

ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

Xây dựng hệ thống hồ dữ liệu phân tích dữ liệu
chuyển bay sử dụng các thành phần của hệ sinh thái
Hadoop

TRẦN NGỌC BẢO

bao.tn215529@sis.hust.edu.vn

Ngành Kỹ thuật máy tính

Giảng viên hướng dẫn: TS. Trần Việt Trung

Chữ kí GVHD

Khoa: Khoa học máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 12/2024

LỜI CẢM ƠN

Trong những năm tháng thanh xuân rực rỡ, em thật may mắn và tự hào khi được trở thành sinh viên của Đại học Bách khoa Hà Nội. Học tập và rèn luyện trong môi trường chuyên nghiệp, năng động tại đây là một trải nghiệm quý giá, đáng nhớ trong cuộc đời. Bốn năm gắn bó với mái trường Bách khoa là hành trình đầy ắp những thử thách, vấp ngã nhưng cũng tràn đầy niềm tự hào và vinh quang. Trước hết, em xin gửi lời tri ân sâu sắc đến gia đình – chỗ dựa vững chắc và nguồn động lực to lớn giúp em vượt qua mọi khó khăn trên con đường học vấn. Em cũng chân thành cảm ơn TS. Trần Việt Trung, người đã tận tình hướng dẫn, định hướng và đồng hành cùng em trong quá trình thực hiện đồ án này. Đồng thời, em bày tỏ lòng biết ơn đến các thầy cô giảng viên của Đại học Bách khoa Hà Nội, đặc biệt là các thầy cô thuộc trường Công nghệ Thông tin và Truyền thông, những người đã truyền đạt cho em những kiến thức quý báu – hành trang vững chắc cho tương lai. Cuối cùng, em không thể không nhắc đến những người bạn đồng hành tuyệt vời – những người luôn sẵn sàng sẻ chia, hỗ trợ em trong suốt thời gian học tập. Những năm tháng tươi đẹp ấy sẽ mãi là ký ức đáng trân trọng và là nguồn động lực to lớn để em không ngừng phấn đấu trên hành trình phía trước.

TÓM TẮT NỘI DUNG ĐỒ ÁN

Dữ liệu lớn ngày càng được xem như một "tài nguyên" cực kỳ giá trị trong hiện tại và tương lai. Việc lưu trữ và xử lý dữ liệu lớn đóng vai trò thiết yếu đối với các doanh nghiệp, đặc biệt là trong việc quyết định mô hình hệ thống và công nghệ phù hợp để triển khai. Hiện nay, các hệ thống dữ liệu lớn có thể được triển khai theo các mô hình như kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu, cùng với nhiều hệ sinh thái công nghệ hỗ trợ như Hadoop, Amazon Web Services, Google Cloud Platform. Việc lựa chọn mô hình và hệ sinh thái phù hợp để xây dựng hệ thống đòi hỏi phải giải quyết các thách thức về khối lượng lưu trữ và xử lý, chi phí và bảo mật.

Đồ án này được em phát triển và thực hiện với mục tiêu xây dựng một hệ thống hồ dữ liệu phân tích chuyển bay dựa trên các công nghệ của hệ sinh thái Hadoop. Em đã lấy một nguồn dữ liệu về chuyển bay tại Hoa Kỳ trên Kaggle để đáp ứng được yêu cầu dữ liệu lớn áp dụng cho hệ thống. Sau đó em dựng lên một hệ thống trên nền tảng ảo hóa Kubernetes để giả lập quá trình lấy dữ liệu theo chu kỳ, lấy và xử lý dữ liệu thời gian thực. Tiếp tiến hành lưu trữ dữ liệu và xử lý dữ liệu theo lô. Cuối cùng là thực hiện phân tích, truy vấn và trực quan hóa dữ liệu để cho ra kết quả cuối cùng. Luồng xử lý dữ liệu và các bước trong luồng xử lý đều được lập lịch và giám sát thực hiện.

Hệ thống được em xây dựng trong đồ án tốt nghiệp này đã thực hiện được một luồng xử lý dữ liệu lớn hoàn chỉnh, là một hệ thống phân tán, có các mô-đun chức năng riêng biệt, có khả năng mở rộng hệ thống, dễ dàng sử dụng và quản lý. Đặc biệt, hệ thống của em có thể đáp ứng được việc triển khai một sản phẩm trên môi trường thực tế trong việc lưu trữ và xử lý dữ liệu lớn cho doanh nghiệp giúp tiết kiệm chi phí.

Sinh viên thực hiện
(Ký và ghi rõ họ tên)

ABSTRACT

Big Data is increasingly regarded as an invaluable "resource" in both the present and the future. Storing and processing Big Data play a crucial role for businesses, particularly in determining the appropriate system model and technology for implementation. Currently, Big Data systems can be deployed using models such as data warehouse, data lake, and lakehouse, supported by various technological ecosystems like Hadoop, Amazon Web Services, and Google Cloud Platform. Selecting the right model and ecosystem to build these systems involves addressing challenges related to storage and processing capacity, costs, and security.

This project is developed with the goal of building a flight data lake system based on technologies from the Hadoop ecosystem. I used a dataset of U.S. flight information available on Kaggle to meet the Big Data requirements of the system. The system was deployed on a Kubernetes-based virtualization platform to simulate cyclical data collection and real-time data ingestion and processing. The project also included storing data, batch processing, analyzing, querying, and visualizing data to produce final results. The data processing flow and all steps within it were scheduled and monitored throughout execution.

The system developed in this graduation project successfully implements a complete Big Data processing flow. It is a distributed system with distinct functional modules, scalability, and ease of use and management. Notably, the system is capable of being deployed in real-world environments to support enterprises in storing and processing Big Data, thereby helping to reduce costs.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Đặt vấn đề.....	1
1.2 Mục tiêu và phạm vi đề tài.....	1
1.3 Định hướng giải pháp.....	2
1.4 Bố cục đồ án	3
CHƯƠNG 2. KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU.....	4
2.1 Khảo sát hiện trạng	4
2.2 Tổng quan chức năng	6
2.2.1 Yêu cầu chức năng	6
2.2.2 Yêu cầu phi chức năng.....	8
CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG.....	9
3.1 Apache Hadoop.....	9
3.1.1 HDFS	9
3.1.2 YARN.....	9
3.2 Apache Spark.....	10
3.2.1 Spark SQL	10
3.2.2 Spark Streaming	10
3.2.3 Spark YARN	11
3.3 Apache Airflow.....	11
3.4 Apache Kafka	12
3.5 Trino	13
3.6 Apache Hive	14
3.7 Apache Superset.....	14
3.8 Kubernetes.....	15

CHƯƠNG 4. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG	17
4.1 Thiết kế hệ thống.....	17
4.1.1 Kiến trúc tổng quan hệ thống	17
4.1.2 Mô-đun thu thập dữ liệu	18
4.1.3 Mô-đun lập lịch tác vụ.....	21
4.1.4 Mô-đun lưu trữ dữ liệu	23
4.1.5 Mô-đun xử lý dữ liệu.....	25
4.1.6 Mô-đun truy vấn dữ liệu	34
4.1.7 Mô-đun trực quan hóa dữ liệu	37
4.2 Triển khai hệ thống.....	38
4.2.1 Triển khai cụm Hadoop	39
4.2.2 Triển khai cụm Airflow	41
4.2.3 Triển khai nguồn phát dữ liệu.....	42
4.2.4 Triển khai cụm Kafka.....	43
4.2.5 Triển khai cụm Hive.....	44
4.2.6 Triển khai cụm Trino	45
4.2.7 Triển khai cụm Superset	46
4.2.8 Kết quả triển khai.....	46
CHƯƠNG 5. KẾT QUẢ THỰC NGHIỆM.....	49
5.1 Kết quả lập lịch và tự động hóa	49
5.2 Tính chịu lỗi của hệ thống	50
5.3 Tính khả mở của hệ thống	51
5.4 Kết quả lưu trữ dữ liệu	52
5.5 Hiệu suất truy vấn dữ liệu.....	55
5.6 Kết quả trực quan hóa dữ liệu	57

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	62
6.1 Kết luận.....	62
6.2 Hướng phát triển.....	62
TÀI LIỆU THAM KHẢO.....	64

DANH MỤC HÌNH VẼ

Hình 2.1	Kiến trúc kho dữ liệu, hồ dữ liệu, hồ kho dữ liệu. Nguồn [2]	4
Hình 2.2	Sơ đồ phân rã chức năng hệ thống	7
Hình 4.1	Kiến trúc tổng quan hệ thống	17
Hình 4.2	Đồ thị luồng xử lý dữ liệu trên Airflow	21
Hình 4.3	Dữ liệu nguồn tại máy chủ Flask	24
Hình 4.4	Triển khai cụm HDFS	39
Hình 4.5	Triển khai cụm YARN	40
Hình 4.6	Triển khai cụm Airflow	41
Hình 4.7	Triển khai nguồn phát dữ liệu	42
Hình 4.8	Triển khai cụm Kafka	43
Hình 4.9	Triển khai cụm Hive	44
Hình 4.10	Triển khai cụm Trino	45
Hình 4.11	Triển khai cụm Superset	46
Hình 4.12	Cụm Kubernetes	46
Hình 4.13	Tổng quan thành phần của cụm Kubernetes	47
Hình 4.14	Các Deployment của hệ thống	47
Hình 4.15	Các Statefulset của hệ thống	47
Hình 4.16	Các Service của hệ thống	48
Hình 4.17	Các Persistent Volume Claim của hệ thống	48
Hình 5.1	Kết quả lập lịch tác vụ trên Airflow (1)	49
Hình 5.2	Kết quả lập lịch tác vụ trên Airflow (2)	49
Hình 5.3	Tính chịu lỗi trên cụm Kafka (1)	50
Hình 5.4	Tính chịu lỗi trên cụm Kafka (2)	50
Hình 5.5	Tính chịu lỗi khi sử dụng Kubernetes (1)	50
Hình 5.6	Tính chịu lỗi khi sử dụng Kubernetes (2)	51
Hình 5.7	Tính khả mở khi sử dụng Kubernetes (1)	51
Hình 5.8	Tính khả mở khi sử dụng Kubernetes (2)	51
Hình 5.9	Tính khả mở khi sử dụng Kubernetes (3)	52
Hình 5.10	Kết quả lưu trữ trên cụm Kafka (1)	52
Hình 5.11	Kết quả lưu trữ trên cụm Kafka (2)	52
Hình 5.12	Kết quả lưu trữ trên cụm Kafka (3)	53
Hình 5.13	Kết quả lưu trữ dữ liệu trên HDFS (1)	53
Hình 5.14	Kết quả lưu trữ dữ liệu trên HDFS (2)	54
Hình 5.15	Kết quả lưu trữ dữ liệu trên HDFS (3)	54

Hình 5.16	Hiệu suất truy vấn dữ liệu trên Trino (1)	55
Hình 5.17	Hiệu suất truy vấn dữ liệu trên Trino (2)	55
Hình 5.18	Hiệu suất truy vấn dữ liệu trên Trino (3)	56
Hình 5.19	Hiệu suất truy vấn dữ liệu trên Trino (4)	57
Hình 5.20	Độ trễ trung bình theo hãng hàng không năm 2018	58
Hình 5.21	Tỉ trọng chuyến bay và tỉ trọng chuyến bay bị hủy theo ngày trong tháng 1 năm 2019	58
Hình 5.22	Phân phối mạng lưới tiếp thị hàng không quý 1 năm 2020	59
Hình 5.23	Tỉ trọng chuyến bay và chuyến bay bị hủy theo từng hãng hàng không trong quý 4 năm 2021	60
Hình 5.24	Số chuyến bay theo tháng qua các năm	60

DANH MỤC BẢNG BIỂU

Bảng 2.1	So sánh kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu	5
Bảng 4.1	Dữ liệu thời gian của chuyến bay	29
Bảng 4.2	Hãng vận chuyển tiếp thị và hãng vận chuyển điều hành của chuyến bay	30
Bảng 4.3	Địa điểm xuất phát và địa điểm đến của chuyến bay	31
Bảng 4.4	Thời gian khởi hành của chuyến bay	32
Bảng 4.5	Thời gian vận hành của chuyến bay	32
Bảng 4.6	Thời gian di chuyển của chuyến bay	33
Bảng 4.7	Thời gian trễ của chuyến bay	33
Bảng 4.8	Dữ liệu khác của chuyến bay	34
Bảng 4.9	Cấu hình của cụm Hadoop	40
Bảng 5.1	Hiệu suất câu truy vấn trên Trino	56

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	Ý nghĩa
API	Giao diện lập trình ứng dụng (Application Programming Interface)
CSV	Tập giá trị phân cách bằng dấu phẩy (Comma Sperated Value)
DAG	Đồ thị có hướng không chu trình (Directed Acyclic Graph)
ETL	Trích xuất, biến đổi, tải (Extract, Transform, Load)
HDFS	Hệ thống lưu trữ dữ liệu phân tán của Hadoop (Hadoop Distributed File System)
SQL	Ngôn ngữ truy vấn có cấu trúc (Structure Query Language)
YARN	Trình Điều Phối Tài Nguyên (Yet Another Resource Negotiator)

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong lĩnh vực hàng không, dữ liệu chuyến bay ngày càng trở nên đa dạng và khổng lồ, bao gồm thông tin về lịch trình, tình trạng bay, thời gian trễ, thời tiết, và nhiều yếu tố khác. Để khai thác hiệu quả nguồn dữ liệu này, các doanh nghiệp cần xây dựng một hệ thống phân tích dữ liệu chuyến bay nhằm cung cấp thông tin chi tiết, hỗ trợ ra quyết định và tối ưu hóa hoạt động. Tuy nhiên, một trong những thách thức lớn nhất là lựa chọn mô hình triển khai và hệ sinh thái công nghệ phù hợp để đáp ứng nhu cầu xử lý dữ liệu lớn.

Hiện nay, có ba mô hình triển khai chính đó là: kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu, mỗi mô hình đều mang lại lợi ích riêng trong việc quản lý và phân tích dữ liệu. Việc lựa chọn hệ sinh thái công nghệ như Hadoop, Amazon Web Services hay Google Cloud Platform cũng rất quan trọng, vì chúng cung cấp công cụ mạnh mẽ nhưng đi kèm chi phí và yêu cầu triển khai khác nhau. Với hệ thống phân tích dữ liệu như chuyến bay, cần đầu tư tài nguyên và đảm bảo hệ thống hoạt động ổn định để đáp ứng nhu cầu thực tiễn.

Do đó, để xây dựng một hệ thống phân tích dữ liệu chuyến bay hiệu quả, cần cân nhắc kỹ lưỡng giữa các yếu tố như khả năng lưu trữ, tốc độ xử lý, chi phí đầu tư, và mức độ bảo mật. Quyết định đúng đắn sẽ giúp doanh nghiệp không chỉ tối ưu hóa việc quản lý và phân tích dữ liệu mà còn tăng khả năng cạnh tranh, giảm thiểu rủi ro và hỗ trợ đưa ra các quyết định chiến lược trong hoạt động kinh doanh.

1.2 Mục tiêu và phạm vi đề tài

Như đã khái quát trong phần 1.1, các hệ thống lưu trữ dữ liệu bao gồm kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu, mỗi loại có ưu và nhược điểm riêng. Kho dữ liệu được thiết kế với cấu trúc cao, hỗ trợ phân tích chuyên sâu và ra quyết định, nhưng chi phí duy trì cao và khó xử lý dữ liệu không cấu trúc. Hồ dữ liệu linh hoạt hơn, cho phép lưu trữ dữ liệu thô với chi phí thấp nhưng tập trung vào lưu trữ hơn là xử lý. Hồ kho dữ liệu kết hợp ưu điểm của cả hai, hỗ trợ lưu trữ dữ liệu thô và phân tích mạnh mẽ, nhưng gặp khó khăn trong tối ưu hóa truy vấn phức tạp và đảm bảo chất lượng dữ liệu.

Khi lựa chọn công nghệ cho hệ thống, các nền tảng mã nguồn mở như Apache Hadoop, Apache Spark và các dịch vụ đám mây như Amazon Web Services, Google Cloud Platform, Microsoft Azure đều cung cấp các công cụ mạnh mẽ, giúp nâng cao hiệu quả của kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu. Hệ sinh thái Hadoop

phổ biến trong xử lý và lưu trữ dữ liệu lớn, trong khi Google BigQuery và Amazon Redshift là các giải pháp kho dữ liệu đám mây nhanh và linh hoạt, nhưng đi kèm chi phí cao và phụ thuộc vào nền tảng đám mây tương ứng.

Do đó, mục tiêu và phạm vi của đồ án này là xây dựng một hệ thống phân tích dữ liệu chuyển bay bằng cách kết hợp các ưu điểm vượt trội của mô hình hồ dữ liệu và hệ sinh thái Hadoop. Cụ thể, hệ thống tận dụng tính linh hoạt của hồ dữ liệu trong việc lưu trữ mọi loại dữ liệu, bao gồm dữ liệu có cấu trúc, không cấu trúc và bán cấu trúc, cùng khả năng dễ dàng tích hợp với các công cụ phân tích dữ liệu tiên tiến như học máy, trực quan hóa và truy vấn dữ liệu phức tạp. Bên cạnh đó, đồ án còn khai thác thế mạnh của hệ sinh thái Hadoop, bao gồm khả năng lưu trữ và xử lý khối lượng dữ liệu lớn trên các hệ thống phân tán, chi phí đầu tư thấp nhờ sử dụng phần cứng tiêu chuẩn, và sự hỗ trợ mạnh mẽ từ các công cụ như HDFS, MapReduce, Hive, Spark.

Hệ thống được thiết kế để không chỉ lưu trữ dữ liệu chuyển bay từ các nguồn khác nhau mà còn thực hiện toàn bộ chu trình xử lý, từ thu thập dữ liệu theo thời gian thực, xử lý dữ liệu theo lô, đến phân tích, truy vấn và trực quan hóa. Đặc biệt, hệ thống này hướng đến việc tối ưu hóa hiệu suất, giảm chi phí vận hành và đáp ứng được yêu cầu triển khai thực tế trong các doanh nghiệp hàng không, hỗ trợ họ ra quyết định nhanh chóng và hiệu quả dựa trên dữ liệu.

1.3 Định hướng giải pháp

Để giải quyết bài toán đặt ra, em đã tiến hành chia nhỏ bài toán và xử lý yêu cầu theo từng bước đó là: thu thập dữ liệu, lưu trữ dữ liệu, xử lý dữ liệu và trực quan hóa dữ liệu. Đầu tiên về dữ liệu chuyển bay, em đã chọn một bộ dữ liệu trên Kaggle lấy từ bảng dữ liệu "Marketing Carrier On-Time Performance" của cơ sở dữ liệu "On-Time" trên trang thư viện dữ liệu TranStats [1] để đảm bảo có một bộ dữ liệu đủ lớn, có nhiều thông tin giá trị cho việc phân tích và xử lý. Và để có thể mô phỏng một luồng dữ liệu trong thời gian thực, em đã xây dựng lên một máy chủ lưu trữ dữ liệu nguồn thô ban đầu để có thể liên tục trả về dữ liệu khi được gọi API lấy dữ liệu.

Em sẽ sử dụng Apache Kafka để xử lý các luồng dữ liệu từ nhiều nguồn, chuyển tiếp chúng vào hệ thống để phân tích mà không làm gián đoạn hoạt động. Để tự động hóa các quy trình ETL và đảm bảo rằng dữ liệu luôn được làm mới và chuẩn hóa, Apache Airflow được sử dụng để lên lịch và quản lý các tác vụ ETL. Airflow giúp tự động hóa các bước nhập dữ liệu từ các nguồn bên ngoài vào hồ dữ liệu và kiểm soát quá trình chuyển đổi và tải dữ liệu vào kho lưu trữ chính.

Dữ liệu sau đó được lưu trữ an toàn, phân tán, có khả năng truy xuất nhanh

chóng trên nhiều máy chủ trong HDFS dưới định dạng file Parquet. Khi dữ liệu đã được lưu trữ trong hồ dữ liệu, Apache Spark được sử dụng để xử lý và phân tích dữ liệu song song, mạnh mẽ và hiệu quả, trong đó Spark SQL cho các truy vấn SQL có cấu trúc và Spark Streaming cho việc xử lý dữ liệu theo thời gian thực.

Em sử dụng Apache Hive giúp tổ chức dữ liệu thành các bảng có cấu trúc và tăng hiệu quả trong việc truy xuất thông tin. Ngoài ra, hệ thống còn triển khai Trino cung cấp khả năng truy vấn dữ liệu phân tán trên hồ dữ liệu kết nối thông qua Hive. Cuối cùng, Apache Superset được sử dụng để tạo ra các biểu đồ và báo cáo trực quan từ kết quả phân tích dữ liệu chuyên bay.

Hệ thống được triển khai trên nền tảng Kubernetes, mang lại khả năng mở rộng linh hoạt, tự động hóa cao và quản lý hiệu quả, đáp ứng tốt các yêu cầu xử lý dữ liệu lớn. Kubernetes cho phép điều chỉnh tài nguyên dựa trên khối lượng công việc, giám sát và cô lập các thành phần để bảo vệ dữ liệu và giảm thiểu rủi ro. Tính linh hoạt và tự động hóa không chỉ tối ưu hóa hiệu suất và chi phí vận hành mà còn đảm bảo hệ thống đáp ứng nhanh chóng nhu cầu trong môi trường thực tế.

1.4 Bố cục đề án

Phần còn lại của báo cáo đề án tốt nghiệp này được tổ chức như sau.

Chương 2 trình bày khảo sát nhu cầu sử dụng hồ dữ liệu trong các hệ thống phân tích dữ liệu chuyên bay, giới thiệu về tổng quan chức năng của hệ thống và chỉ ra các yêu cầu chức năng, phi chức năng của hệ thống.

Chương 3 giới thiệu về các công nghệ và lý do sử dụng công nghệ trong hệ sinh thái Hadoop ứng với các chức năng cụ thể.

Chương 4 sẽ làm rõ phần thiết kế và triển khai hệ thống từ kiến trúc tổng quan hệ thống tới từng mô đun cụ thể bao gồm lập lịch tác vụ, thu thập dữ liệu, lưu trữ dữ liệu, xử lý dữ liệu, truy vấn dữ liệu, trực quan hoá dữ liệu, và triển khai hệ thống thông qua cụm Kubernetes trên nền tảng ảo.

Chương 5 trình bày về kết quả thực nghiệm thu được sau khi triển khai hệ thống trên cụm Kubernetes. Các kết quả này bao gồm kết quả hoạt động của các mô-đun đã triển khai ở trên.

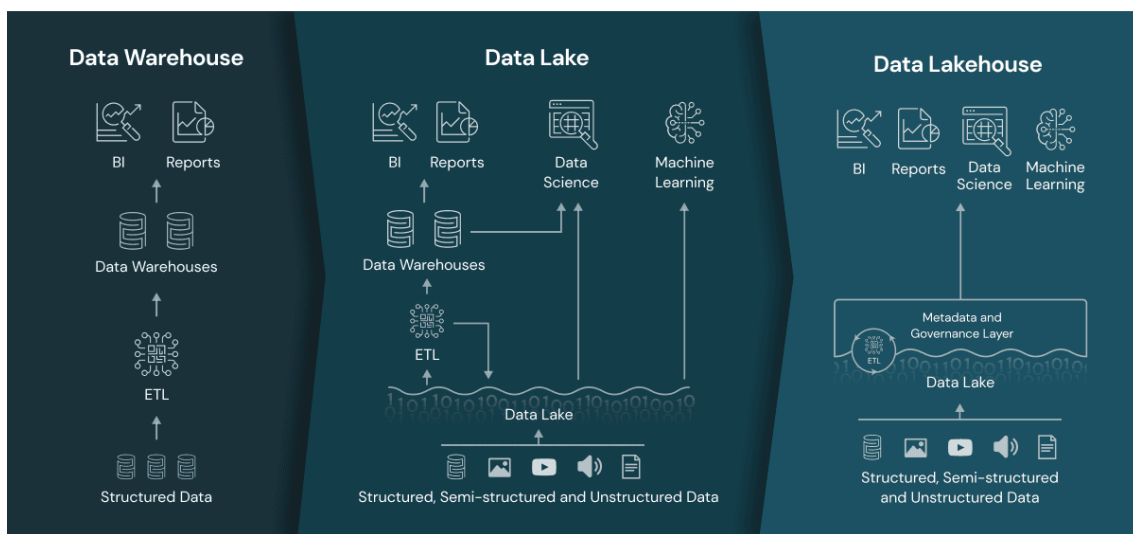
Chương 6 sẽ đi đến kết luận về đề án, trình bày những ưu và nhược điểm của hệ thống và trình bày về hướng phát triển trong tương lai.

CHƯƠNG 2. KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU

Trong chương 2, em sẽ trình bày khảo sát hiện trạng của các doanh nghiệp nói chung và các hãng hàng không nói riêng liên quan đến quá trình hình thành ý tưởng để xây dựng lên hệ thống cho đề án này. Tiếp đó, em sẽ đề xuất các yêu cầu cơ bản của hệ thống, bao gồm cả yêu cầu chức năng và phi chức năng.

2.1 Khảo sát hiện trạng

Hiện nay, các doanh nghiệp và tổ chức đang phải đối mặt với khối lượng dữ liệu ngày càng lớn và phức tạp, đòi hỏi một hệ thống dữ liệu lớn mạnh mẽ để thu thập, lưu trữ, và xử lý hiệu quả. Các hệ thống phổ biến hiện nay như kho dữ liệu (Data Warehouse), hồ dữ liệu (Data Lake) và hồ kho dữ liệu (Data Lakehouse) là các kiến trúc được lựa chọn tùy theo mục đích và nhu cầu của doanh nghiệp.



