**TÀI LIỆU HƯỚNG DẪN THỰC HÀNH**

**Tên bài thực hành:**

**Xây dựng Spotify ETL bằng Python và Airflow**

**10/2023**

**MỤC LỤC**

[1. CƠ SỞ LÝ THUYẾT 2](#_Toc147407991)

[1.1. Giới thiệu chung 2](#_Toc147407992)

[1.2. Nguyên lý hoạt động 11](#_Toc147407993)

[1.3. Nguyên lý dịch & gỡ rối 12](#_Toc147407994)

[2. BÀI THÍ NGHIỆM THỰC HÀNH 14](#_Toc147407995)

[2.1. Mục đích - Yêu cầu 14](#_Toc147407996)

[**2.1.1. Mục đích 14**](#_Toc147407997)

[**2.1.2. Yêu cầu: 14**](#_Toc147407998)

[**2.1.3. Thời gian thực hiện 14**](#_Toc147407999)

[2.2. Chuẩn bị 14](#_Toc147408000)

[**2.2.1. Danh mục thiết bị thực hành 14**](#_Toc147408001)

[2.3. Nội dung 15](#_Toc147408002)

[**2.3.1. Các bước thực hiện 15**](#_Toc147408003)

[**2.3.4. Kết quả thực hiện 35**](#_Toc147408004)

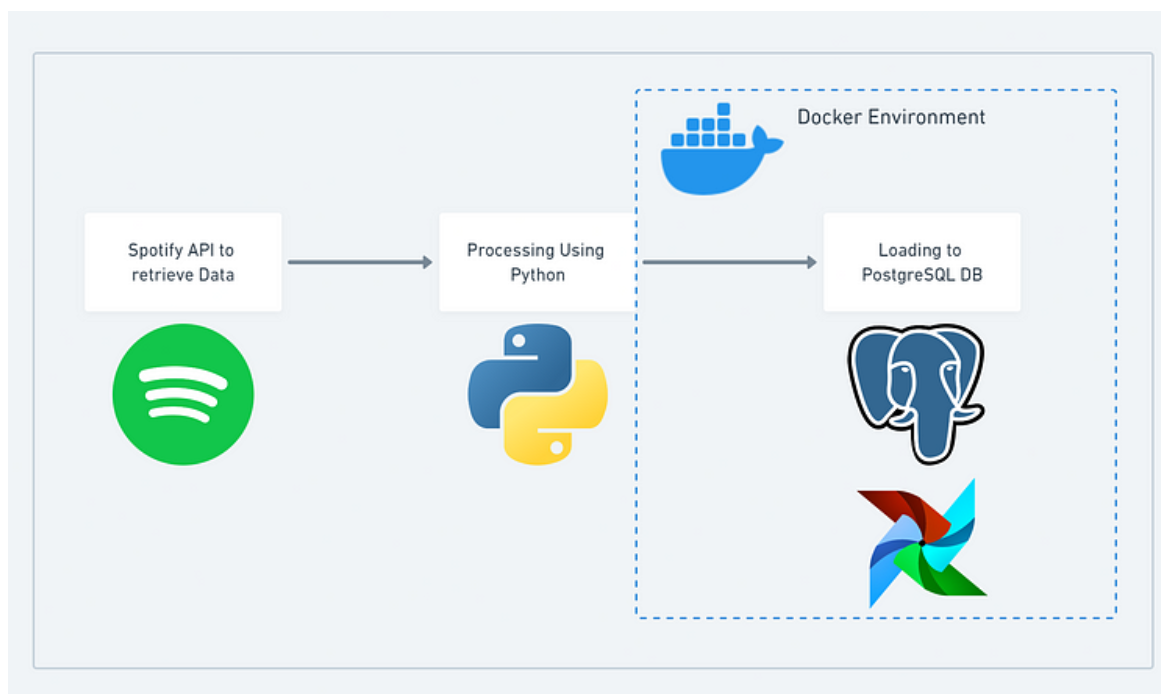
[TÀI LIỆU HƯỚNG DẪN THỰC HÀNH 35](#_Toc147408005)

[PHỤ LỤC 35](#_Toc147408006)

# 1. CƠ SỞ LÝ THUYẾT

## 1.1. Giới thiệu chung

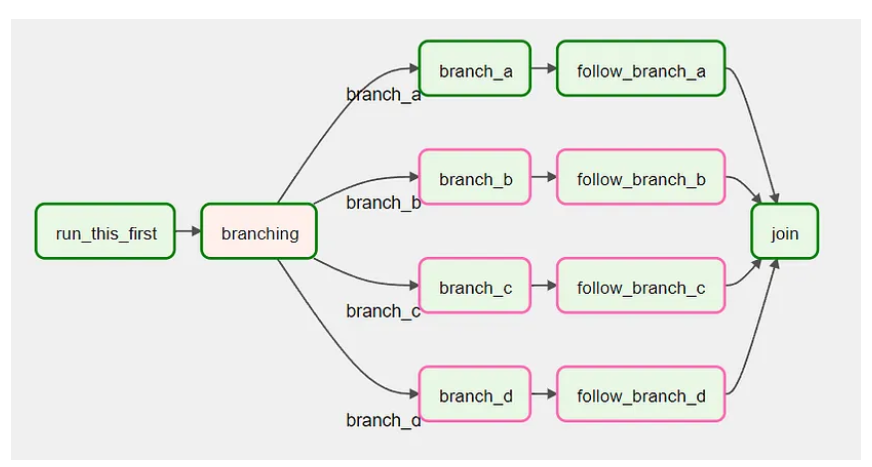
***Làm quen với kiến trúc***



Hình 1 Kiến trúc của hệ thống Spotify ETL bằng Python và Airflow

**+** Khái niệm***:***

* Python:
* Python là một ngôn ngữ lập trình thông dịch và có trình biên dịch.
* Nổi tiếng với cú pháp đơn giản và dễ đọc, Python là một trong những ngôn ngữ phổ biến nhất cho phát triển ứng dụng web, phân tích dữ liệu, trí tuệ nhân tạo, và nhiều lĩnh vực khác.
* Python có một hệ sinh thái đa dạng của các thư viện và framework hỗ trợ cho các loại ứng dụng khác nhau.
* API’s (Application Programming Interfaces):
* API là giao diện cho phép các ứng dụng khác nhau tương tác và trao đổi dữ liệu.
* Các API cho phép các ứng dụng và dịch vụ truy cập và sử dụng các tính năng và dữ liệu từ ứng dụng hoặc dịch vụ khác một cách an toàn và theo quy định.
* Các API có thể được triển khai dưới nhiều hình thức, bao gồm RESTful APIs, GraphQL, SOAP, và nhiều công nghệ khác.
* Docker:
* Docker là một nền tảng công nghệ containerization, cho phép đóng gói ứng dụng và tất cả các phụ thuộc của nó vào một container độc lập.
* Containers giúp đơn giản hóa việc triển khai và quản lý ứng dụng trên các môi trường khác nhau như máy tính cá nhân, máy chủ, hoặc cloud.
* Docker đã trở thành một công cụ quan trọng trong việc phát triển và triển khai ứng dụng.
* Airflow:
* Apache Airflow là một hệ thống quản lý lịch trình và tự động hóa các tác vụ trong hệ thống dựa trên luồng làm việc.
* Airflow cho phép bạn xây dựng, lên lịch, và giám sát các luồng công việc phức tạp thông qua mã Python hoặc dạng đồ thị.
* Nó thường được sử dụng trong quản lý dữ liệu, xử lý dữ liệu, và tự động hóa các quy trình kinh doanh.
* PostgreSQL:
* PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở (RDBMS) mạnh mẽ và phổ biến.
* Nó hỗ trợ nhiều tính năng tiên tiến như ACID transactions, indexes, triggers, và nhiều loại dữ liệu khác nhau.
* PostgreSQL thường được sử dụng trong các dự án phát triển ứng dụng web và quản lý dữ liệu do tính bảo mật và khả năng mở rộng của nó.
* DAG là viết tắt của Đồ thị tuần hoàn có hướng, là một tập hợp các nhiệm vụ được xác định theo thứ tự thực hiện.
* API Hooking (Hook API) là một kỹ thuật sử dụng trong lập trình máy tính để theo dõi và can thiệp vào hoạt động của các hàm và API (Application Programming Interface) trong các ứng dụng hoặc hệ điều hành. Kỹ thuật này thường được sử dụng để theo dõi và kiểm soát các sự kiện, hoặc thay đổi cách hoạt động của các ứng dụng mà người dùng viết hoặc cài đặt.



