

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



SOICT

BÀI TẬP LỚN
LƯU TRỮ VÀ XỬ LÝ DỮ LIỆU LỚN
ĐỀ TÀI: PHÂN TÍCH DỮ LIỆU THỜI TIẾT

NHÓM 16

Sinh viên thực hiện:	Trương Công Đạt	20215346
	Hoàng Đình Hùng	20210399
	Hoàng Công Phú	20215451
	Nguyễn Hoàng Phúc	20215452
	Trần Hồ Khánh Ly	20210561

Giảng viên hướng dẫn: TS.Trần Việt Trung

Hà Nội, Ngày 10 tháng 12 năm 2024

MỤC LỤC

CHƯƠNG 1. ĐẶT VẤN ĐỀ	1
1.1 Bài toán được lựa chọn.....	1
1.2 Tính phù hợp của bài toán với big data.....	1
1.3 Phạm vi và giới hạn của project.....	2
CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ.....	3
2.1 Kiến trúc tổng thể	3
2.2 Chi tiết từng component và vai trò.....	4
2.2.1 Dữ liệu	4
2.2.2 Terraform	5
2.2.3 Docker	5
2.2.4 Mage	6
2.2.5 Google Cloud Storage (GCS)	7
2.2.6 Google BigQuery.....	8
2.2.7 Spark	9
2.2.8 Kafka.....	11
2.2.9 Streamlit	13
2.3 Data flow và component interaction diagrams	13
2.3.1 Luồng dữ liệu từ web sang GCS và Bigquery.....	13
2.3.2 Streaming dữ liệu với kafka.....	17
2.3.3 Xử lý dữ liệu với spark.....	22
2.3.4 Trực quan hóa dữ liệu và kết quả với streamlit.....	27

CHƯƠNG 3. CHI TIẾT TRIỂN KHAI.....	29
3.1 Source code và documentation.....	29
3.2 Configuration files theo môi trường.....	29
3.3 Deployment strategy	30
3.4 Monitoring setup	30
CHƯƠNG 4. BÀI HỌC KINH NGHIỆM.....	32
4.1 Kinh nghiệm về Data Ingestion	32
4.1.1 Xử lý nhiều nguồn dữ liệu đa dạng.....	32
4.1.2 Đảm bảo Data Quality	32
4.1.3 Xử lý Late Arriving Data	33
4.2 Kinh nghiệm về Stream Processing	33
4.2.1 Kinh nghiệm 1:Exactly-once processing.....	33
4.2.2 Kinh nghiệm 2: State management.....	34
4.2.3 Kinh nghiệm 3: Recovery mechanism	34
4.3 Kinh nghiệm về Batch Processing	35
4.3.1 Kết nối giao diện Spark Web UI từ web browser	35
4.3.2 Vấn đề chia partition khi submit spark job	35
4.4 Kinh nghiệm về Data Storage.....	35
4.5 Kinh nghiệm về Performance Optimization.....	36
4.6 Kinh nghiệm về Monitoring & Debugging	37
4.7 Kinh nghiệm về Data Quality & Testing.....	38
4.7.1 Data validation	38
4.7.2 Unit testing.....	38
4.8 Kinh nghiệm về Security & Governance	38

4.9 Kinh nghiệm về Fault Tolerance.....	39
4.9.1 Kinh nghiệm 1: Failure recovery.....	39
4.9.2 Kinh nghiệm 2: Data replication.....	39
4.9.3 Kinh nghiệm 3: Backup strategies	40
4.9.4 Kinh nghiệm 4: Disaster recovery	40
4.10 Các kinh nghiệm khác	40
4.10.1 Kinh nghiệm 1: Giả lập streaming với dữ liệu không liên tục.....	40
4.10.2 Kinh nghiệm 2: Xóa nhầm máy ảo trên Google Cloud	41

DANH MỤC HÌNH VẼ

Hình 2.1	Kiến trúc hệ thống.	3
Hình 2.2	Dashboard của Mage	6
Hình 2.3	Hình ảnh Buckets trong GCS	7
Hình 2.4	Metadata của một Parquet	8
Hình 2.5	Giao diện chính của BigQuery Studio	9
Hình 2.6	Một ví dụ về query độ ẩm	9
Hình 2.7	Máy ảo trên Google Cloud Platform.	10
Hình 2.8	Giao diện hiển thị của Spark	11
Hình 2.9	Giao diện hiển thị của Spark Jobs.	11
Hình 2.10	Zookeeper	12
Hình 2.11	Broker	12
Hình 2.12	Pipeline của dữ liệu lượng mưa trong Mage	14
Hình 2.13	Pipeline về các trường weather trong dữ liệu	14
Hình 2.14	Lập lịch chạy pipeline của lượng mưa theo ngày	15
Hình 2.15	Lập lịch chạy pipeline của weather theo ngày	15
Hình 2.16	Hình ảnh về Cloud Run	16
Hình 2.17	Kết quả dữ liệu về thời tiết sau khi đưa lên data lake.	16
Hình 2.18	Kết quả dữ liệu về lượng mưa sau khi đưa lên data lake.	16
Hình 2.19	Streaming Flow	17
Hình 2.20	Quá trình đọc dữ liệu của 2 producer.	20
Hình 2.21	Messages từ Kafka Cluster.	21
Hình 2.22	Dữ liệu được lưu trữ vào GCS.	21
Hình 2.23	Dữ liệu được lưu trữ vào Google Bigquery.	21
Hình 2.24	Dữ liệu được lưu trữ vào Google Bigquery.	22
Hình 2.25	Khi spark job thực hiện gộp dữ liệu với 2 node workers được thực thi xong.	23
Hình 2.26	Khi spark job thực hiện gộp dữ liệu với 2 node workers thì 1 worker bị kill.	23
Hình 2.27	Khi spark job thực hiện gộp dữ liệu với 1 worker còn lại.	24
Hình 2.28	Khi spark job thực hiện biến đổi tính toán với 2 node workers được thực thi xong.	24
Hình 2.29	Các trường dữ liệu về thời tiết được tính toán.	25
Hình 2.30	Thông tin dữ liệu sau khi thực hiện tính toán.	25

Hình 2.31	Tính toán dữ liệu bão với 2 worker	26
Hình 2.32	Tính toán dữ liệu bão với 1 worker	26
Hình 2.33	Các trường dữ liệu về bão được tính toán.	27
Hình 2.34	Thông tin dữ liệu bão sau khi thực hiện tính toán.	27
Hình 2.35	Giao diện chung streamlit.	27
Hình 2.36	Giao diện detect bão.	28
Hình 3.1	Quick CPU/Mem/Disk in Monitoring	30
Hình 3.2	CPU/Mem/Net/Disk detail in Monitoring	31
Hình 3.3	Monitoring sum query in bigquery and request count in GCS Bucket	31

DANH MỤC BẢNG

Bảng 2.1	Danh sách các thông số đo lường và mô tả chi tiết	5
Bảng 2.2	Cấu trúc dữ liệu thu thập từ API thời tiết.	18
Bảng 4.1	So sánh hiệu năng giữa các giải pháp phân cụm	37

CHƯƠNG 1. ĐẶT VẤN ĐỀ

1.1 Bài toán được lựa chọn

Trong bối cảnh biến đổi khí hậu toàn cầu đang diễn ra ngày càng phức tạp, việc nghiên cứu và phân tích dữ liệu khí hậu đã trở thành một yếu tố quan trọng để hỗ trợ các quyết định liên quan đến quản lý tài nguyên, hoạch định chính sách và ứng phó với các thách thức môi trường. Dữ liệu khí hậu không chỉ cung cấp thông tin về các hiện tượng tự nhiên mà còn là cơ sở để dự báo và xây dựng các kịch bản về biến đổi khí hậu trong tương lai.

Một trong những nguồn dữ liệu khí hậu đáng tin cậy và phổ biến nhất hiện nay là Climate Data Store (CDS), được phát triển bởi Trung tâm Dự báo Thời tiết Trung hạn Châu Âu (ECMWF) trong khuôn khổ chương trình Copernicus. CDS cung cấp một kho dữ liệu phong phú, bao gồm thông tin về nhiệt độ, lượng mưa, gió, độ ẩm và nhiều yếu tố khí hậu khác, với phạm vi không gian và thời gian đa dạng. Điều này giúp các nhà nghiên cứu, tổ chức và cá nhân có thể tiếp cận, khai thác và xử lý dữ liệu khí hậu một cách hiệu quả để phục vụ cho nhiều mục đích khác nhau.

Trong bài tập lớn này, nhóm tập trung vào việc tìm hiểu nguồn dữ liệu từ Climate Data Store. Mục tiêu của project này là xây dựng một data pipeline tự động thu thập, biến đổi và biểu diễn lượng dữ liệu trên nhằm phục vụ cho các bài toán phân tích sau này. Mage được sử dụng để xây dựng pipeline download data từ web về và upload lên Google Cloud Storage (GCS) và Google BigQuery. Trong BigQuery, dữ liệu được làm sạch, biến đổi sử dụng Spark nhằm tạo ra bộ dữ liệu cuối cùng cho mục đích trực quan hóa. Về trực quan hóa dữ liệu, nhóm sử dụng streamlit với những ưu điểm về biểu diễn nội suy. Ngoài ra nhóm cũng giả lập dữ liệu dưới dạng streaming và sử dụng kafka để xử lý. Hệ thống cũng được tích hợp với Google Monitoring và Google Logging để có thể theo dõi tổng thể, cảnh báo khi xảy ra các bất thường và dễ dàng xử lý khi gặp lỗi.

1.2 Tính phù hợp của bài toán với big data

Ngoài tính cấp thiết và tính thời sự của dữ liệu về thời tiết, nhóm dựa chọn bộ dữ liệu này vì đáp ứng đủ tiêu chí 5V của môn học, cụ thể:

- **Khối lượng dữ liệu lớn (Volume):** Dữ liệu khí hậu bao gồm thông tin về nhiều yếu tố như nhiệt độ, độ ẩm, lượng mưa, gió và các chỉ số khác, được thu thập trên phạm vi toàn cầu với độ phân giải cao. Bên cạnh đó, dữ liệu này thường được lưu trữ theo chuỗi thời gian dài (theo giờ, ngày, tháng hoặc năm), dẫn đến khối lượng dữ liệu cực kỳ lớn, đáp ứng đặc

trưng "Volume" của Big Data.

- **Tốc độ phát sinh dữ liệu (Velocity):** Dữ liệu khí hậu liên tục được cập nhật từ các trạm quan trắc, vệ tinh và các mô hình dự báo. Tính chất liên tục này đòi hỏi một hệ thống xử lý dữ liệu thời gian thực hoặc gần thời gian thực để đảm bảo khai thác dữ liệu kịp thời, phục vụ các ứng dụng như dự báo thời tiết hoặc cảnh báo thiên tai.
- **Tính đa dạng của dữ liệu (Variety):** Dữ liệu khí hậu đến từ nhiều nguồn khác nhau và có nhiều định dạng, bao gồm dữ liệu dạng bảng (các chỉ số đo đạc), dữ liệu raster (bản đồ khí hậu), và các dữ liệu phi cấu trúc khác. Sự đa dạng này yêu cầu các phương pháp lưu trữ và xử lý dữ liệu phù hợp để quản lý hiệu quả.
- **Giá trị khai thác từ dữ liệu (Value):** Dữ liệu khí hậu có giá trị lớn trong việc hỗ trợ ra quyết định ở nhiều lĩnh vực, từ hoạch định chính sách, quản lý tài nguyên đến dự báo thời tiết và ứng phó với biến đổi khí hậu. Bài toán này cho thấy rõ giá trị của việc xử lý Big Data, không chỉ dừng lại ở lưu trữ mà còn mở rộng đến phân tích và khai thác thông tin quan trọng.
- **Tính xác thực của dữ liệu (Veracity):** Dữ liệu từ CDS được cung cấp bởi các tổ chức uy tín như ECMWF, đã qua kiểm định và hiệu chỉnh nghiêm ngặt, đảm bảo độ tin cậy cao. Do đó, bộ dữ liệu này hoàn toàn có thể đáp ứng được yêu cầu cho việc học tập, nghiên cứu các công nghệ thu thập, lưu trữ và xử lý dữ liệu được giới thiệu trong môn học.

1.3 Phạm vi và giới hạn của project

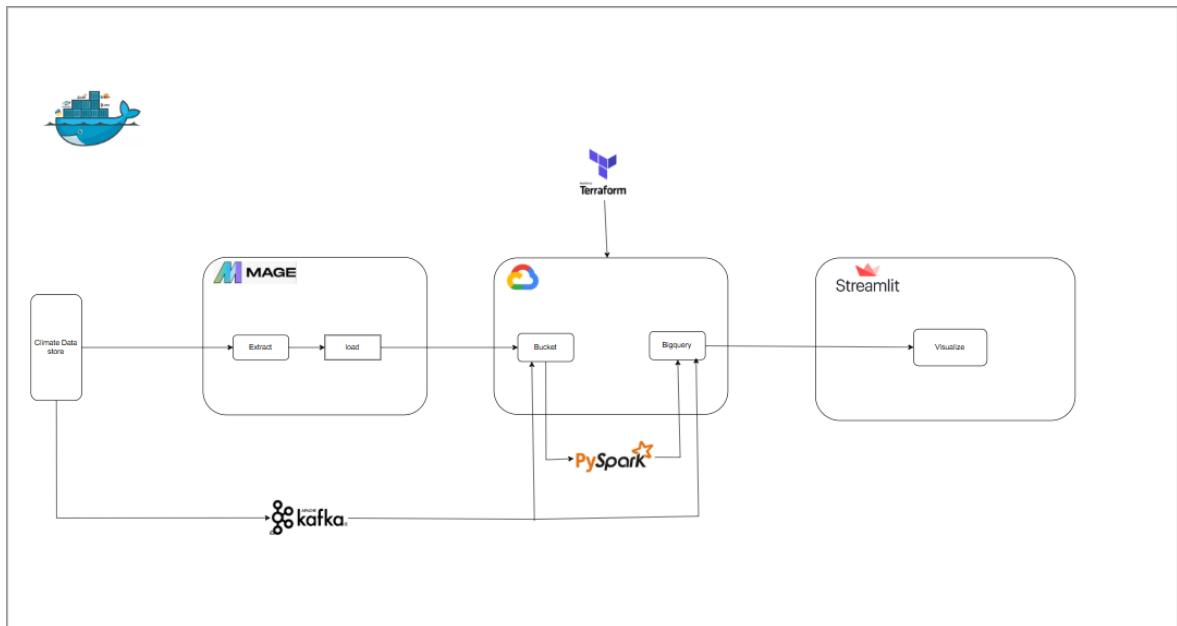
Trong phạm vi và giới hạn của môn học, nhóm tập trung vào việc xây dựng hệ thống xử lý dữ liệu khí hậu tự động từ nguồn Climate Data Store (CDS) với các bước chính:

- Thu thập và lưu trữ dữ liệu trên Google Cloud Storage (GCS).
- Xử lý, làm sạch và biến đổi dữ liệu bằng Apache Spark trên nền tảng Google BigQuery.
- Trực quan hóa dữ liệu qua Streamlit, bao gồm hướng gió, nhiệt độ cảm nhận, áp suất mực nước biển, áp suất bề mặt, tổng diện tích mây bao phủ, tổng lượng tinh thể băng trong mây, tổng lượng nước trong mây, thông tin các cơn bão.
- Thủ nghiệm giả lập streaming bằng cách sử dụng Kafka.
- Do giới hạn thời gian và tài nguyên, phạm vi xử lý dữ liệu từ 2022-2024 với dữ liệu thời tiết ở Việt Nam (vĩ độ từ 8° - 24° vĩ bắc, kinh độ từ 102° - 112° kinh đông).

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

2.1 Kiến trúc tổng thể

Kiến trúc và các công nghệ sử dụng:



Hình 2.1: Kiến trúc hệ thống.

Kiến trúc xử lý dữ liệu được nhóm sử dụng là kiến trúc Lambda kết hợp 2 hướng xử lý:

- **Batch Processing (Xử lý lô):** Xử lý một lượng lớn dữ liệu trong khoảng thời gian từ tháng 1/2022 đến 10/2024.
- **Stream Processing (Xử lý thời gian thực):** Giả lập xử lý thời gian thực vào ngày 01/11/2024.

