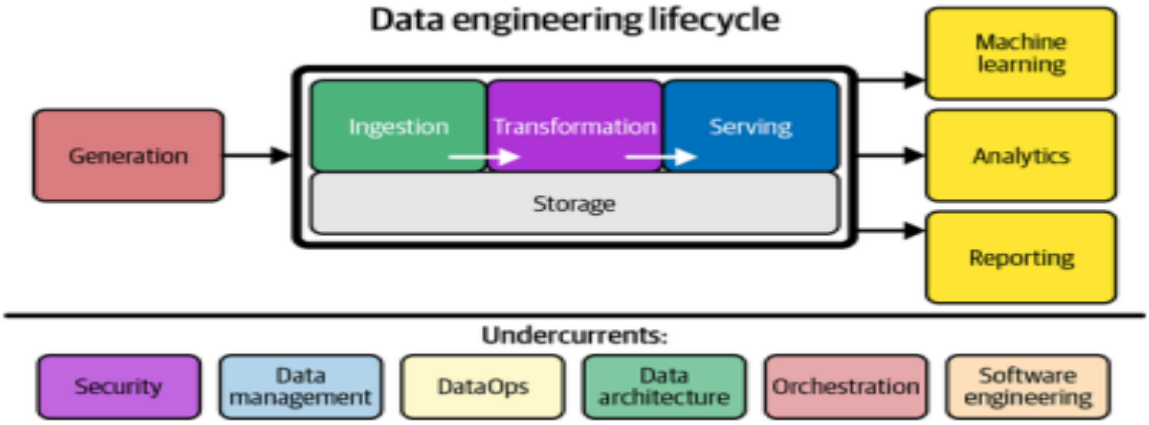
CHƯƠNG 4: THIẾT KẾ

4.1. Tổng quan về vòng đời dữ liệu



4.1.1. Các giai đoạn chính:

- **Generation:** dữ liệu được sinh ra từ các hệ thống nguồn: DBMS, Web, Iot, Sensor, API
- **Storage (lưu trữ):** Dữ liệu thô và đã xử lý cần được lưu trữ một cách an toàn, bền bỉ và có khả năng truy cập hiệu quả.
- **Ingestion (thu thập):** Quá trình di chuyển dữ liệu từ nơi nó được Generation vào lớp Storage.
- **Transformation (biến đổi):** Dữ liệu thô được chuyển đổi thành một dạng hữu ích, đáng tin cậy và sẵn sàng cho việc sử dụng. Quá trình này có thể xảy ra trước hoặc sau khi nạp vào Data Warehouse (ETL vs ELT).
- **Serving (phục vụ):** Cung cấp dữ liệu đã được biến đổi cho các bên liên quan: Data Analytist, Data Scientsit, Software Engineering.

4.1.2. Các yếu tố nền tảng:

- **Data Management (Quản lý Dữ liệu):** Bao gồm quản trị (governance), theo dõi dòng chảy (lineage) và đảm bảo chất lượng dữ liệu. DataOps (Data Operations): Áp dụng các nguyên tắc của DevOps vào dữ liệu, tập trung vào tự động hóa, giám sát và cộng tác để tăng tốc độ và độ tin cậy của pipeline.
- **Data Architecture (Kiến trúc Dữ liệu):** Việc thiết kế tổng thể hệ thống, lựa chọn công nghệ và đảm bảo chúng hoạt động hài hòa với nhau.

- **Orchestration (Điều phối):** Quản lý sự phụ thuộc, lịch trình và thực thi các tác vụ trong pipeline. Đây là vai trò chính của Airflow.
- **Software Engineering (Kỹ thuật Phần mềm):** Áp dụng các phương pháp tốt nhất của kỹ thuật phần mềm như kiểm soát phiên bản (Git), kiểm thử (testing), và viết code sạch vào các pipeline dữ liệu.
- **Security (Bảo mật):** Đảm bảo an toàn dữ liệu ở mọi giai đoạn, từ lúc nghỉ (at rest) đến lúc di chuyển (in transit), thông qua mã hóa và quản lý truy cập (IAM).

