Home About Contact

Big Data Project

AERO: AIR TRAFFIC EXPLORATION & REAL-TIME ORCHESTRATION

Presented by Group 9

GROUP MEMBERS

Dao Xuan Quang Minh	20225449
Trinh The Minh	20225513
Nguyen Minh Duong	20225439
Vu Ngoc Ha	20225490
Bui Anh Duong	20225489

TABLE CONTENT

1. PROBLEM DEFINITON

1.1. Context and Background

- **Domain:** US National Airspace System (NAS) – one of the world's busiest networks.
- **Issue:** Persistent flight delays and cancellations cause:
  - **Economic Loss:** Billions in operational costs.
  - **Passenger Impact:** Erosion of trust and scheduling chaos.
- **The Gap:** Legacy systems (spreadsheets/RDBMS) cannot handle the scale of modern aviation data.

1. PROBLEM DEFINITON

1.2. Problem Statement (The “3 Vs”)

- **Volume (Storage & Compute):** 37+ years of records (1987–2024) > 80GB
  - Consequence: Single-node processing leads to Out-of-Memory (OOM) errors.
- **Velocity (Latency):** Flight statuses change in seconds. Legacy $T+1$ batch processing cannot support real-time operational adjustments.
  - Consequence: Lack of real-time visibility for operational adjustments.
- **Variety (Fragmentation):** Historical trends and live streams reside in disconnected silos.

1. PROBLEM DEFINITON

1.3. Project Objectives

Goal: Engineer a cloud-native End-to-End Big Data Pipeline on Google Cloud Platform (GCP)

Key Objectives

- **Modern Data Stack:** Transition from Hadoop to a scalable ELTV model (BigQuery, Kubernetes).
- **Batch Processing Layer:** Analyze 30+ years of historical data (Seasonal trends, carrier reliability).
- **Streaming Layer:** Implement low-latency ingestion using Apache Kafka & Spark Structured Streaming.
- **Data Democratization:** Provide interactive, auto-refreshing dashboards via Looker Studio.

1. PROBLEM DEFINITON

1.4. Scope and Limitations

Scope

- **Geographic Focus:** US Domestic Flights (High-quality, complex dataset)
- **Primary Focus:** Data Engineering (Pipeline construction, Orchestration, Optimization)

Data Sources

- **Historical Data:** US Bureau of Transportation Statistics
- **Real-time Data:** Live flight streams via AviationStack API

Limitations

- **API Constraints:** Rate limiting on real-time streams
- **Infrastructure:** Dependent on GCP resource availability and cost optimization strategies

2. DATA SOURCES

2.1. Overview

To satisfy the **ELTV** architecture, the system integrates two distinct data categories:

- **Historical Data (Batch Layer):**
  - Focus on Long-term trend analysis & pattern recognition.
  - Role: The "Knowledge Base" of the system.
- **Real-time Data (Streaming Layer):**
  - Focus on Immediate operational monitoring.
  - Role: The "Live Pulse" of the system.

→ **Goal:** Correlate past performance patterns with current operational anomalies.

2. DATA SOURCES

2.2. Historical Data (Batch Layer)

Source: United States Bureau of Transportation Statistics (BTS).

Dataset Profile:

- Time Span: 1987 – 2024 (37 Years).
- Scale: ~200 Million flight records.
- Volume: > 80 GB raw data (CSV format).
- Coverage: Granular view of nearly every US domestic flight.

Key Data Fields:

- **Critical Metric:** ArrDelay (Arrival Delay in minutes) – The primary target for analysis.
- **Dimensions:** Temporal (Year, Quarter, Day), Spatial (Origin/Dest Airport), and Operational (Airline, Tail Number).

