TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO

MÔN: PROJECT 1

Xây dựng một quy trình DevOps

Sinh viên thực hiện: Phan Minh Anh Tuấn

Mã số sinh viên: 20205227

Lớp: Việt Pháp 01 - K65

Giáo viên hướng dẫn: Nguyễn Mạnh Tuấn _____

Chữ kí GVHD

HÀ NỘI, 03/2023

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Lời nói đầu

Hiện nay, chu trình phát triển phần mềm chia làm hai giai đoạn chính, đó là phát triển (Deployments) và vận hành (Operations). Giai đoạn phát triển bao gồm các công việc về thiết kế, coding... Giai đoạn vận hành bao gồm các công việc về hạ tầng hệ thống, bảo mật, triển khai lên máy chủ... Hai giai đoạn này tách rời nhau, khiến cho quá trình làm việc trở nên khó khăn hơn. Đó là lý do DevOps ra đời.

DevOps được ghép từ Developement (Dev) và Operations (Ops), nghĩa là sự kết hợp của bên phía phát triển sản phẩm và bên vận hành hệ thống giúp rút ngắn quá trình phát triển sản phẩm, ra mắt các phiên bản phần mềm mới nhanh hơn.

Từ các lý do trên, có thể thấy tầm quan trọng của DevOps trong ngành Công Nghệ Thông Tin nói chung và ngành Công Nghệ Phần Mềm nói riêng. Em chọn chủ đề này để nghiên cứu trong phạm vi môn project I.

1.2 Chi tiết công việc thực hiện

Nội dụng công việc:

- Tìm hiểu về quy trình DevOps.
- Tìm hiểu và lựa chọn các công cụ phù hợp cho các giai đoạn của quy trình DevOps để tạo tools chain.
- Xây dựng tutorial thiết lập quy trình DevOps cho phát triển một ứng dụng Web đơn giản nhằm tự động hóa các giai đoạn: code -> build -> test -> deploy.
- Tìm hiểu về các vấn đề an toàn trong quy trình DevOps.
- Xây dựng website áp dụng các quy trình DevOps.

Công cụ phát triển:

- Các file config: GitLab, Dockerfile, YAML.
- Môi trường phát triển: Ubuntu server.

Lịch trình thực hiện:

- Tuần 1: Tìm hiểu về CI/CD.
- Tuần 2-3: Xây dựng quy trình CI/CD trên GitLab.
- Tuần 4-5: Tìm hiểu về môi trường triển khai: docker, container, severless...

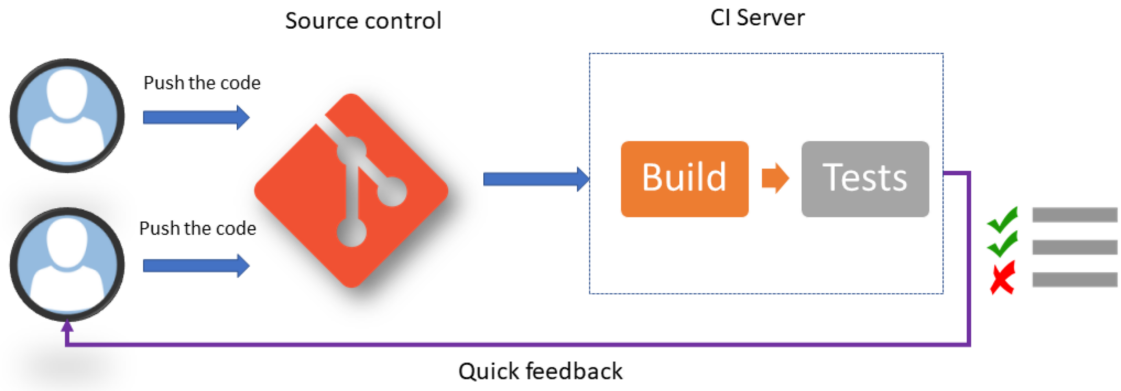
CHƯƠNG 2. QUÁ TRÌNH CI/CD

Nội dung chính của chương này sẽ tìm hiểu về quá trình CI/CD, một yếu tố quan trọng trong quy trình DevOps. Có 3 thành phần chính bao gồm continuous integration, continuous delivery và continuous deployment.

