# Mục lục

Contents

[Mục lục 1](#_Toc119255114)

[I. TỔNG QUAN 2](#_Toc119255115)

[1. Định nghĩa về AirFlow 2](#_Toc119255116)

[2. Quy trình làm việc 2](#_Toc119255117)

[3. Kiến trúc 3](#_Toc119255118)

[4. Cài đặt 4](#_Toc119255119)

[4.1. Cài đặt Airflow bằng pip 4](#_Toc119255120)

[4.2. Cài đặt Airflow bằng Docker 5](#_Toc119255121)

[5. Chạy Airflow 6](#_Toc119255122)

[5.1. CLI 6](#_Toc119255123)

[5.2. Web Interface 7](#_Toc119255124)

[5.3. RestAPI 7](#_Toc119255125)

[6. DAGs 7](#_Toc119255126)

[6.1. Context manager 8](#_Toc119255127)

[6.2. Standard constructor 8](#_Toc119255128)

[6.3. Decorator 8](#_Toc119255129)

[6.4. Sự khác biệt giữa Context manager và Decorator 8](#_Toc119255130)

[7. Xác định thứ tự xử lý của các tasks 9](#_Toc119255131)

[8. Cách viết kết hợp Decorator và Context manager 10](#_Toc119255132)

[9. cài đặt pip package cho task 11](#_Toc119255133)

[10. Chạy File DAG 12](#_Toc119255134)

