

# Docker

- <https://github.com/adrianhajdin/docker-course>
- <https://docs.docker.com/desktop/release-notes/>

Docker is an open-source platform that automates the deployment, scaling, and management of applications by using containerization. Here's a breakdown of what Docker is and how it works:

## 1. Containerization

- **Containers:** Docker containers are lightweight, standalone, and executable software packages that include everything needed to run a piece of software: the code, runtime, system tools, libraries, and settings.
- **Isolation:** Containers provide isolated environments, which means they can run consistently across different environments (e.g., your local machine, a cloud server) without being affected by the underlying system. This makes Docker great for avoiding the "it works on my machine" problem.

## 2. Images

- Docker containers are built from **images**, which are snapshots of the application and its dependencies at a specific point in time.
- **Dockerfile:** A Dockerfile is a script that contains instructions on how to build a Docker image. It specifies what software and dependencies to install and configure.

## 3. Portability and Efficiency

- **Portability:** Docker containers can run on any system that supports Docker (Linux, Windows, macOS), making them portable across different environments.
- **Efficiency:** Unlike virtual machines (VMs), Docker containers share the host system's kernel, making them faster to start and more lightweight (they don't include a full OS like VMs do).

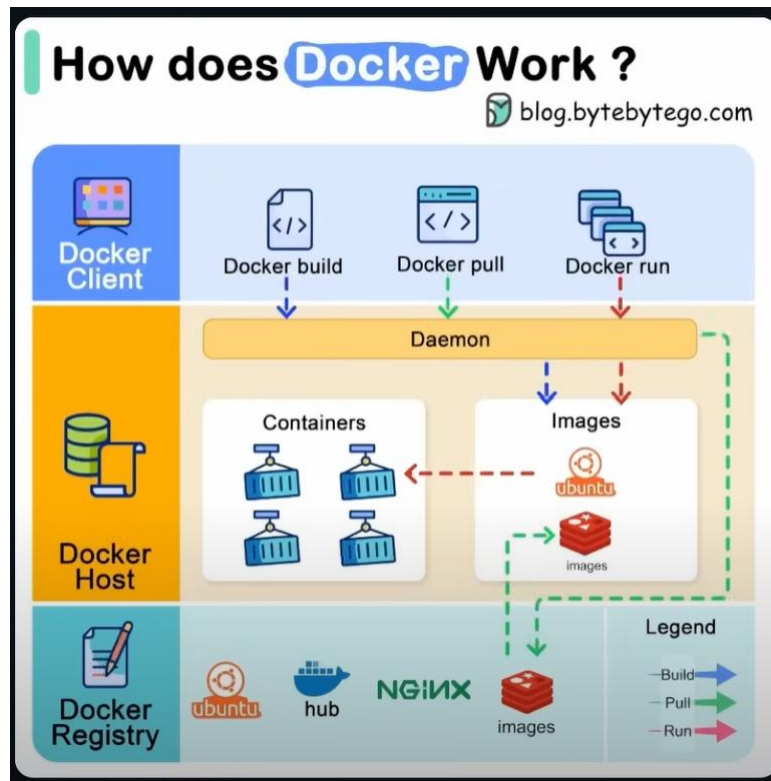
## 4. Key Components of Docker

- **Docker Engine:** The core part of Docker that runs and manages containers.
- **Docker Hub:** A public registry where Docker users can find and share container images. You can pull existing images or upload your own.
- **Docker Compose:** A tool to define and run multi-container Docker applications. It allows you to manage multiple containers as part of a single application.

## 5. Use Cases

- **Development and Testing:** Developers use Docker to create consistent environments across different stages of development, from testing to production.
- **Microservices Architecture:** Docker is widely used for building microservices, where each service runs in its own container.
- **Continuous Integration/Continuous Deployment (CI/CD):** Docker plays a significant role in automating the testing and deployment of applications.

In short, Docker simplifies the process of deploying and managing applications by using container technology, which is portable, fast, and resource efficient.



## Docker Image vs Container vs Volume

In Docker, the terms **Image**, **Container**, and **Volume** refer to different components of the containerization system. Each plays a specific role in how Docker packages, runs, and stores application data. Here's a detailed comparison:

### 1. Docker Image

- **Definition:** A Docker Image is a read-only template that contains the instructions for creating a Docker container. It includes the application code, runtime, libraries, environment variables, and any other dependencies required for the app to run.
- **Immutable:** Once built, an image cannot be modified. Instead, you can create a new image by layering changes over an existing image.
- **Layers:** Docker images are composed of multiple layers, each layer representing a change or instruction (e.g., adding files, installing dependencies). These layers are cached, which helps speed up the building process.
- **Building:** You typically build an image using a **Dockerfile**, which contains instructions to create the environment needed for the application.
- **Usage:** You use an image to instantiate containers. An image can be reused to create multiple containers.
- **Example:**
  - You might pull an image from Docker Hub, such as nginx, and use it to create a running web server container.

