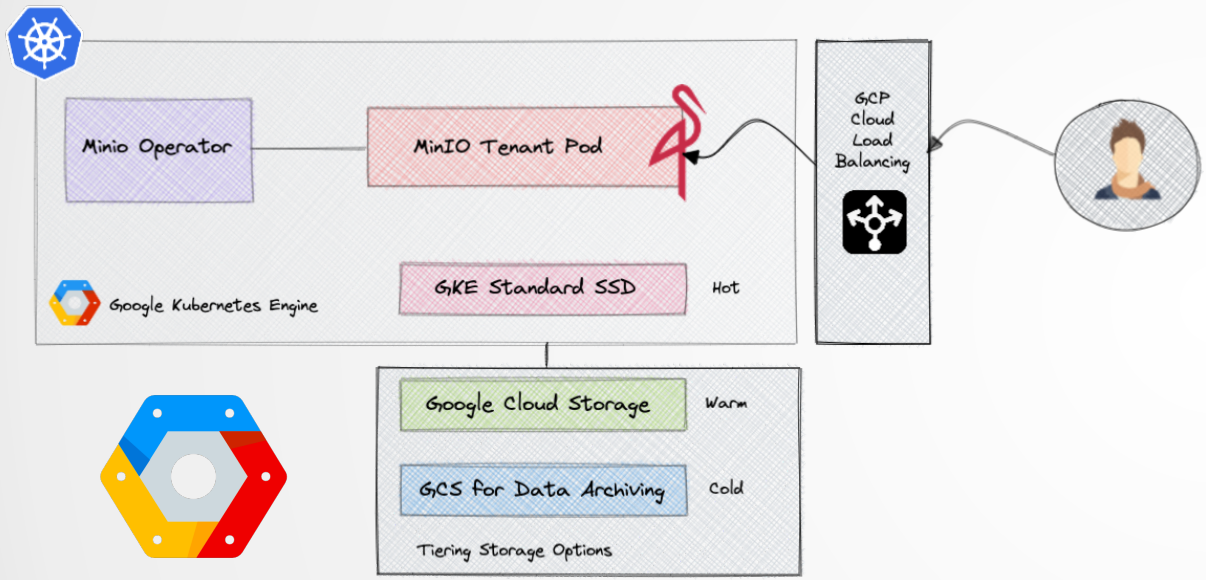
- Kubernetes Native DNA
- **Use of Operator Pattern (CRDs) using Helm Chart**
- Resource Orchestration
- Non-Disruptive Upgrades
- Cluster Expansion
- High-Availability

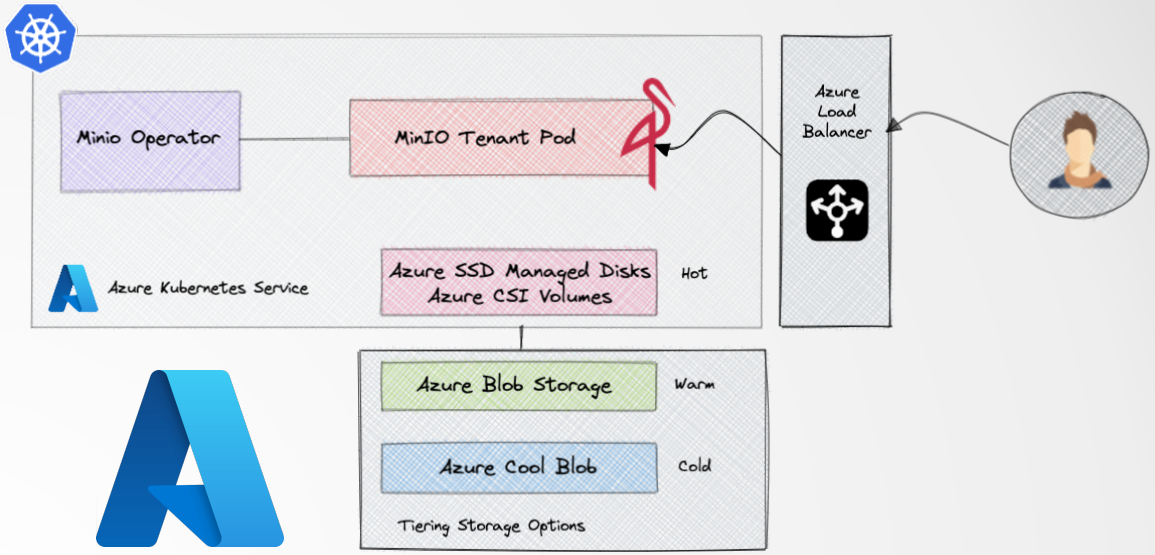
## S3 Select

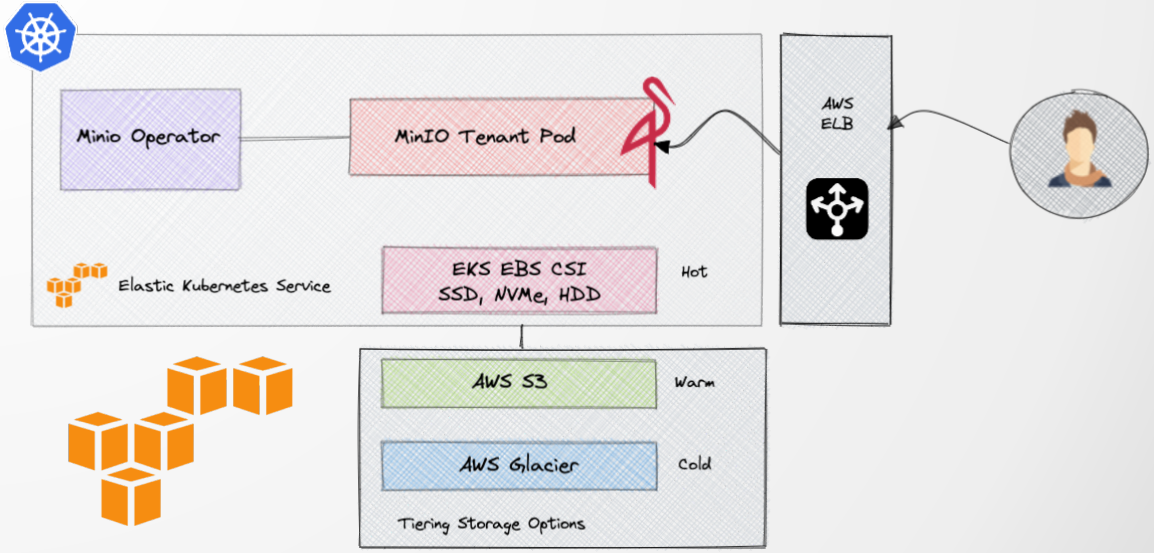
- **SQL** Expressions to Retrieve a Subset of S3 Object
- **SELECT** & **WHERE** Clause to Fetch Data
- CSV, JSON, or Apache Parquet
- GZIP or BZIP2 Compression
- **Python, Spark & Presto Engines**

# MinIO & Managed Kubernetes

Managed Kubernetes Offerings - GKE, AKS & EKS and
MinIO's Tight Cloud Integration



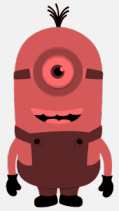



# Deployment Modes

Methods in how to Submit an Apache Spark Application to the Server. Different Modes available

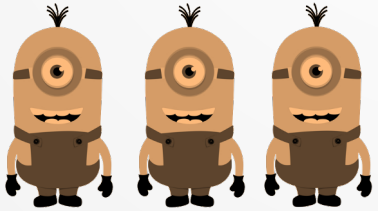
## Local

Runs on a Single JVM ~ Laptop or Single Node, Executor and Cluster Manager Runs on Same Host



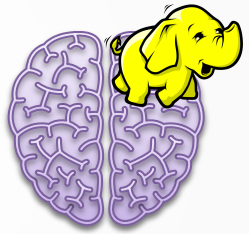
## Standalone

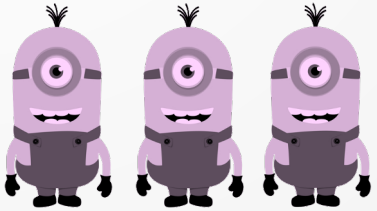
Run on Any Node in Cluster. Each Node Launch Own Executor JVM. Allocated Arbitrarily to Any Host



## Yarn

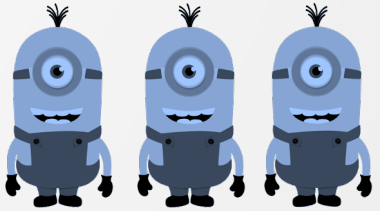
Runs with YARN (Cluster) Application Master. Node Manager & Resource Allocate Containers for Execution