Các thành phần trong hệ thống:

- **Thành phần xử lý thời gian thực (Stream Layer):** Dữ liệu từ Climate Data Store được gửi tới Kafka. Dữ liệu thời gian thực từ Kafka được chuyển đến Spark để thực hiện các phép biến đổi và cập nhật vào BigQuery.
- **Thành phần xử lý lô (Batch Layer):** Mage giúp thu thập dữ liệu lịch sử từ Climate Data Store và tải lên Google Cloud Storage (GCS) để lưu trữ. Sau khi dữ liệu được tải lên GCS, Spark thực hiện việc xử lý, làm sạch và biến đổi dữ liệu. Dữ liệu đã qua xử lý được tải vào BigQuery.

- Thành phần tích hợp dữ liệu (Serving Layer):** BigQuery đóng vai trò là kho dữ liệu hợp nhất (Serving Layer), nơi chứa cả dữ liệu thời gian thực và dữ liệu lô đã xử lý. Streamlit kết nối trực tiếp với BigQuery, giúp hiển thị dữ liệu và tạo giao diện trực quan cho người dùng.

2.2 Chi tiết từng component và vai trò

2.2.1 Dữ liệu

- Dữ liệu thời tiết lấy từ *Climate Data Store* được tổ chức theo giờ.
- Dữ liệu được lấy trong khoảng thời gian từ 01/01/2022 – 31/10/2024 và trong phạm vi Việt Nam.
- Kích thước dữ liệu 15GB với gần 80000000 bản ghi.
- Các trường dữ liệu bao gồm:

Thông số	Mô tả
Time	Thời gian đo
Latitude	Vĩ độ
Longitude	Kinh độ
10m u-component of wind	Thành phần gió theo hướng Đông – Tây tại độ cao 10 mét
10m v-component of wind	Thành phần gió theo hướng Bắc – Nam tại độ cao 10 mét
2m dewpoint temperature	Nhiệt độ điểm sương ở độ cao 2 mét
2m temperature	Nhiệt độ không khí ở độ cao 2 mét
Mean sea level pressure	Áp suất trung bình tại mực nước biển
Sea surface temperature	Nhiệt độ bề mặt nước biển
Surface pressure	Áp suất bề mặt tại điểm đo (trên mặt đất hoặc trên mặt biển)
Total cloud cover	Tổng độ che phủ của mây
Total column cloud ice water	Tổng lượng nước trong cột khí quyển ở dạng tinh thể băng
Total column cloud liquid water	Tổng lượng nước trong cột khí quyển ở dạng tinh thể lỏng
Total precipitation	Lượng mưa

Thông số	Mô tả
----------	-------

Bảng 2.1: Danh sách các thông số đo lường và mô tả chi tiết

2.2.2 Terraform

Terraform được sử dụng để quản lý và tự động hóa việc triển khai cơ sở hạ tầng trên Google Cloud Platform. Nó cho phép định nghĩa các tài nguyên như Google Cloud Storage, Google BigQuery dưới dạng code (IaC), giúp việc triển khai và quản lý dễ dàng hơn. Ngoài ra, Terraform cũng được dùng để cấu hình và triển khai các dịch vụ khác như Cloud Run cho Mage và quản lý cơ sở hạ tầng cho Kafka và PySpark.

Lợi ích:

- Infrastructure as Code (IaC):** Cơ sở hạ tầng được quản lý dưới dạng code, giúp dễ dàng theo dõi, phiên bản hóa và tự động hóa việc triển khai.
- Triển khai đơn giản và nhanh chóng:** Tự động hóa việc tạo và cấu hình các tài nguyên, giảm thiểu lỗi do cấu hình thủ công.
- Khả năng tái sử dụng:** Các module Terraform có thể được tái sử dụng cho các dự án khác.

Nhược điểm:

- Đòi hỏi kiến thức về HCL:** Cần học ngôn ngữ HCL để viết cấu hình Terraform.
- Quản lý state file:** Cần quản lý state file để theo dõi trạng thái của cơ sở hạ tầng.

2.2.3 Docker

Docker được sử dụng để đóng gói, phân phối và chạy các ứng dụng trong môi trường cô lập (container). Trong dự án này, Docker được sử dụng để đóng gói ứng dụng Mage và tạo môi trường chạy nhất quán cho Kafka và Zookeeper. Việc sử dụng Docker giúp đơn giản hóa việc triển khai và đảm bảo tính nhất quán giữa các môi trường khác nhau (phát triển, kiểm thử, sản xuất).

Lợi ích:

- Tính nhất quán:** Docker đảm bảo ứng dụng chạy giống nhau trên mọi môi trường, từ máy tính cá nhân của nhà phát triển đến môi trường production trên cloud.
- Cô lập:** Các ứng dụng chạy trong container được cô lập với nhau và với hệ điều hành host, tránh xung đột về dependency và tài nguyên.

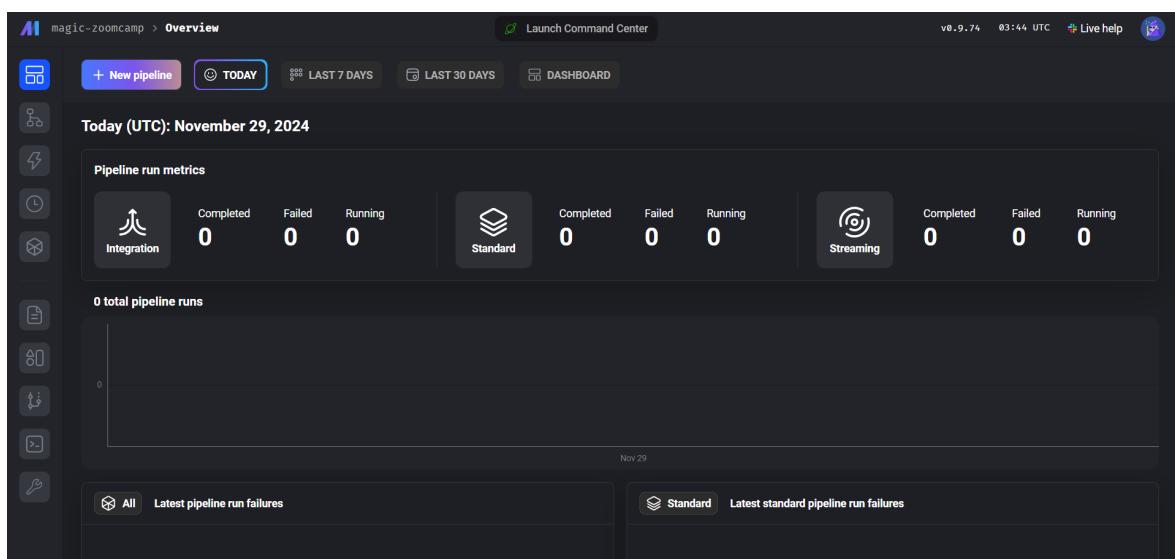
- **Triển khai dễ dàng:** Docker image có thể dễ dàng được triển khai lên các nền tảng cloud như Google Cloud Run.
- **Khả năng mở rộng:** Dễ dàng scale ứng dụng bằng cách chạy nhiều container.

Nhược điểm:

- **Độ phức tạp:** Việc quản lý nhiều container và orchestration có thể trở nên phức tạp.
- **Hiệu suất:** Chạy ứng dụng trong container có thể ảnh hưởng đến hiệu suất, tuy nhiên ảnh hưởng này thường không đáng kể.

2.2.4 Mage

Mage là một phần mềm vận hành workflow tương tự như Airflow, cho phép người dùng xây dựng, giám sát và vận hành data pipelines một cách hiệu quả. Mage có thể biến bất cứ hàm nào trong Python thành một đơn vị có thể giám sát và vận hành được. Bên cạnh các chức năng như logging, scheduling, caching, Mage còn cung cấp các hàm, tiện ích có sẵn cho phép xử lý lỗi và chạy các tác vụ song song. Đây cũng là lí do nhóm lựa chọn sử dụng Mage cho project này.



Hình 2.2: Dashboard của Mage

Trong Mage, các pipeline dữ liệu được cấu thành từ các cell, mỗi cell có một nhiệm vụ riêng với các hàm template hỗ trợ tương tác với các dịch vụ cloud như Google cloud storage và Big query:

- Data loader: Extract dữ liệu.
- Transformer: Biến đổi dữ liệu.

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

- Data exporter: Load dữ liệu vào data lake và data warehouse.

2.2.5 Google Cloud Storage (GCS)

Google Cloud Storage (GCS) là một dịch vụ lưu trữ đối tượng trên cloud, cung cấp khả năng lưu trữ dữ liệu với dung lượng lớn, độ tin cậy cao và chi phí hợp lý. Trong dự án này, GCS được sử dụng như một kho lưu trữ dữ liệu thô (Data Lake) cho dữ liệu thời tiết được thu thập từ Climate Data Store. Dữ liệu được lưu trữ dưới dạng các file Parquet, một định dạng cột hiệu quả cho việc truy vấn và phân tích dữ liệu lớn.

Lợi ích:

- **Khả năng mở rộng:** GCS có thể lưu trữ một lượng dữ liệu khổng lồ và tự động mở rộng theo nhu cầu.
- **Độ tin cậy cao:** Dữ liệu được lưu trữ một cách an toàn và có khả năng phục hồi cao.
- **Tích hợp với các dịch vụ khác của GCP:** GCS tích hợp chặt chẽ với các dịch vụ khác như BigQuery, Dataproc và Dataflow, giúp dễ dàng xử lý và phân tích dữ liệu.
- **Chi phí hợp lý:** GCS cung cấp nhiều lựa chọn lưu trữ với mức giá khác nhau, phù hợp với nhu cầu và ngân sách của dự án.

Nhược điểm:

- **Chi phí truy xuất dữ liệu:** Cần lưu ý đến chi phí truy xuất dữ liệu, đặc biệt là khi truy xuất một lượng lớn dữ liệu thường xuyên.

The screenshot shows the Google Cloud Storage interface for a bucket named 'weather_bigdata_20241'. The bucket is located in the US (multiple regions in United States) and has a standard storage class, public access set to 'Not public', and soft delete protection. The 'OBJECTS' tab is selected, displaying a list of objects. The objects are part of a main folder named 'weather_all_main'. The list includes several files with names starting with '_SUCCESS' and 'part-' followed by various identifiers. Each object entry shows its name, size (e.g., 0 B, 57.3 MB), type (application/octet-stream), creation date (Dec 7, 2024), storage class (Standard), last modified date (Dec 7, 2024), public access (Not public), and version history (indicated by three dots). There are also icons for more actions like edit, delete, and copy.

Hình 2.3: Hình ảnh Buckets trong GCS

The screenshot shows the 'Object details' page for a file named 'part-00000-6519171a-ed6b-4b38-a574-649a8d5f21b0-c000.snappy.parquet'. The 'Overview' section displays the following metadata:

	Value
Type	application/octet-stream
Size	57.3 MB
Created	Dec 7, 2024, 7:07:38 PM
Last modified	Dec 7, 2024, 7:07:38 PM
Storage class	Standard
Custom time	—
Public URL	Not applicable
Authenticated URL	https://storage.cloud.google.com/weather_bigdata_20241/weather_all_main/part-00000-6519171a-ed6b-4b38-a574-649a8d5f21b0-c000.snappy.parquet
gsutil URI	gs://weather_bigdata_20241/weather_all_main/part-00000-6519171a-ed6b-4b38-a574-649a8d5f21b0-c000.snappy.parquet

The 'Permissions' section shows:

- Public access: Not public
- Protection: Version history —
- Retention expiration time: None
- Object retention retain until time: None
- Bucket retention retain until time: None

Hình 2.4: Metadata của một Parquet

2.2.6 Google BigQuery

Google BigQuery là một kho dữ liệu phân tích doanh nghiệp, có khả năng mở rộng cao trên cloud. Trong dự án này, BigQuery được sử dụng để lưu trữ và phân tích dữ liệu thời tiết đã được xử lý. Dữ liệu từ GCS được tải vào BigQuery, nơi nó có thể được truy vấn bằng SQL để phân tích và trực quan hóa.

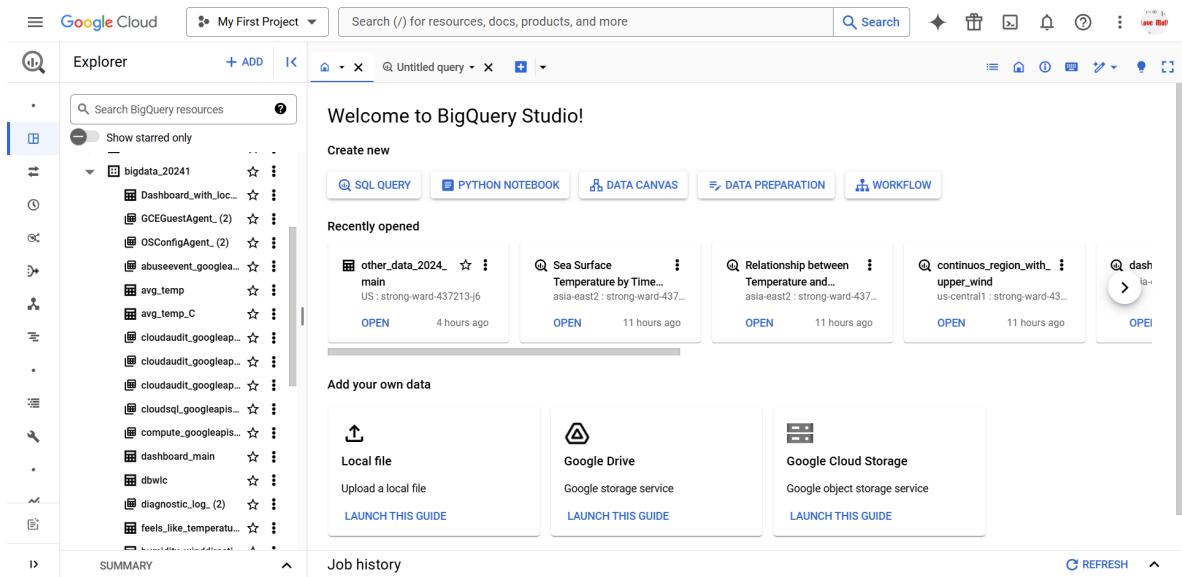
Lợi ích:

- Hiệu suất truy vấn cao:** BigQuery được thiết kế để xử lý các truy vấn trên tập dữ liệu lớn với tốc độ rất nhanh.
- Khả năng mở rộng:** BigQuery có thể xử lý petabyte dữ liệu và tự động mở rộng theo nhu cầu.
- Phân tích SQL:** BigQuery hỗ trợ SQL tiêu chuẩn, giúp dễ dàng phân tích dữ liệu.
- Tích hợp với các dịch vụ khác của GCP:** BigQuery tích hợp với các dịch vụ khác như Data Studio, Looker giúp dễ dàng chia sẻ kết quả phân tích.
- Serverless:** Không cần quản lý cơ sở hạ tầng, giúp tiết kiệm thời gian và công sức.

Nhược điểm:

- Chi phí:** BigQuery tính phí dựa trên dung lượng lưu trữ và lượng dữ liệu được xử lý. Cần quản lý chi phí một cách hiệu quả.

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ



Hình 2.5: Giao diện chính của BigQuery Studio

This screenshot shows a specific query named '1.Relative Humidity' in the BigQuery Studio. The query code is as follows:

```
1 SELECT
2   latitude,
3   longitude,
4   valid_time,
5   ((EXP((17.625 * (d2m - 273.15)) / (d2m - 273.15) + 243.84)) / EXP((17.625 * (t2m - 273.15)) / (t2m - 273.15) + 243.84)) * 100) AS
6   relative_humidity
7   FROM
8   `strong-ward-437213-j6.bigdata_20241.other_data_2024_main`
9   ORDER BY
10   valid_time, latitude, longitude
11
```

The results table below the code shows data for 18 rows, with columns: Row, latitude, longitude, valid_time, and relative_humidity. The results are as follows:

Row	latitude	longitude	valid_time	relative_humidity
1	8.0	102.0	2024-01-01 00:00:00 UTC	87.85151769034...
2	8.0	102.25	2024-01-01 00:00:00 UTC	87.61027401996...
3	8.0	102.5	2024-01-01 00:00:00 UTC	86.34171743496...
4	8.0	102.75	2024-01-01 00:00:00 UTC	84.29721962686...
5	8.0	103.0	2024-01-01 00:00:00 UTC	82.67973621276...
6	8.0	103.25	2024-01-01 00:00:00 UTC	82.85321681214...
7	8.0	103.5	2024-01-01 00:00:00 UTC	82.173666995667
...