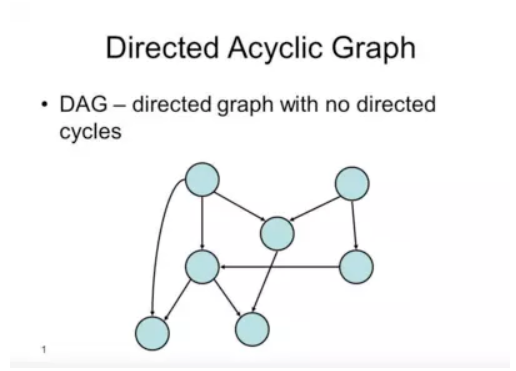
# TỔNG QUAN

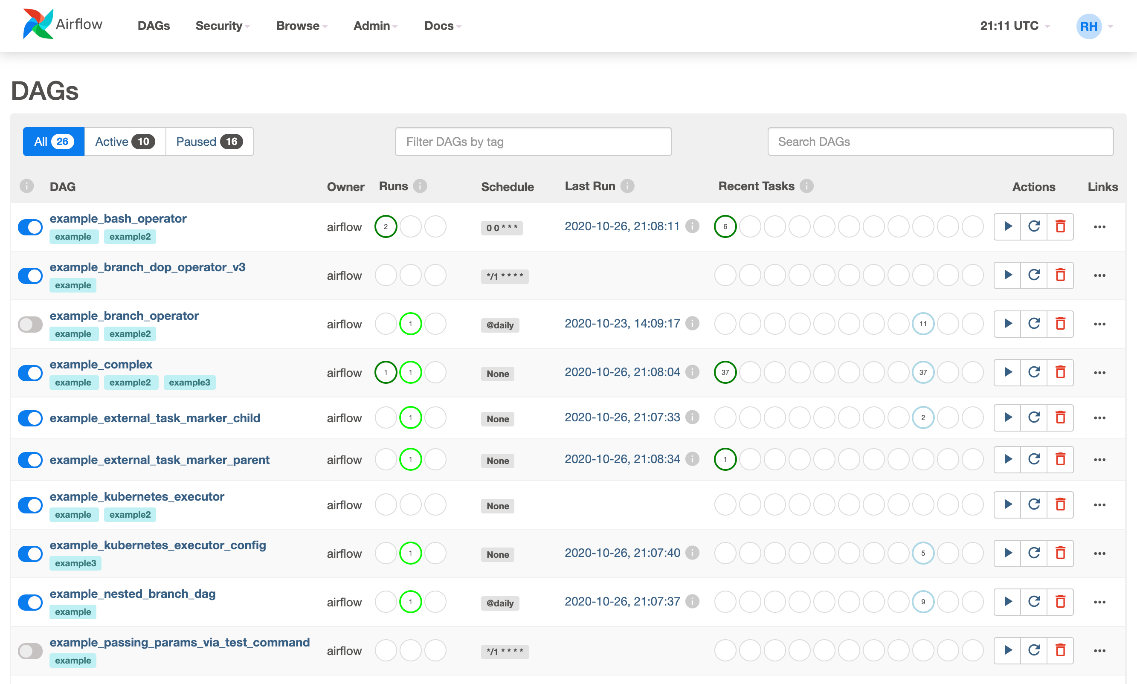
## Định nghĩa về AirFlow

* **Apache AirFlow** là một nền tảng mã nguồn mở để phát triển, lập kế hoạch, và giám sát quy trình làm việc theo định hướng hàng loạt. Khung Python có thể mở rộng của Airflow cho phép bạn xây dựng quy trình làm việc kết nối với hầu hết mọi công nghệ. Giao diện web giúp quản lý trạng thái dòng công việc của bạn. Luồng không khí là Có thể triển khai theo nhiều cách, thay đổi từ một quy trình duy nhất trên máy tính xách tay của bạn đến thiết lập phân tán để hỗ trợ ngay cả quy trình làm việc lớn nhất.

## Quy trình làm việc

* Airflow có thể tự động hóa quy trình công việc của bạn bằng DAGs ( Directed Acyclic Graphs ).



Airflow sẽ thực thi các phần công việc của bạn thông qua các worker mà vẫn đảm bảo các dependencies

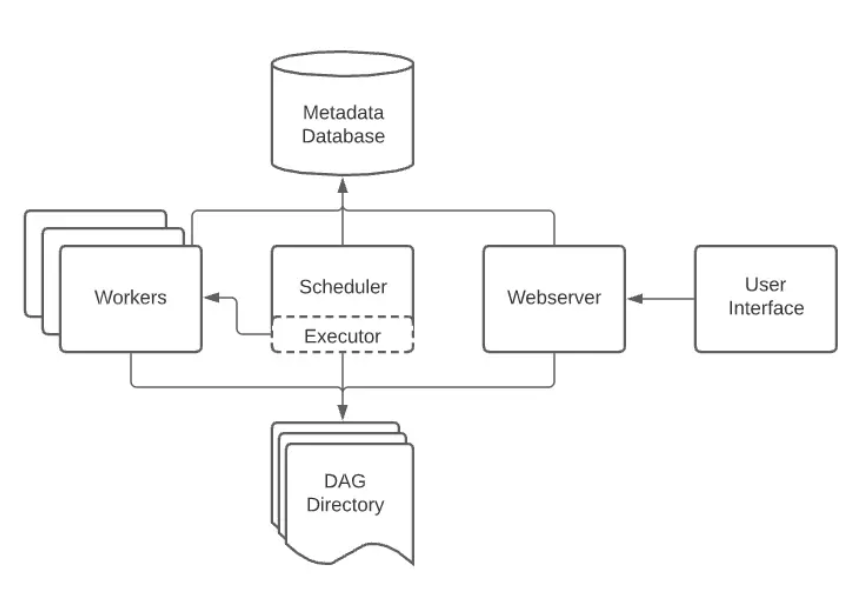
Nguyên tắc của Airflow:

* Tính năng động ( Dynamic ) : Airflow pipeline được config bằng code Python, cho phép bạn thay đổi code dễ dàng để tùy biến luồng làm việc của bạn.
* Tính tăng trưởng ( Scalable ) : Ví dụ đơn giản là bạn có thể mở rộng các task về xử lý dữ liệu để tiết kiệm thời gian
* Tính gọn gàng ( Elegant ) : code gọn gàng, ngăn nắp, rõ ràng giúp bạn đọc hiểu code nhanh chóng.
* Tính mở rộng ( Extensible ) : Bạn có thể thêm thắt thư viện, modules, packages, ... phù hợp với môi trường của bạn

