# DOCKER

## Giới thiệu

Docker là một nền tảng containerization, cho phép đóng gói ứng dụng cùng toàn bộ môi trường chạy (thư viện, cấu hình, hệ điều hành tối thiểu) vào một thực thể gọi là container.

So sánh với máy ảo (VM):

* VM: cần hypervisor, mỗi máy ảo có hệ điều hành riêng → nặng, tốn RAM/CPU.
* Docker: dùng chung kernel của host, container chỉ là tiến trình cô lập → nhẹ, khởi chạy nhanh.

Ưu điểm:

* Triển khai đồng nhất giữa môi trường Dev/Test/Prod.
* Dễ dàng chia sẻ và tái sử dụng.
* Phù hợp cho mô hình microservices và CI/CD.

## Kiến thức nền tảng

Docker hoạt động dựa trên các tính năng của nhân Linux, bao gồm Namespaces, Control Groups (Cgroups) và Union File System (UnionFS/OverlayFS). Đây là ba trụ cột kỹ thuật chính giúp container tồn tại.

### Namespaces – Cơ chế cô lập

Namespaces cung cấp khả năng cô lập tiến trình, mạng, hệ thống file và các tài nguyên khác, giúp mỗi container có môi trường riêng biệt, như một “máy ảo nhẹ”.

* PID namespace: tạo cây tiến trình riêng cho container.
* NET namespace: tạo stack mạng độc lập (interface, IP, routing table).
* MNT namespace: tách biệt không gian file system.
* UTS namespace: cho phép container có hostname riêng.
* IPC namespace: cô lập cơ chế giao tiếp liên tiến trình.

### Cgroups – Quản lý tài nguyên

Cgroups (Control Groups) giúp giới hạn, theo dõi và ưu tiên việc sử dụng tài nguyên (CPU, RAM, I/O, network). Nhờ đó, Docker đảm bảo mỗi container chỉ được cấp tài nguyên theo mức cấu hình, tránh tình trạng một container chiếm hết tài nguyên của host.

Ví dụ:

docker run --cpus=2 -m 512m myapp

→ Container chỉ được dùng tối đa 2 CPU và 512 MB RAM.

### UnionFS (OverlayFS/overlay2) – Hệ thống file theo lớp

UnionFS cho phép hợp nhất nhiều filesystem thành một. Docker sử dụng OverlayFS (overlay2) để quản lý image theo cơ chế layer.

* Image: gồm nhiều layer chỉ-đọc (Ubuntu → Node.js → App).
* Container: thêm một layer ghi (writable layer) khi chạy.

Cơ chế copy-on-write đảm bảo chỉ khi có thay đổi mới copy file từ lớp gốc sang lớp ghi, giúp tiết kiệm tài nguyên và khởi động nhanh.

## Các khái niệm chính

* Image: bản thiết kế (template/blueprint) bất biến của ứng dụng, gồm nhiều layers. Ví dụ: nginx:latest.
* Container: phiên bản đang chạy (instance) của image, là một môi trường cô lập.
* Registry: kho lưu trữ và chia sẻ image (ví dụ: Docker Hub, GCR, ECR).
* Dockerfile: file văn bản chứa tập hợp hướng dẫn để build image.
* Volume/Mount: cơ chế lưu trữ dữ liệu bền vững bên ngoài container, tách biệt khỏi vòng đời container.
* Network: hệ thống kết nối container ↔ container ↔ host, cho phép giao tiếp và phối hợp dịch vụ.

## Các lệnh cơ bản

### Image

docker pull nginx # tải image

docker images # liệt kê image

docker rmi nginx # xóa image

docker build -t myapp . # build image từ Dockerfile

### Container

docker run -d -p 8080:80 nginx # chạy container

docker ps -a # liệt kê container

docker logs -f <id> # xem log

docker exec -it <id> bash # vào container

docker stop <id> && docker rm <id> # dừng và xóa

### Volume

docker volume create myvol

docker run -d -v myvol:/data alpine

### Network

docker network create mynet

docker run -d --network mynet --name db postgres

docker run -d --network mynet --name api node:18

## COPY và docker cp

Trong Dockerfile:

WORKDIR /app

COPY . .

* Dấu . đầu tiên: thư mục host (context).
* Dấu . sau: thư mục trong image (/app).