Hình 2.6: Một ví dụ về query độ ẩm

2.2.7 Spark

Apache Spark là một nền tảng xử lý dữ liệu phân tán mã nguồn mở, được thiết kế để xử lý khối lượng lớn dữ liệu (big data) với tốc độ cao. Spark cung cấp một giao diện lập trình dễ sử dụng và hỗ trợ nhiều ngôn ngữ lập trình phổ biến như Python, Scala, Java và R. Apache Spark hỗ trợ batch processing hiệu quả, giúp xử lý khối lượng lớn dữ liệu tĩnh trong một hoặc nhiều bước tính toán. Batch processing thường được sử dụng cho các tác vụ như xử lý ETL (Extract, Transform, Load).

Transform, Load) tổng hợp dữ liệu, phân tích dữ liệu lớn, và tạo các báo cáo định kỳ.

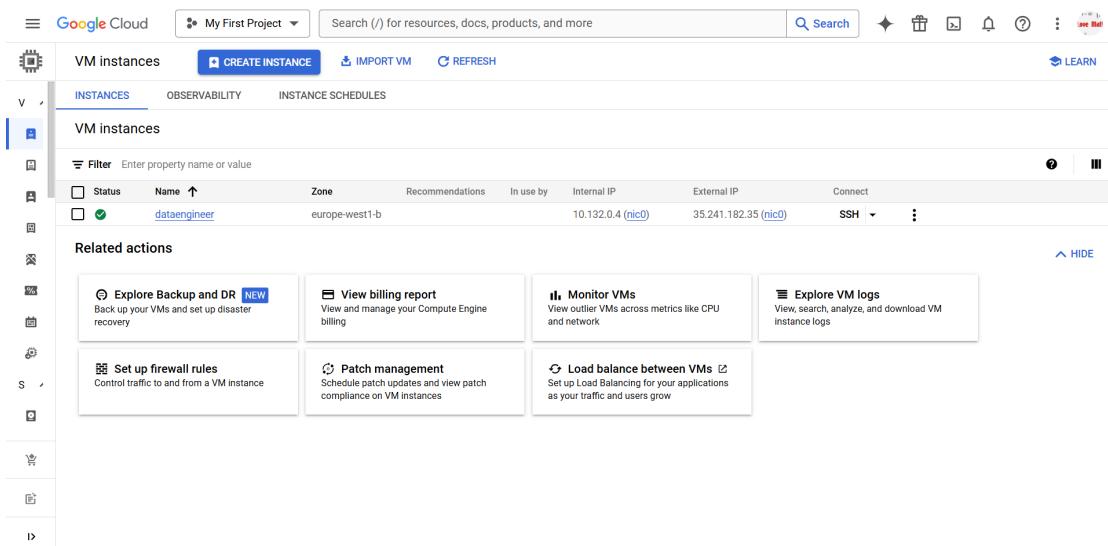
Ưu điểm:

- Hiệu suất cao nhờ cơ chế in-memory computing giúp tăng tốc độ xử lý batch processing so với các hệ thống truyền thống.
- Dễ dàng mở rộng, hỗ trợ dữ liệu phân tán.
- Tích hợp nhiều nguồn dữ liệu khác nhau như Cassandra, HBase, SQL...
- Dễ dàng tích hợp với hệ sinh thái dữ liệu lớn, chẳng hạn như Apache Kafka, Hive hoặc các công cụ học máy như Mlib.
- Tái sử dụng dữ liệu trong bộ nhớ và hỗ trợ chạy trên các hệ thống cloud.

Nhược điểm:

- Việc xử lý trong bộ nhớ yêu cầu nhiều tài nguyên, đặc biệt là RAM.
- Đối với các tác vụ đơn giản, Spark không mang lợi ích đáng kể so với các công cụ như Pandas hoặc SQL thông thường.
- Spark thường có độ trễ khi khởi tạo các job, không lý tưởng với các tác vụ cần kết quả nhanh.

Trong bài tập lớn, nhóm sử dụng Spark để xử lý dữ liệu theo batch processing. Apache Spark được cài đặt trên máy ảo của Google Cloud Platform (GCP) với cấu hình gồm 1 master node và 2 worker node.



Hình 2.7: Máy ảo trên Google Cloud Platform.

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

The screenshot shows the Apache Spark master interface at <http://34.140.61.65:8080>. It displays the following information:

- URL: spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077
- Alive Workers: 2
- Cores in use: 4 Total, 0 Used
- Memory in use: 6.0 GiB Total, 0.0 B Used
- Resources in use:
- Applications: 0 Running, 0 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241203030126-10.132.0.3-42923	10.132.0.3:42923	ALIVE	2 (0 Used)	3.0 GiB (0.0 B Used)	
worker-20241203030439-10.132.0.3-43123	10.132.0.3:43123	ALIVE	2 (0 Used)	3.0 GiB (0.0 B Used)	

Running Applications (0)
Completed Applications (0)

Hình 2.8: Giao diện hiển thị của Spark.

Thông tin về các Spark jobs được hiển thị qua localhost:4040 từ máy tính cá nhân (local).

The screenshot shows the Apache Spark Jobs UI at <http://localhost:4040/jobs/>. It displays the following information:

User: phuc_0703
Total Uptime: 3.0 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 5

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	parquet at NativeMethodAccessImpl.java:0 parquet at NativeMethodAccessImpl.java:0 (kill)	2024/11/13 00:32:55	13 s	0/1	0/8 (4 running)

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at NativeMethodAccessImpl.java:0 count at NativeMethodAccessImpl.java:0	2024/11/13 00:32:52	97 ms	1/1 (1 skipped)	1/1 (8 skipped)
3	count at NativeMethodAccessImpl.java:0 count at NativeMethodAccessImpl.java:0	2024/11/13 00:31:37	1.3 min	1/1	8/8
2	showString at NativeMethodAccessImpl.java:0	2024/11/13 00:30:51	46 s	1/1	1/1

Hình 2.9: Giao diện hiển thị của Spark Jobs.

2.2.8 Kafka

Kafka là một nền tảng truyền tải dữ liệu theo thời gian thực (real-time data streaming platform). Vì nguồn dữ liệu nhóm xử dụng được cập nhật ba ngày một lần, vì thế nhóm tiến hành giả lập luồng xử lý dữ liệu realtime. Do đó, Kafka được sử dụng để mô phỏng dòng dữ liệu thời tiết từ website Climate Data Store. Dữ liệu từ nguồn gốc ban đầu được tải về theo từng giờ, nhờ Kafka, dữ liệu được chuyển đổi thành luồng dữ liệu liên tục (streaming) giống như việc nhận dữ liệu thời gian thực từ các trạm đo.

Kafka cung cấp khả năng lưu trữ ngắn hạn dữ liệu thời tiết trong các topic, đảm bảo rằng dữ

liệu có thể được truy cập ngay cả khi một thành phần trong hệ thống bị lỗi hoặc cần khởi động lại.

Với dự án này, nhóm sử dụng kafka như là một thành phần trung gian giữa Nguồn dữ liệu (Climate Data Store) với Hệ thống lưu trữ (Google Cloud Storage - GCS) và Google Bigquery. Kafka cluster sẽ được xây dựng sử dụng Docker trên một máy ảo của GCP. Cluster gồm có 1 zookeeper và 2 brokers. Zookeeper được sử dụng để quản lý metadata, điều phối các broker và phân phối vai trò leader cho các phân vùng (partitions), đảm bảo tính nhất quán và khả năng chịu lỗi của hệ thống. Hai broker chịu trách nhiệm lưu trữ, quản lý và truyền tải dữ liệu, với mỗi broker xử lý một hoặc nhiều phân vùng của topic. Zookeeper giám sát trạng thái của các broker, trong khi broker đảm bảo lưu trữ dữ liệu an toàn, phục vụ yêu cầu từ producer và consumer, và hỗ trợ nhân bản (replication) để tăng khả năng chịu lỗi và hiệu suất.

- Cấu hình Zookeeper trong Docker:

```
zookeeper:
  image: confluentinc/cp-zookeeper:7.2.0
  hostname: zookeeper
  container_name: zookeeper
  ports:
    - '2181:2181'
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
```

Hình 2.10: Zookeeper

- Cấu hình của một Kafka broker trong Docker:

```
broker1:
  image: confluentinc/cp-kafka:7.2.0
  hostname: broker1
  container_name: broker1
  depends_on:
    - zookeeper
  ports:
    - '9092:9092'
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_LISTENERS: PLAINTEXT://broker1:9092,PLAINTEXT_HOST://broker1:9092
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker1:9092,PLAINTEXT_HOST://34.143.148.1:9092
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 2
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 2
    KAFKA_DEFAULT_REPLICATION_FACTOR: 2
    KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 2
    KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
    CONFLUENT_SUPPORT_CUSTOMER_ID: "anonymous"
    CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: broker1:9092
    CONFLUENT_METRICS_REPORTER_ZOOKEEPER_CONNECT: zookeeper:2181
    CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 2
```

Hình 2.11: Broker

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

Ngoài ra, nhóm còn cái đặt thêm một vài thành phần trong hệ sinh thái kafka bao gồm Schema Registry, Control Center, Kafka REST. Schema Registry giúp quản lý và bảo vệ các schema dữ liệu, đảm bảo tính nhất quán khi trao đổi dữ liệu giữa các producer và consumer. Control Center cung cấp giao diện người dùng để giám sát và quản lý Kafka cluster, cho phép theo dõi hiệu suất, tạo topic, và thiết lập các cảnh báo. Còn Kafka REST cung cấp API RESTful, giúp các ứng dụng không phải Java có thể tương tác với Kafka, gửi và nhận dữ liệu từ các topic mà không cần sử dụng Kafka client. Những công cụ này giúp tăng tính linh hoạt, dễ dàng tích hợp và quản lý hệ thống Kafka hiệu quả hơn.

Về cơ chế xử lý dữ liệu, nhóm sử dụng 2 producer tương ứng với 2 bootstraps server là broker1 và broker2 để gửi dữ liệu vào topic, 1 consumer để tiêu thụ dữ liệu. Hai producer sẽ gửi dữ liệu vào cùng một topic có tên là weathers, và Kafka phân phối dữ liệu này vào các partition của topic. Topic này có 4 partition, được phân phối trên hai broker, với mỗi broker chịu trách nhiệm lưu trữ một số partition. Khi consumer kết nối đến Kafka cluster, nó sẽ tiêu thụ dữ liệu từ các partition của topic. Consumer sẽ nhận dữ liệu từ các broker chứa partition mà nó cần đọc. Hệ thống Kafka hỗ trợ cơ chế replication, đảm bảo mỗi partition có bản sao trên ít nhất một broker khác, giúp tăng tính sẵn sàng và khả năng chịu lỗi. Nếu một broker bị hỏng, Kafka tự động chuyển sang broker còn lại để tiếp tục phục vụ yêu cầu. Về phía Consumer, sẽ lấy dữ liệu và lưu trữ vào Google Cloud Storage và Google Bigquery, phục vụ cho quá trình xử lý về sau.

2.2.9 Streamlit

Streamlit là một thư viện Python mã nguồn mở được thiết kế để tạo các ứng dụng web một cách nhanh chóng và dễ dàng từ mã Python. Nó đặc biệt hữu ích cho các nhà khoa học dữ liệu, kỹ sư học máy và những người làm việc với dữ liệu, cho phép họ trực quan hóa và tương tác với dữ liệu theo cách trực quan mà không cần kiến thức sâu về lập trình web.

Trong project này, streamlit đóng vai trò là một giao diện cho phép người sử dụng có thể xem, và thông kê thông tin một cách trực quan. Các dữ liệu về thời tiết sẽ được tạo thành các ảnh giúp dễ dàng quan sát toàn cảnh theo ngày. Ngoài ra các dữ liệu sẽ được biểu diễn thành các biểu đồ đường theo thời điểm giúp quan sát biến động thời tiết theo giờ.

2.3 Data flow và component interaction diagrams

2.3.1 Luồng dữ liệu từ web sang GCS và Bigquery

Trong project, nhóm sử dụng 2 pipeline chính:

- Pipeline dữ liệu lượng mưa: Data loadder lấy dữ liệu từ Climate Data Store qua API và Data exporter đưa vào Google cloud storage.

The screenshot shows the Mage AI interface with the 'rain_pipeline' selected. On the left, the file structure is visible with 'crawl_rain.py' open. The code defines a 'crawl_rain' function that loads data from an API using a template and a specific client. It sets the dataset to 'reanalysis-era5-single-levels' and the year to 2024. The right side shows a data flow diagram with two nodes: 'crawl_rain' (Data loader) and 'rain_to_gcs' (Data exporter). The 'crawl_rain' node is connected to the 'rain_to_gcs' node.

```

import io
import pandas as pd
import requests
import glob
import cdsapi
import xarray as xr
import os
from datetime import datetime, timedelta
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test

@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    client = cdsapi.Client()
    dataset = "reanalysis-era5-single-levels"
    year = 2024

    # Đặt start_date là ngày 24/10/2024
    # start_date = datetime.now() - timedelta(days=8)
    # end_date = datetime.now() - timedelta(days=8)
    start_date = datetime(2022, 1, 1)
    end_date = datetime(2024, 10, 14)

    # Tạo thư mục crawl_weather nếu chưa tồn tại
    os.makedirs("crawl_weather", exist_ok=True)

```

Hình 2.12: Pipeline của dữ liệu lượng mưa trong Mage

- Pipeline về các trường còn lại trong dữ liệu: Tương tự với pipeline dữ liệu lượng mưa nhưng do có sự khác biệt về cách chọn thời gian đo nên các trường dữ liệu còn lại được tách thành 1 pipeline riêng.