## Kiến trúc

Airflow bao gồm các components sau:

* Scheduler: chạy workflow ( luồng công việc ) , gửi các tasks tới executor
* Executor: quản lý các workers, xử lý các tác vụ đang chạy
* Webserver: giao diện web cho phép kiểm tra, kích hoạt, sửa lỗi các tasks và DAGs
* Folder of DAG files: đọc bởi scheduler và executor ( với mọi worker mà executor có )
* Metadata database: nơi lưu trạng thái của scheduler, executor, webserver



## Cài đặt

Airflow documentation đề cập tới khá nhiều phương pháp để cài đặt airflow, nhưng để tiện thử nghiệm, tôi sẽ dùng docker-compose, cài đặt airflow dưới dạng container. Còn trong môi trường sản phẩm, bạn có thể dùng Kubernetes và Helm charts để config sản phẩm theo ý mình.

Nếu không quen docker hay container bận vẫn có thể cài airflow bằng pip install package.

### Cài đặt Airflow bằng pip

Trước hết chúng ta cần cài đặt dependencies của Linux

|  |
| --- |
| sudo apt-get install -y --no-install-recommends \  freetds-bin \  krb5-user \  ldap-utils \  libffi6 \  libsasl2-2 \  libsasl2-modules \  libssl1.1 \  locales \  lsb-release \  sasl2-bin \  sqlite3 \  unixodbc |

Setup nơi chưa AirFlow, có thể thay đổi đường dẫn theo yêu cầu:

|  |
| --- |
| export AIRFLOW\_HOME=~/airflow |

Assign 3 biến environment: AIRFLOW\_VERSION, PYTHON\_VERSION và CONSTRANT\_URL

|  |
| --- |
| AIRFLOW\_VERSION=2.3.3  PYTHON\_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"  CONSTRAINT\_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW\_VERSION}/constraints-${PYTHON\_VERSION}.txt" |

Cài đặt Airflow bằng pip package

|  |
| --- |
| pip install "apache-airflow==${AIRFLOW\_VERSION}" --constraint "${CONSTRAINT\_URL}" |

Sau khi cài đặt xong khởi tạo database, user cũng như khởi chạy các components của airflow

|  |
| --- |
| airflow standalone |

Mở trình duyệt lên, paste localhost:8080 để xem thành quả.

### Cài đặt Airflow bằng Docker

Vì là dùng docker-compose để cài đặt nên trước hết cần cài docker compose plugin

|  |
| --- |
| sudo curl -L "https://github.com/docker/compose/releases/download/v2.9.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  sudo chmod +x /usr/local/bin/docker-compose |

Kiểm tra phiên bản

|  |
| --- |
| docker-compose --version |

Sau khi cài xong docker compose, tải file docker-compose yaml của airflow

|  |
| --- |
| curl -LfO 'https://airflow.apache.org/docs/apache-airflow/2.3.3/docker-compose.yaml' |

Trong file yaml nàu có chưa các servives:

* airflow-scheduler: giám sát các tasks cũng như chạy tasks khi đã có đủ dependencies
* airflow-webserver: webserver có local domain http://localhost:8080
* airflow-worker: các worker chạy các tasks theo lệnh của scheduler
* airflow-init: dịch vụ khởi tạo ban đầu ( tạo account, migrate database, ... )
* postgres: cơ sở dữ liệu
* redis: cầu nối truyền dẫn các lệnh từ scheduler tới worker

Nếu muốn cài thêm một số thư viện python hoặc nâng cấp airflow providers thì có thể điều chỉnh file doker-compose yaml vừa tải ở trên

Giả sử trong file docker-compose.yaml bên trên, dùng image apache/airflow:latest nên thay đổi một line trong file:

|  |
| --- |
| image: ${AIRFLOW\_IMAGE\_NAME:-apache/airflow:|version|}  =>  image: ${AIRFLOW\_IMAGE\_NAME:-apache/airflow:latest} |