Hình 2.1: Kiến trúc kho dữ liệu, hồ dữ liệu, hồ kho dữ liệu. Nguồn [2]

Việc lựa chọn hệ thống lưu trữ và xử lý dữ liệu phù hợp trở thành một quyết định chiến lược, mỗi loại đều có những đặc điểm riêng biệt và phù hợp với các nhu cầu khác nhau của doanh nghiệp. Tuy nhiên, việc hiểu rõ ưu điểm và nhược điểm của từng kiến trúc này là rất quan trọng để có thể đưa ra quyết định đúng đắn. So sánh chi tiết ba hệ thống này, nhằm giúp các doanh nghiệp hiểu rõ hơn về cách mỗi hệ thống có thể hỗ trợ họ trong việc lưu trữ và xử lý dữ liệu lớn.

Tiêu chí	Kho dữ liệu	Hồ dữ liệu	Hồ kho dữ liệu
Loại dữ liệu	Dữ liệu có cấu trúc	Dữ liệu có cấu trúc, phi cấu trúc, bán cấu trúc	Dữ liệu có cấu trúc và phi cấu trúc
Môi trường sử dụng	Phân tích dữ liệu có cấu trúc, báo cáo kinh doanh	Lưu trữ dữ liệu lớn, dữ liệu thô hoặc dữ liệu phi cấu trúc	Kết hợp giữa phân tích dữ liệu có cấu trúc và phi cấu trúc
Chi phí	Thường cao do yêu cầu phần cứng và phần mềm đắt tiền	Chi phí lưu trữ thấp hơn	Chi phí cao hơn hồ dữ liệu nhưng thấp hơn kho dữ liệu
Mở rộng	Mở rộng có thể tốn kém và phức tạp	Dễ dàng mở rộng, có thể lưu trữ lượng dữ liệu khổng lồ	Mở rộng linh hoạt, nhưng yêu cầu công nghệ phức tạp
Phân tích dữ liệu	Tối ưu cho phân tích dữ liệu có cấu trúc	Hạn chế đối với phân tích dữ liệu phức tạp hoặc SQL	Hỗ trợ phân tích dữ liệu có cấu trúc và phi cấu trúc một cách hiệu quả
Truy vấn	Hỗ trợ mạnh mẽ cho truy vấn SQL	Hỗ trợ SQL hạn chế, cần các công cụ bổ sung như Hive	Hỗ trợ SQL, kết hợp với khả năng phân tích dữ liệu phi cấu trúc
Bảo mật	Quản lý bảo mật tốt, quyền truy cập chi tiết	Quản lý bảo mật khó khăn đối với dữ liệu không cấu trúc	Hỗ trợ bảo mật mạnh mẽ nhưng phức tạp hơn trong việc quản lý dữ liệu

Bảng 2.1: So sánh kho dữ liệu, hồ dữ liệu và hồ kho dữ liệu

Các hãng hàng không thường lựa chọn hệ thống hồ dữ liệu khi triển khai dữ liệu lớn vì khả năng lưu trữ linh hoạt và đa dạng các loại dữ liệu, bao gồm có cấu trúc, bán cấu trúc và phi cấu trúc, phù hợp với nguồn dữ liệu phức tạp như thông tin hành khách, nhật ký hoạt động máy bay, và dữ liệu cảm biến. Hồ dữ liệu cho phép tích hợp nhanh chóng dữ liệu từ nhiều hệ thống khác nhau, như quản lý đặt vé, lịch bay, bảo trì, và phân tích hành vi khách hàng. Hơn nữa, hệ thống này hỗ trợ phân tích thời gian thực và chuyên sâu, giúp dự đoán nhu cầu khách hàng, tối ưu hóa giá vé, phân tích bảo trì để ngăn ngừa sự cố và xử lý các tình huống khẩn cấp. So với kho dữ liệu truyền thống, hồ dữ liệu có chi phí thấp hơn nhờ sử dụng các nền tảng đám mây và cung cấp môi trường lý tưởng để phát triển ứng dụng AI và học máy, phục vụ dự đoán hành vi khách hàng, tối ưu lịch trình và nâng cao an toàn bay. Với

khả năng mở rộng linh hoạt, hồ dữ liệu đáp ứng tốt sự phát triển nhanh chóng về khối lượng dữ liệu trong ngành hàng không.

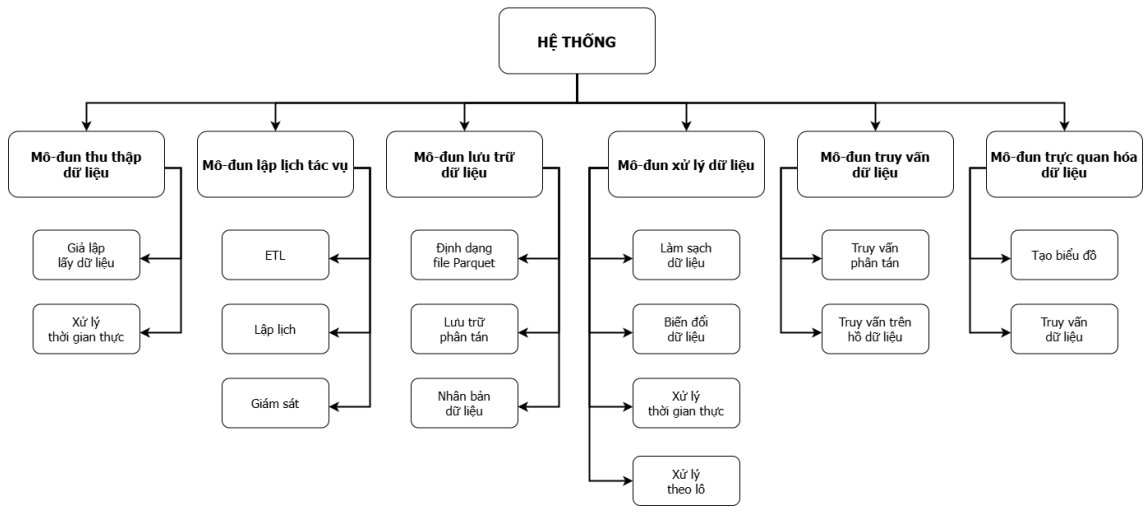
Mặc dù ngày nay có ngày càng nhiều công nghệ mới nổi trong lĩnh vực dữ liệu lớn, hệ sinh thái Hadoop vẫn giữ một vị trí quan trọng trong việc lưu trữ và xử lý dữ liệu phân tán quy mô lớn. Hadoop, với các thành phần cốt lõi như HDFS và YARN, đã mở ra một cuộc cách mạng trong việc xử lý và lưu trữ dữ liệu lớn từ nhiều nguồn khác nhau, đặc biệt là dữ liệu phi cấu trúc. Mặc dù các công nghệ mới và các giải pháp lưu trữ đám mây mang đến nhiều cải tiến về hiệu suất và khả năng tích hợp, Hadoop vẫn được nhiều doanh nghiệp lựa chọn nhờ vào khả năng xử lý dữ liệu khối lượng lớn, chi phí hợp lý và tính linh hoạt trong việc mở rộng. Hệ sinh thái Hadoop có thể tích hợp với các công nghệ hiện đại khác như Apache Spark, Apache Hive, Apache HBase và Trino, giúp tăng cường khả năng phân tích và truy vấn dữ liệu, đồng thời vẫn duy trì vai trò quan trọng trong các hệ thống dữ liệu lớn, đặc biệt là trong những môi trường yêu cầu xử lý dữ liệu theo lô hoặc lưu trữ dữ liệu phi cấu trúc. Vì vậy, dù có sự xuất hiện của nhiều công nghệ tiên tiến, Hadoop vẫn là một phần không thể thiếu trong chiến lược dữ liệu lớn của nhiều doanh nghiệp.

Các hãng hàng không thường lựa chọn các công nghệ trong hệ sinh thái Hadoop khi triển khai dữ liệu lớn vì khả năng xử lý và lưu trữ dữ liệu khổng lồ một cách hiệu quả nhờ kiến trúc phân tán, phù hợp với lượng dữ liệu đa dạng. Với chi phí hợp lý nhờ sử dụng phần cứng thông thường, Hadoop cung cấp khả năng mở rộng linh hoạt từ vài máy chủ đến hàng nghìn máy, đáp ứng nhu cầu tăng trưởng dữ liệu nhanh chóng. Hệ sinh thái Hadoop bao gồm nhiều công cụ mạnh mẽ như HDFS để lưu trữ dữ liệu, MapReduce để xử lý song song, Hive và Impala để truy vấn dữ liệu, Spark để phân tích nhanh, và Kafka hoặc Spark Streaming để xử lý thời gian thực. Và đặc biệt với đặc trưng là hệ thống mã nguồn mở, Hadoop mang lại tính linh hoạt cao, cho phép tùy chỉnh và tích hợp với các công nghệ khác mà không phụ thuộc vào nhà cung cấp độc quyền.

2.2 Tổng quan chức năng

2.2.1 Yêu cầu chức năng

Hệ thống dữ liệu lớn được em xây dựng trong đồ án sẽ bao gồm các mô-đun cơ bản nhằm hỗ trợ lưu trữ, xử lý và phân tích dữ liệu quy mô lớn một cách hiệu quả được thể hiện qua sơ đồ phân rã chức năng sau:



Hình 2.2: Sơ đồ phân rã chức năng hệ thống

Đầu tiên, mô-đun lập lịch tác vụ đảm bảo tự động hóa quá trình thu thập, xử lý và chuyển giao dữ liệu, để điều phối toàn bộ quy trình từ thu thập, xử lý, lưu trữ đến phân tích, hệ thống sử dụng Apache Airflow để lập lịch và giám sát các luồng công việc. Hệ thống có khả năng thu thập và tiếp nhận dữ liệu từ nhiều nguồn khác nhau, bao gồm cả dữ liệu có cấu trúc, phi cấu trúc và dữ liệu thời gian thực được giả lập qua API với Flask trên máy chủ Python. Dữ liệu thô này được chuyển tiếp và quản lý luồng một cách đáng tin cậy bằng Apache Kafka chính là hoạt động chính của mô-đun thu thập dữ liệu. Sau khi thu thập, dữ liệu được lưu trữ phân tán, được nhân bản với định dạng file Parquet trong mô-đun lưu trữ dữ liệu sử dụng HDFS.

Mô-đun xử lý dữ liệu chịu trách nhiệm làm sạch, chuẩn hóa và chuyển đổi dữ liệu thời gian thực với Spark Streaming, biến đổi dữ liệu thành dạng phù hợp cho phân tích. Dữ liệu đã xử lý được tổ chức thành các bảng truy vấn bằng Apache Hive, cho phép sử dụng các truy vấn SQL quen thuộc, và kết hợp với Spark SQL để thực hiện xử lý dữ liệu theo lô và phân tích phức tạp. Mô-đun truy vấn dữ liệu của hệ thống hỗ trợ truy vấn phân tán trên các nguồn dữ liệu của hồ dữ liệu thông qua Trino.

Hệ thống cung cấp khả năng trực quan hóa dữ liệu thông qua Apache Superset đóng vai trò là mô-đun trực quan hóa dữ liệu của hệ thống, giúp hiển thị kết quả phân tích dưới dạng bảng điều khiển, biểu đồ và báo cáo trực quan, hỗ trợ người dùng đưa ra quyết định nhanh chóng. Cuối cùng, hệ thống cần được triển khai trong môi trường container hóa với Kubernetes, nhằm đảm bảo khả năng triển khai linh hoạt, quản lý tài nguyên hiệu quả và hỗ trợ các yêu cầu đa nền tảng trong các môi trường phát triển, thử nghiệm và sản xuất.

2.2.2 Yêu cầu phi chức năng

Hệ thống cần đáp ứng các yêu cầu phi chức năng quan trọng để đảm bảo hiệu quả, độ tin cậy và khả năng mở rộng.

Thứ nhất, hệ thống phải có khả năng mở rộng cao, cho phép xử lý khối lượng dữ liệu lớn và gia tăng nhanh chóng từ các nguồn dữ liệu chuyển bay trong thời gian thực mà không ảnh hưởng đến hiệu suất, tối ưu hóa cho việc khai thác dữ liệu hiệu quả.

Thứ hai, hệ thống phải đảm bảo tính sẵn sàng và độ tin cậy cao, với khả năng dự phòng dữ liệu và xử lý lỗi, đặc biệt trong các thành phần như Kafka, HDFS và Kubernetes, để giảm thiểu thời gian ngừng hoạt động, đảm bảo không mất mát trong quá trình truyền tải.

Thứ ba, hệ thống cần đạt được hiệu suất cao, với khả năng xử lý dữ liệu thời gian thực và thực hiện các truy vấn phân tích phức tạp trong thời gian ngắn. Hệ thống cũng cần phải có khả năng tích hợp tốt, đảm bảo hệ thống có thể dễ dàng kết nối và tương tác với các công cụ, nền tảng hoặc hệ thống bên ngoài đặc biệt là API và các dịch vụ dữ liệu đám mây phổ biến hiện nay.

Thứ tư, hệ thống phải được thiết kế để dễ bảo trì, với cấu trúc rõ ràng, khả năng giám sát hiệu suất và ghi log đầy đủ để hỗ trợ việc phát hiện và xử lý sự cố nhanh chóng, giúp các tác vụ được thực hiện đúng thời gian và không bị gián đoạn.

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

Trong chương trước, em đã trình bày về việc khảo sát thực trạng cùng các kiến trúc hệ thống lưu trữ và xử lý dữ liệu hiện có. Trong chương này, em sẽ tập trung trình bày cơ sở lý thuyết cũng như các công nghệ mà em lựa chọn để xây dựng hệ thống. Các công nghệ được giới thiệu sẽ đi từ Apache Hadoop tới các thành phần trong hệ sinh thái của Hadoop ứng với từng chức năng riêng biệt trong hệ thống và cuối cùng là công nghệ triển khai trên nền tảng ảo hóa Kubernetes.

3.1 Apache Hadoop

Apache Hadoop [3] là một framework mã nguồn mở được thiết kế để xử lý và lưu trữ khối lượng dữ liệu lớn trong các môi trường phân tán. Hadoop cung cấp một hệ sinh thái mạnh mẽ cho việc xử lý và lưu trữ dữ liệu ở quy mô lớn, bao gồm nhiều thành phần cốt lõi, trong đó HDFS và YARN là hai thành phần quan trọng nhất.

3.1.1 HDFS

HDFS là hệ thống tệp phân tán của Hadoop, được thiết kế để lưu trữ dữ liệu lớn trên cụm máy tính với khả năng chịu lỗi cao. Kiến trúc master-slave của HDFS gồm NameNode quản lý siêu dữ liệu và DataNode lưu trữ dữ liệu thực tế. Dữ liệu được chia thành các khối lớn (128MB hoặc 256MB) để tối ưu hóa hiệu suất truyền tải mạng. Với cơ chế sao lưu mặc định 3 bản sao trên các DataNode, HDFS đảm bảo tính sẵn sàng và khả năng chịu lỗi, cho phép truy cập dữ liệu ngay cả khi một số node gặp sự cố. HDFS phù hợp cho các tác vụ đọc/ghi tuần tự và phân tích dữ liệu lớn với ít thay đổi.

3.1.2 YARN

YARN là hệ thống quản lý tài nguyên và lập lịch công việc của Hadoop, giúp điều phối và phân bổ tài nguyên trên cụm máy tính. Bằng cách tách biệt quản lý tài nguyên và lập lịch công việc, YARN cải thiện tính linh hoạt và khả năng mở rộng. YARN gồm ba thành phần chính: ResourceManager (điều phối tài nguyên toàn cụm), NodeManager (theo dõi tài nguyên sẵn có trên mỗi node), và ApplicationMaster (quản lý, giám sát thực thi công việc của từng ứng dụng). Hệ thống này hỗ trợ chạy nhiều loại ứng dụng, từ MapReduce truyền thống đến các framework hiện đại như Apache Spark và Apache Tez.

Sự kết hợp giữa HDFS và YARN trong Apache Hadoop cung cấp một nền tảng mạnh mẽ, có thể mở rộng và linh hoạt để lưu trữ và xử lý dữ liệu lớn. HDFS đảm bảo dữ liệu được lưu trữ an toàn và có thể truy cập nhanh chóng, trong khi YARN

điều phối và tối ưu hóa việc sử dụng tài nguyên của hệ thống, cho phép các ứng dụng chạy hiệu quả và nhanh chóng trên các cụm máy tính phân tán. Với sự kết hợp giữa HDFS và YARN, Apache Hadoop mang lại một nền tảng linh hoạt, mạnh mẽ cho việc lưu trữ và xử lý dữ liệu lớn, giúp các tổ chức khai thác tối đa giá trị từ dữ liệu của họ.

3.2 Apache Spark

Apache Spark [4] là một framework mã nguồn mở mạnh mẽ được thiết kế để xử lý dữ liệu lớn với tốc độ cao và khả năng tính toán phân tán, đặc biệt nổi bật với khả năng xử lý dữ liệu trong bộ nhớ, giúp giảm thiểu thời gian trễ so với các hệ thống truyền thống như Hadoop MapReduce. Spark không chỉ hỗ trợ các công việc xử lý theo lô truyền thống mà còn có khả năng xử lý dữ liệu thời gian thực, phân tích trực tuyến, và học máy, nhờ vào một hệ sinh thái rộng lớn với nhiều thư viện tích hợp. Hai thành phần quan trọng trong hệ sinh thái Spark là Spark SQL và Spark Streaming, mỗi thành phần đều mang lại những khả năng mạnh mẽ cho việc xử lý và phân tích dữ liệu.

3.2.1 Spark SQL

Spark SQL là một module của Spark, hỗ trợ xử lý dữ liệu có cấu trúc và bán cấu trúc thông qua giao diện SQL hoặc API DataFrame/Dataset. Nó cung cấp cú pháp SQL tiêu chuẩn và sử dụng Catalyst Optimizer để tối ưu hóa truy vấn, cải thiện hiệu suất xử lý dữ liệu. Spark SQL hỗ trợ các nguồn dữ liệu phổ biến như Hive, JDBC, JSON, Parquet, và tích hợp tốt với Apache Hive, cho phép truy vấn dữ liệu trong HDFS hoặc HBase, đồng thời duy trì các tính năng như phân vùng dữ liệu và hàm phân tích.

3.2.2 Spark Streaming

Spark Streaming là một module của Apache Spark, thiết kế để xử lý và phân tích dữ liệu trong thời gian thực bằng cách chia luồng dữ liệu thành các micro-batch và xử lý chúng liên tục. Khác với các hệ thống streaming truyền thống, Spark Streaming xử lý dữ liệu trong bộ nhớ, giúp đạt hiệu suất cao khi xử lý khối lượng lớn. Nó có thể kết hợp với các nguồn dữ liệu streaming như Apache Kafka, Amazon Kinesis, Flume và tích hợp với Spark SQL, cho phép thực hiện các truy vấn SQL phức tạp trên dữ liệu streaming.

Bên cạnh đó, sự tích hợp giữa Apache Spark và Hadoop YARN, cho phép quản lý tài nguyên hiệu quả trong các môi trường phân tán. Với vai trò là nền tảng quản lý tài nguyên và lập lịch tác vụ trong hệ sinh thái Hadoop, YARN hỗ trợ Spark chạy trên cùng một cụm Hadoop với các ứng dụng khác như MapReduce, Hive hoặc HBase, đảm bảo chia sẻ tài nguyên và tối ưu hóa hiệu suất.

3.2.3 Spark YARN

Apache Spark hoạt động trên YARN với hai chế độ chính: Client Mode và Cluster Mode. Trong Client Mode, Driver chạy trên máy khách, phù hợp cho các ứng dụng tương tác nhưng không thích hợp cho ứng dụng dài hạn. Các Executor được YARN quản lý trên các node trong cụm. Trong Cluster Mode, Driver chạy trên container trong cụm YARN, tách biệt với máy khách, đảm bảo tính ổn định và cho phép ứng dụng tiếp tục chạy ngay cả khi máy khách ngắt kết nối. Cluster Mode thích hợp cho các ứng dụng dài hạn và môi trường sản xuất, dù phức tạp hơn trong triển khai. Việc lựa chọn giữa hai chế độ phụ thuộc vào yêu cầu cụ thể của ứng dụng.

Sự kết hợp giữa Spark và YARN mang lại nhiều lợi ích, bao gồm tận dụng tài nguyên chung, mở rộng quy mô linh hoạt, tích hợp với các công cụ trong hệ sinh thái Hadoop, khả năng chịu lỗi cao và quản lý tài nguyên động. Khi chạy trên YARN, Spark sử dụng một ApplicationMaster để quản lý ứng dụng, yêu cầu tài nguyên từ ResourceManager và khởi động các container để thực thi tác vụ. Spark YARN phù hợp với xử lý dữ liệu lớn, môi trường dùng chung và tối ưu hóa hiệu suất, kết hợp hiệu suất của Spark với khả năng quản lý tài nguyên hiệu quả của YARN.

3.3 Apache Airflow

Apache Airflow [5] là một nền tảng mã nguồn mở mạnh mẽ dùng để tự động hóa, lập lịch và giám sát các luồng công việc phức tạp trong môi trường dữ liệu lớn. Airflow cho phép người dùng định nghĩa các luồng công việc dưới dạng đồ thị có hướng không chu trình (DAG), với các tác vụ có thể thực thi song song hoặc tuần tự. Điểm mạnh của Airflow là khả năng tích hợp với nhiều công nghệ trong hệ sinh thái dữ liệu lớn và đám mây, giúp xây dựng các pipeline dữ liệu phức tạp.

Airflow hỗ trợ tích hợp với nhiều công nghệ phổ biến như Apache Hadoop, Apache Spark, Apache Kafka, AWS, GCP và Microsoft Azure. Người dùng có thể xây dựng các DAG để quản lý job Hadoop hoặc Spark, như chạy các tác vụ MapReduce hoặc Spark jobs trên HDFS và giám sát trạng thái của chúng. Các Operator của Airflow, như HadoopOperator và SparkSubmitOperator, giúp dễ dàng tương tác với các công nghệ này và thực thi các tác vụ trên Hadoop hoặc submit các job Spark. Airflow tích hợp với Apache Kafka để xử lý dữ liệu thời gian thực, cho phép gửi và nhận thông điệp từ các topic Kafka. Các DAG có thể theo dõi sự kiện từ Kafka và kích hoạt các tác vụ tiếp theo khi có dữ liệu mới. Ngoài ra, Airflow còn hỗ trợ tích hợp với các dịch vụ đám mây như AWS Lambda, Google Cloud Storage, BigQuery, và Azure Data Lake, giúp chạy các tác vụ trên các nền tảng này.

Airflow có khả năng tích hợp với các công cụ ETL, hệ thống quản lý cơ sở dữ liệu như MySQL, PostgreSQL, MongoDB, và các dịch vụ API, giúp xây dựng các luồng công việc phức tạp. Người dùng có thể thu thập dữ liệu từ nhiều nguồn, thực hiện xử lý dữ liệu và lưu trữ kết quả vào các hệ thống cơ sở dữ liệu hoặc kho dữ liệu đám mây. Nhờ khả năng tích hợp mạnh mẽ và linh hoạt, Apache Airflow đã trở thành công cụ lý tưởng cho việc xây dựng và quản lý các luồng xử lý dữ liệu phức tạp trong môi trường dữ liệu lớn, giúp tự động hóa và tối ưu hóa quy trình xử lý dữ liệu và ứng dụng phân tích.

3.4 Apache Kafka

Apache Kafka [6] là nền tảng phân tán mạnh mẽ dành cho truyền tải dữ liệu thời gian thực, xử lý dữ liệu lớn với độ trễ thấp và tính khả dụng cao. Kafka hoạt động theo mô hình thuê bao/xuất bản, với dữ liệu được ghi vào các topic và tiêu thụ bởi các consumer. Kiến trúc Kafka bao gồm các thành phần như Kafka topic (lưu trữ thông điệp), Kafka partition (lưu trữ thông điệp theo thứ tự Kafka offset), và Kafka broker (lưu trữ và gửi thông điệp). Kafka sử dụng nhiều broker và partition nhân bản để đảm bảo tính sẵn sàng và hiệu suất.

Apache Kafka tích hợp dễ dàng với Spark Streaming, giúp xây dựng các luồng xử lý dữ liệu thời gian thực hiệu quả. Kafka cung cấp dữ liệu thời gian thực (như sự kiện hoặc log hệ thống), trong khi Spark Streaming xử lý và phân tích dữ liệu này bằng cách chia nhỏ thành các micro-batch và xử lý song song. Sự kết hợp giữa Kafka và Spark Streaming tạo ra các hệ thống phân tích dữ liệu thời gian thực mạnh mẽ. Khi tích hợp với Spark Streaming, Kafka cung cấp giao diện Kafka DStream (Direct Stream), cho phép Spark dễ dàng tiêu thụ dữ liệu từ các topic Kafka. DStream là kiểu dữ liệu trong Spark Streaming đại diện cho dòng dữ liệu từ các nguồn, giúp Spark tiếp cận dữ liệu streaming nhanh chóng và hiệu quả. Điều này giúp xây dựng các ứng dụng phân tích dữ liệu thời gian thực, như phân tích log, phát hiện gian lận, và giám sát sự kiện hệ thống, bằng cách sử dụng Kafka để thu thập sự kiện và gửi đến Spark Streaming để xử lý.

