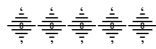


TRƯỜNG ĐẠI HỌC PHENIKAA
KHOA CÔNG NGHỆ THÔNG TIN



Đề tài

“Hệ Thống Quản Lý Đơn Hàng Phân Tán”

Course/Học phần : Ứng dụng phân tán

Class/Lớp : CSE702063-1-3-24(N05)

Instructor/Giảng viên : Thầy Phạm Kim Thành

Thành viên

Mã sinh viên

Phạm Thế Minh

22010075

Nguyễn Ngọc An

22010450

1. Giới thiệu	2
1.1. Lý do chọn đề tài	3
1.2. Mục tiêu của dự án	3
1.3. Phạm vi thực hiện	3
1.4. Công nghệ sử dụng	4
2. Kiến thức nền tảng	4
2.1. Hệ thống phân tán	4
Khái niệm	4
Đặc điểm chính	4
Ứng dụng trong dự án	4
2.2. Giới thiệu về Apache Airflow	5
Apache Airflow là gì?	5
Kiến trúc tổng quan	5
DAG và các thành phần chính	5
2.3. Lý do sử dụng Airflow trong dự án	5
2.4. Giới thiệu về Flask và vai trò trong hệ thống	6
Flask là gì?	6
Vai trò trong dự án	6
3. Phân tích và thiết kế hệ thống	6
3.1. Bài toán cần giải quyết	6
3.2. Kiến trúc tổng thể hệ thống	7
Sơ đồ kiến trúc tổng quan:	7
3.3. Thiết kế DAG trong Apache Airflow	7
1. Kiểm tra tồn kho	7
2. Chọn kho phù hợp	7
3. Gửi đơn hàng đến node đã chọn	8
4. Cập nhật trạng thái đơn hàng	8
Biểu đồ DAG minh họa (mô tả logic):	8
3.4. Thiết kế các node kho hàng (Warehouse Node)	8
3.5. Quản lý trạng thái và cơ sở dữ liệu	9
4. Triển khai và cài đặt	9
4.1. Công cụ và công nghệ sử dụng	9
4.2. Cấu trúc thư mục dự án	9
4.3. Hướng dẫn cài đặt và chạy hệ thống	10
Bước 1: Cài đặt Docker & Docker Compose	10
Bước 2: Khởi động hệ thống Airflow	11
Bước 3: Khởi động các node kho hàng	11
Bước 4: Tạo và kích hoạt DAG	12
Bước 5: Theo dõi trạng thái	12
4.4. Các vấn đề gặp phải khi cài đặt	14
5. Kết luận	14

1. Giới thiệu

1.1. Lý do chọn đề tài

Trong thời đại chuyển đổi số hiện nay, các doanh nghiệp ngày càng có nhu cầu quản lý dữ liệu và quy trình nghiệp vụ một cách tự động, chính xác và có khả năng mở rộng. Đặc biệt trong các hệ thống lớn như thương mại điện tử, việc xử lý đơn hàng nhanh chóng, ổn định và chịu lỗi tốt là một thách thức quan trọng. Việc thiết kế một hệ thống xử lý đơn hàng theo mô hình **ứng dụng phân tán** không chỉ giúp nâng cao hiệu suất mà còn tăng tính sẵn sàng và khả năng phục hồi của hệ thống.

Apache Airflow là một nền tảng mạnh mẽ, mã nguồn mở, chuyên dùng để lập lịch và điều phối luồng công việc (workflow). Với khả năng xác định DAG (Directed Acyclic Graph) linh hoạt, Airflow cho phép lập trình các chuỗi tác vụ theo thời gian, phù hợp với yêu cầu của các hệ thống xử lý dữ liệu phân tán hiện đại.

Do đó, nhóm chúng em quyết định xây dựng một hệ thống quản lý đơn hàng mô phỏng theo kiến trúc phân tán, trong đó **Apache Airflow đóng vai trò trung tâm trong việc điều phối luồng xử lý đơn hàng** giữa các node kho khác nhau. Đề tài này giúp nhóm hiểu rõ hơn về mô hình ứng dụng phân tán, cũng như cách sử dụng Airflow trong thực tế.

