HANOI UNIVERSITY OF
SCIENCE AND TECHNOLOGY

A Scalable Lakehouse for Financial Analytics

Big Data Storage and Processing Capstone Project

Trinh Hoang Anh
Bui Khac Chien
Bach Nhat Minh
Tran Quang Minh
Vo Ta Quang Nhat

Supervised by: Dr. Tran Viet Trung

Table of Contents

1. The Challenge: Taming Financial Data

Financial markets generate data at extreme volume, velocity, and variety, creating significant engineering challenges that demand a unified architecture.

Heterogeneous Sources

Integrating real-time streams (Finnhub quotes) with massive historical datasets (Stooq prices, GDELT global news).

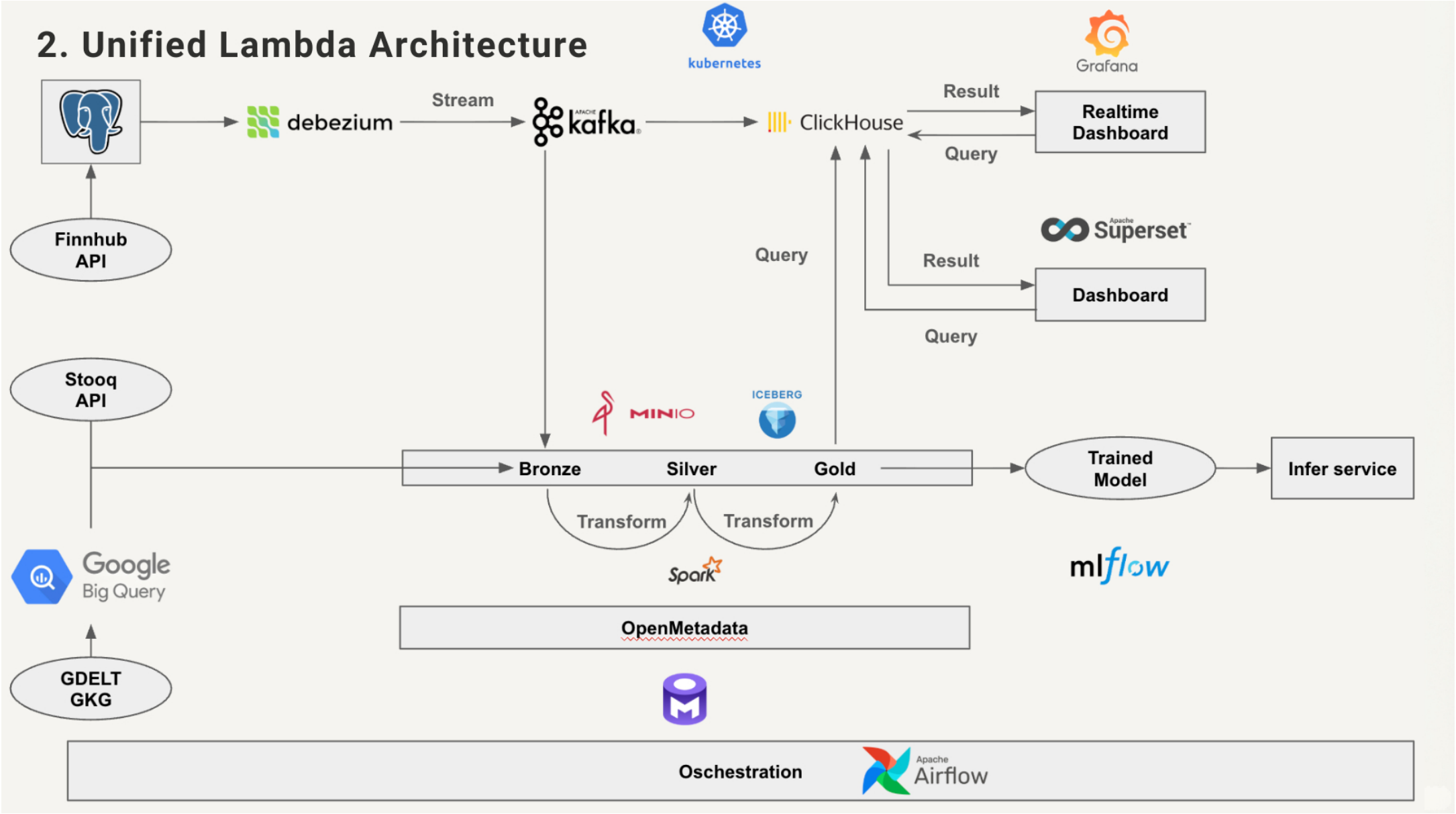
Dual Analytical Demands

Supporting low-latency dashboards for real-time monitoring **and** deep, historical analysis for ML models.

