## **TERMINAL**

Terminal là một giao diện dòng lệnh nơi bạn nhập lệnh và nhận kết quả. Nó có thể là một ứng dụng đồ họa (như GNOME Terminal, Konsole trên Linux, hoặc Terminal trên macOS) hoặc một thiết bị ảo (như tty trên các hệ thống Unix).

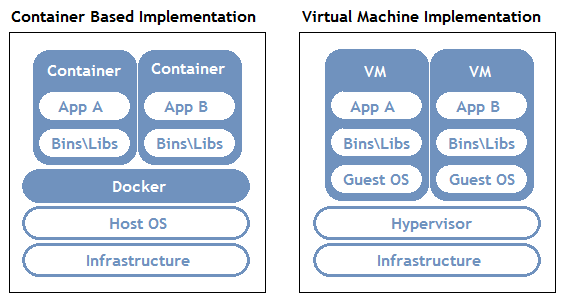
Nói cách khác, terminal là nơi bạn gõ lệnh, trong khi Bash là chương trình thực thi các lệnh đó. Bạn có thể sử dụng nhiều shell khác nhau (như zsh, fish) trong cùng một terminal.

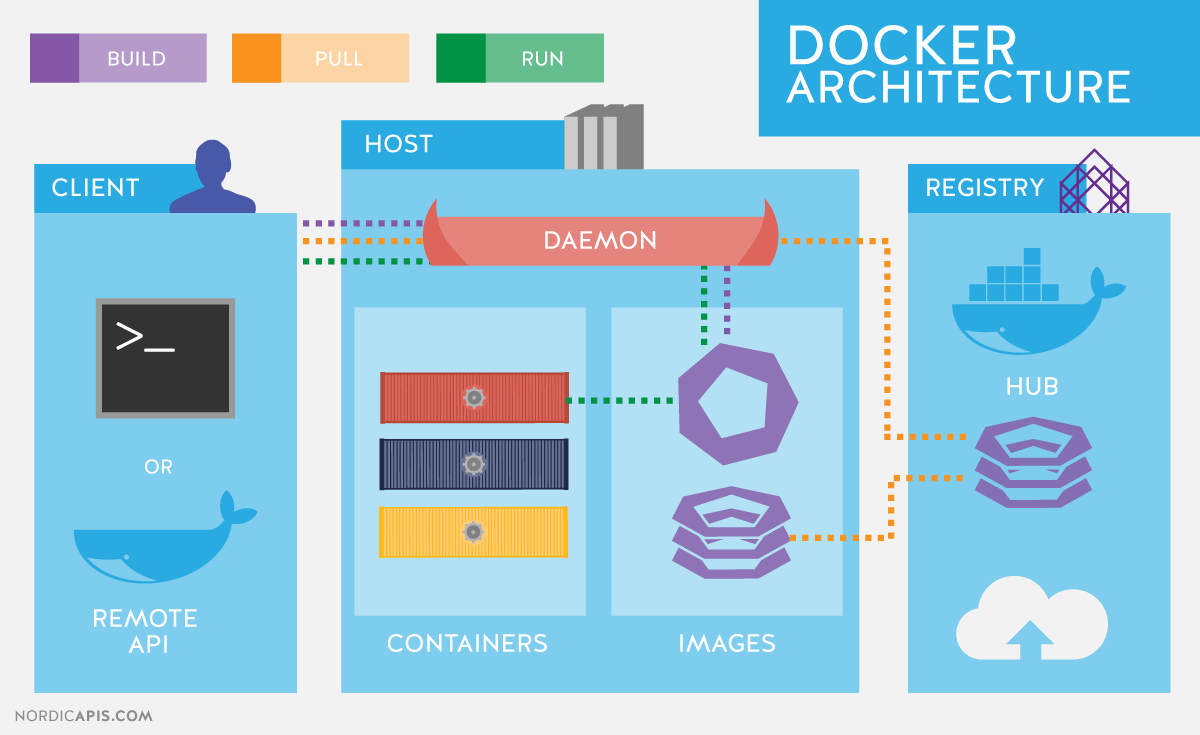
## **BASH**

**Bash (Bourne Again Shell)** là một shell và ngôn ngữ kịch bản (scripting language) phổ biến trên hệ điều hành Unix và Unix-like như Linux và macOS. Đây là một công cụ dòng lệnh mạnh mẽ được sử dụng để thực hiện các lệnh, viết script tự động hóa các tác vụ, và quản lý hệ thống.

