

Aim: Explore Docker commands for content management. develop a single containerized application using docker

Objective:

Understand and utilize fundamental Docker commands for managing containers and images.

Build and deploy a containerized application to demonstrate Docker's efficiency in content management.

Gain hands-on experience in using Docker for application packaging, testing, and running in isolated environments.

Description:

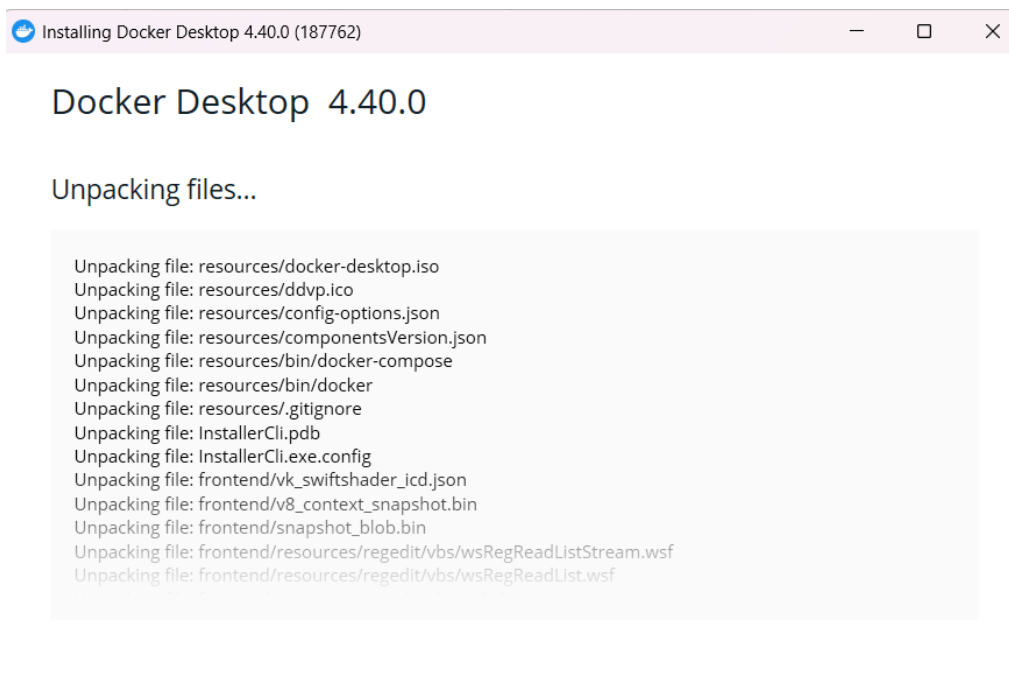
Docker is a platform designed to make it easier to create, deploy, and run applications using containers. Containers allow developers to package applications with all parts they need, such as libraries and other dependencies, and ship it all out as one package. This project involves building a simple web application and running it inside a Docker container while exploring key Docker commands that help manage container content and configuration.

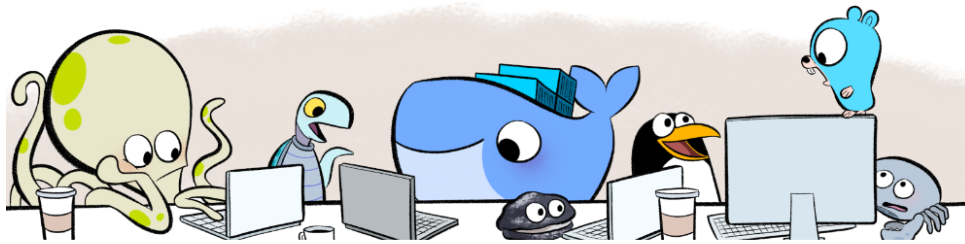
Steps Required:

Setup and Requirements:

- Docker installed on the system - <https://docs.docker.com/desktop/setup/install/windows-install/>
- Basic knowledge of Dockerfile and image building
- A simple web application (e.g., a static HTML page or a Node.js app)

Implementation:





Docker Subscription Service Agreement

By selecting **accept**, you agree to the [Subscription Service Agreement](#), the [Docker Data Processing Agreement](#), and the [Data Privacy Policy](#).

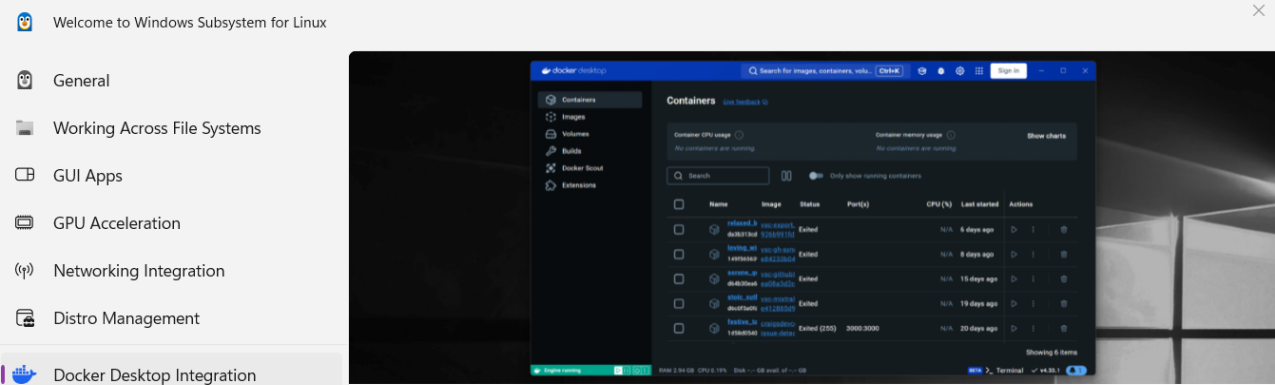
Commercial use of Docker Desktop at a company of more than 250 employees OR more than \$10 million in annual revenue requires a paid subscription (Pro, Team, or Business). [See subscription details](#)