Spark Streaming và Kafka hỗ trợ "cam kết giao hàng ít nhất một lần" (at-least-once delivery guarantee), đảm bảo mỗi sự kiện được xử lý ít nhất một lần, ngay cả khi có sự cố trong quá trình truyền tải. Điều này đảm bảo tính chính xác và đáng tin cậy trong các ứng dụng yêu cầu xử lý dữ liệu thời gian thực, như hệ thống giám sát hoặc khuyến nghị. Với khả năng tích hợp mạnh mẽ giữa Apache Kafka và Apache Spark Streaming, các tổ chức có thể xây dựng các giải pháp phân tích dữ liệu thời gian thực, giúp theo dõi và phản ứng nhanh chóng với các sự kiện và dữ liệu mới, tạo ra những ứng dụng có khả năng mở rộng và đáp ứng các nhu cầu phức tạp về

xử lý dữ liệu.

3.5 Trino

Trino [7] là hệ thống truy vấn phân tán mã nguồn mở, được thiết kế để xử lý các truy vấn dữ liệu quy mô lớn và kết hợp dữ liệu từ nhiều nguồn mà không di chuyển dữ liệu. Trino tối ưu hóa hiệu suất và khả năng mở rộng, cho phép phân tích dữ liệu phức tạp trên các kho dữ liệu phân tán mà không gặp vấn đề về hiệu suất hay quy mô. Hệ thống hỗ trợ SQL chuẩn và kết nối với nhiều hệ thống lưu trữ dữ liệu như Hadoop HDFS, Apache Hive, Amazon S3, Google BigQuery, MySQL, PostgreSQL, Apache Cassandra, và các dịch vụ đám mây khác.

Trino có tính năng truy vấn liên kết (federated query), cho phép thực hiện truy vấn trên dữ liệu phân tán từ nhiều nguồn mà không cần di chuyển hoặc sao chép dữ liệu. Điều này giúp tối ưu hóa chi phí, thời gian và bảo mật dữ liệu, đồng thời giữ cho dữ liệu dễ dàng truy cập từ các hệ thống phân tán khác nhau. Ví dụ, Trino có thể kết nối với kho dữ liệu lớn trên HDFS và cơ sở dữ liệu quan hệ như MySQL trong một truy vấn duy nhất, mang lại tính linh hoạt cao cho các ứng dụng phân tích dữ liệu.

Trino sử dụng kiến trúc máy chủ truy vấn SQL phân tán, trong đó mỗi nút đảm nhận một phần công việc truy vấn, giúp xử lý dữ liệu nhanh chóng và hiệu quả. Các truy vấn được phân tán và xử lý đồng thời, tối ưu hóa thời gian phản hồi và giảm thiểu độ trễ. Trino hỗ trợ multi-stage query execution, tối ưu hóa các bước truy vấn phức tạp như lọc, tính toán, và kết hợp dữ liệu từ nhiều nguồn, giúp cải thiện hiệu quả công việc phân tích và báo cáo dữ liệu.

Trino có khả năng mở rộng linh hoạt, chạy trên các cụm máy chủ lớn, truy vấn dữ liệu quy mô petabyte mà không ảnh hưởng đến hiệu suất. Nó hoạt động hiệu quả trong môi trường đám mây hoặc on-premises và tích hợp với các công cụ phân tích như Apache Superset, Tableau, và các công cụ BI khác. Với kiến trúc connector-based, Trino cho phép thêm các connector mới để kết nối với nhiều nguồn dữ liệu khác, mở rộng khả năng tích hợp với các hệ thống lưu trữ và xử lý dữ liệu.

Một điểm mạnh của Trino là khả năng tối ưu hóa truy vấn phức tạp thông qua cost-based optimization, ước tính chi phí các chiến lược truy vấn và chọn phương án tối ưu nhất. Điều này giúp giảm thiểu thời gian thực thi và mang lại hiệu suất vượt trội cho hệ thống xử lý dữ liệu lớn trong thời gian ngắn. Trino là lựa chọn lý tưởng cho các tổ chức cần phân tích dữ liệu nhanh chóng, mở rộng và linh hoạt trong môi trường dữ liệu lớn, nhờ khả năng kết hợp dữ liệu từ nhiều nguồn, thực hiện truy vấn SQL phức tạp và tối ưu hóa hiệu suất trong môi trường phân tán.

3.6 Apache Hive

Apache Hive [8] là một nền tảng phân tích dữ liệu lớn trên HDFS, sử dụng SQL-like query language để giúp người dùng truy vấn và xử lý dữ liệu mà không cần viết code MapReduce phức tạp. Hive hỗ trợ các truy vấn có cấu trúc, tối ưu hóa cho môi trường Hadoop, và tích hợp tốt với các công nghệ khác như Apache Spark, HDFS và Trino, mở rộng khả năng phân tích dữ liệu lớn trong các môi trường đa dạng.

Tích hợp với HDFS là điểm mạnh của Hive, cho phép lưu trữ dữ liệu phân tán và thực hiện truy vấn mà không cần di chuyển dữ liệu. Dữ liệu trong Hive được lưu dưới các định dạng như văn bản, Parquet, ORC, Avro trên HDFS, giúp tối ưu hóa chi phí lưu trữ và xử lý dữ liệu lớn. HDFS cung cấp khả năng chịu lỗi cao và mở rộng, giúp Hive hoạt động hiệu quả trong môi trường phân tán.

Tích hợp với Apache Spark giúp cải thiện hiệu suất của Hive bằng cách sử dụng Spark engine thay thế MapReduce để thực thi các truy vấn, giảm độ trễ và tăng hiệu suất, đặc biệt đối với các công việc tính toán phức tạp. Người dùng có thể thực hiện truy vấn SQL trên Hive và tận dụng khả năng xử lý trong bộ nhớ của Spark để phân tích dữ liệu lớn nhanh chóng và hiệu quả.

Tích hợp với Trino giúp Hive mở rộng khả năng xử lý và truy vấn dữ liệu, cho phép thực hiện các truy vấn liên kết giữa Hive và các nguồn dữ liệu khác như MySQL, PostgreSQL, hay kho dữ liệu đám mây. Trino, với khả năng tối ưu hóa truy vấn mạnh mẽ, giúp thực hiện các truy vấn SQL phức tạp trên dữ liệu phân tán mà không gặp phải vấn đề về hiệu suất, mang lại khả năng truy vấn dữ liệu từ nhiều hệ thống phân tán trong một lần thực thi.

Với khả năng tích hợp mạnh mẽ với HDFS, Spark và Trino, Apache Hive trở thành một giải pháp lý tưởng cho các tổ chức cần một công cụ mạnh mẽ để lưu trữ, truy vấn và phân tích dữ liệu lớn trong môi trường phân tán, đồng thời tận dụng sức mạnh của các công nghệ phân tán hiện đại để xử lý dữ liệu với hiệu suất tối ưu.

3.7 Apache Superset

Apache Superset [9] là công cụ mã nguồn mở mạnh mẽ cho phân tích và trực quan hóa dữ liệu, cho phép tạo báo cáo, bảng điều khiển và biểu đồ trực quan từ dữ liệu phức tạp. Một trong những tính năng nổi bật của Superset là khả năng tích hợp linh hoạt với Trino, giúp thực hiện các truy vấn SQL trên dữ liệu phân tán từ nhiều nguồn khác nhau mà không cần di chuyển dữ liệu. Sự tích hợp này giúp truy vấn và phân tích dữ liệu lớn hiệu quả, truy xuất nhanh chóng từ các kho dữ liệu phân tán như Hadoop, Amazon S3, Google BigQuery, MySQL, PostgreSQL và nhiều hệ

thống khác.

Khi tích hợp với Trino, Superset tận dụng khả năng xử lý truy vấn mạnh mẽ của Trino để truy vấn dữ liệu từ nhiều nguồn mà không cần chuyển dữ liệu sang hệ thống trung gian, giúp tiết kiệm thời gian và tài nguyên. Điều này cho phép Superset thực hiện các truy vấn phân tán và kết hợp dữ liệu từ nhiều nguồn mà không gặp phải vấn đề về hiệu suất hay phức tạp trong việc di chuyển hoặc sao chép dữ liệu. Kết hợp này mang lại nhiều lợi ích, đặc biệt trong môi trường dữ liệu phân tán, với khả năng xử lý truy vấn SQL phức tạp và truy vấn dữ liệu theo thời gian thực, giúp các phân tích trong Superset luôn được cập nhật và chính xác. Superset sau đó có thể trực quan hóa kết quả này qua biểu đồ và bảng điều khiển, giúp người dùng phân tích và ra quyết định dựa trên dữ liệu.

Việc tích hợp Superset với Trino mang lại lợi ích lớn về khả năng mở rộng và tính linh hoạt trong việc kết nối với các hệ thống dữ liệu đám mây và on-premises. Superset có thể sử dụng Trino để truy vấn dữ liệu từ nhiều dịch vụ đám mây như Amazon Redshift, Google BigQuery, hoặc kho dữ liệu Hadoop như HDFS mà không gặp phải các vấn đề về hiệu suất hay khả năng mở rộng của công cụ phân tích truyền thống. Tính kết hợp mạnh mẽ này cho phép người dùng khai thác và trực quan hóa dữ liệu phân tán từ nhiều nguồn khác nhau một cách hiệu quả. Trino tối ưu hóa tốc độ và độ chính xác của các truy vấn SQL phức tạp, trong khi Superset cung cấp giao diện trực quan, giúp người dùng dễ dàng tương tác và đưa ra quyết định thông minh dựa trên dữ liệu phân tích.

3.8 Kubernetes

Kubernetes [10] là một hệ thống mã nguồn mở mạnh mẽ, được phát triển bởi Google và hiện duy trì bởi Cloud Native Computing Foundation (CNCF), giúp tự động hóa việc triển khai, quản lý và mở rộng các ứng dụng container trong môi trường đám mây hoặc hạ tầng vật lý. Kubernetes quản lý các container trong cụm phân tán, cho phép triển khai và vận hành ứng dụng nhất quán trên nhiều môi trường khác nhau, đồng thời giảm thiểu sự phức tạp và tăng cường khả năng mở rộng cho các ứng dụng phân tán quy mô lớn.

Một trong những tính năng nổi bật của Kubernetes là quản lý container tự động, bao gồm việc tự động triển khai, mở rộng và duy trì các container trong cụm. Kubernetes sử dụng Pod, đơn vị triển khai cơ bản chứa một hoặc nhiều container, để nhóm các container liên quan lại với nhau. Các Pod này được duy trì và triển khai bởi các Kubernetes controllers như ReplicaSet, đảm bảo số lượng Pod luôn ổn định. Deployment trong Kubernetes cho phép cập nhật ứng dụng mà không làm gián đoạn dịch vụ, giúp triển khai phiên bản mới một cách mượt mà.

Tính năng mở rộng tự động của Kubernetes giúp ứng dụng điều chỉnh số lượng PodPod linh hoạt theo nhu cầu thực tế, tối ưu hóa tài nguyên và giảm chi phí. Điều này rất quan trọng trong môi trường đám mây, nơi tài nguyên có thể thay đổi tùy thuộc vào lượng truy cập hoặc tải công việc. Kubernetes hỗ trợ mở rộng cả về quy mô chiều ngang (thêm Pod khi cần) và chiều dọc (tăng cường tài nguyên cho các Pod hiện tại). Kubernetes hỗ trợ quản lý dịch vụ mạng, giúp các pod giao tiếp với nhau và với dịch vụ bên ngoài thông qua đối tượng Service. Service giúp điều phối và phân phối lưu lượng giữa các pod một cách linh hoạt và ổn định, với các tính năng như phát hiện dịch vụ dựa trên DNS, cân bằng tải và chính sách mạng để kiểm soát truy cập và bảo mật.

Kubernetes hỗ trợ quản lý trạng thái và lưu trữ thông qua Persistent Volume (PV) và Persistent Volume Claim (PVC), giúp cung cấp khả năng lưu trữ dữ liệu lâu dài cho các ứng dụng container. Điều này đảm bảo các ứng dụng duy trì trạng thái của mình mà không bị mất dữ liệu, ngay cả khi các Pod bị di dời hoặc tái khởi động. Kubernetes cung cấp hệ thống quản lý cấu hình và bảo mật qua Secret và ConfigMap, giúp quản lý thông tin nhạy cảm như mật khẩu và chứng chỉ SSL. Kubernetes cũng hỗ trợ kiểm soát quyền truy cập với RBAC (Role-Based Access Control), giúp giới hạn quyền truy cập và hành động trên hệ thống.

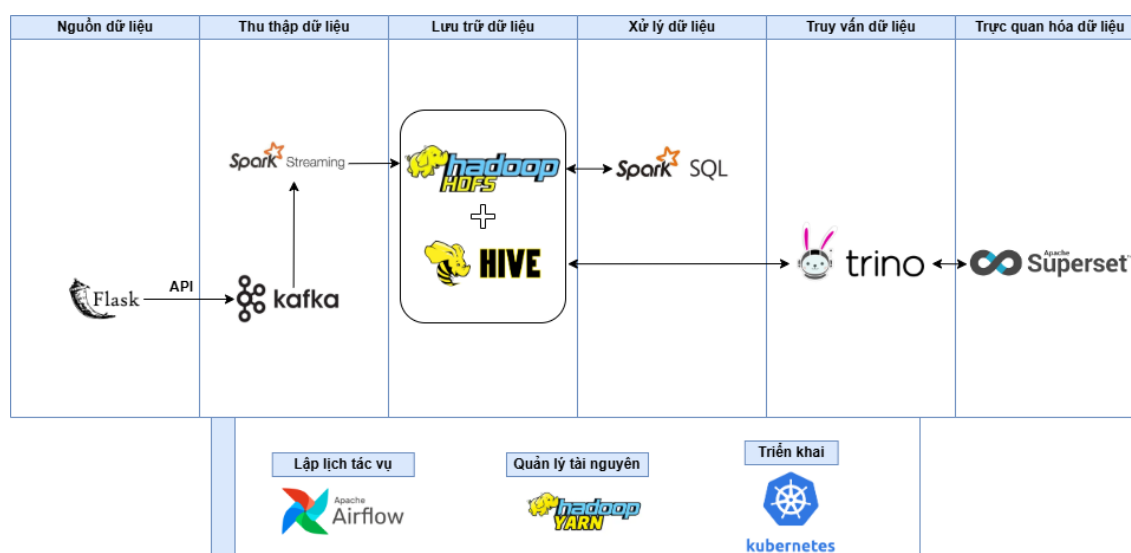
Với khả năng hỗ trợ các tính năng như quản lý container tự động, mở rộng linh hoạt, quản lý dịch vụ mạng, và bảo mật mạnh mẽ, Kubernetes đã trở thành tiêu chuẩn công nghiệp trong việc triển khai và quản lý ứng dụng container, đặc biệt trong các môi trường đám mây và hạ tầng phân tán. Nó mang lại khả năng mở rộng, tính sẵn sàng cao, và tính linh hoạt vượt trội, cho phép các tổ chức triển khai và vận hành các ứng dụng phân tán với độ tin cậy và hiệu quả cao.

CHƯƠNG 4. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG

Chương 2 đã giới thiệu về nhu cầu lưu trữ và xử lý dữ liệu lớn trong doanh nghiệp, đồng thời đưa ra kiến trúc tổng quan của hệ thống cùng các thành phần chính. Các thành phần này được liên kết với các công nghệ cụ thể, đã được phân tích chi tiết trong chương 3. Trong chương này, em sẽ tập trung vào việc thiết kế hệ thống để đảm bảo đáp ứng đầy đủ các yêu cầu của một hệ thống hồ dữ liệu phân tích dữ liệu chuyển bay, sau đó triển khai hệ thống trên nền tảng Kubernetes.

4.1 Thiết kế hệ thống

4.1.1 Kiến trúc tổng quan hệ thống



Hình 4.1: Kiến trúc tổng quan hệ thống

Hình 4.1 là kiến trúc tổng quan hệ thống do em đề xuất. Luồng xử lý dữ liệu trong hệ thống bắt đầu từ việc giả lập quá trình thu thập dữ liệu chuyển bay qua việc gọi API từ ứng dụng Flask, nơi dữ liệu được gửi đến Apache Kafka đảm nhiệm vai trò cổng giao tiếp và hàng đợi tin nhắn. Sau đó dữ liệu được chuyển tiếp cho Spark Streaming để tiếp tục xử lý.

Spark Streaming chịu trách nhiệm xử lý dữ liệu luồng theo thời gian thực, thực hiện các thao tác như chuyển đổi, lọc, và phân tích dữ liệu trước khi lưu trữ. Dữ liệu được em lưu trữ dưới định dạng Parquet giúp giảm tài nguyên lưu trữ và tối ưu cho việc truy vấn dữ liệu. Đồng thời, dữ liệu cũng được em lưu trữ phân tán, bền vững trong hệ thống HDFS. Tài nguyên tính toán cung cấp cho Apache Spark để xử lý dữ liệu của hệ thống được quản lý bởi Hadoop YARN, giúp điều phối tài nguyên và tối ưu hóa các tác vụ xử lý, đảm bảo hiệu suất cao cho hệ thống.

Dữ liệu trong HDFS xử lý thông qua Spark SQL hỗ trợ tốt cho cả dữ liệu theo lô và dữ liệu thời gian thực. Để đảm bảo tính tự động hóa và tối ưu hóa quy trình xử lý, Apache Airflow được triển khai để quản lý các luồng công việc, giúp điều phối và lập lịch các tác vụ, đảm bảo tính liên tục và chính xác trong toàn bộ quy trình xử lý dữ liệu.

Khi dữ liệu được chuẩn bị sẵn sàng, Trino với sự kết hợp với Apache Hive để có thể kết nối tới HDFS sẽ đóng vai trò là công cụ truy vấn phân tán, cho phép người dùng thực hiện các truy vấn phức tạp và kết hợp dữ liệu từ nhiều nguồn khác nhau với tốc độ nhanh và độ trễ thấp. Trino sau đó chuyển dữ liệu đã phân tích, truy vấn, với kết quả cuối là các dataset đến Apache Superset, hỗ trợ tạo các biểu đồ, bảng biểu, và báo cáo tương tác, mang lại cái nhìn trực quan về dữ liệu để hỗ trợ ra quyết định.

Toàn bộ hệ thống được triển khai trên nền Kubernetes, đảm bảo khả năng mở rộng linh hoạt, quản lý hiệu quả tài nguyên và duy trì tính sẵn sàng cao, đáp ứng yêu cầu của các hệ thống hồ dữ liệu hiện đại. Sau đây, em sẽ đi vào thiết kế chi tiết từng thành phần của hệ thống trên.

4.1.2 Mô-đun thu thập dữ liệu

Đầu tiên, em sẽ xây dựng mô-đun thu thập dữ liệu bằng cách giả lập quá trình này thông qua việc tạo một máy chủ sử dụng ngôn ngữ lập trình Python và framework Flask. Máy chủ này sẽ đóng vai trò là nơi lưu trữ dữ liệu thô ban đầu, mô phỏng nguồn dữ liệu thực tế. Bằng cách này, máy chủ có khả năng cung cấp dữ liệu theo thời gian thực khi có yêu cầu thông qua API. Đây sẽ là nền tảng để thử nghiệm và đánh giá các chức năng xử lý dữ liệu trong các bước tiếp theo.

```

1 from flask import Flask, request, jsonify
2 import pandas as pd
3
4 app = Flask(__name__)
5
6 @app.route('/api/get_data', methods=['GET'])
7 def get_data():
8     try:
9         year = request.args.get('year')
10        month = request.args.get('month')
11        offset = int(request.args.get('offset'))
12        limit = int(request.args.get('limit'))
13
14        df = pd.read_csv(f'/data/Flights_{year}_{month}.csv')
15        cnt = len(df)
16

```

```

17         if offset + limit >= cnt:
18             result = df.iloc[offset:cnt-offset]
19             return jsonify({'status': 'complete', 'data': result.
to_dict(orient='records')})
20
21         result = df.iloc[offset:offset+limit]
22         return jsonify({'status': 'success', 'data': result.
to_dict(orient='records')})
23     except Exception as e:
24         print(e)
25         return jsonify({'status': 'error', 'data': 'error'})
26
27 if __name__ == '__main__':
28     app.run(debug=True, port=5000)

```

Đoạn mã nguồn trên tạo một API phục vụ dữ liệu từ tệp CSV dựa trên các tham số truy vấn. Đầu tiên, Flask được khởi tạo với "app = Flask(__name__)". API có đường dẫn là "/api/get_data", hỗ trợ phương thức "GET", và lấy các tham số "year", "month", "offset", "limit" từ yêu cầu HTTP. Dữ liệu được đọc từ tệp CSV theo định dạng tên "Flights_year_month.csv" bằng thư viện pandas. API trả về một phần dữ liệu dựa trên "offset" và "limit", hoặc toàn bộ phần còn lại nếu vượt quá giới hạn số dòng của tệp. Nếu có lỗi, API trả về thông báo lỗi với trạng thái 'error'. Cuối cùng, ứng dụng chạy trên cổng 5000 với chế độ gỡ lỗi bật.

Dữ liệu sẽ được thu thập định kỳ hàng tháng thông qua mô-đun lập lịch tác vụ trong phần 4.1.3, đảm bảo tự động hóa quá trình lấy dữ liệu. Cụ thể, mô-đun này sẽ thực hiện các yêu cầu gọi API liên tục trong một khoảng thời gian cho đến khi toàn bộ dữ liệu của tháng đó được tải về đầy đủ. Sau khi dữ liệu được thu thập, nó sẽ được gửi đi theo từng phần và đẩy liên tục lên các topic tương ứng trong hệ thống hàng đợi thông điệp của cụm Kafka.

```

1 from confluent_kafka import Producer
2 import requests
3 import json
4
5 def delivery_callback(err, msg):
6     if err:
7         print('ERROR: Message failed delivery: {}'.format(err))
8     else:
9         print("Produced event to topic {topic}: key = {key:12}".
format(topic=msg.topic(), key=msg.key().decode('utf-8')))
10
11 def extract_data_def():
12     global year

```

```

13     global month
14
15     config = {'bootstrap.servers': 'kafka:30092', 'acks': 'all'}
16     producer = Producer(config)
17
18     topic = f'flight_data_{year}'
19     url = 'http://service:5000/api/get_data'
20     params = {'year': year, 'month': month, 'offset': 0, 'limit':
21               100}
22
23     while True:
24         try:
25             r = requests.get(url=url, params=params)
26             data = r.json()
27
28             if data['status'] == 'error':
29                 break
30
31             if data['status'] == 'success' or data['status'] == '
complete':
32                 key = str(year) + '_' + str(month) + '_' + str(
33                     params['offset'])
34                 print(key)
35                 value = json.dumps(data['data'])
36                 producer.produce(topic, value, key, callback=
37                     delivery_callback)
38
39                 if data['status'] == 'complete':
40                     break
41
42                 params['offset'] += 100
43         except Exception as e:
44             print(e)
45             break
46
47     producer.flush()

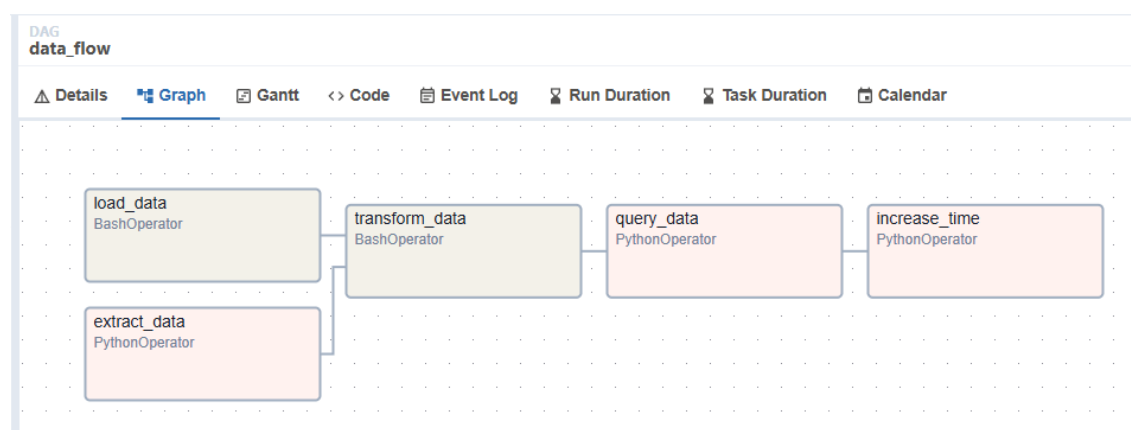
```

Đoạn mã nguồn trên khởi tạo một hàm cho một PythonOperator thuộc một DAG trong Airflow (Chi tiết trong 4.1.3) thực hiện việc lấy dữ liệu từ API được tạo ra từ máy chủ Flask ở trên và gửi dữ liệu lên các topic tương ứng theo năm lên hệ thống Kafka. Hàm "delivery_callback" kiểm tra và thông báo kết quả của việc gửi dữ liệu, hiển thị lỗi nếu có. Trong hàm "extract_data_def", một đối tượng "Producer" của Kafka được khởi tạo với cấu hình gồm địa chỉ cụm Kafka và chế độ xác nhận. Hàm này xác định topic dựa trên biến "year" và sử dụng API để tải dữ liệu theo từng phần

nhỏ dựa trên tham số "offset" và "limit". Sau mỗi lần lấy dữ liệu thành công, dữ liệu được chuyển đổi thành chuỗi JSON và gửi lên Kafka với một khóa xác định, đồng thời offset được tăng dần để lấy các phần tiếp theo. Quá trình này lặp lại cho đến khi API báo trạng thái "complete" hoặc xảy ra lỗi. Cuối cùng, "producer.flush()" được gọi để đảm bảo tất cả các tin nhắn được gửi hoàn tất.

