docker® **Crash Course Tutorial**

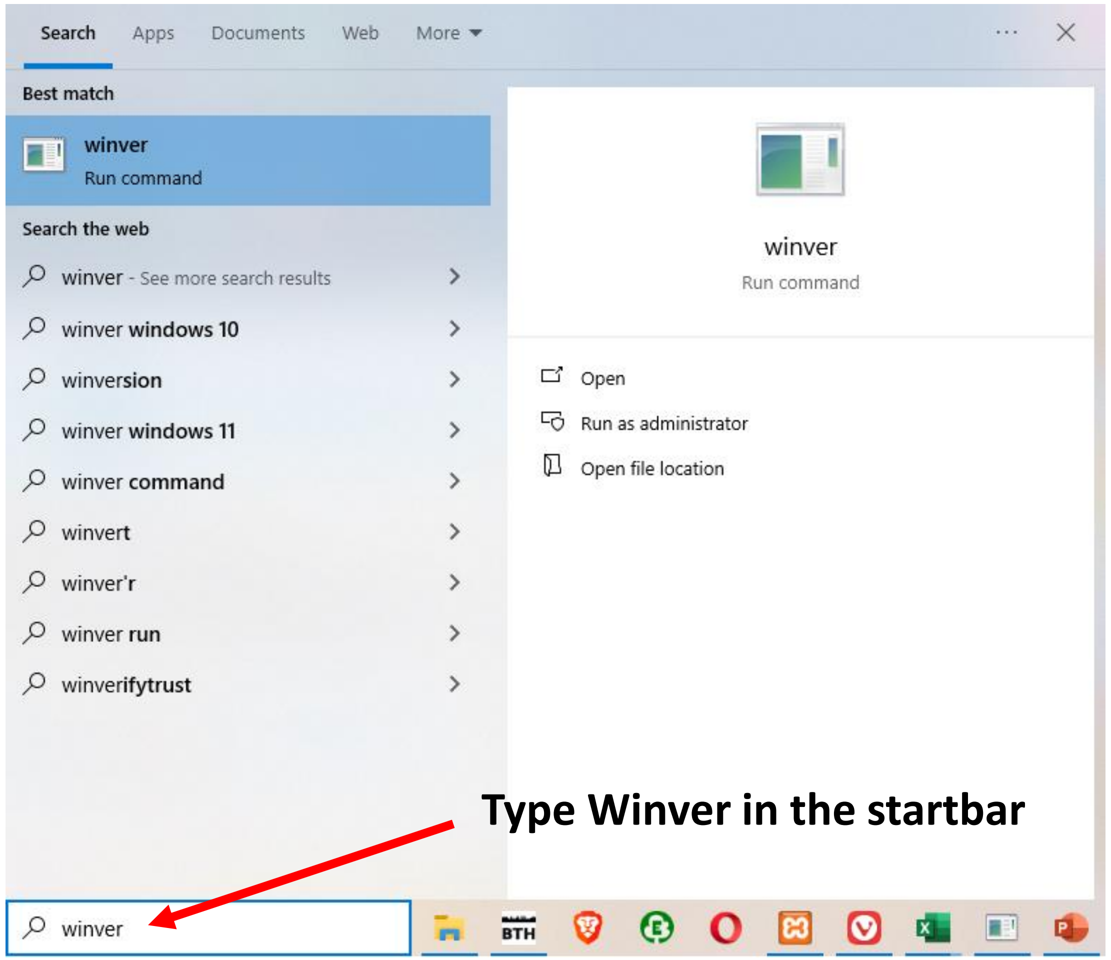
Docker Crash Course Tutorial keywords

What is docker compared to a VM? | Installing docker on windows with WSL (Windows Subsystem for Linux) | installing docker desktop | Image and container theory | pulling a parent image | Docker Hub | Running an image | Command line to running image | create Dockerfile | build image from Dockerfile | dockerignore file | container optional parameters | container port mapping | open container in the browser | docker ps |docker ps -a | docker run | docker stop | docker laker caching | manipulating dockerfile to exploit layer caching | observing image build times | delete an image | force deleting an image | image in use dangling | delete container before deleting image | delete dangling image | delete multiple images | delete multiple containers | image versioning | docker system prune | docker image versioning tag | run container on specific image version | What are volumes | why use volumes | soring persistent data in volumes | docker run with volumes | docker anonymous volumes | docker-compose.yaml | docker-compose up | docker-compose down & options | Dockerising a React App | dockerfile for react app | .dockerignore for react app | adding react app to the docker-compose.yaml | spinning up multiple app from using docker composer | create docker repo | build image to upload to repo | push image to repo | verify and review tags | pull image from repo

#1 Installing Docker on windows machine

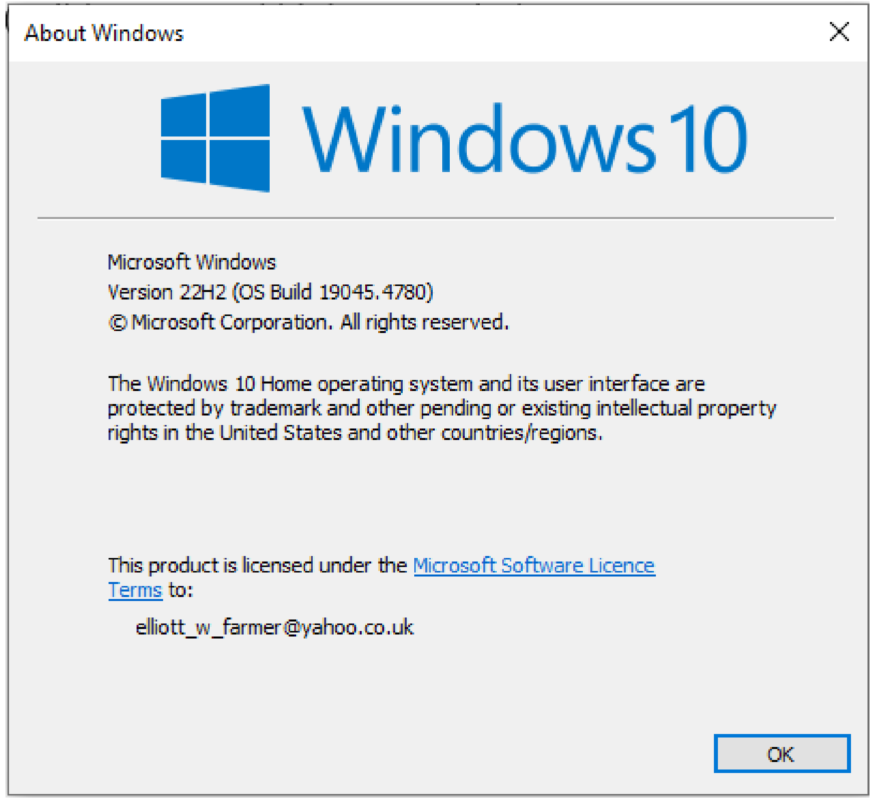
System Requirements for WSL

Install WSL (Windows Subsystem for Linux) on windows 10 or 11 – System requirements:
https://learn.microsoft.com/en-us/windows/wsl/install#prerequisites
You must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11



Type Winver in the startbar

My machine is running 20H2 build 19045 so it is higher than the minimum windows 10 requirement



Microsoft Windows
Version 22H2 (OS Build 19045.4780)
© Microsoft Corporation. All rights reserved.

The Windows 10 Home operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

This product is licensed under the Microsoft Software Licence Terms to:

elliott_w_farmer@yahoo.co.uk

"Windows 10 May 2020 Update (also known as version 2004 and codenamed "20H1") is the ninth major update to Windows 10. It carries the build number 10.0. 19041"

Manual Installation of WSL (1)

https://learn.microsoft.com/en-us/windows/wsl/install-manual

Step 1: Open PowerShell **as Administrator (Start menu > PowerShell > right-click > Run as Administrator)** and enter this command:

```
PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux
/all /norestart

Deployment Image Servicing and Management tool
Version: 10.0.19041.3636

Image Version: 10.0.19045.4780

Enabling feature(s)
[==========================100.0%==========================]
The operation completed successfully.
PS C:\Windows\system32>
```

Manual Installation of WSL (2)

Step 2: Check requirements for running WSL 2

To update to WSL 2, you must be running Windows 10...
- For x64 systems: **Version 1903** or later, with **Build 18362.1049** or later.
- For ARM64 systems: **Version 2004** or later, with **Build 19041** or later.

Step 3: Enable Virtual Machine feature

Before installing WSL 2, you must enable the **Virtual Machine Platform** optional feature. Your machine will require virtualization capabilities to use this feature. Open PowerShell as Administrator and run:

```
PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all
/norestart

Deployment Image Servicing and Management tool
Version: 10.0.19041.3636

Image Version: 10.0.19045.4780

Enabling feature(s)
[==========================100.0%==========================]
The operation completed successfully.
PS C:\Windows\system32>
```

Restart your machine to complete the WSL install and update to WSL 2.

Manual Installation of WSL (4)

Step 4: Download the Linux kernel update package

The Linux kernel update package installs the most recent version of the WSL 2 Linux kernel for running WSL inside the Windows operating system image. (To run WSL from the Microsoft Store, with more frequently pushed updates, use wsl.exe --install or wsl.exe --update.).

WSL2 Linux kernel update package for x64 machines

Run the update package downloaded in the previous step. (Double-click to run - you will be prompted for elevated permissions, select 'yes' to approve this installation.) Once the installation is complete, move on to the next step - setting WSL 2 as your default version when installing new Linux distributions. (Skip this step if you want your new Linux installs to be set to WSL 1).

Step 5: Set WSL 2 as your default version

Open PowerShell and run this command to set WSL 2 as the default version when installing a new Linux distribution:

```
PS C:\Windows\system32> wsl --set-default-version 2
For information on key differences with WSL 2 please visit https://aka.ms/wsl2
The operation completed successfully.
C:\Windows\system32>
```

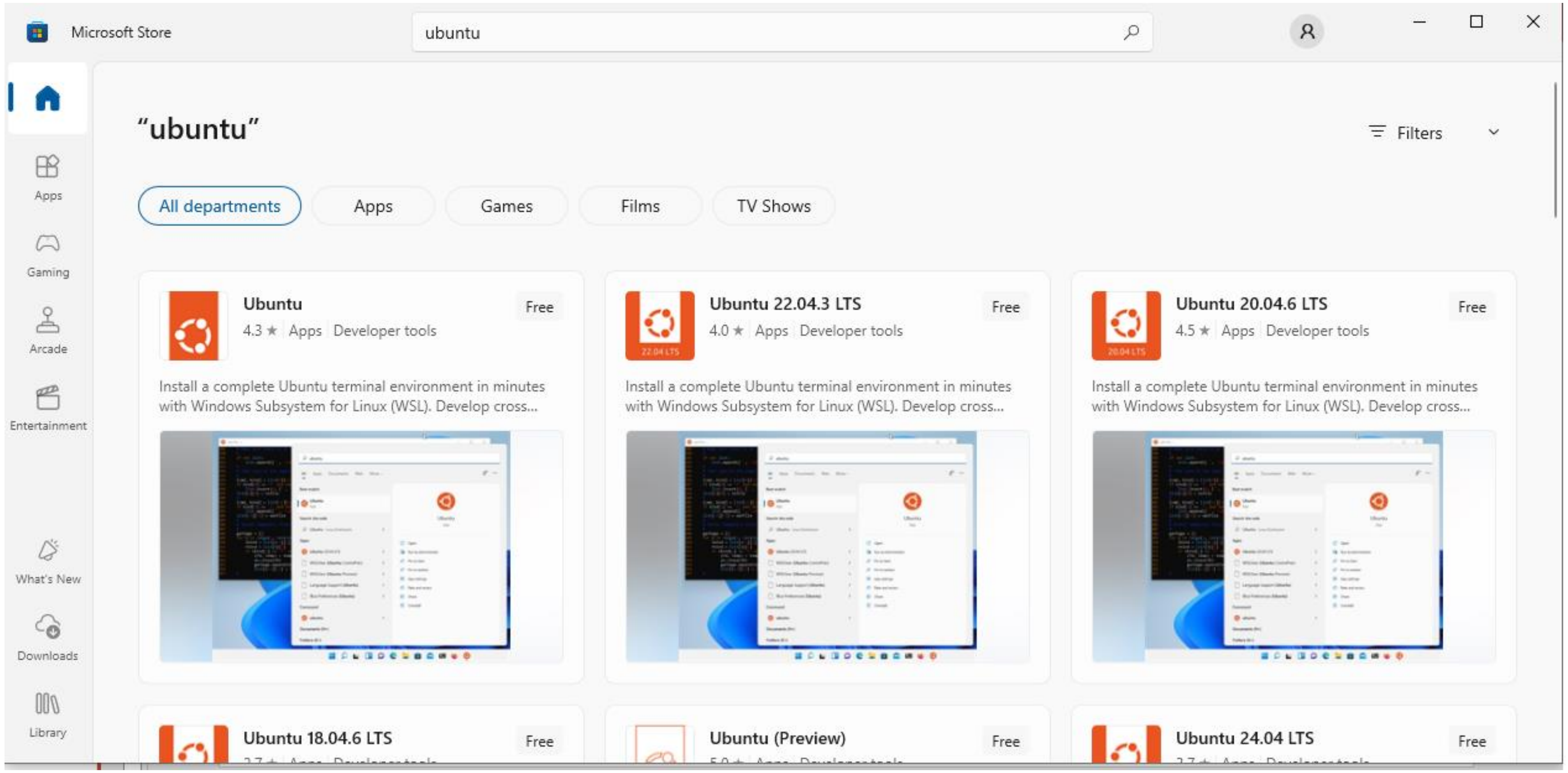
Manual Installation of WSL (6)

Step 6: Install your Linux distribution of choice

Open the Microsoft Store and select your favourite Linux distribution. The link in the Microsoft article does not point to the store so I opened it from the windows start button typing in store. In store I searched for “Ubuntu”. Then select the “Get” button to download the package.

When the installation is complete, from the store, I click the “open” button.

The first time you launch a newly installed Linux distribution, a console window will open and you'll be asked to wait for a minute or two for files to de-compress and be stored on your PC. All future launches should take less than a second.



“ubuntu”

Ubuntu — Free — 4.3 ★ Apps Developer tools
Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross...

Ubuntu 22.04.3 LTS — Free — 4.0 ★ Apps Developer tools
Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross...

Ubuntu 20.04.6 LTS — Free — 4.5 ★ Apps Developer tools
Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross...

Ubuntu 18.04.6 LTS — Free

Ubuntu (Preview) — Free

Ubuntu 24.04 LTS — Free

Manual Installation of WSL (7)

Step 7: The Ubuntu Linux terminal when opened for the first time prompts me to create a user name and password.

```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: elliott
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
Windows Subsystem for Linux is now available in the Microsoft Store!
You can upgrade by running 'wsl.exe --update' or by visiting https://aka.ms/wslstorepage
Installing WSL from the Microsoft Store will give you the latest WSL updates, faster.
For more information please visit https://aka.ms/wslstoreinfo

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This message is shown once a day. To disable it please create the
/home/elliott/.hushlogin file.
elliott@DESKTOP-U93252R:~$
```

Install docker desktop for windows

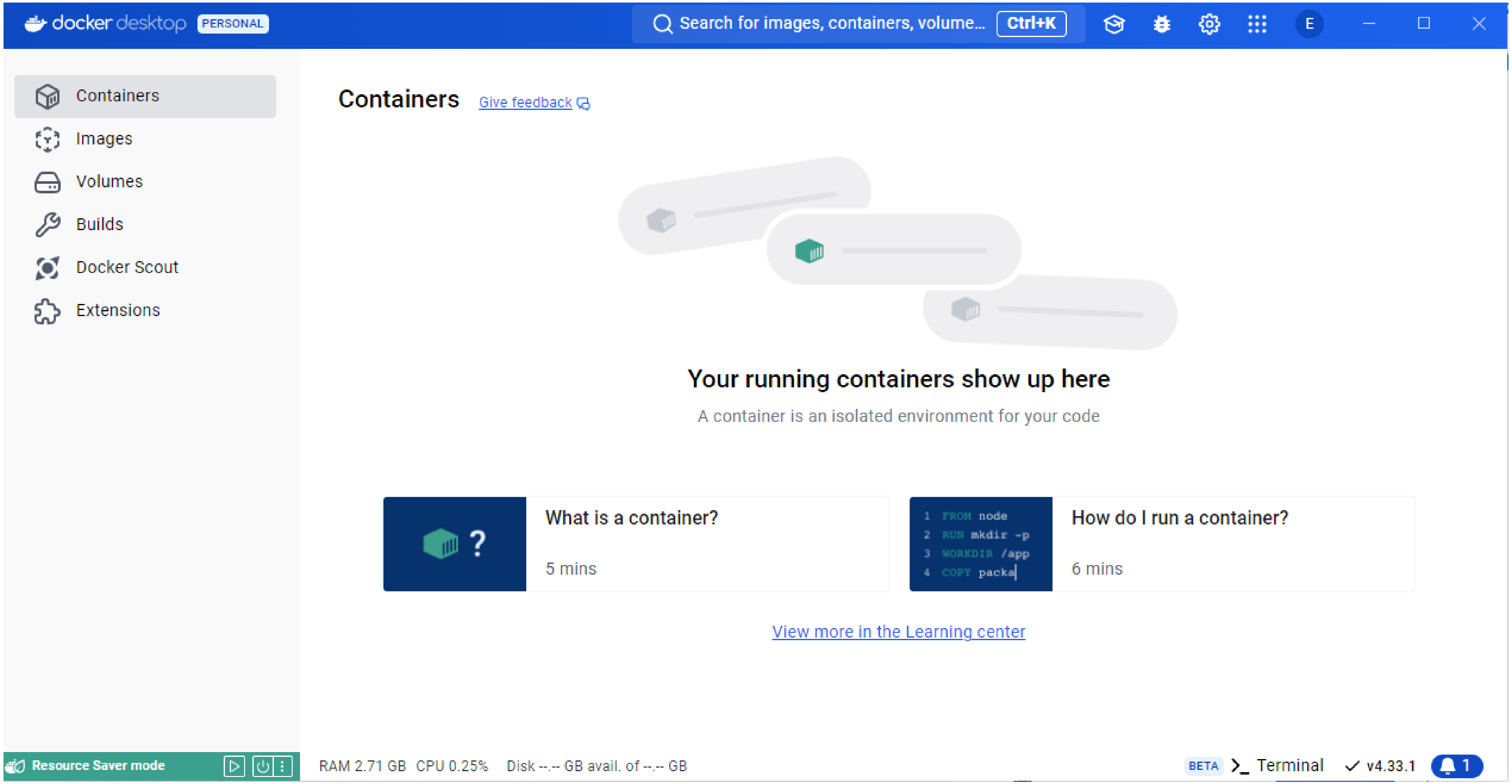
https://docs.docker.com/desktop/install/windows-install/

- WSL version 1.1.3.0 or later.
- Windows 11 64-bit: Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2 or higher.
- Windows 10 64-bit:
  - We recommend **Home or Pro 22H2 (build 19045) or higher**, or Enterprise or Education 22H2 (build 19045) or higher.
  - Minimum required is Home or Pro 21H2 (build 19044) or higher, or Enterprise or Education 21H2 (build 19044) or higher.
- **Turn on the WSL 2 feature on Windows**. For detailed instructions, refer to the Microsoft documentation.
- The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 or Windows 11:
  - **64-bit processor with Second Level Address Translation (SLAT)**
  - **4GB system RAM**
  - Enable **hardware virtualization in BIOS**. For more information, see Virtualization.

For more information on setting up WSL 2 with Docker Desktop, see WSL.

I installed the downloaded .exe and created an account to open Docker desktop

Opening Docker Desktop



Containers Give feedback

Your running containers show up here

A container is an isolated environment for your code

What is a container?

5 mins

```
FROM node
RUN mkdir -p
WORKDIR /app
COPY packa
```

How do I run a container?

6 mins

View more in the Learning center

RAM 2.71 GB CPU 0.25% Disk --.-- GB avail. of --.-- GB

#2 Docker Images & Containers

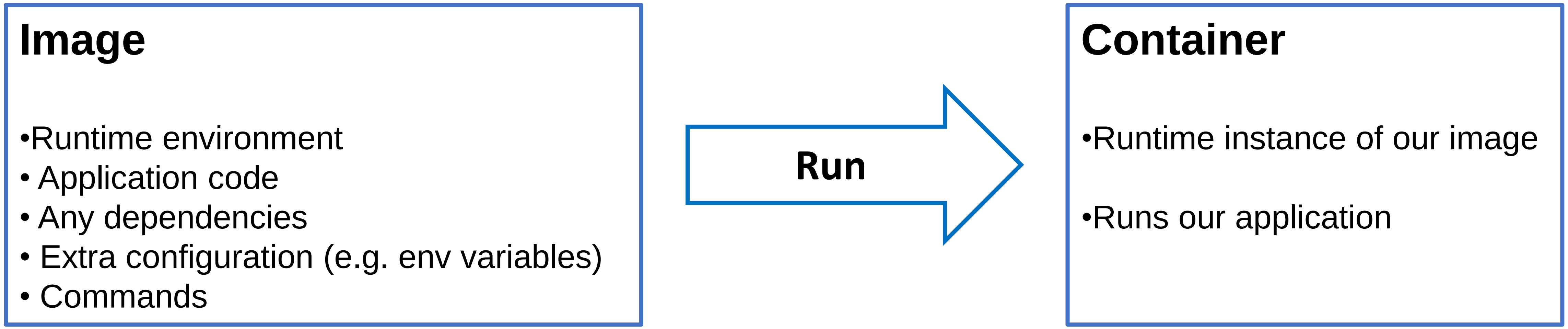
[What are Docker Images & Containers?]

[Docker Images & Containers]

[Docker image components]

What are Docker Images & Containers?

Docker Images are like blueprints for containers and contain the following stored inside them:



Image

- Runtime environment
- Application code
- Any dependencies
- Extra configuration (e.g. env variables)
- Commands

Container

- Runtime instance of our image
- Runs our application

Images also have their own file system which is independent of the computer. Images are read only which means that once it is created it cannot be changed. If you need to change something about an image then you need to create a new image.

Containers are runnable instances of those images. When we run an image it creates a container which is a process which runs our application as per outlined in the image so it will have the correct runtime environment, application code, dependencies and extra config.

Docker Images & Containers

Isolated Process

Container

- Runtime instance of our image
- Runs our application

Containers are an isolated process meaning that they run independently from any other process on the computer so it is a bit like our applications being run in in it's own box somewhere on our computer.

I can therefore make an image that contains everything I need to make that application run packaged inside of it. (the OS, the NodeJS version or python version, dependencies, source code etc)

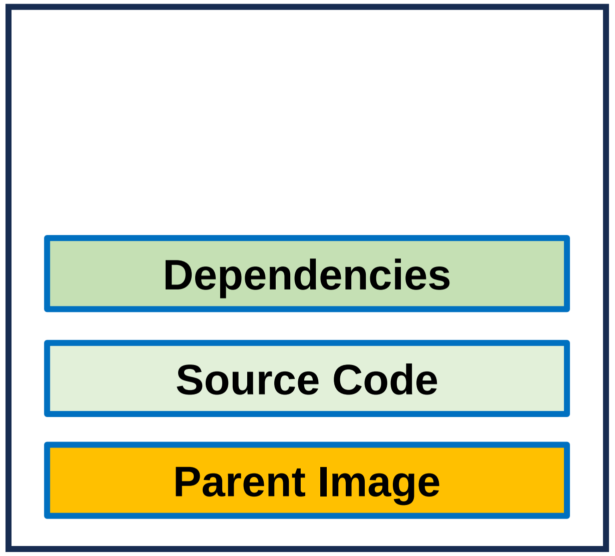
Then I can run that image to create a container to run my application and it does not matter what version of NodeJS or Python or whatever it is that I am using for the application that's installed on my computer because it's all running inside the container.

Because the image has the correct versions of everything inside it for it to run. This means that I can just share the docker image with anyone that needs to run the application independent of what versions they have installed on their computer because that is all prepacked inside the image.

This means that the image can be run anywhere such as another computer or production server.

Docker image components

Images are made up of several "layers" where each layer adds something else to the image incrementally so the order of the layers does matter. Normally we start with a parent image



Parent Image: Describes the OS & sometimes the runtime environment.

So we could have a specific parent image that has NodeJS 17 on a Linux distro. This parent layer in itself is a premade docker image so we are just creating a new image on top of the parent image. It usually contains a light operating system and runtime environment.

The next layers that we build on top of that parent image can be anything else that we would add to our image such as copying source code to the image and dependencies.

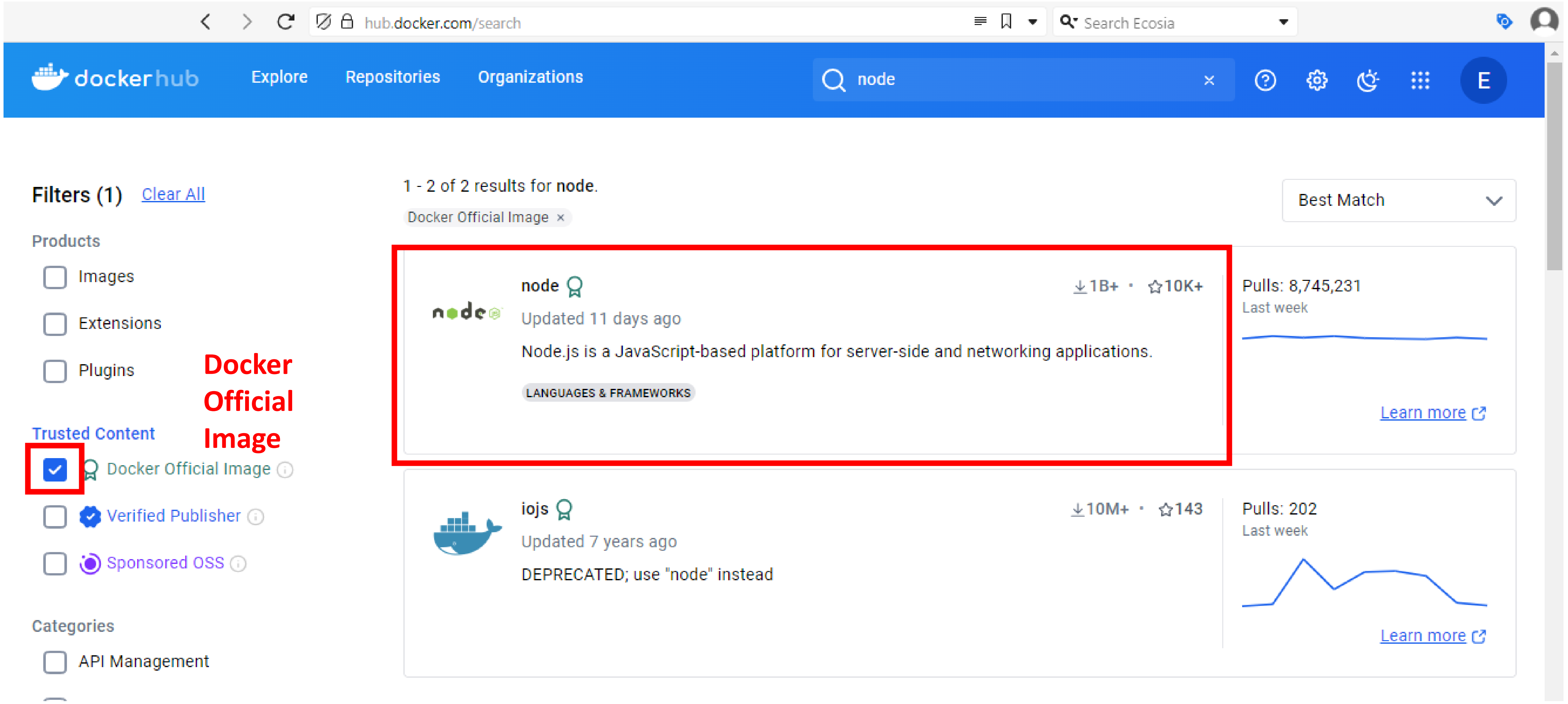
Parent images can be found at docker Hub which is like git hub but contains an an online repository of images
https://hub.docker.com

Lets say that we want an image that runs NodeJS then the initial layer of our image would be a parent node image.

#3 Parent Images & Docker hub

Parent images from Docker Hub

In docker hub I run a search for Node which returns multiple results. I can filter results to only show official Docker Images.



hub.docker.com/search

Search Ecosia

Filters (1) Clear All

Products

- [ ] Images
- [ ] Extensions
- [ ] Plugins

Trusted Content

- [✓] Docker Official Image
- [ ] Verified Publisher
- [ ] Sponsored OSS

Categories

- [ ] API Management

Docker Official Image

1 - 2 of 2 results for **node**.

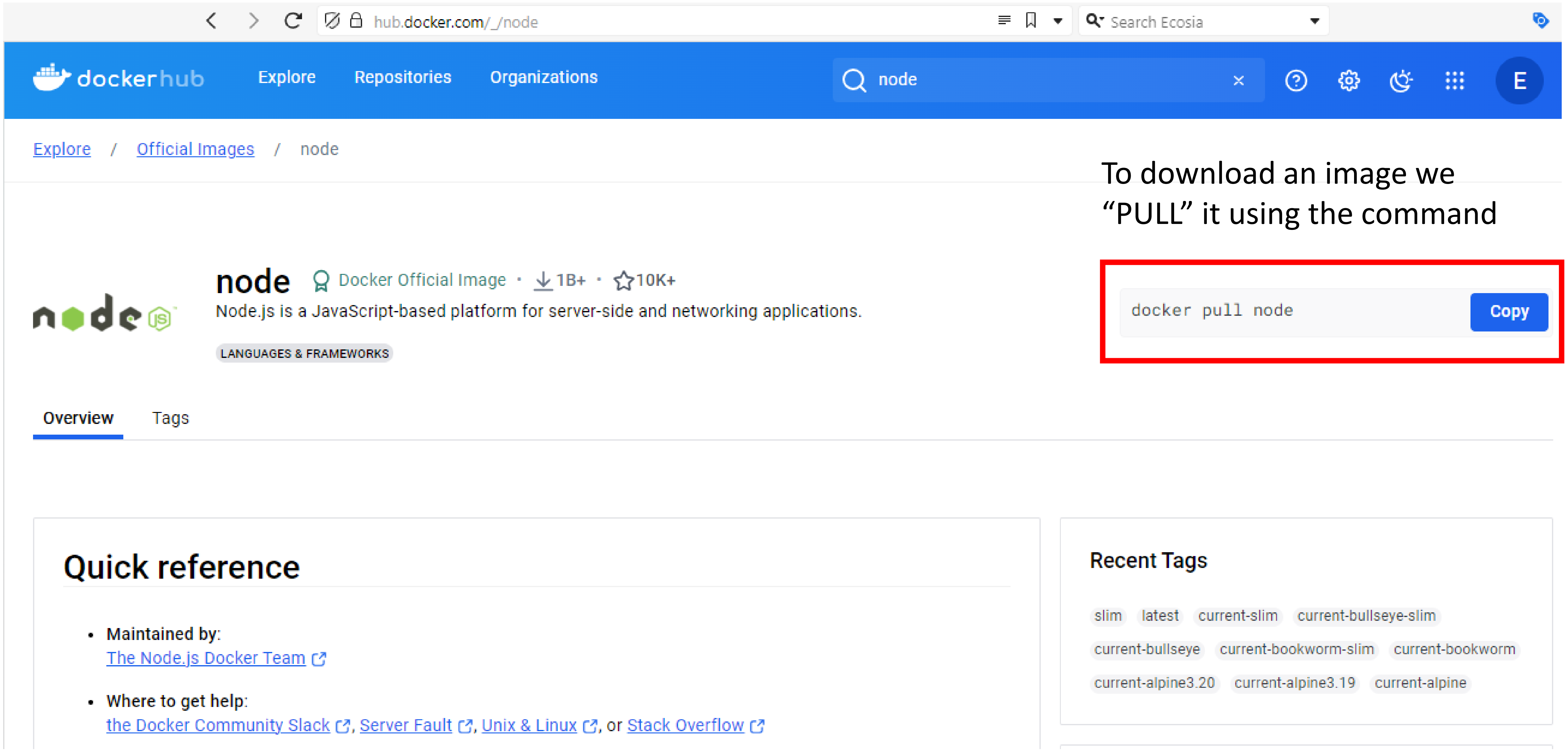
Docker Official Image ×

Best Match

node
↓1B+ • ☆10K+
Updated 11 days ago
Node.js is a JavaScript-based platform for server-side and networking applications.
LANGUAGES & FRAMEWORKS
Pulls: 8,745,231
Last week
Learn more

iojs
↓10M+ • ☆143
Updated 7 years ago
DEPRECATED; use "node" instead
Pulls: 202
Last week
Learn more

Docker Parent image pull



hub.docker.com/_/node

Explore / Official Images / node

node Docker Official Image • 1B+ • 10K+

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

LANGUAGES & FRAMEWORKS

```
docker pull node
```

Quick reference

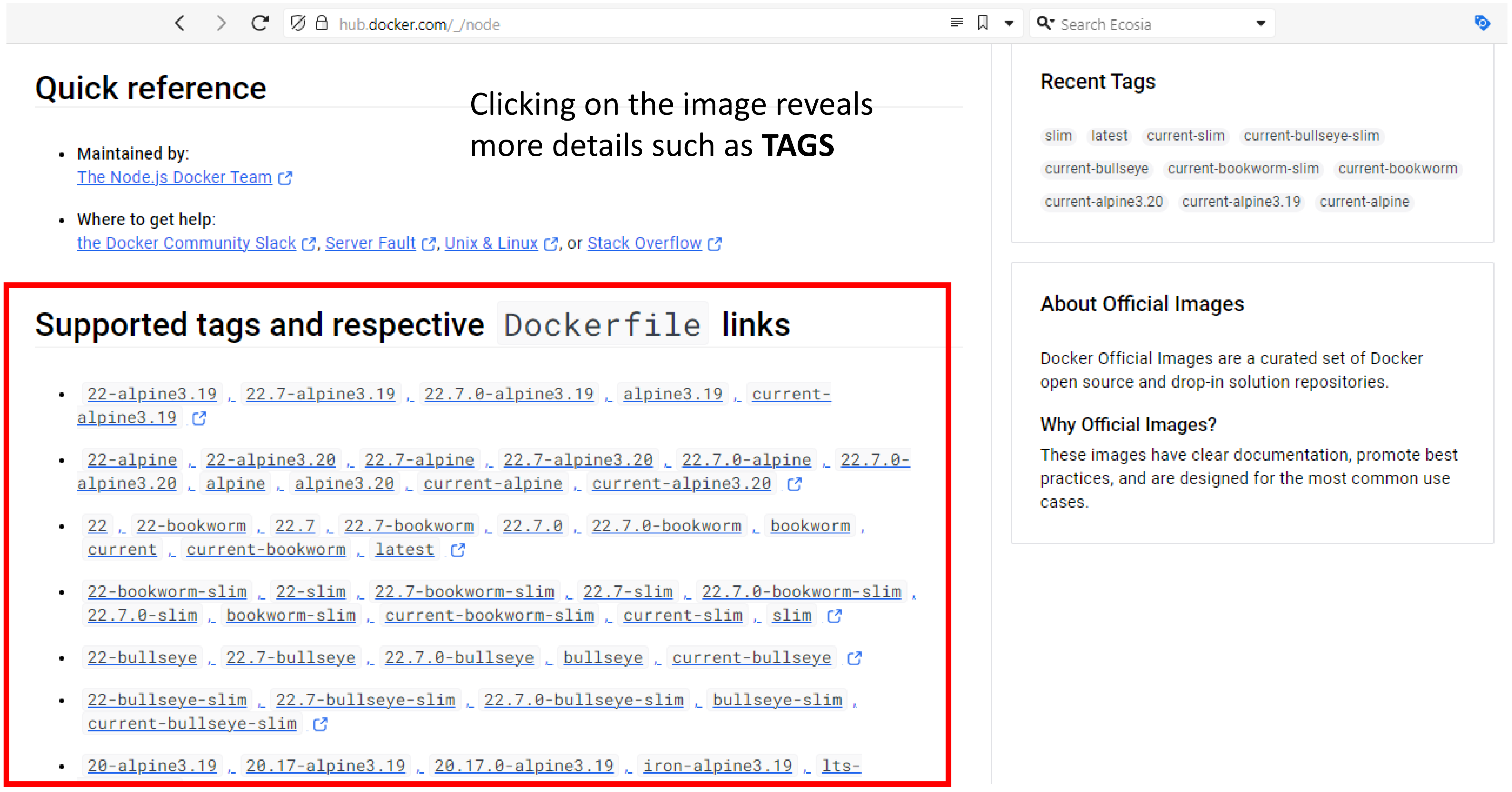
- Maintained by:
  The Node.js Docker Team
- Where to get help:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Recent Tags

slim latest current-slim current-bullseye-slim current-bullseye current-bookworm-slim current-bookworm current-alpine3.20 current-alpine3.19 current-alpine