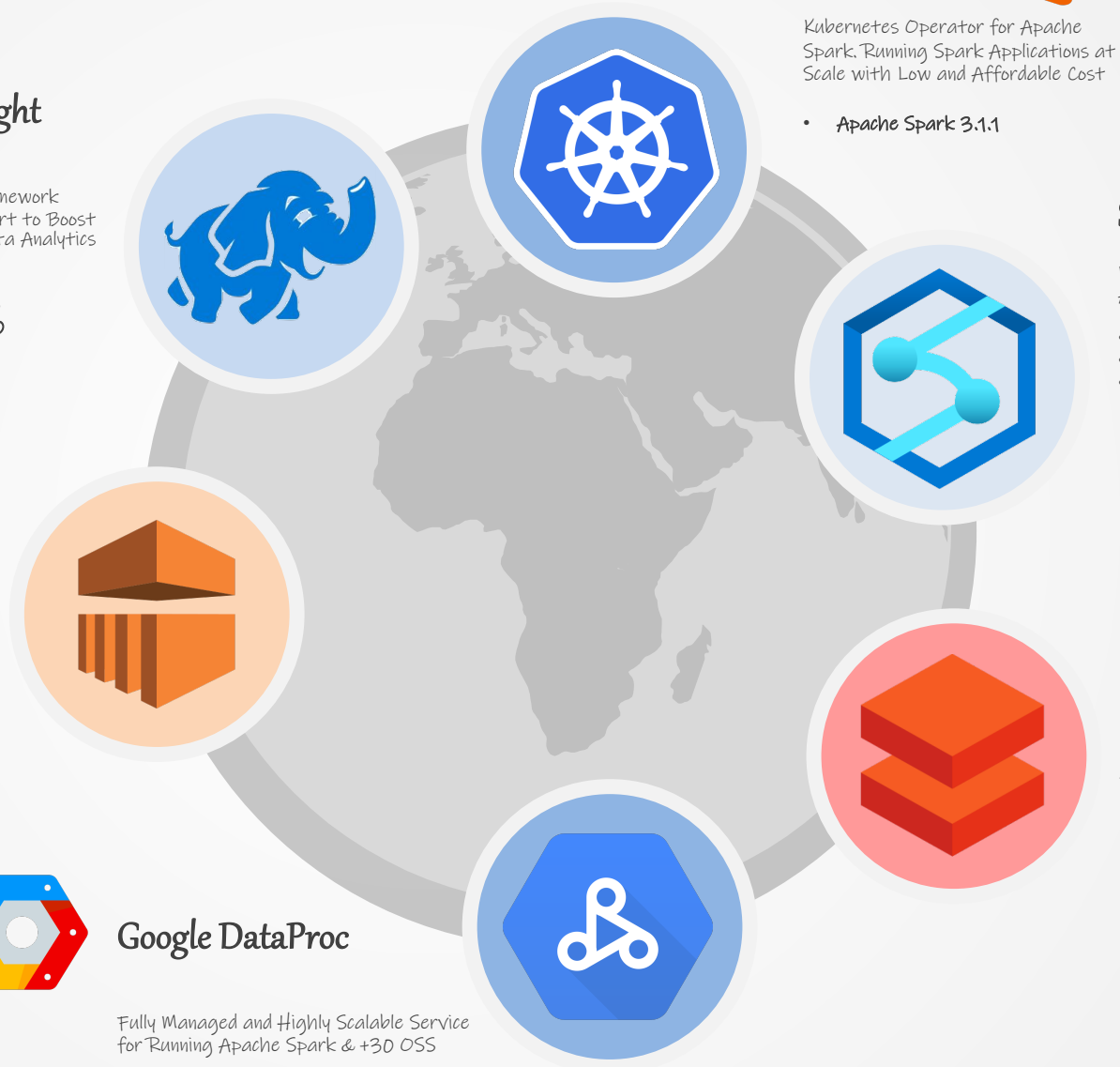
## Kubernetes

Runs in a Kubernetes Pod. Each Worker Runs Within on Pod Context, use Kubernetes Master for Cluster Management



# Deployment Options

Different Options to Deploy Apache Spark Cluster either on Cloud or K8S



## Azure HDInsight

Parallel Processing Framework with In-Memory Support to Boost Performance of Big Data Analytics

- HDInsight 4.0
- Ubuntu 16.0.4 LTS
- **Apache Spark 3.0.0**

## Kubernetes

Kubernetes Operator for Apache Spark. Running Spark Applications at Scale with Low and Affordable Cost

- **Apache Spark 3.1.1**

## Spark Pools

Microsoft's Implementation of Apache Spark in Azure Synapse Compatible with Azure Storage

- **Apache Spark 3.0.0**
- Delta Lake Integration
- .NET for Apache Spark

## Amazon EMR

Industry-Leading Cloud Big Data Platform for Processing Vast Amount of Data Using Open-Source Tools

- EMR 6.3.0
- Linux AMI
- **Apache Spark 3.1.1**

## Databricks

Data + AI Company. Open-Source Apache Spark, Founded in 2013 by Original Creators of Apache Spark

- Apache Spark
- Delta Lake
- MLFlow
- SQL Analytics

- Runtime 8.3
- **Apache Spark 3.1.1**

## Google DataProc

Fully Managed and Highly Scalable Service for Running Apache Spark & +30 OSS

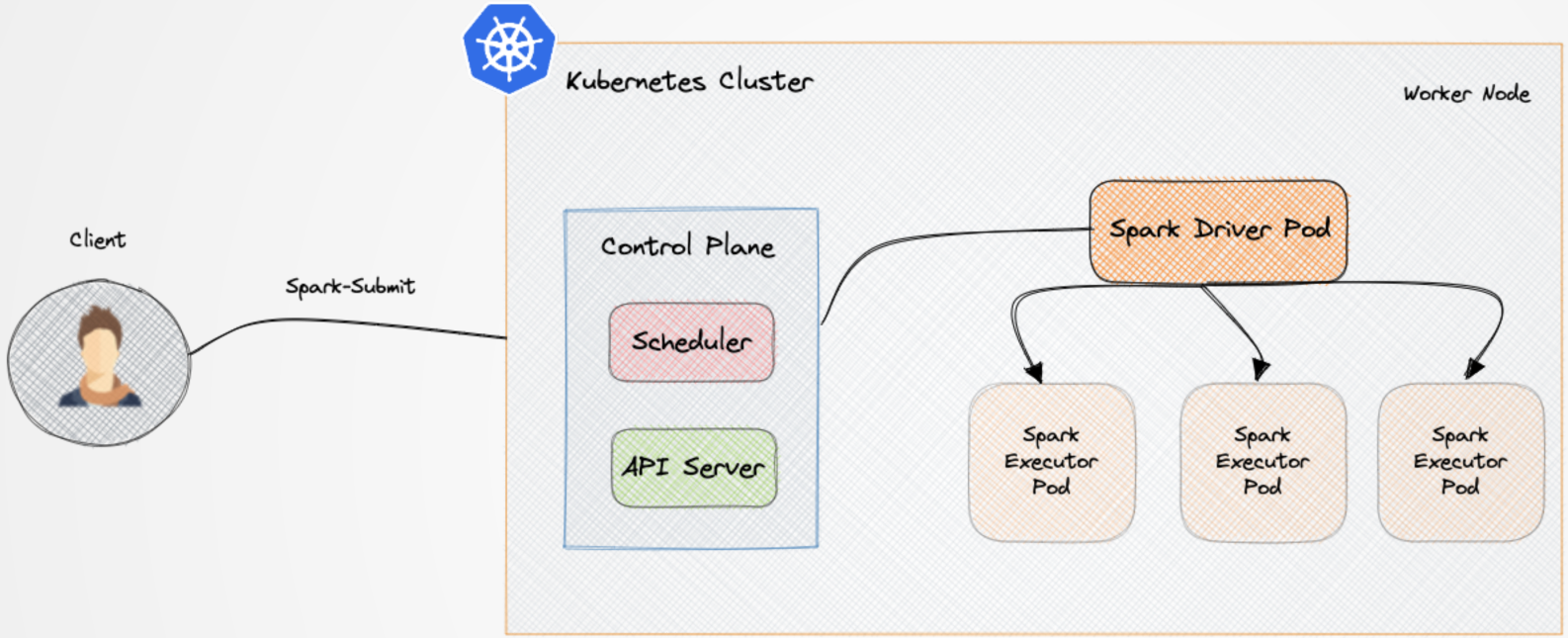
- BORG = Kubernetes
- Debian, Ubuntu & CentOS
- **Apache Spark 3.1.1**

# Spark on Kubernetes

Use of Native Kubernetes Scheduler for Manage Spark Applications in a Lightweight Fashion

Apache Spark GA in 3.1

Typically - 50% to 70% of Cost Savings




## Cost Optimization

- Use of SSDs or Large Disks for Best-Shuffle Performance
- Optimize Pod Sizes to Avoid Wasting Capacity (90%)
- spark.executor.cores=4
  spark.kubernetes.executor.request.cores=3600m
- Enable App-Level Dynamic Allocation

spark.dynamicAllocation.enabled=true
spark.dynamicAllocation.shuffleTracking.enabled=true

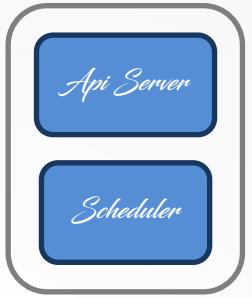
- Enable Cluster-Level Autoscaling
- Use Spot Nodes to Reduce Cloud Costs

# Spark-on-Kubernetes ~ Operator

Operator to Deploy and Manage the Lifecycle Process of an Apache Spark Application on Kubernetes

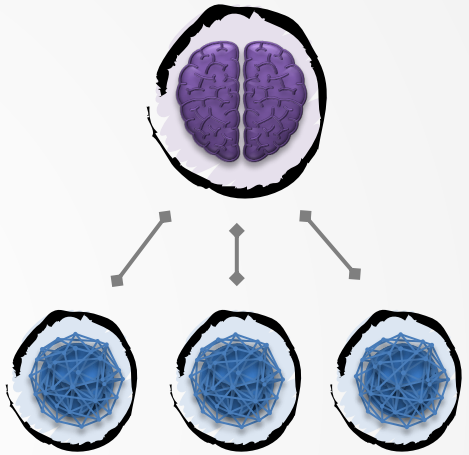
## Spark-Submit

Submit an Apache Spark Program and Launches on Cluster




## Spark Driver

Creates Spark Context, Session, & Executes User Code



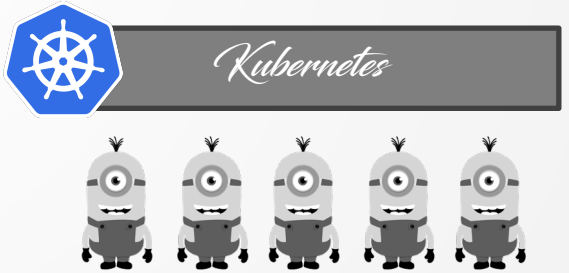
## Spark Executor

Executes Individual Tasks & Return Result Set to Driver

## Spark-on-K8s-Operator

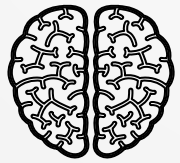
**Kubernetes Operator** ~ Method of Packaging, Deploying and Managing an Application. Extends Functionality to Create, Configure, and Manage Instances of Complex Applications

