Docker Tutorial

What is container :

A process running on host that isolated from host process

What is image :

When we need to run container we need to use an image. Image contain everything needed to run an application - all dependencies, configurations, scripts, binaries. So that means it portable and can be run on any where and any OS.

Docker installation

- Mac
  - Enter this site : https://docs.docker.com/desktop/install/mac-install/
  - Then following the description.
- Ubuntu (full docs on this site : https://docs.docker.com/engine/install/ubuntu/)
  - Set up Docker's `apt` repository.
  - install docker package

    ```
    sudo apt-get install \
    docker-ce docker-ce-cli \
    containerd.io \
    docker-buildx-plugin \
    docker-compose-plugin \
    ```

  - Add a user group docker to your user via

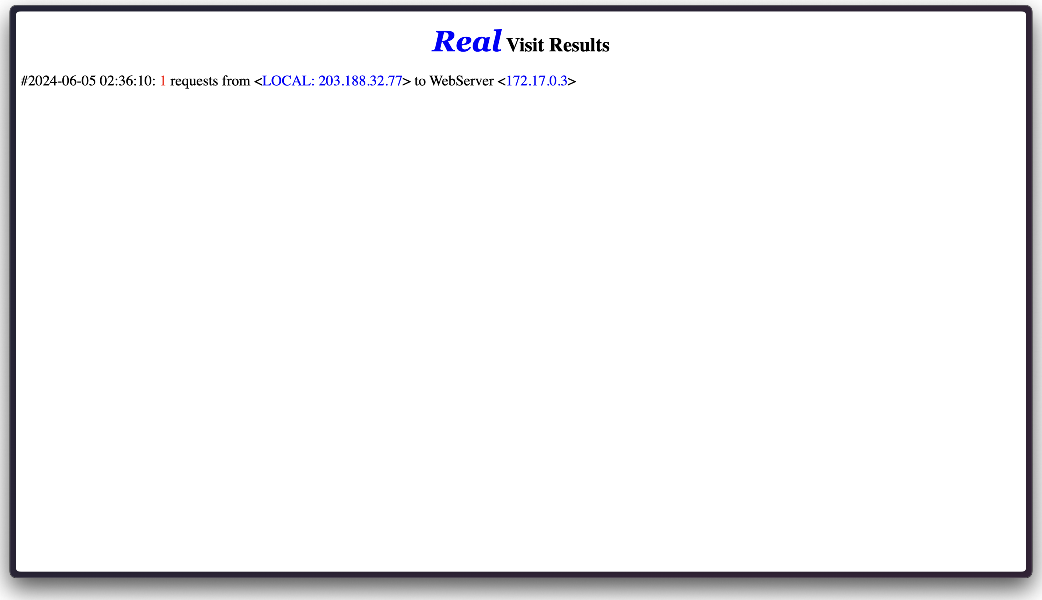
    ```
    sudo usermod -aG docker $USER

    -a mean append user group from exist usergroup
    -G mean group
    ```

```
-aG mean append new group from existing group
```

- Now we are getting and initial setup docker on our device already, next we will try to run some container to show simple page.

  ```
  docker run -dp 80:80 yeasy/simple-web:latest
  ```

  If you are work perfectly you will see the result like this via enter IP: 127.0.0.1 or localhost



Real **Visit Results**

```
#2024-06-05 02:36:10: 1 requests from <LOCAL: 203.188.32.77> to WebServer <172.17.0.3>
```

Create docker image

when you implement your application you need to build an docker image for that project to be run it on container, in this chapter we will show some simple way to create docker image via dockerFile and try to run.