To download an image we "PULL" it using the command

Docker Parent Image Details



hub.docker.com/_/node

Search Ecosia

Quick reference

- Maintained by:
  The Node.js Docker Team

- Where to get help:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Clicking on the image reveals more details such as **TAGS**

Supported tags and respective `Dockerfile` links

- `22-alpine3.19`, `22.7-alpine3.19`, `22.7.0-alpine3.19`, `alpine3.19`, `current-alpine3.19`
- `22-alpine`, `22-alpine3.20`, `22.7-alpine`, `22.7-alpine3.20`, `22.7.0-alpine`, `22.7.0-alpine3.20`, `alpine`, `alpine3.20`, `current-alpine`, `current-alpine3.20`
- `22`, `22-bookworm`, `22.7`, `22.7-bookworm`, `22.7.0`, `22.7.0-bookworm`, `bookworm`, `current`, `current-bookworm`, `latest`
- `22-bookworm-slim`, `22-slim`, `22.7-bookworm-slim`, `22.7-slim`, `22.7.0-bookworm-slim`, `22.7.0-slim`, `bookworm-slim`, `current-bookworm-slim`, `current-slim`, `slim`
- `22-bullseye`, `22.7-bullseye`, `22.7.0-bullseye`, `bullseye`, `current-bullseye`
- `22-bullseye-slim`, `22.7-bullseye-slim`, `22.7.0-bullseye-slim`, `bullseye-slim`, `current-bullseye-slim`
- `20-alpine3.19`, `20.17-alpine3.19`, `20.17.0-alpine3.19`, `iron-alpine3.19`, `lts-`

Recent Tags

slim latest current-slim current-bullseye-slim current-bullseye current-bookworm-slim current-bookworm current-alpine3.20 current-alpine3.19 current-alpine

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

Docker Pull command

```
PS C:\Users\ellio> docker pull node
Using default tag: latest
latest: Pulling from library/node
903681d87777: Pull complete
3cbbe86a28c2: Pull complete
6ed93aa58a52: Pull complete
787c78da4383: Pull complete
43646240185: Pull complete
d59df365b3bf: Pull complete
24505dd295d9: Pull complete
cafde2261323: Pull complete
Digest:
sha256:54b7a9a6bb4ebfb623b5163581426b83f0a
b39292e4df2c808ace95ab4cba94f
Status: Downloaded newer image for
node:latest
docker.io/library/node:latest
PS C:\Users\ellio>
```

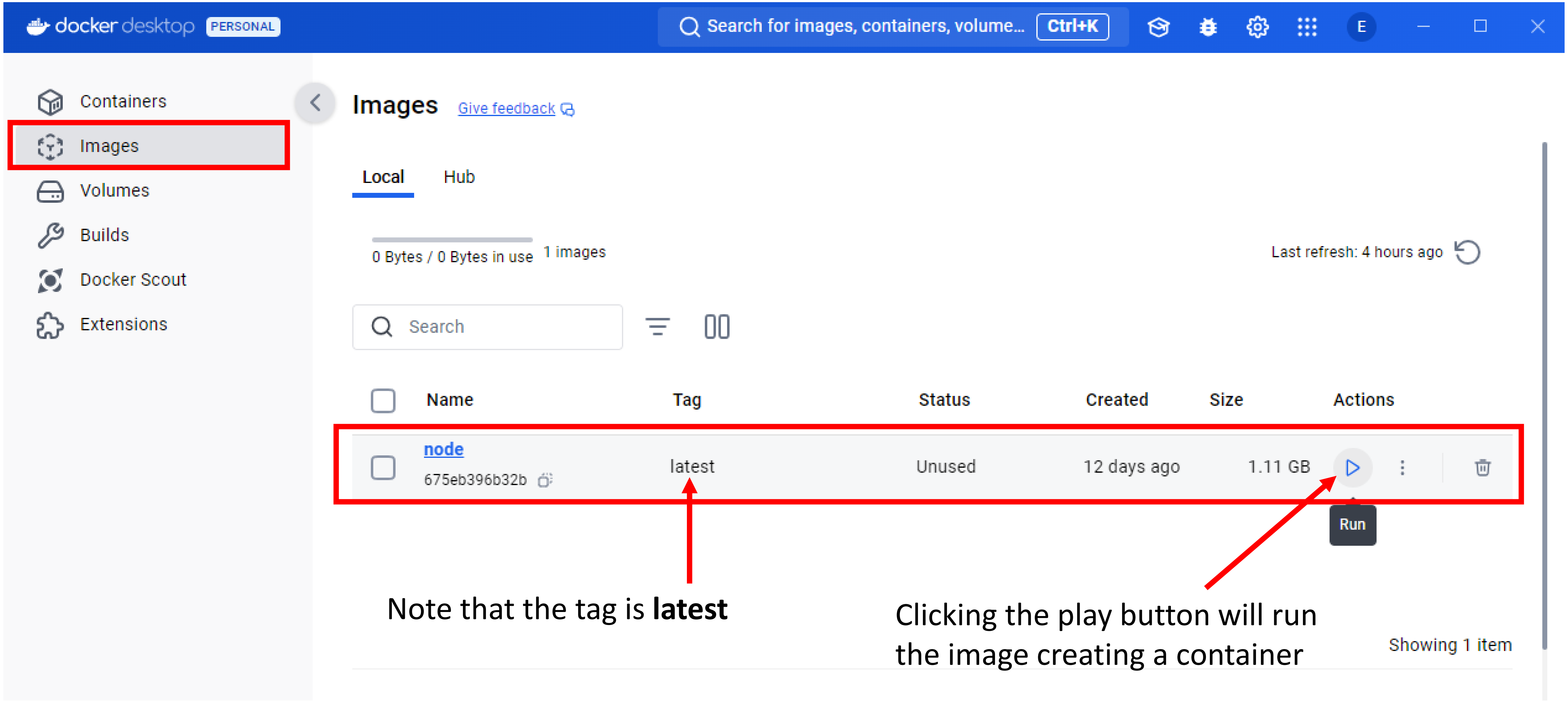
Tags are optional parameters that specify thigs such as the version of the image and the underlying Linux distribution.

For example we could chose Node version 18 running on alpine (alpine is a light weight Linux distro). This would give us the optional parameters of "18.20.4-alpine".

It is always better to specify the version of the image otherwise Docker hub will download the latest version (default tag).

Docker commands are run in a terminal, i.e. PowerShell. Note that it does not matter where we do the pull from (in this case C:/users/Ellio>) because docker will store the image in a special place.

Docker Hub Images



docker desktop PERSONAL

Search for images, containers, volume... Ctrl+K

Containers
Images
Volumes
Builds
Docker Scout
Extensions

Images Give feedback

Local Hub

0 Bytes / 0 Bytes in use 1 images

Last refresh: 4 hours ago

Search

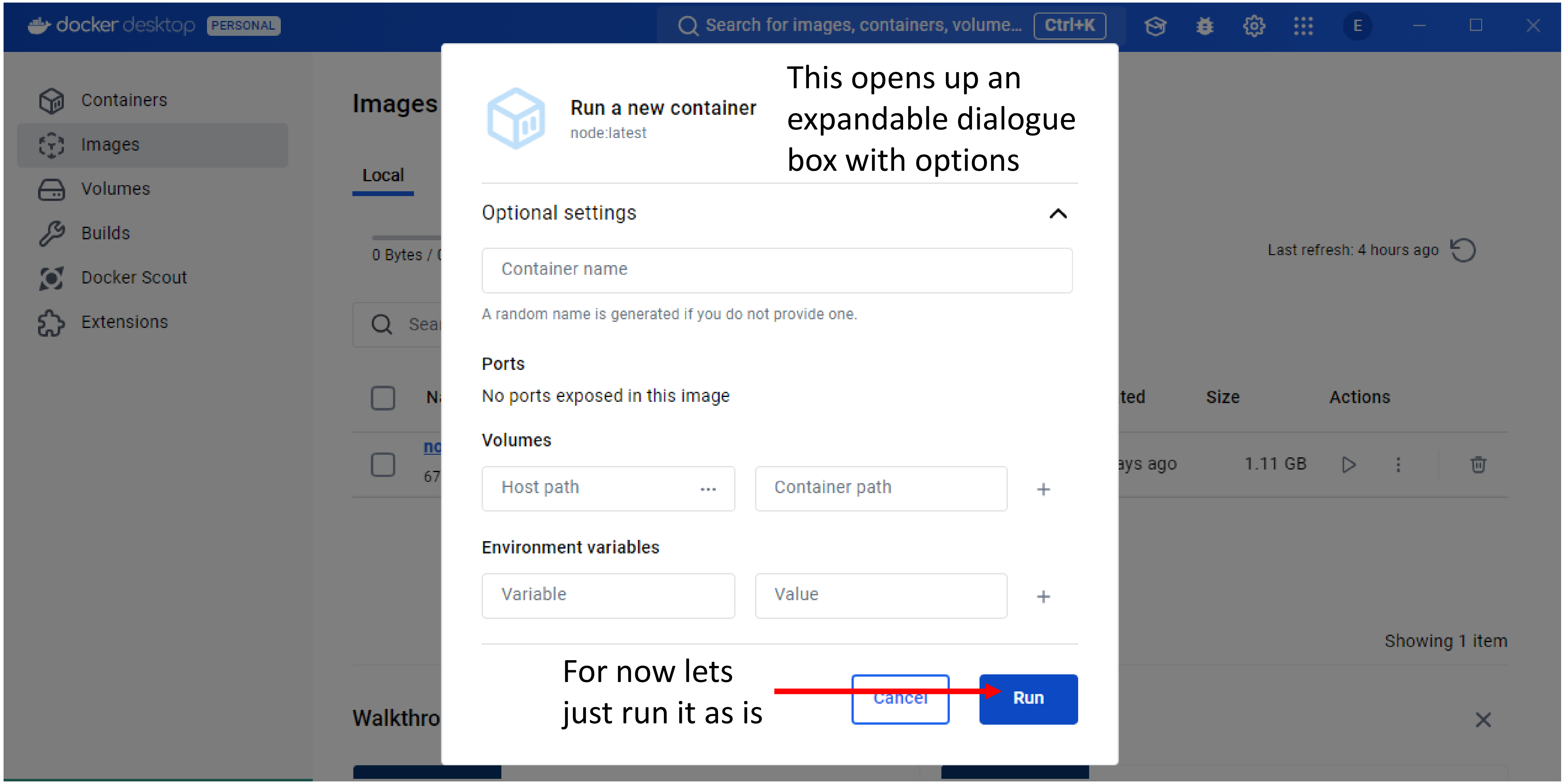
Name	Tag	Status	Created	Size	Actions
node 675eb396b32b	latest	Unused	12 days ago	1.11 GB	Run

Showing 1 item

Note that the tag is **latest**

Clicking the play button will run the image creating a container

Run Docker Image



Run a new container
node:latest

This opens up an expandable dialogue box with options

Optional settings

Container name

A random name is generated if you do not provide one.

Ports

No ports exposed in this image

Volumes

Host path ... Container path +

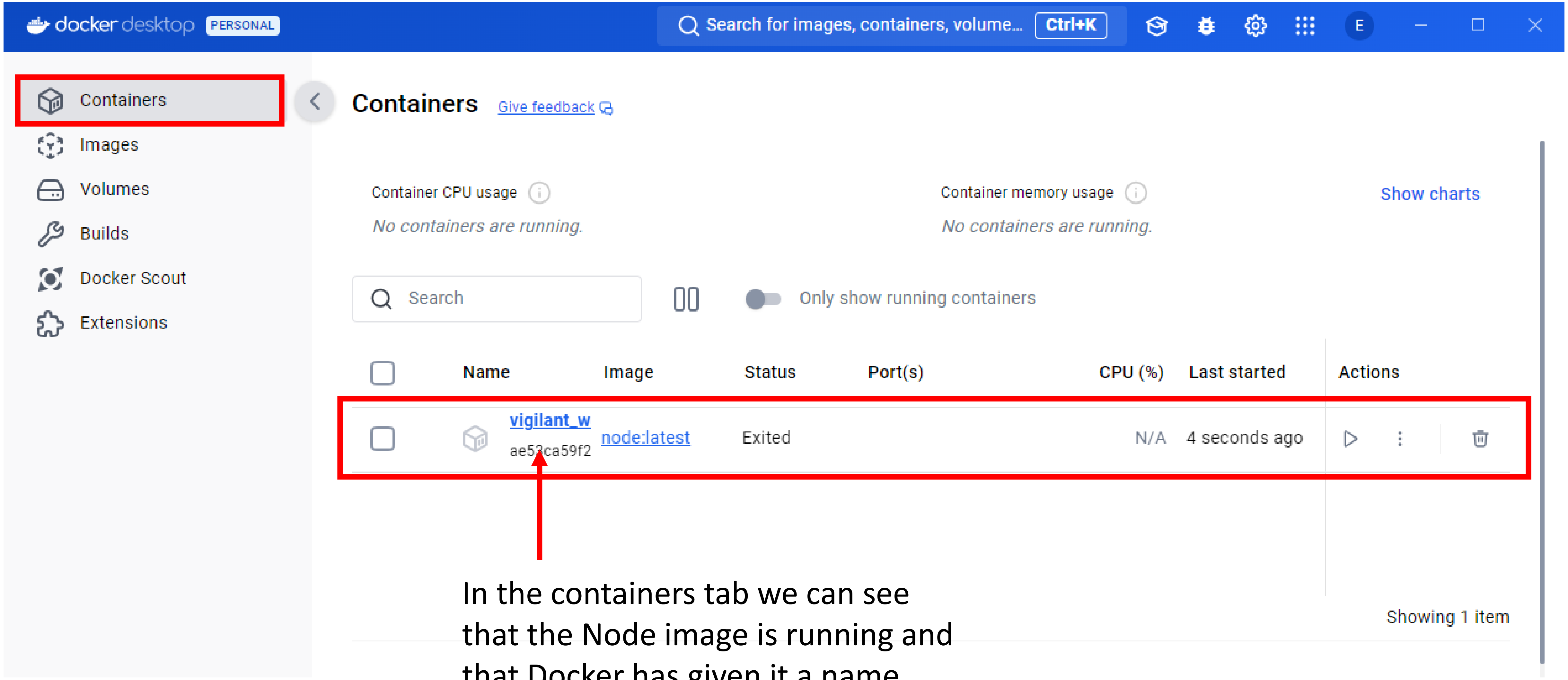
Environment variables

Variable Value +

For now lets just run it as is

Cancel Run

Run Docker image Container



Containers

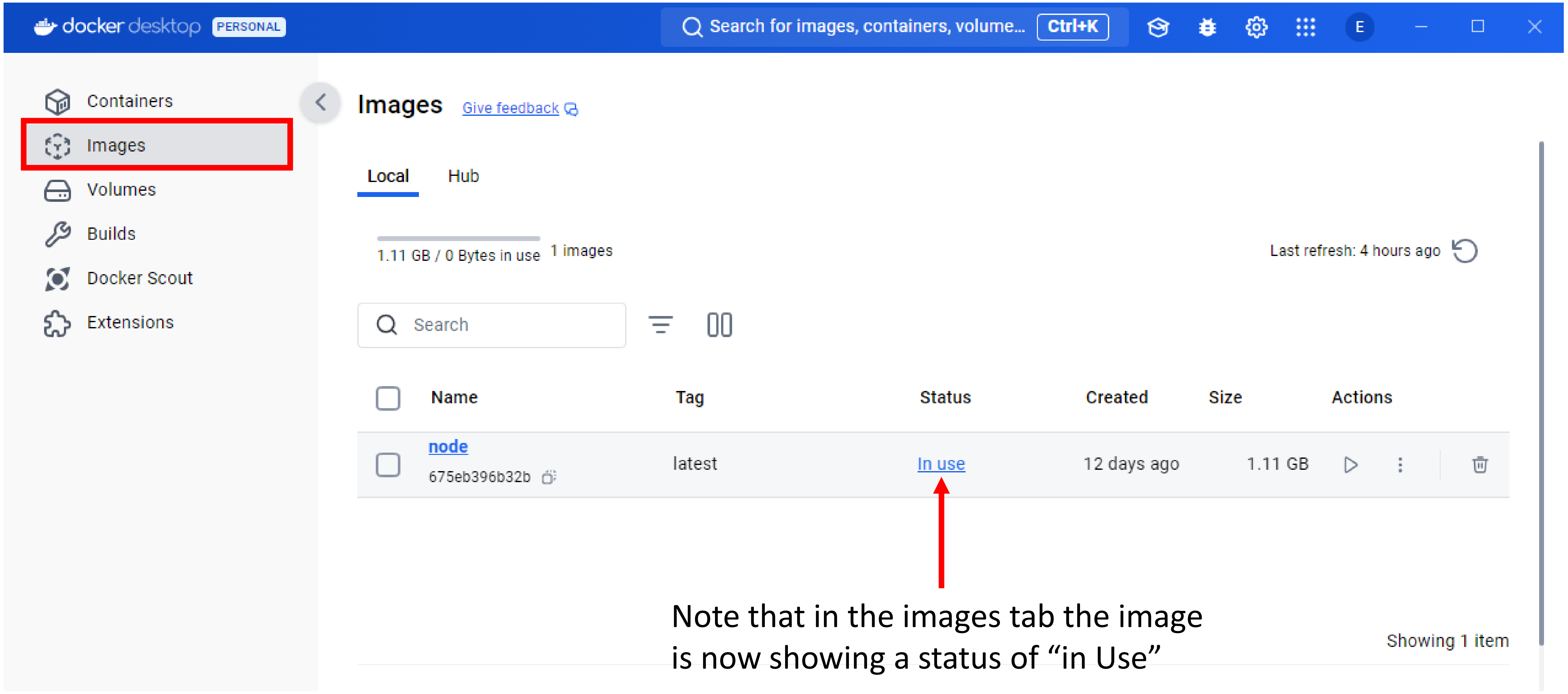
Container CPU usage
No containers are running.

Container memory usage
No containers are running.

	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
	vigilant_w ae52ca59f2	node:latest	Exited		N/A	4 seconds ago	

In the containers tab we can see that the Node image is running and that Docker has given it a name

Docker desktop Image status



1.11 GB / 0 Bytes in use 1 images

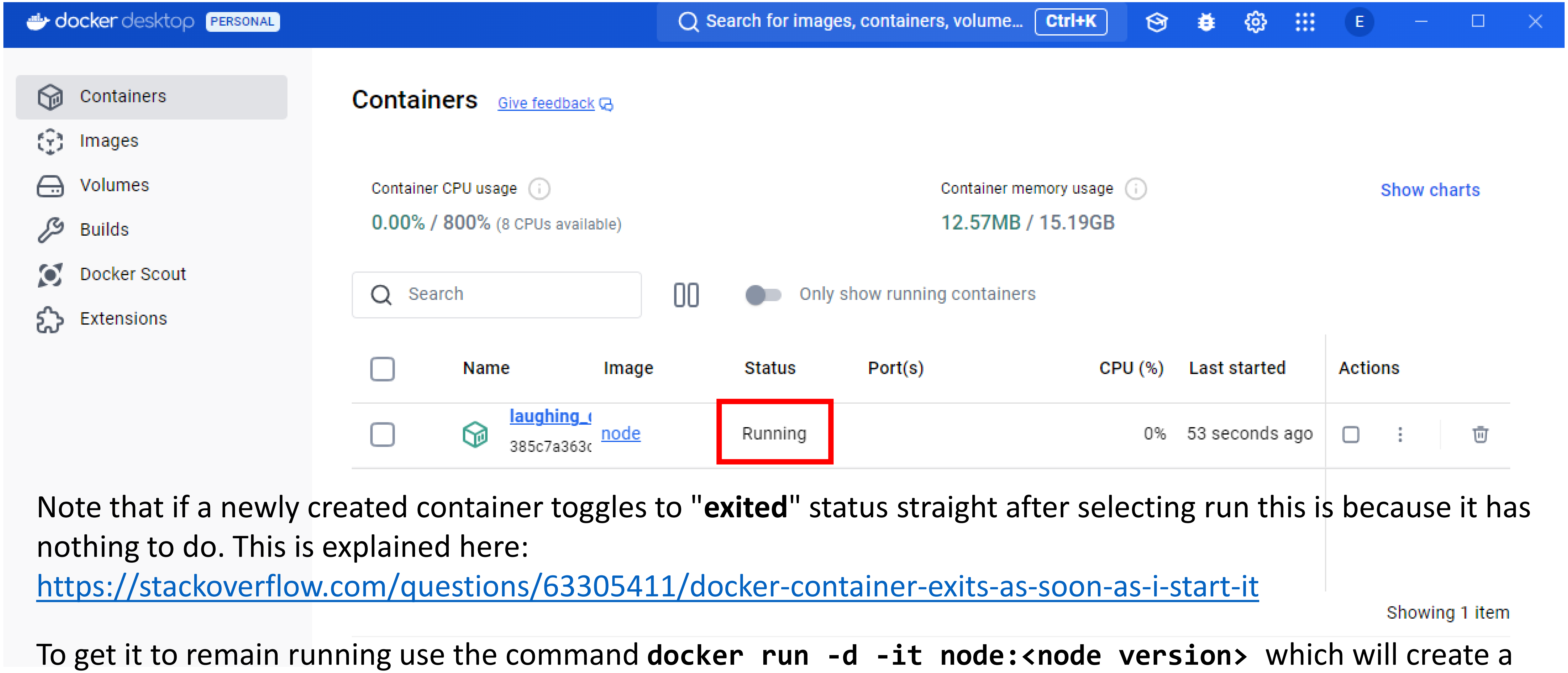
Last refresh: 4 hours ago

Name	Tag	Status	Created	Size	Actions
node 675eb396b32b	latest	In use	12 days ago	1.11 GB	

Showing 1 item

Note that in the images tab the image is now showing a status of "in Use"

Docker exited status



Containers

Container CPU usage
0.00% / 800% (8 CPUs available)

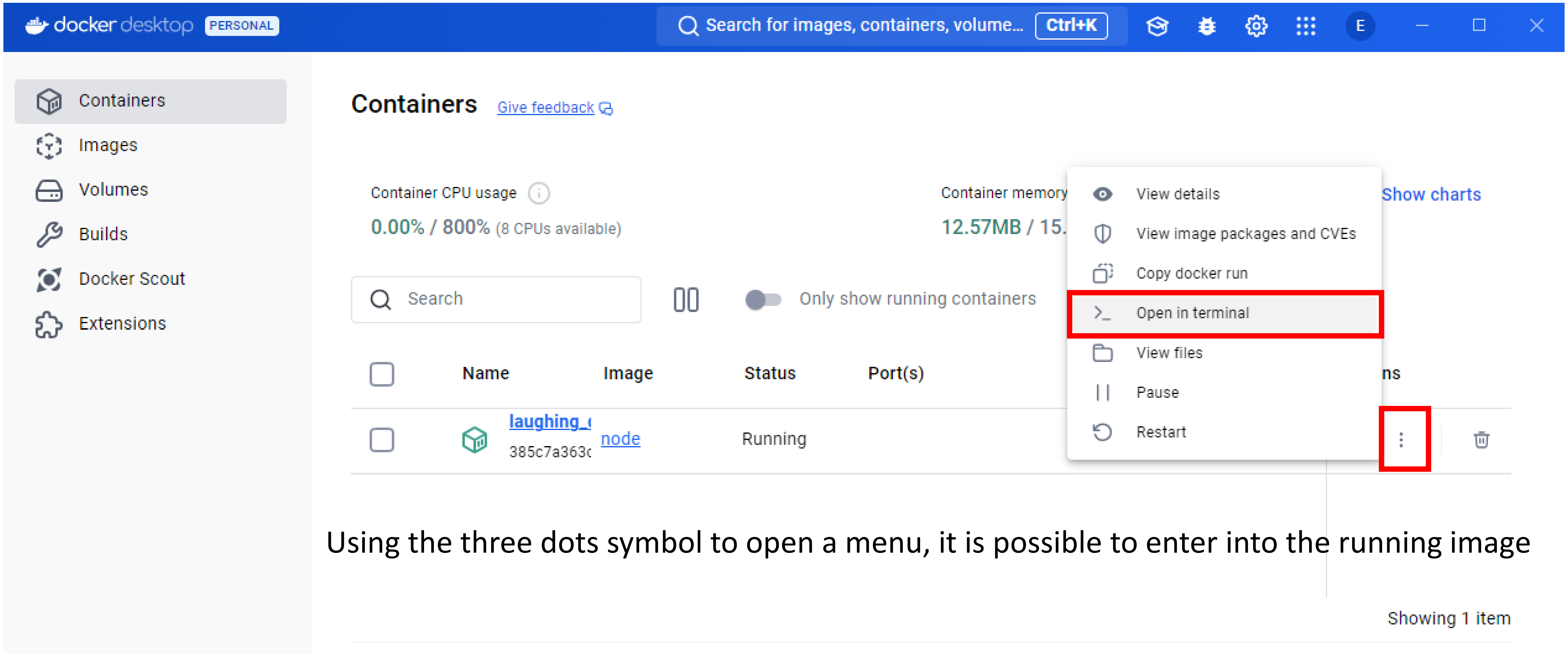
Container memory usage
12.57MB / 15.19GB

	Name	Image	Status	Port(s)	CPU (%)	Last started
	laughing_(385c7a363(	node	Running		0%	53 seconds ago

Note that if a newly created container toggles to "**exited**" status straight after selecting run this is because it has nothing to do. This is explained here:
https://stackoverflow.com/questions/63305411/docker-container-exits-as-soon-as-i-start-it

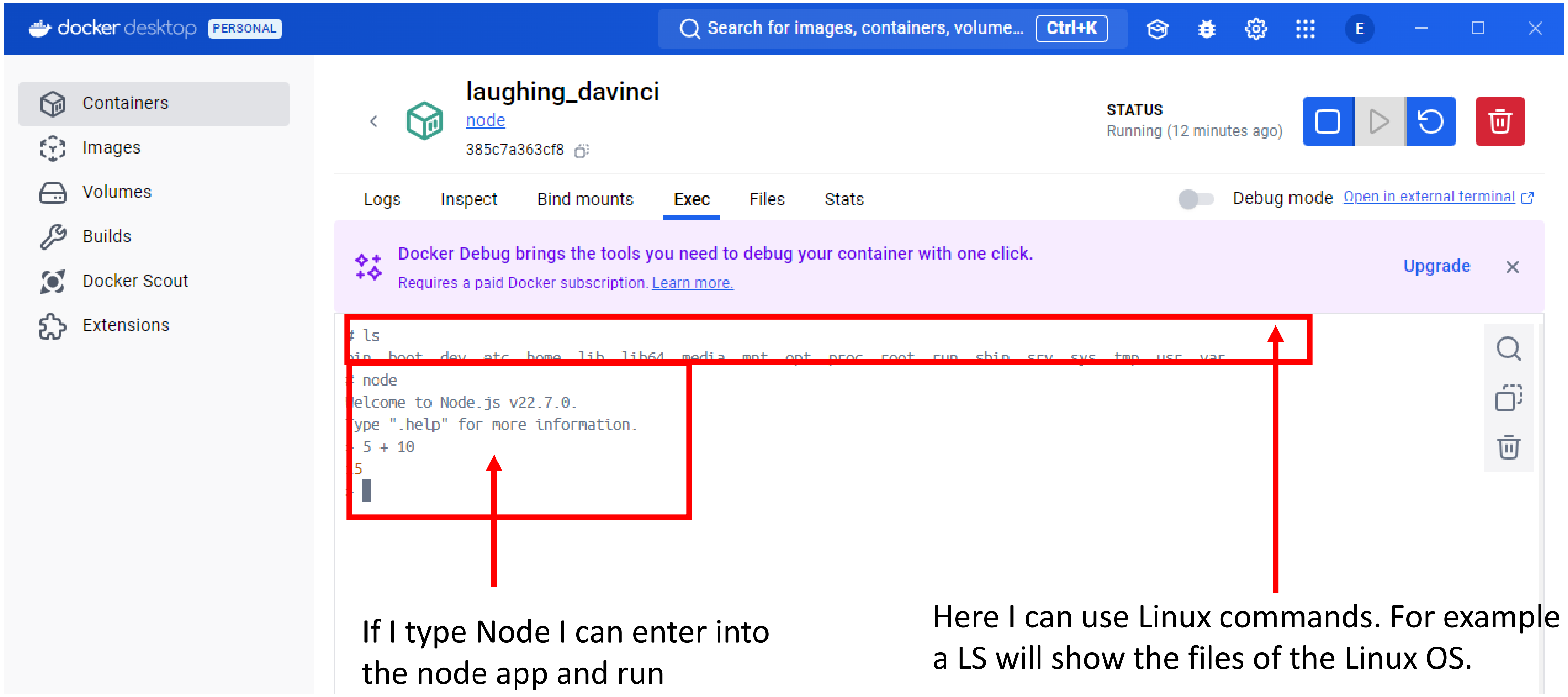
To get it to remain running use the command **`docker run -d -it node:<node version>`** which will create a new container which will stay running. the old container can be deleted.

CLI to Running Container



Using the three dots symbol to open a menu, it is possible to enter into the running image

CLI commands in Docker image OS and application



Docker Debug brings the tools you need to debug your container with one click.

Requires a paid Docker subscription. Learn more.

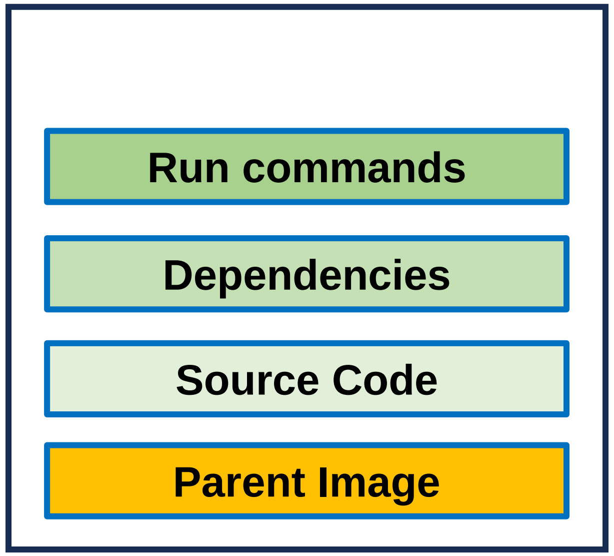
```
# ls
in  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# node
elcome to Node.js v22.7.0.
ype ".help" for more information.
 5 + 10
5
```

If I type Node I can enter into the node app and run commands. For example 10 + 5.

Here I can use Linux commands. For example a LS will show the files of the Linux OS.

#4 Dockerfile

What is a Dockerfile?



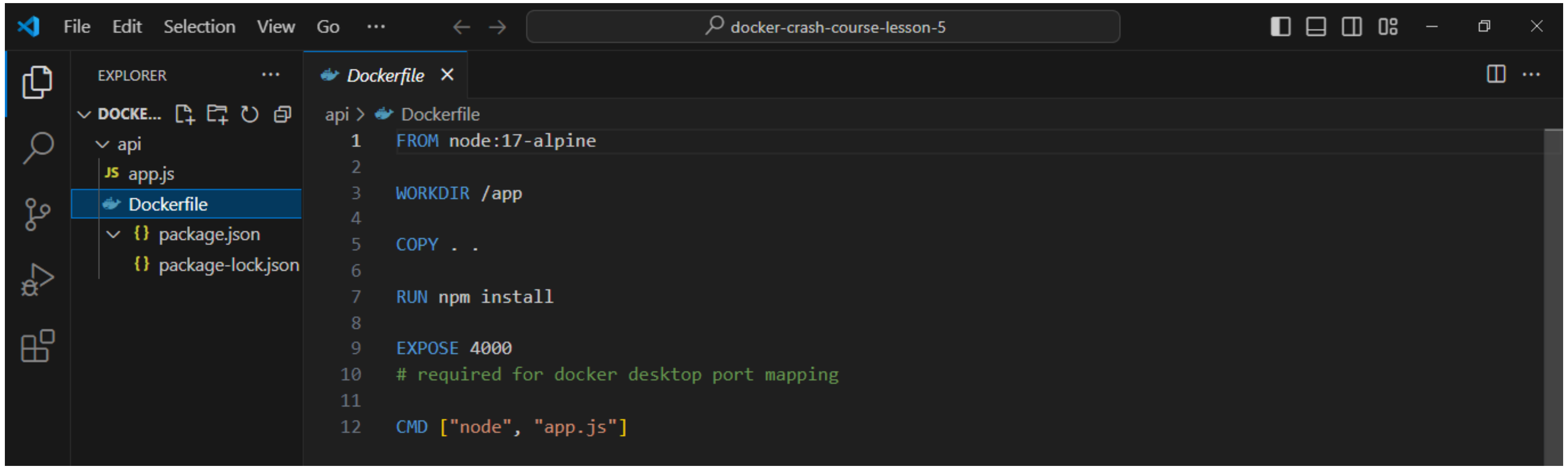
Docker images are made up of different layers consisting first of a parent image with layers on top that customise the image to do what we want.

To do this we need to create a Dockerfile which is like a set of instructions to docker on how to create these layers on an image.

The dockerfile is a file that instructs Docker how to create the image. It is like a set of instructions on how to create the image.