1. **ls**: Liệt kê các tệp và thư mục trong thư mục hiện tại.
2. **cd [directory]**: Thay đổi thư mục hiện tại.
3. **pwd**: Hiển thị đường dẫn đến thư mục hiện tại.
4. **mkdir [directory]**: Tạo một thư mục mới.
5. **rmdir [directory]**: Xóa một thư mục (chỉ khi thư mục trống).
6. **rm [file]**: Xóa một tệp.
7. **cp [source] [destination]**: Sao chép tệp hoặc thư mục từ nguồn đến đích.
8. **mv [source] [destination]**: Di chuyển hoặc đổi tên tệp hoặc thư mục.
9. **cat [file]**: Hiển thị nội dung của tệp.
10. **more [file]** hoặc **less [file]**: Xem nội dung tệp theo trang.
11. **head [file]**: Hiển thị các dòng đầu tiên của tệp.
12. **tail [file]**: Hiển thị các dòng cuối cùng của tệp.
13. **echo [text]**: Hiển thị văn bản hoặc giá trị của biến.
14. **chmod [permissions] [file]**: Thay đổi quyền truy cập của tệp.
15. **chown [user]:[group] [file]**: Thay đổi chủ sở hữu và nhóm của tệp.
16. **find [directory] -name [filename]**: Tìm tệp theo tên trong thư mục.
17. **grep [pattern] [file]**: Tìm kiếm các dòng trong tệp khớp với mẫu.
18. **ps**: Liệt kê các tiến trình đang chạy.
19. **top**: Hiển thị các tiến trình đang chạy và thông tin hệ thống theo thời gian thực.
20. **kill [PID]**: Gửi tín hiệu để dừng tiến trình có ID là PID.
21. **history**: Hiển thị lịch sử các lệnh đã thực thi.
22. **man [command]**: Hiển thị trang hướng dẫn (manual) cho lệnh.

## **DOCKER OVERVIEW**





1. **Docker Client:** Giao diện người dùng để gửi lệnh, như docker run, docker build, đến Docker Daemon.
2. **Docker Host:** Chứa Docker Daemon, chịu trách nhiệm tạo, chạy và quản lý các container. Thành phần chính gồm:

* **Docker Daemon**: Xử lý các lệnh từ Docker Client.
* **Containers**: Các thực thể runtime chứa ứng dụng.
* **Images**: Template chứa ứng dụng và môi trường để tạo container.
* **Volumes** **và** **Networks**: Quản lý dữ liệu và giao tiếp giữa các container.

1. **Docker Registry**: Kho lưu trữ các Docker images. Phổ biến nhất là Docker Hub, hoặc registry nội bộ.

* **Docker** right now is the best way to have an portable application that can be work on any machine, any Operating System and any infrastructure. So, you do not have to waste your time and effort thinking about the machines and the infrastructures and operating system.

[Các Lệnh Docker Cơ Bản | Docker CheatSheet - Viblo](https://viblo.asia/p/cac-lenh-docker-co-ban-docker-cheatsheet-GyZJZjKGJjm)

## **LAB 1 - RUNNING YOUR FIRST CONTAINER**

### Step 1 : Run your first container

1. **Open a terminal on your local computer**

A screenshot of a computer

Description automatically generated

1. **Run*****docker container run -t ubuntu top***

A screenshot of a computer

Description automatically generated

Khi chạy lệnh docker container run -it ubuntu top, Docker sẽ:

1. Tải image: Docker kiểm tra xem image ubuntu đã có sẵn trên máy chưa. Nếu chưa, nó sẽ tải về từ Docker Hub.
2. Chạy container: Docker khởi tạo container từ image ubuntu và chạy lệnh top, hiển thị thông tin về tiến trình bên trong container.
3. Namespace isolation: Container sử dụng các namespace của Linux để cô lập các tài nguyên (PID, hệ thống file) khỏi hệ điều hành chính. Vì vậy, khi chạy lệnh top trong container, chỉ thấy tiến trình trong container, không thấy các tiến trình từ host.
4. Chia sẻ kernel: Container không có kernel riêng, nó sử dụng kernel của hệ thống host. Image Ubuntu chỉ cung cấp hệ thống file và công cụ của Ubuntu.
5. **Inspect the container with****docker container exec**

