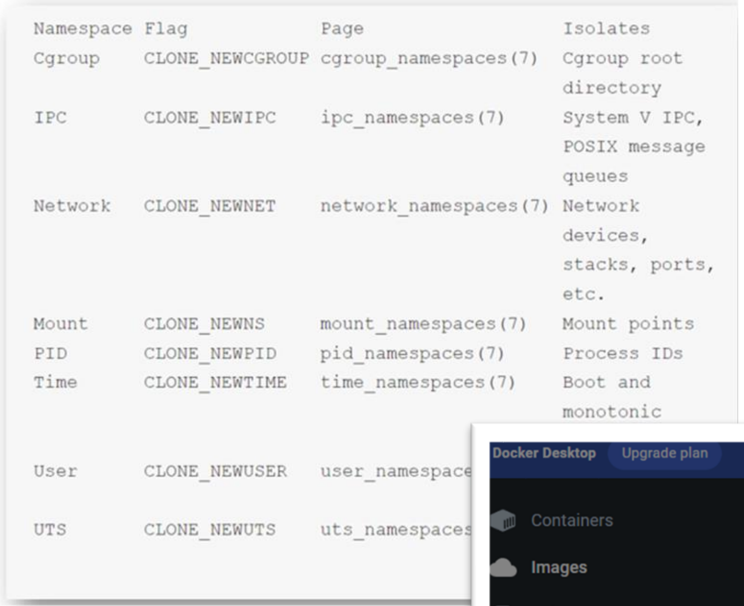
Containerization 3

By Majid Babaei

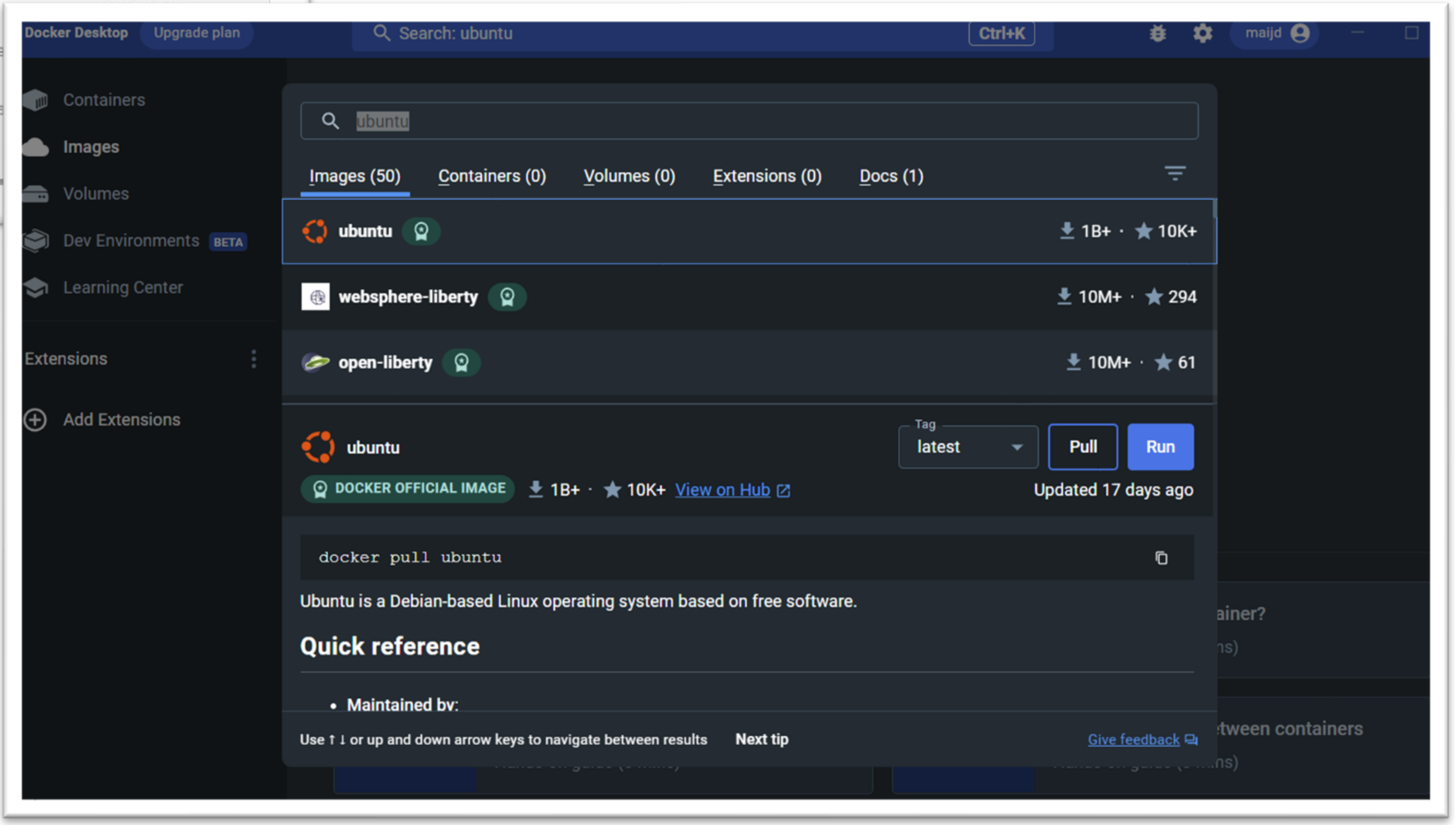
Types of Namespaces

Each type of namespace is different, and it provides isolation for different resources in our system.

If we check the namespaces in the Linux manual pages, we can see a list of namespace types:



Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	mount_namespaces(7)	Mount points
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic
User	CLONE_NEWUSER	user_namespace	
UTS	CLONE_NEWUTS	uts_namespaces	



ubuntu	1B+ · ★ 10K+
websphere-liberty	10M+ · ★ 294
open-liberty	10M+ · ★ 61

ubuntu

Tag: latest — Pull — Run

DOCKER OFFICIAL IMAGE · 1B+ · ★ 10K+ View on Hub

Updated 17 days ago

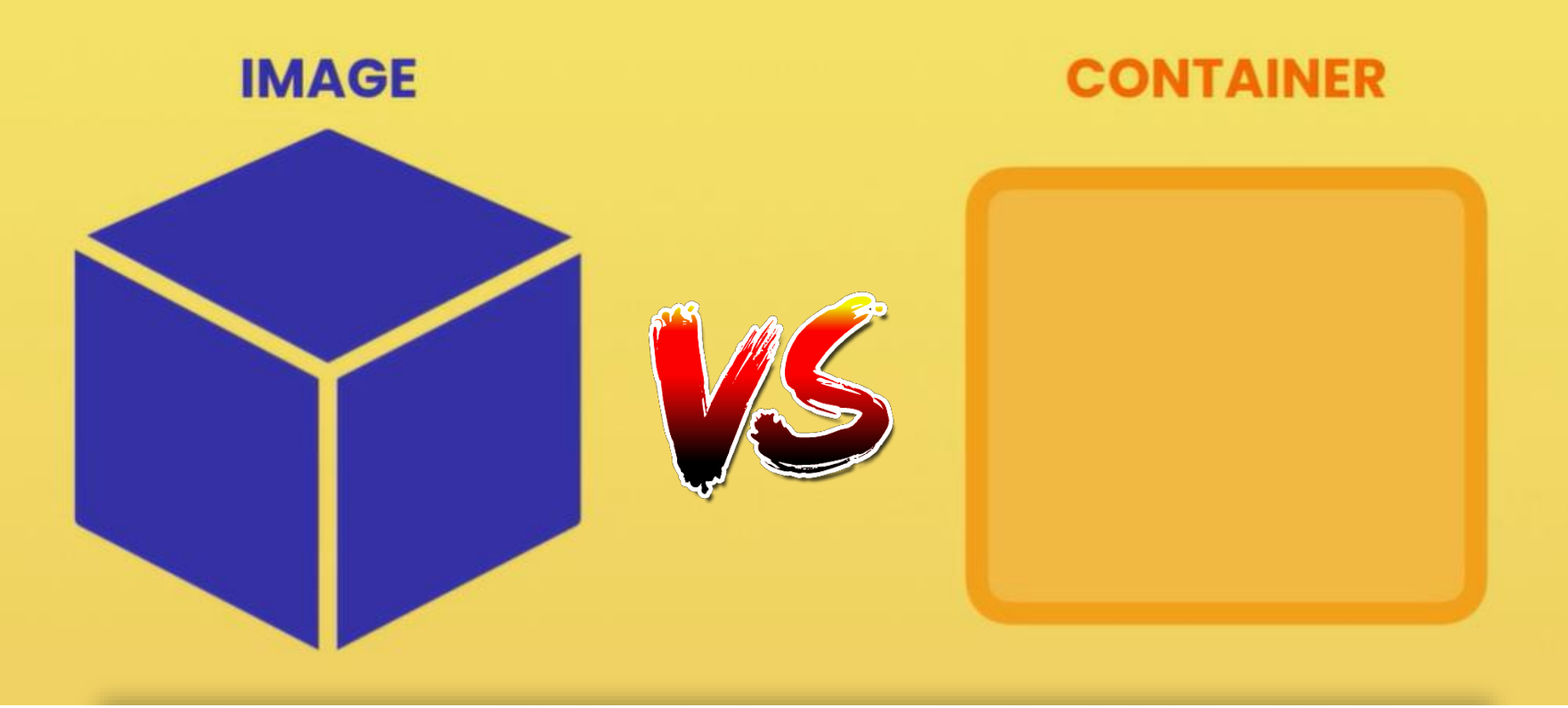
```
docker pull ubuntu
```

Ubuntu is a Debian-based Linux operating system based on free software.

Quick reference

- Maintained by:

Use ↑ ↓ or up and down arrow keys to navigate between results   Next tip   Give feedback



what an image is? How it is different from a container?

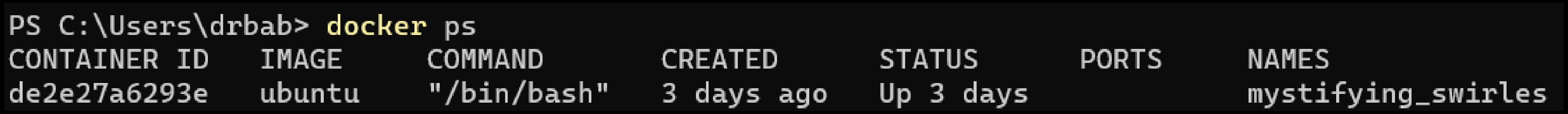
- A cut-down OS
- Third-party libraries
- Application files
- Environment variables

An image includes everything an application needs to run!
(all the files and configuration settings)

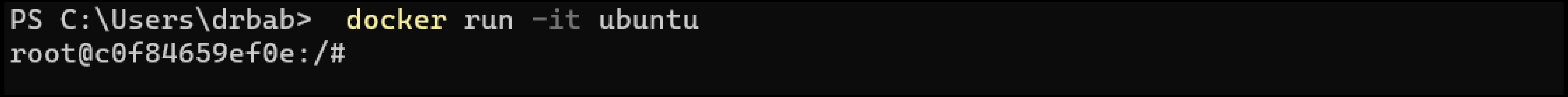
- Provides an isolated environment
- Can be stopped & restarted
- Is just a process!

A container provides an isolated environment for executing an application. It is a special process whose filesystem is provided by the Image

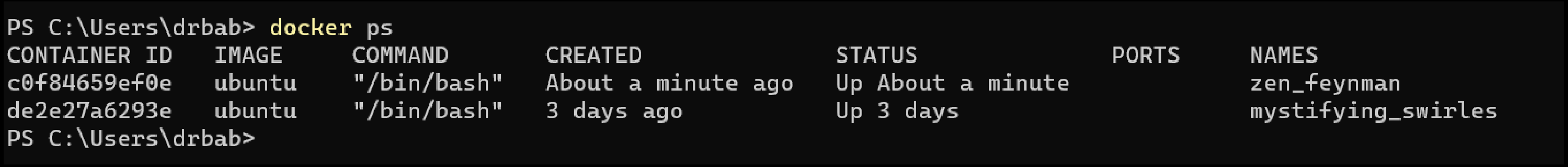
Start two containers from one image



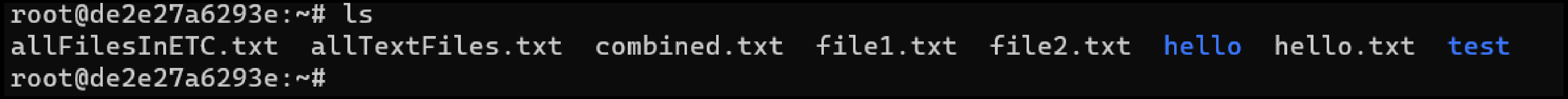
```
PS C:\Users\drbab> docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED      STATUS      PORTS     NAMES
de2e27a6293e   ubuntu    "/bin/bash"   3 days ago   Up 3 days             mystifying_swirles
```



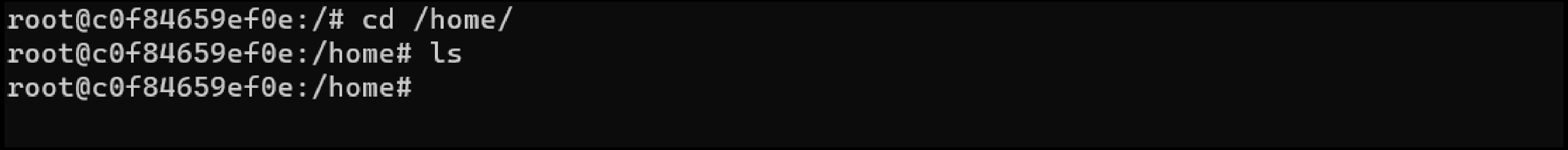
```
PS C:\Users\drbab>  docker run -it ubuntu
root@c0f84659ef0e:/#
```



```
PS C:\Users\drbab> docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS              PORTS     NAMES
c0f84659ef0e   ubuntu    "/bin/bash"   About a minute ago   Up About a minute             zen_feynman
de2e27a6293e   ubuntu    "/bin/bash"   3 days ago           Up 3 days                     mystifying_swirles
PS C:\Users\drbab>
```



```
root@de2e27a6293e:~# ls
allFilesInETC.txt  allTextFiles.txt  combined.txt  file1.txt  file2.txt  hello  hello.txt  test
root@de2e27a6293e:~#
```

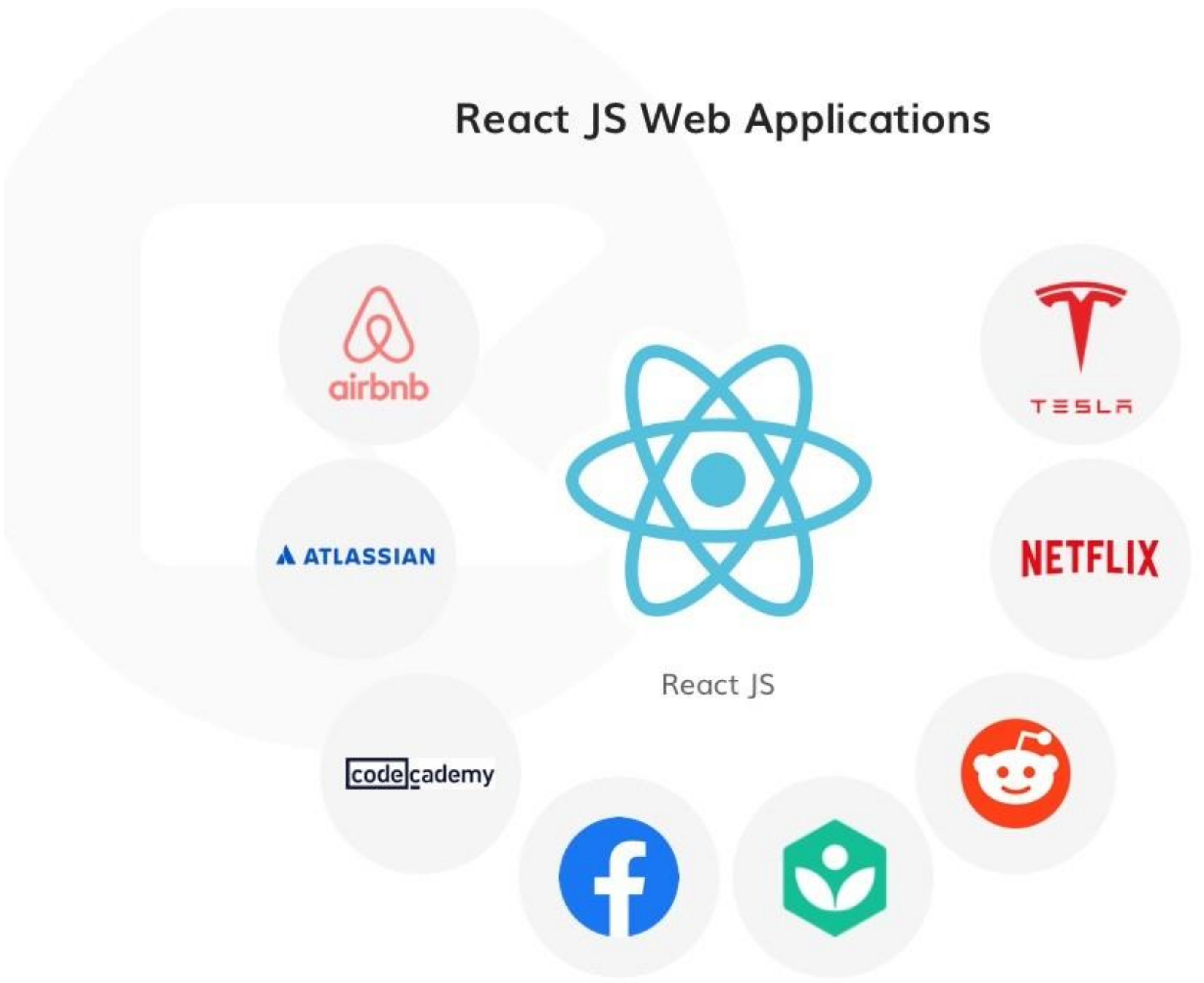


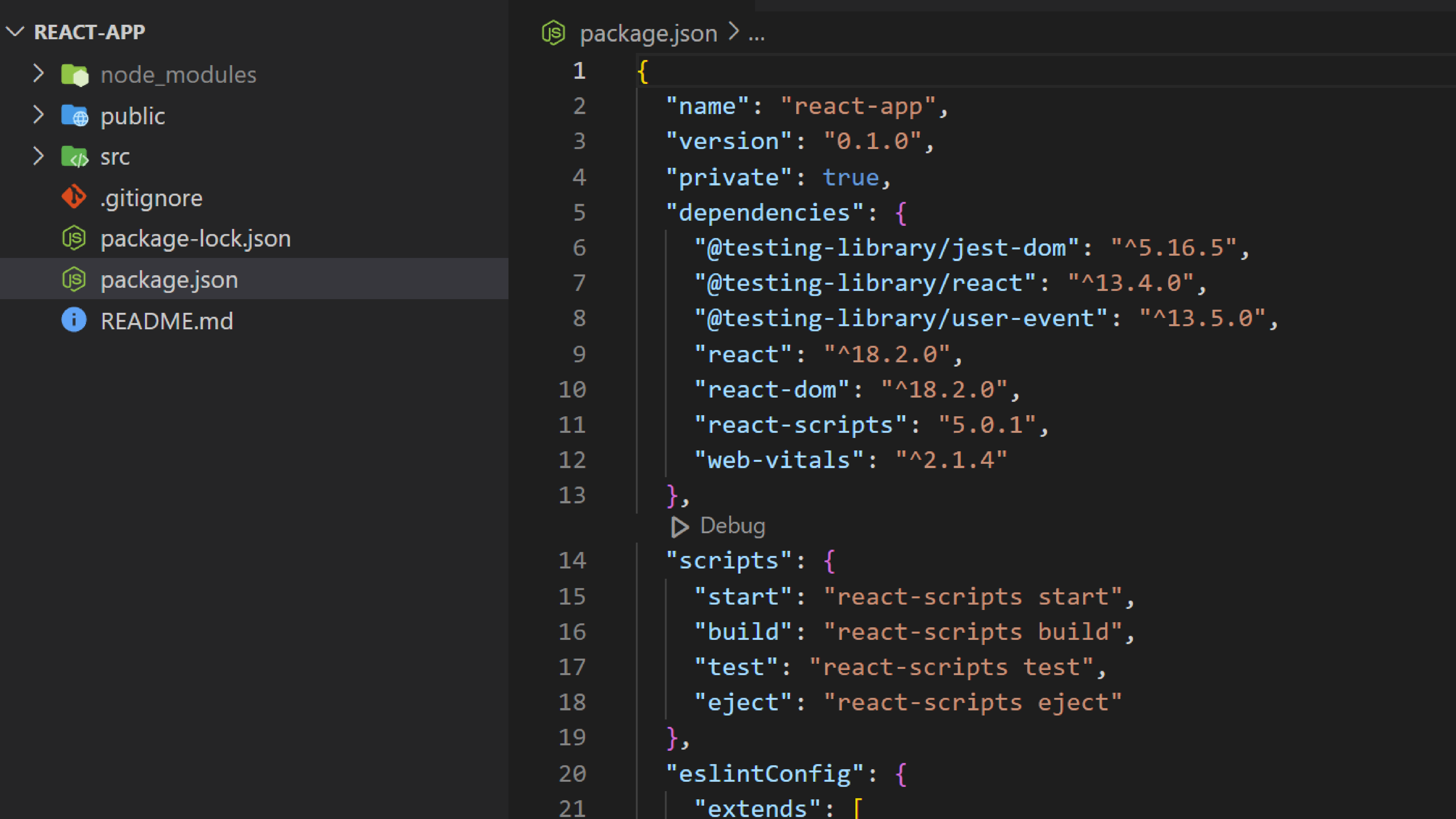
```
root@c0f84659ef0e:/# cd /home/
root@c0f84659ef0e:/home# ls
root@c0f84659ef0e:/home#
```

By default, containers don’t share a file system.