4.2. Lựa chọn mô hình và tổng quan kiến trúc

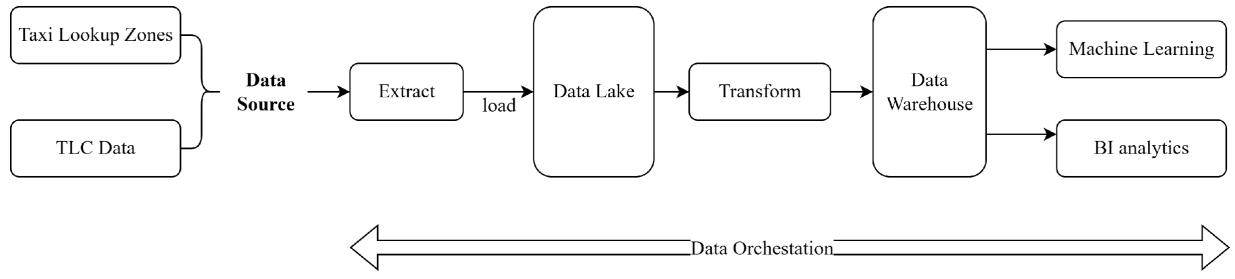
4.2.1. Lựa chọn mô hình xử lý:

Mô hình ELT thay vì ETL truyền thống vì các lí do sau:

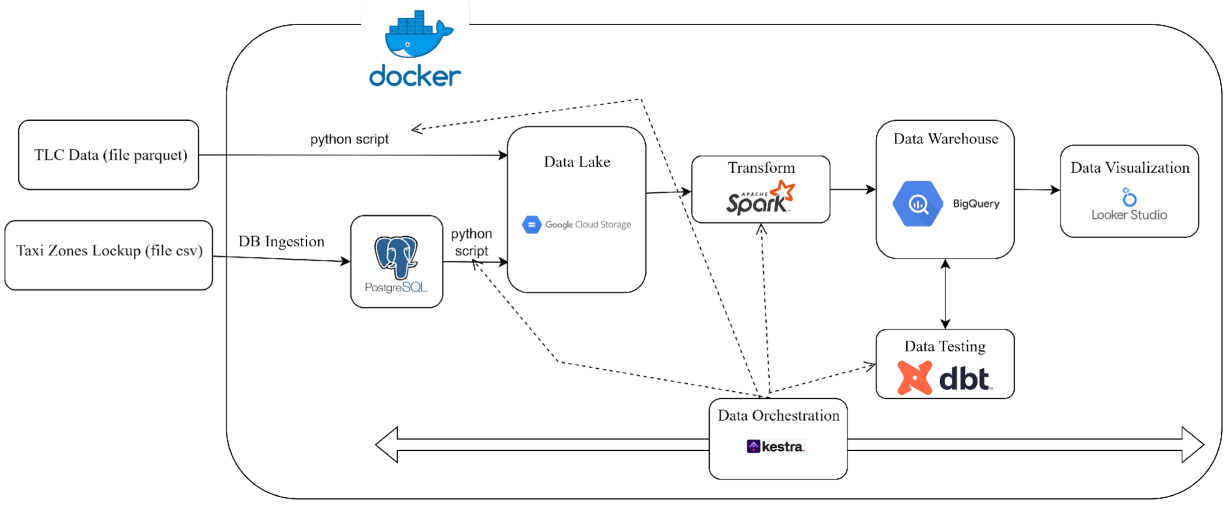
- Tận dụng sức mạnh tính toán của kho dữ liệu hiện đại.
- Tăng tính linh hoạt: Dữ liệu thô được lưu trữ trong Data Lake, cho phép xử lý lại hoặc áp dụng các logic biến đổi mới trong tương lai mà không cần trích xuất lại từ nguồn.
- Tối ưu chi phí và hiệu năng khi nạp dữ liệu theo lô.