### 2. Docker Container

- **Definition:** A container is a running instance of a Docker image. It includes the application and all its dependencies, but also the runtime state, like memory, processes, and file system changes.

- **Ephemeral:** Containers are designed to be short-lived. Once a container stops, it can be restarted or removed, but its changes (outside of volumes) are lost unless committed to a new image.
  - **Isolation:** Containers provide process and filesystem isolation. This means that different containers running on the same host do not interfere with each other or the host system.
  - **Mutable:** Although the image is immutable, a container can change its internal state while it runs (e.g., writing files or logs). However, these changes are not reflected in the original image.
  - **Lifecycle:** You can create, run, stop, pause, and remove containers. Each container is tied to an image but represents a distinct instance with its own running environment.
  - **Example:**
    - If you run an image like nginx, you will create a container that starts an Nginx web server. If you stop or remove the container, the underlying image remains unchanged.
- 

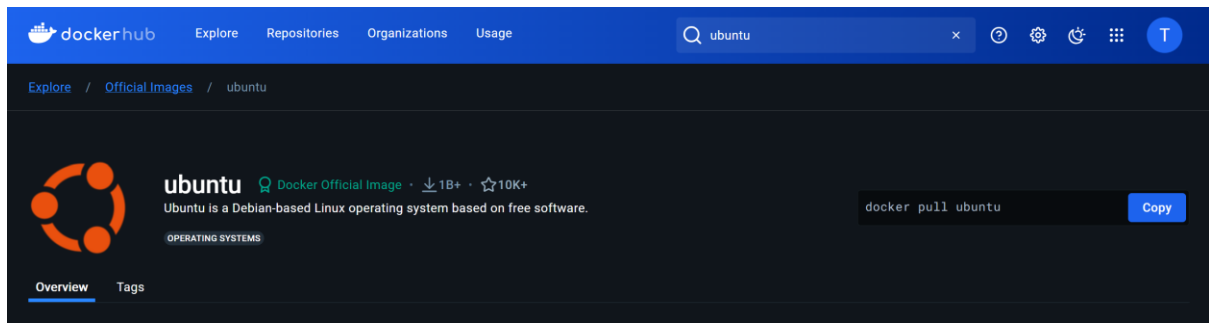
### 3. Docker Volume

- **Definition:** A volume is a persistent storage solution in Docker. It allows you to store data outside the container's filesystem so that data remains even after the container is stopped or removed.
- **Persistence:** While containers themselves are ephemeral, volumes provide a way to store data persistently. When a container writes to a volume, the data persists on the host machine or a remote storage system, even if the container is deleted.
- **Decoupling Data from Containers:** Volumes are used to decouple application data from the container itself. This is especially useful when you need to persist databases, log files, or user-generated content.
- **Types:**
  - **Anonymous Volumes:** Created by Docker, tied to a specific container, but not named.
  - **Named Volumes:** Explicitly named by the user, making it easier to manage and reuse across containers.
  - **Bind Mounts:** Allow containers to access specific directories on the host's filesystem.
- **Sharing:** Volumes can be shared across multiple containers. For example, if you're running a web app and a database in separate containers, both containers can use the same volume to read and write data.
- **Example:**
  - When you run a MySQL container, you would use a volume to store the database data. This way, if you stop or remove the MySQL container, the data in the volume remains intact.