CONTAINER
CONTAINER
CONTAINER

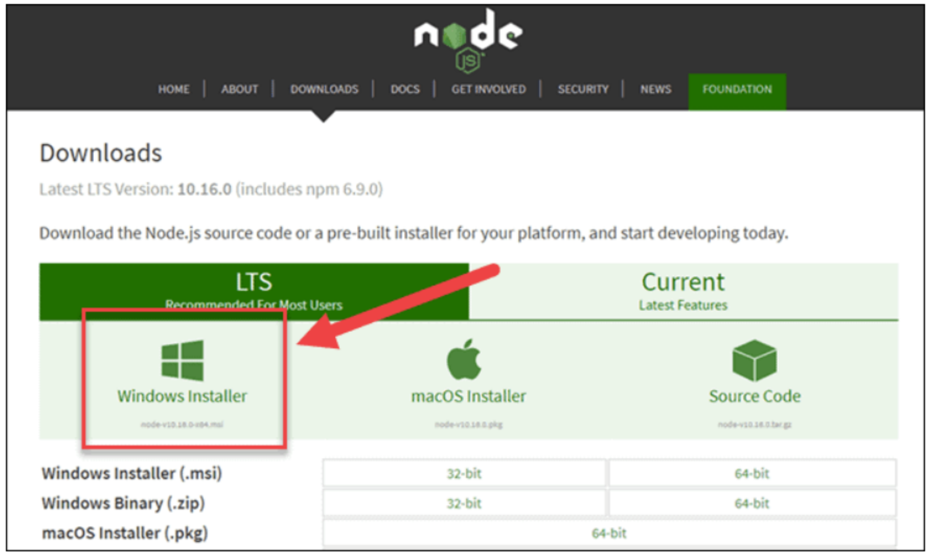
Let’s package a React application into a docker image





```json
{
  "name": "react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
```

First install Node to have npm package manager and install the dependencies of this application


HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Downloads

Latest LTS Version: **10.16.0** (includes npm 6.9.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	

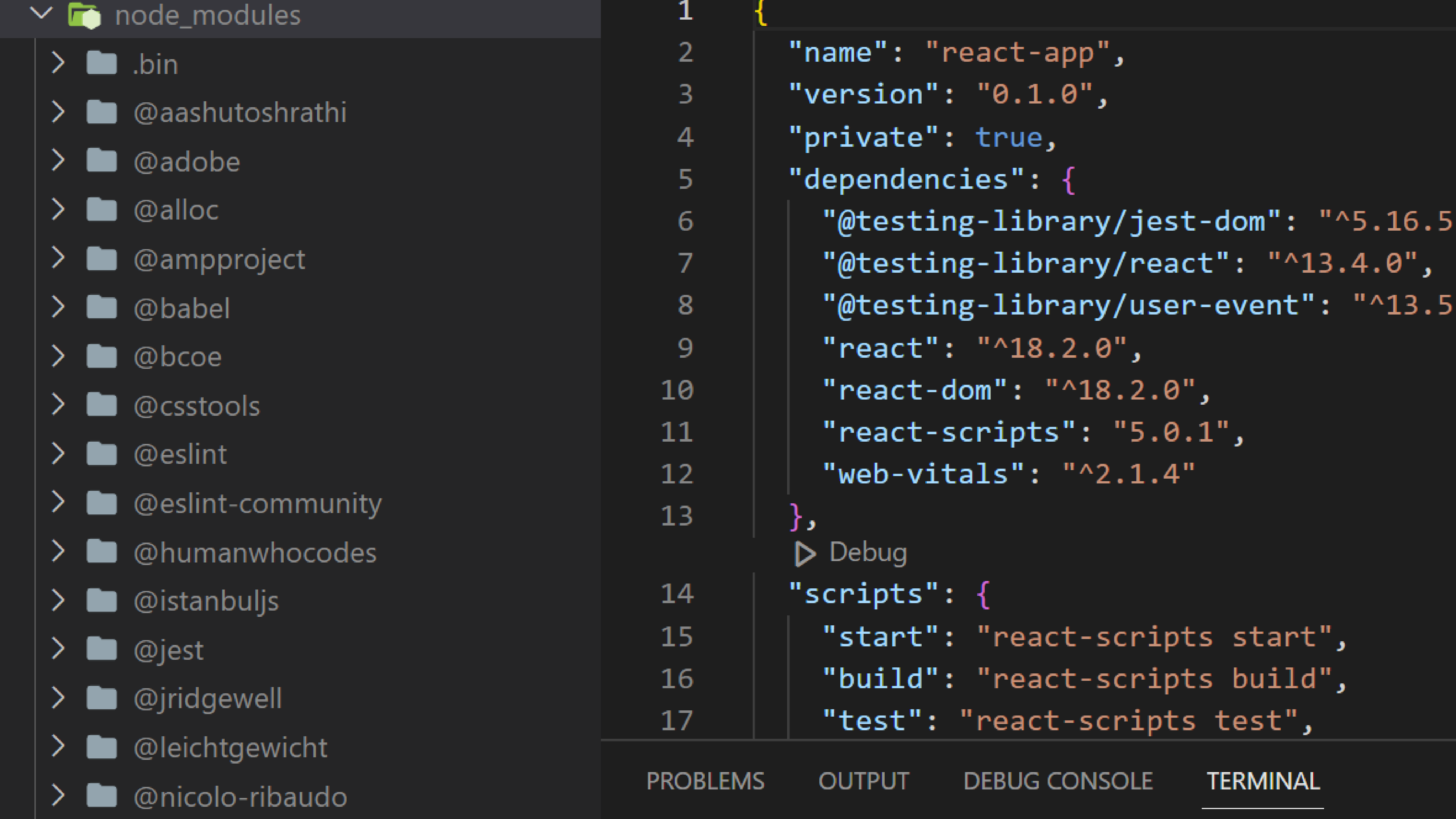
> node -v

Then we need to automatically download and install all the dependencies

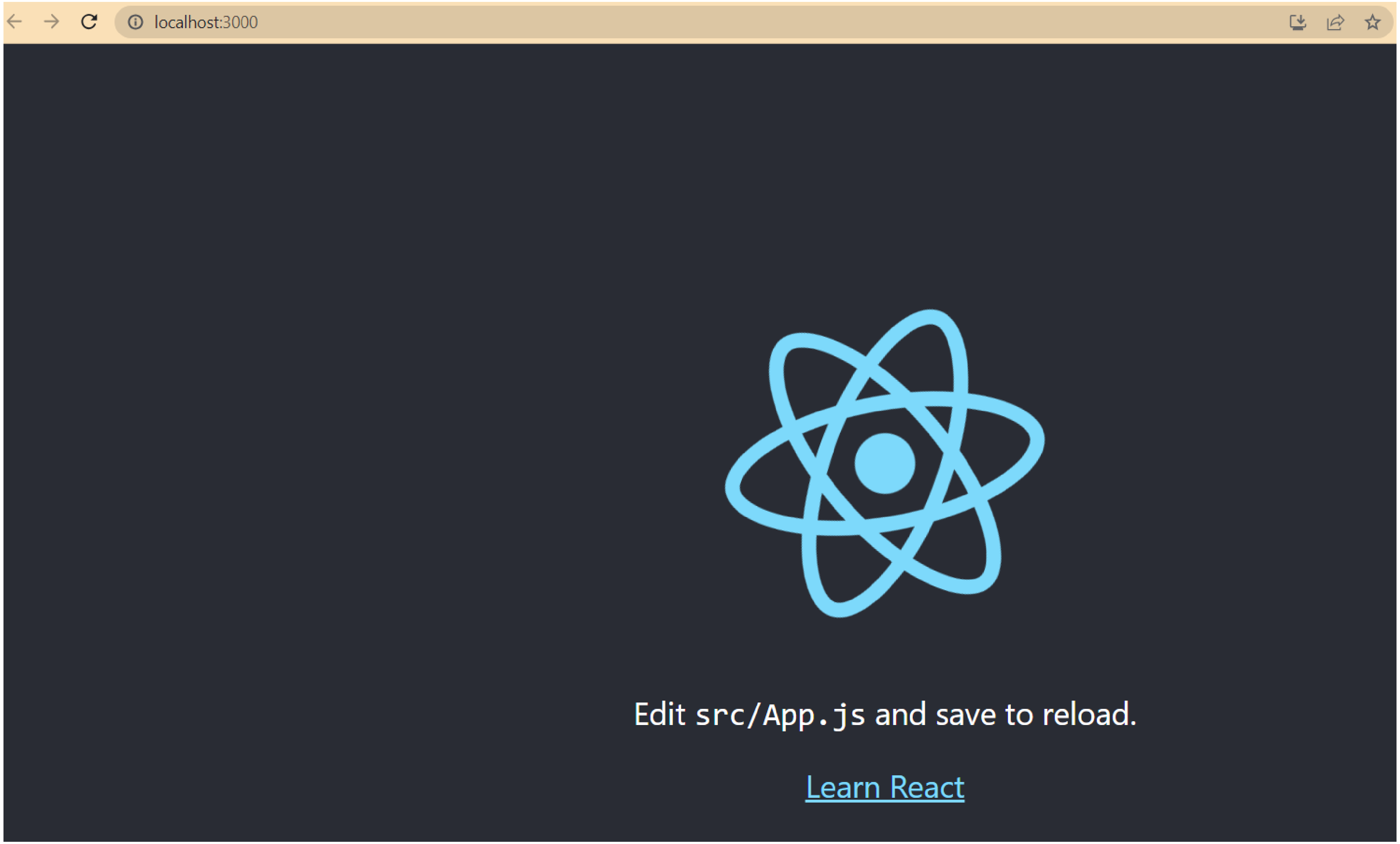
> npm install

Finally start the project

> npm run start



```
{
  "name": "react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
```



Edit `src/App.js` and save to reload.

Learn React

A Dockerfile contains instructions for building an image

- *FROM: to specify the base image. We take this base image that contains some files or directories and build on top of it*

- *WORTDIR: to specify the working directory. Once we define this all the other commands will be executed in the current working directory*

- *COPY and ADD*

- *RUN: to execute the OS commands*

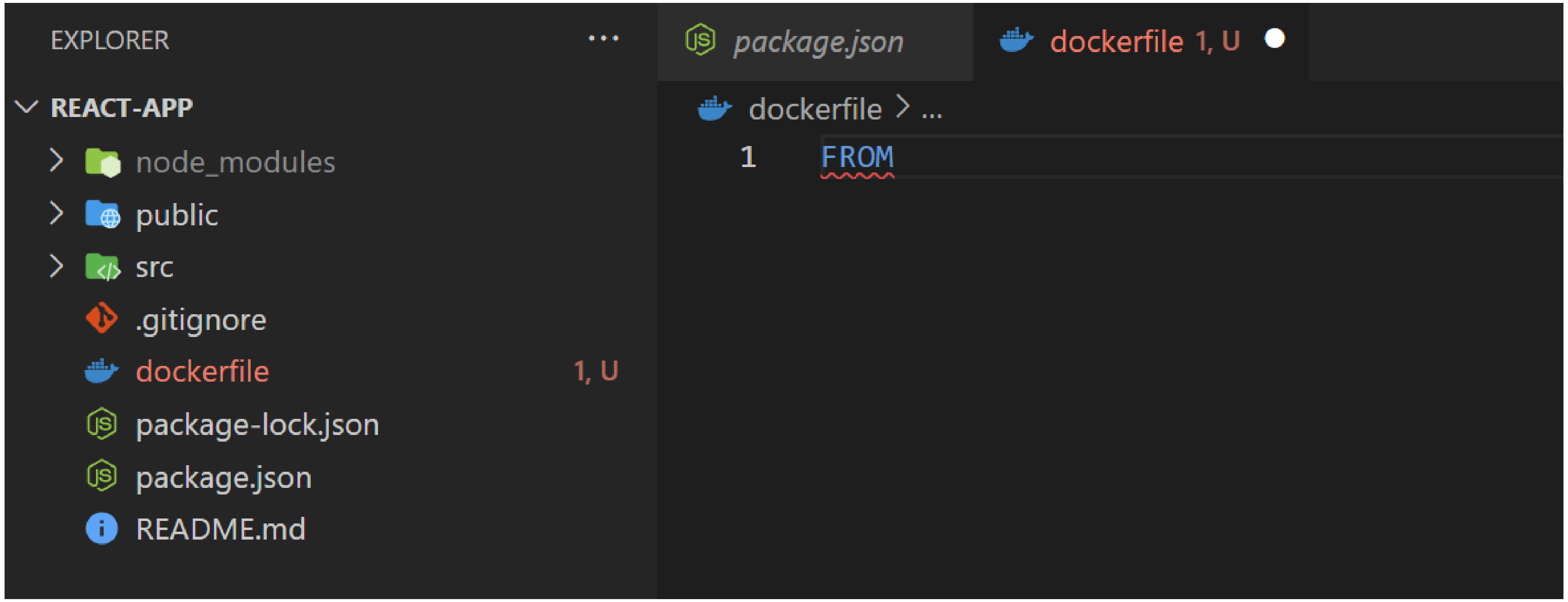
- *ENV: to set environmental variables*

- *EXPOSE: to tell the docker our container is starting on a given port*

- *USER: to specify the user that should run the application (typically user with limit privileges)*

- *CMD: the command that should be executed we when start the container*

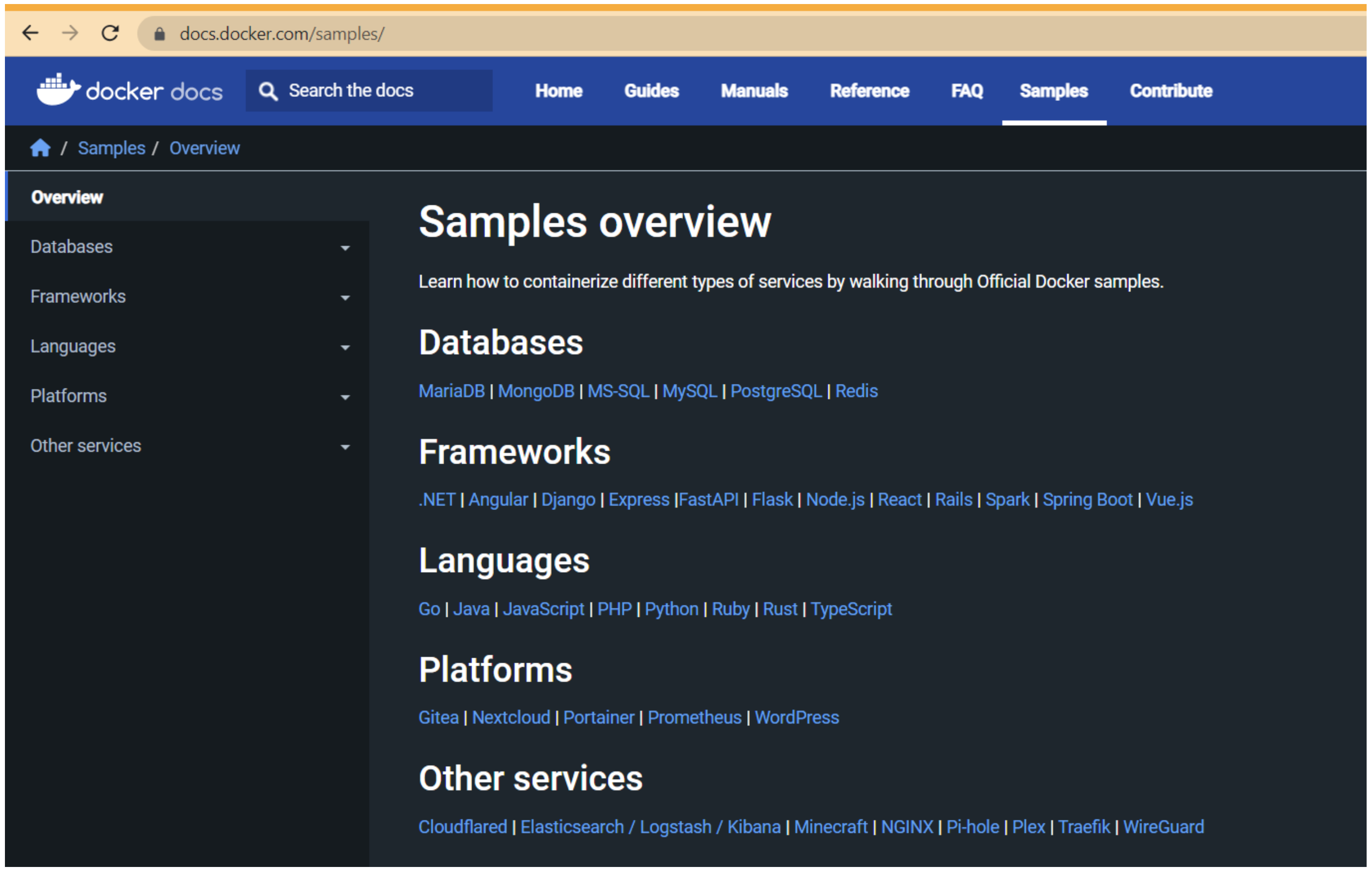
- *ENTRYPOINT: It is used to configure the executables that will always run after the container is initiated*



```
FROM
```

- *A base image can be an OS, e.g., Linux or Windows, or an OS + Runtime environment, e.g., .NET on Windows 11, Node on Ubuntu.*

- https://docs.docker.com/samples/
- *Docker samples for different purposes/OS/CPU ARC*



Samples overview

Learn how to containerize different types of services by walking through Official Docker samples.

Databases

MariaDB | MongoDB | MS-SQL | MySQL | PostgreSQL | Redis

Frameworks

.NET | Angular | Django | Express |FastAPI | Flask | Node.js | React | Rails | Spark | Spring Boot | Vue.js

Languages

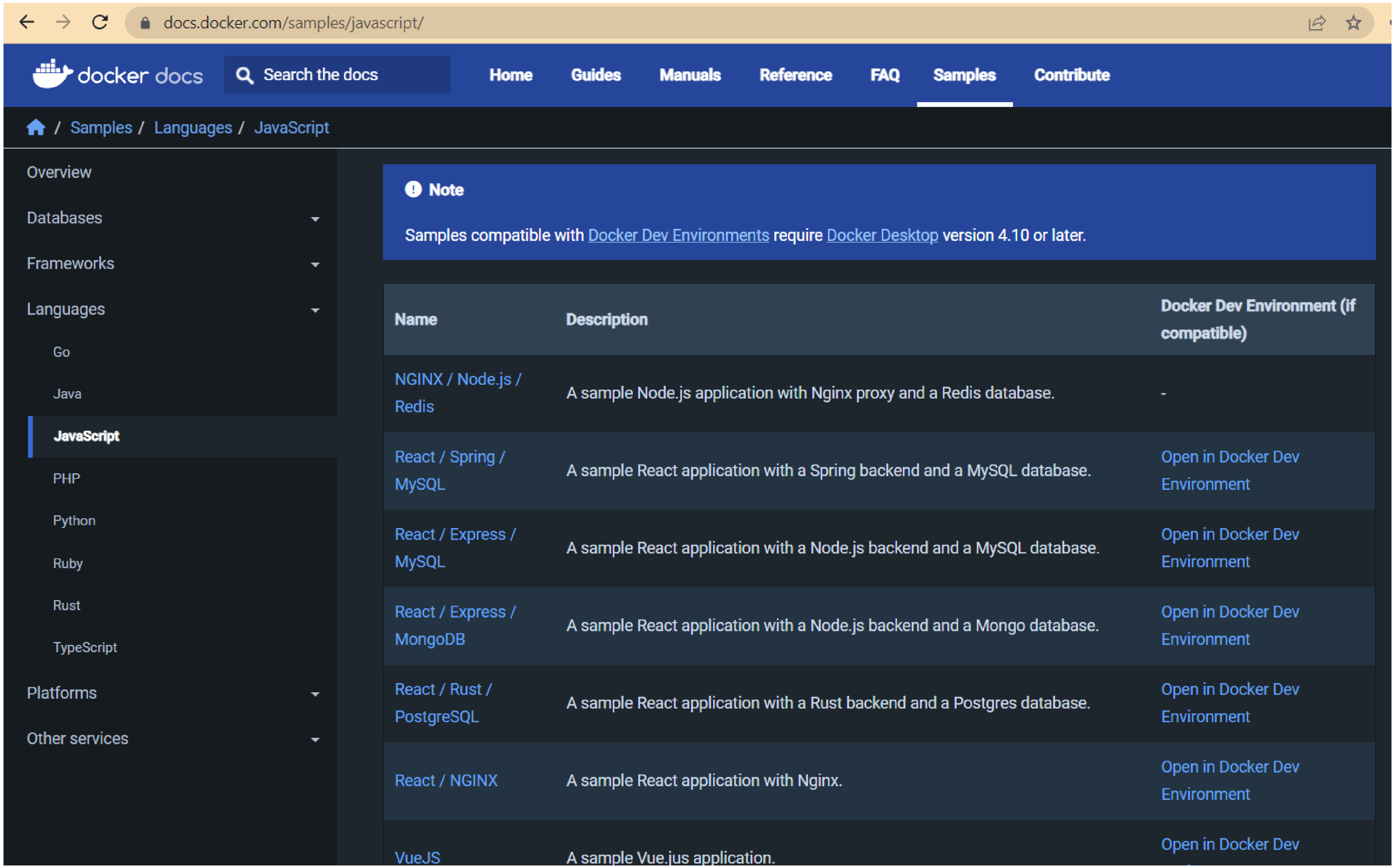
Go | Java | JavaScript | PHP | Python | Ruby | Rust | TypeScript

