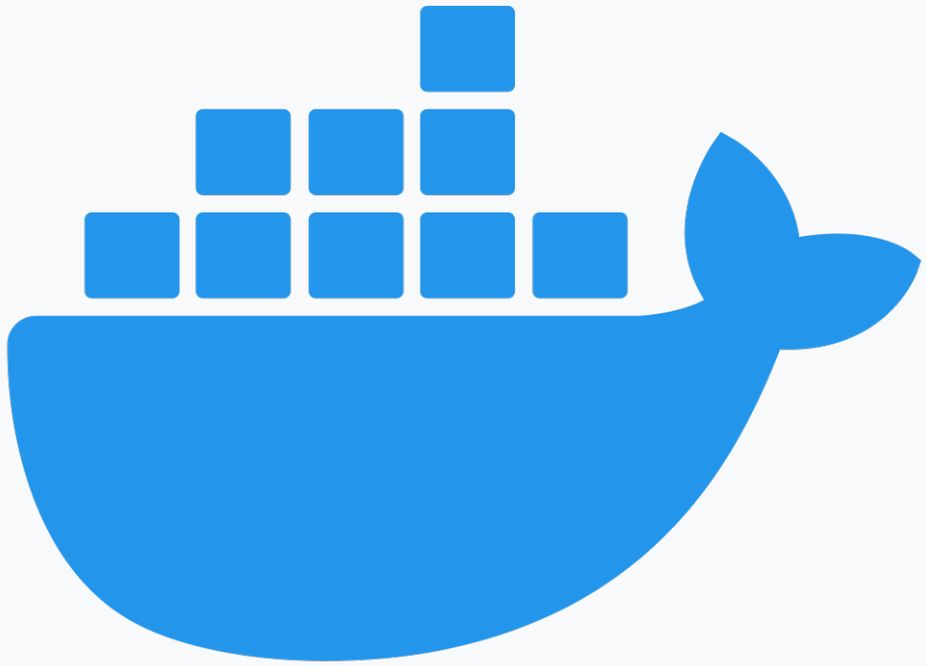
CODEMOBILES
COMPANY LIMITED



Basic

CODEMOBILES COMPANY LIMITED

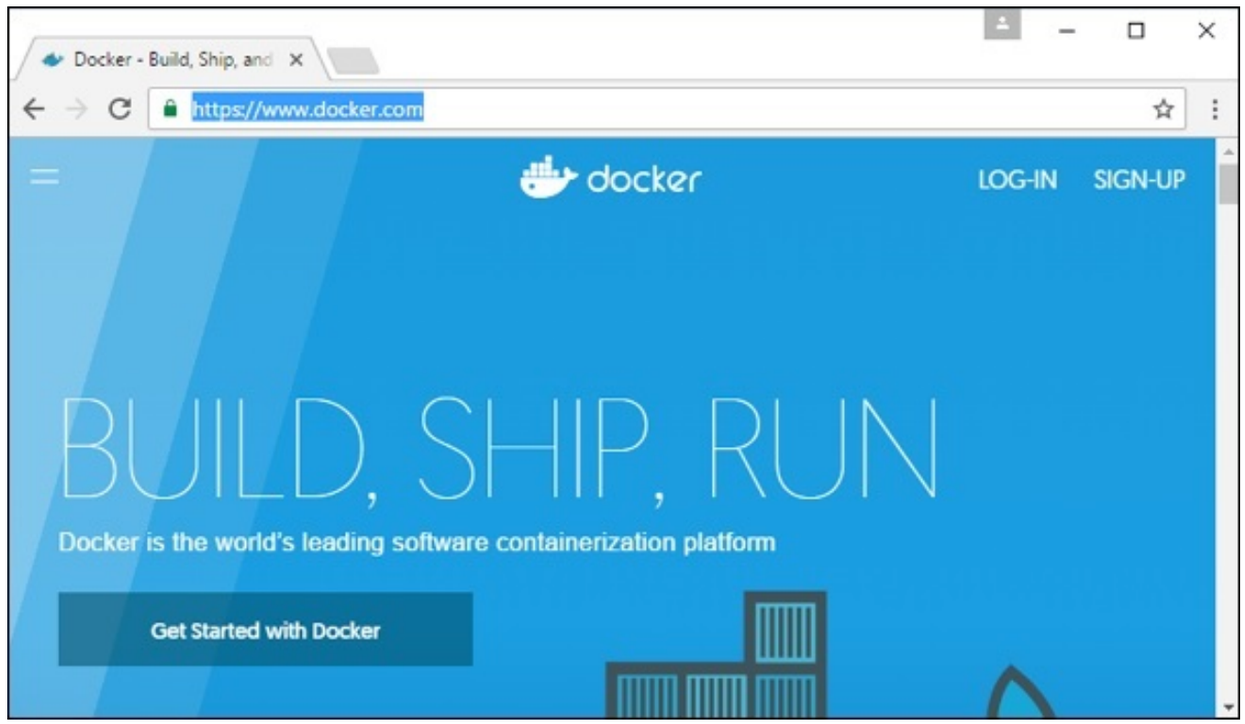
What is Docker?

- Docker
- NodeJS
- VSCode

CODEMOBILES COMPANY LIMITED

What is Docker?

Docker is a container management service. The keywords of Docker are **develop,** **ship** and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.



CODEMOBILES COMPANY LIMITED

Dock is ท่าเรือ

ทั้งหมด ค้นรูป Maps วิดีโอ ช็อปปิ้ง เพิ่มเติม เครื่องมือ

ผลการค้นหาประมาณ 489,000,000 รายการ (0.57 วินาที)

อังกฤษ ⇄ ไทย

dock

däk

ท่าเรือ

Thāreụ̄x

CODEMOBILES
COMPANY LIMITED

Dock

Components of Docker

Docker has the following components

- **Docker for Mac** − It allows one to run Docker containers on the Mac OS.
- **Docker for Linux** − It allows one to run Docker containers on the Linux OS.
- **Docker for Windows** − It allows one to run Docker containers on the Windows OS.
- **Docker Engine** − It is used for building Docker images and creating Docker containers.
- **Docker Hub** − This is the registry which is used to host various Docker images.
- **Docker Compose** − This is used to define applications using multiple Docker containers.

Docker Installation

Docker Desktop for Mac

Docker Desktop for Windows

Docker for Linux

Docker Desktop

Developer productivity tools and a local Kubernetes environment.

Download for Mac - Intel Chip

Download for Mac - Apple Chip

Download for Windows

View Linux Engine

https://www.docker.com/get-started

CODEMOBILES COMPANY LIMITED

Docker CLI

Cheatsheet for Docker CLI

Run a new Container

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

Manage Images

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

Info & Stats

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

To test that Docker runs properly, we can use the Docker **run command** to download and run a simple **HelloWorld Docker container**.

The working of the Docker **run command** is given below –

```
docker run
```

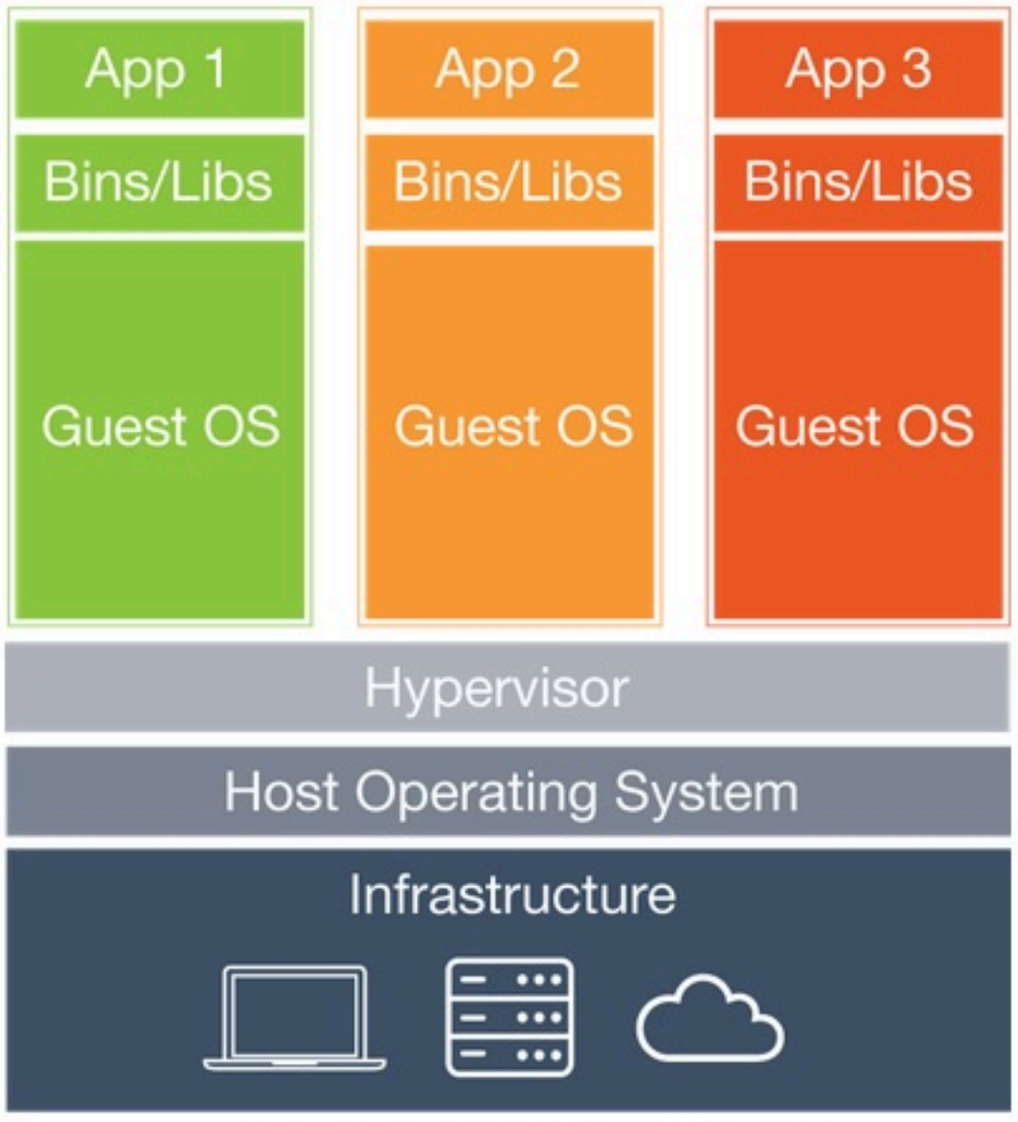
This command is used to run a command in a Docker container.

