# **Các lệnh cơ bản**

### **Pull image**

Để download một image thì chúng ra sử dụng lệnh

docker pull image-name

vd mình muốn tải image nginx xuống thì chỉ cần chạy

docker pull nginx

### **Chạy Container**

Tải image xuống rồi thì phải tạo vào chạy container chứ nhỉ

docker run thamso Image command thamsolenh

Ví dụ chạy container từ image vừa pull xuống ở trên

docker run -it -p 8080:80 nginx

-it: là interactive terminal nghĩa là nó sẽ chạy trong terminal của chúng ta. Tiếp theo sẽ lấy ra một cái port để truy cập vào, ban đầu sẽ là 8080 và port bên trong container nginx là 80. cuối cùng là tên image là nginx.

Bây giờ mở trình duyệt ra và truy cập vào http://localhost:8080 xem có gì k nhé. Vậy là chúng ta đã tạo được 1 serve nginx trên máy của chúng ta rồi



### **Xem các container**

Để xem các container đang chạy, chỉ cần gõ lệnh

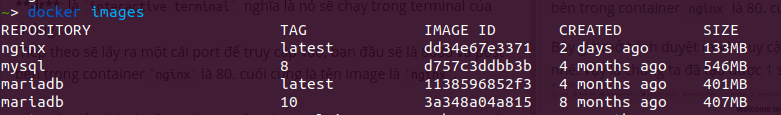
docker ps

Screenshot from 2021-08-19 21-44-43.png

Sẽ có một sô thông tin như, id , name, thời gian tạo,thời gian chạy, port..

Để xem chúng ta đang có những image nào thì bạn chỉ cần gõ

docker images



Bây giờ muốn dừng cái container chỉ cần

ctrl + c

Nếu các bạn chạy container với tham số là -d (detach). thì muốn dừng container các bạn cần chạy lệnh

docker stop + id\_container

có thể thay id bằng name của container Bây giờ chúng ta gõ docker ps thì sẽ không thấy container nginx nữa. Để xem tất cả các container đang chạy hay đã dừng thì chỉ cần gõ

docker ps -a

Nếu muốn khởi động lại container vừa stop ở trên bạn chỉ cần gõ

docker start + id\_container hoặc + name\_container

### **Xóa container**

để xóa container bạn chỉ cần chạy lệnh + id của container

docker rm + id\_container

* Lưu ý: có thể xóa nhiều container cùng lúc, không thể xóa được các container đang chạy, muốn xóa bạn phải exit contaier đó đi trước khi xóa.

### **Xóa image**

Tương tự như xóa container, lệnh xóa image

docker rmi + id\_image

Cũng có thể xóa image bằng tên của image

docker rmi + name\_image

Lưu ý là chúng ta cũng không thể xóa được image khi mà image đó đang được sử dụng để chạy một container nào đó.

### **Đặt tên cho container**

Ở phần chạy container mình quên mất chưa nói đến phần đặt tên cho container. các bạn chỉ cần thêm tham số --name tênmuốnđặt tênimage Ví dụ mình muốn có container với tên là myNginx của image nginx thì chỉ cần chạy

docker run -it -p 8082:80 --name myNginx nginx

### **Vào bên trong 1 container**

Để vào bên trong 1 container để xem các file bên trong của nó bạn có thể dùng lệnh

docker exec -it + name\_container bash

Ví dụ mình muốn chui vào container myNginx vừa tạo ở trên

docker exec -it myNginx bash

rồi bây giờ các bạn có thể chạy ls -la để xem tất cả các file trong container đó

### **Xem log của container**

Chúng ta có thể xem log của container bằng lệnh

docker logs + name\_container

### Dockerfile python flask

—------------------------------------------------------------------------------------------------------------------------

Dockerfile

##### Dockerfile #####

FROM python:3.10-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

COPY . .