A screenshot of a computer

Description automatically generated

1. Chạy lệnh docker container exec: Bạn có thể "vào" container đang chạy bằng cách sử dụng lệnh này. Ví dụ: docker container exec -it <CONTAINER\_ID> bash (sh) sẽ khởi chạy một shell bash trong container.
2. Isolation của container: Container không phải là một máy ảo, mà là một nhóm các tiến trình chạy cách ly trong không gian PID riêng. Bạn có thể thấy điều này khi chạy lệnh ps -ef bên trong container; chỉ các tiến trình của container (như top, bash, ps) xuất hiện.
3. Namespace trong Linux: Docker sử dụng nhiều namespace của Linux để cung cấp cách ly cho container, như PID, NET, IPC, USER, và UTC.
4. Chạy container trên Windows/Mac: Docker trên các hệ điều hành này sử dụng một Linux subsystem để chạy các container Linux.
5. **Clean up the container running the top processes by typing: <ctrl>-c, list all containers and remove the containers by their id.**

docker ps -a

docker rm <CONTAINER ID>

A screenshot of a computer

Description automatically generated

### Step 2 : Run Multiple Containers

1. Docker Hub: Là registry công cộng chứa các Docker images chính thức và cộng đồng. Bạn có thể tìm kiếm các image đã được xác thực và chính thức để đảm bảo an toàn và tính tương thích cho môi trường sản xuất.
2. Chạy Nginx:

* Lệnh: docker container run --detach --publish 8080:80 --name nginx nginx
* Điều này kéo image Nginx từ Docker Hub, khởi chạy Nginx server trong background, và ánh xạ port 80 của container đến port 8080 của host.

A computer screen with white text

Description automatically generated

* Truy cập Nginx qua localhost:8080.

A screenshot of a computer

Description automatically generated

1. Chạy MongoDB:

* Lệnh: docker container run --detach --publish 8081:27017 --name mongo mongo:4.4

A computer screen with white text

Description automatically generated

* Điều này kéo image MongoDB phiên bản 4.4, khởi chạy MongoDB server trong background, và ánh xạ port 27017 của container đến port 8081 của host.
* Truy cập MongoDB qua localhost:8081 (cảnh báo vì MongoDB không cung cấp giao diện HTTP trên port này).

A screenshot of a computer

Description automatically generated

1. Kiểm tra các container đang chạy:

* Lệnh: docker container ls
* Bạn sẽ thấy các container Nginx và MongoDB đang chạy cùng với thông tin về port mappings và tên container.

A black screen with white text

Description automatically generated

1. Lợi ích của container:

* Cung cấp sự cách ly giữa các ứng dụng và các phiên bản khác nhau của phần mềm.
* Không cần cài đặt thêm phần mềm trên host; các dependencies được đóng gói trong container.
* Có thể chạy nhiều container trên cùng một host để tận dụng tài nguyên hiệu quả.

Sử dụng Docker Hub và Docker containers giúp dễ dàng triển khai các dịch vụ và quản lý môi trường phát triển và sản xuất mà không gặp phải các vấn đề xung đột hoặc phụ thuộc phần mềm.

### Step 3 : Clean Up

1. Danh sách các container đang chạy : docker container ls

A black screen with white text

Description automatically generated

1. Dừng các container : docker container stop <id>

A black background with white text

Description automatically generated

1. Xóa các container đã dừng : docker system prune

A computer screen with white text

Description automatically generated

## **LAB 2 - ADDING VALUE WITH CUSTOM DOCKER IMAGES**

### Step 1 : Create a python app (without using Docker)

echo 'from flask import Flask

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

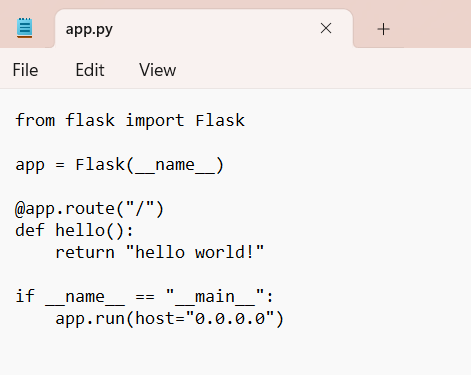
@app.route("/")

def hello():

return "hello world!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0")' > app.py