Syntax

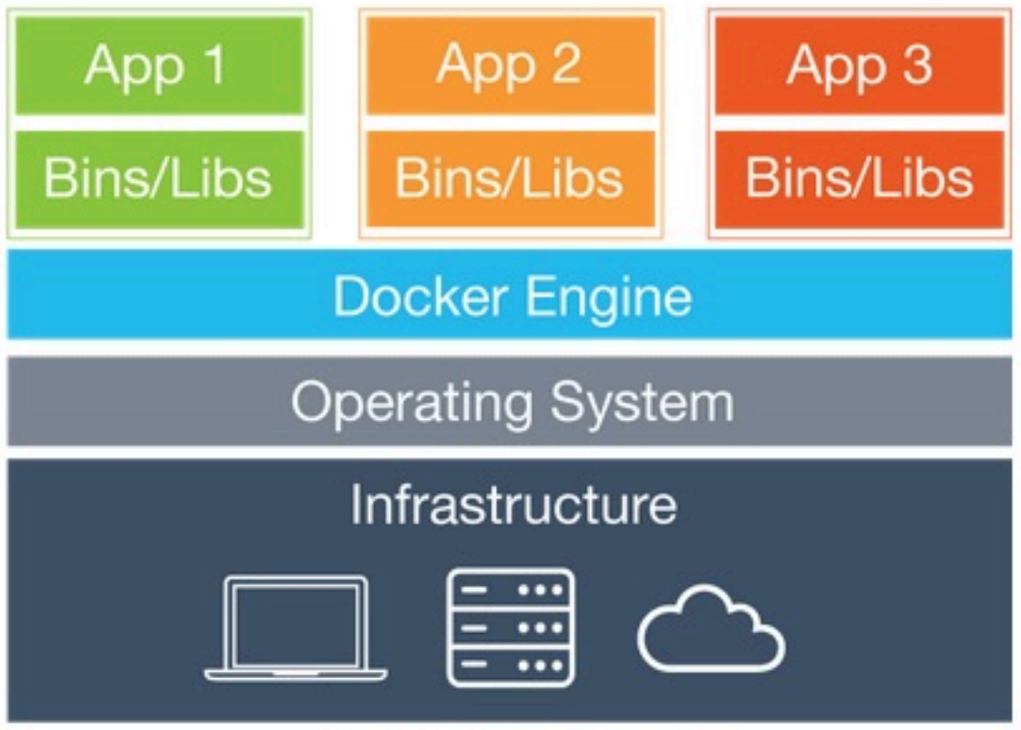
```
docker run image
```

CODEMOBILES COMPANY LIMITED

Container vs VMs

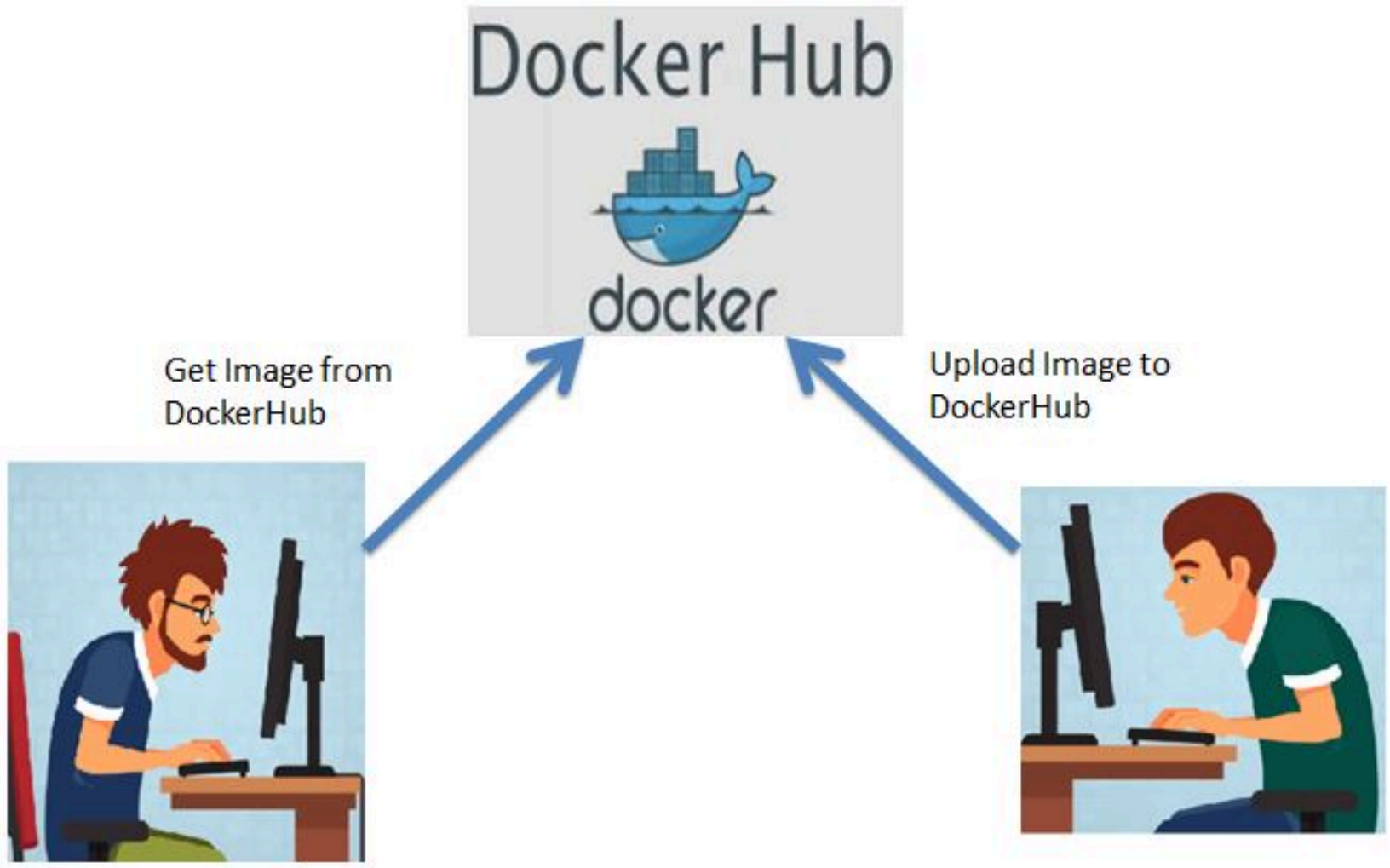


Virtual Machines



Containers

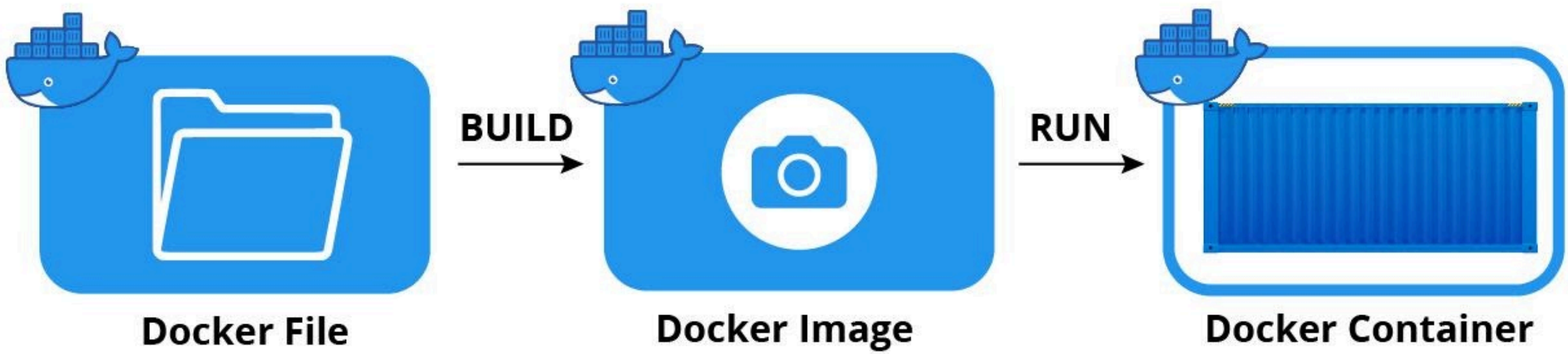
Docker Hub



Share Work Environment using Docker

CODEMOBILES
COMPANY LIMITED

Docker Image



```
sudo docker images
```

Output

When we run the above command, it will produce the following result −

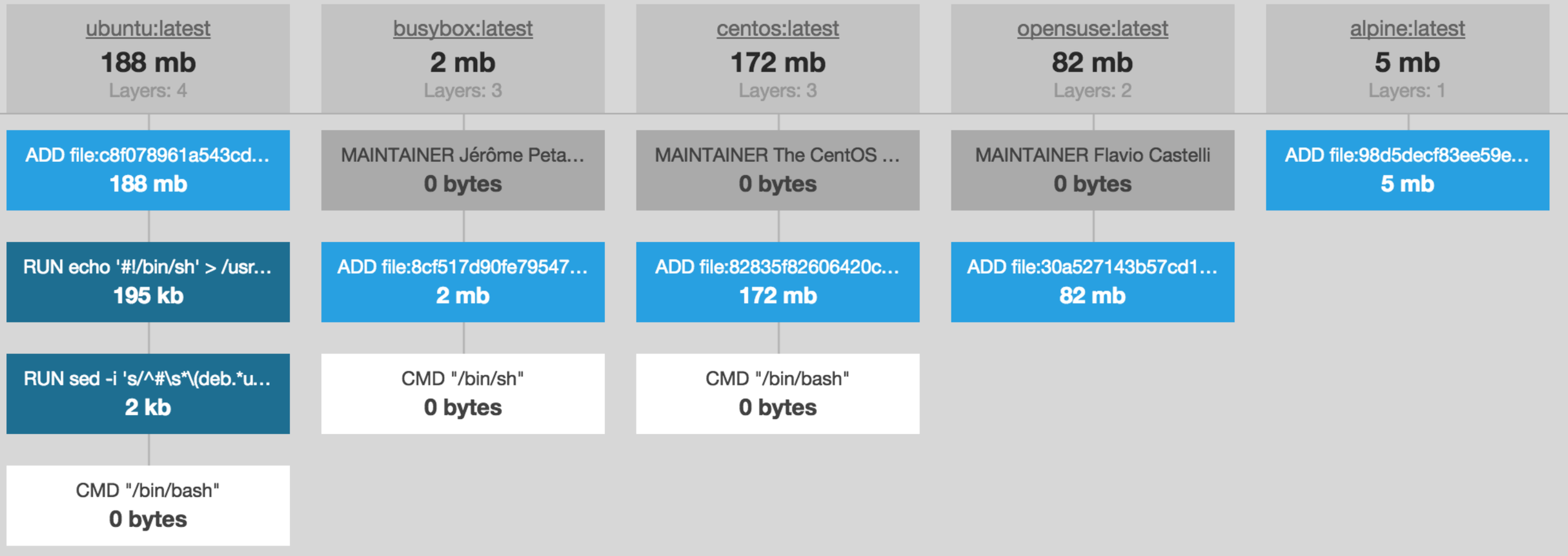
```
demo@ubuntuserver:~$ sudo docker images
[sudo] password for demo:
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
newcentos           latest              7a86f8ffcb25        9 days ago
196.5 MB
jenkins             latest              998d1854867e        2 weeks ago
714.1 MB
centos              latest              97cad5e16cb6        4 weeks ago
196.5 MB
demo@ubuntuserver:~$ _
```

From the above output, you can see that the server has three images: **centos, newcentos,** and **jenkins**. Each image has the following attributes −

- **TAG** − This is used to logically tag images.
- **Image ID** − This is used to uniquely identify the image.
- **Created** − The number of days since the image was created.
- **Virtual Size** − The size of the image.

CODEMOBILES COMPANY LIMITED

Docker Linux Dist.



https://brianchristner.io/docker-image-base-os-size-comparison/

Docker Image

- docker run --rm -it node node
- docker run --rm -it python python
- docker run --rm -it go go version

Image management commands

command	description
`docker images`	list all local images
`docker history` *image*	show the image history (list of ancestors)
`docker inspect` *image...*	show low-level infos (in json format)
`docker tag` *image tag*	tag an image
`docker commit` *container image*	create an image (from a container)
`docker import` *url*|- [*tag*]	create an image (from a tarball)
`docker rmi` *image...*	delete images

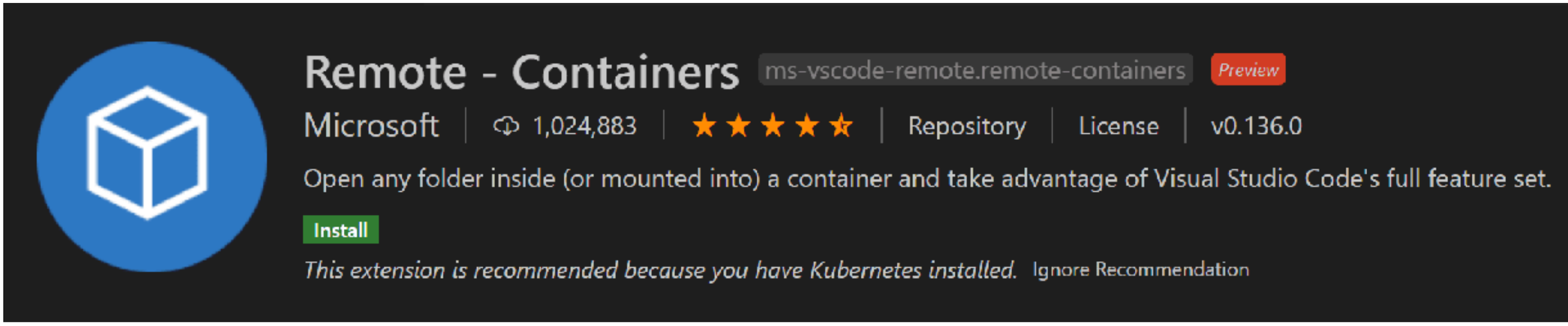
- docker run --rm -it node /bin/bash
- apt-get update
- apt-get install vim
- npm init -y
- npx node ...