1.2. Mục tiêu của dự án

- Xây dựng một hệ thống xử lý đơn hàng phân tán bao gồm nhiều node kho, mỗi node có thể hoạt động độc lập.
- Sử dụng Apache Airflow để điều phối việc gửi, kiểm tra và cập nhật trạng thái đơn hàng giữa các node kho.
- Minh họa các tính chất quan trọng của hệ thống phân tán như: chịu lỗi, tính mở rộng, phân mảnh dữ liệu (sharding), và truyền thông giữa các node.
- củng cố kỹ năng triển khai API backend bằng Flask và triển khai hệ thống bằng Docker Compose.

1.3. Phạm vi thực hiện

- Triển khai hệ thống gồm:
 - Một DAG quản lý luồng công việc đơn hàng (Airflow).
 - Nhiều node kho được triển khai dưới dạng Flask API.
- Tập trung vào **luồng xử lý đơn hàng**, chưa bao gồm các chức năng nâng cao như giao diện người dùng hay phân quyền người dùng.

- Cài đặt, kiểm thử và chạy hệ thống trên môi trường máy ảo hoặc Docker Compose.
- Viết tài liệu hướng dẫn và báo cáo chi tiết về kiến trúc, luồng xử lý và cách tích hợp Airflow vào hệ thống.

1.4. Công nghệ sử dụng

Công nghệ	Mục đích sử dụng
Apache Airflow	Điều phối các DAG xử lý đơn hàng
Python	Ngôn ngữ lập trình chính
Flask	Xây dựng các node kho dưới dạng RESTful API
Docker & Docker Compose	Triển khai môi trường container hóa cho Airflow và các node
PostgreSQL	Hệ quản trị cơ sở dữ liệu
GitHub	Quản lý mã nguồn và cộng tác nhóm

2. Kiến thức nền tảng

2.1. Hệ thống phân tán

Khái niệm

Hệ thống phân tán là một tập hợp các máy tính độc lập (thường được gọi là node) phối hợp với nhau để người dùng cuối nhìn thấy như một hệ thống duy nhất. Mỗi node trong hệ thống có thể xử lý dữ liệu, giao tiếp với các node khác và thực hiện một phần công việc nhất định.

Đặc điểm chính

- **Tính trong suốt (Transparency):** Người dùng không cần biết các thành phần bên dưới phân tán như thế nào.
- **Khả năng mở rộng (Scalability):** Hệ thống có thể mở rộng bằng cách thêm node mới mà không ảnh hưởng đến hệ thống hiện tại.
- **Khả năng chịu lỗi (Fault-tolerance):** Hệ thống có thể tiếp tục hoạt động ngay cả khi một số node gặp sự cố.
- **Cân bằng tải (Load balancing):** Công việc được phân phối hợp lý giữa các node để tối ưu hiệu suất.

Ứng dụng trong dự án

Trong dự án này, mỗi **node kho hàng** (warehouse node) là một thực thể độc lập, có thể nhận và xử lý đơn hàng. Hệ thống sử dụng Apache Airflow để điều phối quá trình xử lý giữa các node này, đảm bảo khả năng mở rộng và chịu lỗi tốt.

2.2. Giới thiệu về Apache Airflow

Apache Airflow là gì?

Apache Airflow là một nền tảng mã nguồn mở dùng để lập lịch và điều phối các luồng công việc phức tạp. Airflow cho phép định nghĩa một **DAG (Directed Acyclic Graph)** – biểu đồ có hướng và không chu trình – để mô tả quan hệ phụ thuộc giữa các tác vụ (task).

Kiến trúc tổng quan

Thành phần	Mô tả
Scheduler	Lên lịch và kích hoạt các task dựa trên DAG đã định nghĩa.
Executor	Thực thi các task đã được kích hoạt.
Web Server	Giao diện web cho phép theo dõi DAG, log, trạng thái task,...
Metadata DB	Lưu trữ thông tin DAG, trạng thái task,...
Worker	(Tuỳ thuộc Executor) thực hiện công việc thực tế.