Khởi tạo môi trường:

Trước khi chạy Airflow lần đầu, bạn cần phải chuẩn bị môi trường: taoko files, folers cần thiết, khởi tạo cở sở dữ liệu

|  |
| --- |
| mkdir -p ./dags ./logs ./plugins  echo -e "AIRFLOW\_UID=$(id -u)" > .env |

* ./dags : nơi chứa files DAG
* ./logs : nơi chứa log của executor và scheduler
* ./plugins : nơi chứa các plugin tùy chỉnh của bạn

Ps: Ở một số hệ điều hành, nếu không set biến AIRFLOW\_UID sẽ hiện lên cảnh báo, bạn có thể phớt lờ chúng đi. Thay vào đó, bạn có thể tạo 1 file .evn trong cùng folder docker-compose.yaml với nội dung:

|  |
| --- |
| AIRFLOW\_UID=50000 |

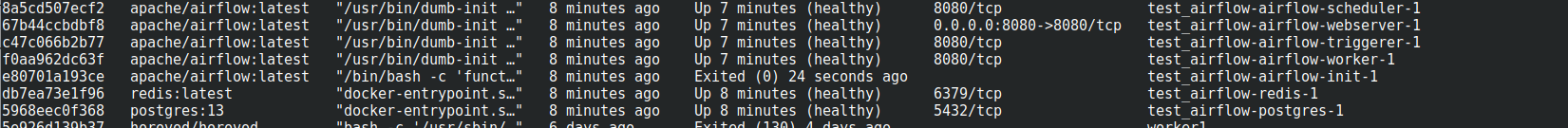
Với mọi hệ thống, bạn cần khởi tạo cơ sở dữ liệu và tài khoản trước

|  |
| --- |
| docker-compose up airflow-init |

## Chạy Airflow

Sau khi khởi tạo cơ sở dữ liệu cũng như tài khoản airflow, giờ là lúc chạy framework này. Trong folder chưa file docker-compose.yaml của airflow, chạy lệnh

|  |
| --- |
| docker-compose up |



Ok, Airflow container đã chạy và tôi có thể sử dụng các dịch vụ của framework này bằng 3 cách:

* CLI
* Web interface
* RestAPI

Xóa container và image nếu thử nghiệm xong tránh nặng cho máy:

|  |
| --- |
| docker-compose down --volumes --rmi all |

Hoặc chỉ xóa mỗi container

|  |
| --- |
| docker-compose down --volumes --remove-orphans |

### CLI

Kiểm tra thông tin của Airflow thông qua dịch vụ **airflow info** bằng lệnh docker

|  |
| --- |
| docker-compose run airflow-worker airflow info |

Nhưng với cách này sẽ phải tạo thêm instance docker nên không khuyến khích dùng docker CLI

Ngoài docker CLI ra thì bạn cũng có thể dùng một file shell để chạy lệnh Airflow.Chú ý là file shell mà bạn tải về phải có cùng phiên bản với Airflow container đang dùng.

|  |
| --- |
| curl -LfO 'https://airflow.apache.org/docs/apache-airflow/2.3.3/airflow.sh'  chmod +x airflow.sh |

Chạy câu lệnh trong folder có chứa file shell

|  |
| --- |
| ./airflow.sh info |

Hoặc chọc thẳng vào Airflow container

|  |
| --- |
| ./airflow.sh bash |

### Web Interface

Sau khi chạy Airflow container sẽ cung cấp một giao diện web: <http://localhost:8080>

Tài khoản mặc định: airflow

Mật khẩu mặc định: airflow

Nếu không muốn dùng tài khoản mặc định thì có thể thêm 2 env này vào file docker-compose.yaml

