ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

NGUYỄN KHÁNH LINH - 22520769

TRẦN THIÊN THANH - 22521367

PHẠM THỊ CẨM TIÊN - 22521473

THÁI NGỌC DIỄM TRINH - 22521541

BÁO CÁO ĐỒ ÁN

MÔN QUẢN TRỊ MẠNG VÀ HỆ THỐNG

ĐỀ TÀI: TRIỂN KHAI ỨNG DỤNG WEB VỚI DOCKER CONTAINER

GIẢNG VIÊN HƯỚNG DẪN

THS. TRẦN THỊ DUNG

TP. HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin trân trọng cảm ơn cô ThS. Trần Thị Dung - giảng viên hướng dẫn đồ án môn học Quản trị mạng và hệ thống, người đã trực tiếp chỉ bảo và hướng dẫn tận tình cho chúng em trong suốt quá trình thực hiện và hoàn thành đề tài này.

Với tất cả sự cố gắng, nhóm chúng em đã hoàn thành đồ án theo đúng kế hoạch đã đề ra và hoàn thiện hết sức có thể. Tuy nhiên, nhóm chúng em vẫn còn non trẻ và chưa có đủ kinh nghiệm nên vẫn sẽ có những thiếu sót xảy ra. Chúng em kính mong nhận được những lời góp ý và chỉ bảo của cô để đề tài này ngày càng hoàn thiện hơn. Và cũng xin cảm ơn các thành viên trong nhóm đã nỗ lực đóng góp, thực hiện công việc của mình một cách tốt nhất để hoàn thành đề tài này.

Chúng em xin chân thành cảm ơn!

Thành phố Hồ Chí Minh, tháng 11 năm 2024

Nhóm sinh viên thực hiện

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

I. Tổng quan 8
1.1. Giới thiệu 8
1.2. Thành phần 8
1.3. Hoạt động 9
1.4. Viết Dockerfile 10
1.5. Viết docker-compose.yml 12
II. Triển khai 14
2.1. Mô hình 14
2.1.1. Static web 14
2.1.2. Web Application 15
2.2. Cài đặt 16
2.2.1. Cài đặt WSL2 16
2.2.2. Cài đặt Docker Desktop 16
2.3. Cấu hình 18
2.3.1. Cấu hình cơ bản 18
2.3.2. Cấu hình nâng cao 22
III. Kết quả và kết luận 35
3.1. Kết quả 35
3.2. Kết luận 35

DANH MỤC HÌNH

Hình 1.1 Hoạt động của Docker 9
Hình 1.2. Cấu trúc của file docker-compose.yml 12
Hình 2.1. Mô hình của website tĩnh 14
Hình 2.2. Mô hình của ứng dụng web 15
Hình 2.3. Giao diện cài đặt Docker Desktop (1) 16
Hình 2.4. Giao diện cài đặt Docker Desktop (2) 17
Hình 2.5. Giao diện thỏa thuận dịch vụ đăng ký của Docker Desktop 17
Hình 2.6. Dockerfile cho triển khai một website tĩnh 18
Hình 2.7. Tạo Docker image 18
Hình 2.8. Danh sách các image đang tồn tại trên máy tính 19
Hình 2.9. Chạy Docker container 19
Hình 2.10. Giao diện website được triển khai 20
Hình 2.11. Tạo repository trên Docker Hub 20
Hình 2.12. Đăng nhập vào Docker Hub 20
Hình 2.13. Gắn tag cho image và kiểm tra lại 21
Hình 2.14. Đẩy image lên Docker Hub 21
Hình 2.15. Danh sách image đã push lên Docker Hub 21
Hình 2.16. Kéo image về máy 22
Hình 2.17. Liệt kê danh sách image 22
Hình 2.18. Chạy container 22
Hình 2.19. Cấu trúc thư mục của ứng dụng web 23
Hình 2.20. Dockerfile của frontend 24
Hình 2.21. Dockerfile của backend 25
Hình 2.22. Nội dung file docker-compose.yml 26
Hình 2.23. Xây dựng các image và khởi chạy các container 27
Hình 2.24. Danh sách các container tồn tại trên máy 27
Hình 2.25. Giao diện của ứng dụng web 28
Hình 2.26. Dừng hoặc gỡ các container 28
Hình 2.27. Tạo repository trên Docker Hub 28
Hình 2.28. Đăng nhập Docker 29
Hình 2.29. Gắn tag cho các image và liệt kê danh sách các image 29

Hình 2.30. Đẩy image lên Docker Hub 29
Hình 2.31. Danh sách các image trên Docker Hub 30
Hình 2.32. Kéo các image về máy.....30
Hình 2.33. Danh sách các image đang tồn tại trên máy 30
Hình 2.34. Khởi chạy lần lượt các container.....31
Hình 2.35. Giao diện ứng dụng web.....31
Hình 2.36. Giao diện trang đăng ký 32
Hình 2.37. Giao diện trang đăng nhập.....32
Hình 2.38. Giao diện sau khi đăng nhập thành công.....33
Hình 2.39. Database MongoDB bảng users 33
Hình 2.40. Database MongoDB bảng todos.....34

DANH MỤC BẢNG

Bảng 1.1. So sánh triển khai bằng Docker và Manual 12
Bảng 2. Bảng phân công công việc 39
Bảng 3. Bảng điểm tự đánh giá 40

I. Tổng quan

1.1. Giới thiệu

Docker là một nền tảng mở để người dùng phát triển, vận chuyển và chạy các ứng dụng một cách dễ dàng trên nền tảng ảo hóa độc lập. Nó cho phép người dùng tách biệt các ứng dụng khỏi cơ sở hạ tầng để có thể phân phối phần mềm nhanh chóng thông qua các container. Bằng cách tận dụng các phương pháp của Docker để vận chuyển, thử nghiệm, triển khai, người dùng có thể giảm đáng kể độ trễ giữa quá trình viết mã và triển khai của ứng dụng.

Docker được xem là hệ điều hành dành riêng cho container. Tương tự như các máy ảo ảo hóa phần cứng máy chủ, các container sẽ ảo hóa hệ điều hành của máy chủ. Sau khi Docker được cài đặt trên mỗi máy chủ, nó cung cấp các lệnh cơ bản có thể được sử dụng để tạo, khởi chạy và dừng các container. Đặc biệt, các container có tính di động, người dùng có thể di chuyển ứng dụng của mình từ môi trường này sang môi trường khác mà không lo có sự khác biệt về hệ điều hành cũng như cấu hình của phần cứng.

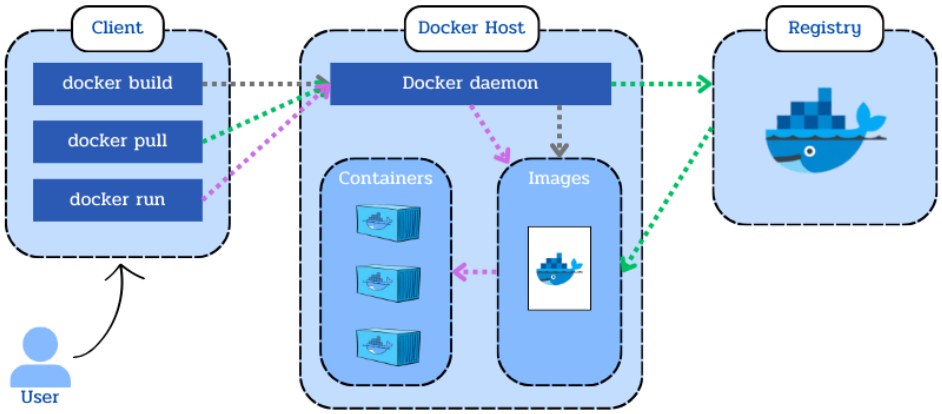
1.2. Thành phần

Để có cái nhìn tổng quát hơn về quá trình triển khai ứng dụng web bằng Docker, chúng ta cần tìm hiểu chi tiết hơn về các thành phần của Docker:

- Docker Client: là giao diện dòng lệnh để người dùng tương tác với Docker. Bằng cách sử dụng Docker Client, người dùng có thể gửi các yêu cầu tới Docker Daemon để thực hiện các hoạt động như tạo, chạy, quản lý, … các container.
- Docker Engine: là thành phần cốt lõi của Docker. Đây được xem như là một nền tảng phần mềm dùng cho mục đích khởi tạo các container, quản lý các container trên một máy tính hoặc một máy chủ. Docker Engine bao gồm:
    - Docker Daemon: dùng để tạo và quản lý các đối tượng như image, container, network, volume.
    - Docker API: là một controller cho Docker Daemon, giúp cho những công việc cụ thể từ Docker Client mà Docker Daemon phải làm.
    - Docker Client: Docker Client sử dụng Docker API để gửi lệnh tới Docker Daemon.

- Docker Registry: là dịch vụ cloud có khả năng tự động hóa các công việc liên tục và cho phép chia sẻ các ứng dụng giữa các máy với nhau. Nó cho phép người dùng thực hiện các thao tác như push, pull với các image.
- Dockerfile: là tệp văn bản chứa một tập các hướng dẫn về các bước thực thi phần mềm và dùng làm cấu trúc để xây dựng Docker Image.
- Docker Image: là một file bất biến - không thay đổi, chứa các source code, libraries, dependencies, tools và các file khác cần thiết cho một ứng dụng để chạy. Docker Image có thể chia sẻ, triển khai trên nhiều máy khác nhau một cách độc lập.
- Docker Container: là một run-time environment nhỏ gọn chứa tất cả các thành phần cần thiết của một ứng dụng như mã nguồn, thư viện, và các công cụ, giúp đảm bảo ứng dụng có thể chạy độc lập và đồng nhất trên mọi môi trường. Đặc biệt, các container sử dụng chung tài nguyên của hệ thống nên nó rất nhẹ và giúp cái thao tác kết nối, tương tác diễn ra nhanh chóng và tiện lợi hơn.
- Docker Compose: là công cụ hỗ trợ xác định và chạy các ứng dụng multi-container. Nó có thể xử lý đồng thời multi-container trong sản xuất, staging, phát triển, thử nghiệm và CI.

1.3. Hoạt động



Hình 1.1 Hoạt động của Docker

Về cơ bản, Docker Client và Docker Daemon có thể chạy trên cùng một hệ thống hoặc Docker Client có thể kết nối với Docker Daemon từ xa.

Docker Client gửi các yêu cầu đến Docker Daemon để thực hiện các hành động như xây dựng container, chạy container, tải xuống image, ... và client có thể giao tiếp với nhiều hơn một daemon.

Docker Daemon quản lý các đối tượng docker như image, container, mạng, … và là nơi lắng nghe, thực thi các yêu cầu API Docker như "docker build", "docker run",… kết nối với Docker registry để thực hiện các hành động như tải hoặc đẩy các Docker image.

Sau khi image được tạo thành công người dùng có thể đẩy nó lên các repository trên docker bằng lệnh "docker push" để lưu trữ ở Registry. Người dùng có thể lấy và sử dụng các Docker image từ Registry bằng cách chạy lệnh "docker pull", "docker run".

1.4. Viết Dockerfile

Trong Dockerfile, đầu tiên người dùng cần xác định image cơ sở mà Docker sử dụng để xây dựng container bằng cách thêm nội dung sau: *FROM <image>*

- Ví dụ : FROM ubuntu:20.04 ⇒ Người dùng sử dụng image cơ sở là Ubuntu phiên bản 20.04.

Sau đó, thiết lập thư mục làm việc cho container bằng lệnh: *WORKDIR /path/to/directory*. Lệnh này sẽ chỉ định nơi các lệnh tiếp theo sẽ chạy.

- Ví dụ : WORKDIR /app ⇒ Các lệnh tiếp theo sẽ được thực thi trong thư mục app.

Sao chép tất cả các tập tin và thư mục từ dự án trên máy người dùng vào container image bằng cách sử dụng *COPY <source> <destination>*

- Ví dụ : COPY . . ⇒ Sao chép tất cả các tệp và thư mục từ thư mục hiện tại trên máy (thư mục Dockerfile đang tồn tại) vào thư mục làm việc hiện tại trên container.

Kế tiếp, sử dụng *RUN <command>* để thực thi các lệnh bên trong container trong quá trình xây dựng image với mục đích cài đặt phần mềm, cấu hình môi trường hoặc thực hiện bất kỳ bước thiết lập nào cần thiết trước khi chạy ứng dụng.

- Ví dụ : RUN npm install -g yarn ⇨ Cài đặt yarn toàn cục trong container.

Sau đó, người dùng có thể thông báo với Docker về các cổng mạng cụ thể mà container sẽ lắng nghe trong thời gian chạy bằng *EXPOSE <port>* (hoặc *EXPOSE*

<*port*>/<*protocol*>). Tuy nhiên khi kết hợp với cờ -p khi chạy lệnh “docker run”, các cổng đang mở có thể được ánh xạ tới các cổng trên máy chủ.

- Ví dụ : EXPOSE 80 ⇒ Thông báo ứng dụng bên trong container sẽ lắng nghe trên cổng 80.