2. DATA SOURCES

2.2. Historical Data (Batch Layer)

Field	Description
Year	Year of the flight. Helps in identifying long-term trends or seasonal patterns.
Quarter	Quarter of the year (1–4). Useful for analyzing seasonal variations.
Month	Month of the flight. Allows for monthly breakdown of data.
DayOfMonth	Day of the month when the flight occurred. Useful for precise date-specific analysis.
DayOfWeek	Day of the week (1 for Monday, 7 for Sunday). Important for identifying weekly trends.
FlightDate	Exact date of the flight. A critical field for time-series analysis.

Reporting Airline	Unique carrier code for the reporting airline. Helps distinguish between airlines.
Tail Number	Unique aircraft tail number. Useful for tracking performance by specific aircraft.
Flight Number	Reporting Flight number reported by the airline. Enables tracking individual flights.
OriginAirportID	Unique ID of the origin airport. Essential for identifying departure locations.
OriginCityName	City name of the origin airport. Useful for grouping and summarizing data by city.
DestAirportID	Unique ID of the destination airport. Important for analyzing arrival locations.
DestCityName	City name of the destination airport. Enables grouping data by destination city.
ArrDelay	Arrival delay in minutes. The critical performance metric for this study.

Key Fields in the US Bureau of Transportation Dataset

2. DATA SOURCES

2.3. Real-time Data (Streaming Layer)

Source: AviationStack API

- Format: Continuous stream of JSON messages
- Ingestion Flow: Polling Mechanism → Apache Kafka

Distinguishing Attributes

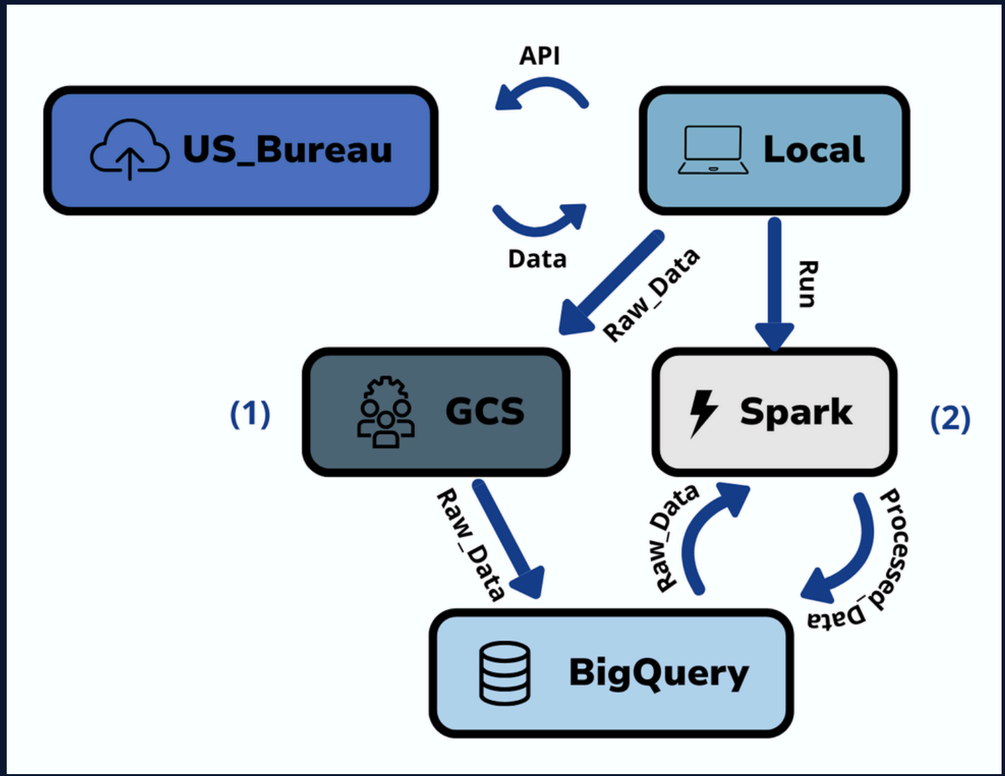
Unlike historical data (which is static/finalized), this stream provides:

- **Live Status:** Tracking states like Active, Scheduled, Landed, Cancelled
- **Geolocation:** Real-time Latitude / Longitude (Enabling live map visualization)
- **Dynamic ETA:** Estimated Arrival Time updated based on current conditions

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.1. Architecture Evolution: From Legacy to Cloud-Native

Historical Pipeline (Initialization)



- **Goal:** Ingest 37 years of data (1987–2024).
- **Flow:** Local Script

→ **GCS (Data Lake/Bronze Layer)**

→ Spark Transformation

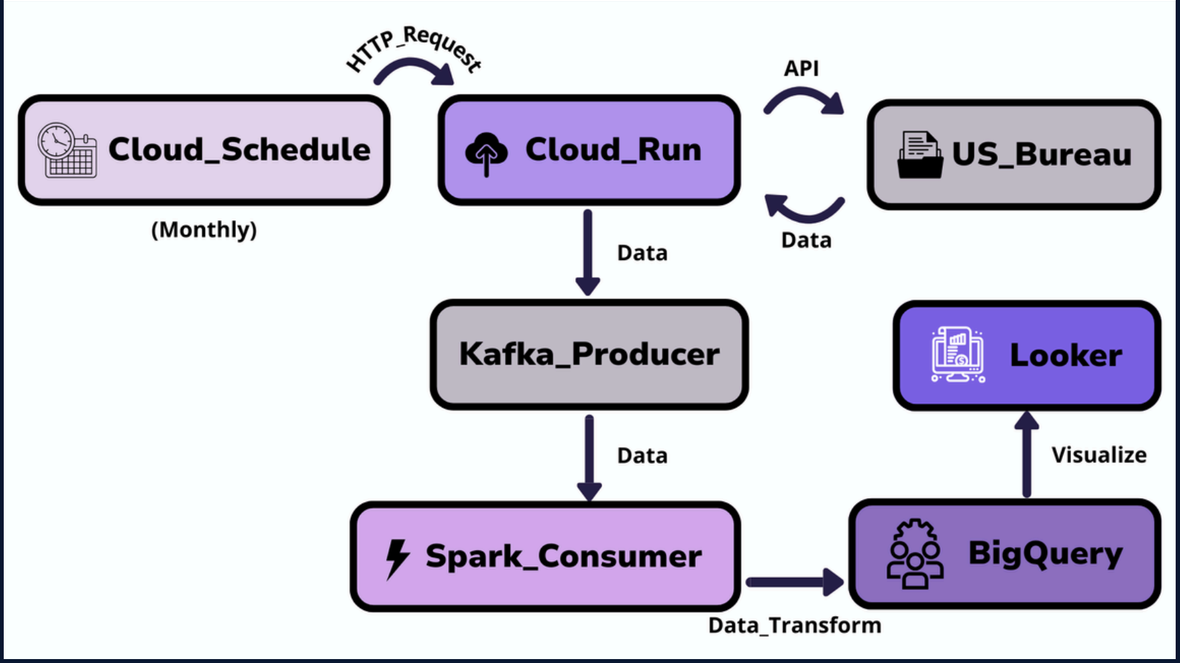
→ BigQuery

- **Key:** Focus on high-throughput bulk loading.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.2. Detailed Pipeline Design

Monthly Pipeline (Automation)