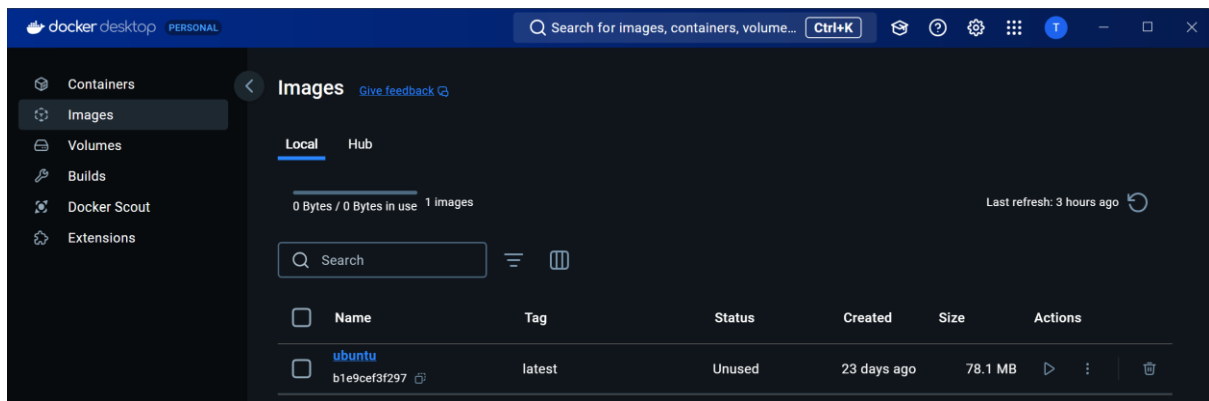
## Tutorial

### Try existing Docker Image from Docker Hub

1. Look for ubuntu in <https://hub.docker.com/>



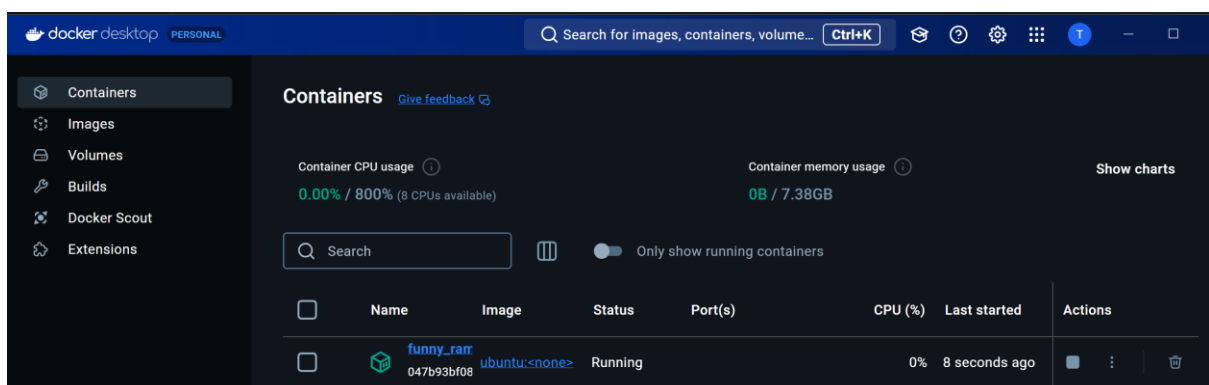
## 2. Run docker pull ubuntu in the terminal (VS Code)



## 3. Run docker run -it ubuntu

### What Happens When You Run docker run -it ubuntu?

- Image Retrieval:** Docker checks if the ubuntu image is available locally. If not, it pulls the latest version from Docker Hub.
- Container Creation:** Docker creates a new container based on the Ubuntu image.
- Interactive Terminal:** The `-it` flag provides you with an interactive terminal (a shell) inside the newly created Ubuntu container. You are now "inside" the container and can execute shell commands, just as if you were using a regular Ubuntu machine.
- Run Commands:** Once inside the container, you can run typical Linux commands, install software, or make modifications.



## 4. Play around with the ubuntu machine (from the image)

```

root@28c4fc4463f4:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@28c4fc4463f4:/# cd home/
root@28c4fc4463f4:/home# ls
ubuntu
root@28c4fc4463f4:/home# mkdir hello
root@28c4fc4463f4:/home# ls
hello ubuntu
root@28c4fc4463f4:/home# cd hello
root@28c4fc4463f4:/home/hello# touch hello-ubuntu.txt
root@28c4fc4463f4:/home/hello# ls
hello-ubuntu.txt
root@28c4fc4463f4:/home/hello# exit
exit
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning>

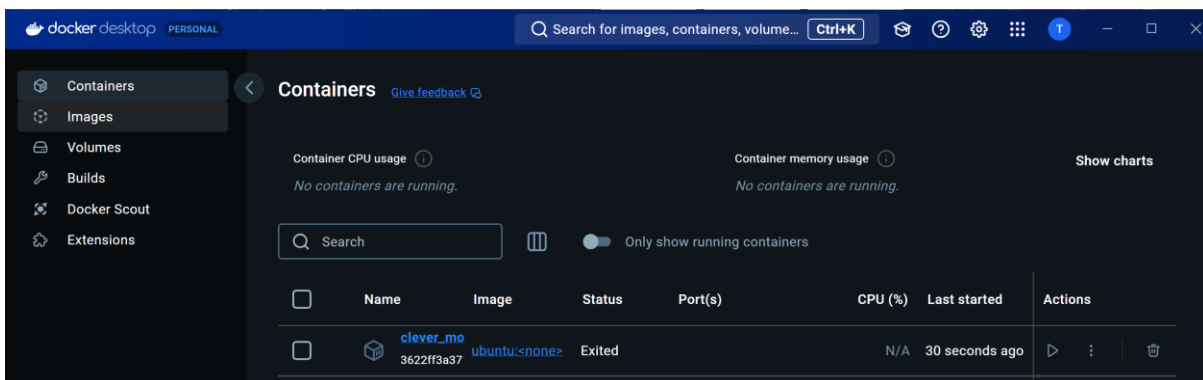
```