DAG và các thành phần chính

- **DAG (Directed Acyclic Graph):** Xác định luồng công việc gồm nhiều tác vụ, có thứ tự thực thi rõ ràng.
 - **Task:** Một đơn vị công việc nhỏ, ví dụ như gọi API, xử lý dữ liệu,...
 - **Operator:** Mẫu thực thi của task, ví dụ `PythonOperator`, `BashOperator`, `HttpOperator`,...
-

2.3. Lý do sử dụng Airflow trong dự án

Dự án hướng đến mô hình xử lý đơn hàng phân tán, gồm nhiều bước như gửi yêu cầu, xác nhận đơn, cập nhật kho,... Các bước này phụ thuộc lẫn nhau và cần được thực thi theo thứ tự hợp lý. Airflow là công cụ lý tưởng cho mục tiêu này vì:

- Cho phép định nghĩa rõ ràng luồng xử lý thông qua DAG.

- Theo dõi trạng thái của từng task trong quá trình xử lý.
 - Dễ tích hợp với các API node kho thông qua [HttpOperator](#) hoặc [PythonOperator](#).
 - Cấu hình dễ dàng bằng Python, thuận tiện cho việc mở rộng.
 - Giao diện web trực quan giúp dễ dàng theo dõi trạng thái hệ thống.
-

2.4. Giới thiệu về Flask và vai trò trong hệ thống

Flask là gì?

Flask là một micro web framework viết bằng Python, rất nhẹ và linh hoạt, thích hợp cho việc xây dựng các API đơn giản và nhanh chóng.

Vai trò trong dự án

Trong dự án này, mỗi **node kho hàng** được triển khai như một dịch vụ Flask API độc lập. Mỗi node cung cấp các endpoint như:

- Nhận đơn hàng mới ([/orders](#))
- Kiểm tra tồn kho ([/check-stock](#))
- Cập nhật trạng thái đơn hàng ([/update-status](#))

Airflow sẽ gọi các API này như là các task trong DAG để điều phối toàn bộ quá trình xử lý đơn hàng.

3. Phân tích và thiết kế hệ thống

3.1. Bài toán cần giải quyết

Hệ thống cần giải quyết bài toán quản lý đơn hàng trong một môi trường phân tán, nơi có nhiều kho hàng (node) hoạt động độc lập. Khi một đơn hàng mới được tạo ra, hệ thống sẽ:

1. Gửi yêu cầu kiểm tra tồn kho đến tất cả các node.
2. Tìm node phù hợp (còn đủ hàng).
3. Gửi đơn hàng đến node đó để xử lý.

4. Cập nhật trạng thái đơn hàng sau khi xử lý xong.

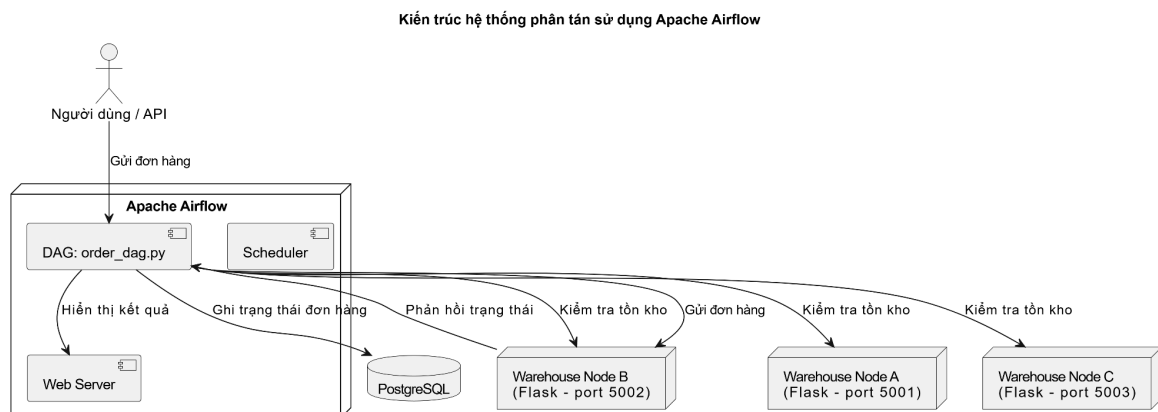
Toàn bộ quá trình này cần được thực hiện theo đúng thứ tự, đồng bộ và có khả năng mở rộng, chịu lỗi.