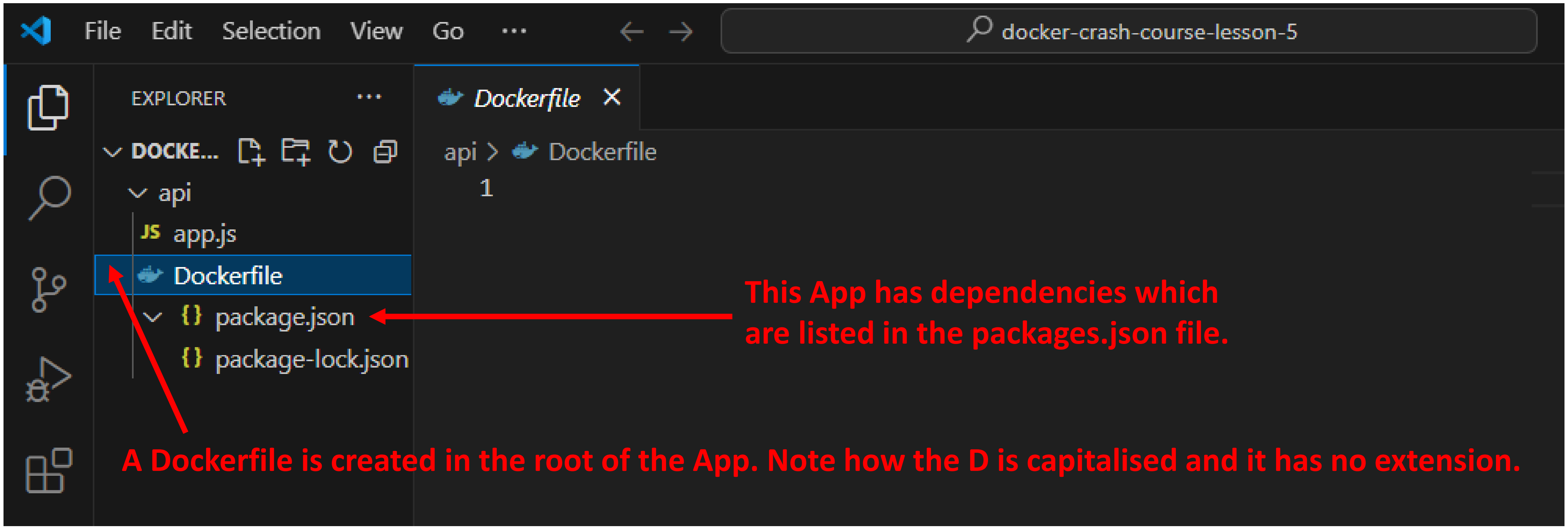
To continue we need to download the sample node app where this lesson is in branch 5. https://github.com/iamshaunjp/docker-crash-course/tree/lesson-5

The unzipped folder of the lesson App can be opened in VS Code



```
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 4000
# required for docker desktop port mapping

CMD ["node", "app.js"]
```

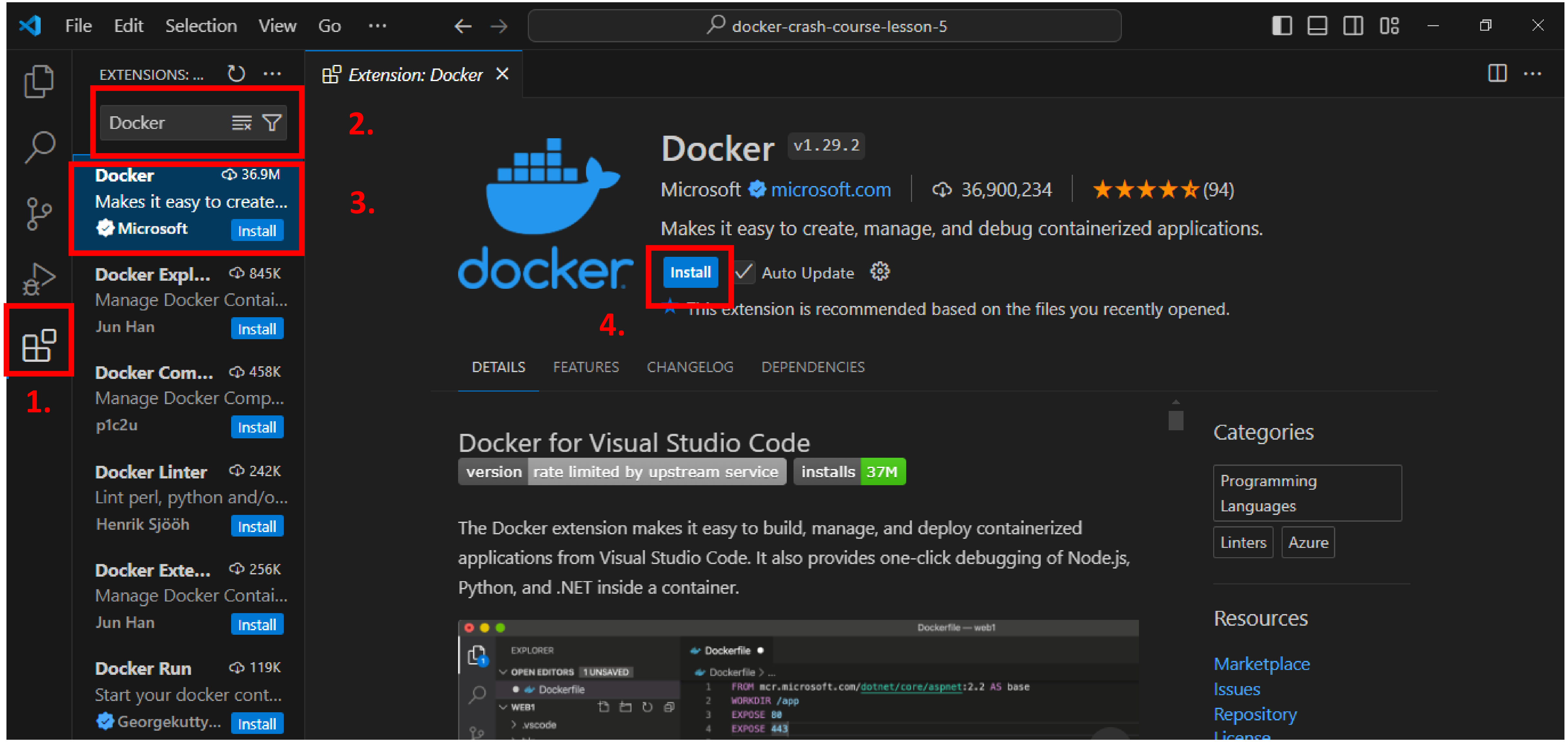
How to create a Dockerfile



This App has dependencies which are listed in the packages.json file.

A Dockerfile is created in the root of the App. Note how the D is capitalised and it has no extension.

Install VS Code Docker extensions



Docker 36.9M
Makes it easy to create...
Microsoft Install

Docker Expl... 845K
Manage Docker Contai...
Jun Han Install

Docker Com... 458K
Manage Docker Comp...
p1c2u Install

Docker Linter 242K
Lint perl, python and/o...
Henrik Sjööh Install

Docker Exte... 256K
Manage Docker Contai...
Jun Han Install

Docker Run 119K
Start your docker cont...
Georgekutty... Install

Docker v1.29.2

Microsoft microsoft.com | 36,900,234 | ★★★★★ (94)

Makes it easy to create, manage, and debug containerized applications.

This extension is recommended based on the files you recently opened.

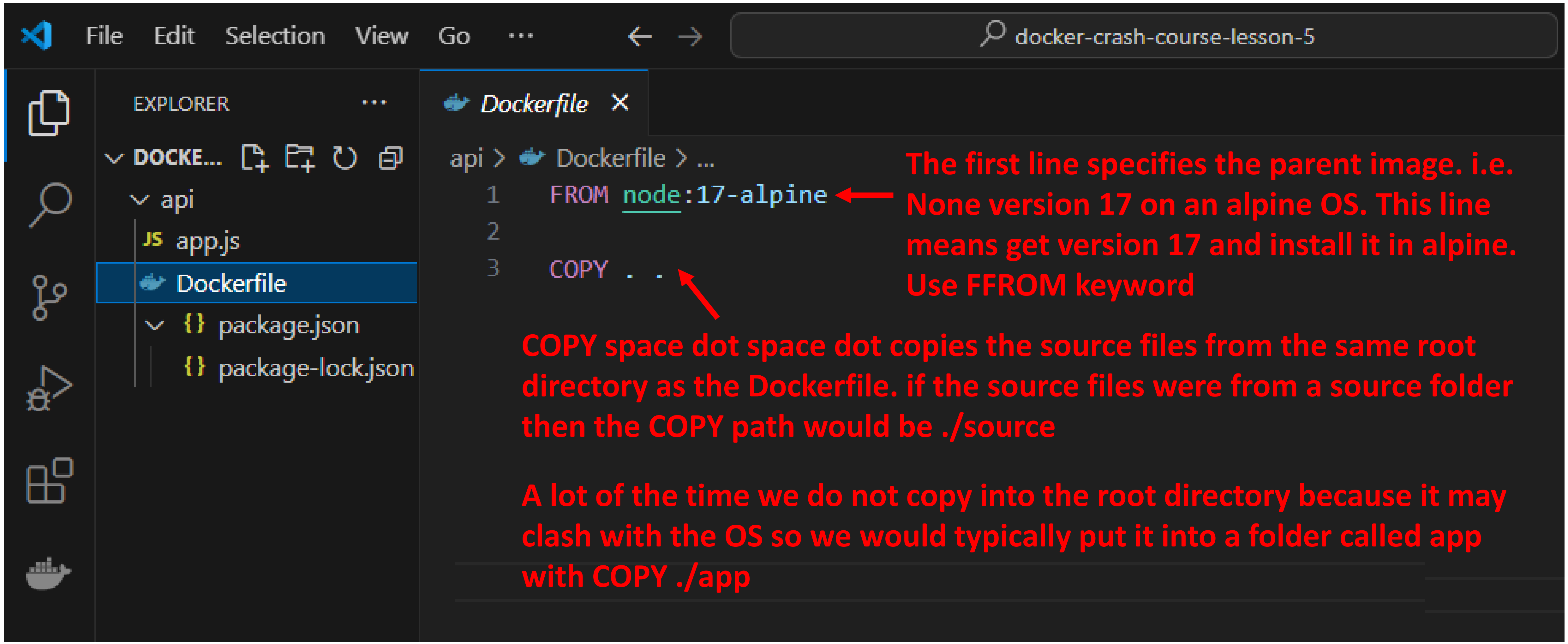
Docker for Visual Studio Code

version | rate limited by upstream service | installs | 37M

The Docker extension makes it easy to build, manage, and deploy containerized applications from Visual Studio Code. It also provides one-click debugging of Node.js, Python, and .NET inside a container.

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443
```

Create a Dockerfile (1)



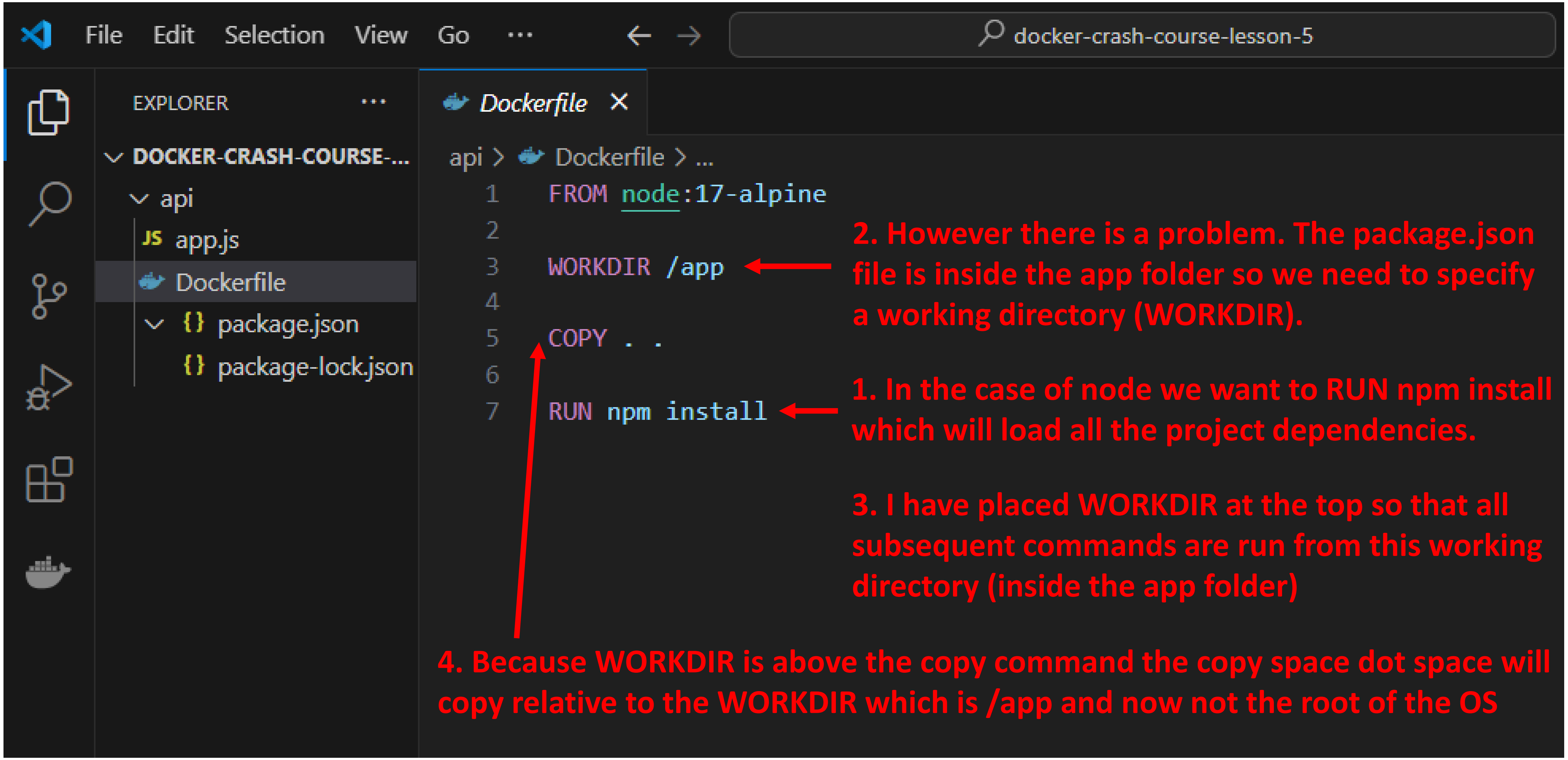
```
FROM node:17-alpine

COPY . .
```

The first line specifies the parent image. i.e. None version 17 on an alpine OS. This line means get version 17 and install it in alpine. Use FFROM keyword

COPY space dot space dot copies the source files from the same root directory as the Dockerfile. if the source files were from a source folder then the COPY path would be ./source

A lot of the time we do not copy into the root directory because it may clash with the OS so we would typically put it into a folder called app with COPY ./app

Create a Dockerfile (2)



```
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install
```

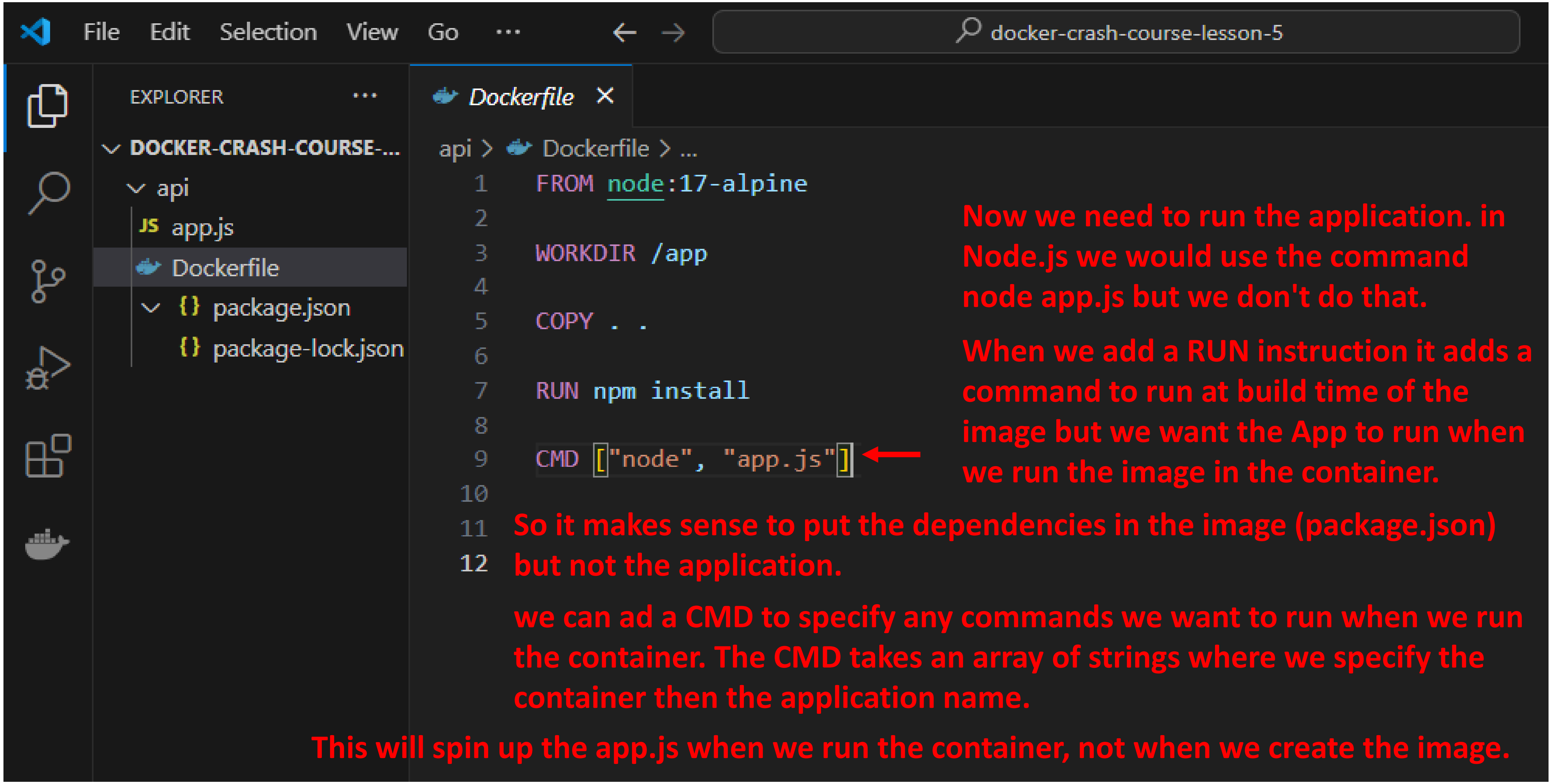
2. However there is a problem. The package.json file is inside the app folder so we need to specify a working directory (WORKDIR).

1. In the case of node we want to RUN npm install which will load all the project dependencies.

3. I have placed WORKDIR at the top so that all subsequent commands are run from this working directory (inside the app folder)

4. Because WORKDIR is above the copy command the copy space dot space will copy relative to the WORKDIR which is /app and now not the root of the OS

Create a Dockerfile (3)



```
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install

CMD ["node", "app.js"]
```

Now we need to run the application. in Node.js we would use the command node app.js but we don't do that.

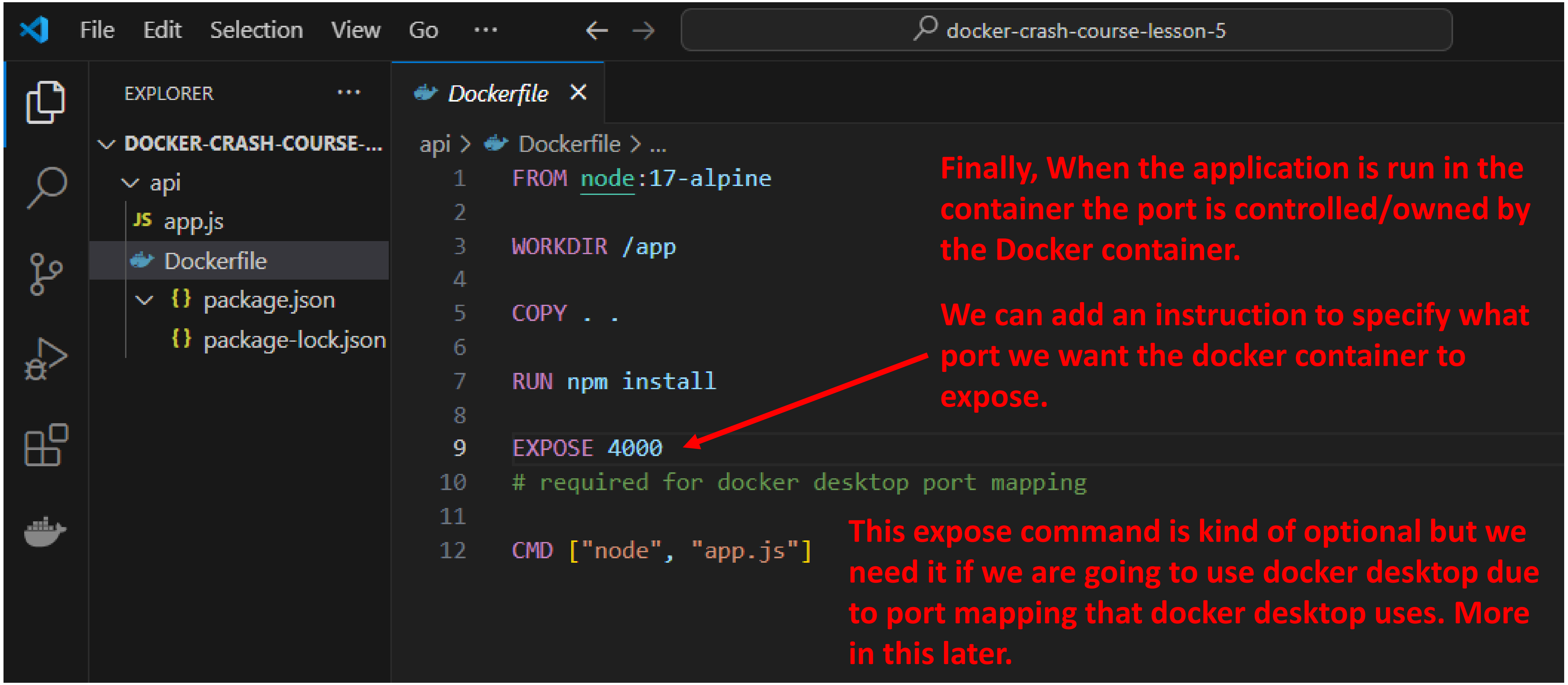
When we add a RUN instruction it adds a command to run at build time of the image but we want the App to run when we run the image in the container.

So it makes sense to put the dependencies in the image (package.json) but not the application.

we can ad a CMD to specify any commands we want to run when we run the container. The CMD takes an array of strings where we specify the container then the application name.

This will spin up the app.js when we run the container, not when we create the image.

Create a Dockerfile (4)



```
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 4000
# required for docker desktop port mapping

CMD ["node", "app.js"]
```

Finally, When the application is run in the container the port is controlled/owned by the Docker container.

We can add an instruction to specify what port we want the docker container to expose.

This expose command is kind of optional but we need it if we are going to use docker desktop due to port mapping that docker desktop uses. More in this later.

Create an Image with Dockerfile

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-5/api
$ ls
app.js  Dockerfile  package.json  package-lock.json

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-5/api
$ docker build -t myapp .
[+] Building 16.4s (10/10) FINISHED                                                                docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                               0.1s
 => => transferring dockerfile: 180B                                                                               0.0s
 => [internal] load metadata for docker.io/library/node:17-alpine                                                  1.8s
 => [auth] library/node:pull token for registry-1.docker.io                                                        0.0s
 => [internal] load .dockerignore                                                                                  0.1s
 => => transferring context: 2B                                                                                    0.0s
 => [1/4] FROM docker.io/library/node:17-alpine@sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595d44092b625e2c    5.6s
 => => resolve docker.io/library/node:17-alpine@sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595d44092b625e2c    0.1s
 => => sha256:1bedfac31d6a1e001d4e5d45ea1aba8f53e5f54b5555ce2c415a65a7041b074f 45.89MB / 45.89MB                   1.9s
 => => sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595d44092b625e2cd557ddbf 1.43kB / 1.43kB                     0.0s
 => => sha256:c7bde48048debf58dba50f8d2ba674854bdf7dfc8c43bd468f19a5212facfdbe 1.16kB / 1.16kB                     0.0s
 => => sha256:57488723f0872b65eb586f4fde54d5c25c16cde94da3bde8b338cf2af2aceb1c 6.67kB / 6.67kB                     0.0s
 => => sha256:6463b5f3dbb1d524374fd51f430ea4837e794edd1c508bad449f93a86be57ccb 2.34MB / 2.34MB                     1.4s
 => => sha256:df9b9388f04ad6279a7410b85cedfdcb2208c0a003da7ab5613af71079148139 2.81MB / 2.81MB                     1.9s
 => => sha256:885e68a88c76f90ebf7b390469107ac661410a590df8939c237fa720ca91efb3 451B / 451B                         1.6s
 => => extracting sha256:df9b9388f04ad6279a7410b85cedfdcb2208c0a003da7ab5613af71079148139                          0.1s
 => => extracting sha256:1bedfac31d6a1e001d4e5d45ea1aba8f53e5f54b5555ce2c415a65a7041b074f                          2.5s
 => => extracting sha256:6463b5f3dbb1d524374fd51f430ea4837e794edd1c508bad449f93a86be57ccb                          0.1s
 => => extracting sha256:885e68a88c76f90ebf7b390469107ac661410a590df8939c237fa720ca91efb3                          0.0s
 => [internal] load build context                                                                                  0.1s
 => => transferring context: 34.25kB                                                                               0.0s
 => [2/4] WORKDIR /app                                                                                             4.9s
 => [3/4] COPY . .                                                                                                 0.2s
 => [4/4] RUN npm install                                                                                          3.2s
 => exporting to image                                                                                             0.3s
 => => exporting layers                                                                                            0.2s
 => => writing image sha256:25ae2bdc48f6e54e0441b7ed7fa37e2ea4b3cdc4446059fdc05c99f2f9b879db                       0.0s
 => => naming to docker.io/library/myapp                                                                           0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/wxcrb93b8lhv27ug6lw7wvb2a

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-5/api
```

From the VS code terminal (view -> terminal) I can ensure I am in the directory where the Dockerfile is located and run the command **`docker build –t myapp .`** Where:

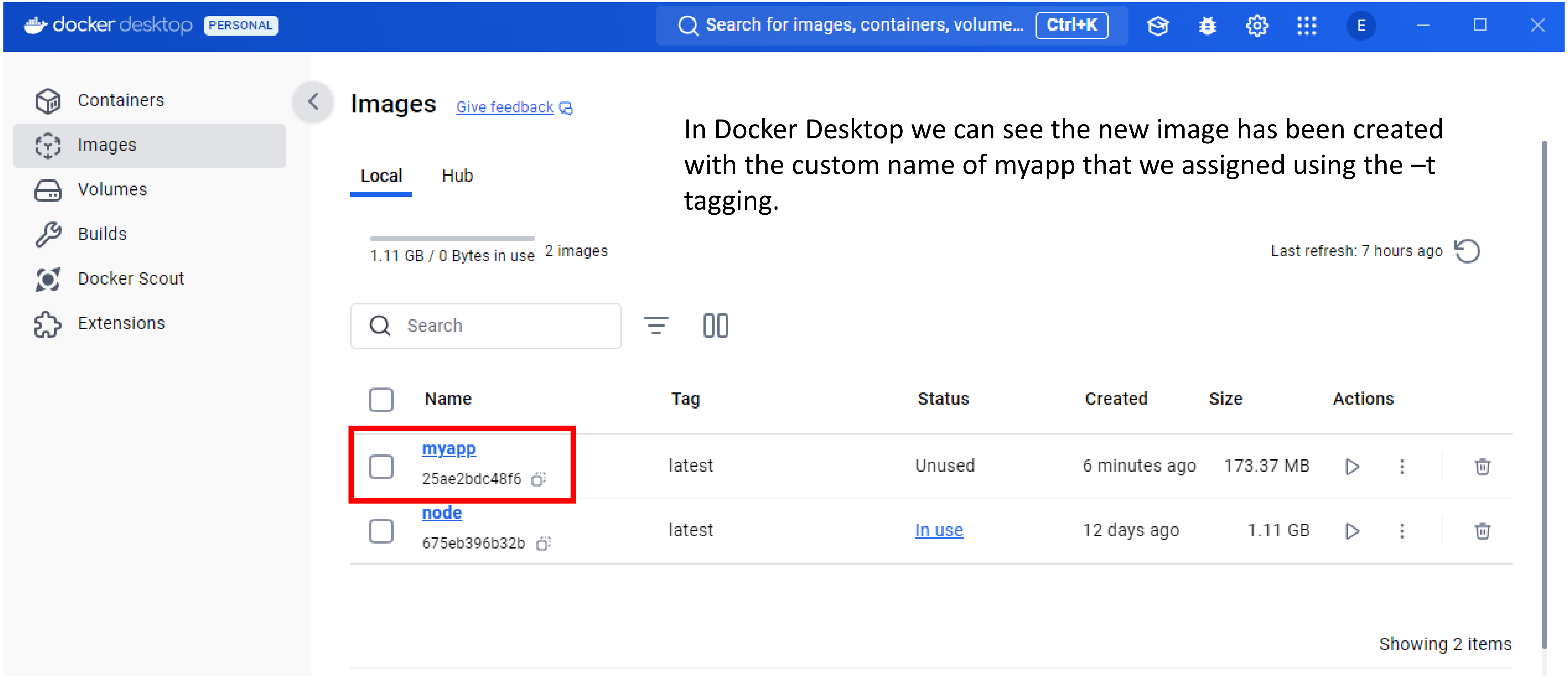
-t is adding a tag of a custom name

. note that the dot at the end of the command is a relative path to the Dockerfile.

Note that in the terminal you see the build process of the image where each line is essentially a new layer being added.

Verify Image in docker desktop

Note that the build image process does not add any additional files to the VS code folder because it is created within Docker



1.11 GB / 0 Bytes in use 2 images

Last refresh: 7 hours ago

Name	Tag	Status	Created	Size	Actions
myapp 25ae2bdc48f6	latest	Unused	6 minutes ago	173.37 MB	
node 675eb396b32b	latest	In use	12 days ago	1.11 GB	

Showing 2 items

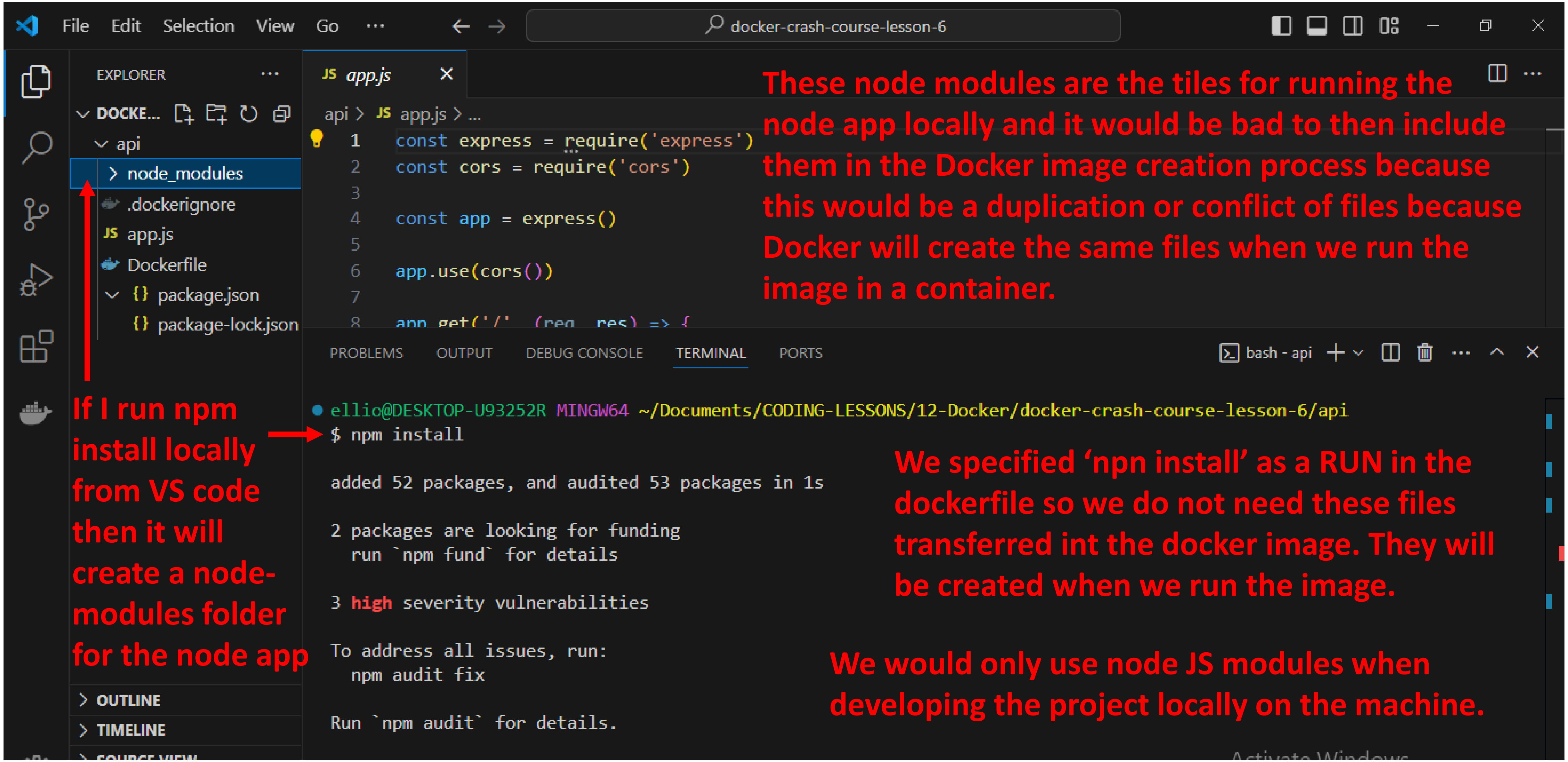
In Docker Desktop we can see the new image has been created with the custom name of myapp that we assigned using the –t tagging.

#5 Dockerignore

[Why use Dockerignore?]