Dùng docker cp:

docker cp ./file.txt myct:/app/ # host → container

docker cp myct:/app/logs ./logs # container → host

## Volume và Mount

* Volume: do Docker quản lý, bền vững.
* Bind mount: gắn trực tiếp thư mục/file host vào container.
* Tmpfs: dữ liệu trên RAM, mất khi container stop.

# Volume cho DB

docker run -d -v pgdata:/var/lib/postgresql/data postgres

# Bind mount cho code dev

docker run -it -v $PWD:/app -w /app node:18 bash

# Tmpfs

docker run -d --mount type=tmpfs,target=/cache redis

## Docker Network

* bridge: mặc định, cho phép container trong cùng mạng giao tiếp.
* host: dùng chung network host (Linux only).
* none: không có mạng.
* macvlan: container có IP thật trong LAN.
* overlay: cho cluster multi-host.

Ví dụ:

docker network create appnet

docker run -d --network appnet --name db postgres

docker run -d --network appnet --name api node:18

→ api gọi db:5432.

## Dockerfile và Multi-stage build

Ví dụ Node.js:

FROM node:18-alpine AS build

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

FROM node:18-alpine

WORKDIR /app

COPY --from=build /app .

EXPOSE 3000

CMD ["node", "server.js"]

→ Multi-stage build giúp giảm kích thước image.

## Docker Compose

version: "3.9"

services:

db:

image: postgres:15

volumes:

- dbdata:/var/lib/postgresql/data

api:

build: .

volumes:

- ./src:/app

ports:

- "3000:3000"

depends\_on:

- db

volumes:

dbdata:

→ Quản lý nhiều container trong một file YAML.

## Bảo mật và Quản trị

Không chạy container với root (dùng USER).

Quét lỗ hổng image bằng Trivy, Snyk.

Giới hạn tài nguyên:

docker run --cpus=1.5 --memory=512m myapp

Giám sát bằng Prometheus + cAdvisor.

Dọn dẹp tài nguyên không dùng:

docker system prune

docker volume prune

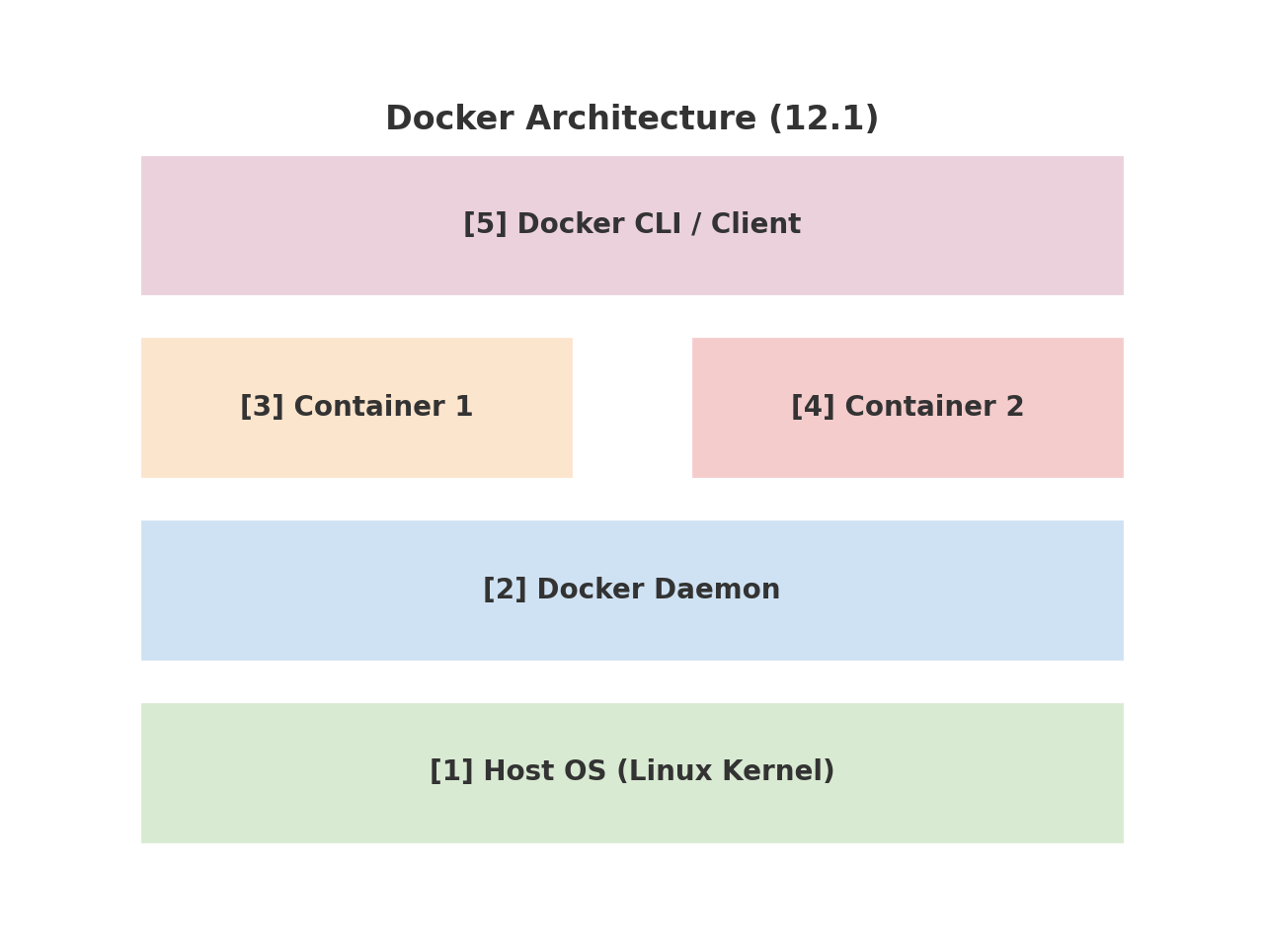
## Kết luận

Docker đã thay đổi cách thức triển khai và vận hành ứng dụng, mang lại sự đồng nhất, linh hoạt và tiết kiệm tài nguyên. Với cơ chế nền tảng từ Linux (Namespaces, Cgroups, UnionFS), Docker cung cấp một giải pháp containerization nhẹ, nhanh và đáng tin cậy.

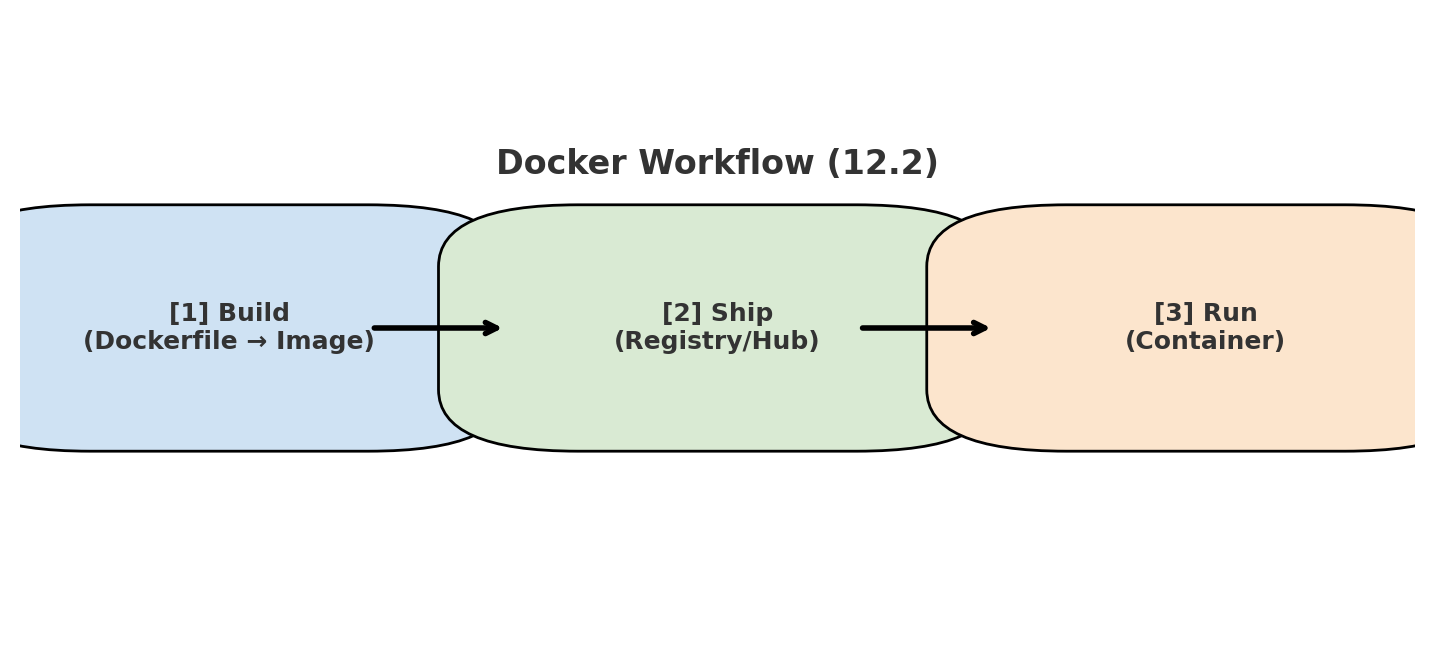
Trong môi trường phát triển, Docker giúp dễ dàng tái tạo môi trường; trong môi trường sản xuất, Docker kết hợp với các công cụ như Kubernetes mang lại khả năng mở rộng mạnh mẽ.

