

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN**



BÁO CÁO THỰC TẬP TỐT NGHIỆP

Tìm hiểu về Docker

Sinh viên thực hiện (trưởng nhóm xếp số 1):

B21DCAT090 Nguyễn Minh Hiệu

Giảng viên hướng dẫn: TS Nguyễn Ngọc Diệp

HÀ NỘI 6-2025

Mục lục

Docker: Nền Tảng Ảo Hóa Container và Ứng Dụng Trong Kỹ Thuật Phần Mềm Hiện Đại ..	3
1.1 Tổng Quan Khoa Học về Docker	3
1.2 Cơ Chế Hoạt Động: Container và Sự Khác Biệt Với Máy Ảo (VM)	3
1.2.1 Máy Ảo (Virtual Machine - VM)	3
1.2.2 Docker Container	3
1.3 Lợi Ích Cốt Lõi và Tầm Quan Trọng của Docker	4
1.4 Kiến Trúc và Thành Phần Chính của Docker	4
1.5 Các Thực Thể Quan Trọng Trong Docker	5
1.5.1 Docker Images	5
1.5.2 Docker Containers	5
1.5.3 Dockerfile	5
1.5.4 Docker Compose	5
1.6 Cài Đặt Docker	6
1.6.1 Trên Ubuntu/Linux	6
1.6.2 Trên Windows và macOS	8
1.7 Các Lệnh Docker Cơ Bản	9
1.7.1 Kiểm Tra Phiên Bản và Thông Tin Hệ Thống	9
1.7.2 Làm Việc Với Images	9
1.7.3 Làm Việc Với Containers	10
1.8 Quản Lý Dữ Liệu với Docker Volumes	11
1.9 Quản Lý Mạng với Docker Networking	11
1.10 Bài tập Thực hành	12
Bài tập 1: Container cơ bản	12
Bài tập 2: Dockerfile	15
Bài tập 3: Docker Compose	18
Bài tập 4: Networking	22

Docker: Nền Tảng Ảo Hóa Container và Ứng Dụng Trong Kỹ Thuật Phần Mềm Hiện Đại

1.1 Tổng Quan Khoa Học về Docker

Docker là một nền tảng **mã nguồn mở** được phát triển để chuẩn hóa và tự động hóa toàn bộ vòng đời của ứng dụng thông qua công nghệ **ảo hóa cấp độ hệ điều hành**, hay còn gọi là **containerization**. Mục tiêu cốt lõi của Docker là đóng gói các ứng dụng và mọi thứ chúng cần để chạy – bao gồm mã nguồn, runtime, thư viện hệ thống, biến môi trường, và tệp cấu hình – vào các đơn vị **tiêu chuẩn hóa, gọn nhẹ và có tính di động cao** được gọi là **container**.

Về bản chất, Docker thiết lập một **lớp trừu tượng hóa** mạnh mẽ, cho phép các nhà phát triển và quản trị viên hệ thống tách biệt ứng dụng khỏi cơ sở hạ tầng vật lý hoặc ảo hóa bên dưới. Điều này đảm bảo rằng một ứng dụng được đóng gói trong container sẽ hoạt động **nhất quán và đáng tin cậy** trên mọi môi trường có cài đặt Docker Engine, từ môi trường phát triển cục bộ đến hệ thống sản xuất phân tán.

1.2 Cơ Chế Hoạt Động: Container và Sự Khác Biệt Với Máy Ảo (VM)

Để hiểu rõ giá trị vượt trội của Docker, điều quan trọng là phải phân biệt công nghệ **container** của nó với công nghệ **máy ảo (Virtual Machine - VM)** truyền thống.

1.2.1 Máy Ảo (Virtual Machine - VM)

Một Máy ảo (VM) hoạt động bằng cách sử dụng một lớp phần mềm gọi là hypervisor (ví dụ: VMware ESXi, KVM, VirtualBox) để mô phỏng toàn bộ một hệ thống phần cứng vật lý hoàn chỉnh. Trên phần cứng ảo hóa này, một hệ điều hành khách (Guest OS) riêng biệt được cài đặt và khởi động. Ứng dụng sau đó sẽ chạy trên Guest OS này.

Đặc điểm:

- **Mức độ cách ly cao:** Mỗi VM có nhân hệ điều hành (kernel) riêng, cung cấp sự cô lập mạnh mẽ giữa các ứng dụng.
- **Tiêu tốn tài nguyên:** Mỗi Guest OS yêu cầu tài nguyên CPU, RAM và không gian đĩa đáng kể, dẫn đến thời gian khởi động lâu và chi phí overhead cao.

1.2.2 Docker Container

Ngược lại, Docker container không ảo hóa phần cứng. Thay vào đó, tất cả các container trên một máy chủ chia sẻ chung nhân (kernel) của hệ điều hành chủ (Host OS). Docker Engine, thành phần cốt lõi, tận dụng các tính năng sẵn có của kernel Linux như Namespaces và Control Groups (cgroups) để tạo ra các môi trường biệt lập.

Cơ chế Namespaces: Cung cấp sự cô lập về tài nguyên hệ thống (như PID, NET, IPC, MNT, UTS), đảm bảo mỗi container chỉ nhìn thấy và tương tác với các tài nguyên của riêng nó.

Cơ chế Control Groups (cgroups): Cho phép kiểm soát và giới hạn việc sử dụng tài nguyên (CPU, RAM, I/O) của từng container, ngăn chặn một container chiếm dụng quá nhiều tài nguyên và ảnh hưởng đến hiệu suất của các container khác hoặc của Host OS. Mỗi container chỉ đóng gói mã nguồn của ứng dụng cùng với các thư viện và thành phần phụ thuộc cần thiết cho nó, không bao gồm một hệ điều hành khách đầy đủ.