3.2. Kiến trúc tổng thể hệ thống

Hệ thống bao gồm các thành phần chính:

- **Apache Airflow**: Điều phối luồng công việc xử lý đơn hàng.
- **Các node kho (Warehouse nodes)**: Mỗi node là một dịch vụ Flask API độc lập.
- **PostgreSQL**: Lưu trữ trạng thái và thông tin đơn hàng.
- **Docker Compose**: Quản lý triển khai toàn bộ hệ thống trong môi trường container.

Sơ đồ kiến trúc tổng quan:



3.3. Thiết kế DAG trong Apache Airflow

Luồng DAG chính được định nghĩa trong file `order_dag.py`. DAG này thực hiện 4 bước chính:

1. Kiểm tra tồn kho

- Gửi yêu cầu kiểm tra số lượng hàng đến tất cả các node.
- Các node phản hồi số lượng hiện có.

2. Chọn kho phù hợp

- Dựa vào kết quả kiểm tra, Airflow chọn node còn đủ hàng.
- Nếu không có node nào đủ, workflow dừng và thông báo lỗi.

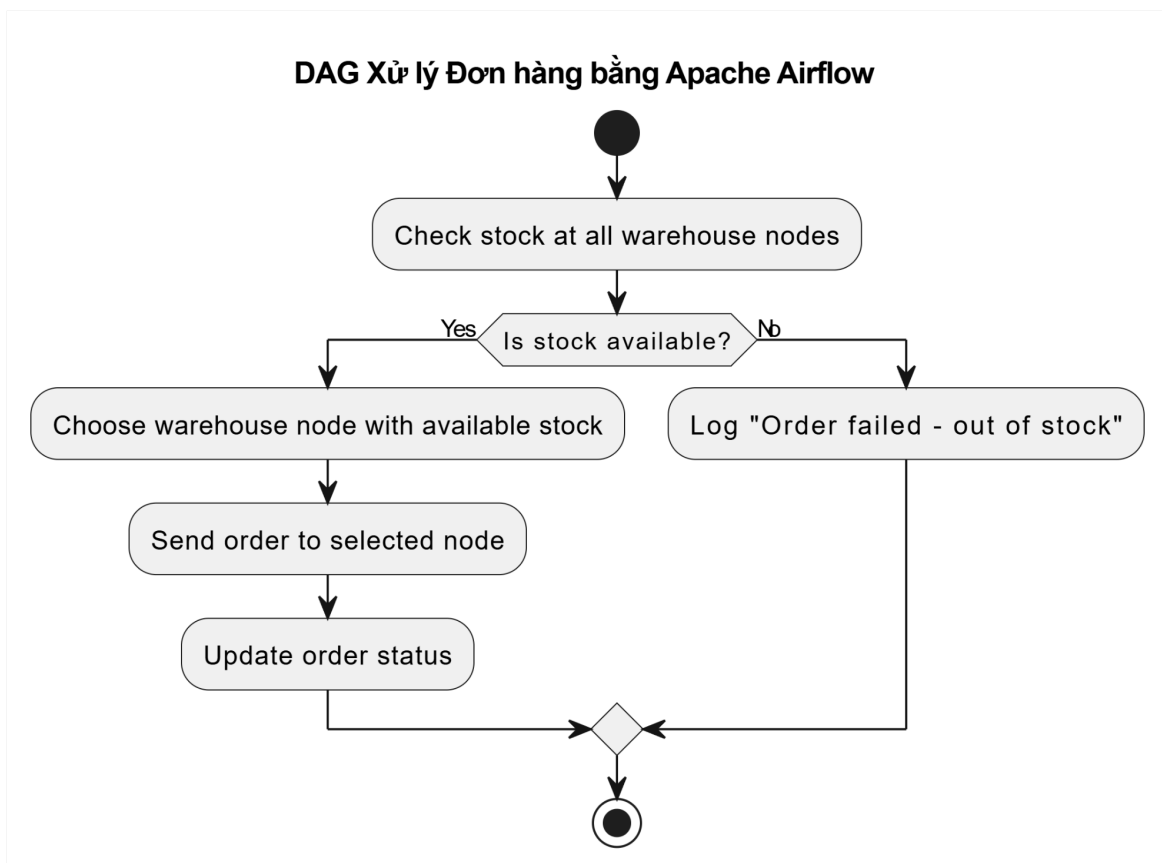
3. Gửi đơn hàng đến node đã chọn

- Airflow gửi đơn hàng đến node được chọn để tiến hành xử lý đơn.

