Docker

Each container is seperate
To run container from same image

docker run -it ubuntu
- Each container is an isolated

1. We have to create images with dependencies included

2. Adding Dockerfile

FROM
WORKDIR
COPY
ADD
RUN
ENV
EXPOSE
USER
CMD
ENTRYPOINT

To build image

```
docker -t react .
```

-t : tag
react : name
. : for location of Dockersfile

docker images
or
docker image ls

To run the image
→ docker run -it react
name or container id

This will run the container in shell but we want it in the bash mode so

```
# docker run -it react bash
```

Alpine doesn't come with bash.
↳ docker run -it react sh

Dockerfile

```
FROM node:(exact path)
WORKDIR /app
COPY . .
```

← To set default Working Directory

To copy files with spaces we use array ["A", "B"]
A = copy with space
B = To location

ADD file.zip . ← Automatically uncompressed

ADD https:// . ← Add from URL

These are the two options for which we can use add else we going to copy cmd

- In project files there is folder called node_modules which contains the dependesies which are in the extracted format when creating a image we would like to avoid using these files since these files makes image bigger + extra transfer time every time.

→ What we can do is exclude the folder and thoes dependencies will be still downloaded

To exclude a folder .dockerignore

↓

Benefits

① lower size

② faster transfer

after running the build we have to install the dependencies using npm install

npm install can be added to Dockerfile

RUN npm install

```
ENV API_URL = Http://app.com
ENV API_URL Http://app.com
```
← older syntax

To view this in container $ENV

- Exposing Ports

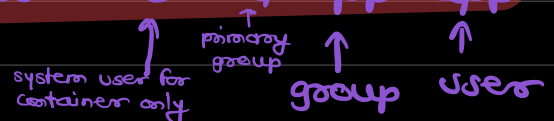
```
EXPOSE 3000
```

By default Host mapping for the ports is required

- Setting the user

```
addgroup app
adduser -S -G app app
```

addgroup app → setting group first



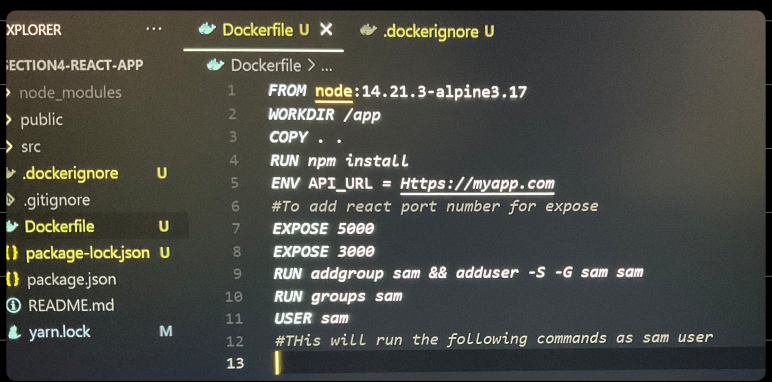
Adding user & then setting primary group as app

combined — addgroup app && adduser -S -G app app

for security we take this step so we can add

This to our Dockerfile and add extra security layer

- Here is screenshot after adding Everything



- Defining Entrypoints

- CMD to run the commands spefically last mentioned command in Docker file

ex. CMD npm start

Difference betn RUN vs CMD

RUN runs at the start of container

- CMD is for running at runtime

Best use syntax for CMD ["npm", "start"]

ENTRYPOINT

Similar to CMD But it cannot be easily overwritten using
```
docker run react-app
    echo hellow
```
This will overrite the CMD from Dockerfile.

CMD or ENTRYPOINT is based on preference

• Speeding the Builds

To improve the Build we can make use of cache

Dockerfile optimization is important for this

— It checks if layers are re-used or not if no changes then it will use the same layer.

* We have to seperate the copy of dependencies in the docker files to better optimize it
  → Because if we changed like a single line that will affect the docker-file and

the whole file has to be copied again. that slows down the build.

soln → Copying dependencies seperately

* Optimization →
  Dockerfiles which doesn't change frequently has to be on the top.
  — Because whenever there is change in docker file it will copy all the changed data and all files will also be transfered again

- Removing Image

`docker image prune` ← To remove temp images

- `docker container prune`
- `docker images` ← To see list of images

- Tagging images

- It is necessary to tag images properly for production build

`docker build -t react:main .`

↑

(:) used for tagging

Same image can have multiple tags
to remove the same with multiple tag

- `docker image remove react2.1:latest`

↓

tag name

• To change the tag of current build image

→ `docker image tag react2.2:latest`
`react2.2:updated`

• latest tag to images doesn't mean the latest tag in reality.