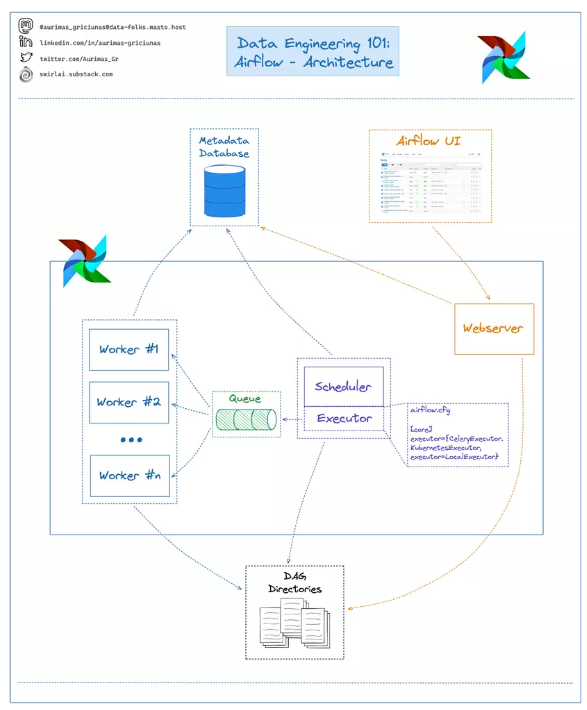
Hình 2 Hình ảnh minh họa về Dags

**+** Đặc điểm

* Python:
* Python là một ngôn ngữ lập trình mã nguồn mở, dễ đọc và dễ hiểu.
* Đa năng: Python được sử dụng cho nhiều mục đích, bao gồm phát triển ứng dụng web, khoa học dữ liệu, máy học, và nhiều lĩnh vực khác.
* Cộng đồng lớn: Python có cộng đồng phát triển và hỗ trợ lớn, có nhiều thư viện và framework sẵn sàng.
* API (Application Programming Interface):
* Giao diện lập trình ứng dụng: API là tập hợp các quy tắc và hướng dẫn cho phép các ứng dụng giao tiếp với nhau.
* Hỗ trợ tích hợp: API cho phép các ứng dụng và dịch vụ khác nhau tương tác và chia sẻ dữ liệu một cách dễ dàng.
* Apache Airflow:
* Quản lý luồng công việc: Apache Airflow là một công cụ quản lý và lên lịch công việc phức tạp.
* Điều khiển công việc: Airflow cho phép bạn xác định các quy trình, lên lịch và theo dõi chúng.
* Docker:
* Đóng gói ứng dụng: Docker cho phép đóng gói ứng dụng và các phụ thuộc của nó vào các container độc lập, giúp đảm bảo tính đồng nhất và di động khi triển khai.
* Tích hợp dễ dàng: Docker có khả năng tích hợp dễ dàng với các công cụ và hệ thống khác, tạo môi trường ứng dụng cô lập.
* PostgreSQL:
* Hệ quản trị cơ sở dữ liệu (DBMS): PostgreSQL là một hệ quản trị cơ sở dữ liệu mạnh mẽ và mã nguồn mở. Nó hỗ trợ nhiều tính năng cao cấp như quan hệ, mảng, JSON và nhiều loại dữ liệu khác.
* An toàn và bảo mật: PostgreSQL cung cấp nhiều tính năng bảo mật, bao gồm xác thực, mã hóa và kiểm soát truy cập.
* API Hooking:
* Theo dõi và ghi lại hoặc can thiệp vào hoạt động của hàm hoặc API.
* Thay đổi cách hoạt động của các ứng dụng mà không cần sửa đổi mã nguồn của chúng.
* Tạo các ứng dụng phụ trợ, chẳng hạn như các công cụ gỡ lỗi (debugging tools), phân tích mã độc hại (malware analysis tools), hoặc tạo các ứng dụng mở rộng (extensions) cho các ứng dụng khác.
* Cách thức hoạt động: API Hooking thường được thực hiện bằng cách chèn mã thay thế (hook) vào hàm hoặc API mục tiêu. Khi hàm đó được gọi, mã hook sẽ được thực hiện trước hoặc sau khi hàm gốc được thực thi. Mã hook có thể thực hiện các công việc như ghi lại tham số, kết quả hoặc can thiệp vào quá trình thực thi của hàm.

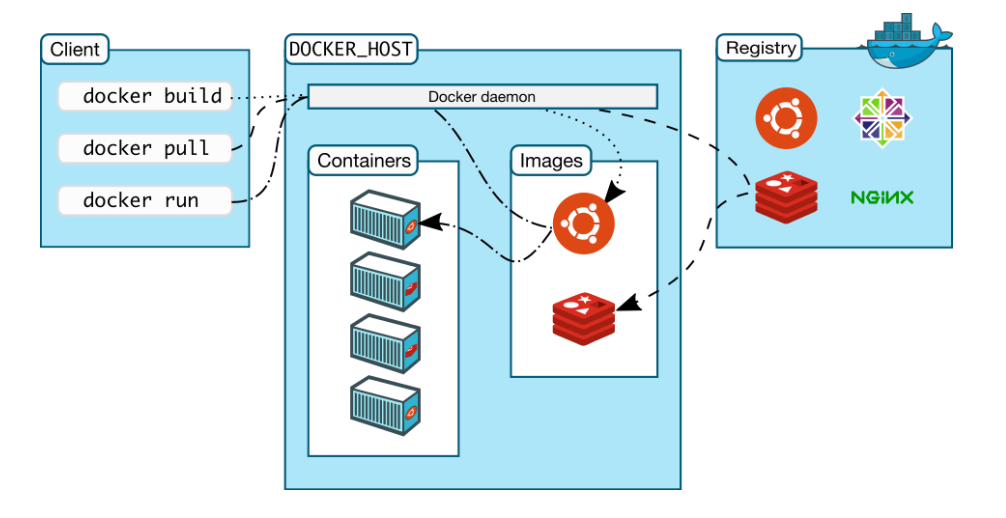
+ Cấu trúc

* Apache Airflow



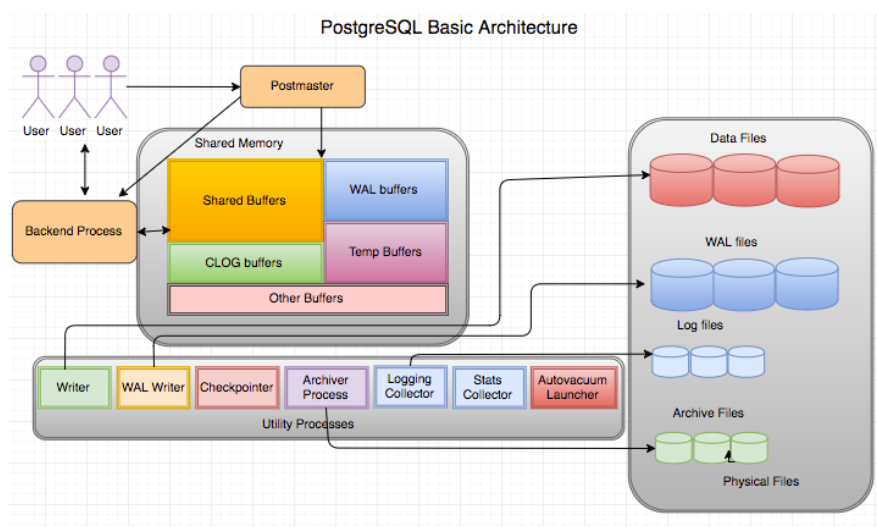
Hình 3 Cấu trúc Apache Airflow

* Scheduler: Một trong những thành phần quan trọng nhất, xử lý cả việc kích hoạt workflow đã lên lịch và gửi Task cho executor để chạy.
* Executor các thành phần được sử dụng để chạy các Task, chúng có thể chạy trực tiếp trên Scheduler hoặc thực thi trên các Worker, tùy vào cách cấu hình
* WebServer: Chứa UI dạng Web được sử dụng để quản lý các Task, DagRun hoặc các giá trị của hệ thống
* Docker



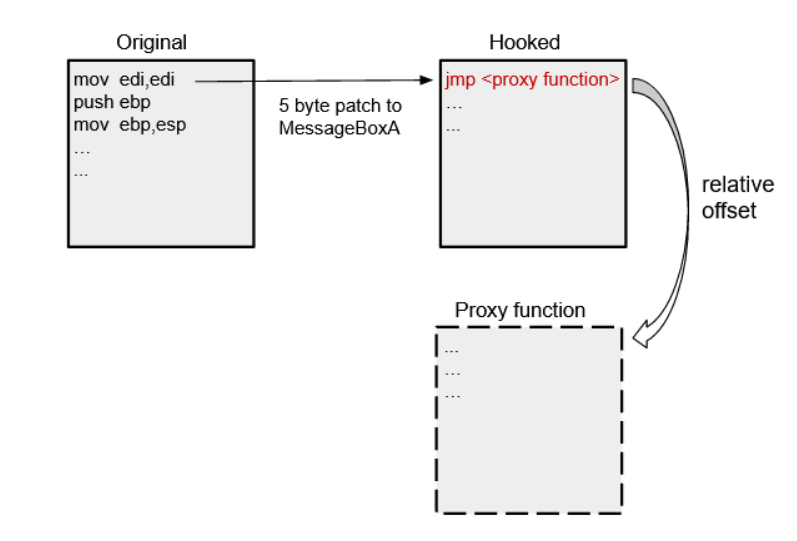
Hình 4 Kiến trúc docker

* Docker Daemon: Docker daemon chạy trên các máy host. Người dùng sẽ không tương tác trực tiếp với các daemon, mà thông qua Docker Client.
* Docker Client: Là giao diện người dùng của Docker, nó cung cấp cho người dùng giao diện dòng lệnh và thực hiện phản hồi với các Docker daemon.
* Docker images: Là một template chỉ cho phép đọc, ví dụ một image có thể chứa hệ điều hành Ubuntu và web app. Images được dùng để tạo Docker container. Docker cho phép chúng ta build và cập nhật các image có sẵn một cách cơ bản nhất, hoặc bạn có thể download Docker images của người khác.
* Docker Container: Docker container có nét giống với các directory. Một Docker container giữ mọi thứ chúng ta cần để chạy một app. Mỗi container được tạo từ Docker image. Docker container có thể có các trạng thái run, started, stopped, moved và deleted.
* PostgreSQL