4.2.2. Sơ đồ kiến trúc Logic:

Kiến trúc logic của hệ thống dữ liệu



4.3. Sơ đồ kiến trúc vật lý



4.4. Phân tích vai trò và lý do lựa chọn công nghệ

4.4.1. Nền tảng "container hóa": Docker

- Vai trò: Đóng góp ứng dụng các thành phần phụ thuộc của chúng vào một đơn vị độc lập gọi là container.

- Lý do lựa chon:

- **Tính nhất quán của môi trường:** đảm bảo mã nguồn thực thi đồng nhất trên mọi môi trường, từ máy phát triển (local) đến môi trường sản xuất (producction), loại bỏ xung đột về thư viện.
- **Khả năng tái sản xuất (Reproducbility):** Cho phép bất kì ai cũng có thể tái tạo và thực thi pipeline một cách chính xác.
- **Tính cô lập:** Các container chạy trong môi trường cô lập, không ảnh hưởng đến hệ điều hành chủ hoặc các container khác.

4.4.2. Tầng thu thập dữ liệu (DB Ingestion)

- **Công nghệ: Python** kết hợp với thư viện psycopg2 hoặc SQLAlchemy để tương tác với **PostgresSQL.**

- Vai trò: PostgreSQL mô phỏng một hệ thống cơ sở dữ liệu giao dịch trực tuyến (OLTP) nguồn, chứa dữ liệu nghiệp vụ (ví dụ: Taxi Zones Lookup). Các script Python thực hiện quá trình trích xuất dữ liệu từ CSDL này.

- Lý do lựa chọn:

- **Python:** Là ngôn ngữ phổ biến trong kỹ thuật dữ liệu nhờ hệ sinh thái thư viện phong phú (pandas, google-cloud-storage) và cú pháp linh hoạt.
- **PostgreSQL:** Là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở mạnh mẽ, ổn định, rất phù hợp để mô phỏng các hệ thống nguồn trong thực tế.

4.4.3. Tầng lưu trữ dữ liệu thô (Data Lake)

- **Công nghệ:** Google Cloud Storage (GCS)

- Vai trò: Là kho lưu trữ trung tâm cho toàn bộ dữ liệu thô (raw data) và dữ liệu đã qua tiền xử lý, dưới các định dạng file tối ưu như Parquet.

- Lý do lựa chọn:

- **Chi phí hiệu quả và năng mở rộng:** GCS cung cấp dịch vụ lưu trữ đối tượng (object storage) với chi phí thấp và khả năng mở rộng gần như không giới hạn.
- **Nền tảng cho kiến trúc tách biệt:** là thành phần lưu trữ cốt lõi, cho phép các engine tính toán khác nhau (Spark, BigQuery) truy cập và xử lý dữ liệu một cách độc lập.

4.4.4. Tầng Biến đổi Dữ liệu (Transformation)

- **Công nghệ:** Apache Spark và dbt (Data Build Tool).

- Vai trò:

- **Apache Spark:** thực thi các tác vụ biến đổi dữ liệu phức tạp, quy mô lớn, đặc biệt là xử lí dữ liệu thô từ DataLake trước khi nạp vào DataWarehouse.
- **dbt:** quản lý logic biến đổi dữ liệu bên trong Data Warehouse. Biên dịch các mô hình viết bằng SQL thành các câu lệnh SQL thực thi trực tiếp trên BigQuery.

- Lý do lựa chọn:

- **Apache Spark:** là framework tiêu chuẩn cho xử lý dữ liệu phân tán, có khả năng xử lý các tập dữ liệu vượt quá dung lượng bô nhớ của một máy đơn lẻ.
- **dbt:** Thúc đẩy các nguyên tắc kỹ thuật phần mềm trong quá trình phân tích. Cho phép quản lí phiên bản (version control), kiểm thử dữ liệu (data testing), và tạo tài liệu tự động (documentation), nâng cao độ tin cậy và khả năng bảo trì của pipeline.