2.1 Continuous integration (CI)

Đây là một kĩ thuật phát triển phần mềm, trong đó các thành viên trong một team thường xuyên tích hợp code của họ với nhau thông qua GitHub, GitLab... Mỗi lần tích hợp đều được build và test tự động để phát hiện và sửa lỗi một cách nhanh chóng. Vậy continuous integration là một quá trình tự động cho phép kiểm tra độ ổn định và hoàn thiện của code mỗi khi thành viên của nhóm thực hiện thêm/sửa đổi.

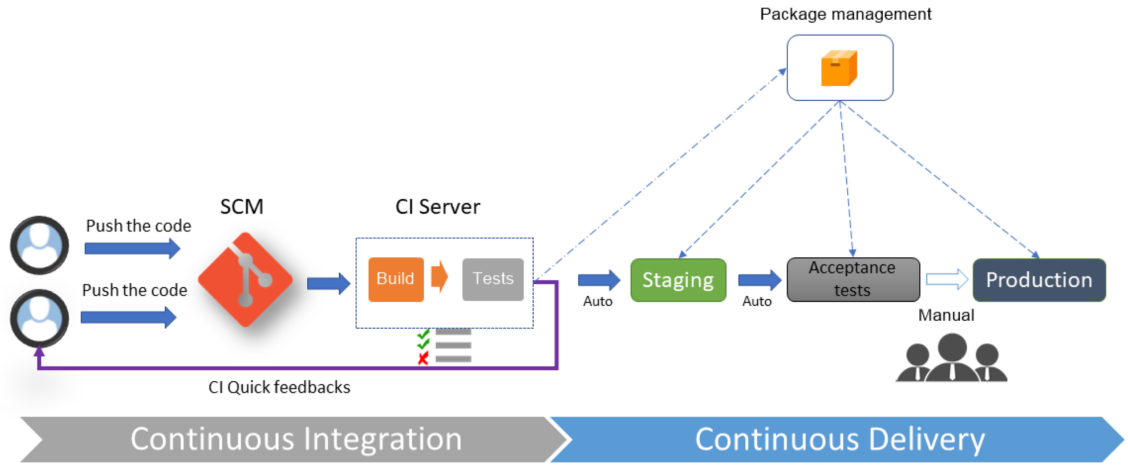
Để triển khai CI ta cần có một Source Code Manager (SCM) cho phép đồng bộ code của toàn bộ thành viên trong nhóm. Phổ biến nhất là Git. Một số Automatic Build Manager (CI server) hỗ trợ continuous integration như GitLab CI, Azure Pipelines, GitHub Actions ...



Hình 2.1: Continuous intergration pipeline

2.2 Continuous delivery (CD)

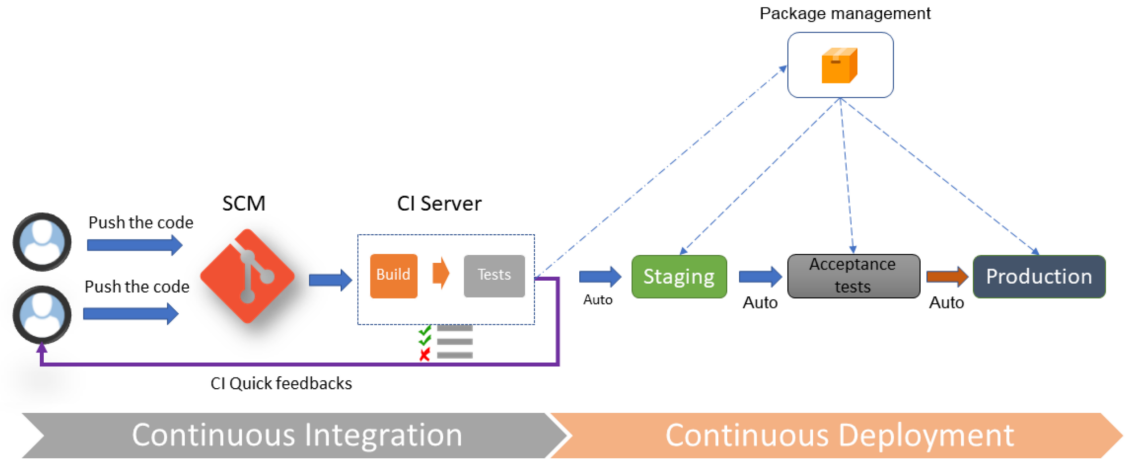
Mỗi một lần CI hoàn thành, bước tiếp theo là deploy sản phẩm **tự động** tại các môi trường testing hoặc staging và deploy **thủ công** lên môi trường production. Những bài test này bao gồm UI testing, load testing, integration testing, API testing... Tự động hoàn toàn quy trình release phần mềm. Đó được gọi là Continuous delivery (CD). Mục tiêu của CD là test toàn bộ sản phẩm tạo ra, khác với CI chỉ test từng thành phần.