Hình 5 kiến trúc của PostgreSQL đơn giản

* Bộ nhớ chung: Bộ nhớ chung(Shared Memory) là bộ nhớ dành riêng cho lưu trữ cơ sở dữ liệu và nhật ký giao dịch. Các thành phần quan trọng của bộ nhớ chung có thể kể đến là Shared Buffer, WAL Buffer, CLOG Buffer, Temp Buffer, Work Memory và Vacuum Buffer.
* Shared Buffer: Thao tác đọc và ghi trong bộ nhớ luôn nhanh hơn bất kỳ thao tác nào khác. Thế nên các cơ sở dữ liệu luôn cần bộ nhớ để truy cập nhanh dữ liệu, mỗi khi có truy cập READ và WRITE xuất hiện. Trong PostgreSQL đấy chính là Shared Buffer (được điều khiển bởi tham số shared\_buffers). Dung lượng RAM được cấp phát cho Shared Buffer sẽ là cố định trong suốt thời gian chạy PostgreSQL. Shared Buffer có thể được truy cập bởi tất cả tiến trình server và người dùng kết nối đến cơ sở dữ liệu. Dữ liệu được ghi hay chỉnh sửa trong Shared Buffer được gọi là dirty data, và các đơn vị thao tác trong csdl block (hay page) thay đổi được gọi là dirty block hay dirty page. Dirty data sẽ được ghi vào file vật lý liên tục trên ở đĩa, các file này được gọi là file dữ liệu (data file). Mục đích của Shared Buffer là để giảm thiểu các tác vụ I/O lên đĩa (DISK IO). Để đạt được mục đích đó, nó phải đáp ứng được những yêu cầu sau: Phải truy cập bộ nhớ đệm lớn(hàng chục, trăm gigabites) nhanh chóng, Tối thiểu hoá xung đột khi nhiều người dùng truy cập cùng lúc, Các blocks được sử dụng thường xuyên phải ở trong bộ đệm càng lâu càng tốt.
* WAL Buffer: WAL Buffer còn gọi là transaction log buffers, là bộ nhớ đệm để lưu trữ dữ liệu WAL. Dữ liệu WAL là thông tin về những thay đổi đối với dữ liệu thực tế và dùng để tạo lại dữ liệu trong quá trình sao lưu và phục hồi cơ sở dữ liệu. Dữ liệu WAL được ghi trong file vật lý ở các vị trí liên tục gọi là WAL segments hoặc checkpoint segments. WAL Buffer được điều khiển bởi tham số wal\_buffers, nó được cấp phát bởi RAM của hệ điều hành. Mặc dù nó cũng có thể được truy cập bởi tất cả tiến trình server và người dùng, nhưng nó không phải là một phần của Shared Bufer. WAL Buffer nằm ngoài Shared Buffer và rất nhỏ nếu so sánh với Shared Buffer. Dữ liệu WAL lần đầu tiên sửa đổi sẽ được ghi vào WAL Buffer trước khi được ghi vào WAL Segments trên ổ đĩa. Theo thiết lập mặc định, nó sẽ được phân bổ với kích thước bằng 1/16 Shared Buffer.
* CLOG Buffer: CLOG là viết tắt của commit log, và CLOG Buffer là bộ đệm dành riêng cho lưu trữ các trang commit log được cấp phát bởi RAM của hệ điều hành. Các trang commit log chứa nhật ký về giao dịch và các thông tin khác từ dữ liệu WAL. Các commit log chứa trạng thái commit của tất cả giao dịch và cho biết một giao dịch đã hoàn thành hay chưa. Không có tham số cụ thể để kiểm soát vùng nhớ này. Sẽ có công cụ cơ sở dữ liệu tự động quản lý với số lượng rất nhỏ. Đây là thành phần nhớ dùng chung, có thể được truy cập bởi tất cả tiến trình server và người dùng của csdl PostgreSQL.
* Memory for Lock: Thành phần nhớ này là để lưu trữ tất cả các khóa(lock) nặng được sử dụng bởi PostgreSQL. Các khoá này được chia sẻ trên tất cả tiến trình server hay user kết nối đến csdl. Một thiết lập (không mặc định) giữa hai tham số là max\_locks\_per\_transaction và max\_pred\_locks\_per\_transaction sẽ ảnh hưởng theo một cách nào đó đến kích thước của bộ nhớ này.
* Vacuum Buffer: Đây là lượng bộ nhớ tối đa được sử dụng cho mỗi tiến trình autovacuum worker, được điều khiển bởi tham số autovacuum\_work\_mem. Bộ nhớ được cấp phát bởi RAM của hệ điều hành. Tất cả thiết lập tham số chỉ có hiệu quả khi tiến trình auto vacuum được bật, nếu không các thiết lập này sẽ không ảnh hưởng đến VACUUM đang chạy ở ngữ cảnh khác. Thành phần nhớ này không được chia sẻ bởi bất kỳ tiến trình máy chủ hay người dùng nào.
* Work Memory: Đây là bộ nhớ dành riêng cho một thao tác sắp xếp hoặc bảng băm cho một truy vấn nào đó, được điều khiển bởi tham số work\_mem. Thao tác sắp xếp có thể là ORDER BY, DISTINCT hay MERGE JOIN. Thao tác trên bảng băm có thể là hash-join hoặc truy vấn IN. Các câu truy vấn phức tạp hơn như nhiều thao tác sắp xếp hoặc nhiều bảng băm có thể được cấp phát bởi tham số work\_mem. Vì lý do đó không nên khai báo work\_mem với giá trị quá lớn, vì nó có thể dẫn đến việc sử dụng vùng nhớ của hệ điều hành chỉ cho một câu truy vấn lớn, khiến hệ điều hành thiếu RAM cho các tiến trình cần thiết khác.
* Maintenance Work Memory: Đây là lượng nhớ tối đa mà RAM sử dụng cho các hoạt động bảo trì(maintenance). Các hoạt động bảo trì có thể là VACUUM, CREATE INDEX hay FOREIGN KEY, và được kiểm soát bởi tham số maintenance\_work\_mem.
* Temp Buffer: Các cơ sở dữ liệu cần một hay nhiều bảng mẫu, và các block(page) của bảng mẫu này cần nơi để lưu trữ. Temp Buffer sinh ra nhằm mục đích này, bằng cách sử dụng một phần RAM, được xác định bởi tham số temp\_buffer. Temp Buffer chỉ được sử dụng để truy cập bảng tạm thời trong phiên người dùng. Không có liên hệ gì giữa temp buffer với các file mẫu được tạo trong thư mục pgsql\_tmp để thực hiện sắp xếp lớn hay bảng băm.
* API HOOKING



Hình 6 Cấu trúc của API HOOKING

* Hooking Engine (Máy nạp Hook): Đây là một thành phần quan trọng của API Hooking, nó chịu trách nhiệm quản lý việc thực hiện hook (chèn mã thay thế) vào các hàm hoặc API mục tiêu. Hooking engine cần có khả năng theo dõi và can thiệp vào các cuộc gọi hàm một cách đáng tin cậy.
* Hooked Function (Hàm bị Hook): Đây là hàm hoặc API mà bạn muốn theo dõi hoặc can thiệp vào. Thông thường, bạn sẽ xác định các hàm cụ thể trong ứng dụng mục tiêu để hook.
* Hooking Mechanism (Cơ chế Hooking): Cơ chế này xác định cách hook sẽ được thực hiện. Các cơ chế phổ biến bao gồm Function Hooking, Inline Hooking, System Call Hooking, và COM Hooking. Mỗi cơ chế có cách hoạt động và ứng dụng khác nhau.
* Hook Procedure (Thủ tục Hook): Đây là mã thay thế được chèn vào trước hoặc sau khi hàm bị hook được gọi. Thủ tục hook có thể thực hiện các công việc như ghi lại tham số, thay đổi kết quả hoặc can thiệp vào quá trình thực thi của hàm.
* Unhooking Mechanism (Cơ chế Bỏ Hook): Cần có cơ chế để loại bỏ hook khi không cần thiết nữa để tránh xung đột và tối ưu hóa hiệu suất. Unhooking đảm bảo rằng ứng dụng hoạt động bình thường sau khi hook đã được thực hiện.
* Synchronization (Đồng bộ hóa): Khi có nhiều hooks cùng thực hiện trên một hàm hoặc API, cần phải có cơ chế đồng bộ hóa để đảm bảo rằng các hooks không xung đột với nhau.
* Trampoline Function (Hàm Trampoline): Đây là một hàm phụ thuộc hook procedure để chuyển quyền điều khiển cho hàm gốc sau khi hook procedure đã hoàn thành. Hàm này giúp tránh vòng lặp vô tận khi hook procedure gọi lại chính nó.
* Logging và Handling (Ghi nhật ký và Xử lý): Nếu mục tiêu của hooking là ghi lại hoặc can thiệp vào hoạt động của hàm, bạn cần phải có cơ chế để xử lý và ghi lại thông tin.
* Bypass và Tránh Sự Nhận Diện (Bypass and Anti-Detection Techniques): Một số ứng dụng hoặc hệ thống có thể cố gắng phát hiện và ngăn chặn việc hooking. Trong trường hợp này, bạn có thể cần sử dụng các kỹ thuật bypass và anti-detection để tránh bị phát hiện.