Interacting with the container

command	description
`docker attach` *container*	attach to a running container (stdin/stdout/stderr)
`docker cp` *container:path hostpath*|- `docker cp` *hostpath*|- *container:path*	copy files from the container copy files into the container
`docker export` *container*	export the content of the container (tar archive)
`docker exec` *container args...*	run a command in an existing container (**useful** for debugging)
`docker wait` *container*	wait until the container terminates and return the exit code
`docker commit` *container image*	commit a new docker image (snapshot of the container)

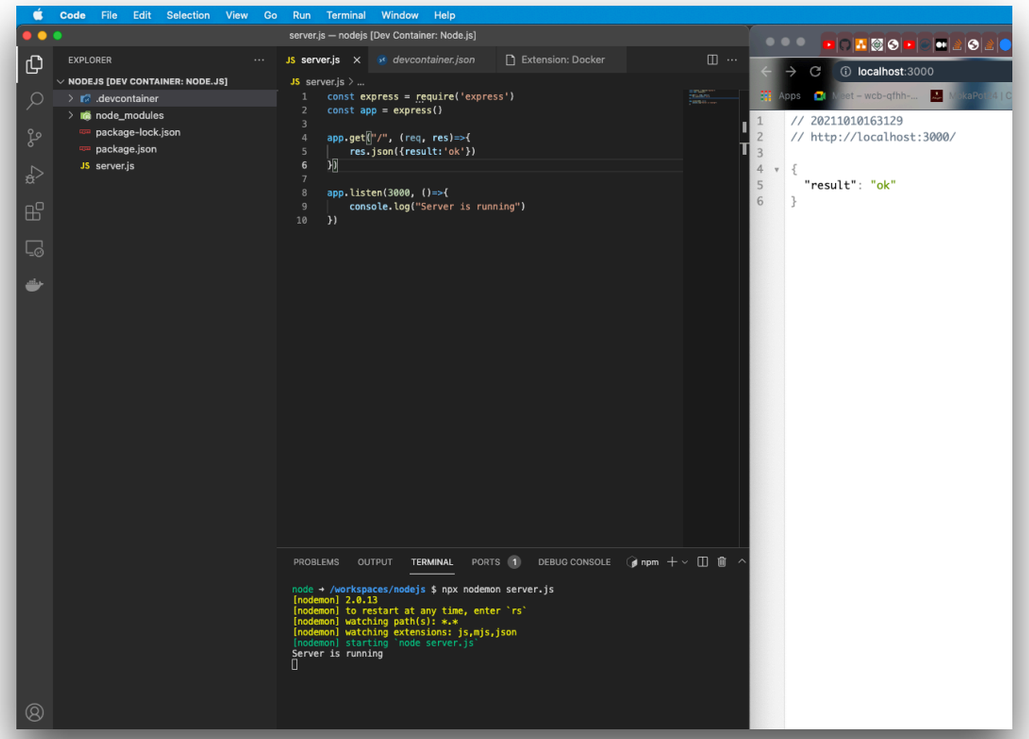
Remote development in Containers

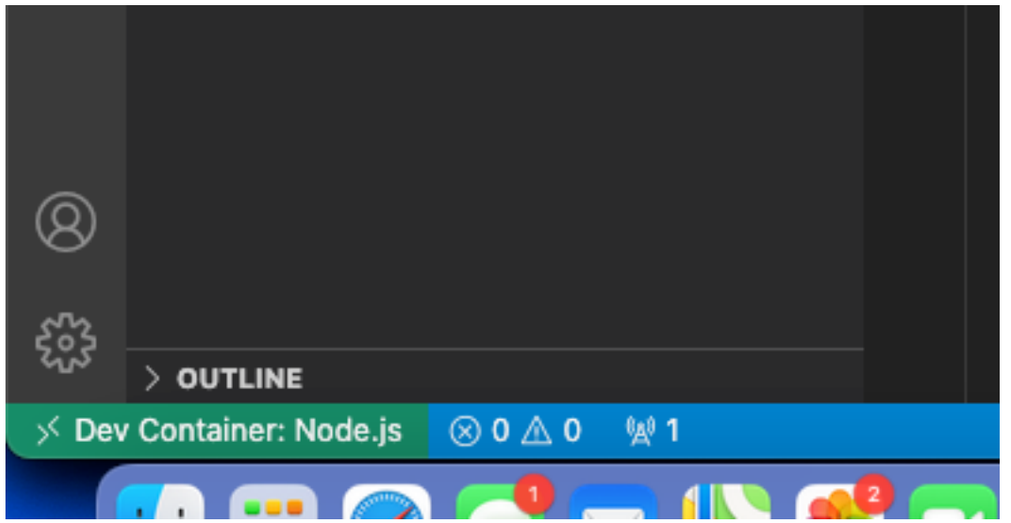
https://code.visualstudio.com/docs/remote/containers-tutorial

