# **File docker-compose.yaml**

---

x-airflow-common:

*# image: ${AIRFLOW\_IMAGE\_NAME:-apache/airflow:2.8.2}*

  &airflow-common

  build: .

  environment: &airflow-common-env

    AIRFLOW\_\_CORE\_\_EXECUTOR: CeleryExecutor

    AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

    AIRFLOW\_\_CELERY\_\_RESULT\_BACKEND: db+postgresql://airflow:airflow@postgres/airflow

    AIRFLOW\_\_CELERY\_\_BROKER\_URL: redis://:@redis:6379/0

    AIRFLOW\_\_CORE\_\_FERNET\_KEY: ''

    AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_AT\_CREATION: 'true'

    AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'true'

    AIRFLOW\_\_API\_\_AUTH\_BACKENDS: 'airflow.api.auth.backend.basic\_auth,airflow.api.auth.backend.session'

    AIRFLOW\_\_SCHEDULER\_\_ENABLE\_HEALTH\_CHECK: 'true'

    \_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:-}

  volumes:

    - ${AIRFLOW\_PROJ\_DIR:-.}/dags:/opt/airflow/dags

    - ${AIRFLOW\_PROJ\_DIR:-.}/logs:/opt/airflow/logs

    - ${AIRFLOW\_PROJ\_DIR:-.}/config:/opt/airflow/config

    - ${AIRFLOW\_PROJ\_DIR:-.}/plugins:/opt/airflow/plugins

  user: "${AIRFLOW\_UID:-50000}:0"

  depends\_on: &airflow-common-depends-on

    redis:

      condition: service\_healthy

    postgres:

      condition: service\_healthy

services:

  postgres:

    image: postgres:13

    environment:

      POSTGRES\_USER: airflow

      POSTGRES\_PASSWORD: airflow

      POSTGRES\_DB: airflow

    volumes:

      - postgres-db-volume:/var/lib/postgresql/data

    healthcheck:

      test: [ "CMD", "pg\_isready", "-U", "airflow" ]

      interval: 10s

      retries: 5

      start\_period: 5s

    restart: always

    networks:

      - qdrant-network

  qdrant:

    image: qdrant/qdrant:v1.4.1

    container\_name: qdrant\_db

    ports:

      - "6333:6333"

    networks:

      - qdrant-network

  mongo:

    image: mongo:4.0.8

    container\_name: mongodb

    command: mongod --auth

    environment:

      MONGO\_INITDB\_ROOT\_USERNAME: admin

      MONGO\_INITDB\_ROOT\_PASSWORD: admin

      MONGO\_INITDB\_DATABASE: fit-iuh

      MONGO\_DATA\_DIR: /data/db

      MONGO\_LOG\_DIR: /dev/null

    ports:

      - "27017:27017"

    networks:

      - qdrant-network

*# myapp:*

*#   build: ../lab7\_app*

*#   ports:*

*#     - "8989:5123"*

*#   networks:*

*#     - qdrant-network*

*#   depends\_on:*

*#     - qdrant*

*#     - mongo*

  redis:

    image: redis:latest

    expose:

      - 6379

    healthcheck:

      test: [ "CMD", "redis-cli", "ping" ]

      interval: 10s

      timeout: 30s

      retries: 50

      start\_period: 30s

    restart: always

    networks:

      - qdrant-network

  airflow-webserver:

    <<: \*airflow-common

    command: webserver

    ports:

      - "8080:8080"

    healthcheck:

      test:

        [

          "CMD",

          "curl",

          "--fail",

          "http://localhost:8080/health"

        ]

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    restart: always

    networks:

      - qdrant-network

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

  airflow-scheduler:

    <<: \*airflow-common

    command: scheduler

    healthcheck:

      test:

        [

          "CMD",

          "curl",

          "--fail",

          "http://localhost:8974/health"

        ]

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    restart: always

    networks:

      - qdrant-network

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

  airflow-worker:

    <<: \*airflow-common

    command: celery worker

    healthcheck:

*# yamllint disable rule:line-length*

      test:

        - "CMD-SHELL"