Hình 2.2: Continuous delivery pipeline

2.3 Continuous deployment

Điểm khác duy nhất giữa Continuous deployment và Continuous delivery là Continuous deployment tự động hóa việc deploy lên môi trường production



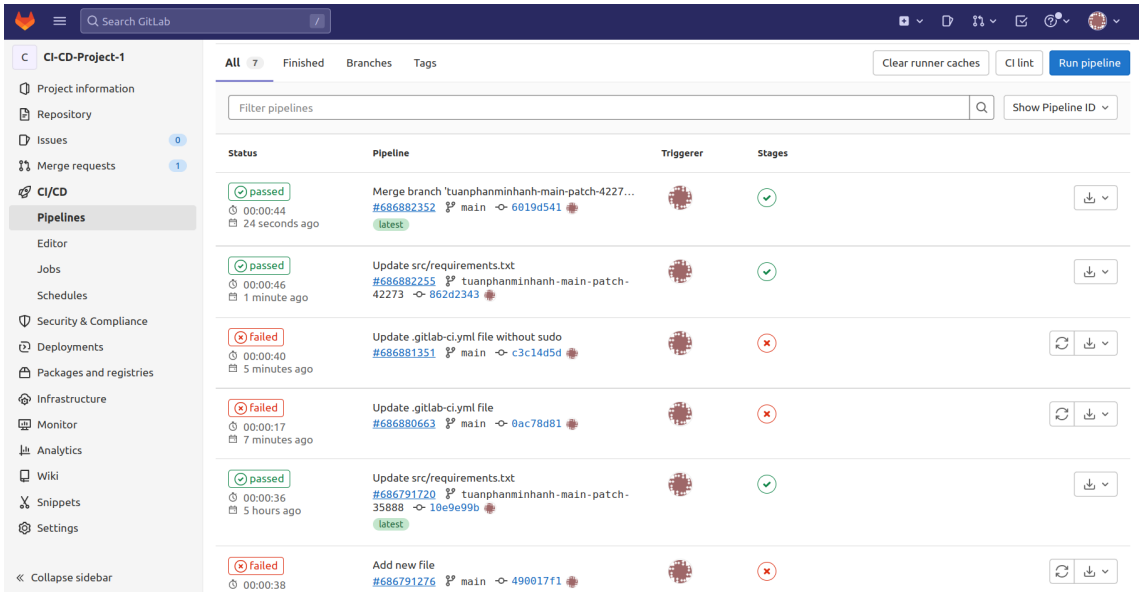
Hình 2.3: Continuous deployment pipeline

2.4 Continuous delivery hay Continuous deployment ?

Tùy thuộc vào mục tiêu và yêu cầu để chọn practice phù hợp. Trong đó continuous delivery phù hợp với các công ty muốn kiểm soát sản phẩm cuối đến với người dùng, còn continuous deployment sẽ nhanh hơn do tự động hóa cả quy trình cuối.

