TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN
CÁC CÔNG NGHỆ MỚI TRONG PHÁT TRIỂN PHẦN MỀM

Giáo viên hướng dẫn: GS.TS. Phan Viết Hoàng
Sinh viên thực hiện: 2111858 – Nguyễn Thị Thùy Linh

Đà Lạt, 12/2024

MỤC LỤC

I. Development frameworks dùng trong containers

Trong môi trường sử dụng container (như Docker), các development frameworks cho frontend, backend, và database được triển khai độc lập và kết hợp với nhau để xây dựng các ứng dụng.

1. Frontend (Giao diện người dùng)

Frontend trong container được xây dựng để hiển thị giao diện người dùng. Các framework thường được sử dụng như:

- React.js: Một thư viện JavaScript để xây dựng giao diện người dùng.
- Vue.js: Một framework dễ học và phù hợp với các dự án vừa và nhỏ.
- Next.js: Một framework dựa trên React, hỗ trợ server-side rendering (SSR) và tối ưu hóa SEO.
- Angular: Một framework mạnh mẽ cho các ứng dụng quy mô lớn.

Triển khai Frontend trong container:

- Một container chứa mã frontend và các công cụ build (Webpack, Vite, hoặc Parcel).
- Sau khi build, mã tĩnh được phục vụ bởi nginx hoặc Node.js Express.
- Ví dụ: Một container chạy nginx để phân phối nội dung React đã build.

Công cụ trong Docker:

- Dockerfile để build ứng dụng frontend.
- Multi-stage build giúp tối ưu hóa kích thước container.

2. Backend (Xử lý logic và API)

Backend là phần xử lý logic, quản lý API và kết nối với cơ sở dữ liệu. Node.js rất phổ biến cho backend nhờ khả năng xử lý I/O hiệu quả. Các framework Node.js thường dùng:

- Express.js: Nhẹ, dễ cấu hình, phổ biến để xây dựng RESTful APIs.
- Nest.js: Framework mạnh mẽ, hỗ trợ cấu trúc dự án lớn, tích hợp tốt TypeScript.
- Fastify: Hiệu suất cao, phù hợp với các ứng dụng cần xử lý nhanh.
- Hapi.js: Framework linh hoạt, hỗ trợ xây dựng API lớn và bảo mật.

Triển khai Backend trong container:

- Một container chạy mã backend, thường được quản lý bằng Node.js.
- Cấu hình các biến môi trường (API keys, URL database) qua file .env.

Công cụ trong Docker:

- Docker Compose: Kết hợp backend với các dịch vụ liên quan như database, cache.
- PM2: Quản lý và giám sát tiến trình Node.js trong container.

3. Database (Lưu trữ dữ liệu) Database-Container: Lắng nghe trên cổng 27017 (MongoDB).

Node.js tương tác với cơ sở dữ liệu thông qua các thư viện hoặc ORM (Object Relational Mapping). Database không sử dụng Node.js trực tiếp nhưng đóng vai trò quan trọng trong hệ thống:

- Relational DBs (SQL): PostgreSQL, MySQL, MariaDB.
- ORM phổ biến: Sequelize, TypeORM.
- NoSQL DBs: MongoDB, DynamoDB.
- Thư viện phổ biến: Mongoose (cho MongoDB).

Triển khai Database trong container:

- Database thường chạy trong container riêng (ví dụ: postgres:latest hoặc mongo:latest).
- Dữ liệu được lưu trữ trên volume để không mất dữ liệu khi container tắt.

Công cụ trong Docker:

- Volume: Lưu trữ dữ liệu ngoài container.
- Docker Compose: Kết nối container database với backend.

II. Phân tích các lệnh hữu dụng nhất trong Docker (build, run, volume, v.v.). Giải thich tại sao?

1. docker build

Mô tả: Khởi chạy một container từ image đã build.

Cú pháp: **docker run [OPTIONS] <image_name>**

Lý do hữu ích:

- Cho phép bạn tạo image tùy chỉnh dựa trên cấu hình cụ thể của ứng dụng.
- Tagging (-t): Giúp bạn quản lý nhiều phiên bản của image (ví dụ: app:v1, app:latest).
- Tích hợp tốt với CI/CD để tự động hóa quá trình build.

2. docker run

Mô tả: Khởi chạy một container từ image đã build.

Cú pháp: **docker run [OPTIONS] <image_name>**

Lý do hữu ích:

- Cho phép bạn chạy và kiểm tra ứng dụng trong môi trường độc lập.
- Các tùy chọn linh hoạt như:
    (-p): Gắn cổng container với cổng host (-p 8080:80).
    (-d): Chạy container ở chế độ nền (detached).
    (--name): Đặt tên cho container để dễ quản lý.

3. docker ps

Mô tả: Hiển thị danh sách các container đang chạy.

Cú pháp: **docker ps [OPTIONS]**

Lý do hữu ích:

- Giúp bạn theo dõi trạng thái container, bao gồm cổng kết nối và ID.
- Tuỳ chọn (-a): Liệt kê cả container đã dừng.

4. docker logs

Mô tả: Xem log của container.

Cú pháp: **docker logs [OPTIONS] <container_name|ID>**

Lý do hữu ích:

- Hữu ích khi debug hoặc kiểm tra trạng thái của ứng dụng chạy trong container.
- Tùy chọn (-f): Theo dõi log theo thời gian thực.

5. docker volume

Mô tả: Quản lý dữ liệu persistent bằng volume.

Cú pháp: **docker volume <COMMAND>**

Lý do hữu ích:

- Giữ dữ liệu không bị mất khi container bị xóa.
- Tách biệt dữ liệu và logic ứng dụng.
- Dễ dàng chia sẻ dữ liệu giữa các container.

6. docker network

Mô tả: Tạo và quản lý mạng giữa các container.

Cú pháp: **docker network <COMMAND>**

Lý do hữu ích:

- Đảm bảo các container có thể giao tiếp an toàn và hiệu quả.
- Loại bỏ sự phụ thuộc vào IP tĩnh.

- Hỗ trợ mô hình microservices.

7. docker-compose

Mô tả: Sử dụng để quản lý nhiều container với file cấu hình docker-compose.yml.

Cú pháp: **docker-compose <COMMAND>**

Lý do hữu ích:

- Giảm thiểu cấu hình phức tạp khi làm việc với nhiều container.
- Tự động hóa việc build, chạy và kết nối container.

8. docker exec

Mô tả: Chạy lệnh trong container đang chạy.

Cú pháp: **docker exec [OPTIONS] <container_name|ID> <command>**

Lý do hữu ích:

- Cho phép truy cập container để kiểm tra hoặc sửa lỗi mà không cần khởi động lại.

9. docker rm và docker rmi

Mô tả:

- docker rm: Xóa container.
- docker rmi: Xóa image.

Lý do hữu ích:

- Giải phóng tài nguyên không cần thiết.
- Dọn dẹp container và image cũ.

10. docker prune

Mô tả: Xóa các tài nguyên Docker không sử dụng (container, image, volume, network).

Cú pháp: **docker system prune**

Lý do hữu ích:

- Giải phóng không gian lưu trữ.
- Tự động hóa việc dọn dẹp hệ thống.

Link GitHub: https://github.com/TLinhh210/DA1.git