[Dockerignore file]

Why use Dockerignore?



```
const express = require('express')
const cors = require('cors')

const app = express()

app.use(cors())

app.get('/', (req, res) => {
```

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ npm install

added 52 packages, and audited 53 packages in 1s

2 packages are looking for funding
  run `npm fund` for details

3 high severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

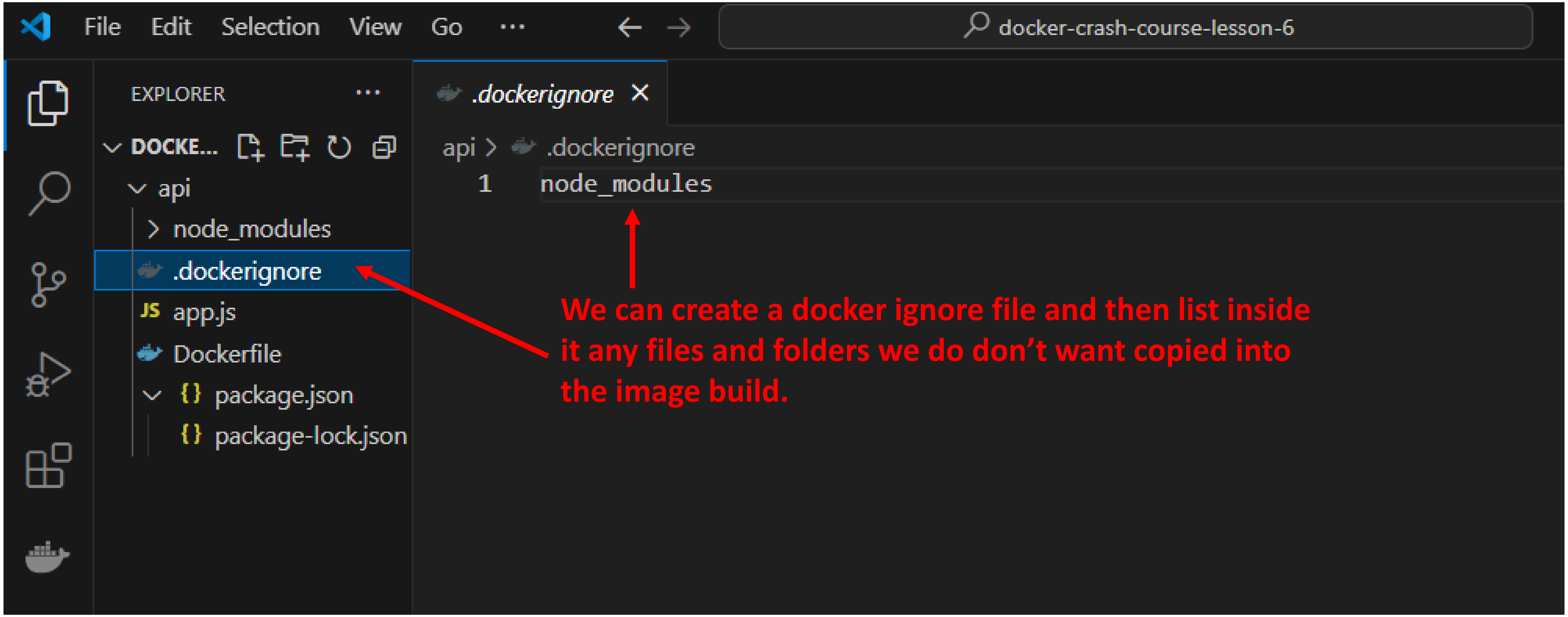
These node modules are the tiles for running the node app locally and it would be bad to then include them in the Docker image creation process because this would be a duplication or conflict of files because Docker will create the same files when we run the image in a container.

If I run npm install locally from VS code then it will create a node-modules folder for the node app

We specified 'npn install' as a RUN in the dockerfile so we do not need these files transferred int the docker image. They will be created when we run the image.

We would only use node JS modules when developing the project locally on the machine.

Dockerignore file



```
node_modules
```

We can create a docker ignore file and then list inside it any files and folders we do don't want copied into the image build.

#6 Starting & Stopping containers

Optional parameters when starting a container

open running container in the browser

Stopping a container in Docker Desktop

running container in the browser

Start Container From Terminal

Show active Docker processes

Stop Container Process from Terminal

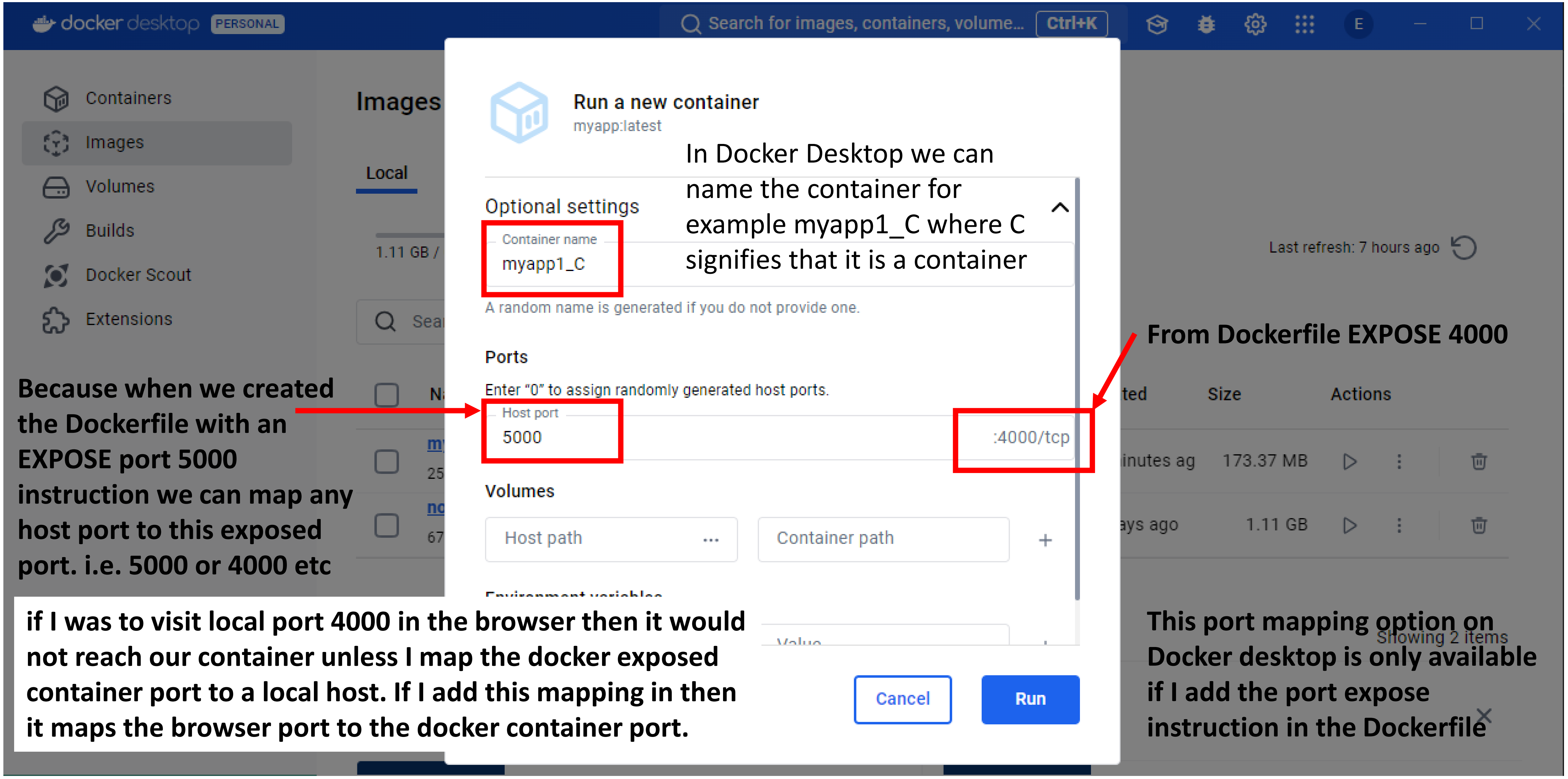
Start Container with Port Mapping

Docker ps command

Docker ps –a command

Restart an existing container

Optional parameters when starting a container



Run a new container
myapp:latest

Optional settings

Container name: myapp1_C

A random name is generated if you do not provide one.

Ports

Enter "0" to assign randomly generated host ports.

Host port: 5000 :4000/tcp

Volumes

Host path ... Container path +

Environment variables

In Docker Desktop we can name the container for example myapp1_C where C signifies that it is a container

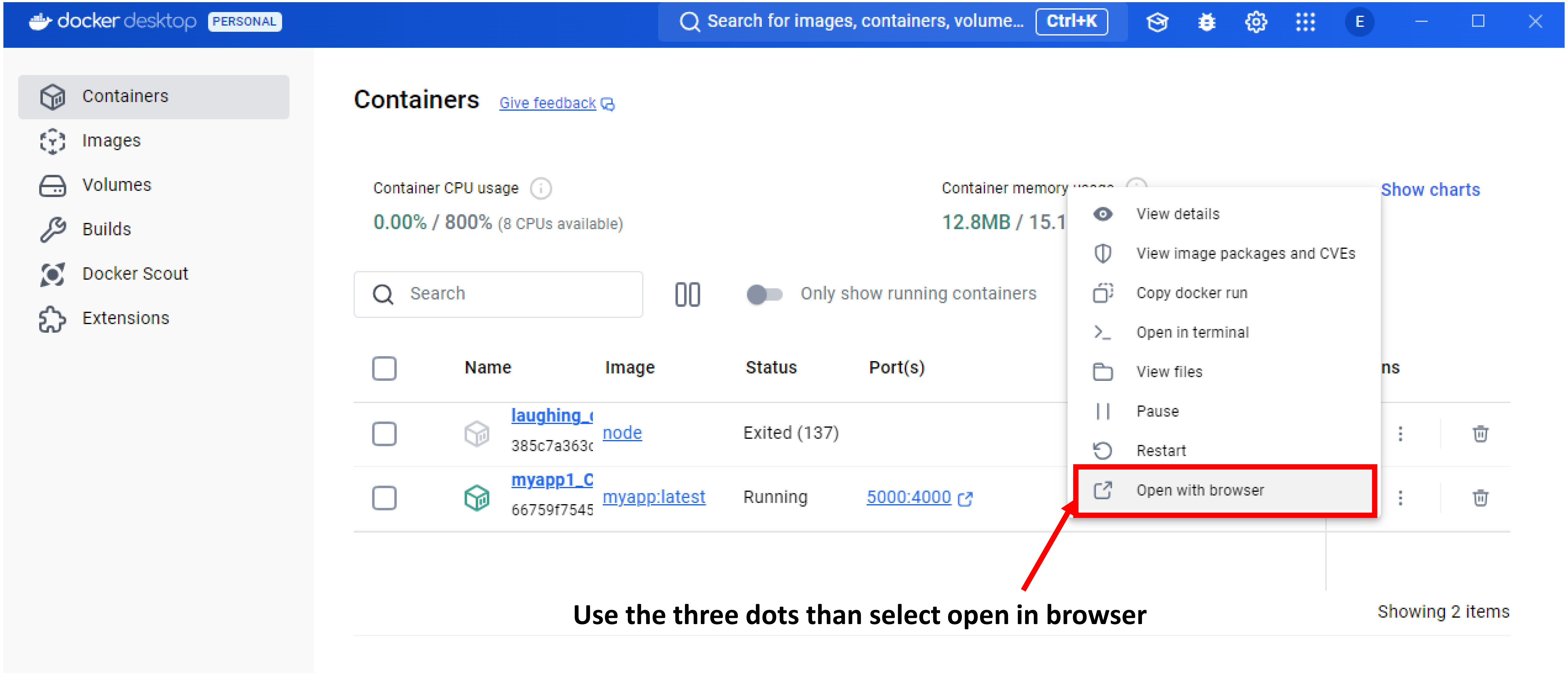
From Dockerfile EXPOSE 4000

Because when we created the Dockerfile with an EXPOSE port 5000 instruction we can map any host port to this exposed port. i.e. 5000 or 4000 etc

if I was to visit local port 4000 in the browser then it would not reach our container unless I map the docker exposed container port to a local host. If I add this mapping in then it maps the browser port to the docker container port.

This port mapping option on Docker desktop is only available if I add the port expose instruction in the Dockerfile

open running container in the browser



Containers

Container CPU usage
0.00% / 800% (8 CPUs available)

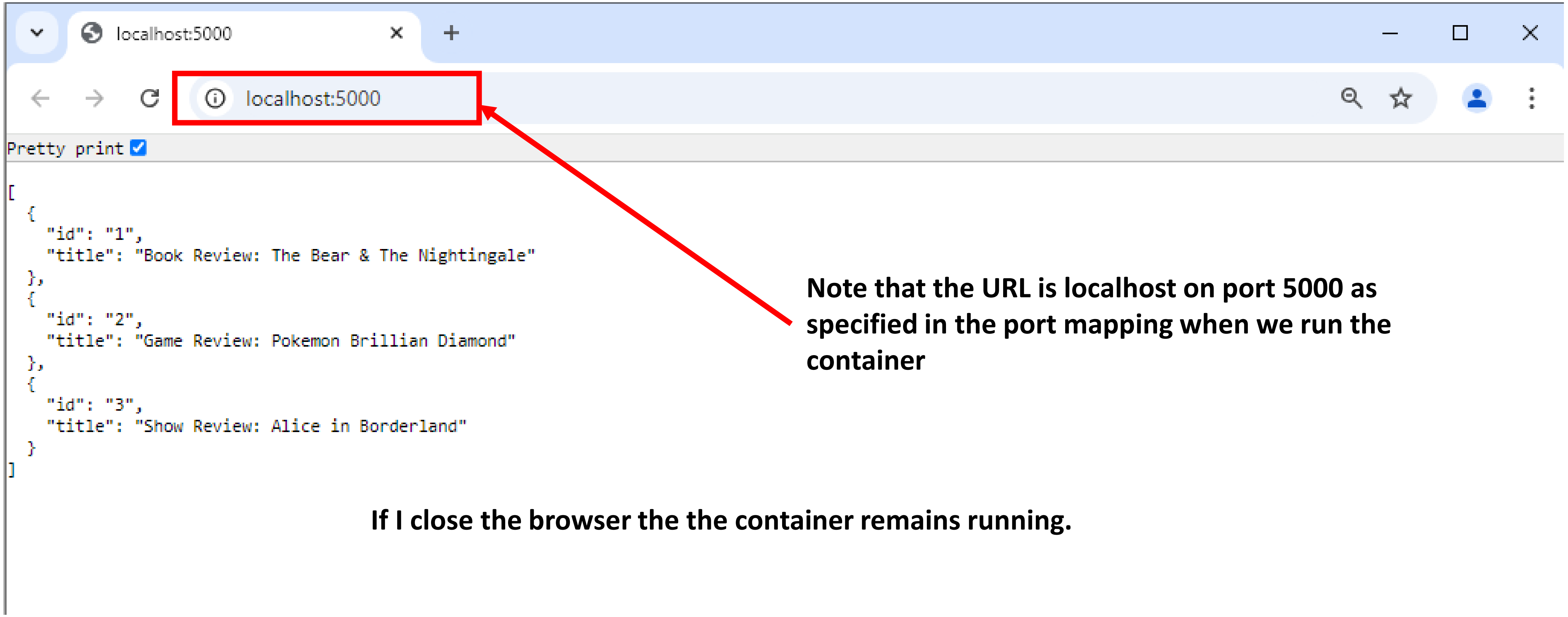
Container memory usage
12.8MB / 15.1

Name	Image	Status	Port(s)
laughing_ 385c7a363	node	Exited (137)	
myapp1_C 66759f7545	myapp:latest	Running	5000:4000

Showing 2 items

Use the three dots than select open in browser

running container in the browser

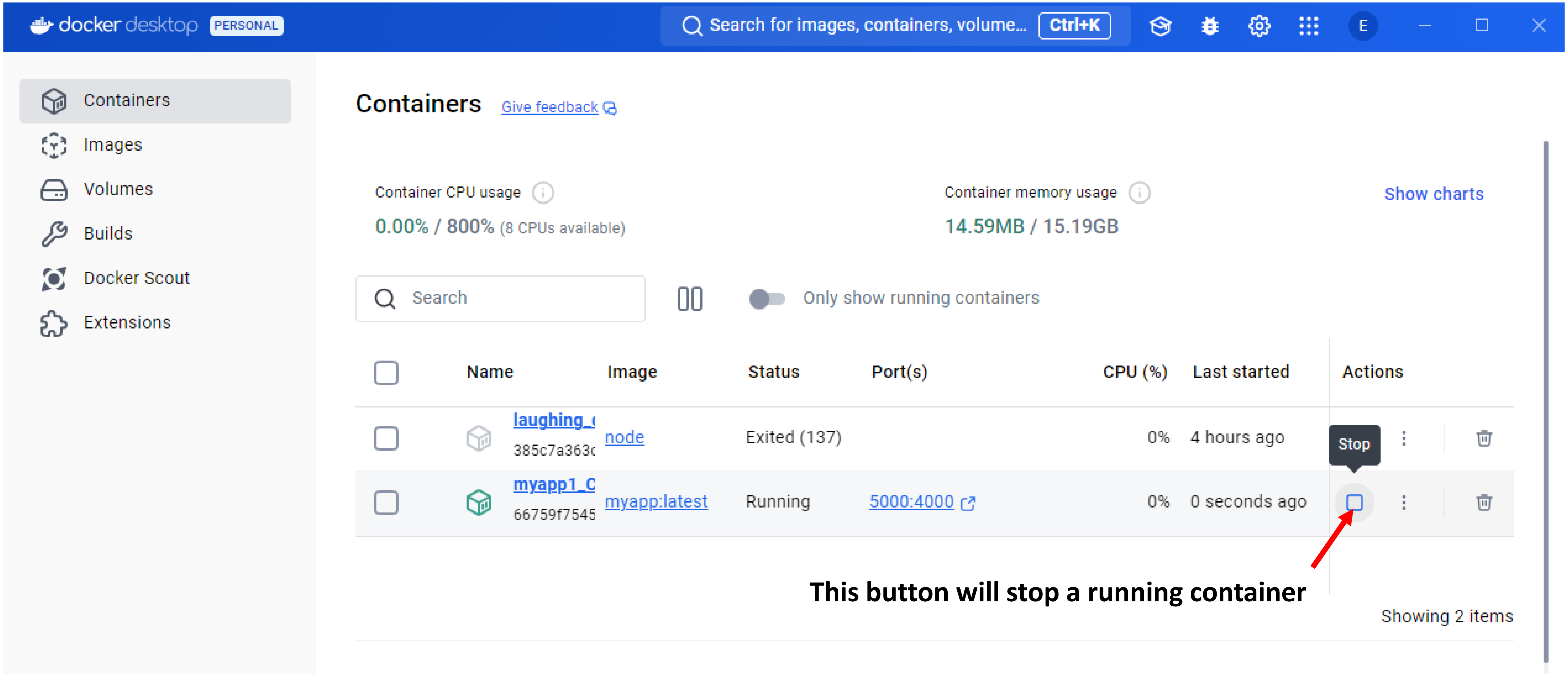


```
[
  {
    "id": "1",
    "title": "Book Review: The Bear & The Nightingale"
  },
  {
    "id": "2",
    "title": "Game Review: Pokemon Brillian Diamond"
  },
  {
    "id": "3",
    "title": "Show Review: Alice in Borderland"
  }
]
```

Note that the URL is localhost on port 5000 as specified in the port mapping when we run the container

If I close the browser the the container remains running.

Stopping a container in Docker Desktop



Containers

Container CPU usage
0.00% / 800% (8 CPUs available)

Container memory usage
14.59MB / 15.19GB

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
laughing_c 385c7a363c	node	Exited (137)		0%	4 hours ago	
myapp1_C 66759f7545	myapp:latest	Running	5000:4000	0%	0 seconds ago	

Showing 2 items

This button will stop a running container

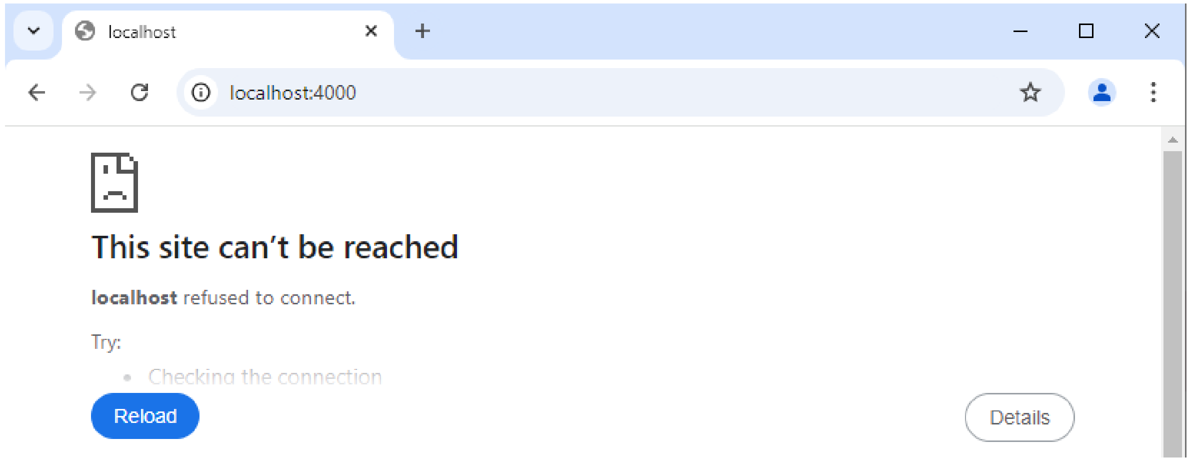
Start Container From Terminal

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED             SIZE
myapp         latest    25ae2bdc48f6   About an hour ago   173MB
node          latest    675eb396b32b   12 days ago         1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api


ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name myapp_c2 myapp
listening for requests on port 4000
```

Docker images command in the terminal will list all the images

We can run an image by creating a new container and adding optional parameters where:

--name myapp_c2 is the name of the new container



This site can't be reached

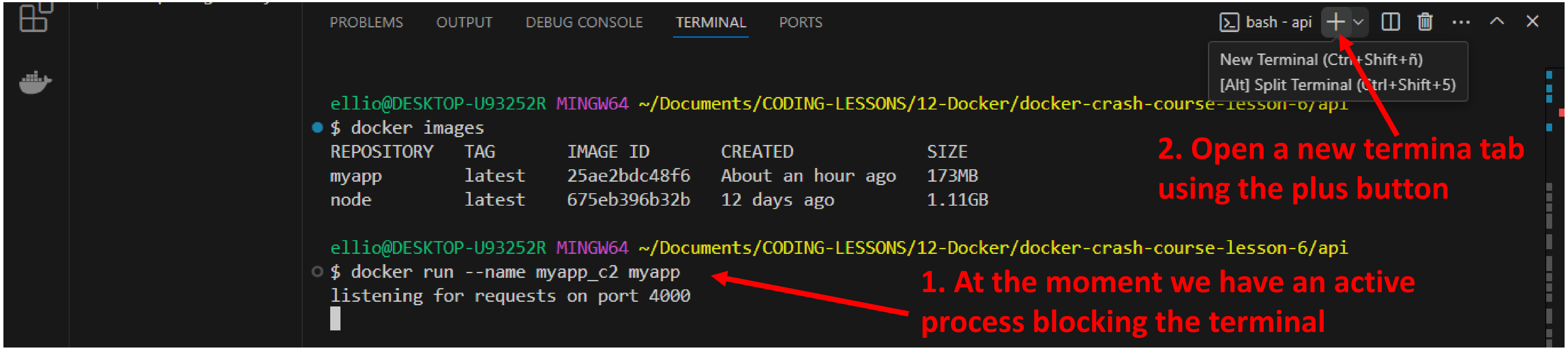
localhost refused to connect.

Try:

- Checking the connection

The Command line says it is listening on port 4000 but when I visit localhost port 4000 in the browser it does not actually work – No port mapping

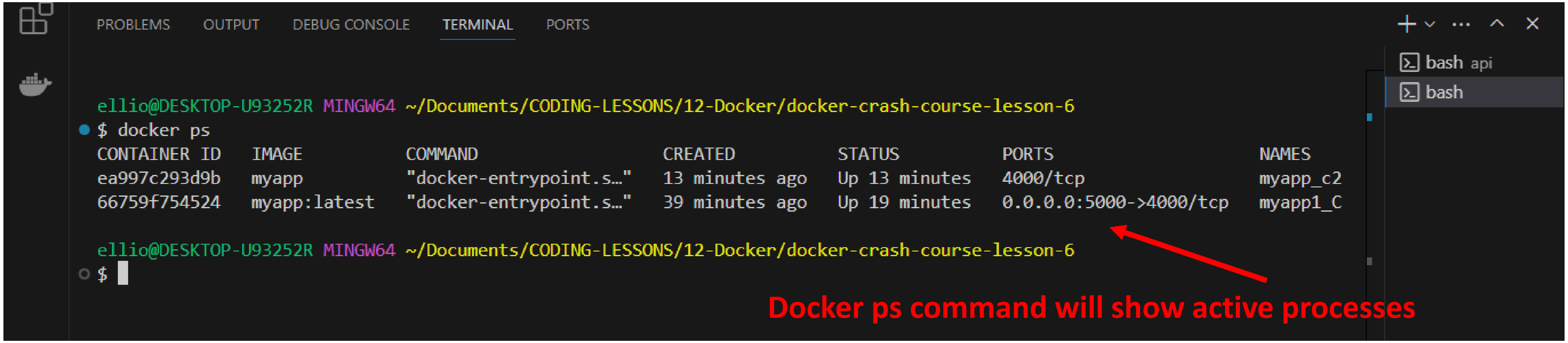
Show active Docker processes


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash - api
New Terminal (Ctrl+Shift+ñ)
[Alt] Split Terminal (Ctrl+Shift+5)

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED             SIZE
myapp        latest    25ae2bdc48f6   About an hour ago   173MB
node         latest    675eb396b32b   12 days ago         1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name myapp_c2 myapp
listening for requests on port 4000
```

2. Open a new termina tab using the plus button

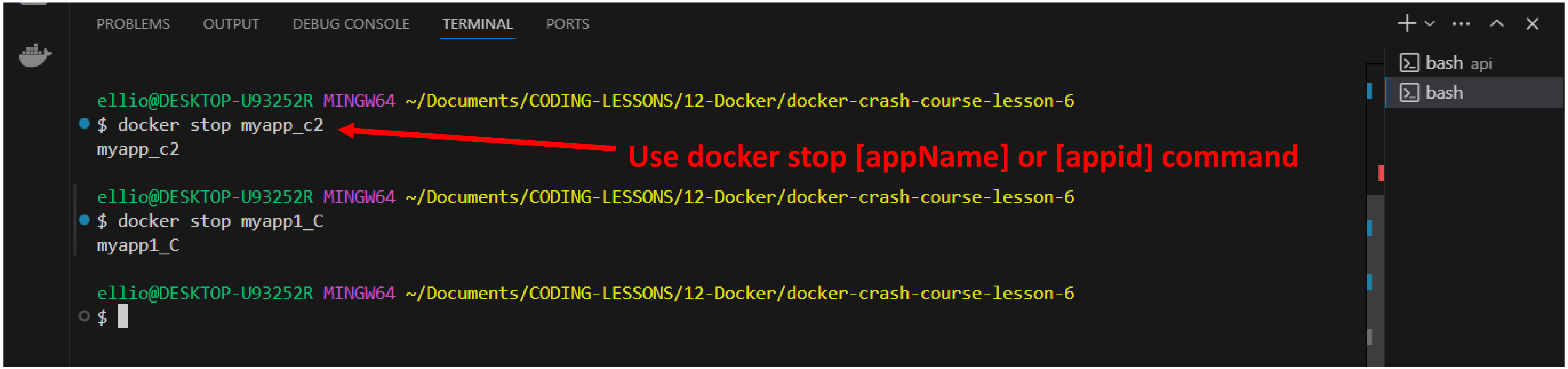
1. At the moment we have an active process blocking the terminal



```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
ea997c293d9b   myapp          "docker-entrypoint.s…"   13 minutes ago   Up 13 minutes   4000/tcp                 myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   39 minutes ago   Up 19 minutes   0.0.0.0:5000->4000/tcp   myapp1_C

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ 
```

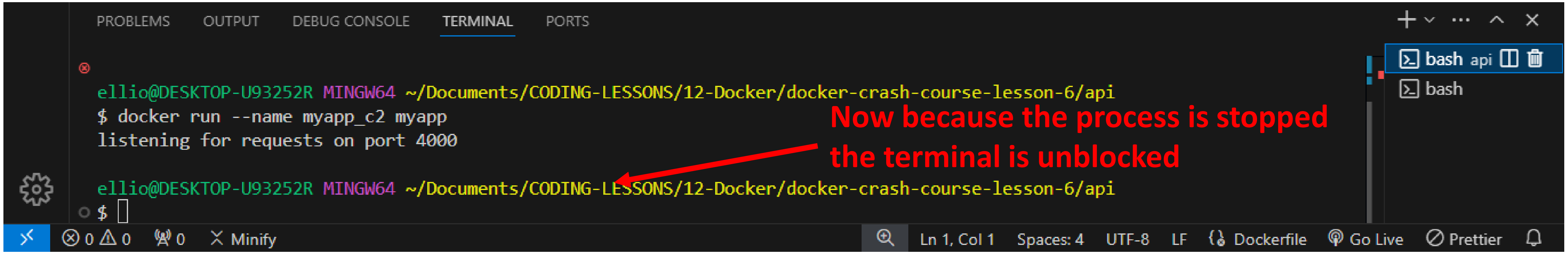
Docker ps command will show active processes

Stop Container Process from Terminal



```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker stop myapp_c2
myapp_c2

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker stop myapp1_C
myapp1_C

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$
```

Use docker stop [appName] or [appid] command



```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name myapp_c2 myapp
listening for requests on port 4000

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$
```

Now because the process is stopped the terminal is unblocked

Start Container with Port Mapping

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name myapp_c3 -p 4000:4000 -d myapp
acb60a67e55a9bbb8bb5694ac05cb5a177d2cfecde5618130098987d633cd479

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
```

```
docker run --name myapp_c3 -p 4000:4000 -d myapp
```

Name of container

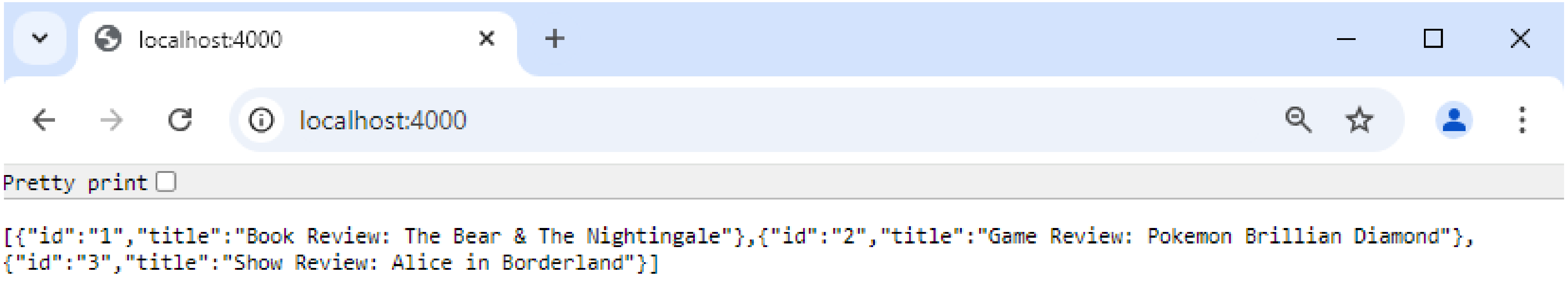
-p = port mapping

4000 = Localhost port

4000 = container port

-d means that the container is run in detached mode independently of the terminal so it does not block the terminal

myapp is the image



```
[{"id":"1","title":"Book Review: The Bear & The Nightingale"},{"id":"2","title":"Game Review: Pokemon Brillian Diamond"},
{"id":"3","title":"Show Review: Alice in Borderland"}]
```

Docker ps command

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS
NAMES
acb60a67e55a   myapp     "docker-entrypoint.s…"   9 minutes ago   Up 9 minutes   0.0.0.0:4000->4000/tcp
myapp_c3

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker stop myapp_c3
myapp_c3

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$
```

Docker ps only shows active running containers

Docker ps command

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED            STATUS                          PORTS     NAMES
acb60a67e55a   myapp          "docker-entrypoint.s…"   12 minutes ago     Exited (137) 2 minutes ago                myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   42 minutes ago     Exited (137) 24 minutes ago               myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   About an hour ago   Exited (137) 24 minutes ago
myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   5 hours ago        Exited (137) About an hour ago
laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
```

Docker ps –a will show all containers

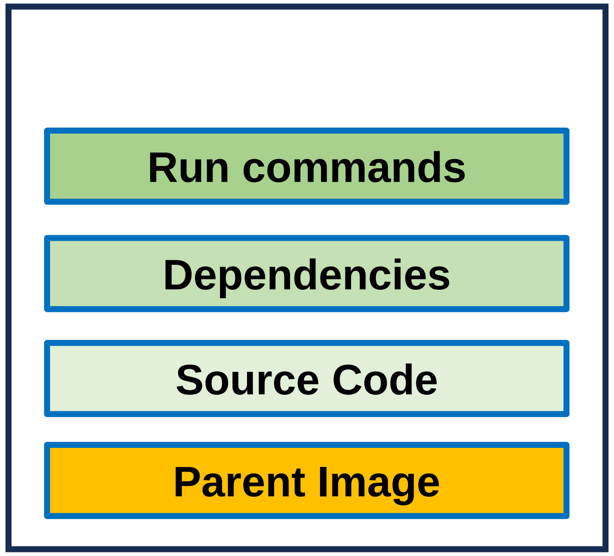
Restart an existing container

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker start myapp_c3
myapp_c3
```

Docker start command will restart an existing container. We do not need to specify ports because that was previously done when we created the container and is saved automatically

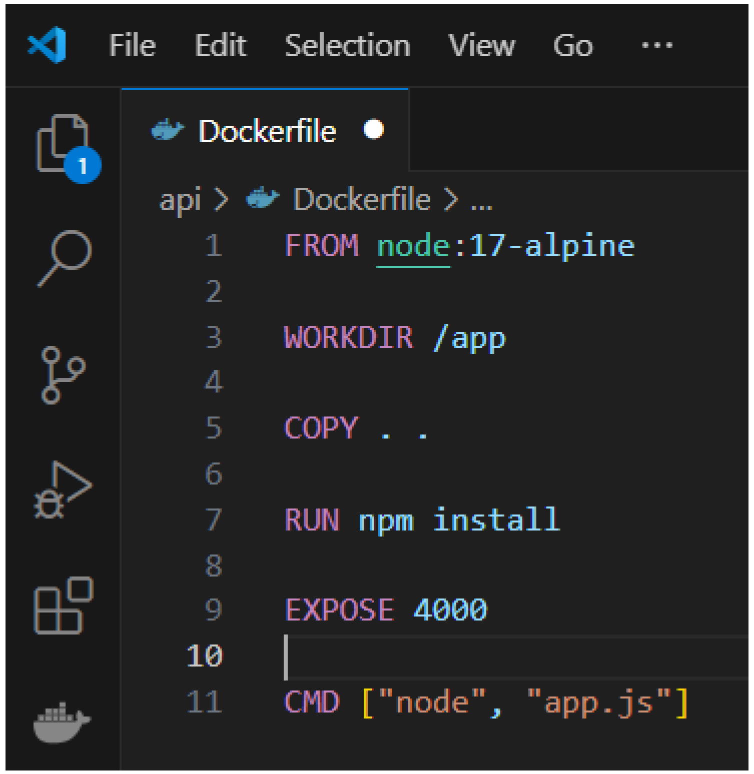
#7 Docker Layer Caching

Why do we need Docker Layer Caching?



Every line in the docker file kind of represents a new layer in the image that we are creating because each line adds something new to the image. Each line adds something new to the image.

Each time we add a new line to the Dockerfile we are essentially changing the image. each new layer creates extra work for docker to do to create the image.



```dockerfile
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 4000

CMD ["node", "app.js"]
```

Looking at the build process output in the terminal we can see Docker working through each line of the Dockerfile to build the image and see how long it took to complete.