Platforms

Gitea | Nextcloud | Portainer | Prometheus | WordPress

Other services

Cloudflared | Elasticsearch / Logstash / Kibana | Minecraft | NGINX | Pi-hole | Plex | Traefik | WireGuard

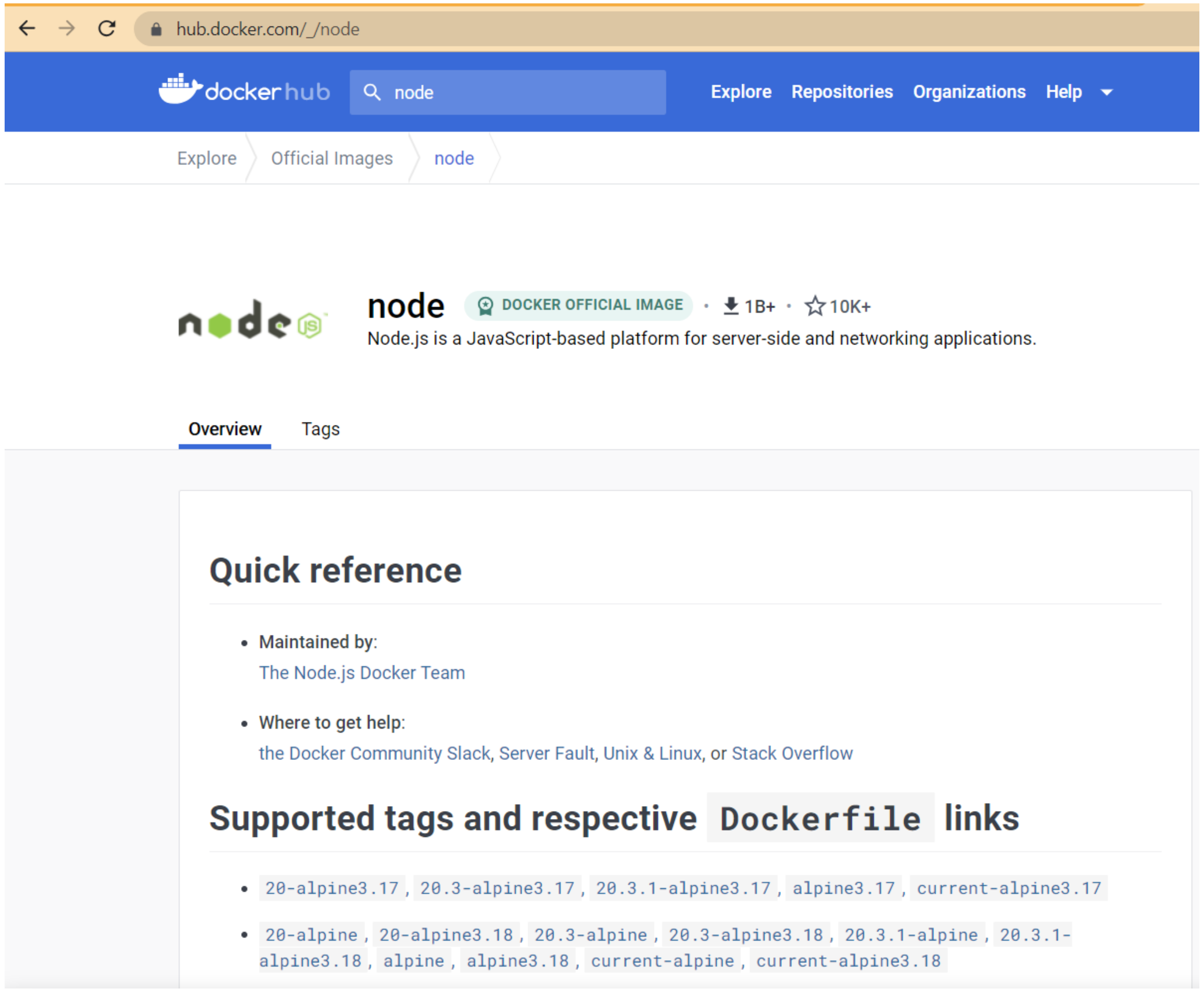

docs.docker.com/samples/javascript/

Samples / Languages / JavaScript

Note

Samples compatible with Docker Dev Environments require Docker Desktop version 4.10 or later.

Name	Description	Docker Dev Environment (if compatible)
NGINX / Node.js / Redis	A sample Node.js application with Nginx proxy and a Redis database.	-
React / Spring / MySQL	A sample React application with a Spring backend and a MySQL database.	Open in Docker Dev Environment
React / Express / MySQL	A sample React application with a Node.js backend and a MySQL database.	Open in Docker Dev Environment
React / Express / MongoDB	A sample React application with a Node.js backend and a Mongo database.	Open in Docker Dev Environment
React / Rust / PostgreSQL	A sample React application with a Rust backend and a Postgres database.	Open in Docker Dev Environment
React / NGINX	A sample React application with Nginx.	Open in Docker Dev Environment
VueJS	A sample Vue.jus application.	Open in Docker Dev


hub.docker.com/_/node

docker hub
node
Explore Repositories Organizations Help
Explore › Official Images › node

node DOCKER OFFICIAL IMAGE • 1B+ • 10K+

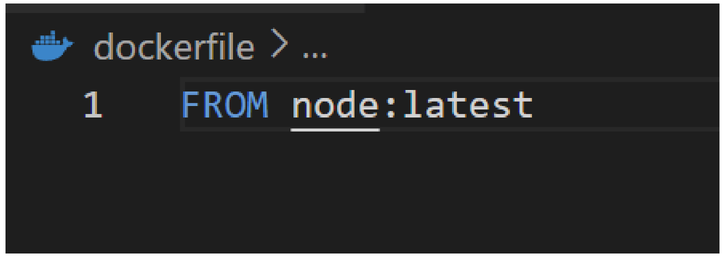
Node.js is a JavaScript-based platform for server-side and networking applications.

Quick reference

- Maintained by:
  The Node.js Docker Team
- Where to get help:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

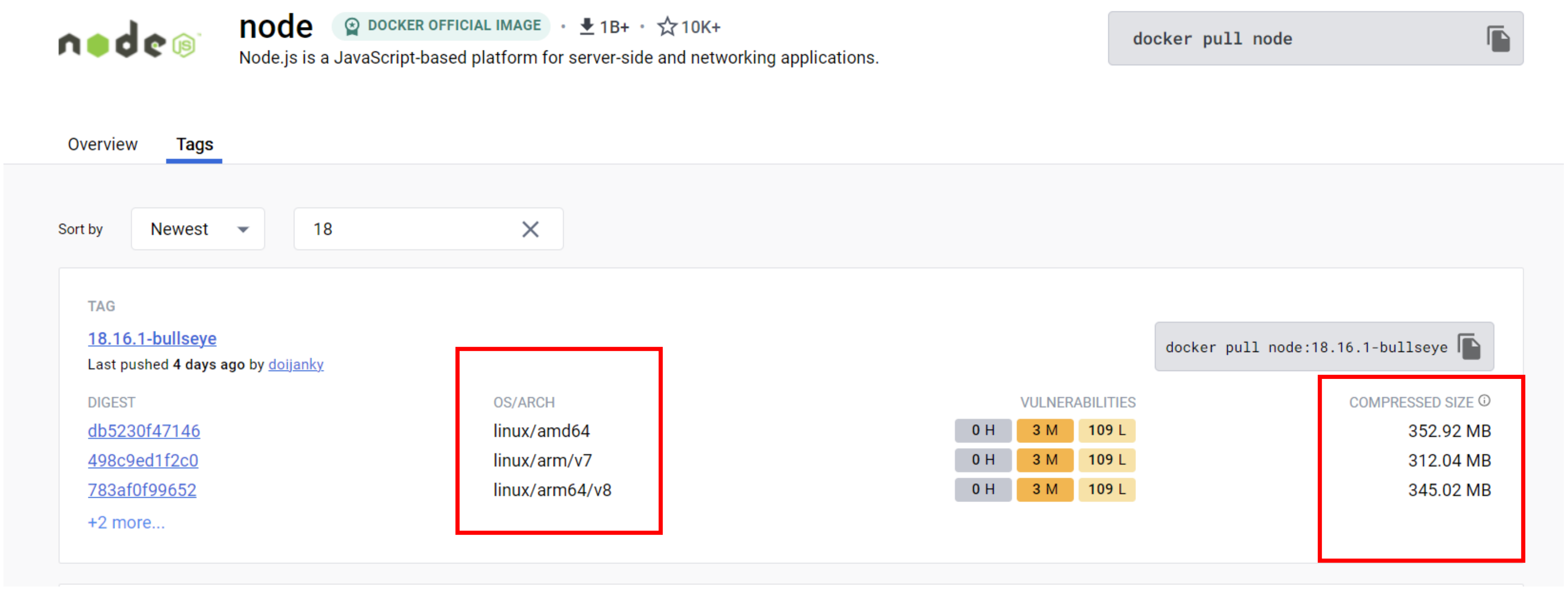
Supported tags and respective `Dockerfile` links

- `20-alpine3.17`, `20.3-alpine3.17`, `20.3.1-alpine3.17`, `alpine3.17`, `current-alpine3.17`
- `20-alpine`, `20-alpine3.18`, `20.3-alpine`, `20.3-alpine3.18`, `20.3.1-alpine`, `20.3.1-alpine3.18`, `alpine`, `alpine3.18`, `current-alpine`, `current-alpine3.18`



```
FROM node:latest
```

- *Your application will be built with different version of Node!*
- *The behavior of your app gets unpredictable*
- *Always use a specific version!*



Node.js is a JavaScript-based platform for server-side and networking applications.

```
docker pull node
```

18

TAG

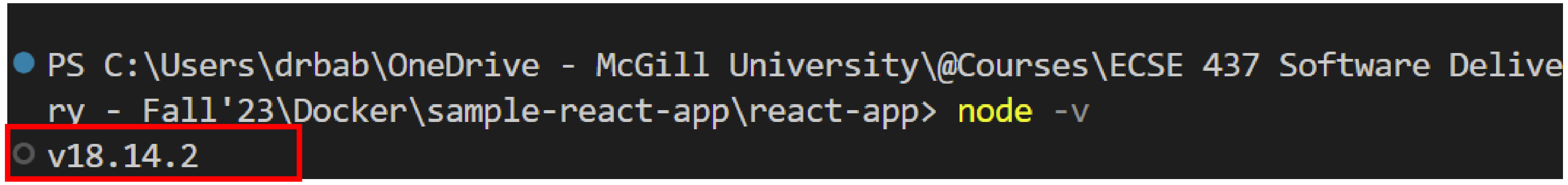
18.16.1-bullseye

Last pushed **4 days ago** by doijanky

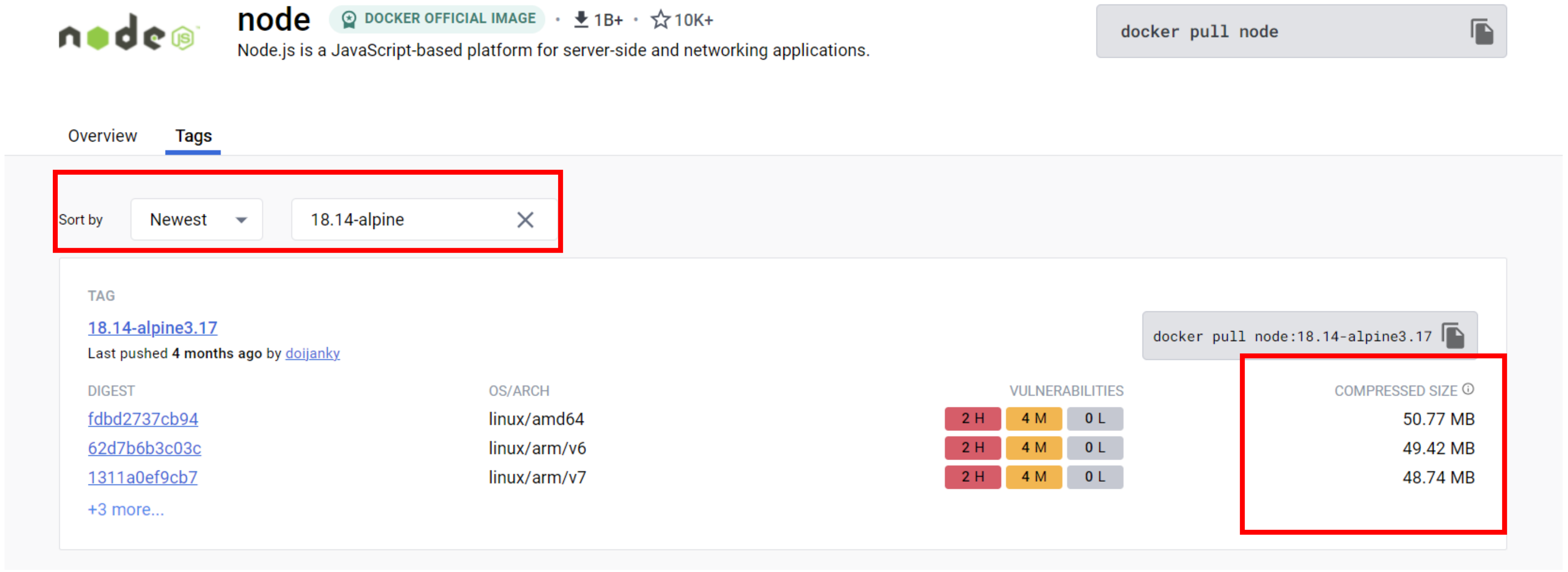
```
docker pull node:18.16.1-bullseye
```

DIGEST	OS/ARCH	VULNERABILITIES			COMPRESSED SIZE
db5230f47146	linux/amd64	0 H	3 M	109 L	352.92 MB
498c9ed1f2c0	linux/arm/v7	0 H	3 M	109 L	312.04 MB
783af0f99652	linux/arm64/v8	0 H	3 M	109 L	345.02 MB

+2 more...



```
PS C:\Users\drbab\OneDrive - McGill University\@Courses\ECSE 437 Software Delive
ry - Fall'23\Docker\sample-react-app\react-app> node -v
v18.14.2
```



node DOCKER OFFICIAL IMAGE · 1B+ · 10K+

Node.js is a JavaScript-based platform for server-side and networking applications.

```
docker pull node
```

Sort by: Newest — 18.14-alpine

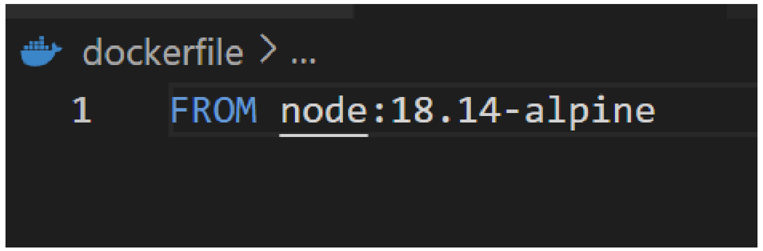
TAG

18.14-alpine3.17

Last pushed **4 months ago** by doijanky

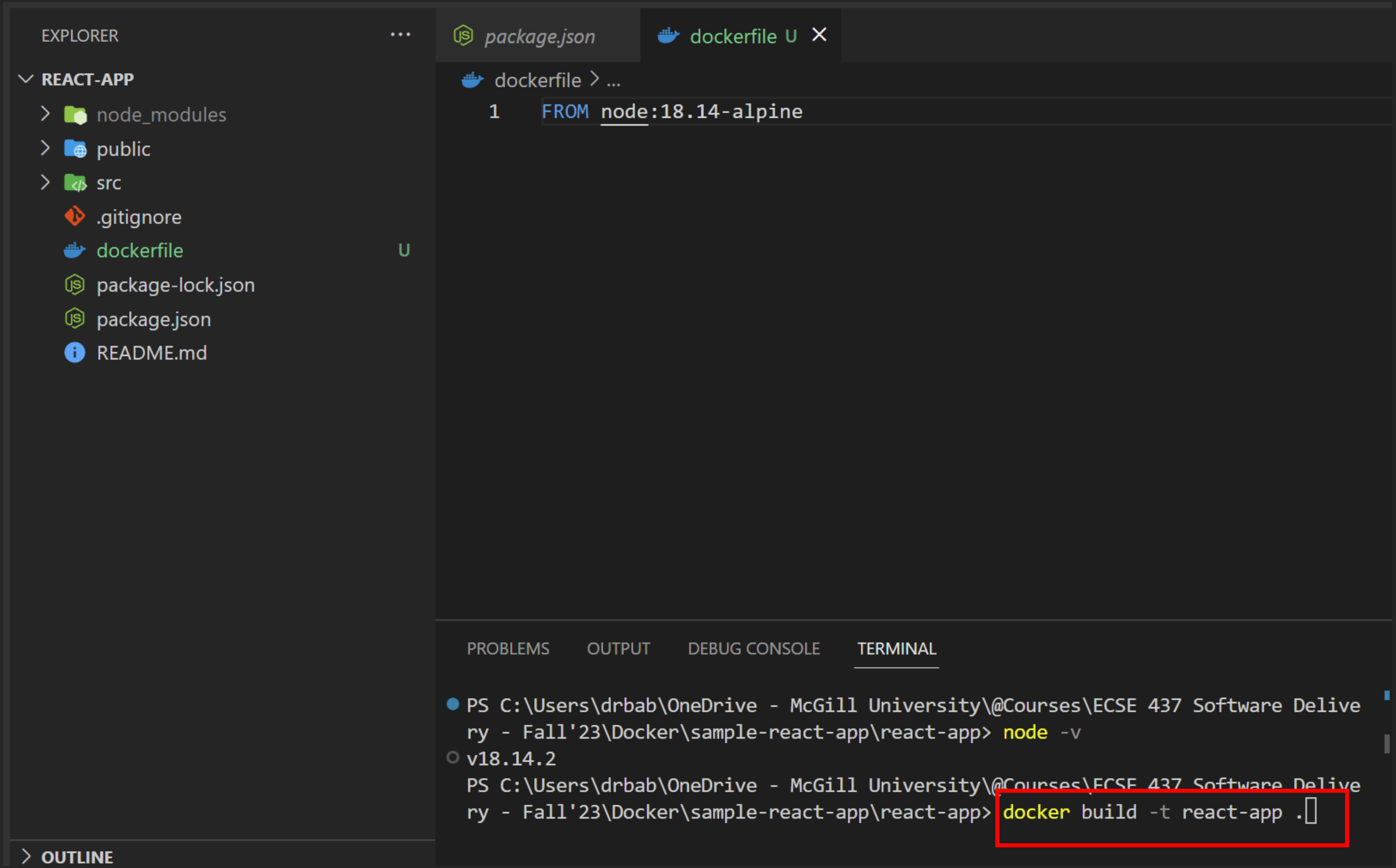
```
docker pull node:18.14-alpine3.17
```

DIGEST	OS/ARCH	VULNERABILITIES	COMPRESSED SIZE
fdbd2737cb94	linux/amd64	2 H 4 M 0 L	50.77 MB
62d7b6b3c03c	linux/arm/v6	2 H 4 M 0 L	49.42 MB
1311a0ef9cb7	linux/arm/v7	2 H 4 M 0 L	48.74 MB
+3 more...			