CMD [ "python3", "-m" , "flask", "run" ]

—------------------------------------------------------------------------------------------------------------------------

### **1. FROM python:3.10-slim-buster**

* **Ý nghĩa:** Chọn image cơ sở là python:3.10-slim-buster.
* **Chi tiết:**
  + python:3.10 là image chứa Python 3.10.
  + slim-buster là một phiên bản nhẹ của Debian Buster, giúp giảm dung lượng image.

### **2. WORKDIR /app**

* **Ý nghĩa:** Thiết lập thư mục làm việc bên trong container là /app.
* **Chi tiết:**
  + Tất cả các lệnh sau đó sẽ chạy trong thư mục /app.
  + Giúp tổ chức thư mục gọn gàng, dễ quản lý.

### **3. COPY requirements.txt requirements.txt**

* **Ý nghĩa:** Sao chép file requirements.txt từ máy host vào container.
* **Chi tiết:**
  + requirements.txt chứa danh sách thư viện cần cài đặt.
  + Sao chép riêng file này giúp tối ưu Docker build cache, tránh cài lại thư viện nếu không cần thiết.

### **4. RUN pip3 install -r requirements.txt**

* **Ý nghĩa:** Cài đặt các thư viện Python cần thiết được liệt kê trong requirements.txt.
* **Chi tiết:**
  + Sử dụng pip3 install -r requirements.txt để cài đặt dependencies.
  + Nếu requirements.txt không thay đổi, Docker cache giúp bỏ qua bước cài đặt này khi build lại container.

### **5. COPY . .**

* **Ý nghĩa:** Sao chép toàn bộ mã nguồn từ thư mục hiện tại trên máy host vào container.
* **Chi tiết:**
  + Dấu . đầu tiên là thư mục nguồn (máy host).
  + Dấu . thứ hai là thư mục đích (/app do WORKDIR đặt trước đó).

### **6. CMD [ "python3", "-m" , "flask", "run" ]**

* **Ý nghĩa:** Lệnh mặc định khi container chạy.
* **Chi tiết:**
  + Chạy Flask bằng python3 -m flask run.
  + Flask sẽ khởi chạy ứng dụng theo biến môi trường FLASK\_APP.

—------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Docker Compose là một công cụ hỗ trợ xác định và chạy các ứng dụng multi-container . Docker Compose có thể xử lý đồng thời multi-container trong sản xuất, staging, phát triển, thử nghiệm và CI.

Docker Compose hoạt động bằng cách áp dụng các quy tắc được xác định trong tệp docker-compose.yaml.

Giả sử bạn đang phát triển một ứng dụng web có cấu trúc như sau:

* Một container chứa ứng dụng web Node.js, chạy trên cổng 3000.
* Một container chứa cơ sở dữ liệu MySQL, chạy trên cổng 3306.

Để triển khai ứng dụng này bằng Docker Compose, bạn cần tạo một tệp docker-compose.yml với các thông tin sau:

Yaml

—------------------------------------------------------------------------------------------------------------------------

version: "3.9"   
services:

web:

build: .

ports: - "3000:3000"

depends\_on:

- db

db:

image:

mysql:latest

environment:

MYSQL\_ROOT\_PASSWORD: password

ports: - "3306:3306"

—------------------------------------------------------------------------------------------------------------------------

Trong đó:

* version: phiên bản của Docker Compose
* services: danh sách các dịch vụ cần triển khai
* web: định nghĩa dịch vụ web, bao gồm việc build image từ Dockerfile và ánh xạ cổng 3000 của container với cổng 3000 của máy host.
* depends\_on cho biết dịch vụ này cần phụ thuộc vào dịch vụ db.
* db: định nghĩa dịch vụ cơ sở dữ liệu MySQL, bao gồm việc sử dụng image MySQL, đặt mật khẩu cho root user và ánh xạ cổng 3306 của container với cổng 3306 của máy host.

