DOCKER
INDEX

INTRODUCTION

▷ WHAT IS DOCKER?

A platform for building, running and shipping applications

▷ REASONS WHY SAME APPLICATION MAY NOT WORK IN DIFFERENT ENVIRONMENT

- One or more files missing
- Software version mismatch
- Different configuration settings

▷ CONTAINER VS VM

CONTAINER	VIRTUAL MACHINE
An isolated environment for running an application	An abstraction of a machine (physical hardware)

▽ Problems with VM

PROBLEMS

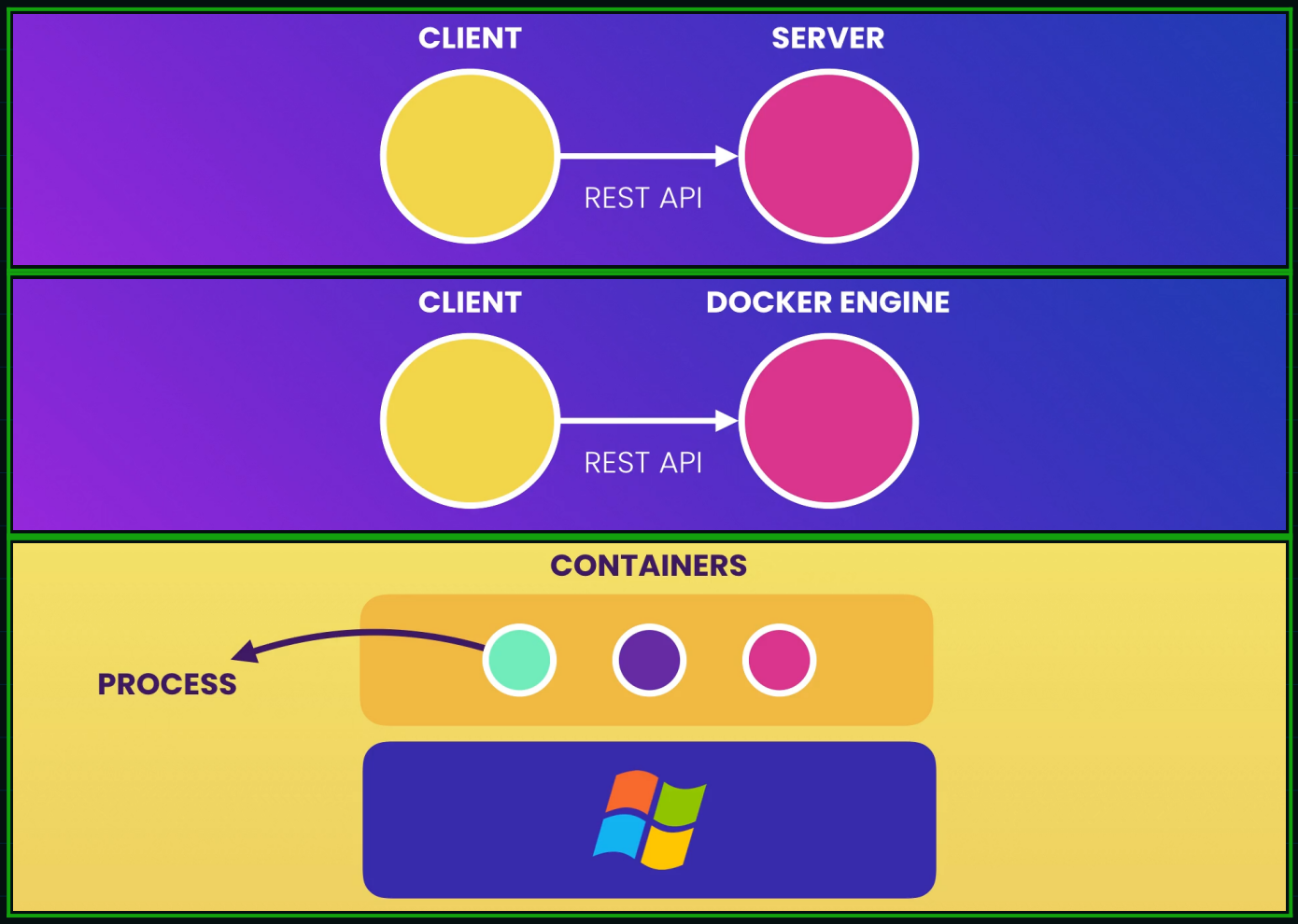
- Each VM needs a full-blown OS
- Slow to start
- Resource intensive