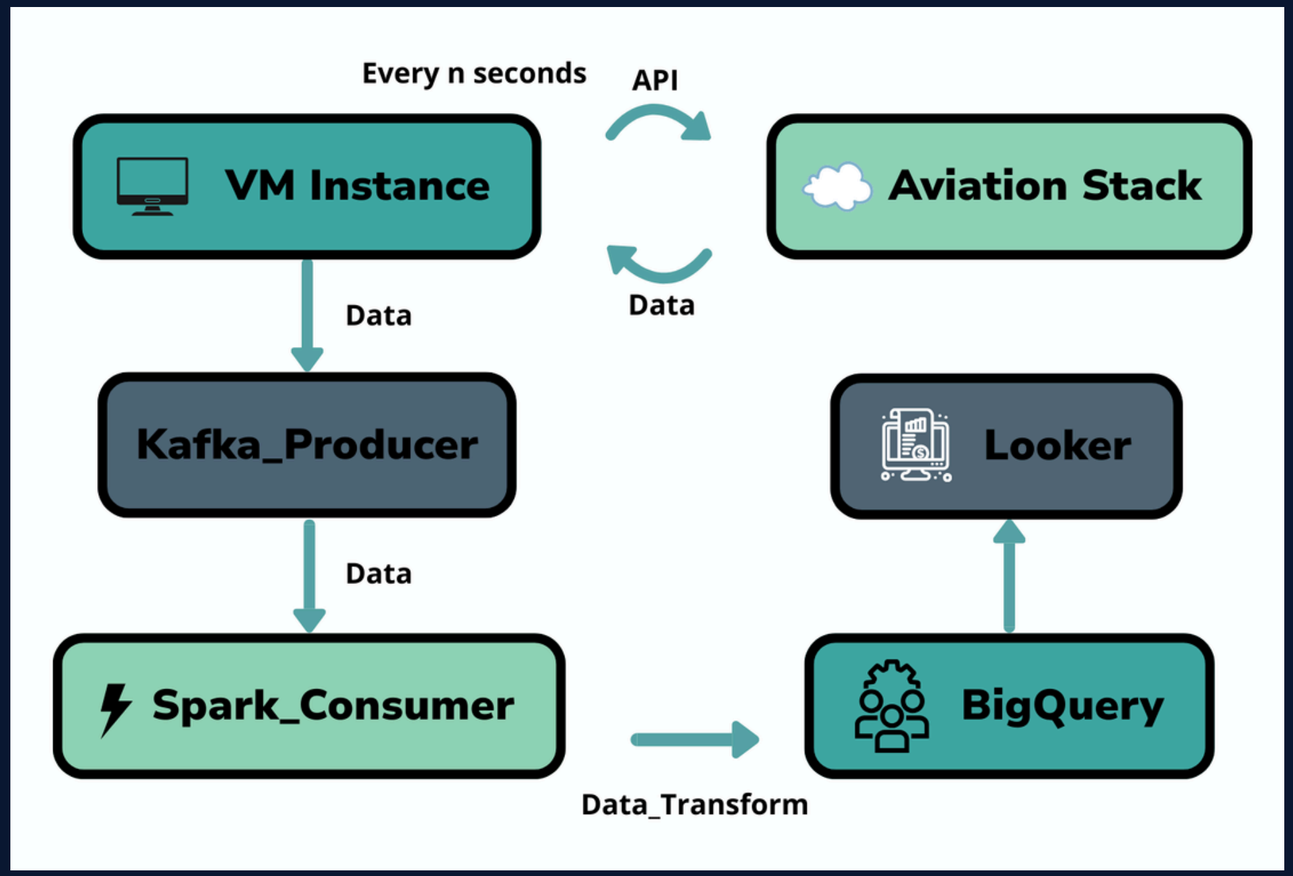


- **Goal:** Keep data current without reprocessing history
- **Flow:**
  - Cloud Scheduler: Triggers monthly jobs.
  - Cloud Run: Serverless extraction of new reports.
  - Unified Ingestion: Pushes data to Kafka to reuse the streaming logic (Ensures consistency).

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.2. Detailed Pipeline Design

Real-time Streaming Pipeline



3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.2. Detailed Pipeline Design

Real-time Streaming Pipeline

Goal: Provide continuous situational awareness.