4.1.3 Mô-đun lập lịch tác vụ

Tiếp theo, em xây dựng mô-đun lập lịch tác vụ bằng cách sử dụng Airflow để thiết kế luồng xử lý dữ liệu cho hệ thống. Trong Airflow, em tạo một biểu đồ phụ thuộc công việc DAG, trong đó mỗi tác vụ được đại diện bởi một operator (Hình 4.2). Mỗi operator đảm nhiệm một giai đoạn cụ thể trong quy trình xử lý dữ liệu: thu thập, xử lý, lưu trữ và truy vấn. DAG giúp kết nối các tác vụ theo trình tự logic, đảm bảo luồng dữ liệu diễn ra liên tục, không bị gián đoạn. Ngoài ra, Airflow cung cấp các công cụ giám sát và kiểm tra lỗi, giúp theo dõi trạng thái thực thi của từng tác vụ, phát hiện và xử lý lỗi kịp thời, từ đó đảm bảo tính ổn định và hiệu quả của toàn bộ hệ thống.



Hình 4.2: Đồ thị luồng xử lý dữ liệu trên Airflow

Đồ thị trong hình 4.2 có PythonOperator "extract_data" đã được mô tả trong phần 4.1.3, hai BashOperator "load_data" và "transform_data" sẽ chạy các câu lệnh giúp nạp các tệp mã nguồn Spark lên trên cụm YARN để xử lý dữ liệu, được mô tả chi tiết trong phần 4.1.6. Còn 2 PythonOperator "query_data" và "increase_time" được em xây dựng như sau:

```

1 import os
2
3 def query_data_def():
4     global year
5     global month
6
7     string = ''
  
```



```

8     with open('/opt/airflow/sql/trino/template/month.sql', 'r')
as f:
9         string = f.read()
10
11        string_replace = string.replace('{year}', str(year)).replace(
'{month}', str(month))
12        with open(f'/opt/airflow/sql/trino/month/month_{year}_{month}
}.sql', 'w') as f:
13            f.write(string_replace)
14
15        os.system(f'cd /opt/airflow/source && ./trino --server http
://trino:8080 --file /opt/airflow/sql/trino/month/month_{year}
_{month}.sql')
16
17        if month % 3 == 0:
18            string = ''
19            with open('/opt/airflow/sql/trino/template/quarter.sql',
'r') as f:
20                string = f.read()
21
22                quarter = int(month / 3)
23                string_replace = string.replace('{year}', str(year)).
replace('{quarter}', str(quarter))
24                with open(f'/opt/airflow/sql/trino/quarter/quarter_{year}
_{quarter}.sql', 'w') as f:
25                    f.write(string_replace)
26
27                os.system(f'cd /opt/airflow/source && ./trino --server
http://trino:8080 --file /opt/airflow/sql/trino/quarter/
quarter_{year}_{quarter}.sql')
28
29                if month % 12 == 0:
30                    string = ''
31                    with open('/opt/airflow/sql/trino/template/year.sql',
'r') as f:
32                        string = f.read()
33
34                        string_replace = string.replace('{year}', str(year))
35                        with open(f'/opt/airflow/sql/trino/year/year_{year}.
sql', 'w') as f:
36                            f.write(string_replace)
37
38                        os.system(f'cd /opt/airflow/source && ./trino --
server http://trino:8080 --file /opt/airflow/sql/trino/year/
year_{year}.sql')

```

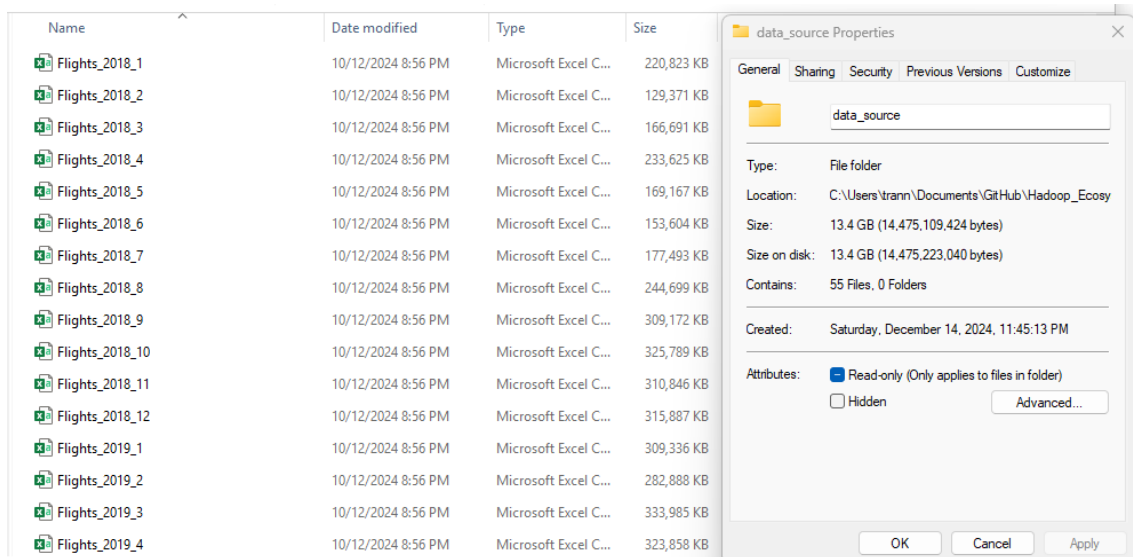
Đoạn mã nguồn trên khởi tạo một hàm cho PythonOperator "query_data" thực hiện việc tạo và chạy các truy vấn SQL trên hệ thống Trino, với logic xử lý dữ liệu theo tháng, quý, và năm. Hàm "query_data_def" đầu tiên đọc một tệp mẫu SQL "month.sql", thay thế các placeholder "{year}" và "{month}" bằng giá trị cụ thể của "year" và "month", sau đó ghi truy vấn đã cập nhật vào một tệp mới. Tệp này được thực thi trên Trino thông qua lệnh hệ thống "os.system". Nếu "month" là tháng cuối của một quý, mã nguồn sẽ tiếp tục xử lý tương tự với tệp mẫu "quarter.sql", thay thế "{year}" và "{quarter}" bằng giá trị tương ứng, và thực thi truy vấn SQL cho cả quý. Cuối cùng, nếu "month" là tháng 12, một truy vấn tương tự được tạo từ tệp mẫu "year.sql" để xử lý dữ liệu tổng kết của cả năm. Tất cả các truy vấn được thực hiện thông qua cli của Trino với đường dẫn và tệp SQL tương ứng, việc truy vấn dữ liệu trên Trino sẽ được mô tả chi tiết trong phần 4.1.7.

Đối với hàm sử dụng cho PythonOperator "increase_time" sử dụng PySpark để xử lý dữ liệu thời gian được lưu trữ trong hệ thống HDFS cho dữ liệu được thu thập định kỳ hàng tháng, em đã điều chỉnh quy trình thu thập dữ liệu. Thay vì thu thập dữ liệu theo đúng chu kỳ hàng tháng, em rút ngắn khoảng thời gian giả lập để phù hợp với tiến độ thực hiện đồ án. Đầu tiên, nó khởi tạo một "SparkSession" và đọc tệp CSV từ đường dẫn "hdfs://hadoop-hadoop-hdfs-nn:9000/time", sau đó lấy dòng đầu tiên của dữ liệu để cập nhật giá trị của "year" và "month". Hàm "increase_time_def" được định nghĩa để tăng thời gian: nếu "month" là 12, giá trị "year" sẽ tăng lên 1 và "month" đặt lại thành 1; nếu không, chỉ tăng "month" thêm 1. Sau khi tính toán giá trị mới, một DataFrame Spark được tạo ra từ dữ liệu cập nhật và ghi đè trở lại vào cùng đường dẫn trong HDFS với tùy chọn "header=true".

4.1.4 Mô-đun lưu trữ dữ liệu

Trong hệ thống có 3 nơi lưu trữ dữ liệu đó là: (i) dữ liệu nguồn tại máy chủ Flask cung cấp API; (ii) dữ liệu thời gian thực được lưu trữ tại cụm Kafka; (iii) dữ liệu lưu trên HDFS.

Thứ nhất, về dữ liệu nguồn tại máy chủ Flask, dữ liệu được lưu trữ dưới dạng các file CSV có tên dưới dạng: "Flights_{year}_{month}", trong đó "year" là năm và "month" là tháng của file dữ liệu. Tổng dung lượng của dữ liệu nguồn khoảng 14.5 GiB và mô tả trên được thể hiện qua hình 4.3.



Hình 4.3: Dữ liệu nguồn tại máy chủ Flask

Thứ hai, về dữ liệu thời gian thực được lưu trữ tại cụm Kafka được chia theo các topic, partition, offset. Trong đó topic được chia theo các năm, cụm Kafka sẽ có 3 partition và được nhân bản 3 lần để đảm bảo dữ liệu được thu thập và lưu trữ an toàn.

Thứ ba, về dữ liệu lưu trữ trên HDFS là nơi lưu trữ dữ liệu chính của hệ thống. HDFS lưu trữ dữ liệu sau quá trình tiền xử lý Spark Streaming, dữ liệu sau quá trình xử lý theo lô của Spark SQL và các bảng dữ liệu được tạo ra sau quá trình truy vấn dữ liệu bằng Trino. Dữ liệu trên HDFS được em tổ chức lưu trữ như sau:

```

1 staging/
2   |_ 2018/
3     |_ 1/
4       |_ file.Parquet
5       |_ ...
6     |_ ...
7   |_ ...
8 processed_data/
9   |_ process_data.db/
10    |_ 2018/
11      |_ 1/
12        |_ file.Parquet
13        |_ ...
14      |_ ...
15    |_ ...
16 datasets/
17   |_ datasets_month/
18     |_ marketing_airline_network_2018_1/
19     |_ flights_across_days_of_the_week_2020_2/

```

```

20     |_ ...
21   |_ datasets_quarter/
22     |_ marketing_airline_network_2018_1/
23     |_ flights_across_days_of_the_week_2019_3/
24     |_ ...
25   |_ datasets_year/
26     |_ marketing_airline_network_2018/
27     |_ flights_across_days_of_the_week_2021/
28     |_ ...
29   |_ datasets/
30     |_ flights_all_year_month/
31     |_ marketing_airline_network_all_year/
32     |_ ...

```

Trên HDFS, thư mục "staging/" được sử dụng để lưu trữ dữ liệu thô, được tổ chức theo cấu trúc phân cấp với các thư mục con theo năm và tháng. Mỗi thư mục tháng chứa các tệp dữ liệu dạng Parquet, đây là kết quả của quá trình thu thập dữ liệu. Thư mục "processed_data/" chứa dữ liệu đã qua xử lý, được tổ chức theo năm và tháng với các thư mục con lưu trữ dữ liệu sau quá trình tiền xử lý và xử lý dữ liệu. Thư mục này còn bao gồm cơ sở dữ liệu "process_data.db", nơi các tệp dữ liệu đã xử lý được lưu trữ và tổ chức theo dạng cơ sở dữ liệu bằng Spark SQL kết hợp với Hive. Thư mục "datasets/" chứa các bộ dữ liệu đầu ra được phân loại theo các nhóm thời gian như tháng, quý, năm và các loại báo cáo. Các thư mục con trong "datasets/" được đặt tên theo thời gian và loại báo cáo, ví dụ như "datasets_month/", "datasets_quarter/", "datasets_year/", và "datasets/", nơi lưu trữ các tệp dữ liệu đã được truy vấn và xử lý sẵn sàng để sử dụng cho các bước phân tích và trực quan hóa dữ liệu. Mỗi tệp dữ liệu trong các thư mục này đều có thể được sử dụng để tạo các báo cáo, bảng điều khiển, hoặc để phân tích thêm trong hệ thống.

4.1.5 Mô-đun xử lý dữ liệu

Dữ liệu sẽ được xử lý, biến đổi sử dụng Spark Streaming và Spark SQL. Quá trình này được thực hiện thông qua việc Airflow nộp mã nguồn Spark lên cụm YARN trong chế độ Client Mode. Quá trình xử lý dữ liệu gồm hai phần chính là xử lý dữ liệu theo thời gian thực (BashOperator "load_data") và xử lý dữ liệu theo lô (BashOperator "transform_data") đã nêu qua trong mô-đun lập lịch tác vụ 4.1.4, được em thiết kế chi tiết như sau:

```

1 year = int(argv[1])
2 month = int(argv[2])
3
4 spark = SparkSession.builder \
5     .appName("Load data") \

```

```

6      .config("spark.streaming.stopGracefullyOnShutdown", True) \
7      .config('spark.jars.packages', 'org.apache.spark:spark-sql-
kafka-0-10_2.12:3.5.1') \
8      .getOrCreate()
9
10 if (year == 2022 and month >= 8) or year > 2022:
11     pass
12 else:
13     df = spark.readStream \
14         .format("kafka") \
15         .option("kafka.bootstrap.servers", "kafka:30092") \
16         .option("subscribe", f"flight_data_{year}") \
17         .load()
18
19     json_df = df.selectExpr("CAST(key AS STRING) as msg_key", "
CAST(value AS STRING) as msg_value")
20
21     json_schema = StructType([
22         StructField('Year', StringType(), True),
23         StructField('Quarter', StringType(), True),
24         ...
25         StructField('Div5TailNum', StringType(), True),
26         StructField('Duplicate', StringType(), True)
27     ])
28
29     json_expanded_df = json_df.withColumn("msg_value", from_json(
json_df["msg_value"], json_schema)).select("msg_value.*")
30
31     writing_df = json_expanded_df.writeStream \
32         .format("Parquet") \
33         .option("format", "append") \
34         .option("path", "hdfs://hadoop-hadoop-hdfs-nn:9000/
staging/" + str(year) + "/" + str(month)) \
35         .option("checkpointLocation", "hdfs://hadoop-hadoop-hdfs-
nn:9000/tmp/" + str(year) + "/" + str(month)) \
36         .outputMode("append") \
37         .start()
38
39     def stop_query():
40         writing_df.stop()
41
42     timer = threading.Timer(24 * 60 * 60, stop_query)
43     timer.start()
44
45     writing_df.awaitTermination()

```

Đối với Spark Streaming, dữ liệu thời gian thực từ Kafka được xử lý và lưu trữ vào HDFS. Quá trình bắt đầu bằng việc khởi tạo một SparkSession với các cấu hình đặc biệt hỗ trợ tích hợp Kafka và xử lý dữ liệu dạng luồng thông qua "readStream". Dữ liệu được đọc từ Kafka topic "flight_data_{year}" và chuyển đổi thành định dạng chuỗi JSON. Sau đó, JSON được ánh xạ vào một schema đã xác định trước "json_schema" để phân tách thành các cột dữ liệu riêng biệt, giúp dễ dàng truy cập và thao tác. Dữ liệu sau khi được xử lý sẽ được ghi vào HDFS dưới dạng tệp Parquet, lưu tại đường dẫn "/staging/year/month". Việc ghi sử dụng chế độ "append" để thêm dữ liệu mới liên tục, đồng thời kết hợp "checkpointing" tại một đường dẫn tạm thời để đảm bảo quá trình ghi có khả năng phục hồi trong trường hợp xảy ra sự cố, tăng độ tin cậy cho hệ thống. Cuối cùng, Spark Streaming chờ hoàn tất quá trình xử lý thông qua "awaitTermination()", đảm bảo toàn bộ luồng dữ liệu được xử lý liên mạch và không bị gián đoạn. Còn quá trình xử lý dữ liệu theo lô được em thiết kế như sau:

```
1 datawarehouse_location = 'hdfs://hadoop-hadoop-hdfs-nn:9000/
   processed_data'
2 spark = SparkSession.builder.appName("Transform data").config("
   spark.sql.warehouse.dir", datawarehouse_location).
   enableHiveSupport().getOrCreate()
3
4 year = int(argv[1])
5 month = int(argv[2])
6
7 if (year == 2022 and month >= 8) or year > 2022:
8     pass
9 else:
10     df = spark.read.Parquet("hdfs://hadoop-hadoop-hdfs-nn:9000/
   staging/" + str(year) + "/" + str(month))
11
12     # Select columns not too much null
13     df_select = df.select("Year", "Quarter", "Month", "DayOfMonth",
   "DayOfWeek", "FlightDate",
14                             ...
   "FirstDepTime", "TotalAddGTime", "
15     LongestAddGTime"
16                             )
17
18     # Process duplicated rows
19     df_duplicate = df_select.dropDuplicates()
20
21     # Process date data
22     df_process_date_tmp = df_duplicate \
```

```

23     .withColumn("DepTimeBlkStart", when(length(df_duplicate.
DepTimeBlk) == 9, substring(df_duplicate.DepTimeBlk, 1, 4).
cast("int")).otherwise(-1)) \
24     .withColumn("DepTimeBlkEnd", when(length(df_duplicate.
DepTimeBlk) == 9, substring(df_duplicate.DepTimeBlk, 6, 4).
cast("int")).otherwise(-1)) \
25     .withColumn("ArrTimeBlkStart", when(length(df_duplicate.
ArrTimeBlk) == 9, substring(df_duplicate.ArrTimeBlk, 1, 4).
cast("int")).otherwise(-1)) \
26     .withColumn("ArrTimeBlkEnd", when(length(df_duplicate.
ArrTimeBlk) == 9, substring(df_duplicate.ArrTimeBlk, 6, 4).
cast("int")).otherwise(-1))
27
28     df_process_date = df_process_date_tmp \
29     .withColumn("DepTimeBlkDistance", df_process_date_tmp.
DepTimeBlkEnd - df_process_date_tmp.DepTimeBlkStart) \
30     .withColumn("ArrTimeBlkDistance", df_process_date_tmp.
ArrTimeBlkEnd - df_process_date_tmp.ArrTimeBlkStart)
31
32     # Create table if not exists
33     spark.sql("create schema if not exists processed_data")
34     spark.sql("use processed_data")
35     create_table_sql = f"""
36         create table if not exists flight_{year} (
37             Year int, Quarter int, Month int, DayofMonth int,
DayOfWeek int, FlightDate date,
38             ...
39             FirstDepTime int, TotalAddGTime float,
LongestAddGTime float
40         )
41     """
42     spark.sql(create_table_sql)
43
44     # Cast data type and Insert data
45     df_process_date.createOrReplaceTempView("temp_view")
46     insert_into_sql = f"""
47         insert into table flight_{year} select
48             cast(Year as int), cast(Quarter as int), cast(Month as
int), cast(DayofMonth as int),
49             ...
50             cast(FirstDepTime as int), cast(TotalAddGTime as float),
cast(LongestAddGTime as float)
51         from temp_view
52     """
53     spark.sql(insert_into_sql)

```

Đối với Spark SQL thực hiện xử lý và chuyển đổi dữ liệu từ HDFS, sau đó lưu trữ vào một bảng trong Hive. Đầu tiên, "SparkSession" được khởi tạo với hỗ trợ Hive, và thư mục kho dữ liệu được cấu hình tại "hdfs://hadoop-hadoop-hdfs-nn:9000/processed_data". Giá trị "year" và "month" được truyền từ dòng lệnh. dữ liệu Parquet từ HDFS trong đường dẫn "/staging/year/month" được đọc vào. Dữ liệu được chọn lọc các cột cần thiết thông qua "select", các trường dữ liệu chọn lọc được mô tả chi tiết như sau:

Bảng 4.1 mô tả thông tin về dữ liệu thời gian của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
Year	int	Năm
Quarter	int	Quý
Month	int	Tháng
DayOfMonth	int	Ngày trong tháng
DayOfWeek	int	Ngày trong tuần
FlightDate	date	Ngày bay (yyyymmdd)

Bảng 4.1: Dữ liệu thời gian của chuyến bay

Bảng 4.2 mô tả thông tin về dữ liệu hãng vận chuyển tiếp thị và hãng vận chuyển điều hành của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
Marketing_Airline_Network	varchar(255)	Mã hãng vận chuyển tiếp thị duy nhất
Operated_or_Brande d_C ode_Share_Partners	varchar(255)	Hãng vận chuyển được báo cáo điều hành hoặc đối tác chia sẻ mã thương hiệu
DOT_ID_Marketing_Air line	varchar(255)	Mã số nhận dạng được Bộ Giao thông Vận tải Hoa Kỳ cấp để xác định một hãng hàng không
IATA_Code_Marketing_ Airline	varchar(255)	Mã được IATA cấp và thường được sử dụng để nhận dạng hãng hàng không
Flight_Number_Marketin g_Airline	int	Số hiệu chuyến bay
Originally_Scheduled_Co de_Share_Airline	varchar(255)	Mã hãng vận chuyển điều hành theo lịch duy nhất
DOT_ID_Originally_Sch eduled_ Code_Share_Airline	varchar(255)	Mã số nhận dạng được Bộ Giao thông Vận tải Hoa Kỳ cấp để xác định một hãng hàng không
IATA_Code_Originally_ Scheduled_ Code_Share_Airline	varchar(255)	Mã được IATA cấp và thường được sử dụng để nhận dạng hãng hàng không
Flight_Num_Originally_ Sched- uled_Code_Share_Airline	varchar(255)	Số hiệu chuyến bay
Operating_Airline	varchar(255)	Mã hãng vận chuyển điều hành duy nhất
DOT_ID_Operating_Air line	varchar(255)	Mã số nhận dạng được Bộ Giao thông Vận tải Hoa Kỳ cấp để xác định một hãng hàng không
IATA_Code_Operating_ Airline	varchar(255)	Mã được IATA cấp để nhận dạng hãng hàng không
Tail_Number	varchar(255)	Số đuôi máy bay
Flight_Number_Operatin g_Airline	varchar(255)	Số hiệu chuyến bay

Bảng 4.2: Hãng vận chuyển tiếp thị và hãng vận chuyển điều hành của chuyến bay

Bảng 4.3 mô tả thông tin về dữ liệu địa điểm xuất phát và địa điểm đến của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
OriginAirportID	varchar(255)	Sân bay xuất phát, mã nhận dạng sân bay
OriginAirportSeq ID	varchar(255)	Sân bay xuất phát, mã nhận dạng chuỗi sân bay
OriginCityMarket ID	varchar(255)	Sân bay xuất phát, mã nhận dạng thị trường thành phố
Origin	varchar(255)	Sân bay xuất phát
OriginCityName	varchar(255)	Tên thành phố của sân bay xuất phát
OriginState	varchar(255)	Mã bang của sân bay xuất phát
OriginStateFips	varchar(255)	Mã Fips của bang sân bay xuất phát
OriginStateName	varchar(255)	Tên bang của sân bay xuất phát
OriginWac	varchar(255)	Mã khu vực thế giới của sân bay xuất phát
DestAirportID	varchar(255)	Sân bay đến, mã nhận dạng sân bay
DestAirportSeqID	varchar(255)	Sân bay đến, mã nhận dạng chuỗi sân bay
DestCityMarketID	varchar(255)	Sân bay đến, mã nhận dạng thị trường thành phố
Dest	varchar(255)	Sân bay đến
DestCityName	varchar(255)	Tên thành phố của sân bay đến
DestState	varchar(255)	Mã bang của sân bay đến
DestStateFips	varchar(255)	Mã Fips của bang sân bay đến
DestStateName	varchar(255)	Tên bang của sân bay đến
DestWac	varchar(255)	Mã khu vực thế giới của sân bay đến

Bảng 4.3: Địa điểm xuất phát và địa điểm đến của chuyến bay

Bảng 4.4 mô tả thông tin về dữ liệu thời gian khởi hành của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
CRSDepTime	int	Giờ khởi hành theo lịch trình (giờ địa phương: hhmm)
DepTime	int	Giờ khởi hành thực tế (giờ địa phương: hhmm)
DepDelay	float	Chênh lệch phút giữa giờ khởi hành dự kiến và thực tế
DepDelayMinutes	float	Chênh lệch phút giữa giờ khởi hành dự kiến và thực tế.
DepDel15	float	Chỉ báo trễ khởi hành, 15 phút trở lên (1=Có)
DepartureDelayGroups	int	Nhóm độ trễ khởi hành, chia khoảng thời gian thành các nhóm 15 phút
DepTimeBlk	int	Khoảng thời gian khởi hành, chia theo khung giờ

Bảng 4.4: Thời gian khởi hành của chuyến bay

Bảng 4.5 mô tả thông tin về dữ liệu thời gian vận hành của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
TaxiOut	float	Thời gian lăn bánh từ cổng ra đường băng (phút)
WheelsOff	int	Giờ cất cánh thực tế (hhmm, giờ địa phương)
WheelsOn	int	Giờ hạ cánh thực tế (hhmm, giờ địa phương)
TaxiIn	float	Thời gian lăn bánh từ đường băng vào cổng (phút)

Bảng 4.5: Thời gian vận hành của chuyến bay

Bảng 4.6 mô tả thông tin về dữ liệu thời gian di chuyển của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
CRSArrTime	int	Giờ đến theo lịch trình (hhmm, giờ địa phương)
ArrTime	int	Giờ đến thực tế (hhmm, giờ địa phương)
ArrDelay	float	Chênh lệch phút giữa giờ đến dự kiến và thực tế
ArrDelayMinutes	float	Chênh lệch phút giữa giờ đến dự kiến và thực tế
ArrDel15	float	Arrival Delay Indicator, 15 Minutes or More (1=Yes)
ArrivalDelayGroups	int	Nhóm độ trễ đến, chia khoảng thời gian thành các nhóm 15 phút
ArrTimeBlk	int	Khoảng thời gian đến, chia theo khung giờ

Bảng 4.6: Thời gian di chuyển của chuyến bay

Bảng 4.7 mô tả thông tin về dữ liệu thời gian trễ của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
CarrierDelay	float	Độ trễ do hãng vận chuyển gây ra (phút)
WeatherDelay	float	Độ trễ do thời tiết gây ra (phút)
NASDelay	float	Độ trễ do Hệ thống Hàng không Quốc gia (NAS) gây ra (phút)
SecurityDelay	float	Độ trễ do vấn đề an ninh (phút)
LateAircraftDelay	float	Độ trễ do máy bay đến muộn (phút)

Bảng 4.7: Thời gian trễ của chuyến bay

Bảng 4.8 mô tả thông tin về dữ liệu khác của chuyến bay.