4.4.5. Kho dữ liệu (Data Warehouse)

- Công nghệ: Google Big Query

- Vai trò: lưu trữ dữ liệu đã được cấu trúc, làm sạch và mô hình hóa, sẵn sàng cho các truy vấn phân tích (OLAP) và báo cáo.

- Lý do lựa chọn:

- **Kiến trúc Serverless:** Hoàn toàn không cần quản lý hạ tầng, cho phép đội ngũ tập trung vào việc khai thác dữ liệu thay vì vận hành.
- **Hiệu năng truy vấn cao:** Kiến trúc phân tán và lưu trữ dạng cột (columnar storage) cho phép BigQuery thực thi các truy vấn phức tạp trên hàng Terabyte dữ liệu trong vài giây.
- **Tích hợp hệ sinh thái:** Tích hợp liền mạch với GCS, dbt, và công cụ BI như Looker Studio.

4.4.6. Tầng Điều phối Dữ liệu (Data Orchestration)

- Công nghệ: Kestra

- Vai trò: điều hành toàn bộ hệ thống, lên lịch (scheduling), điều phối, giám sát việc thực thi của toàn bộ pipeline dữ liệu theo một luồng công việc workflow đã định nghĩa.

- Lý do lựa chọn:

- **Quản lý phụ thuộc phức tạp:** đảm bảo các tác vụ thực thi đúng thứ tự (ví dụ: *dbt run* chỉ chạy sau khi quá trình nạp dữ liệu vào BigQuery hoàn tất.
- **Tự động hóa và giám sát:** Tự động hóa toàn bộ quy trình, cung cấp giao diện để giám sát trạng thái, ghi nhận lịch sử (logging), và gửi lời cảnh báo khi có sự cố. Kestra là một công cụ hiện đại, sử dụng mô hình khai báo (declarative) giúp định nghĩa workflow dễ dàng hơn.

4.4.7. Tầng phục vụ và trực quan hóa (Serving & Visualization)

- Công nghệ: Looker Studio

- Vai trò: Kết nối với Data Warehouse (BigQuery) để xây dựng các báo cáo (reports) và bảng điều khiển (dashboards) tương tác, phục vụ cho người dùng cuối (end-users).

- Lý do lựa chọn:

- **Tích hợp chặt chẽ:** là sản phẩm của Google, Loocker Studio có trình kết nối (connector) gốc tới BigQuery, cho phép truy vấn dữ liệu trực tiếp với hiệu năng cao.
- **Dễ sử dụng và miễn phí:** Cung cấp giao diện kéo-thả trực quan, giúp người dùng nghiệp vụ có thể tự tạo báo cáo mà không cần kiến thức kỹ thuật sâu.

4.5. Thiết kế chi tiết Data Lake (trên Google Cloud Storage)

Sau khi xác định kiến trúc tổng quan dựa trên mô hình ELT ở các phần trước, phần này sẽ đi sau và thiết kế chi tiết Data Lake.

4.5.1. Cấu trúc thư mục:

Hệ thống sẽ áp dụng cấu trúc thư mục phân lớp (layered architecture) trên Google Cloud Storage (GCS) để phân tách dữ liệu dựa trên mức độ xử lý của chúng:

- **Lớp raw (/raw)**
    - **Mục đích:** Chứa dữ liệu thô, không qua bất kì chỉnh sửa nào, được tải trực tiếp từ nguồn NYC TLC). Đảm bảo dữ liệu gốc cho các nhu cầu xử lý lại.
    - **Cấu trúc:** Dữ liệu được phân vùng theo loại dịch vụ (service_type), năm (year), và tháng (month) để tối ưu hóa truy vấn và xử lý gia tăng.

```
gs://<your-project-bucket>/raw/
├── tlc_trip_records/
│   ├── service_type=yellow/
│   │   ├── year=2024/
│   │   │   └── month=01/
│   │   │       └── data_2024_01.parquet
│   ├── service_type=green/
│   │   └── ...
│   └── service_type=fhvhv/
│       └── ...
└── taxi_zone_lookup/
    └── taxi_zone_lookup.csv
```