The Workflow:

- **Polling (VM Instance):** Producer polls AviationStack API every n seconds.
- **Buffering (Apache Kafka):** Decouples ingestion from processing; handles high-velocity bursts.
- **Processing (Spark Structured Streaming):**
  - Consumes events from Kafka.
  - Computes live metrics (Current Delays).
- **Serving (BigQuery Storage Write API):** Writes directly to tables for instant visualization on Looker.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.3. Implementation Strategy and Technology Stack

Core Technology Selection

Driven by the "Three Vs" of Big Data:

- **Ingestion:** Apache Kafka (on Kubernetes)
  - High throughput & Fault tolerance
- **Processing:** Apache Spark (Unified Engine)
  - Handles both Batch & Streaming layers (Code reusability)
  - Exactly-Once Semantics via Checkpointing (Reliability)
- **Warehousing:** Google BigQuery
  - Serverless & Scalable storage
  - Optimized via Partitioning (by Date) & Clustering (by Airline)

    → Minimizes query costs & latency

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.3. Implementation Strategy and Technology Stack

Containerization & Orchestration Strategy

Docker

- All components (Python Producers, Spark Executors) are packaged as Docker images
- Ensures consistency between Development (Local) and Production (Cloud)

Kubernetes (GKE)

- Lifecycle Management: Manages the deployment of the entire pipeline
- Self-Healing: Automatically restarts failed pods (e.g., if a Spark worker crashes)
- Horizontal Scaling: Dynamically adds Spark workers during heavy load periods (processing 200M records) and scales down to save costs

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.4. Source Code Implementation Detail

Overview

```
Big-Data-Project-AERO/
|-- src/
|    |-- extract/
|    |-- load/
|    |-- transform/
|-- setup/
|    |-- setup_bigquery.py
|    |-- load_historical_data.py
|    |-- spark_transformation.py
|-- flow/
|    |-- collect_data_flow.py
|-- k8s/
     |-- deployment.yaml
     |-- kafka/
```

BigQuery Optimization

Schema: Rigorous typing (Date, Float64).

Table Strategy:

- flights_raw: Partitioned by Date, Clustered by Airline/Origin/Dest.
- flights_processed: Optimized for time-series.
- flights_analytics: Pre-aggregated stats.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.4. Source Code Implementation Detail

ETL Implementation (Batch & Orchestration)

Orchestration with Prefect

- **Parallelism:** Uses ThreadPoolExecutor to speed up downloads/uploads of 30+ years of data.
- **Staging:** All data lands in GCS (gs://aero_data/) first.

Historical Loading

- Idempotency: Uses WRITE_TRUNCATE to ensure re-runs don't duplicate data.

Spark Transformation

- Feature Engineering: Generates flags: Is_Delayed, Is_Cancelled.
- Null Handling: Fills missing values (e.g., CarrierDelay = 0.0) for accurate aggregation.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.4. Source Code Implementation Detail

Real-time Logic Implementation

- **Ingestion:** Reads from Kafka topic flights (startingOffsets="earliest").
- **Transformation (AeroSparkProcessor):**
  - Parsing: Enforces strict StructType schema on JSON payloads.
  - Logic: Calculates departure_delay_minutes.
  - Categorization: Tags flights dynamically:
    - On-Time, Minor (0-15m), Moderate (15-60m), Major (>60m).
- **Watermarking:** Applies a 10-minute watermark to handle late-arriving data correctly.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

3.4. Source Code Implementation Detail

Cloud-Native Deployment

Containerization (Docker)