## Sơ đồ minh họa

### Volume & Mount



### Network (bridge)



# THỰC NGHIỆM

## Thực nghiệm docker trên dự án đặt lịch khám bệnh

Dự án: Hệ thống Đặt lịch khám bệnh trực tuyến (Frontend: Next.js, Backend: Express/Sequelize, Database: MySQL, Reverse Proxy: Nginx cho production).

Mục tiêu: Chuẩn hóa môi trường chạy, đơn giản hóa triển khai, tách biệt các tầng, thử nghiệm chạy đa môi trường (local và production), phân tích các vấn đề khi container hóa.

## Mục tiêu cụ thể

* Đóng gói backend và frontend thành image riêng.
* Kết nối qua mạng nội bộ Docker (bridge network).
* Quản lý biến môi trường cho local và production.
* Tích hợp reverse proxy + SSL termination (production).
* Ghi nhận và xử lý lỗi phát sinh (CORS, SSL, base URL, rebuild).

## Kiến trúc Docker

### Local (docker-compose.yml)

* frontend: Next.js (cổng 3000).
* backend: Express API (cổng 8080).
* db-mysql: MySQL 8.0 (port 3306 container, 3307 host).
* Network: app-network (bridge).
* Volume: db\_data lưu dữ liệu MySQL.

### Production (deploy/docker-compose.prod.yml)

Thêm nginx reverse proxy + SSL (Let’s Encrypt cert).

* frontend và backend dùng image build sẵn (registry).
* Expose nội bộ, chỉ Nginx publish cổng 80/443.

Luồng Request Production:

Client (HTTPS 443) → Nginx → (/api/\* → backend 8080 | / → frontend) → DB nội bộ.

### Dockerfile

Backend: node:20-alpine, copy code, cài deps, expose 8080, npm start.

Frontend: multi-stage build → build Next.js (stage 1), copy artifact sang runtime (stage 2), expose 3000.

### Biến môi trường

Local: NEXT\_PUBLIC\_API\_URL=http://localhost:8080/api/.

Production:

NEXT\_PUBLIC\_API\_URL=https://<domain>/api;

ALLOWED\_ORIGINS;

JWT\_SECRET;

DATABASE\_URL.

Rủi ro: lộ mật khẩu nếu commit .env. Giải pháp: Docker secrets / Vault.

### Quy trình chạy

Local:

docker compose build

docker compose up -d

curl http://localhost:8080/api/health

Production:

docker compose -f deploy/docker-compose.prod.yml pull

docker compose -f deploy/docker-compose.prod.yml up -d

curl -k https://<domain>/api/health

Kết quả & Đánh giá

* Khởi động local: ~20–40s.
* Restart policy tự phục hồi.
* Chuyển đổi local ↔ production rõ ràng (http ↔ https, qua Nginx).
* Docker đảm bảo môi trường nhất quán, image frontend nhỏ gọn nhờ multi-stage.
* Cần tối ưu thêm bảo mật (non-root, secrets) & cache deps.

Kết luận thực nghiệm:

Docker đơn giản hóa triển khai, đảm bảo tính nhất quán.