Tên cột	Kiểu dữ liệu	Mô tả
Cancelled	float	Hủy chuyến (1=Hủy)
CRSElapsedTime	float	Thời gian dự kiến cho chuyến bay (phút)
ActualElapsedTime	float	Thời gian thực tế của chuyến bay (phút)
AirTime	float	Thời gian bay thực tế (phút)
Distance	float	Khoảng cách từ sân bay xuất phát đến sân bay đến (dặm)
DistanceGroup	int	Nhóm khoảng cách
FirstDepTime	int	Giờ khởi hành cổng đầu tiên tại sân bay xuất phát
TotalAddGTime	float	Tổng thời gian trên mặt đất cách xa cổng cho chuyến bay trở về cổng hoặc chuyến bay bị hủy
LongestAddGTime	float	Thời gian chờ lâu nhất từ cổng ra máy bay cho chuyến bay về hoặc chuyến bay bị hủy

Bảng 4.8: Dữ liệu khác của chuyến bay

Sau đó, em loại bỏ các bản ghi trùng lặp bằng "dropDuplicates", và xử lý các cột thời gian "DepTimeBlk" và "ArrTimeBlk" bằng cách tách thành các giá trị bắt đầu, kết thúc và tính khoảng cách giữa chúng. Các cột này được thêm mới vào DataFrame dưới dạng các cột tính toán. Sau đó, mã kiểm tra và tạo một schema Hive "processed_data" nếu chưa tồn tại, chuyển sang sử dụng schema này và tạo bảng Hive tương ứng cho từng năm "flight_{year}". Cuối cùng, dữ liệu được chuyển đổi kiểu dữ liệu cho các cột cần thiết và chèn vào bảng Hive từ một "temp_view". Điều này đảm bảo dữ liệu được lưu trữ có cấu trúc và sẵn sàng cho các truy vấn bằng Trino tiếp theo.

4.1.6 Mô-đun truy vấn dữ liệu

Dữ liệu sau khi được xử lý bởi Spark sẽ được truy vấn và phân tích bằng Trino. Quá trình này tập trung vào việc khai thác thông tin từ dữ liệu chuyến bay để tạo ra các báo cáo phù hợp, được phân tích theo các mốc thời gian như tháng, quý, năm và so sánh giữa các năm. Công việc này được thực hiện bởi PythonOperator "query_data" như đã trình bày trong mục 4.1.4. Các báo cáo đầu ra bao gồm: (i) phân tích mạng lưới tiếp thị hàng không; (ii) phân bố chuyến bay theo các ngày trong tuần; (iii) phân bố điểm xuất phát của các chuyến bay; (iv) hủy chuyến bay theo các ngày trong tuần; (v) tổng số chuyến bay bị hủy và phân bố theo từng hãng

hàng không; (vi) phân tích đặc điểm của các ngày trong tháng; (vii) phân tích độ trễ trung bình theo từng hãng hàng không; (viii) số lượng chuyến bay theo từng tháng qua các năm; (ix) phân bố điểm xuất phát của các chuyến bay qua từng năm. Em đã thiết kế chi tiết các câu truy vấn tương ứng cho từng báo cáo này, có thể nêu ra một số ví dụ như sau:

```

1 create table if not exists hive.datasets_month.
    total_cancellations_and_flights_per_carrier_{year}_{month}
2 with (format = 'Parquet')
3 as select
4 s.Year, s.Quarter, s.Month,
5 s.Marketing_Airline_Network,
6 (cast(count(1) as real) / b.total_flights * 100)
    percentage_flights, (cast(count_if(Cancelled = 1) as real) / b
    .total_cancellations * 100) percentage_cancellations
7 from hive.processed_data.flight_{year} s,
8 (select Year, Quarter, Month, count(1) total_flights, count_if(
    Cancelled = 1) total_cancellations from hive.processed_data.
    flight_{year} where Month = {month} group by Year, Quarter,
    Month) b
9 where s.Year = b.Year and s.Quarter = b.Quarter and s.Month = b.
    Month and s.Month = {month}
10 group by s.Year, s.Quarter, s.Month, s.Marketing_Airline_Network,
    b.total_flights, b.total_cancellations;

```

Đoạn mã nguồn trên tạo bảng theo tháng với tên "total_cancellations_and_flights_per_carrier_{year}_{month}", lưu trữ dưới định dạng Parquet. Bảng này chứa thông tin về tỷ lệ phần trăm số chuyến bay và tỷ lệ phần trăm số chuyến bay bị hủy theo từng hãng hàng không trong một tháng cụ thể. Dữ liệu được lấy từ bảng "flight_{year}" trong schema "processed_data". Đầu tiên, một bảng tạm "b" được tạo với các thông tin tổng số chuyến bay (total_flights) và tổng số chuyến bay bị hủy (total_cancellations) theo từng năm, quý, tháng, bằng cách nhóm dữ liệu từ "flight_{year}" và lọc theo tháng "{month}". Sau đó, bảng chính sử dụng bảng tạm này để tính toán tỷ lệ phần trăm số chuyến bay (percentage_flights) và tỷ lệ phần trăm chuyến bay bị hủy (percentage_cancellations) cho từng hãng hàng không. Các phép tính này được thực hiện bằng cách đếm tổng số bản ghi và đếm các bản ghi có giá trị "Cancelled = 1", kết hợp với bảng tạm để chuẩn hóa kết quả theo tỷ lệ phần trăm. Kết quả cuối cùng được nhóm theo năm, quý, tháng, và hãng hàng không.

```

1 create table if not exists hive.datasets_quarter.day_of_month_{
    year}_{quarter}
2 with (format = 'Parquet')
3 as select

```

```

4 s.Year, s.Quarter, s.Month, s.DayOfMonth,
5 (cast(count(1) as real) / b.all * 100) percentage_all, (cast(
    count_if(s.Cancelled = 1) as real) / b.cancelled * 100)
    percentage_cancelled
6 from hive.processed_data.flight_{year} s,
7 (select Year, Quarter, Month, count(1) all, count_if(Cancelled =
    1) cancelled from hive.processed_data.flight_{year} where
    Quarter = {quarter} group by Year, Quarter, Month) b
8 where s.Year = b.Year and s.Quarter = b.Quarter and s.Month = b.
    Month and s.Quarter = {quarter}
9 group by s.Year, s.Quarter, s.Month, s.DayOfMonth, b.all, b.
    cancelled;

```

Đoạn mã nguồn trên tạo bảng theo quý với tên "day_of_month_{year}_{quarter}". Bảng này cung cấp thông tin về tỷ lệ phần trăm tổng số chuyến bay và tỷ lệ phần trăm số chuyến bay bị hủy theo từng ngày trong tháng, trong một quý cụ thể của năm "{year}". Dữ liệu được lấy từ bảng "flight_{year}" trong schema "processed_data". Đầu tiên, một bảng tạm "b" được tạo để tổng hợp tổng số chuyến bay (all) và số chuyến bay bị hủy (cancelled) theo năm, quý và tháng, bằng cách lọc theo quý ({quarter}) và nhóm dữ liệu. Sau đó, bảng chính sử dụng bảng tạm này để tính tỷ lệ phần trăm tổng số chuyến bay (percentage_all) và tỷ lệ phần trăm chuyến bay bị hủy (percentage_cancelled) cho từng ngày trong tháng. Các phép tính được thực hiện bằng cách đếm tổng số bản ghi và các bản ghi có giá trị "Cancelled = 1", sau đó chuẩn hóa kết quả thành tỷ lệ phần trăm dựa trên bảng tạm. Dữ liệu cuối cùng được nhóm theo năm, quý, tháng, ngày trong tháng và các giá trị tổng hợp từ bảng tạm.

```

1 create table if not exists hive.datasets_year.
    mean_delay_by_aircraft_carrier_{year}
2 with (format = 'Parquet')
3 as select
4 Year, Quarter, Month, Marketing_Airline_Network,
5 avg(CarrierDelay) carrier_delay,
6 avg(WeatherDelay) weather_delay,
7 avg(NASDelay) nas_delay,
8 avg(SecurityDelay) security_delay,
9 avg(LateAircraftDelay) late_aircraft_delay
10 from hive.processed_data.flight_{year}
11 group by Year, Month, Quarter, Marketing_Airline_Network;

```

Đoạn mã nguồn trên tạo bảng theo năm với tên "mean_delay_by_aircraft_carrier_{year}". Bảng này cung cấp thông tin về độ trễ trung bình của các chuyến bay theo từng loại nguyên nhân, được phân nhóm theo năm, quý, tháng và mạng lưới hãng hàng

không "Marketing_Airline_Network". Dữ liệu được lấy từ bảng "flight_{year}" trong schema "processed_data". Các cột độ trễ bao gồm "CarrierDelay", "WeatherDelay", "NASDelay", "SecurityDelay" và "LateAircraftDelay" được tính giá trị trung bình (avg) cho từng nhóm dữ liệu. Kết quả cuối cùng được nhóm theo các cột "Year", "Quarter", "Month" và "Marketing_Airline_Network", cung cấp cái nhìn chi tiết về mức độ trễ trung bình của các hãng hàng không trong năm "{year}".

```

1 create table if not exists hive.datasets.flights_all_year_month
2 with (format = 'Parquet')
3 as select ((Year - 2000) * 100 + Month) year_month, count(1) cnt
4   from hive.processed_data.flight_2018 group by Year, Month
5 union
6 select ((Year - 2000) * 100 + Month) year_month, count(1) cnt
7   from hive.processed_data.flight_2019 group by Year, Month
8 union
9 select ((Year - 2000) * 100 + Month) year_month, count(1) cnt
10  from hive.processed_data.flight_2020 group by Year, Month
11 union
12 select ((Year - 2000) * 100 + Month) year_month, count(1) cnt
13  from hive.processed_data.flight_2021 group by Year, Month
14 union
15 select ((Year - 2000) * 100 + Month) year_month, count(1) cnt
16  from hive.processed_data.flight_2022 group by Year, Month;

```

Đoạn mã nguồn tạo bảng giữa các năm với tên "flights_all_year_month", lưu trữ dưới định dạng Parquet, để tổng hợp số lượng chuyến bay theo từng tháng qua các năm. Mỗi bản ghi trong bảng chứa cột "year_month" là giá trị duy nhất đại diện cho năm và tháng (tính bằng công thức $((Year - 2000) * 100 + Month)$) và "cnt" là tổng số chuyến bay trong tháng đó. Dữ liệu được lấy từ các bảng "flight_2018", "flight_2019", "flight_2020", "flight_2021" và "flight_2022" thuộc schema "processed_data". Các kết quả theo năm được gộp lại thông qua phép "union", đảm bảo bảng cuối cùng chứa toàn bộ dữ liệu chuyến bay từ năm 2018 đến 2022, được nhóm theo cột "Year" và "Month".

4.1.7 Mô-đun trực quan hóa dữ liệu

Sau khi truy vấn dữ liệu chuyến bay bằng Trino và tạo các bảng kết quả phân chia theo tháng, quý, năm và giữa các năm, dữ liệu từ các bảng này sẽ được nhập vào Superset dưới dạng các dataset để thực hiện trực quan hóa. Quá trình trực quan hóa dữ liệu được tổ chức thành các bảng điều khiển (dashboard), bao gồm bảng điều khiển cho từng năm riêng biệt và một bảng điều khiển so sánh dữ liệu giữa các năm. Trong mỗi bảng điều khiển, dữ liệu được trình bày qua các biểu đồ (chart) theo từng loại kết quả phân tích từ bước truy vấn dữ liệu, giúp người dùng dễ dàng

theo dõi và đánh giá thông tin chi tiết.

Mỗi dataset của các báo cáo đầu ra có đặc trưng dữ liệu và kết quả riêng biệt, do đó cần lựa chọn loại biểu đồ phù hợp để trình bày thông tin một cách hiệu quả. Dựa trên quá trình phân tích và thiết kế, em đã lựa chọn các biểu đồ cho từng báo cáo như sau: Biểu đồ cột được sử dụng cho các báo cáo (i) phân bố chuyến bay theo các ngày trong tuần, (ii) phân bố điểm xuất phát của các chuyến bay, (iii) hủy chuyến bay theo các ngày trong tuần, (iv) tổng số chuyến bay bị hủy và phân bố theo từng hãng hàng không, (v) phân tích đặc điểm của các ngày trong tháng, (vi) phân tích độ trễ trung bình theo từng hãng hàng không, (vii) phân bố điểm xuất phát của các chuyến bay qua từng năm. Biểu đồ tròn được áp dụng cho báo cáo phân tích mạng lưới tiếp thị hàng không. Biểu đồ đường được sử dụng để thể hiện số lượng chuyến bay theo từng tháng qua các năm. Chi tiết cụ thể về các biểu đồ này sẽ được trình bày trong phần kết quả 5.6.

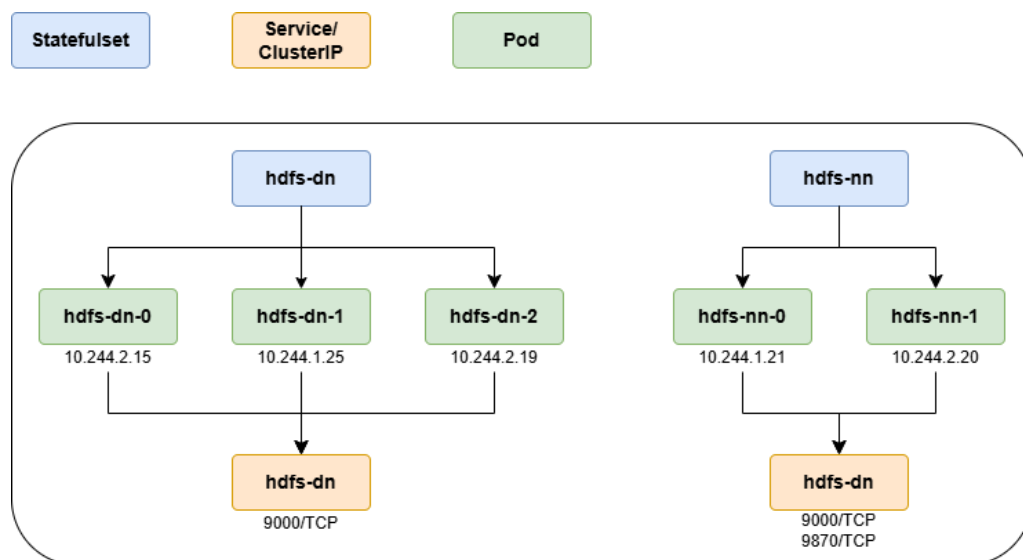
4.2 Triển khai hệ thống

Trong hệ thống của mình, em đã triển khai toàn bộ các ứng dụng và dịch vụ trên một cụm Kubernetes. Để thiết lập môi trường Kubernetes, em sử dụng Minikube, một công cụ đơn giản hóa việc cài đặt Kubernetes bằng cách tạo một máy ảo trên máy tính cá nhân. Minikube giúp em triển khai một cụm Kubernetes cơ bản với cấu hình mặc định là một node duy nhất. Tuy nhiên, để đáp ứng yêu cầu của hệ thống, em đã mở rộng cấu hình cụm Kubernetes thành 3 node. Cụm này bao gồm 1 node master, đóng vai trò là control plane, chịu trách nhiệm quản lý và điều phối toàn bộ hoạt động của cụm và 2 node còn lại được cấu hình làm worker node. Các worker node được gán nhãn phù hợp để phân biệt vai trò của chúng và là nơi các ứng dụng chính của hệ thống được triển khai.

Trong cụm Kubernetes này, em đã thiết lập một không gian tên (namespace) riêng biệt với tên là "hadoop-ecosystem". Toàn bộ các ứng dụng của hệ thống đều được triển khai trong không gian tên này, đảm bảo tính tổ chức và dễ dàng quản lý. Các ứng dụng được cài đặt trên hai worker node để tối ưu hóa việc phân bổ tài nguyên và đảm bảo hiệu suất cao. Các ứng dụng trong hệ thống giao tiếp với nhau thông qua các dịch vụ Kubernetes loại ClusterIP Service, giúp tạo địa chỉ IP nội bộ cho từng dịch vụ. Điều này cho phép các thành phần trong cụm có thể kết nối và trao đổi dữ liệu một cách an toàn và hiệu quả mà không cần lộ địa chỉ ra bên ngoài. Dưới đây, em sẽ trình bày chi tiết từng bước triển khai cho các mô-đun trong hệ thống.

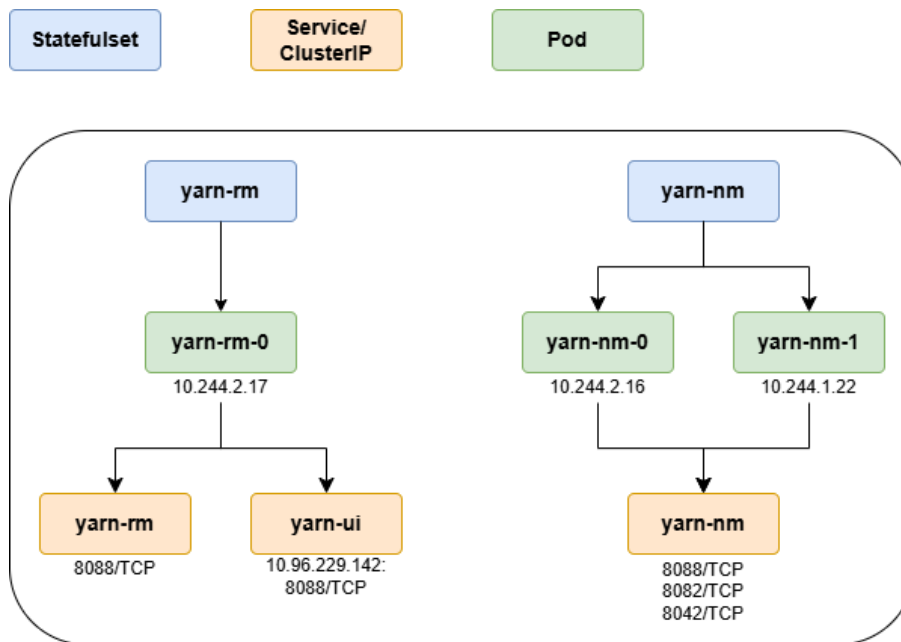
4.2.1 Triển khai cụm Hadoop

Hình 4.4 và 4.5 là cụm Hadoop được triển khai trên Kubernetes với các thành phần chính gồm HDFS và YARN, được quản lý bằng Statefulset để đảm bảo tính ổn định và định danh duy nhất cho từng Pod. Các Pod được phân phối đồng đều trên 2 node worker của cụm Kubernetes để đảm bảo cân bằng tải và tính sẵn sàng cao.



Hình 4.4: Triển khai cụm HDFS

Hình 4.4 là cụm HDFS bao gồm 2 Pod NameNode (hadoop-hadoop-hdfs-nn-0, hadoop-hadoop-hdfs-nn-1) để đảm bảo tính sẵn sàng cao của mô-đun lưu trữ dữ liệu, chịu trách nhiệm quản lý metadata. Cùng với 3 Pod DataNode (hadoop-hadoop-hdfs-dn-0, hadoop-hadoop-hdfs-dn-1, hadoop-hadoop-hdfs-dn-2) để lưu trữ dữ liệu. Các thành phần của cụm giao tiếp với nhau thông qua ClusterIP Service với các cổng mở phù hợp: 9000, 9870 cho NameNode và 9000 cho DataNode



Hình 4.5: Triển khai cụm YARN

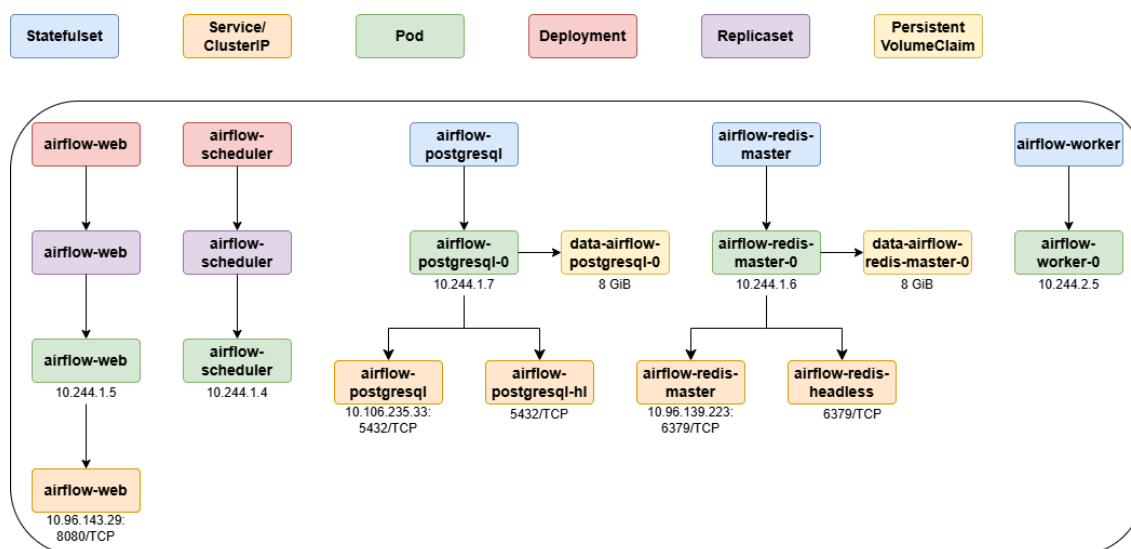
Trong khi YARN bao gồm 1 Pod ResourceManager (hadoop-hadoop-yarn-rm-0) để điều phối tài nguyên và 2 Pod NodeManager (hadoop-hadoop-yarn-nm-0, hadoop-hadoop-yarn-nm-1) để quản lý tài nguyên trên các node. Cụm YARN được triển khai để là nơi xử lý dữ liệu bằng Spark Streaming và Spark SQL trong hệ thống. Các thành phần của cụm giao tiếp với nhau thông qua ClusterIP Service với các cổng mở phù hợp: 8088, 8082, 8042 cho NodeManager và 8088 cho ResourceManager.

Thuộc tính	Giá trị
Số nhân bản dữ liệu	3
Kích thước khối	128 MiB
Số RAM mỗi NodeManager	4 GiB
Số nhân CPU mỗi NodeManager	3

Bảng 4.9: Cấu hình của cụm Hadoop

Bảng 4.9 có chứa các cấu hình quan trọng của cụm Hadoop. Trong đó, cấu hình số nhân bản là 3 để đảm bảo tính chịu lỗi của mô-đun lưu trữ dữ liệu, mỗi Pod NodeManager được cung cấp tài nguyên gồm có 4 GiB RAM và 3 nhân CPU, tổng cộng tài nguyên của mô-đun xử lý dữ liệu gồm có 8 GiB RAM và 6 nhân CPU.

4.2.2 Triển khai cụm Airflow

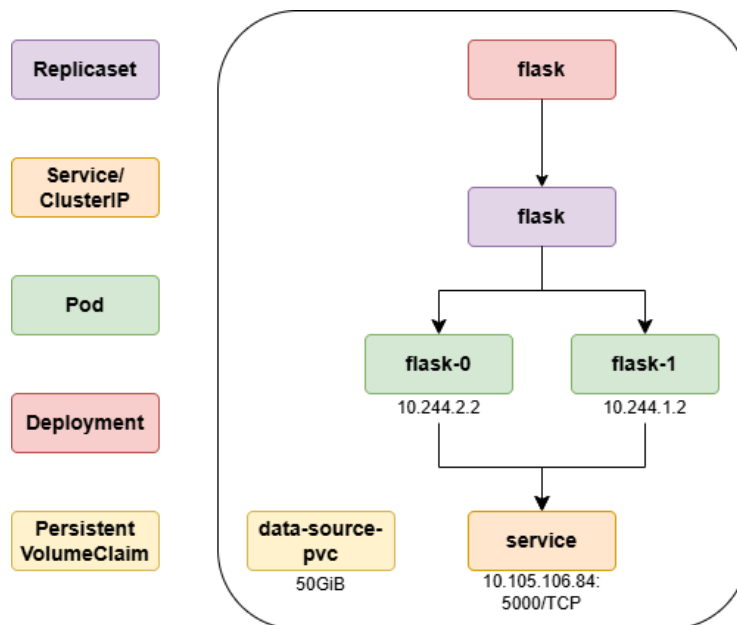


Hình 4.6: Triển khai cụm Airflow

Hình 4.6 là cụm Airflow được em triển khai trên Kubernetes bao gồm các thành phần chính: PostgreSQL, Redis, Airflow Scheduler, Airflow Webserver và Airflow Worker. PostgreSQL đóng vai trò là nơi lưu trữ cơ sở dữ liệu của cụm Airflow, Redis đóng vai trò là nơi lưu trữ dữ liệu thông điệp giữa Scheduler với Worker và Worker là nơi chạy các DAG được cấu hình của cụm Airflow đều được triển khai dưới dạng Statefulset, với mỗi Statefulset hiện tại chỉ có một Pod (airflow-postgresql-0, airflow-redis-master-0, airflow-worker-0) để phù hợp với tài nguyên máy tính em có thể đáp ứng, có thể tăng số Pod lên để có tính sẵn sàng cao cho cụm Airflow. Trong đó Pod "airflow-postgresql-0" và Pod "airflow-redis-master-0" sẽ kết nối với một Persistent Volume Claim tương ứng để lưu trữ dữ liệu cho Pod.