- **Lớp Cleaned (/cleaned)**
    - **Mục đích:** Trong tương lai, lớp này có thể được bổ sung để lưu trữ dữ liệu đã qua các bước làm sạch cơ bản (ví dụ: chuẩn hóa tên cột, ép kiểu dữ liệu) nhưng chưa được tổng hợp hay biến đổi theo logic nghiệp vụ.

4.5.2. Quy ước Đặt tên và Quản lý Phiên bản

- **Quy ước Đặt tên file:** Các file dữ liệu hành trình sẽ được đặt tên thống nhất (ví dụ: *data_[year]_[month].parquet*) bên trong các thư mục đã được phân vùng. Điều này giúp đơn giản hóa các kịch bản tự động hóa và truy vết.

- **Quản lý Phiên bản (Object Versioning):** Tính năng này sẽ được **kích hoạt** trên GCS bucket. Đây là một cơ chế an toàn quan trọng, cho phép khôi phục

lại các phiên bản cũ của file trong trường hợp xảy ra lỗi ghi đè không mong muốn, đảm bảo tính toàn vẹn của dữ liệu thô.

4.6. Thiết kế chi tiết Data Lake (trên Google Cloud Storage)

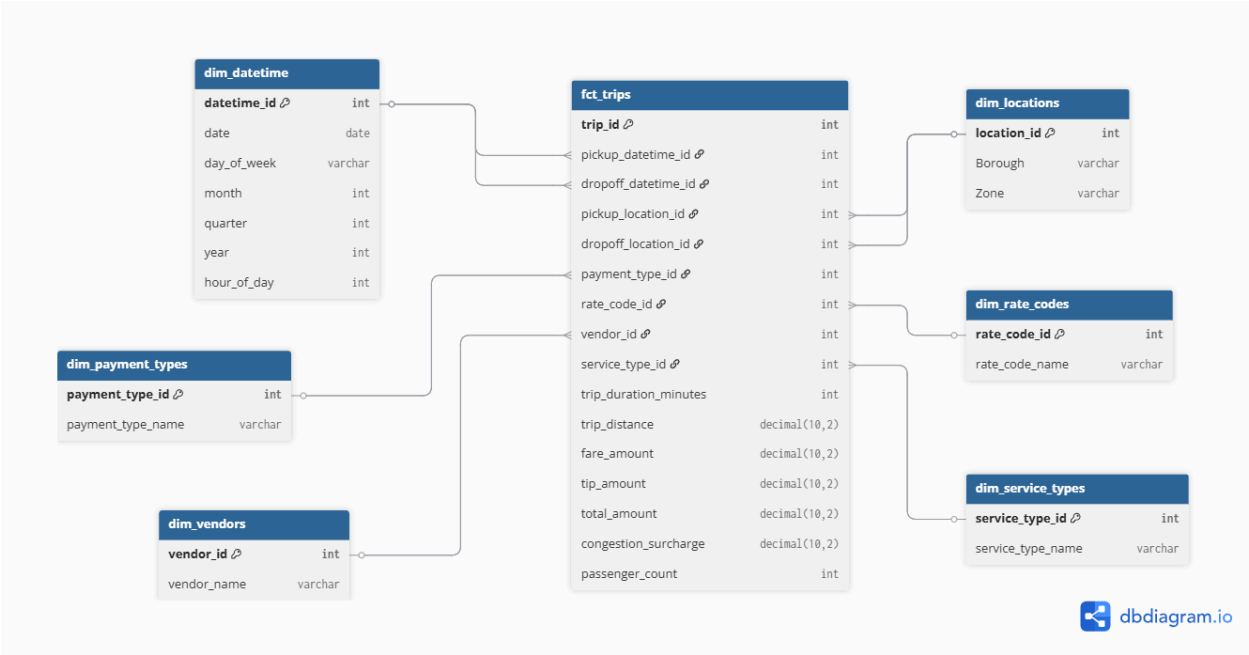
4.5.1. Mô hình Dữ liệu (Star Schema)