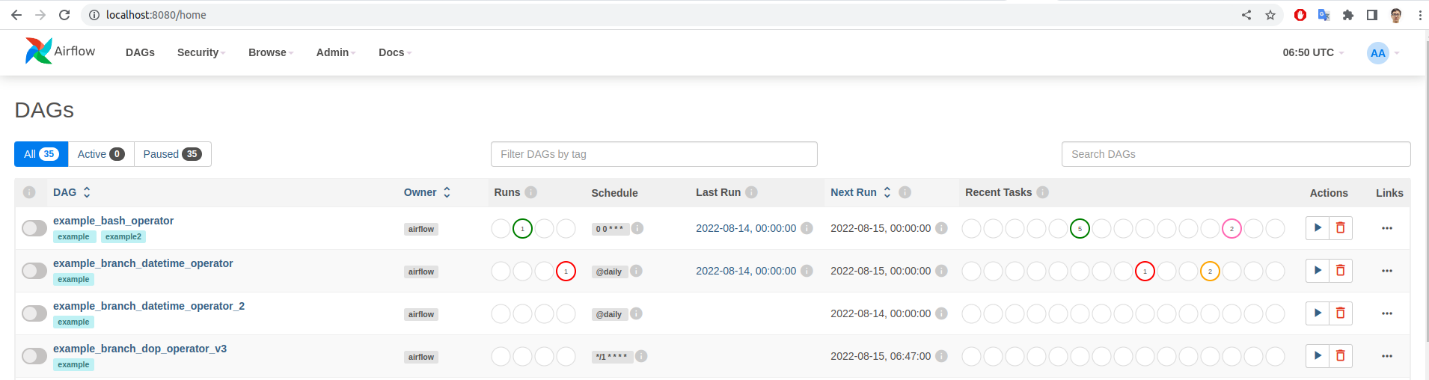
|  |
| --- |
| \_AIRFLOW\_WWW\_USER\_USERNAME  \_AIRFLOW\_WWW\_USER\_PASSWORD |

### RestAPI

Airflow cũng cung cấp API Endpoint, có thể dùng postman để test

|  |
| --- |
| ENDPOINT\_URL="http://localhost:8080/"  curl -X GET \  --user "airflow:airflow" \  "${ENDPOINT\_URL}/api/v1/pools" |

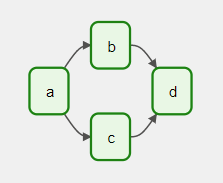
Sau khi đăng nhập tài khoản airflow trên web <http://localhost:8080> sẽ hiện ra một danh sách các DAG



P/s: Nếu không muốn WebUI chứa các file DAG mẫu mà nhà phát triển cung cấp thì có thể thay đổi env AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'false' trong file docker-compose.yaml .

## DAGs

1 DAG (Directed Acyclic Graph) sẽ kết nối các tasks, xây dựng liên kết, tự động chạy task, ...



Hình bên trên bao gồm 4 task: A, B, C, D có mối liên kết với nhau và chạy theo thứ tự mà người dùng quy định: a -> b, c -> d. Nếu task a chết thì sẽ dẫn đến task b, c, d chết theo. Bạn cũng có thể tùy chỉnh thời gian để các tasks này chạy vào thời điểm mong muốn. Bạn có thể yên tâm một điều là DAG sẽ không quan tâm đến code của bạn, mục tiêu của nó chỉ là workflow mà thôi.

Có 3 cách để khai báo DAG trong code của bạn:

* Context manager
* Standard constructor
* Decorator

### Context manager

Với cách này sẽ bọc code trong DAG bằng with

|  |
| --- |
| with DAG(  "my\_dag\_name", start\_date=pendulum.datetime(2021, 1, 1, tz="UTC"),  schedule\_interval="@daily", catchup=False  ) as dag:  op = EmptyOperator(task\_id="task") |

### Standard constructor

Với cách này, sẽ khai báo một constructer gọi tới class DAG

|  |
| --- |
| my\_dag = DAG("my\_dag\_name", start\_date=pendulum.datetime(2021, 1, 1, tz="UTC"),  schedule\_interval="@daily", catchup=False)  op = EmptyOperator(task\_id="task", dag=my\_dag) |