## *1.2. Nguyên lý hoạt động*

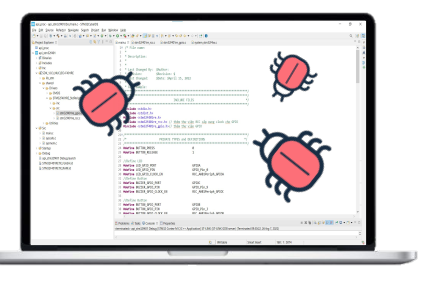
***Nguyên lý hoạt động của quy trình này có thể được mô tả như sau:***

* Khởi tạo quá trình ETL: Quá trình bắt đầu bằng việc khởi tạo quá trình ETL pipeline. Pipeline này có nhiệm vụ lấy dữ liệu từ nguồn Spotify API, chuyển đổi dữ liệu và sau đó đưa nó vào cơ sở dữ liệu PostgreSQL.
* Trích xuất (Extract): Sử dụng API của Spotify để trích xuất dữ liệu từ dịch vụ Spotify. Điều này có thể là thông tin về bài hát, album, hoặc dữ liệu khác từ Spotify. Dữ liệu trích xuất có thể ở định dạng JSON hoặc XML, tùy thuộc vào API của Spotify. Python thường được sử dụng để gửi các yêu cầu HTTP đến API và lấy dữ liệu trả về.
* Biến đổi (Transform): Sau khi dữ liệu được trích xuất, quá trình biến đổi sẽ thực hiện một số chuyển đổi cơ bản trên dữ liệu. Điều này có thể bao gồm việc lọc, chuyển đổi định dạng, hoặc tính toán các chỉ số mới từ dữ liệu ban đầu. Python được sử dụng để thực hiện các biến đổi này. Các thư viện và công cụ thông thường được sử dụng bao gồm Pandas cho xử lý dữ liệu và các thư viện khác cho việc thao tác dữ liệu.
* Kiểm tra Chất lượng Dữ liệu (Data Quality Check): Sau khi biến đổi dữ liệu, quá trình kiểm tra chất lượng dữ liệu sẽ kiểm tra tính đúng đắn của dữ liệu sau khi biến đổi. Điều này có thể bao gồm kiểm tra dữ liệu trống rỗng, kiểm tra giá trị hợp lệ và các kiểm tra khác để đảm bảo chất lượng dữ liệu. Nếu dữ liệu không đáp ứng các tiêu chuẩn chất lượng, quá trình sẽ báo lỗi hoặc quyết định cách xử lý dữ liệu lỗi.
* Nạp (Load): Sau khi dữ liệu đã được trích xuất và biến đổi, nó sẽ được nạp vào cơ sở dữ liệu PostgreSQL. Quá trình này bao gồm việc xác định cơ sở dữ liệu và bảng cần nạp dữ liệu, sau đó sử dụng SQL hoặc các thư viện Python để thực hiện nạp dữ liệu.
* Tự động hóa thông qua Apache Airflow: Apache Airflow được sử dụng để xác định luồng làm việc và lên lịch chạy quá trình ETL pipeline. Các bước trong quá trình ETL được định nghĩa dưới dạng các nhiệm vụ (tasks) trong Airflow. Airflow sẽ tự động thực hiện các nhiệm vụ này theo lịch được đặt trước, và nó cũng quản lý lỗi và tái thử lại nhiệm vụ nếu cần.
* Thời gian ước tính: Thời gian ước tính [4–7 giờ] chỉ là một ước tính và có thể thay đổi tùy thuộc vào khối lượng dữ liệu, phức tạp của quá trình biến đổi, và hiệu suất hệ thống***.***

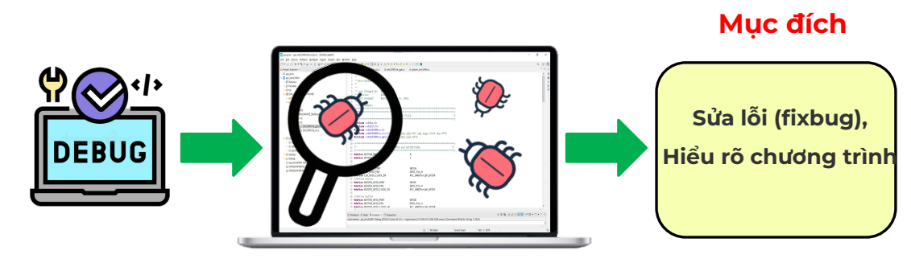
## *1.3. Nguyên lý dịch & gỡ rối*

***+*** Bug được hiểu là thuật ngữ dùng để chỉ các lỗi kỹ thuật có thể xảy ra trong quá trình thiết kế và vận hành các chương trình lập trình.

+ Bug khiến cho phần mềm, ứng dụng không thực thi được hoặc thực thi sai.

**

+ Debug là quá trình tìm kiếm và phát hiện nguyên nhân gây ra lỗi

****

***+ Các tùy chọn với chế độ debug***

* *Python:*
* *Logging: Sử dụng module logging để tạo các bản ghi (logs) để theo dõi luồng của chương trình và giúp xác định vị trí của lỗi.*
* *pdb Debugger: Sử dụng bộ gỡ lỗi tích hợp (integrated debugger) của Python, gọi là pdb, để bắt lỗi và theo dõi giá trị biến trong chương trình.*
* *API:*
* *Postman: Sử dụng Postman hoặc một công cụ tương tự để thực hiện các yêu cầu API và theo dõi các phản hồi.*
* *Curl: Sử dụng lệnh curl để gửi yêu cầu HTTP và nhận phản hồi từ API.*
* *Docker:*
* *Docker Logs: Sử dụng lệnh docker logs để xem các logs từ các container Docker đang chạy.*
* *Docker Inspect: Sử dụng lệnh docker inspect để kiểm tra thông tin chi tiết về một container.*
* *Apache Airflow:*
* *Airflow Logs: Kiểm tra các logs của các công việc và dòng thời gian trong giao diện người dùng Airflow hoặc sử dụng lệnh airflow logs.*
* *Airflow Variables: Kiểm tra các biến môi trường và biến tùy chỉnh trong Airflow để đảm bảo các giá trị đúng đắn.*
* *PostgreSQL:*
* *SQL Queries: Sử dụng lệnh psql hoặc các công cụ quản lý cơ sở dữ liệu PostgreSQL để thực hiện các truy vấn SQL và kiểm tra dữ liệu.*
* *PostgreSQL Logs: Xem các logs của PostgreSQL để tìm hiểu về các hoạt động và lỗi của cơ sở dữ liệu.*

# 2. BÀI THÍ NGHIỆM THỰC HÀNH

## 2.1. Mục đích - Yêu cầu

### 2.1.1. Mục đích

***+*** Hiểu cách tương tác với API để lấy dữ liệu

***+*** Xử lý Dataframe trong Pandas

***+*** Thiết lập Airflow và PostgreSQL thông qua Docker-Compose.

***+*** Học cách tạo DAG trong luồng không khí

### 2.1.2. Yêu cầu:

* Nắm vững kiến thức được giới thiệu, tự tìm hiểu thêm nhiều chức năng khác
* Kiến thức về API
* Hiểu docker và docker-compose là gì
* Python và SQL trung cấp
* Sự hiểu biết cơ bản về Airflow sẽ giúp ích
* Hoàn thành bài code thực hành theo yêu cầu của giảng viên

### 2.1.3. Thời gian thực hiện

+ Thời gian mỗi buổi thực hành là từ 3-4 giờ, chia làm 10-12 nhóm nhỏ, mỗi nhóm 2 sinh viên/1 máy tính. Sinh viên tìm hiểu cơ sở lý thuyết ngắn gọn và các bước thực hiện có thể thao tác dễ dàng.

## 2.2. Chuẩn bị

### 2.2.1. Danh mục thiết bị thực hành

+ Phần cứng: 01 máy tính. Tất cả được được đồng bộ theo số thứ tự từ 1-20

+ Phần mềm: Airflow, Pyhton, Docker, PostgreSQL.

**Danh mục thiết bị thực hành phòng lab**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stt | Tên thiết bị | Số lượng | Thông số  kỹ thuật | Vai trò |
| 1 | Máy tính để bàn | 01 | - Intel Core i3-2100 3.1 Ghz/3M Cache  - 4GB DDR3 (nâng cấp lên ram 8GB thêm 300K)  -SSD 120G Gb chuẩn SATA 3 - 6Gb/s Ổ SSD | -Tính toán, viết chương trình, chạy chương trình |