Đặc điểm:

- **Gọn nhẹ và hiệu quả:** Không có Guest OS, container khởi động gần như tức thời (mili giây) và tiêu thụ tài nguyên tối thiểu.
- **Tính di động cao:** Chạy nhất quán trên mọi môi trường có Docker Engine, nhờ sự đóng gói ứng dụng và môi trường vào một đơn vị độc lập.

1.3 Lợi Ích Cốt Lõi và Tầm Quan Trọng của Docker

Việc áp dụng Docker mang lại những lợi ích chiến lược, cách mạng hóa quy trình phát triển và vận hành phần mềm:

- **Tính nhất quán và Khả năng tái lập môi trường:** Docker loại bỏ vấn đề kinh điển "Nó chạy được trên máy của tôi!" bằng cách đảm bảo môi trường phát triển, thử nghiệm và sản xuất là hoàn toàn đồng nhất. Điều này giảm thiểu lỗi do khác biệt cấu hình, đơn giản hóa gỡ lỗi và tăng tốc độ phát hành.
- **Tối ưu hóa tài nguyên và Hiệu suất:** Nhờ kích thước nhỏ gọn và chia sẻ kernel, container tiêu thụ ít tài nguyên hơn đáng kể so với VM. Điều này cho phép triển khai mật độ ứng dụng cao hơn trên cùng một cơ sở hạ tầng, giảm chi phí phần cứng và năng lượng, đồng thời cải thiện hiệu suất tổng thể của hệ thống.
- **Thúc đẩy Kiến trúc Microservices và Văn hóa DevOps:** Docker là nền tảng lý tưởng cho kiến trúc microservices, cho phép chia nhỏ ứng dụng phức tạp thành các dịch vụ độc lập, dễ dàng phát triển, triển khai và mở rộng. Nó cũng là công cụ thiết yếu trong DevOps, tạo ra quy trình CI/CD liền mạch và nhất quán giữa các nhóm phát triển và vận hành.
- **Tính di động và linh hoạt tối đa:** Container Docker có thể chạy trên bất kỳ hệ thống nào có Docker Engine (Windows, macOS, Linux, đám mây công cộng hoặc trung tâm dữ liệu riêng), mang lại sự linh hoạt chưa từng có trong việc di chuyển và triển khai ứng dụng, tránh sự phụ thuộc vào một nhà cung cấp cụ thể.

1.4 Kiến Trúc và Thành Phần Chính của Docker

Docker hoạt động trên **kiến trúc client-server** với các thành phần chính sau:

- **Docker Engine:** Là thành phần cốt lõi, chạy dưới dạng một **daemon** (dockerd) trên máy chủ (host). Daemon này chịu trách nhiệm chính trong việc xây dựng (build), chạy, và quản lý các **Docker Images** và **Containers**.
- **Docker CLI (Command Line Interface):** Là giao diện dòng lệnh cho phép người dùng tương tác với Docker Engine bằng cách gửi các lệnh (ví dụ: docker run, docker build).

- **Docker API (Application Programming Interface):** Là giao diện RESTful cung cấp một cách lập trình để các ứng dụng khác (ví dụ: công cụ CI/CD, hệ thống quản lý container) tương tác với Docker Engine.

1.5 Các Thực Thể Quan Trọng Trong Docker

1.5.1 Docker Images

Docker Images là các **bản thiết kế tĩnh, không thể thay đổi (immutable blueprints)** cho container. Một image bao gồm tất cả những gì một ứng dụng cần để chạy: mã nguồn, thư viện, runtime, các biến môi trường, và các tệp cấu hình. Images được xây dựng theo từng lớp (layers), giúp tái sử dụng và tối ưu hóa dung lượng.

1.5.2 Docker Containers

Docker Containers là các **thể hiện (instances) đang chạy** của Docker Images. Khi bạn chạy một Docker Image, một container sẽ được khởi tạo. Container cung cấp một môi trường biệt lập, nhẹ và di động để ứng dụng của bạn hoạt động.

1.5.3 Dockerfile

Dockerfile là một tệp văn bản chứa một tập hợp các chỉ thị (instructions) tuần tự được Docker Engine sử dụng để **tự động hóa quá trình xây dựng một Docker Image**. Mỗi chỉ thị trong Dockerfile tạo ra một lớp mới trong image, đảm bảo tính tái lập và hiệu quả.

Cấu trúc Dockerfile cơ bản:

```
FROM ubuntu:20.04
# Thiết lập thư mục làm việc bên trong container
WORKDIR /app
# Copy files từ host vào container
COPY . .
# Chạy lệnh trong quá trình build image (ví dụ: cài đặt dependencies)
RUN apt-get update && apt-get install -y python3
# Thiết lập biến môi trường
ENV PYTHON_VERSION=3.9
# Mở port mà ứng dụng trong container sẽ lắng nghe
EXPOSE 8080
# Lệnh sẽ được chạy khi container khởi động
CMD ["python3", "app.py"]
```

1.5.4 Docker Compose

Docker Compose là một công cụ mạnh mẽ cho phép định nghĩa và chạy các ứng dụng Docker **đa-container (multi-container applications)**. Thay vì quản lý từng container một cách riêng lẻ, Docker Compose sử dụng một tệp **YAML** (docker-compose.yml) để định cấu hình tất cả các dịch vụ, mạng và volumes cần thiết cho một ứng dụng, sau đó cho phép khởi động hoặc dừng toàn bộ stack chỉ bằng một lệnh duy nhất.