The screenshot shows the Mage AI interface with the 'weather_pipeline' selected. On the left, the file structure is visible with 'crawl_weather.py' open. The code defines a 'crawl_weather' function that loads data from an API using a template and a specific client. It sets the dataset to 'reanalysis-era5-single-levels' and the year to 2024. The right side shows a data flow diagram with four nodes: 'crawl_weather' (Data loader), 'load_weather_from_gcs' (Data loader), 'weather_to_gcs_partitioned' (Data exporter), and 'weather_bq' (Data exporter). The 'crawl_weather' node is connected to 'load_weather_from_gcs', which is then connected to both 'weather_to_gcs_partitioned' and 'weather_bq'.

```

import glob
import cdsapi
import xarray as xr
import os
from datetime import datetime, timedelta
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test

@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    client = cdsapi.Client()
    dataset = "reanalysis-era5-single-levels"
    year = 2024

    # Đặt start_date là ngày 24/10/2024
    # start_date = datetime.now() - timedelta(days=8)
    # end_date = datetime.now() - timedelta(days=8)
    start_date = datetime(2022, 1, 1)
    end_date = datetime(2024, 10, 14)

    # Tạo thư mục crawl_weather nếu chưa tồn tại
    os.makedirs("crawl_weather", exist_ok=True)

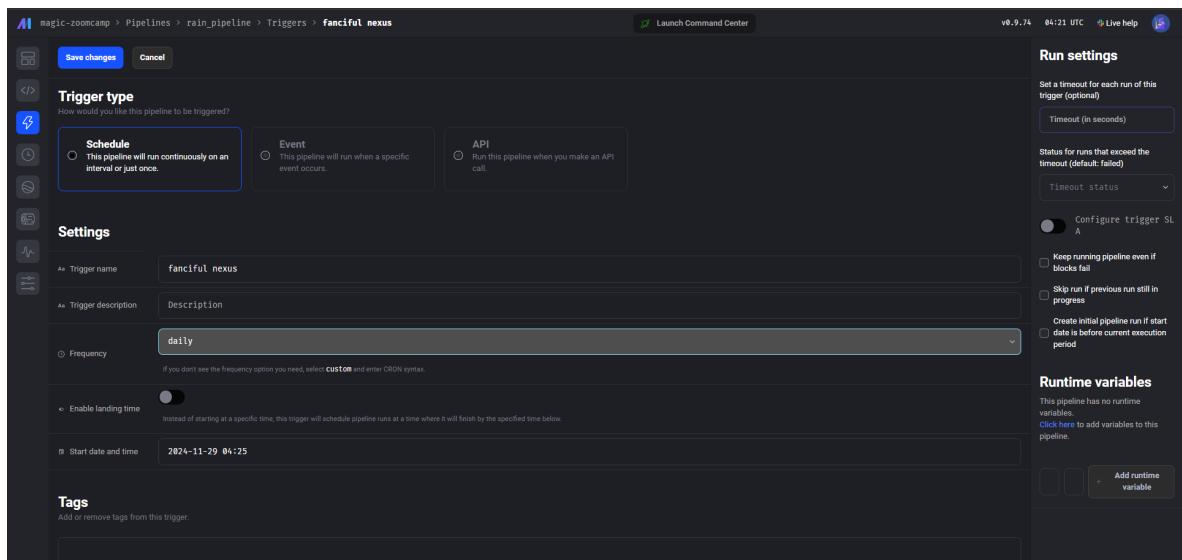
```

Hình 2.13: Pipeline về các trường weather trong dữ liệu

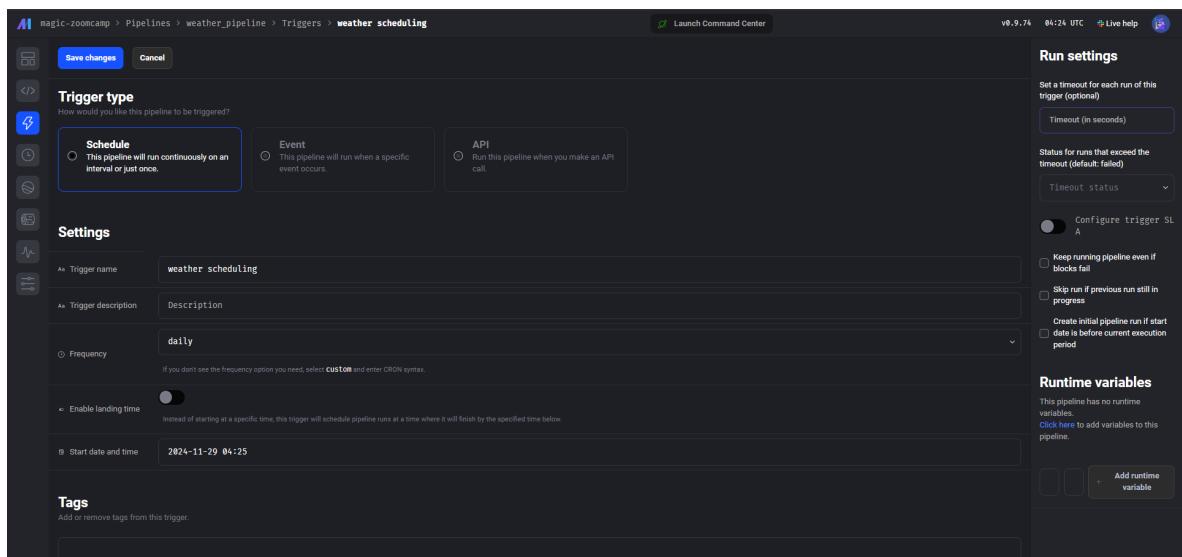
- Ngoài ra, 1 pipeline dữ liệu được sử dụng để lấy dữ liệu từ Google cloud storage và đưa vào Big query phục vụ trong quá trình phân tích dữ liệu.

Lập lịch chạy các pipeline theo ngày:

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

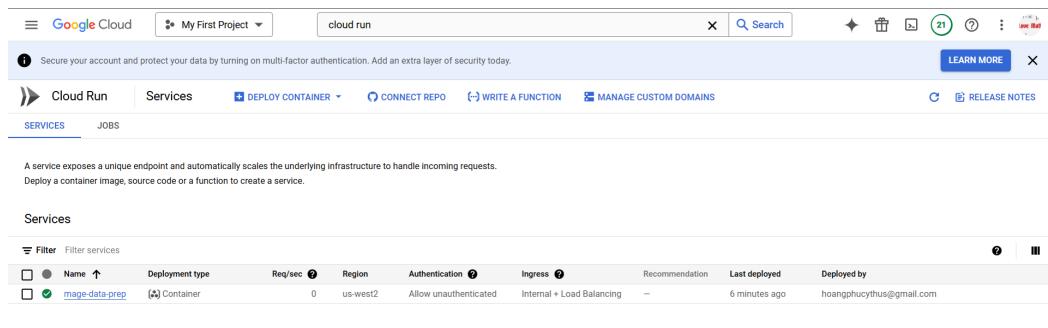


Hình 2.14: Lập lịch chạy pipeline của lượng mưa theo ngày



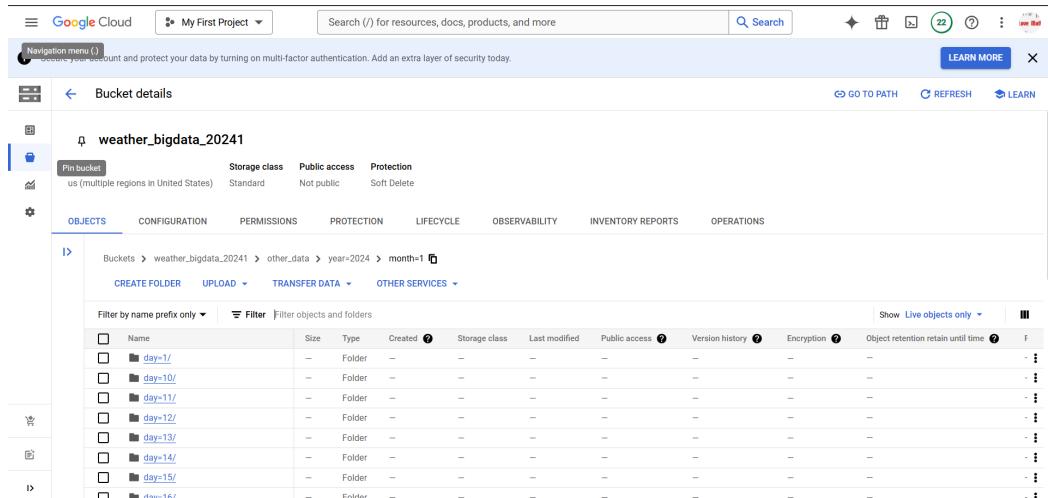
Hình 2.15: Lập lịch chạy pipeline của weather theo ngày

Tất cả code của để thực hiện các pipeline được đóng gói thành 1 Docker image và được triển khai lên Cloud Run. Cloud Run là môi trường máy tính được quản lý tổng thể để triển khai và mở rộng quy mô các container phi máy chủ mà không cần lo lắng về việc chuẩn bị máy móc vật lý, cấu hình các cụm hoặc autoscaling.

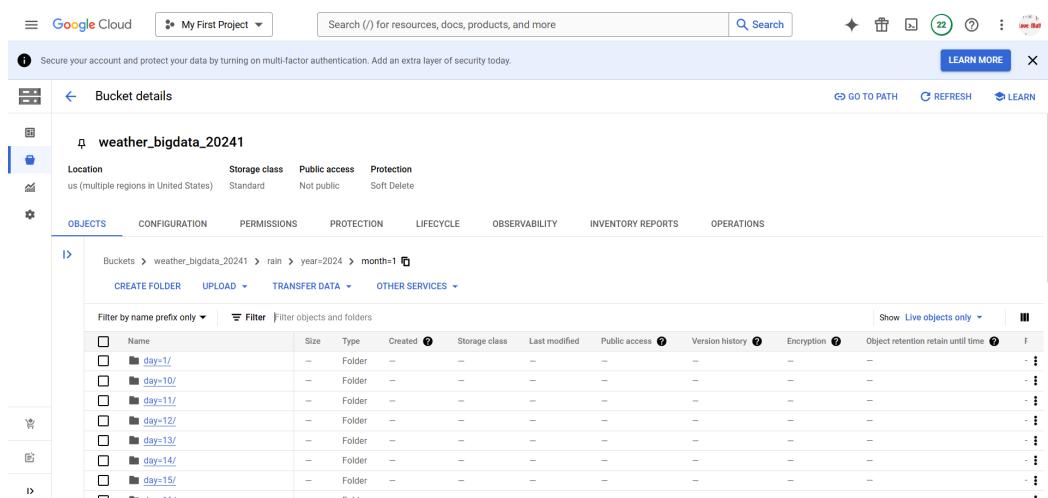


Hình 2.16: Hình ảnh về Cloud Run

Như vậy nhóm đã xây dựng các pipeline để trích xuất dữ liệu từ API về data lake. Các pipeline được triển khai, có thể chạy tự động theo lịch và có khả năng chịu lỗi.



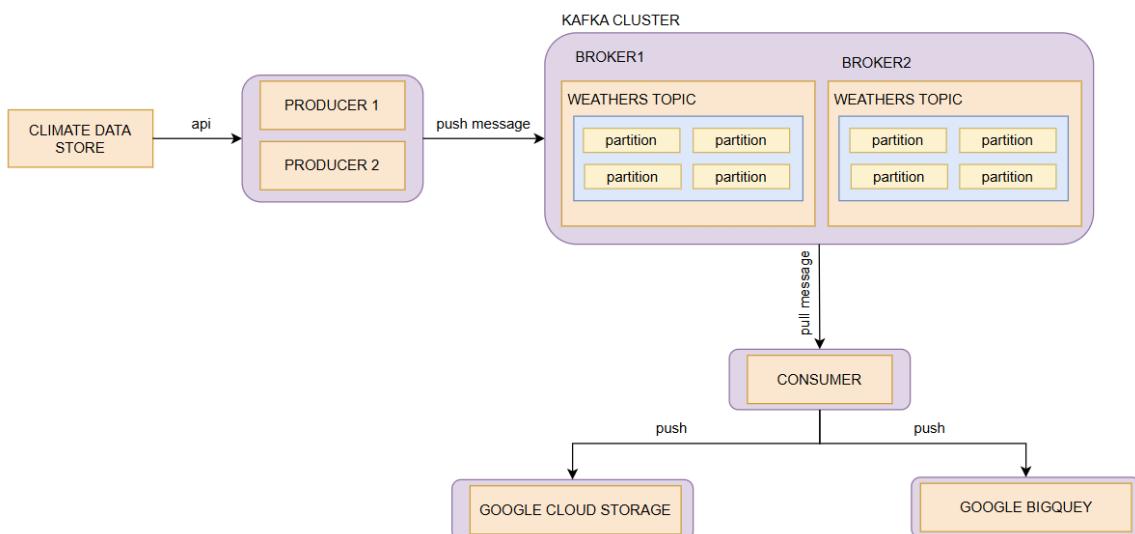
Hình 2.17: Kết quả dữ liệu về thời tiết sau khi đưa lên data lake.



Hình 2.18: Kết quả dữ liệu về lượng mưa sau khi đưa lên data lake.

2.3.2 Streaming dữ liệu với kafka

Để giả lập streaming dữ liệu, nhóm tiến hành thu thập dữ liệu từ trang web Climate Data Store, dữ liệu được lấy thông qua một API và gửi đến hai Producers (Producer 1 và Producer 2). Các Producers này tiếp tục đẩy dữ liệu vào Kafka Cluster, nơi chứa hai Brokers (Broker1 và Broker2). Hai Brokers này đóng vai trò là trung gian để tiếp nhận, lưu trữ tạm thời và quản lý dữ liệu trong Weathers Topic, được chia thành bốn Partitions để tối ưu hóa việc phân phối và xử lý song song. Một Consumer được cấu hình để lấy dữ liệu từ các Partitions này, xử lý và chuẩn hóa dữ liệu. Sau đó, dữ liệu đã xử lý được đẩy đến hai hệ thống đích: Google Cloud Storage và Google BigQuery, phục vụ cho các quá trình xử lý dữ liệu sau này. Sơ đồ minh họa được biểu diễn như hình vẽ dưới đây:



Hình 2.19: Streaming Flow

a) Cấu trúc dữ liệu:

Trước hết nhóm định nghĩa các trường thông tin sẽ được xử lý streaming. Nhóm sử dụng PySpark, một thư viện của Apache Spark dành cho Python, với module `pyspark.sql.types`, nhóm định nghĩa schema cho dữ liệu, sử dụng `T.StructType()` để xác định cấu trúc tổng thể của bảng dữ liệu, còn `T.StructField()` để xác định từng trường dữ liệu, bao gồm tên trường, kiểu dữ liệu (như `T.TimestampType()`, `T.FloatType()`, `T.StringType()`, v.v.) và tùy chọn có cho phép giá trị null hay không. Schema này giúp PySpark hiểu và xử lý dữ liệu theo đúng cấu trúc đã định. Cấu trúc này được định nghĩa trong file `schema.py` với các trường cụ thể như sau:

Cột	Kiểu Dữ liệu	Mô tả
time	Timestamp	Thời gian ghi nhận dữ liệu thời tiết.
latitude	Float	Vĩ độ, chỉ tọa độ địa lý (hướng Bắc - Nam).
longitude	Float	Kinh độ, chỉ tọa độ địa lý (hướng Đông - Tây).
number	Integer	Một chỉ số hoặc mã định danh cho bản ghi dữ liệu.
step	String	Bước thời gian hoặc thông tin bổ sung liên quan đến thời điểm dự báo.
surface	Float	Dữ liệu liên quan đến bề mặt (có thể là độ cao hoặc thông số bề mặt khác).
valid_time	Timestamp	Thời gian hợp lệ của dữ liệu dự báo hoặc đo đạc.
u10	Float	Tốc độ gió thành phần <i>u</i> tại độ cao 10m (hướng Đông-Tây).
v10	Float	Tốc độ gió thành phần <i>v</i> tại độ cao 10m (hướng Bắc-Nam).
d2m	Float	Nhiệt độ điểm sương tại độ cao 2m.
t2m	Float	Nhiệt độ không khí tại độ cao 2m.
msl	Float	Áp suất khí quyển tại mực nước biển (mean sea level).
sst	Float (nullable)	Nhiệt độ bề mặt biển (<i>nullable</i> , có thể có giá trị null).
sp	Float	Áp suất khí quyển tại bề mặt đất.
tcc	Float	Tổng lượng mây che phủ (từ 0 đến 1).
tciw	Float	Hàm lượng nước băng trong mây (ice water content).
tclw	Float	Hàm lượng nước lỏng trong mây (liquid water content).

Bảng 2.2: Cấu trúc dữ liệu thu thập từ API thời tiết.

b) Thu thập dữ liệu từ trang web:

Để thu thập dữ liệu cho mục đích streaming, nhóm thiết lập hàm `fetch_weather_data` sử dụng thư viện `cdsapi` để tải dữ liệu khí tượng từ Copernicus Climate Data Store(CDS) dựa trên một thời gian thực cụ thể, được truyền vào dưới dạng đối tượng `datetime`. Do dữ liệu được cập nhật với độ trễ 8 ngày, nhóm tiến hành thu thập các dữ liệu có trước đó 10 ngày. Từ `datetime`, thông tin thời gian (năm, tháng, ngày, giờ) được trích xuất và sử dụng để tạo yêu cầu tải dữ liệu. Dữ liệu được tải về dưới định dạng GRIB và lưu tạm thời trong một thư mục dựa trên thời gian. Sau đó, file GRIB được mở và chuyển đổi thành `DataFrame` bằng thư viện `xarray`, rồi được xử lý để tạo danh sách các bản ghi, mỗi bản ghi gồm mã định danh (thời gian, vĩ độ, kinh độ) và dữ liệu liên quan. Cuối cùng, thư mục lưu trữ tạm thời được xóa để giải phóng không gian, và

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

danh sách các bản ghi được trả về để sử dụng tiếp.

c) Tạo Luồng Dữ Liệu với Kafka Producer:

Trước hết, nhóm triển khai một lớp WeatherCSVProducer để xử lý việc lấy và gửi dữ liệu thời tiết lên một Kafka topic. Trong đó, phương thức read_records nhận một đối tượng datetime làm tham số, gọi hàm fetch_weather_data để thu thập dữ liệu thời tiết, sau đó trả về danh sách các bản ghi (dạng cặp key-value). Sau đó, phương thức publish gửi từng bản ghi từ danh sách dữ liệu tới một Kafka topic chỉ định, với mỗi bản ghi được chuyển đổi thành byte (key và value). Nếu việc gửi thành công, thông tin về topic, partition, và offset sẽ được in ra.Thêm vào đó, có khoảng nghỉ ngắn (sleep_time) giữa các lần gửi để kiểm soát tốc độ. Lớp này kết hợp Kafka Producer và hàm thu thập dữ liệu để tạo một luồng dữ liệu từ nguồn đến Kafka topic. Chi tiết trong file producer.py.