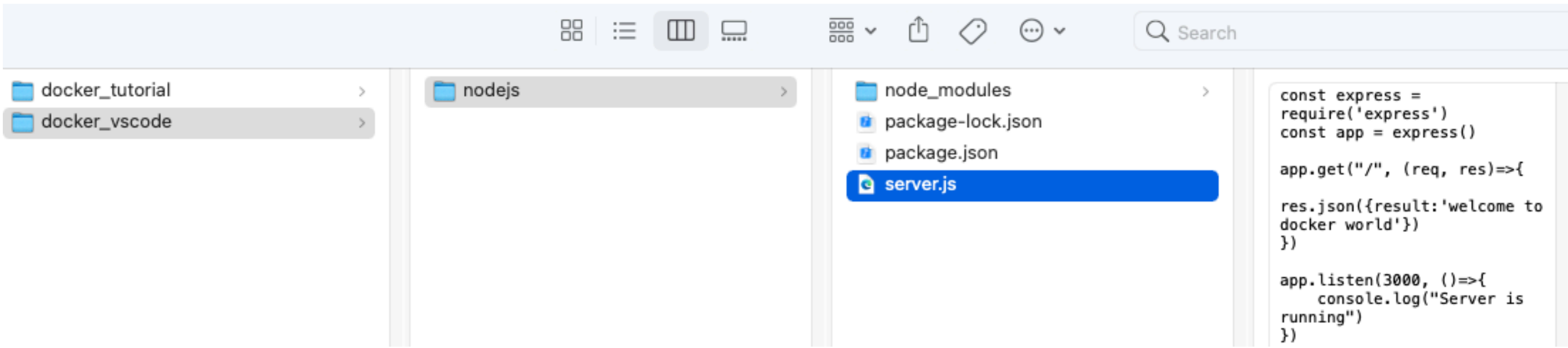


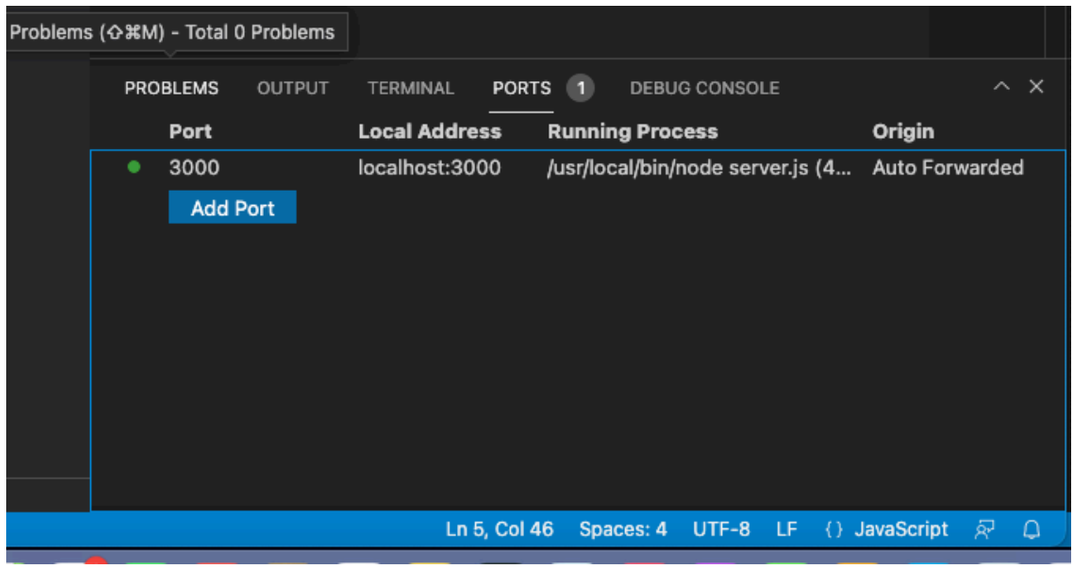
CODEMOBILES COMPANY LIMITED

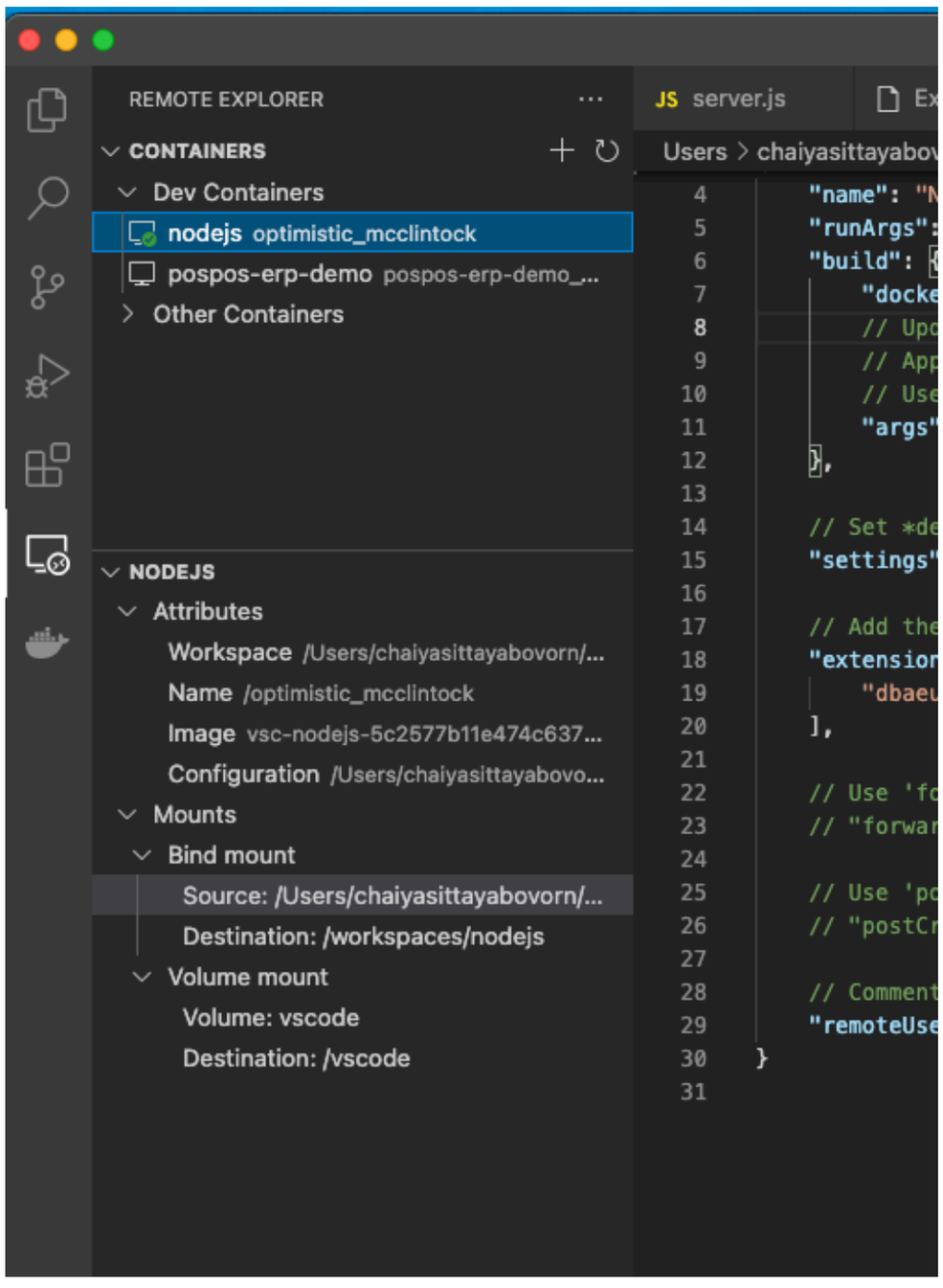
Docker for Developer








Problems (⇧⌘M) - Total 0 Problems
PROBLEMS
OUTPUT
TERMINAL
PORTS 1
DEBUG CONSOLE
Port
Local Address
Running Process
Origin
3000
localhost:3000
/usr/local/bin/node server.js (4...
Auto Forwarded
Add Port
Ln 5, Col 46
Spaces: 4
UTF-8
LF
JavaScript


REMOTE EXPLORER
CONTAINERS
Dev Containers
nodejs optimistic_mcclintock
pospos-erp-demo pospos-erp-demo_...
Other Containers
NODEJS
Attributes
Workspace /Users/chaiyasittayabovorn/...
Name /optimistic_mcclintock
Image vsc-nodejs-5c2577b11e474c637...
Configuration /Users/chaiyasittayabovo...
Mounts
Bind mount
Source: /Users/chaiyasittayabovorn/...
Destination: /workspaces/nodejs
Volume mount
Volume: vscode
Destination: /vscode
JS server.js
Users > chaiyasittayabov
"name": "N
"runArgs":
"build": {
"docke
// Upc
// App
// Use
"args"
},
// Set *de
"settings"
// Add the
"extensior
"dbaeu
],
// Use 'fc
// "forwar
// Use 'po
// "postCr
// Comment
"remoteUse
}

CODEMOBILES COMPANY LIMITED

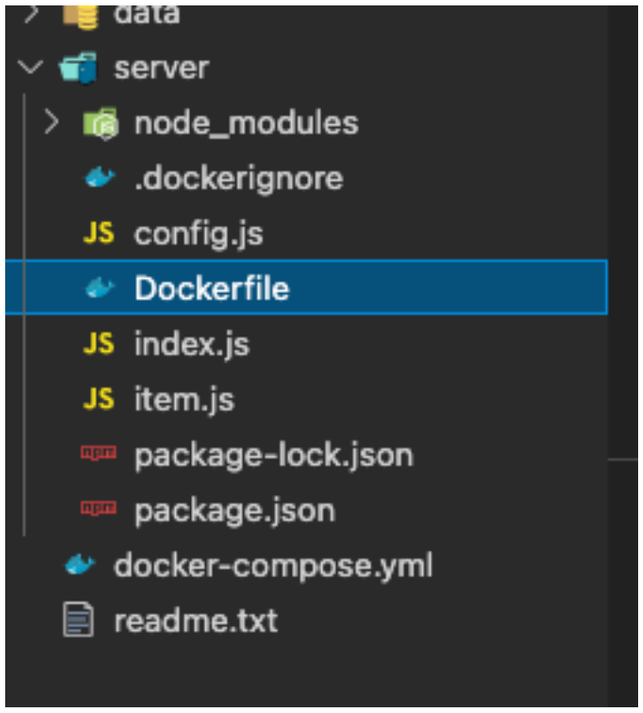
Debug Image

docker run --rm -it --entrypoint sh image:tag

```
CMDevM1 [~/Desktop/Training/Docker/workshop/docker_tutorial/docker_mean_demo/server]$ docker run --rm -it --entrypoint sh meanserver:1.0
/usr/src/app # ls -lrt
total 44
-rw-r--r--    1 root     root         22383 Oct 10 10:06 package-lock.json
-rw-r--r--    1 root     root           327 Oct 10 10:27 package.json
-rw-r--r--    1 root     root           192 Oct 10 10:44 config.js
-rw-r--r--    1 root     root           271 Oct 10 11:01 item.js
-rw-r--r--    1 root     root          1221 Oct 11 02:50 index.js
drwxr-xr-x   74 root     root          4096 Oct 11 02:50 node_modules
/usr/src/app #
```

Dockerfile

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.




```
server > Dockerfile > ...
1   FROM node:15.14.0-alpine3.10
2
3   WORKDIR /usr/src/app
4   COPY package*.json ./
5   RUN npm install
6   COPY . .
7   EXPOSE 8080
8   CMD [ "npm", "start" ]
9
```

Builder main commands

command	description
`FROM` *image*|`scratch`	base image for the build
`MAINTAINER` *email*	name of the mainainer (metadata)
`COPY` *path dst*	copy *path* from the context into the container at location *dst*
`ADD` *src dst*	same as `COPY` but untar archives and accepts http urls
`RUN` *args...*	run an arbitrary command inside the container
`USER` *name*	set the default username
`WORKDIR` *path*	set the default working directory
`CMD` *args...*	set the default command
`ENV` *name value*	set an environment variable

CODEMOBILES COMPANY LIMITED

Dockerfile

- FROM
- WORKDIR
- COPY
- RUN
- EXPOSE
- CMD

r > Dockerfile > ...

```
FROM node:15.14.0-alpine3.10

WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD [ "npm", "start" ]
```

CODEMOBILES COMPANY LIMITED

Dockerfile

- ENV
- AS
- ENTRYPOINT
- MAINTAINER
- DEPEND
- ADD
- LINK
- NETWORK
- VOLUME

Dockerfile client U × | readme.txt | environment.prod.ts | app.com

client > Dockerfile > ...

```
1   FROM node:15.14.0-alpine3.10 AS builder
2   WORKDIR /usr/src/app
3
4   COPY package.json package-lock.json ./
5   RUN npm install
6
7   ENV PATH="./node_modules/.bin:$PATH"
8   COPY . ./
9   RUN ng build --prod
10
11  FROM nginx:1.19.10-alpine
12  COPY --from=builder /usr/src/app/dist/client /usr/share/nginx/html
13
```

Build Dockerfile

docker build --no-cache -t image_name:1.0 .

- build : to build
- --no-cache : to re-build completely without using cache
- -t : to tag image name and version
- image_name:1.0 : name and tag of image
- . : target folder

CODEMOBILES COMPANY LIMITED

Debug Build Dockerfile

DOCKER_BUILDKIT=0 docker build --no-cache -t meanstackapp:1.0 .

- Set Env DOCKER_BUILDKIT=0 : to debug or see console output from run command

Debug with entrypoint

docker run -it --rm **--entrypoint sh** image:tag

This override/intercept the base entrypoint - in order to shell to view content. Use this option to verify content in image

```
[root@berded tmp]# docker run -it --rm --entrypoint sh tmp:1.0
/usr/src/app #
```

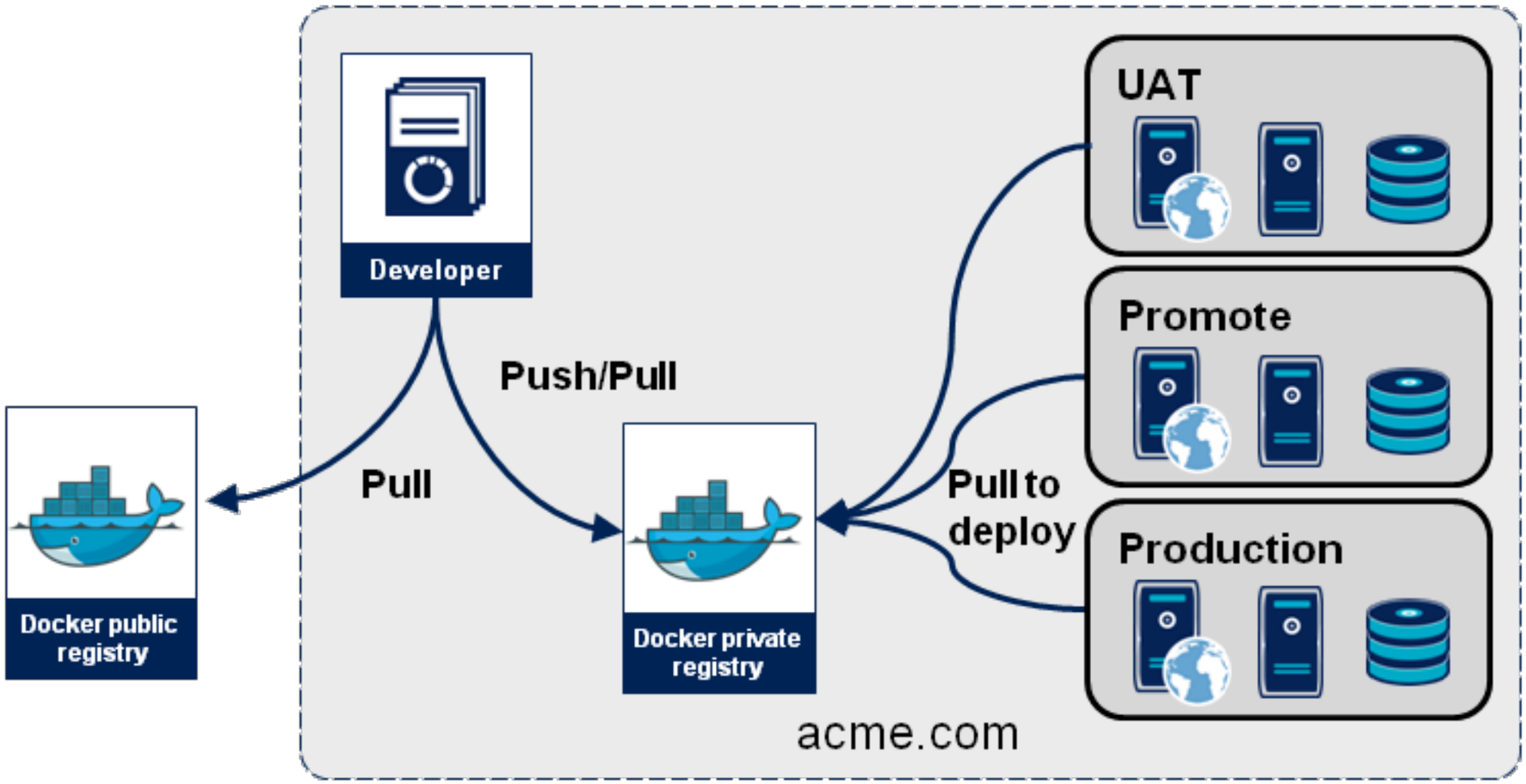
CODEMOBILES COMPANY LIMITED

Using Git in Dockerfile

client > Dockerfile > ...