Ví dụ cấu trúc docker-compose.yml:

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
    depends_on:
      - db

  db:
    image: postgres:13
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

1.6 Cài Đặt Docker

1.6.1 Trên Ubuntu/Linux

Cập nhật package list

```
sudo apt-get update
```

Cài đặt các dependencies

```
sudo apt-get install ca-certificates curl gnupg
```

Thêm Docker repository

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o  
/usr/share/keyrings/docker-archive-keyring.gpg
```

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-  
archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ sudo apt-get update  
Hit:1 https://download.docker.com/linux/ubuntu noble InRelease  
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease  
Hit:3 http://us.archive.ubuntu.com/ubuntu noble InRelease  
Hit:4 http://us.archive.ubuntu.com/ubuntu noble-updates InRelease  
Hit:5 http://us.archive.ubuntu.com/ubuntu noble-backports InRelease  
Reading package lists... Done  
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.  
student@LabtainerVMware:~$ sudo apt-get install ca-certificates curl gnupg  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
ca-certificates is already the newest version (20240203).  
curl is already the newest version (8.5.0-2ubuntu10.6).  
gnupg is already the newest version (2.4.4-2ubuntu17.2).  
gnupg set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 310 not upgraded.  
student@LabtainerVMware:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg  
student@LabtainerVMware:~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
student@LabtainerVMware:~$
```

Cài đặt Docker

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

```
student@LabtainerVMware:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
The following additional packages will be installed:  
  docker-ce-rootless-extras docker-compose-plugin  
Suggested packages:  
  cgroupfs-mount | cgroup-lite docker-model-plugin  
The following packages will be upgraded:  
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras  
  docker-compose-plugin  
5 upgraded, 0 newly installed, 0 to remove and 305 not upgraded.  
Need to get 87.3 MB of archives.  
After this operation, 2,901 kB disk space will be freed.  
Do you want to continue? [Y/n] Y  
Get:1 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-cli amd64 5:28.3.1-1~ubuntu.24.04~noble [16.5 MB]  
Get:2 https://download.docker.com/linux/ubuntu noble/stable amd64 containerd.io amd64 1.7.27-1 [30.5 MB]
```

Khởi động Docker

`sudo systemctl start docker`

`sudo systemctl enable docker`

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ sudo systemctl start docker  
student@LabtainerVMware:~$ sudo systemctl enable docker  
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.  
Executing: /usr/lib/systemd/systemd-sysv-install enable docker  
student@LabtainerVMware:~$
```

Kiểm tra cài đặt

`sudo docker run hello-world`

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
e6590344b1a5: Pull complete  
Digest: sha256:940c619fbd418f9b2b1b63e25d8861f9cc1b46e3fc8b018ccfe8b78f19b8cc4f  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (amd64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash
```

1.6.2 Trên Windows và macOS

Trên Windows

1. Tải Docker Desktop từ trang chính thức của Docker
2. Chạy file cài đặt và làm theo hướng dẫn
3. Kích hoạt tính năng Hyper-V trong quá trình cài đặt
4. Khởi động lại máy tính sau khi cài đặt hoàn tất