4. Cập nhật trạng thái đơn hàng

- Sau khi node xử lý xong, Airflow cập nhật trạng thái đơn hàng trong hệ thống.

Biểu đồ DAG minh họa (mô tả logic):



3.4. Thiết kế các node kho hàng (Warehouse Node)

Mỗi node được triển khai bằng Flask, gồm các API:

- **POST /check-stock**: Trả về số lượng còn lại của một sản phẩm.
- **POST /order**: Nhận đơn hàng mới và xử lý.

- **POST /update-status**: Cập nhật trạng thái đơn hàng.

Node hoạt động độc lập và có thể chạy trên cổng riêng biệt trong Docker Compose, ví dụ:

- Node A: **localhost:5001**
- Node B: **localhost:5002**
- Node C: **localhost:5003**

3.5. Quản lý trạng thái và cơ sở dữ liệu

- Airflow sử dụng PostgreSQL để lưu metadata DAG.
- Trạng thái đơn hàng được ghi nhận lại (trong thực tế có thể mở rộng lưu vào MongoDB, Redis,...).
- Cơ chế log task cũng được quản lý bởi Airflow, có thể xem chi tiết từng lần chạy DAG.

4. Triển khai và cài đặt

4.1. Công cụ và công nghệ sử dụng

Thành phần	Công nghệ sử dụng
Điều phối công việc	Apache Airflow
API node kho hàng	Python Flask
Cơ sở dữ liệu	PostgreSQL
Triển khai container	Docker, Docker Compose
Hệ điều hành	Linux hoặc Windows có Docker
Giao tiếp giữa các thành phần	HTTP REST API

4.2. Cấu trúc thư mục dự án

```

Ung_dung_phan_tan/
├─ airflow/
|   └─ dags/

```

```

|   |   └─ order_dag.py           # DAG định nghĩa luồng công việc
chính
|   └─ docker-compose.yml         # Triển khai Airflow
└─ node/
    │   └─ node1/
    │       └─ main.py             # Flask API cho Node 1
    │   └─ node2/
    │       └─ main.py             # Flask API cho Node 2
    │   └─ node3/
    │       └─ main.py             # Flask API cho Node 3
    └─ docker-compose.yml         # Khởi chạy các node kho bằng Flask
└─ README.md                     # Tài liệu mô tả dự án

```

4.3. Hướng dẫn cài đặt và chạy hệ thống

Bước 1: Cài đặt Docker & Docker Compose

- Yêu cầu Docker và Docker Compose đã được cài đặt sẵn:
 - Docker Desktop

Kiểm tra:

```
docker -v
```

```
docker-compose -v
```

○

```

PS D:\bài trên lớp\New folder\UDPT\quanlikho\distributed_order_system\warehouse_node> docker-compose up -d
>>
time="2025-06-02T13:36:07+07:00" level=warning msg="D:\\bài trên lớp\\New folder\\UDPT\\quanlikho\\distributed_order_syst
em\\warehouse_node\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoi
d potential confusion"
[+] Running 3/3
 ✓ Container warehouse_node-node-hn-1   Started      0.6s
 ✓ Container warehouse_node-node-hcm-1   Started      0.6s
 ✓ Container warehouse_node-node-dn-1    Started      0.6s
PS D:\bài trên lớp\New folder\UDPT\quanlikho\distributed_order_system\warehouse_node>

```

```
PS D:\bài trên lớp\New folder\UDPT\quanlikho\distributed_order_system\airflow> docker-compose up -d
>>
time="2025-06-02T13:35:48+07:00" level=warning msg="D:\\bài trên lớp\\New folder\\UDPT\\quanlikho\\distributed_order_system\\airflow\\docker-c
ompose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 4/4
 ✓ Container airflow-postgres-1      Started      1.0s
 ✓ Container airflow-airflow-scheduler-1 Started      1.3s
 ✓ Container airflow-airflow-init-1  Started      1.3s
 ✓ Container airflow-airflow-webserver-1 Started      1.5s
PS D:\bài trên lớp\New folder\UDPT\quanlikho\distributed_order_system\airflow> |
```