**Spark-on-K8s-Operator** ~ Aims to Specifying and Running Apache Spark Applications as Easy and Idiomatic as Running Kubernetes Workloads Instances





spark-pi.yaml



## Spark Operator

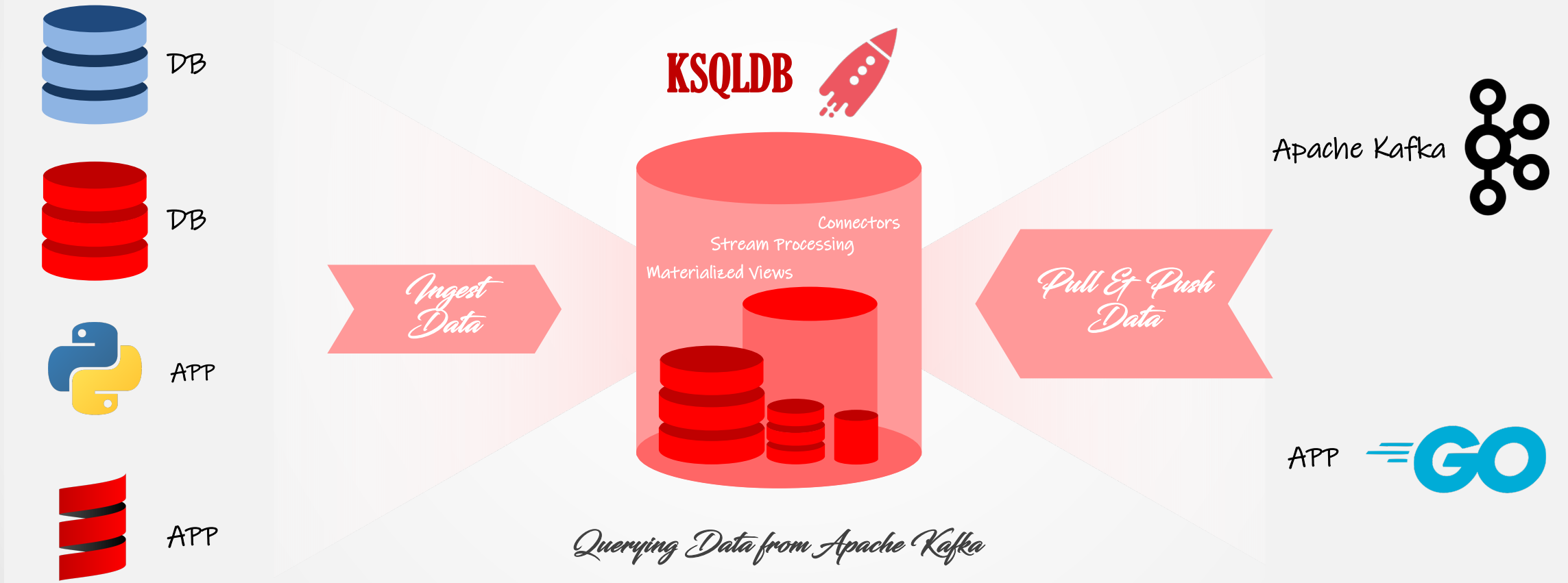
- Controllers
- Submission Runner
- Pod Monitor
- Mutating Admission

## Deployment Mode

Runs in a Kubernetes Pod. Each Worker Runs Within on Pod Context, use Kubernetes Master for Cluster Management

# Single Mental Model for Event Streams

The Database Purpose-Built for Stream Processing Applications
that uses Apache Kafka as Backbone System




*Querying Data from Apache Kafka*

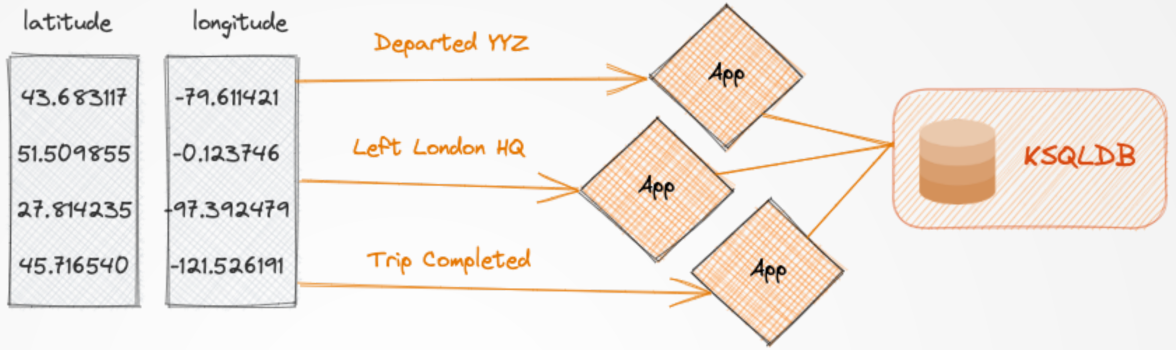
**Build Connectors** = Create Connectors in inside KSQLDB using a Create Statement
**Stream Processing** = Aggregation, Joins Processing Streaming
**Materialized View** = Create Objects from Streaming Processing Results

# Query Modes

Kinds of Query Models: Push, Pull and Persistent. Different Ways to Work with Row of Events in Real-Time

## Push Queries

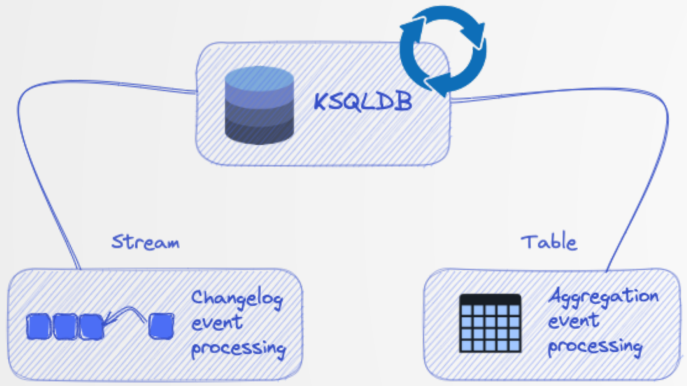
Subscribe to a query's result as it changes in real-time. Emit refinements to react quickly




```
SELECT * FROM pageviews
  WHERE ROWTIME >= 1510923225000
    AND ROWTIME <= 1510923228000
  EMIT CHANGES;
```

## Persistent Queries

Server-Side Queries that runs Indefinitely Processing Rows of Events. Queries derived from Streams and Tables

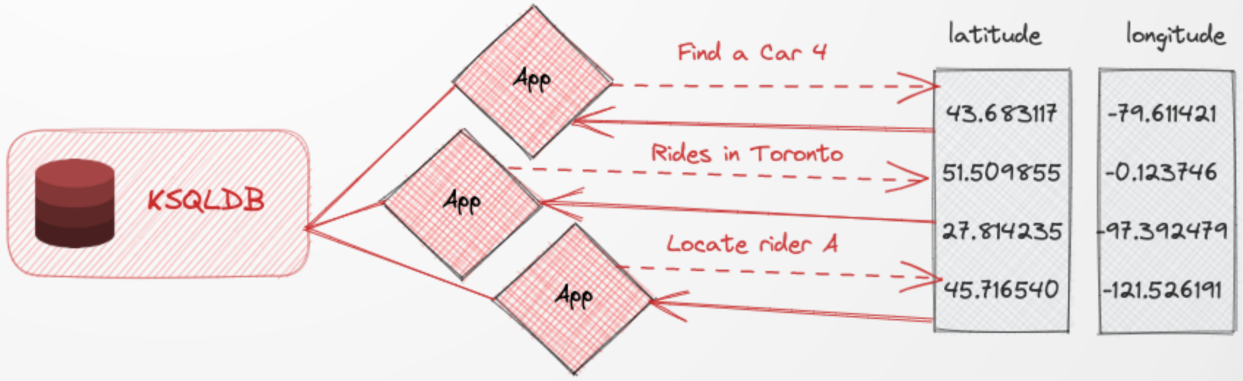



## Pull Queries

Get current values from the Materialized View and Terminates. Result not Persisted in a Kafka Topic

```
CREATE TABLE GRADES
(
    ID INT PRIMARY KEY, GRADE STRING, RANK INT
)
WITH
(
    kafka_topic = 'test_topic',
    value_format = 'JSON',
    partitions = 4
);


SELECT * FROM TOP_TEN_RANKS
WHERE ID = 5;
```