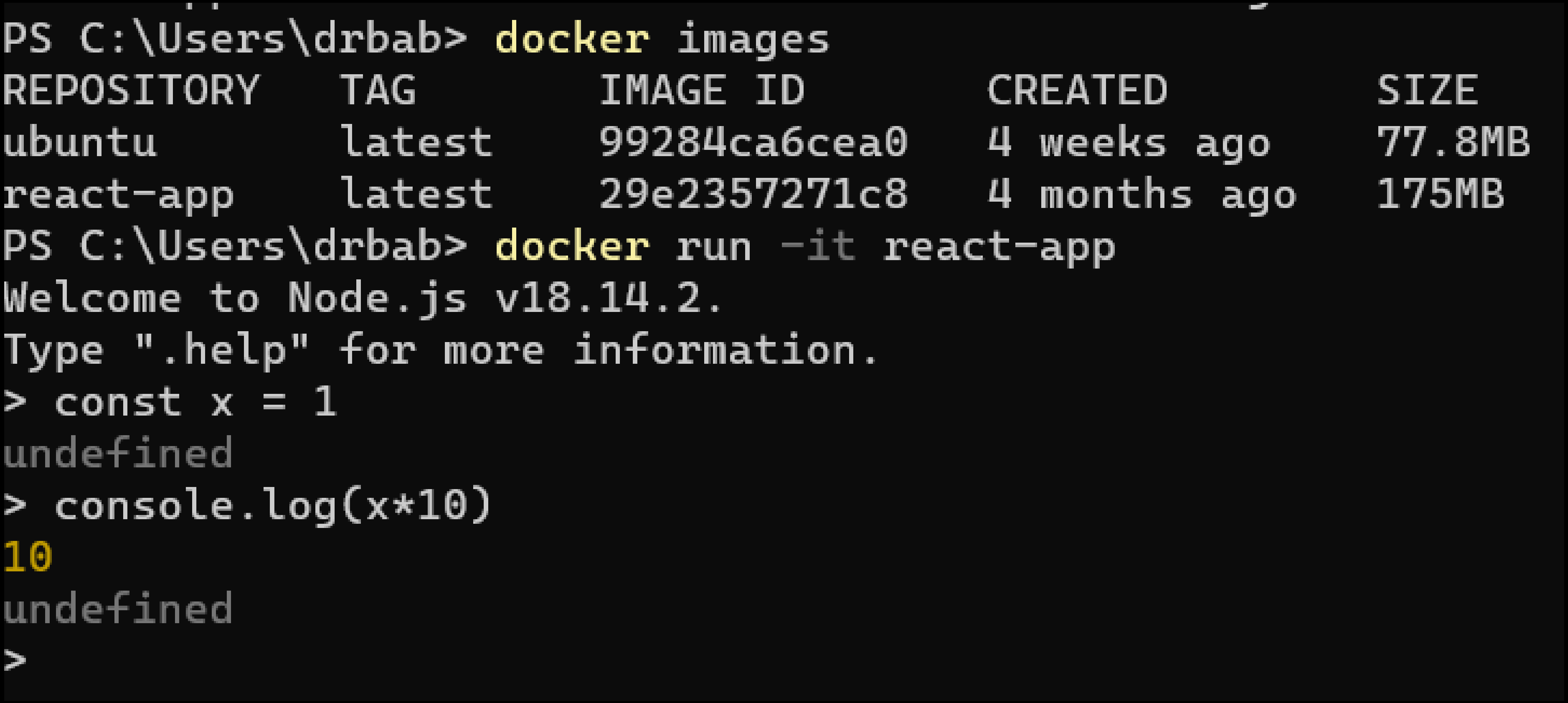
dockerfile > ...

```dockerfile
FROM node:18.14-alpine
```

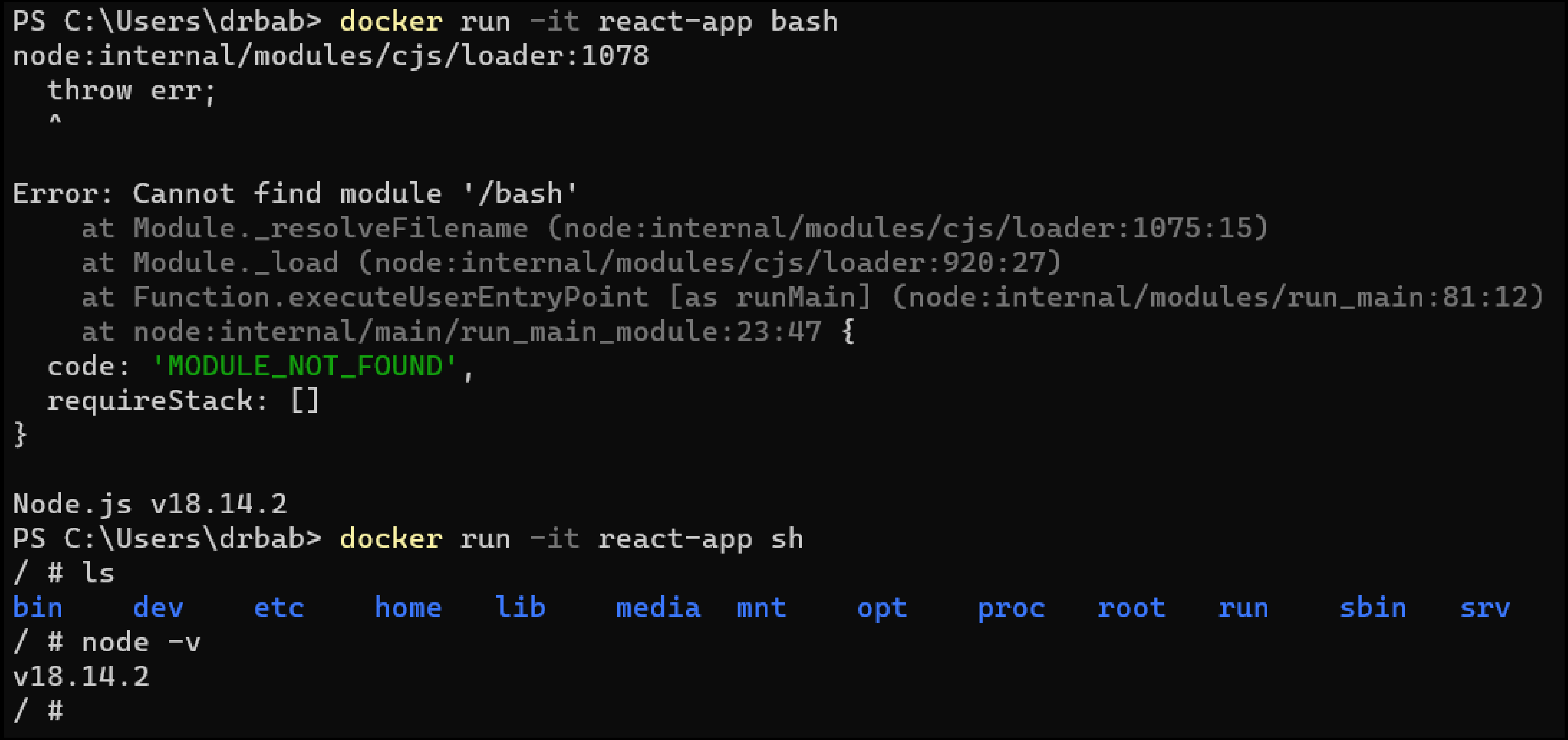


```
FROM node:18.14-alpine
```

```
PS C:\Users\drbab\OneDrive - McGill University\@Courses\ECSE 437 Software Delive
ry - Fall'23\Docker\sample-react-app\react-app> node -v
v18.14.2
PS C:\Users\drbab\OneDrive - McGill University\@Courses\ECSE 437 Software Delive
ry - Fall'23\Docker\sample-react-app\react-app> docker build -t react-app .
```

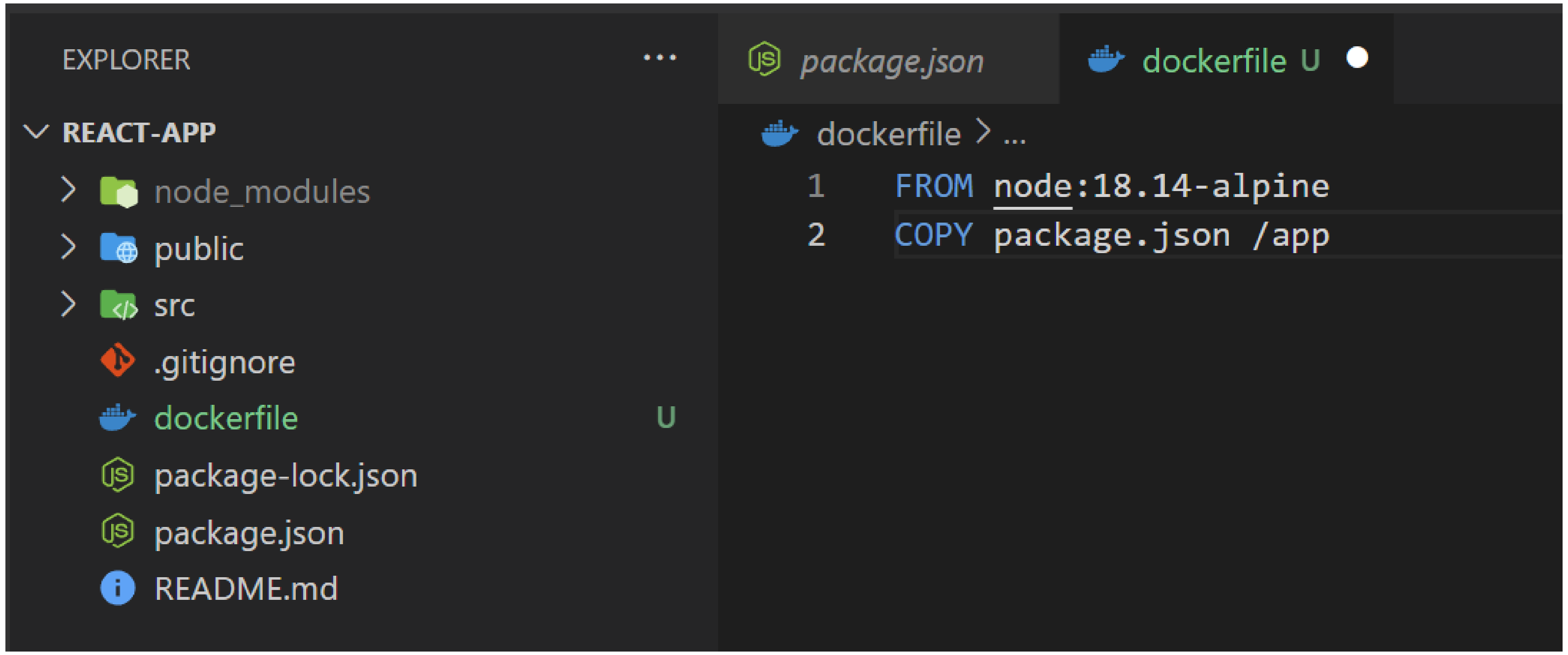


```
PS C:\Users\drbab> docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       latest    99284ca6cea0   4 weeks ago     77.8MB
react-app    latest    29e2357271c8   4 months ago    175MB
PS C:\Users\drbab> docker run -it react-app
Welcome to Node.js v18.14.2.
Type ".help" for more information.
> const x = 1
undefined
> console.log(x*10)
10
undefined
>
```

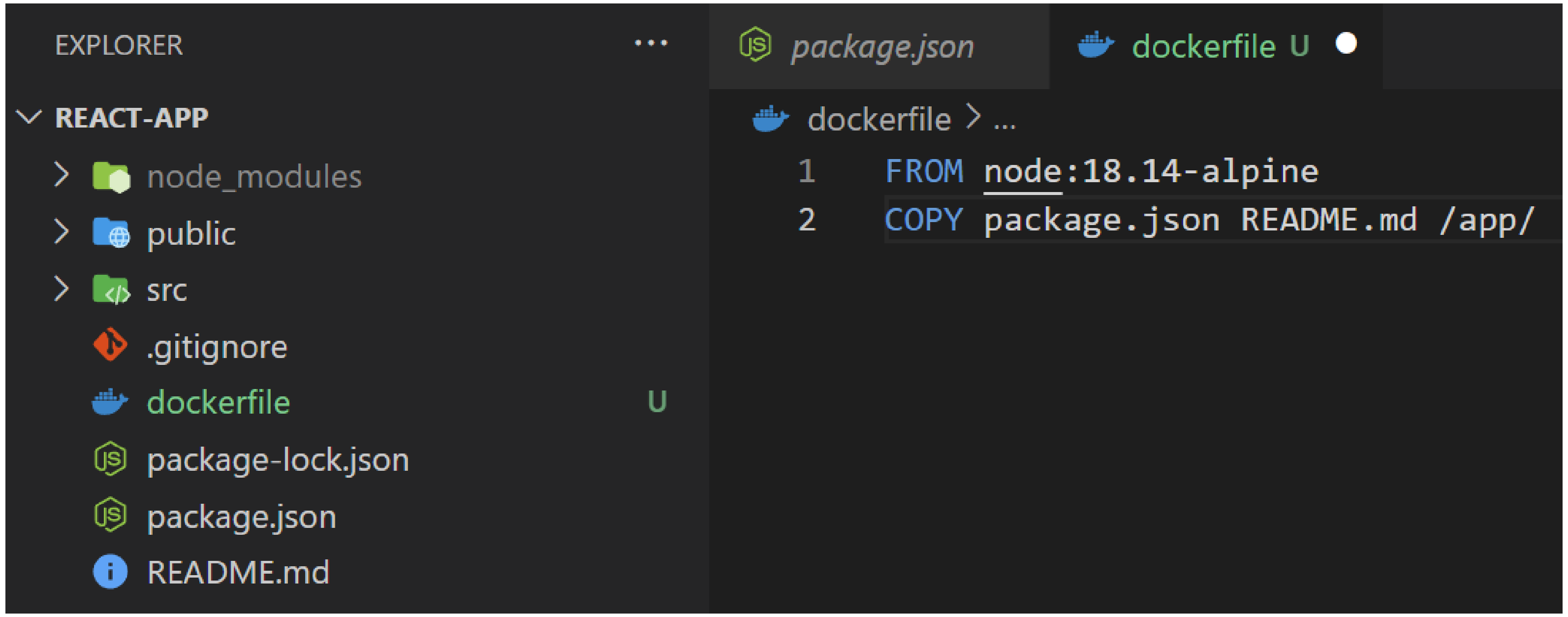


```
PS C:\Users\drbab> docker run -it react-app bash
node:internal/modules/cjs/loader:1078
  throw err;
  ^

Error: Cannot find module '/bash'
    at Module._resolveFilename (node:internal/modules/cjs/loader:1075:15)
    at Module._load (node:internal/modules/cjs/loader:920:27)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:23:47 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v18.14.2
PS C:\Users\drbab> docker run -it react-app sh
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv
/ # node -v
v18.14.2
/ #
```

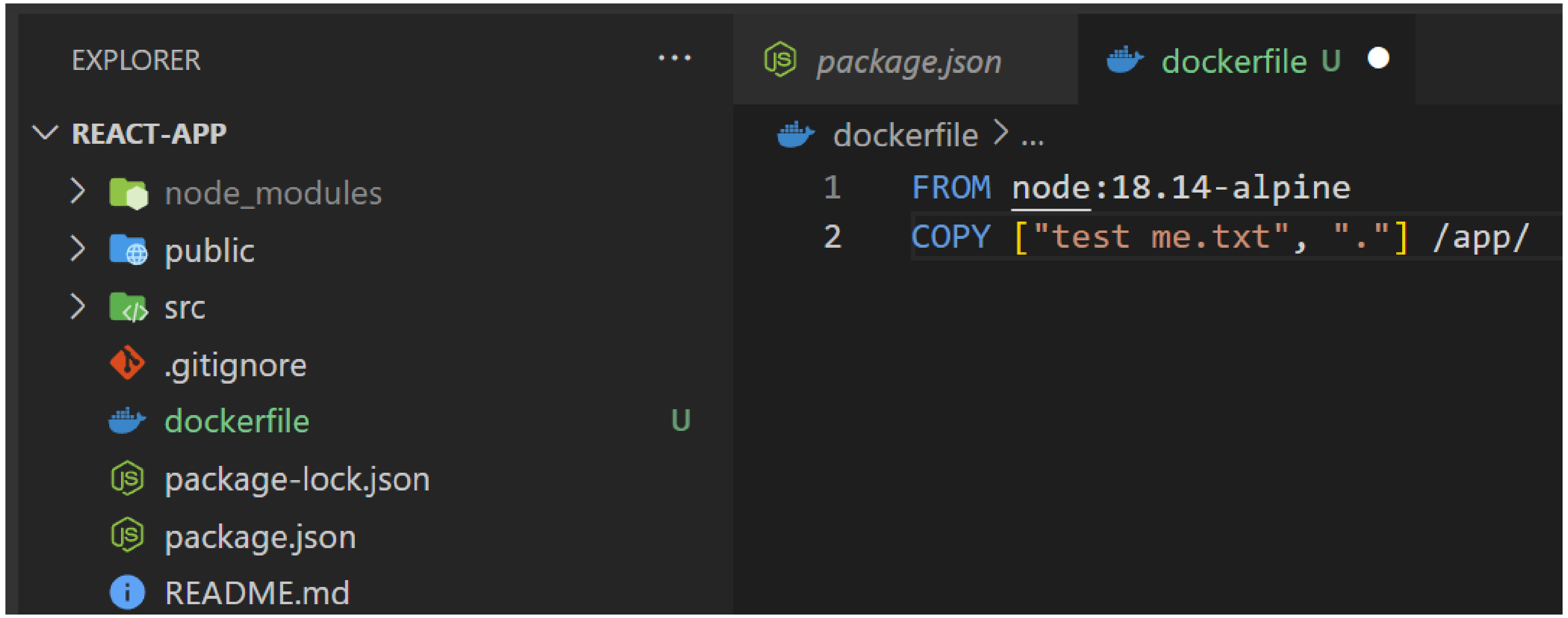
Now that we have the base image, the next step is to copy the application files into the image!



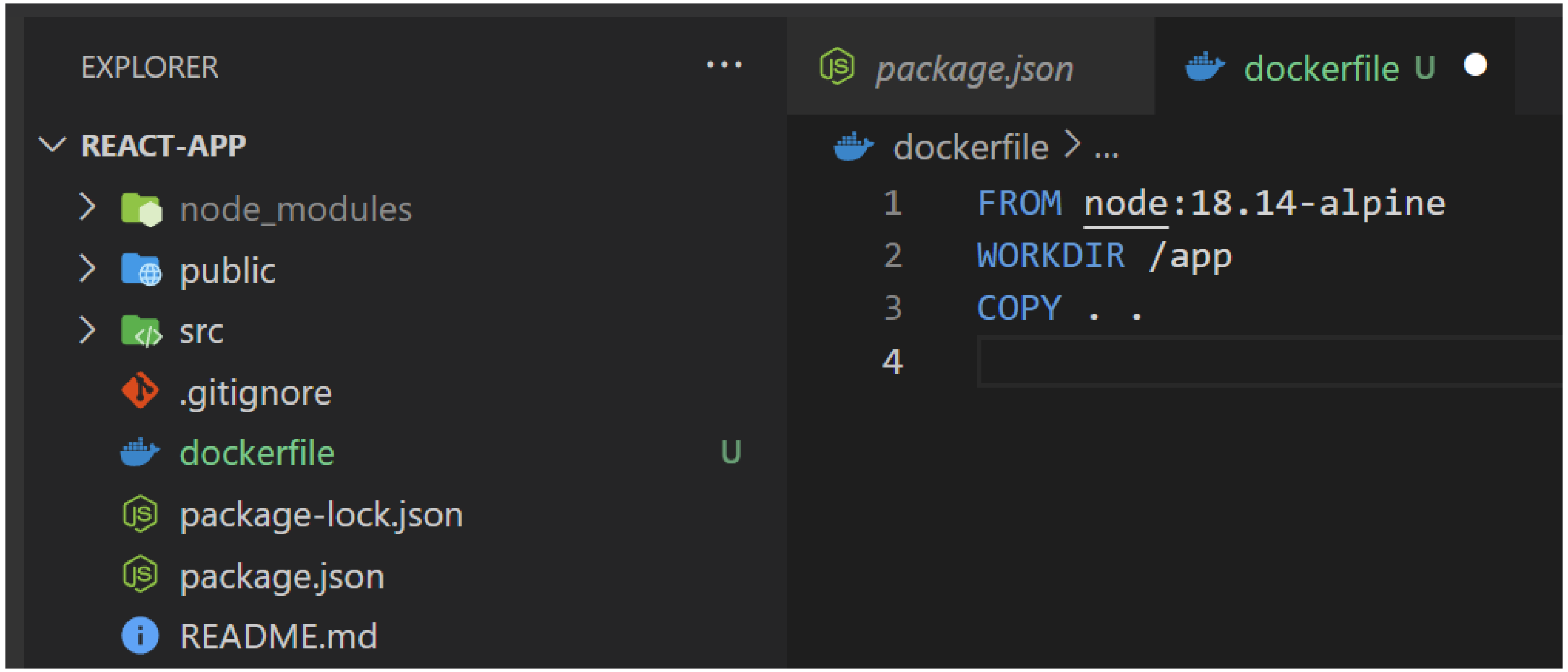
```
FROM node:18.14-alpine
COPY package.json /app
```