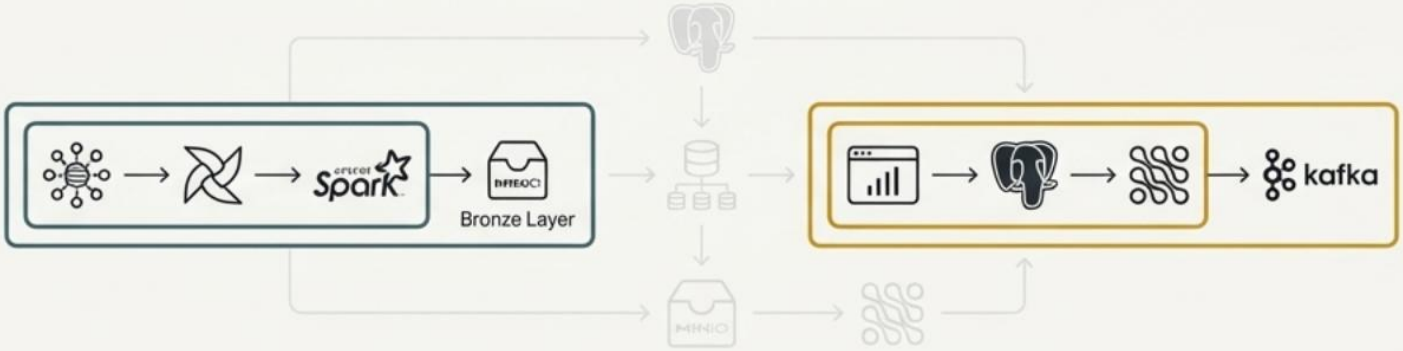
Data Integrity & Governance

Ensuring consistency, quality, and traceability across a massive, evolving dataset from raw ingestion to aggregated insights.

Project Mission: To design and implement a scalable Big Data Lakehouse capable of ingesting, processing, and analyzing financial data in both real-time and batch modes.

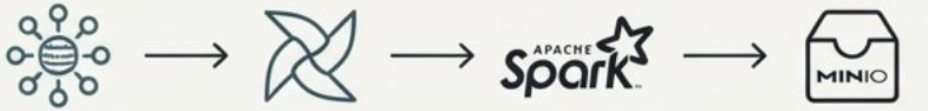

2. Unified Lambda Architecture
kubernetes
Grafana
debezium
Stream
kafka
ClickHouse
Result
Query
Realtime Dashboard
Finnhub API
Apache Superset
Query
Result
Dashboard
Query
Stooq API
MINIO
ICEBERG
Bronze
Silver
Gold
Trained Model
Infer service
Transform
Transform
Spark
mlflow
Google Big Query
OpenMetadata
GDELT GKG
Oschestration
Apache Airflow

2a. Ingestion: A Hybrid, Purpose-Built Strategy



Batch Layer (High Throughput)

Sources: GDELT, Stooq.



Principle: A 'Direct-to-Lake' approach bypasses relational databases to maximize throughput and avoid I/O bottlenecks for bulk historical data.

Speed Layer (Low Latency & Integrity)

Source: Finnhub API.

Principle: Uses PostgreSQL as a transactional buffer for data persistence deduplication. Debezium's Change Data Capture on the database logs creates a real-time, event-driven stream without impacting the source.

2b. Processing: The Medallion Lakehouse on Spark & Iceberg

Bronze (Raw)
Immutable, raw data from sources (GDELT, Stooq) stored as-is. Partitioned by date for efficient writes. The single source of truth for reprocessing.