Hệ thống sẽ triển khai mô hình sao (Star Schema), một lựa chọn tối ưu cho các truy vấn phân tích, bao gồm một bảng Fact trung tâm và các bảng Dimension xung quanh.

- Bảng Fact (fct_trips):
    - **Mô tả:** Bảng lớn nhất, chứa các số liệu định lượng về mỗi chuyến đi và các khóa ngoại trỏ đến các bảng Dimension. Mỗi dòng đại diện cho một chuyến đi hợp lệ.
    - **Các cột số liệu:** trip_duration_minutes, trip_distance, fare_amount, tip_amount, total_amount, congestion_surcharge, passenger_count.
    - **Các khóa ngoại:** pickup_datatime_id, dropoff_datatime_id, pickup_locaction_id, dropoff_location_id, payment_type_id, rate_code_id, vendor_id, service_type_id.
- Các bảng Dimension (dim.*)
    - **dim_datetime:** Bảng chiều thời gian, được sinh ra để phân tích dữ liệu theo các đơn vị thời gian khác nhau (ngày, tháng, quý, năm, thứ trong tuần, giờ trong ngày).
    - **dim_locations:** Chứa thông tin chi tiết về các khu vực (LocationID, Borough, Zone). Bảng này được join vào fct_trips hai lần để xác định thông tin điểm đón và điểm trả.
    - **dim_payment_types:** Ánh xạ mã phương thức thanh toán (payment_type) sang tên đầy đủ (ví dụ: 'Credit card', 'Cash').
    - **dim_rate_codes:** Ánh xạ mã giá cước (RatecodeID) sang tên đầy đủ (ví dụ: 'Standard rate', 'JFK').

- **dim_service_types:** Bảng tự định nghĩa để chuẩn hóa các loại hình dịch vụ ('Yellow', 'Green', 'FHVHV'), phục vụ cho việc so sánh và lọc dữ liệu.
- **dim_vendors:** Ánh xạ mã nhà cung cấp (VendorID) sang tên đầy đủ.

Mô hình thiết kế data warehouse



4.5.2. Thiết kế tối ưu hiệu năng truy vấn cho Data Warehouse:

- **Phân vùng (Partitioning):**
  - **Chiến lược:** Bảng fct_trips sẽ được phân vùng theo cột thời gian pickup_datetime với mức độ chi tiết là theo **NGÀY (DAY)**.
  - **Lợi ích:** Khi người dùng truy vấn dữ liệu trong một khoảng thời gian nhất định (ví dụ: WHERE DATE(pickup_datetime) BETWEEN '2024-01-01' AND '2024-01-31'), BigQuery sẽ chỉ quét các phân vùng liên quan (partition pruning) thay vì quét toàn bộ bảng, giúp giảm đáng kể lượng dữ liệu xử lý và tiết kiệm chi phí.
  - **Cấu hình DDL:** PARTITION BY DATE(pickup_datetime)
- **Phân cụm (Clustering):**

- **Chiến lược:** Bảng fct_trips (sau khi đã phân vùng) sẽ được phân cụm theo các cột thường được sử dụng trong các mệnh đề lọc (WHERE) hoặc nhóm (GROUP BY).
- **Lợi ích:** Dữ liệu có cùng giá trị trên các cột cluster sẽ được lưu trữ gần nhau về mặt vật lý, giúp tăng tốc độ truy vấn đáng kể khi lọc hoặc tổng hợp theo các cột này.
- **Cấu hình DDL:** CLUSTER BY service_type, pickup_location_id, dropoff_location_id.