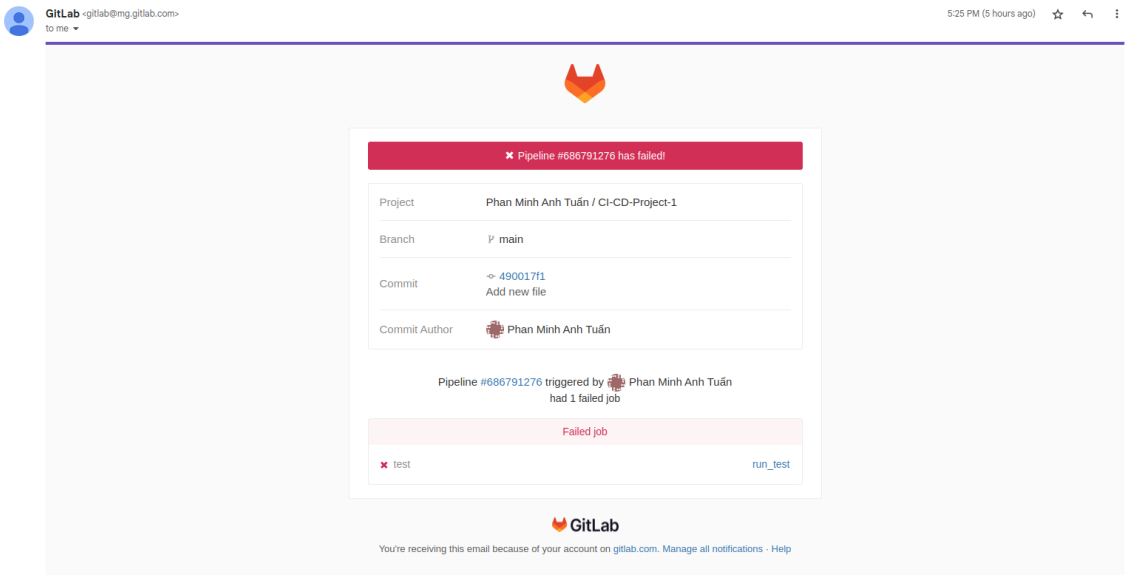
2.5 Xây dựng quy trình CI/CD trên GitLab

GitLab là một kho lưu trữ và quản lý code mã nguồn mở. GitLab hỗ trợ việc tạo các pipeline CI/CD một cách dễ dàng và miễn phí.



Hình 2.4: Các pipelines quá trình CI/CD

GitLab hỗ trợ thông báo lỗi về email của người dùng khi có pipeline xảy ra lỗi.



Hình 2.5: Mail gửi về khi quá trình gặp lỗi

Do đó, em lựa chọn GitLab làm công cụ để minh họa quá trình CI/CD đơn giản. Để minh họa, em sử dụng một repository mã nguồn mở: **python-demoapp**.

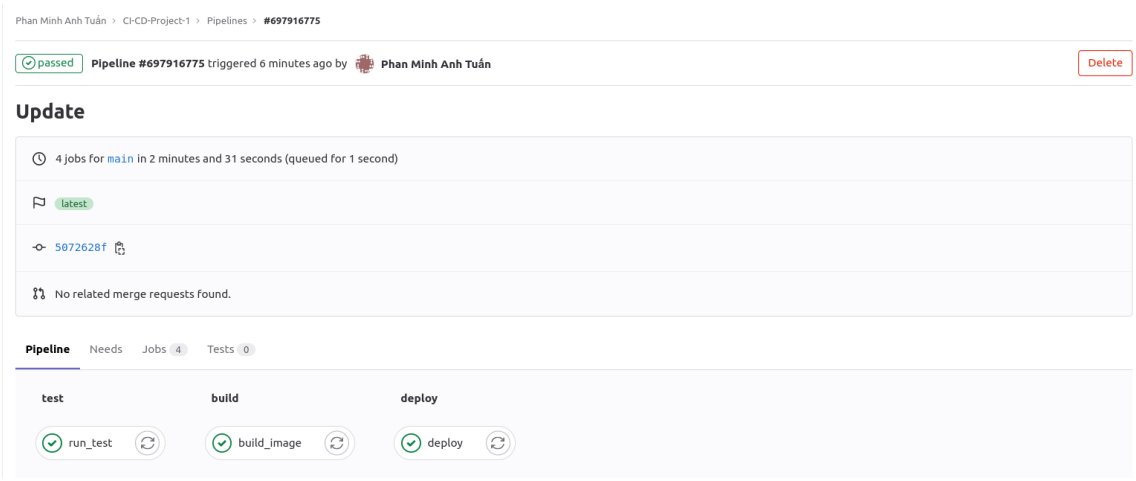
Chi tiết các bước thực hiện như sau:

- Bước 1: Chạy các file test đã chuẩn bị sẵn bao gồm UI testing, load testing... trên môi trường của GitLab.
- Bước 2: Tạo docker images từ Dockerfile trong repository, sau đó push lên Dockerhub.
- Bước 3: Truy cập vào server đã chuẩn bị sẵn, kéo docker images từ docker hub về và sau đó chạy docker container.

```
stages:
    - test
    - build
    - deploy


run_test:
    stage: test
    image: python:3.9-slim-buster
    before_script:
        - apt-get update && apt-get install make
    script:
        - make test

build_image:
    stage: build
    image: docker:20.10.16
    services:
        - docker:20.10.16-dind
    variables:
        DOCKER_TLS_CERTDIR: "/certs"
    before_script:
        - docker login -u $USER -p $PASS
    script:
        - docker build -t pmatuan/demo-app-python:v1 .
        - docker push pmatuan/demo-app-python:v1


deploy:
    stage: deploy
    before_script:
        - chmod 400 $SSH_KEY
    script:
        - ssh -o StrictHostKeyChecking=no -i $SSH_KEY pmat@20.187.72.154 "
            docker login -u $USER -p $PASS &&
            docker run -d -p 5000:5000 pmatuan/demo-app-python:v1"
```

Hình 2.6: File .gitlab-ci.yml minh họa 3 bước trên

Sau khi commit code, quá trình CI/CD được tự động thực hiện. Giao diện khi thực hiện 3 bước trên thành công như sau.



Hình 2.7: Chạy thành công run_test, build_image, deploy

Truy cập vào máy chủ, kiểm tra các docker container có đang được chạy hay không.

```
pmat@ServerDeploy:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
986570b447f9   pmatuan/demo-app-python:v1   "gunicorn -b 0.0.0.0…"   6 minutes ago   Up 6 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   practical_wescoff
```

Hình 2.8: Container đang được chạy trên máy chủ

Kết luận: Từ kết quả trên, có thể thấy em đã xây dựng thành công một quy trình CI/CD đơn giản trên GitLab.

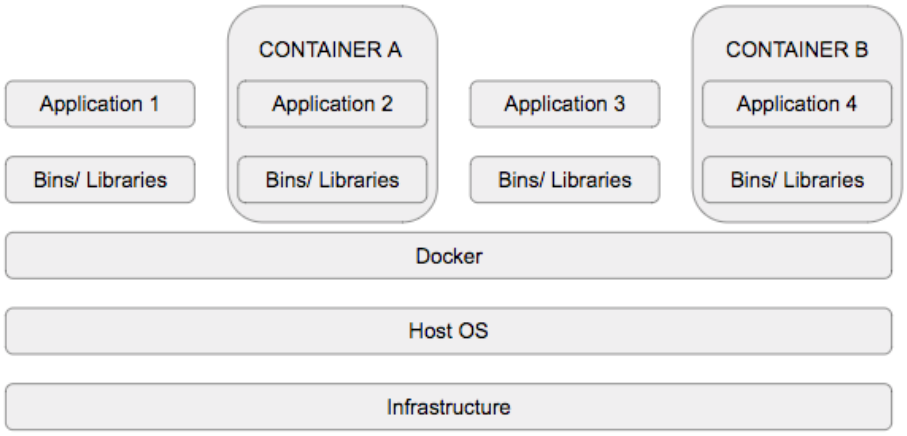
CHƯƠNG 3. MÔI TRƯỜNG TRIỂN KHAI

Chương 3 bao gồm 2 phần chính. Phần đầu em sẽ trình bày cơ bản về Docker, cách để tạo một Docker images và chạy container từ Dockerfile. Phần sau em sẽ trình bày về kiến trúc serverless.