### Decorator

|  |
| --- |
| @dag(start\_date=pendulum.datetime(2021, 1, 1, tz="UTC"),  schedule\_interval="@daily", catchup=False)  def generate\_dag():  op = EmptyOperator(task\_id="task")  dag = generate\_dag() |

### Sự khác biệt giữa Context manager và Decorator

Nếu để chọn cách nào trong 3 cách trên thì tôi sẽ chọn viết DAG bằng decorator. Tại sao ? Bởi vì cách viết của context manager và standard constructor phức tạp, dài dòng và mơ hồ. Chính vì thế các phiên bản của Airflow từ 2.x trở lên cung cấp thêm cách viết bằng decorator cho người dùng dễ dàng sử dụng

1. Import modules: 2 cách viết khác nhau sẽ có 2 cách import thư viện khác nhau
   * Context manager:

|  |
| --- |
| from airflow import DAG  from airflow.operators.python import PythonOperator |

* + Decorator:

|  |
| --- |
| from airflow.decorators import dag, task |

1. Khởi tạo DAG:
   * Context manager: bọc tất cả các tasks với DAG bằng keyword **with**
   * Decorator: khởi tạo @dag đầu file

Ví dụ :

|  |
| --- |
| @dag(  start\_date=datetime.now(),  schedule\_interval=None,  catchup=False,  tags=['example'],  ...  ) |

Start\_date: thời gian bắt đầu

Schedule\_interval: lịch trình chạy ( @once, @hourly, @daily, @weekly, @monthly, @yearly )

catchup: nếu bạn có start\_date, end\_date ( optional ) và schedule\_interval thì khi bạn để giá trị của catchup là True thì schedule\_interval sẽ không bị giới hạn và thực thi tasks tức thì.

1. Khởi tạo tasks trong DAG: các tasks ở đây là extract, transform, loading
   * Context manager: Truyền các function ETL vào PythonOperator ( Operator có trách nhiệm xử lý tasks trong phần kiến trúc đề cập bên trên) và gán 1 biến bất kỳ

|  |
| --- |
| extract\_task = PythonOperator(  task\_id='extract',  python\_callable=extract,  ) |

* + Decorator: đơn giản dùng như python decorator

|  |
| --- |
| @task()  def extract():  ... |

1. Viết doc cho function
   * Context manager: từ các biến tượng trưng cho các task gọi hàm doc\_md
   * Decorator: không thay đổi gì, viết doc trong hàm như bình thường
2. Xác định thứ tự xử lý của các tasks:
   * Context manager: extract\_task >> transform\_task >> load\_task
   * Decorator: cách viết này sẽ rõ ràng hơn cho người đọc

|  |
| --- |
| order\_data = extract()  order\_summary = transform(order\_data)  load(order\_summary["total\_order\_value"]) |

## 7. Xác định thứ tự xử lý của các tasks

Xác định thứ tự xử lý của các tasks ( Task Dependencies ): 1 task không phải lúc nào cũng hoạt động 1 mình, sẽ có các tasks phụ thuộc vào nó hoặc nó phụ thuộc vào. Vì vậy, một nhiệm vụ quan trọng của DAG là tạo mối quan hệ của các tasks.

* Cách 1: dùng toán tử << và >>

|  |
| --- |
| first\_task >> [second\_task, third\_task]  third\_task << fourth\_task |

* Cách 2: dùng hàm set\_dowstream và set\_upstream

|  |
| --- |
| first\_task.set\_downstream(second\_task, third\_task)  third\_task.set\_upstream(fourth\_task) |

Trường hợp các tasks giao nhau

|  |
| --- |
| from airflow.models.baseoperator import cross\_downstream  # Replaces  # [op1, op2] >> op3  # [op1, op2] >> op4  cross\_downstream([op1, op2], [op3, op4]) |