Tiếp theo, để thiết lập 2 producer cho việc thu thập dữ liệu, nhóm sử dụng hàm producer_instance tạo producer để thu thập và gửi dữ liệu thời tiết đến một Kafka topic với các tùy chọn cấu hình linh hoạt. Ban đầu, các biến môi trường như tên topic, địa chỉ Kafka broker (kafka_address), và cổng (broker) được tải từ file .env hoặc được truyền dưới dạng tham số. Một Kafka Admin Client được khởi tạo để đảm bảo topic có đủ số phân vùng bằng cách tạo thêm phân vùng nếu cần. Sau đó, một Kafka producer (WeatherCSVProducer) được cấu hình với các thiết lập như độ tin cậy (acks='all') và ID của broker. Dữ liệu thời tiết tại thời điểm hiện tại (datetime.now()) được thu thập thông qua phương thức read_records và gửi đến Kafka topic với thời gian nghỉ giữa các lần gửi được tùy chỉnh bằng tham số dòng lệnh (-time). Cuối cùng, thời gian xử lý toàn bộ quá trình được đo lường và in ra để theo dõi hiệu suất.

d) Xử lý dữ liệu từ Consumer vào Google Cloud Storage (GCS) và Google BigQuery:

Về cơ bản, nhóm sử dụng PySpark Structured Streaming để kết nối và xử lý dữ liệu từ Kafka, sau đó ghi kết quả vào Google Cloud Storage (GCS) và Google BigQuery. Cụ thể, dữ liệu đầu vào được lấy từ một WEATHERS TOPIC có chứa các thông tin về thời tiết, bao gồm các biến như tốc độ gió, nhiệt độ, độ ẩm và các thông số khí tượng khác.

Trước hết, PySpark đọc dữ liệu từ Kafka bằng cách sử dụng phương thức readStream để tạo một DataFrame stream, nơi các bản tin Kafka được xử lý theo từng batch. Cấu hình của Kafka bao gồm địa chỉ bootstrap servers, tên topic, và chế độ bắt đầu đọc. Dữ liệu từ Kafka sẽ được chuyển đổi thành các cột tương ứng với kiểu dữ liệu được định nghĩa trong WEATHER_SCHEMA, bao gồm các trường như time, latitude, longitude, tcc (cloud cover), msl (sea level pressure), và các biến khí tượng khác. Các giá trị của trường value trong Kafka message sẽ được tách thành

các phần tử và ánh xạ vào các cột phù hợp.

Để minh họa tính toán 1 query, nhóm tiến hành tính toán tốc độ gió từ hai thành phần của gió u_{10} và v_{10} (tốc độ gió theo phương bắc-nam và phương đông-tây). Công thức tính tốc độ gió là: $\text{wind_speed} = \sqrt{u_{10}^2 + v_{10}^2}$, được thực hiện bằng các phép toán vector trong PySpark thông qua các hàm F.pow và F.sqrt.

Sau khi dữ liệu được xử lý và tính toán các thông số cần thiết, hệ thống ghi kết quả vào hai đích khác nhau: GCS và BigQuery. Đầu tiên, dữ liệu sẽ được ghi vào GCS dưới dạng file Parquet để lưu trữ lâu dài và có thể được truy vấn sau này. Để đảm bảo tính nhất quán và không mất mát dữ liệu trong quá trình ghi, Spark sử dụng cơ chế checkpointing. Các file Parquet sẽ được ghi vào thư mục đã định nghĩa trong GCS_STORAGE_PATH. Đồng thời, dữ liệu cũng được ghi vào BigQuery để phục vụ cho việc truy vấn và phân tích dữ liệu theo thời gian thực. Tính năng checkpoint trong BigQuery cũng giúp hệ thống duy trì trạng thái và tiếp tục xử lý dữ liệu từ điểm cuối cùng khi gặp sự cố hoặc tạm dừng.

Toàn bộ quá trình được cấu hình để thực thi liên tục với các trigger và thời gian xử lý được thiết lập là mỗi 10 giây, đảm bảo dữ liệu được cập nhật kịp thời và không bị gián đoạn. Mỗi lần dữ liệu được xử lý, các batch dữ liệu mới từ Kafka sẽ được kiểm tra và tính toán, với việc ghi vào GCS và BigQuery thực hiện qua các stream writer. Cuối cùng, chương trình sẽ chờ đợi và tiếp tục lặp lại cho đến khi dừng hoặc có lỗi xảy ra.

e) Kết quả thực thi chương trình:

Quá trình đọc dữ liệu từ hai producer

```

Produced record to topic weathers partition [1] @ offset 50 from producer2
Produced record to topic weathers partition [2] @ offset 52 from producer1
Produced record to topic weathers partition [2] @ offset 53 from producer2
Produced record to topic weathers partition [1] @ offset 51 from producer1
Produced record to topic weathers partition [0] @ offset 61 from producer2
Produced record to topic weathers partition [3] @ offset 62 from producer1
Produced record to topic weathers partition [2] @ offset 54 from producer2
Produced record to topic weathers partition [1] @ offset 52 from producer1
Produced record to topic weathers partition [2] @ offset 55 from producer2
Produced record to topic weathers partition [2] @ offset 56 from producer1
Produced record to topic weathers partition [0] @ offset 62 from producer2
Produced record to topic weathers partition [2] @ offset 57 from producer1
Produced record to topic weathers partition [1] @ offset 53 from producer2
Produced record to topic weathers partition [2] @ offset 58 from producer1
Produced record to topic weathers partition [2] @ offset 59 from producer2
Produced record to topic weathers partition [0] @ offset 63 from producer1
Produced record to topic weathers partition [2] @ offset 60 from producer2
Produced record to topic weathers partition [0] @ offset 64 from producer1
Produced record to topic weathers partition [3] @ offset 63 from producer2
Produced record to topic weathers partition [3] @ offset 64 from producer1
Produced record to topic weathers partition [1] @ offset 54 from producer2
Produced record to topic weathers partition [0] @ offset 65 from producer1
Produced record to topic weathers partition [1] @ offset 55 from producer2
Produced record to topic weathers partition [1] @ offset 56 from producer1
Produced record to topic weathers partition [3] @ offset 65 from producer2
Produced record to topic weathers partition [3] @ offset 66 from producer1
Produced record to topic weathers partition [3] @ offset 67 from producer2
Produced record to topic weathers partition [0] @ offset 66 from producer1
Produced record to topic weathers partition [3] @ offset 68 from producer2

```

Hình 2.20: Quá trình đọc dữ liệu của 2 producer.

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

Hình 2.21: Messages từ Kafka Cluster.

- Dữ liệu sau khi được đưa vào Google Cloud Storage(GCS) và Google Bigquery:

Hình 2.22: Dữ liệu được lưu trữ vào GCS.

Hình 2.23: Dữ liệu được lưu trữ vào Google Bigquery.

Hình 2.24: Dữ liệu được lưu trữ vào Google Bigquery.

2.3.3 Xử lý dữ liệu với spark

Đầu tiên, nhóm sử dụng Spark để tiền xử lý và làm sạch dữ liệu với các thao tác như: loại bỏ đi những giá trị rỗng, chuyển đổi các cột thông tin thời gian, chuyển đổi nhiệt độ từ Kelvin sang độ C. Code được viết sử dụng PySpark để thực hiện các tính toán như: tốc độ gió, áp suất hơi nước, mật độ không khí, hướng gió...

Công thức tính mật độ không khí:

$$air_density = \frac{sp}{R * (t2m + 273.15)} \quad (2.1)$$

trong đó sp là áp suất không khí, R là hằng số khí, $t2m$ là nhiệt độ tính bằng độ C.

Công thức tính áp suất hơi nước:

$$vapor_pressure = 6.11 \times 10^{\frac{7.5 \times d2m}{d2m + 237.3}} \quad (2.2)$$

Công thức tính hướng gió:

$$wind_direction = \begin{cases} 360 + \left(180 + \frac{180}{\pi} \arctan\left(\frac{u_{10}}{v_{10}}\right)\right), & \text{if } 180 + \frac{180}{\pi} \arctan\left(\frac{u_{10}}{v_{10}}\right) < 0 \\ 180 + \frac{180}{\pi} \arctan\left(\frac{u_{10}}{v_{10}}\right), & \text{otherwise} \end{cases} \quad (2.3)$$

Công thức tính tốc độ gió:

$$wind_speed = \sqrt{u10^2 + v10^2} \quad (2.4)$$

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

Công thức tính độ ẩm tương đối:

$$\text{humidity} = \left(\frac{\text{vapor_pressure}}{\text{saturation_vapor_pressure}} \right) \cdot 100 \quad (2.5)$$

Công thức tính nhiệt độ thực tế cảm nhận:

$$\text{apparent_temperature} = t2m + 0.33 \cdot \text{vapor_pressure} - 0.70 \cdot \text{wind_speed} - 4.00 \quad (2.6)$$

Kết quả:

Đối với việc gộp dữ liệu khi sử dụng 2 node workers, tổng thời gian thực hiện là 31 phút.

The screenshot shows the Spark Master UI at the URL <http://34.38.208.106:8080>. The page displays the following information:

- Spark Master at spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077**
- URL:** spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077
- Alive Workers:** 2
- Cores in use:** 4 Total, 0 Used
- Memory in use:** 8.0 GiB Total, 0.0 B Used
- Resources in use:**
- Applications:** 0 Running, 1 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241209175411-10.132.0.4-33971	10.132.0.4:33971	ALIVE	2 (0 Used)	4.0 GiB (0.0 B Used)	
worker-20241209175414-10.132.0.4-46737	10.132.0.4:46737	ALIVE	2 (0 Used)	4.0 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209175826-0000	merge-data	4	1024.0 MiB		2024/12/09 17:58:26	phuc_0703	FINISHED	31 min

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209175826-0000	merge-data	4	1024.0 MiB		2024/12/09 17:58:26	phuc_0703	FINISHED	31 min

Hình 2.25: Khi spark job thực hiện gộp dữ liệu với 2 node workers được thực thi xong.

Ngoài ra, trong quá trình thực hiện gộp dữ liệu với 2 node workers, nhóm đã thử kill 1 worker (DEAD) và chỉ để lại 1 worker hoạt động (ALIVE)

The screenshot shows the Spark Master UI at the URL <http://34.38.208.106:8080>. The page displays the following information:

- Spark Master at spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077**
- URL:** spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077
- Alive Workers:** 1
- Cores in use:** 2 Total, 0 Used
- Memory in use:** 4.0 GiB Total, 0.0 B Used
- Resources in use:**
- Applications:** 0 Running, 1 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (2)

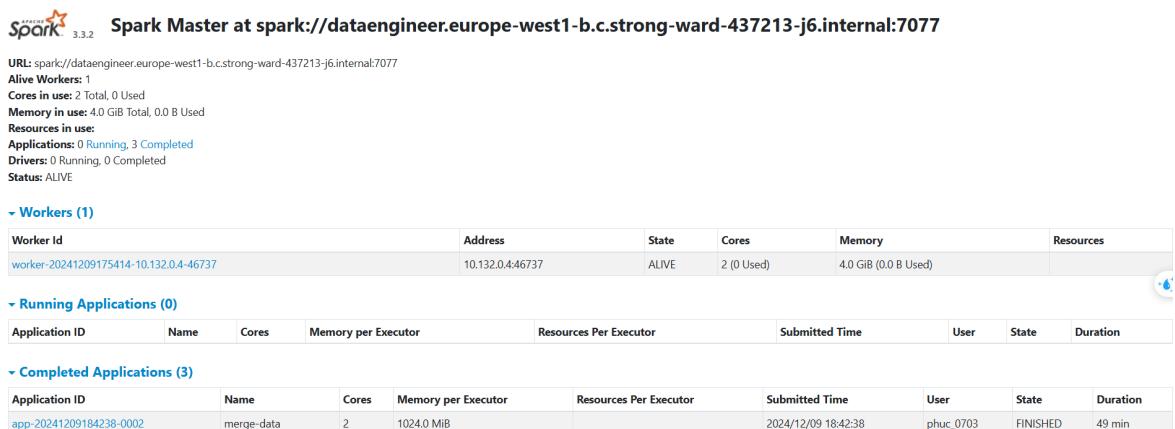
Worker Id	Address	State	Cores	Memory	Resources
worker-20241209175411-10.132.0.4-33971	10.132.0.4:33971	DEAD	2 (2 Used)	4.0 GiB (1024.0 MiB Used)	
worker-20241209175414-10.132.0.4-46737	10.132.0.4:46737	ALIVE	2 (2 Used)	4.0 GiB (1024.0 MiB Used)	

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209184238-0002	(kill) merge-data	2	1024.0 MiB		2024/12/09 18:42:38	phuc_0703	RUNNING	9.2 min

Hình 2.26: Khi spark job thực hiện gộp dữ liệu với 2 node workers thì 1 worker bị kill.

Kết quả 1 worker còn lại tiếp tục thực hiện công việc với thời gian là 49 phút, hiển nhiên lâu hơn khi 2 workers hoạt động bình thường.



The screenshot shows the Spark Master interface at `spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077`. It displays the following information:

- Alive Workers:** 1
- Memory in use:** 4.0 GiB Total, 0.0 B Used
- Resources in use:**
- Applications:** 0 Running, 3 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241209175414-10.132.0.4-46737	10.132.0.4:46737	ALIVE	2 (0 Used)	4.0 GiB (0.0 B Used)	

Running Applications (0)

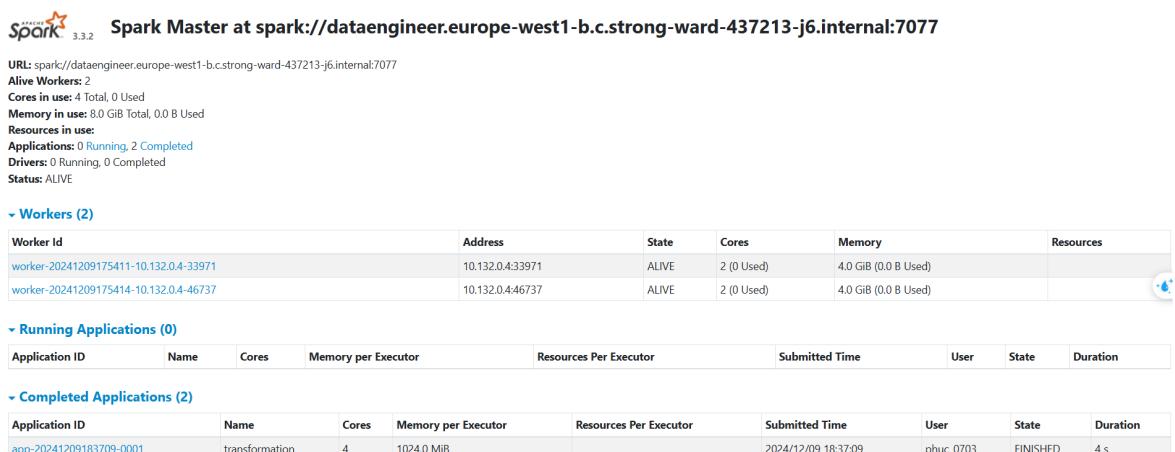
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209184238-0002	merge-data	2	1024.0 MiB		2024/12/09 18:42:38	phuc_0703	FINISHED	49 min

Completed Applications (3)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209184238-0002	merge-data	2	1024.0 MiB		2024/12/09 18:42:38	phuc_0703	FINISHED	49 min

Hình 2.27: Khi spark job thực hiện gộp dữ liệu với 1 worker còn lại.