[View Full Terms](#)[Accept](#)[Close](#)

```
and 'wsl.exe --install <Distro>' to install.
PS C:\WINDOWS\system32> wsl --list --online
>>
The following is a list of valid distributions that can be installed.
Install using 'wsl.exe --install <Distro>'.
```

NAME	FRIENDLY NAME
AlmaLinux-8	AlmaLinux OS 8
AlmaLinux-9	AlmaLinux OS 9
AlmaLinux-Kitten-10	AlmaLinux OS Kitten 10
Debian	Debian GNU/Linux
FedoraLinux-42	Fedora Linux 42
SUSE-Linux-Enterprise-15-SP5	SUSE Linux Enterprise 15 SP5
SUSE-Linux-Enterprise-15-SP6	SUSE Linux Enterprise 15 SP6
Ubuntu	Ubuntu
Ubuntu-24.04	Ubuntu 24.04 LTS
archlinux	Arch Linux
kali-linux	Kali Linux Rolling
openSUSE-Tumbleweed	openSUSE Tumbleweed
openSUSE-Leap-15.6	openSUSE Leap 15.6
Ubuntu-18.04	Ubuntu 18.04 LTS
Ubuntu-20.04	Ubuntu 20.04 LTS
Ubuntu-22.04	Ubuntu 22.04 LTS
OracleLinux_7_9	Oracle Linux 7.9
OracleLinux_8_7	Oracle Linux 8.7
OracleLinux_9_1	Oracle Linux 9.1

```
PS C:\WINDOWS\system32> wsl --install -d Ubuntu
>>
Downloading: Ubuntu
[===== 14.6% ]
```



The screenshot shows the Docker Desktop application window. On the left is a sidebar with navigation options: General, Working Across File Systems, GUI Apps, GPU Acceleration, Networking Integration, Distro Management, Docker Desktop Integration (highlighted), VS Code Integration, and Settings. The main area displays the 'Containers' tab, which lists several containers. The containers listed are:

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
ubuntu-24.04	ubuntu:24.04	Exited		N/A	6 days ago	Stop, Restart, Logs
ubuntu-24.04	ubuntu:24.04	Exited		N/A	8 days ago	Stop, Restart, Logs
ubuntu-24.04	ubuntu:24.04	Exited		N/A	15 days ago	Stop, Restart, Logs
ubuntu-24.04	ubuntu:24.04	Exited		N/A	19 days ago	Stop, Restart, Logs
ubuntu-24.04	ubuntu:24.04	Exited (255)	3000-3000	N/A	20 days ago	Stop, Restart, Logs

Below the table, it says 'Showing 6 items'.

Docker Desktop Integration

Docker Desktop works great with WSL to help you develop with Linux containers.

Some of the benefits of using Docker Desktop with WSL are:

- You can run Docker commands in WSL or in Windows, using the same Docker daemon and images.
- You can share files and folders between Windows and Linux seamlessly, using the automatic mount of Windows drives in WSL.
- You can use your preferred Windows tools and editors to work on Linux code and files, and vice versa, thanks to the interoperability of WSL.

[Learn More About Using WSL with Docker](#)

```
PS C:\WINDOWS\system32> wsl -d Ubuntu
>>
Provisioning the new WSL instance Ubuntu
This might take a while...
Create a default Unix user account: geeta
New password:
Retype new password:
passwd: password updated successfully
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

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

System information as of Sun Apr 20 07:14:58 UTC 2025

System load:  0.06          Processes:            32
Usage of /:   0.1% of 1006.85GB  Users logged in:     0
Memory usage: 5%            IPv4 address for eth0: 172.17.247.16
Swap usage:   0%

This message is shown once a day. To disable it please create the
/home/geeta/.hushlogin file.
geeta@Geeta: /mnt/c/WINDOWS/system32$
```

```
C:\Users\geeta>docker --version
Docker version 28.0.4, build b8034c0

C:\Users\geeta>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:c41088499908a59aae84b0a49c70e86f4731e588a737f1637e73c8c09d995654
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

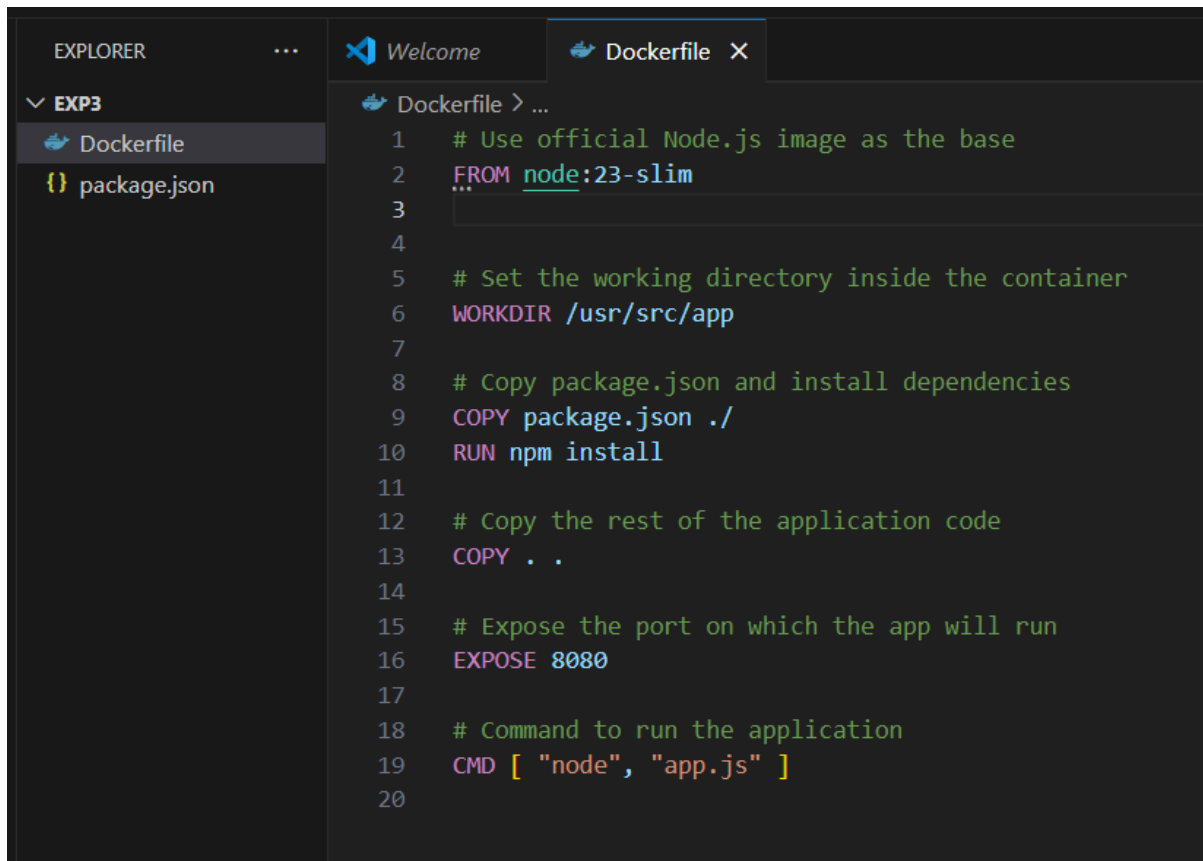
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