Trường hợp các tasks tạo thành dây chuyền

|  |
| --- |
| from airflow.models.baseoperator import chain  #1  # op1 >> op2 >> op3 >> op4  chain(op1, op2, op3, op4)  #2  # op1 >> op2 >> op3 >> op4 >> op5 >> op6  chain(\*[EmptyOperator(task\_id='op' + i) for i in range(1, 6)])  #3  # op1 >> op2 >> op4 >> op6  # op1 >> op3 >> op5 >> op6  chain(op1, [op2, op3], [op4, op5], op6) |

## 8. Cách viết kết hợp Decorator và Context manager

Nếu xảy ra trường hợp bạn dùng PythonOperator để chạy task A và dùng EmailOperator để chạy task B thì bạn cũng có thể kết hợp 2 cách viết. Ví dụ:

|  |
| --- |
| from airflow.decorators import dag, task  from airflow.operators.email\_operator import EmailOperator  from datetime import datetime  from typing import Dict  import requests  import logging  API = "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs\_currencies=usd&include\_market\_cap=true&include\_24hr\_vol=true&include\_24hr\_change=true&include\_last\_updated\_at=true"  @dag(schedule\_interval='@daily', start\_date=datetime(2021, 12, 1), catchup=False)  def taskflow():  @task(task\_id='extract', retries=2)  def extract\_bitcoin\_price() -> Dict[str, float]:  return requests.get(API).json()['bitcoin']  @task(multiple\_outputs=True)  def process\_data(response: Dict[str, float]) -> Dict[str, float]:  logging.info(response)  return {'usd': response['usd'], 'change': response['usd\_24h\_change']}  @task  def store\_data(data: Dict[str, float]):  logging.info(f"Store: {data['usd']} with change {data['change']}")  email\_notification = EmailOperator(  task\_id='email\_notification',  to='noreply@xxx.com',  subject='dag completed',  html\_content='the dag has finished'  )  store\_data(process\_data(extract\_bitcoin\_price())) >> email\_notification  dag = taskflow() |

## 9. cài đặt pip package cho task

Trong trường hợp task của bạn chạy cần một thư viện chỉ định. Chúng ta có thể dùng cách viết Decorator @task.virtualenv hoặc cách viết Context Manager PythonVirtualenvOperator. Cụ thể như sau:

* Decorator:

|  |
| --- |
| @task.virtualenv(  task\_id="virtualenv\_python", requirements=["numpy"], system\_site\_packages=False  )  def mul\_number(numbs: list):  import numpy  return int(numpy.prod(numpy.array(numbs))) |

* Context Manager:

|  |
| --- |
| def mul\_number(numbs: list):  import numpy  return int(numpy.prod(numpy.array(numbs)))  virtualenv\_task = PythonVirtualenvOperator(  task\_id="virtualenv\_python",  python\_callable=mul\_number(numbs),  requirements=["numpy"],  system\_site\_packages=False,  ) |

Ps: Nếu bạn có nhiều thư viện cần cài thì có thể thêm đường dẫn tới file requirements.txt: requirements="path\to\requirements.txt"

## 10. Chạy File DAG

1. Tạo 1 file DAG mẫu

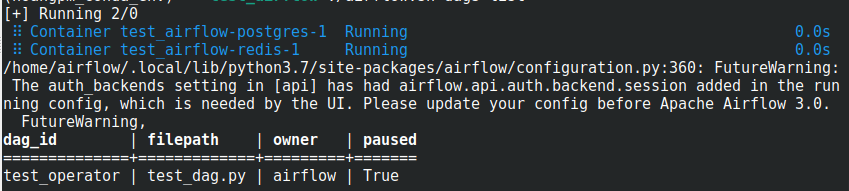
|  |
| --- |
| import random  from datetime import datetime  from airflow.decorators import dag, task  @dag(  start\_date=datetime.now(),  schedule\_interval=None,  catchup=False,  tags=['example'],  )  def test\_operator():  @task()  def get\_random\_number():  n1 = random.choice(range(100))  n2 = random.choice(range(100))  return [n1, n2]  @task()  def add\_number(numbs: list):  return sum(numbs)  @task.virtualenv(  task\_id="virtualenv\_python", requirements=["numpy"], system\_site\_packages=False  )  def mul\_number(numbs: list):  import numpy  return int(numpy.prod(numpy.array(numbs)))    @task()  def display(add\_res, mul\_res):  return (add\_res, mul\_res)  task1 = get\_random\_number()  task2 = add\_number(task1)  task3 = mul\_number(task1)  display(task2, task3)  test\_dag = test\_operator() |