## 2.3. Nội dung

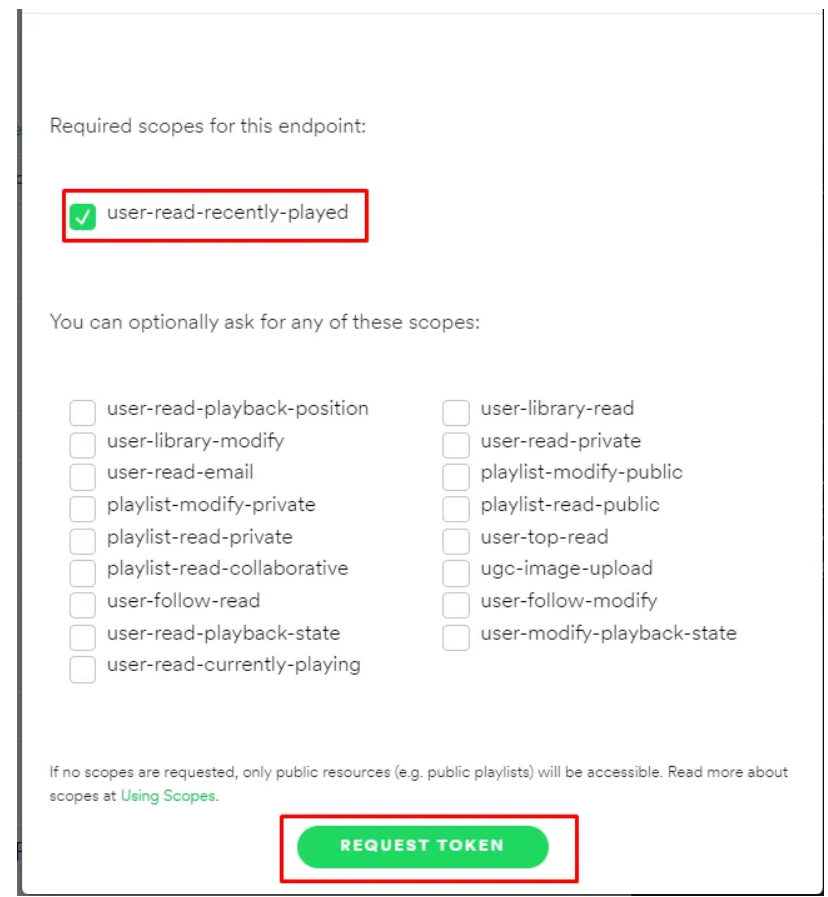
### 2.3.1. Các bước thực hiện

***Nội dung 1:*** Xây dựng Spotify ETL bằng Python và Airflow

**Bước 1: Tạo một project mới “PROJECT1”**

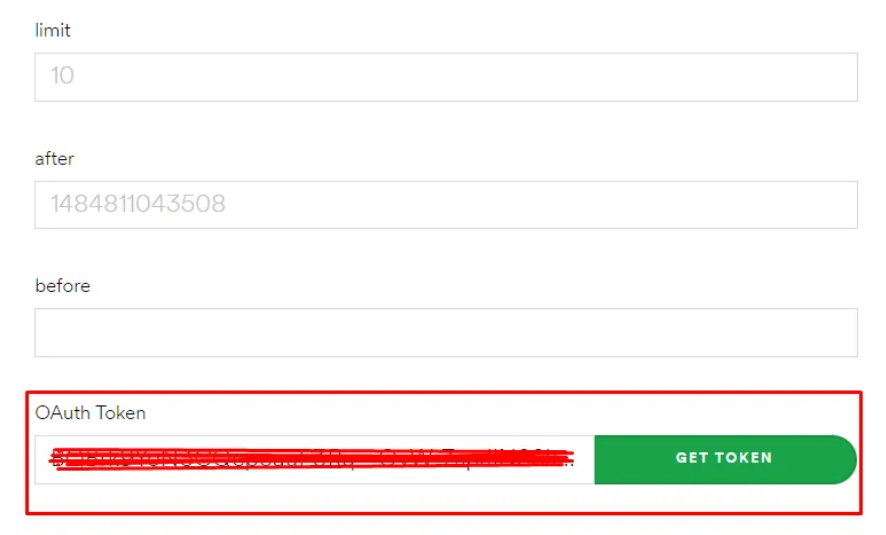
**Bước 2: Xây dựng ETL Pipeline**

|  |
| --- |
| ***/\*******Bộ dữ liệu:****Trong bài thực hành này, chúng ta sử dụng API của Spotify, vì vậy vui lòng tạo một tài khoản cho chính bạn. Sau khi tạo tài khoản, hãy đến đường link https://developer.spotify.com/documentation/web-api/reference/get-recently-played?limit=&after=&before=. Bạn sẽ có thể thấy biểu tượng nhận mã thông báo, nhấp vào đó và chọn người dùng đã thao tác gần đây và nhấp vào nhận mã thông báo.\*/* |

****

Hình 7 Cửa sổ đăng ký

|  |
| --- |
| /\* Bạn có thể thấy mã thông báo của mình như thế này.\*/ |



Hình 8 Hình ảnh minh họa mã thông báo

**Bước 3: tạo và gõ lệnh trong file Extract.py**

|  |
| --- |
| */\* Extract.py - sử dụng mã thông báo này để trích xuất dữ liệu từ Spotify. Chúng ta đang tạo hàm return\_dataframe(). Mã python bên dưới giải thích cách trích xuất dữ liệu API và chuyển đổi nó thành Dataframe.*  *\*/*  import pandas as pd  import requests  from datetime import datetime  import datetime  USER\_ID = "YOUR\_USER\_ID"  TOKEN = "YOUR\_TOKEN"  # Creating an function to be used in other python files  def return\_dataframe():  input\_variables = {  "Accept" : "application/json",  "Content-Type" : "application/json",  "Authorization" : "Bearer {token}".format(token=TOKEN)  }  today = datetime.datetime.now()  yesterday = today - datetime.timedelta(days=1 #no of Days u want the data for)  yesterday\_unix\_timestamp = int(yesterday.timestamp()) \* 1000  # Download all songs you've listened to "after yesterday", which means in the last 24 hours  r = requests.get("https://api.spotify.com/v1/me/player/recently-played?limit=50&after={time}".format(time=yesterday\_unix\_timestamp), headers = input\_variables)  data = r.json()  song\_names = []  artist\_names = []  played\_at\_list = []  timestamps = []  # Extracting only the relevant bits of data from the json object  for song in data["items"]:  song\_names.append(song["track"]["name"])  artist\_names.append(song["track"]["album"]["artists"][0]["name"])  played\_at\_list.append(song["played\_at"])  timestamps.append(song["played\_at"][0:10])  # Prepare a dictionary in order to turn it into a pandas dataframe below  song\_dict = {  "song\_name" : song\_names,  "artist\_name": artist\_names,  "played\_at" : played\_at\_list,  "timestamp" : timestamps  }  song\_df = pd.DataFrame(song\_dict, columns = ["song\_name", "artist\_name", "played\_at", "timestamp"])  return song\_df |

**Bước 4: tạo và gõ lệnh trong file Transform.py**

|  |
| --- |
| /\* xuất tệp Extract để lấy dữ liệu.  + def Data\_Quality(load\_df): Được sử dụng để kiểm tra frame dữ liệu trống, thực thi các ràng buộc duy nhất, kiểm tra các giá trị null. Vì những dữ liệu này có thể làm hỏng cơ sở dữ liệu của chúng ta nên điều quan trọng là chúng ta phải thực thi các biện pháp kiểm tra Chất lượng Dữ liệu này.  + def Transform\_df(load\_df): Sau đó chúng ta viết một số logic theo yêu cầu. Chúng ta muốn biết nghệ sĩ yêu thích của mình nên chúng ta tiến hành nhóm các bài hát mà nghệ sĩ đó đã nghe. Lưu ý: Bước này không bắt buộc, bạn có thể triển khai nó hoặc bất kỳ logic nào khác nếu muốn nhưng hãy đảm bảo rằng bạn thực thi ràng buộc chính.  \*/  import Extract  import pandas as pd  # Set of Data Quality Checks Needed to Perform Before Loading  def Data\_Quality(load\_df):  #Checking Whether the DataFrame is empty  if load\_df.empty:  print('No Songs Extracted')  return False    #Enforcing Primary keys since we don't need duplicates  if pd.Series(load\_df['played\_at']).is\_unique:  pass  else:  #The Reason for using exception is to immediately terminate the program and avoid further processing  raise Exception("Primary Key Exception,Data Might Contain duplicates")    #Checking for Nulls in our data frame  if load\_df.isnull().values.any():  raise Exception("Null values found")  # Writing some Transformation Queries to get the count of artist  def Transform\_df(load\_df):  #Applying transformation logic  Transformed\_df=load\_df.groupby(['timestamp','artist\_name'],as\_index = False).count()  Transformed\_df.rename(columns ={'played\_at':'count'}, inplace=True)  #Creating a Primary Key based on Timestamp and artist name  Transformed\_df["ID"] = Transformed\_df['timestamp'].astype(str) +"-"+ Transformed\_df["artist\_name"]  return Transformed\_df[['ID','timestamp','artist\_name','count']]  if \_\_name\_\_ == "\_\_main\_\_":  #Importing the songs\_df from the Extract.py  load\_df=Extract.return\_dataframe()  Data\_Quality(load\_df)  #calling the transformation  Transformed\_df=Transform\_df(load\_df)  print(Transformed\_df) |