▽ Advantages of a container

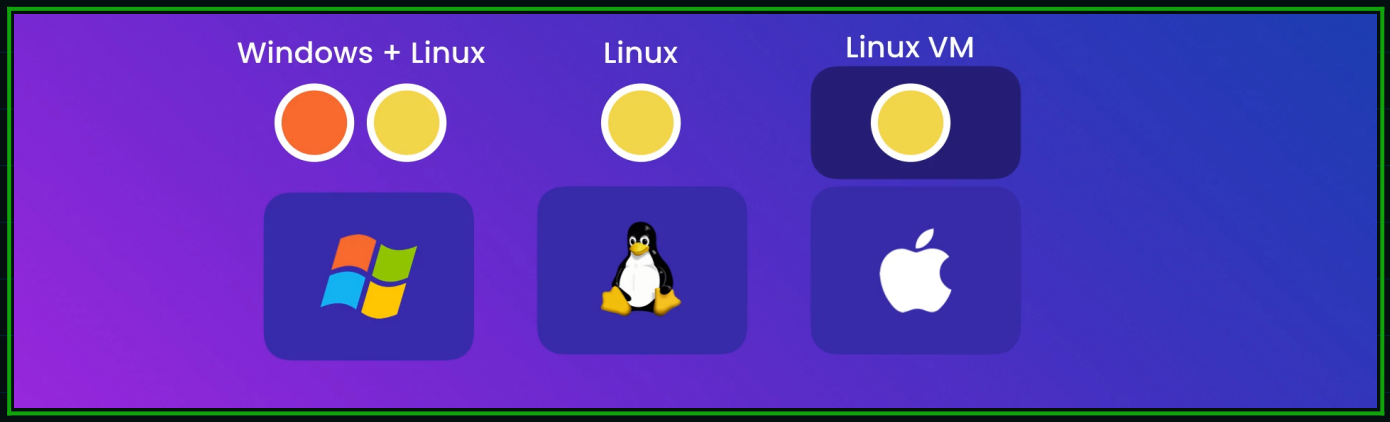
CONTAINERS

- Allow running multiple apps in isolation
- Are lightweight
- Use OS of the host
- Start quickly
- Need less hardware resources