```
● PS C:\Users\geeta\OneDrive\Desktop\exp3> npm init -y
Wrote to C:\Users\geeta\OneDrive\Desktop\exp3\package.json:

{
  "name": "exp3",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "description": ""
}
```



```
EXPLORER
EXP3
  Dockerfile
  package.json

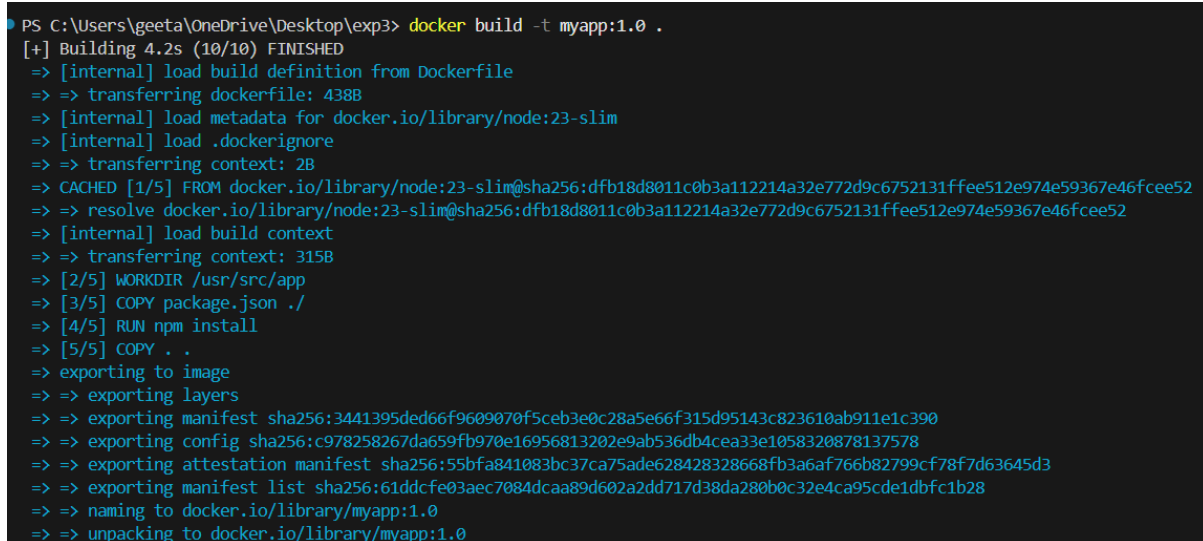
Dockerfile
1  # Use official Node.js image as the base
2  FROM node:23-slim
3
4
5  # Set the working directory inside the container
6  WORKDIR /usr/src/app
7
8  # Copy package.json and install dependencies
9  COPY package.json ./
10 RUN npm install
11
12 # Copy the rest of the application code
13 COPY . .
14
15 # Expose the port on which the app will run
16 EXPOSE 8080
17
18 # Command to run the application
19 CMD [ "node", "app.js" ]
20
```

Part 1: Docker Commands for Content Management

Docker provides several useful commands for managing containers and images related to content management. Below are some essential Docker commands:

1. Build an Image from a Dockerfile

- **Command:** `docker build -t <image_name>:<tag> <path_to_dockerfile>`
- **Example:** `docker build -t myapp:1.0.`



```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker build -t myapp:1.0 .
[+] Building 4.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 438B
=> [internal] load metadata for docker.io/library/node:23-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/5] FROM docker.io/library/node:23-slim@sha256:dfb18d8011c0b3a112214a32e772d9c6752131ffee512e974e59367e46fcee52
=> => resolve docker.io/library/node:23-slim@sha256:dfb18d8011c0b3a112214a32e772d9c6752131ffee512e974e59367e46fcee52
=> [internal] load build context
=> => transferring context: 315B
=> [2/5] WORKDIR /usr/src/app
=> [3/5] COPY package.json ./
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:3441395ded66f9609070f5ceb3e0c28a5e66f315d95143c823610ab911e1c390
=> => exporting config sha256:c978258267da659fb970e16956813202e9ab536db4cea33e1058320878137578
=> => exporting attestation manifest sha256:55bfa841083bc37ca75ade628428328668fb3a6af766b82799cf78f7d63645d3
=> => exporting manifest list sha256:61ddcf03aec7084dcaa89d602a2dd717d38da280b0c32e4ca95cde1dbfc1b28
=> => naming to docker.io/library/myapp:1.0
=> => unpacking to docker.io/library/myapp:1.0
```

This command builds an image from the Dockerfile located in the current directory (.)

List Docker Images

- **Command:** `docker images`
- **Example:** `docker images`

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
myapp                1.0            61ddcfe03aec   44 seconds ago 332MB
node-redis-api-app   latest         3ef30ad3329e   21 minutes ago 364MB
hello-world          latest         c41088499908   2 months ago  20.4kB
redis                latest         fdbbaea47b9a   3 months ago  173MB
```