Silver (Cleaned & Enriched)
Contains validated, structured, and deduplicated data. Key transformations include schema enforcement, temporal normalization (UTC to EST), and linking news events to specific stock tickers.

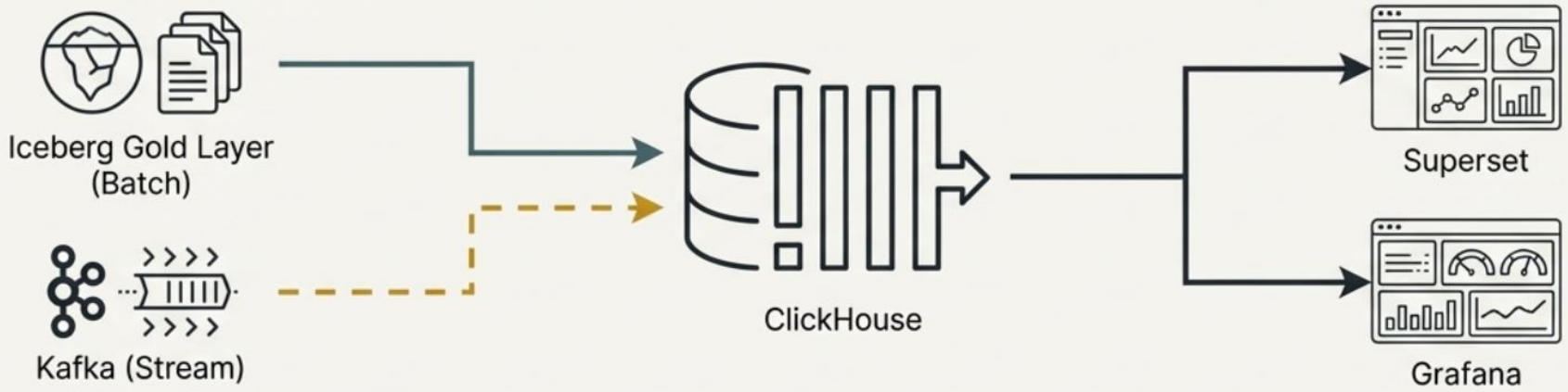
Gold (Aggregated)
Business-level aggregates optimized for analytics and ML. Examples: daily sentiment scores, news volume, and volatility metrics per ticker.

Key Enabler - Apache Iceberg

Stored on MinIO, Iceberg provides ACID transactions, schema evolution, and time-travel directly on the data lake, combining data warehouse reliability with data lake scalability.

2c. Serving: Unifying Batch and Stream with ClickHouse

ClickHouse provides a single, high-performance OLAP access point for both deep historical data and low-latency real-time streams.



Integration Strategies:

- **Batch Data:** Utilizes the Iceberg Table Engine for **zero-copy queries** directly on Parquet files in MinIO. Aggregated data is also loaded into `MergeTree` tables for maximum dashboard performance.
- **Stream Data:** Ingests high-throughput streams from Kafka into optimized `MergeTree` tables designed for time-series queries.

Outcome: Sub-second query performance for interactive dashboards and immediate visibility of market data.

3. The Stack: A Containerized & Cloud-Native Foundation

The entire system is built on a containerized, cloud-agnostic infrastructure, enabling reproducible and scalable deployments.

Deployment & Orchestration

Processing & Storage

Machine Learning

Workflow & Metadata

Streaming & Database

Core Principles: Infrastructure as Code (IaC) • Scalability • Fault Tolerance

4. Notable Lesson #1: Hybrid Ingestion is a Necessity, Not a Choice

Problem

Clash Grotesk Medium

A "one-size-fits-all" ingestion approach created severe bottlenecks.

Routing massive batch files (GDELT) through PostgreSQL caused I/O overhead, while writing API streams directly to the lake risked data duplication.

Solution

Clash Grotesk Medium

For Batch

Direct-to-Lake

Spark writes directly to MinIO, bypassing transactional overhead for maximum throughput.

For Stream

Transactional Buffering

API -> Postgres -> CDC, using the database for its strengths in enforcing integrity and deduplication before streaming.