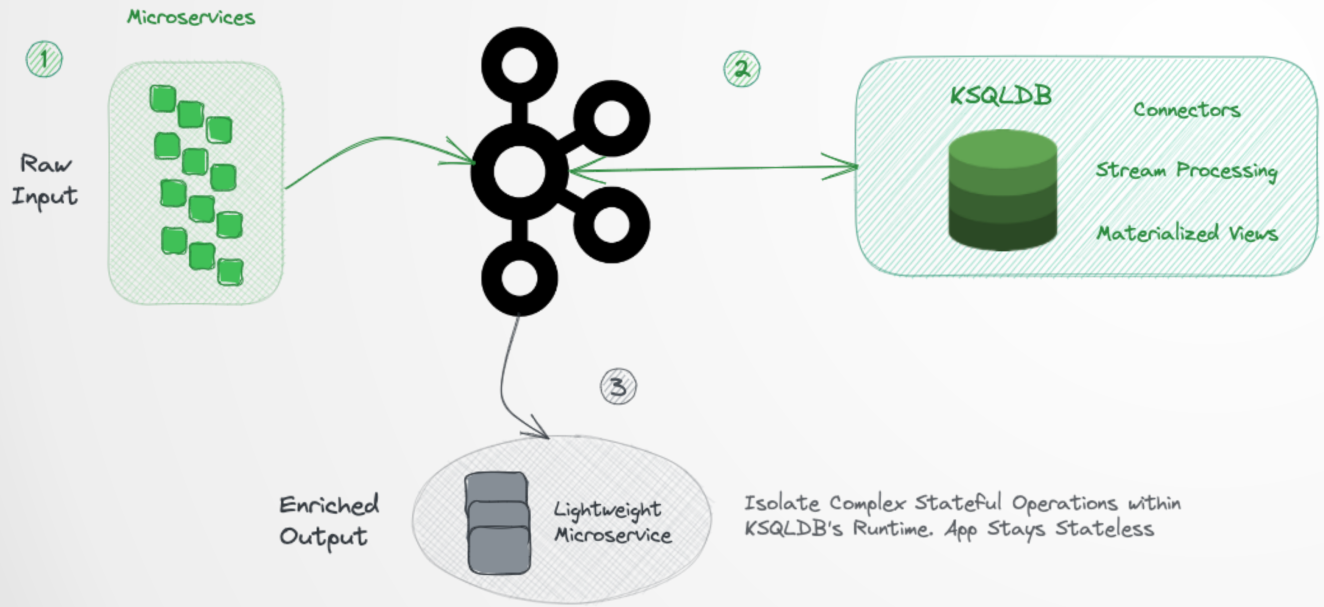


# Use-Cases

Different Approaches in how to leverage the Best of KSQLDB
for Microservices, Big Data and Analytics

## Event-Driven Microservice

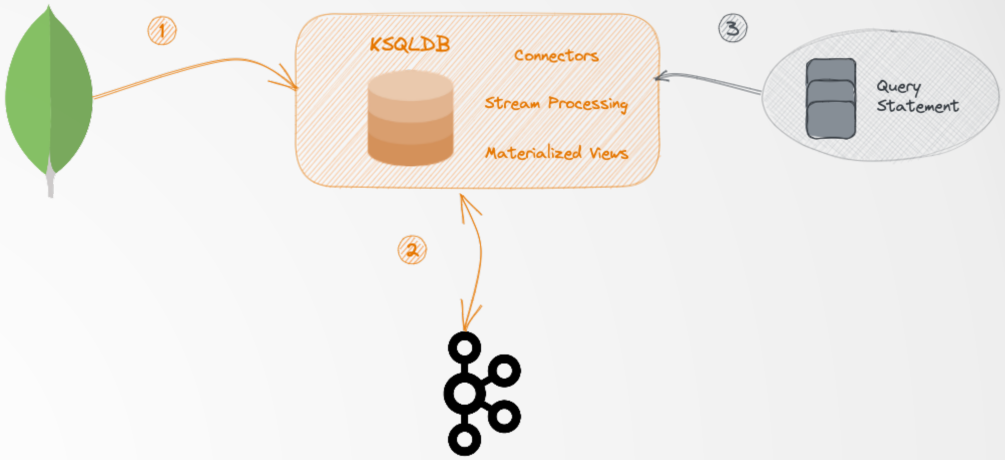
Pattern which a piece of code communicates with the outside world through messages called events. Only receives and emits information defined per a communication channel




Isolate Complex Stateful Operations within KSQLDB's Runtime. App Stays Stateless

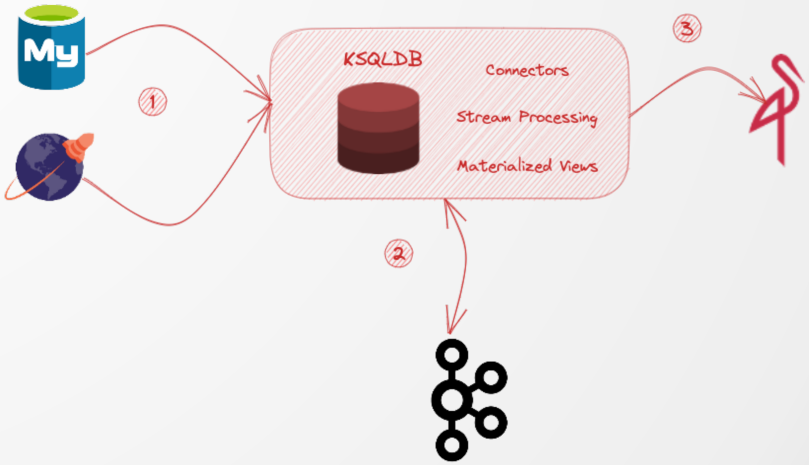
## Materialized Cache

Precomputing results of a query and storing for fast read access
KSQLDB consolidates this complexity by slimming the architecture tp two things
Kafka (Storage) and Compute (KSQLDB)




## Streaming ETL

Gluing all the pieces together, Streamline how you
write and deploy streaming data pipelines by boiling it down to two things
Kafka (Store) and Compute (KSQLDB)



# Features

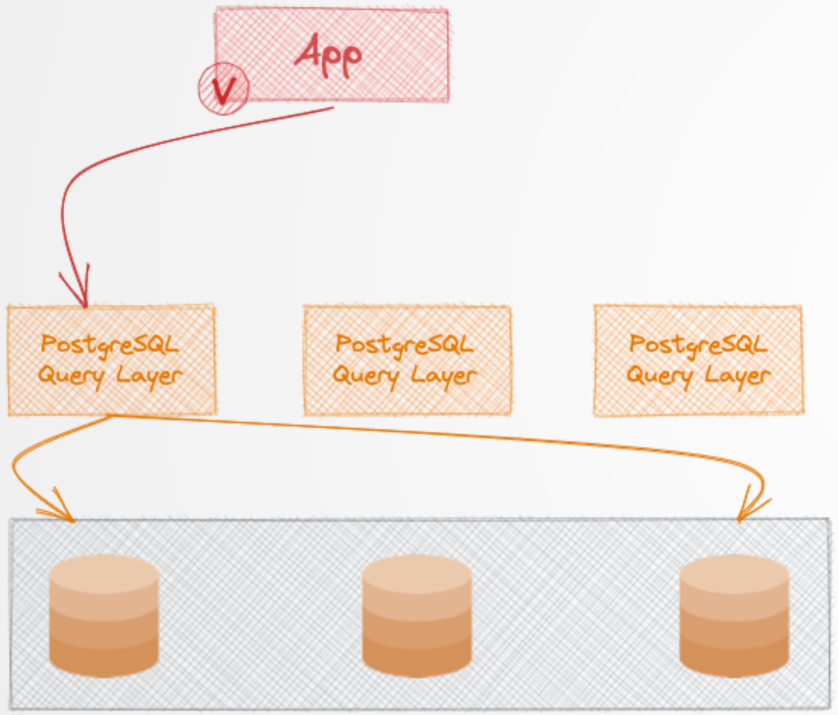
Kubernetes Native Data Platform for RDBMS & NoSQL Databases in a Single Deployment Unit


## Investment

- Series C Funding of **$188** Million, Company Value of $1.3B
- 4K+ Slack Users
- 5.5K+ GitHub Stars
- **1M+** Clusters Deployed
- 220+ Contributors

# Relational Database for Cloud Era

Reimagine the Relational Database for the Cloud. Uniquely Combines Enterprise-Grade Relational Database with Horizontal Scalability




## Characteristics

### PostgresSQL Compatibility

- Wire Compatible
- Query Layer Reuse
- Lift & Shift Migrations

### Advanced Relational Database Capabilities

- Triggers, Functions, SPs, Secondary Indexes
- Distributed ACID Transactions

### Resilience and High Availability

- Node, Zone, Region and Data Center Failures
- Fast Failover with Re-Replicating Data Automatically
- Zero Downtime during Upgrades or Patching

### Horizontal Scalability

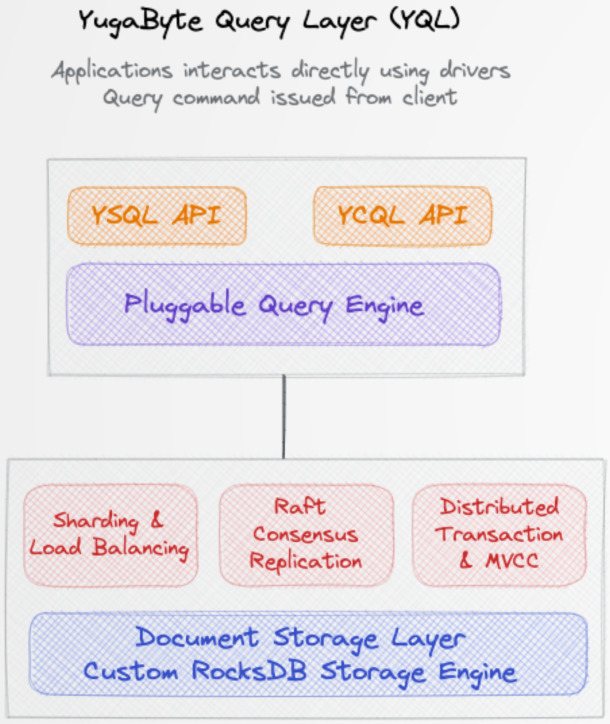
- Scale-Out without Disruption or Downtime
- Adding New Nodes into Cluster
- Automatically Rebalances Load