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker build -t myapp2 .
[+] Building 11.3s (10/10) FINISHED                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                         0.0s
 => => transferring dockerfile: 134B                                                                         0.0s
 => [internal] load metadata for docker.io/library/node:16-alpine                                            1.7s
 => [auth] library/node:pull token for registry-1.docker.io                                                  0.0s
 => [internal] load .dockerignore                                                                            0.0s
 => => transferring context: 52B                                                                             0.0s
 => [1/4] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787      3.6s
 => => resolve docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787      0.0s
 => => sha256:93b3025fe10392717d06ec0d012a9ffa2039d766a322aac899c6831dd93382c2 2.34MB / 2.34MB              0.3s
 => => sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787 1.43kB / 1.43kB              0.0s
 => => sha256:72e89a86be58c922ed7b1475e5e6f151537676470695dd106521738b060e139d 1.16kB / 1.16kB              0.0s
 => => sha256:2573171e0124bb95d14d128728a52a97bb917ef45d7c4fa8cfe76bc44aa78b73 6.73kB / 6.73kB              0.0s
 => => sha256:7264a8db6415046d36d16ba98b79778e18accee6ffa71850405994cffa9be7de 3.40MB / 3.40MB              1.4s
 => => sha256:eee371b9ce3ffdbb8aa703b9a14d318801ddc3468f096bb6cfeabbeb715147f9 36.63MB / 36.63MB            1.7s
 => => sha256:d9059661ce70092af66d2773666584fc8addcb78a2be63f720022f4875577ea9 452B / 452B                  1.3s
 => => extracting sha256:7264a8db6415046d36d16ba98b79778e18accee6ffa71850405994cffa9be7de                   0.2s
 => => extracting sha256:eee371b9ce3ffdbb8aa703b9a14d318801ddc3468f096bb6cfeabbeb715147f9                   1.4s
 => => extracting sha256:93b3025fe10392717d06ec0d012a9ffa2039d766a322aac899c6831dd93382c2                   0.1s
 => => extracting sha256:d9059661ce70092af66d2773666584fc8addcb78a2be63f720022f4875577ea9                   0.0s
 => [internal] load build context                                                                            0.0s
 => => transferring context: 34.25kB                                                                        0.0s
 => [2/4] WORKDIR /app                                                                                       3.0s
 => [3/4] COPY . .                                                                                           0.1s
 => [4/4] RUN npm install                                                                                    2.6s
 => exporting to image                                                                                       0.2s
 => => exporting layers                                                                                      0.1s
 => => writing image sha256:5655d386938f67013f9d94992b10a96baddafcdd94f0eed663b9d9f96b2262a4                0.0s
 => => naming to docker.io/library/myapp2
```

[1/4] FROM node:16-alpine. This is where docker is downloading the parent image from the docker hub repository.

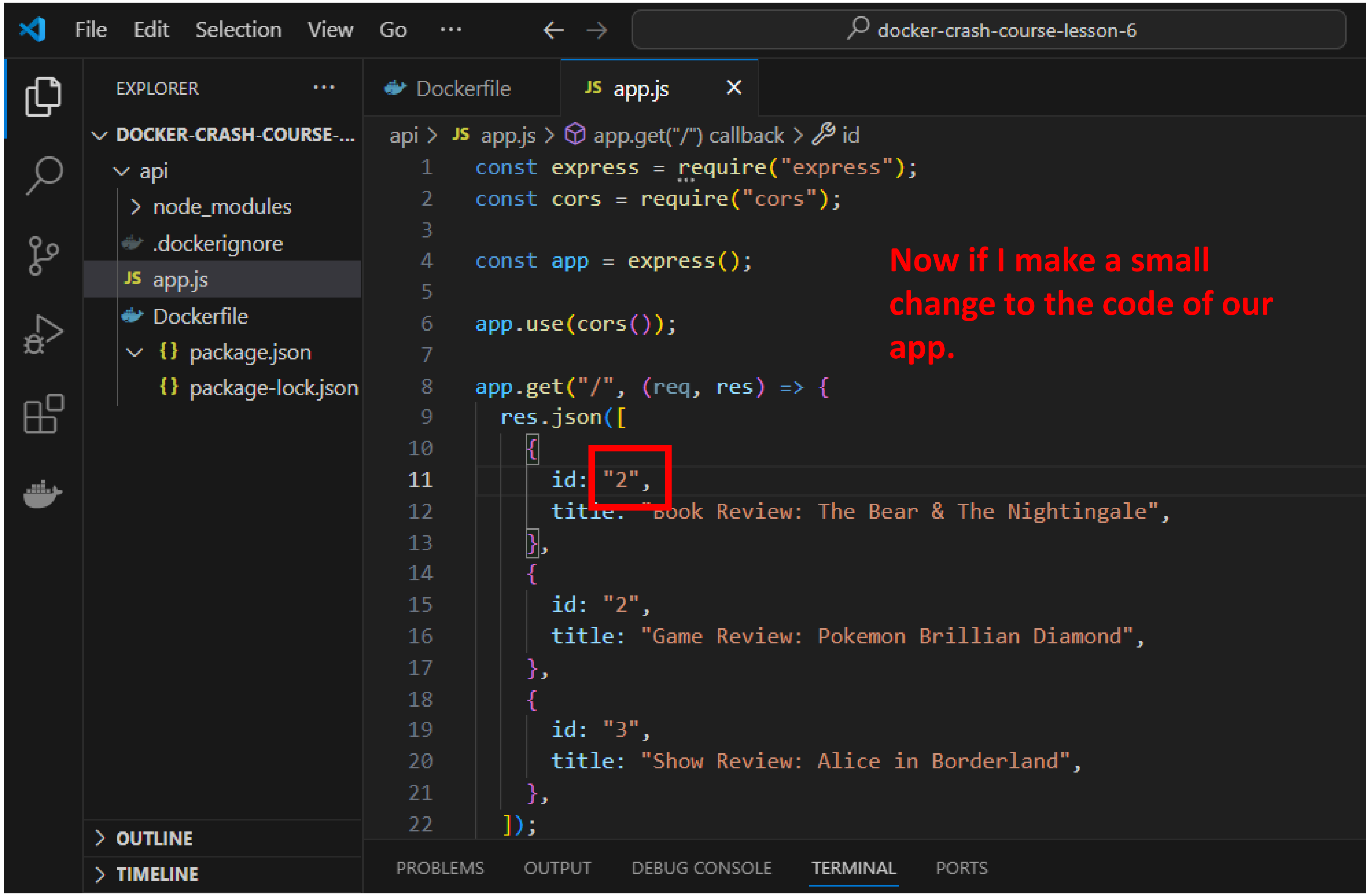
[2/4] FROM WORKDIR /app. It is creating the working directory and deciding what to add to the image based on the dockerIngnore.

[3/4] Copy . . It is copying the image

[3/4] Run npm install. It is installing the dependencies

At the top it gives us a total time for completing the build. In this case 11.3 seconds. Note that this is only for a simple test app with a few lines of code. For a real world app, Each time we make a change to our code we need to build a new image because images are read only. This could become very time consuming each time we change the app and have to create a new docker image.

Docker Layer Caching


File Edit Selection View Go ···
docker-crash-course-lesson-6
EXPLORER
DOCKER-CRASH-COURSE-...
api
node_modules
.dockerignore
app.js
Dockerfile
package.json
package-lock.json
OUTLINE
TIMELINE
Dockerfile
app.js
api > app.js > app.get("/") callback > id
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```javascript
const express = require("express");
const cors = require("cors");

const app = express();

app.use(cors());

app.get("/", (req, res) => {
  res.json([
    {
      id: "2",
      title: "Book Review: The Bear & The Nightingale",
    },
    {
      id: "2",
      title: "Game Review: Pokemon Brillian Diamond",
    },
    {
      id: "3",
      title: "Show Review: Alice in Borderland",
    },
  ]);
```

Now if I make a small change to the code of our app.

Docker Layer Caching seen in Build command output

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/mip4nl718mb6evrzyzfehpy1n

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker build -t myapp3 .
[+] Building 4.7s (10/10) FINISHED                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                  0.0s
 => => transferring dockerfile: 134B                                                                  0.0s
 => [internal] load metadata for docker.io/library/node:16-alpine                                     1.0s
 => [auth] library/node:pull token for registry-1.docker.io                                           0.0s
 => [internal] load .dockerignore                                                                     0.0s
 => => transferring context: 52B                                                                      0.0s
 => [1/4] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9d  0.0s
 => [internal] load build context                                                                     0.0s
 => => transferring context: 658B                                                                     0.0s
 => CACHED [2/4] WORKDIR /app                                                                         0.0s
 => [3/4] COPY . .                                                                                    0.1s
 => [4/4] RUN npm install                                                                             3.1s
 => exporting to image                                                                                0.2s
 => => exporting layers                                                                               0.2s
 => => writing image sha256:e7cf174cfe429e5102ef8a31e49e94046718eac82ef5a0964d31796bde339145          0.0s
 => => naming to docker.io/library/myapp3                                                             0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/538k0vkqgz5etbwd2negpkr9d

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
```

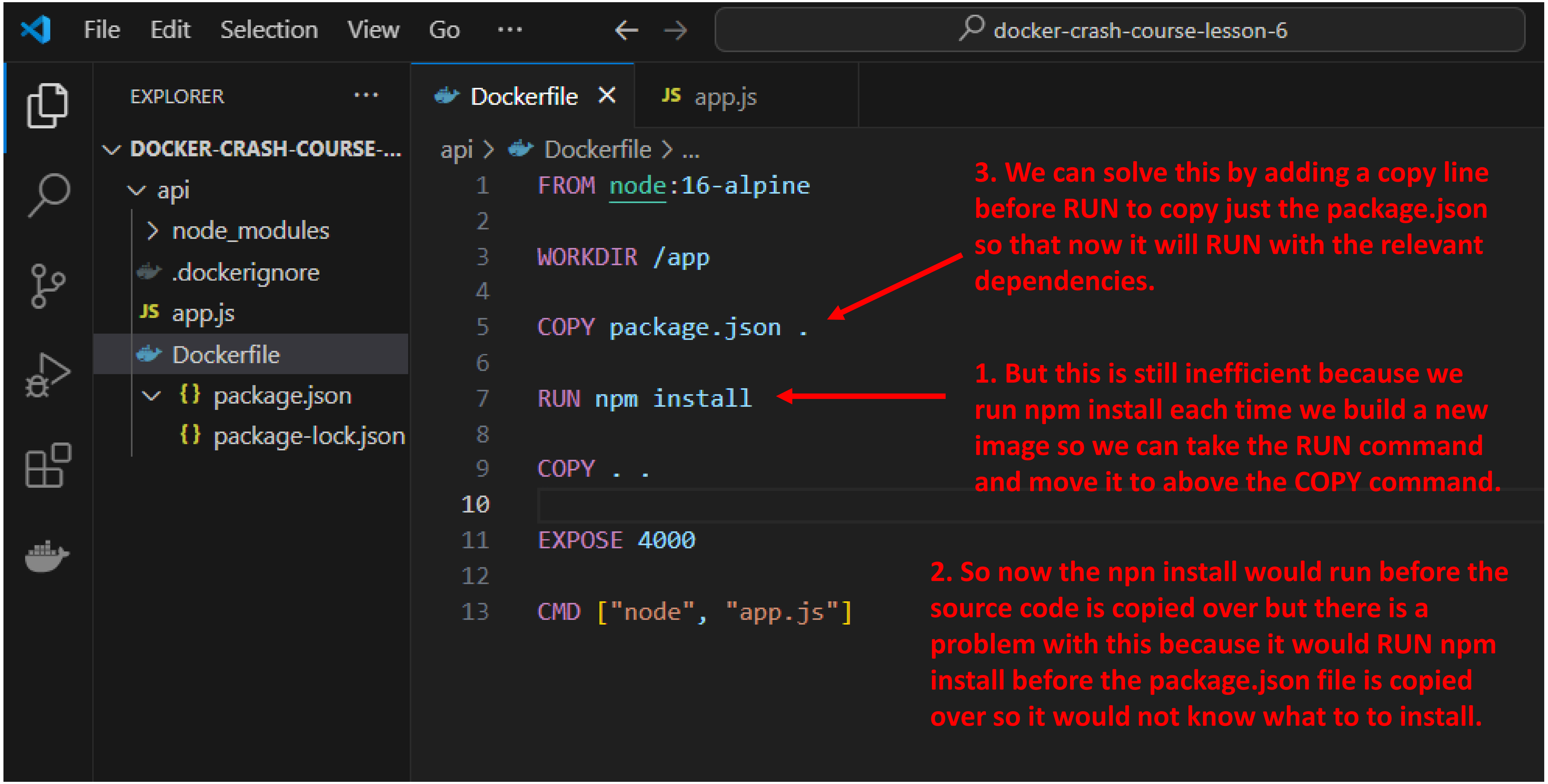
Note that when the new image is now built it took a lot less time to do it - `Building 4.7s`. This is because docker caches each layer. The first time we build an image it stores the image in the cache at each layer. When we build a new image it looks in the cache to see if there is an existing cache that can be used - `CACHED [2/4] WORKDIR /app`.

In our case we made a change to the code files which affect the copy layer so Docker uses the first two layers of the cache adding additional layers on top where stuff has changed.

Pulling from the cache is quicker than downloading a new image from the repository if the parent image has not changed.

But why does it not use other layers of the cache that have not changed. Because the changes affect all higher layers so it will take from the cache layer.

Exploiting Docker Layer Caching



```dockerfile
FROM node:16-alpine

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 4000

CMD ["node", "app.js"]
```

3. We can solve this by adding a copy line before RUN to copy just the package.json so that now it will RUN with the relevant dependencies.

1. But this is still inefficient because we run npm install each time we build a new image so we can take the RUN command and move it to above the COPY command.

2. So now the npn install would run before the source code is copied over but there is a problem with this because it would RUN npm install before the package.json file is copied over so it would not know what to to install.

Docker multi-layer Caching seen in Build command output

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker build -t myapp4 .
[+] Building 6.2s (11/11) FINISHED                                                                 docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 155B                                                                              0.0s
 => [internal] load metadata for docker.io/library/node:16-alpine                                                 1.0s
 => [auth] library/node:pull token for registry-1.docker.io                                                       0.0s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 52B                                                                                  0.0s
 => [1/5] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9d  0.0s
 => [internal] load build context                                                                                 0.0s
 => => transferring context: 287B                                                                                 0.0s
 => CACHED [2/5] WORKDIR /app                                                                                     0.0s
 => [3/5] COPY package.json .                                                                                     0.1s
 => [4/5] RUN npm install                                                                                         4.7s
 => [5/5] COPY . .                                                                                                0.1s
 => exporting to image                                                                                            0.2s
 => => exporting layers                                                                                           0.2s
 => => writing image sha256:6649d4def80e413bce9bf41190966062af3e1bad706625be20619135be881867                      0.0s
 => => naming to docker.io/library/myapp4                                                                         0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/omtr0bjqys2g5zuv8p61ubtxn

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
```

We can see that when we build the image we start from a cached version (`CACHED [2/5] WORKDIR /app`) because the first two layers have not changed from the previous build.

We also see it has added extra layers for the copy json packages (`[3/5] COPY package.json`) and run npm install (`RUN npm install`) layers before copying the image.

Exploiting Layer Caching to reduce build time

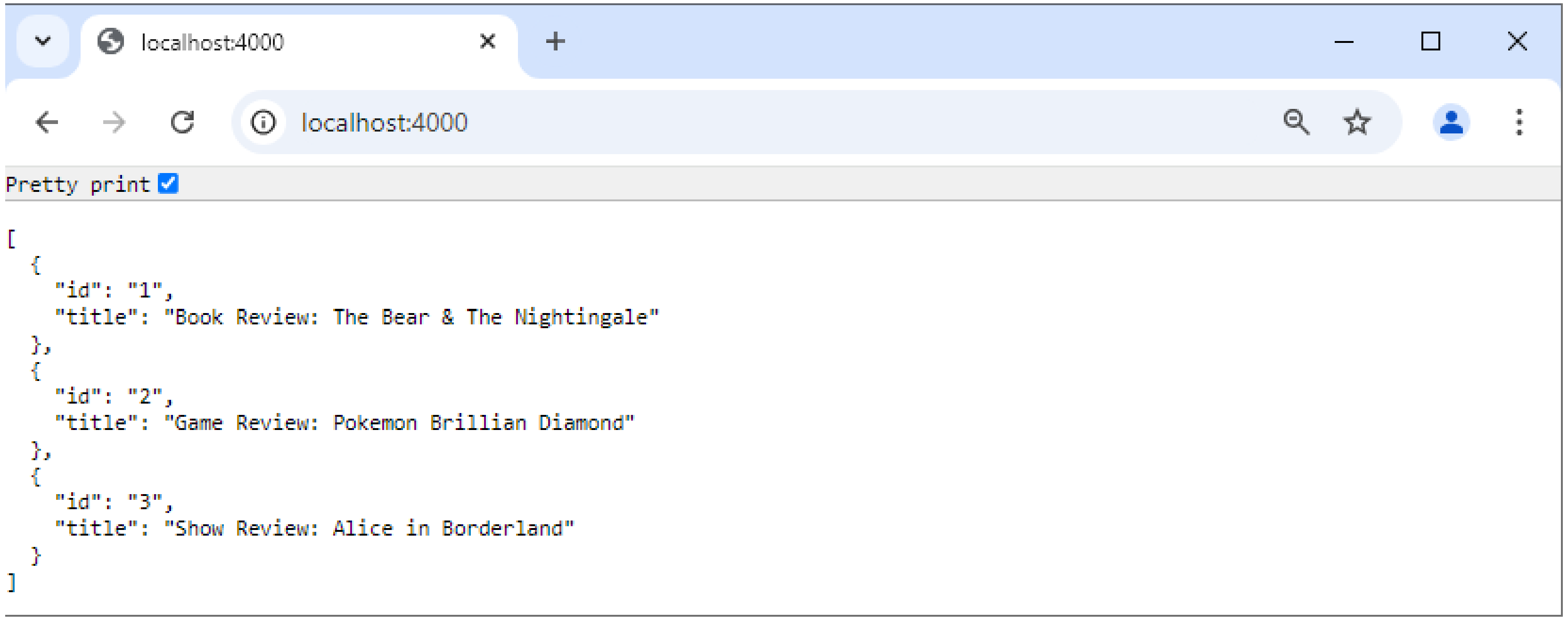
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker build -t myapp5 .
[+] Building 1.2s (11/11) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                            0.0s
 => => transferring dockerfile: 155B                                                                            0.0s
 => [internal] load metadata for docker.io/library/node:16-alpine                                               0.9s
 => [auth] library/node:pull token for registry-1.docker.io                                                     0.0s
 => [internal] load .dockerignore                                                                               0.0s
 => => transferring context: 52B                                                                                0.0s
 => [1/5] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9d 0.0s
 => [internal] load build context                                                                               0.0s
 => => transferring context: 658B                                                                               0.0s
 => CACHED [2/5] WORKDIR /app                                                                                   0.0s
 => CACHED [3/5] COPY package.json .                                                                            0.0s
 => CACHED [4/5] RUN npm install                                                                                0.0s
 => [5/5] COPY . .                                                                                              0.1s
 => exporting to image                                                                                          0.1s
 => => exporting layers                                                                                         0.0s
 => => writing image sha256:9aee05e0648f94993cd0743a2c14de4260da74cff43789239b4ef97b939eaad4                    0.0s
 => => naming to docker.io/library/myapp5                                                                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/qa4xqra15sl944tij07k5rha9

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
```

Now if I change something again in the app.js file and build the package again I see that the build time is significantly reduced (`Building 1.2s`).

We can see that when we build the it has used multiple layers of cache (`CACHED [2/5] WORKDIR /app`, `CACHED [3/5] COPY package.json .`, `CACHED [4/5] RUN npm install`) to speed up the build.

Verify Layer Cached built image runs in Container

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name myapp5_c -p 4000:4000 myapp5
listening for requests on port 4000
```



```
[
  {
    "id": "1",
    "title": "Book Review: The Bear & The Nightingale"
  },
  {
    "id": "2",
    "title": "Game Review: Pokemon Brillian Diamond"
  },
  {
    "id": "3",
    "title": "Show Review: Alice in Borderland"
  }
]
```

Now I have run the image built with layer caching, to create a new container and I can see that it successfully loads in the browser.

#8 Managing Images & Containers

View images and ALL Containers

View all images

docker images

View all containers

docker ps -a (Note that docker ps will only show running containers)

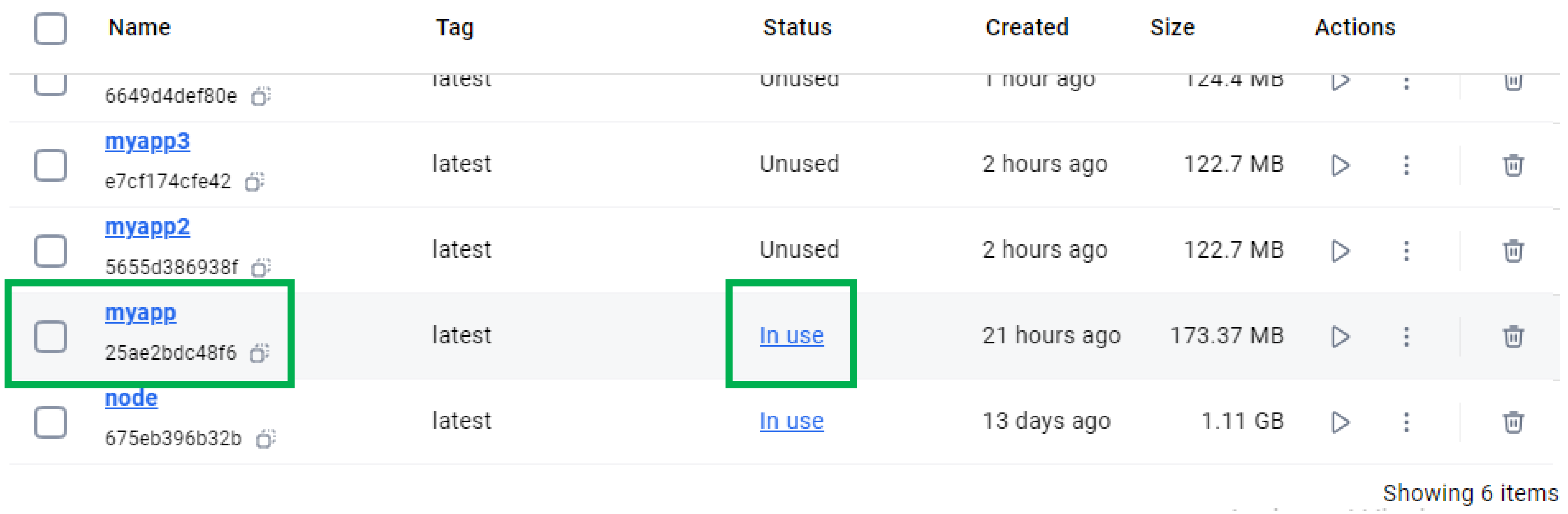
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED             SIZE
myapp5       latest    9aee05e0648f   51 minutes ago      124MB
myapp4       latest    6649d4def80e   58 minutes ago      124MB
myapp3       latest    e7cf174cfe42   About an hour ago   123MB
myapp2       latest    5655d386938f   2 hours ago         123MB
myapp        latest    25ae2bdc48f6   21 hours ago        173MB
node         latest    675eb396b32b   12 days ago         1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                       PORTS                    NAMES
d357c925c502   myapp5         "docker-entrypoint.s…"   45 minutes ago   Up 45 minutes                0.0.0.0:4000->4000/tcp   myapp5_c
acb60a67e55a   myapp          "docker-entrypoint.s…"   19 hours ago     Exited (137) 18 hours ago                             myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   19 hours ago     Exited (137) 19 hours ago                             myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   20 hours ago     Exited (137) 19 hours ago                             myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   23 hours ago     Exited (137) 20 hours ago                             laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$
```

Deleting an Image

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED             SIZE
myapp5        latest    9aee05e0648f   57 minutes ago      124MB
myapp4        latest    6649d4def80e   About an hour ago   124MB
myapp3        latest    e7cf174cfe42   About an hour ago   123MB
myapp2        latest    5655d386938f   2 hours ago         123MB
myapp         latest    25ae2bdc48f6   21 hours ago        173MB
node          latest    675eb396b32b   12 days ago         1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker image rm myapp
Error response from daemon: conflict: unable to remove repository reference "myapp" (must force) - container 66759f754524 is using its
referenced image 25ae2bdc48f6

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$
```

To delete an image the linux remove command is used (rm). However we cannot delete an image that is being used by a container.

We can see this in docker desktop where the image shows as "in use"



Name	Tag	Status	Created	Size	Actions
6649d4def80e	latest	Unused	1 hour ago	124.4 MB	
myapp3 e7cf174cfe42	latest	Unused	2 hours ago	122.7 MB	
myapp2 5655d386938f	latest	Unused	2 hours ago	122.7 MB	
myapp 25ae2bdc48f6	latest	In use	21 hours ago	173.37 MB	
node 675eb396b32b	latest	In use	13 days ago	1.11 GB	

Showing 6 items

Force Deleting an Image

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker image rm myapp4
Untagged: myapp4:latest
Deleted: sha256:6649d4def80e413bce9bf41190966062af3e1bad706625be20619135be881867

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
myapp5       latest    9aee05e0648f   About an hour ago    124MB
myapp3       latest    e7cf174cfe42   2 hours ago          123MB
myapp2       latest    5655d386938f   2 hours ago          123MB
myapp        latest    25ae2bdc48f6   21 hours ago         173MB
node         latest    675eb396b32b   12 days ago          1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker image rm myapp5 -f
Untagged: myapp5:latest

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    9aee05e0648f   About an hour ago    124MB
myapp3       latest    e7cf174cfe42   2 hours ago          123MB
myapp2       latest    5655d386938f   2 hours ago          123MB
myapp        latest    25ae2bdc48f6   21 hours ago         173MB
node         latest    675eb396b32b   12 days ago          1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
```

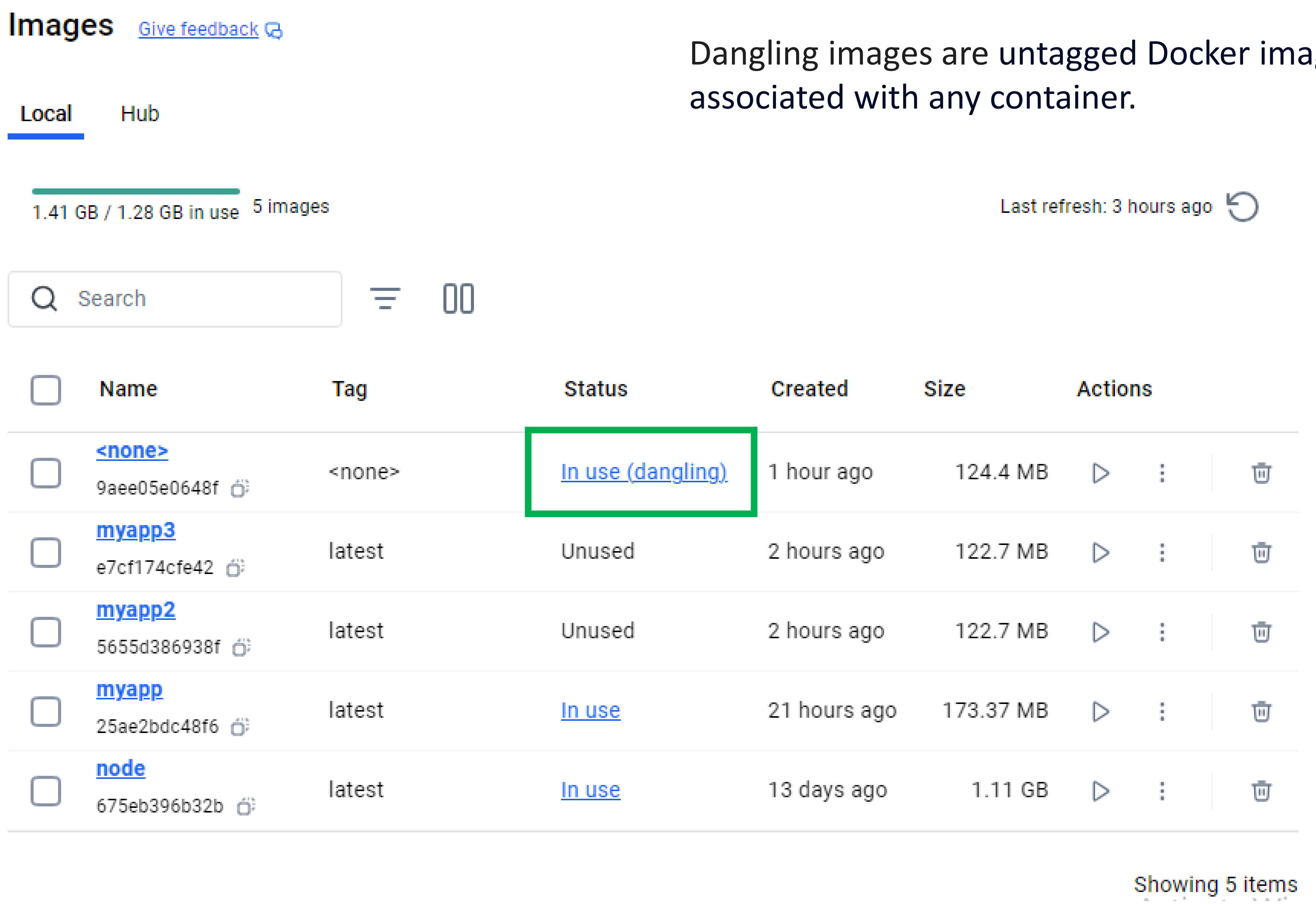
We can delete (remove) an image that is not in use.

To delete an image that is in use we have to add the –f (force) tag.

The image is (kind of) deleted (removed).

Image in use Dangling

Dangling images are untagged Docker images that are not associated with any container.



Images Give feedback

Local Hub

1.41 GB / 1.28 GB in use 5 images

Last refresh: 3 hours ago

Search

	Name	Tag	Status	Created	Size	Actions
☐	<none> 9aee05e0648f	<none>	In use (dangling)	1 hour ago	124.4 MB	
☐	myapp3 e7cf174cfe42	latest	Unused	2 hours ago	122.7 MB	
☐	myapp2 5655d386938f	latest	Unused	2 hours ago	122.7 MB	
☐	myapp 25ae2bdc48f6	latest	In use	21 hours ago	173.37 MB	
☐	node 675eb396b32b	latest	In use	13 days ago	1.11 GB	

Showing 5 items

Delete container before deleting image

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
<none>        <none>    9aee05e0648f   About an hour ago    124MB
myapp3        latest    e7cf174cfe42   2 hours ago          123MB
myapp2        latest    5655d386938f   2 hours ago          123MB
myapp         latest    25ae2bdc48f6   21 hours ago         173MB
node          latest    675eb396b32b   12 days ago          1.11GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                   CREATED             STATUS                       PORTS                    NAMES
d357c925c502   9aee05e0648f   "docker-entrypoint.s…"   About an hour ago   Up About an hour             0.0.0.0:4000->4000/tcp   myapp5_c
acb60a67e55a   myapp          "docker-entrypoint.s…"   19 hours ago        Exited (137) 19 hours ago                             myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   20 hours ago        Exited (137) 19 hours ago                             myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   20 hours ago        Exited (137) 19 hours ago                             myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   24 hours ago        Exited (137) 20 hours ago                             laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker container rm myapp5_c
Error response from daemon: cannot remove container "/myapp5_c": container is running: stop the container before removing or force remove

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker container rm myapp5_c -f
myapp5_c
```

We can see that mtapp5_c container is using the **9aee05e0648f** image

We should stop any containers that we are not using to preserve PC memory but if we try and delete a running container it throw an error but we can force delete a running container.

Delete Dangling image

The container myapp5_c has been deleted

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                       PORTS     NAMES
acb60a67e55a   myapp          "docker-entrypoint.s…"   19 hours ago    Exited (137) 19 hours ago              myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   20 hours ago    Exited (137) 20 hours ago              myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   20 hours ago    Exited (137) 19 hours ago              myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   24 hours ago    Exited (137) 20 hours ago              laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    9aee05e0648f   About an hour ago    124MB
myapp3       latest    e7cf174cfe42   2 hours ago          123MB
myapp2       latest    5655d386938f   2 hours ago          123MB
myapp        latest    25ae2bdc48f6   21 hours ago         173MB
node         latest    675eb396b32b   12 days ago          1.11GB
```

But the dangling image still remains

Note that a dangling image has no name tag so we cannot delete it by name

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker image rm myapp5
Error response from daemon: No such image: myapp5:latest

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker image rm 9aee05e0648f
Deleted: sha256:9aee05e0648f94993cd0743a2c14de4260da74cff43789239b4ef97b939eaad4
```

We have to delete a dangling image by the image ID

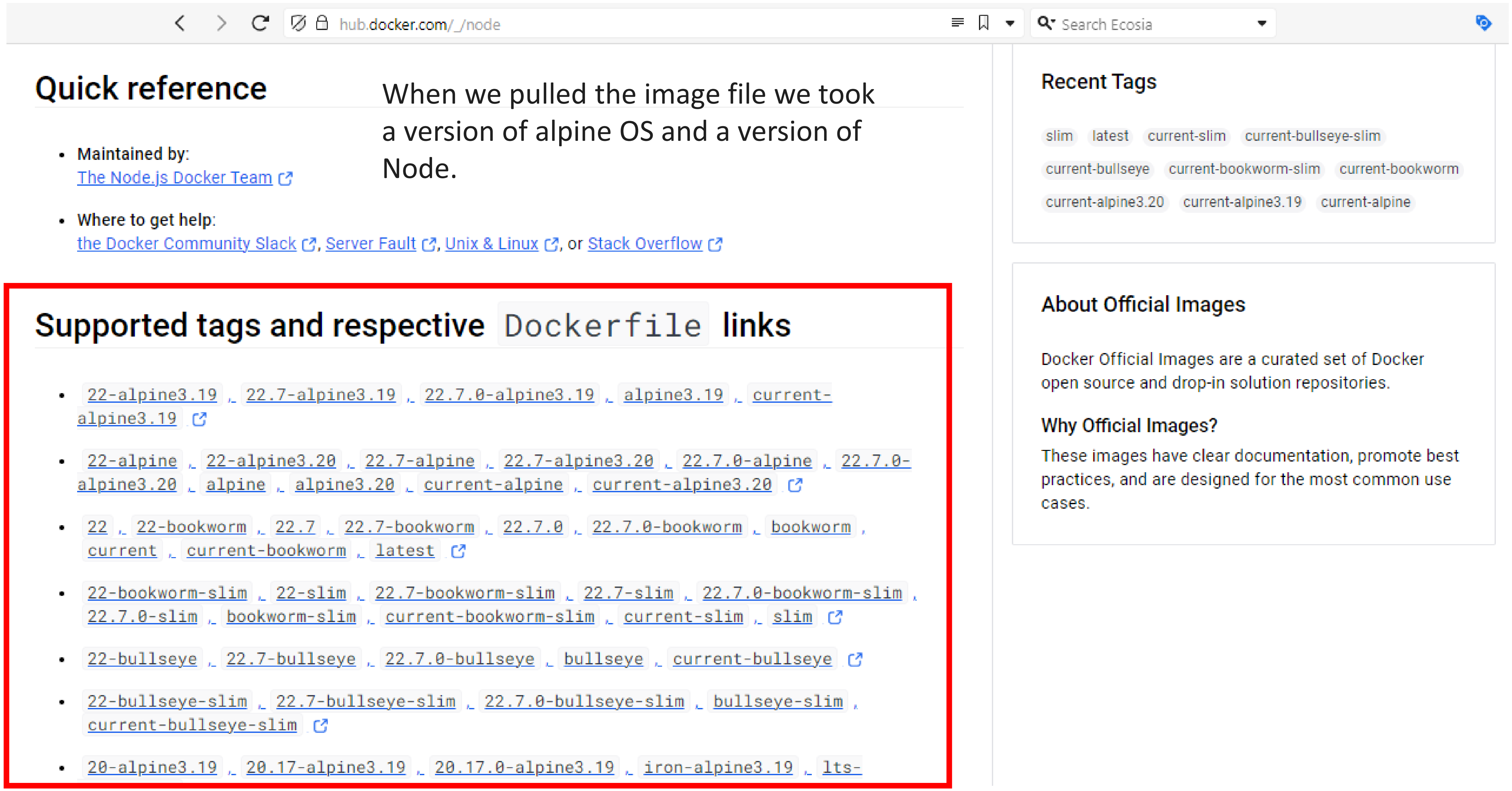
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
myapp3       latest    e7cf174cfe42   2 hours ago     123MB
myapp2       latest    5655d386938f   2 hours ago     123MB
myapp        latest    25ae2bdc48f6   21 hours ago    173MB
node         latest    675eb396b32b   12 days ago     1.11GB
```

Delete multiple containers

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                        PORTS     NAMES
acb60a67e55a   myapp          "docker-entrypoint.s…"   20 hours ago     Exited (137) 19 hours ago               myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   20 hours ago     Exited (137) 20 hours ago               myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   21 hours ago     Exited (137) 20 hours ago               myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   24 hours ago     Exited (137) 20 hours ago               laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker container rm myapp_c3 myapp_c2 myapp1_C
myapp_c3
myapp_c2
myapp1_C

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                        PORTS     NAMES
385c7a363cf8   node      "docker-entrypoint.s…"   24 hours ago     Exited (137) 21 hours ago               laughing_davinci

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
```

Multiple containers (or images) can be deleted by tacking on additional names or IDs to the RM command.