Đối với việc thực hiện các biến đổi tính toán, tổng thời gian thực hiện là 4 giây.



The screenshot shows the Spark Master interface at `spark://dataengineer.europe-west1-b.c.strong-ward-437213-j6.internal:7077`. It displays the following information:

- Alive Workers:** 2
- Memory in use:** 8.0 GiB Total, 0.0 B Used
- Resources in use:**
- Applications:** 0 Running, 2 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241209175411-10.132.0.4-33971	10.132.0.4:33971	ALIVE	2 (0 Used)	4.0 GiB (0.0 B Used)	
worker-20241209175414-10.132.0.4-46737	10.132.0.4:46737	ALIVE	2 (0 Used)	4.0 GiB (0.0 B Used)	

Running Applications (0)

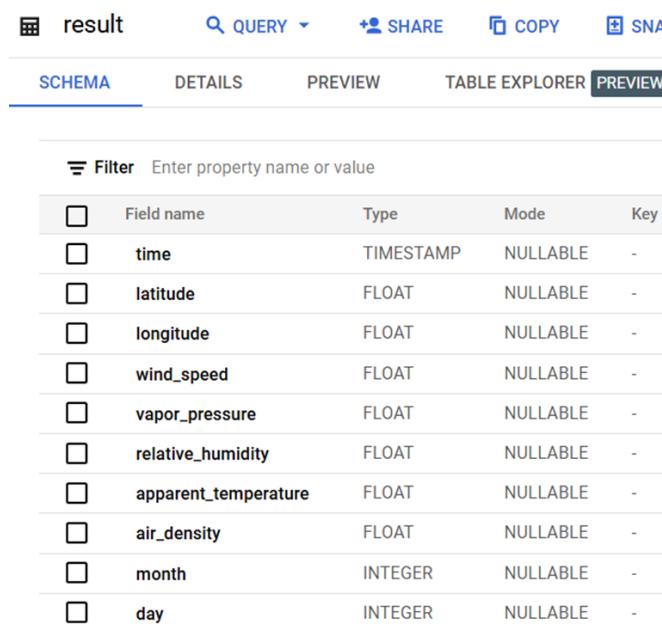
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209183709-0001	transformation	4	1024.0 MiB		2024/12/09 18:37:09	phuc_0703	FINISHED	4 s

Completed Applications (2)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209183709-0001	transformation	4	1024.0 MiB		2024/12/09 18:37:09	phuc_0703	FINISHED	4 s

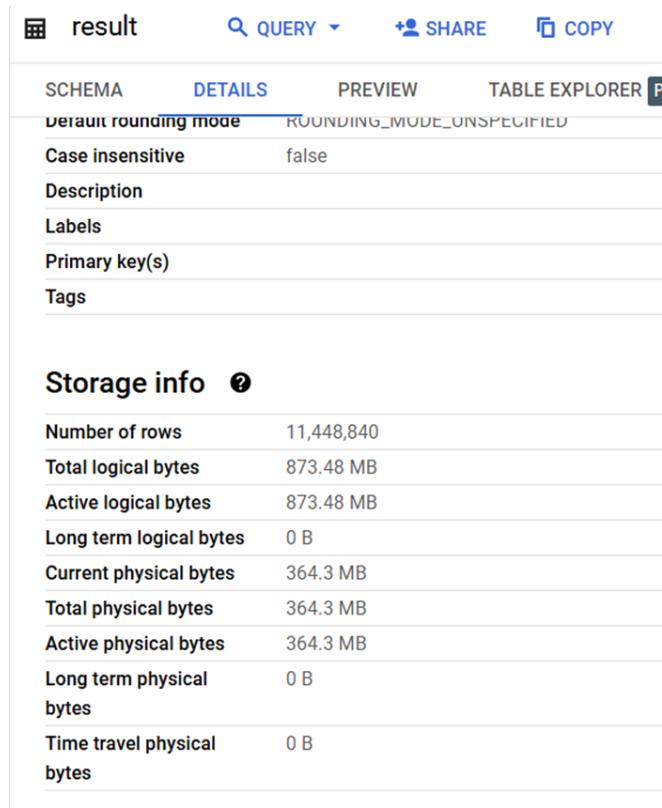
Hình 2.28: Khi spark job thực hiện biến đổi tính toán với 2 node workers được thực thi xong.

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ



Field name	Type	Mode	Key
time	TIMESTAMP	NULLABLE	-
latitude	FLOAT	NULLABLE	-
longitude	FLOAT	NULLABLE	-
wind_speed	FLOAT	NULLABLE	-
vapor_pressure	FLOAT	NULLABLE	-
relative_humidity	FLOAT	NULLABLE	-
apparent_temperature	FLOAT	NULLABLE	-
air_density	FLOAT	NULLABLE	-
month	INTEGER	NULLABLE	-
day	INTEGER	NULLABLE	-

Hình 2.29: Các trường dữ liệu về thời tiết được tính toán.



Number of rows	11,448,840
Total logical bytes	873.48 MB
Active logical bytes	873.48 MB
Long term logical bytes	0 B
Current physical bytes	364.3 MB
Total physical bytes	364.3 MB
Active physical bytes	364.3 MB
Long term physical bytes	0 B
Time travel physical bytes	0 B

Hình 2.30: Thông tin dữ liệu sau khi thực hiện tính toán.

Tiếp theo, nhóm thực hiện xử lý dữ liệu cho thuật toán phát hiện và tracking bão. Thuật toán sử dụng 3 trường chính của dữ liệu là: u10 (thành phần gió theo hướng Đông – Tây tại độ cao 10 mét), v10 (thành phần gió theo hướng Bắc – Nam tại độ cao 10 mét), msl (áp suất trung bình tại mực nước biển). Trong giai đoạn phát hiện, các điểm cực trị (cực tiểu cho bão và cực đại cho áp cao) được xác định từ các nhóm điểm lân cận đủ lớn (ít nhất 9 điểm) và ghi nhận vị trí trung tâm của chúng. Giai đoạn theo dõi liên kết các vị trí cực trị qua các bước thời gian, sử dụng khoảng cách tìm kiêm tối đa dựa trên tốc độ di chuyển (80 km/h) và loại bỏ các đường đi ngắn hơn 12 giờ.

Thuật toán sử dụng Spark để lọc các trường dữ liệu nói trên theo giờ sắp xếp theo chiều tăng dần của thời gian và kinh độ, giảm dần của vĩ độ. Sau đó sử dụng numpy để tính toán các chỉ số của cơn bão như: ví trí tâm bão, tốc độ gió tối đa, diện tích ảnh hưởng, chênh lệch áp suất vùng gần tâm bão và ma trận khu vực ảnh hưởng. Các thông số này được lưu vào Bigquery phục vụ cho việc truy xuất trực quan hóa dữ liệu (ma trận khu vực ảnh hưởng được duỗi thành vector 1 chiều cùng với kích thước ban đầu được lưu dưới dạng chuỗi).

Kết quả:

Khi sử dụng 2 node worker, tổng thời gian thực hiện là 13.8 phút:

Spark Master at spark://dataengineer.europe-west1-b.c strong-ward-437213-j6.internal:7077								
URL: spark://dataengineer.europe-west1-b.c strong-ward-437213-j6.internal:7077								
All Workers	0	0	0	0	0	0		
Cores in use:	0	0	0	0	0	0		
Memory in use:	0.0 GB Total, 0.0 B Used							
Resources in use:								
Applications in use:								
Drivers in use:								
Workers in use:								
Workers (2)								
Worker ID	Name	Cores	Memory	Resources	User	State		
worker-20241209150000-181120-4-42319	10.132.0.4:43519	2 (0 Used)	4.0 GB (0.0 B Used)					
worker-20241209150000-181120-4-10055	10.132.0.4:43595	2 (0 Used)	4.0 GB (0.0 B Used)					
+ Running Applications (0)	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User		
+ Completed Applications (1)	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209150000	storm_d69	2	1024.0 MB		2024-12-09 10:09:14	pduc_0703	FINISHED	13.8 ms

Hình 2.31: Tính toán dữ liệu bão với 2 worker

Khi kill 1 worker và chạy job với worker còn lại, thời gian là 16 phút:

Spark Master at spark://dataengineer.europe-west1-b.c strong-ward-437213-j6.internal:7077								
URL: spark://dataengineer.europe-west1-b.c strong-ward-437213-j6.internal:7077								
All Workers	0	0	0	0	0	0		
Cores in use:	0	0	0	0	0	0		
Memory in use:	0.0 GB Total, 0.0 B Used							
Resources in use:								
Applications in use:								
Drivers in use:								
Workers in use:								
Workers (1)								
Worker ID	Name	Cores	Memory	Resources	User	State		
worker-2024120915414-10.132.0.4-40737	10.132.0.4:440737	4 (0 Used)	4.0 GB (0.0 B Used)					
+ Running Applications (0)	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
+ Completed Applications (3)	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241209154238-0002	storm_d69	2	1024.0 MB		2024-12-09 15:42:38	pduc_0703	FINISHED	16 min

Hình 2.32: Tính toán dữ liệu bão với 1 worker

CHƯƠNG 2. KIẾN TRÚC VÀ THIẾT KẾ

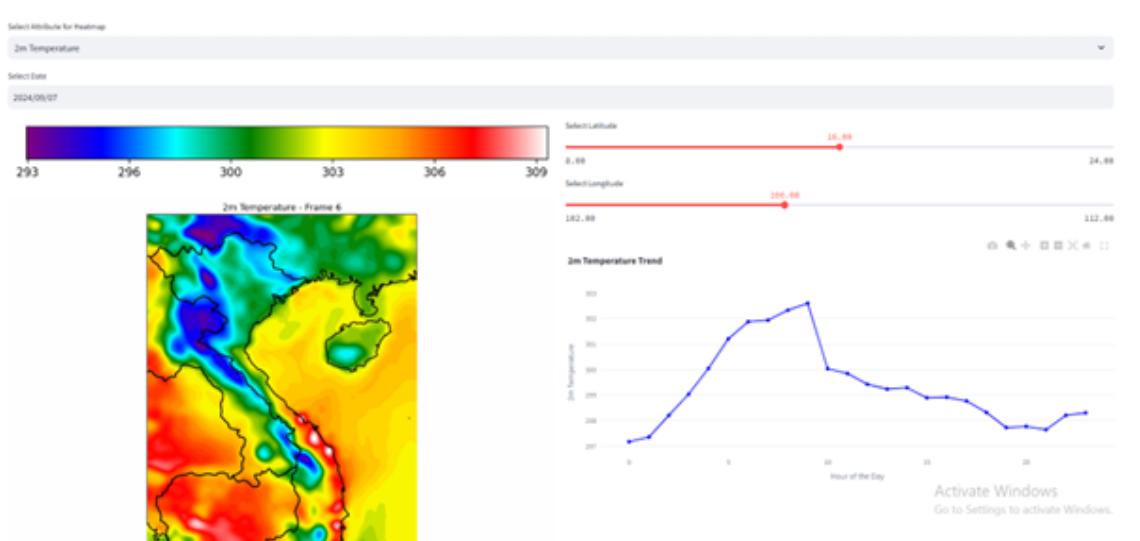
Dữ liệu trên Bigquery:

Hình 2.33: Các trường dữ liệu về bão được tính toán.

Hình 2.34: Thông tin dữ liệu bão sau khi thực hiện tính toán.

2.3.4 Trực quan hóa dữ liệu và kết quả với streamlit

Giao diện chung:



Hình 2.35: Giao diện chung streamlit.

**Hình 2.36:** Giao diện detect bão.

Giao diện bao gồm 3 phần chính:

- Hiển thị ảnh dưới dạng heatmap mô tả điều kiện thời tiết theo ngày
- Biểu đồ cho ta biết biến động của thời tiết theo các khung giờ trong ngày
- Hiển thị thông tin của các cơn bão nhiệt đới trong ngày

Chi tiết quá trình xử lý dữ liệu:

Bước 1: Sau khi chọn ngày, các dữ liệu thời tiết trong ngày sẽ được lấy về từ bigquery.

Bước 2: Dữ liệu sau khi được lấy về sẽ được tách thành các trường riêng biệt theo các yếu tố thời tiết. Đồng thời hệ thống sẽ tiến hành truy vấn để kiểm tra về thông tin bão trong ngày.

Bước 3: Các dữ liệu thời tiết sẽ được tái cấu trúc lại tạo thành các mảng dữ liệu 3 chiều (ứng với thời gian, vĩ độ, tung độ)

Bước 4: Các dữ liệu về yếu tố thời tiết sẽ được vẽ thành ảnh heatmap như phần 1, đồng thời các dữ liệu biến động sẽ được biểu diễn thành các đồ thị 2, các dữ liệu về bão sẽ được hiển thị như phần 3.

CHƯƠNG 3. CHI TIẾT TRIỂN KHAI

3.1 Source code và documentation.

Repository của đề tài được lưu trữ tại GitHub Repository: **analyze_weather_data**. Repository này chứa toàn bộ mã nguồn và tài liệu hướng dẫn để triển khai hệ thống. Mã nguồn sử dụng các công nghệ chính như Mage để xây dựng pipelines dữ liệu, Terraform để quản lý cơ sở hạ tầng, Docker để đóng gói và triển khai các dịch vụ, Apache Spark để xử lý batch, Kafka để giả lập truyền tải và xử lý luồng dữ liệu thời gian thực, và Streamlit để xây dựng giao diện trực quan hóa dữ liệu. Mã nguồn được trình bày riêng thành từng thư mục nhỏ tương ứng với các công nghệ tương ứng được sử dụng trong project. Ngoài ra, project được mô tả tổng quan trọng file README.md, hướng dẫn để triển khai hệ thống được mô tả trong reproduce.md.

3.2 Configuration files theo môi trường

Trong bài tập lớn này, Terraform được sử dụng để triển khai cơ sở hạ tầng trên Google Cloud Platform (GCP). Terraform là công cụ Infrastructure as Code (IaC) mạnh mẽ, cho phép tự động hóa việc tạo và quản lý tài nguyên trên GCP như Kubernetes Cluster (GKE), Virtual Machines (VM), Storage Buckets, và nhiều dịch vụ khác. Việc sử dụng Terraform giúp tăng tính nhất quán, giảm thiểu lỗi trong việc thiết lập môi trường, đồng thời hỗ trợ quản lý các cấu hình phức tạp một cách hiệu quả.

Thư mục `terraform` được nhóm tổ chức thành 2 file nhỏ gồm `main.tf` và `variables.tf` trong đó:

- **main.tf:** Đây là file chính, nơi định nghĩa các tài nguyên GCP cần triển khai. Trong file `main.tf`, nhóm cấu hình phiên bản Terraform yêu cầu và thiết lập provider Google, sử dụng thông tin dự án, khu vực và credentials. Tiếp theo, tài nguyên được khai báo, bao gồm một Google Cloud Storage Bucket để lưu trữ dữ liệu, với các tính năng như phiên bản hóa và quy tắc vòng đời tự động xóa dữ liệu sau 30 ngày. Ngoài ra, một BigQuery Dataset được tạo để phân tích dữ liệu lớn. Cấu hình cũng bao gồm việc thiết lập các firewall rules để mở cổng kết nối cho Kafka Broker, đảm bảo khả năng giao tiếp giữa các thành phần trong hệ thống. Hai máy ảo (VMs) được triển khai, một cho Kafka và một cho Data Engineer, để phục vụ các tác vụ xử lý dữ liệu.
- **variables.tf:** Khai báo các biến sử dụng trong Terraform để dễ dàng thay đổi theo môi trường. File này định nghĩa các biến quan trọng và giá trị mặc định cần thiết để triển khai tài nguyên trên Google Cloud Platform (GCP). Các biến bao gồm `project_id` (ID dự án GCP), `region` và `zone` (khu vực và zone của GCP), cùng với `storage_class` (lớp

lưu trữ cho Google Cloud Storage), BQ_DATASET (BigQuery Dataset), và credentials (đường dẫn tệp chứng chỉ GCP). Network xác định mạng cho các tài nguyên và vm_image định nghĩa hệ điều hành cho máy ảo. Quy trình triển khai terraform được trình bày trong file **reproduce.md**.