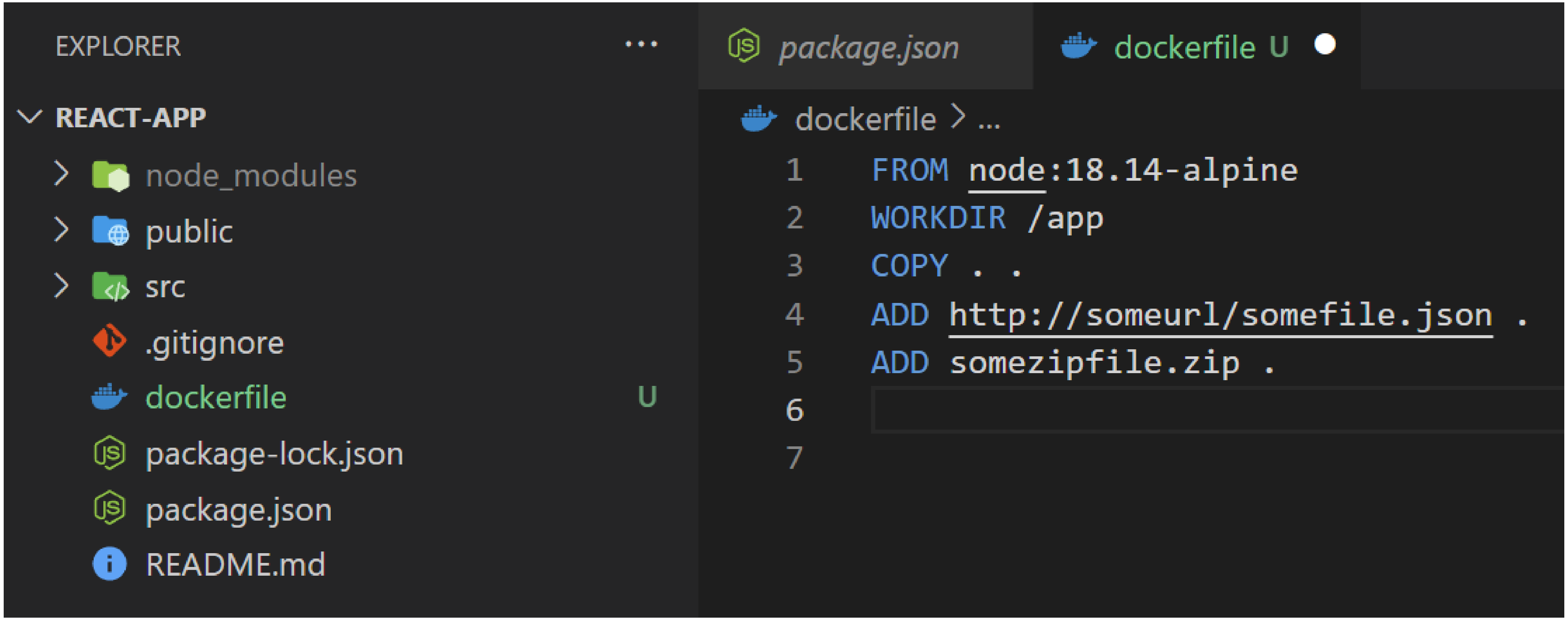
```
FROM node:18.14-alpine
COPY package.json README.md /app/
```



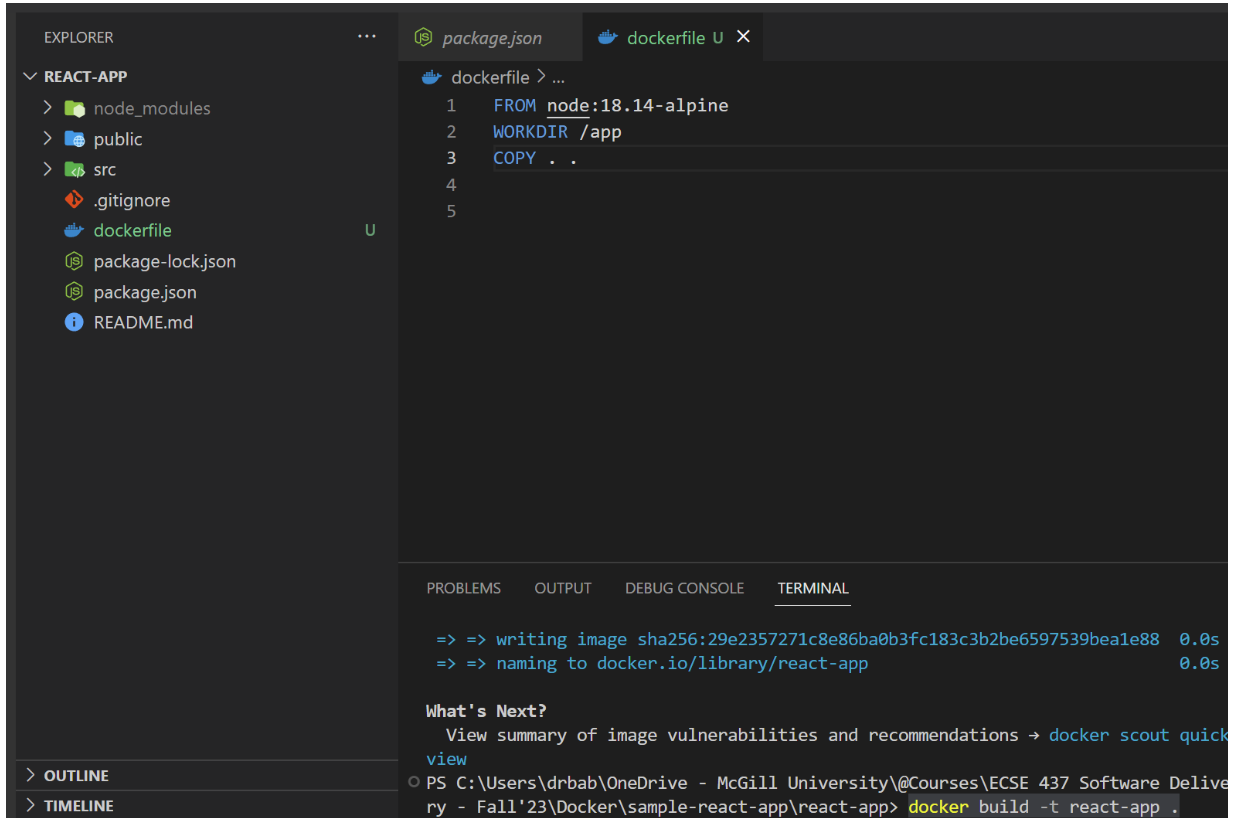
```
FROM node:18.14-alpine
COPY ["test me.txt", "."] /app/
```



```
FROM node:18.14-alpine
WORKDIR /app
COPY . .

```

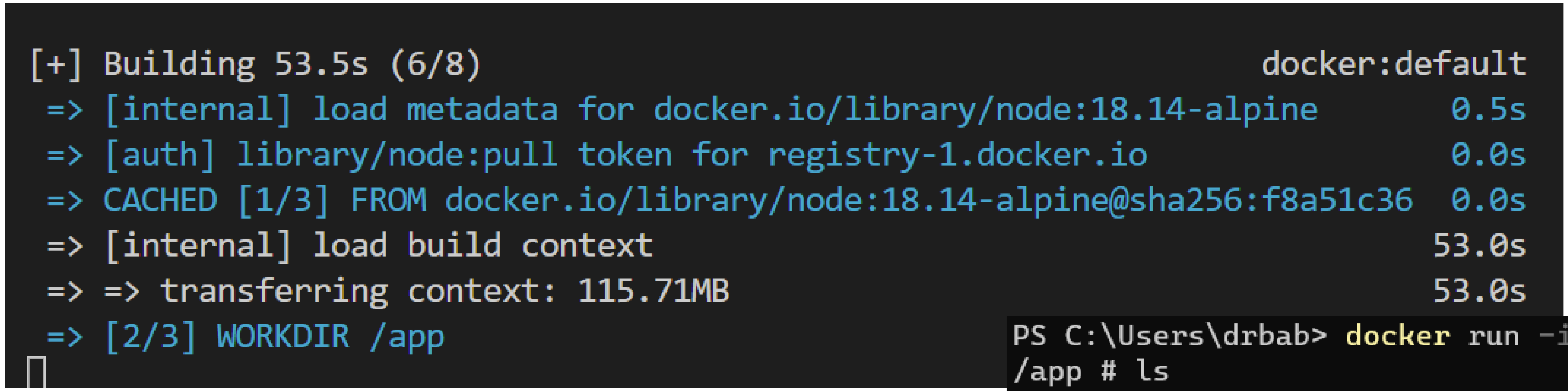


```
FROM node:18.14-alpine
WORKDIR /app
COPY . .
ADD http://someurl/somefile.json .
ADD somezipfile.zip .

```

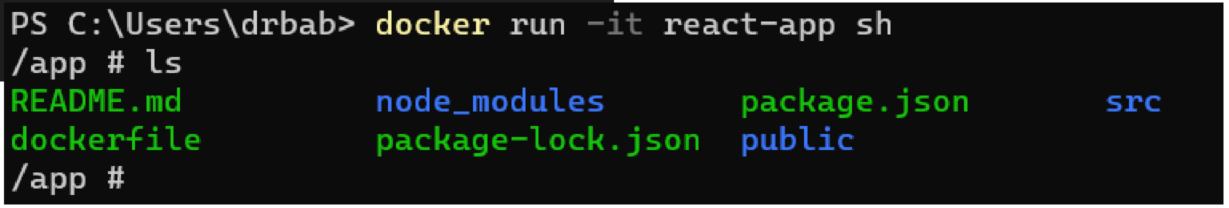


```
FROM node:18.14-alpine
WORKDIR /app
COPY . .
```

```
 => => writing image sha256:29e2357271c8e86ba0b3fc183c3b2be6597539bea1e88  0.0s
 => => naming to docker.io/library/react-app                               0.0s

What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quick
view
PS C:\Users\drbab\OneDrive - McGill University\@Courses\ECSE 437 Software Delive
ry - Fall'23\Docker\sample-react-app\react-app> docker build -t react-app .
```



```
[+] Building 53.5s (6/8)                                            docker:default
 => [internal] load metadata for docker.io/library/node:18.14-alpine          0.5s
 => [auth] library/node:pull token for registry-1.docker.io                   0.0s
 => CACHED [1/3] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36     0.0s
 => [internal] load build context                                            53.0s
 => => transferring context: 115.71MB                                        53.0s
 => [2/3] WORKDIR /app
```



```
PS C:\Users\drbab> docker run -it react-app sh
/app # ls
README.md           node_modules        package.json        src
dockerfile          package-lock.json   public
/app #
```

- When we include all files using “docker build -t react-app .”, docker client take everything in the current directory (i.e., context directory or build context)

- Docker client send everything to docker engine or docker daemon

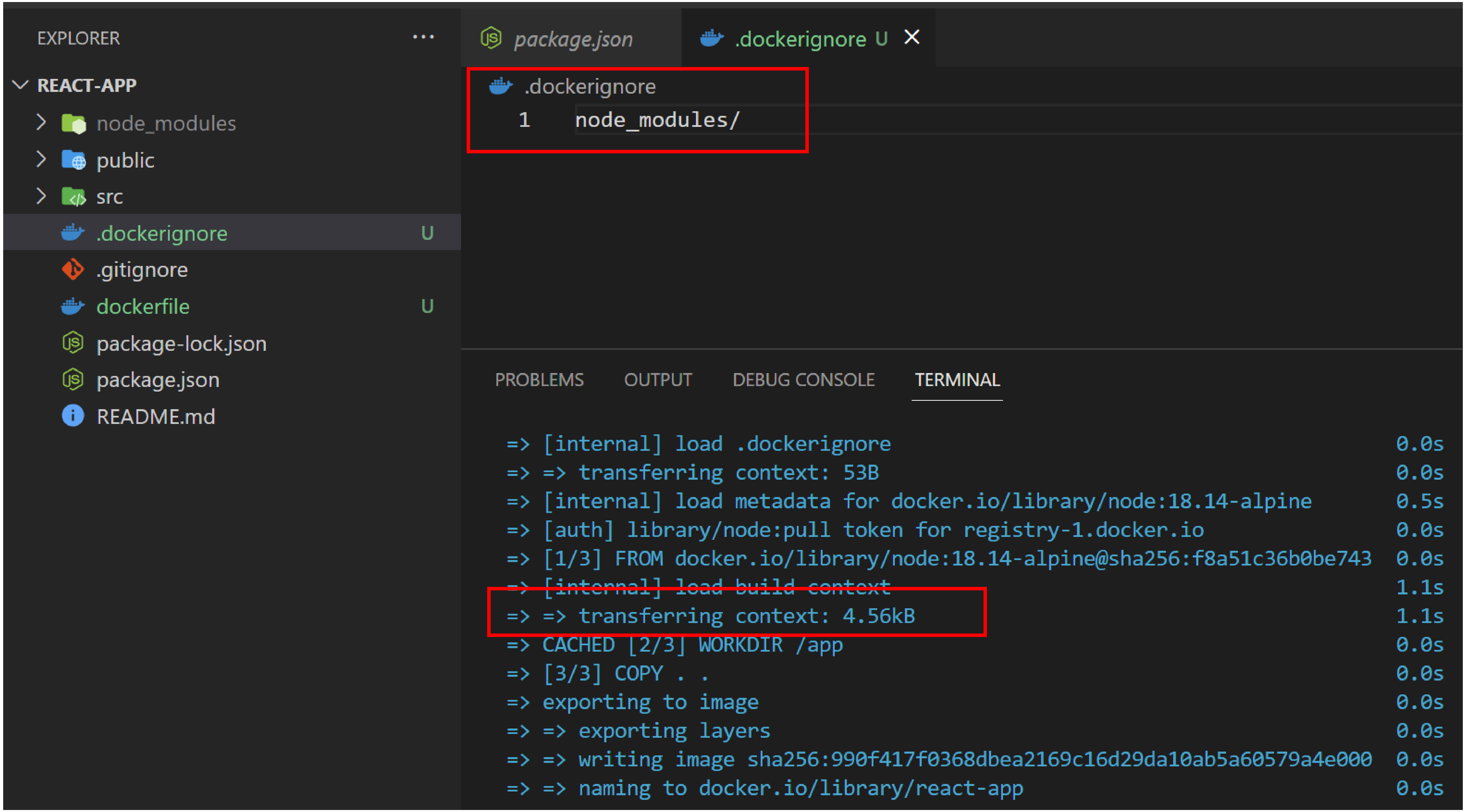
- For a very simple app with no feature the size of build context is about 150 MB

- In deployment, docker client will talk to the docker engine on a different machine

- Whatever we have in the build context has to be transferred over the network

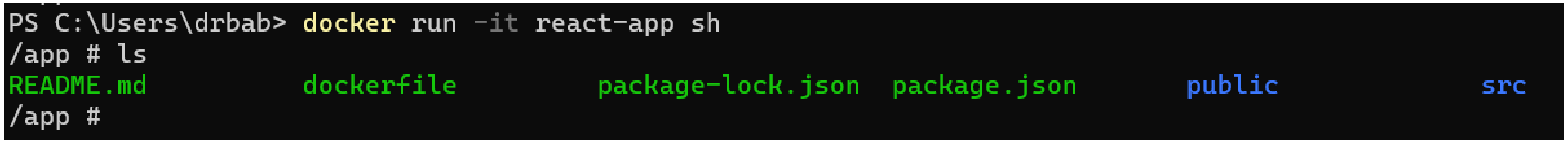
- We don’t really need to transfer node_module directory, because all dependencies are defined in package.json file

- We can exclude this directory in build (to have a faster build) and restore it on the target image



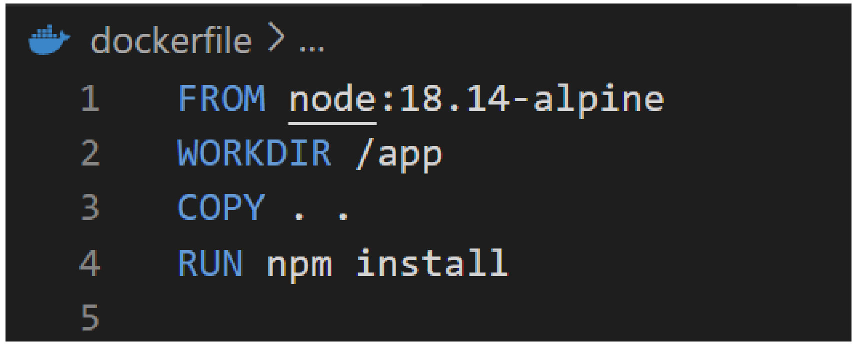
```
.dockerignore
node_modules/
```

```
=> [internal] load .dockerignore                                           0.0s
=> => transferring context: 53B                                            0.0s
=> [internal] load metadata for docker.io/library/node:18.14-alpine        0.5s
=> [auth] library/node:pull token for registry-1.docker.io                 0.0s
=> [1/3] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36b0be743   0.0s
=> [internal] load build context                                           1.1s
=> => transferring context: 4.56kB                                         1.1s
=> CACHED [2/3] WORKDIR /app                                               0.0s
=> [3/3] COPY . .                                                          0.0s
=> exporting to image                                                      0.0s
=> => exporting layers                                                     0.0s
=> => writing image sha256:990f417f0368dbea2169c16d29da10ab5a60579a4e000   0.0s
=> => naming to docker.io/library/react-app                                0.0s
```



```
PS C:\Users\drbab> docker run -it react-app sh
/app # ls
README.md          dockerfile          package-lock.json  package.json       public             src
/app #
```

- The next step is to install the project dependencies using npm
- We can use RUN command
- With this command we can execute all commands that we normally execute in the terminal session



dockerfile > ...

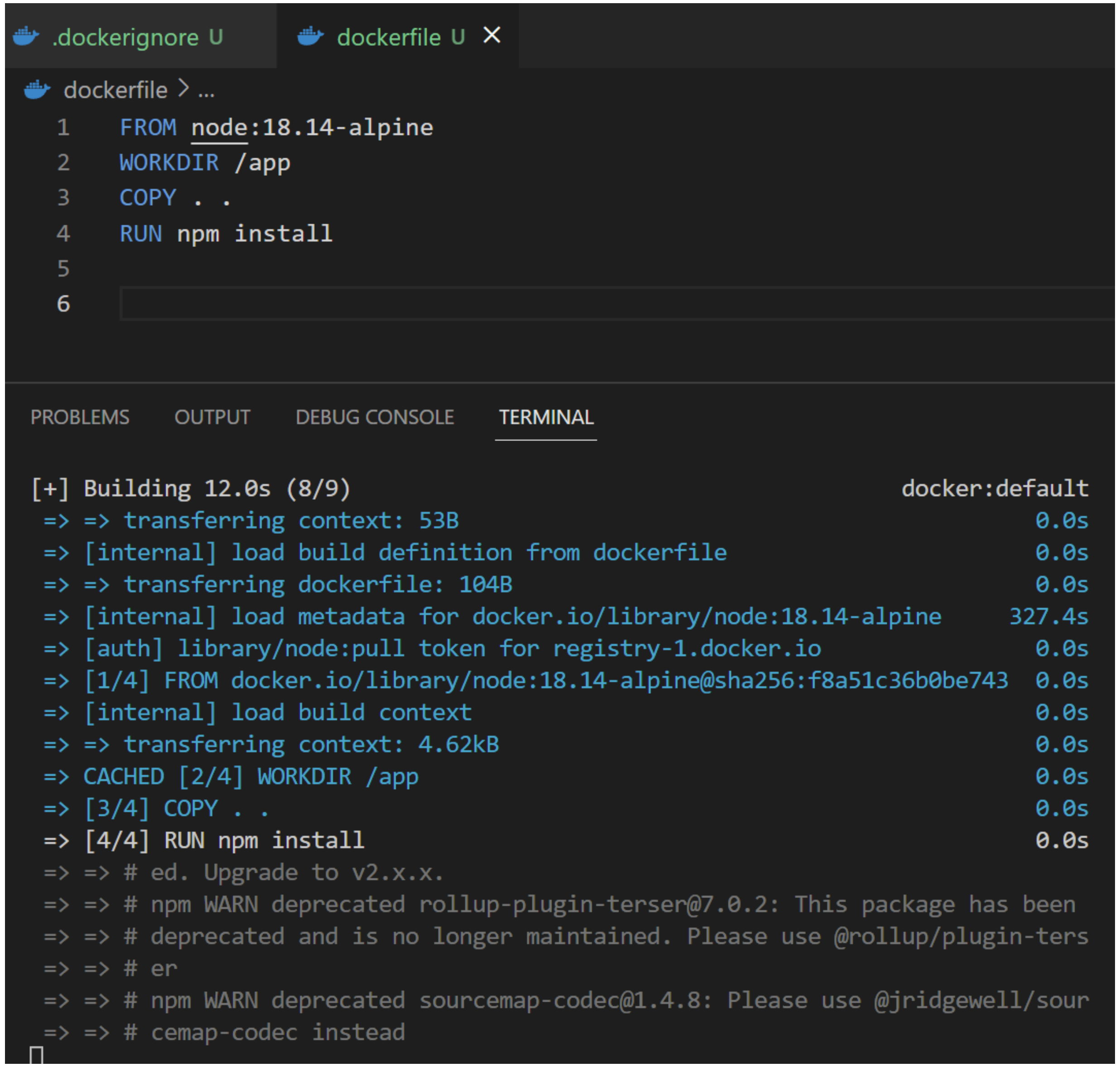
```
FROM node:18.14-alpine
WORKDIR /app
COPY . .
RUN npm install
```

NOTE:

RUN apt install python

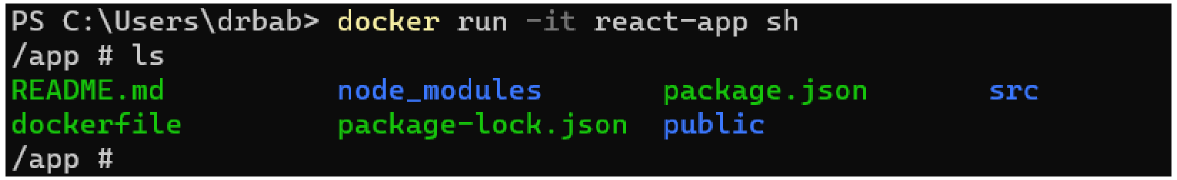
If you run this you will get an error because alpine doesn't have apt package manger! Instead it uses apk.