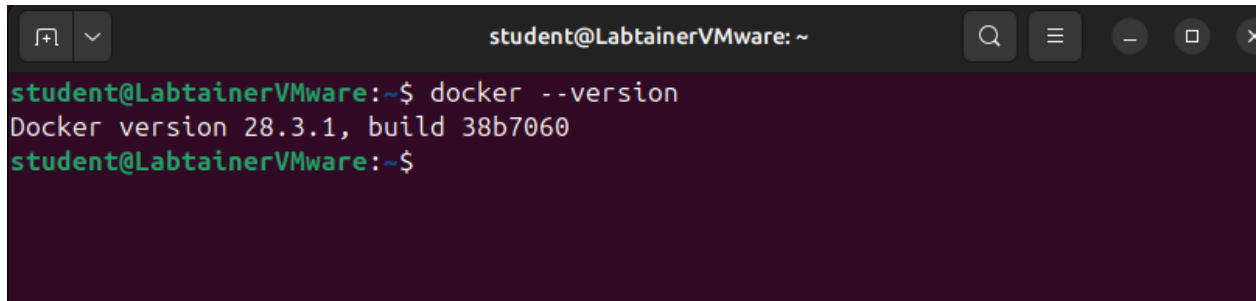
Trên macOS

Tải Docker Desktop từ trang chính thức và cài đặt như các ứng dụng thông thường⁴

1.7 Các Lệnh Docker Cơ Bản

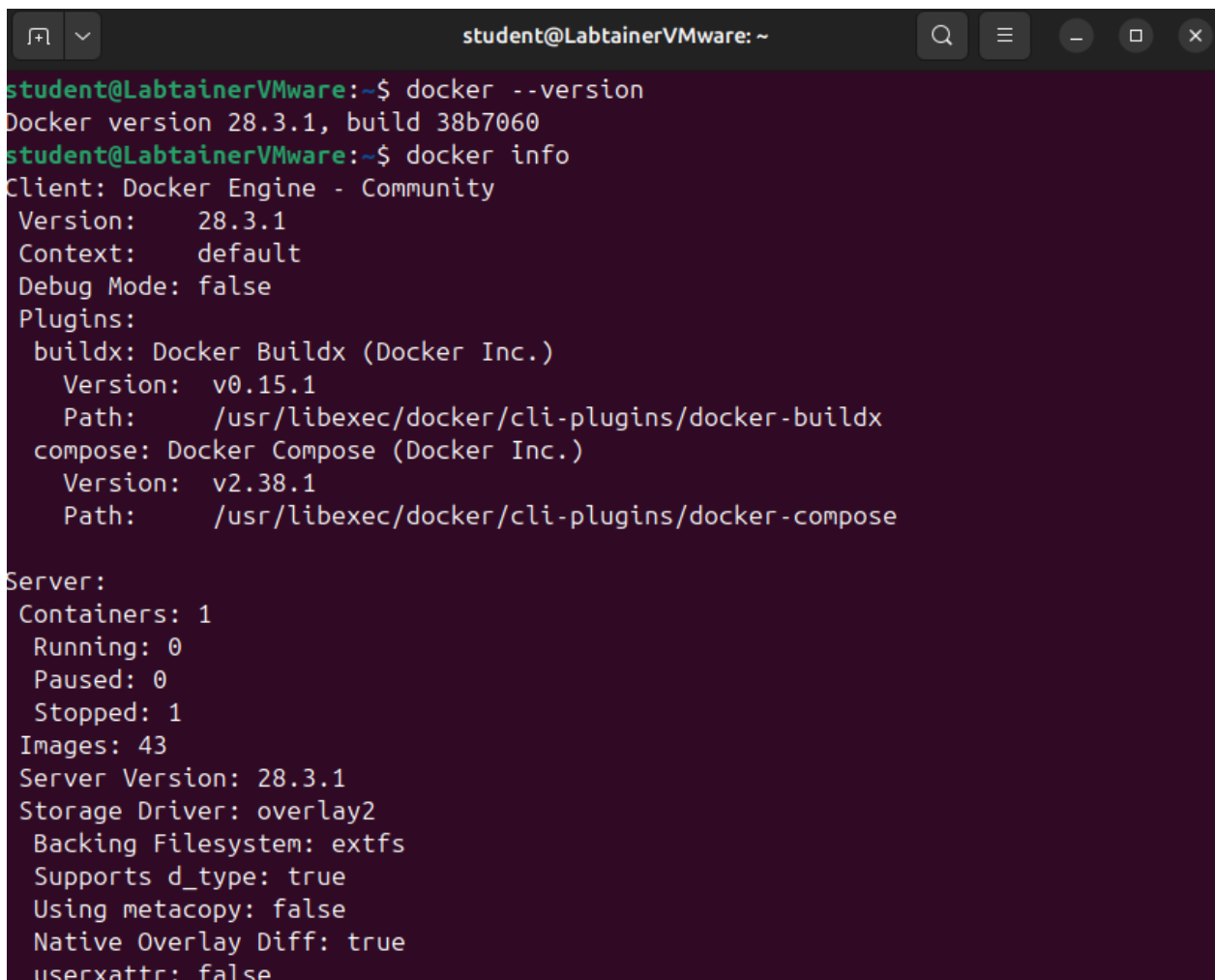
1.7.1 Kiểm Tra Phiên Bản và Thông Tin Hệ Thống

`docker --version`



```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ docker --version  
Docker version 28.3.1, build 38b7060  
student@LabtainerVMware:~$
```

`docker info`



```
student@LabtainerVMware:~$ docker --version  
Docker version 28.3.1, build 38b7060  
student@LabtainerVMware:~$ docker info  
Client: Docker Engine - Community  
Version:      28.3.1  
Context:      default  
Debug Mode:   false  
Plugins:  
  buildx: Docker Buildx (Docker Inc.)  
    Version:  v0.15.1  
    Path:      /usr/libexec/docker/cli-plugins/docker-buildx  
  compose: Docker Compose (Docker Inc.)  
    Version:  v2.38.1  
    Path:      /usr/libexec/docker/cli-plugins/docker-compose  
Server:  
Containers: 1  
  Running: 0  
  Paused: 0  
  Stopped: 1  
Images: 43  
Server Version: 28.3.1  
Storage Driver: overlay2  
  Backing Filesystem: extfs  
  Supports d_type: true  
  Using metacopy: false  
  Native Overlay Diff: true  
userxattr: false
```

1.7.2 Làm Việc Với Images

Tải image từ Docker Hub

`docker pull <image-name>`

Liệt kê tất cả images

docker images

Xóa image

docker rmi <image-id>

Tìm kiếm image trên Docker Hub

docker search <image-name>

1.7.3 Làm Việc Với Containers

Chạy container từ image

docker run <image-name>

Chạy container ở chế độ background

docker run -d <image-name>

Chạy container với port mapping

docker run -p <host-port>:<container-port> <image-name>

Liệt kê containers đang chạy

docker ps

Liệt kê tất cả containers

docker ps -a

Dừng container

docker stop <container-id>

Xóa container

docker rm <container-id>

Truy cập vào container đang chạy

docker exec -it <container-id> bash

Lệnh Docker Compose cơ bản

Khởi động tất cả services

docker-compose up

Khởi động ở background

docker-compose up -d

Dừng và xóa containers

docker-compose down

Xem logs

docker-compose logs

Build lại services

docker-compose build

Khởi động lại service cụ thể

docker-compose restart web

1.8 Quản Lý Dữ Liệu với Docker Volumes

Docker Volumes là cơ chế lưu trữ dữ liệu bên ngoài containers, đảm bảo dữ liệu không bị mất khi container bị xóa.

Các loại volumes

1. **Named Volumes:** Được quản lý hoàn toàn bởi Docker
2. **Bind Mounts:** Mount trực tiếp thư mục từ host
3. **Anonymous Volumes:** Volumes tạm thời không có tên

Các lệnh làm việc với volumes

Tạo volume

docker volume create my-volume

Liệt kê volumes

docker volume ls

Kiểm tra thông tin volume

docker volume inspect my-volume

Chạy container với volume

docker run -v my-volume:/data nginx

Bind mount

docker run -v /host/path:/container/path nginx

Xóa volume

docker volume rm my-volume

Xóa tất cả volumes không sử dụng

docker volume prune

1.9 Quản Lý Mạng với Docker Networking

Docker cung cấp nhiều loại mạng để containers có thể giao tiếp với nhau⁸:

Các loại network drivers

- **Bridge:** Mạng mặc định cho containers standalone
- **Host:** Container sử dụng trực tiếp network của host

- **None:** Container không có network interface
- **Overlay:** Cho Docker Swarm và multi-host networking

Lệnh networking cơ bản

Liệt kê networks

docker network ls

Tạo network

docker network create my-network

Kiểm tra thông tin network

docker network inspect my-network

Kết nối container với network

docker network connect my-network container-name

Ngắt kết nối

docker network disconnect my-network container-name

Xóa network

docker network rm my-network

1.10 Bài tập Thực hành

Bài tập 1: Container cơ bản

1. Chạy container nginx với port mapping 8080:80
2. Tạo container MySQL với password và port tùy chỉnh
3. Thực hành mount volume để lưu trữ dữ liệu

1. Chạy container nginx với port mapping 8080:80

Pull image nginx

```

student@LabtainerVMware: ~
student@LabtainerVMware:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
3da95a905ed5: Pull complete
5c8e51cf0087: Pull complete
9bbbd7ee45b7: Pull complete
48670a58a68f: Pull complete
ce7132063a56: Pull complete
23e05839d684: Pull complete
ee95256df030: Pull complete
Digest: sha256:93230cd54060f497430c7a120e2347894846a81b6a5dd2110f7362c5423b4abc
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
student@LabtainerVMware:~$

```

Chạy container với port mapping

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ docker run -d --name nginx-server -p 8080:80 nginx  
5c3abec971e1eab0448baadb8909f0c35dada159ab811d6309d26c7667bbe24d  
student@LabtainerVMware:~$ docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS  
PORTS  
5c3abec971e1   nginx    "/docker-entrypoint...." 16 seconds ago Up 16 seconds  
0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   nginx-server  
student@LabtainerVMware:~$ curl http://localhost:8080  
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome to nginx!</title>  
<style>  
html { color-scheme: light dark; }  
body { width: 35em; margin: 0 auto;  
font-family: Tahoma, Verdana, Arial, sans-serif; }  
</style>  
</head>  
<body>  
<h1>Welcome to nginx!</h1>  
<p>If you see this page, the nginx web server is successfully installed and  
working. Further configuration is required.</p>  
  
<p>For online documentation and support please refer to  
<a href="http://nginx.org/">nginx.org</a>.<br/>  
Commercial support is available at  
<a href="http://nginx.com/">nginx.com</a>.</p>  
  
<p><em>Thank you for using nginx.</em></p>  
</body>  
</html>
```

2. Tạo container MySQL với password và port tùy chỉnh

Chạy container MySQL:

```
docker run -d --name test-mysql -e MYSQL_ROOT_PASSWORD=strong_password -p 3307:3306 mysql
```

- -e MYSQL_ROOT_PASSWORD=strong_password: Thiết lập password cho user root
- -p 3307:3306: Map port 3307 của host với port 3306 mặc định của MySQL

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ docker run -d --name test-mysql -e MYSQL_ROOT_PASSWORD=strong_password -p 3307:3306 mysql  
Unable to find image 'mysql:latest' locally  
latest: Pulling from library/mysql  
3d2798b2072a: Pull complete  
f4b2a1d21561: Pull complete  
515701765b17: Pull complete  
c55f07fe1b6e: Pull complete  
610c16d564ee: Pull complete  
9db7a8f5c310: Pull complete  
e6048afa6840: Pull complete  
56483034e3f1: Pull complete  
4d45e5094a07: Pull complete  
3b536f25676c: Pull complete  
Digest: sha256:f1049ce35b3986b84c08184de43a0b2109ae037a4a10a23ecf373a893daeadf7  
Status: Downloaded newer image for mysql:latest  
9f90d0684d923ed18dffd5a61595e76e74061c9b2757f89378bbbbac0b72b3  
student@LabtainerVMware:~$
```

Kiểm tra port mapping:

docker port test-mysql

Kết nối tới MySQL từ host:

mysql --host=127.0.0.1 --port=3307 -u root -p

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ mysql --host=127.0.0.1 --port=3307 -u root -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 11  
Server version: 9.3.0 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2025, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

3. Thực hành mount volume để lưu trữ dữ liệu

Tạo named volume và chạy container với volume mount

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ docker volume create my_data_volume  
my_data_volume  
student@LabtainerVMware:~$ docker run -d \  
  --name my_postgres \  
  -e POSTGRES_PASSWORD=mysecretpassword \  
  -v my_data_volume:/var/lib/postgresql/data \  
  -p 5432:5432 \  
  postgres  
Unable to find image 'postgres:latest' locally  
latest: Pulling from library/postgres  
8da95a905ed5: Already exists  
0374722b7db6: Pull complete  
9e48f0d5503b: Pull complete  
c883d509c82b: Pull complete  
e9068395fab4: Pull complete  
f68d74b3ee3b: Pull complete  
420b84acc7f: Pull complete  
761152bb4395: Pull complete  
1eb73c80cbec: Pull complete  
0bce49c39241: Pull complete  
f028345d1934: Pull complete  
57bfaf72b9da: Pull complete  
ebb41c3a1a26: Pull complete  
a56d6197194e: Pull complete  
Digest: sha256:3962158596daaef3682838cc8eb0e719ad1ce520f88e34596ce8d5de1b6330a1  
Status: Downloaded newer image for postgres:latest  
1dcfc35dc54f0d8a084f39eec92592c554630ea029ea3ba441b1cec286f5399  
student@LabtainerVMware:~$
```

Liệt kê volume và kiểm tra volume