Và cuối cùng, sử dụng *CMD ["executable", "param1", "param2"]* để chỉ thị lệnh mặc định sẽ được thực thi khi container khởi động.

- Ví dụ : CMD ["node", "server.js"] => Yêu cầu Docker chạy lệnh node server.js khi container bắt đầu chạy.⇨

So sánh giữa triển khai bằng Docker với triển khai thủ công:

Docker deployment	**Manual deployment**
1. Tạo Dockerfile.	1. Cài đặt môi trường và phần mềm cần thiết để chạy web như ngôn ngữ lập trình, web server, cơ sở dữ liệu, các công cụ hỗ trợ (npm, python3,...) trên máy Developer.
2. Xây dựng Docker image từ Dockerfile	2. Chuyển mã nguồn và các cài đặt môi trường bằng các công cụ như npm install, pip install,..
3. Chạy Docker container từ Docker image ở máy Developer.	3. Chạy web thủ công thông qua các công cụ hoặc các lệnh trực tiếp ở máy Developer.
4. Đẩy Docker image lên Docker registry để lưu trữ.	4. Mã nguồn và các cài đặt phụ thuộc thường được sao chép trực tiếp từ Developer lên host mà không lưu trữ.
5. Kéo Docker image từ Docker registry về máy Production.	5. Tải mã nguồn, cài đặt môi trường, phần mềm và các công cụ cần thiết để chạy web trên máy Production.

6. Chạy Docker container từ Docker image ở máy Production.	6. Chạy web thủ công thông qua các công cụ hoặc các lệnh trực tiếp ở máy Production.
7. Cập nhật web Docker bằng cách xây dựng lại Docker image và đẩy nó lên lại registry. Máy Production chỉ cần lấy image và chạy container mới.	7. Cập nhật mã nguồn trên máy Developer, có thể cần khởi động lại các dịch vụ (web server, database,...). Máy Production cần lấy mã nguồn mới từ máy Developer và phải chạy lại các lệnh cài đặt cần thiết để chạy web.

Bảng 1.1. So sánh triển khai bằng Docker và Manual

1.5. Viết docker-compose.yml

Cấu trúc cơ bản của file docker-compose.yml sẽ trông như dưới:

```
services:
  <tên_service>:
    container_name: <tên_container>
    restart: always
    build:
      context: <tên_thư_mục>
    ports:
      - "<host_port>:<container_port>"
    volumes:
      - <đường_dẫn_host>:<đường_dẫn_container>
    depends_on:
      - <tên_service_khác>
```

Hình 1.2. Cấu trúc của file docker-compose.yml

Cụ thể:

- services: <tên_service>: Tên của service (dịch vụ).
- container_name: Tên của container chạy dịch vụ này.
- restart: Cài đặt chính sách khởi động lại (ở đây là always - luôn khởi động lại nếu container dừng).
- build: context: Thư mục chứa Dockerfile và các tệp liên quan để build image.

- ports: Danh sách ánh xạ cổng giữa host và container (theo định dạng <host_port>:<container_port>).
- volumes: Liên kết thư mục/volume giữa host và container (theo định dạng <đường_dẫn_host>:<đường_dẫn_container>).
- depends_on: Xác định các dịch vụ khác mà dịch vụ này phụ thuộc

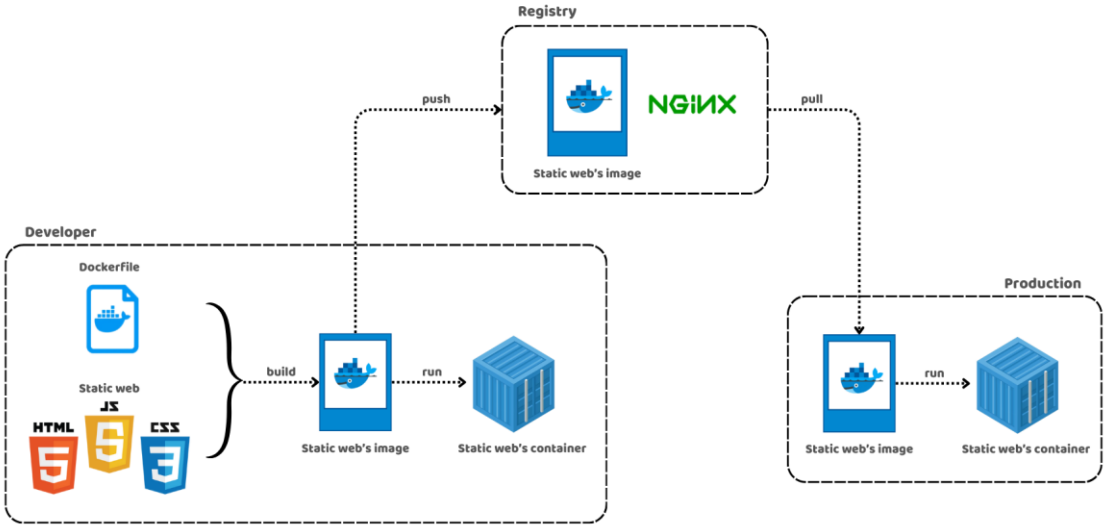
II. Triển khai

2.1. Mô hình

Mục tiêu:

- Cơ bản: Triển khai thành công một trang web tĩnh - chỉ có front-end.
- Nâng cao: Triển khai thành công một trang web có đầy đủ front-end, back-end và database.

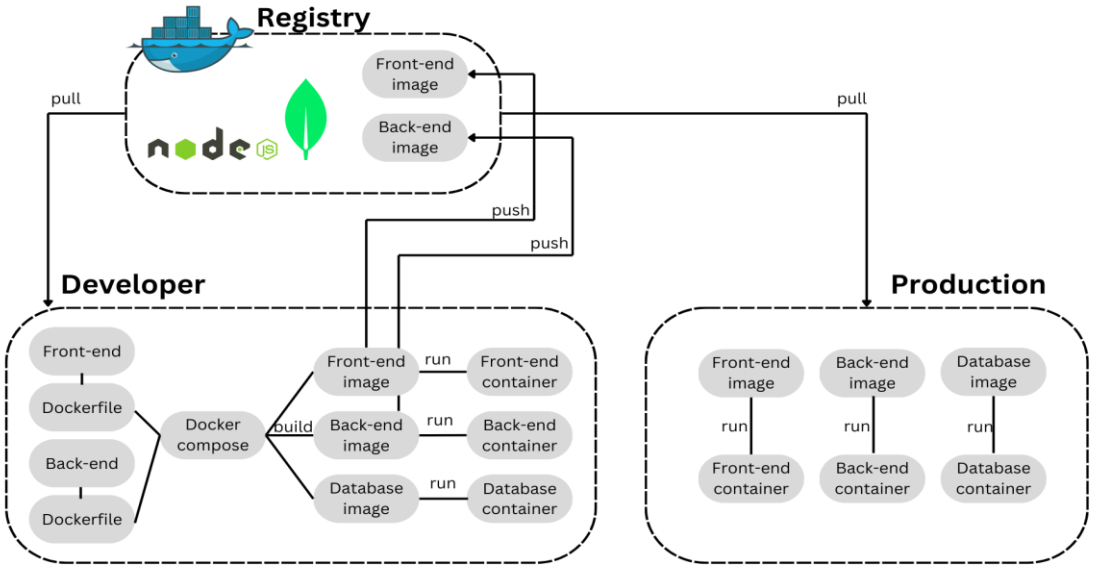
2.1.1. Static web



Hình 2.1. Mô hình của website tĩnh

- Registry: là nơi lưu trữ và phân phối Docker image giúp người dùng dễ quản lý các image của mình và người dùng khác có thể dễ dàng tìm kiếm và sử dụng các image được chia sẻ. Docker Hub là registry mặc định mà Docker sử dụng, người dùng cũng có thể sử dụng các private registry.
- Developer: Ở đây Developer tạo Dockerfile để chỉ định cách xây dựng Docker image từ mã nguồn static web do Developer phát triển. Sau đó, tiến hành xây dựng một image từ Dockerfile vừa tạo. Sau khi tạo xong image, Developer có thể chạy container trên máy local. Hoặc có thể đẩy Docker image vừa tạo lên Docker registry để lưu trữ và chia sẻ.
- Production: Khi Production muốn sử dụng một image được lưu trữ trên Registry để triển khai ở nơi khác, Production chỉ cần kéo image đó về máy bằng lệnh “docker pull”, sau đó chạy container từ image vừa lấy được.

2.1.2. Web Application



Hình 2.2. Mô hình của ứng dụng web

- Registry: Là nơi lưu trữ các image Docker sau khi được tạo ra, giúp chia sẻ và triển khai dễ dàng. Docker Hub là registry phổ biến nhất, nhưng có thể sử dụng các registry khác như GitLab Container Registry hoặc Azure Container Registry.
- Developer: Dockerfile cho từng dịch vụ frontend và backend được tạo để định nghĩa môi trường cần thiết. Đáng chú ý là các Dockerfile sẽ pull image Node.js từ Docker Hub làm nền tảng (base image) cho cả frontend và backend, vì cả hai đều chạy trên nền JavaScript và cần môi trường runtime phù hợp để đảm bảo ứng dụng hoạt động đúng cách. Docker Compose được sử dụng để định nghĩa và quản lý toàn bộ ứng dụng có nhiều container, bao gồm cả cơ sở dữ liệu MongoDB. Thay vì phải tạo, chạy và kết nối từng container bằng các lệnh Docker riêng lẻ, Docker Compose cho phép định nghĩa tất cả các container cùng cấu hình liên quan trong một tệp docker-compose.yml. Khi Docker Compose được chạy, nó sẽ pull image MongoDB từ registry để tạo container cho cơ sở dữ liệu. Sau khi hoàn thành việc build các image frontend, backend, và database, đăng nhập vào Docker Hub để push lên Registry nhằm mục đích lưu trữ và triển khai. Khi cần kiểm thử, các image có thể được run để tạo các container tương ứng, từ đó chạy ứng dụng web trên Docker.

- Production: Đây là môi trường triển khai ứng dụng chính thức, nơi các image từ Registry được kéo về và chạy để phục vụ người dùng cuối.

2.2. Cài đặt

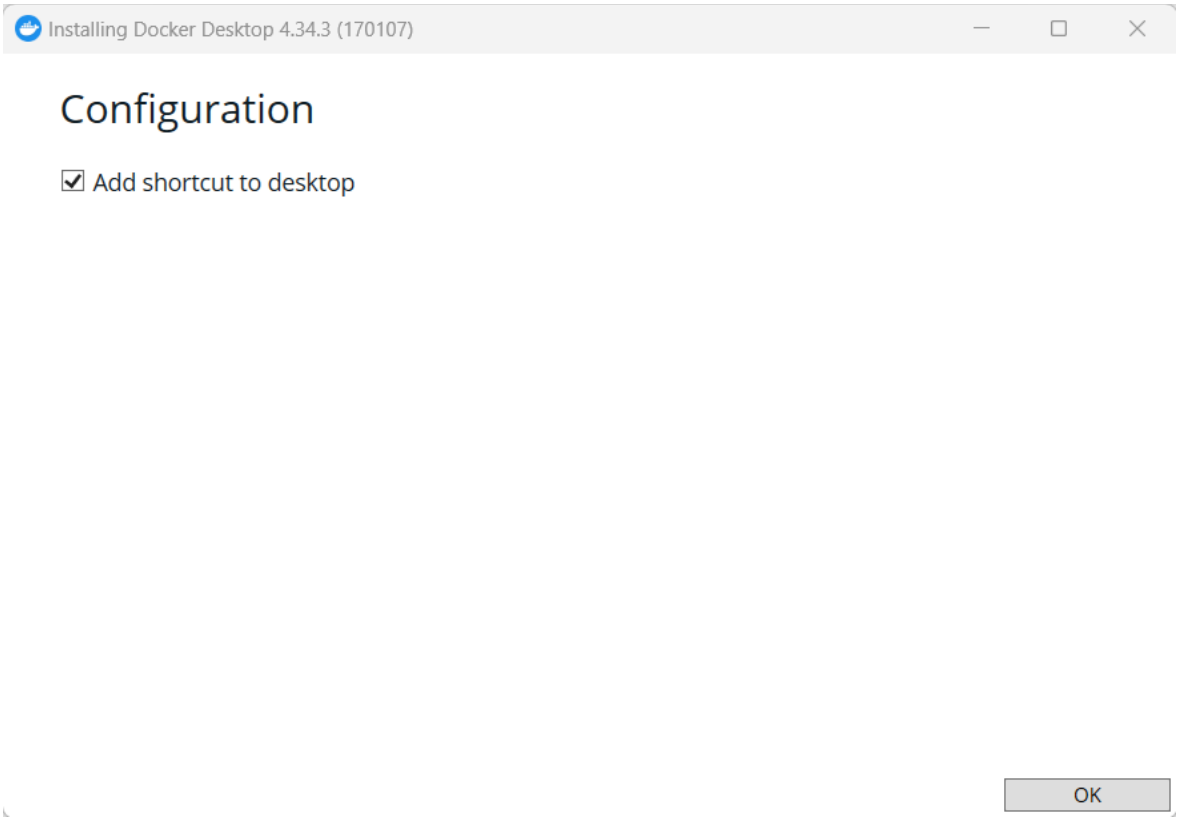
Nhóm chọn chạy trên môi trường Windows để cài Docker, vì vậy cần cài thêm WSL2 để cung cấp nền tảng để Docker Desktop hoạt động đúng như mong muốn.