1. Firstly, I will implement some simple nodes application to be an example app for building image.
   a. Create empty folder to store our source code.
   b. execute this command

   ```
   npm init -y #init node project
   ```

```
npm i express # install express library
```

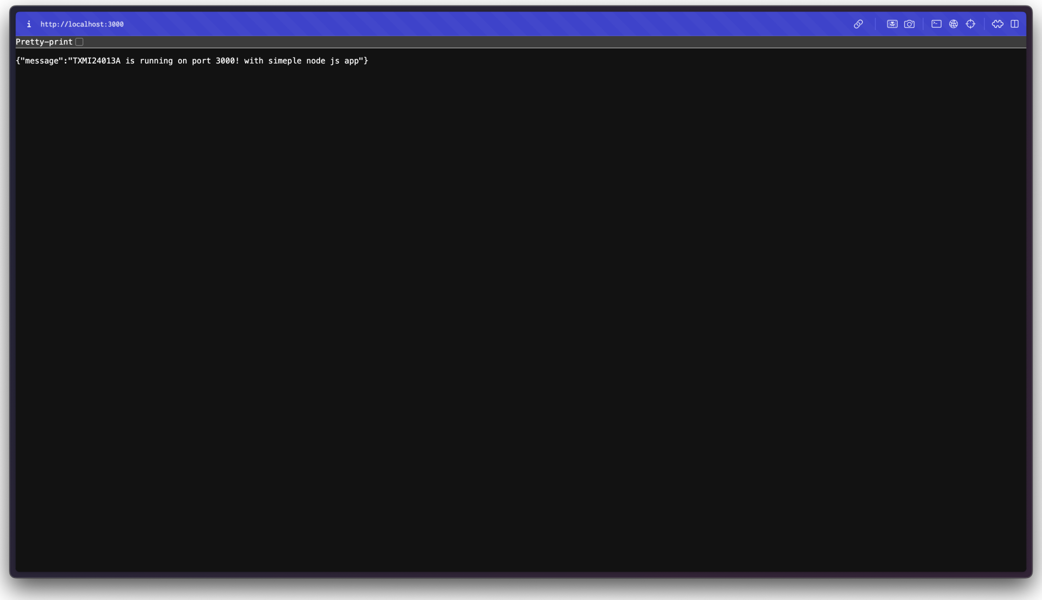
c. go to package.json then add `“type” : “module”` into your package.json file. the result should be like this.

```
{
  "name": "simple-node-app",
  "version": "1.0.0",
  "main": "index.js", //added line
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.19.2"
  }
}
```

d. create file index.js then write some simple code like this.

```
import express from "express";
import os from "os";

const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.send({
    message: `${os.hostname()} is running on port ${por
    simeple node js app`,
  });
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

e. try to run this app via command `node index.js` , then enter website with localhost:3000 the result should be



```
{"message":"TXMI24013A is running on port 3000! with simeple node js app"}
```

f. Next, we will write DockerFile to create a docker image for this simple app.

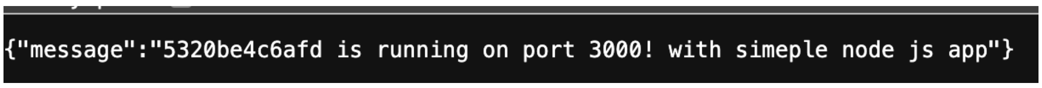
   i. Create DockerFile then write some simple script like this

```
// DockerFile

FROM node:lts-alpine
WORKDIR /usr/src/app
COPY ["package.json", "package-lock.json*", "./"]
RUN npm install && mv node_modules ../
COPY . .
EXPOSE 3000
RUN chown -R node /usr/src/app
USER node
CMD ["node", "index.js"]
```

   ii. try to build image via `docker build . -t simple-app`

iii. if everything work perfectly when you type `docker image` you need to see an simple-app image

iv. running an app via `docker run -dp 80:3000 simple-app` , then enter website localhost:80 you will able to see the same result as before running without docker but device name will be change to docker container id.



