Install docker:

after installation:

Now test docker:

docker run hello-world

build a simple node app

Instructions in Dockerfile are layers

they can be cached by docker engine

so that build of an image should fast

Dockerfile

A Dockerfile is a simple text file that contains a list of commands that the Docker client calls while creating an image.

FROM : Node:16.3.1

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

CM [“node”, “server.js”]

docker build .

docker build -t <imagename> .

docker image rm <imagename>

docker images

docker image ls

docker —help

docker run —help

docker build —help

docker run <imageName>

docker run -i -t —name <containerName> <imageName>

docker run -it —name <containerName> <imageName>

docker run -d -p 3000:3001 —name <containerName> <imageName>

docker rm <containerName> -f

docker run -d -p 3000:3001 rm —name<containerName> <imageName>

docker ps

docker ps -a

docker system prune

docker container prune

docker rm <containerName>

docker stop <containerName or id>

docker start <containerName or id>

docker inspect <containerName or id>

enter to the container by using following command:

docker exec -it <containerName> bash

here we can see all files of our project: in Dockerfile we actually use copy command which copy all files and folder in container that we don't want to do so for that we can use .dockerignore file and mention which files and folder should not be copied inside container

now create a .dockerignore

add:

node\_modules

Dockerfile

.git

.gitignore

.dockerignore

Now images are read only if we change something inside our code we need to rebuild our image again and run our container agains

To overcome this in development process: how we can sink our project directory with running container:

there so many types of volumes available in docker like anonymous vol, named vol, but in this case we use bind mounts:

When you use a bind mount, a file or directory on the host machine is mounted into a container.

when every we create a container docker also create volume for that in following dir:

/var/lib/docker/volumes/hash/\_data

but these volumes are temporary

docker run -v pathtoLocalmachinefolder:pathtoContainerfolder -p 3000:3000 -d —name <containerName> <imageName>

in window:cmd

docker run -v %cd%:/app -p 3000:3000 -d —name <containerName> <imageName>

in window: powersell:

docker run -v ${pwd}:/app -p 3000:3000 -d —name <containerName> <imageName>

in mac and linux:

docker run -v $(pwd):/app -p 3000:3000 -d —name <containerName> <imageName>

now make changes in your code not changes will not be reflected

check code by this command:

docker exec -it <containerName> bash

you will see changes are there.

we need to restart node process

we can do this by using nodemon

so install nodemon in local env

make changes in start

“start” : “node index.js”

“dev”: “nodemon index.js”

in window if have issue in docker

“dev”: “nodemon L index.js”

now rebuild our image again

also make changes in Dockerfile

CMD [“npm”,”run”, “dev”]

docker logs <containerName>

docker run – Runs a command in a new container.

docker start – Starts one or more stopped containers

docker stop – Stops one or more running containers

docker build – Builds an image form a Docker file

docker pull – Pulls an image or a repository from a registry

docker push – Pushes an image or a repository to a registry

docker export – Exports a container’s filesystem as a tar archive

docker exec – Runs a command in a run-time container

docker search – Searches the Docker Hub for images

docker attach – Attaches to a running container

docker commit – Creates a new image from a container’s changes

Todays we will cover:

1. few more docker commands

2. how to use docker in development synk file in container

3. Know about Dockerfile

4. Know about .dockerignore

5. Add ENV in Dockerfile

6. About Volumes Add bind mount volume to our application

7. uses of —rm flag and -v flag with docker run command

8. check docker logs

9. Check files in docker container

10. make bind mount read only

11. Push image to docker hub

multi container app

12. Docker compose

Docker Commands

docker run -d -p 4000:3000 —name my-app —rm <imageId>

docker container prune

docker logs <containerId>

docker system —help

Logs

docker logs <containerId>

docker logs —tail 100 <containerId>

Stream Logs From a Specific Point in Time

docker logs --since 2019-03-02 <container ID>

docker push

docker pull

docker rename

Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote.

When an operator executes docker run, the container process that runs is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.

Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

.dockerignore file

Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

https://docs.docker.com/engine/reference/builder/

The beauty of bind mounts during development

During development, we want our application’s source code to update within the container whenever we change something. This is achieved by docker using bind mounts which allows our source-code to be accessed and modified by both the running container and the host system.

Simply mount your entire source code folder into a docker container and it will pick up the changes (bi-directionally!).