2.2.1. Cài đặt WSL2

- Đầu tiên, khởi chạy command prompt.
- Tiếp theo, nhập lệnh **wsl --install** để cài đặt.
- Sau khi quá trình cài đặt thực hiện xong thì khởi động lại và nhập một số thông tin theo yêu cầu là hoàn tất.

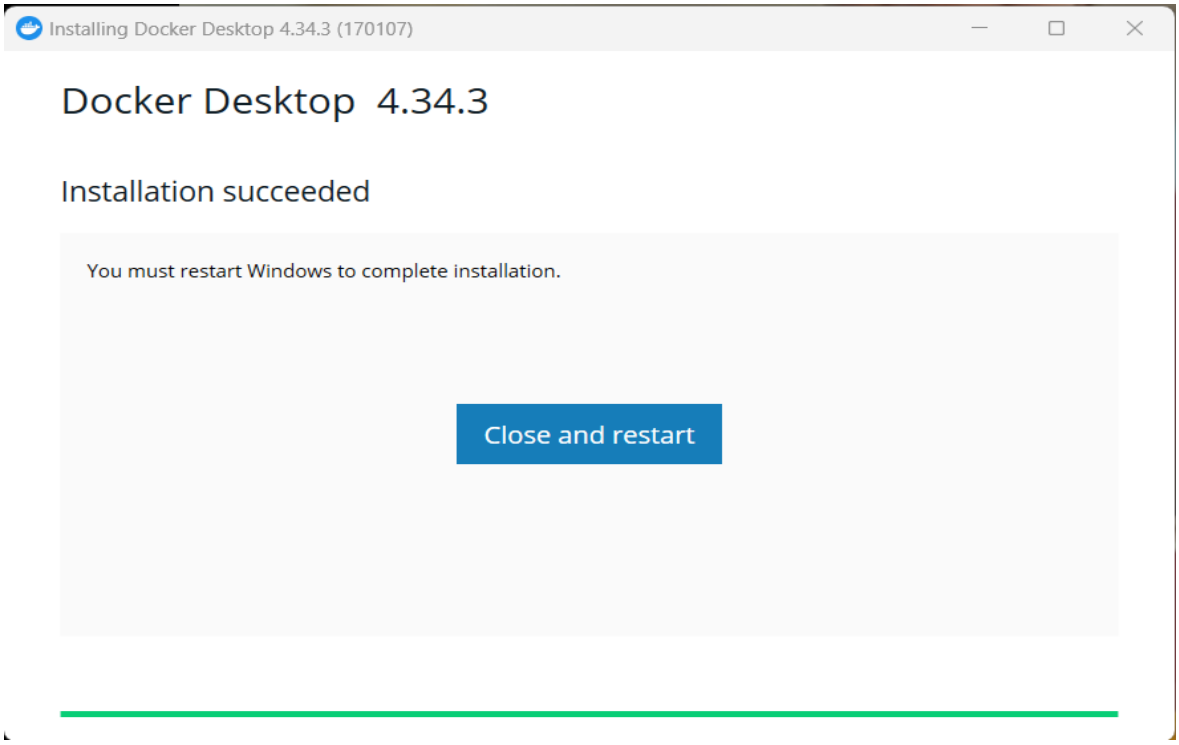
2.2.2. Cài đặt Docker Desktop

- Bước 1: Truy cập vào trang chủ docker và tải xuống tệp Docker Desktop Installer.exe
- Bước 2: Chạy trình cài đặt với quyền quản trị.
- Bước 3: Giữ nguyên các tùy chọn mặc định và nhấn OK.



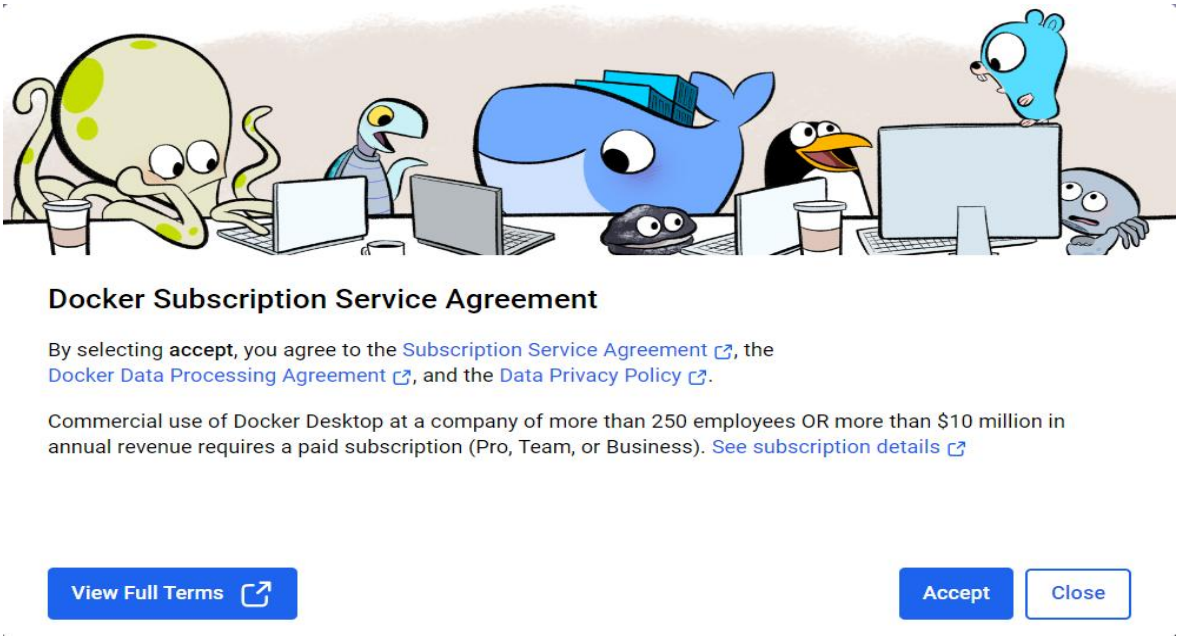
Hình 2.3. Giao diện cài đặt Docker Desktop (1)

- Bước 4: Chờ trình cài đặt cài đặt Docker cho hệ thống.
- Bước 5: Sau khi trình cài đặt chạy xong, nhấn chọn “Close and restart” để hoàn thành quá trình cài đặt.



Hình 2.4. Giao diện cài đặt Docker Desktop (2)

- Bước 6: Nhấn chọn “Accept” để chấp nhận các thỏa thuận dịch vụ đăng ký Docker. Docker Desktop đã được cài đặt thành công.



Hình 2.5. Giao diện thỏa thuận dịch vụ đăng ký của Docker Desktop

2.3. Cấu hình

2.3.1. Cấu hình cơ bản

Source code: https://github.com/solivaquaant/landing-page

Để triển khai một trang web tĩnh cơ bản, nhóm tham khảo một mẫu website tĩnh chỉ sử dụng các ngôn ngữ cơ bản như HTML, CSS, JavaScript. Nhóm bắt đầu triển khai trang web bằng việc viết Dockerfile, có nội dung như hình bên dưới:

```
FROM nginx:alpine

COPY ./ /usr/share/nginx/html

EXPOSE 80
```

Hình 2.6. Dockerfile cho triển khai một website tĩnh

Nhóm sẽ giải thích chi tiết nội dung của tệp Dockerfile mà nhóm đã viết:

- FROM nginx:alpine - Thiết lập docker image cơ sở là nginx image với tag là alpine.
- COPY . /usr/share/nginx/html - Sao chép tất cả các tệp và thư mục có trong thư mục hiện tại vào thư mục gốc của nginx.
- EXPOSE 80 - chỉ định container sẽ lắng nghe các kết nối trên port 80.

Sau khi viết Dockerfile, nhóm bắt đầu với lệnh xây dựng docker image từ Dockerfile, mở terminal và di chuyển đến thư mục chứa Dockerfile. Nhóm sử dụng câu lệnh **docker build -t landing-page .**

```
PS D:\source-code\landing-page> docker build -t landing-page .
[+] Building 4.7s (8/8) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                              0.1s
 => => transferring dockerfile: 99B                                                                                               0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                                   3.6s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                                      0.0s
 => [internal] load .dockerignore                                                                                                 0.1s
 => => transferring context: 2B                                                                                                   0.0s
 => [internal] load build context                                                                                                 0.2s
 => => transferring context: 40.06kB                                                                                              0.2s
 => CACHED [1/2] FROM docker.io/library/nginx:alpine@sha256:2140dad235c130ac861018a4e13a6bc8aea3a35f3a40e20c1b060d51a7efd250      0.0s
 => [2/2] COPY . /usr/share/nginx/html                                                                                            0.3s
 => exporting to image                                                                                                            0.2s
 => => exporting layers                                                                                                           0.1s
 => => writing image sha256:6c1b101d5a7b36851cabe2ac66da290aedbff5bd480e60ead1c97f4af599a9ea                                      0.0s
 => => naming to docker.io/library/landing-page                                                                                   0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/n4305p5rqsu1kmj1tgrlyyflm

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\source-code\landing-page>
```

Hình 2.7. Tạo Docker image

Trong đó:

- docker build: là câu lệnh để tạo ra một image mới dựa trên các chỉ thị được định nghĩa bên trong file Dockerfile
- -t: dùng để đặt tên và tag cho image mới.
- landing-page: là tên nhóm đặt cho image đang build. Tùy theo nhu cầu mà người dùng có thể gắn tag cho image bằng cú pháp <tên_image>:<tên_tag>, nếu không đặt tag thì sẽ được đặt theo mặc định là "lastest".
- .: chỉ định đường dẫn đến thư mục hiện tại và Docker sẽ tìm kiếm tệp Dockerfile trong thư mục này và sử dụng nó để xây dựng hình ảnh.

Sau khi hoàn tất quá trình xây dựng image, nhóm kiểm tra lại image vừa tạo bằng lệnh **docker image ls** để liệt kê tất cả các image có sẵn trong máy.

```
PS D:\source-code\landing-page> docker image ls
REPOSITORY                TAG       IMAGE ID       CREATED              SIZE
landing-page              latest    6c1b101d5a7b   39 seconds ago       47MB
solivaquaant/to-do-list   backend   e1486bae746d   59 minutes ago       161MB
solivaquaant/to-do-list   frontend  07c41db65274   About an hour ago    652MB
solivaquaant/to-do-list   mongo     77c59b638412   12 days ago          855MB
PS D:\source-code\landing-page>
```

Hình 2.8. Danh sách các image đang tồn tại trên máy tính

Kế tiếp, nhóm sẽ khởi chạy một container với câu lệnh **docker run -d --rm -p 80:80 landing-page**.

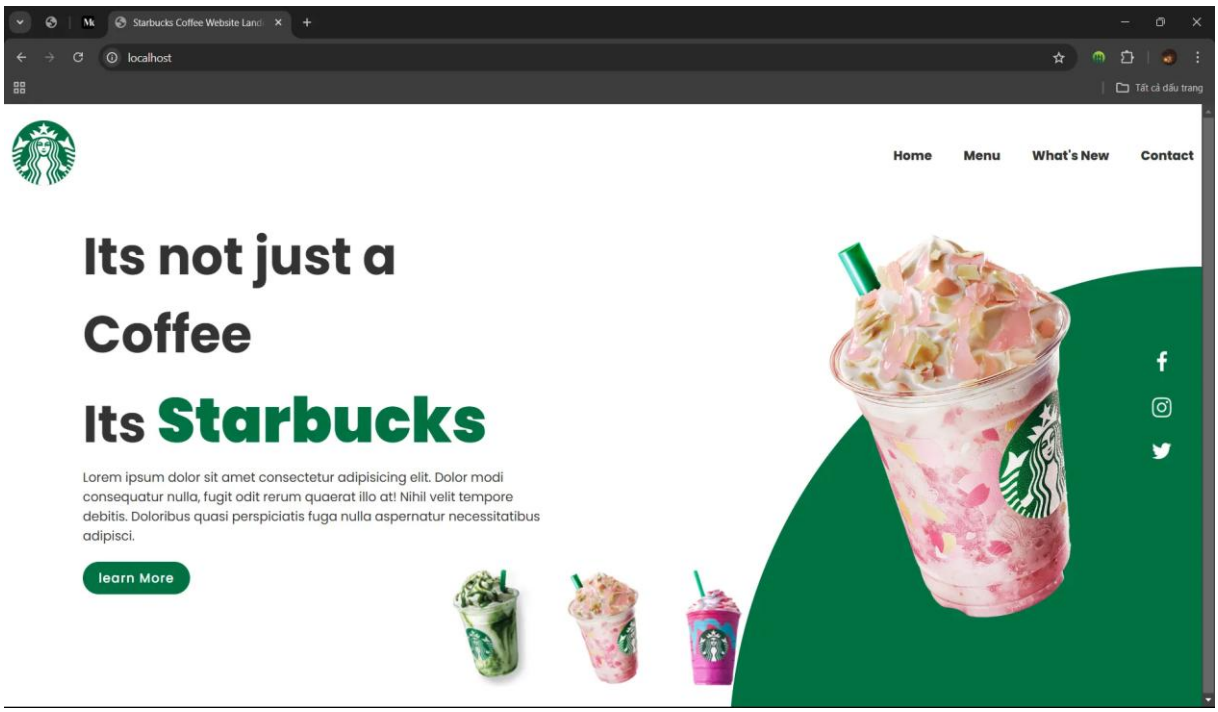
```
PS D:\source-code\landing-page>  docker run -d --rm -p 80:80 landing-page
ffec129dbab4d25d4aa0f305c001b00c57ab7b0fec250f3b3aef2be82f162467
PS D:\source-code\landing-page>
```