**Bước 5: tạo và gõ lệnh trong file Load.py**

|  |
| --- |
| /\* Trong bước này, chúng ta sử dụng sqlalchemy và SQLite để tải dữ liệu của mình vào cơ sở dữ liệu và lưu tệp vào thư mục dự án.\*/  import Extract  import Transform  import sqlalchemy  import pandas as pd  from sqlalchemy.orm import sessionmaker  import requests  import json  from datetime import datetime  import datetime  import sqlite3  DATABASE\_LOCATION = "sqlite:///my\_played\_tracks.sqlite"  if \_\_name\_\_ == "\_\_main\_\_":  #Importing the songs\_df from the Extract.py  load\_df=Extract.return\_dataframe()  if(Transform.Data\_Quality(load\_df) == False):  raise ("Failed at Data Validation")  Transformed\_df=Transform.Transform\_df(load\_df)  #The Two Data Frame that need to be Loaded in to the DataBase  #Loading into Database  engine = sqlalchemy.create\_engine(DATABASE\_LOCATION)  conn = sqlite3.connect('my\_played\_tracks.sqlite')  cursor = conn.cursor()  #SQL Query to Create Played Songs  sql\_query\_1 = """  CREATE TABLE IF NOT EXISTS my\_played\_tracks(  song\_name VARCHAR(200),  artist\_name VARCHAR(200),  played\_at VARCHAR(200),  timestamp VARCHAR(200),  CONSTRAINT primary\_key\_constraint PRIMARY KEY (played\_at)  )  """  #SQL Query to Create Most Listened Artist  sql\_query\_2 = """  CREATE TABLE IF NOT EXISTS fav\_artist(  timestamp VARCHAR(200),  ID VARCHAR(200),  artist\_name VARCHAR(200),  count VARCHAR(200),  CONSTRAINT primary\_key\_constraint PRIMARY KEY (ID)  )  """  cursor.execute(sql\_query\_1)  cursor.execute(sql\_query\_2)  print("Opened database successfully")  #We need to only Append New Data to avoid duplicates  try:  load\_df.to\_sql("my\_played\_tracks", engine, index=False, if\_exists='append')  except:  print("Data already exists in the database")  try:  Transformed\_df.to\_sql("fav\_artist", engine, index=False, if\_exists='append')  except:  print("Data already exists in the database2")  #cursor.execute('DROP TABLE my\_played\_tracks')  #cursor.execute('DROP TABLE fav\_artist')  conn.close()  print("Close database successfully") |

**Bước 6: hoàn thành đường dẫn ETL cho project**

|  |
| --- |
| ***E:\DE\PROJECTS\SPOTIFY\_ETL\Project1 /\* đường dẫn lưu thư dự án của bạn\*/***  ***│ Extract.py***  ***│ Load.py***  ***│ my\_played\_tracks.sqlite***  ***│ Spotify\_etl.py***  ***│ Transform.py***  ***└───*** |

**Bước 7: Sinh ra tệp .splite**

|  |
| --- |
| */\** *chạy Load.py, bạn có thể thấy tệp .sqlite sẽ được lưu vào thư mục dự án, để kiểm tra dữ liệu bên trong đầu tệp bằng cách truy cập link sau https://inloop.github.io/sqlite-viewer/ và thả tệp của bạn.\*/* |

****

Hình 9 giao diện kiểm tra tệp

**Bước 8: Tự động hóa thông qua Airflow**

|  |
| --- |
| ***/\**** *Một số điểm quan trọng dưới đây:*   * *Chúng ta đã hoàn thành một ETL và bản thân đây là một dự án nhỏ nên sẽ tiết kiệm được công việc.* * *Bây giờ chúng ta sẽ mở rộng điều này với luồng không khí bằng cách sử dụng docker.* * *Tại sao docker? chúng tôi đang sử dụng docker vì nó dễ cài đặt và bảo trì hơn và nó độc lập với hệ điều hành.* * *Làm cách nào để thiết lập luồng không khí bằng Docker? Thực hiện theo hướng dẫn được cung cấp trong blog này .* * *Các bạn cần thay đổi riêng file Yaml theo hướng dẫn trên vui lòng tham khảo tại đây* * *Sau khi thiết lập luồng không khí, hãy đặt dags của bạn vào thư mục dags.* * *Sau khi docker hoạt động, bạn có thể thấy 4 dịch vụ đang chạy.\*/* |

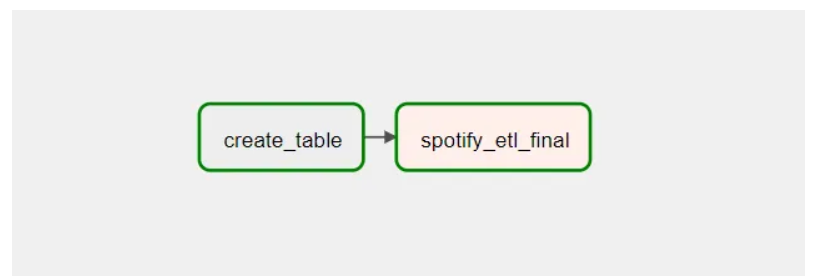


Hình 10 Sau khi docker hoạt động, bạn có thể thấy 4 dịch vụ đang chạy

|  |
| --- |
| C:\USERS\SIDHA\DOCKER\AIRFLOW /\* đường dẫn thư mục airflow trên máy cá nhân của bạn\*/  │ docker-compose.yml  ├───dags  │ │ YOUR\_DAGS.py  ├───logs  ├───plugins  └───scripts |

**Bước 9: Truy cập giao diện người dùng Airflow**

|  |
| --- |
| */\** + *Sau khi đã thiết lập Airflow, chúng ta có thể xem giao diện người dùng bằng cách truy cập cổng 8080. Nhập tên người dùng và mật khẩu.*  *+ Vì vậy, bên trong dag của chúng ta cần tạo các nhiệm vụ để hoàn thành công việc của mình. Để đơn giản, chúng ta sẽ sử dụng hai tác vụ, một tác vụ là tạo Bảng Postgres và tác vụ khác để tải Dữ liệu vào Bảng, dag của chúng ta sẽ trông như thế này.\*/* |

****

Hình 11 Hai tác vụ của Dags

**Bước 10: Tạo và cấu hình file Spotify\_etl.py**

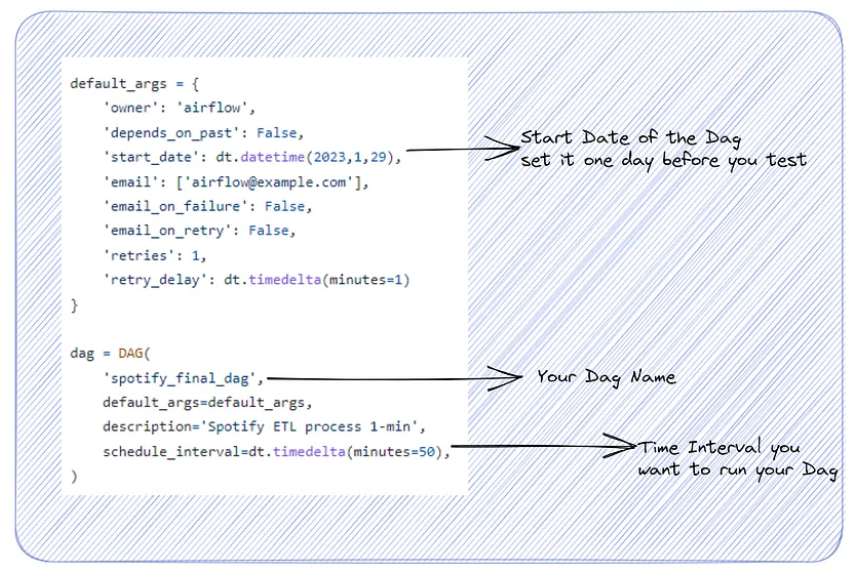
|  |
| --- |
| /\* *Trong File Python này sẽ viết logic để trích xuất dữ liệu từ API → Thực hiện Kiểm tra chất lượng → Chuyển đổi dữ liệu.*  *+ yesterday = today — datetime.timedelta(days=1) → Xác định số ngày bạn muốn có dữ liệu, thay đổi theo ý muốn vì công việc của chúng ta là tải hàng ngày mà tôi đã đặt thành 1.*  *+ def Spotify\_etl() → Hàm cốt lõi trả về frame dữ liệu cho tệp python DAG.*  *Tập tin này cần được đặt bên trong thư mục dags*  *\*/*  *import pandas as pd*  *import requests*  *from datetime import datetime*  *import datetime*  *import pandas as pd*  *import requests*  *from datetime import datetime*  *import datetime*  *USER\_ID = "YOUR\_USERNAME\_HERE"*  *TOKEN = "YOUR\_TOKEN\_HERE"*  *print('started')*  *# Creating an function to be used in other pyrhon files*  *def return\_dataframe():*  *input\_variables = {*  *"Accept" : "application/json",*  *"Content-Type" : "application/json",*  *"Authorization" : "Bearer {token}".format(token=TOKEN)*  *}*    *today = datetime.datetime.now()*  *yesterday = today - datetime.timedelta(days=1)*  *yesterday\_unix\_timestamp = int(yesterday.timestamp()) \* 1000*  *# Download all songs you've listened to "after yesterday", which means in the last 24 hours*  *r = requests.get("https://api.spotify.com/v1/me/player/recently-played?limit=50&after={time}".format(time=yesterday\_unix\_timestamp), headers = input\_variables)*  *data = r.json()*  *song\_names = []*  *artist\_names = []*  *played\_at\_list = []*  *timestamps = []*  *# Extracting only the relevant bits of data from the json object*  *for song in data["items"]:*  *song\_names.append(song["track"]["name"])*  *artist\_names.append(song["track"]["album"]["artists"][0]["name"])*  *played\_at\_list.append(song["played\_at"])*  *timestamps.append(song["played\_at"][0:10])*    *# Prepare a dictionary in order to turn it into a pandas dataframe below*  *song\_dict = {*  *"song\_name" : song\_names,*  *"artist\_name": artist\_names,*  *"played\_at" : played\_at\_list,*  *"timestamp" : timestamps*  *}*  *song\_df = pd.DataFrame(song\_dict, columns = ["song\_name", "artist\_name", "played\_at", "timestamp"])*  *return song\_df*  *def Data\_Quality(load\_df):*  *#Checking Whether the DataFrame is empty*  *if load\_df.empty:*  *print('No Songs Extracted')*  *return False*    *#Enforcing Primary keys since we don't need duplicates*  *if pd.Series(load\_df['played\_at']).is\_unique:*  *pass*  *else:*  *#The Reason for using exception is to immediately terminate the program and avoid further processing*  *raise Exception("Primary Key Exception,Data Might Contain duplicates")*    *#Checking for Nulls in our data frame*  *if load\_df.isnull().values.any():*  *raise Exception("Null values found")*  *# Writing some Transformation Queries to get the count of artist*  *def Transform\_df(load\_df):*  *#Applying transformation logic*  *Transformed\_df=load\_df.groupby(['timestamp','artist\_name'],as\_index = False).count()*  *Transformed\_df.rename(columns ={'played\_at':'count'}, inplace=True)*  *#Creating a Primary Key based on Timestamp and artist name*  *Transformed\_df["ID"] = Transformed\_df['timestamp'].astype(str) +"-"+ Transformed\_df["artist\_name"]*  *return Transformed\_df[['ID','timestamp','artist\_name','count']]*  *def spotify\_etl():*  *#Importing the songs\_df from the Extract.py*  *load\_df=return\_dataframe()*  *Data\_Quality(load\_df)*  *#calling the transformation*  *Transformed\_df=Transform\_df(load\_df)*  *print(load\_df)*  *return (load\_df)*  *spotify\_etl()* |