Run a Container from an Image

- **Command:** `docker run -d --name <container_name> <image_name>:<tag>`
- **Example:** `docker run -d --name mycontainer myapp:1.0`

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker run -d --name mycontainer myapp:1.0
a8ffdd9b4a3cf928358b29583a641d2dd06e717bcbd5994fc9e284b44ec38c73
```

This runs the container in detached mode (-d) from the specified image.

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ Container memory usage ⓘ [Show charts](#)

No containers are running.

Search Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	○ mycontainer	a8ffdd9b4a3c	myapp:1.0		N/A	52 seconds ago	

[Images](#) / [myapp:1.0](#)

[<](#) **myapp:1.0** IN USE
61ddcfe03aec

CREATED
28 minutes ago

SIZE
331.79 MB

Layers (15)

Vulnerabilities

Packages

0	# debian.sh --arch 'amd64' out/ 'bookworm' '@1743984000'	85.25 MB
1	RUN /bin/sh -c groupadd --gid 1000 node && useradd --ui...	69.63 KB
2	ENV NODE_VERSION=23.11.0	0 B
3	RUN /bin/sh -c ARCH= OPENSSL_ARCH= && dpkgArch="\$...	155.43 MB
4	ENV YARN_VERSION=1.22.22	0 B
5	RUN /bin/sh -c set -ex && savedAptMark="\$(apt-mark sho...	7.32 MB
6	COPY docker-entrypoint.sh /usr/local/bin/ # buildkit	20.48 KB
7	ENTRYPOINT ["docker-entrypoint.sh"]	0 B
8	CMD ["node"]	0 B
9	WORKDIR /usr/src/app	16.38 KB
10	COPY package.json ./ # buildkit	20.48 KB

This

You can use Docli

[Enable background](#)

11	RUN /bin/sh -c npm install # buildkit	3.04 MB
12	COPY . . # buildkit	20.48 KB
13	EXPOSE map[8080/tcp:{}]	0 B
14	CMD ["node" "app.js"]	0 B

List Running Containers

- **Command:** *docker ps*
- **Example:** *docker ps*

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker ps
=> CACHED [1/5] FROM docker.io/library/node:23-slim@sha256:dfb18d8011c0b3a112214a32e772d9c6752131ffee512e974e59367e46fcee52
=> => resolve docker.io/library/node:23-slim@sha256:dfb18d8011c0b3a112214a32e772d9c6752131ffee512e974e59367e46fcee52
=> [internal] load build context
=> => transferring context: 315B
=> [2/5] WORKDIR /usr/src/app
=> [3/5] COPY package.json ./
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:3441395ded66f9609070f5ceb3e0c28a5e66f315d95143c823610ab911e1c390
=> => exporting config sha256:c978258267da659fb970e16956813202e9ab536db4cea33e1058320878137578
=> => exporting attestation manifest sha256:55bfa841083bc37ca75ade628428328668fb3a6af766b82799cf78f7d63645d3
=> => exporting manifest list sha256:61ddcfe03aec7084dcaa89d602a2dd717d38da280b0c32e4ca95cde1dbfc1b28
=> => naming to docker.io/library/myapp:1.0
=> => unpacking to docker.io/library/myapp:1.0
```

This shows a list of all running containers.

Stop a Running Container

- **Command:** *docker stop <container_name>*
- **Example:** *docker stop mycontainer*

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker stop mycontainer
mycontainer
PS C:\Users\geeta\OneDrive\Desktop\exp3>
```

Remove a Container

- **Command:** *docker rm <container_name>*
- **Example:** *docker rm mycontainer*

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker rm mycontainer
mycontainer\geeta\OneDrive\Desktop\exp3>
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker rm mycontainer
Error response from daemon: No such container: mycontainer
PS C:\Users\geeta\OneDrive\Desktop\exp3>
```

Containers [Give feedback](#)View all your running containers and applications. [Learn more](#)**Your running containers show up here**

A container is an isolated environment for your code

Remove an Image

- **Command:** `docker rmi <image_name>:<tag>`
- **Example:** `docker rmi myapp:1.0`

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker rmi myapp:1.0
Untagged: myapp:1.0
Deleted: sha256:61ddcfe03aec7084dcaa89d602a2dd717d38da280b0c32e4ca95cde1dbfc1b28
PS C:\Users\geeta\OneDrive\Desktop\exp3>
```

View Logs of a Running Container

- **Command:** `docker logs <container_name>`
- **Example:** `docker logs mycontainer`

```
PS C:\Users\geeta\OneDrive\Desktop\exp3> docker logs mycontainer
Error response from daemon: No such container: mycontainer
PS C:\Users\geeta\OneDrive\Desktop\exp3>
```

Part 2: Developing a Simple Containerized Application

Let's create a simple Node.js web application that serves a "Hello, Docker!" message. We will containerize this application using Docker.

Step 1: Create the Application

1. **Create a new directory for your application.**

```
PS C:\Users\geeta\OneDrive\Desktop> mkdir my-docker-app

Directory: C:\Users\geeta\OneDrive\Desktop

Mode                LastWriteTime         Length Name
----                -
d-----          20-04-2025         14:21           my-docker-app
```


Create the Node.js application:

- Create a file called app.js with the following content:

```
JS app.js > ...
1  const http = require('http');
2  const port = 8080;
3
4  const requestHandler = (req, res) => {
5    res.write('Hello, Docker!');
6    res.end();
7  };
8
9  const server = http.createServer(requestHandler);
10
11 server.listen(port, () => {
12   console.log(`Server is running on http://localhost:${port}`);
13 });
14
```

Create a package.json file:

- Run npm init -y to generate a basic package.json file, and then run: npm install http

```
PS C:\Users\geeta\OneDrive\Desktop\my-docker-app> npm init -y
Wrote to C:\Users\geeta\OneDrive\Desktop\my-docker-app\package.json:

{
  "name": "my-docker-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "description": ""
}
```

```
npm install http
>> C:\Users\geeta\OneDrive\Desktop\my-docker-app>
added 1 package, and audited 2 packages in 1s
found 0 vulnerabilities
PS C:\Users\geeta\OneDrive\Desktop\my-docker-app>
```

Step 2: Create the Dockerfile

1. Create a Dockerfile in the same directory with the following content:

```
{ } package-lock.json Dockerfile X JS app.js
Dockerfile > ...
1  # Use official Node.js image as the base
2  FROM node:23-slim
3
4
5  # Set the working directory inside the container
6  WORKDIR /usr/src/app
7
8  # Copy package.json and install dependencies
9  COPY package.json ./
10 RUN npm install
11
12 # Copy the rest of the application code
13 COPY . .
14
15 # Expose the port on which the app will run
16 EXPOSE 8080
17
18 # Command to run the application
19 CMD [ "node", "app.js" ]
20
```

2. App.js

```
package-lock.json Dockerfile JS app.js X
JS app.js > ...
1  const http = require('http');
2  const port = 8080;
3
4  const requestHandler = (req, res) => {
5    res.write('Hello, Docker!');
6    res.end();
7  };
8
9  const server = http.createServer(requestHandler);
10
11 server.listen(port, () => {
12   console.log(`Server is running on http://localhost:${port}`);
13 });
14
```

Step 3: Build the Docker Image

1. Build the Docker image using the command: `docker build -t mynodeapp:1.0 .`

```
PS C:\Users\geeta\OneDrive\Desktop\my-docker-app> docker build -t mynodeapp:1.0 .
[+] Building 164.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 431B
=> [internal] load metadata for docker.io/library/node:14
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> => sha256:0d27a8e861329007574c6766fba946d48e20d2c8e964e873de352603f22c4ceb 450B / 450B
=> => sha256:0c8cc2f24a4dcb64e602e086fc9446b0a541e8acd9ad72d2e90df3ba22f158b3 2.29MB / 2.29MB
=> => sha256:5f32ed3c3f278edda4fc571c880b5277355a29ae8f52b52cdf865f058378a590 35.24MB / 35.24MB
=> => sha256:6f51ee005deac0d99898e41b8ce60ebf250ebe1a31a0b03f613aec6bbc9b83d8 4.19kB / 4.19kB
=> => sha256:d9a8df5894511ce28a05e2925a75e8a4acbd0634c39ad734dfdba8e23d1b1569 191.85MB / 191.85MB
=> => sha256:1de76e268b103d05fa8960e0f77951ff54b912b63429c34f5d6adfd09f5f9ee2 51.88MB / 51.88MB
=> => sha256:3d2201bd995cccf12851a50820de03d34a17011dcb9ac9fd3a50c952cbb131 10.00MB / 10.00MB
=> => sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3b0fbfc7c05cca5 7.86MB / 7.86MB
=> => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca845ebc747976c408 50.45MB / 50.45MB
=> => extracting sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3b0fbfc7c05cca5
=> => extracting sha256:3d2201bd995cccf12851a50820de03d34a17011dcb9ac9fd3a50c952cbb131
=> => extracting sha256:1de76e268b103d05fa8960e0f77951ff54b912b63429c34f5d6adfd09f5f9ee2
=> => extracting sha256:d9a8df5894511ce28a05e2925a75e8a4acbd0634c39ad734dfdba8e23d1b1569
=> => extracting sha256:6f51ee005deac0d99898e41b8ce60ebf250ebe1a31a0b03f613aec6bbc9b83d8
=> => extracting sha256:5f32ed3c3f278edda4fc571c880b5277355a29ae8f52b52cdf865f058378a590
=> => extracting sha256:0c8cc2f24a4dcb64e602e086fc9446b0a541e8acd9ad72d2e90df3ba22f158b3
=> => extracting sha256:0d27a8e861329007574c6766fba946d48e20d2c8e964e873de352603f22c4ceb
=> [internal] load build context
=> => transferring context: 2.47kB
=> [auth] library/node:pull token for registry-1.docker.io
=> [2/5] WORKDIR /usr/src/app
=> [3/5] COPY package.json ./
```

Step 4: Run the Docker Container

1. Run the container: `docker run -d -p 8081:8080 --name mynodeapp-container mynodeapp:1.0`