Hình 2.9. Chạy Docker container

Câu lệnh bao gồm các thành phần sau:

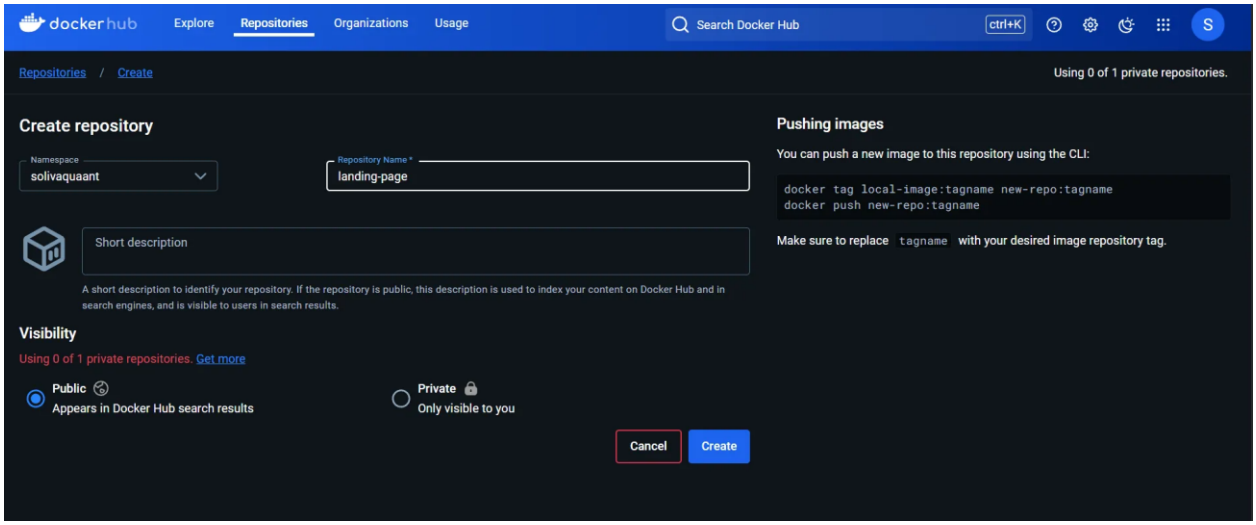
- docker run: là lệnh để tạo và chạy một container.
- -d: chạy với chế độ detached mode - container sẽ chạy ở nền và không chiếm giữ cửa số dòng lệnh hiện tại.
- -p 80:80: mở cổng để kết nối từ host đến container. Ở đây, các yêu cầu từ máy host trên port 80 sẽ được chuyển đến port 80 của container để xử lý.

Lúc này có thể truy cập vào đường dẫn http://localhost:80/ để xem trang web mà nhóm vừa triển khai.



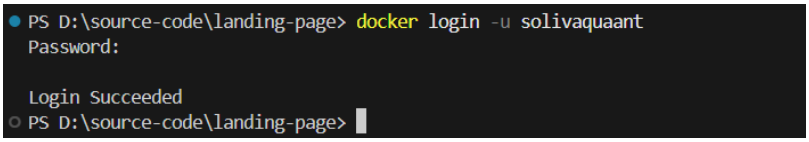
Hình 2.10. Giao diện website được triển khai

Phục vụ cho quá trình "vận chuyển" và triển khai trên các máy tính khác, nhóm sẽ đẩy image của dự án lên Docker Hub. Trước hết, nhóm tạo 1 repository trên Docker Hub.



Hình 2.11. Tạo repository trên Docker Hub

Tại terminal, nhóm đăng nhập vào Docker Hub bằng câu lệnh **docker login -u** <tên_tài_khoản> và nhập mật khẩu.



Hình 2.12. Đăng nhập vào Docker Hub

Trước khi đẩy lên Docker Hub, nhóm sẽ gắn tag cho image bằng lệnh **docker tag** <tên_image_chỉ_định> <tên_tài_khoản>/<tên_image>**:**<tên_tag>. Việc gắn tag này giúp cho việc quản lý của người dùng. Sau khi gắn tag, nhóm sử dụng lệnh **docker image ls** để xem danh sách các image đang có trong máy và thấy có một bản sao của image được chỉ định đã được tạo ra.

```
PS D:\source-code\landing-page> docker tag landing-page solivaquaant/landing-page:v1
PS D:\source-code\landing-page> docker image ls
REPOSITORY                  TAG       IMAGE ID       CREATED             SIZE
landing-page                latest    9e8a47e2415e   3 minutes ago       47MB
solivaquaant/landing-page   v1        9e8a47e2415e   3 minutes ago       47MB
solivaquaant/to-do-list     backend   e1486bae746d   About an hour ago   161MB
solivaquaant/to-do-list     frontend  07c41db65274   About an hour ago   652MB
solivaquaant/to-do-list     mongo     77c59b638412   12 days ago         855MB
PS D:\source-code\landing-page>
```

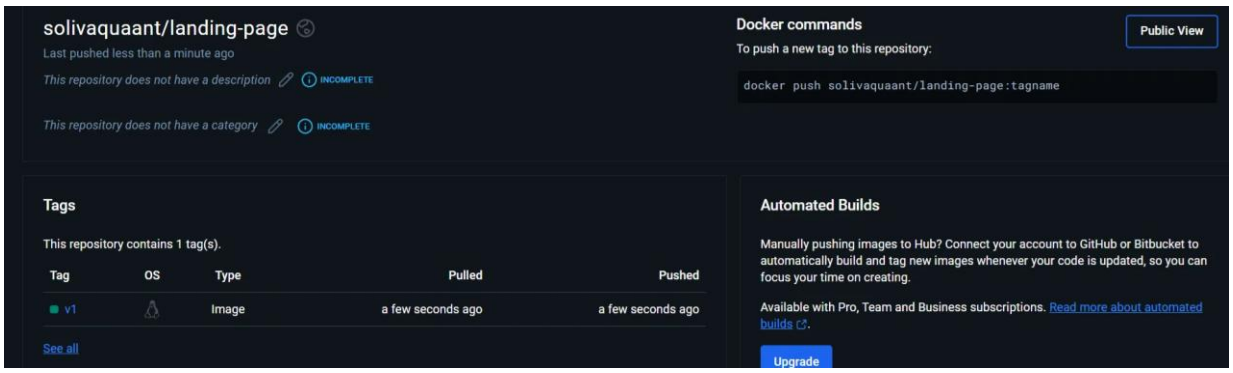
Hình 2.13. Gắn tag cho image và kiểm tra lại

Sau khi đã hoàn tất quá trình gắn tag, nhóm sẽ đẩy các image lên Docker Hub với lệnh **docker push** <tên_tài_khoản>/<tên_image>**:**<tên_tag>

```
PS D:\source-code\landing-page> docker push solivaquaant/landing-page:v1
The push refers to repository [docker.io/solivaquaant/landing-page]
ea8927f15a71: Pushed
528b47987bcf: Mounted from library/nginx
a533c9e2e114: Mounted from library/nginx
6033613561cc: Mounted from library/nginx
0de02d5b2d31: Mounted from library/nginx
f80bfdacda57: Mounted from library/nginx
1241fe31c0bf: Mounted from library/nginx
4e9e0d6ba2cc: Mounted from library/nginx
63ca1fbb43ae: Mounted from solivaquaant/to-do-list
v1: digest: sha256:b1f458eb91a3326a1053d3859f3b511becc19e61bdec4b8c3e6bcb91acf5e6ec size: 2198
PS D:\source-code\landing-page>
```

Hình 2.14. Đẩy image lên Docker Hub

Khi image được đẩy lên thành công, ở phần Repository trong giao diện Docker Hub sẽ hiển thị thông tin của image đó.



Hình 2.15. Danh sách image đã push lên Docker Hub

Lúc này, các máy khác đã có thể dễ dàng kéo image về và chạy một cách dễ dàng ở khắp mọi nơi chỉ với lệnh **docker pull -a** <tên_tài_khoản>/<tên_image>.

```
C:\Users\ASUS> docker pull -a solivaquaant/landing-page
v1: Pulling from solivaquaant/landing-page
43c4264eed91: Already exists
d1171b13e412: Already exists
596d53a7de88: Already exists
f99ac9ba1313: Already exists
fd072e74e282: Already exists
379754eea6a7: Already exists
45eb579d59b2: Already exists
472934715761: Already exists
002439eecfb8: Already exists
Digest: sha256:b1f458eb91a3326a1053d3859f3b511becc19e61bdec4b8c3e6bcb91acf5e6ec
Status: Downloaded newer image for solivaquaant/landing-page
docker.io/solivaquaant/landing-page

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview solivaquaant/landing-page
```

Hình 2.16. Kéo image về máy

Sau khi kéo image về máy, mọi người có thể dùng lệnh **docker image ls** để liệt kê các image tồn tại trong máy.

```
C:\Users\ASUS>docker image ls
REPOSITORY                TAG       IMAGE ID       CREATED             SIZE
solivaquaant/landing-page   v1        9e8a47e2415e   7 minutes ago       47MB
solivaquaant/to-do-list     backend   e1486bae746d   About an hour ago   161MB
solivaquaant/to-do-list     frontend  07c41db65274   About an hour ago   652MB
```

Hình 2.17. Liệt kê danh sách image

Chạy container dựa trên image vừa pull về từ Docker Hub bằng lệnh **docker run** như đã nêu ở trên.

```
C:\Users\ASUS>docker run -d --rm -p 80:80 solivaquaant/landing-page:v1
71a1cf4b6fe2289f6356d265825fe8e67571fd48b9bbe7ebf56774df0fe00004
```

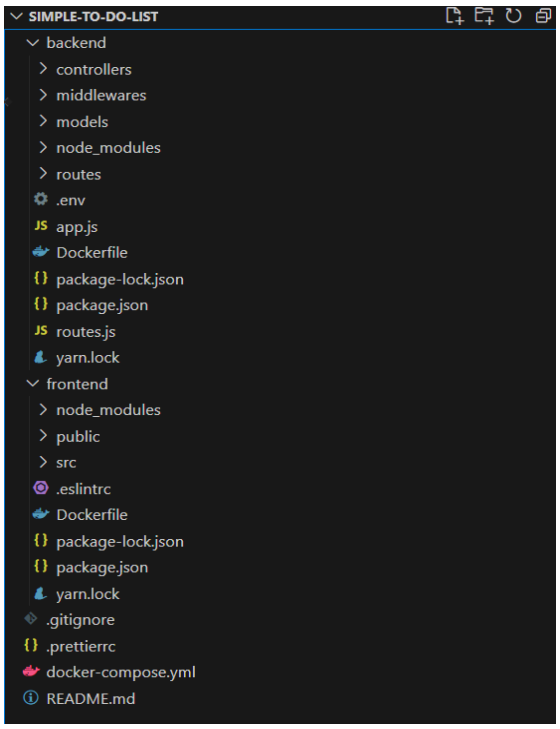
Hình 2.18. Chạy container

Truy cập vào trang web vừa triển khai bằng đường dẫn http://localhost:80/

2.3.2. Cấu hình nâng cao

Source code: https://github.com/solivaquaant/simple-to-do-list

Để triển khai 1 website có đầy đủ frontend, backend và database, nhóm sẽ viết 2 file Dockerfile riêng dành cho frontend và backend, 1 file docker-compose.yml để kết nối các dịch vụ.



Hình 2.19. Cấu trúc thư mục của ứng dụng web

Đầu tiên sẽ viết Dockerfile cho frontend:

- FROM node:18-alpine: Sử dụng image node phiên bản 18 với Alpine Linux làm nền tảng cho container, giúp giảm kích thước container.
- WORKDIR /frontend: Đặt thư mục làm việc trong container là /frontend. Các lệnh tiếp theo sẽ được thực hiện trong thư mục này.
- COPY package*.json ./: Sao chép các tệp package.json và package-lock.json từ máy tính vào thư mục hiện tại (/frontend) trong container.
- RUN npm install: Chạy lệnh npm install để cài đặt các gói phụ thuộc cho ứng dụng frontend.
- COPY . .: Sao chép toàn bộ nội dung của thư mục dự án trên máy tính vào thư mục hiện tại (/frontend) trong container.
- RUN npm run build: Chạy lệnh npm run build để build ứng dụng thành các tệp tĩnh trong thư mục build.
- RUN npm install -g serve: Cài đặt global package serve, một công cụ để phục vụ các tệp tĩnh, giúp triển khai ứng dụng frontend dưới dạng tĩnh.
- EXPOSE 3000: Mở cổng 3000 để container có thể nhận các kết nối trên cổng này.