**Bước 11: Tạo và cấu hình file spotify\_final\_dag.py**

|  |
| --- |
| /\* *Đây là phần quan trọng nhất bạn cần chú ý. Trước tiên, hãy tìm hiểu những kiến ​​thức cơ bản về L DAG tại đường link sau https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html , quá trình này có thể mất khoảng 15 phút hoặc bạn có thể tìm kiếm trên youtube. Sau những điều cơ bản, hãy làm theo hướng dẫn dưới đây.*  *+ từ airflow.operators.python\_operator import PythonOperator → chúng ta đang sử dụng toán tử python để thực hiện các hàm python chẳng hạn như chèn DataFrame vào bảng.*  *+ từ Airflow.providers.postgres.operators.postgres import PostgresOperator → chúng ta đang sử dụng toán tử Postgres để tạo bảng trong cơ sở dữ liệu Postgres của mình.*  *+ từ Airflow. hooks.base\_hook import BaseHook → Hook là một bản tóm tắt của một API cụ thể cho phép Airflow tương tác với hệ thống bên ngoài. Hook được tích hợp vào nhiều toán tử, nhưng chúng cũng có thể được sử dụng trực tiếp trong mã DAG. Chúng ta đang sử dụng một cái móc ở đây để kết nối Cơ sở dữ liệu Postgres từ hàm python của chúng ta*  *+từ Spotify\_etl nhập Spotify\_etl → Nhập hàm Spotify\_etl từ Spotify\_etl.py\*/*  *import datetime as dt*  *from airflow import DAG*  *from airflow.operators.python\_operator import PythonOperator*  *from airflow.hooks.base\_hook import BaseHook*  *from airflow.providers.postgres.hooks.postgres import PostgresHook*  *from airflow.providers.postgres.operators.postgres import PostgresOperator*  *from sqlalchemy import create\_engine*  *from airflow.utils.dates import days\_ago*  *from spotify\_etl import spotify\_etl*  *default\_args = {*  *'owner': 'airflow',*  *'depends\_on\_past': False,*  *'start\_date': dt.datetime(2023,1,29),*  *'email': ['airflow@example.com'],*  *'email\_on\_failure': False,*  *'email\_on\_retry': False,*  *'retries': 1,*  *'retry\_delay': dt.timedelta(minutes=1)*  *}*  *dag = DAG(*  *'spotify\_final\_dag',*  *default\_args=default\_args,*  *description='Spotify ETL process 1-min',*  *schedule\_interval=dt.timedelta(minutes=50),*  *)*  *def ETL():*  *print("started")*  *df=spotify\_etl()*  *#print(df)*  *conn = BaseHook.get\_connection('postgre\_sql')*  *engine = create\_engine(f'postgresql://{conn.login}:{conn.password}@{conn.host}:{conn.port}/{conn.schema}')*  *df.to\_sql('my\_played\_tracks', engine, if\_exists='replace')*  *with dag:*  *create\_table= PostgresOperator(*  *task\_id='create\_table',*  *postgres\_conn\_id='postgre\_sql',*  *sql="""*  *CREATE TABLE IF NOT EXISTS my\_played\_tracks(*  *song\_name VARCHAR(200),*  *artist\_name VARCHAR(200),*  *played\_at VARCHAR(200),*  *timestamp VARCHAR(200),*  *CONSTRAINT primary\_key\_constraint PRIMARY KEY (played\_at)*  *)*  *"""*  *)*  *run\_etl = PythonOperator(*  *task\_id='spotify\_etl\_final',*  *python\_callable=ETL,*  *dag=dag,*  *)*  *create\_table >> run\_etl* |

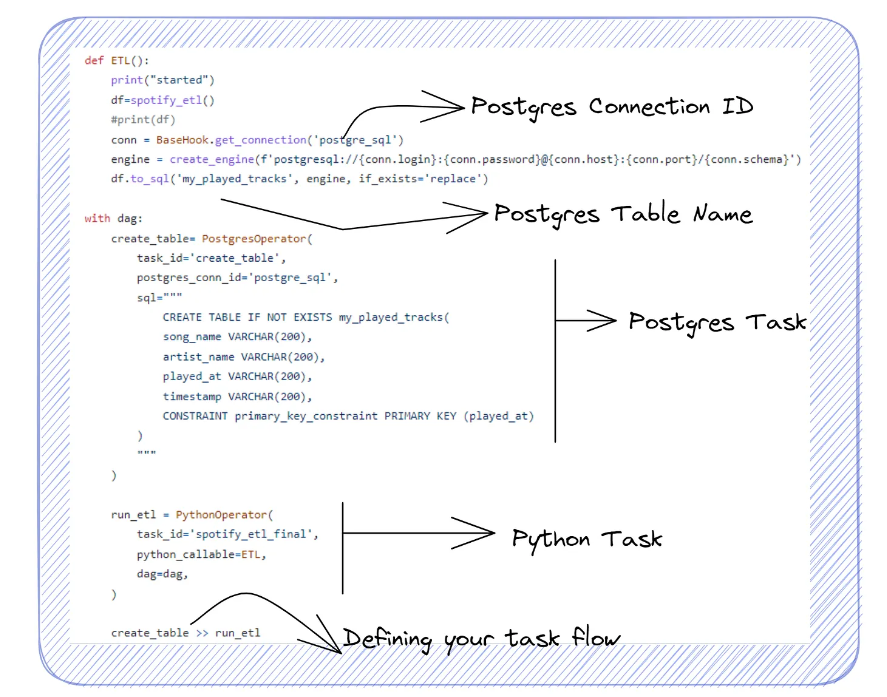
**Chú thích code:**

* Thiết lập các đối số mặc định và khoảng thời gian. Chúng ta có thể thay đổi ngày bắt đầu khoảng thời gian theo nhu cầu của mình.

****

Hình 12 Thiết lập các đối số mặc định và khoảng thời gian

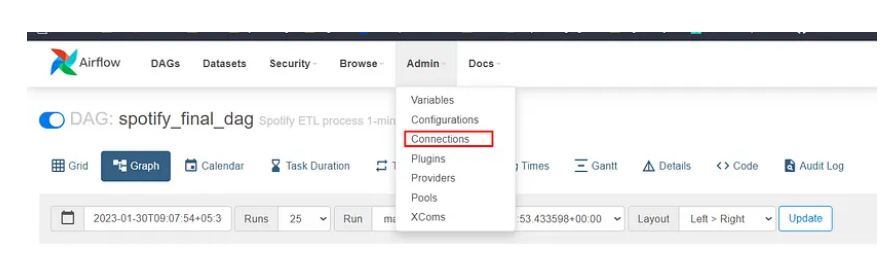
* **Hiểu kết nối và nhiệm vụ của Postgres.**
* conn = BaseHook.get\_connection('[ID kết nối của bạn]') → Kết nối với cơ sở dữ liệu Postgres của bạn.
* df.to\_sql('[Tên bảng của bạn]', engine, if\_exists='replace') → Tải DF vào bảng
* create\_table >> run\_etl → Xác định luồng tác vụ