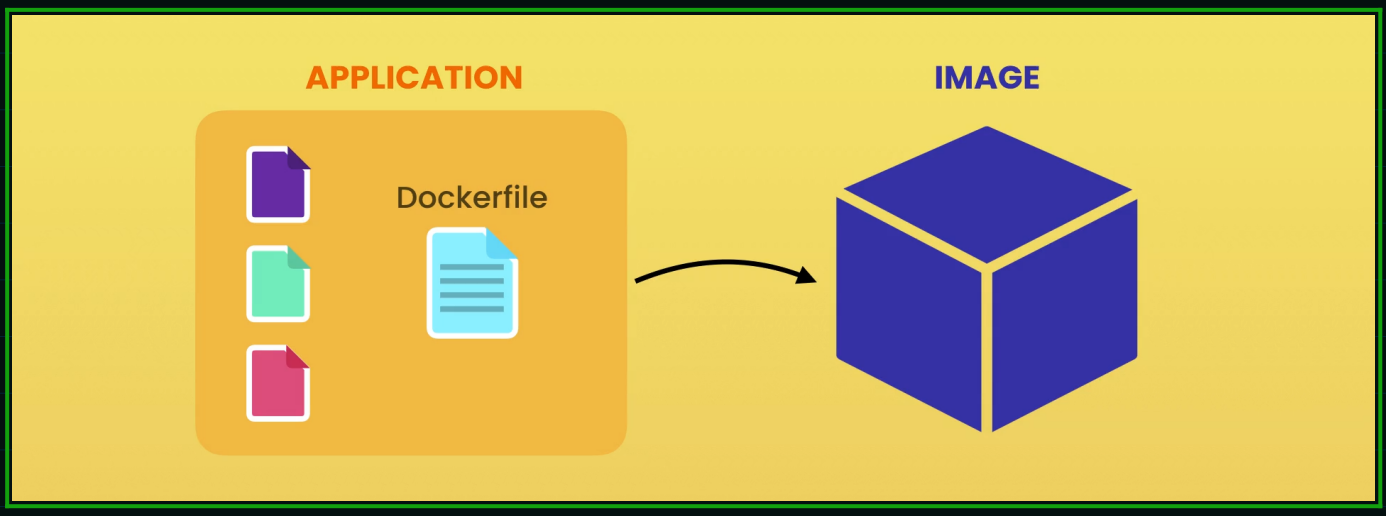
▷ ARCHITECTURE OF DOCKER



▷ TYPES OF CONTAINERS USED ON DIFFERENT OS



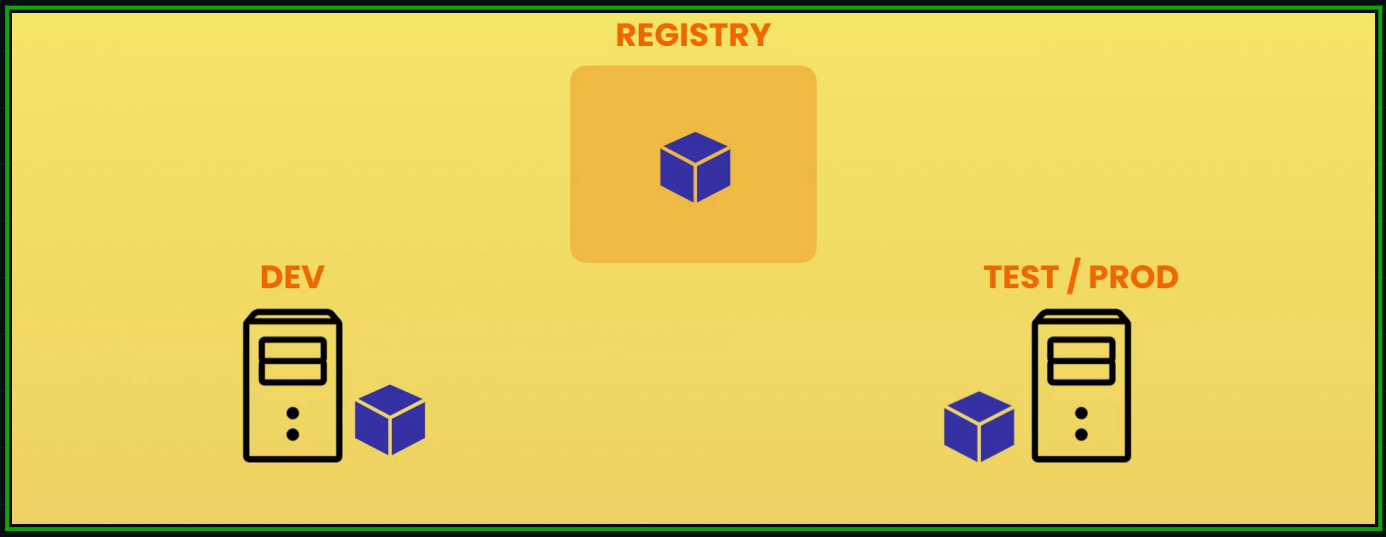
▷ DEVELOPMENT WORKFLOW



IMAGE

- A cut-down OS
- A runtime environment (eg Node)
- Application files
- Third-party libraries
- Environment variables

▷ SHARING OF DOCKER IMAGES



BUILDING DOCKER IMAGES

▷ IMAGES AND CONTAINERS

IMAGE

- A cut-down OS
- Third-party libraries
- Application files
- Environment variables

CONTAINER

- Provides an isolated environment
- Can be stopped & restarted
- Is just a process!