- CMD ["serve", "-s", "build"]: Đặt lệnh mặc định khi container khởi động là serve -s build, chạy ứng dụng frontend tĩnh từ thư mục build.

```
FROM node:18-alpine

WORKDIR /frontend

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

RUN npm install -g serve

EXPOSE 3000

CMD ["serve", "-s", "build"]
```

Hình 2.20. Dockerfile của frontend

Tiếp theo sẽ viết Dockerfile cho backend như sau:

- FROM node:18-alpine: Sử dụng image node phiên bản 18 với Alpine Linux, một bản phân phối Linux nhẹ, làm nền tảng cho container.
- WORKDIR /backend: Đặt thư mục làm việc trong container là /backend.
- COPY package*.json ./: Sao chép các tệp package.json và package-lock.json vào thư mục hiện tại (/backend) trong container.
- RUN npm install: Chạy lệnh npm install để cài đặt các gói phụ thuộc được liệt kê trong package.json.
- COPY . .: Sao chép toàn bộ nội dung của thư mục dự án trên máy tính vào thư mục hiện tại (/backend) trong container.
- EXPOSE 3001: Mở cổng 3001 để container có thể nhận các kết nối trên cổng này.
- CMD ["node", "app.js"]: Đặt lệnh mặc định để chạy khi container khởi động là node app.js, khởi chạy ứng dụng.

```
FROM node:18-alpine

WORKDIR /backend

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3001

CMD ["node", "app.js"]
```

Hình 2.21. Dockerfile của backend

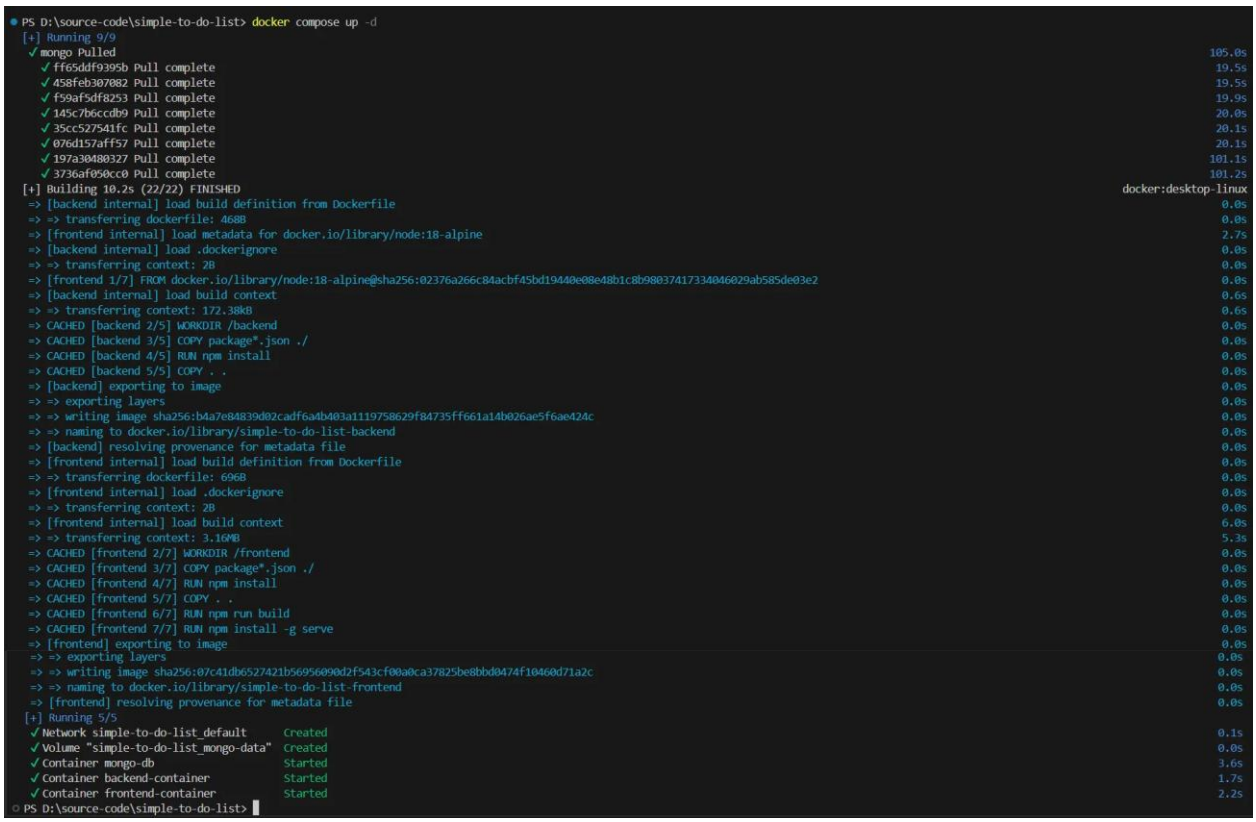
Kế đó, nhóm sẽ tạo 1 file docker-compose.yml:

- services: Định nghĩa các dịch vụ, mỗi dịch vụ tương ứng với một docker image. Trong trường hợp này, chúng em có 3 dịch vụ là backend, frontend và mongo.
- container_name: Đặt tên cho container để dễ quản lý.
- restart: always: Đảm bảo container tự động khởi động lại nếu gặp sự cố.
- build: context: <thư mục>: Xác định thư mục nguồn để Docker tìm và xây dựng Dockerfile của frontend.
- ports: Mở cổng kết nối từ máy host đến container.
- depends_on: Chỉ ra rằng dịch vụ này cần có một dịch vụ tiên quyết được khởi chạy trước. Trong trường hợp này, dịch vụ "backend" chỉ được khởi chạy nếu dịch vụ "mongo" đã khởi chạy, và dịch vụ "frontend" chỉ được khởi chạy nếu dịch vụ "backend" đã khởi chạy.
- env_file: Đọc biến môi trường từ file .env trong thư mục backend. Do ở backend thì connection string đến MongoDB được lưu trong file .env
- volumes: Gắn volume mongo-data vào thư mục /data/db trong container, lưu trữ dữ liệu của MongoDB để đảm bảo không mất dữ liệu khi container bị xóa.
- volumes: mongo-data: Định nghĩa volume mongo-data được gắn vào container mongo nhằm lưu trữ dữ liệu của MongoDB để đảm bảo dữ liệu sẽ không bị mất khi container khởi động lại.

```yaml
services:
  frontend:
    container_name: frontend-container
    restart: always
    build:
      context: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend

  backend:
    container_name: backend-container
    restart: always
    build:
      context: ./backend
    ports:
      - "3001:3001"
    env_file:
      - ./backend/.env
    depends_on:
      - mongo

  mongo:
    container_name: mongo-db
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db

volumes:
  mongo-data:
```

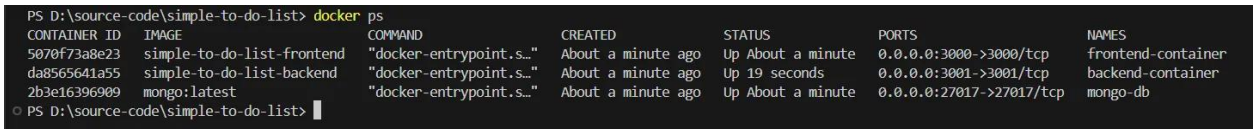
Hình 2.22. Nội dung file docker-compose.yml

Để xây dựng các image và khởi chạy các container, sử dụng lệnh: **docker compose up -d**



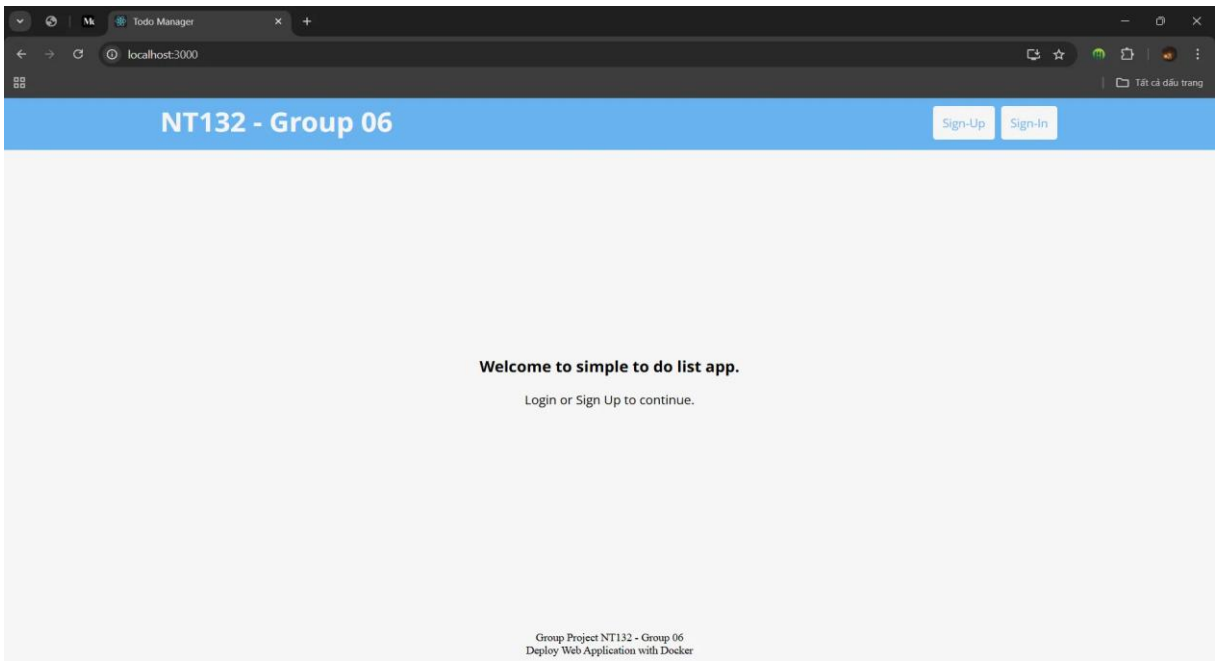
Hình 2.23. Xây dựng các image và khởi chạy các container

Sau khi thực hiện câu lệnh trên, docker sẽ tiến hành build các image được khai báo trong các dịch vụ, sử dụng các image được build hoặc pull và khởi chạy container. Lúc này có thể liệt kê các container đang chạy trên máy tính bằng lệnh **docker ps.**



Hình 2.24. Danh sách các container tồn tại trên máy

Sử dụng trình duyệt truy cập vào đường dẫn http://localhost:3000/ để truy cập trang web đã được triển khai.



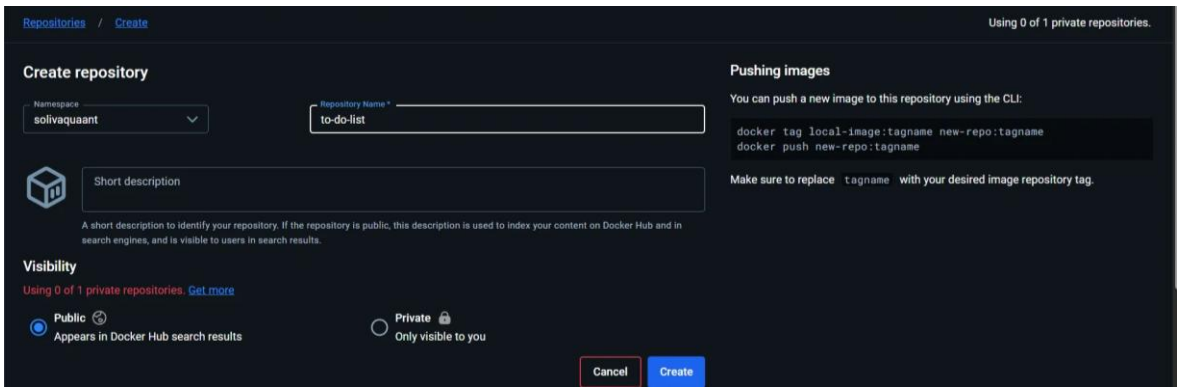
Hình 2.25. Giao diện của ứng dụng web

Trong một số trường hợp cần thiết, có thể sử dụng lệnh **docker compose down** để tiến hành dừng và gỡ bỏ các container đang được chạy.

```
PS D:\source-code\mern-todo-main\simple-to-do-list> docker compose down
 [+] Running 4/4
  ✓ Container frontend-container       Removed
  ✓ Container backend-container        Removed
  ✓ Container mongo-db                 Removed
  ✓ Network simple-to-do-list_default  Removed
PS D:\source-code\mern-todo-main\simple-to-do-list>
```

Hình 2.26. Dừng hoặc gỡ các container

Để phục vụ cho quá trình triển khai trên các máy khác, nhóm sẽ đẩy các image lên Docker Hub. Đầu tiên, nhóm tạo 1 repository trên Docker Hub.