```
PS C:\Users\geeta\OneDrive\Desktop\my-docker-app> docker run -d -p 8081:8080 --name mynodeapp-container mynodeapp:1.0
b8942dd602ef929d9bc547720ff42af7ee1ff2a4635671c0397c090d3e233a93
PS C:\Users\geeta\OneDrive\Desktop\my-docker-app>
```

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ

No containers are running.

Container memory usage ⓘ

No containers are running.

[Show charts](#)

Search



Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	mynodeapp-container	b8942dd602ef	mynodeapp:1.0	8081:8080	N/A	5 minutes ago	▶ ⋮ 🗑️

[Images](#) / mynodeapp:1.0

[<](#) **mynodeapp:1.0** IN USE
f0b7b86abe3c [🔗](#)

CREATED
8 minutes ago

SIZE
1.34 GB

[Recommended fixes](#) ▾

[Run](#) ▾



Layers (20)

0	ADD file:40953ed6e6f96703b2e0c13288437c2aaf8b3df3...	130.11 MB
1	CMD ["bash"]	0 B
2	set -eux; apt-get update; apt-get install -y --no-install-reco...	18.92 MB
3	set -ex; if ! command -v gpg > /dev/null; then apt-get upda...	18.78 MB
4	apt-get update && apt-get install -y --no-install-recommen...	160.46 MB
5	set -ex; apt-get update; apt-get install -y --no-install-recom...	543.22 MB
6	groupadd --gid 1000 node && useradd --uid 1000 --gid nod...	405.5 KB
7	ENV NODE_VERSION=14.21.3	0 B
8	ARCH= && dpkgArch="\$(dpkg --print-architecture)" && cas...	117.58 MB
9	ENV YARN_VERSION=1.22.19	0 B

Vulnerabilities

Packages

Analyzed by **docker scout**

[Give feedback](#)



This image has not been analyzed

You can use Docker Scout to analyze local images and list its vulnerabilities.

[Start analysis](#)

[Enable background indexing in Settings](#) so your results are always ready.

RAM 3.19 GB CPU 0.00% Disk: 2.84 GB used (limit 1006.85 GB)

[Terminal](#) ✓ v4.40.0

⌵

Docker Content Management A x

localhost:8081 x

Semester 2 - Google Drive x

+

← → ↺

localhost:8081 ⓘ

🗑️

V26

Profile

2025

DQ

S2

Apply

J

Hello, Docker!

The screenshot displays the Docker Desktop interface. On the left, the 'Containers' tab shows a list of running containers: 'elastic_yonath' and 'bold_pascal'. The 'Resource usage' panel on the right shows 'Container CPU usage' at 0.00% / 1200% and 'Container memory usage' at 8.16MB / 7.45GB. Below this, the 'Images' tab shows the 'mynodeapp:1.0' image. The 'Layers' section lists the image's layers, including 'node:23-bookworm-slim' and 'mynodeapp:1.0'. The 'Vulnerabilities' section shows 24 vulnerabilities, with a list of CVEs and their severities. The 'Packages' section shows 332 packages. The bottom status bar indicates RAM usage at 1.51 GB and CPU usage at 0.00%.

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ 0.00% / 1200% (12 CPUs available) Container memory usage ⓘ 8.16MB / 7.45GB

Search Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	Actions
<input type="checkbox"/>	elastic_yonath	9893804698cc	mynodeapp	8080:8080	Play Logs Stop
<input type="checkbox"/>	bold_pascal	b64d18a81900	mynodeapp	8081:8080	Play Logs Stop

Showing 2 items

RAM 1.40 GB CPU 0.00% Disk: 2.84 GB used (limit 1006.85 GB) v4.40.0

Resource usage ⓘ

Container CPU usage ⓘ 0.00% / 1200% 12 CPUs available

Container memory usage ⓘ 8.16MB / 7.45GB

Images / mynodeapp:1.0

mynodeapp:1.0 IN USE 24ff584bb19b

CREATED 3 minutes ago SIZE 331.92 MB Recommended fixes [Run](#) [Stop](#) [Delete](#)

Analyzed by **docker scout** [Give feedback](#)

Layers (15)

Layer	Image	Size
9	WORKDIR /usr/src/app	16.38 KB
10	COPY package.json ./ # buildkit	20.48 KB
11	RUN /bin/sh -c npm install # buildkit	3.13 MB
12	COPY . # buildkit	49.15 KB
13	EXPOSE map[8080/tcp:{}]	0 B
14	CMD ["node" "app.js"]	0 B

Vulnerabilities (24) Packages (332) [Give feedback](#)

Package or CVE name Fixable ☐ Show excepted ☐ Reset filters

CVE ID	Severity	Fixable	Present in	Affected packages
.. CVE-2018-20796	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2019-1010022	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2019-1010023	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2010-4756	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2019-1010024	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2019-1010025	N/A L	<input type="checkbox"/>	deb / debian	
.. CVE-2019-9192	N/A L	<input type="checkbox"/>	deb / debian	

RAM 1.51 GB CPU 0.00% Disk: 2.84 GB used (limit 1006.85 GB) Terminal v4.40.0

Example 2: Docker Implementation on Mini Project

1. Create a Dockerfile

The Dockerfile is a text file that contains instructions for Docker to create an image. In this case, you'll use a lightweight web server (like Nginx) to serve your static files.

In the root of your Word Search project directory, create a file named Dockerfile (without any file extension).

Dockerfile:

```
# Use an official Nginx image from Docker Hub
FROM nginx:alpine
```

```
# Set the working directory inside the container
WORKDIR /usr/share/nginx/html

# Copy the contents of the local directory to the Nginx container directory
COPY . .

# Expose port 80 so that it can be accessed from the outside
EXPOSE 80

# Start the Nginx service when the container is run
CMD ["nginx", "-g", "daemon off;"]
```

- FROM nginx:alpine: This uses an official, minimal version of the Nginx web server.
- WORKDIR /usr/share/nginx/html: Sets the working directory inside the container to where Nginx serves files.
- COPY . .: Copies all files from your local directory (where the Dockerfile is located) into the working directory inside the container.
- EXPOSE 80: Exposes port 80 for the web server.
- CMD ["nginx", "-g", "daemon off;"]: Tells Docker to run Nginx in the foreground.