Các thành phần còn lại của cụm Airflow là: Scheduler, Webserver được triển khai dưới dạng Deployment với số lượng Pod tương ứng là 1 cho mỗi thành phần, quản lý thông qua các Replicaset. Cụm sử dụng ClusterIP Service để kết nối nội bộ, với các cổng dịch vụ: 8080 cho giao diện Web của Airflow, 5432 cho PostgreSQL, 6379 cho Redis và 8793 cho Worker. Tất cả các thành phần đều đang hoạt động ổn định và phân phối trên 2 node worker trong cụm Kubernetes (hadoop-ecosystem-m02, hadoop-ecosystem-m03).

4.2.3 Triển khai nguồn phát dữ liệu

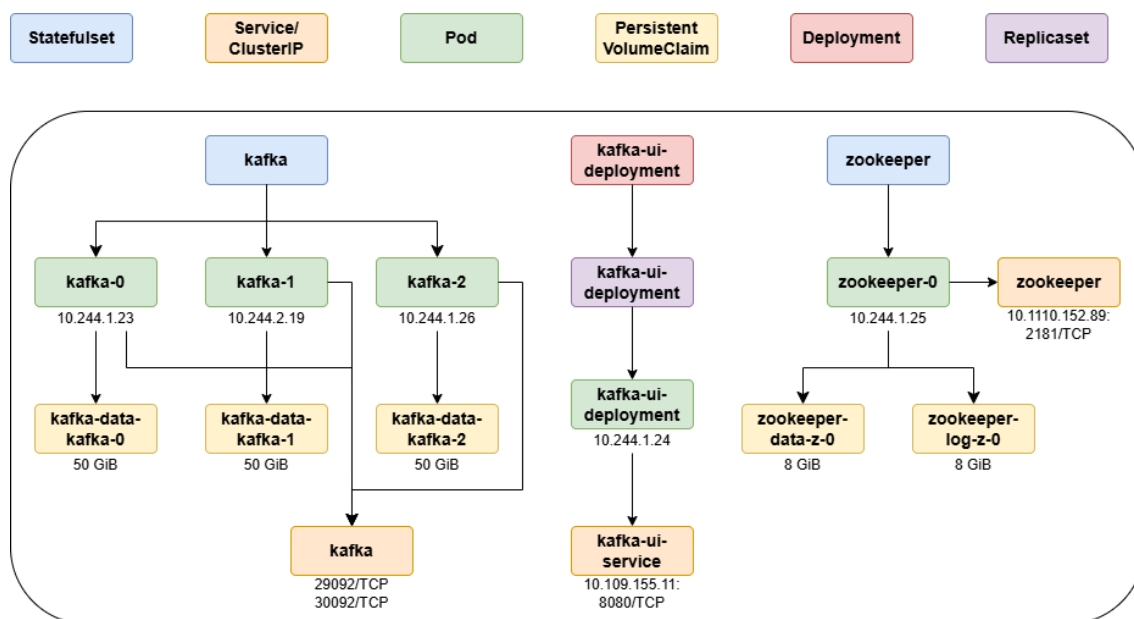


Hình 4.7: Triển khai nguồn phát dữ liệu

Hình 4.7 minh họa việc triển khai nguồn phát dữ liệu dưới dạng Deployment trên Kubernetes, với cấu hình gồm 2 Pod. Việc sử dụng 2 Pod nhằm đảm bảo tính sẵn sàng cao, cho phép hệ thống tiếp tục hoạt động ngay cả khi một trong các Pod gặp sự cố. Các Pod này được quản lý bởi một Replicaset, đảm bảo số lượng Pod luôn được duy trì theo cấu hình mong muốn. Nguồn phát dữ liệu được kết nối và cung cấp dịch vụ thông qua một ClusterIP Service. Dịch vụ này được gán địa chỉ IP nội bộ là 10.102.178.84, mở cổng 5000/TCP để cung cấp API phục vụ cho quá trình thu thập dữ liệu.

Để đảm bảo khả năng lưu trữ và xử lý dữ liệu, nguồn phát dữ liệu được liên kết với một Persistent Volume Claim (PVC). PVC này cung cấp khả năng lưu trữ ổn định và liên tục, cho phép dữ liệu của nguồn phát được lưu trữ một cách bền vững, ngay cả khi các Pod bị khởi động lại hoặc di chuyển giữa các node.

4.2.4 Triển khai cụm Kafka



Hình 4.8: Triển khai cụm Kafka

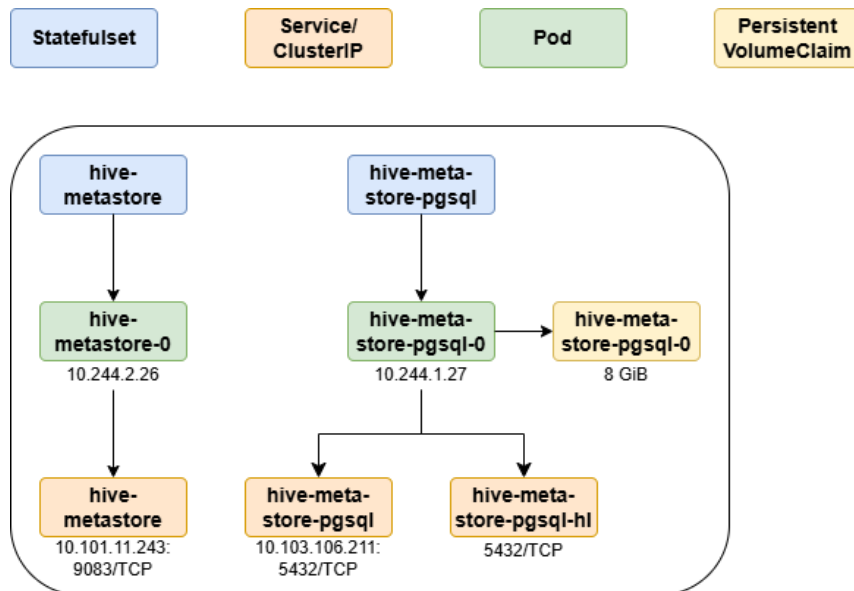
Hình 4.8 minh họa cách triển khai cụm Kafka trên Kubernetes, được cấu hình dưới dạng Statefulset nhằm đảm bảo tính ổn định và định danh cố định cho từng broker. Cụm Kafka này bao gồm 3 broker (kafka-0, kafka-1, kafka-2), mỗi broker được triển khai trong một Pod riêng biệt. Mỗi Pod trong Statefulset được liên kết với một Persistent Volume Claim, cung cấp lưu trữ ổn định và lâu dài cho dữ liệu của các broker.

Để quản lý cụm Kafka, một Statefulset Zookeeper được triển khai kèm theo 1 Pod và 2 Persistent Volume Claim, dùng để lưu trữ dữ liệu của Zookeeper và log của cụm Kafka. Bên cạnh đó, một Deployment kafka-ui được thiết lập, cung cấp giao diện web giúp tương tác với cụm Kafka thuận tiện hơn.

Một ClusterIP Service (kafka) được cấu hình mở cổng 29092/TCP để giao tiếp nội cụm và cổng 30092/TCP cho phép các ứng dụng trong Kubernetes kết nối và trao đổi dữ liệu với Kafka. Giao diện web của kafka-ui có địa chỉ IP là 10.109.155.11 và hoạt động trên cổng 8080/TCP. Trong khi đó, Zookeeper của cụm Kafka hoạt động trên cổng 2181/TCP.

Việc triển khai Kafka theo kiến trúc này giúp đảm bảo tính sẵn sàng cao, khả năng mở rộng linh hoạt, và khả năng lưu trữ dữ liệu lâu dài, phù hợp với các yêu cầu về xử lý dữ liệu phân tán trong hệ thống.

4.2.5 Triển khai cụm Hive

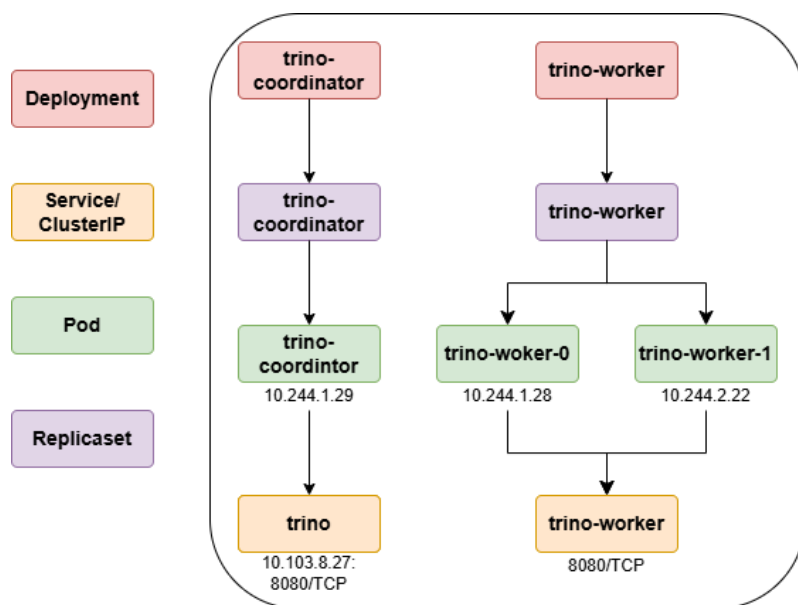


Hình 4.9: Triển khai cụm Hive

Hình 4.9 minh họa cụm Hive được triển khai trên Kubernetes, bao gồm hai thành phần chính là PostgreSQL và Hive Metastore. Trong đó, PostgreSQL được sử dụng làm cơ sở dữ liệu để lưu trữ dữ liệu của Hive, còn Hive Metastore đóng vai trò là kho lưu trữ siêu dữ liệu cho các bảng và dữ liệu được quản lý bởi Hive. Cả hai thành phần này đều được triển khai dưới dạng Statefulset, cho phép đảm bảo trạng thái ổn định và hỗ trợ việc mở rộng dễ dàng trong tương lai. Hiện tại, mỗi Statefulset chỉ bao gồm một Pod (hive-metastore-postgresql-0 và hive-metastore-0), tuy nhiên cấu hình có thể được mở rộng để tăng cường tính sẵn sàng cao cho hệ thống.

Pod "hive-metastore-postgresql-0" được liên kết với một Persistent Volume Claim. Cụm Hive sử dụng các ClusterIP Service để hỗ trợ giao tiếp giữa các thành phần bên trong cụm Kubernetes. Dịch vụ hive-metastore được cấu hình trên cổng 9083/TCP, cho phép các ứng dụng hoặc thành phần khác trong hệ thống kết nối và sử dụng metadata của Hive. Bên cạnh đó, dịch vụ hive-metastore-postgresql hoạt động trên cổng 5432/TCP, cung cấp giao diện kết nối với cơ sở dữ liệu PostgreSQL.

4.2.6 Triển khai cụm Trino

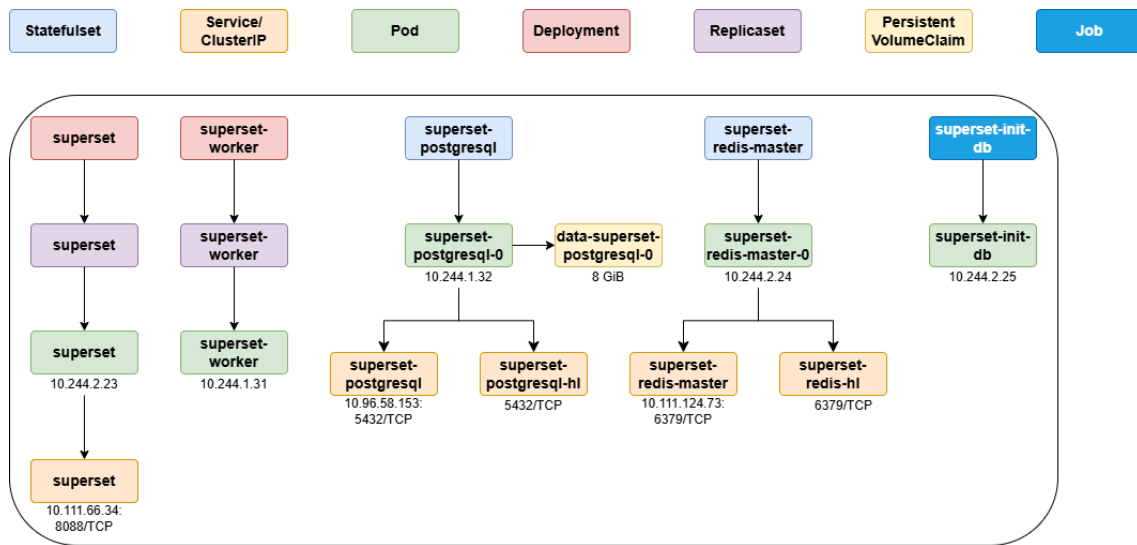


Hình 4.10: Triển khai cụm Trino

Hình 4.10 minh họa cụm Trino mà em đã triển khai trên nền tảng Kubernetes, bao gồm các thành phần chính: Trino Coordinator và Trino Worker. Trino Coordinator được triển khai dưới dạng một Deployment, với một Pod (trino-coordinator) đang hoạt động và được quản lý bởi một Replicaset. Tương tự, Trino Worker cũng được triển khai dưới dạng Deployment, với hai Pod (trino-worker-0 và trino-worker-1) đang chạy và được quản lý bởi một Replicaset riêng biệt.

Các dịch vụ trong cụm Trino sử dụng loại dịch vụ ClusterIP. Trong đó, dịch vụ chính "trino" với cổng 8080 cung cấp giao diện web của cụm Trino, trong khi dịch vụ "trino-worker" đảm nhận việc giao tiếp nội bộ giữa các thành phần trong cụm. Tất cả các thành phần của hệ thống đều hoạt động ổn định và được phân phối trên hai node worker trong cụm Kubernetes. Điều này giúp đảm bảo khả năng mở rộng linh hoạt cũng như tính sẵn sàng cao cho các tác vụ truy vấn dữ liệu, đảm bảo hiệu suất và độ tin cậy của hệ thống.

4.2.7 Triển khai cụm Superset



Hình 4.11: Triển khai cụm Superset

Hình 4.11 mô tả cụm Superset mà em đã triển khai trên nền tảng Kubernetes, bao gồm các thành phần chính: PostgreSQL, Redis, Superset Webserver và Superset Worker. PostgreSQL đóng vai trò là nơi lưu trữ cơ sở dữ liệu của cụm Superset, Redis đóng vai trò là nơi lưu trữ dữ liệu thông điệp giữa Webserver với Worker đều được triển khai dưới dạng Statefulset, với mỗi Statefulset có một Pod đang hoạt động, lần lượt là "superset-postgresql-0" và "superset-redis-master-0". Pod "superset-postgresql-0" kết nối với một Persistent Volume Claim để lưu trữ dữ liệu.

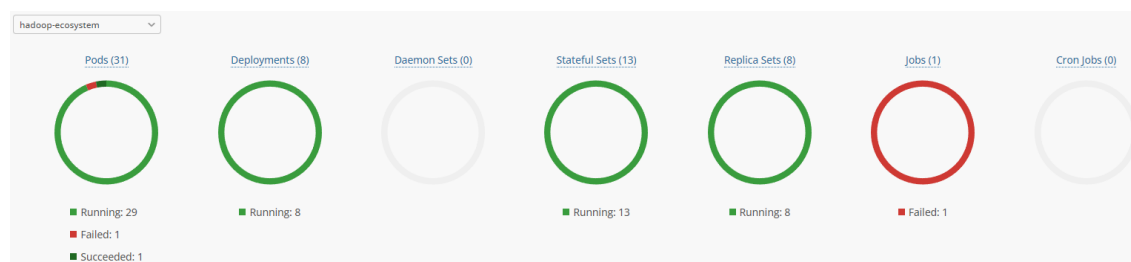
Các thành phần còn lại của Superset, bao gồm Webserver và Worker, được triển khai dưới dạng Deployment. Mỗi Deployment có một Pod đang chạy, được quản lý và giám sát bởi các Replicaset tương ứng. Bên cạnh đó, còn có một job Superset Init để khởi tạo cơ sở dữ liệu cho cụm Superset. Hệ thống sử dụng dịch vụ ClusterIP để thiết lập kết nối nội bộ giữa các thành phần. Các dịch vụ này bao gồm cổng 8088 cho giao diện web của Superset, cổng 5432 cho PostgreSQL và cổng 6379 cho Redis, cho phép các thành phần giao tiếp hiệu quả và đồng bộ trong môi trường Kubernetes.

4.2.8 Kết quả triển khai

<input type="checkbox"/> Name	<input type="checkbox"/> CPU	<input type="checkbox"/> Memory	<input type="checkbox"/> Disk	<input type="checkbox"/> Taint	<input type="checkbox"/> Roles	<input type="checkbox"/> Version	<input type="checkbox"/> Age	<input type="checkbox"/> Conditions
<input type="checkbox"/> hadoop-ecosystem				0	control-plane	v1.31.0	8h	Ready
<input type="checkbox"/> hadoop-ecosystem-m02				0	worker	v1.31.0	8h	Ready
<input type="checkbox"/> hadoop-ecosystem-m03				0	worker	v1.31.0	8h	Ready

Hình 4.12: Cụm Kubernetes

Hình 4.12 là kết quả triển khai cụm Kubernetes bằng Minikube với 3 node đã được gán nhãn thành 1 node control-plane và 2 node worker.



Hình 4.13: Tổng quan thành phần của cụm Kubernetes

Hình 4.13 là tổng quan các trạng thái đang chạy, lỗi và thành công của các Pod, Deployment, Daemonset, Statefulset, Replicaset, Job, Cronjob của cụm Kubernetes được dựng lên.

<input type="checkbox"/> Name	Namespace	Pods	Replicas	Age	Conditions
<input type="checkbox"/> flask	hadoop-ecosystem	2/2	2	8h	Available Progressing
<input type="checkbox"/> trino-worker	hadoop-ecosystem	2/2	2	6h49m	Available Progressing
<input type="checkbox"/> airflow-scheduler	hadoop-ecosystem	1/1	1	8h	Available Progressing
<input type="checkbox"/> airflow-web	hadoop-ecosystem	1/1	1	8h	Available Progressing
<input type="checkbox"/> kafka-ui-deployment	hadoop-ecosystem	1/1	1	3h34m	Available Progressing
<input type="checkbox"/> superset	hadoop-ecosystem	1/1	1	6h34m	Available Progressing
<input type="checkbox"/> superset-worker	hadoop-ecosystem	1/1	1	6h34m	Available Progressing
<input type="checkbox"/> trino-coordinator	hadoop-ecosystem	1/1	1	6h49m	Available Progressing

Hình 4.14: Các Deployment của hệ thống


Hình 4.14 là thông tin của các công nghệ được triển khai bằng Deployment đã được trình bày ở các phần trước bao gồm: Flask, cụm Trino, Airflow scheduler và webserver, Kafka-ui, Superset webserver và worker.

<input type="checkbox"/> Name	Namespace	Pods	Replicas	Age
<input type="checkbox"/> airflow-postgresql	hadoop-ecosystem	1/1	1	8h
<input type="checkbox"/> airflow-redis-master	hadoop-ecosystem	1/1	1	8h
<input type="checkbox"/> airflow-worker	hadoop-ecosystem	1/1	1	8h
<input type="checkbox"/> hadoop-hadoop-hdfs-dn	hadoop-ecosystem	3/3	3	7h11m
<input type="checkbox"/> hadoop-hadoop-hdfs-nn	hadoop-ecosystem	2/2	2	7h11m
<input type="checkbox"/> hadoop-hadoop-yarn-nm	hadoop-ecosystem	1/1	1	7h11m
<input type="checkbox"/> hadoop-hadoop-yarn-rm	hadoop-ecosystem	1/1	1	6h55m
<input type="checkbox"/> hive-metastore	hadoop-ecosystem	1/1	1	6h55m
<input type="checkbox"/> hive-metastore-postgresql	hadoop-ecosystem	1/1	1	6h55m
<input type="checkbox"/> kafka	hadoop-ecosystem	3/3	3	3h39m
<input type="checkbox"/> superset-postgresql	hadoop-ecosystem	1/1	1	6h39m
<input type="checkbox"/> superset-redis-master	hadoop-ecosystem	1/1	1	6h39m
<input type="checkbox"/> zookeeper	hadoop-ecosystem	1/1	1	3h39m

Hình 4.15: Các Statefulset của hệ thống


Hình 4.16 là thông tin của các công nghệ được triển khai bằng Statefulset đã được trình bày ở các phần trước bao gồm: postgresql và redis của cụm Airflow, postgresql và redis của cụm Superset, cụm Hadoop, cụm Hive, zookeeper và 3 broker của cụm Kafka.

CHƯƠNG 4. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG

<input type="checkbox"/> Name	 Namespace	Type	Cluster IP	Ports	External IP	Selector	Age	Status
<input type="checkbox"/> airflow	hadoop-ecosystem	ClusterIP	10.96.143.29	8080:http/TCP	-	app.kubern	8h	Active
<input type="checkbox"/> airflow-postgresql	hadoop-ecosystem	ClusterIP	10.106.235.33	5432:tcp-postgre	-	app.kubern	8h	Active
<input type="checkbox"/> airflow-postgresql-hl	hadoop-ecosystem	ClusterIP	None	5432:tcp-postgre	-	app.kubern	8h	Active
<input type="checkbox"/> airflow-redis-headless	hadoop-ecosystem	ClusterIP	None	6379:redis/TCP	-	app.kubern	8h	Active
<input type="checkbox"/> airflow-redis-master	hadoop-ecosystem	ClusterIP	10.96.139.223	6379:redis/TCP	-	app.kubern	8h	Active
<input type="checkbox"/> airflow-worker-hl	hadoop-ecosystem	ClusterIP	None	8793:worker/TCF	-	app.kubern	8h	Active
<input type="checkbox"/> hadoop-hadoop-hdfs-dn	hadoop-ecosystem	ClusterIP	None	9000/TCP	-	app.kubern	7h15m	Active
<input type="checkbox"/> hadoop-hadoop-hdfs-nn	hadoop-ecosystem	ClusterIP	None	9000/TCP, 9870/1	-	app.kubern	7h15m	Active
<input type="checkbox"/> hadoop-hadoop-yarn-nm	hadoop-ecosystem	ClusterIP	None	8088/TCP, 8082/1	-	app.kubern	7h15m	Active
<input type="checkbox"/> hadoop-hadoop-yarn-rm	hadoop-ecosystem	ClusterIP	None	8088/TCP	-	app.kubern	7h15m	Active
<input type="checkbox"/> hadoop-hadoop-yarn-ui	hadoop-ecosystem	ClusterIP	10.96.229.142	8088/TCP	-	app.kubern	7h15m	Active
<input type="checkbox"/> hive-metastore	hadoop-ecosystem	ClusterIP	10.101.11.243	9083/TCP	-	app.kubern	6h59m	Active
<input type="checkbox"/> hive-metastore-postgresql	hadoop-ecosystem	ClusterIP	10.103.106.21	5432:tcp-postgre	-	app.kubern	6h59m	Active
<input type="checkbox"/> hive-metastore-postgresql-headless	hadoop-ecosystem	ClusterIP	None	5432:tcp-postgre	-	app.kubern	6h59m	Active
<input type="checkbox"/> kafka	hadoop-ecosystem	ClusterIP	None	29092/TCP, 3009	-	service=kaf	3h43m	Active
<input type="checkbox"/> kafka-ui-service	hadoop-ecosystem	ClusterIP	10.109.155.11	8080/TCP	-	app=kafka-1	3h43m	Active
<input type="checkbox"/> service	hadoop-ecosystem	ClusterIP	10.105.106.84	5000/TCP	-	app=flask	8h	Active
<input type="checkbox"/> superset	hadoop-ecosystem	ClusterIP	10.111.66.34	8088:http/TCP	-	app=supers	6h43m	Active
<input type="checkbox"/> superset-postgresql	hadoop-ecosystem	ClusterIP	10.96.58.153	5432:tcp-postgre	-	app.kubern	6h43m	Active
<input type="checkbox"/> superset-postgresql-hl	hadoop-ecosystem	ClusterIP	None	5432:tcp-postgre	-	app.kubern	6h43m	Active
<input type="checkbox"/> superset-redis-headless	hadoop-ecosystem	ClusterIP	None	6379:redis/TCP	-	app.kubern	6h43m	Active
<input type="checkbox"/> superset-redis-master	hadoop-ecosystem	ClusterIP	10.111.124.73	6379:redis/TCP	-	app.kubern	6h43m	Active
<input type="checkbox"/> trino	hadoop-ecosystem	ClusterIP	10.103.8.27	8080:http/TCP	-	app.kubern	6h59m	Active
<input type="checkbox"/> trino-worker	hadoop-ecosystem	ClusterIP	None	8080:http/TCP	-	app.kubern	6h59m	Active
<input type="checkbox"/> zookeeper	hadoop-ecosystem	ClusterIP	10.110.152.89	2181/TCP	-	service=zoo	3h43m	Active

Hình 4.16: Các Service của hệ thống

Hình 4.16 là thông tin của toàn bộ Service của cụm Kubernetes được triển khai trong hệ thống.