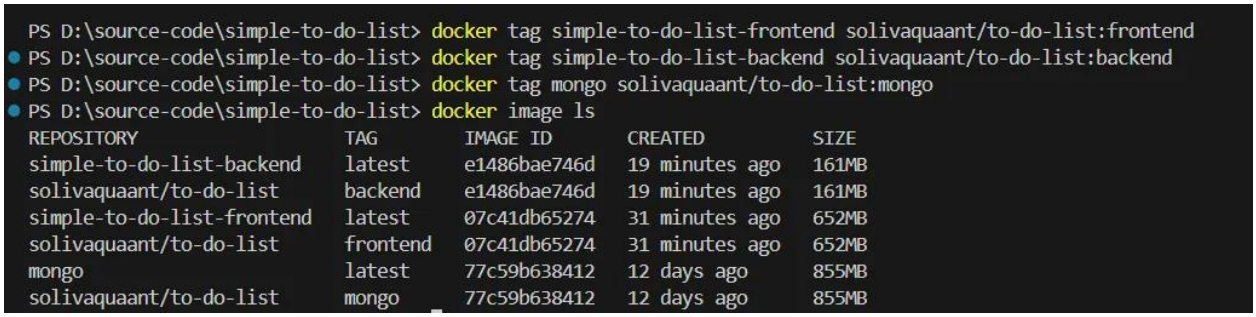


Hình 2.27. Tạo repository trên Docker Hub

Tiến hành đăng nhập vào tài khoản docker bằng lệnh **docker login -u** <*username*>, sau đó nhập mật khẩu.

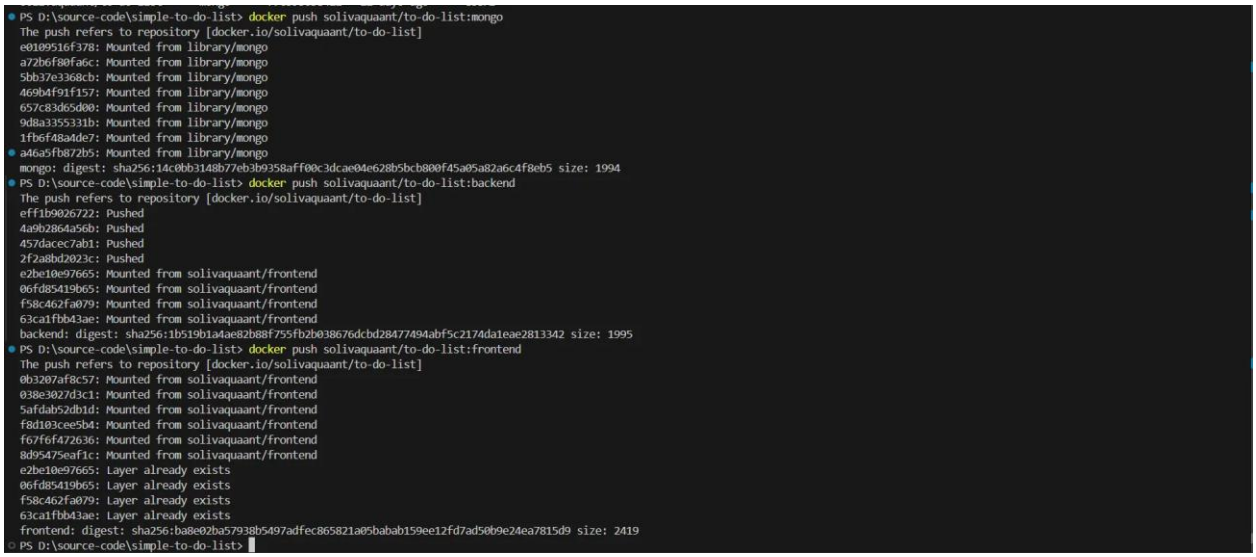
Hình 2.28. Đăng nhập Docker

Để việc quản lý được thuận tiện và dễ dạng hơn, nhóm đã thực hiện gắn tag cho image bằng lệnh **docker tag** *<tên_image> <username>/<tên_repo>:<tag>*. Có thể dùng lệnh **docker image ls** để kiểm tra lại các tag đã gán cho các image.



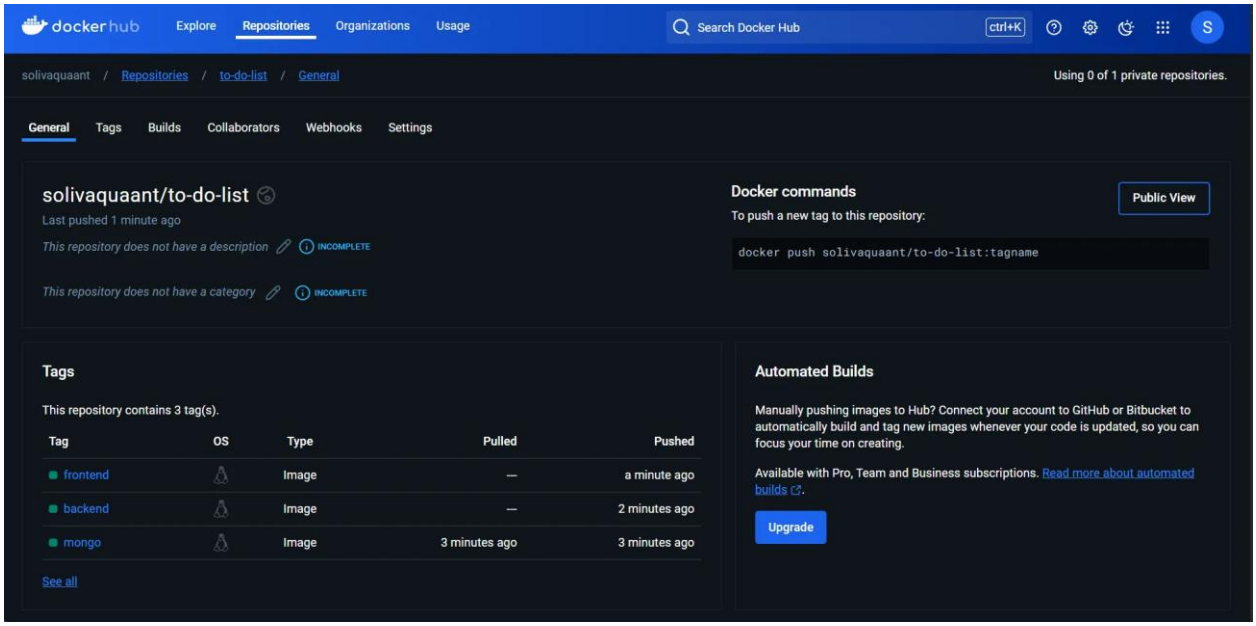
Hình 2.29. Gắn tag cho các image và liệt kê danh sách các image

Sau khi đã gắn tag, nhóm đẩy image từ máy tính lên Docker Hub bằng lệnh **docker push** *<username>/<tên_repo>:<tag>*



Hình 2.30. Đẩy image lên Docker Hub

Để kiểm tra xem các image đã được push lên thành công hay không, nhóm sẽ vào Docker Hub để xem. Quá trình đẩy các image lên diễn ra suôn sẻ.



Hình 2.31. Danh sách các image trên Docker Hub

Để một máy tính khác có thể triển khai được website trên, có thể tải repository về máy tính bằng lệnh **docker pull -a** *<username>/<tên_repos>*

```
C:\Users\ASUS>docker pull -a solivaquaant/to-do-list
backend: Pulling from solivaquaant/to-do-list
Digest: sha256:1b519b1a4ae82b88f755fb2b038676dcbd28477494abf5c2174da1eae2813342
frontend: Pulling from solivaquaant/to-do-list
Digest: sha256:ba8e02ba57938b5497adfec865821a05babab159ee12fd7ad50b9e24ea7815d9
mongo: Pulling from solivaquaant/to-do-list
Digest: sha256:14c0bb3148b77eb3b9358aff00c3dcae04e628b5bcb800f45a05a82a6c4f8eb5
Status: Downloaded newer image for solivaquaant/to-do-list
docker.io/solivaquaant/to-do-list

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview solivaquaant/to-do-list
```

Hình 2.32. Kéo các image về máy

Dùng lệnh **docker image ls** để kiểm tra các image đã được tải về đủ.

```
C:\Users\ASUS>docker image ls
REPOSITORY                  TAG         IMAGE ID       CREATED          SIZE
solivaquaant/to-do-list     backend     e1486bae746d   47 minutes ago   161MB
solivaquaant/to-do-list     frontend    07c41db65274   59 minutes ago   652MB
solivaquaant/to-do-list     mongo       77c59b638412   12 days ago      855MB

C:\Users\ASUS>
```

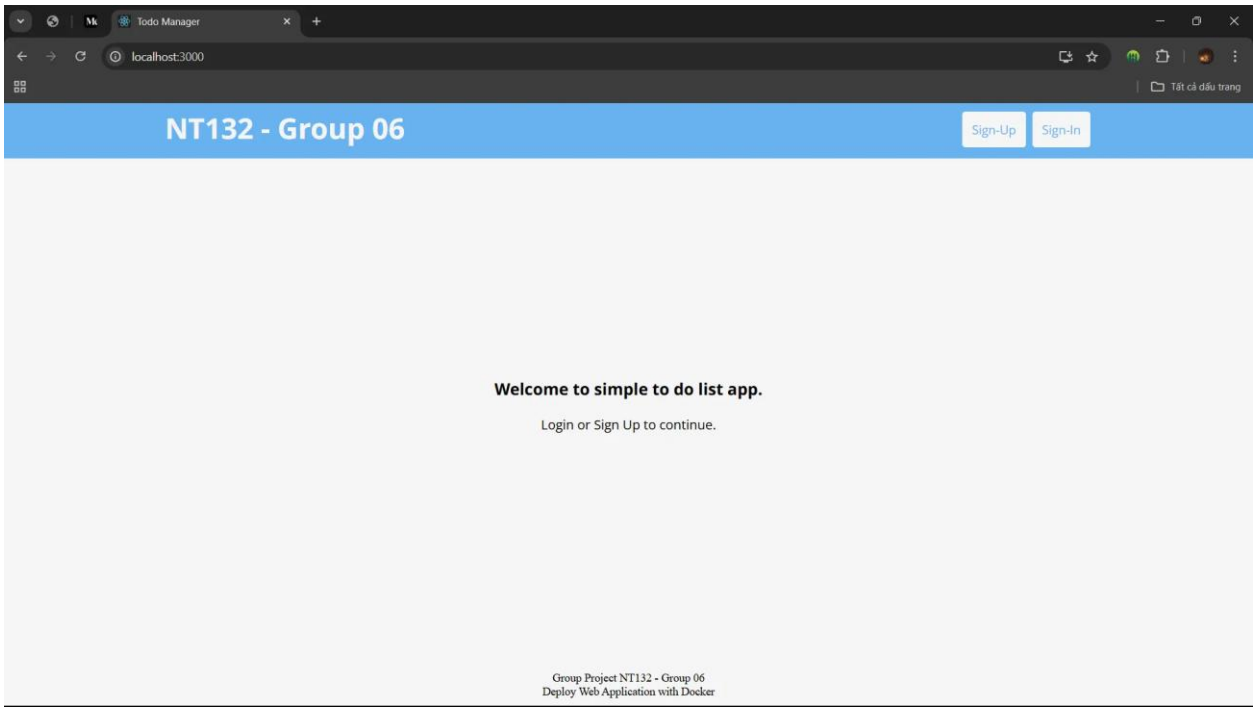
Hình 2.33. Danh sách các image đang tồn tại trên máy

Chạy lần lượt các container dựa trên các image bằng lệnh **docker run -d --rm -p** *<port_trên_máy_host>*:*<port_của_container>* *<username>*/*<tên_image>*:*<tag>*