Image Versioning



hub.docker.com/_/node

Search Ecosia

Quick reference

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

When we pulled the image file we took a version of alpine OS and a version of Node.

Supported tags and respective `Dockerfile` links

- `22-alpine3.19`, `22.7-alpine3.19`, `22.7.0-alpine3.19`, `alpine3.19`, `current-alpine3.19`
- `22-alpine`, `22-alpine3.20`, `22.7-alpine`, `22.7-alpine3.20`, `22.7.0-alpine`, `22.7.0-alpine3.20`, `alpine`, `alpine3.20`, `current-alpine`, `current-alpine3.20`
- `22`, `22-bookworm`, `22.7`, `22.7-bookworm`, `22.7.0`, `22.7.0-bookworm`, `bookworm`, `current`, `current-bookworm`, `latest`
- `22-bookworm-slim`, `22-slim`, `22.7-bookworm-slim`, `22.7-slim`, `22.7.0-bookworm-slim`, `22.7.0-slim`, `bookworm-slim`, `current-bookworm-slim`, `current-slim`, `slim`
- `22-bullseye`, `22.7-bullseye`, `22.7.0-bullseye`, `bullseye`, `current-bullseye`
- `22-bullseye-slim`, `22.7-bullseye-slim`, `22.7.0-bullseye-slim`, `bullseye-slim`, `current-bullseye-slim`
- `20-alpine3.19`, `20.17-alpine3.19`, `20.17.0-alpine3.19`, `iron-alpine3.19`, `lts-`

Recent Tags

slim latest current-slim current-bullseye-slim current-bullseye current-bookworm-slim current-bookworm current-alpine3.20 current-alpine3.19 current-alpine

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

Image Versioning tag

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                      PORTS     NAMES
acb60a67e55a   myapp          "docker-entrypoint.s…"   20 hours ago   Exited (137) 19 hours ago             myapp_c3
ea997c293d9b   myapp          "docker-entrypoint.s…"   20 hours ago   Exited (137) 20 hours ago             myapp_c2
66759f754524   myapp:latest   "docker-entrypoint.s…"   21 hours ago   Exited (137) 20 hours ago             myapp1_C
385c7a363cf8   node           "docker-entrypoint.s…"   24 hours ago   Exited (137) 20 hours ago             laughing_davinci
```

The image is denoted by its name then a colon then a tag to identify the version.

Docker system prune (1)

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
```

To clean up our lab environment we can remove all images, containers and volumes with a **docker system prune –a** command.

Docker system prune (2)

```
Deleted Containers:
385c7a363cf8ee6b4166ea272f2c7dcabf8f79de038d59f1ff15e22ccc28f11c

Deleted Images:
untagged: myapp:latest
deleted: sha256:25ae2bdc48f6e54e0441b7ed7fa37e2ea4b3cdc4446059fdc05c99f2f9b879db
untagged: myapp2:latest
deleted: sha256:5655d386938f67013f9d94992b10a96baddafcdd94f0eed663b9d9f96b2262a4
untagged: myapp3:latest
deleted: sha256:e7cf174cfe429e5102ef8a31e49e94046718eac82ef5a0964d31796bde339145
untagged: node:latest
untagged: node@sha256:54b7a9a6bb4ebfb623b5163581426b83f0ab39292e4df2c808ace95ab4cba94f
deleted: sha256:675eb396b32bb59364b89b3e05c198cbdd574eefc0ac9a0d2b9329b366da889f
deleted: sha256:69818a7fb64eaa4fb05e75998f8eb8f0f20f3d4d0ffbcc37090305bf30404034
deleted: sha256:21fc637509729cc07e26e5c4d66fca95e678d0f4194cdbd25f423e443db9c4ad
deleted: sha256:fb561c1d14bca545470766907c242250be2e8fd5c094aee4efa6b8a9cc4b489a
deleted: sha256:3882edd1e30858a0cc51ab30979e3ae477f00f3cfdf401eb9e3f04c8d0657a69
deleted: sha256:29052490f610e28ac46fb90ef58dd6daea743246b5f8a31247ec49bb4dc4c7f8
deleted: sha256:f1927c507a2107d532c655187f48c2b1b716bd0bcaafa14182d6f1361bd0fbc1
deleted: sha256:e2bab150e41cdd3ca2ad8f90cb8a41d18412e2a0a0da6970faae4fc5e3a91efe
deleted: sha256:8f4ceb8cc1a2056b98f0424fad4715dd334aecc9769186b3ea0394f131524e27


...
```

From the prune command output we can see all images being removed.

Docker system prune (2)

```
Deleted build cache objects:
suc3kbko32c6ou889fja2c0jn
ihjziwlmc13u0y8t71szlkm48
mdgjyq4fgondvtmxmg5ovdox3
pwlwhlsz9qw8hurm0cbfic0r4
dch8zk78kzrw9qaav0vwmjoyj
aq8by7ccu1tshrm0mghmzbzb9
q07g420b5w1oitsdobzfw4fch
s6v7v2v4d4klbmi5aedogiumv
jycsv9lnnx0gqecra0nkdctx8
vtdkg0eya5jw412ko8wb2upj9
4fdgwf9t9as1ddlugolfqej1u
ysgfa1grmt8jd3ycqkrjwju7n
se50tc4hz48p70t650lzvrool
h024fophy7gi14gsus8s8jyrt
9lbb1wm0msx2yo9yj4jmubgpi
wv5l3wjf05l1y6ntb9hqybydt
e1tc8fswj0q5w44fy1606d3a7
y3o3ex0yvw0lvui6uywh7oz77
6dqwgy0jqzmln2fg1cknkjpde
388tezwhttkdmbmskfwv5euq1
g0n2iqr36oil7manlaaqxcnj4
mny6yyfvd9c789zvr4qhwqtgq
htngje2leo4syggz8vksfxszt
qmcjkvcdgn8iqmgzwqp43qltp

Total reclaimed space: 1.136GB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
```

From the prune command output we can see all cache objects being removed.

This command is **irreversible** and work will be lost if the command is used unwisely.

How to add Docker image version tag

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker build -t myapp:v1 .
[+] Building 10.8s (11/11) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                   0.1s
 => => transferring dockerfile: 155B                                                                                                   0.0s
 => [internal] load metadata for docker.io/library/node:16-alpine                                                                      1.9s
 => [auth] library/node:pull token for registry-1.docker.io                                                                            0.0s
 => [internal] load .dockerignore                                                                                                      0.1s
 => => transferring context: 52B                                                                                                       0.0s
 => [1/5] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787                4.2s
 => => resolve docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787                0.1s
 => => sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787 1.43kB / 1.43kB                                         0.0s
 => => sha256:72e89a86be58c922ed7b1475e5e6f151537676470695dd106521738b060e139d 1.16kB / 1.16kB                                         0.0s
 => => sha256:2573171e0124bb95d14d128728a52a97bb917ef45d7c4fa8cfe76bc44aa78b73 6.73kB / 6.73kB                                         0.0s
 => => sha256:7264a8db6415046d36d16ba98b79778e18accee6ffa71850405994cffa9be7de 3.40MB / 3.40MB                                         0.4s
 => => sha256:eee371b9ce3ffdbb8aa703b9a14d318801ddc3468f096bb6cfeabbeb715147f9 36.63MB / 36.63MB                                       1.8s
 => => sha256:93b3025fe10392717d06ec0d012a9ffa2039d766a322aac899c6831dd93382c2 2.34MB / 2.34MB                                         0.6s
 => => extracting sha256:7264a8db6415046d36d16ba98b79778e18accee6ffa71850405994cffa9be7de                                              0.2s
 => => sha256:d9059661ce70092af66d2773666584fc8addcb78a2be63f720022f4875577ea9 452B / 452B                                             0.7s
 => => extracting sha256:eee371b9ce3ffdbb8aa703b9a14d318801ddc3468f096bb6cfeabbeb715147f9                                              1.5s
 => => extracting sha256:93b3025fe10392717d06ec0d012a9ffa2039d766a322aac899c6831dd93382c2                                              0.1s
 => => extracting sha256:d9059661ce70092af66d2773666584fc8addcb78a2be63f720022f4875577ea9                                              0.0s
 => [internal] load build context                                                                                                      0.1s
 => => transferring context: 34.28kB                                                                                                   0.0s
 => [2/5] WORKDIR /app                                                                                                                 0.6s
 => [3/5] COPY package.json .                                                                                                          0.2s
 => [4/5] RUN npm install                                                                                                              3.0s
 => [5/5] COPY . .                                                                                                                     0.2s
 => exporting to image                                                                                                                 0.3s
 => => exporting layers                                                                                                                0.3s
 => => writing image sha256:8f44707112bd68623f5a250b24bd2841e3dc8b58fb476d437f173d73ca91227c                                           0.0s
 => => naming to docker.io/library/myapp:v1                                                                                            0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/r4qcd60i02emvc88rlvbciyd2
```

When building the image from the command line we can manually specify our own tag by using **docker build –t [app name]:[version]**

Run container on Specific Image Version

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
myapp         v1        8f44707112bd   3 minutes ago   124MB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6/api
$ docker run --name mmyapp_c -p 4000:4000 myapp:v1
listening for requests on port 4000
```

Now when we view the docker images we can see the version tag.

To run a container on this specific version we just use the run command adding :version to the end of the image name.
docker run –name myapp_c p4000:4000 [app name]:[version]

#9 Docker Volumes

Why do we need Volumes?

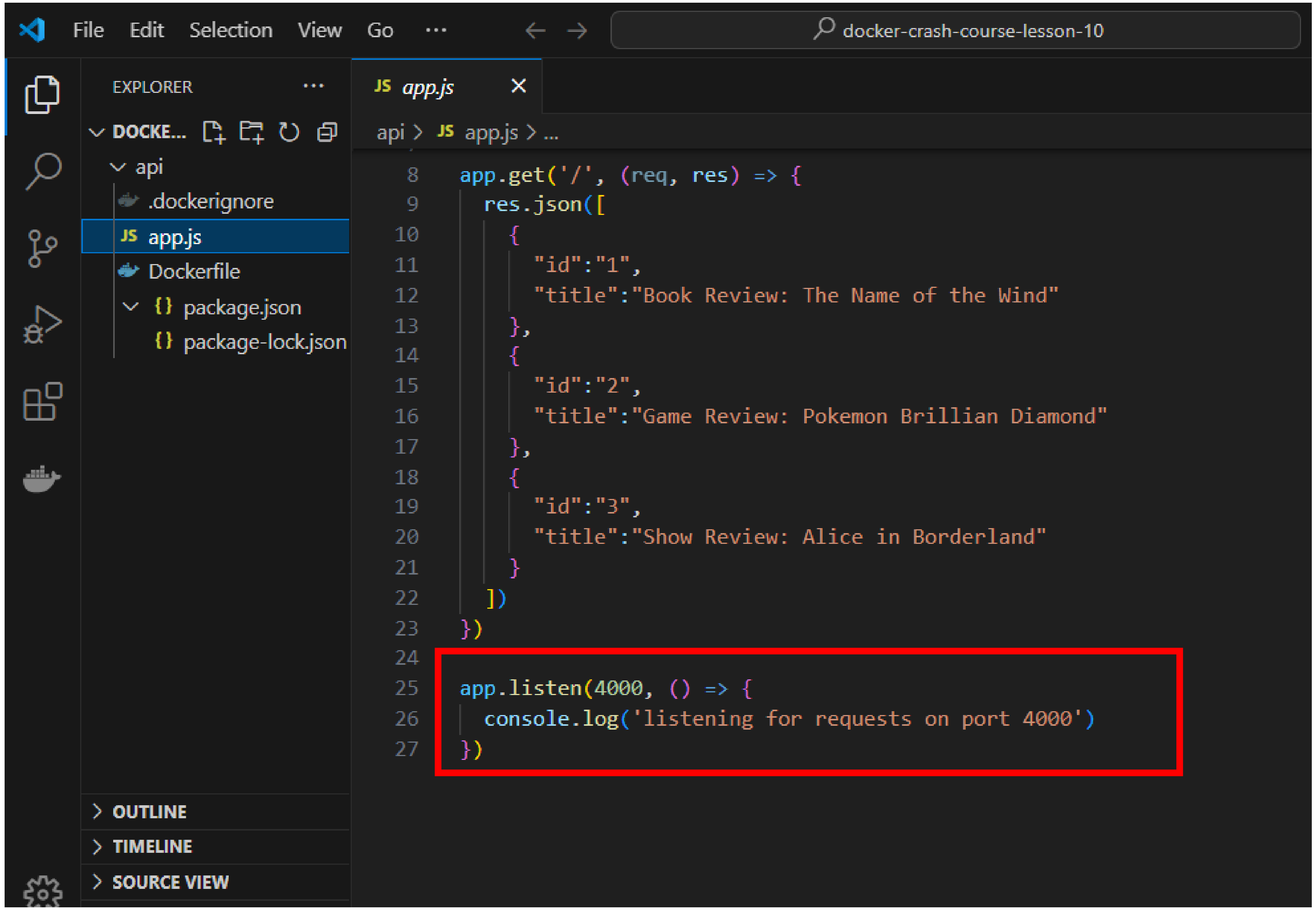
Reminder:

1. Images are read only so once changes are made to an app then a new image has to be built.
2. Docker run will always create a new container from an image.
3. Whereas docker start will start an existing container.
4. Docker start will run a container in detached mode.
5. Whereas Docker run, will by default block, the command line.

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                    NAMES
72c1d37bae00   myapp:v1   "docker-entrypoint.s…"   34 minutes ago   Up 34 minutes   0.0.0.0:4000->4000/tcp   mmyapp_c

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker stop 72c1d37bae00
72c1d37bae00

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-6
```

There is one running container that we can stop using the ID or container name.

Changes to the source code



```
app.get('/', (req, res) => {
  res.json([
    {
      "id":"1",
      "title":"Book Review: The Name of the Wind"
    },
    {
      "id":"2",
      "title":"Game Review: Pokemon Brillian Diamond"
    },
    {
      "id":"3",
      "title":"Show Review: Alice in Borderland"
    }
  ])
})

app.listen(4000, () => {
  console.log('listening for requests on port 4000')
})
```

The source code has changed to include additional lines to console log.

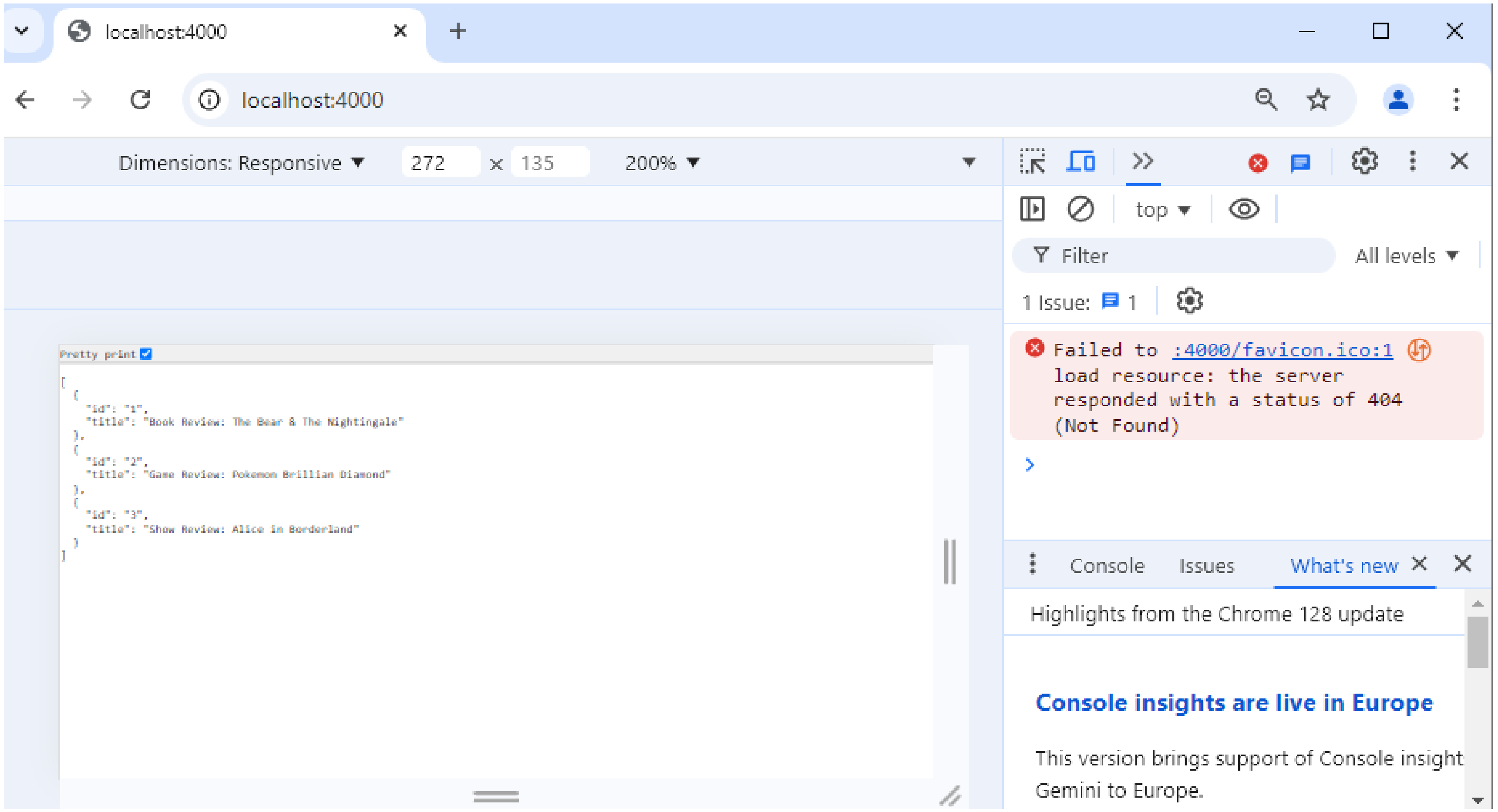
Changes to the source code are Non persistent

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS                       PORTS     NAMES
72c1d37bae00   myapp:v1   "docker-entrypoint.s…"   44 minutes ago   Exited (137) 9 minutes ago             mmyapp_c

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker start mmyapp_c
mmyapp_c
```

When a container is started, (Not created) any changes to the code will not be freflected because the image is read only and changes are non persistent.

To see these changes we would need to build a new image then run a new container to see these changes and this could be a long winded way of doing things.

This is not viable in the fast paced world of software and app dev but fortunately there is a way round this using Volumes.



```
[
  {
    "id": "1",
    "title": "Book Review: The Bear & The Nightingale"
  },
  {
    "id": "2",
    "title": "Game Review: Pokemon Brillian Diamond"
  },
  {
    "id": "3",
    "title": "Show Review: Alice in Borderland"
  }
]
```

```
Failed to :4000/favicon.ico:1
load resource: the server
responded with a status of 404
(Not Found)
```

Highlights from the Chrome 128 update

Console insights are live in Europe

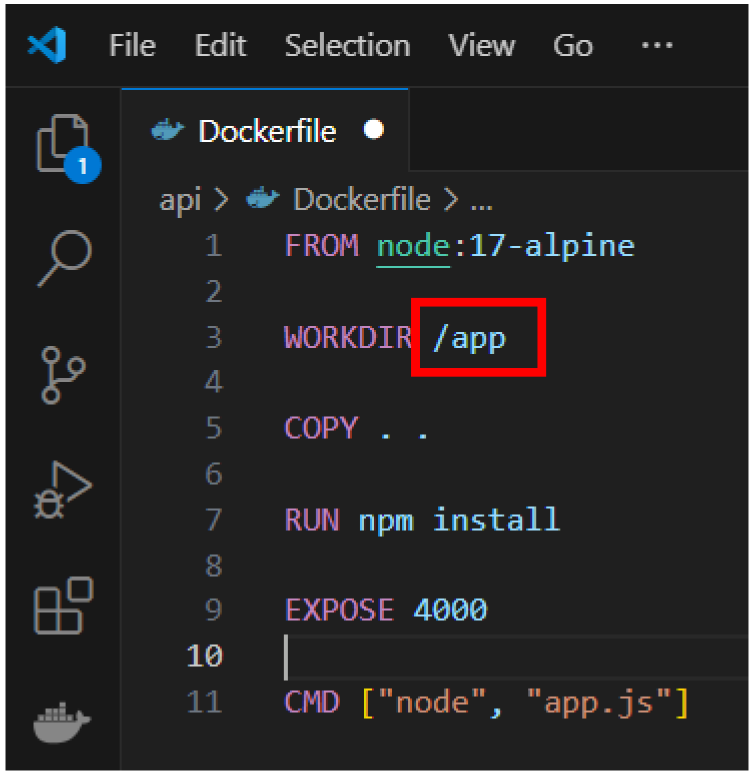
This version brings support of Console insight
Gemini to Europe.

Volumes introduction

Since the image does not update then the image is still running the old code.

Volumes are a feature of docker that allow us to specify folders on our host computer that can be made available to folders in the running container so that any changes in the computer will also be reflected in the folder of the running container.

We could map the entire projcect folder, the api folder, to the app folder in our container because we specified that in the docker file. This means that if any file is created, deleted or updated then these changes would be reflected in the container.



```
FROM node:17-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 4000

CMD ["node", "app.js"]
```

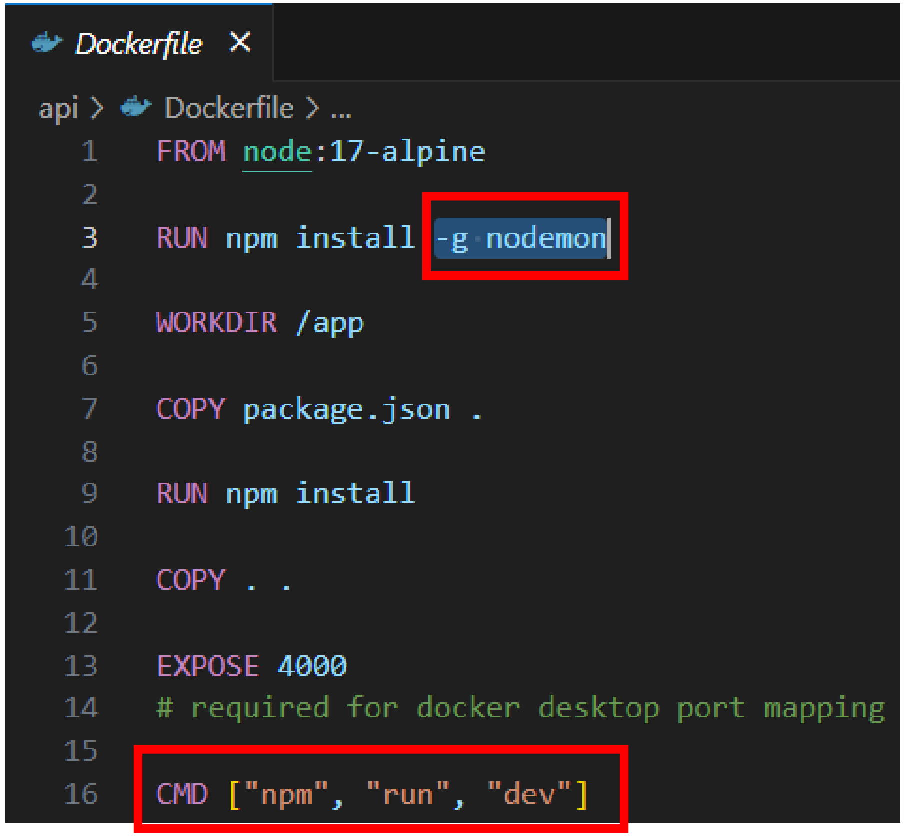
We would see the update without having to build a new image. This is a way that we can make changes to the project without having to build a new image every time.

Everything in the api folder of the project would be mapped to the app folder of the container so changes in the dev environment would be reflected in the docker container.

One important thing to note is that the image itself does not change. Volumes just map directories between containers and host computer. the image remains the same.

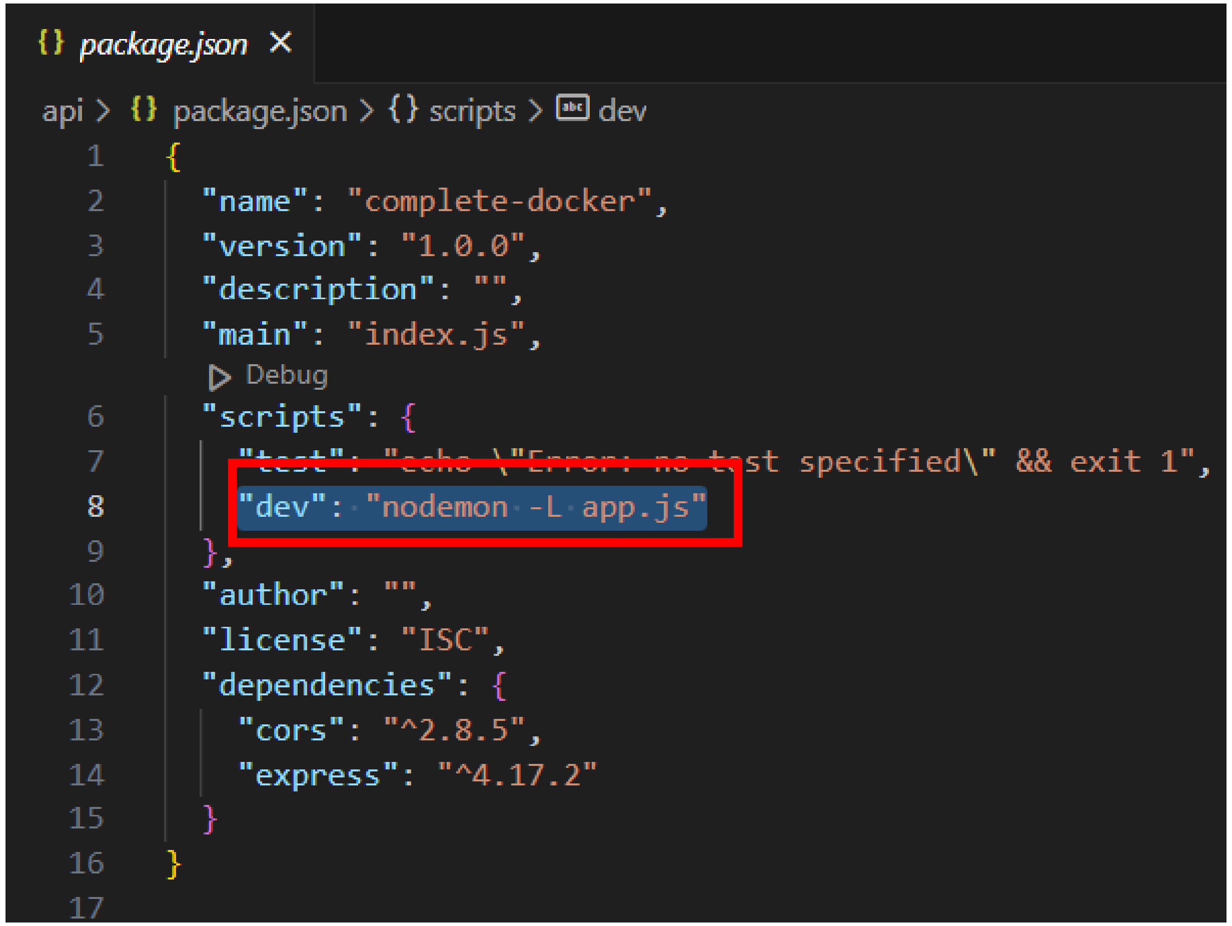
While live developing volumes can be used but if you wantd to share the app with others or deploy to a server then you would have to build a new image.

Setting up Node to Reflect Changes



```dockerfile
FROM node:17-alpine

RUN npm install -g nodemon

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 4000
# required for docker desktop port mapping

CMD ["npm", "run", "dev"]
```

First we modify the docker file to include nodemon globally. Nodemon watches the JS and json code files for any changes and restarts the node server automatically when changes are detected. Without this we would have to restart the node server automatically each time we make a change to files.



```json
{
  "name": "complete-docker",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon -L app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.2"
  }
}
```

To trigger nodemon we are going to do it with a dev script from the packages.json file. Note the -L flag which is a requirement to get nodemon working on a windows machine.

Finally the CMD of the Dockerfile is modified to call the dev script. We could have put this directly into the Dockerfile but it is cleaner code to put it in the packages.json.

Build a new image to reflect Node Changes

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker build -t myapp:nodemon .
[+] Building 20.8s (12/12) FINISHED                                                   docker:desktop-linux
...
Omitted for berevity
...
 => [internal] load build context                                                                 0.1s
 => => transferring context: 34.35kB                                                              0.0s
 => [2/6] RUN npm install -g nodemon                                                              6.3s
 => [3/6] WORKDIR /app                                                                            0.2s
 => [4/6] COPY package.json .                                                                     0.2s
 => [5/6] RUN npm install                                                                         5.7s
 => [6/6] COPY . .                                                                                0.2s
 => exporting to image                                                                            0.5s
 => => exporting layers                                                                           0.4s
 => => writing image sha256:eec02eafe8d1f7ac6fe9eb3b1e9acfee5a1b48900ac62fc5d00a3c640ca09d01      0.0s
 => => naming to docker.io/library/myapp:nodemon                                                  0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/lobfxf85d0cszqaf1xl7fy4oo
```

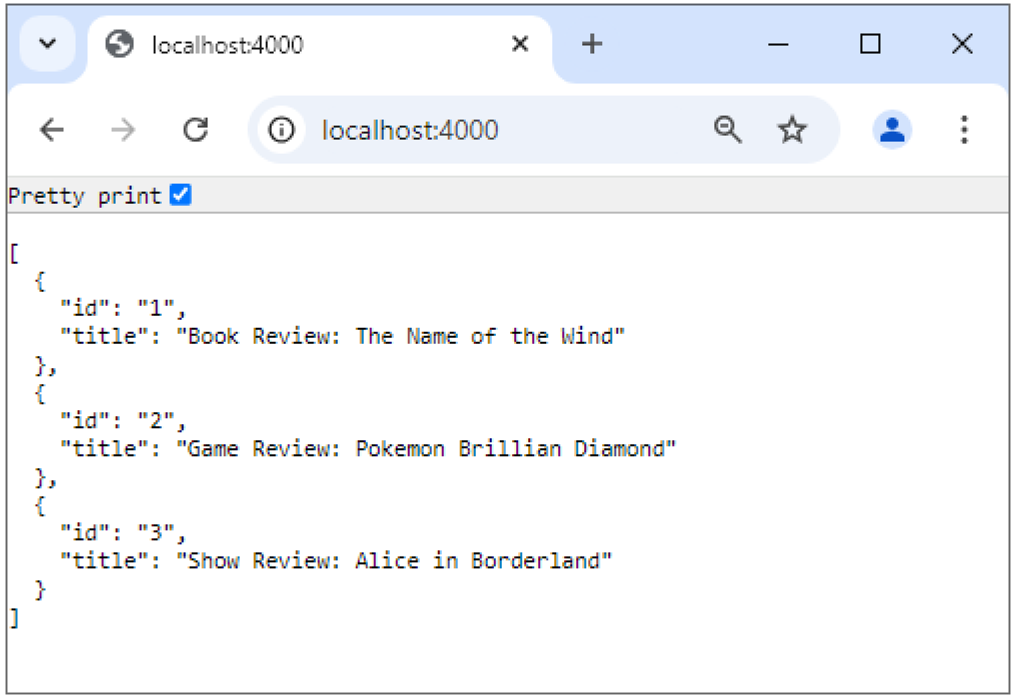
The Image is built

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker run --name myapp_c_nodemon -p 4000:4000 --rm myapp:nodemon

> complete-docker@1.0.0 dev
> nodemon -L app.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
listening for requests on port 4000
```

Now I create the container with the run command. I give the container a name and create port mapping. Note the –rm part. This automatically deletes the container when I stiop it to keep a clean Docker environment. Finally I specify the image and version

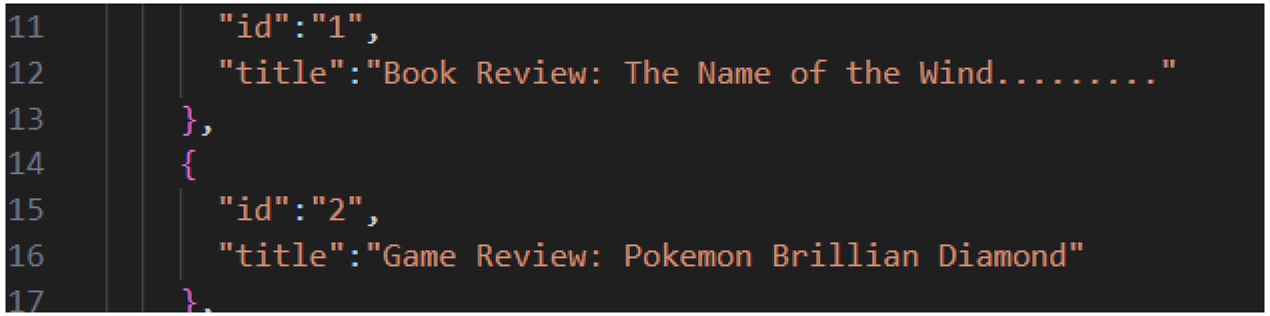
This now outputs some nodemon

Verify image, container and nodemon



```
[
  {
    "id": "1",
    "title": "Book Review: The Name of the Wind"
  },
  {
    "id": "2",
    "title": "Game Review: Pokemon Brillian Diamond"
  },
  {
    "id": "3",
    "title": "Show Review: Alice in Borderland"
  }
]
```

The app opens in the browser on port 4000.



```
      "id":"1",
      "title":"Book Review: The Name of the Wind........."
    },
    {
      "id":"2",
      "title":"Game Review: Pokemon Brillian Diamond"
```

The app.js file is modified slightly and the browser is refreshed. Why are the changes not being reflected?