### Geographic Distribution

- Flexible Deployment Options for Distribution
- Sync and Async Data Replication
- Geo-Partitioning

# Layered Architecture

2 Logical Layers - YugaByte Query Layer and
DocDB - Distributed Document Store




## DocDB Document Store

Distributed Document Store that guarantees:

* Strong write consistency
* Extremely Resilient to failures
* Automatic Sharding and Load Balancing
* Zone, Region, and Cloud Aware Data Placement
* Tunable Read Consistency

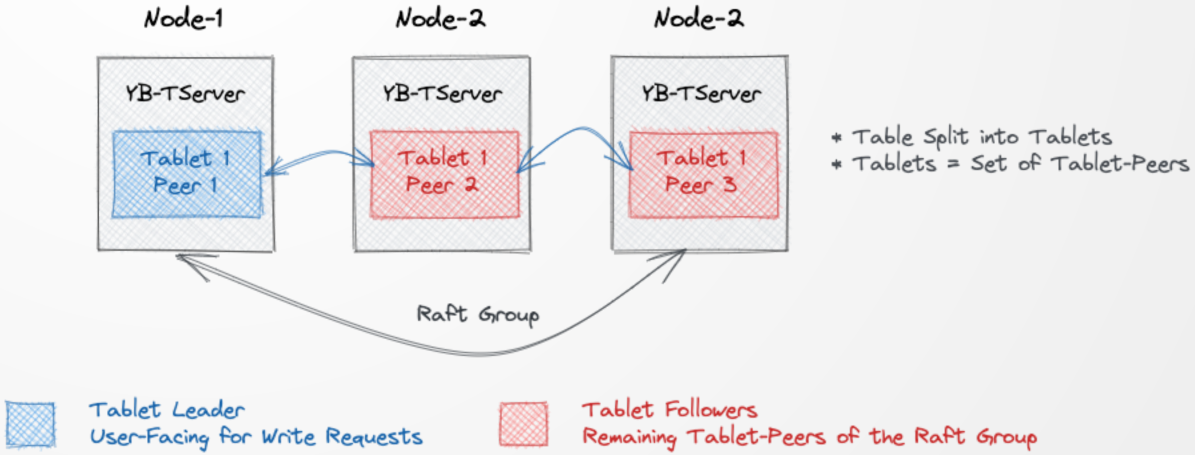
## Characteristics

### YugaByte SQL (YSQL) & Cloud SQL (YCQL)

- **YSQL** = Distributed SQL API Built to Reuse PostgresSQL Language
- **YCQL** = Semi-Relational Language that Roots Apache Cassandra

### DocB

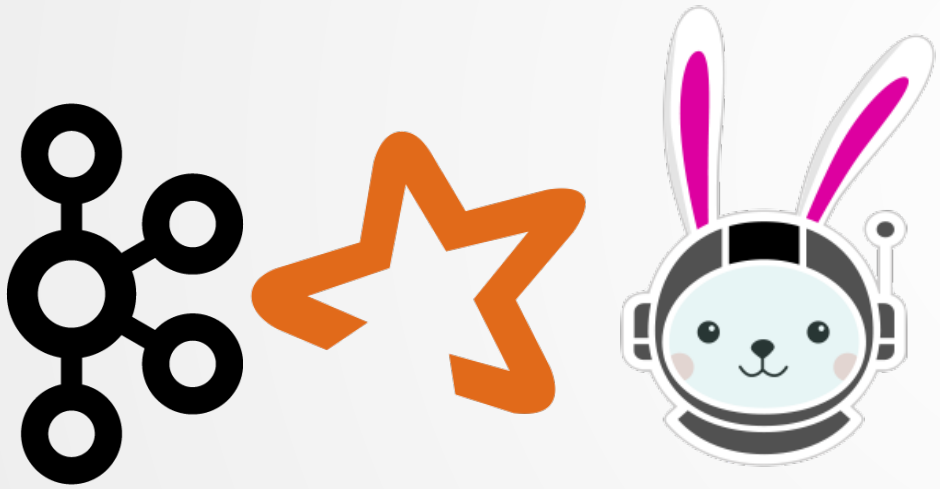
- **Sharding** = Table Sharded into Number of Tablets
- **Replication** = Replication Factor using Raft Consensus Algorithm, Tablet Level
- **Persistence** = Log-Structured Row & Document-Oriented Storage
- **Transactions** = Single-Row and Multi-Row Transactions using ACID Theorem



# YB-Master & YB-TServer

Distributed Architecture on Top of a Relational Database System
Intelligent System for Distributed Transactions

Integrations

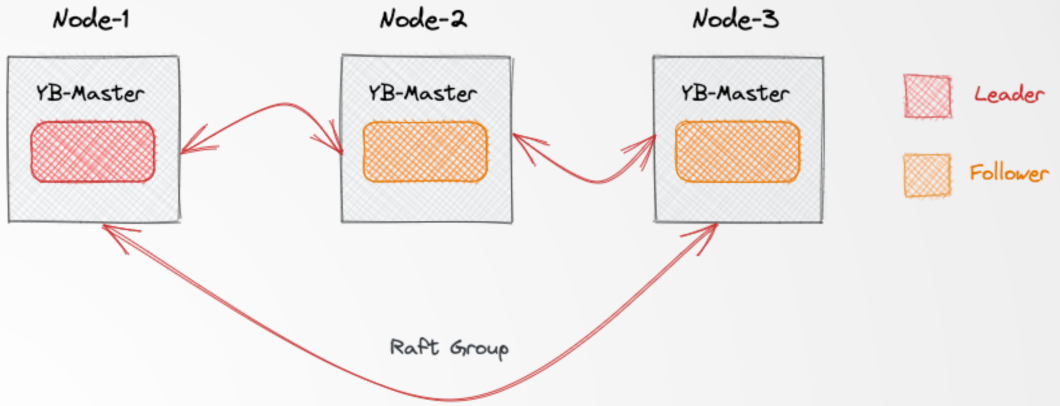


YugaByte JDBC Driver

- Distributed YSQL Built On Postgres JDBC Driver
- Cluster-Aware & Topology-Aware,

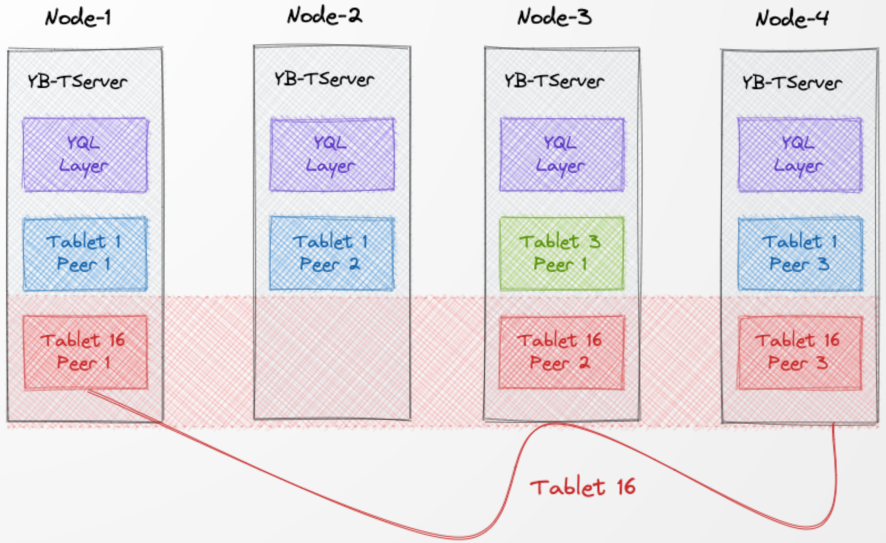
## YB-Master Service

Keeper of Metadata and Records, Responsible for Coordinating Background Ops (LB, Replication) and Administrative Operations




## YB-TServer Service

Tablet Server Responsible for IO Requests. Table is Split or Sharded, into Tablets. Table is composed of One or More Table-Peers, depending of the replication factor stablished



# SQL vs. NoSQL

Best of SQL and NoSQL Databases together into one Unified Platform to simplify development of scalable cloud services

| Characteristics | SQL | NoSQL | YugaByteDB |
|---|---|---|---|
| Data Model | Well-Defined Schema (Tables, Rows, Columns) | Schemaless | Both |
| API | SQL | Various | Fully-Relational SQL + Semi-Relational SQL |
| Consistency | Strong Consistency | Eventual Consistency | Strong Consistency |
| Transactions | ACID Transactions | No Transactions | ACID Transactions |
| High Write Throughput | No | Sometimes | Yes |
| Tunable Read Latency | No | Yes | Yes |

| Characteristics | SQL | NoSQL | YugaByteDB |
|---|---|---|---|
| Automatic Sharding | No | Sometimes | Yes |
| Linear Scalability | No | Yes | Yes |
| Fault Tolerance | No - Manual Setup | Yes - Smart Client Detects Failed Nodes | Yes - Smart Client Detects Failed Nodes |
| Data Resilience | No | Yes - Rebuilds Cause High Latencies | Yes - Automatic, Efficient Data Rebuilds |
| Geo-Distributed | No - Manual Setup | Sometimes | Yes |
| Low Latency Reads | No | Yes | Yes |
| Read Replica Support | No - Manual Setup | No - No Async Replication | Yes - Sync and Async Replication Options |

