Docker Notes

Docker provides its own virtual Operating system for an application in a containerised way.

Features

- Consistency across environment - any machine
- Reduces confusion
- Isolation - boundary and improves security
- Portability - between development, testing and production
- Version control - tracks versions and can rollback to previous version like git
- Scalability - can create multiple copies of the application
- DevOps integration

Components of Docker

Images :

Lightweight standalone executable packages like instructions/recipe to cook a food

It has all the code, runtime, libraries and the system tools.

Containers:

They are runnable instance/execution environment for images.

One can run multiple containers for a single image.

Volumes:

They are persistent data storage mechanisms outside containers

Help for data durability, intermediary between container and a host machine

Imagine it as a shared folder outside a container

Network:

They act as a communication channel for different containers or for a container and a host machine, help to maintain isolation

Think of it like a restaurant kitchen

Docker workflow

Docker client : sends all commands either via CLI or GUI

Docker host: They are docker daemon, background running processes, listens to commands from CLI/GUI

Docker registry: Docker hub, centralised repository to pull or push images

Commands:

1) Create a Dockerfile and specify the instructions first. The layout will be typically:

FROM
WORKDIR:
COPY
RUN
EXPOSE
ENV
ARG
VOLUME
CMD
ENTRYPOINT

2) To pull an image, for eg ubuntu,

docker pull ubuntu

3) To run iteratively,

docker run -it ubuntu

Let's create our own image.

4) Create a hello.js file and a Dockerfile [Alpine is a lightweight version of linux]

5) docker build -t hello-docker .

6) docker run hello-docker

7) npm create vite@latest react-docker

8) Let's create a new docker file.

FROM node:23 -alpine [Must always be the first instruction]

RUN addgroup app && adduser -S -G app app [add a system user to the group]

USER app

WORKDIR /app [set working directory to app]

COPY package*.json./

USER root [change ownership of files to the root user]

RUN chown -R app:app [changes the ownership for a given file]

USER app

RUN npm install

COPY . .

EXPOSE 5173

CMD npm run dev

This won't run in localhost as its isolated environment

Port Mapping:

Map port of docker container and that of the host machine

9) docker run -p 5173:5173

Modify the package.json also, in script -> dev -> vite —host

10) docker ps -a [shows all containers]

11) docker stop <container-id>

To get rid of all the inactive containers,

12) docker container prune

13) docker rm <id> —force

This time when we edit a file, it wont show any changes.

14) docker run -p 5173:5173 -v “$(pwd):/app” -v /app/node_modules react-docker

This mounts current working directory to the container directory , -v stands for volume

Publishing a docker image:

15) docker login

16) docker tag react-docker <username>/<image_name>

Note that tag always accept 2 arguments

17) docker push <username>/<image_name>

This will push it to the docker hub

Docker compose

It is a yml file to configure services, networks and volumes

18) docker init

The structure of compose.yml file will be:

```
Services:
  Web:
     Build:
        Context:
     Ports:
     Volumes:
```

19) sudo docker compose up [with root privileges]

Docker compose watch

This listens to changes.

It has 3 components :

1) sync: move changes files
2) Rebuild: creation of new container images
3) Sync -restart: merges sync and rebuild

We need to create build section in yml file

This helps us to build backend, frontend and api in single command

20) docker pull mongo:latest

21) sudo docker compose up

In mern-docker container, 3 images will be created for web, api and db will be created in the Docker GUI.

To enable automatic reloading,

22) sudo docker compose watch

For every save, update happens in real time.

You can test it by installing libraries like npm i colours

Docker scout:

It is proactive about security vulnerabilities, scans container images and creates a detailed list called SBOM - Software bill of materials, which checks the list and updates regularly.

It is used in docker desktop, hub and the CLI.

Vulnerabilities can be seen (CVEs) for an image which can be analysed