## 5. Use exit command or trash icon to exit the image

```

PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu latest b1e9cef3f297 3 weeks ago 78.1MB
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning> docker run -it ubuntu
root@3622ff3a3719:/# exit
exit
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning>

```



## Create First Image (Hello Docker Demo) – Just one JS Script

1. Create hello-docker folder
2. Create hello.js file
3. Create Dockerfile
4. (Terminal) docker build -t hello-docker .

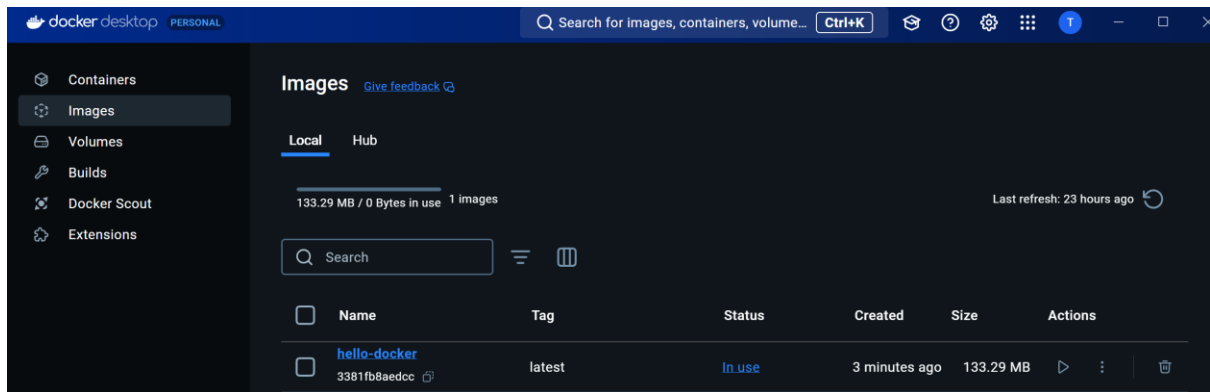
### Key Points to Remember:

- **docker build:** Builds a new Docker image.
- **-t hello-docker:** Tags the image with the name hello-docker, making it easy to reference later.
- **.(dot):** The build context, indicating that Docker should look in the current directory for the Dockerfile and other files.

```

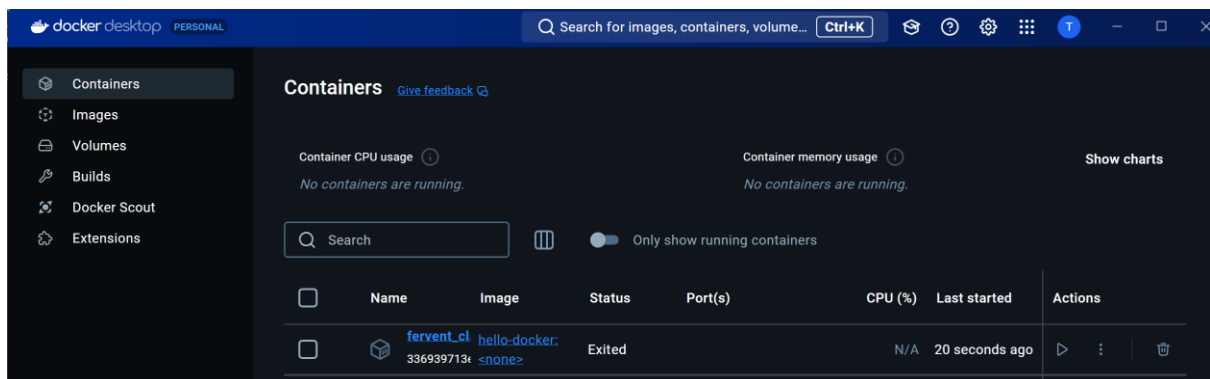
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning\docker-course-testing\hello-docker> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hello-docker latest 3381fb8aedcc 31 seconds ago 133MB
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning\docker-course-testing\hello-docker>