3.3 Deployment strategy

Chuẩn bị môi trường: Cần thiết lập các công cụ cần thiết bao gồm Terraform, Docker, công cụ quản lý phiên bản GitHub. Đảm bảo rằng các tài nguyên như Google Cloud Storage (GCS), BigQuery, Virtual Machine phải có quyền truy cập. Chi tiết về GCP và Terraform xem tại: [Link](#).

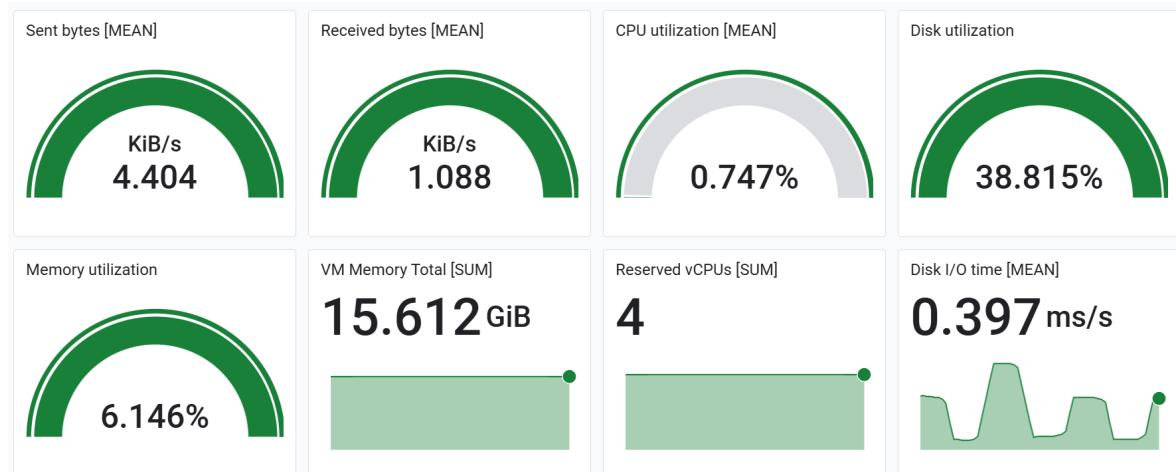
Triển khai hạ tầng (Infrastructure Deployment): Sử dụng Terraform đã được thiết lập trước đó để triển khai hạ tầng theo hướng dẫn tại [Link](#).

Triển khai các dịch vụ: Được đóng gói và trình bày trong thư mục reproduce. Có thể triển khai theo hướng dẫn tại: [Link](#).

3.4 Monitoring setup

Cấu trúc của Monitoring gồm các phần chính như sau:

- **Quick CPU/Mem/Disk:** có thể xem được phần trăm lượng CPU, Memory và Disk mà VM Instance sử dụng. Ngoài ra, còn xem được cả tổng core CPU, Total RAM



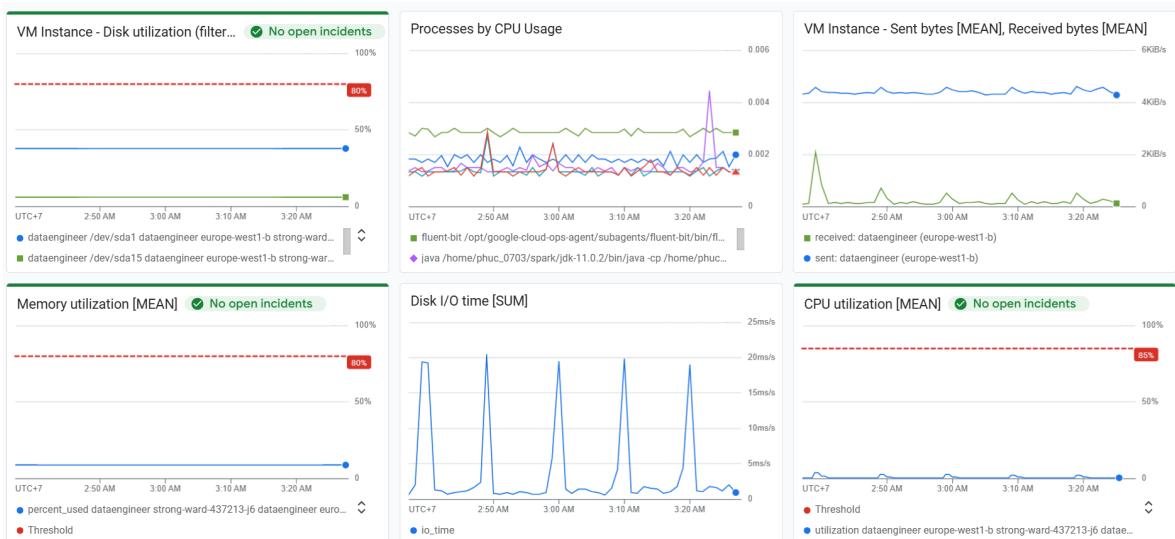
Hình 3.1: Quick CPU/Mem/Disk in Monitoring

- **CPU/Mem/Net/Disk:** sẽ hiển thị chi tiết thông tin về các thành phần quan trọng của hệ thống, bao gồm CPU, Memory, Network, và Disk, thông qua các biểu đồ Linechart. Mỗi biểu đồ sẽ thể hiện các chỉ số tài nguyên theo thời gian, đồng thời hiển thị ngưỡng

CHƯƠNG 3. CHI TIẾT TRIỂN KHAI

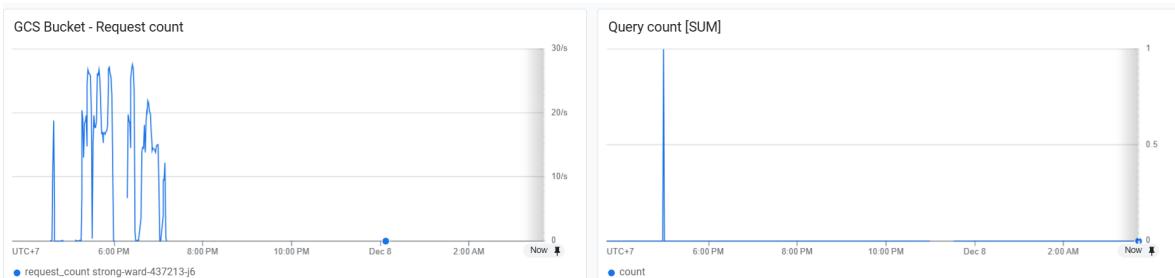
(threshold) để giúp người dùng nhận diện kịp thời khi các chỉ số này vượt quá mức cho phép.

- **Alerting Setup:** Các cảnh báo sẽ được thiết lập dựa trên ngưỡng (threshold) đã định sẵn cho từng chỉ số quan trọng. Cảnh báo sẽ được kích hoạt khi một chỉ số vượt quá hoặc thấp hơn ngưỡng cho phép, giúp nhanh chóng nhận diện và xử lý kịp thời các vấn đề. Các cảnh báo cấu hình được gửi thông báo qua gmail.
- **CPU Utilization:** Cảnh báo khi CPU sử dụng trên 85% trong 5 phút liên tiếp.
- **Memory Utilization:** Cảnh báo khi bộ nhớ sử dụng trên 80%.
- **Disk Utilization:** Cảnh báo khi mức sử dụng ổ đĩa vượt quá 80% dung lượng.



Hình 3.2: CPU/Mem/Net/Disk detail in Monitoring

Ngoài ra, nhóm cũng triển khai phần theo dõi các thông số cơ bản của GCS Bucket là Request Count, và Query SUM trong bigquery để admin có thể tiện theo dõi



Hình 3.3: Monitoring sum query in bigquery and request count in GCS Bucket

CHƯƠNG 4. BÀI HỌC KINH NGHIỆM

4.1 Kinh nghiệm về Data Ingestion

4.1.1 Xử lý nhiều nguồn dữ liệu đa dạng

Vấn đề: Các thông số thời tiết trong dữ liệu, như lượng mưa, nhiệt độ, áp suất, và độ che phủ của mây, có cấu trúc và ý nghĩa khác nhau. Việc xử lý tất cả các thông số trong một pipeline gây khó khăn trong quản lý và làm giảm hiệu suất do tính phức tạp của dữ liệu.

Giải pháp:

- Giải pháp đã thử: Sử dụng pipeline chỉ có 1 luồng chung để xử lý toàn bộ các thông số, từ việc tải dữ liệu, biến đổi, và lưu trữ nhưng cách này khiến pipeline trở nên cồng kềnh, khó xử lý lỗi và không thể tối ưu hóa từng bước cho từng loại dữ liệu do có sự khác biệt về thời gian và cách thức đo lường của dữ liệu lượng mưa.
- Giải pháp đã chọn: Phân loại dữ liệu dựa trên bản chất của thông số: Luồng riêng cho lượng mưa, vì đây là thông số có sự khác biệt trong cách lấy thời gian đo lường. Luồng khác riêng cho các thông số thời tiết còn lại như nhiệt độ, áp suất, độ che phủ mây. Sau đó join 2 luồng này lại dựa trên thời gian hợp lệ. Sử dụng Mage để thiết kế các pipeline riêng biệt, mỗi pipeline có nhiệm vụ cụ thể, dễ giám sát và tối ưu hóa. Đảm bảo rằng mỗi pipeline đều có logging và cơ chế phát hiện lỗi riêng.

Bài học rút ra: Việc chia nhỏ các pipeline không chỉ giúp quản lý dữ liệu tốt hơn mà còn nâng cao hiệu suất xử lý. Thiết kế pipeline cần dựa trên đặc điểm dữ liệu thực tế để tránh sự cồng kềnh không cần thiết. Các công cụ như Mage hỗ trợ tối ưu việc phân chia và giám sát pipeline.

4.1.2 Đảm bảo Data Quality

Vấn đề: Dữ liệu thời tiết được tải từ API của Climate Data Store có giới hạn tốc độ (rate limit), tức là chỉ cho phép một số lượng yêu cầu nhất định trong khoảng thời gian cụ thể. Khối lượng dữ liệu lớn (15GB, gần 80 triệu bản ghi) khiến việc tải dữ liệu toàn bộ một lần trở nên không khả thi, dễ xảy ra lỗi timeout và bị API từ chối do vượt quá giới hạn.

Giải pháp:

- Giải pháp đã thử: Tải toàn bộ dữ liệu trong một lần bằng cách thiết lập khoảng thời gian 2022-2024. Tuy nhiên, cách này dẫn đến lỗi khi API giới hạn số lượng bản ghi trả về cho mỗi lần gọi. Hơn nữa, nếu xảy ra lỗi giữa chừng, toàn bộ quá trình tải phải khởi động lại,

gây lãng phí tài nguyên.

- Giải pháp đã chọn: Chia dữ liệu thành từng tháng, và lập lịch tải dữ liệu theo từng ngày. Sử dụng Mage để xây dựng pipeline tự động hóa việc tải dữ liệu, với các cell đảm nhận việc gọi API, kiểm tra lỗi và lưu dữ liệu vào Google Cloud Storage. Thiết lập cơ chế retry sau 120s và 3 lần retry trong trường hợp xảy ra lỗi mạng hoặc lỗi API.

Bài học rút ra: Quản lý dữ liệu lớn yêu cầu chia nhỏ khối lượng công việc để phù hợp với giới hạn của hệ thống. Sử dụng các công cụ như Mage giúp tự động hóa và xử lý lỗi hiệu quả hơn, giảm rủi ro do can thiệp thủ công. Lập lịch pipeline định kỳ là cách hiệu quả để duy trì luồng dữ liệu ổn định trong thời gian dài.

4.1.3 Xử lý Late Arriving Data

Vấn đề: Trong quá trình vận hành pipeline, dữ liệu có thể không đến đúng thời điểm, dẫn đến sự thiếu sót trong các bản ghi dữ liệu. Với dữ liệu thời tiết, việc thiếu dữ liệu của một ngày hoặc một khoảng thời gian nhất định có thể ảnh hưởng đến kết quả phân tích.

Giải pháp:

- Giải pháp đã thử: Định kỳ kiểm tra dữ liệu thiếu, sử dụng các script để kiểm tra sự dày đặc của dữ liệu theo khoảng thời gian cố định. Tuy nhiên, cách này đòi hỏi kiểm tra thủ công và thiếu hiệu quả khi dữ liệu bị thiếu ở nhiều thời điểm khác nhau.
- Giải pháp đã chọn: Data Backfill của Mage để xử lý dữ liệu muộn. Tạo pipeline riêng để backfill, pipeline backfill có nhiệm vụ bổ sung dữ liệu cho các khoảng thời gian bị thiếu, pipeline này được kích hoạt khi xác định được khoảng thời gian dữ liệu bị thiếu dựa trên log và sau đó tải bổ sung vào hệ thống.

Bài học rút ra: Việc tách pipeline backfill ra khỏi pipeline chính đảm bảo hệ thống không bị gián đoạn trong khi vẫn bổ sung được dữ liệu cần thiết. Backfill tự động phát hiện và bổ sung dữ liệu bị thiếu nhờ cơ chế kiểm tra log. Mage cho phép chạy các pipeline đồng thời, giúp quá trình backfill diễn ra nhanh chóng mà không làm giảm hiệu suất của pipeline chính.

4.2 Kinh nghiệm về Stream Processing

4.2.1 Kinh nghiệm 1: Exactly-once processing

Vấn đề: Khi hệ thống xảy ra sự cố, việc xử lý dữ liệu sẽ bị gián đoạn, một vài dữ liệu sẽ có thể bị mất. Hơn nữa, hệ thống cần phải có cơ chế để đảm bảo dữ liệu chỉ được xử lý 1 lần. Điều này là cần thiết trong điều kiện bigquery không hỗ trợ transactional writes. Dữ liệu có thể bị ghi trùng lặp nếu luồng bị khởi động lại.

Giải pháp cuối cùng:

- Đặt failOnDataLoss thành true để Spark không bỏ qua các thông điệp Kafka đã mất. Điều này giúp hệ thống đảm bảo dữ liệu được xử lý đầy đủ.
- Đảm bảo Ghi Dữ liệu là Idempotent: Thêm cột Batch ID làm một phần của tên file hoặc metadata để tránh trùng lặp dữ liệu.
- Sử dụng checkpointLocation trong cả hai luồng ghi (GCS và BigQuery để hỗ trợ việc khôi phục trạng thái của hệ thống khi xảy ra sự cố.

Bài học

- Exactly-once processing không hoàn toàn miễn phí; nó có thể yêu cầu thêm tài nguyên để lưu trạng thái, checkpoint.
- Trạng thái cần được lưu trữ để tránh việc mất dữ liệu khi xảy ra lỗi.
- Cần có cơ chế xử lý để đảm bảo dữ liệu vào bigquery không bị xử lý nhiều lần để giảm thiểu chi phí về thời gian và chi phí xử lý làm sạch dữ liệu sau này.

4.2.2 Kinh nghiệm 2: State management

Vấn đề: Khi làm việc với streaming, cần có cơ chế để phục hồi lại trạng thái nếu hệ thống gặp sự cố hoặc khi tái khởi động.

Giải pháp cuối cùng:

- Checkpointing là một cơ chế trong Spark Streaming để lưu trữ trạng thái và các thông tin tiến trình của các tác vụ trong một thư mục đặc biệt. Khi sử dụng checkpoint, trạng thái sẽ được lưu lại một cách an toàn và có thể phục hồi khi cần
- Checkpointing giúp duy trì trạng thái trong trường hợp hệ thống bị lỗi hoặc bị gián đoạn.
- Dữ liệu trong checkpoint được lưu trữ vào GCS để phục hồi khi hệ thống restart.

Bài học

- Khi sử dụng checkpointing, phải đảm bảo rằng trạng thái của hệ thống được lưu trữ đúng cách để đảm bảo tính nhất quán và phục hồi dữ liệu chính xác trong hệ thống xử lý streaming.