        - 'celery --app airflow.providers.celery.executors.celery\_executor.app inspect ping -d "celery@$${HOSTNAME}" || celery --app airflow.executors.celery\_executor.app inspect ping -d "celery@$${HOSTNAME}"'

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    environment:

      <<: \*airflow-common-env

      DUMB\_INIT\_SETSID: "0"

    restart: always

    networks:

      - qdrant-network

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

  airflow-triggerer:

    <<: \*airflow-common

    command: triggerer

    healthcheck:

      test:

        [

          "CMD-SHELL",

          'airflow jobs check --job-type TriggererJob --hostname "$${HOSTNAME}"'

        ]

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    restart: always

    networks:

      - qdrant-network

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

  airflow-init:

    <<: \*airflow-common

    entrypoint: /bin/bash

    command:

      - -c

      - |

        if [[ -z "${AIRFLOW\_UID}" ]]; then

          echo

          echo -e "\033[1;33mWARNING!!!: AIRFLOW\_UID not set!\e[0m"

          echo "If you are on Linux, you SHOULD follow the instructions below to set "

          echo "AIRFLOW\_UID environment variable, otherwise files will be owned by root."

          echo "For other operating systems you can get rid of the warning with manually created .env file:"

          echo "    See: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#setting-the-right-airflow-user"

          echo

        fi

        one\_meg=1048576

        mem\_available=$$(($$(getconf \_PHYS\_PAGES) \* $$(getconf PAGE\_SIZE) / one\_meg))

        cpus\_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)

        disk\_available=$$(df / | tail -1 | awk '{print $$4}')

        warning\_resources="false"

        if (( mem\_available < 4000 )) ; then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"

          echo "At least 4GB of memory required. You have $$(numfmt --to iec $$((mem\_available \* one\_meg)))"

          echo

          warning\_resources="true"

        fi

        if (( cpus\_available < 2 )); then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"

          echo "At least 2 CPUs recommended. You have $${cpus\_available}"

          echo

          warning\_resources="true"

        fi

        if (( disk\_available < one\_meg \* 10 )); then

          echo

          echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for Docker.\e[0m"

          echo "At least 10 GBs recommended. You have $$(numfmt --to iec $$((disk\_available \* 1024 )))"

          echo

          warning\_resources="true"

        fi

        if [[ $${warning\_resources} == "true" ]]; then

          echo

          echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see above)!\e[0m"

          echo "Please follow the instructions to increase amount of resources available:"

          echo "   https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#before-you-begin"

          echo

        fi

        mkdir -p /sources/logs /sources/dags /sources/plugins

        chown -R "${AIRFLOW\_UID}:0" /sources/{logs,dags,plugins}

        exec /entrypoint airflow version

    environment:

      <<: \*airflow-common-env

      \_AIRFLOW\_DB\_MIGRATE: 'true'

      \_AIRFLOW\_WWW\_USER\_CREATE: 'true'

      \_AIRFLOW\_WWW\_USER\_USERNAME: ${\_AIRFLOW\_WWW\_USER\_USERNAME:-airflow}

      \_AIRFLOW\_WWW\_USER\_PASSWORD: ${\_AIRFLOW\_WWW\_USER\_PASSWORD:-airflow}

      \_PIP\_ADDITIONAL\_REQUIREMENTS: ''

    user: "0:0"

    volumes:

      - ${AIRFLOW\_PROJ\_DIR:-.}:/sources

    networks:

      - qdrant-network

  airflow-cli:

    <<: \*airflow-common

    profiles:

      - debug

    environment:

      <<: \*airflow-common-env

      CONNECTION\_CHECK\_MAX\_COUNT: "0"

    command:

      - bash

      - -c

      - airflow

  flower:

    <<: \*airflow-common

    command: celery flower

    profiles:

      - flower

    ports:

      - "5555:5555"

    healthcheck:

      test: [ "CMD", "curl", "--fail", "http://localhost:5555/" ]

      interval: 30s

      timeout: 10s

      retries: 5

      start\_period: 30s

    restart: always

    networks:

      - qdrant-network

    depends\_on:

      <<: \*airflow-common-depends-on

      airflow-init:

        condition: service\_completed\_successfully

networks:

  qdrant-network:

    driver: bridge

volumes:

  postgres-db-volume:

# **File: “.env” : *AIRFLOW\_UID=501***

# **Các lệnh**

Sau khi có file docker-compose.yaml

Có thể setup thư mục như:

20001955\_TruongVanThong

├── airflow

│   ├── docker-compose.yml

│   ├── Dockerfile

│   ├── fit\_news.py

│   ├── requirements.txt

│   └── data\_iuh\_title.json

├── app

│   ├── Dockerfile

│   ├── requirements.txt

│   ├── app.py

Thư mục airflow:

requirements.txt: Thư viện

qdrant-client==1.8.0

beautifulsoup4==4.12.3

requests==2.31.0

pymongo==4.3.3

Dockerfile: setup

FROM apache/airflow:2.8.2

RUN pip install apache-airflow==${AIRFLOW\_VERSION}

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

Data\_iuh\_title.json: Thầy cho không cho thì sửa hàm *insertVectorDB*

fit\_news.py: Crawl data tùy thầy cho

import uuid

import requests

import pymongo

import datetime as dt

import random

import numpy as np

from bs4 import BeautifulSoup

import json

from airflow import DAG

from airflow.operators.bash\_operator import BashOperator

from airflow.operators.python\_operator import PythonOperator

from qdrant\_client import QdrantClient

from qdrant\_client.http.models import Distance, VectorParams, PointStruct

def printCollections():

    client = QdrantClient(host="qdrant\_db", port=6333)

    collections = client.get\_collections()

    collectionNames = [

        collection.name for collection in collections.collections]

    if collectionNames:

        print("Collections: ", collectionNames)

    else:

        print("No collection in database")

def insertMongoDB(data: dict):

    try:

        client = pymongo.MongoClient(

            "mongodb://admin:admin@mongodb:27017")

        db = client["fit-iuh"]

        news = db["news"]

*# check if news['title'] already exists*

        if news.find\_one({"title": data['title']}):

            return 0

        result = news.insert\_one(data)

        return 1

    except Exception as e:

        return 0

def createCollection():

    client = QdrantClient(host="qdrant\_db", port=6333)

    collections = client.get\_collections()

    collectionNames = [

        collection.name for collection in collections.collections]

    if "fit-iuh-news" not in collectionNames:

        client.recreate\_collection(

            collection\_name="fit-iuh-news",

            vectors\_config=VectorParams(size=1536, *#openAI: 1536*

                                        distance=Distance.COSINE)

        )

        print("Create collection successfully")

    else:

        print("Collection already exists")

def getNew(element):

    try:

        a\_tag = element.find('a')

        title = a\_tag['title']

        href = a\_tag['href']

        date = element.find(

            class\_='content-date').text

        return {

            "title": title,

            "href": href,

            "date": dt.datetime.strptime(date.strip(), "%d-%m-%Y")

        }

    except Exception as e:

        return {}

def getContentNews(url):

    url = "https://fit.iuh.edu.vn/"+url

    response = requests.get(url)

    soup = BeautifulSoup(response.text, 'html.parser')

    content = soup.select(".left-content > .content-list")

    content = content[0].text

    return content

def findLatestTimeNews():

    client = pymongo.MongoClient(

        "mongodb://admin:admin@mongodb:27017")

    db = client["fit-iuh"]

    news = db["news"]

    latestNews = news.find().sort("date", -1).limit(1)

    latest\_news\_item = next(latestNews, None)

    if latest\_news\_item:

        return latest\_news\_item['date']

    return dt.datetime.now() + dt.timedelta(days=1)

def crawlData():

    fitUrl = "https://fit.iuh.edu.vn/"

    latestTimeNews = findLatestTimeNews()

    pageNum = range(5)

    count = 0

    for page in pageNum:

        response = requests.get(

            fitUrl+f"news.html@102@Tin-tuc-Su-kien@p={page}")

        if response.status\_code == 200:

            soup = BeautifulSoup(response.text, 'html.parser')

            selectorContents = ".content-list > .content > [class\*=content-info]"

            contents = soup.select(selectorContents)

            for content in contents:

                new = getNew(content)

                if new:

                    if new['date'] > latestTimeNews:

                        print(new)

                        return

                    count += insertMongoDB(new)

    if count:

        print(f"Insert {count} news successfully")

    else:

        print("No news to insert")

def updateNews():

*# find all news in database not have content*

    client = pymongo.MongoClient(

        "mongodb://admin:admin@mongodb:27017")

    db = client["fit-iuh"]

    news = db["news"]

    newsNotHaveContent = news.find({"content": {"$exists": False}})

    for new in newsNotHaveContent:

        content = getContentNews(new['href'])

        news.update\_one({"\_id": new['\_id']}, {"$set": {"content": content}})

    print("Update news successfully")

*# def insertVectorDB():*

*#     qdrantClient = QdrantClient(host="qdrant\_db", port=6333)*

*#     mongoClient = pymongo.MongoClient(*

*#         "mongodb://admin:admin@mongodb:27017")*

*#     db = mongoClient["fit-iuh"]*

*#     news = db["news"]*

*#     allNews = news.find()*

*#     for new in allNews:*

*#         id = str(new.pop('\_id'))*

*#         vector = np.random.rand(100)*

*#         point = PointStruct(id=str(uuid.uuid4()),*

*#                             vector=vector.tolist(),*

*#                             payload=new)*

*#         qdrantClient.upsert(collection\_name="fit-iuh-news", points=[point])*

def insertVectorDB():

    qdrantClient = QdrantClient(host="qdrant\_db", port=6333)

    mongoClient = pymongo.MongoClient(

        "mongodb://admin:admin@mongodb:27017")

    with open("/opt/airflow/dags/data\_iuh\_title.json", "r") as f:

        data = json.load(f)

    db = mongoClient["fit-iuh"]

    news = db["news"]

    allNews = news.find()

    for new in allNews:

        id = str(new.pop('\_id'))

        if new['title'] in data:

            vector = data[new['title']]

            point = PointStruct(id=str(uuid.uuid4()),

                                vector=vector,

                                payload=new)

            qdrantClient.upsert(collection\_name="fit-iuh-news", points=[point])

default\_args = {

    'owner': 'VanThong',

    'start\_date': dt.datetime.now() - dt.timedelta(minutes=9),

    'retries': 1,

    'retry\_delay': dt.timedelta(minutes=10),

}

with DAG('ThongDag',

         default\_args=default\_args,

         tags=['LAB6'],

         schedule\_interval=dt.timedelta(minutes=10),      *# '0 \* \* \* \*',*

         ) as dag:

    create\_collection = PythonOperator(task\_id='create\_collection\_qdrant',

                                       python\_callable=createCollection)

    crawl\_data = PythonOperator(task\_id='crawl\_data',

                                python\_callable=crawlData)

    update\_content = PythonOperator(task\_id='update\_news',

                                    python\_callable=updateNews)

    insert\_vector = PythonOperator(task\_id='insert\_vector',

                                   python\_callable=insertVectorDB)

    end = BashOperator(task\_id='end',

                       bash\_command='echo "Chương trình đã hoàn thành....."')

create\_collection >> crawl\_data >> update\_content >> insert\_vector >> end

Build bên airflow trước:

**cd airflow 🡪 docker compose up --build**

Chờ hơi lâu 🡪 Sau đó sẽ có các thư mục xuất hiện trong aiflow trong đó có thư mục: “dags” ta copy file fit\_news.py và data\_iuh\_title.json

*Trick lỏ để nó add nhanh và trong airflow: new 1 PS mới*

*> docker ps*

*tìm IMAGE: a460389c308b airflow-airflow-scheduler chỉ lấy 3 số đầu là hiểu*

***>*** *docker exec -it a46 bash*

***>*** *default@a460389c308b:/opt/airflow$* ***airflow scheduler***

**Tiếp theo lên web check  
Airflow:** [**http://localhost:8080/**](http://localhost:8080/)

**Qdrant: http://localhost:6333/dashboard**

Xong phần trên nếu xanh hết và trong Qdrant có data

Phần app: Khó điên

File: app.py

import numpy as np

import flask

from flask import request, make\_response

from qdrant\_client import QdrantClient

app = flask.Flask(\_\_name\_\_)