Because the file that I have modified is not inside a volume, it is inside the container which nodemon is watching. This is where volumes now come into play.

Setting up Volume (1)

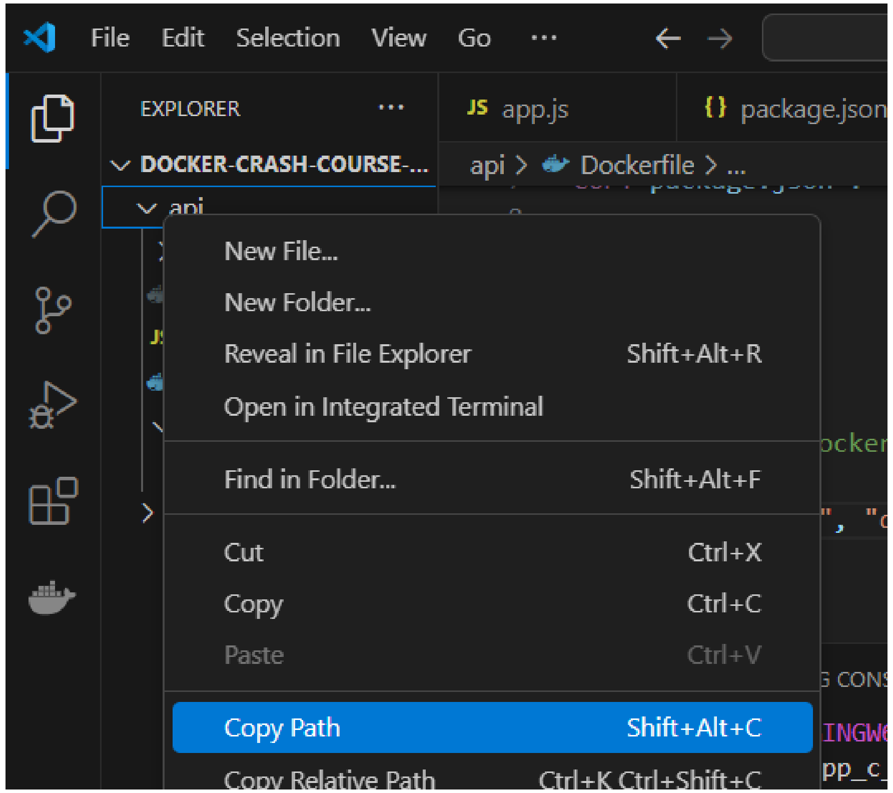
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10
$ docker ps
CONTAINER ID   IMAGE            COMMAND                    CREATED          STATUS           PORTS
     NAMES
c06452eeb7f1   myapp:nodemon    "docker-entrypoint.s…"    10 minutes ago   Up 10 minutes    0.0.0.0:4000->4000/tcp   myapp_c_nodemon

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10
$ docker stop myapp_c_nodemon
myapp_c_nodemon

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10
```

First we stop the running container myapp_c_nodemon

Because the container was built with a –rm, it is deleted when it is stopped

Setting up Volume (2)

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker run --name myapp_c_nodemon -p 4000:4000 --rm  -v "C:\Users\ellio\Documents\CODING-LESSONS\12-
Docker\docker-crash-course-lesson-10\api:/app" myapp:nodemon

> complete-docker@1.0.0 dev
> nodemon -L app.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
listening for requests on port 4000
```



```
docker run \
--name myapp_c_nodemon \
-p 4000:4000 \
--rm \
-v "C:\Users\ellio\Documents\CODING-LESSONS\12-
Docker\docker-crash-course-lesson-10\api:/app"
\ myapp:nodemon
```

This time we run the image to create the container as before but inserting a volume mapping clause (-v) where we specify the **absolute path to the source folder** (found by right clicking in VS code and copying path) then a colon and the **destination path in the container**

Anonymous Volume for node_modules

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-10/api
$ docker run --name myapp_c_nodemon -p 4000:4000 --rm -v "C:\Users\ellio\Documents\CODING-LESSONS\12-Docker\docker-
crash-course-lesson-10\api:/app" -v /app/node_modules myapp:nodemon

> complete-docker@1.0.0 dev
> nodemon -L app.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
listening for requests on port 4000
```

Now when the project folder changes it will be kept in sync with our container.

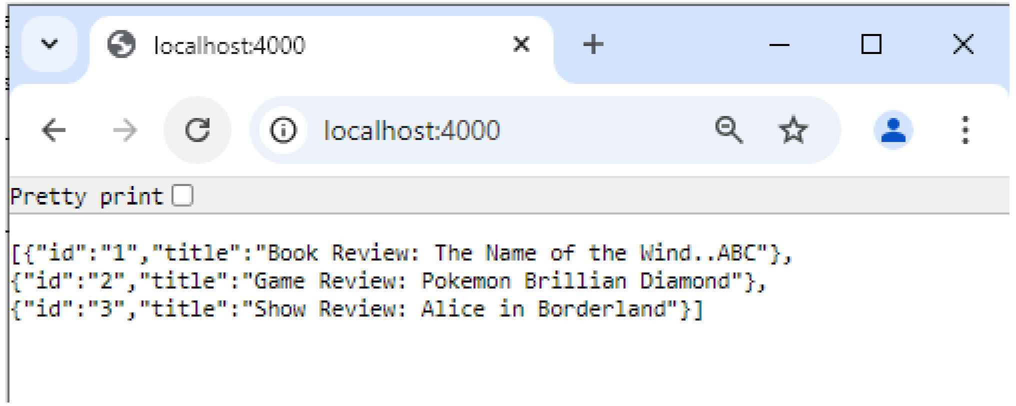
However there is a problem. If something happns to the node_modules folder in the project files then the app would not run from the container.

We need a way to map our volume to the container. we can do this by adding another volume called an **anonymous volume** which will map the containers node_modules container to somewhere else on our computer. In this case we map the node_modules to the **node_modules folder in the container.**

If the node modules folder in the project files is deleted then the app will still run.

Verify Volumes

```
[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
listening for requests on port 4000
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
listening for requests on port 4000
```



```
[{"id":"1","title":"Book Review: The Name of the Wind..ABC"},
{"id":"2","title":"Game Review: Pokemon Brillian Diamond"},
{"id":"3","title":"Show Review: Alice in Borderland"}]
```

Now when changes are made in the code the **nodemon restarts the node server** which can be seen in the terminal. The changes are also reflected in the browser.

#10 Docker Compose

Why Docker Compose?

Previously we typed out a log command to specify the container, ports, volumes and image. This will get tedious especially if we are opening multiple containers simultaneously such as a MongoDB, an node JSapp and a react front end.

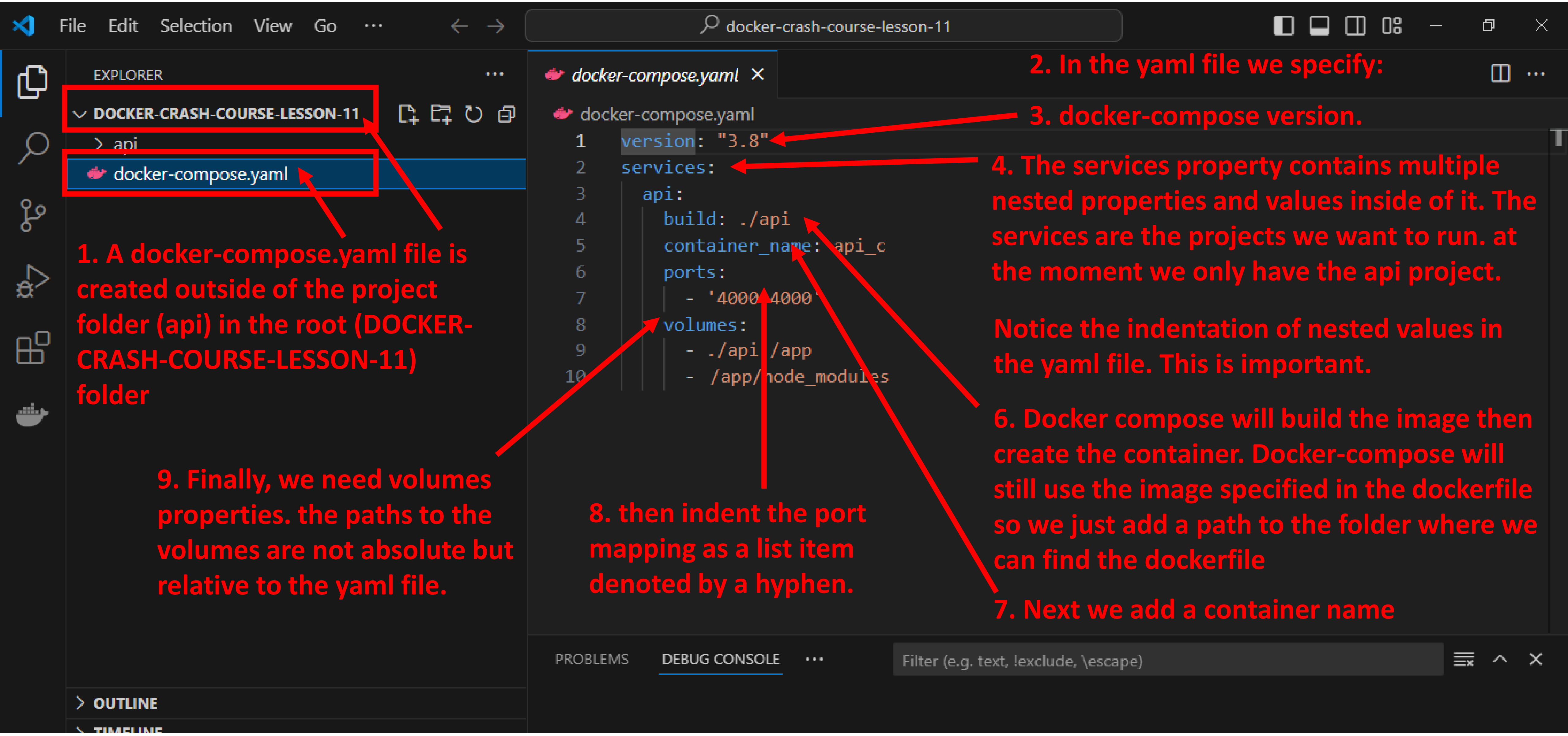
We can use a docker compose file to list out all the containers that we want to open then we just call the compose file to run multiple containers.

The compose goes in the root directory of the project.

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - unused build cache

Are you sure you want to continue? [y/N] y
Deleted build cache objects:
jt29u3lpwhbega6b3qegnt7aj
u95h9jmbm4he3k0xm49fc9yac
s5ygf7ef5wk2kcrdlervp6z0z

Total reclaimed space: 34.35kB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
```

To start this I am going to clean up the docker lab by erasing images, caches, containers using docker system prune.

docker-compose.yaml file


File Edit Selection View Go ···
docker-crash-course-lesson-11
EXPLORER
DOCKER-CRASH-COURSE-LESSON-11
api
docker-compose.yaml
OUTLINE
TIMELINE
PROBLEMS
DEBUG CONSOLE
Filter (e.g. text, !exclude, \escape)

```yaml
version: "3.8"
services:
  api:
    build: ./api
    container_name: api_c
    ports:
      - '4000:4000'
    volumes:
      - ./api:/app
      - /app/node_modules
```

1. A docker-compose.yaml file is created outside of the project folder (api) in the root (DOCKER-CRASH-COURSE-LESSON-11) folder

2. In the yaml file we specify:

3. docker-compose version.

4. The services property contains multiple nested properties and values inside of it. The services are the projects we want to run. at the moment we only have the api project.

Notice the indentation of nested values in the yaml file. This is important.

6. Docker compose will build the image then create the container. Docker-compose will still use the image specified in the dockerfile so we just add a path to the folder where we can find the dockerfile

7. Next we add a container name

8. then indent the port mapping as a list item denoted by a hyphen.

9. Finally, we need volumes properties. the paths to the volumes are not absolute but relative to the yaml file.

docker-compose up

From the folder containing the composer

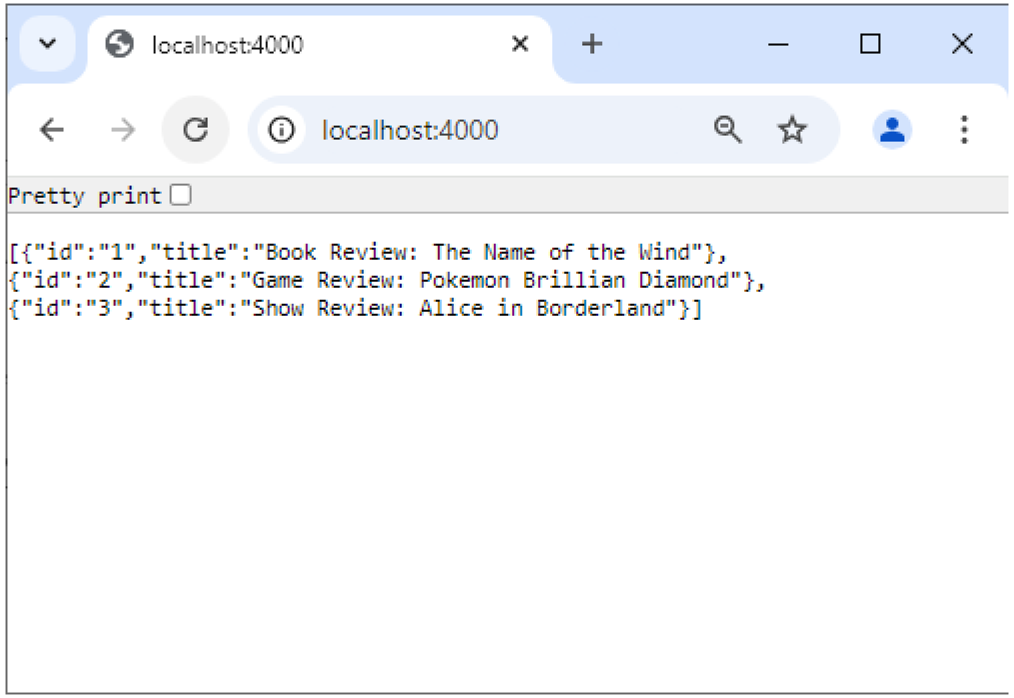
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ ls
api/  docker-compose.yaml

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ docker-compose up
time="2024-09-05T01:21:16+02:00" level=warning msg="C:\\Users\\ellio\\Documents\\CODING-LESSONS\\12-Docker\\docker-crash-course-lesson-11\\docker-
compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 1.2s (13/13) FINISHED                                                     docker:desktop-linux
 => [api internal] load build definition from Dockerfile                                               0.0s
 => => transferring dockerfile: 231B                                                                   0.0s
 => [api internal] load metadata for docker.io/library/node:17-alpine                                  1.0s
 => [api auth] library/node:pull token for registry-1.docker.io                                        0.0s
 => [api internal] load .dockerignore                                                                  0.0s
 => => transferring context: 52B                                                                       0.0s
 => [api 1/6] FROM docker.io/library/node:17-alpine@sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595  0.0s
 => [api internal] load build context                                                                  0.0s
 => => transferring context: 160B                                                                      0.0s
 => CACHED [api 2/6] RUN npm install -g nodemon                                                        0.0s
 => CACHED [api 3/6] WORKDIR /app                                                                      0.0s
 => CACHED [api 4/6] COPY package.json .                                                               0.0s
 => CACHED [api 5/6] RUN npm install                                                                   0.0s
 => CACHED [api 6/6] COPY . .                                                                          0.0s
 => [api] exporting to image                                                                           0.0s
 => => exporting layers                                                                                0.0s
 => => writing image sha256:da4b783d6f33989dbeb0e5eed5434a9d0d0db58fd123d3f8a98234a6a0ac5290           0.0s
 => => naming to docker.io/library/docker-crash-course-lesson-11-api                                   0.0s
 => [api] resolving provenance for metadata file                                                       0.0s
```

Use docker-compose up command. We see from the console output that the image is being built, then the container

Verify docker-compose up

```
0.0s
[+] Running 2/2
 ✓ Network docker-crash-course-lesson-11_default  Cr...                                    0.1s
 ✓ Container api_c                                Created                                  0.4s
Attaching to api_c
api_c  |
api_c  | > complete-docker@1.0.0 dev
api_c  | > nodemon -L app.js
api_c  |
api_c  | [nodemon] 3.1.4
api_c  | [nodemon] to restart at any time, enter `rs`
api_c  | [nodemon] watching path(s): *.*
api_c  | [nodemon] watching extensions: js,mjs,cjs,json
api_c  | [nodemon] starting `node app.js`
api_c  | listening for requests on port 4000


v View in Docker Desktop   o View Config   w Enable Watch
```

The webpage loads.



```
[{"id":"1","title":"Book Review: The Name of the Wind"},
{"id":"2","title":"Game Review: Pokemon Brillian Diamond"},
{"id":"3","title":"Show Review: Alice in Borderland"}]
```

docker-compose down

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ docker images
REPOSITORY                           TAG       IMAGE ID       CREATED          SIZE
docker-crash-course-lesson-11-api   latest    da4b783d6f33   24 minutes ago   177MB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ docker ps
CONTAINER ID   IMAGE                               COMMAND                  CREATED         STATUS         PORTS                    NAMES
f712f6e1c7b3   docker-crash-course-lesson-11-api   "docker-entrypoint.s…"   8 minutes ago   Up 8 minutes   0.0.0.0:4000->4000/tcp   api_c

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
```

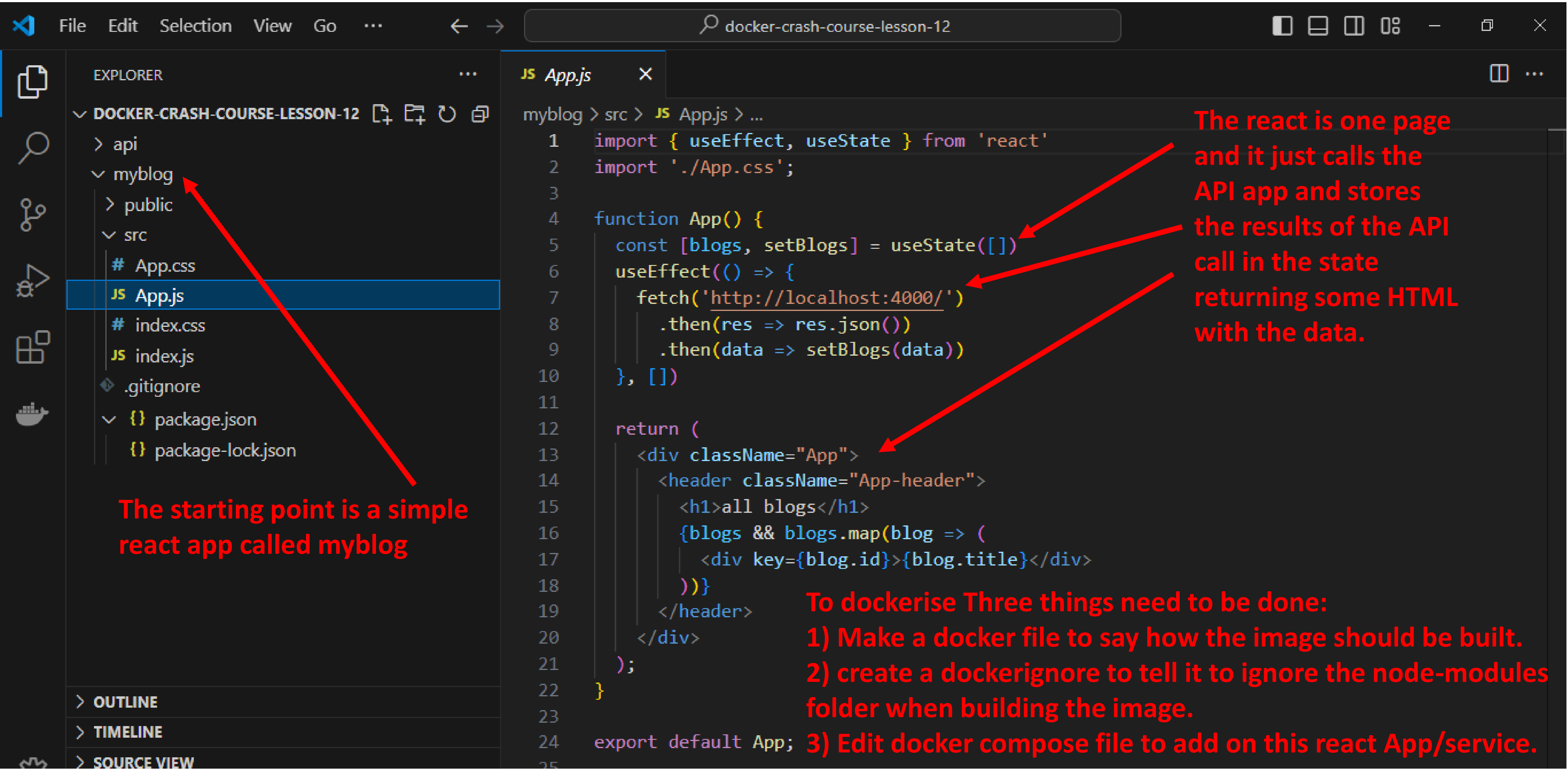
We can see the image and the container that docker-composer created.

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
$ docker-compose down --rmi all -v
time="2024-09-05T01:33:27+02:00" level=warning msg="C:\\Users\\ellio\\Documents\\CODING-LESSONS\\12-Docker\\docker-crash-course-lesson-11\\docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
 ✓ Container api_c                               Removed                                                    0.6s
 ✓ Image docker-crash-course-lesson-11-api:latest  Removed                                                    0.0s
 ✓ Network docker-crash-course-lesson-11_default   R...                                                       0.3s

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-11
```

The docker-compose down shuts down the container. We cab also append the –rmi (remove image tag) all (all images) and also –v (to remove volumes)

#11 Dockerising a React App

React App overview



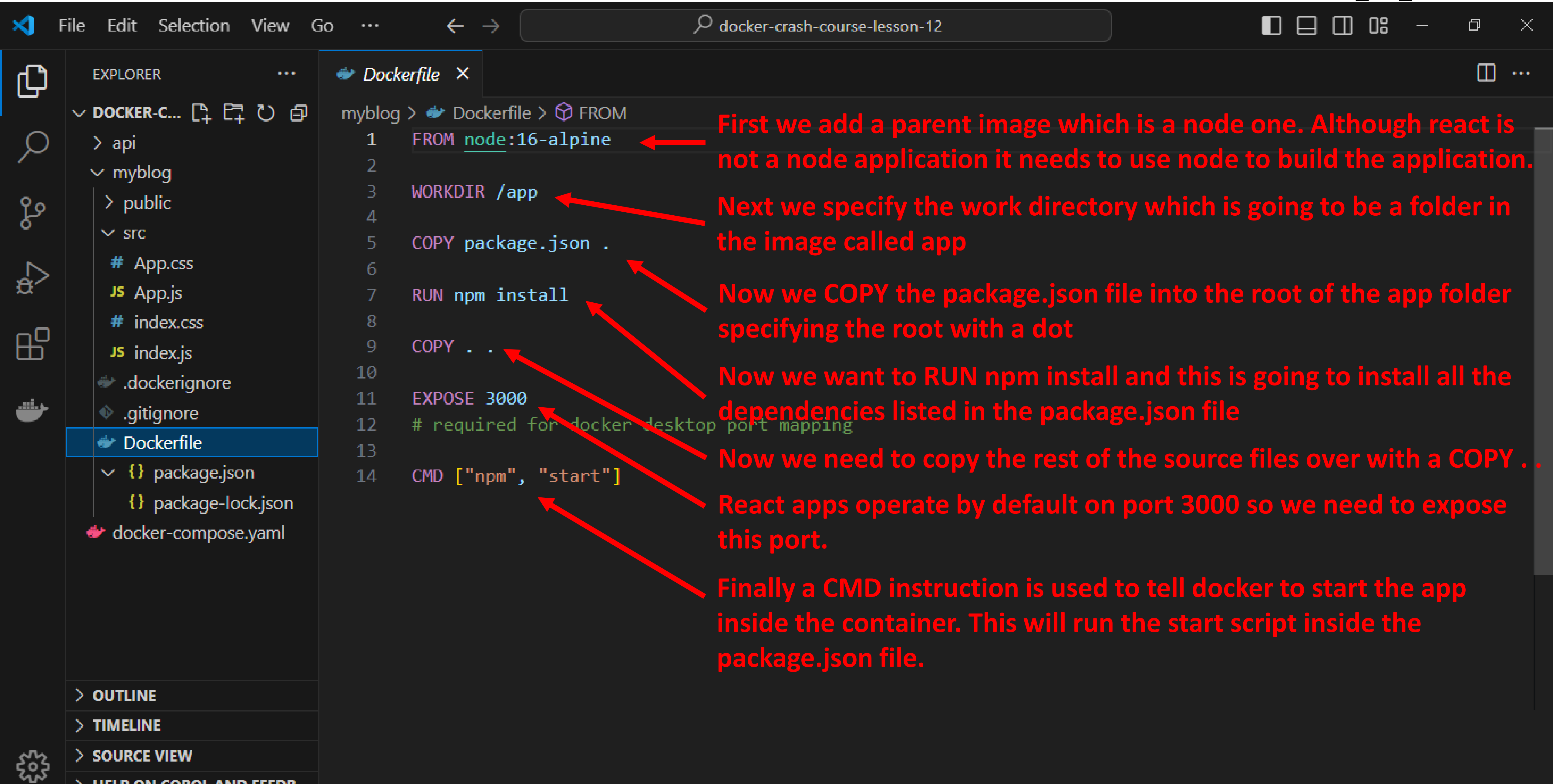
```
import { useEffect, useState } from 'react'
import './App.css';

function App() {
  const [blogs, setBlogs] = useState([])
  useEffect(() => {
    fetch('http://localhost:4000/')
      .then(res => res.json())
      .then(data => setBlogs(data))
  }, [])

  return (
    <div className="App">
      <header className="App-header">
        <h1>all blogs</h1>
        {blogs && blogs.map(blog => (
          <div key={blog.id}>{blog.title}</div>
        ))}
      </header>
    </div>
  );
}

export default App;
```

The react is one page and it just calls the API app and stores the results of the API call in the state returning some HTML with the data.

The starting point is a simple react app called myblog

To dockerise Three things need to be done:
1) Make a docker file to say how the image should be built.
2) create a dockerignore to tell it to ignore the node-modules folder when building the image.
3) Edit docker compose file to add on this react App/service.

Create the Dockerfile for the React App



```dockerfile
FROM node:16-alpine

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000
# required for docker desktop port mapping

CMD ["npm", "start"]
```

First we add a parent image which is a node one. Although react is not a node application it needs to use node to build the application.

Next we specify the work directory which is going to be a folder in the image called app

Now we COPY the package.json file into the root of the app folder specifying the root with a dot

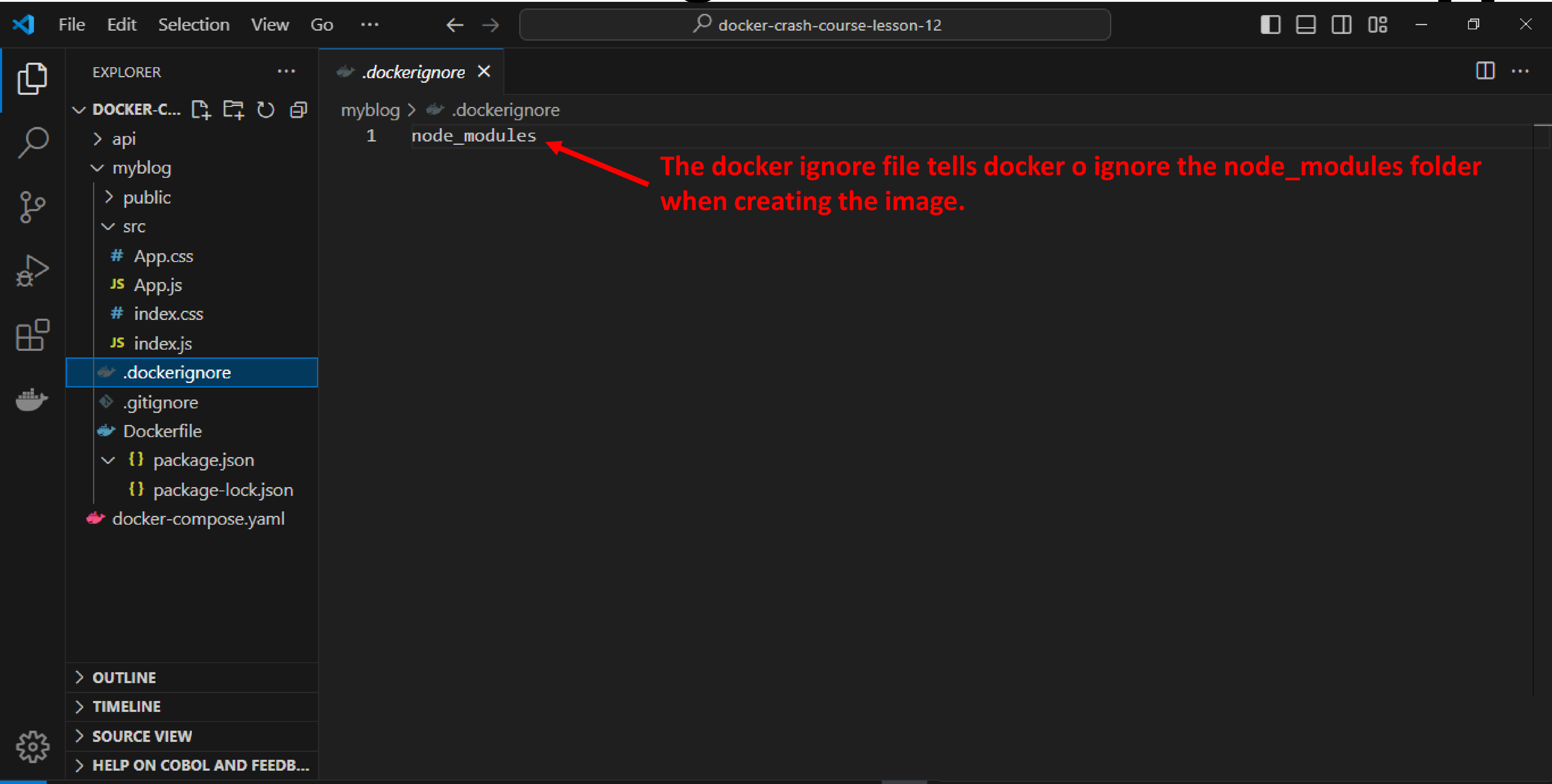
Now we want to RUN npm install and this is going to install all the dependencies listed in the package.json file

Now we need to copy the rest of the source files over with a COPY . .

React apps operate by default on port 3000 so we need to expose this port.

Finally a CMD instruction is used to tell docker to start the app inside the container. This will run the start script inside the package.json file.

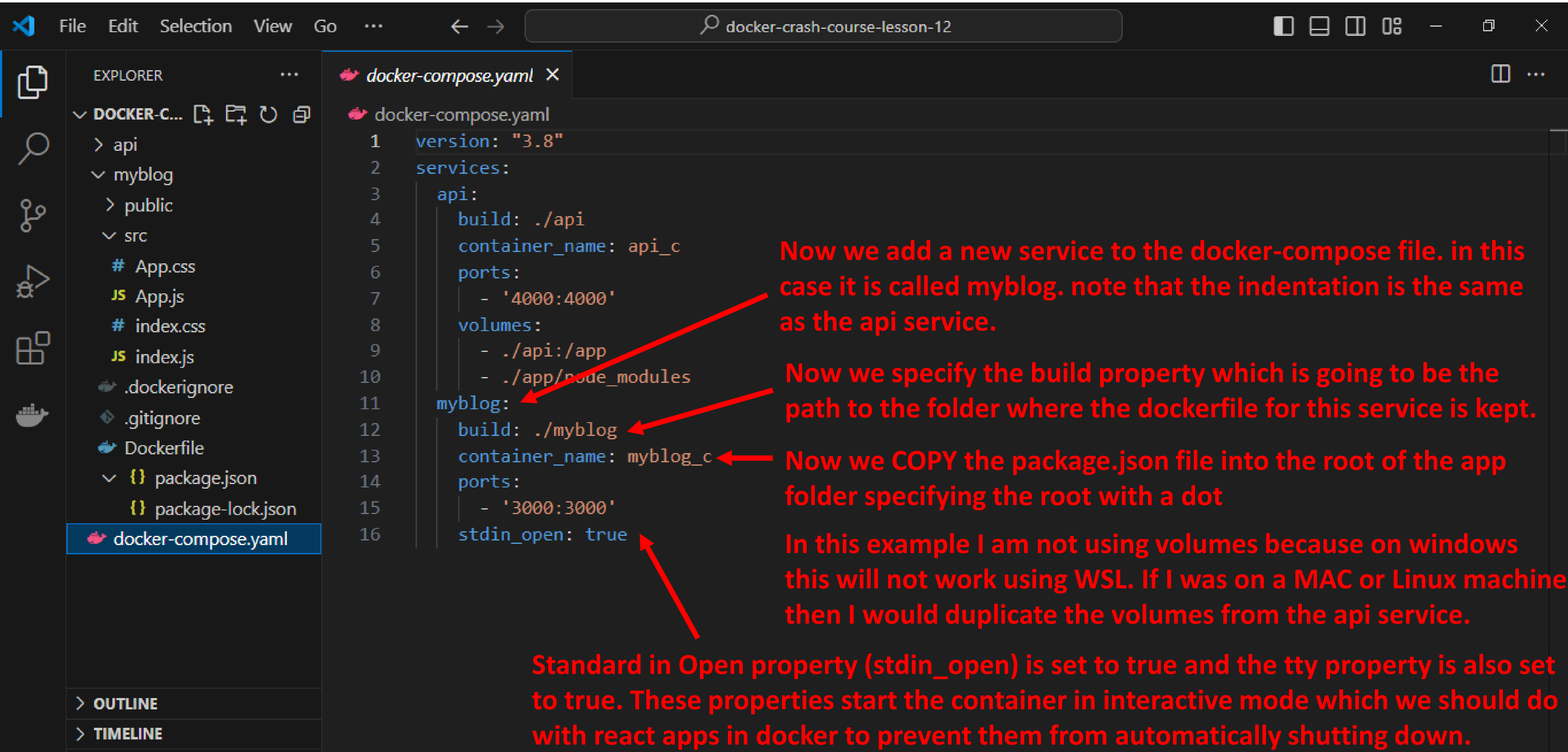
Create the Dockeignore for the React App



