TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# Flight Delays and Cancellations Pipeline

Group 1
Phạm Anh Tú 20225463
Etienne Fontbonne 20250006S
Eliott Moskowicz 20250075S
Florian Dorchies 20250072S
Philippe Marchand 20250083S

ONE LOVE. ONE FUTURE.

# Problem Statement

Air travel delays are a persistent issue affecting passengers, airlines, and airport operations worldwide.

Delays can arise from a complex interplay of factors such as :

- weather conditions
- air traffic congestion
- carrier performance
- airport infrastructure
- scheduling inefficiencies.

We aim to **process** and **visualize** various statistics about this dataset to inform customers about the performance of different airlines and routes.

# Dataset

2015 Flight Delays and Cancellations dataset

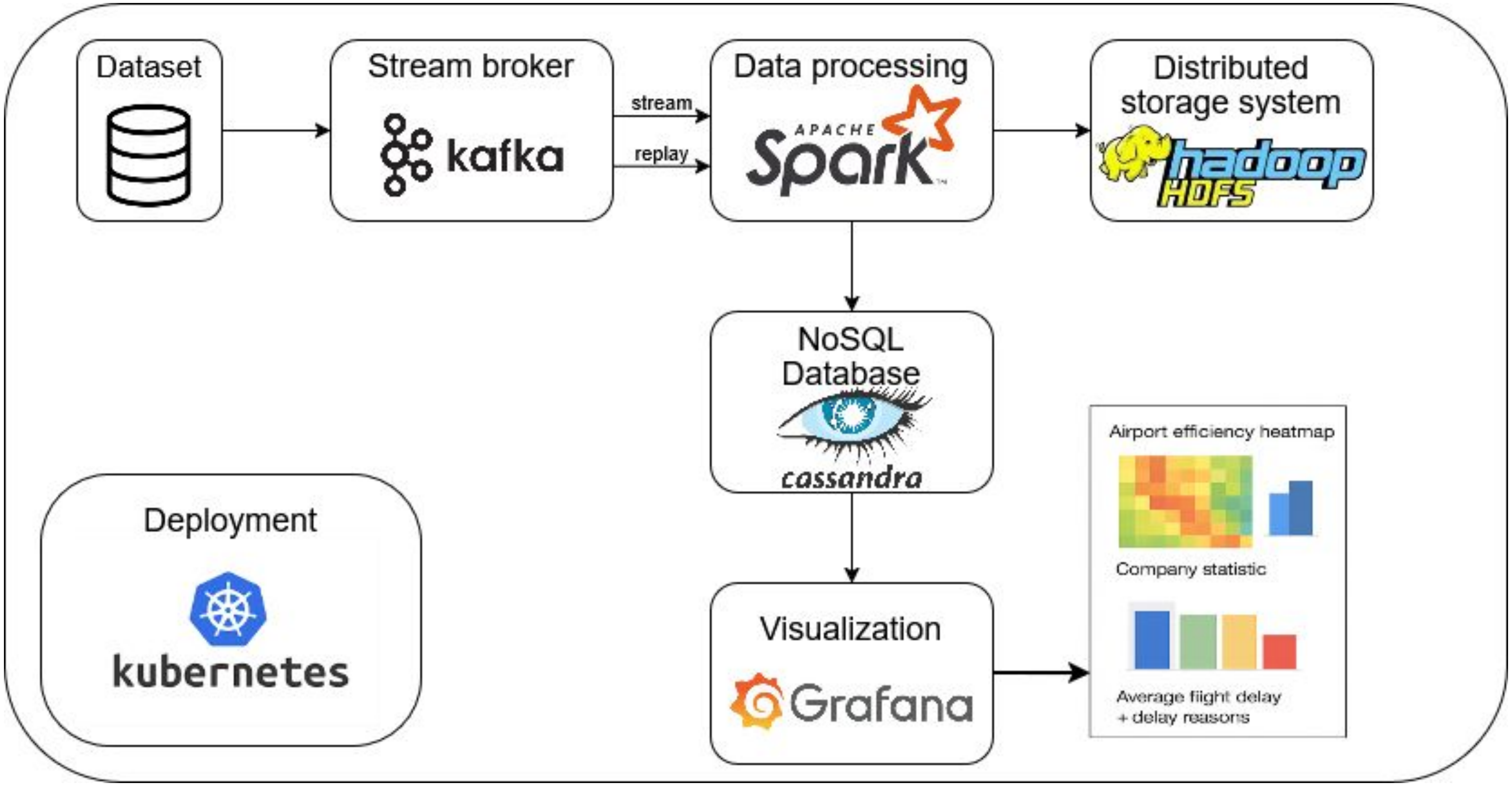
Tracks the on-time performance of domestic flights in the USA operated by large air carriers.