Bước 2: Khởi động hệ thống Airflow

Di chuyển vào thư mục `airflow`:

```
cd airflow
```

Dùng Docker Compose để khởi chạy Airflow:

```
docker-compose up -d
```

1. Truy cập giao diện Airflow:

- URL: <http://localhost:8080>
- Tài khoản mặc định (nếu chưa đổi):
 - Username: `airflow`
 - Password: `airflow`

Bước 3: Khởi động các node kho hàng

Di chuyển sang thư mục `node`:

```
cd ../node
```

Dùng Docker Compose để khởi động 3 node Flask:

`docker-compose up -d`

1. Mỗi node sẽ chạy trên cổng khác nhau:


- Node 1: <http://localhost:5001>
- Node 2: <http://localhost:5002>
- Node 3: <http://localhost:5003>

Bước 4: Tạo và kích hoạt DAG

1. Vào giao diện Airflow tại <http://localhost:8080>
2. Tìm DAG có tên `order_dag`
3. Bật DAG → Click **Trigger DAG** để tạo luồng xử lý đơn hàng
4. Theo dõi từng bước thực hiện trong giao diện trực quan của Airflow

Bước 5: Theo dõi trạng thái

- Dùng tab **Graph View** hoặc **Tree View** trong Airflow để kiểm tra:
 - Node nào được chọn để xử lý đơn hàng.
 - Trạng thái hoàn thành của từng task trong DAG.
 - Log chi tiết tại từng bước.



DAGs

Cluster Activity


Datasets


Security

Browse

Admin

Docs

13:36 +07 (+07:00) 

Triggerred order_workflow, it should start any moment now. 

DAGs

All **1**

Active **1**


Paused **0**




Running **0**

Failed **0**

Filter DAGs by tag

Search DAGs

Auto-refresh 

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
 order_workflow <div>order warehouse</div>	airflow	<div><div>3</div><div>1</div><div>4</div></div>	None	2025-06-02, 13:36:48		<div><div>2</div><div>1</div></div>	  ...	

«


<

1

>

»

Showing 1-1 of 1 DAGs



DAGs

Cluster Activity


Datasets

Security

Browse

Admin

Docs

12:18 +07 (+07:00) 

DAGs

All **1**

Active **1**


Paused **0**




Running **1**

Failed **0**

Filter DAGs by tag

Search DAGs

Auto-refresh 

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
 order_workflow <div>order warehouse</div>	airflow	<div><div>1</div><div>3</div></div>	None	2025-06-02, 12:15:06		<div><div>2</div><div>1</div></div>	  ...	

«


<

1

>

»

Showing 1-1 of 1 DAGs



DAGs

Cluster Activity


Datasets

Security

Browse

Admin

Docs

13:36 +07 (+07:00) 

DAGs

All **1**

Active **1**


Paused **0**




Running **0**

Failed **0**

Filter DAGs by tag

Search DAGs

Auto-refresh 

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
 order_workflow <div>order warehouse</div>	airflow	<div><div>3</div><div>4</div></div>	None	2025-06-02, 12:33:16		<div><div>3</div></div>	  ...	