Key Takeaway

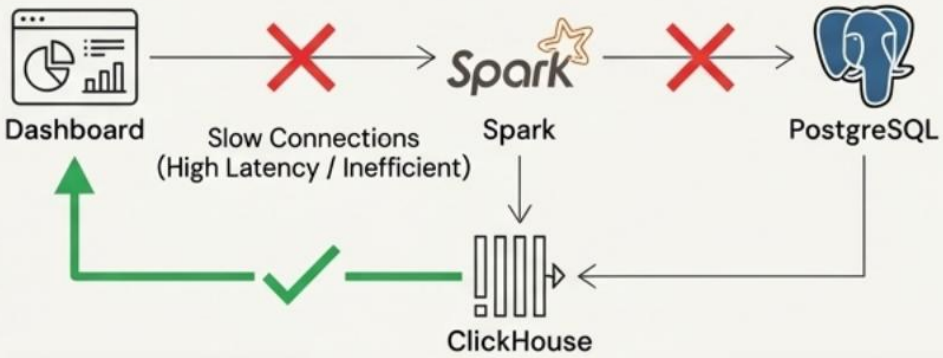
Don't put a database in the middle unless you have a transactional reason. Use raw object storage for bulk throughput.

4. Notable Lesson #2: Separate Processing from Serving for Low-Latency BI

Problem

Dashboards were sluggish and timed out. Connecting Superset directly to Spark was slow (high latency), and using PostgreSQL for large-scale aggregations was inefficient (row-oriented).

Solution



Dedicated OLAP Serving Layer

Implemented **ClickHouse** as a dedicated **OLAP serving layer**. Its columnar storage and vectorized query execution are purpose-built for the fast aggregation queries required by BI dashboards, delivering sub-second responses.

Key Takeaway

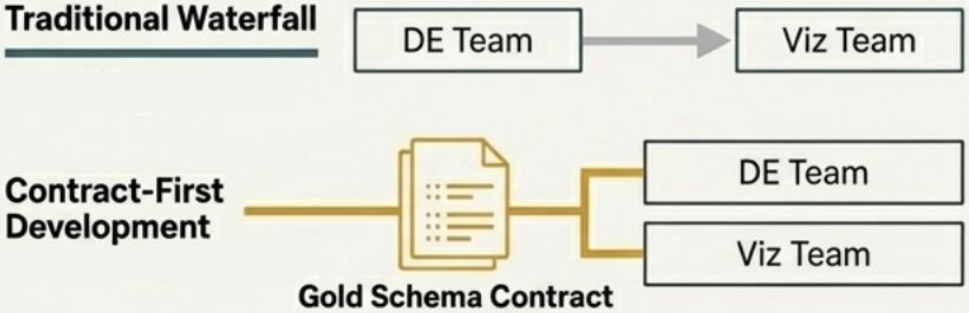
Spark is for throughput (ETL processing). ClickHouse is for latency (interactive serving). Use the right engine for the query pattern. Satoshi Regular

4. Notable Lesson #3: Data Schemas as Contracts Enable Parallel Work

Problem

The visualization team was blocked, waiting for the data engineering pipeline to be completed. This forced a slow, sequential "waterfall" process.

Solution



Adopted **Contract-First Development.** The teams agreed on the final "Gold" layer data schema (column names, types, format) upfront. The visualization team built dashboards using mock data matching this contract, while the DE team built the pipeline to produce it. Integration required zero changes to the dashboard logic.

Key Takeaway

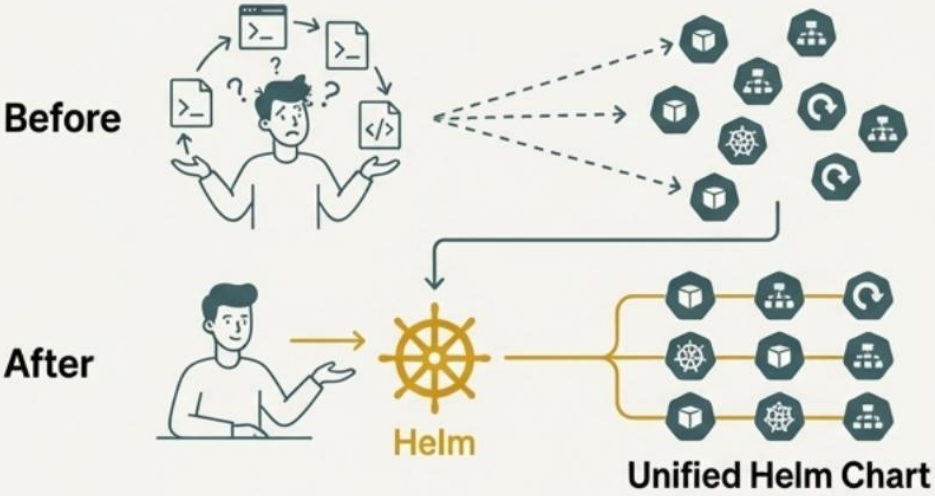
A well-defined data contract decouples teams, is the foundation for agile development in data projects, and drastically reduces integration risk.