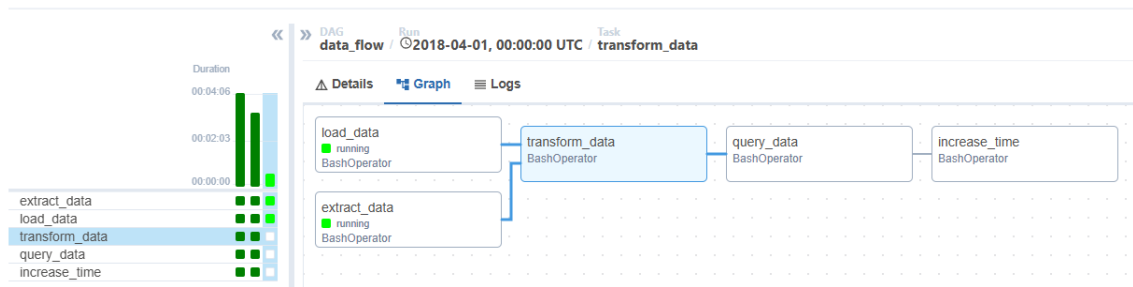
<input type="checkbox"/> Name	 Namespace	Storage class	Size	Pods	Age	Status
<input type="checkbox"/> data-airflow-postgresql-0	hadoop-ecosystem	standard	8Gi		8h	Bound
<input type="checkbox"/> data-hive-metastore-postgresql-0	hadoop-ecosystem	standard	8Gi		7h32m	Bound
<input type="checkbox"/> data-kafka-controller-0	hadoop-ecosystem	standard	50Gi		7h18m	Bound
<input type="checkbox"/> data-kafka-controller-1	hadoop-ecosystem	standard	50Gi		7h18m	Bound
<input type="checkbox"/> data-kafka-controller-2	hadoop-ecosystem	standard	50Gi		7h18m	Bound
<input type="checkbox"/> data-source-pvc	hadoop-ecosystem	standard	50Gi		9h	Bound
<input type="checkbox"/> data-superset-postgresql-0	hadoop-ecosystem	standard	8Gi		6h45m	Bound
<input type="checkbox"/> kafka-data-kafka-0	hadoop-ecosystem	standard	50Gi		4h10m	Bound
<input type="checkbox"/> kafka-data-kafka-1	hadoop-ecosystem	standard	50Gi		4h9m	Bound
<input type="checkbox"/> kafka-data-kafka-2	hadoop-ecosystem	standard	50Gi		4h9m	Bound
<input type="checkbox"/> redis-data-airflow-redis-master-0	hadoop-ecosystem	standard	8Gi		8h	Bound
<input type="checkbox"/> zookeeper-data-zookeeper-0	hadoop-ecosystem	standard	8Gi		3h45m	Bound
<input type="checkbox"/> zookeeper-log-zookeeper-0	hadoop-ecosystem	standard	8Gi		3h45m	Bound

Hình 4.17: Các Persistent Volume Claim của hệ thống

Hình 4.17 là thông tin của toàn bộ Persistent Volume Claim của cụm Kubernetes được triển khai trong hệ thống.

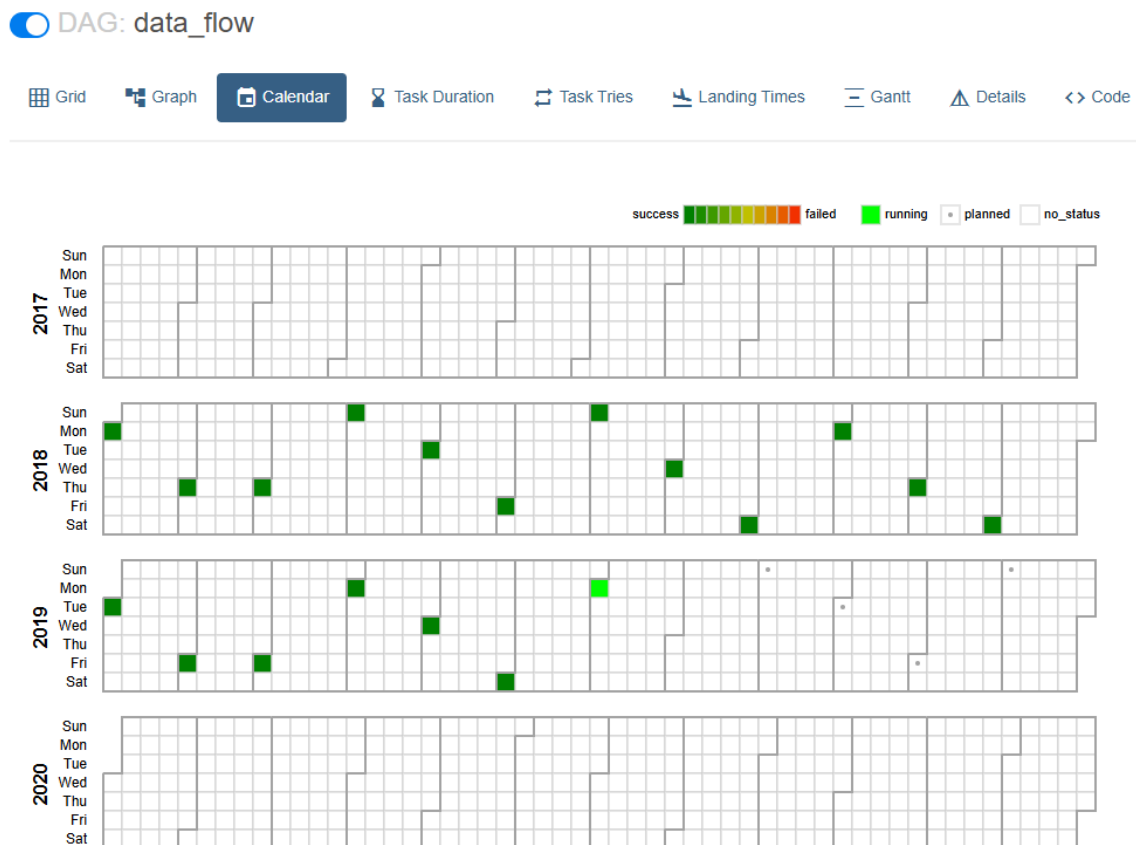
CHƯƠNG 5. KẾT QUẢ THỰC NGHIỆM

5.1 Kết quả lập lịch và tự động hóa



Hình 5.1: Kết quả lập lịch tác vụ trên Airflow (1)

Hình 5.1 là kết quả hoạt động của DAG "data_flow" là luồng xử lý dữ liệu chính của hệ thống trên Airflow. Nhờ có Airflow, hệ thống có thể lập lịch và giám sát quá trình chạy các tác vụ từ thu thập, lưu trữ, xử lý tới truy vấn dữ liệu. Khi tác vụ bị lỗi có thể chạy lại và lưu log đầy đủ hỗ trợ cho quá trình sửa lỗi hệ thống.



Hình 5.2: Kết quả lập lịch tác vụ trên Airflow (2)

Hình 5.2 là kết quả giả lập quá trình lập lịch lấy dữ liệu theo hàng tháng, cho thấy luồng xử lý dữ liệu được xử lý định kỳ và hoàn toàn tự động.

5.2 Tính chịu lỗi của hệ thống

UI for Apache Kafka

83b5a60 v0.7.2

Dashboard

kafka

Brokers

Topics

Consumers

Brokers

Uptime

Broker Count3

Active Controller3

Version3.8-IV0

Partitions

Online3 of 3

URP0

In Sync Replicas3 of 3

Out Of Sync Replicas0

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1	N/A	-	1	-	1	9092	broker01
2	N/A	-	1	-	1	9092	broker02
3	N/A	-	1	-	1	9092	broker03

Hình 5.3: Tính chịu lỗi trên cụm Kafka (1)

UI for Apache Kafka

83b5a60 v0.7.2

Dashboard

kafka

Brokers

Topics

Consumers

Brokers

Uptime

Broker Count2

Active Controller2

Version3.8-IV0

Partitions

Online2 of 3

URP0

In Sync Replicas3 of 3

Out Of Sync Replicas0

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1	N/A	-	1	-	1	9092	broker01
2	N/A	-	1	-	1	9092	broker02

Hình 5.4: Tính chịu lỗi trên cụm Kafka (2)

Hình 5.3 và 5.4 thể hiện tính chịu lỗi của cụm Kafka trong hệ thống. Khi hoạt động bình thường, cụm Kafka sẽ có 3 broker, ở đây broker số 3 được Zookeeper lựa chọn làm leader cho cụm 3 broker. Sau khi em tắt broker số 3 đi, Zookeeper lập tức chọn broker khác làm leader, ở đây là broker số 2 và cụm Kafka vẫn hoạt động bình thường với 2/3 số broker với số nhân bản vẫn là 3.

Name	Namespace	Cont...	CPU	Memor	Restart	Controlled B	Node	QoS	Age	Status
airflow-postgresql-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	27h	Running
airflow-redis-master-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	27h	Running
airflow-scheduler-56f5578b59-rp8h	hadooop-ecosyste	■	N/A	N/A	4	ReplicaSet	hadoop-ec	Burstable	27h	Running
airflow-web-7f65794fd-cdkn6	hadooop-ecosyste	■	N/A	N/A	6	ReplicaSet	hadoop-ec	Burstable	27h	Running
airflow-worker-0	hadooop-ecosyste	■	N/A	N/A	4	StatefulSet	hadoop-ec	Burstable	27h	Running
flask-78d877bfd4-84ssp	hadooop-ecosyste	■	N/A	N/A	2	ReplicaSet	hadoop-ec	BestEffort	28h	Running
flask-78d877bfd4-jdqj8	hadooop-ecosyste	■	N/A	N/A	2	ReplicaSet	hadoop-ec	BestEffort	28h	Running
hadoop-hadoop-hdfs-dn-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hadoop-hadoop-hdfs-dn-1	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hadoop-hadoop-hdfs-dn-2	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hadoop-hadoop-hdfs-nn-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hadoop-hadoop-hdfs-nn-1	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hadoop-hadoop-yarn-nm-0	hadooop-ecosyste	■	N/A	N/A	3	StatefulSet	hadoop-ec	Guarantee	26h	Running
hadoop-hadoop-yarn-nm-1	hadooop-ecosyste	■	N/A	N/A	3	StatefulSet	hadoop-ec	Guarantee	26h	Running
hadoop-hadoop-yarn-rm-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
hive-metastore-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	BestEffort	25h	Running
hive-metastore-postgresql-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
kafka-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	BestEffort	22h	Running
kafka-1	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	BestEffort	22h	Running
kafka-2	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	BestEffort	22h	Running
kafka-ui-deployment-649b8cfb9c-x2qhn	hadooop-ecosyste	■	N/A	N/A	1	ReplicaSet	hadoop-ec	Burstable	22h	Running
superset-6d87b4d7bf-zdlwh	hadooop-ecosyste	■	N/A	N/A	2	ReplicaSet	hadoop-ec	BestEffort	25h	Running
superset-init-db-d8jqp	hadooop-ecosyste	■	N/A	N/A	0	Job	hadoop-ec	BestEffort	25h	Failed
superset-init-db-pbjlm	hadooop-ecosyste	■	N/A	N/A	0	Job	hadoop-ec	BestEffort	25h	Succeeded
superset-postgresql-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	25h	Running
superset-redis-master-0	hadooop-ecosyste	■	N/A	N/A	2	StatefulSet	hadoop-ec	BestEffort	25h	Running
superset-worker-5466cbd8f-qcql2	hadooop-ecosyste	■	N/A	N/A	2	ReplicaSet	hadoop-ec	BestEffort	25h	Running
trino-coordinator-6f85d689-sjy5	hadooop-ecosyste	■	N/A	N/A	0	ReplicaSet	hadoop-ec	BestEffort	14m	Running
trino-worker-6797f565f6-h9zxx	hadooop-ecosyste	■	N/A	N/A	0	ReplicaSet	hadoop-ec	BestEffort	14m	Running

Hình 5.5: Tính chịu lỗi khi sử dụng Kubernetes (1)

Hình 5.5 thể hiện tính chịu lỗi khi sử dụng Kubernetes để triển khai hệ thống.

Các Pod trong cụm Kubernetes được triển khai dưới dạng các Replicaset và Statefulset giúp kiểm soát được số lượng và trạng thái hoạt động của từng Replicaset và Statefulset. Các Pod luôn được restart lại khi gặp lỗi để đảm bảo trạng thái hoạt động luôn là running.








trino-coordinator-56885fc5b-js779	1/1	Running	2 (17h ago)	25h
trino-coordinator-6f855d689-sjvw5	0/1	Running	0	22s
trino-worker-64c5ff4b4f-j29bg	1/1	Running	2 (17h ago)	25h
trino-worker-64c5ff4b4f-xnqqz	1/1	Running	2 (17h ago)	25h
trino-worker-6797f565f6-h9zxk	0/1	Running	0	22s

Hình 5.6: Tính chịu lỗi khi sử dụng Kubernetes (2)

Hình 5.4 cũng thể hiện tính chịu lỗi khi sử dụng Kubernetes để triển khai hệ thống. Khi em tiến hành dùng Helm để cập nhật cụm Trino, em đã chọn chiến lược (strategy) cập nhật là RollingUpdate để đảm bảo hệ thống luôn hoạt động ngay cả khi tiếp hành cập nhật Pod, Pod cũ sẽ tồn tại và được xóa sau khi Pod mới được khởi tạo thành công.

5.3 Tính khả mở của hệ thống

Tính khả mở là một đặc điểm nổi bật và quan trọng của một hệ thống được triển khai bằng Kubernetes. Khi hạ tầng có đủ tài nguyên, sử dụng Kubernetes giúp dễ dàng mở rộng hệ thống theo cả chiều dọc lẫn chiều ngang bằng việc thay đổi các file YAML trong mã nguồn, sau đó cập nhật hệ thống bằng Kubectl hoặc Helm; hoặc mở rộng (scale up) một cụm nào đó bằng ngay câu lệnh của Kubectl.

<input type="checkbox"/> hadoop-hadoop-hdfs-dn-0	hadoop-ecosyste 	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-dn-1	hadoop-ecosyste 	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-dn-2	hadoop-ecosyste 	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-dn-3	hadoop-ecosyste 	N/A	N/A	0	StatefulSet	hadoop-ec	Burstable	114s	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-nn-0	hadoop-ecosyste 	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-nn-1	hadoop-ecosyste 	N/A	N/A	2	StatefulSet	hadoop-ec	Burstable	26h	Running
<input type="checkbox"/> hadoop-hadoop-hdfs-nn-2	hadoop-ecosyste 	N/A	N/A	0	StatefulSet	hadoop-ec	Burstable	114s	Running

Hình 5.7: Tính khả mở khi sử dụng Kubernetes (1)

Hình 5.7 là kết quả khi em tăng số NameNode từ 2 lên 3 và tăng số DataNode từ 3 lên 4 của cụm HDFS được em thực hiện bằng cách thay đổi các file YAML trong mã nguồn, sau đó cập nhật hệ thống bằng Helm.

```
C:\Users\trann>kubectl scale --replicas=2 statefulset/zookeeper
statefulset.apps/zookeeper scaled

C:\Users\trann>kubectl scale --replicas=4 statefulset/kafka
statefulset.apps/kafka scaled

C:\Users\trann>
```

Hình 5.8: Tính khả mở khi sử dụng Kubernetes (2)

<input type="checkbox"/> kafka	hadoop-ecosystem	4/4	4	23h
<input type="checkbox"/> superset-postgresql	hadoop-ecosystem	1/1	1	26h
<input type="checkbox"/> superset-redis-master	hadoop-ecosystem	1/1	1	26h
<input type="checkbox"/> zookeeper	hadoop-ecosystem	2/2	2	23h

Hình 5.9: Tính khả mở khi sử dụng Kubernetes (3)

Hình 5.8 câu lệnh Kubectl để tăng số broker trong cụm Kafka từ 3 lên 4, tăng số Zookeeper trong cụm Kafka từ 2 lên 3 và hình 5.9 là kết quả sau khi mở rộng cụm Kafka.

5.4 Kết quả lưu trữ dữ liệu

UI for Apache Kafka

3b5a90 v0.7.2

Dashboard

kafka -

Brokers

Topics

Consumers

Brokers

Uptime

Broker Count3

Active Controller2

Version3.8-IV0

Partitions

Online3 of 3

URP0

In Sync Replicas3 of 3

Out Of Sync Replicas0

Broker ID

Disk usage

Partitions skew ⓘ

Leaders

Leader skew

Online partitions

Port

Host

1

3.63 MB, 1 segment(s)

-

1

-

1

9092

broker01

2

1.95 MB, 1 segment(s)

-

1

-

1

9092

broker02

3

2.79 MB, 1 segment(s)

-

1

-

1

9092

broker03

Hình 5.10: Kết quả lưu trữ trên cụm Kafka (1)

Hình 5.10 là thông tin về cụm 3 broker của Kafka đang hoạt động 3/3, có thông tin giá trị lượng dữ liệu hiện tại được lưu trên mỗi broker và broker số 2 được Zookeeper lựa chọn làm leader cho cụm 3 broker này.

Partitions	Replication Factor	URP	In Sync Replicas	Type	Segment Size	Segment Count	Clean Up Policy	Message Count
3	1	0	3 of 3	External	8 MB	3	DELETE	30

Partition ID	Replicas	First Offset	Next Offset	Message Count
0	3	0	10	10
1	1	0	13	13
2	2	0	7	7

Hình 5.11: Kết quả lưu trữ trên cụm Kafka (2)

Hình 5.11 là thông tin về một topic theo năm của dữ liệu trong cụm Kafka với số partition là 3 và số nhân bản là 3 giống như trong thiết kế phần 4.1.4, ngoài ra còn có thứ tự thông điệp tiếp theo (offset), lượng thông điệp đã nhận được của cụm và của mỗi partition.

	Offset	Partition	Timestamp	Key Preview	Value Preview
+	0	1	1/5/2025, 11:14:20	2018_1_0	[{"ActualElapsedTime": 59.0, "AirTime": 38.0, "Arr...
+	1	1	1/5/2025, 11:14:24	2018_1_100	[{"ActualElapsedTime": 84.0, "AirTime": 64.0, "Arr...
+	0	0	1/5/2025, 11:14:29	2018_1_200	[{"ActualElapsedTime": 96.0, "AirTime": 69.0, "Arr...
+	2	1	1/5/2025, 11:14:33	2018_1_300	[{"ActualElapsedTime": 109.0, "AirTime": 89.0, "Arr...
+	3	1	1/5/2025, 11:14:37	2018_1_400	[{"ActualElapsedTime": 83.0, "AirTime": 58.0, "Arr...
+	4	1	1/5/2025, 11:14:41	2018_1_500	[{"ActualElapsedTime": 134.0, "AirTime": 92.0, "Arr...
+	1	0	1/5/2025, 11:14:46	2018_1_600	[{"ActualElapsedTime": 66.0, "AirTime": 46.0, "Arr...
+	0	2	1/5/2025, 11:14:52	2018_1_700	[{"ActualElapsedTime": 108.0, "AirTime": 91.0, "Arr...
+	5	1	1/5/2025, 11:15:03	2018_1_900	[{"ActualElapsedTime": 48.0, "AirTime": 27.0, "Arr...
+	1	2	1/5/2025, 11:15:04	2018_1_800	[{"ActualElapsedTime": 110.0, "AirTime": 72.0, "Arr...
+	2	0	1/5/2025, 11:15:09	2018_1_1000	[{"ActualElapsedTime": 88.0, "AirTime": 72.0, "Arr...
+	2	2	1/5/2025, 11:15:14	2018_1_1100	[{"ActualElapsedTime": 109.0, "AirTime": 92.0, "Arr...

Hình 5.12: Kết quả lưu trữ trên cụm Kafka (3)

Hình 5.11 là thông tin chi tiết về các thông điệp được gửi đến cụm Kafka. Thông tin cho biết: thông điệp nằm ở partition nào, offset của thông điệp trên partition đó, khóa (key) và giá trị (value) của thông điệp. Ngoài ra còn có thông tin về timestamp cho thấy dữ liệu được gửi liên tục tới cụm Kafka với khoảng cách giữa các thông điệp là từ 4 đến 6 giây.

Summary

Security is off.	
Safemode is off.	
1,279 files and directories, 1,106 blocks (1,106 replicated blocks, 0 erasure coded block groups) = 2,385 total filesystem object(s).	
Heap Memory used 94.36 MB of 533.5 MB Heap Memory. Max Heap Memory is 3.46 GB.	
Non Heap Memory used 75.07 MB of 76.65 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.	
Configured Capacity:	2.95 TB
Configured Remote Capacity:	0 B
DFS Used:	13.46 GB (0.45%)
Non DFS Used:	178.52 GB
DFS Remaining:	2.61 TB (88.56%)
Block Pool Used:	13.46 GB (0.45%)
DataNodes usages% (Min/Median/Max/stdDev):	0.45% / 0.45% / 0.45% / 0.00%
Live Nodes	3 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Sun Jan 05 13:05:55 +0700 2025
Last Checkpoint Time	Sun Jan 05 13:05:49 +0700 2025
Enabled Erasure Coding Policies	RS-6-3-1024k

Hình 5.13: Kết quả lưu trữ dữ liệu trên HDFS (1)

Hình 5.13 là thống kê tổng quan của dữ liệu được lưu trữ trên HDFS trong đó có 3 Live Nodes như trong triển khai để đảm bảo tính phân tán của hệ thống.

In operation

Show 25 entries Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✓ 41dbb611d097:9866 (172.18.0.4:9866)	http://41dbb611d097:9864	1s	21m	1006.85 GB	1106	4.49 GB (0.45%)	3.2.1
✓ d1cb574eaa2d:9866 (172.18.0.6:9866)	http://d1cb574eaa2d:9864	1s	67m	1006.85 GB	1106	4.49 GB (0.45%)	3.2.1
✓ fc15d3782c90:9866 (172.18.0.7:9866)	http://fc15d3782c90:9864	1s	67m	1006.85 GB	1106	4.49 GB (0.45%)	3.2.1

Showing 1 to 3 of 3 entries

Previous 1 Next

Hình 5.14: Kết quả lưu trữ dữ liệu trên HDFS (2)

Hình 5.14 cho thấy dữ liệu được lưu trữ trên 3 datanode của cụm HDFS, dữ liệu được chia đồng đều trên cả 3 node cho thấy tính cân bằng tải của mô-đun lưu trữ dữ liệu.

Browse Directory

/processed_data/processed_data.db/flight_2020 Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxr-xr-x	airflow	supergroup	48.44 MB	Jan 01 21:22	3	128 MB	part-00000-56584430-0635-4ee1-ad97-358a6be9b338-c000
-rwxr-xr-x	airflow	supergroup	37.54 MB	Jan 01 21:38	3	128 MB	part-00000-5a55eea5-2778-414f-8cdf-cd78eaa8801e-c000
-rwxr-xr-x	airflow	supergroup	48.44 MB	Jan 01 21:25	3	128 MB	part-00000-6a997ee2-b7b1-414f-a089-a6c77fa423dd-c000
-rwxr-xr-x	airflow	supergroup	30.45 MB	Jan 01 21:36	3	128 MB	part-00000-9200bc90-e94c-4de6-80cb-1a6503222c54-c000
-rwxr-xr-x	airflow	supergroup	49.12 MB	Jan 01 21:20	3	128 MB	part-00000-b7151389-64d8-437c-aa03-194ae2249e33-c000
-rwxr-xr-x	airflow	supergroup	49.5 MB	Jan 01 21:28	3	128 MB	part-00000-d0b3b685-13be-4ab5-b1d4-f9b608edbc95-c000
-rwxr-xr-x	airflow	supergroup	48.7 MB	Jan 01 21:22	3	128 MB	part-00001-56584430-0635-4ee1-ad97-358a6be9b338-c000
-rwxr-xr-x	airflow	supergroup	37.67 MB	Jan 01 21:38	3	128 MB	part-00001-5a55eea5-2778-414f-8cdf-cd78eaa8801e-c000

Hình 5.15: Kết quả lưu trữ dữ liệu trên HDFS (3)

Hình 5.15 cho thấy dữ liệu được lưu trữ dưới các file Parquet trên cụm HDFS với số nhân bản (replication) là 3 để đảm bảo tính chống chịu lỗi của mô-đun lưu trữ dữ liệu.

5.5 Hiệu suất truy vấn dữ liệu

Session	Execution
User airflow 	Resource Group global
Principal airflow	Submission Time 2025-01-05 2:00pm
Source trino-cli	Completion Time 2025-01-05 2:00pm
Catalog	Elapsed Time 8.71s
Schema	Queued Time 423.00us
Time zone UTC	Analysis Time 180.17ms
Client Address 172.18.0.10	Planning Time 350.99ms
Client Tags	Execution Time 8.53s
Session Properties	
Resource Estimates	

Hình 5.16: Hiệu suất truy vấn dữ liệu trên Trino (1)

Resource Utilization Summary	Timeline
CPU Time 51.04s	Parallelism
Planning CPU Time 197.30ms	5.86
Scheduled Time 4.61m	Scheduled Time/s
Input Rows 8.09M	31.8
Input Data 170MB	Input Rows/s
Physical Input Rows 8.09M	929K
Physical Input Data 2.51GB	Input Bytes/s
Physical Input Read Time 58.75s	19.5MB
Internal Network Rows 1.25K	Physical Input Bytes/s
Internal Network Data 74.5KB	43.8MB
Peak User Memory 80.3MB	Memory Utilization
Peak Total Memory 80.3MB	0B
Cumulative User Memory 191MB*seconds	
Output Rows 1.00	
Output Data 9.00B	
Written Rows 120	
Logical Written Data 3.63KB	
Physical Written Data 1.64KB	

Hình 5.17: Hiệu suất truy vấn dữ liệu trên Trino (2)

Hình 5.16 và 5.17 là thông tin đánh giá câu truy vấn tạo bảng tên "marketing_airline_network_{year}" trong cả năm 2019 tương tự như biểu đồ 5.22. Hiệu suất của câu truy vấn bằng cụm Trino được em mô tả, tóm tắt lại trong bảng 5.1.