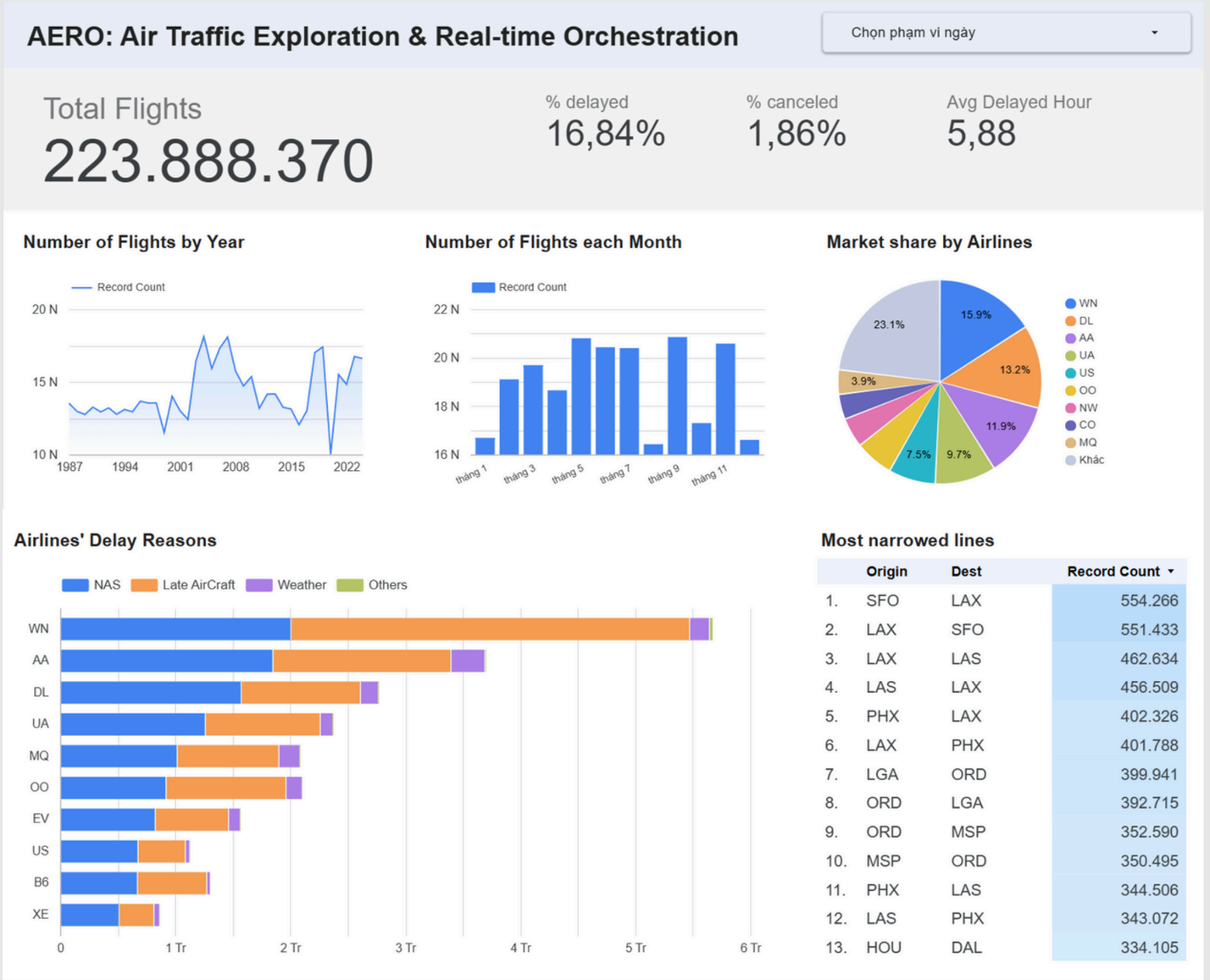
- Consistent environments with pre-installed dependencies (pyspark, kafka-python).

Kubernetes Orchestration (GKE)

- **StatefulSets (Kafka):** Ensures persistence of flight logs and stable network identity.
- **Deployments (Apps):** Manages stateless Spark Drivers and Consumers.
- **Local Dev:** docker-compose for rapid local testing before cloud deployment.