▷ DOCKERFILE

A Dockerfile contains instructions for building an image

DOCKERFILE

- FROM
- WORKDIR
- COPY
- ADD
- RUN
- ENV
- EXPOSE
- USER
- CMD
- ENTRYPOINT

▷ DOCKERIGNORE FILE

▽ Files to ignore

```
//.dockerignore file
node_modules
build
```

▷ DOCKERFILE

```
//Dockerfile
FROM node:18.12.1-alpine3.17
WORKDIR /app
# ADD source dest: can copy files from url and can copy files from a
compressed archive
COPY . .
ENV API_KEY="test"
EXPOSE 3000
RUN npm install && npm install -g serve && npm run build
RUN addgroup app && adduser -S -G app app
USER app
# CMD command argument: shell form, Spawns an extra process (shell)
# CMD [ "executable" ]: executable form
CMD [ "serve","-s","build" ]
# ENTRYPOINT [ "executable" ]
```

▷ OPTIMIZING BUILDS

▽ Building images uses the concept of layering; Image: Collection of layers

▽ Every instruction is a layer

▽ If for new build, Cached layers can be used

▽ If any layer is changed then docker has to rebuild all the layers appearing after the changed layer


```
FROM node:18.12.1-alpine3.17
WORKDIR /app
# ADD source dest: can copy files from url and can copy files from a compressed archive
COPY package*.json ./
RUN npm install && npm install -g serve
ENV API_KEY="test"
EXPOSE 3000
COPY . .
RUN npm run build
RUN addgroup app && adduser -S -G app app
USER app
# CMD command argument: shell form, Spawns an extra process (shell)
# CMD [ "executable" ]: executable form
CMD [ "serve","-s","build" ]
# ENTRYPOINT [ "executable" ]
```

▷ REMOVING IMAGES

▽ Dangling images

▸ The images with no name and no tag

```
~/Documents/AYSTIC/testing-docker  main ?2                    10.9s  18.12.1
❯ docker image ls
REPOSITORY      TAG                 IMAGE ID       CREATED          SIZE
react-app       latest              e076c89181b2   13 seconds ago   456MB
<none>          <none>              f13f5bbf7ba8   58 seconds ago   456MB
<none>          <none>              df0accee9756   20 minutes ago   455MB
nginx           latest              1403e55ab369   3 days ago       142MB
httpd           latest              73c10eb9266e   3 days ago       145MB
node            18.12.1-alpine3.17  6d7b7852bcd3   11 days ago      169MB
ubuntu          latest              6b7dfa7e8fdb   2 weeks ago      77.8MB
mysql           latest              7484689f290f   2 weeks ago      538MB
alpine          latest              49176f190c7e   4 weeks ago      7.05MB
centos          7                   eeb6ee3f44bd   15 months ago    204MB
ubuntu          14.04               13b66b487594   21 months ago    197MB
elasticsearch   2                   5e9d896dc62c   4 years ago      479MB
```

```
docker image prune
//to remove all the stopped containers
docker container prune
```

▷ TAGGING AN IMAGE

```
//during building
docker build -t name:tag .

//after building
docker image tag old-name new-name
```

▽ Latest tag may not point to the latest image. We have to explicitly tag the newer image as latest

▷ SHARING IMAGES

```
docker login
docker image tag <username>/<repo>:tag
docker push <username>/<repo>:tag
```

▷ SAVING AND LOADING IMAGES

▽ Transfer files without docker hub

```
docker image save --output <filename> <image-name>
docker image load --input <filename>
```

WORKING WITH CONTAINERS

```
//mapping ports
docker container run -p host:container <name>
//detaching
docker container run -d <name>
//running a container interactively
docker container run -it <name>
//starting a container interactively
docker container start -ia <name>
//executing a command in the running container
docker container exec <name> <command>
//executing a command in the running container in interactive mode
docker container exec -it <name> <command>
//passing env variables to container
docker container run -e <name>=<value> <image>
```