- 3 tables:
  - airlines : 2 columns
  - airports : 7 columns
  - flights : 31 columns
- 4 data types
- Source : kaggle

Over 1M rows in the dataset.

# Proposed pipeline

## Kappa architecture



# Kafka

- **Main role:** Data ingestion
- **Producer :** sends flights events into a Kafka topic
- **Broker :** central server that stores and distributes messages
- Using **Zookeeper :**
  - Zookeeper ensures consistency across the cluster.
  - Handles broker leader election and monitors node status
- Produces through a python file

# Spark

- **Main role:** Data Processing
- **Transformations:**
  - Parse JSON messages using robust type casting
  - Read and cached static data
  - Handle missing values and data formatting
  - Add timestamp
  - Custom UDF: eparate on-time flights and delayed flights according to US laws.

# Components

## Spark

- **Aggregations:**
  - Airline-level statistics (on-time, delayed, cancelled flights and calculate average delay).
  - Route-level average delays enriched with geographic information.
  - Hourly delay trends (avg, std, median, ...) and statistical summaries.
  - Delay statistics grouped by delay reasons

# Spark

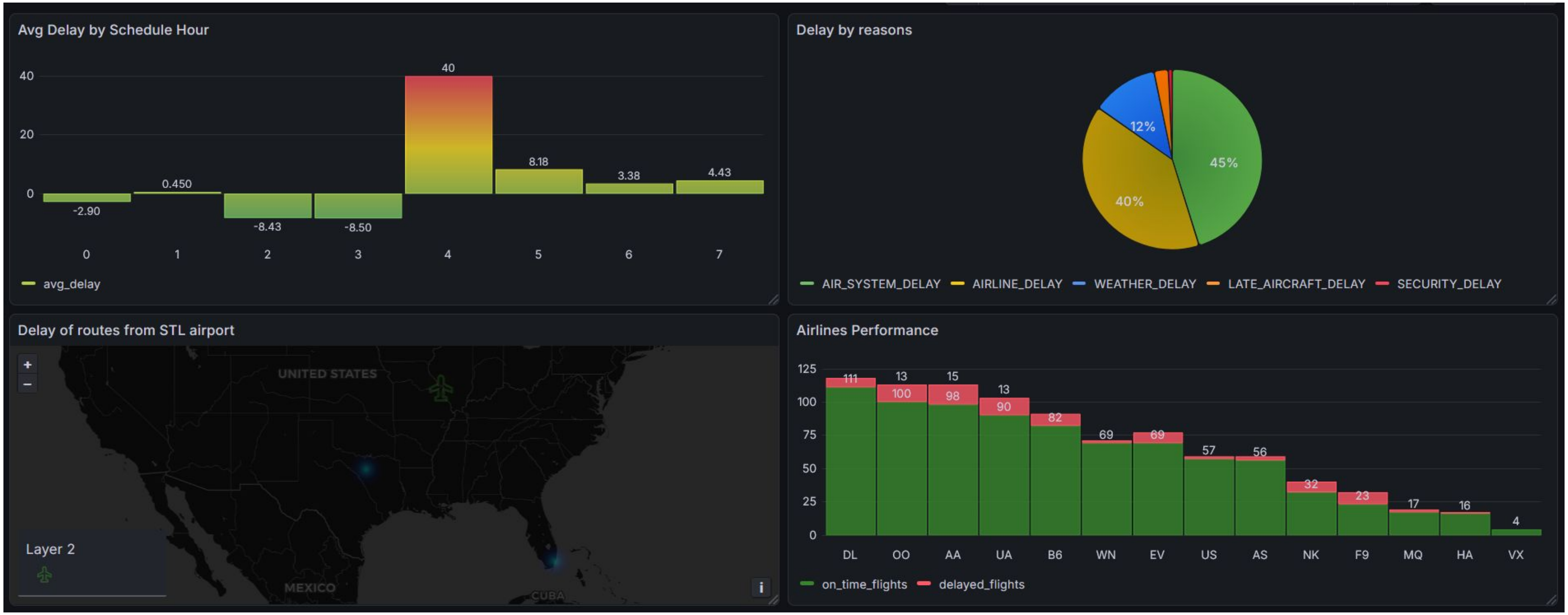
- **Key optimizations:**
  - Controlled micro-batch size  and backpressure
  - Partition alignment with Kafka
  - Caching static datasets
  - Resource limits via Docker/Kubernetes
- **Result:**
  - Stable startup
  - Predictable latency