```
{"message":"5320be4c6afd is running on port 3000! with simeple node js app"}
```

Push image to registry

- Public registry

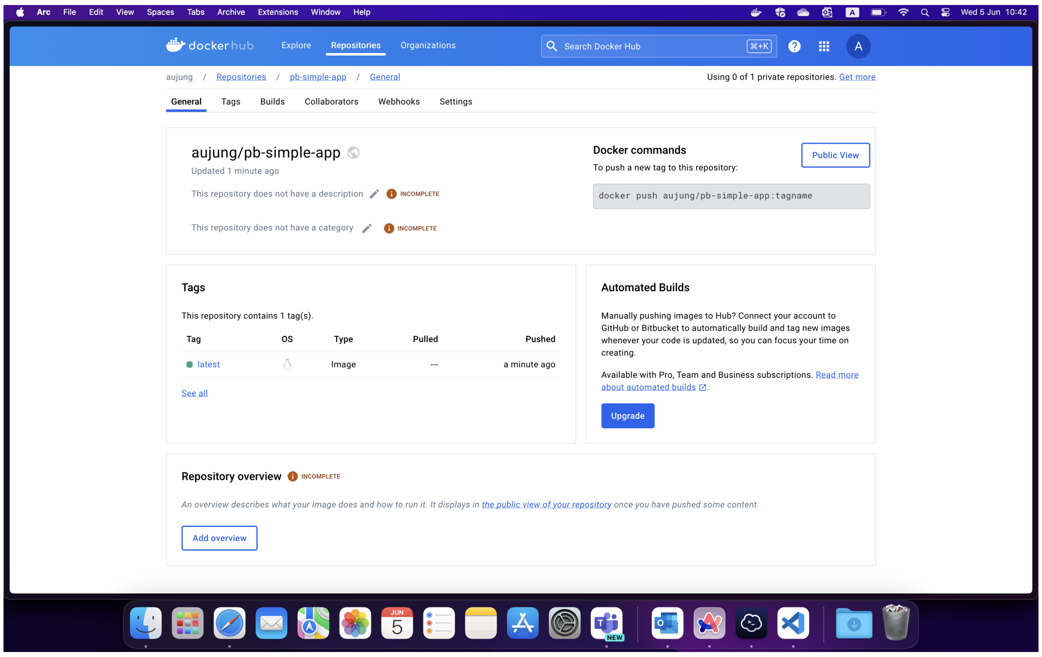
  Form the previous chapter we had created an docker image call `simple-app` in this part we will push it on DockerHub public registry

  before we push we need to create an repository on docker hub.

  - Firstly, we need to give an tag to simple-app before pushing to registry, then push it.

    ```
    docker tag simple-app YOUR-USER-NAME/REPO_NAME

    docker push YOUR-USER-NAME/REPO_NAME
    ```

  - If everything work fine you will see the result look like this



Using 0 of 1 private repositories. Get more

aujung/pb-simple-app

Updated 1 minute ago

This repository does not have a description — INCOMPLETE

This repository does not have a category — INCOMPLETE

Docker commands

To push a new tag to this repository:

```
docker push aujung/pb-simple-app:tagname
```

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a minute ago

See all

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Repository overview — INCOMPLETE

An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.

- Private registry

  In this part everything is the same as public repository but there are some several thing need to change in execution command.

  ```
  docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNA

  docker push NAME[:TAG]
  ```

Running container

There are several to run a docker container a way we use it already is run with `docker run` and another version is writing `docker-compose` file,So in this chapter we will create simple docker-compose file to run our simple-app

1. create `docker-compose.yml` then write some script like this.

   ```
   version: "3"
   services:
     simple-app:
       image: IMAGETAG //IMAGE TAG FROM previous section
       ports:
         - "80:3000"
   ```

2. run docker `compose up -d` to run container with simple-app image'

Container file system volumes and bind mounts

Each container also gets its own "scratch space" to create/update/remove files. Any changes won't be seen in another container, even if they're using the same image.

volume

Volumes provide the ability to connect specific filesystem paths of the container back to the host machine. If you mount a directory in the container, changes in that directory are also seen on the host machine. If you mount that same directory across container restarts, you'd see the same files.

Networking

Container networking refers to the ability for containers to connect to and communicate with each other, or to non-Docker workloads.