## **Các lệnh cơ bản trong Docker Compose**

* docker-compose up: Khởi động các container
* docker-compose down: Dừng và xóa các container
* docker-compose ps: Hiển thị trạng thái của các container
* docker-compose build: Tạo image từ Dockerfile trong mỗi dịch vụ
  + docker-compose up --build -d : chạy xong trả lại terminal
* docker-compose restart: Khởi động lại các container
* docker-compose stop: Dừng các container
* docker-compose rm: Xóa các container không sử dụng
* docker-compose logs: Hiển thị các logs của các container
* docker-compose config: Hiển thị các cấu hình của Docker Compose
* docker-compose exec: Thực thi một lệnh trên một container
* docker-compose port: Hiển thị các port của các container
* docker-compose top: Hiển thị các process đang chạy trong các container

Lưu ý: Các lệnh trên phải được thực hiện trong thư mục chứa file docker-compose.yml.  
**Docker vs. Docker Compose?**

**—---------Phân biệt docker và docker compose—-----------**

| **Docker** | **Docker Compose** |
| --- | --- |
| Chạy container đơn lẻ | Chạy nhiều container cùng lúc |
| Dùng lệnh docker run | Dùng file docker-compose.yml |
| Không tự động kết nối giữa các container | Tự động kết nối các container trong cùng một mạng |

⚡ **Tóm lại:**

* **Docker**: Dùng để đóng gói & chạy ứng dụng trong container.
* **Docker Compose**: Dùng để quản lý & chạy nhiều container một cách dễ dàng.

—----------------------------------------------------------------------------------------------------------------------------------

tạo môi trường ảo: python -m venv venv

dùng môi trường ảo: venv\Scripts\activate

thoát : deactivate

celery -A celery\_worker.celery worker --loglevel=info --pool=solo chạy celery

—---------------------------các lệnh git—--------------------------------------------------------------------------

### **🔥 Cập nhật Git**

1️⃣ **Kiểm tra trạng thái thay đổi**

Sao chépChỉnh sửa

git status

2️⃣ **Thêm tất cả các tệp đã chỉnh sửa**

git add .

git add path/to/file.py

3️⃣ **Commit thay đổi với nội dung mô tả**

git commit -m "Cập nhật bảng events: sửa kiểu dữ liệu timestamp và numeric"

4️⃣ **Push lên GitHub hoặc GitLab** Nếu bạn đã thiết lập remote repository, dùng lệnh:

git push origin main # Nếu nhánh chính là main

## **🛠 Tạo nhánh mới và chuyển sang nhánh đó**

🔹 **Cách 1: Tạo và chuyển ngay sang nhánh mới**

git checkout -b <ten-nhanh-moi>

git checkout -b feature-update-events

📌 Lệnh này sẽ tạo nhánh mới có tên feature-update-events và tự động chuyển sang nhánh đó.

## **🚀 Tiến hành chỉnh sửa code**

### **✅ Lưu thay đổi và commit**

**Kiểm tra thay đổi:**  
git status

**Thêm tất cả file đã chỉnh sửa vào Git:**  
git add .

**Commit thay đổi với mô tả:**  
git commit -m "Cập nhật API sự kiện và sửa cấu trúc thư mục"

## **🔄 Đẩy nhánh lên GitHub**

git push origin <ten-nhanh-moi>

git push origin feature-update-events

## **🔗 Gộp nhánh vào nhánh chính (main)**

Sau khi hoàn tất và muốn hợp nhất nhánh mới với main:

Chuyển về nhánh main:  
git checkout main

Cập nhật nhánh main với GitHub:  
git pull origin main

Gộp nhánh mới vào main:  
git merge <ten-nhanh-moi>  
git merge feature-update-events

**Xóa nhánh không cần thiết nữa**:  
git branch -d <ten-nhanh-moi>

Hoặc nếu nhánh đã được đẩy lên GitHub:  
git push origin --delete <ten-nhanh-moi>

### **🎯 Tóm tắt nhanh**

1️⃣ git checkout -b feature-update-events → **Tạo và chuyển sang nhánh mới** 2️⃣ git add . && git commit -m "Mô tả commit" → **Commit thay đổi** 3️⃣ git push origin feature-update-events → **Đẩy nhánh lên GitHub** 4️⃣ git checkout main && git merge feature-update-events → **Gộp nhánh vào main**