```



## 5. Run the image

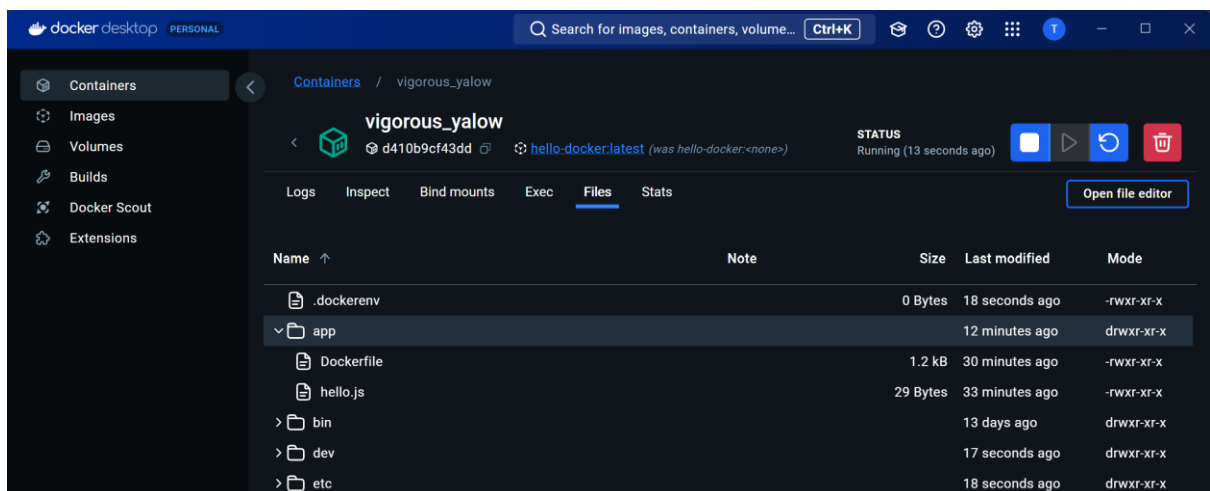
```
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning\docker-course-testing\hello-docker> docker run hello-docker
Hello Docker!
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning\docker-course-testing\hello-docker> 
```



## 6. Run the operating system of an image

### a. (Terminal) docker run -it hello-docker sh

```
PS C:\Users\Vince\Downloads\PersonalTraining\docker-learning\docker-course-testing\hello-docker> docker run -it hello-docker sh
/app # node hello.js
Hello Docker!
/app # 
```

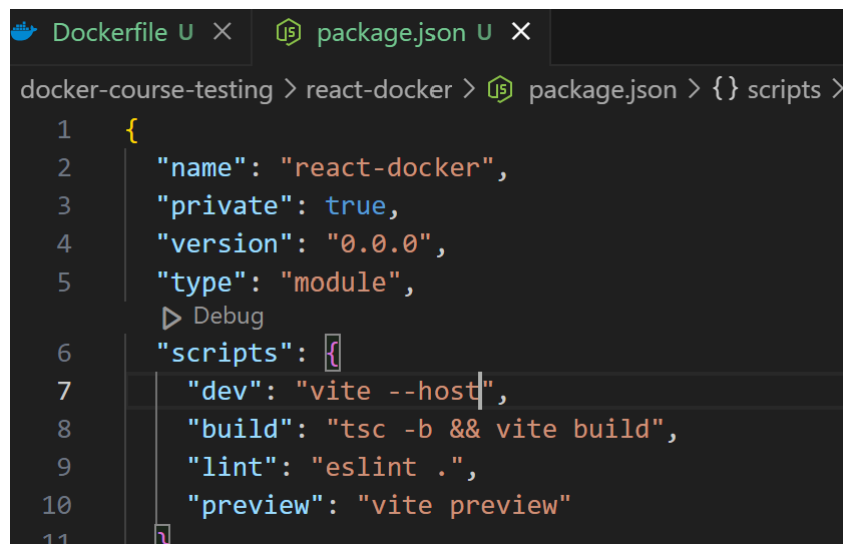


### Key Points:

1. **Run an Interactive Container:** The -it flags allow you to interact with the container via a shell in real time.
2. **Using the hello-docker Image:** The container is based on the hello-docker image, which you created earlier.
3. **Starting a Shell (sh):** The sh command opens a simple shell environment inside the container, allowing you to execute commands interactively.

### Create React Image (Without compose.yaml)

1. Create a React project (Terminal) `npm create vite@latest react-docker`
2. React -> Typescript
3. Create a Dockerfile
4. Create a .dockerignore
5. Add --host to the package.json

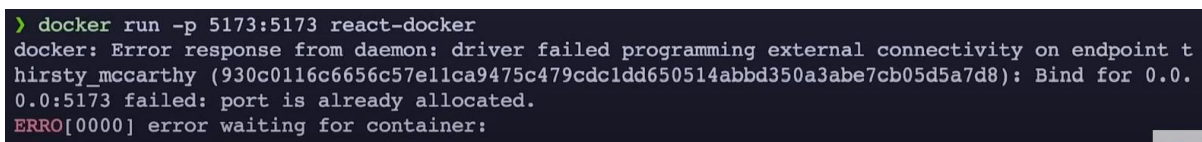


```
Dockerfile U x package.json U x
docker-course-testing > react-docker > package.json > {} scripts >
1  {
2    "name": "react-docker",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite --host",
8      "build": "tsc -b && vite build",
9      "lint": "eslint .",
10     "preview": "vite preview"
11   }
```

6. (Terminal) `docker build -t react-docker .`
7. (Terminal) `docker run -p 5173:5173 react-docker`