# Overview & Features

Realtime Distributed OLAP DataStore, Designed to Answer OLAP Queries with Low Latency

## Characteristics

- Blazing Fast OLAP Engine
- LinkedIN, Uber, Target, Slack, Stripe
- Designed for Real-Time Analytics

## History

- Pinot Noir ~ Name of Grape for Wine
- Developed Internally at LinkedIN in 2014
- Open-Sourced in 2015
- Entered in Apache Incubator in 2018
- Top-Level Project in 2021

## Key Capabilities

- Columnar-Oriented Storage
- Pluggable Indexing Technologies
- Horizontally Scale & Fault-Tolerant
- Performs Anomaly Detection using ThirdEye
- Joins using Presto & Trino

## Apache Pinot

Realtime Distributed OLAP DataStore, Designed for Answering OLAP Queries with Low-Latency

## Performance

- 100K + Queries per Second at ms Latency
- Ingestion of Million of Events per Second
- 50+ User-Facing Analytics

## Users

- PQL ~ SQL using Apache Calcite
- API ~ RestAPIs for Broker & Controller
- External Clients ~ JDBC, Java, Python & Go

## Core Components

- Controller (Helix & Zookeeper) ~ Manage State & Health
- Broker ~ Route Queries
- Server ~ Host Segments (Data) ~ Realtime & Offline Tables
- Minion ~ Purge Data

# Components

Major Components and Logical Abstractions used in Apache Pinot Engine

### Real-Time Server

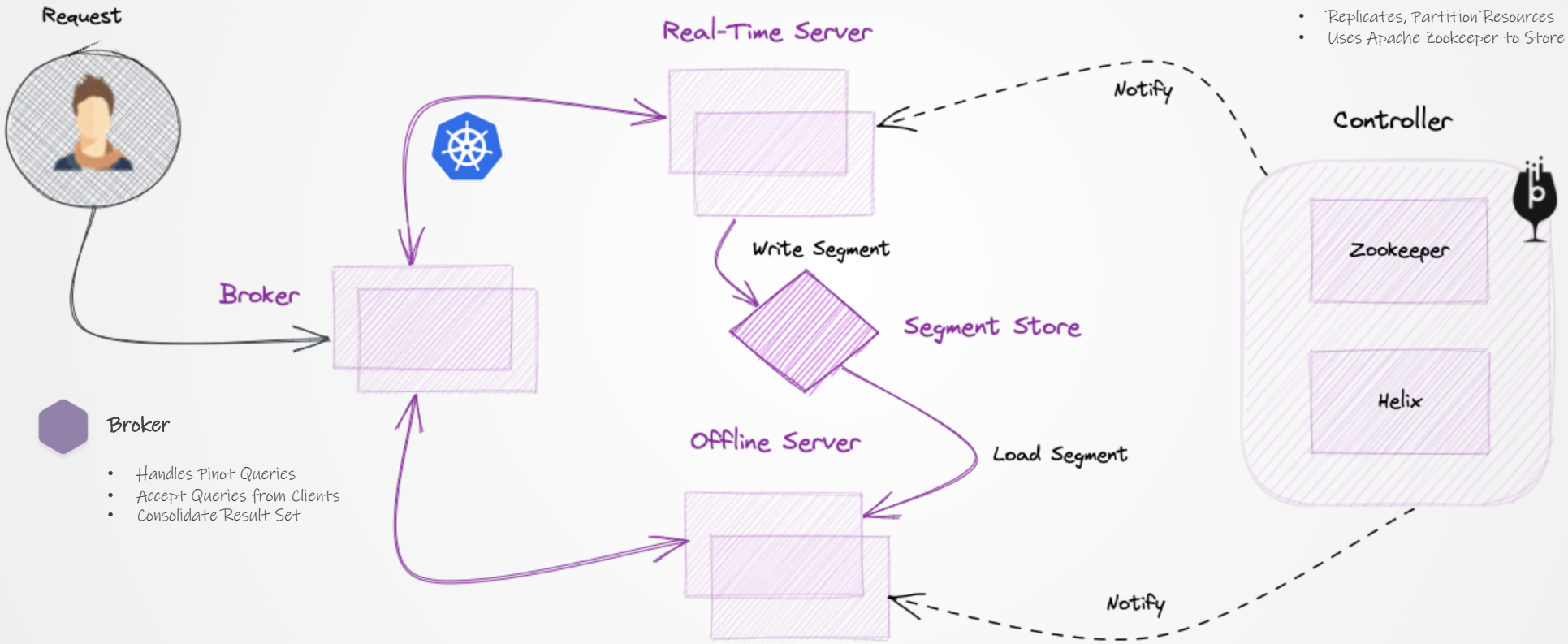
- Directly Ingests from Real-Time Systems
- Kafka, Pulsar, Kinesis
- Segments In-Memory Based on Thresholds
- Persisted on Segment Store

### Controller

- Global Metadata
- Manage Components
- Mapping Segments and Servers
- Admin Endpoint

### Apache Helix & Zookeeper

- Replicates, Partition Resources
- Uses Apache Zookeeper to Store State and Metadata




### Broker

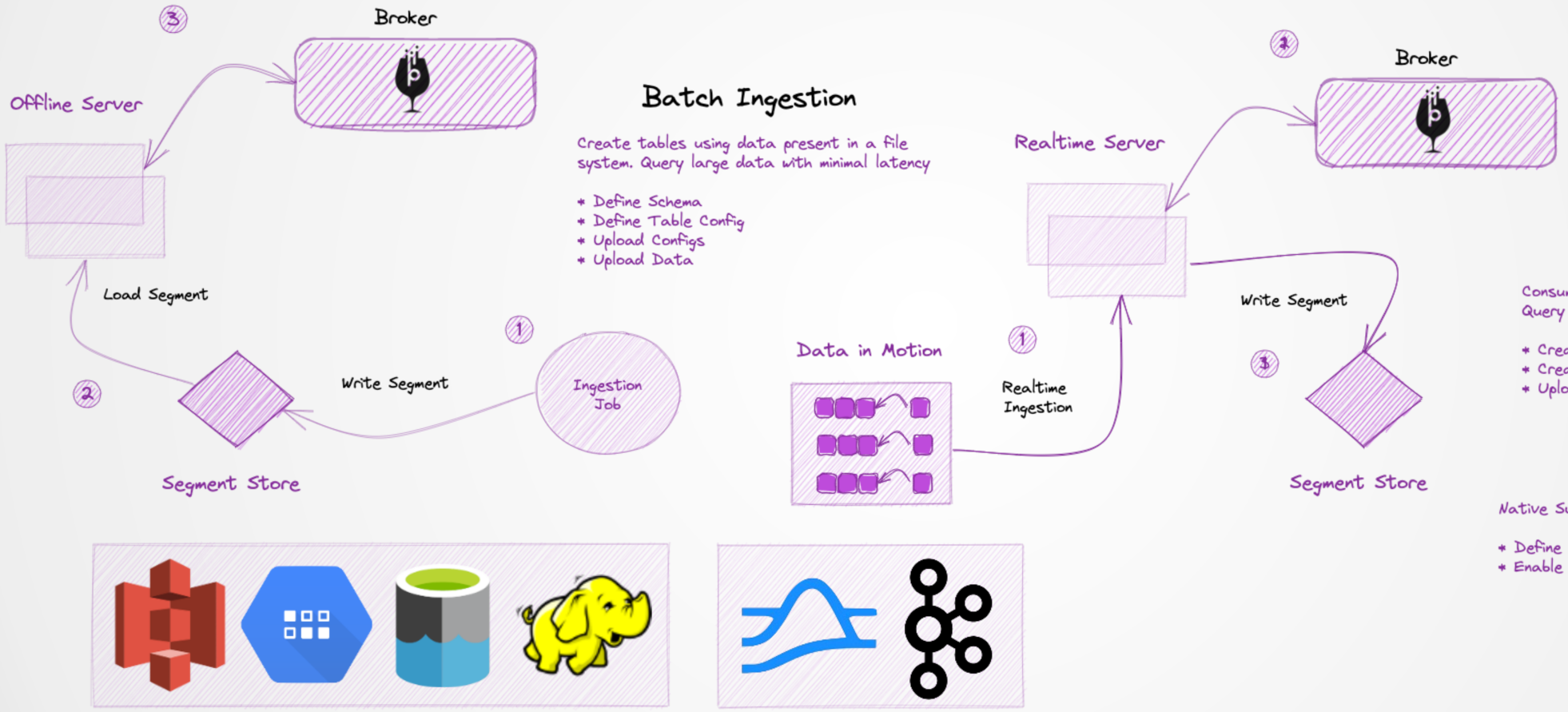
- Handles Pinot Queries
- Accept Queries from Clients
- Consolidate Result Set