Pushing image to docker

- set the appropriate tag similar to your docker repo name

name /react-app

`docker image tag 511 name /react-app:1`

↓

Once done we can run docker push

```
docker push name/react-app:1
```

- Saving and loading images

- Saving

```
docker image save -o react.tar react-app
```

image name

Loading

- `docker image load -i react.tar`

- Section - 5
- Working with Containers

Running Container in detatched mode

- `docker run -d react2.1`

Will run the container in background

- docker running with name

- `docker run -d --name blue react2.1`

```
PS Z:\Subjects\DevOps\docker_files\Section 4- Images\section4-react-app\section4-react-app> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS      NAMES
596290d34e9d   aprazors/react-app:1   "docker-entrypoint.s…"   4 seconds ago   Up 3 seconds   3000/tcp   boring_maxwell
PS Z:\Subjects\DevOps\docker_files\Section 4- Images\section4-react-app\section4-react-app> docker run -d --name sky-high aprazors/react-app:1
a8f38d85b92f4cf7e4cd14cd7d32179f6406b6e01eb432a619cb88e720a462c7
PS Z:\Subjects\DevOps\docker_files\Section 4- Images\section4-react-app\section4-react-app> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS      NAMES
a8f38d85b92f   aprazors/react-app:1   "docker-entrypoint.s…"   4 seconds ago    Up 3 seconds    3000/tcp   sky-high
596290d34e9d   aprazors/react-app:1   "docker-entrypoint.s…"   47 seconds ago   Up 45 seconds   3000/tcp   boring_maxwell
PS Z:\Subjects\DevOps\docker_files\Section 4- Images\section4-react-app\section4-react-app>
```

- logs

`docker logs (container-id)`

`docker logs --help` for more

Publishing a port

EXPOSE 3000 ← let's container know that we are using port 3000 at container level. However, we need to enable this ports at Host level.

- `docker run -d -p 3000:3000 --name PS react2:1`
  - Host port can be set anything

- Executing commands in running container

- docker exec ← to run commands on running container

- docker run ← to start the container and run commands

- `docker exec 3f3 ls-l`

`docker run` ← running new container
`docker start` ← running stopped container

- Container file system
- Containers have ephemeral storage like AWS instance store.
- We have to use EBS like storage for persistancy in containers.

Volumes

- `docker volume create app-data`
- `docker volume inspect app-data`

- `docker run -d -p 3000:3000 -v app-data:/app/data react2:1`

- We added RUN mkdir data in docker file to create directory for mount point in the dockerfile for the volume.
  (It should be added after creating user & WORKDIR)

- Best way to create volume is to do with running container but make sure you have the entrypoint in dockerfile for

mounting the drive.

- `docker run -d -p 500:3000 -v app-data --name reactx2 react2.2`

- Copying files bet^n the host & containers

↓ Container → Host

docker cp `containerid:/app/log.txt` .

name : location

↓ Host to container cp

docker cp Hello.txt 00d:/app

- Sharing source code with container.

- for production always create new build with proper tag for minor changes

- for development we can create a mapping

- Removing docker images at once

- `docker rmi -f $(docker image ls -q)`

To remove with stopped container

- `docker image rmi -f $(docker image ls -aq)`

Docker Compose

- docker-compose.yml
  → To install all the dependencies of frontend, Backend and other in one file

1. `docker-compose up`

- creating compose file
  — is more like a configuration file

`docker compose --help`
— This commands works a whole impacts multiple services
↳ made necessary changes into compose

2. Docker compose build

To force full rebuild
- docker-compose build --no-cache

starting Applications

- docker-compose up --help
- docker-compose up --build
- docker-compose up -d
- docker compose ps

TO take it down
- docker compose down

- Docker Networking
  - Automatic networking bet^n each Application in that container.
-

- Viewing logs

docker compose logs
docker logs imageid -f
↑
To follows.

- Publishing changes
* Mapping a directory. in compose.

- Some dependencies related to particular Application might be missing thoes has to be installed.

Simplest way
`npm install` in that folder

We can add persistant storage into docker-compose.yml

```
api:-
  volumes:
    - ./backend:/app
```

So we mapped volume into backend folder so whenever there is change in backend that can automatically can

be reflected.

- Migrating the database

- To view the volumes
  - docker volume ls
  - ex. vidly_vidly ← vo
    - ↑ Application name

- docker-entrypoint.sh

Contains all the scripts or entrypoint command that should run

- Running Test

~ learn on your own.

Section - 7

- Deployment

- Deployment option.

  - Docker orchestration - kubernetes.