```
student@LabtainerVMware: ~  
student@LabtainerVMware:~$ docker volume ls  
DRIVER      VOLUME NAME  
local       827de8c71fb3df5fd9dc83bd107378309e45872952b445cc3db568a194acfa5d  
local       my_data_volume  
student@LabtainerVMware:~$ docker volume inspect my_data_volume  
[  
  {  
    "CreatedAt": "2025-07-05T02:22:17-07:00",  
    "Driver": "local",  
    "Labels": null,  
    "Mountpoint": "/var/lib/docker/volumes/my_data_volume/_data",  
    "Name": "my_data_volume",  
    "Options": null,  
    "Scope": "local"  
  }  
]  
student@LabtainerVMware:~$
```

Bài tập 2: Dockerfile

1. Tạo Dockerfile cho ứng dụng Python Flask đơn giản
2. Build image và chạy container
3. Optimize Dockerfile với multi-stage builds

1. Tạo Dockerfile cho ứng dụng Python Flask đơn giản

Bước 1: Tạo cấu trúc thư mục

```
mkdir flask-docker
```

```
cd flask-docker
```

Bước 2: Tạo file Flask app

Tạo file app.py:

```
from flask import Flask  
app = Flask(__name__)  
@app.route('/')  
def hello_world():  
    return '<h1>Hello from Flask & Docker!</h1>'  
  
@app.route('/health')  
def health_check():  
    return {'status': 'healthy', 'message': 'Flask app is running'}  
  
if __name__ == "__main__":  
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Bước 3: Tạo file requirements.txt

Flask==3.0.0

Bước 4: Tạo Dockerfile đơn giản

```
# Sử dụng Python official image
FROM python:3.11-slim
# Thiết lập working directory
WORKDIR /app
# Copy requirements file
COPY requirements.txt .
# Cài đặt dependencies
RUN pip install --no-cache-dir -r requirements.txt
# Copy application code
COPY . .
# Expose port
EXPOSE 5000
# Chạy application
CMD ["python", "app.py"]
```

2. Build image và chạy container

Build Docker image

docker build -t flask-app .


```
student@LabtainerVMware: ~/flask-docker
student@LabtainerVMware:~/flask-docker$ docker build -t flask-app .
[+] Building 30.5s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 382B                                0.1s
=> [internal] load metadata for docker.io/library/python:3.11-slim 6.5s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:139020233cc412ef 10.7s
=> => resolve docker.io/library/python:3.11-slim@sha256:139020233cc412ef 0.0s
=> => sha256:483d0dd375188d7d3b2d66116d5974d2b67e6988c614 3.51MB / 3.51MB 1.3s
=> => sha256:02a5d22e0d6f85a6ac1c7fb356e9eed39a981dec1 16.21MB / 16.21MB 3.8s
=> => sha256:471797cdda8c4cd4a9795c8cb56078e627b3fc7486fd8574 249B / 249B 2.7s
=> => sha256:139020233cc412efe4c8135b0efe1c7569dc8b28ddd8 9.13kB / 9.13kB 0.0s
=> => sha256:153bae509ec02c9ac789e2e35f3cbe94be446b59c3af 1.75kB / 1.75kB 0.0s
=> => sha256:0b14a859cd8a15104c5f194ef813fccc2749d74bc7 5.37kB / 5.37kB 0.0s
=> => extracting sha256:483d0dd375188d7d3b2d66116d5974d2b67e6988c6146eb2c 1.3s
=> => extracting sha256:02a5d22e0d6f85a6ac1c7fb356e9eed39a981dec1fac1205 6.5s
=> => extracting sha256:471797cdda8c4cd4a9795c8cb56078e627b3fc7486fd85748 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 799B                                     0.0s
=> [2/5] WORKDIR /app                                           0.3s
=> [3/5] COPY requirements.txt .                                0.1s
=> [4/5] RUN pip install --no-cache-dir -r requirements.txt     11.3s
=> [5/5] COPY . .                                               0.1s
=> exporting to image                                           0.9s
=> => exporting layers                                           0.9s
=> => writing image sha256:8fce4c5cae4f786577e7b7847b0bbbc40578bf182a7659 0.0s
=> => naming to docker.io/library/flask-app                       0.0s
student@LabtainerVMware:~/flask-docker$
```

Kiểm tra image đã tạo:

docker images

Chạy container

docker run -d --name my-flask-app -p 5000:5000 flask-app

```
student@LabtainerVMware: ~/flask-docker
Files student@LabtainerVMware:~/flask-docker$ docker run -d --name my-flask-app -p 5000:5000 flask-app
2b05bd818a954d19e6fd0727d5586d3a1a9f32792dbf85b143a358fdc633f9e1
student@LabtainerVMware:~/flask-docker$
```

Kiểm tra container đang chạy:

docker ps

Test ứng dụng:

curl http://localhost:5000

curl <http://localhost:5000/health>

```
student@LabtainerVMware: ~/flask-docker
student@LabtainerVMware:~/flask-docker$ curl http://localhost:5000
<h1>Hello from Flask & Docker!</h1>student@LabtainerVMware:~/flask-docker$ curl h
curl http://localhost:5000/health
{
  "message": "Flask app is running",
  "status": "healthy"
}
student@LabtainerVMware:~/flask-docker$
```

Xem logs của container:

docker logs my-flask-app

```
student@LabtainerVMware: ~/flask-docker
student@LabtainerVMware:~/flask-docker$ docker logs my-flask-app
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.5:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 621-830-486
172.17.0.1 - - [05/Jul/2025 09:43:57] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [05/Jul/2025 09:44:03] "GET /health HTTP/1.1" 200 -
student@LabtainerVMware:~/flask-docker$
```

Bài tập 3: Docker Compose

1. Tạo stack WordPress + MySQL bằng Docker Compose
2. Thêm phpMyAdmin để quản lý database
3. Cấu hình volumes cho persistent data

Tạo file docker-compose.yml với nội dung sau:

```
version: '3.9'

services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
```