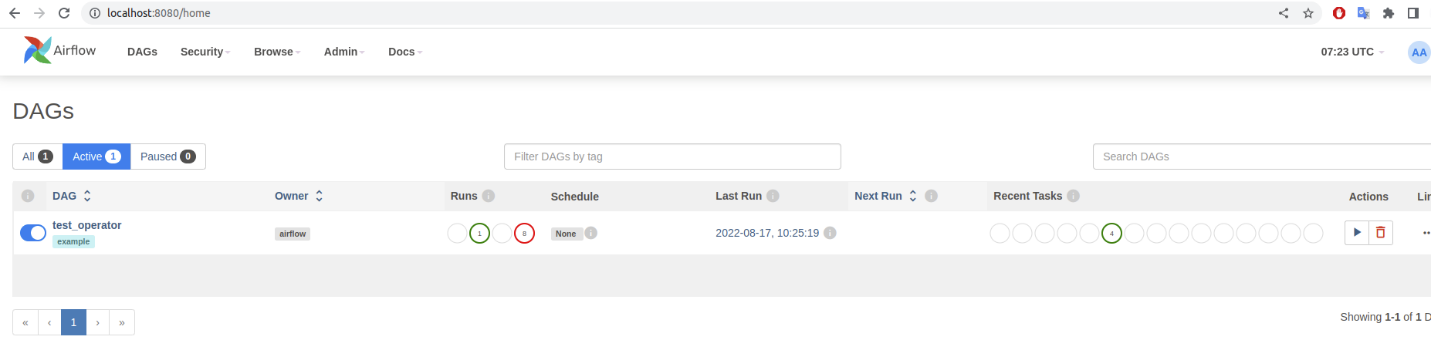
2. Chuyển file DAG này vào thư mục **dags/** mà Airflow container đã mount.

|  |
| --- |
| volumes:  - ./dags:/opt/airflow/dags |

3. Refresh lại WebUI để cập nhập trạng thái mới nhất. Nếu bạn kiểm tra airflow database sẽ thấy thông tin file DAG đã được lưu lại. Dùng airflow shell script để kiểm tra

|  |
| --- |
| ./airflow.sh dags list |





1. Ban đầu DAG test\_operator trong trạng thái tạm dừng, chúng ta cần kích hoạt nó, sau đó trigger DAG nằm trong mục Actions ở ảnh trên. Mục Runs sẽ thống kê trạng thái của các lần chạy DAG, như ở bên trên thì tôi có 1 lần thành công và 8 lần thất bại ( do chạy thử để sửa code ). Ngoài ra bạn có thể click vào tên file DAG để xem chi tiết ( kiến trúc, ngày giờ chạy, thời gian chạy của mỗi task, ... )

Vậy là xong, chỉ cần 4 bước để chạy 1 file DAG, sau khi chạy xong các bạn có thể dễ dàng quản lý, quan sát workflow thông qua WebUI

P/s: Nếu các bạn muốn xóa file DAG trên web thì làm theo các bước sau:

1. Xóa file DAG trong thư mục dags/
2. Dùng câu lệnh ./airflow.sh dags delete DAG\_ID - DAG\_ID ở đây là tên file DAG, ví dụ "test\_operator" để xóa bản ghi trong cơ sở dữ liệu hoặc dùng câu lệnh /airflow.sh db reset
3. Trên Web UI, ấn vào icon thùng rác trong mục Actions