4. Notable Lesson #4: Robust Deployment Requires Infrastructure as Code

Problem

Managing 15+ Kubernetes resources with manual `kubectl apply` commands and shell scripts was brittle, error-prone, and lacked versioning.

Solution



Migrated all Kubernetes YAML manifests into a unified **Helm chart**. This templated the entire application, centralizing configuration in a single `values.yaml` file and enabling versioned, atomic deployments and rollbacks with a single command.

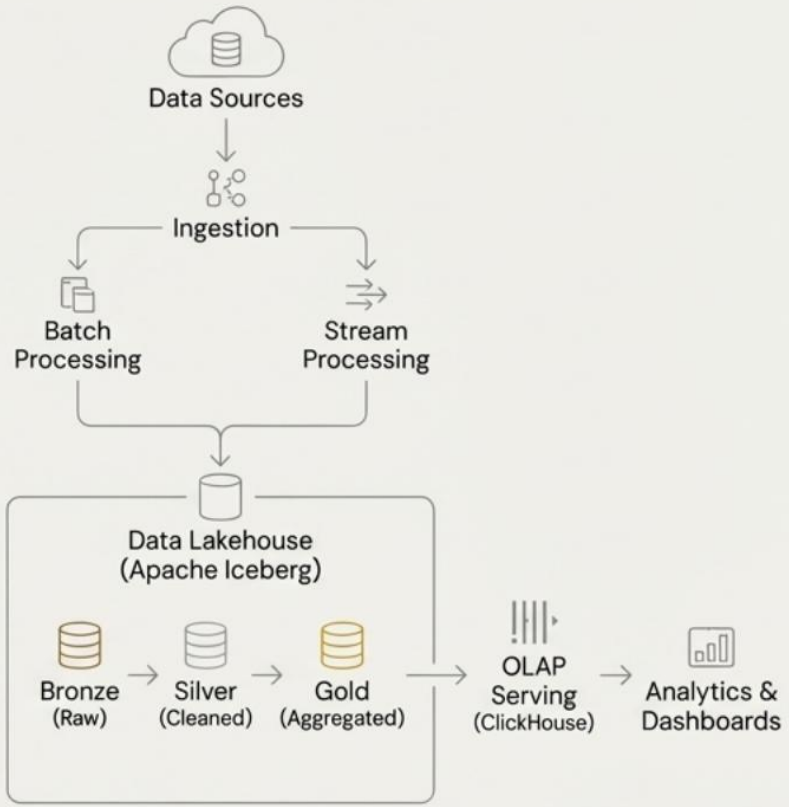
Key Takeaway

Helm treats infrastructure as versioned code (IaC), preventing configuration drift and making complex deployments manageable, resilient, and repeatable.

5. Conclusion: An End-to-End Financial Data Platform

We successfully designed and implemented a scalable, end-to-end Big Data Lakehouse that addresses the core challenges of complex financial analysis.

- **Unified Architecture:** Integrated batch and stream processing via a Lambda pattern.
- **Data Quality & Reliability:** Ensured curated, reliable data ae data with the Medallion model on Apache Iceberg.
- **High-Performance Analytics:** Delivered low-latency dashboards through a dedicated OLAP serving layer (ClickHouse).
- **Robust & Reproducible System:** Built on a modern, containerized infrastructure managed with Infrastructure as Code (Helm).



Simplified Architecture Overview