MYSQL_USER: wordpress
MYSQL_PASSWORD: wordpress

volumes:

- db_data:/var/lib/mysql

networks:

- wpsite

phpmyadmin:

depends_on:

- db

image: phpmyadmin/phpmyadmin

restart: always

ports:

- '8080:80'

environment:

PMA_HOST: db

MYSQL_ROOT_PASSWORD: password

networks:

- wpsite

wordpress:

depends_on:

- db

image: wordpress:latest

restart: always

ports:

- '8000:80'

environment:

WORDPRESS_DB_HOST: db:3306

WORDPRESS_DB_USER: wordpress

WORDPRESS_DB_PASSWORD: wordpress

WORDPRESS_DB_NAME: wordpress

volumes:

- ./wp-app:/var/www/html

networks:

- wpsite

networks:

wpsite:

volumes:

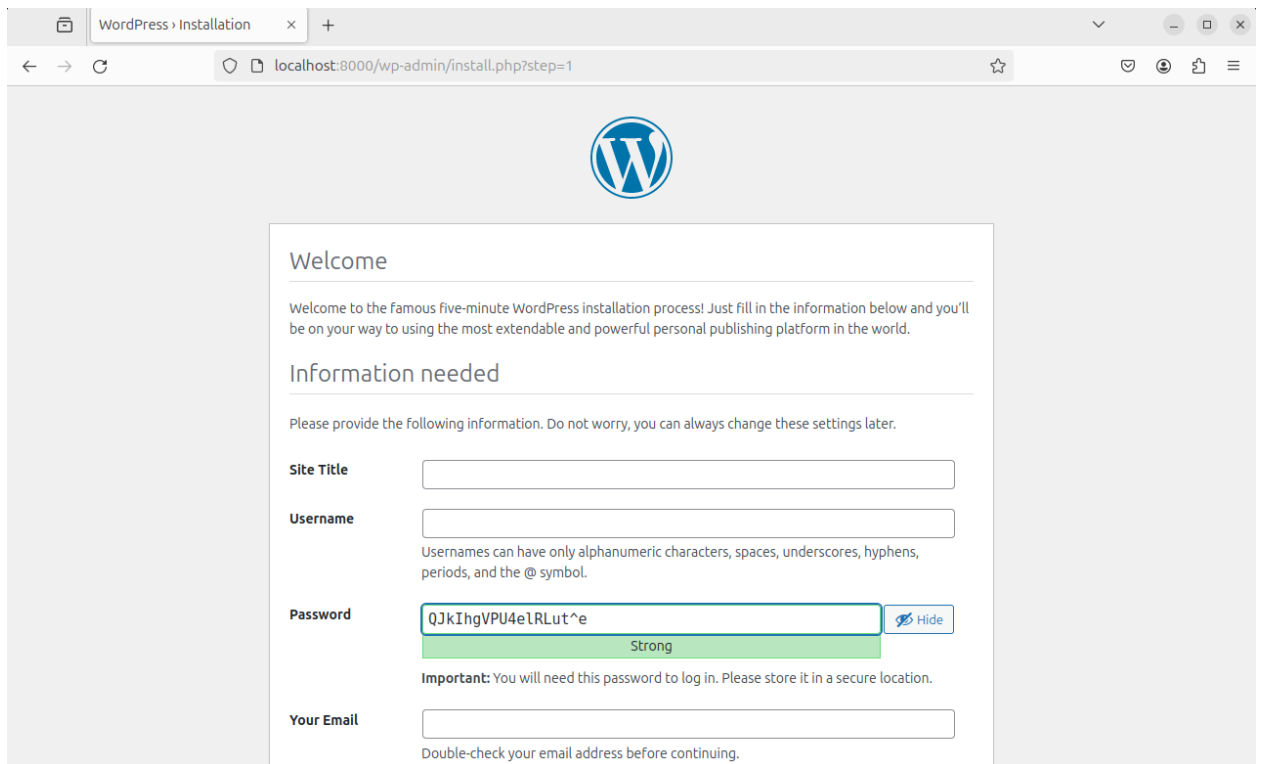
db_data:

Chạy lệnh: *docker-compose up -d*

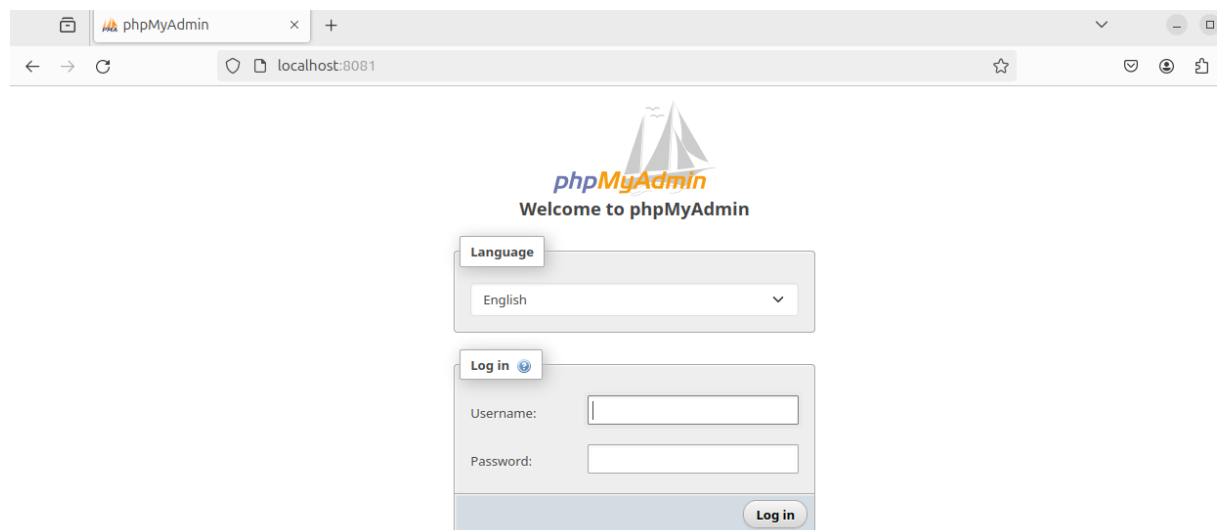
```
student@LabtainerVMware: ~/wordpress-stack
student@LabtainerVMware:~/wordpress-stack$ ls
docker-compose.yml
student@LabtainerVMware:~/wordpress-stack$ docker compose up -d
WARN[0000] /home/student/wordpress-stack/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 57/57
  ✓ wordpress Pulled 154.9s
  ✓ db Pulled 143.4s
  ✓ phpmyadmin Pulled 110.9s
```

