CONTAINER HÓA ỨNG DỤNG REACT VỚI DOCKER

Mục tiêu: Cài đặt Docker vào môi trường Vagrant, làm quen với vòng đời của container, và áp dụng kiến thức để chạy ứng dụng React từ buổi 2 bên trong Docker.

Phần A: Cài đặt Docker vào Môi trường Vagrant

1. **Tạo script Cài đặt Docker:**
   Trong thư mục `scripts/` của dự án, tạo một file mới tên là `install-docker.sh`.

```
touch scripts/install-docker.sh
```

   Mở file và thêm nội dung sau. script này tuân theo hướng dẫn cài đặt chính thức của Docker cho Ubuntu.

```
#!/bin/bash

echo "Starting Docker installation..."

# Gỡ bỏ các phiên bản cũ nếu có
sudo apt-get remove docker docker-engine docker.io containerd runc

# Cập nhật và cài các gói cần thiết
sudo apt-get update
sudo apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Thêm GPG key chính thức của Docker
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
# Thiết lập repository của Docker
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Cài đặt Docker Engine
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Thêm user 'vagrant' vào group 'docker' để không cần dùng sudo khi chạy lệnh docker
sudo usermod -aG docker vagrant

echo "Docker installation finished."
echo "You may need to log out and log back in for group changes to take effect."
```

2. **Cập nhật `Vagrantfile`:**
   Mở `Vagrantfile` và thêm script mới này vào chuỗi provisioning.

```
# ... các dòng cấu hình khác ...
  config.vm.provision "shell", path: "scripts/bootstrap.sh"
  config.vm.provision "shell", path: "scripts/install-node.sh"
  config.vm.provision "shell", path: "scripts/install-docker.sh" # Thêm dòng này
# ...
```

3. **Tái tạo Môi trường:**
   Hủy và tạo lại máy ảo để script cài đặt được chạy.

```
vagrant destroy -f
vagrant up
```

4. Kiểm tra cài đặt Docker:
Sau khi máy ảo khởi động xong, SSH vào và kiểm tra.

```
vagrant ssh
```

Bên trong máy ảo, chạy lệnh sau. **không cần `sudo`** vì script đã thêm user `vagrant` vào group `docker`.

```
docker --version
```

Kết quả mong đợi: Hiển thị phiên bản Docker, ví dụ: `Docker version 20.10.21, build ...`

5. Commit các thay đổi:
Thoát máy ảo (`exit`) và commit.

```
git add Vagrantfile scripts/install-docker.sh
git commit -m "Feat: Provision environment with Docker Engine"
```

Phần B: Thực hành các Lệnh Docker Cơ bản

Tất cả các lệnh trong phần này được thực hiện bên trong máy ảo Vagrant.

1. **Chạy Container đầu tiên:**

```
docker run hello-world
```

Quan sát: Docker sẽ tự động `pull` image `hello-world` về và chạy nó. Container này chỉ in ra một thông điệp chào mừng rồi thoát.

2. **Kiểm tra Image và Container:**

```
docker images          # Bạn sẽ thấy image 'hello-world'
docker ps              # Sẽ không có gì, vì container đã chạy xong và dừng lại
docker ps -a           # Bạn sẽ thấy container 'hello-world' ở trạng thái 'Exited'
```

3. **Chạy Container ở chế độ Tương tác (Interactive):**

Chúng ta sẽ chạy một container Ubuntu và truy cập vào shell của nó.

```
docker run -it ubuntu:20.04 bash
```

Quan sát: Dấu nhắc lệnh của sẽ thay đổi thành `root@<container-id>:/#`. Bây giờ đang ở bên trong một container Ubuntu, không phải máy ảo Vagrant.

- Thử gõ `ls -la`, `pwd`, `apt update`. sẽ thấy đây là một môi trường Linux hoàn toàn mới và cô lập.
- Gõ `exit` để thoát khỏi container.

4. **Chạy một Web Server Nginx:**

```
docker run --name my-nginx-server -p 8000:80 -d nginx
```

- `--name my-nginx-server`: Đặt tên cho container là `my-nginx-server`.
- `-p 8000:80`: Ánh xạ cổng 8000 của máy ảo Vagrant tới cổng 80 (cổng web mặc định) của container Nginx.
- `-d`: Chạy ở chế độ nền.
- `nginx`: Tên của image cần chạy.