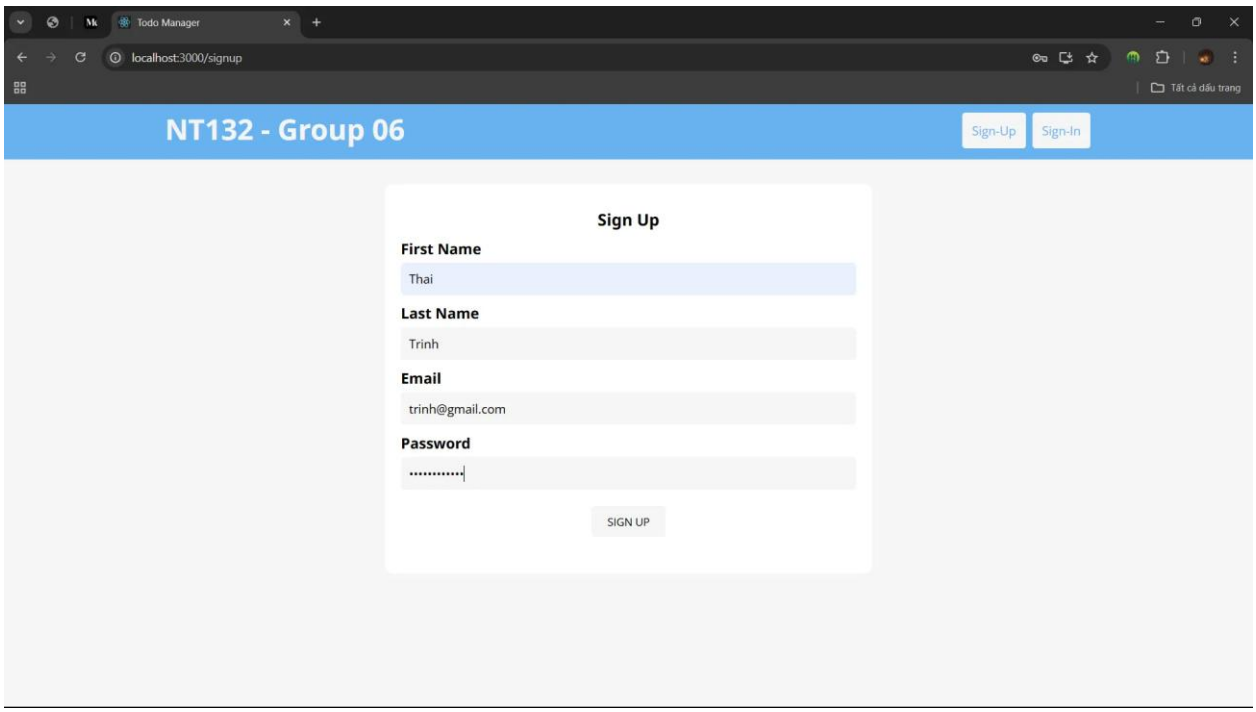
```
C:\Users\ASUS>docker run -d --rm -p 3001:3001 solivaquaant/to-do-list:backend
af882aa6683a100d79653da27634da4a68f75e9deb83389c5544bf636a43ca51

C:\Users\ASUS>docker run -d --rm -p 3000:3000 solivaquaant/to-do-list:frontend
49696810afd0ffc18ee210ca5aa50f86a808c97b7dfc4625d27e9b70f315cf7b

C:\Users\ASUS>docker run -d --rm -p 27017:27017 solivaquaant/to-do-list:mongo
2eaaec791266a712756649fef4c493cef681d84a08700ee855024a7e9bdbfbc7
```

Hình 2.34. Khởi chạy lần lượt các container

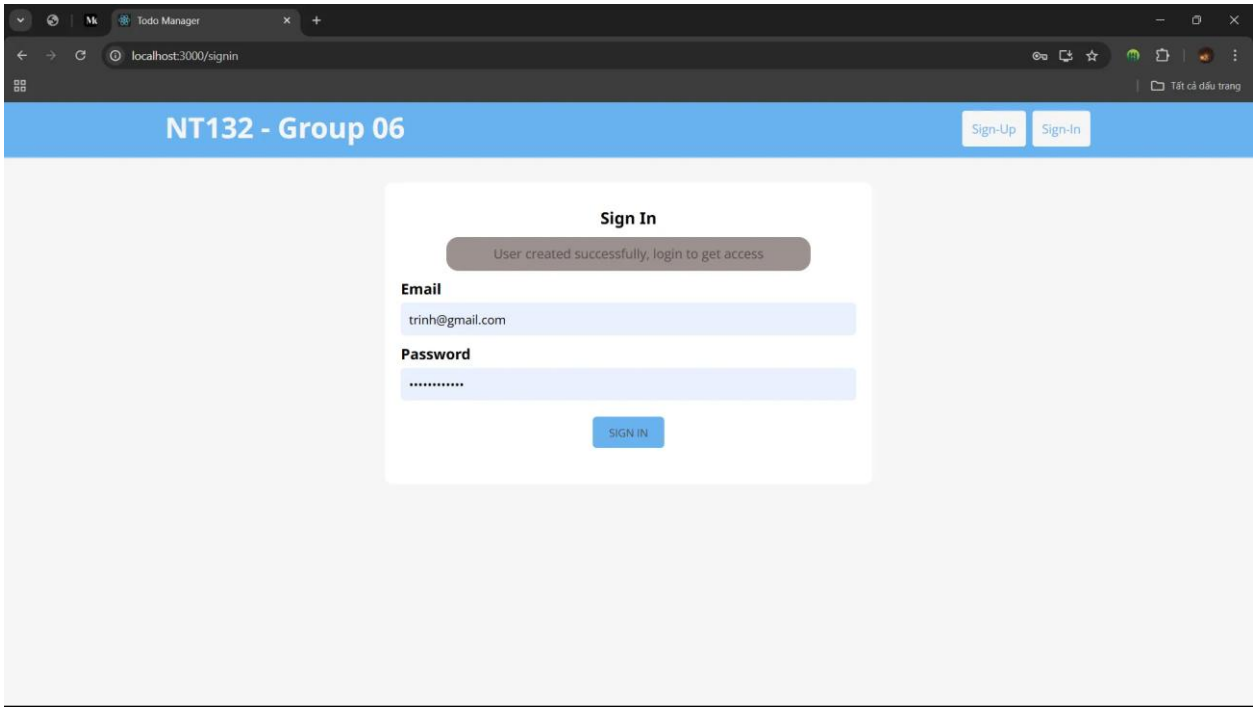
Kiểm tra lại bằng cách truy cập vào đường dẫn http://localhost:3000/



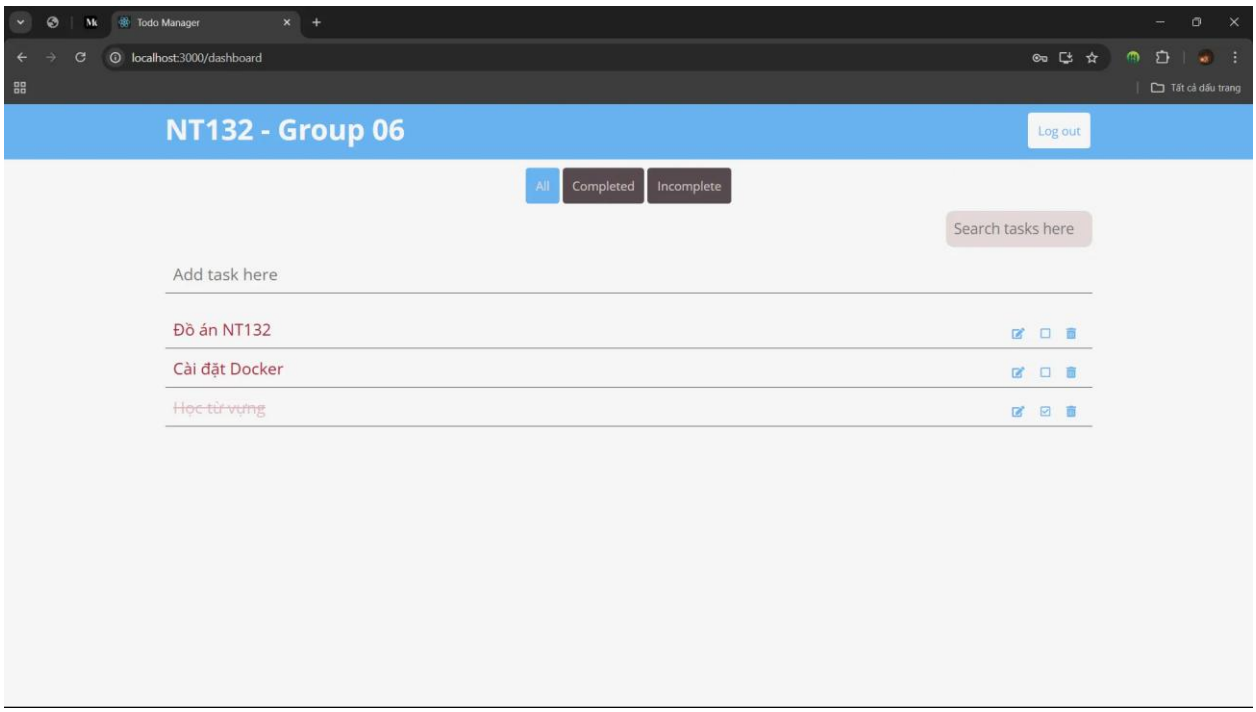
Hình 2.35. Giao diện ứng dụng web



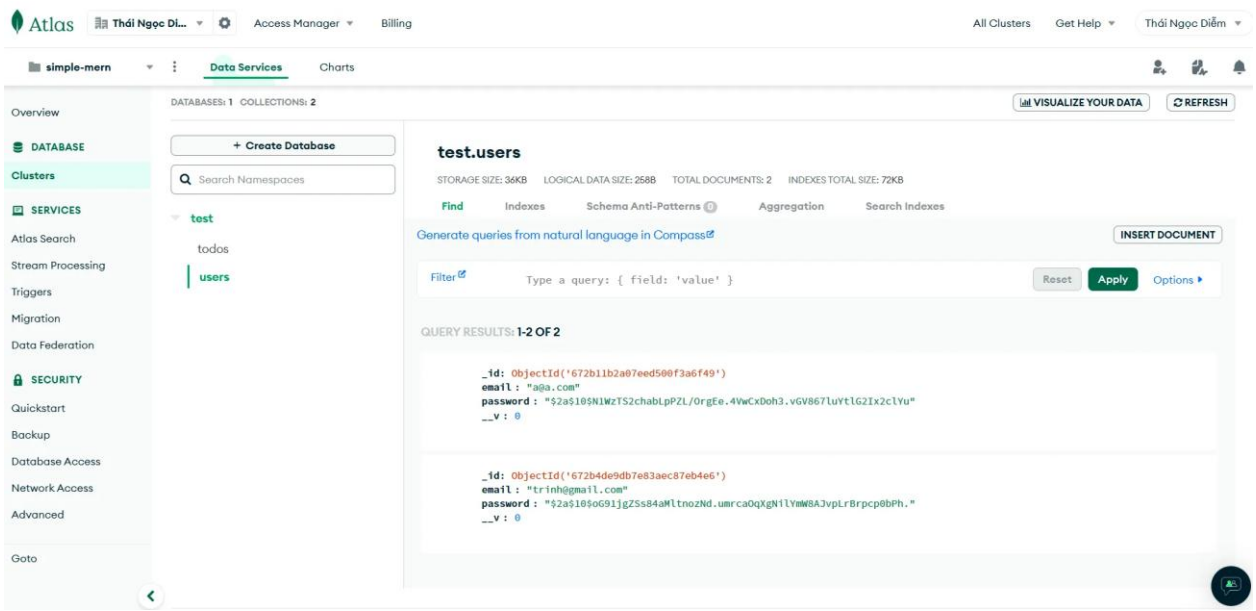
Hình 2.36. Giao diện trang đăng ký



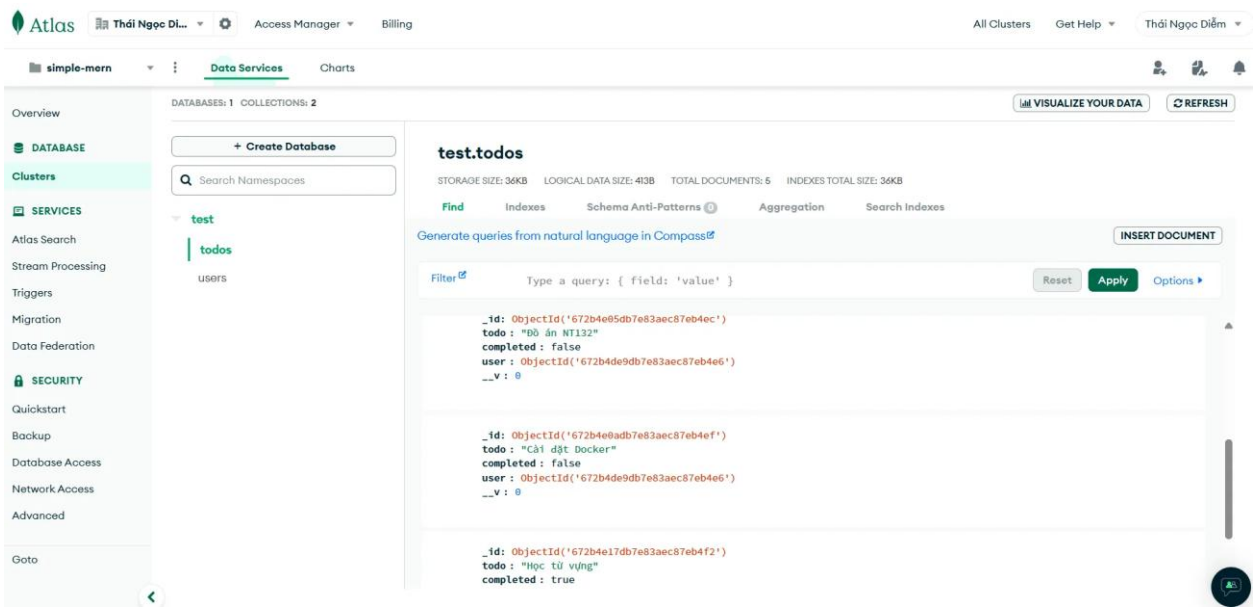
Hình 2.37. Giao diện trang đăng nhập



Hình 2.38. Giao diện sau khi đăng nhập thành công



Hình 2.39. Database MongoDB bảng users



Hình 2.40. Database MongoDB bảng todos

III. Kết quả và kết luận

3.1. Kết quả

Kết quả đạt được:

- Hoàn thành mục tiêu hiểu và triển khai được ứng dụng web bằng Docker. Nắm vững các kiến thức cơ bản như Dockerfile, Docker image, Docker container, Docker Compose,...
- Hiểu được quy trình đóng gói và hoạt động của Docker.
- Thành công triển khai 2 trang web bằng Docker, gồm:
    - Trang web tĩnh
    - Ứng dụng web (frontend, backend và Database)

Ưu điểm:

- Khả năng di động cao: Container có thể chạy trên bất kỳ máy nào chỉ cần có cài đặt Docker.
- Quản lý tài nguyên hiệu quả: các Container chia sẻ cùng một kernel giúp giảm tiêu thụ tài nguyên hệ thống.
- Tốc độ triển khai nhanh: Container nhẹ và triển khai nhanh chóng chỉ mất vài giây.
- Tính bảo mật: Mỗi container là một môi trường riêng biệt giúp giảm thiểu rủi ro bị tấn công.