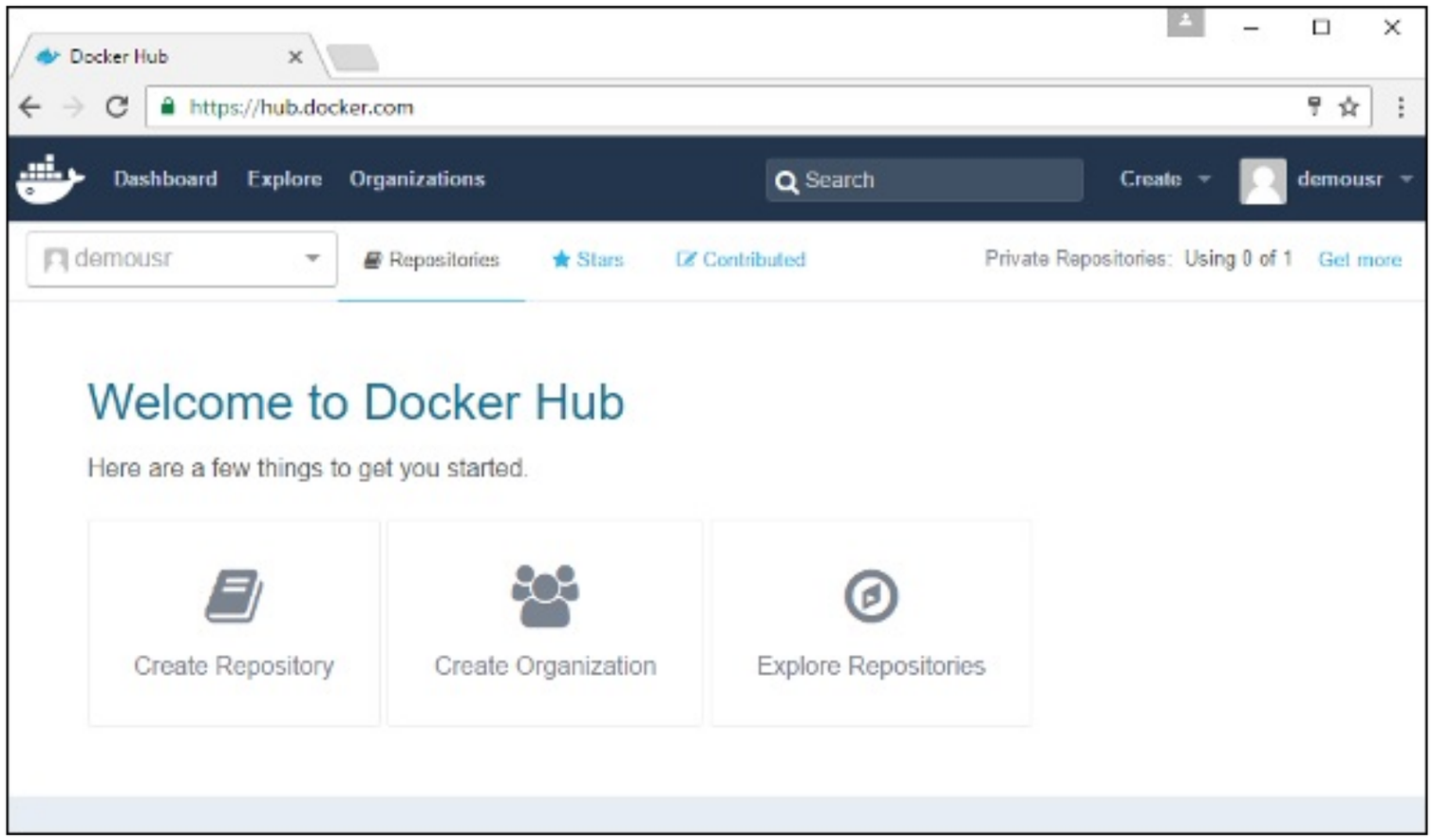
```
1   FROM node:14.18.1-alpine3.14 AS builder
2   WORKDIR /usr/src/app
3
4   COPY package*.json ./
5   RUN apk add --no-cache git
6   RUN npm install
7   RUN command
8
9   # ENV PATH="./node_modules/.bin:$PATH"
```

Docker hub


Developer
Push/Pull
Pull
Docker public registry
Docker private registry
UAT
Promote
Pull to deploy
Production
acme.com