4.2.3 Kinh nghiệm 3: Recovery mechanism

Vấn đề: Cần khôi phục trạng thái và tiếp tục xử lý dữ liệu khi hệ thống gặp sự cố hoặc bị gián đoạn trong quá trình xử lý dòng dữ liệu.

Giải pháp cuối cùng:

- Sử dụng Checkpointing, khi hệ thống bị lỗi hoặc dừng đột ngột, checkpoint sẽ giúp ứng dụng khôi phục lại trạng thái và tiếp tục xử lý từ điểm cuối cùng được lưu trữ thay vì bắt đầu lại từ đầu.

Bài học

- Khi sử dụng checkpointing, phải đảm bảo rằng trạng thái của hệ thống được lưu trữ đúng cách để đảm bảo tính nhất quán và phục hồi dữ liệu chính xác trong hệ thống xử lý streaming.

4.3 Kinh nghiệm về Batch Processing

4.3.1 Kết nối giao diện Spark Web UI từ web browser

Vấn đề: Không thể mở Spark UI, trang mất quá lâu để phản hồi, lỗi ERR_CONNECTION_TIMED_OUT. **Giải pháp**

- Kiểm tra tường lửa
- Kiểm tra master và workers vẫn running
- Kiểm tra địa chỉ External IP
- Kiểm tra cổng nội bộ của máy ảo

Bài học: Đã kiểm tra master và workers vẫn hoạt động, truy cập Spark UI thì cổng nội bộ vẫn được. Vấn đề được giải quyết bằng cách thêm *Firewallrules*, cài đặt cho phép tcp:8080.

4.3.2 Vấn đề chia partition khi submit spark job

Vấn đề: Gặp lỗi khi chia partition theo *year, month, day*

Giải pháp: Chia partition là hằng số bằng cách dùng *.repartition()*

Bài học: nên sử dụng *.repartition()* vì không phụ thuộc vào cấu trúc dữ liệu

4.4 Kinh nghiệm về Data Storage

Vấn đề: Chọn định dạng lưu trữ phù hợp cho dữ liệu thời tiết lớn (15GB, gần 80 triệu bản ghi) để tối ưu hiệu suất truy vấn và tiết kiệm không gian lưu trữ.

Giải pháp:

- Giải pháp đã thử: Đánh giá các định dạng phổ biến như CSV.
- Giải pháp đã chọn: Sử dụng định dạng Parquet.

Bài học: Định dạng cột (columnar) của Parquet cho phép truy vấn chỉ những cột cần thiết, giảm đáng kể lượng dữ liệu đọc vào và tăng tốc độ truy vấn.

4.5 Kinh nghiệm về Performance Optimization

Vấn đề: Một trong những vấn đề lớn của dữ liệu liên quan đến thời tiết này là không có một thuộc tính dữ liệu rõ ràng nào có thể trở thành khóa. Vì vậy trong quá trình truy vấn, với các truy vấn liên quan đến việc lấy dữ liệu trong một khoảng thời gian và vị trí nhất định sẽ tiêu tốn tài nguyên tính toán rất lớn.

Các giải pháp đã thử

- Giải pháp 1: Tạo cluster trên trường time

Đây là cách dễ dàng thực hiện do chỉ cần tạo lại cluster dựa trên trường time

- Giải pháp 2: Tạo cluster trên đồng thời cả ba trường time, latitude và longitude

Do các trường latitude và longitude là số thực nên không thể tạo cluster, do đó trước khi thực hiện cần chuyển đổi về string trước khi thực hiện tạo cluster

Giải pháp được lựa chọn:

- Để kiểm tra hiệu quả, nhóm quyết định thực nghiệm trên 2 query chính thường được sử dụng trong project này:

Query 1:

```

1 SELECT *
2 FROM strong-ward-437213-j6.bigdata_20241.dashboard_main
3 WHERE valid_time >= '{selected_date_str} 00:00:00 UTC'
4   AND valid_time <= '{selected_date_str} 23:00:00 UTC'
5 ORDER BY valid_time, latitude DESC, longitude;
```

Query 2:

```

1 SELECT
2   DATE(TIMESTAMP(valid_time)) AS day,
3   MAX(temperature_celsius) AS max_temperature,
4   MIN(temperature_celsius) AS min_temperature,
5   SUM(total_precipitation) AS total_precipitation
6 FROM `strong-ward-437213-j6.bigdata_20241.dashboard_main`
7 WHERE latitude = {lat_slider}
8   AND longitude = {lon_slider}
```

CHƯƠNG 4. BÀI HỌC KINH NGHIỆM

```
9   AND FORMAT_TIMESTAMP ('%Y-%m', TIMESTAMP(valid_time)) =  
    '{selected_year}-{selected_month:02d}'  
10 GROUP BY day  
11 ORDER BY day;
```

Và sau đây là bảng kết quả:

Giải pháp	QUERY 1		QUERY 2	
	Bytes processed	Slot milliseconds	Bytes processed	Slot milliseconds
Không tạo cluster	7.45 GB	47158	2.83GB	15102
Tạo cluster trên time	75 MB	2392	2.83GB	10541
Tạo cluster trên time, lat, long	79.16 MB	2232	2.6 GB	10358

Bảng 4.1: So sánh hiệu năng giữa các giải pháp phân cụm

Dựa trên kết quả này, nhóm quyết định chọn giải pháp 1 do kết quả giải pháp 2 không quá vượt trội so với giải pháp 1 và giải pháp 1 có thể giữ nguyên kiểu dữ liệu đối với các thuộc tính.

Bài học

- Đối với các dạng dữ liệu không có khóa rõ ràng, ta cần tìm cách truy cập dữ liệu phù hợp nhất với các truy vấn để chọn làm cluster nhằm tối ưu quá trình truy vấn trong cơ sở dữ liệu

4.6 Kinh nghiệm về Monitoring & Debugging

Vấn đề: Giám sát và gỡ lỗi hiệu quả hệ thống xử lý dữ liệu thời tiết tự động để đảm bảo hoạt động ổn định và phát hiện sự cố nhanh chóng.

Giải pháp: Kết hợp Google Cloud Monitoring và Cloud Logging. Thu thập metrics từ các thành phần chính. Thiết lập cảnh báo (Alerting Policies) dựa trên các ngưỡng định sẵn.

Bài học:

- Metrics Collection:** Thu thập metrics từ nhiều nguồn giúp có cái nhìn tổng quan về hoạt động hệ thống. Việc giám sát định kỳ các dashboard giúp phát hiện sớm các vấn đề tiềm ẩn, đặc biệt là tình trạng thiếu tài nguyên (CPU, bộ nhớ) hoặc các task thất bại.
- Alert Configuration:** Cấu hình cảnh báo tự động (Alerting Policies) dựa trên các ngưỡng đã được thiết lập giúp chủ động nhận diện sự cố và can thiệp kịp thời. Nhưng cũng nên điều chỉnh ngưỡng cảnh báo thường xuyên để tránh gặp phải tình trạng cảnh báo giả (false

positives).

4.7 Kinh nghiệm về Data Quality & Testing

4.7.1 Data validation

Vấn đề: Dữ liệu đầu vào không phải lúc nào cũng đảm bảo độ chính xác. Các bản ghi về lượng mưa có thể có thời gian hợp lệ bị NULL. Dữ liệu có định dạng không phù hợp, ngày tháng không theo chuẩn ISO 8601.

Giải pháp: Kiểm tra định dạng và định nghĩa schema, đảm bảo dữ liệu đầu vào tuân theo và được ép cấu trúc mong đợi. Lọc bỏ các trường có giá trị null.

Bài học: Data validation giúp phát hiện vấn đề ngay từ đầu, tránh lan truyền lỗi sang các bước xử lý sau. Kiểm tra định dạng là cách hiệu quả để đảm bảo dữ liệu đầu vào sạch.

4.7.2 Unit testing

Vấn đề: Khi pipeline xử lý dữ liệu lớn (hàng chục triệu bản ghi), thời gian xử lý dài hoặc tiêu tốn nhiều tài nguyên có thể ảnh hưởng đến hiệu suất hệ thống.

Giải pháp: Chia nhỏ tài nguyên cho các worker để xử lý song song. Đo thời gian thực hiện đối với từng job trên các trường hợp: sử dụng standalone mode, tăng giảm số lượng worker và điều chỉnh tài nguyên của mỗi worker.

Bài học rút ra: Performance testing giúp tối ưu hóa pipeline để xử lý dữ liệu lớn hiệu quả hơn. Việc điều chỉnh số partition, số lượng và tài nguyên worker là yếu tố quan trọng để cải thiện hiệu suất.

4.8 Kinh nghiệm về Security & Governance

Vấn đề: Đảm bảo an ninh và tuân thủ các quy định về bảo mật dữ liệu trong hệ thống.

Giải pháp: IAM (Identity and Access Management) của Google Cloud. Bật Cloud Audit Logs cho GCS và BigQuery.

Bài học:

- **Access Control:** Nguyên tắc phân quyền tối thiểu (Principle of Least Privilege) giúp giảm thiểu rủi ro bằng cách chỉ cấp quyền truy cập cần thiết cho từng tài khoản và dịch vụ. IAM của Google Cloud Platform cung cấp cơ chế quản lý quyền truy cập chi tiết và linh hoạt, giúp kiểm soát truy cập vào các tài nguyên GCP (GCS, BigQuery, VM) một cách hiệu quả.
- **Audit Logging:** Bật Cloud Audit Logs cho GCS và BigQuery giúp theo dõi và phát hiện các hoạt động bất thường.

4.9 Kinh nghiệm về Fault Tolerance

4.9.1 Kinh nghiệm 1: Failure recovery

Vấn đề: Khi làm việc với streaming, cần có cơ chế để phục hồi lại trạng thái nếu hệ thống gặp sự cố hoặc khi tái khởi động.

Giải pháp cuối cùng:

- Checkpointing là một cơ chế trong Spark Streaming để lưu trữ trạng thái và các thông tin tiến trình của các tác vụ trong một thư mục đặc biệt. Khi sử dụng checkpoint, trạng thái sẽ được lưu lại một cách an toàn và có thể phục hồi khi cần
- Trong Apache Spark Streaming, trạng thái của các phép toán và các đối tượng được lưu vào checkpoint. Nếu hệ thống gặp sự cố, Spark sẽ khôi phục từ checkpoint và tiếp tục xử lý từ đó.
- Sử dụng replication để sao lưu dữ liệu thành 2 bản (được cài đặt thông qua docker)

Bài học

- Khi sử dụng checkpointing, phải đảm bảo rằng trạng thái của hệ thống được lưu trữ đúng cách để đảm bảo tính nhất quán và phục hồi dữ liệu chính xác trong hệ thống xử lý streaming. Kafka cũng hỗ trợ replication để đảm bảo rằng các bản sao của dữ liệu luôn sẵn có.

4.9.2 Kinh nghiệm 2: Data replication

Vấn đề: Trong streaming, cần sao chép dữ liệu hoặc các sự kiện từ các nguồn đầu vào vào nhiều bản sao khác nhau của hệ thống hoặc các dịch vụ lưu trữ. Điều này đảm bảo rằng nếu có sự cố xảy ra với một bản sao, các bản sao khác có thể tiếp tục phục vụ hoặc giúp khôi phục dữ liệu mà không bị mất mát.

Giải pháp cuối cùng:

- Khi tạo topic trong Kafka, ta có thể cấu hình replication factor. Đây là số lượng bản sao của mỗi partition mà Kafka sẽ tạo ra. Mỗi partition trong Kafka sẽ có một leader và một số followers. Dữ liệu của partition được sao chép từ leader sang các followers. Nếu leader bị lỗi, một follower có thể được chọn làm leader mới và tiếp tục phục vụ các consumer.

Bài học

- Replication factor là một tham số quan trọng khi tạo Kafka Topic. Nó quyết định số lượng bản sao (replicas) của mỗi partition sẽ được tạo ra. Một replication factor quá thấp có thể làm giảm khả năng chịu lỗi của hệ thống, trong khi quá cao có thể gây lãng phí tài nguyên.

4.9.3 Kinh nghiệm 3: Backup strategies

Vấn đề: Cần phải sao lưu dữ liệu trong các hệ thống xử lý dữ liệu streaming, đảm bảo tính toàn vẹn và khả năng phục hồi của dữ liệu khi có sự cố xảy ra.

Giải pháp cuối cùng:

- Sử dụng cơ chế checkpointing ghi lại trạng thái của hệ thống vào một thời điểm cụ thể, giúp hệ thống có thể khôi phục lại từ trạng thái đó khi gặp sự cố hoặc khi quá trình streaming bị gián đoạn.
- Sử dụng chiến lược data replication đã được trình bày ở trên.

Bài học

- Khi sử dụng checkpointing, phải đảm bảo rằng trạng thái của hệ thống được lưu trữ đúng cách để đảm bảo tính nhất quán và phục hồi dữ liệu chính xác trong hệ thống xử lý streaming. Số lượng replication factor cần phải được cân đối.

4.9.4 Kinh nghiệm 4: Disaster recovery

Vấn đề: Mô hình cần thiết kế để bảo vệ và phục hồi hệ thống streaming khi gặp sự cố nghiêm trọng, chẳng hạn như lỗi phần cứng, sự cố mạng, hoặc mất mát dữ liệu.

Giải pháp cuối cùng:

- Sử dụng replication để lưu trữ dữ liệu trên nhiều broker. Nếu một broker bị lỗi, các broker khác vẫn có thể tiếp tục cung cấp dữ liệu mà không gặp gián đoạn, sử dụng topic partitioning để phân phối tải và bảo vệ dữ liệu khi gặp sự cố.

Bài học

- Nhóm sử dụng thêm Kafka Control Center để theo dõi trạng thái của hệ thống streaming.

4.10 Các kinh nghiệm khác

4.10.1 Kinh nghiệm 1: Giả lập streaming với dữ liệu không liên tục

Vấn đề: Để đáp ứng yêu cầu của bài tập lớn, trong điều kiện dữ liệu mà nhóm lựa chọn được cập nhật ba ngày một lần, độ trễ so với thời gian thực là 10 ngày, nhóm cần tìm ra giải pháp để có thể giả lập streaming.

Giải pháp cuối cùng:

- Sử dụng hai producer để lấy dữ liệu từ api của climate data store. Dữ liệu được lấy cách thời gian thực 10 ngày để đảm bảo rằng dữ liệu truy xuất đã chắc chắn được cập nhật. Hai

CHƯƠNG 4. BÀI HỌC KINH NGHIỆM

producer cứ cách một tiếng sẽ bắt đầu làm việc là lấy dữ liệu thời tiết trong vòng 1 tiếng (do thời gian chia nhỏ nhất của bộ dữ liệu là 1 tiếng). Sau đó, producer sẽ rơi vào trạng thái nghỉ trước khi được kích hoạt lại sau một tiếng.

Bài học

- Kinh nghiệm trong việc giả lập streaming, xử lý những nguồn dữ liệu không liên tục và được cập nhật với độ trễ so với thời gian thực.

4.10.2 Kinh nghiệm 2: Xóa nhầm máy ảo trên Google Cloud

Vấn đề: Lỡ tay xóa nhầm máy ảo đã setup **Giải pháp:**

- Kiểm tra snapshots, disk backups
- Cài đặt máy ảo lại từ đầu

Bài học: Mọi nỗ lực tìm kiếm backup hay "thùng rác" đều vô nghĩa. Rất may mắn là trong quá trình làm việc data đều được lưu trữ ở Cloud, và code được đẩy lên github nên nhóm chọn tạo và cài đặt lại trên máy ảo mới. Thiết lập snapshot cho máy ảo mới tạo

TÀI LIỆU THAM KHẢO

- [1] TS. Trần Việt Trung, *Slide Lưu trữ và xử lý dữ liệu lớn..* Trường Công nghệ Thông tin & Truyền thông - Đại học Bách Khoa Hà Nội
- [2] Google Cloud Storage Documentation.
- [3] Apache Kafka Documentation.
- [4] Apache Spark Documentation.
- [5] Streamlit Documentation.
- [6] Climate Data Store.