Nhược điểm:

- Docker Container không thể tự động cập nhật mỗi khi tài nguyên ứng dụng có sự thay đổi, nên sẽ phải rebuild sau mỗi lần cập nhật.
- Có nguy cơ bị tấn công nếu sử dụng các image không đáng tin cậy.
- Docker có thể tốn nhiều tài nguyên khi chạy nhiều container và các ứng dụng lớn.

3.2. Kết luận

Sau 3 tuần kể từ lúc nhận đồ án, nhóm đã thực hiện được hầu hết các yêu cầu cơ bản và nâng cao trong việc triển khai web tĩnh và ứng dụng web bằng docker.

Nhờ việc sử dụng Docker Compose, quá trình triển khai web đã trở nên dễ dàng và nhanh chóng, giúp nâng cao hiệu suất bằng cách giảm thiểu thời gian khởi động và

tăng tốc độ phản hồi, đồng thời đơn giản hóa quá trình quản lý và triển khai nhiều dịch vụ khác nhau. Bên cạnh đó, cơ chế container còn hỗ trợ cho tính mở rộng của quy mô ứng dụng web.

Kết quả, thời gian triển khai một web mới đã giảm từ 2 giờ xuống còn 15 phút, đồng thời tăng khả năng chịu lỗi của hệ thống. Điều này đã góp phần nâng cao chất lượng sản phẩm và tối ưu thời gian của quá trình phát triển web.

Trong quá trình nghiên cứu và triển khai, nhóm cũng gặp một số khó khăn liên quan đến việc quản lý mạng và lưu trữ dữ liệu. Những trải nghiệm do đồ án mang lại đã giúp nhóm có thêm kinh nghiệm lên kế hoạch chi tiết thực hiện một dự án, học được các kiến thức về docker và web, thêm vào đó là củng cố tinh thần làm việc nhóm, hỗ trợ lẫn nhau, phối hợp và phân chia công việc.

Đồ án này đã cho nhóm thấy được nhiều tiềm năng của Docker Compose. Với những kiến thức và kinh nghiệm đã tích lũy được, nhóm tin rằng có thể áp dụng Docker vào các dự án trong tương lai để nâng cao tính bảo mật và khả năng mở rộng.

TÀI LIỆU THAM KHẢO

1. *LearnDocker*. (n.d.). LearnDocker. https://learndocker.online/
2. *Play with Docker Classroom*. (n.d.). https://training.play-with-docker.com/
3. Potnuru, R. (2023, February 4). *Dockerizing Your MERN Stack App: A Step-by-Step Guide.* DEV Community. https://dev.to/itsrakesh/dockerizing-your-mern-stack-app-a-step-by-step-guide-19nh
4. *"What is Docker?"* (2024, September 10). Docker Documentation. https://docs.docker.com/get-started/docker-overview/

PHỤ LỤC

1. Bảng phân công

Tên - MSSV	Công việc phân công	Mức độ hoàn thành
Nguyễn Khánh Linh 22520769	- Tìm hiểu nội dung về Docker. - Cài đặt Docker. - Tìm kiếm trang web để thực hiện yêu cầu cơ bản. - Thực hiện yêu cầu cơ bản. - Soạn nội dung Slide. - Viết báo cáo và trả lời các câu hỏi.	100%
Trần Thiên Thanh 22521367	- Tìm hiểu nội dung về Docker. - Cài đặt Docker. - Tìm kiếm trang web để thực hiện yêu cầu nâng cao. - Thực hiện yêu cầu nâng cao. - Thuyết trình. - Viết báo cáo và trả lời các câu hỏi.	100%
Phạm Thị Cẩm Tiên 22521473	- Tìm hiểu nội dung về Docker. - Cài đặt Docker. - Tìm kiếm trang web để thực hiện yêu cầu cơ bản. - Thực hiện yêu cầu cơ bản. - Thuyết trình. - Viết báo cáo và trả lời các câu hỏi.	100%

Thái Ngọc Diễm Trinh 22521541	- Tìm hiểu nội dung về Docker. - Cài đặt Docker. - Tìm kiếm trang web để thực hiện yêu cầu nâng cao. - Thực hiện yêu cầu nâng cao. - Chuẩn bị video Demo. - Viết báo cáo và trả lời các câu hỏi.	100%

Bảng 2. Bảng phân công công việc

2. Bảng đánh giá

Tiêu chí đánh giá	Điểm tự đánh giá	Ghi chú
Report format (1 point)	1	Báo cáo đầy đủ các nội dung yêu cầu với format thống nhất.
Presentation (1 point)	1	Nhóm xung phong thuyết trình trong buổi đầu. Tất cả thành viên đều chuẩn bị kỹ và thuyết trình tốt, rành mạch, không đọc slide; giúp người nghe hiểu được nội dung. Đồng thời, nhóm có chuẩn bị các câu hỏi ôn tập để tương tác với người nghe.
Theory (2 point)	2	Trình bày khái niệm của Docker, cơ chế hoạt động dựa trên sự hiểu biết của nhóm, không bị copy từ trên Internet.
Demonstration (5 point)	5	Thực hiện đầy đủ và hoàn thành được toàn bộ nội dung phần cơ bản (Deploy a static web) và nâng cao (Deploy a web application on Docker container).

Tổng điểm	**9**	

Bảng 3. Bảng điểm tự đánh giá

3. **Trả lời câu hỏi**

<u>Câu 1:</u> Thì trong trường hợp ví dụ docker container bị vấn đề khiến nó die , thì có cách nào config sao cho khi nó die thì nó vẫn có thể tự động khôi phục lại được không?

Các cách cấu hình để container có thể tự khởi động lại khi gặp sự cố:

- Triển khai ứng dụng với Restart Policy: thêm tùy chọn restart khi chạy lệnh "docker run" hoặc trong tệp cấu hình docker-compose.yml, điều này giúp tự động khởi động lại container khi nó gặp lỗi.
- Sử dụng Docker Swarm: công cụ được tích hợp sẵn trong Docker giúp triển khai và quản lý các container, cung cấp khả năng tự khôi phục container khi có lỗi.
- Triển khai web bằng Docker kết hợp với Kubernetes: Kubernetes quản lý, điều phối các container được tạo ra bởi Docker và nó có cơ chế tự động khôi phục container nếu container bị dừng hoặc lỗi.

<u>Câu 2:</u> Làm thế nào để tạo ra một Docker Image?

Để tạo ra một Docker Image ta thực hiện các bước sau:

- Bước 1: Tạo Dockerfile
  - Tạo Dockerfile tại thư mục dự án.
  - Viết nội dung Dockerfile với nội dung là định nghĩa cách mà Image sẽ được xây dựng.
- Bước 2: Xây dựng Docker Image
  - Điều hướng đến thư mục chứa file Dockerfile đã tạo.
  - Dùng lệnh "**docker build -t** <tên_image> **.**" để dựng Image dựa trên Dockerfile đã viết. Trong đó:
    - **-t** - là tùy chọn dùng để đặt tên cho image.
    - **.** - là đường dẫn tới thư mục hiện tại, nơi chứa Dockerfile.

<u>Câu 3:</u> Làm thế nào để tích hợp HTTPS vào ứng dụng web chạy trên Docker ?

- Bước 1: Chuẩn bị chứng chỉ SSL.
- Bước 2: Cấu hình web server hỗ trợ HTTPS: Có thể sử dụng Nginx hoặc Node.js.
    - Nếu sử dụng Nginx:
        - Cấu hình Nginx để lắng nghe các kết nối HTTPS (443) và chỉ ra nơi lưu chứng chỉ SSL.
        - Chuyển hướng từ HTTP sang HTTPS.
    - Nếu sử dụng Node.js:
        - Tạo một server HTTPS bằng cách đọc file chứng chỉ và khóa private key.
- Bước 3: Tích hợp vào Docker:
    - Định nghĩa lại file Dockerfile để chứa các file cấu hình hoặc chứng chỉ SSL.
    - Khai báo các service cùng các file chứng chỉ SSL để tích hợp HTTPS vào file docker-compose.yml.
- Bước 4: Khởi chạy Docker container và kiểm tra ứng dụng bằng cách truy cập qua HTTPS.

<u>Câu 4:</u> Cách tối ưu hóa Docker Image để giảm kích thước?

- Chọn base image phù hợp: các base image nhẹ như alpine, scratch,... sẽ giúp giảm kích thước image cuối cùng.
- Sử dụng kỹ thuật tối ưu hóa Multi-stage builds: kỹ thuật này sử dụng nhiều giai đoạn để tách việc xây dựng và chạy ứng dụng, người dùng chỉ cần sao chép những phần cần thiết từ giai đoạn build vào image cuối cùng.
- Sử dụng file .dockerignore: file này giúp loại bỏ những file và thư mục không cần thiết khỏi quá trình build.
- Tối ưu hóa cách viết Dockerfile: gộp các lệnh RUN thành một lệnh duy nhất để giảm thiểu layer.

- Sử dụng tính năng Docker BuildKit của Docker: tính năng này cung cấp các option cho việc tối ưu hóa.
- Chỉ cài đặt những dependencies cần thiết.

<u>Câu 5:</u> Docker có phù hợp với mọi loại ứng dụng web không? Hãy cho ví dụ.

Docker phù hợp với các loại ứng dụng sau:

- Các ứng dụng web có kiến trúc Microservice chia ứng dụng thành các dịch vụ nhỏ, độc lập, dễ dàng phát triển, triển khai và quản lý. Docker sẽ đóng gói từng dịch vụ vào các container nhỏ gọn, dễ dàng quản lý và mở rộng.

Ví dụ: Netflix,...

- Các ứng dụng yêu cầu môi trường đồng nhất. Docker tạo ra container với môi trường độc lập, đảm bảo ứng dụng sẽ chạy giống nhau trên mọi hệ thống.

Ví dụ: Wordpress,...

- Các ứng dụng web có lưu lượng truy cập cao, Docker có thể tăng khả năng xử lý bằng cách thêm các container.

Ví dụ: Youtube,...

<u>Câu 6:</u> Làm cách nào có thể lưu thông tin được tạo trong Docker về máy local, không sử dụng copy?

- Bước 1: Tạo 1 thư mục để chia sẻ chung trên máy local và container
- Bước 2: Chia sẻ thư mục đã tạo với 1 container
    - Sử dụng thêm tùy chọn **-v** <local_path>**:**<container_path> trong lệnh run container.
    - Lúc này dữ liệu ở 2 thư mục đã được map với nhau nếu container lưu dữ liệu vào thư mục này thì cũng lưu trên host, khi xóa container thì dữ liệu vẫn được lưu trên máy host.

<u>Câu 7:</u> Nhóm em vẫn chưa hiểu phần triển khai nâng cao tại sao vẫn chạy ở môi trường local?

Triển khai ở môi trường local là một cách minh chứng rằng Docker có thể container hóa ứng dụng, push lên Docker Hub, và pull về để chạy trên bất kỳ môi trường nào, bao gồm local, server, hoặc cloud. Triển khai ở local là bước đầu tiên giúp kiểm tra ứng dụng, tiết kiệm chi phí và đơn giản hóa quy trình trong demo. Khi cần, chúng ta có thể triển khai image tương tự lên server hoặc môi trường production mà không cần thay đổi gì cả. Điều này minh chứng tính portability và linh hoạt của Docker trong mọi môi trường.