Be aware of these differences, depending on the type of the OS you are using.



```
FROM node:18.14-alpine
WORKDIR /app
COPY . .
RUN npm install

```

```
[+] Building 12.0s (8/9)                                          docker:default
 => => transferring context: 53B                                           0.0s
 => [internal] load build definition from dockerfile                       0.0s
 => => transferring dockerfile: 104B                                       0.0s
 => [internal] load metadata for docker.io/library/node:18.14-alpine     327.4s
 => [auth] library/node:pull token for registry-1.docker.io                0.0s
 => [1/4] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36b0be743  0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 4.62kB                                        0.0s
 => CACHED [2/4] WORKDIR /app                                              0.0s
 => [3/4] COPY . .                                                         0.0s
 => [4/4] RUN npm install                                                  0.0s
 => => # ed. Upgrade to v2.x.x.
 => => # npm WARN deprecated rollup-plugin-terser@7.0.2: This package has been
 => => # deprecated and is no longer maintained. Please use @rollup/plugin-ters
 => => # er
 => => # npm WARN deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sour
 => => # cemap-codec instead
```

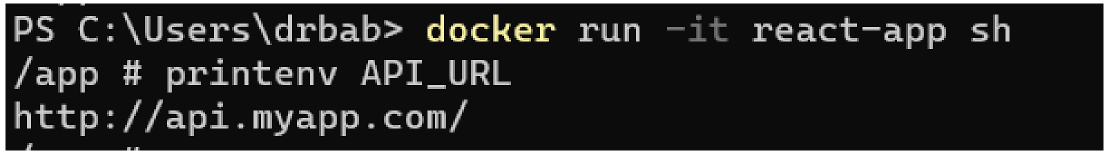


```
PS C:\Users\drbab> docker run -it react-app sh
/app # ls
README.md           node_modules        package.json        src
dockerfile          package-lock.json   public
/app #
```

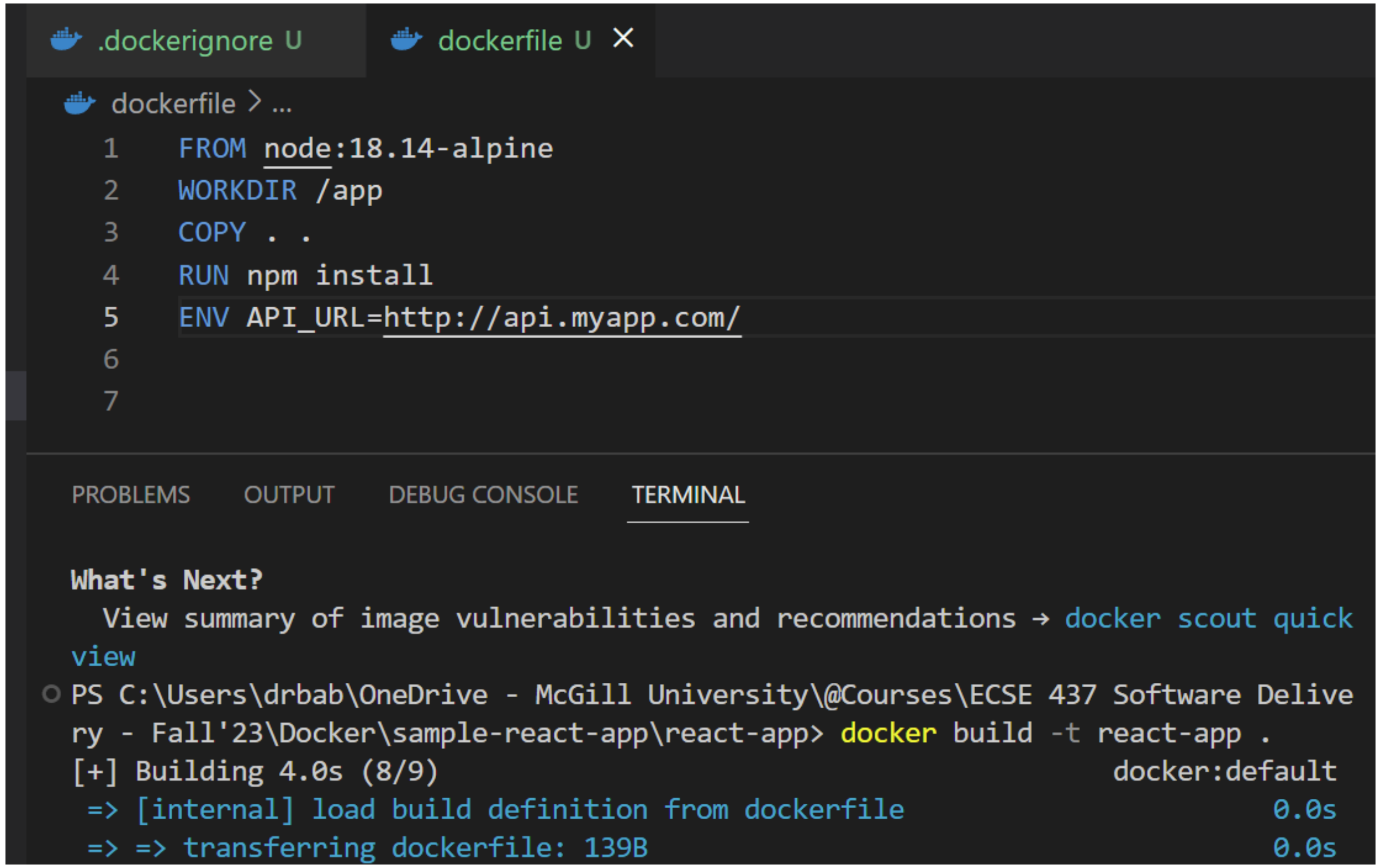
Setting environment variables

Sometimes we need to set env. variables

- The frontend app needs to talk to a backend or an API
- We set the URL of the API using an env. variable
- We can use RUN command
- We use ENV for this purpose



```
PS C:\Users\drbab> docker run -it react-app sh
/app # printenv API_URL
http://api.myapp.com/
```



```
FROM node:18.14-alpine
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
```

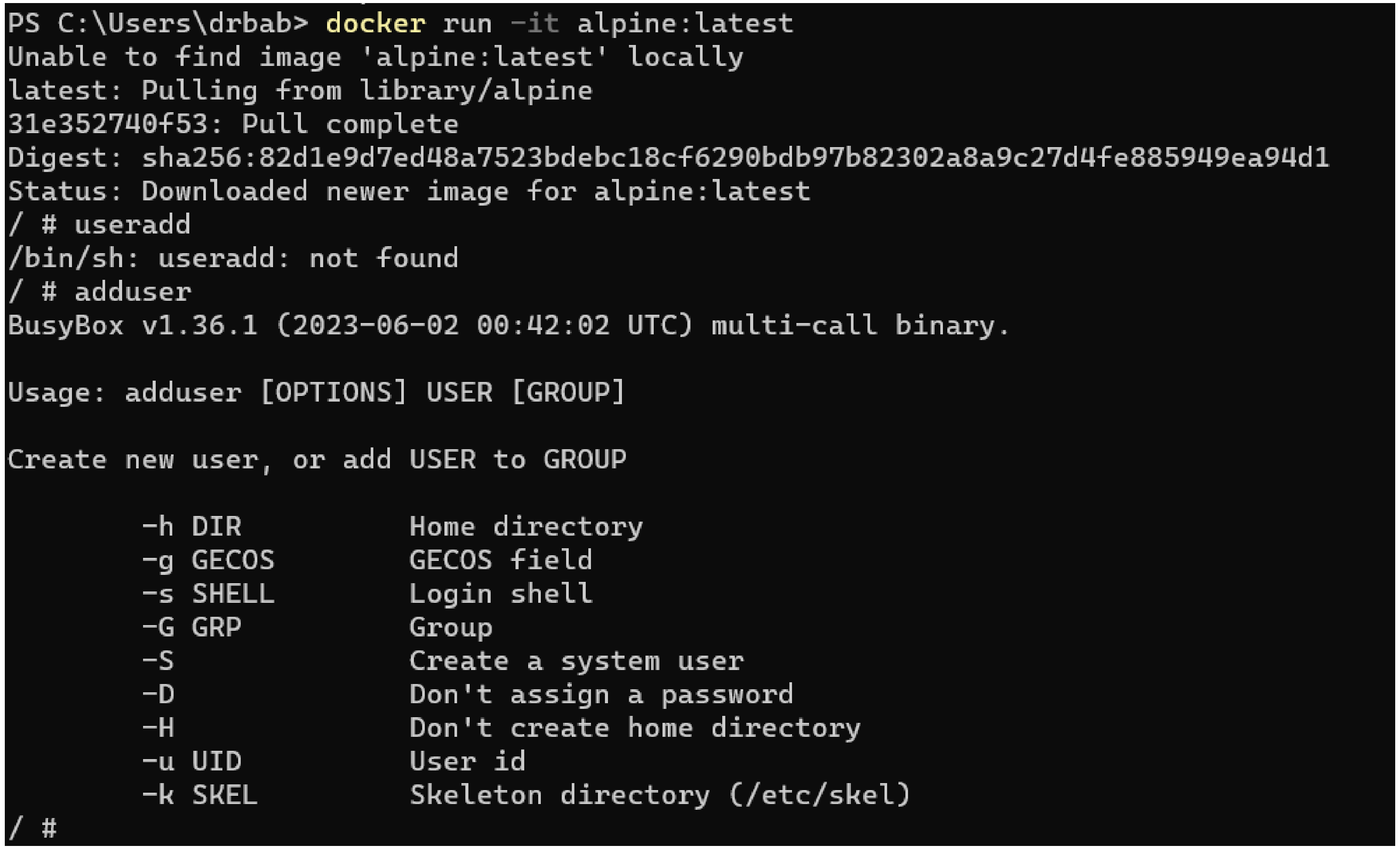
```
What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quick
view
PS C:\Users\drbab\OneDrive - McGill University\@Courses\ECSE 437 Software Delive
ry - Fall'23\Docker\sample-react-app\react-app> docker build -t react-app .
[+] Building 4.0s (8/9)                                              docker:default
 => [internal] load build definition from dockerfile                           0.0s
 => => transferring dockerfile: 139B                                           0.0s
```

Exposing Ports

- When we run an application in the host it will use a port to communicate with users
  - For example, in our ReactJs application: **npm run start**
  - It will listen to localhost/3000
- When we run this application inside the docker container this port will be open on the **container, not on the host!**
- On the same host machine, we can have multiple instances of an app running the same image
- All these containers will be listening to port 3000
- But the port 3000 on the host is not going to be mapped automatically to the ports on the containers
- In the dockerfile we use EXPORT to specify what port this container will be listening on
- The EXPOSE command doesn't automatically publish the port on the host
- It is just a form of documentation to tell us this container will eventually listen to what port to communicate with the host

Setting the User

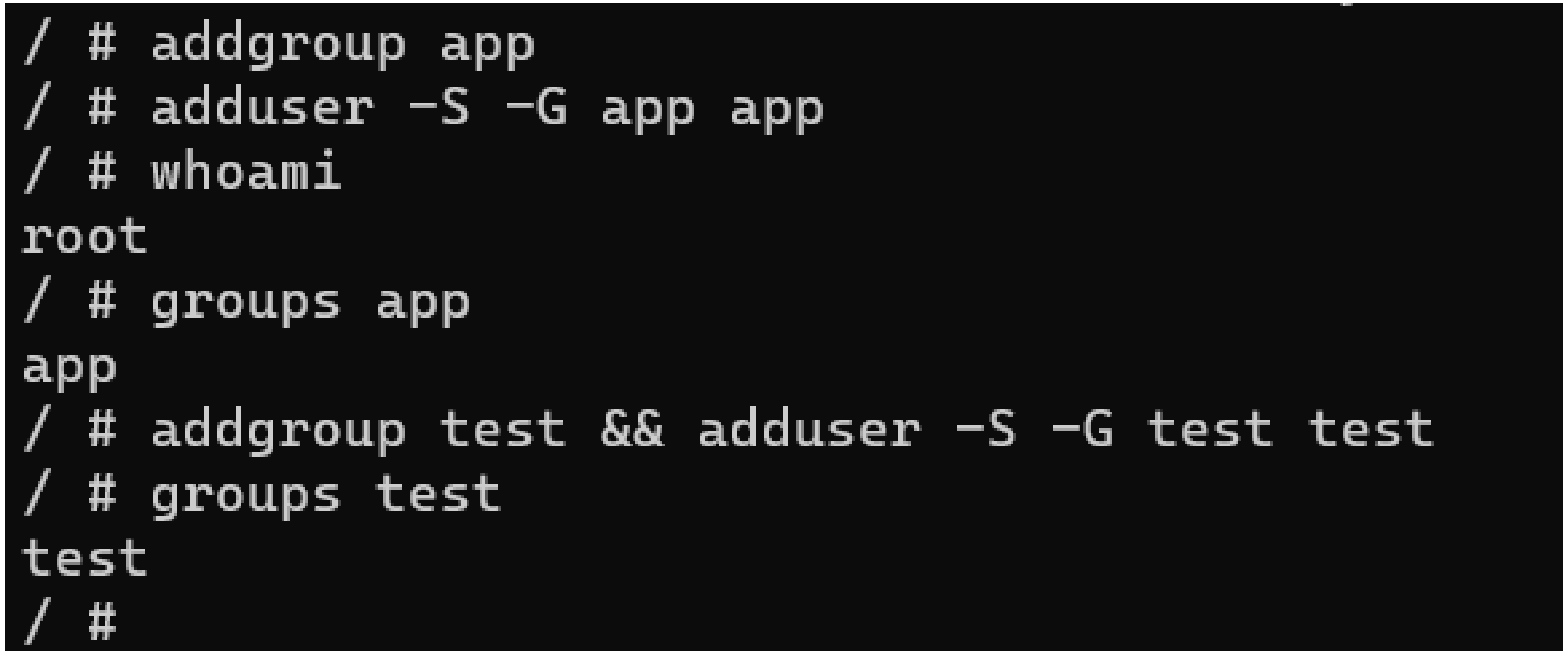
- By default, docker runs our app using the root user that has the higher privileges
- To run our app we should create a new user with limited privileges



```
PS C:\Users\drbab> docker run -it alpine:latest
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
31e352740f53: Pull complete
Digest: sha256:82d1e9d7ed48a7523bdebc18cf6290bdb97b82302a8a9c27d4fe885949ea94d1
Status: Downloaded newer image for alpine:latest
/ # useradd
/bin/sh: useradd: not found
/ # adduser
BusyBox v1.36.1 (2023-06-02 00:42:02 UTC) multi-call binary.

Usage: adduser [OPTIONS] USER [GROUP]

Create new user, or add USER to GROUP

        -h DIR          Home directory
        -g GECOS        GECOS field
        -s SHELL        Login shell
        -G GRP          Group
        -S              Create a system user
        -D              Don't assign a password
        -H              Don't create home directory
        -u UID          User id
        -k SKEL         Skeleton directory (/etc/skel)
/ #
```

Setting the User

- We use –S to create a system user that is not a real user (only for running an app)



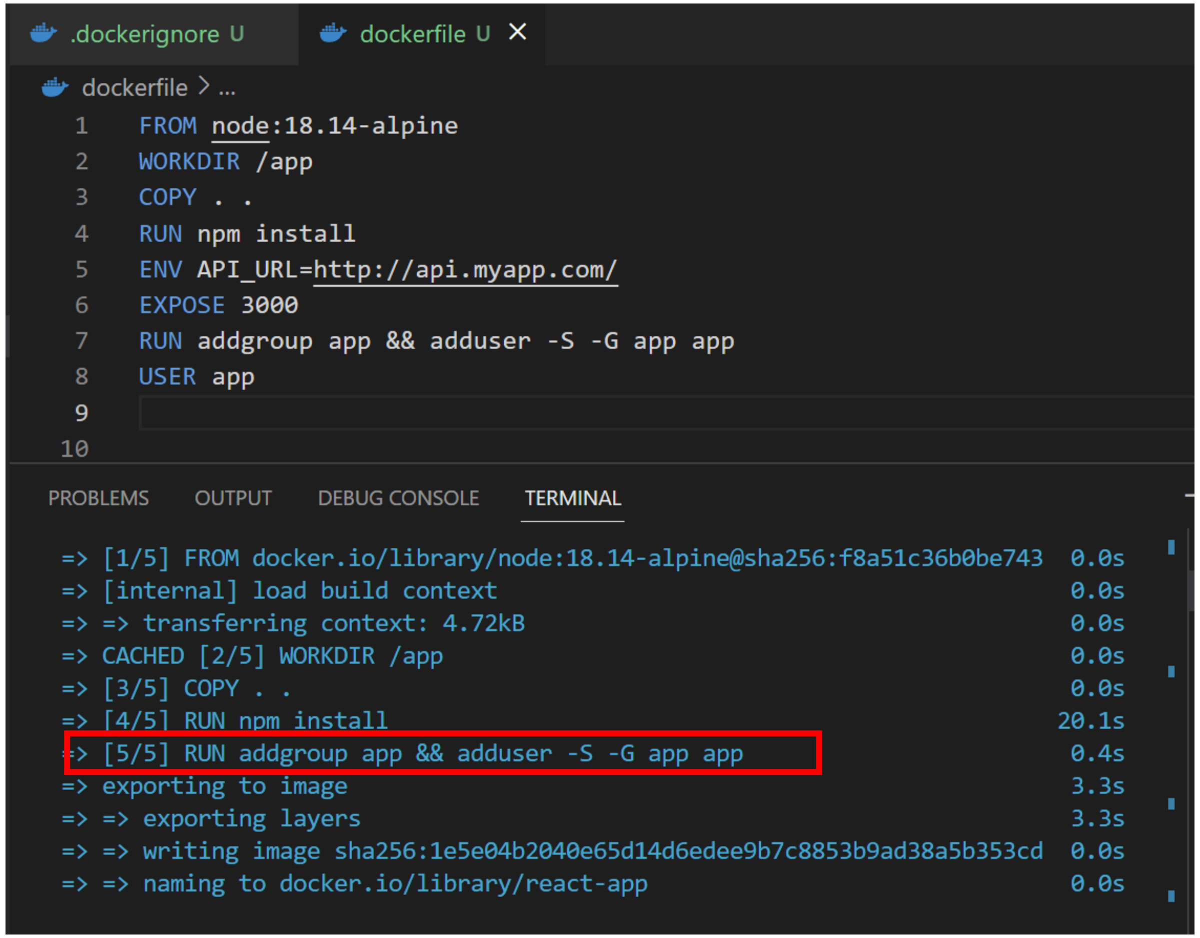
```
/ # addgroup app
/ # adduser -S -G app app
/ # whoami
root
/ # groups app
app
/ # addgroup test && adduser -S -G test test
/ # groups test
test
/ #
```

- We want to create this user inside the container running our image

Setting the User

- We use the RUN command
- Once we create the user and group, we need to switch to the limited user *app*
- All the commands coming after this will be executed with the user *app*

When you rebuild the app image you may notice the 4th step is taking too much time! We will back to this point and find a solution for that!