*# ------------------------API----------------------------------*

*# Không cần url và headers nữa vì không gọi API*

*# url = "https://be12-34-72-112-245.ngrok-free.app"*

*# headers = {*

*#     'mssv': '20001955'*

*# }*

*# def get\_embdding(text):*

*#     response = requests.post(*

*#         url + '/embedding', json={'text': text}, headers=headers)*

*#     return response.json()['embedding']*

*# def complete(message, context):*

*#     response = requests.post(*

*#         url + '/complete', json={'question': message, 'context': context}, headers=headers)*

*#     return response.json()*

*#----------------------------------------------------------------*

def get\_embedding\_random(text):

*# Sinh ngẫu nhiên một vector có chiều dài 1536*

    return np.random.rand(1536).tolist()

def complete(message, context):

*# Sinh ngẫu nhiên một chuỗi 100 ký tự*

    return {"answer": " ".join([message, " ".join([chr(np.random.randint(97, 123)) for \_ in range(100)])])}

def search(query):

    client = QdrantClient(host = "qdrant\_db", port = 6333)

    collections = client.get\_collections()

    collectionNames = [

        collection.name for collection in collections.collections]

*# embed = get\_embdding(query) # Sử dụng hàm của API*

    embed = get\_embedding\_random(query) *# Sử dụng hàm mới tạo vector ngẫu nhiên*

    if "fit-iuh-news" in collectionNames:

        results = client.search(

            collection\_name="fit-iuh-news",

            query\_vector=embed,

            limit = 1,

        )

        return results[0].model\_dump()

    return {"message": "Collection not found"}

@app.route('/get\_collections', methods=['GET'])

def get\_collections():

    client = QdrantClient(host="qdrant\_db", port=6333)

    collections = client.get\_collections()

    collectionNames = [

        collection.name for collection in collections.collections]

    if collectionNames:

        return {"collections": collectionNames}

    else:

        return {"message": "No collection in database"}

@app.route('/search', methods=['POST'])

def searchView():

    query = request.json['query']

    results = search(query)

    return make\_response(results)

@app.route('/complete', methods=['POST'])

def completeView():

    query = request.json['query']

    result = search(query)

    context = result['payload']['title']+'\n'+result['payload']['content']

    results = complete(query, context)

    print(results)

    return make\_response(results)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5123, debug=False)

*# curl -X POST https://localhost:8989/search -H "Content-Type: application/json" -d '{"query": "hội nghị cấp khoa"}'*

Dockerfile

FROM python:3.9.11

WORKDIR /app

COPY requirements.txt /requirements.txt

RUN pip install --upgrade pip

RUN pip install -r /requirements.txt && pip cache purge

COPY . .

CMD ["python", "app.py"]

Requirements.txt

flask==2.2.3

qdrant-client==1.8.0

requests==2.31.0

Werkzeug==2.2.2

Vào lại docker-compose.yaml

Mở cái cmt ra để build app:  
  *# myapp:*

*#   build: ../lab7\_app*

*#   ports:*

*#     - "8989:5123"*

*#   networks:*

*#     - qdrant-network*

*#   depends\_on:*

*#     - qdrant*

*#     - mongo*

../lab\_app tùy vào thư mục người đặt ví dụ đặt app thì chỉ ghi app

Tiếp theo build app:

New PS khác:

> **docker compose up myapp –build**

Vậy là xong bài 10đ trên lý thuyết lab7

Test search như anh Thành chỉ:

import requests

url = "http://localhost:8989"

response = requests.post(f"{url}/search", json = {"query": "Hội nghị cấp khoa?"})

response.json()

output: …

response = requests.post(f"{url}/complete", json = {"query": "Hội nghị cấp khoa được tổ chức ngày nào? bao gồm những nội dung gì?"})

response.json()

output: …