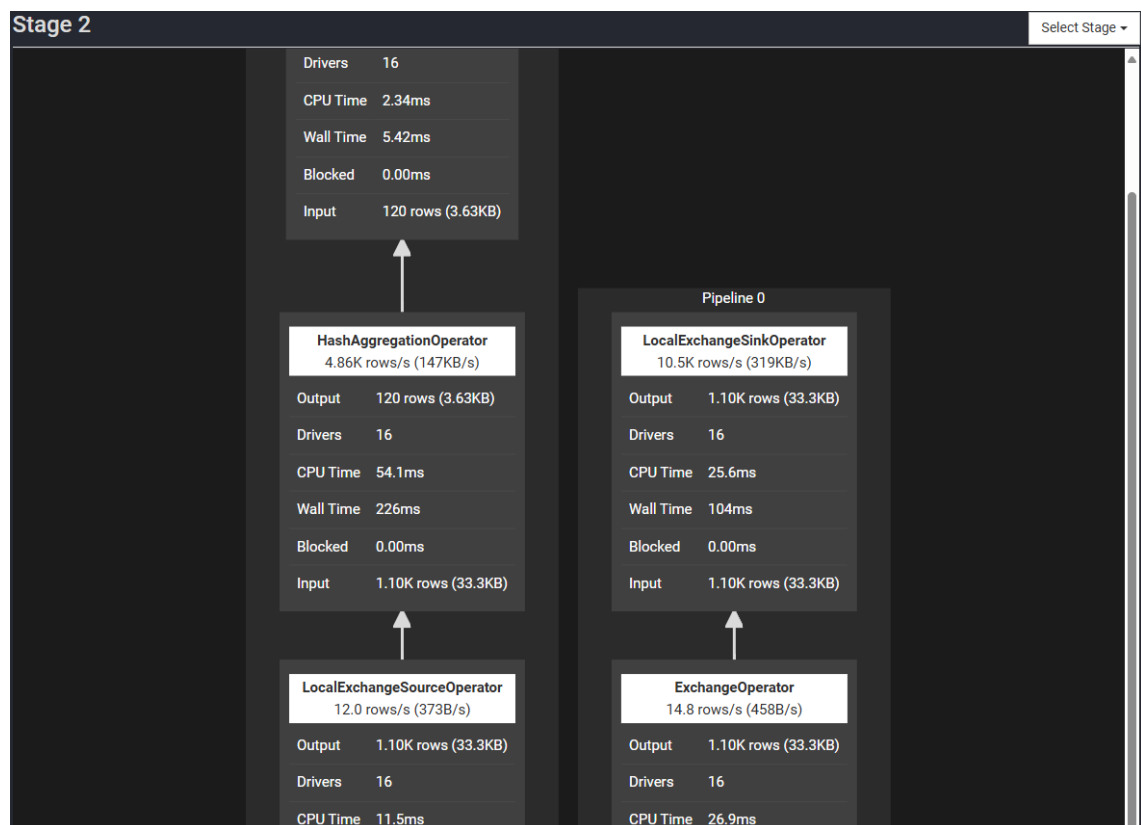
Thuộc tính	Giá trị
Thời gian đợi	423.00 us
Thời gian phân tích	180.17 ms
Thời gian lên kế hoạch	350.99 ms
Thời gian thực hiện	8.53 s
Tổng số bản ghi	8.09 triệu
Kích thước dữ liệu	2.51 GB

Bảng 5.1: Hiệu suất câu truy vấn trên Trino

Với dữ liệu 8.09 triệu bản ghi và kích thước 2.51 GB, Trino thể hiện hiệu suất khá tốt, đặc biệt là về thời gian đợi, phân tích và lên kế hoạch. Tuy nhiên, thời gian thực hiện 8.53 giây có thể được cải thiện nếu dữ liệu được phân phối tối ưu hơn hoặc truy vấn được tối ưu hóa thêm. So với các công nghệ khác, Trino là một lựa chọn mạnh mẽ và có khả năng cạnh tranh, đặc biệt trong môi trường cần xử lý dữ liệu phân tán và trả kết quả nhanh chóng.



Hình 5.18: Hiệu suất truy vấn dữ liệu trên Trino (3)

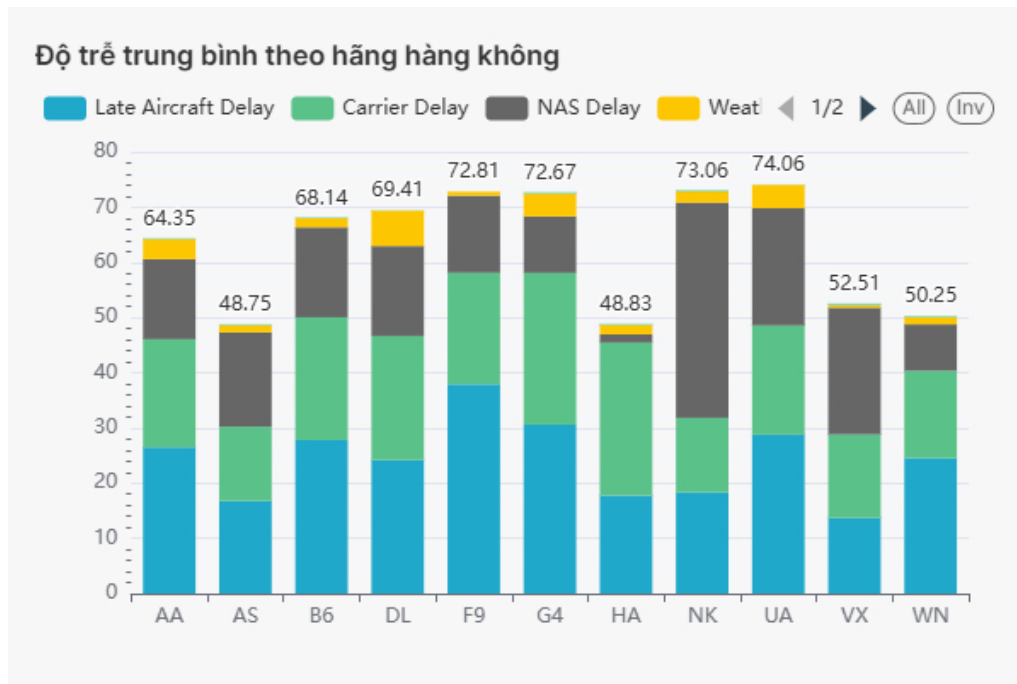


Hình 5.19: Hiệu suất truy vấn dữ liệu trên Trino (4)

Ngoài ra, hình 5.18 và 5.19 còn có thông tin của các màn (stage) và đường ống (pipeline) của các màn theo thiết kế của Trino, giúp câu truy vấn có thể được thực hiện phân tán, song song ngay trên hồ dữ liệu, khiến Trino trở thành một công nghệ truy vấn dữ liệu lớn được sử dụng rộng rãi.

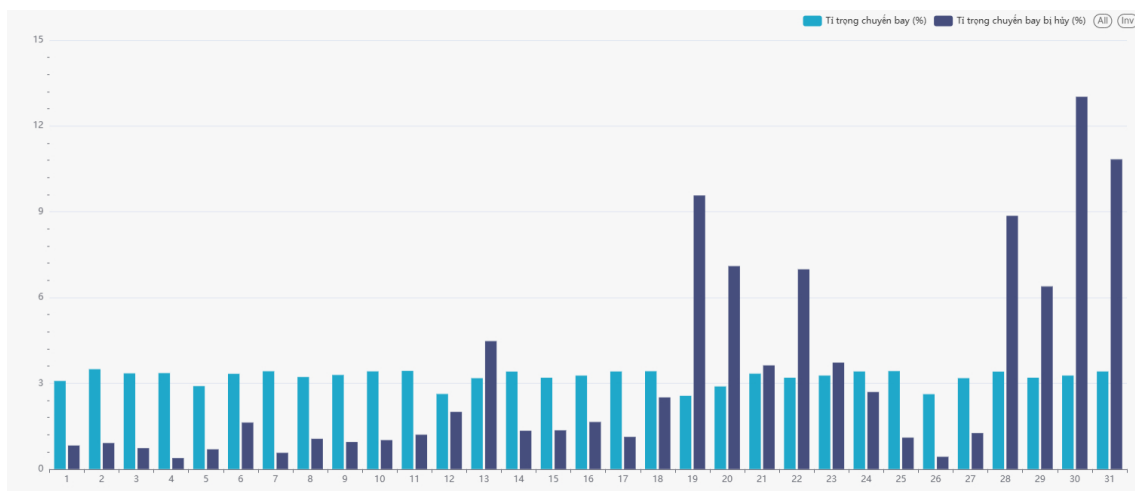
5.6 Kết quả trực quan hóa dữ liệu

Dưới đây là một số biểu đồ trực quan hóa các thông số liên quan đến các chuyến bay do các hãng hàng không tại Hoa Kỳ cung cấp, được thu thập trong giai đoạn từ năm 2018 đến năm 2022. Các biểu đồ này được xây dựng và hiển thị bằng công cụ Superset, giúp phân tích và trình bày dữ liệu một cách trực quan, hỗ trợ việc đánh giá các xu hướng và thông tin quan trọng trong ngành hàng không.



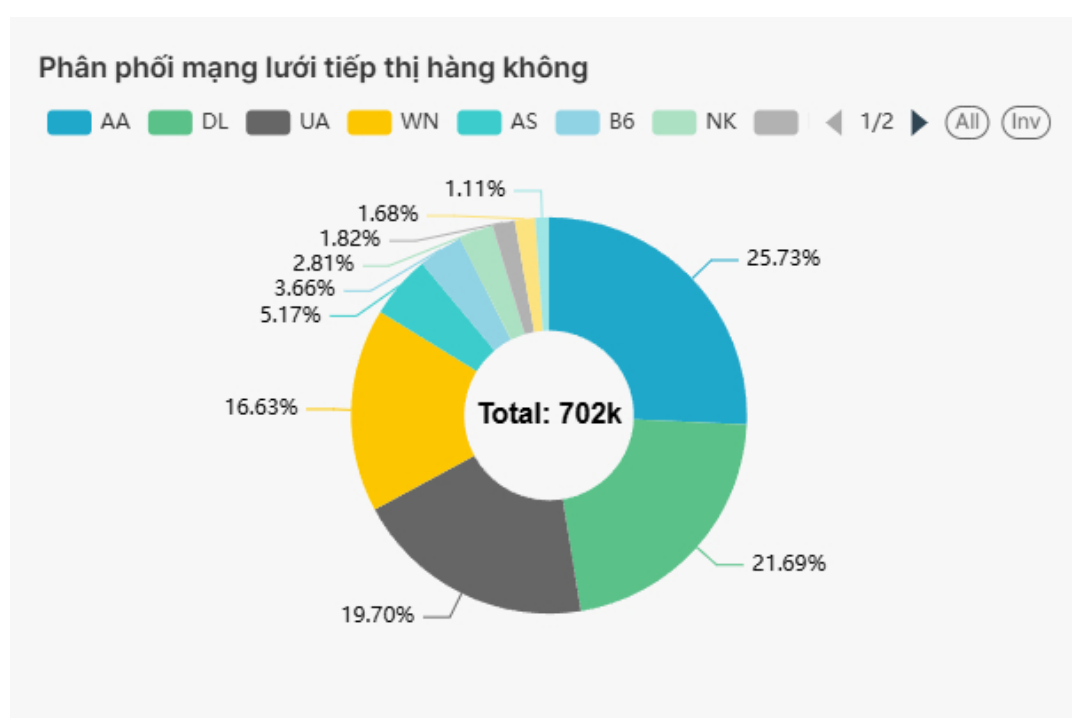
Hình 5.20: Độ trễ trung bình theo hãng hàng không năm 2018

Biểu đồ trong hình 5.20 thể hiện độ trễ trung bình theo các hãng hàng không năm 2018, được chia thành các nguyên nhân: trễ do máy bay đến muộn (Late Aircraft), trễ do hãng hàng không (Carrier), trễ liên quan đến hệ thống không lưu (NAS) và trễ do thời tiết (Weather). Hãng hàng không UA - United Airlines có mức độ trễ trung bình cao nhất (74.06 phút), tiếp theo là NK- Spirit Airlines (73.06 phút). Trong khi đó, HA - Hawaiian Airlines có độ trễ thấp nhất (48.83 phút). Nguyên nhân trễ chủ yếu ở tất cả các hãng là "Late Aircraft Delay," với tỷ lệ vượt trội so với các nguyên nhân khác. Trễ do thời tiết chiếm tỷ lệ nhỏ nhất ở hầu hết các hãng.



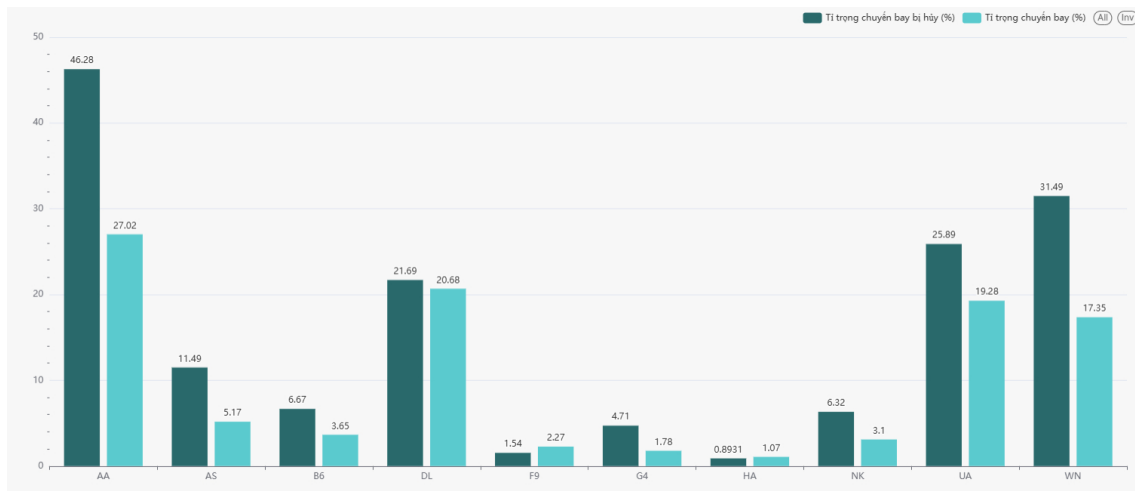
Hình 5.21: Tỉ trọng chuyến bay và tỉ trọng chuyến bay bị hủy theo ngày trong tháng 1 năm 2019

Biểu đồ trong hình 5.21 thể hiện tỉ trọng chuyến bay và tỉ trọng chuyến bay bị hủy theo ngày trong tháng 1 năm 2019. Tỉ trọng chuyến bay (màu xanh nhạt) duy trì khá ổn định trong suốt tháng, dao động khoảng 3-4%. Trong khi đó, tỉ trọng chuyến bay bị hủy (màu xanh đậm) có sự biến động mạnh, đặc biệt tăng đột biến vào các ngày 20, 30 và 31, đạt mức cao nhất hơn 12%. Các ngày còn lại, tỉ trọng chuyến bay bị hủy thường dưới 2%. Điều này cho thấy sự gia tăng đột xuất các chuyến bay bị hủy ở một số ngày cuối tháng, có thể do các yếu tố bất thường như thời tiết hoặc sự cố vận hành.



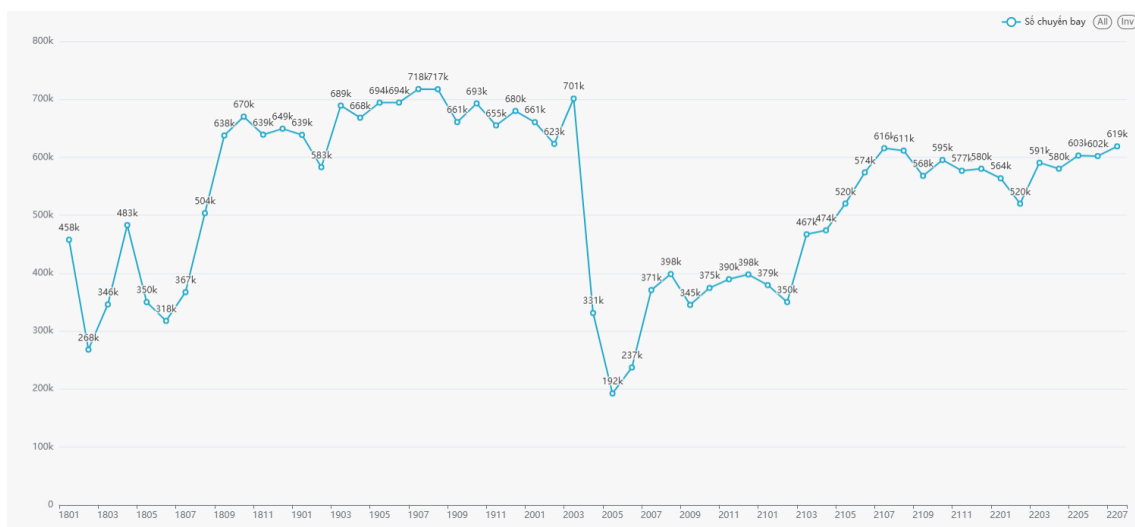
Hình 5.22: Phân phối mạng lưới tiếp thị hàng không quý 1 năm 2020

Biểu đồ trong hình 5.22 thể hiện phân phối mạng lưới tiếp thị hàng không trong quý 1 năm 2020 với tổng số 702 nghìn. Hãng AA - American Airlines chiếm tỉ lệ lớn nhất, đạt 25.73%, tiếp theo là DL - Delta Air Lines với 21.69% và UA - United Airlines với 19.70%. Hãng WN - Southwest Airlines đứng thứ tư với 16.63%. Các hãng khác còn lại chiếm tỉ lệ nhỏ hơn chỉ khoảng dưới 6%. Biểu đồ cho thấy sự phân bố tập trung chủ yếu vào một số hãng lớn, trong khi các hãng nhỏ chỉ chiếm một phần nhỏ trong mạng lưới tiếp thị.



Hình 5.23: Tỉ trọng chuyến bay và chuyến bay bị hủy theo từng hãng hàng không trong quý 4 năm 2021

Biểu đồ trong hình 5.23 thể hiện tỉ trọng chuyến bay (màu xanh nhạt) và tỉ trọng chuyến bay bị hủy (màu xanh đậm) của các hãng hàng không trong quý 4 năm 2021. Hãng AA - American Airlines có tỉ trọng chuyến bay bị hủy cao nhất, chiếm 46.28%, trong khi tỉ trọng chuyến bay là 27.02%. Hãng WN - Southwest Airlines đứng thứ hai với tỉ trọng chuyến bay bị hủy là 31.49% và tỉ trọng chuyến bay là 17.35%. Ngược lại, các hãng HA - Hawaiian Airlines và F9 - Frontier Airlines có tỉ trọng chuyến bay bị hủy thấp nhất, lần lượt là 0.89% và 1.54%, cho thấy hoạt động ổn định hơn. Các hãng như DL - Delta Air Lines, UA - United Airlines có tỉ lệ chuyến bay bị hủy dao động trong khoảng từ 19% đến 25%, tương đối cao so với một số hãng khác. Biểu đồ cho thấy sự chênh lệch rõ rệt về mức độ ổn định trong khai thác chuyến bay giữa các hãng hàng không.



Hình 5.24: Số chuyến bay theo tháng qua các năm

Biểu đồ trong hình 5.24 thể hiện số lượng chuyến bay theo từng tháng qua các

năm từ năm 2018 đến 2022. Một điểm nổi bật là sự sụt giảm đột ngột số chuyến bay vào năm 2020, đặc biệt trong các tháng đầu, do ảnh hưởng của đại dịch COVID-19, với mức thấp nhất rơi vào khoảng 331 nghìn chuyến bay. Sau đó, số lượng chuyến bay bắt đầu phục hồi dần từ cuối năm 2020 và tiếp tục tăng lên ổn định trong các năm sau, phản ánh quá trình khôi phục ngành hàng không sau đại dịch.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Chương 4 đã trình bày chi tiết về quá trình thiết kế và triển khai hệ thống, chương 5 tập trung vào việc phân tích kết quả thực nghiệm, đánh giá hiệu quả và những thách thức trong quá trình thực hiện. Trong chương cuối cùng, em sẽ đưa ra kết luận tổng quát về đồ án và đề xuất các hướng phát triển tiềm năng trong tương lai.

6.1 Kết luận

Trong đồ án này, em đã xây dựng thành công một hệ thống hồ dữ liệu phân tích chuyển bay, sử dụng các khung phần mềm mã nguồn mở trong hệ sinh thái Hadoop. Hệ thống được thiết kế để đáp ứng nhu cầu lưu trữ và xử lý dữ liệu lớn với độ chính xác cao, khả năng tự động hóa, và chi phí tối ưu.

Về mặt chức năng, hệ thống hỗ trợ phát dữ liệu trực tuyến và xử lý dữ liệu phân tán theo luồng gia tăng, với khả năng cập nhật thời gian thực thông qua các cơ chế xử lý dữ liệu và lập lịch công việc hiệu quả. Hệ thống không chỉ đảm bảo khả năng lưu trữ khối lượng dữ liệu lớn với chi phí thấp mà còn hỗ trợ mở rộng linh hoạt nhờ kiến trúc hồ dữ liệu. Các tính năng kiểm thử và trực quan hóa dữ liệu cũng được tích hợp, giúp xác minh tính chính xác của các quy trình xử lý dữ liệu trong hệ thống.

Về hiệu năng, hệ thống thể hiện khả năng lưu trữ và xử lý dữ liệu phân tán hiệu quả trên nhiều nút thông qua các cơ chế xử lý song song mạnh mẽ. Các thành phần hoạt động ổn định trong phạm vi dự án và được thiết kế để dễ dàng mở rộng theo chiều ngang khi tài nguyên tính toán được tăng cường. Hệ thống cũng đảm bảo tính sẵn sàng và khả năng chịu lỗi cao.

Mặc dù đạt được những kết quả đáng kể, đồ án vẫn tồn tại một số hạn chế cần cải thiện và phát triển thêm trong tương lai. Thứ nhất, hệ thống chưa tích hợp các cơ chế bảo mật dữ liệu. Thứ hai, việc lựa chọn bộ dữ liệu chưa thực sự tối ưu, gây khó khăn trong quá trình thiết kế và triển khai. Bên cạnh đó, công cụ quản lý chất lượng dữ liệu chưa được tích hợp, và một tính năng quan trọng là tích hợp các quy trình học máy vẫn chưa được thực hiện.

6.2 Hướng phát triển

Để dự án hoàn thiện hơn, em dự kiến triển khai thêm các giải pháp bảo mật toàn diện, bao gồm bảo mật hệ thống và ứng dụng, nhằm đảm bảo an toàn dữ liệu và hoạt động của hệ thống. Đồng thời, em sẽ tập trung vào việc theo dõi và tối ưu hóa cấp phát tài nguyên cho các ứng dụng, giúp hệ thống vận hành hiệu quả hơn. Bên cạnh đó, cần phát triển tính năng quản lý chất lượng dữ liệu nhằm đảm bảo tính

chính xác và nhất quán trong quá trình xử lý.

Đặc biệt, em sẽ tích hợp các quy trình học máy và trí tuệ nhân tạo, cho phép huấn luyện mô hình trực tiếp trên hồ dữ liệu và ứng dụng các mô hình này để đưa ra quyết định theo thời gian thực. Cuối cùng, em dự định mở rộng thiết kế và triển khai hệ thống từ hạ tầng on-premise sang hạ tầng cloud, tận dụng các lợi thế về khả năng mở rộng, hiệu suất và chi phí của điện toán đám mây.

TÀI LIỆU THAM KHẢO

- [1] TranStats, *Airline on-time performance data*. [Online]. Available: https://www.transtats.bts.gov/tables.asp?QO_VQ=EFD&QO_anzr=Nv4yv0r.
- [2] Databricks, *Three data architectures*. [Online]. Available: <https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html>.
- [3] Apache Software Foundation, *Apache hadoop*. [Online]. Available: <https://apache.github.io/hadoop>.
- [4] Apache Software Foundation, *Apache spark*. [Online]. Available: <https://spark.apache.org/docs/latest>.
- [5] Apache Software Foundation, *Apache airflow*. [Online]. Available: <https://airflow.apache.org/docs>.
- [6] Apache Software Foundation, *Apache kafka*. [Online]. Available: <https://kafka.apache.org/documentation>.
- [7] Trino, *Trino*. [Online]. Available: <https://trino.io/docs/current>.
- [8] Apache Software Foundation, *Apache hive*. [Online]. Available: <https://hive.apache.org/docs/latest>.
- [9] Apache Software Foundation, *Apache superset*. [Online]. Available: <https://superset.apache.org/docs/intro>.
- [10] Cloud Native Computing Foundation, *Kubernetes*. [Online]. Available: <https://kubernetes.io/docs/home>.