The issue with bind-mounting node\_modules

Bind-mounting the source code is amazing, but once we start dual developing, i.e. running our Node.js app both within the container and on the host system (maybe even just to run the tests), things get messy.

Preventing node\_modules to be overwritten by the bind mount

So how do we prevent the node\_modules from our host system to overwrite the node\_modules within the container?

bind mount command

docker run -v $(pwd):/app -d -p 3000:3000 --name myapp my-app-image

Anonymous volume:

docker run -v $(pwd):/app -v /app/node\_modules -d -p 3000:3000 --name myapp my-app-image

now

docker exec -it <containerId> bash

create a new file

touch test file

you will see a new file in your local folder that is not good

Why docker container changes our files:

Start using ENV in docker container

declare env in docker file

or create your own env file

—env PORT:4000

or —

—env-file ./.env

Docker Compose

Docker Compose is a tool that was developed to help define and share multi-container applications.

With docker desktop, Docker compose already installed.

check the version of docker-compose by typing:

docker-compose -v

Defining services in docker-compsoe.yaml file

version: "3.9"

services:

web:

build: .

ports:

- "8000:5000"

redis:

image: "redis:alpine"

Build and run your app with compose:

docker-compose up

//to rebuild again

docker-compose up —build

Edit compose file to add a build mount:

services:

node:

command: nodemon index.js

volumes:

- ./:/usr/local/app/

# the volume above prevents our host system's node\_modules to be mounted

- exclude:/usr/local/app/node\_modules/

volumes:

exclude:

Questions:

What's the result of these commands?

\* docker build -t myimage .

\* docker run --name mycontainer myimage

\* docker stop mycontainer

Docker push image

renaming image:

using docker tag

docker rename command

---------------------

Create Network

---------------------

docker network create goals-net

---------------------

Run MongoDB Container

---------------------

docker run --name mongodb \

-e MONGO\_INITDB\_ROOT\_USERNAME=max \

-e MONGO\_INITDB\_ROOT\_PASSWORD=secret \

-v data:/data/db \

--rm \

-d \

--network goals-net \

mongo

---------------------

Build Node API Image

---------------------

docker build -t goals-node .

---------------------

Run Node API Container

---------------------

docker run --name goals-backend \

-e MONGODB\_USERNAME=max \

-e MONGODB\_PASSWORD=secret \

-v logs:/app/logs \

-v /Users/maximilianschwarzmuller/development/teaching/udemy/docker-complete/backend:/app \

-v /app/node\_modules \

--rm \

-d \

--network goals-net \

-p 80:80 \

goals-node

---------------------

Build React SPA Image

---------------------

docker build -t goals-react .

---------------------

Run React SPA Container

---------------------

docker run --name goals-frontend \

-v /Users/maximilianschwarzmuller/development/teaching/udemy/docker-complete/frontend/src:/app/src \

--rm \

-d \

-p 3000:3000 \

-it \

goals-react

---------------------

Stop all Containers

---------------------

docker stop mongodb goals-backend goals-frontend

----- -- - - - - - - - -- --- - - - - - - - - - - -

Dockerfile

# 1. For build React app

FROM node:lts AS development

# Set working directory

WORKDIR /app

#

COPY package.json /app/package.json

COPY package-lock.json /app/package-lock.json

# Same as npm install

RUN npm ci

COPY . /app

ENV CI=true

ENV PORT=3000

CMD [ "npm", "start" ]

FROM development AS build

RUN npm run build

# 2. For Nginx setup

FROM nginx:alpine

# Copy config nginx

COPY --from=build /app/.nginx/nginx.conf /etc/nginx/conf.d/default.conf

WORKDIR /usr/share/nginx/html

# Remove default nginx static assets

RUN rm -rf ./\*

# Copy static assets from builder stage

COPY --from=build /app/build .

# Containers run nginx with global directives and daemon off

ENTRYPOINT ["nginx", "-g", "daemon off;"]

docker-compose.yml

version: "3.7"

services:

frontend:

build:

context: .

container\_name: frontend

ports:

- "80:80"

.dockerignore

node\_modules

npm-debug.log

build

.dockerignore

\*\*/.git

\*\*/.DS\_Store

\*\*/node\_modules

nginx.conf

server {

listen 80;

location / {

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri /index.html =404;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

}