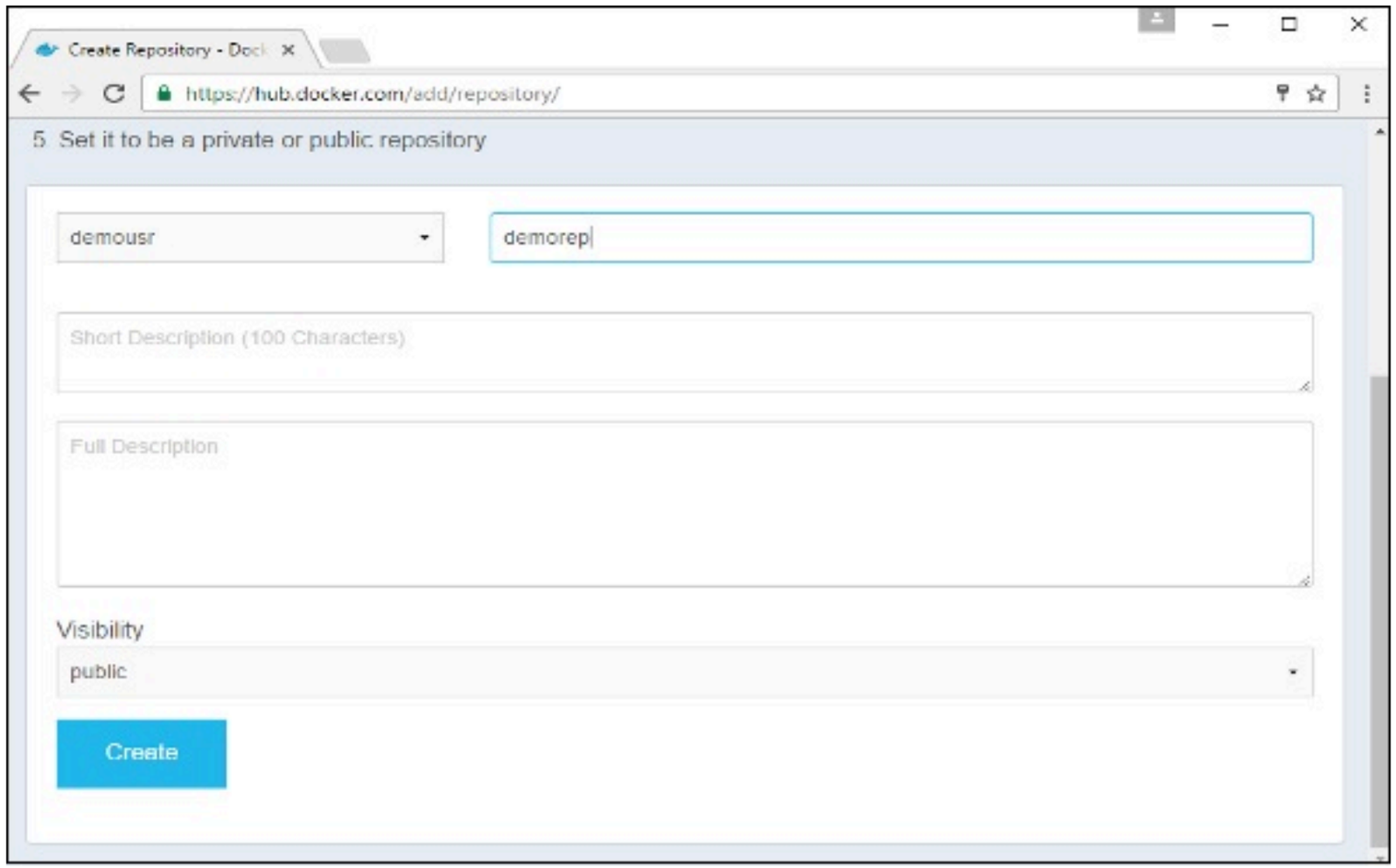
CODEMOBILES
COMPANY LIMITED

Docker hub



CODEMOBILES COMPANY LIMITED

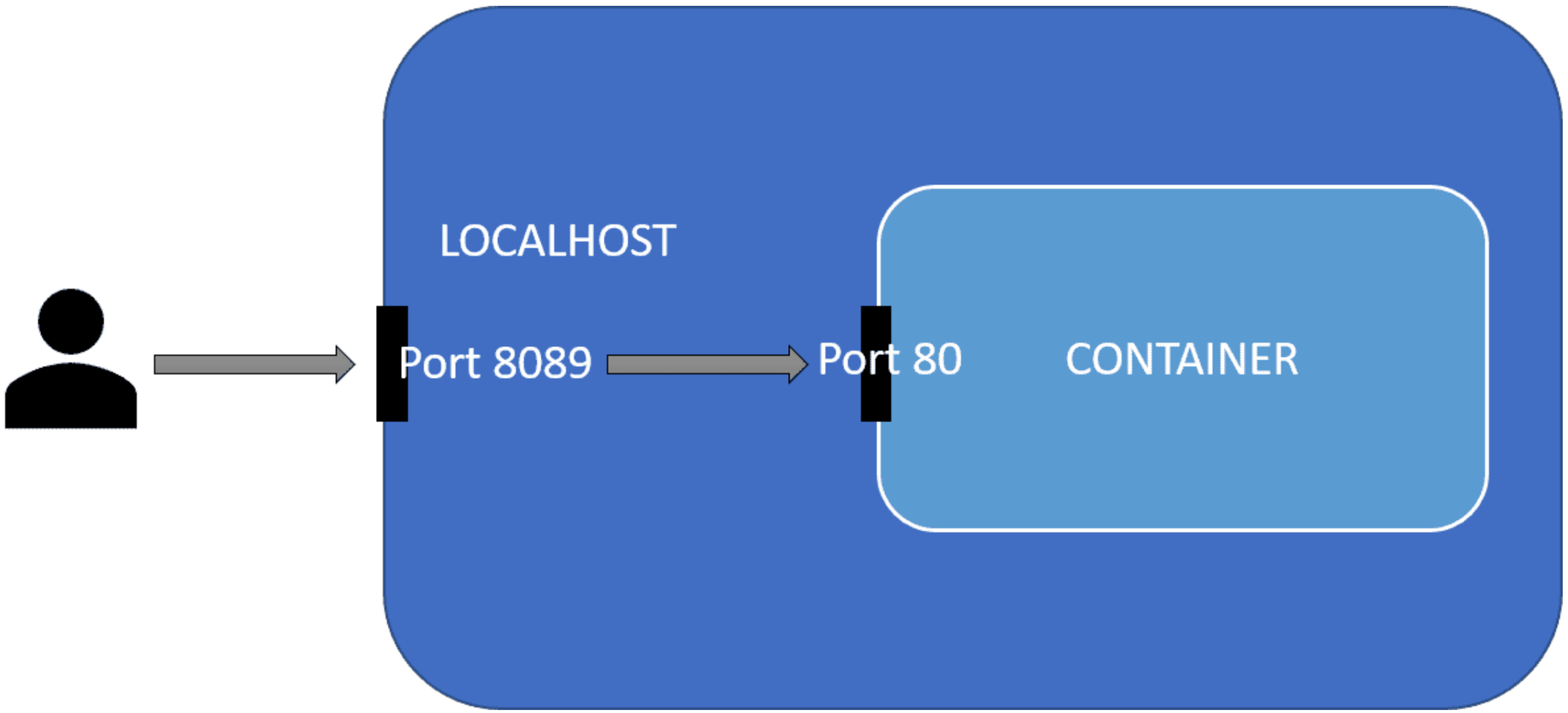
Submit Image to Docker hub



https://www.tutorialspoint.com/docker/docker_public_repositories.htm

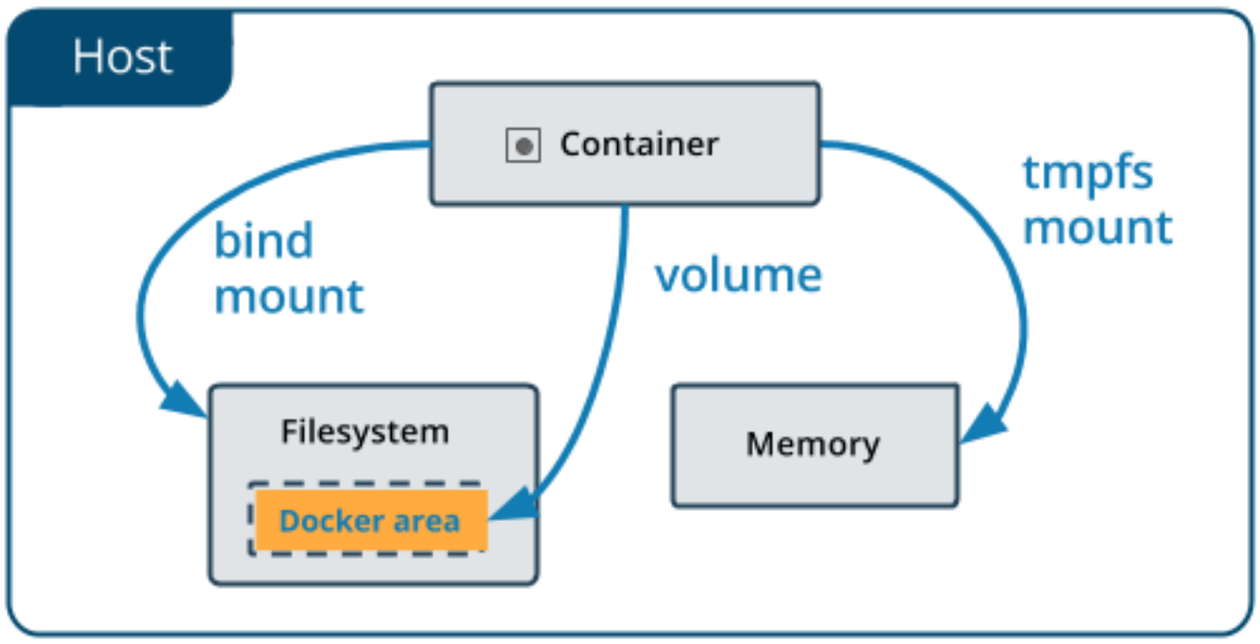
CODEMOBILES COMPANY LIMITED

Docker Port




```
sudo docker run -p 8089:80 -p host_port:container_port jenkins
```

Docker Volume




```
docker volume ls
```

CODEMOBILES COMPANY LIMITED

Remove Containers, Images, Volumes, Networks

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

Removing All Unused Docker Objects

The `docker system prune` command removes all stopped containers, dangling images, and unused networks:

```
$ docker system prune
```

You'll be prompt you to confirm the operation:

```
Output

WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N]
```

Use the `-f` (`--force`) option to bypass the prompt.

If you want to remove all unused images not just the dangling ones, add the `-a` (`--all`) option to the command:

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

Remove Unused Volumes

By default, the command doesn't remove unused volumes to prevent losing important data. To remove all unused volumes, pass the `--volumes` option:

```
$ docker system prune --volumes
```

```
Output
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all volumes not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

List All Containers

You can get a list of all containers by invoking the `docker container ls` command with the `-a` option:

```
$ docker container ls -a
```

or docker ps -a

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

List All Stopped Containers

```
docker container ls -a --filter status=exited --filter status=created
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

Remove all Stoped Containers

Removing all stopped containers

To remove all stopped containers, invoke the `docker container prune` command:

```
$ docker container prune
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

Removing one or more images

To remove one or more Docker images, first, you need to find the IDs of the images:

```
$ docker image ls
```

The output should look something like this:

```
Output
REPOSITORY              TAG                 IMAGE ID            CREATED
centos                  latest              75835a67d134        7 days ago
ubuntu                  latest              2a4cca5ac898        2 months ago
linuxize/fedora         latest              a45d6dca3361        3 months ago
java                    8-jre               e44d62cf8862        3 months ago
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

Remove Dangling Images (Unused)

Removing dangling images

Docker provides a `docker image prune` command that can be used to remove dangled and unused images.

A dangling image is an image that is not tagged and is not used by any container. To remove dangling images, type:

```
$ docker image prune
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

List All Dangling Images (Unused)

Removing all unused images

To remove all images that are not referenced by any existing container, not just the dangling ones, use the `prune` command with the `-a` option:

```
$ docker image prune -a
```

```
Output

WARNING! This will remove all images without at least one container associate
Are you sure you want to continue? [y/N] y
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

Prune Objects

```
# system all objects
docker system prune

# images
docker image prune
docker image prune  --filter "until=12h"

# containers
docker container prune
docker container prune --filter "until=12h"

# volume
docker system prune --volumes

# network
docker network ls
docker network prune
docker network prune --filter "until=12h"
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

CODEMOBILES COMPANY LIMITED

List All Dangling Images (Unused)

Removing images using filters

With the `docker image prune` command, you can also remove images based on a particular condition with the `--filter` option.

At the time of the writing of this article, the currently supported filters are `until` and `label`. You can use more than one filter.

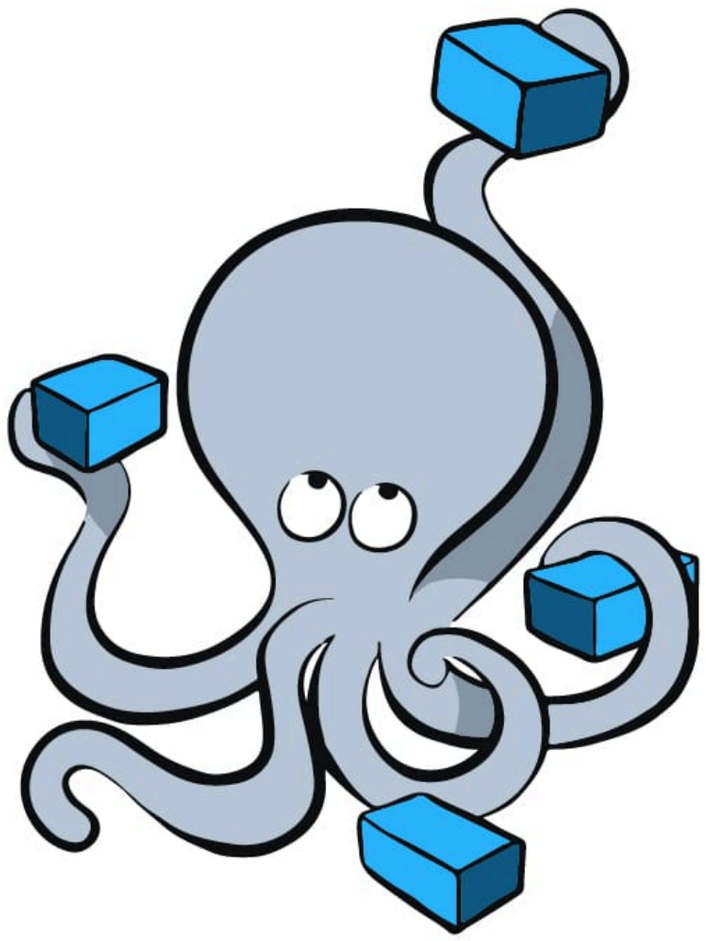
For example, to remove all images that are created more than seven days (168 hours) ago, you would run:

```
$ docker image prune -a --filter "until=168h"
```

https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#removing-all-unused-images

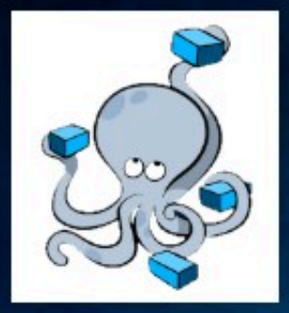
CODEMOBILES
COMPANY LIMITED

Docker-compose

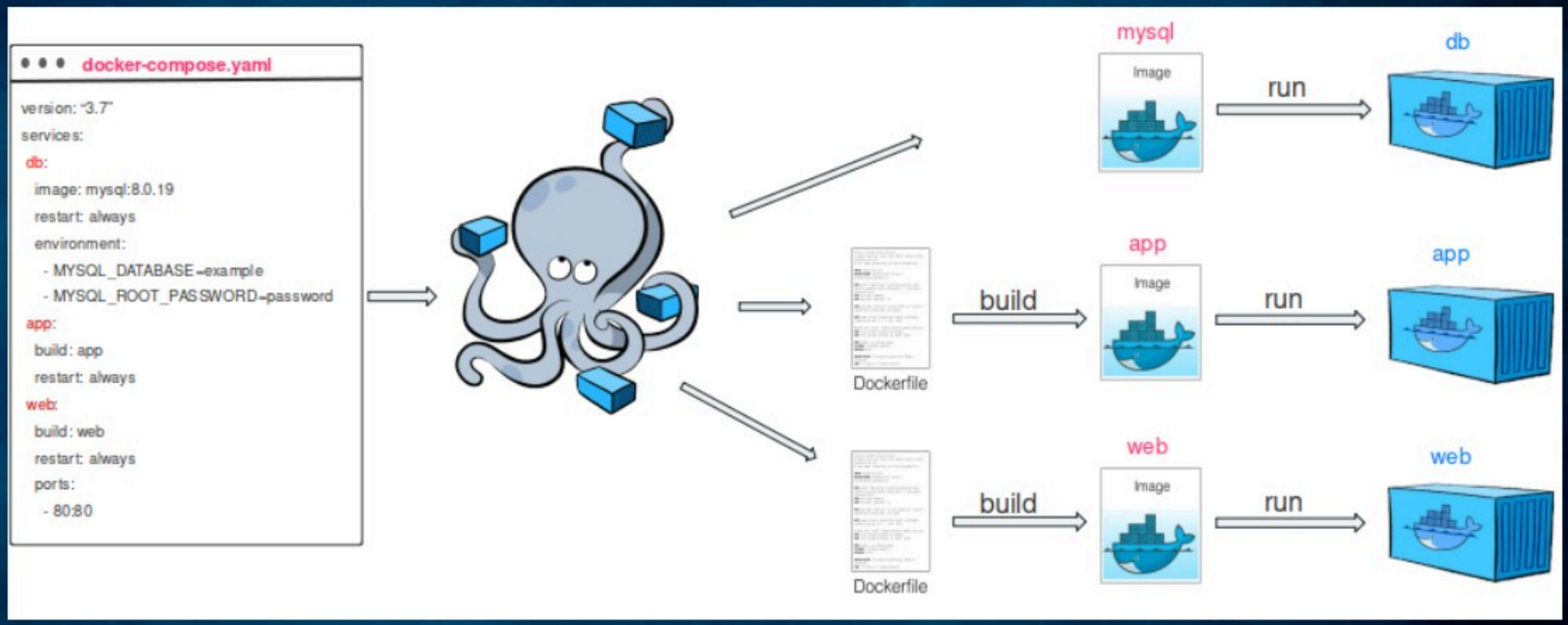


docker

Compose



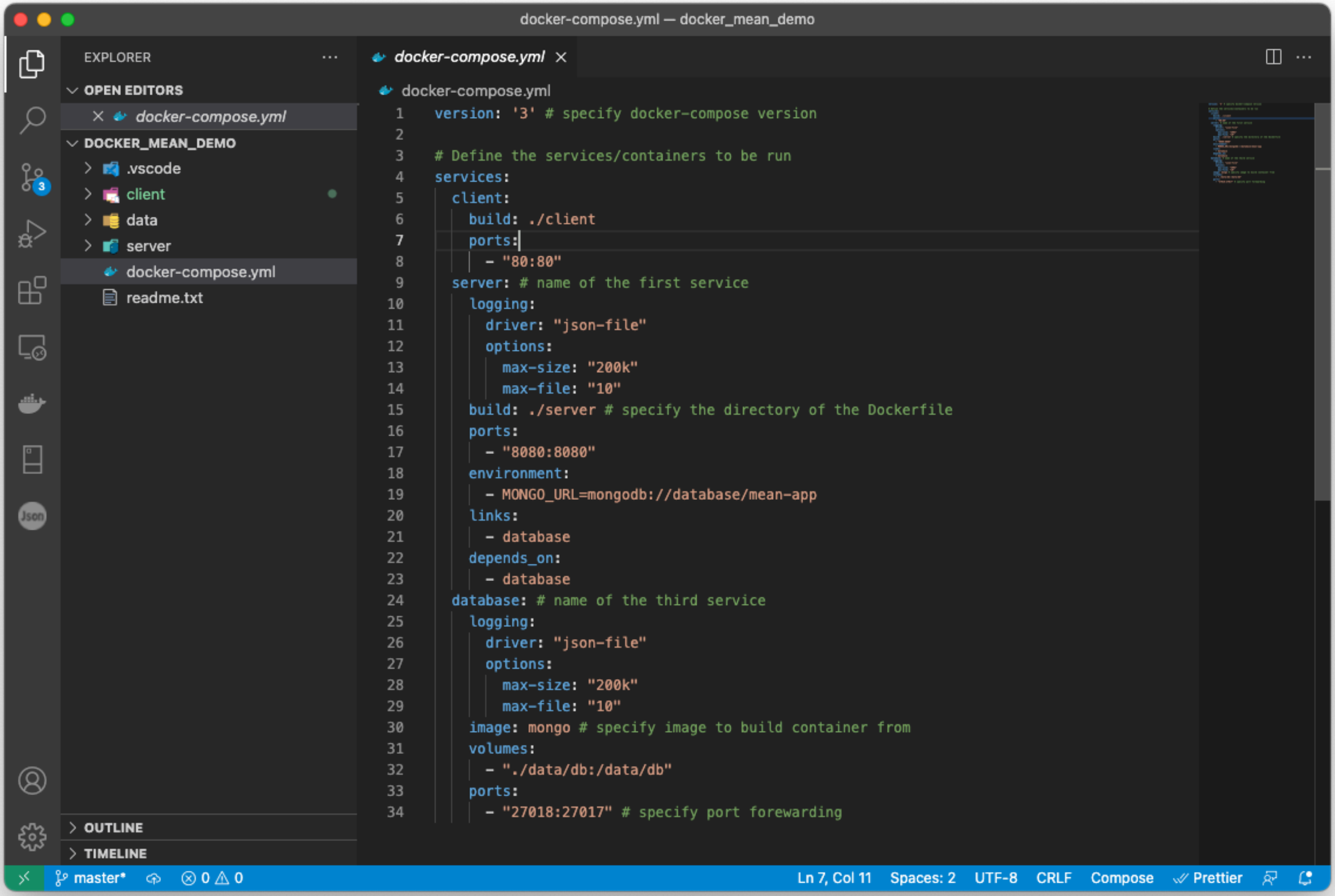
Docker Compose



Run multiple containers as a service from 1 .yaml file

CODEMOBILES COMPANY LIMITED

Docker-compose

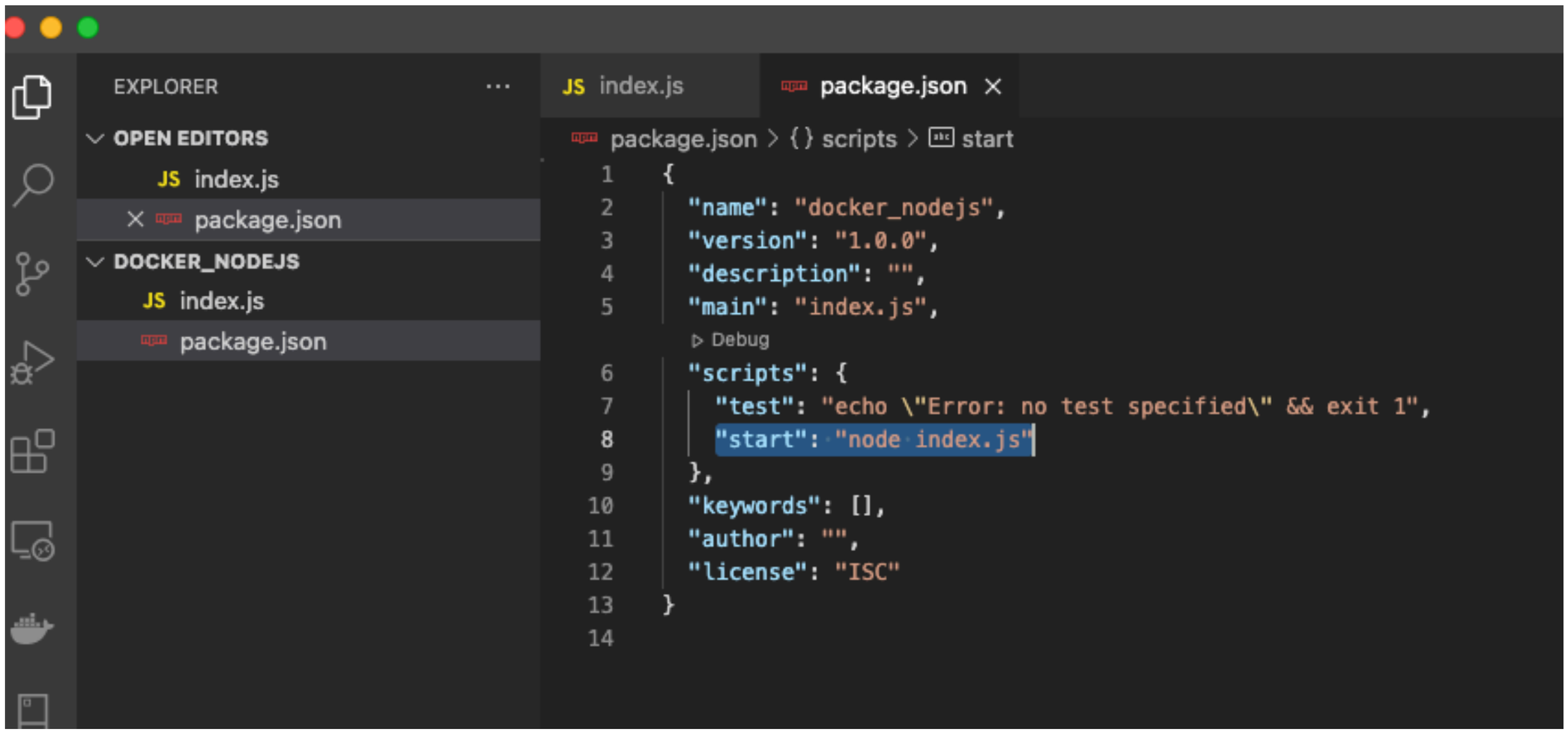


CODEMOBILES COMPANY LIMITED

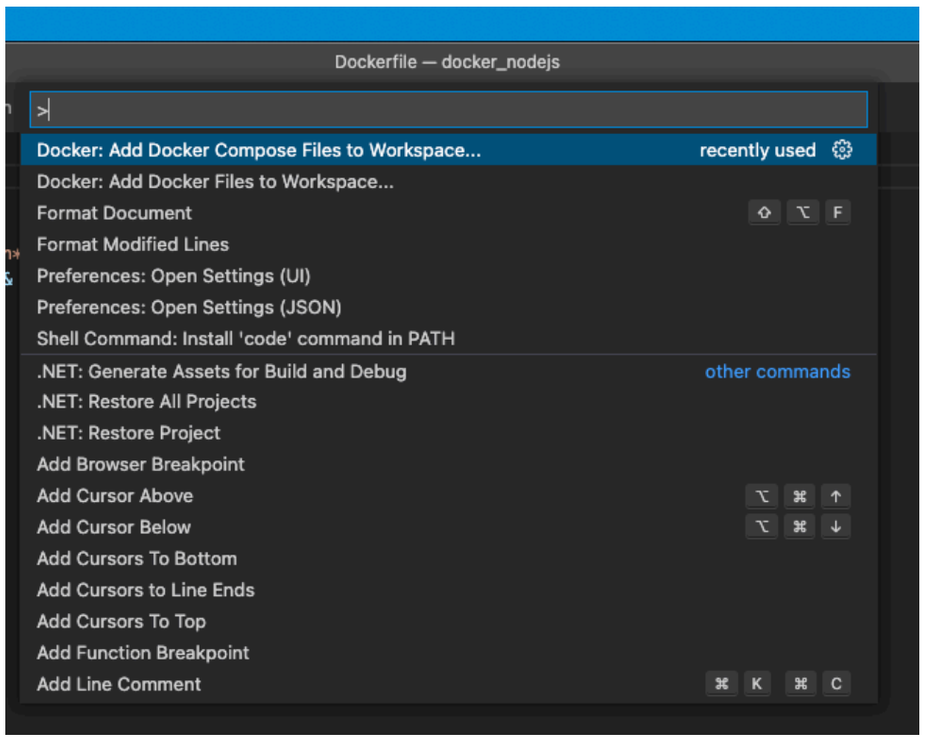
Docker NodeJS Workshop

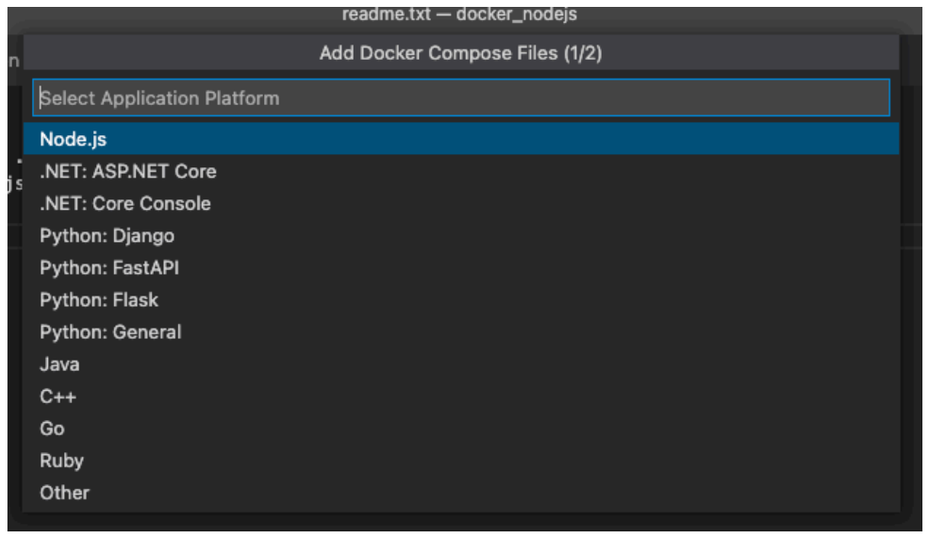
- **Preparation**
  - create a docker project folder like demo1
  - npm init -y
  - npm i express
  - Revise package.json to start index.js file
    - "start": "node index.js"
- Create index.js
- Create Dockerfile with VSCode Docker Extension
- docker build -t cmnodejs:1.0 .
- docker run -it --rm -p 3000:3000 cmnodejs:1.0

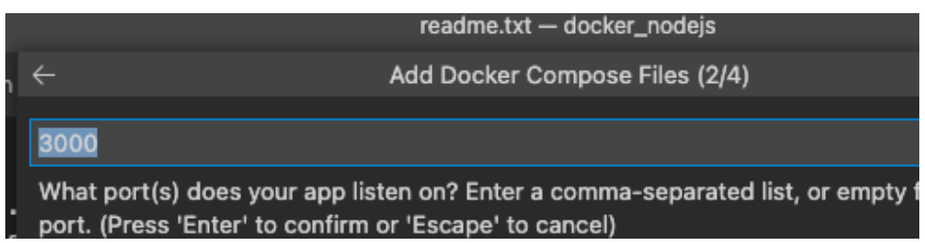
- create a docker project folder like demo1
- npm init -y
- npm i express
- Revise package.json to start index.js file : "start": "node index.js"

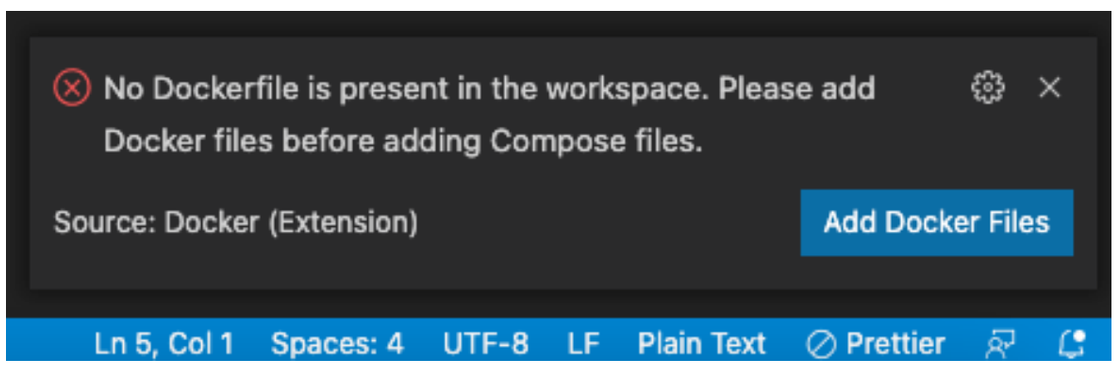


Create Dockerfile with VSCode Extension

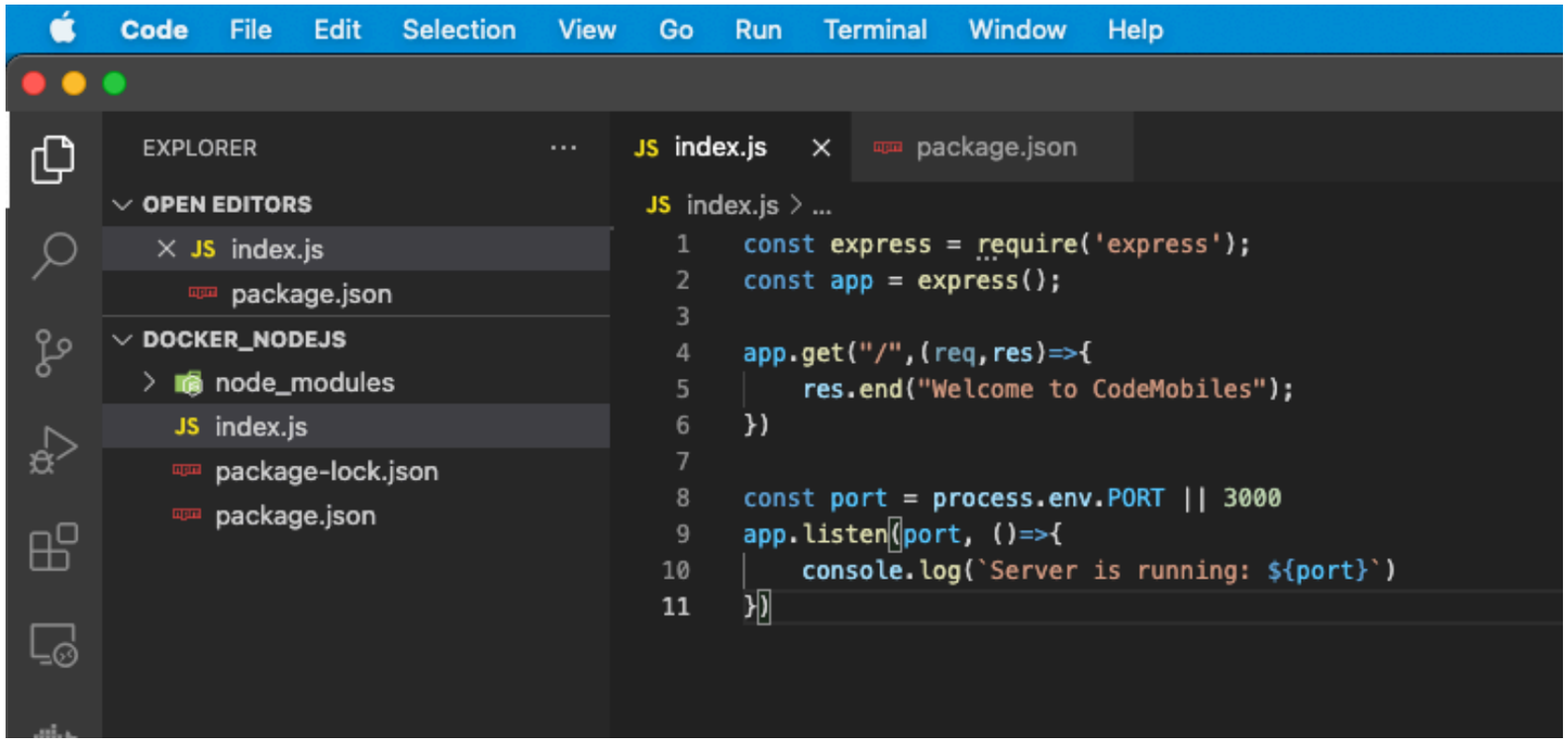








Create nodejs code named : index.js