```
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .dockerignore
        Dockerfile

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> git add .
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> git commit -m "Docker Changes"
error: pathspec '-' did not match any file(s) known to git
error: pathspec 'm' did not match any file(s) known to git
error: pathspec 'Docker Changes' did not match any file(s) known to git
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> git commit -m "Docker Changes"
[main f2b65a7] Docker Changes
 2 files changed, 17 insertions(+)
 /github.com/geeta-seshapalli/Word-Search.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Description: Docker Files update in git

```
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> git pull origin main
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 12 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (12/12), 4.93 KiB | 85.00 KiB/s, done.
From https://github.com/geeta-seshapalli/Word-Search
 * branch          main      -> FETCH HEAD
   cf1817c..3f2afea main      -> origin/main
hint: Waiting for your editor to close the Merge made by the 'ort' strategy.
 Jenkinsfile | 62 ++++++
1 file changed, 62 insertions(+)
cregit commit -m "Docker Changes"
p\WordSearch>
```

Description: Docker Changes

```

1 file changed, 62 insertions(+)
commit 3f2afea..8e1d1fb main -> main
PS C:\Users\geeta\OneDrive\Desktop\WordSearch>

```

Description: Docker Changes

Build the Docker Image

To build the Docker image. Open your terminal, navigate to your project folder, and run the following command:

```

PS C:\Users\geeta\OneDrive\Desktop\WordSearch> docker build -t wordsearch-app .
[*] Building 8.7s (9/9) FINISHED
[+] Building 8.7s (9/9) FINISHED
-> [internal] load build definition from Dockerfile
-> [internal] load metadata for docker.io/library/nginx:alpine
-> [auth] library/nginx:pull token for registry-1.docker.io
-> [internal] load .dockerignore
-> transferring context: 678
-> [1/3] FROM docker.io/library/nginx:alpine@sha256:65645c7bb6a0661892a8b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10
-> resolve docker.io/library/nginx:alpine@sha256:65645c7bb6a0661892a8b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10
-> sha256:39c2ddfd6010082a4a646e7ca44e95aca9bf3eabc00f17f7ccc2954004f2a7d 15.52MB / 15.52MB
-> sha256:34a6464b756511a2e217f0508e11d1a572085d66cd6dc9a55a082ad49a3102 1.40kB / 1.40kB
-> sha256:197eb75867efafcecd4724f17b0972ab048943686a594a45f8eaff8155053 1.21kB / 1.21kB
-> sha256:81bd8ed7ec6789b0cb7f1b47ee731c522f6dba83201ec73cd6bca1350f582948 402B / 402B
-> sha256:d7e9070240863957eb0b5a44a5729963c346266bbaa2947d00628cb5f2d5773 955B / 955B
-> sha256:b464cdf2a6319875aeb27359ec549790ce14d8214fcb16ef915e4530e5ed235 629B / 629B
-> sha256:61ca4f73c802af9e05a32f0de0361bd713b0b53292dc15fb093229f648674 1.74MB / 1.74MB
-> sha256:f19232174bc91741f4f3da96d85011092101a032a293a388b79e99e69c2d5c870 3.64MB / 3.64MB
-> extracting sha256:f19232174bc91741f4f3da96d85011092101a032a293a388b79e99e69c2d5c870
-> extracting sha256:61ca4f73c802af9e05a32f0de0361bd713b0b53292dc15fb093229f648674
-> extracting sha256:b464cdf2a6319875aeb27359ec549790ce14d8214fcb16ef915e4530e5ed235
-> extracting sha256:d7e9070240863957eb0b5a44a5729963c346266bbaa2947d00628cb5f2d5773
-> extracting sha256:81bd8ed7ec6789b0cb7f1b47ee731c522f6dba83201ec73cd6bca1350f582948
-> extracting sha256:197eb75867efafcecd4724f17b0972ab048943686a594a45f8eaff8155053
-> extracting sha256:34a6464b756511a2e217f0508e11d1a572085d66cd6dc9a55a082ad49a3102
-> extracting sha256:39c2ddfd6010082a4a646e7ca44e95aca9bf3eabc00f17f7ccc2954004f2a7d
-> [internal] load build context
-> transferring context: 11.04kB
-> [2/3] WORKDIR /usr/share/nginx/html
-> [3/3] COPY
-> exporting to image
-> exporting layers
-> exporting manifest sha256:ca2911b7510b60965111c58fc958c542e57e4b03cfd4d045b8545541e848fd8d
-> exporting config sha256:a8e85a8be647c1386390f05a39574eff37298e542d4b68d97346d73a8a8108
-> exporting attestation manifest sha256:177c9617b10b0eca2268817cc65ad21545d85b63b0b7e38a16d1d4776b12e3f1
-> exporting manifest list sha256:908262d72f050ad71eff1451b6f25606a909ce54340ac138aaf040e752a

```

Description: Build the Docker Image

```

PS C:\Users\geeta\OneDrive\Desktop\WordSearch> docker run -d -p 8081:80 --name wordsearch-container wordsearch-app
75c8795ccf823e9ae0376f1f91eeda1ff42c93f7904ee275a4a0320890eb3c7
PS C:\Users\geeta\OneDrive\Desktop\WordSearch> docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
da25cb1b4d73	wordsearch-app:latest	"/docker-entrypoint..."	36 seconds ago	Up 36 seconds	80/tcp	flamboyant_raman
75c8795ccf82	wordsearch-app	"/docker-entrypoint..."	About a minute ago	Up About a minute	0.0.0.0:8081->80/tcp	wordsearch-container