Hình 13 Kết nối Postgres

**Bước 12: Thiết lập kết nối Postgres trên giao diện người dùng Airflow**

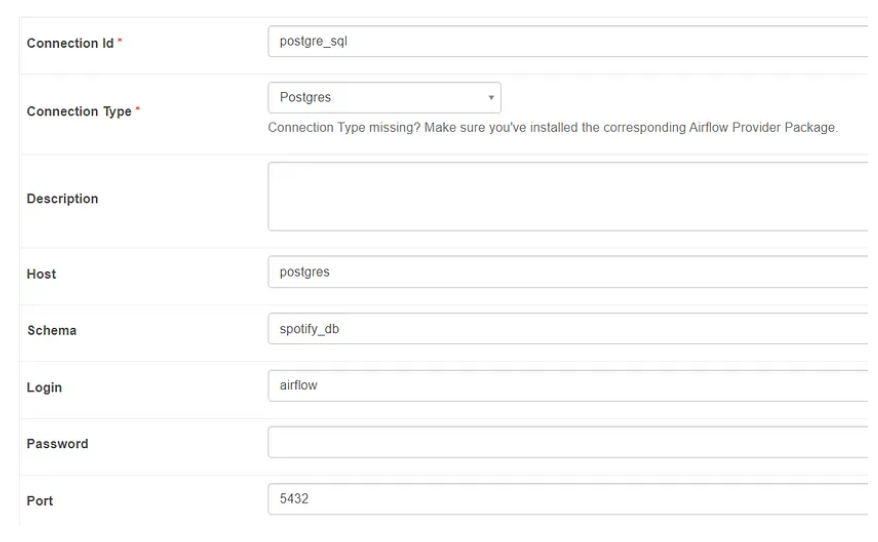
|  |
| --- |
| */\* Đi tới giao diện người dùng Airflow và nhấp vào connection.\*/* |

****

Hình 14 giao diện người dùng Airflow

**Bước 13: Tạo kết nối**

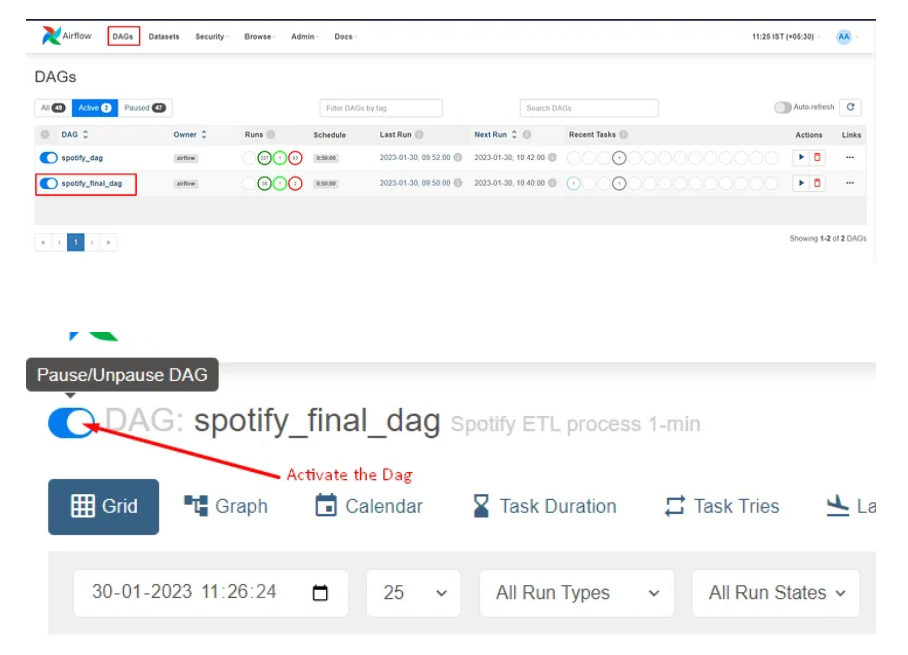
|  |
| --- |
| */\* The connection id: postgre\_sql sẽ là id chúng ta đã sử dụng.*  *Connection Type: Postgres*  *Host: Postgres*  *Schema: Spotify\_db, bạn có thể sử dụng tên riêng của mình nhưng hãy đảm bảo kiểm tra cơ sở dữ liệu phù hợp khi xác thực.*  *Login:: luồng không khí*  *Password: Airflow*  *Port: 5432\*/* |



Hình 15 Giao diện tạo kết nối

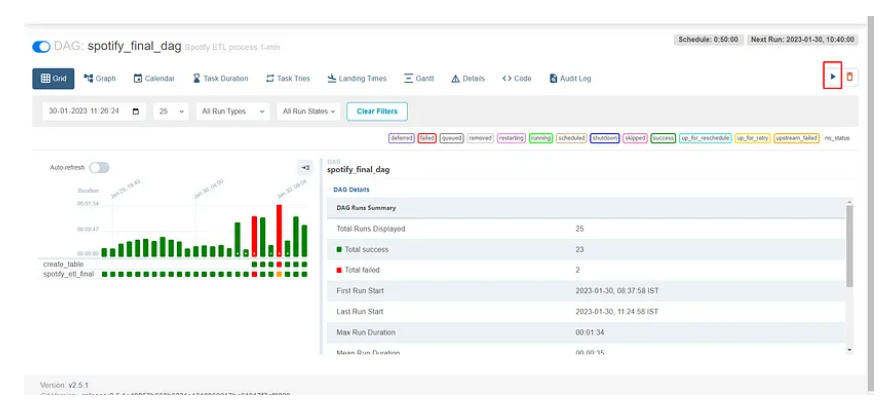
**Bước 14: Triển khai**

|  |
| --- |
| */\* Kiểm tra Dag của bạn trong phần dags\*/* |



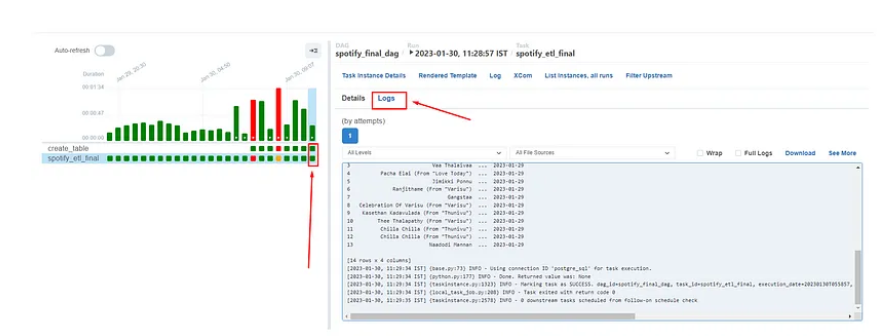
Hình 16 Giao diện kiểm tra Dags

|  |
| --- |
| */\* Sau khi kích hoạt bây giờ hãy chạy dag bằng cách kích hoạt.\*/* |



Hình 17 Kích hoạt Dags

|  |
| --- |
| */\* hoàn thành kiểm tra các bản ghi.\*/* |



Hình 18 Logs thông tin

|  |
| --- |
| */\* Bây giờ chúng ta sẽ xác thực bằng cách kết nối với cơ sở dữ liệu Postgres. Mở terminal và thực thi:* ***pgcli -h localhost -p 5432 -uflow -d Spotify\_db****. Nó sẽ nhắc nhập mật khẩu ariflow. Ở đây Spotify\_db là tên cơ sở dữ liệu hoặc lược đồ kết nối của chúng ta.\*/* |



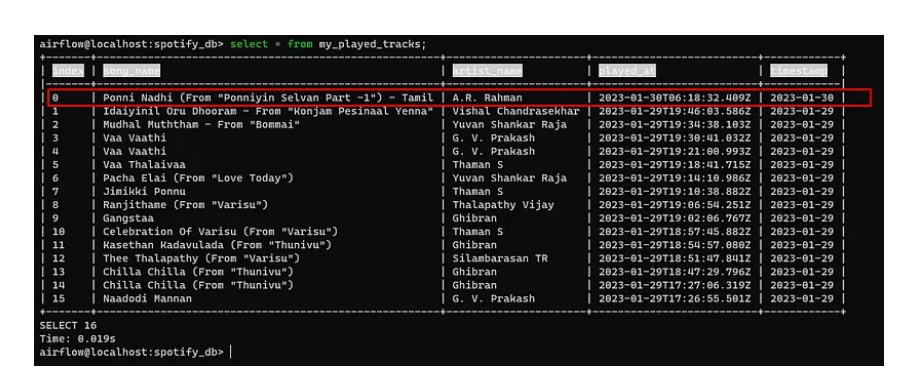
Hình 19 cửa sổ terminal cho việc xác thực

|  |
| --- |
| */\* Nhập \d để xem các bảng.\*/* |



Hình 20 Danh sách các bảng

### 2.3.4. Kết quả thực hiện



Hình 22 Kết quả thu được

# TÀI LIỆU HƯỚNG DẪN THỰC HÀNH

[1] Tài liệu: https://blog.devgenius.io/data-engineering-project-2-building-spotify-etl-using-python-and-airflow-432dd8e4ffa3

# PHỤ LỤC

**Phiếu báo cáo kết quả thực hành (Sinh viên)**

Tên bài: ……………………………………………………………………………...

Họ và tên sinh viên……………………………Mã sinh viên……………………...

………………………………………………………………..……………………...

Nhóm…………………..Lớp……………… ..Ngày…..tháng…..năm…………..

Giáo viên hướng dẫn…………………………Ca thực tập……………………….

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Nội dung  thực hành | Mức độ  hoàn thành (%) | Thời gian  hoàn thành | Đánh giá kết quả (100) |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| Thảo luận sinh viên: | | | | |

**Phiếu đánh giá kết quả thực hành (Giảng viên)**

Tên bài: ……………………………………………………………………………...

Họ và tên sinh viên……………………………Mã sinh viên……………………...

Nhóm…………………..Lớp…………………Ngày…..tháng…..năm…………..

Giáo viên hướng dẫn………………………….Ca thực tập……………………….

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thứ tự** | **Nội dung đánh giá** | **Điểm chuẩn** | **Yêu cầu** | **Ghi chú** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Tổng điểm: | | | | |
| Nhận xét giảng viên: | | | | |