## Cassandra

- **Role:** Serving database for Grafana
- Stores 4 tables:
  - airline_stats: Stores airline-level performance metrics aggregated over streaming batches
  - delay_by_reason: Stores aggregated delay statistics grouped by delay category
  - route_stats: Stores route-level delay analytics between origin and destination airports.
  - hourly_stats: Stores time-series statistics aggregated by scheduled departure hour

# Grafana

- **Role:** End of pipeline visualization



# HDFS

- **Main role :** Cold data storage and checkpoints
- **Data archival :**
  - Save of enriched flights for archival purposes
- **Checkpoints :**
  - Acts as a safe place for Spark streaming checkpoint
  - This allows recovery of streaming jobs after failures, state management and improved resistance

# From Docker to Kubernetes

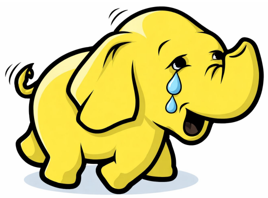
- Firstly started on Docker for easier development
- Then switch to Kubernetes, using Minikube
- Problematic :
    - Need to convert **`docker-compose.yml`** to Kubernetes manifest
    - Preserve process starting order

# From Docker to Kubernetes

- Solution :
  - Use of InitContainers to preserve startup dependencies
  - Define StatefulSet for stable containers (Kafka, Cassandra, ...)
  - Define Deployment to enable easier scaling and updates (Spark, Grafana, ...)
  - Replace configuration files by ConfigMaps
  - Define PVC to ensure durable storage
  - Automatic deployment using a bash file

# From Docker to Kubernetes

- Issues :
  - **HDFS :**
    - Not resistant to pressure
    - Needed too much resources to be stable
    - -> could not be used in the **Kubernetes** deployment
    - Consequences :
      - Local checkpoint in Spark processing
      - No cold storage
  - No access to local repository :
    - Solution :
      - Uses of mounted ConfigMap during deployment
      - Creation of a custom Docker Image for Kafka



# From Docker to Kubernetes

- Deployment script :
  - Checks minikube and dataset status
  - Applying namespace and then all manifests
  - Builds the Kafka Docker Image
  - Loads the ConfigMap

- Then validation with :
  - `kubectl get pods`
  - `kubectl logs <pod>`
  - `minikube logs`

# Challenges

- Not all images are free : Bitnami Spark, Grafana MongoDB plugins .
  - => caused changes in the pipeline
- Latency between data generation and availability for processing
  - Controlling micro-batches size, flow control
- Spark streaming from Kafka produced null metrics and startup delays due to parsing and backlog issues
  - Ingest fields as StringType
  - Data quality checks

HUST

hust.edu.vn fb.com/dhbkhn

# THANK YOU !