3.1 Container hóa ứng dụng với Docker

3.1.1 Container là gì

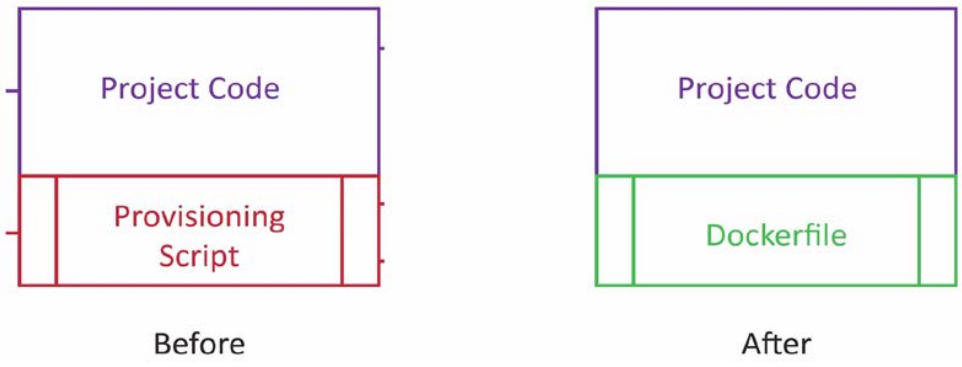
Container là hệ thống đóng gói một ứng dụng với các thành phần cần thiết như thư viện, các chương trình dịch... một cách độc lập.



Hình 3.1: Kiến trúc container

3.1.2 Dockerfile, Docker image, Docker container

Thông thường, để khởi chạy một ứng dụng ta cần project code và provisioning script để chạy chương trình trên máy chủ.



Hình 3.2: Setup mẫu cách Docker thực hiện

Việc sử dụng Docker không can thiệp vào project code, mà thay thế phần provisioning script bằng Dockerfile. Sự kết hợp giữa project code và Dockerfile tạo nên Docker image. Khi chạy Docker image sẽ tạo ra Docker container.

3.1.3 Cấu trúc Dockerfile

Theo tài liệu chính thức trên trang chủ Docker, cấu trúc Dockerfile sẽ gồm những câu lệnh cơ bản sau:

- **FROM:** Chỉ định base image.
- **RUN:** Các câu lệnh được thực thi trong quá trình build image.
- **WORKDIR:** Thư mục làm việc mặc định.
- **ADD/COPY:** Sao chép thư mục vào container.
- **CMD:** Tham số trong lúc thực thi câu lệnh chạy container.
- **EXPOSE:** Thiết lập cổng để truy cập tới container sau khi chạy.
- **ENTRYPOINT:**: Tiến trình mặc định khi chạy container.

3.1.4 Ví dụ minh họa

Để minh họa quá trình tạo Docker container, Docker image từ Dockerfile. Em sẽ tạo một trang web React trên Docker.

```
FROM ubuntu:latest

RUN apt update -y
RUN apt install curl -y
RUN apt install neovim -y
RUN curl -sL https://deb.nodesource.com/setup_18.x -o nodesource_setup.sh
RUN bash nodesource_setup.sh
RUN apt install nodejs -y
RUN npm install -g create-react-app

WORKDIR /home
ADD app /home/
WORKDIR /home/app
CMD ["npm", "start"]
```

Hình 3.3: Cấu trúc Dockerfile

Chi tiết các bước thực hiện trong Dockerfile như sau:

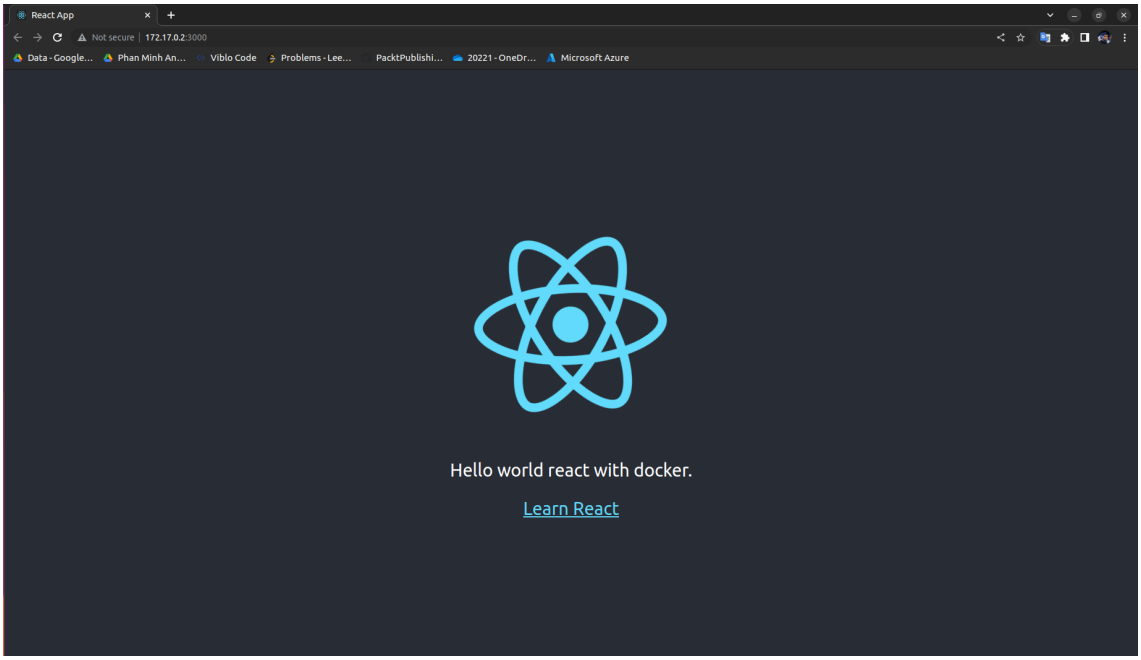
- Bước 1: Sử dụng ubuntu làm base image. Tức là container sẽ chạy dựa trên ubuntu.
- Bước 2: Cài đặt các ứng dụng cần thiết như curl, vim, nodejs, npm và create-react-app.

- Bước 3: Thiết lập thư mục làm việc là /home/.
- Bước 4: Copy thư mục app từ local.
- Bước 4: Thiết lập lại thư mục làm việc là /home/.
- Bước 5: Chạy npm start.

Sau đó chạy câu lệnh *docker build -t demo:v1 -f Dockerfile .* để tạo Docker image từ Dockerfile. Trong đó demo là tên image được tạo, v1 là tag của image.

Tạo container từ image, chạy câu lệnh *docker run demo:v1*. Terminal hiển thị như sau.

Truy cập vào 172.17.0.2:3000, ta được kết quả như sau.



Hình 3.4: Chạy thành công web React trên Docker

Kết luận: Từ kết quả trên, có thể thấy em đã minh họa thành công quá trình tạo Docker container, Docker image từ Dockerfile.

3.2 Serverless

3.2.1 Kiến trúc Serverless

Serverless là một mô hình thực thi điện toán đám mây mà trong đó các cloud providers sẽ quản lý động việc phân bổ tài nguyên, cung cấp cấu hình cần thiết cho các container để tránh sự lãng phí.

Ban đầu, serverless được coi như là Backend as a Service (BaaS), do nó phụ thuộc hoàn toàn vào một phần hay toàn bộ các dịch vụ bên thứ ba. Ngày nay, serverless lại được coi như là Function as a Service do serverless providers coi ứng dụng như một hàm, gọi chúng khi có yêu cầu.

3.2.2 Serverless Computing hoạt động như thế nào?

Serverless providers cung cấp các vùng cho các containers của ứng dụng. Các container hoạt động theo hướng sự kiện (event-driven), tức là chỉ kích hoạt khi có những sự kiện nhất định.

Các sự kiện có thể là:

- HTTP requests
- Upload file
- Cập nhật Database
- ...