▷ PERSISTING DATA USING VOLUMES

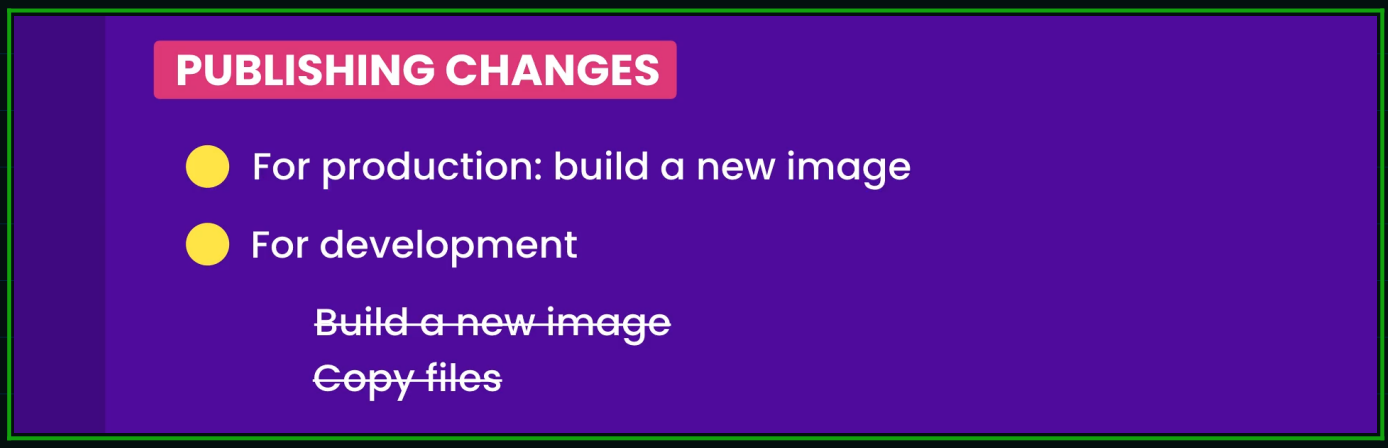
```
~/Documents/AYSTIC/testing-docker  main !1 ?3                    18.12.1
❯ docker volume create app-data
app-data

~/Documents/AYSTIC/testing-docker  main !1 ?3                    18.12.1
❯ docker volume ls
DRIVER    VOLUME NAME
local     app-data

~/Documents/AYSTIC/testing-docker  main !1 ?3                    18.12.1
❯ docker container run -it --volume app-data:/app/app-data alpine
/ # ls
app    bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # ls -R /app/
/app/:
app-data

/app/app-data:
/ #
```

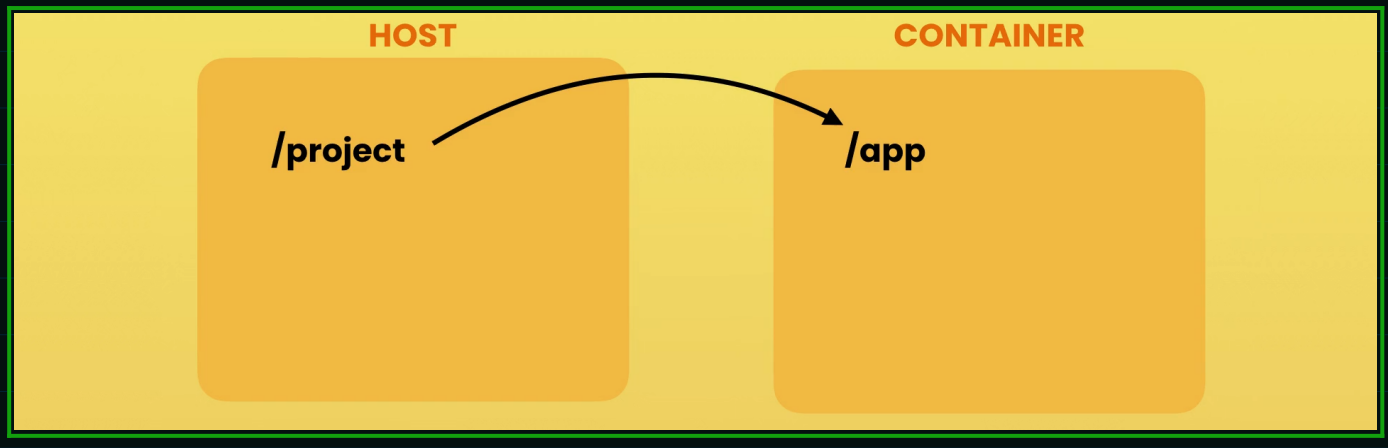
▷ make sure to include the mkdir command by the same user who will be using the volume as by default it will be created by root

▷ COPYING FILES BETWEEN CONTAINER AND HOST

```
docker cp <name>:<path-on-container> <path-on-host>
docker cp <path-on-host> <name>:<path-on-container>
```