Truy cập các dịch vụ

- **WordPress:** <http://localhost:8000>



- **phpMyAdmin:** <http://localhost:8080>
 - Đăng nhập bằng user: root, password: password hoặc user: wordpress, password: wordpress.
 - Do xung đột cổng 8080 ở bài tập 1, chuyển sang port 8081 sửa trong file docker-compose.yml



Một số thao tác liên quan

```
student@LabtainerVMware: ~/wordpress-stack
student@LabtainerVMware:~/wordpress-stack$ docker volume ls
DRIVER      VOLUME NAME
local       827de8c71fb3df5fd9dc83bd107378309e45872952b445cc3db568a194acfa5d
local       my_data_volume
local       wordpress-stack_db_data
student@LabtainerVMware:~/wordpress-stack$ docker volume inspect wordpress-stack_db_data
[
  {
    "CreatedAt": "2025-07-05T02:54:01-07:00",
    "Driver": "local",
    "Labels": {
      "com.docker.compose.config-hash": "656a02aa4b71169a2d3aace7d440339623a99a1d510495fc457c32cc85c1869f",
      "com.docker.compose.project": "wordpress-stack",
      "com.docker.compose.version": "2.38.1",
      "com.docker.compose.volume": "db_data"
    },
    "Mountpoint": "/var/lib/docker/volumes/wordpress-stack_db_data/_data",
    "Name": "wordpress-stack_db_data",
    "Options": null,
    "Scope": "local"
  }
]
```

Bài tập 4: Networking

1. Tạo custom network
2. Chạy nhiều containers trong cùng network
3. Test connectivity giữa các containers

Tạo một mạng bridge mới tên là my-custom-network:

docker network create my-custom-network

```
student@LabtainerVMware:~/wordpress-stack$ docker network create my-custom-network
cbbb7d420d834d8aa68cf8157ab41fe32ee708ef84fca42639ae8bda05bd669a
student@LabtainerVMware:~/wordpress-stack$ docker network ls
NETWORK ID        NAME                DRIVER              SCOPE
a76ffc04d6fc      bridge              bridge              local
d305071c6b7d      host                host                local
cbbb7d420d83      my-custom-network   bridge              local
4e780c024ea8      none                null                local
83ef25512f96      wordpress-stack_wpsite bridge              local
student@LabtainerVMware:~/wordpress-stack$
```

chạy 2 container (nginx và alpine) cùng nằm trong network này

```

student@LabtainerVMware:~/wordpress-stack$ docker run -d --name web1 --network my-custom-network nginx
20636cc5f95c9d9b701d4cd4f62bc7eb2f29aebef7f5d1fb3976d0757ab18d59b
student@LabtainerVMware:~/wordpress-stack$ docker run -d --name web2 --network my-custom-network alpine sleep 1000
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
fe07684b16b8: Pull complete
Digest: sha256:8a1f59ffb675680d47db6337b49d22281a139e9d709335b492be023728e11715
Status: Downloaded newer image for alpine:latest
81d493758d69d0a93bb72d5eb109b0cc06d34d88d86c0768c2c0faeada46d28a
student@LabtainerVMware:~/wordpress-stack$

```

Kết nối vào container web2:

`docker exec -it web2 sh`

Ping tới container web1 bằng tên:

`ping web1`

```

student@LabtainerVMware:~/wordpress-stack$ docker run -d --name web1 --network my-custom-network nginx
20636cc5f95c9d9b701d4cd4f62bc7eb2f29aebef7f5d1fb3976d0757ab18d59b
student@LabtainerVMware:~/wordpress-stack$ docker run -d --name web2 --network my-custom-network alpine sleep 1000
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
fe07684b16b8: Pull complete
Digest: sha256:8a1f59ffb675680d47db6337b49d22281a139e9d709335b492be023728e11715
Status: Downloaded newer image for alpine:latest
81d493758d69d0a93bb72d5eb109b0cc06d34d88d86c0768c2c0faeada46d28a
student@LabtainerVMware:~/wordpress-stack$ docker exec -it web2 sh
/ # ping web1
PING web1 (172.19.0.2): 56 data bytes
64 bytes from 172.19.0.2: seq=0 ttl=64 time=3.254 ms
64 bytes from 172.19.0.2: seq=1 ttl=64 time=0.283 ms
64 bytes from 172.19.0.2: seq=2 ttl=64 time=0.246 ms
^C
--- web1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.246/1.261/3.254 ms
/ #

```

Tài nguyên Học tập

1. **Documentation chính thức:** docs.docker.com
2. **Docker Hub:** hub.docker.com - Repository các images
3. **Tutorials interactives:** docker.com/101-tutorial
4. **Video tutorials:** Nhiều channel YouTube như "TechWorld with Nana", "Hỏi Dân IT"