### Some possible errors:

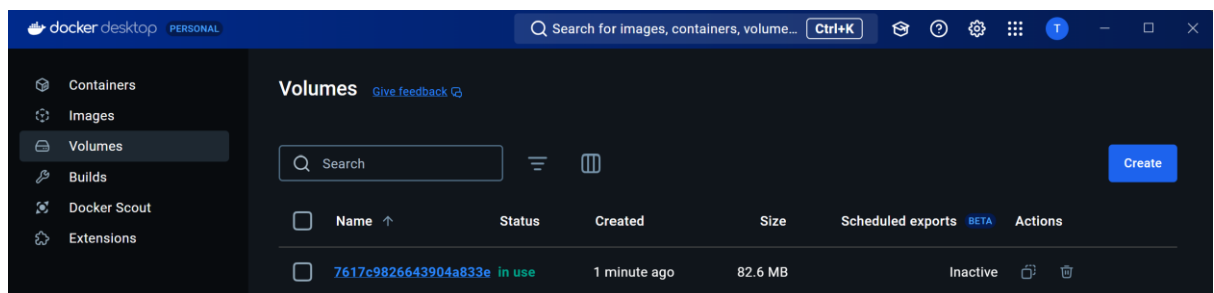
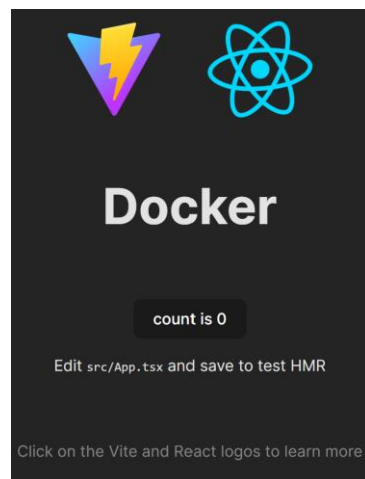
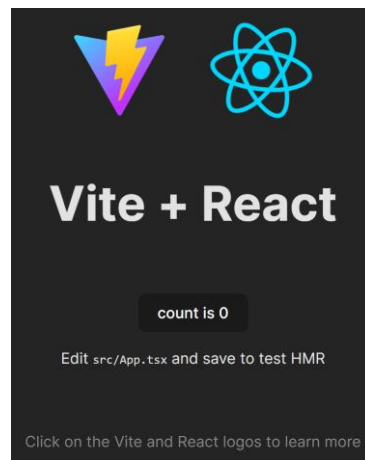
- a. Port 5173 is already allocated.



```
> docker run -p 5173:5173 react-docker
docker: Error response from daemon: driver failed programming external connectivity on endpoint t
hirsty_mccarthy (930c0116c6656c57e11ca9475c479cdc1dd650514abbd350a3abe7cb05d5a7d8): Bind for 0.0.
0.0:5173 failed: port is already allocated.
ERRO[0000] error waiting for container:
```

- (Terminal) `docker ps` [Only active/running containers] `docker ps -a` [All containers]
- (Terminal) `docker stop first3letters` [To stop 1 running container by id]
- (Terminal) `docker container prune` [To remove all stopped containers]
- (Terminal) `docker rm first3letters --force` [To remove 1 running container by id] `docker rm first3letters` [To remove 1 stopped container by id]
- (Terminal) `docker rmi first3letters` [To remove 1 image by id]

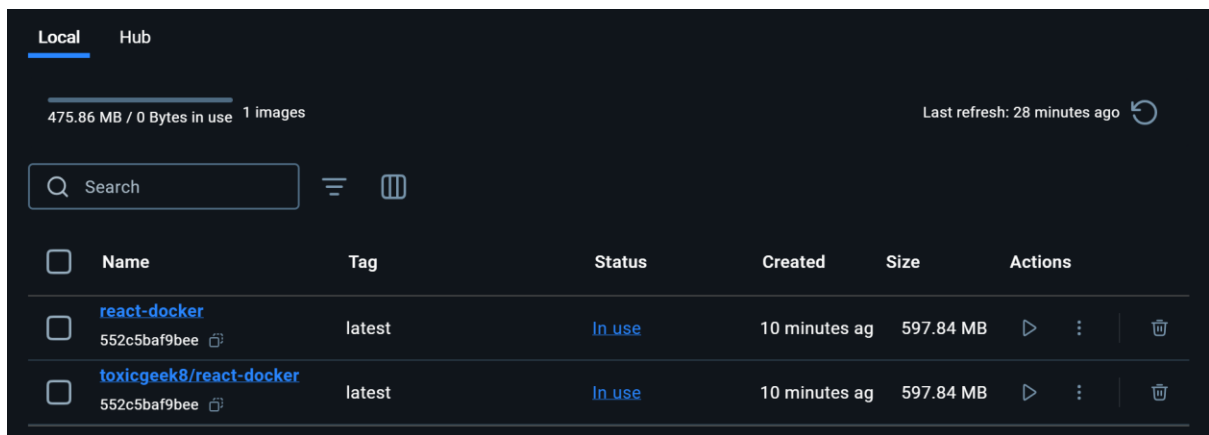
8. To make docker responsive after editing the files and CTRL + S
  - a. (Terminal) `docker run -p 5173:5173 -v "$(pwd):/app" -v /app/node_modules react-docker`