▷ PUBLISHING CHANGES





▽ create a mapping from host to container

```
docker container run -v $(pwd):/app <name>
```

```
//Dockerfile.dev
FROM node:18.12.1-alpine3.16
WORKDIR /app
COPY package*.json ./
RUN npm install
EXPOSE 3000
CMD npm run start

//Building dev image
docker build -f ./Dockerfile.dev --rm --tag reactapp:dev1 .

//running container from dev image
docker container run -d -p 80:3000 --name reactapp -v $(pwd):/app
reactapp:dev1
```

```
//Dockerfile.prod
FROM node:18.12.1-alpine3.16
RUN npm install -g serve
RUN addgroup -g 1001 app && adduser -u 1001 -G app -s /bin/sh -D app
WORKDIR /app
COPY package*.json /app/
RUN npm install
COPY . .
EXPOSE 3000
RUN npm run build
USER app
CMD serve -s build

//building production image
docker build -f ./Dockerfile.prod --rm --tag reactapp:prod1 .

//running production container
docker container run -d -p 80:3000 --name reactapp reactapp:prod1
```

RUNNING MULTI CONTAINER APPS

▷ CLEANING THE WORKSPACE

```
docker container rm -f $(docker container ls -q)
docker container rm -f $(docker container ls -a -q)
docker volume rm -f $(docker volume ls -q)
docker image rm -f $(docker image ls -q)
```

▷ YAML (YAML AIN'T MARKUP LANGUAGE)

- ▽ Data serialization language for all programming languages
- ▽ Other data serialization languages are XML and JSON
- ▽ can store data not commands
- ▽ It is case sensitive

```
---
apple: fruit
1: roll number
weight: 55.34
---
- first
- second
- third
---
mylist:
     - first
     - second
     - third
---
mylist: [first, second, third]
---
#object
person:
     name:
          firstname: abc
          lastname: xyz
     age: 24
...
```

▷ Docker compose file is used to compose multi-container application

▷ DOCKER COMPOSE FILE

▽ Specifications

▽ Documentation

▽ The Compose file is a YAML file defining services, networks, and volumes for a Docker application.

```yaml
//docker-compose.yaml
services:
     frontend:
          build: ./frontend
          ports:
               - 3000:3000
          depends_on:
               - backend
               - db
          links:
               - backend
     backend:
          build: ./backend
          ports:
               - 3001:3001
          environment:
               - DB_URL=mongodb://db/vidly
          depends_on:
               - db
     db:
          image: mongo:4.0-xenial
          volumes:
               - myvol:/data/db
volumes:
# creats a volume vildly_myvol
     myvol:
```

▷ RUNNING TESTS

```
frontend:
     build: ./frontend
     ports:
          - 3000:3000
     depends_on:
          - backend
          - db
     links:
          - backend
     volumes:
          - ./frontend:/app

frontend-test:
     image: vidly_frontend
     volumes:
          - ./frontend:/app
     command: npm test
```

DEPLOYMENT OPTIONS

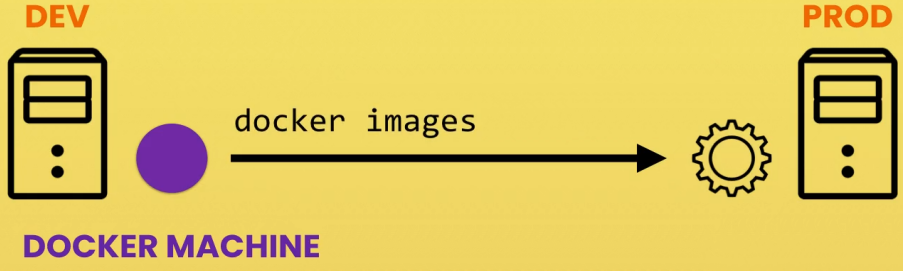
- Single-host deployment
- Cluster deployment

CLUSTER SOLUTIONS

- Docker Swarm
- Kubernetes

VPS OPTIONS

- Digital Ocean
- Google Cloud Platform (GCP)
- Microsoft Azure
- Amazon Web Services (AWS)



```
//frontend Dockerfile.prod
# step1: building stage
FROM node:14.16.0-alpine3.13 AS build-stage
RUN addgroup app && adduser -S -G app app
RUN mkdir app && chown app /app
WORKDIR /app
USER app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# step2: deploying stage
FROM nginx:alpine
COPY --from=build-stage /app/build /usr/share/nginx/html
EXPOSE 80
ENTRYPOINT [ "nginx","-g","daemon off;" ]
```