```
const express = require('express');
const app = express();

app.get("/",(req,res)=>{
    res.end("Welcome to CodeMobiles");
})

const port = process.env.PORT || 3000
app.listen(port, ()=>{
    console.log(`Server is running: ${port}`)
})
```

docker build --no-cache -t cmnodejs:1.0 .

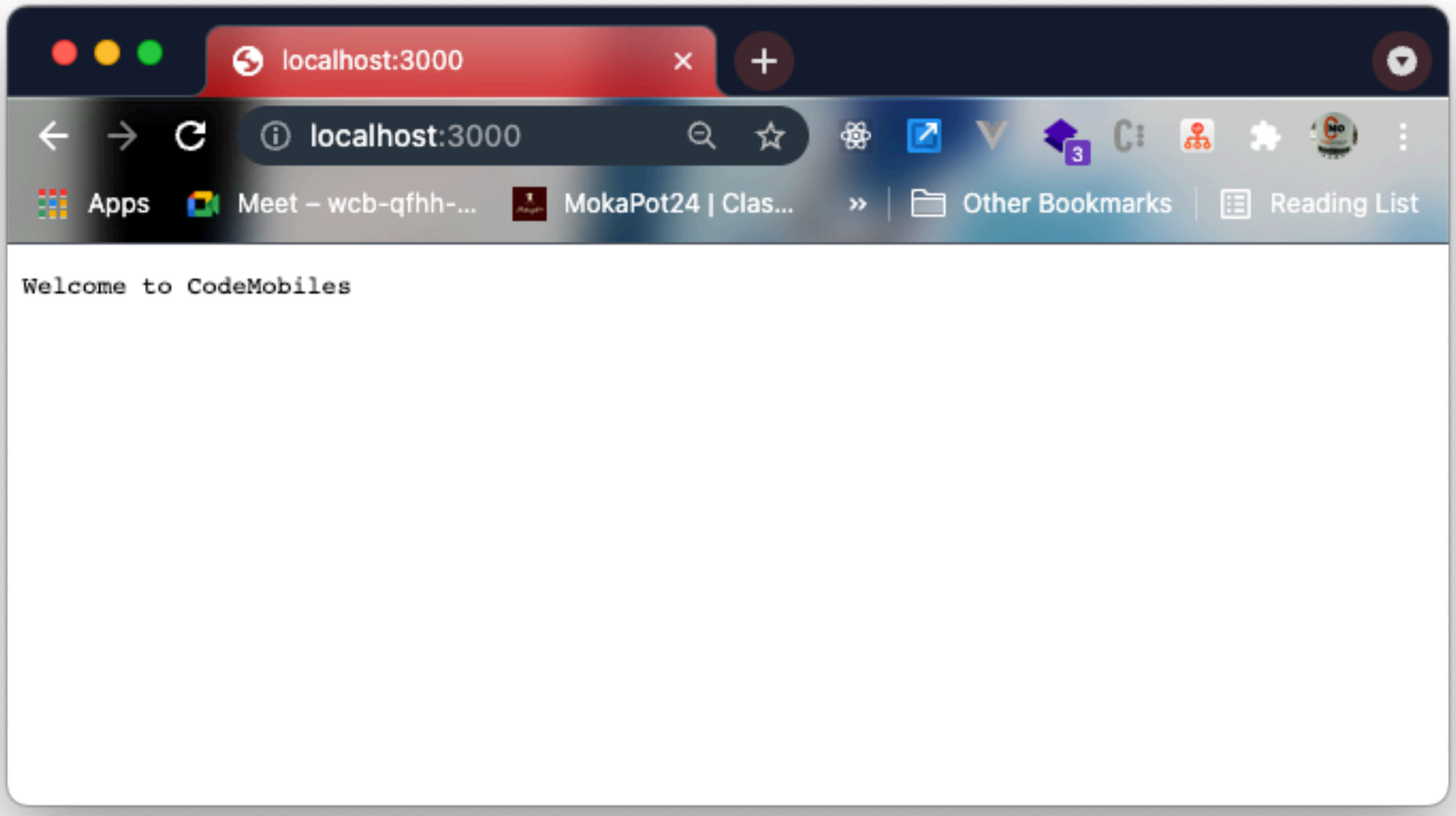
```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

CMDevM1 [~/Desktop/Training/Docker/workshop/docker_tutorial/docker_nodejs]$ docker build --no-cache -t cmnodejs:1.0 .
[+] Building 6.2s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 337B
 => [internal] load .dockerignore
 => => transferring context: 366B
 => [internal] load metadata for docker.io/library/node:14-alpine
 => [auth] library/node:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 245B
 => [1/6] FROM docker.io/library/node:14-alpine@sha256:6e52e0b3bedfb494496488514d18bee7fd503fd4e44289ea012ad02f8f41a312
 => CACHED [2/6] WORKDIR /usr/src/app
 => [3/6] COPY [package.json, package-lock.json*, npm-shrinkwrap.json*, ./]
 => [4/6] RUN npm install --production --silent && mv node_modules ../
 => [5/6] COPY . .
 => [6/6] RUN chown -R node /usr/src/app
 => exporting to image
 => => exporting layers
 => => writing image sha256:32aaec411c9cae9b476686d3d6920d8c57374c6add768c072ed1b988626211e2
 => => naming to docker.io/library/cmnodejs:1.0

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
CMDevM1 [~/Desktop/Training/Docker/workshop/docker_tutorial/docker_nodejs]$
```

docker run -it --rm -p 3000:3000 cmnodejs:1.0

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

CMDevM1 [~/Desktop/Training/Docker/workshop/docker_tutorial/docker_nodejs]$ docker run -it --rm -p 3000:3000 cmnodejs:1.0

> docker_nodejs@1.0.0 start /usr/src/app
> node index.js

Server is running: 3000
```

CODEMOBILES COMPANY LIMITED

Docker NodeJS Workshop



CODEMOBILES
COMPANY LIMITED

Thank You
For Your Attention