```
node_modules
```

The docker ignore file tells docker o ignore the node_modules folder when creating the image.

Edit Docker-compose.yaml to include React App



```yaml
version: "3.8"
services:
  api:
    build: ./api
    container_name: api_c
    ports:
      - '4000:4000'
    volumes:
      - ./api:/app
      - ./app/node_modules
  myblog:
    build: ./myblog
    container_name: myblog_c
    ports:
      - '3000:3000'
    stdin_open: true
```

Now we add a new service to the docker-compose file. in this case it is called myblog. note that the indentation is the same as the api service.

Now we specify the build property which is going to be the path to the folder where the dockerfile for this service is kept.

Now we COPY the package.json file into the root of the app folder specifying the root with a dot

In this example I am not using volumes because on windows this will not work using WSL. If I was on a MAC or Linux machine then I would duplicate the volumes from the api service.

Standard in Open property (stdin_open) is set to true and the tty property is also set to true. These properties start the container in interactive mode which we should do with react apps in docker to prevent them from automatically shutting down.

Launch React App with docker-compose up

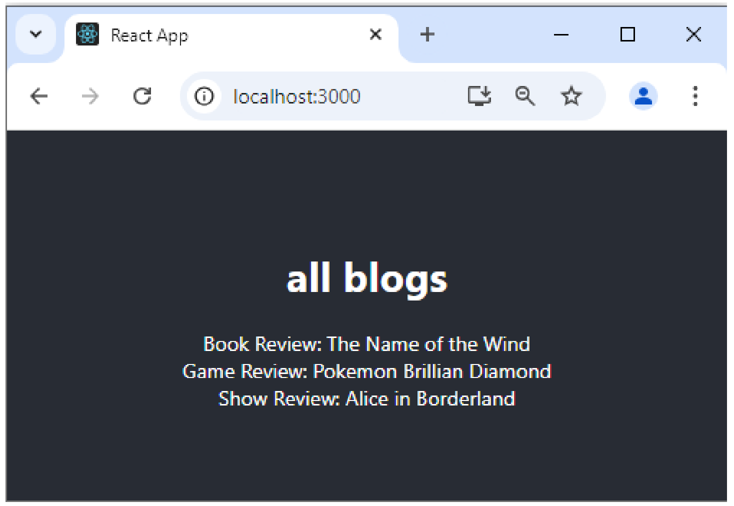
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ ls
api/  docker-compose.yaml  myblog/

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ docker-compose up
time="2024-09-05T18:19:07+02:00" level=warning msg="C:\\Users\\ellio\\Documents\\CODING-LESSONS\\12-Docker\\docker-crash-course-lesson-12\\docker-compose.yaml:
the attribute `version` is obsolete, it will be ignored, please
remove it to avoid potential confusion"
2024/09/05 18:19:07 http2: server: error reading preface from client //./pipe/dockerDesktopLinuxEngine: file has already been closed
[+] Building 0.8s (23/23) FINISHED                                                         docker:desktop-linux
 => [myblog internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 198B                                                                       0.0s
 => [api internal] load build definition from Dockerfile                                                   0.0s
 => => transferring dockerfile: 231B                                                                       0.0s
 => [myblog internal] load metadata for docker.io/library/node:16-alpine                                   0.6s
 => [api internal] load metadata for docker.io/library/node:17-alpine                                      0.6s
 => [myblog internal] load .dockerignore                                                                   0.0s
 => => transferring context: 52B                                                                           0.0s
 => [myblog 1/5] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97  0.0s
 => [myblog internal] load build context                                                                   0.0s
 => => transferring context: 566B                                                                          0.0s
 => CACHED [myblog 2/5] WORKDIR /app                                                                       0.0s
 => CACHED [myblog 3/5] COPY package.json .                                                                0.0s
 => CACHED [myblog 4/5] RUN npm install                                                                    0.0s
 => CACHED [myblog 5/5] COPY . .                                                                           0.0s
 => [myblog] exporting to image                                                                            0.0s
 => => exporting layers                                                                                    0.0s
 => => writing image sha256:eed0ef624cfe2c16894b16ef5386cf2585ea7545795f48432365b6351aab98b0               0.0s
 => => naming to docker.io/library/docker-crash-course-lesson-12-myblog                                    0.0s
 => [api internal] load .dockerignore                                                                      0.0s
 => => transferring context: 52B                                                                           0.0s
 => [api internal] load build context                                                                      0.0s
 => => transferring context: 160B                                                                          0.0s
 => [api 1/6] FROM docker.io/library/node:17-alpine@sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595d44  0.0s
 => CACHED [api 2/6] RUN npm install -g nodemon                                                            0.0s
 => CACHED [api 3/6] WORKDIR /app                                                                          0.0s
 => CACHED [api 4/6] COPY package.json .                                                                   0.0s
 => CACHED [api 5/6] RUN npm install                                                                       0.0s
 => CACHED [api 6/6] COPY . .                                                                              0.0s
 => [api] exporting to image                                                                               0.0s
 => => exporting layers                                                                                    0.0s
 => => writing image sha256:31836d0d8a7c47d1d7aba8634b29be0ac8e199044ba93f05fe8af66d2aa22ba0               0.0s
 => => naming to docker.io/library/docker-crash-course-lesson-12-api                                       0.0s
 => [myblog] resolving provenance for metadata file                                                        0.0s
 => [api] resolving provenance for metadata file                                                           0.0s
```

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ ls
api/  docker-compose.yaml  myblog/

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ docker-compose up
. . .
[+] Running 3/3
 ✓ Network docker-crash-course-lesson-12_default  Creat...                                                  0.0s
 ✓ Container api_c                                Created                                                   0.4s
 ✓ Container myblog_c                             Created                                                   0.1s
Attaching to api_c, myblog_c
myblog_c  |
myblog_c  | > myblog@0.1.0 start
myblog_c  | > react-scripts start
myblog_c  |
api_c     |
api_c     | > complete-docker@1.0.0 dev
api_c     | > nodemon -L app.js
api_c     |
api_c     | [nodemon] 3.1.4
api_c     | [nodemon] to restart at any time, enter `rs`
api_c     | [nodemon] watching path(s): *.*
api_c     | [nodemon] watching extensions: js,mjs,cjs,json
api_c     | [nodemon] starting `node app.js`
api_c     | listening for requests on port 4000
myblog_c  | (node:25) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the
'setupMiddlewares' option.
myblog_c  | (Use `node --trace-deprecation ...` to show where the warning was created)
myblog_c  | (node:25) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the
'setupMiddlewares' option.
myblog_c  | Starting the development server...
myblog_c  |
myblog_c  | One of your dependencies, babel-preset-react-app, is importing the
myblog_c  | "@babel/plugin-proposal-private-property-in-object" package without
myblog_c  | declaring it in its dependencies. This is currently working because
myblog_c  | "@babel/plugin-proposal-private-property-in-object" is already in your
myblog_c  | node_modules folder for unrelated reasons, but it may break at any time.
myblog_c  |
myblog_c  | babel-preset-react-app is part of the create-react-app project, which
myblog_c  | is not maintianed anymore. It is thus unlikely that this bug will
myblog_c  | ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
myblog_c  | your devDependencies to work around this error. This will make this message
myblog_c  | go away.
myblog_c  |
myblog_c  | Compiled successfully!
myblog_c  |
myblog_c  | You can now view myblog in the browser.
myblog_c  |
myblog_c  |   Local:            http://localhost:3000
myblog_c  |   On Your Network:  http://172.18.0.2:3000
myblog_c  |
myblog_c  | Note that the development build is not optimized.
myblog_c  | To create a production build, use npm run build.
myblog_c  |
myblog_c  | asset static/js/bundle.js 1.49 MiB [emitted] (name: main) 1 related asset
myblog_c  | asset index.html 1.67 KiB [emitted]
myblog_c  | asset asset-manifest.json 190 bytes [emitted]
myblog_c  | runtime modules 28.4 KiB 14 modules
myblog_c  | modules by path ./node_modules/ 1.36 MiB 105 modules
myblog_c  | modules by path ./src/ 12.4 KiB
myblog_c  |   modules by path ./src/*.css 8.76 KiB
myblog_c  |     ./src/index.css 2.72 KiB [built] [code generated]
myblog_c  |     ./node_modules/css-loader/dist/cjs.js??ruleSet[1].rules[1].oneOf[5].use[1]!./node_modules/postcss-
loader/dist/cjs.js??ruleSet[1].rules[1].oneOf[5].use[2]!./node_modules/source-map-loader/dist/cjs.js!./src/index.css 1.36 KiB [built] [code generated]
myblog_c  |     ./src/App.css 2.71 KiB [built] [code generated]
myblog_c  |     ./node_modules/css-loader/dist/cjs.js??ruleSet[1].rules[1].oneOf[5].use[1]!./node_modules/postcss-
loader/dist/cjs.js??ruleSet[1].rules[1].oneOf[5].use[2]!./node_modules/source-map-loader/dist/cjs.js!./src/App.css 1.97 KiB [built] [code generated]
myblog_c  |   modules by path ./src/*.js 3.6 KiB
myblog_c  |     ./src/index.js 1.45 KiB [built] [code generated]
myblog_c  |     ./src/App.js 2.15 KiB [built] [code generated]
myblog_c  | webpack 5.94.0 compiled successfully in 5506 ms
myblog_c  | Compiling...
myblog_c  | Compiled successfully!
myblog_c  | assets by status 1.67 KiB [cached] 1 asset
myblog_c  | assets by chunk 1.49 MiB (name: main)
myblog_c  |   asset static/js/bundle.js 1.49 MiB [emitted] (name: main) 1 related asset
myblog_c  |   asset main.b486f9fe41e334ad591a.hot-update.js 357 bytes [emitted] [immutable] [hmr] (name: main) 1 related asset
myblog_c  | assets by path *.json 343 bytes
myblog_c  |   asset asset-manifest.json 315 bytes [emitted]
myblog_c  |   asset main.b486f9fe41e334ad591a.hot-update.json 28 bytes [emitted] [immutable] [hmr]
myblog_c  | Entrypoint main 1.49 MiB (1.57 MiB) = static/js/bundle.js 1.49 MiB main.b486f9fe41e334ad591a.hot-update.js 357 bytes 2 auxiliary assets
myblog_c  | cached modules 1.37 MiB [cached] 111 modules
myblog_c  | runtime modules 28.4 KiB 14 modules
myblog_c  | webpack 5.94.0 compiled successfully in 131 ms


v View in Docker Desktop   o View Config   w Enable Watch
```

Verify React App in browser

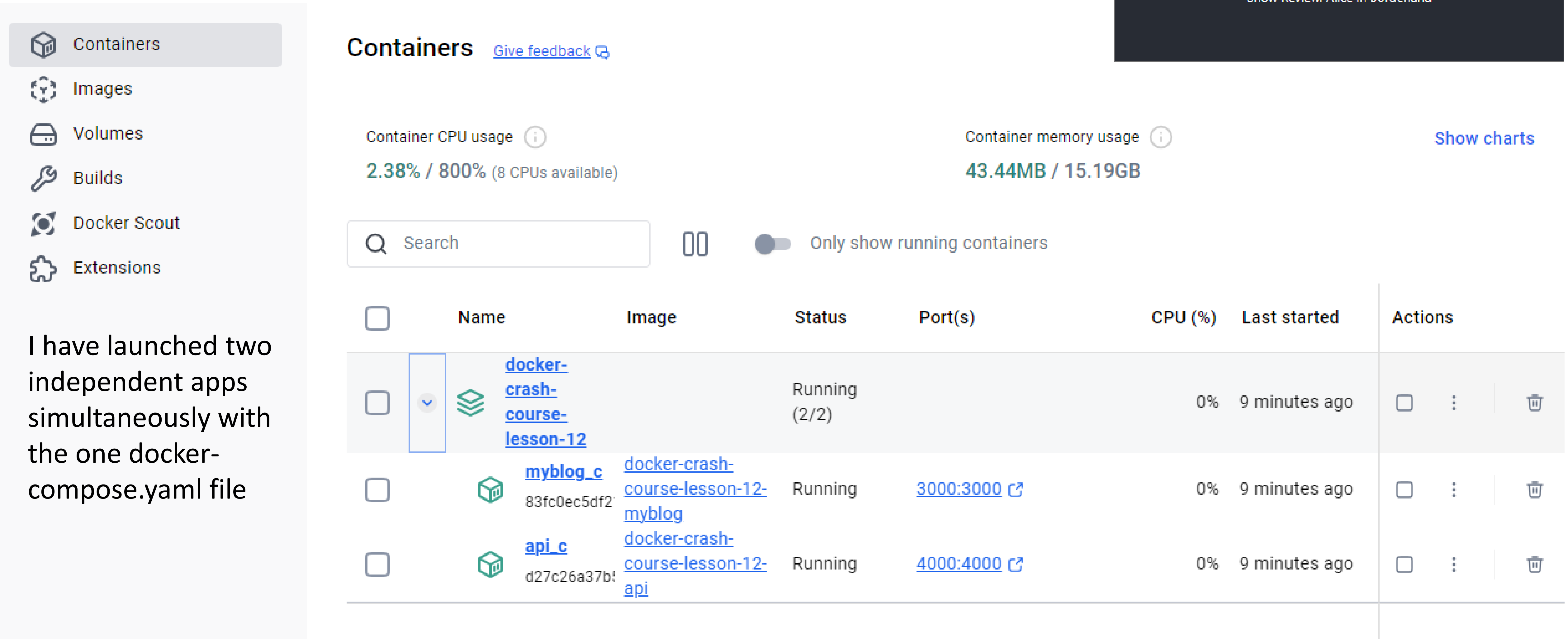
My blogs app loads in the browser and is succefully calling the api app to get the data.

The interesting thing is that the api app is running in its own container and the myblogs app is also running in its own container.



all blogs

Book Review: The Name of the Wind
Game Review: Pokemon Brillian Diamond
Show Review: Alice in Borderland



Containers

Container CPU usage: 2.38% / 800% (8 CPUs available)

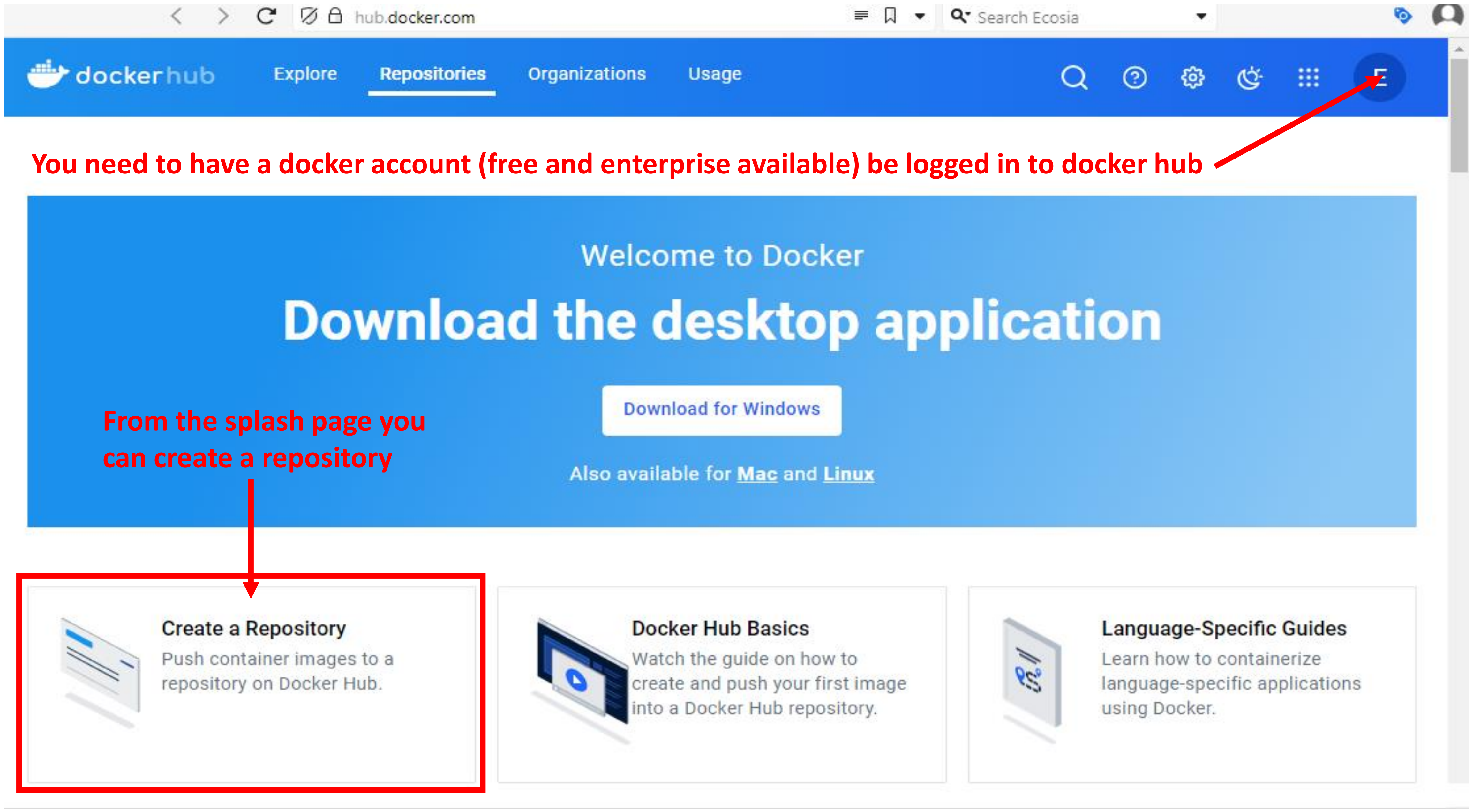
Container memory usage: 43.44MB / 15.19GB

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
docker-crash-course-lesson-12		Running (2/2)		0%	9 minutes ago	
myblog_c 83fc0ec5df2	docker-crash-course-lesson-12-myblog	Running	3000:3000	0%	9 minutes ago	
api_c d27c26a37b	docker-crash-course-lesson-12-api	Running	4000:4000	0%	9 minutes ago	

I have launched two independent apps simultaneously with the one docker-compose.yaml file

#12 Sharing Images on DockerHub

Requirements for sharing images on Docker Hub



You need to have a docker account (free and enterprise available) be logged in to docker hub

Welcome to Docker

Download the desktop application

Download for Windows

Also available for Mac and Linux

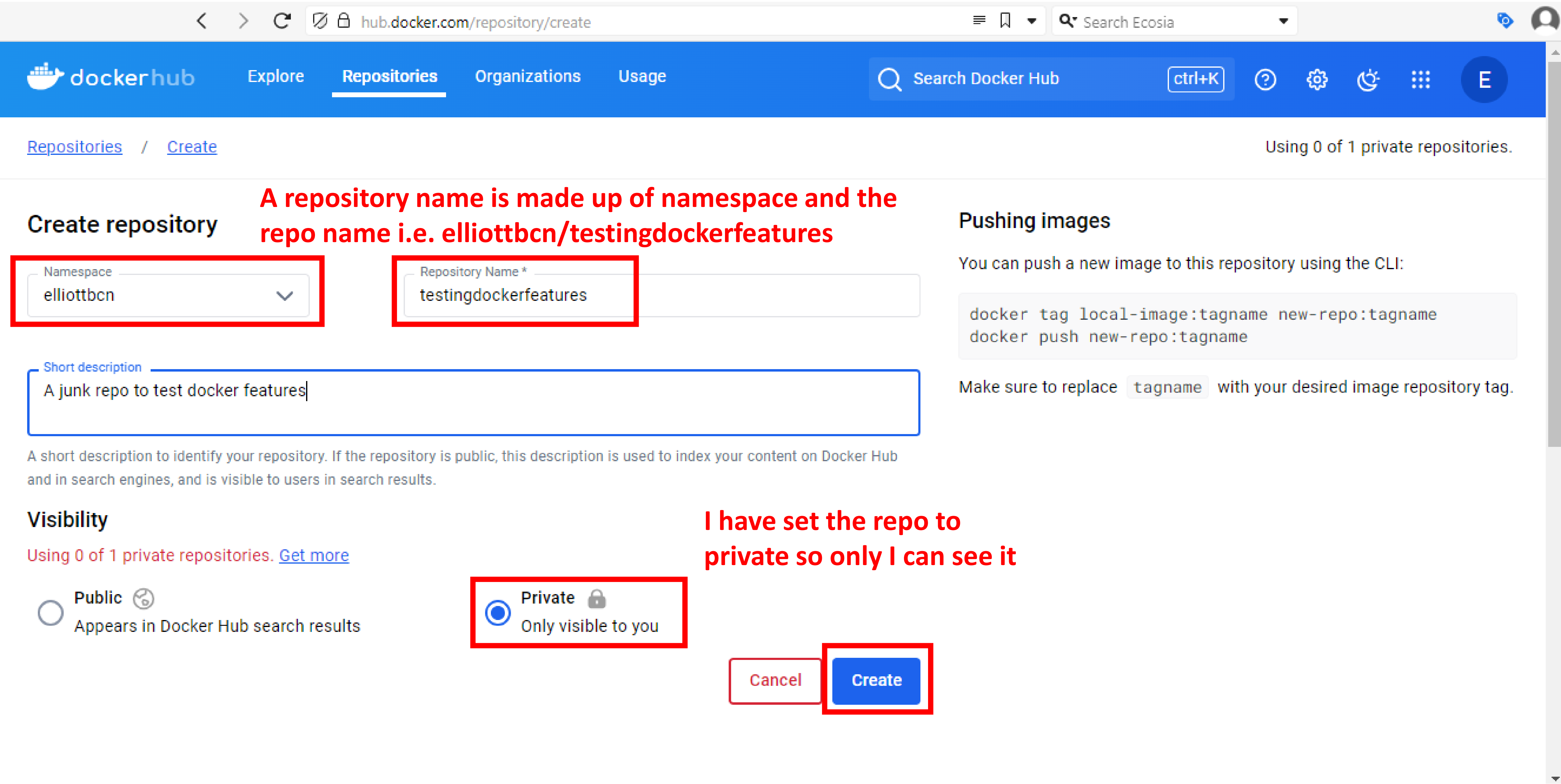
From the splash page you can create a repository

Create a Repository
Push container images to a repository on Docker Hub.

Docker Hub Basics
Watch the guide on how to create and push your first image into a Docker Hub repository.

Language-Specific Guides
Learn how to containerize language-specific applications using Docker.

Create a repository



hub.docker.com/repository/create

Repositories / Create

Using 0 of 1 private repositories.

A repository name is made up of namespace and the repo name i.e. elliottbcn/testingdockerfeatures

Create repository

Namespace: elliottbcn

Repository Name *: testingdockerfeatures

Short description: A junk repo to test docker features

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search results, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. Get more

I have set the repo to private so only I can see it

[ ] Public — Appears in Docker Hub search results

[✓ Private — Only visible to you]

Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Build an image to upload to Docker Hub

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12
$ cd api

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker build -t elliottbcn/testingdockerfeatures .
[+] Building 1.5s (12/12) FINISHED                                                        docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                      0.0s
 => => transferring dockerfile: 231B                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:17-alpine                                         1.1s
 => [auth] library/node:pull token for registry-1.docker.io                                               0.0s
 => [internal] load .dockerignore                                                                         0.0s
 => => transferring context: 52B                                                                          0.0s
 => [1/6] FROM docker.io/library/node:17-alpine@sha256:76e638eb0d73ac5f0b76d70df3ce1ddad941ac63595d440  0.0s
 => [internal] load build context                                                                         0.0s
 => => transferring context: 160B                                                                         0.0s
 => CACHED [2/6] RUN npm install -g nodemon                                                               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                                                                                    0.1s
 => => exporting layers                                                                                   0.0s
 => => writing image sha256:dc6ef9b35d468a6055e22c104f2bcca448dd482437775d7f05bf924c360512a3             0.0s
 => => naming to docker.io/elliottbcn/testingdockerfeatures                                               0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/a3hiys3x2vou41930yxzgrnd9

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
```

Curently I have no images

I need to be in the folder where the project files are and as I want to upload an image for the api app I will change directory.

I build the image and tag it (-t) with a name. This name needs to be specific and made up of docker username / docker repository. The Dot at the end signifies that we will take files from this folder.

Push an Image to Docker Hub

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
elliottbcn/testingdockerfeatures    latest    dc6ef9b35d46   19 hours ago     177MB
```

Now I have an image

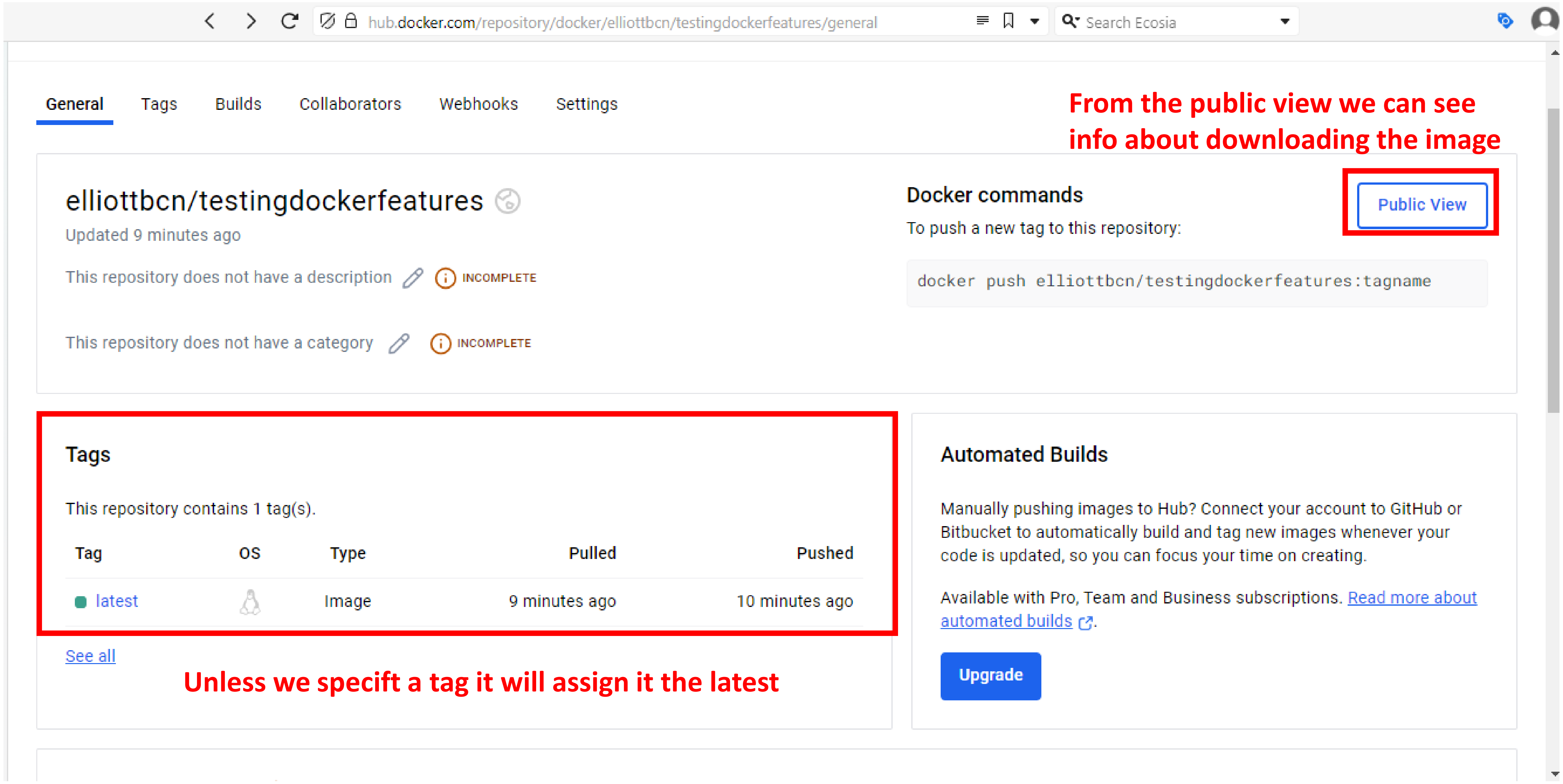
```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker login
Authenticating with existing credentials...
Login Succeeded
```

I need to be logged in to docker locally in this terminal

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker push elliottbcn/testingdockerfeatures
Using default tag: latest
The push refers to repository [docker.io/elliottbcn/testingdockerfeatures]
fdf6799fedfe: Pushed
583180e2c9bc: Pushed
b904eb851b15: Pushed
ca45c02cfc7d: Pushed
72efedfc22f4: Pushed
e6a74996eabe: Mounted from library/node
db2e1fd51a80: Mounted from library/node
19ebba8d6369: Mounted from library/node
4fc242d58285: Mounted from library/node
latest: digest: sha256:d4a6376f26da894c11ffbf282f1ef0a03fa3d9607b709738678f68fa804b1923 size: 2201

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
```

Now I can push the image using the specific name of my dockerusername / repo name

Verify Image is pushed to Docker Hub


hub.docker.com/repository/docker/elliottbcn/testingdockerfeatures/general

General Tags Builds Collaborators Webhooks Settings

From the public view we can see info about downloading the image

elliottbcn/testingdockerfeatures

Updated 9 minutes ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

Docker commands

To push a new tag to this repository:

```
docker push elliottbcn/testingdockerfeatures:tagname
```

Public View

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
🟩 latest		Image	9 minutes ago	10 minutes ago

See all

Unless we specift a tag it will assign it the latest

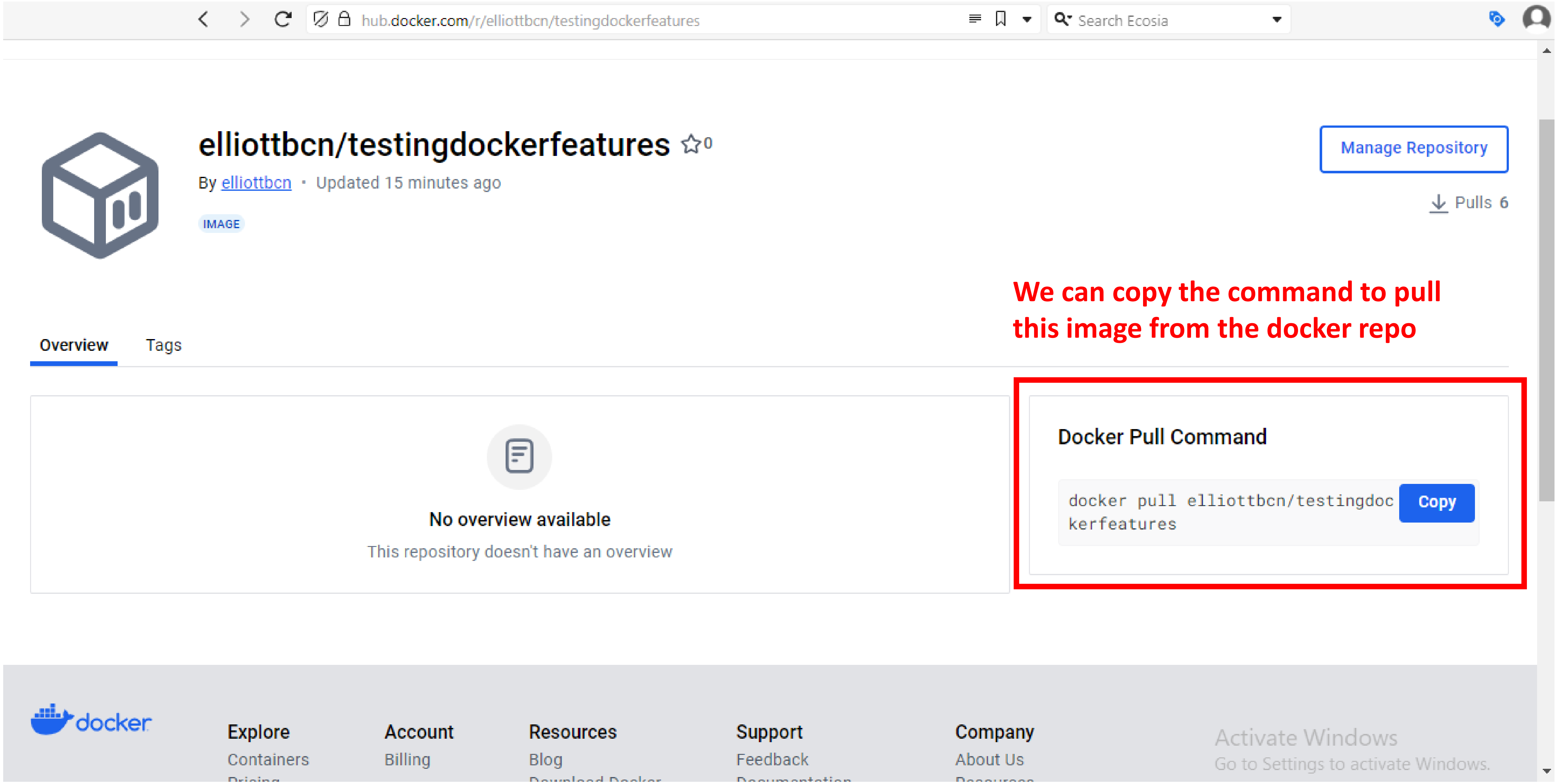
Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Upgrade

Find pull Command in repo



hub.docker.com/r/elliottbcn/testingdockerfeatures

elliottbcn/testingdockerfeatures ☆0

By elliottbcn · Updated 15 minutes ago

IMAGE

Manage Repository

Pulls 6

We can copy the command to pull this image from the docker repo

Overview Tags

No overview available

This repository doesn't have an overview

Docker Pull Command

```
docker pull elliottbcn/testingdockerfeatures
```

Copy

docker

Explore
Containers

Account
Billing

Resources
Blog

Support
Feedback

Company
About Us

Verify Pull Repo Image

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker images
REPOSITORY                        TAG       IMAGE ID       CREATED         SIZE
elliottbcn/testingdockerfeatures  latest    dc6ef9b35d46   19 hours ago    177MB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker image rm elliottbcn/testingdockerfeatures
Untagged: elliottbcn/testingdockerfeatures:latest
Untagged: elliottbcn/testingdockerfeatures@sha256:d4a6376f26da894c11ffbf282f1ef0a03fa3d9607b709738678f68fa804b1923
Deleted: sha256:dc6ef9b35d468a6055e22c104f2bcca448dd482437775d7f05bf924c360512a3

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

First I delete the image on my local machine and verify that it has been removed.

```
ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker pull elliottbcn/testingdockerfeatures
Using default tag: latest
latest: Pulling from elliottbcn/testingdockerfeatures
df9b9388f04a: Already exists
....
0f5e8a6bba28: Already exists
Digest: sha256:d4a6376f26da894c11ffbf282f1ef0a03fa3d9607b709738678f68fa804b1923
Status: Downloaded newer image for elliottbcn/testingdockerfeatures:latest
docker.io/elliottbcn/testingdockerfeatures:latest

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
$ docker images
REPOSITORY                        TAG       IMAGE ID       CREATED         SIZE
elliottbcn/testingdockerfeatures  latest    dc6ef9b35d46   19 hours ago    177MB

ellio@DESKTOP-U93252R MINGW64 ~/Documents/CODING-LESSONS/12-Docker/docker-crash-course-lesson-12/api
```

Now I can pull the image and if I check again I see that the image has been downloaded from docker hub onto my local docker machine.