5. **Kiểm tra Nginx Server:**
   Mở trình duyệt trên máy thật của và truy cập http://localhost:8000.
   Kết quả mong đợi: sẽ thấy trang chào mừng "Welcome to nginx!".
6. **Quản lý Container Nginx:**

```
docker ps               # Xem container 'my-nginx-server' đang chạy
docker logs my-nginx-server # Xem log truy cập của Nginx
docker stop my-nginx-server # Dừng container
# Tải lại trang http://localhost:8000 -> sẽ không thể truy cập được
docker start my-nginx-server # Khởi động lại container
# Tải lại trang http://localhost:8000 -> hoạt động trở lại
```

7. **Cleanup:**

```
docker stop my-nginx-server
docker rm my-nginx-server
docker rm $(docker ps -aq) # Lệnh xóa tất cả các container đã dừng
docker rmi hello-world nginx ubuntu:20.04 # Xóa các image đã tải về
```

Phần C: Container hóa Môi trường Phát triển React

Mục tiêu là chạy server phát triển (`npm start`) của ứng dụng React bên trong một container `node`.

1. **Điều hướng đến thư mục dự án:**

Bên trong máy ảo, vào thư mục chứa code của .

```
cd /var/www/project/
```

2. **Phân tích lệnh `docker run`:**
   Chúng ta cần chạy một container từ image `node:16-alpine` và thực hiện các việc sau:
   a. Ánh xạ cổng `3000` ra ngoài.
   b. Mount thư mục `client/` trên máy ảo vào trong container để code được đồng bộ.
   c. Chỉ định thư mục làm việc bên trong container.
   d. Chạy `npm install` và `npm start`.
3. **Chạy Container Development:**

   Thực thi lệnh phức tạp sau:

```
docker run \
  --name react-dev-container \
  -p 3000:3000 \
  -v $(pwd)/client:/app \
  -w /app \
  -it \
  node:16-alpine \
  sh
```

a. -v $(pwd)/client:/app: Mount thư mục client hiện tại vào thư mục /app trong container.
b. -w /app: Đặt thư mục làm việc mặc định là /app.
c. -it: Chế độ tương tác.
d. node:16-alpine: Image cần dùng.
e. sh: Lệnh để khởi động shell Alpine.

4. **Bên trong Container:**
   Dấu nhắc lệnh của giờ là `/#`. đang ở trong container, tại thư mục `/app`.

- Chạy `ls` để xem code của đã được mount vào.
- Cài đặt các gói phụ thuộc:

```
npm install
```

Khởi động server phát triển:

```
# Chúng ta không cần sửa package.json nữa vì Docker sẽ xử lý port mapping
npm start
```

5. **Kiểm tra và Trải nghiệm Hot-Reloading:**
   a. Mở trình duyệt trên máy thật và truy cập http://localhost:3000. sẽ thấy ứng dụng React.
   b. Thử nghiệm quan trọng: Trên máy thật, mở file `src/client/src/components/Profile.js` và thay đổi `<h1>Your Name</h1>` thành `<h1>My Dockerized Name</h1>`.
   c. Lưu file lại. Quay lại trình duyệt và xem trang tự động cập nhật. Điều này chứng tỏ volume mount đang hoạt động hoàn hảo!

6. **Cleanup:**
    a. Trong terminal của container, nhấn `Ctrl+C` để dừng server.
    b. Gõ `exit` để thoát container.
    c. Xóa container: `docker rm react-dev-container`.

Phần D: Container hóa Môi trường Production

Bây giờ, chúng ta sẽ giả lập việc triển khai sản phẩm. Chúng ta sẽ build ứng dụng React và chạy server Express trong một container.

1. **Build ứng dụng React:**
   Bên trong máy ảo, tại thư mục `/var/www/project/client`, chạy:

```
npm run build
```

Việc này tạo ra thư mục `client/build`.

2. **Chạy Container Production:**

Quay lại thư mục `/var/www/project`. Chúng ta sẽ chạy `server.js` bằng container `node`.

```
docker run \
  --name react-prod-container \
  -p 5000:5000 \
  -v $(pwd):/app \
  -w /app \
  -d \
  node:16-alpine \
  node server.js
```

a. $(pwd): Mount toàn bộ thư mục dự án (my-lamp-project) vào /app.
b. -d: Chạy nền.
c. node server.js: Lệnh để chạy server Express của chúng ta.

3. Kiểm tra:

- Chạy `docker ps` để thấy `react-prod-container` đang chạy.

- Chạy `docker logs react-prod-container` để thấy output "Production server is running...".
- Mở trình duyệt trên máy thật và truy cập http://localhost:5000. sẽ thấy ứng dụng React của mình.

Phần E: Cleanup

1. **Dừng và xóa container:**

```
docker stop react-prod-container
docker rm react-prod-container
```

2. **Cleanup hệ thống Docker:**

Lệnh này sẽ xóa tất cả các container đã dừng, các network không dùng, và các image "dangling" (image không được tag).

```
docker system prune -f
```