```

PS C:\Users\geeta\OneDrive\Desktop\WordSearch>

```

- d: This runs the container in detached mode (in the background).
- p 8081:80: This maps port 80 on the container to port 8080 on your host system. You can change 8081 to any port of your choice.
- name wordsearch-container: This names your container wordsearch-container.
- wordsearch-app: This is the name of the image you created earlier.

The screenshot shows the Docker Scout interface for a Docker image named `wordsearch-app:latest`. The interface includes a top bar with the image name, a status indicator (IN USE), and buttons for 'Recommended fixes', 'Run', and a trash icon. Below the top bar, there's a section for 'Layers (23)' and 'Vulnerabilities (0)'. The 'Layers' section lists the following layers:

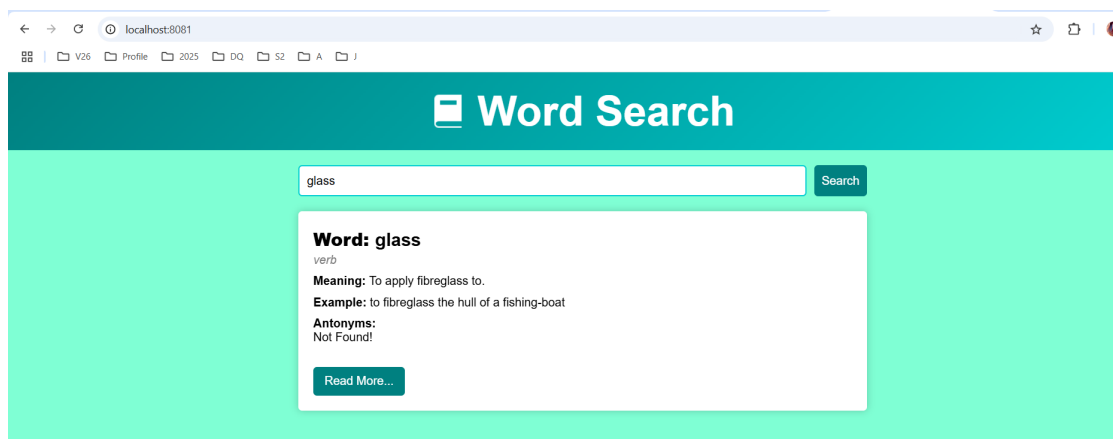
Layer ID	Layer Name	Size
19	WORKDIR /usr/share/nginx/html	4.1 KB
20	COPY . . # buildkit	45.06 KB
21	EXPOSE map[80/tcp:[]]	0 B
22	CMD ["nginx" "-g" "daemon off;"]	0 B

The 'Vulnerabilities' section shows a search bar and filters for 'Fixable' and 'Show excepted'. It also displays a message: 'No vulnerabilities were introduced in the selected layers.'

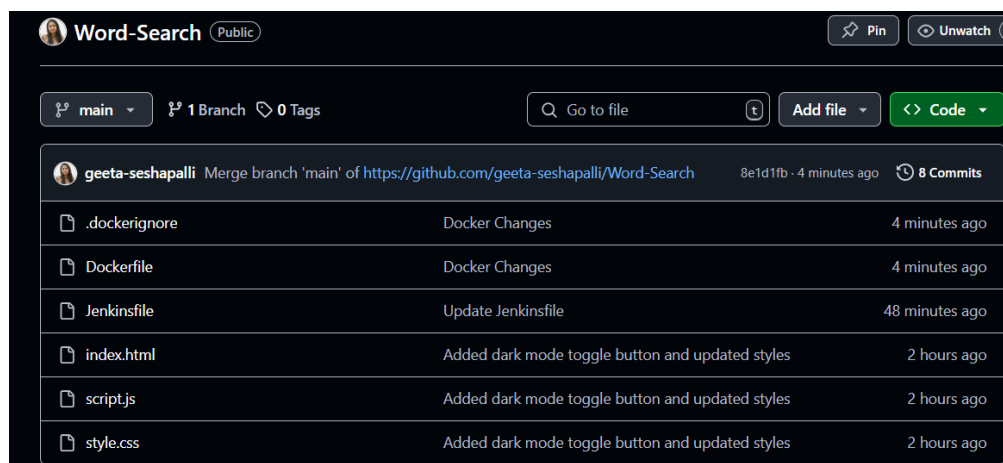
Description: Docker Image

Access Your Application in the Browser

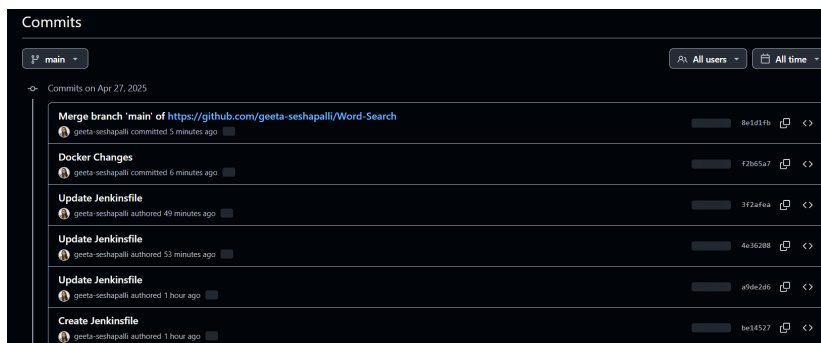
Once the container is running, you should be able to access your WordSearch app in the browser. You mapped port 8081 on your local machine to port 80 in the container, so open your browser and visit:



Description: Access Your Application in the Browser



Description: Docker Changes



Description: Docker Commit Changes



Description: Docker Changes in Jenkins

```
[Pipeline] echo
No build step configured, skipping.
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to Staging)
[Pipeline] echo
Deployment to staging server complete!
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Notify)
[Pipeline] echo
Deployment completed!
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Build and Deployment successful!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Description: Docker Changes in Jenkins

Outcome:

- A functional containerized application that runs independently of the host system's environment
- Practical experience with Docker commands and concepts such as images, containers, volumes, and networking
- Understanding of how Docker facilitates efficient content and configuration management

Conclusion:

Through this project, a strong foundation in Docker's core commands and concepts was developed. The hands-on experience of containerizing a simple application demonstrated the power of Docker in managing application content, dependencies, and deployment with ease. This forms a stepping stone toward container orchestration and more complex DevOps implementations.