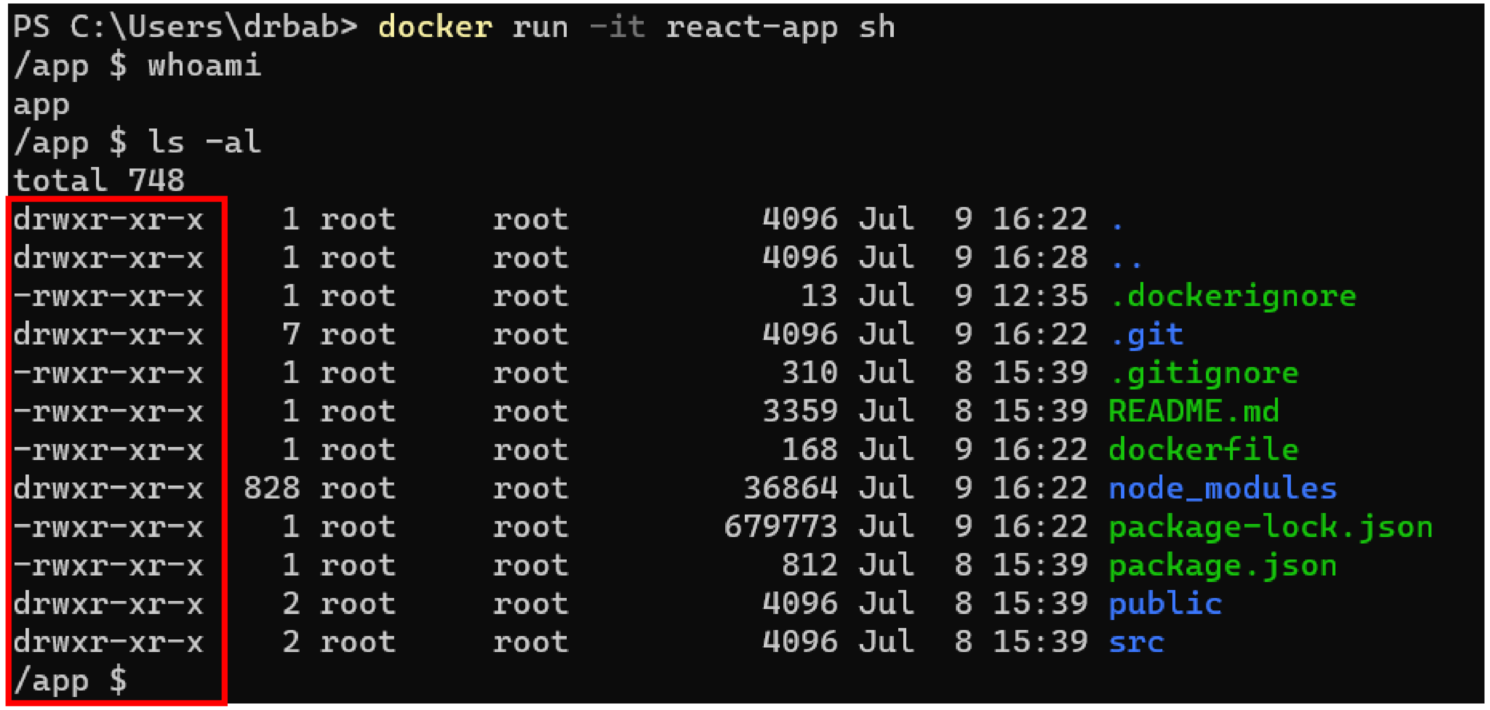


```dockerfile
FROM node:18.14-alpine
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
RUN addgroup app && adduser -S -G app app
USER app
```

```
=> [1/5] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36b0be743  0.0s
=> [internal] load build context                                          0.0s
=> => transferring context: 4.72kB                                        0.0s
=> CACHED [2/5] WORKDIR /app                                              0.0s
=> [3/5] COPY . .                                                         0.0s
=> [4/5] RUN npm install                                                 20.1s
=> [5/5] RUN addgroup app && adduser -S -G app app                        0.4s
=> exporting to image                                                     3.3s
=> => exporting layers                                                    3.3s
=> => writing image sha256:1e5e04b2040e65d14d6edee9b7c8853b9ad38a5b353cd  0.0s
=> => naming to docker.io/library/react-app                               0.0s
```

Setting the User

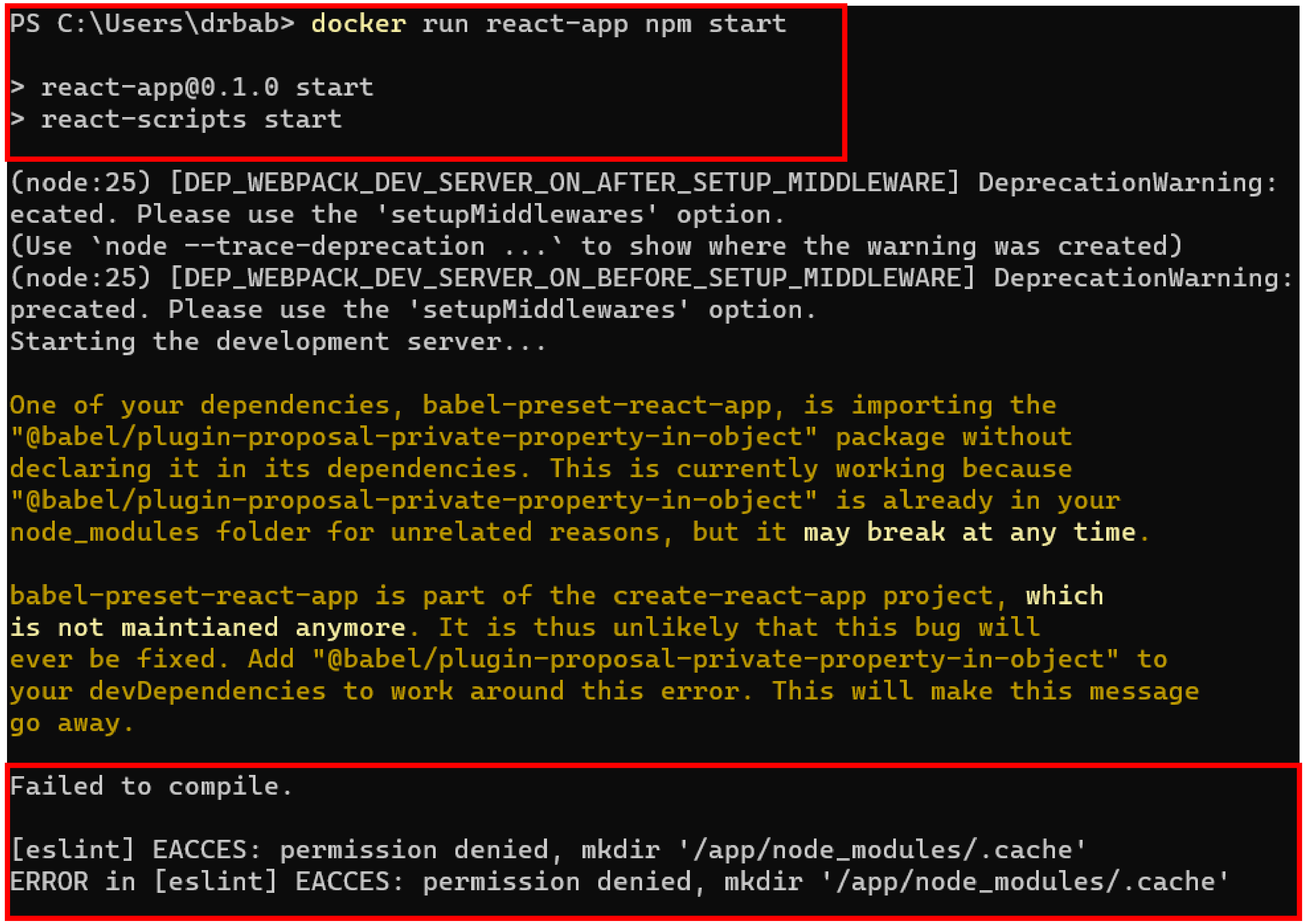
- Let’s verify the user running the app
- If you check the permissions, you will notice the root user has the write permission!



```
PS C:\Users\drbab> docker run -it react-app sh
/app $ whoami
app
/app $ ls -al
total 748
drwxr-xr-x    1 root     root          4096 Jul  9 16:22 .
drwxr-xr-x    1 root     root          4096 Jul  9 16:28 ..
-rwxr-xr-x    1 root     root            13 Jul  9 12:35 .dockerignore
drwxr-xr-x    7 root     root          4096 Jul  9 16:22 .git
-rwxr-xr-x    1 root     root           310 Jul  8 15:39 .gitignore
-rwxr-xr-x    1 root     root          3359 Jul  8 15:39 README.md
-rwxr-xr-x    1 root     root           168 Jul  9 16:22 dockerfile
drwxr-xr-x  828 root     root         36864 Jul  9 16:22 node_modules
-rwxr-xr-x    1 root     root        679773 Jul  9 16:22 package-lock.json
-rwxr-xr-x    1 root     root           812 Jul  8 15:39 package.json
drwxr-xr-x    2 root     root          4096 Jul  8 15:39 public
drwxr-xr-x    2 root     root          4096 Jul  8 15:39 src
/app $
```

Define Entry points

- To run the app we need to execute the command: **npm run start**

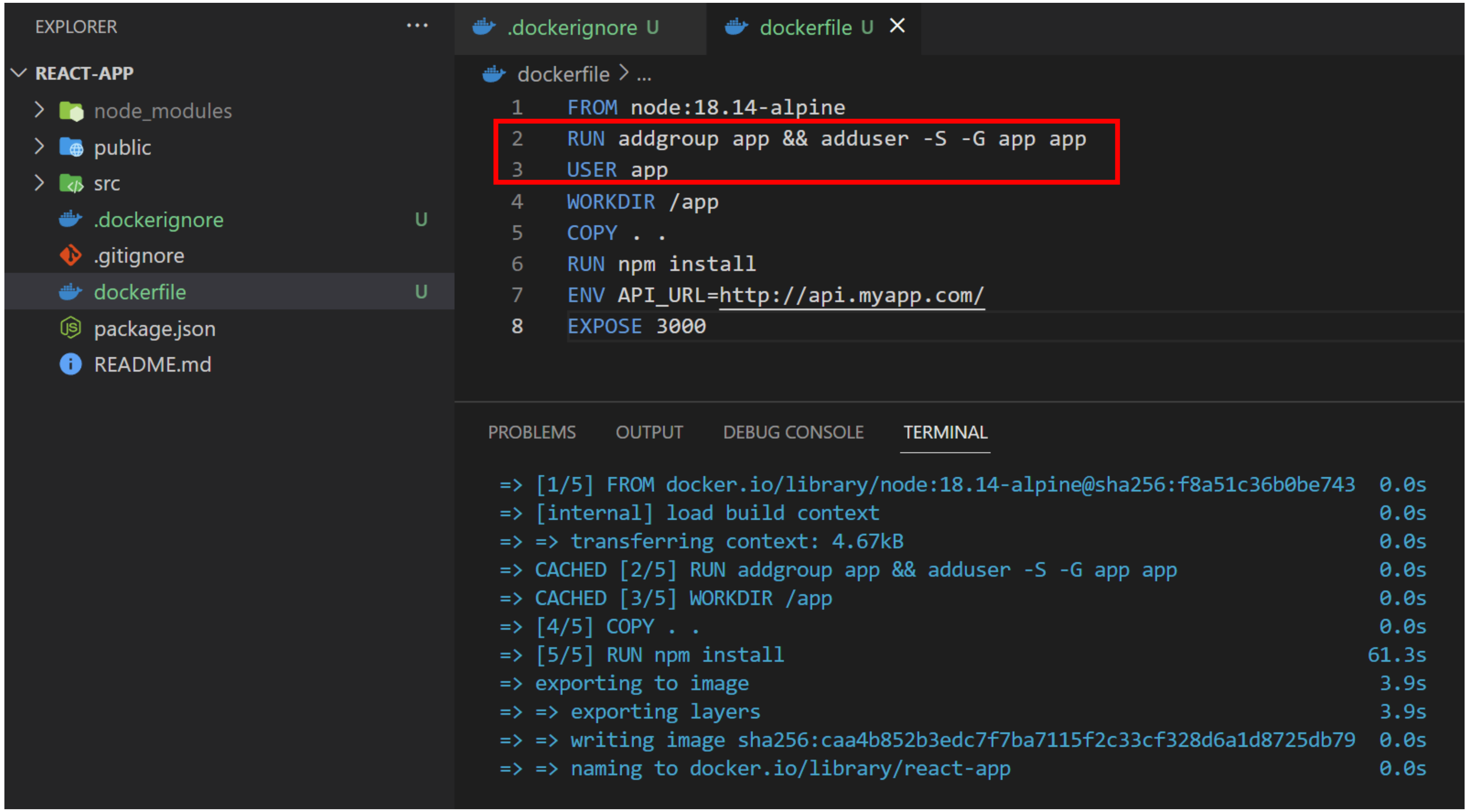


```
PS C:\Users\drbab> docker run react-app npm start

> react-app@0.1.0 start
> react-scripts start

(node:25) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning:
ecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:25) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning:
precated. Please use the 'setupMiddlewares' option.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

Failed to compile.

[eslint] EACCES: permission denied, mkdir '/app/node_modules/.cache'
ERROR in [eslint] EACCES: permission denied, mkdir '/app/node_modules/.cache'
```

But why we are getting this error?!

Define Entry points

- In the dockerfile we set the user at the end!
- All the instruction executed with the root user, then we switch to the limited user!



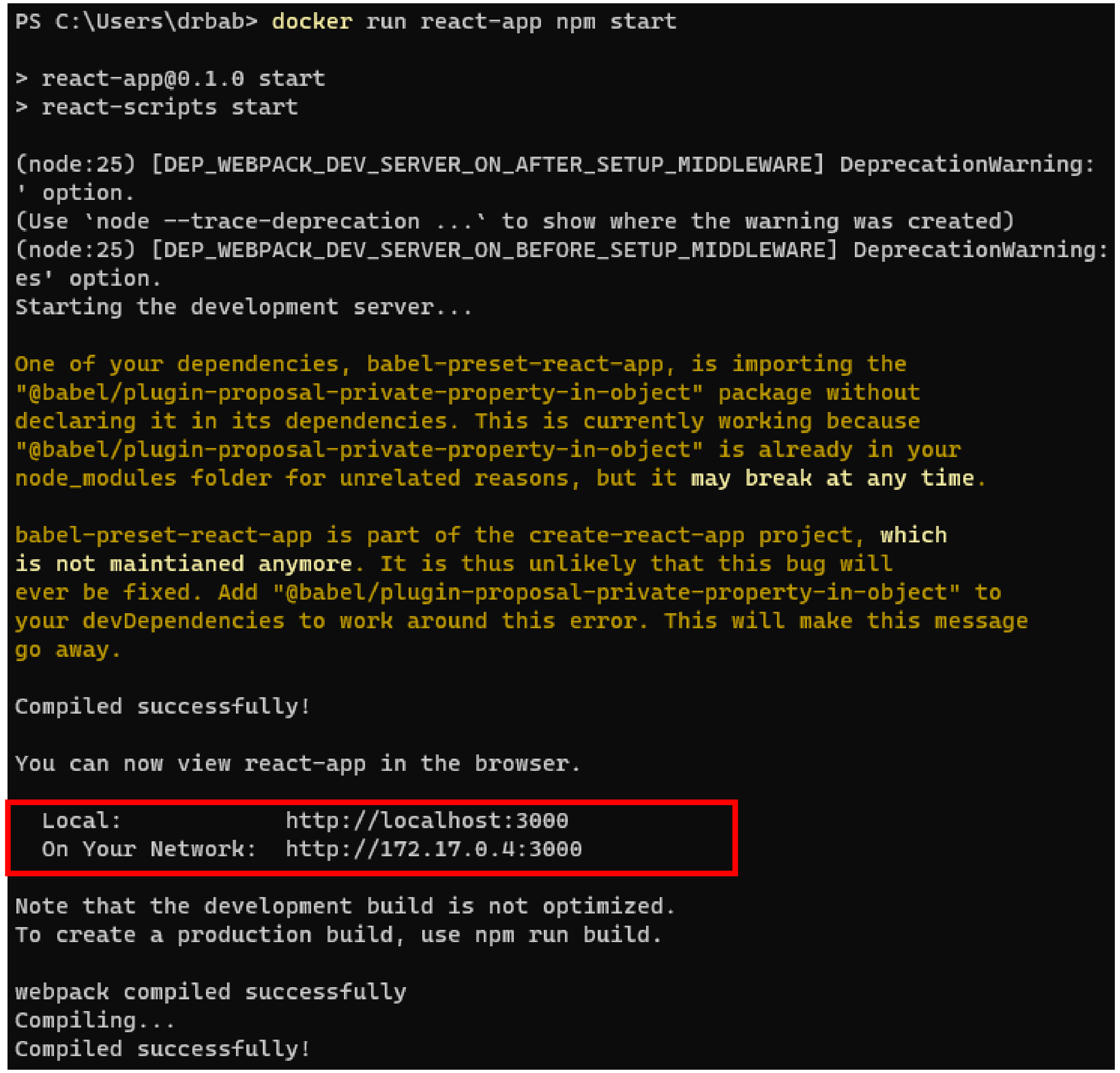
```
FROM node:18.14-alpine
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
```

```
=> [1/5] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36b0be743  0.0s
=> [internal] load build context                                            0.0s
=> => transferring context: 4.67kB                                          0.0s
=> CACHED [2/5] RUN addgroup app && adduser -S -G app app                   0.0s
=> CACHED [3/5] WORKDIR /app                                                0.0s
=> [4/5] COPY . .                                                           0.0s
=> [5/5] RUN npm install                                                   61.3s
=> exporting to image                                                       3.9s
=> => exporting layers                                                      3.9s
=> => writing image sha256:caa4b852b3edc7f7ba7115f2c33cf328d6a1d8725db79  0.0s
=> => naming to docker.io/library/react-app                                 0.0s
```

NOTE: You should remove the package-lock.json file!

Define Entry points

- This is a port 3000 of the container not the localhost!
- We will see how to do the port mapping!

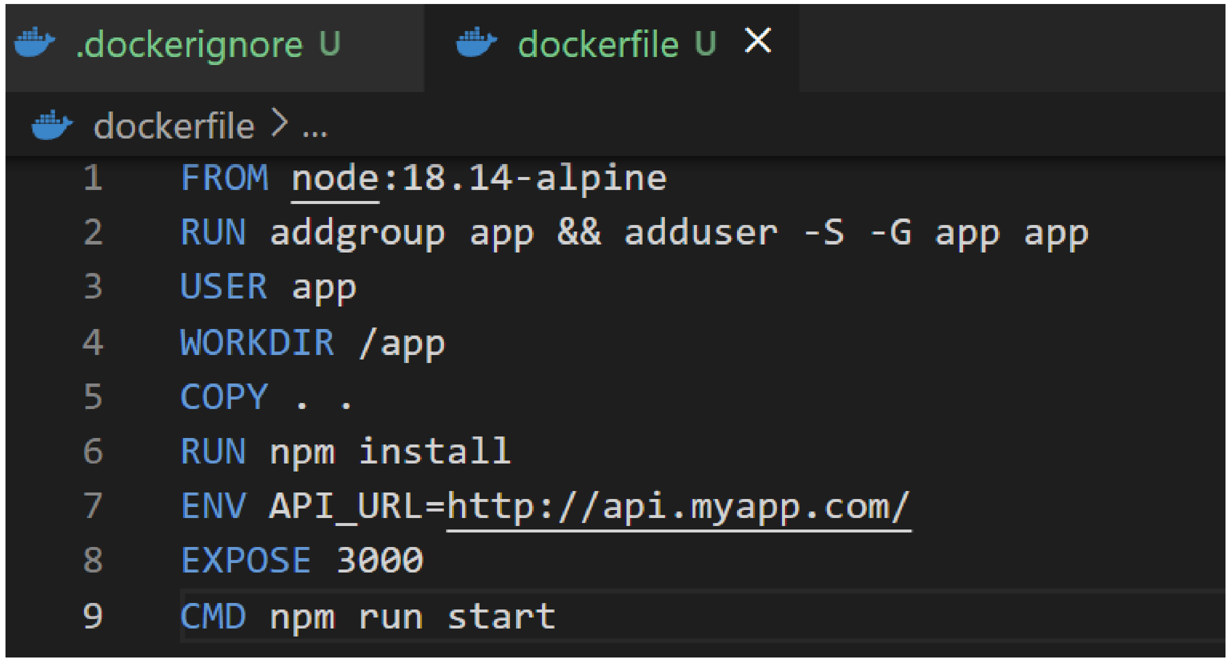


```
PS C:\Users\drbab> docker run react-app npm start

> react-app@0.1.0 start
> react-scripts start

(node:25) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning:
' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:25) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning:
es' option.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

Compiled successfully!

You can now view react-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.17.0.4:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Compiling...
Compiled successfully!
```

Define Entry points

- We don’t want to specify this command every time we run the container!
- We can use CMD to supply the default command
- It does not make sense to have multiple CMD lines in the dockerfile!

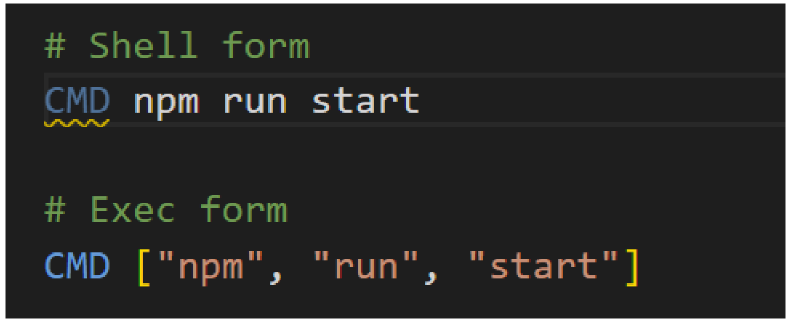
What is the difference between CMD and RUN?! With both, we can execute commands!



```
FROM node:18.14-alpine
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
CMD npm run start
```

RUN vs. CMD

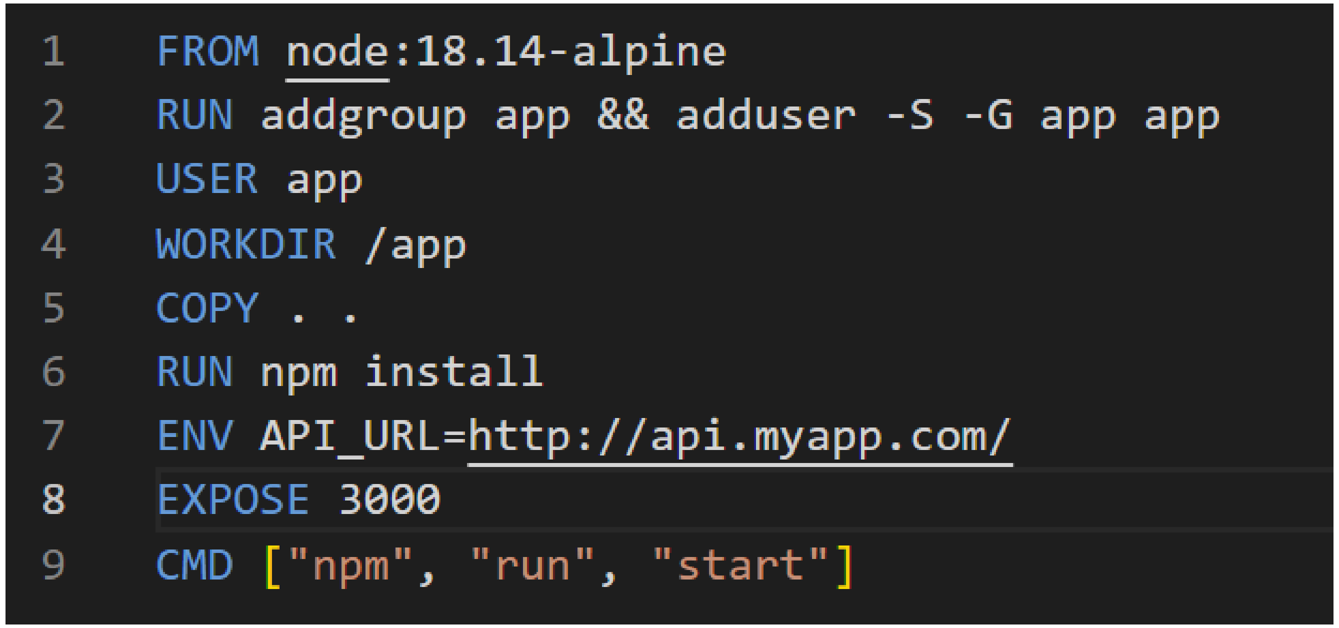
- RUN is the build time instruction, and it is executed at the time of building the image
- CMD is the runtime instruction, and it is executed when starting a container
- CMD has two forms! (Shell and Exec)
  - Shell form: docker execute the command inside a separate shell. On Linux the shell is under /bin/bash or /bin/sh
  - Exec form: it is executed directly with the OS and no need to run an additional shell process. It makes it faster to clean up resources when you stop the container.