### Offline Server

- Downloading Segments from Store
- Host and Server Queries Off

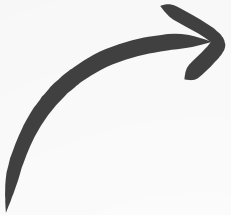
# Ingestion Modes

Batch and Stream Ingestion Processes of
Apache Pinot Real-Time OLAP Engine



# Data Virtualization

Access Data in Any Source, Shape or Structure without Copying from Source

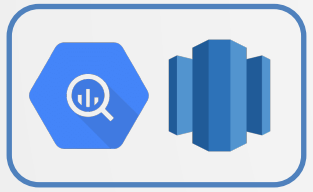


## Processing Query Engine

Connects, Discover, Plan and Analyze Different Data Stores for Query at Scale

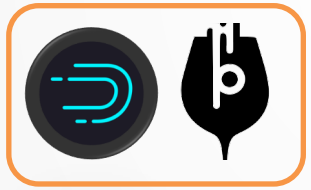
## Modern Data Warehouses

Redshift, BigQuery



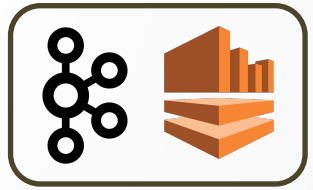
## Real-Time OLAP

Apache Druid & Apache Pinot



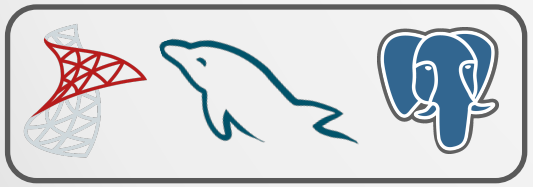
## Real-Time Ingestion

Apache Kafka, Kinesis



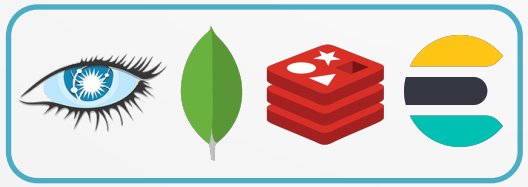
## Relational Databases

SQL Server, MySQL, Postgres



## NoSql

Cassandra, MongoDB, Redis, ElasticSearch



## Data Lake

HDFS, S3, Blob Storage, GCS & MinIO [S3*]

# Overview & Features

Fastest Bunny on Forest, Eating
Any Source of Data

## Trino

- Speed & Scale
- Simplicity
- Versatile
- In-Place Analysis
- Query Federation
- Runs Everywhere
- Trusted
- Trino Software Foundation [OSS]

## Designed For

- Access Data using SQL
- Data Warehousing
- Analytics
- OLAP

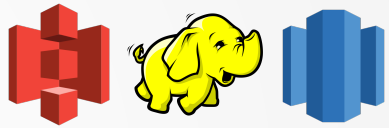
## Features

- Memory Management & Spilling using Compression and Encryption
- Resource Groups
- Distributed Sort
- Dynamic Filtering ~ Query Performance, Network Traffic Reduction & Remote Load Reduction
- Cost Based Optimization - CBO
- Query Pushdown

Trino is a Tool Designed to Efficiently Query Vast Amounts of Data using Distributed Queries in a Terabytes & Petabytes Range

## Query Execution Model

Turn SQL Statements into Queries, Executes Across a Distributed Cluster of Coordinator and Workers

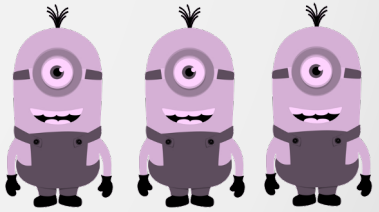


## Coordinator a.k.a The Brain

Parsing Statements, Planning Queries, & Managing Trino Worker Nodes

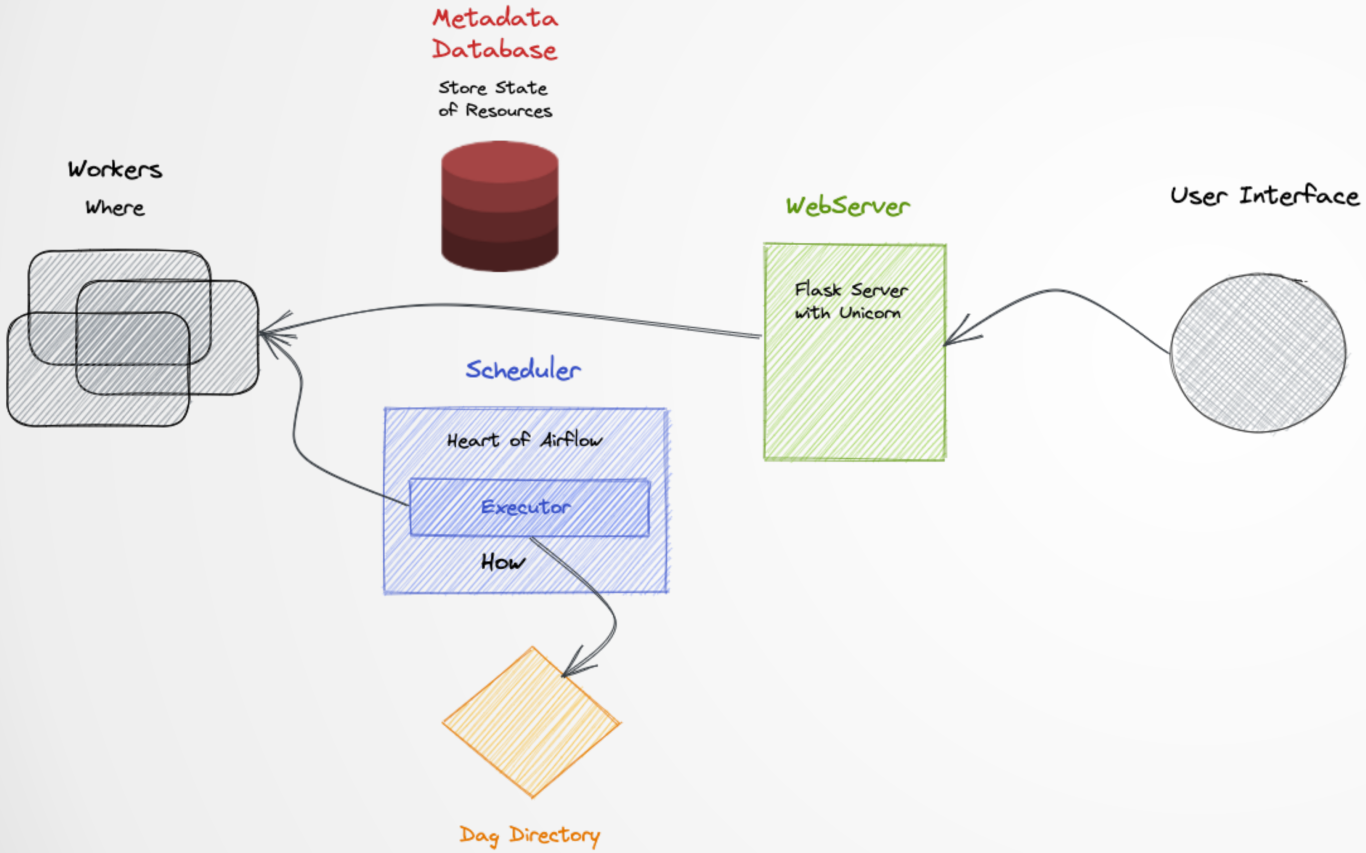
## Workers a.k.a The Slave

Responsible for Executing Tasks & Processing Data. Fetch Data from Connectors and Exchange Intermediate Data in Parallel

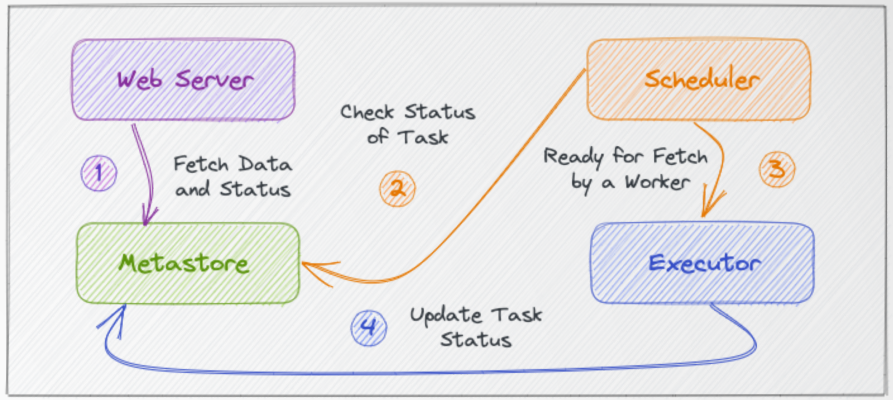


# Core Components & Architecture

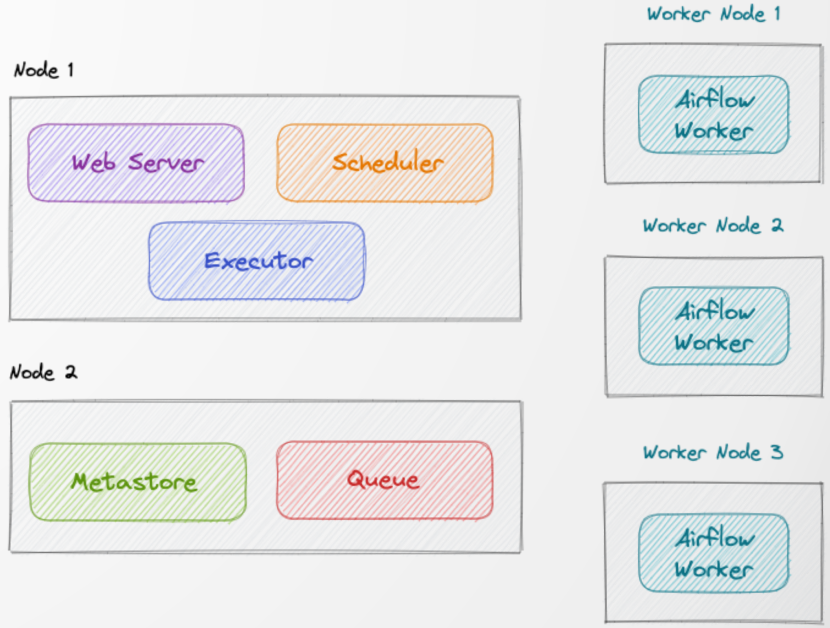
Architecture that Consists of Several Scalable Components to Execute DAGS