## Publish Docker Image

1. (Terminal) `docker login`
2. (Terminal) `docker tag react-docker toxicgeek8/react-docker`





The command `docker tag react-docker toxicgeek8/react-docker` is used to create a new tag for an existing Docker image. This tag allows you to name the image differently or prepare it for uploading to a Docker registry (such as Docker Hub). Here's a breakdown of each part of the command:

### 1. `docker tag`

- The `docker tag` command is used to add a new tag to an existing Docker image.
- A tag in Docker is essentially a label or name you give to an image, which helps in identifying and managing images.
- You can assign multiple tags to the same image, which makes it easier to version and share them.

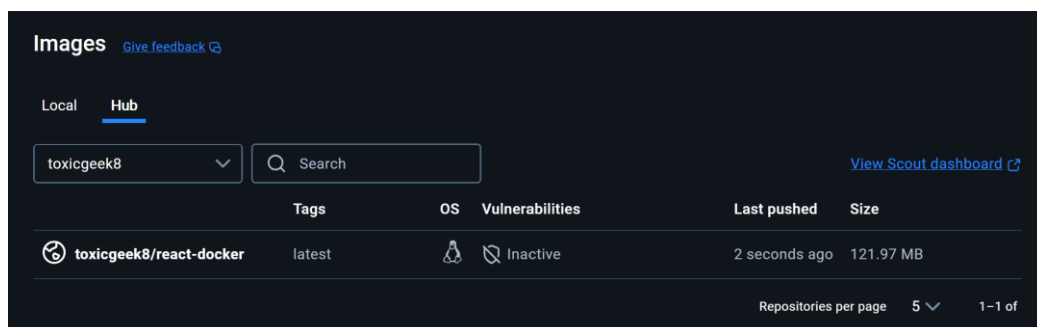
### 2. `react-docker`

- This is the **source image**. It refers to the name or ID of the existing image on your local machine.
- In this case, `react-docker` is the local image that you have built previously (likely using a `docker build` command).
- The image can be referred to by its name (as shown here) or by its image ID.

### 3. `toxicgeek8/react-docker`

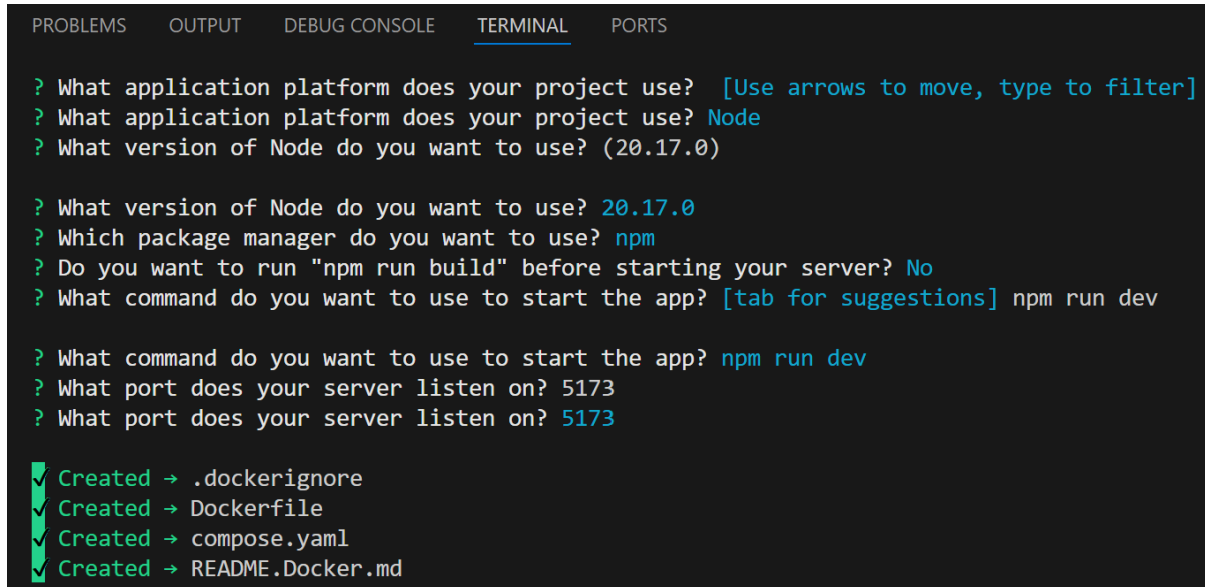
- This is the **target name** (including the repository) that you want to assign to the image.
- It consists of two parts:
  - **toxicgeek8**: This is the **Docker Hub username** or namespace. In this case, it looks like your Docker Hub username is `toxicgeek8`. When you push an image to Docker Hub, you need to prefix the image name with your username so it can be stored in your repository.
  - **react-docker**: This is the name of the image within the `toxicgeek8` repository.
- By tagging the image this way, you prepare it to be pushed to Docker Hub under the `toxicgeek8/react-docker` repository.