«

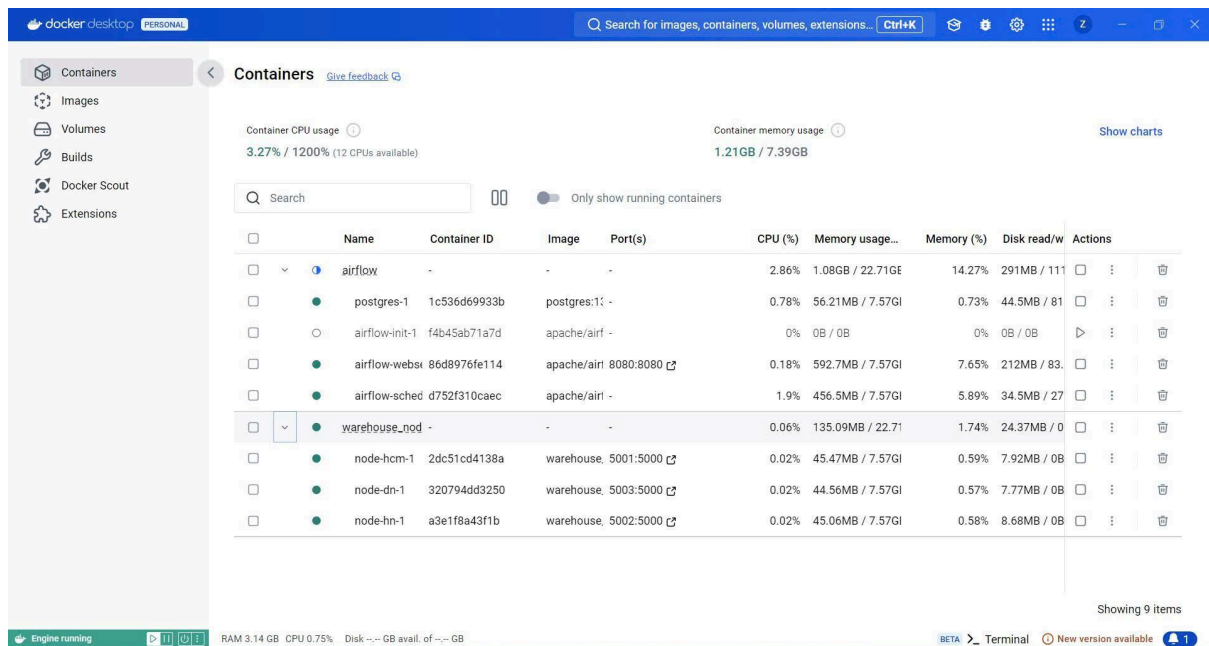
<

1

>

»

Showing 1-1 of 1 DAGs



4.4. Các vấn đề gặp phải khi cài đặt

Vấn đề	Cách khắc phục
Cổng bị chiếm khi chạy node	Kiểm tra tiến trình đang dùng cổng bằng <code>lsof -i :PORT</code>
DAG không hiện trong giao diện	Kiểm tra DAG có được đặt đúng trong <code>dags/</code> và cú pháp đúng
Flask API lỗi 500	Kiểm tra log trong container của node đó bằng <code>docker logs</code>

5. Kết luận

Dự án đã xây dựng thành công một hệ thống ứng dụng phân tán đơn giản để xử lý đơn hàng, với khả năng điều phối công việc dựa trên **Apache Airflow** và giao tiếp giữa các thành phần thông qua **RESTful API**. Việc triển khai các node kho hàng dưới dạng các **dịch vụ Flask độc lập** giúp hệ thống dễ dàng mở rộng, bảo trì và thay thế.

Thông qua quá trình thực hiện, nhóm đã đạt được các mục tiêu chính sau:

- **Hiểu và vận dụng kiến trúc phân tán** trong thực tiễn thông qua việc chia hệ thống thành các thành phần độc lập, giao tiếp qua mạng.
- **Áp dụng Apache Airflow** để định nghĩa và điều phối quy trình công việc theo dạng DAG.

- **Sử dụng Docker Compose** để triển khai hệ thống đa thành phần một cách dễ dàng và tái sử dụng được trên nhiều máy.
- **Thiết kế cơ sở dữ liệu và mô hình tương tác hợp lý** giữa Airflow và các node kho hàng.

Mặc dù hệ thống hiện tại mới chỉ ở mức cơ bản, nó hoàn toàn có thể được mở rộng để phục vụ các nhu cầu lớn hơn, chẳng hạn như:

- Tăng số lượng node kho hàng.
- Áp dụng thuật toán lựa chọn node nâng cao (load balancing, geographic location...).
- Tích hợp thêm hệ thống hàng đợi (RabbitMQ, Kafka) để nâng cao hiệu suất xử lý.
- Triển khai thực tế trên môi trường cloud (AWS, GCP, Azure) với Kubernetes.

Dự án không chỉ giúp nhóm tiếp cận sâu hơn với các công nghệ phân tán, mà còn mang lại trải nghiệm thực tiễn trong việc tổ chức, thiết kế và triển khai một hệ thống hoàn chỉnh theo hướng hiện đại.