## Single Node Architecture




## Multi Nodes using Celery



# Executor Types & Astronomer Registry

Different Flavors in how to Provision and Work with Apache Airflow in Dev and Production Environments

## Local Executor

- Execute Tasks in Separate Processes
- Single Machine
- Non-Distributed Execution
- For Small Deployments and Dev Environment

## Celery Executor

- Most Popular
- Uses Celery Queue System
- Deploy Multiple Workers
- Reads from Queue System (Redis or RabbitMQ)
- Compute Heavy Operations
- Used for Production Environments

## Kubernetes Executor

- Widely Used for Production-Ready Executor
- Spawn a New Pod with Instance
- Can Cause Additional Overhead for Short Running Tasks

## Celery & Kubernetes Executor

- Uses Celery & Kubernetes Combined
- Use Special Kubernetes Queue
- Specify Particular Tasks
- Advantage of Horizontal Auto Worker Scaling
- Delegate Long Running and Heavy Tasks to Executor

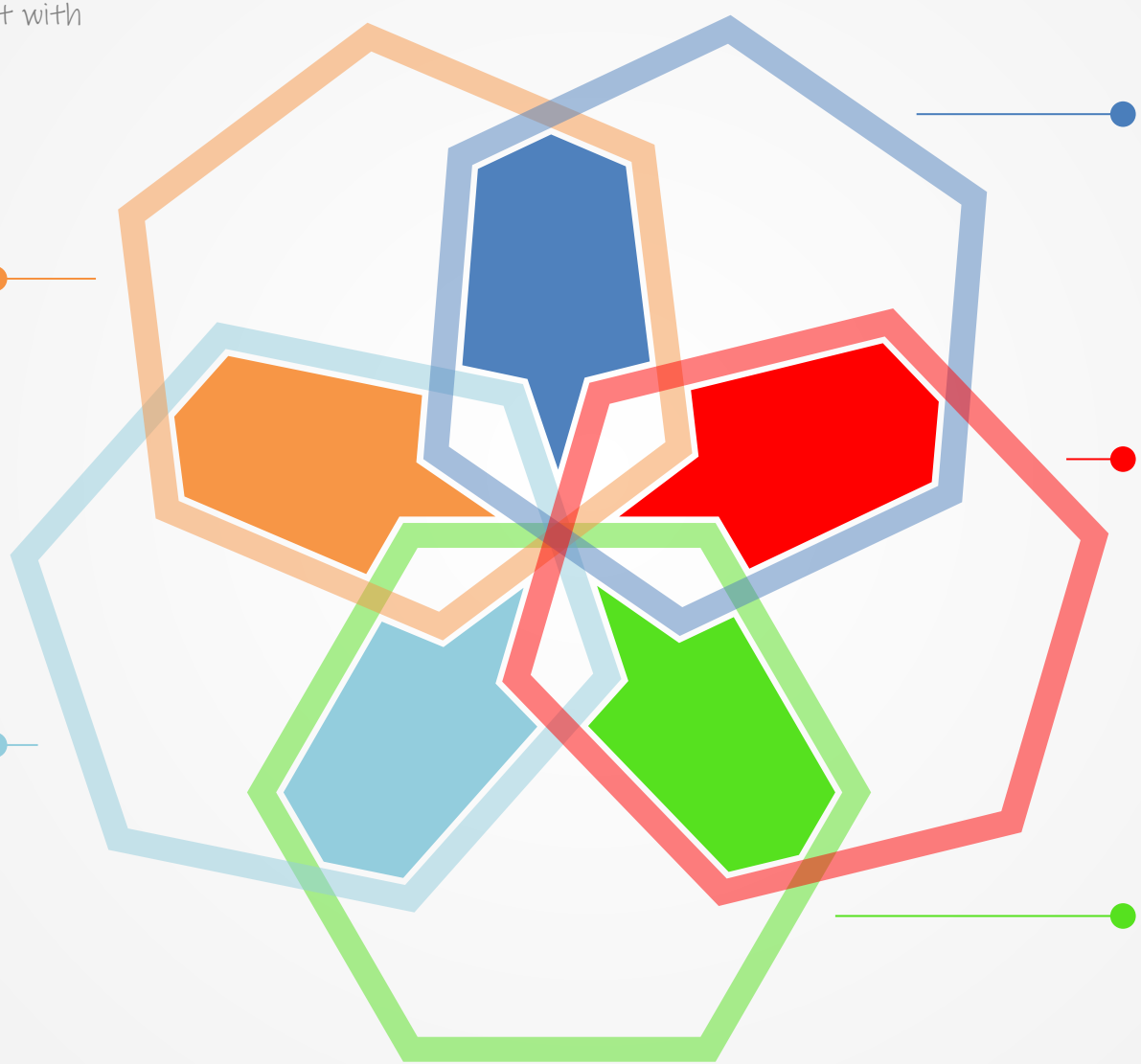
## Astronomer Registry

Building Blocks for Your Apache Airflow Data Pipelines

- Providers
- Modules
- DAGs

# Airflow on Kubernetes

Official Deployment for Airflow on Kubernetes use of Helm Chart for Easy Development and Deployment with Kubernetes Native-Features



## Auto Scaling

Using Celery Executor with Auto Scaler of KEDA Embedded

## Build Images

Build Images to Add Extra Packages and Resources

## Bullet Proof

Fully Integrated with Kubernetes Engine Helm Chart Designed by Astronomer Team

## Backend Databases

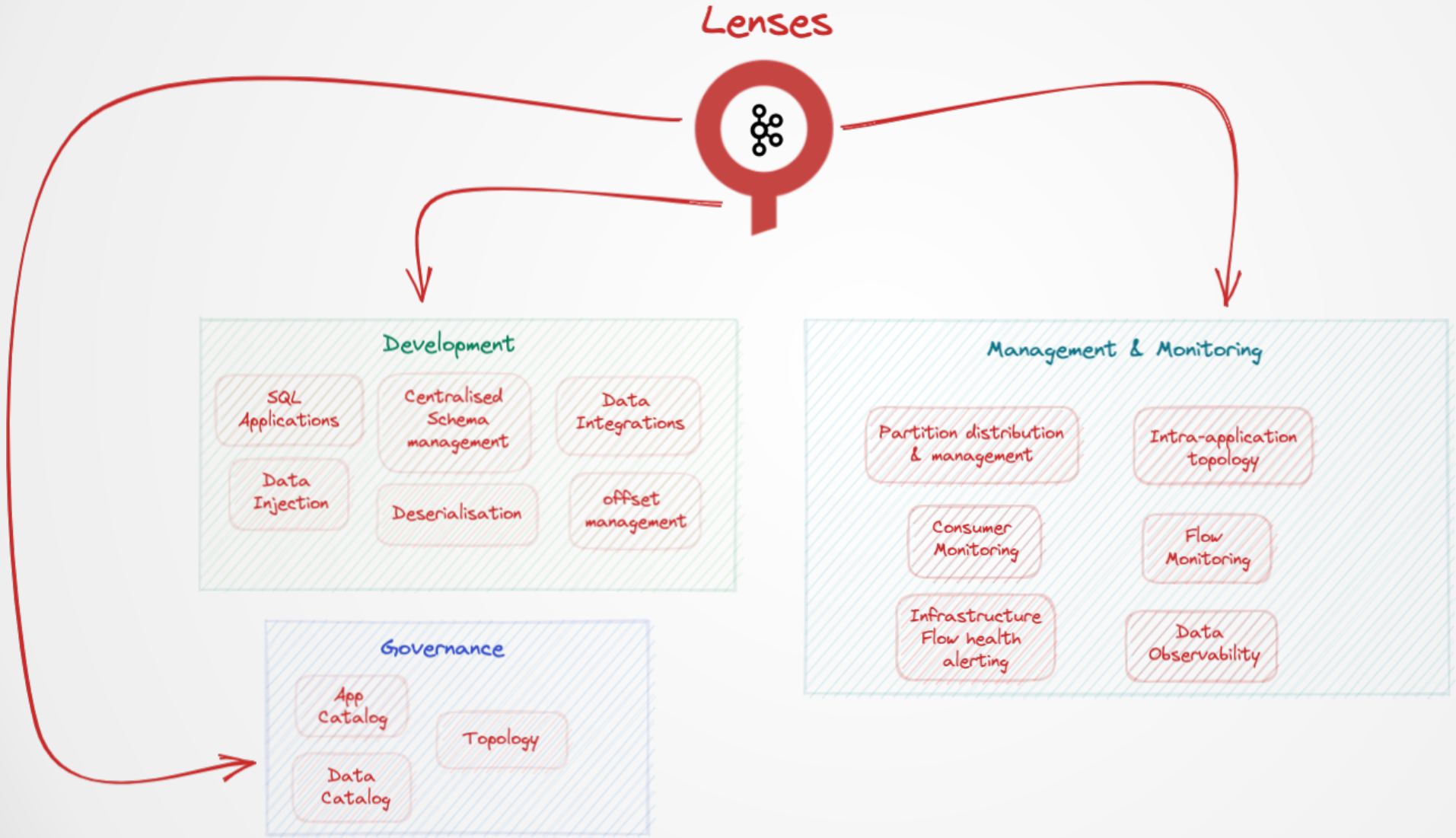
- Postgres (Recommended)
- MySQL
- SQLAlchemy

## Supported Executors

- Local Executor
- Celery Executor
- Celery Kubernetes Executor
- Kubernetes Executor

# Overview & Features

Real-Time Data without PHD in Apache Kafka



# Lenses on Kubernetes

Enabling Lenses for Monitoring and Watchdog Apache Kafka Installations on Kubernetes