### 3. (Terminal) `docker push toxicgeek8/react-docker`



## Create React Image (With compose.yaml) + Using docker init

1. (Terminal) npm create vite@latest vite-project
2. React -> Typescript
3. (Terminal) docker init



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

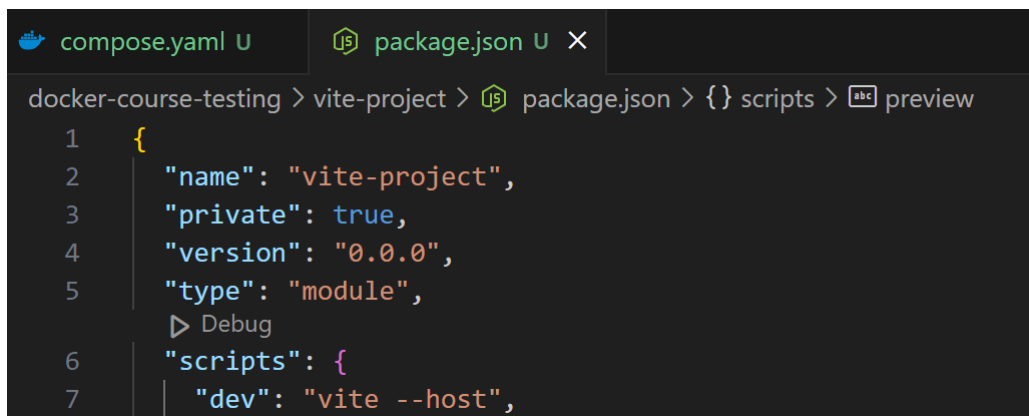
? What application platform does your project use? [Use arrows to move, type to filter]
? What application platform does your project use? Node
? What version of Node do you want to use? (20.17.0)

? What version of Node do you want to use? 20.17.0
? Which package manager do you want to use? npm
? Do you want to run "npm run build" before starting your server? No
? What command do you want to use to start the app? [tab for suggestions] npm run dev

? What command do you want to use to start the app? npm run dev
? What port does your server listen on? 5173
? What port does your server listen on? 5173

✓ Created → .dockerignore
✓ Created → Dockerfile
✓ Created → compose.yaml
✓ Created → README.Docker.md
```

4. Modify Dockerfile (Use the one in the react-docker)
5. Modify compose.yaml
6. Modify package.json (Add --host)



```
compose.yaml U  package.json U X
docker-course-testing > vite-project > package.json > {} scripts > preview
1  {
2    "name": "vite-project",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite --host",
```

7. (Terminal) docker compose up (If return error, open VSCode as an administrator). Must be inside the project directory.

## Create MERN Image (With Docker Compose Watch)

Docker Compose Watch – Listen to any changes

- Rebuilding the app
  - Re-running the container
1. Create Dockerfile for both frontend and backend.
    - (For backend) use port 8000 and npm start

2. Create .dockerignore for both frontend and backend
3. Create compose.yaml
4. Modify package.json (Add --host)
5. (Terminal) docker compose up (If return error, open VSCode as an administrator). Must be inside the project directory.
6. (in new Terminal) docker compose watch -> To enable responsive behaviour when the codes are modified.

### Create Next.js Image (With Docker Compose Watch)

1. (Terminal) docker init

```
? What application platform does your project use? [Use arrows to move, type to filter]
? What application platform does your project use? Node
? What version of Node do you want to use? (20.17.0)

? What version of Node do you want to use? 20.17.0
? Which package manager do you want to use? npm
? Do you want to run "npm run build" before starting your server? No
? What command do you want to use to start the app? [tab for suggestions] (npm start) npm run dev

? What command do you want to use to start the app? npm run dev
? What port does your server listen on? 3000
? What port does your server listen on? 3000

✓ Created → .dockerignore
✓ Created → Dockerfile
✓ Created → compose.yaml
✓ Created → README.Docker.md
```

2. Modify Dockerfile and compose.yaml
3. (Terminal) docker compose up (If return error, open VSCode as an administrator). Must be inside the project directory.
4. (in new Terminal) docker compose watch -> To enable responsive behaviour when the codes are modified.