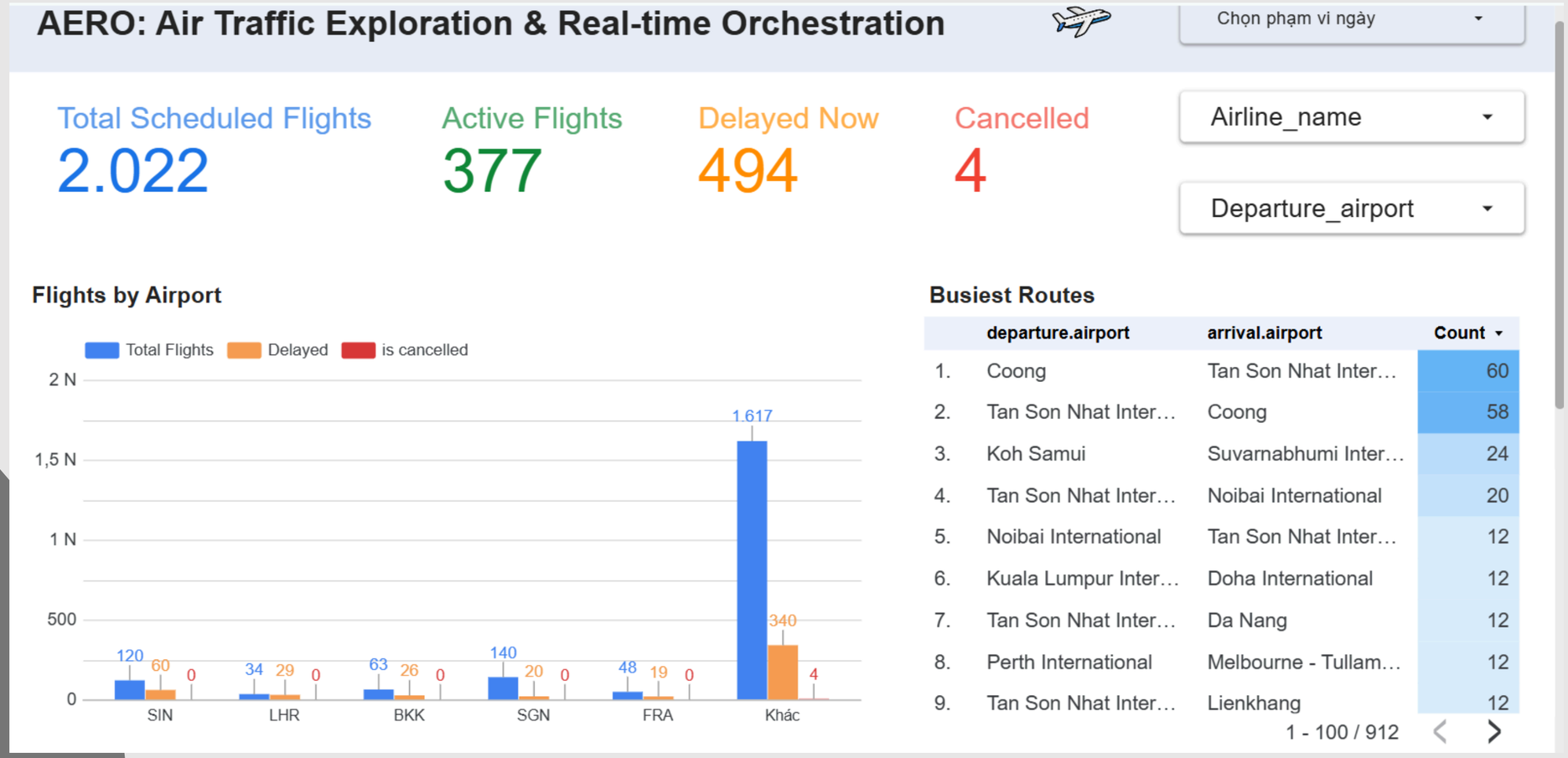
4. CONCLUSION AND FUTURE WORK

BTS Data Source



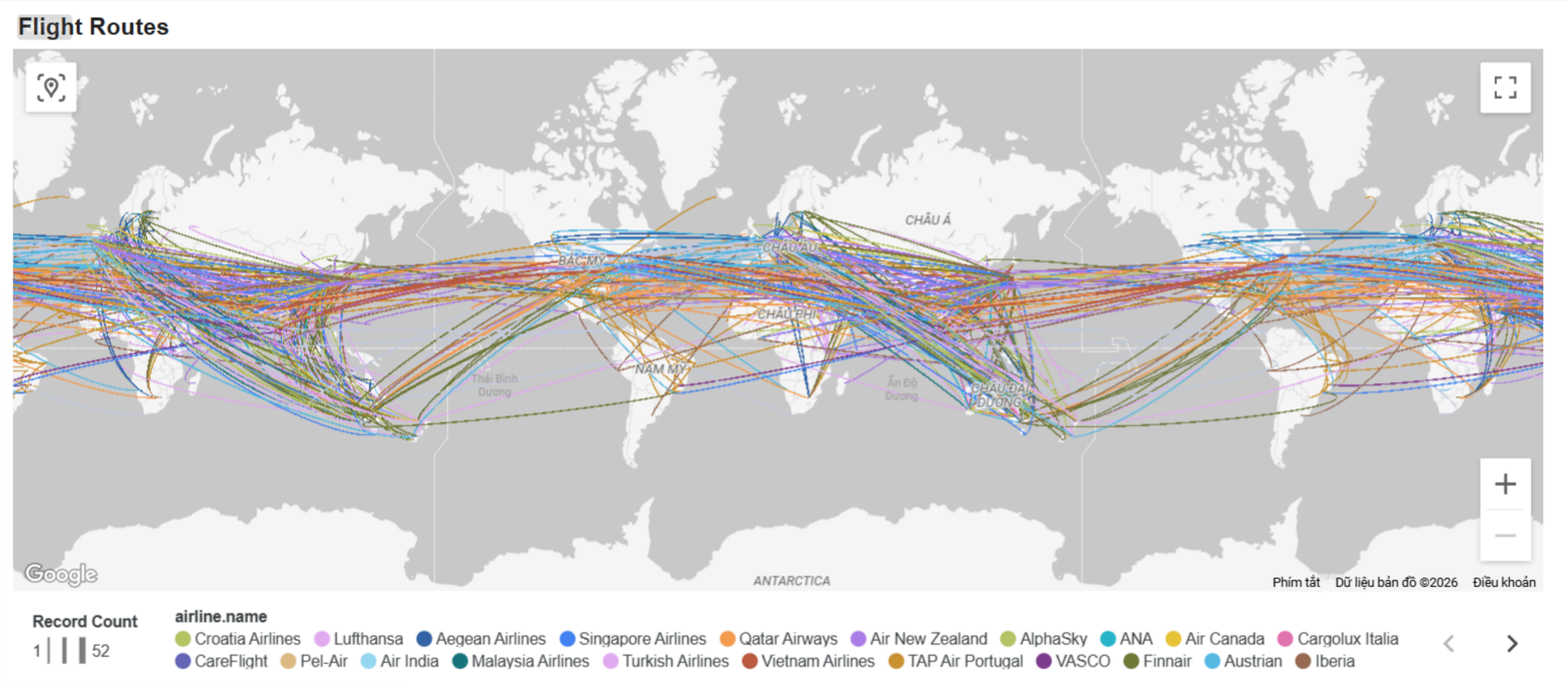
4. CONCLUSION AND FUTURE WORK

Realtime via Aviation Stack



4. CONCLUSION AND FUTURE WORK

Realtime via Aviation Stack



4. CONCLUSION AND FUTURE WORK

Future Work

- **Predictive Analytics:** Integrate Spark MLlib / Vertex AI to predict delays 24h in advance
- **Architecture Evolution:** Move to Kappa Architecture to simplify code maintenance
- **DevOps & FinOps:** Implement CI/CD pipelines and stricter cost optimization strategies

Home About Contact

See You Next Time

THANK YOU

AERO: Air Traffic Exploration & Real-time Orchestration