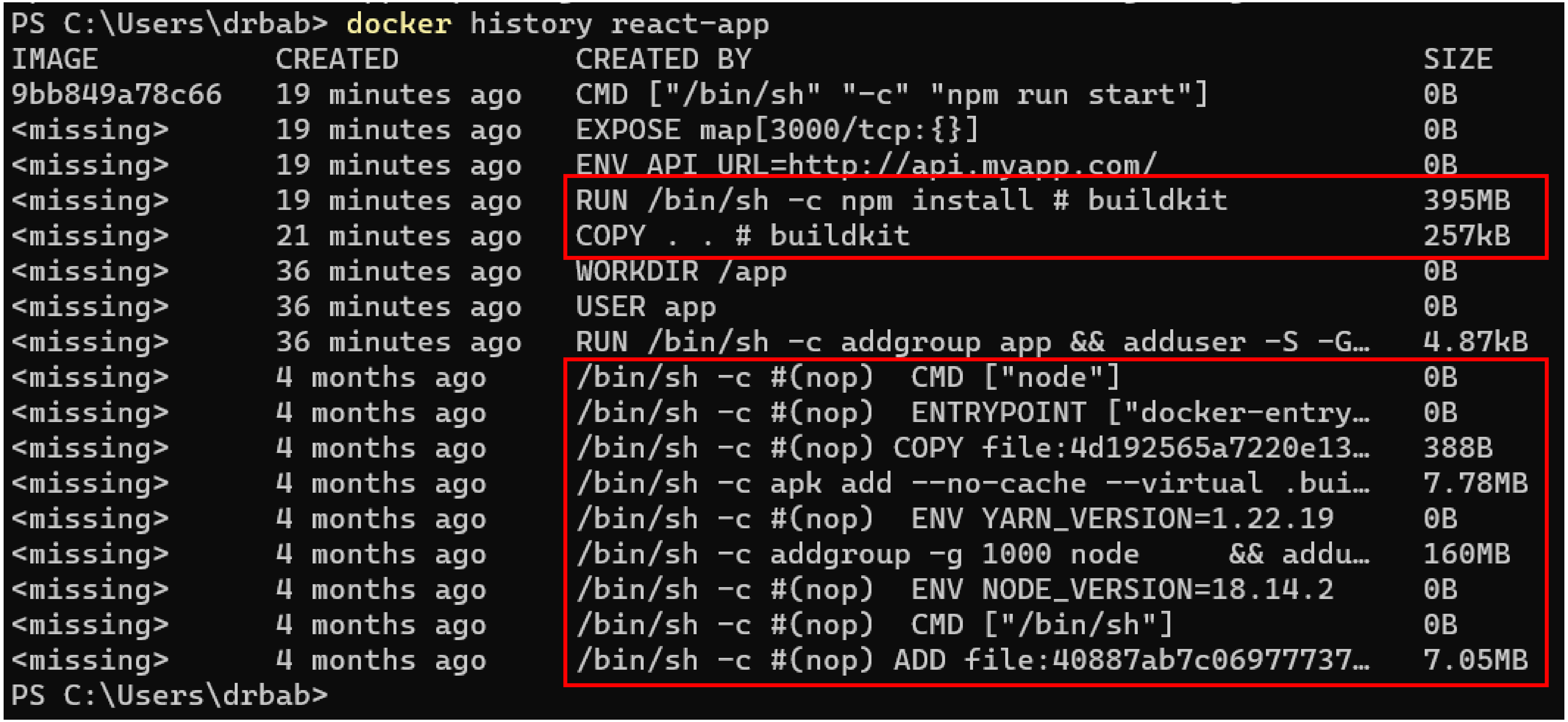
```
# Shell form
CMD npm run start

# Exec form
CMD ["npm", "run", "start"]
```

How to speeding up builds

- The first thing we need to understand is the concept of layers in Docker
- An image is essentially collection of layers
- You can think of a layer as a small file system that only includes modified files
- When docker builds an image it execute each instruction in the dockerfile and create a new layer (include only files that is modified as the result of the instruction)
- Node image itself is several layers!



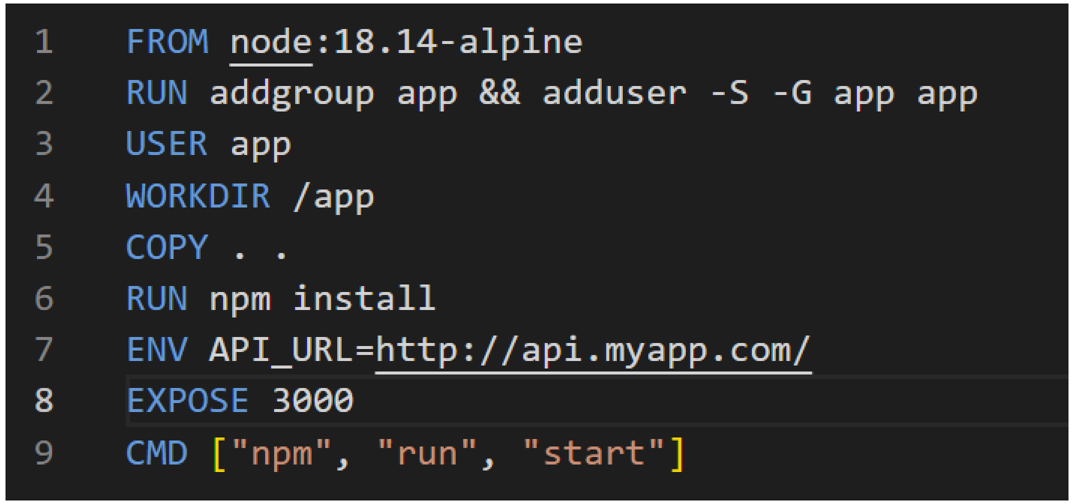
```dockerfile
FROM node:18.14-alpine
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
CMD ["npm", "run", "start"]
```



```
PS C:\Users\drbab> docker history react-app
IMAGE          CREATED          CREATED BY                                      SIZE
9bb849a78c66   19 minutes ago   CMD ["/bin/sh" "-c" "npm run start"]            0B
<missing>      19 minutes ago   EXPOSE map[3000/tcp:{}]                         0B
<missing>      19 minutes ago   ENV API_URL=http://api.myapp.com/               0B
<missing>      19 minutes ago   RUN /bin/sh -c npm install # buildkit           395MB
<missing>      21 minutes ago   COPY . . # buildkit                             257kB
<missing>      36 minutes ago   WORKDIR /app                                    0B
<missing>      36 minutes ago   USER app                                        0B
<missing>      36 minutes ago   RUN /bin/sh -c addgroup app && adduser -S -G…   4.87kB
<missing>      4 months ago     /bin/sh -c #(nop)  CMD ["node"]                 0B
<missing>      4 months ago     /bin/sh -c #(nop)  ENTRYPOINT ["docker-entry…   0B
<missing>      4 months ago     /bin/sh -c #(nop) COPY file:4d192565a7220e13…   388B
<missing>      4 months ago     /bin/sh -c apk add --no-cache --virtual .bui…   7.78MB
<missing>      4 months ago     /bin/sh -c #(nop)  ENV YARN_VERSION=1.22.19     0B
<missing>      4 months ago     /bin/sh -c addgroup -g 1000 node     && addu…   160MB
<missing>      4 months ago     /bin/sh -c #(nop)  ENV NODE_VERSION=18.14.2     0B
<missing>      4 months ago     /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      4 months ago     /bin/sh -c #(nop) ADD file:40887ab7c06977737…   7.05MB
PS C:\Users\drbab>
```

How to speeding up builds

- Docker has a built-in optimization mechanism
- The next time it builds the image if nothing changed it is going to reuse the results in the cache
- If we made a tiny change then the docker need to rebuild the layer
- “COPY . .” is the special instruction because with that docker cannot tell if anything has changed or not!
- It looks into the content of all files! If single line in a file has changed docker cannot use the result of the cache.
- All the instructions after that need to be executed again! *This is where the problem happens!*

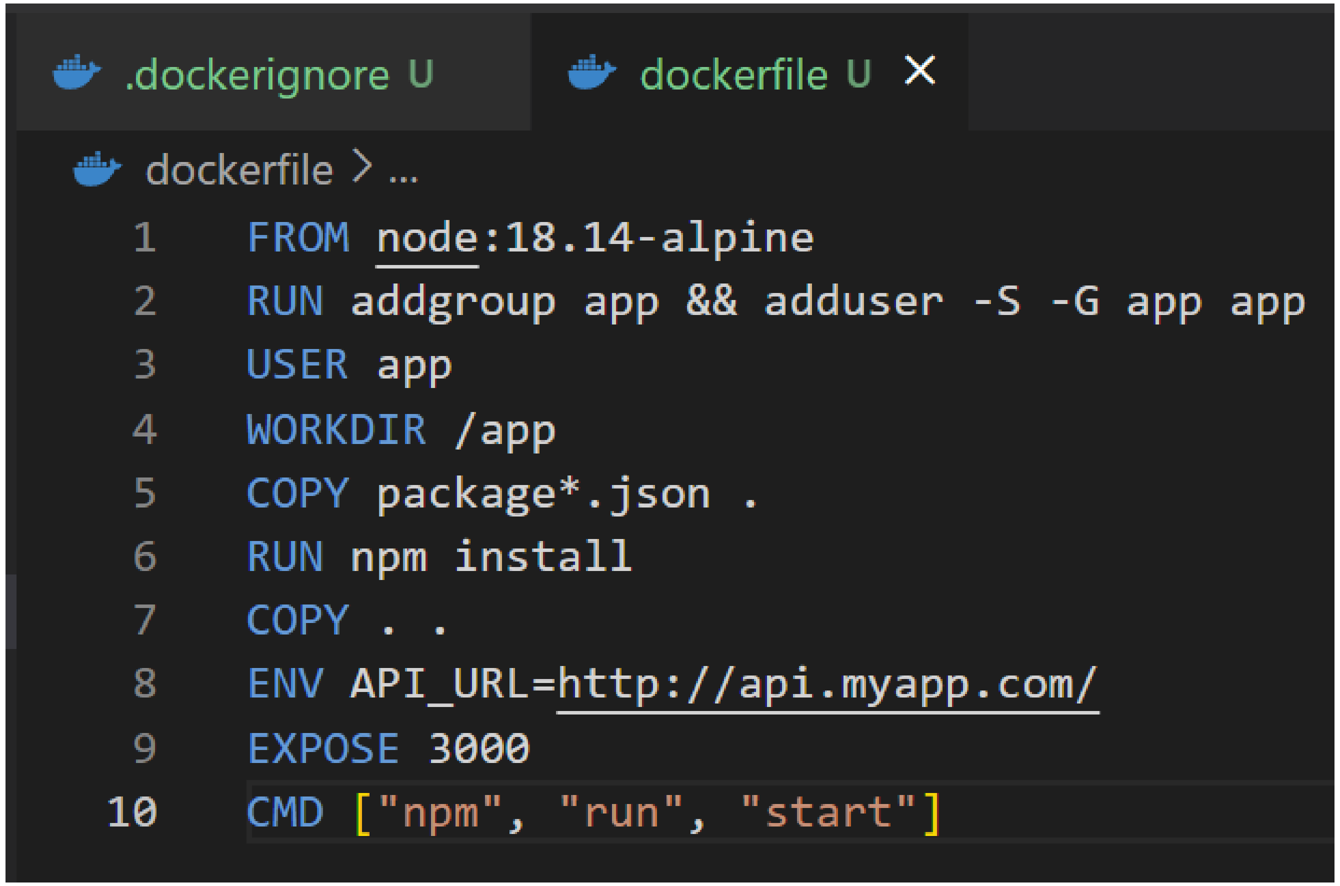


```
FROM node:18.14-alpine
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
COPY . .
RUN npm install
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
CMD ["npm", "run", "start"]
```

How to speeding up builds

- We have to separate the installation of the 3rd party dependencies from copying all the files in our application!
- How do we do that?! Instead of copying all the files in one go, first copy the files needed for installing 3rd party dependencies, i.e., package*.json
- With this new setup, if we haven't change any of our app dependencies, docker is going to reuse this layer from its cache, because package.json is not modified!
- Similarly, docker is not going to re-install dependencies
- "COPY . ." always need to be rebuilt and it is fine!

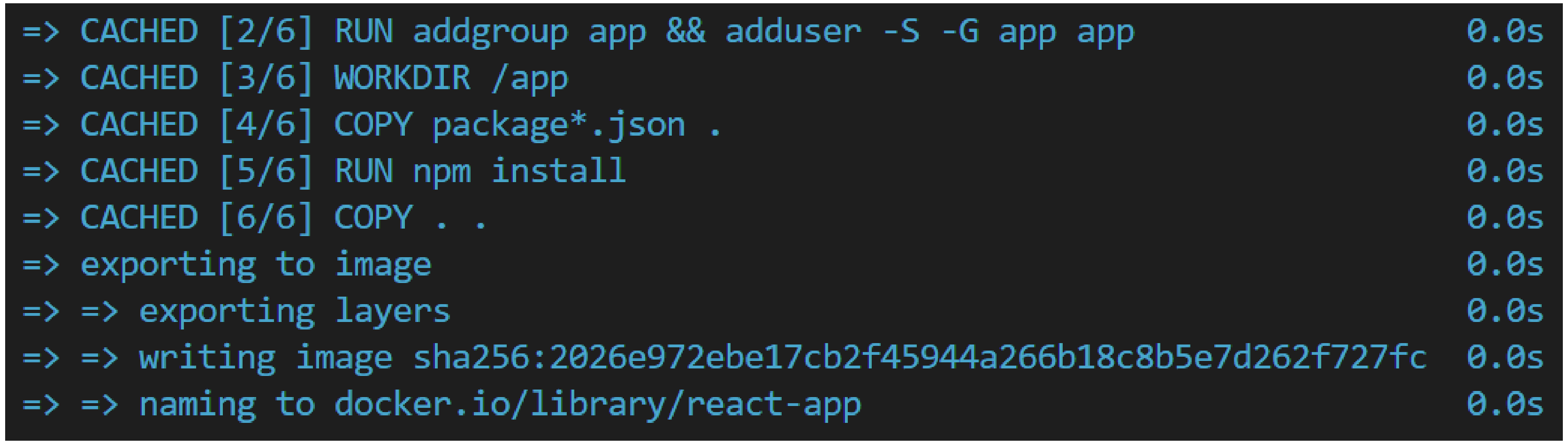
The first time you run, it takes a long time to build the image!



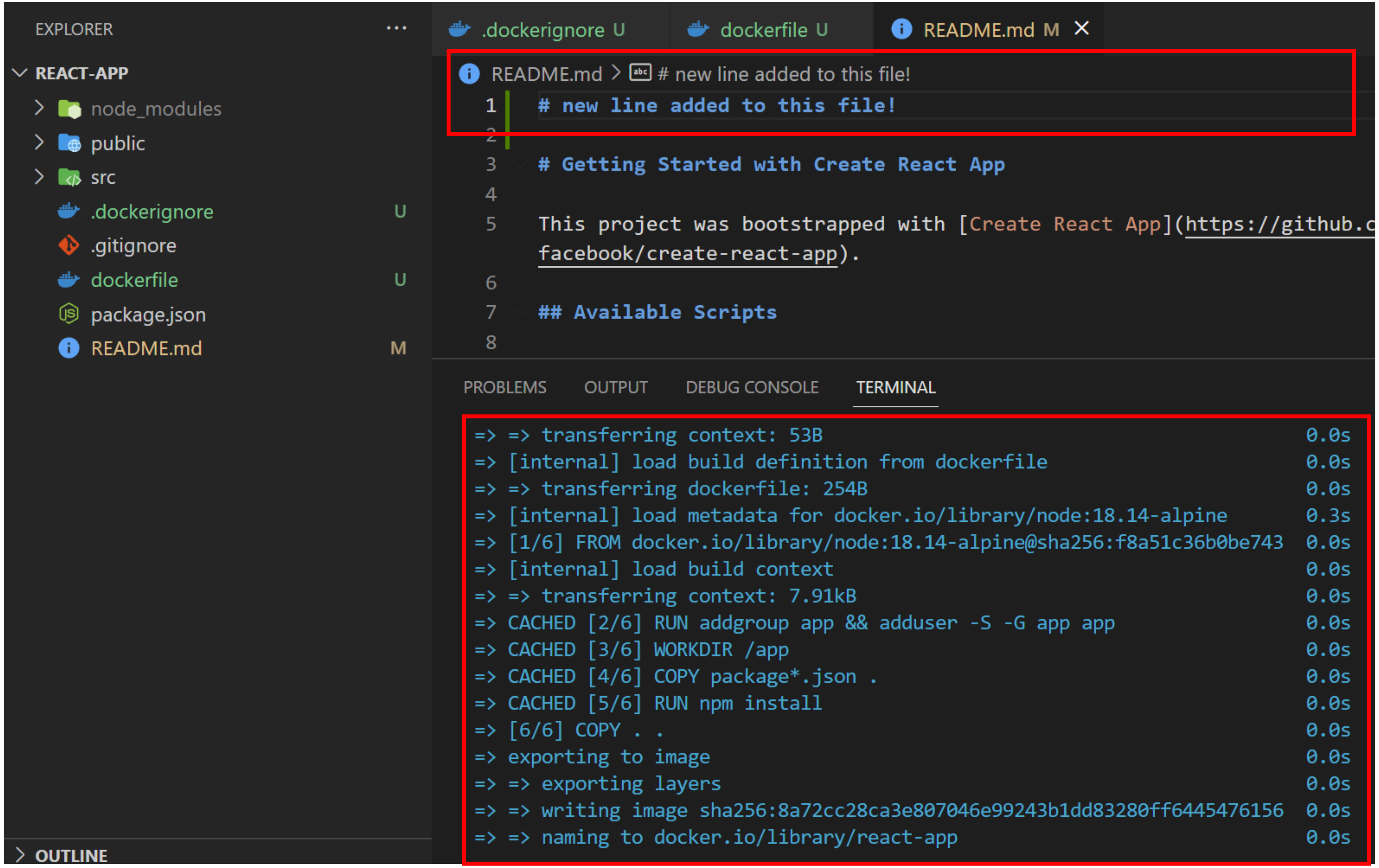
.dockerignore U dockerfile U ×

dockerfile > ...

```dockerfile
FROM node:18.14-alpine
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
CMD ["npm", "run", "start"]
```



```
=> CACHED [2/6] RUN addgroup app && adduser -S -G app app                  0.0s
=> CACHED [3/6] WORKDIR /app                                               0.0s
=> CACHED [4/6] COPY package*.json .                                       0.0s
=> CACHED [5/6] RUN npm install                                            0.0s
=> CACHED [6/6] COPY . .                                                   0.0s
=> exporting to image                                                      0.0s
=> => exporting layers                                                     0.0s
=> => writing image sha256:2026e972ebe17cb2f45944a266b18c8b5e7d262f727fc   0.0s
=> => naming to docker.io/library/react-app                                0.0s
```



```
# new line added to this file!

# Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.c
facebook/create-react-app).

## Available Scripts

```

```
=> => transferring context: 53B                                           0.0s
=> [internal] load build definition from dockerfile                       0.0s
=> => transferring dockerfile: 254B                                       0.0s
=> [internal] load metadata for docker.io/library/node:18.14-alpine      0.3s
=> [1/6] FROM docker.io/library/node:18.14-alpine@sha256:f8a51c36b0be743  0.0s
=> [internal] load build context                                          0.0s
=> => transferring context: 7.91kB                                        0.0s
=> CACHED [2/6] RUN addgroup app && adduser -S -G app app                 0.0s
=> CACHED [3/6] WORKDIR /app                                              0.0s
=> CACHED [4/6] COPY package*.json .                                      0.0s
=> CACHED [5/6] RUN npm install                                           0.0s
=> [6/6] COPY . .                                                         0.0s
=> exporting to image                                                     0.0s
=> => exporting layers                                                    0.0s
=> => writing image sha256:8a72cc28ca3e807046e99243b1dd83280ff6445476156  0.0s
=> => naming to docker.io/library/react-app                               0.0s
```

To create an efficient dockerfile

The instructions/files that don’t change frequently should be on the top!

The instructions/files that change frequently should be on the bottom!