$ python3 –version

$ pip3 –version

A black screen with white text

Description automatically generated

$ pip3 install flask

A computer screen shot of a program

Description automatically generated

$ python3 app.py

A screen shot of a computer

Description automatically generated

### Step 2 : Create and build the Docker Image

1. **Tạo một Dockerfile**

*echo 'FROM python:3.8-alpine RUN pip install flask CMD ["python","app.py"] COPY app.py /app.py' > Dockerfile*

* **FROM python:3.8-alpine**: Sử dụng hình ảnh Python 3.8 dựa trên Alpine Linux làm hình ảnh cơ sở.
* **RUN pip install flask:** Cài đặt Flask bên trong hình ảnh.
* **CMD ["python","app.py"]:** Xác định lệnh sẽ chạy khi container khởi động (trong trường hợp này là chạy app.py với Python).
* **COPY app.py /app.py:** Sao chép tệp app.py từ máy local vào container.

1. **Xây dựng Docker Image**

*docker image build -t python-hello-world .*

A screen shot of a computer screen

Description automatically generated

* **‘-t python-hello-world’**: Gán tag cho hình ảnh với tên python-hello-world.
* **‘ . ’**: Xác định bối cảnh xây dựng (trong trường hợp này là thư mục hiện tại).

1. **Xác nhận Image**

*docker image ls*

A screenshot of a computer screen

Description automatically generated

1. **Xem lịch sử Image**

*docker history python-hello-world*

*docker history python:3.8-alpine*

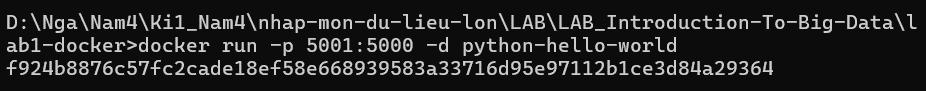
A screen shot of a computer program

Description automatically generated

### Step 3 : Run the Docker Image

1. **Chạy Docker Image**

*docker run -p 5001:5000 -d python-hello-world*



**-p 5001:5000**: Ánh xạ cổng 5000 trong container ra cổng 5001 trên máy chủ. Nếu cổng 5001 đã được sử dụng bởi một ứng dụng khác trên máy chủ của bạn, bạn có thể thay thế bằng một giá trị khác, chẳng hạn như 5002.

**-d**: Chạy container ở chế độ nền (detached mode).

**python-hello-world**: Tên của Docker image bạn đã xây dựng.

1. **Truy cập ứng dụng**

Mở trình duyệt và điều hướng đến localhost:5001 để xem kết quả. Bạn sẽ thấy "hello world!" hiển thị trên trình duyệt.

A screenshot of a computer

Description automatically generated

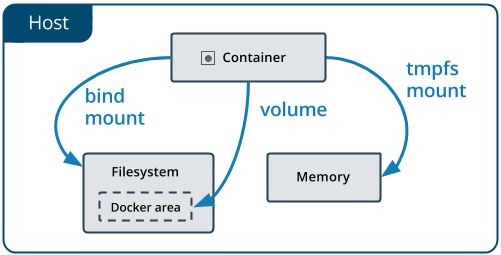
1. **Kiểm tra Log Output của Container**

*docker container logs [container id]*

A screen shot of a computer

Description automatically generated

## **LAB 3 - MANAGING DATA IN CONTAINERS**



**Volumes (Dung lượng)**:

* **Anonymous Volumes**: Được Docker tự động tạo và quản lý. Không có tên cụ thể và thường được sử dụng khi bạn không cần phải chia sẻ dữ liệu giữa các container.
* **Named Volumes**: Có thể đặt tên và dễ dàng chia sẻ giữa các container. Docker quản lý chúng và lưu trữ trong thư mục /var/lib/docker/volumes/ trên host.
* **Host Volumes**: Thư mục trên host được gán trực tiếp vào container. Thường được sử dụng khi cần truy cập dữ liệu từ host mà không thông qua Docker.

**Bind Mounts**:

* **Bind Mounts**: Cho phép bạn ánh xạ một thư mục hoặc tệp từ host vào container. Bạn chỉ định đường dẫn đầy đủ trên host và container. Bind mounts có thể thay đổi hệ thống tệp của host.

**tmpfs Mounts**:

* **tmpfs Mounts**: Lưu trữ dữ liệu trong bộ nhớ tạm thời. Dữ liệu không được lưu trên đĩa và sẽ mất khi container dừng.