

Aim:

To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers

Theory:

1. Docker Architecture:

Docker employs a client-server model. The pivotal components encompass:

- Docker Daemon: Serving as a background process, it orchestrates Docker objects such as images, containers, networks, and volumes.
- Docker Client: Serving as the primary interface for users, it dispatches commands to the Docker Daemon for execution.
- Docker Images: Immutable blueprints housing filesystems and application code.
- Docker Containers: Lightweight, portable, and self-contained entities that execute applications.

2. Container Lifecycle:

- Create: A container materializes from an image using the ``docker run`` command.
- Start: The container commences operation via the ``docker start`` command.
- Pause: Utilizing the ``docker pause`` command, a container can be halted.
- Unpause: A paused container can be reinstated to activity with the ``docker unpause`` command.
- Stop: Halting a container is accomplished via the ``docker stop`` command. This methodically sends a SIGTERM signal, prompting processes within the container to gracefully conclude.
- Kill: If exigencies arise, forcefully halting a container is achieved using the ``docker kill`` command. This instantaneously terminates processes within the container by dispatching a SIGKILL signal.
- Remove: Post-halt, a container can be eliminated from existence via the ``docker rm`` command.

3. Managing Docker Images and Containers:

- Image Management: Docker offers a spectrum of commands:
 - `docker pull``: Fetches images from registries.
 - `docker build``: Constructs images from Dockerfiles.

- docker push: Uploads images to registries.

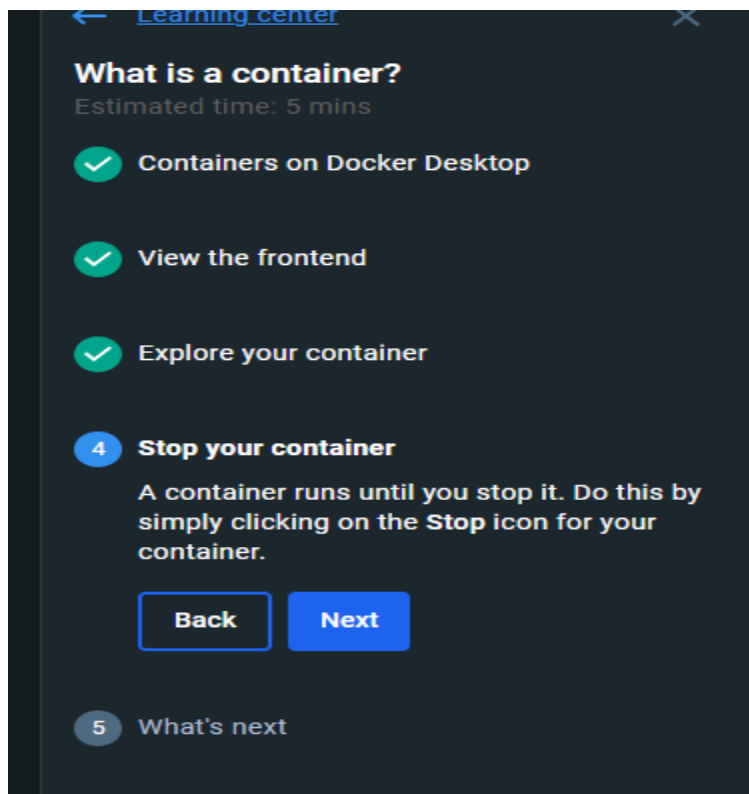
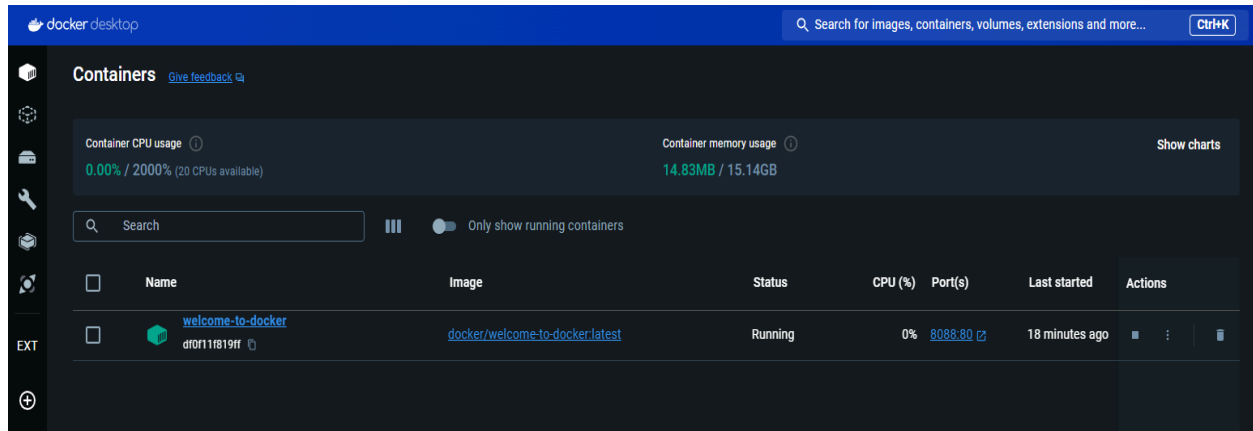
- Container Management: Docker commands wielded in container administration encompass:

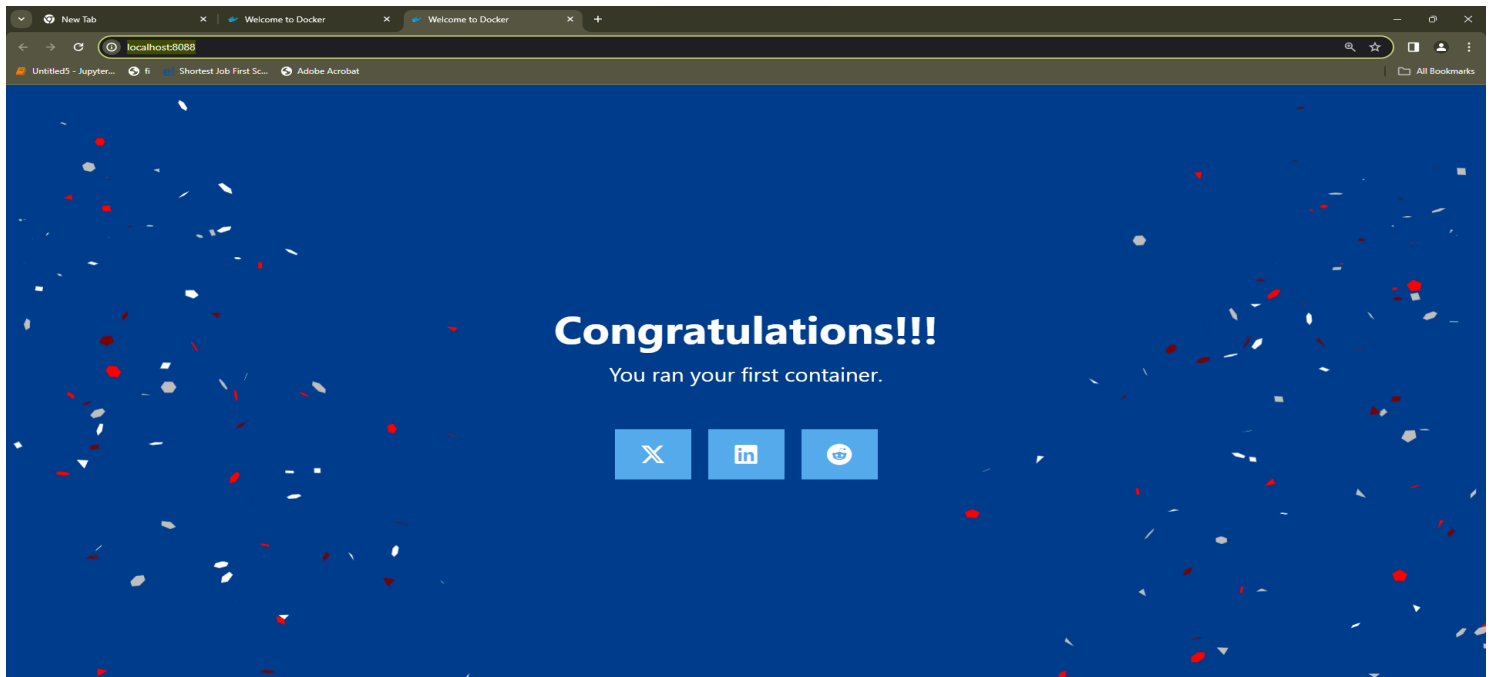
- docker run: Instantiates and launches containers.

- docker ps: Enumerates extant containers.

- docker stop: Halts a running container.

- docker rm: Deletes a container.



A screenshot of the Visual Studio Code editor. The Explorer pane on the left shows the file structure of a project named "WELCOME-TO-DOCKER", including files like .github, public, src, .dockerignore, .gitignore, Dockerfile, MAINTAINERS.md, package-lock.json, package.json, and README.md. The Dockerfile is open in the editor, showing the following content:

```
15 RUN npm install \
16   && npm install -g serve \
17   && npm run build \
18   && npm run start
```

The TERMINAL pane at the bottom shows the output of the Docker build process. It displays a series of build steps, each starting with "[+] Building" followed by a duration and a progress indicator (9/10). The steps include loading build definitions from the Dockerfile and loading metadata for the docker.io/library/node:18-alpine image. The output is as follows:

```
[+] Building 31.8s (9/10)
[+] Building 31.9s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.1s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.2s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.4s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.5s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.7s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.8s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 33.0s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 33.1s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 33.3s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 33.4s (9/10)
=> [internal] load metadata for docker.io/library/node:18-alpine
[+] Building 38.7s (9/10)
=> [internal] load metadata for docker.io/library/node:18-alpine
[+] Building 38.9s (9/10)
=> [internal] load metadata for docker.io/library/node:18-alpine
[+] Building 39.0s (9/10)
=> [internal] load metadata for docker.io/library/node:18-alpine
```

```

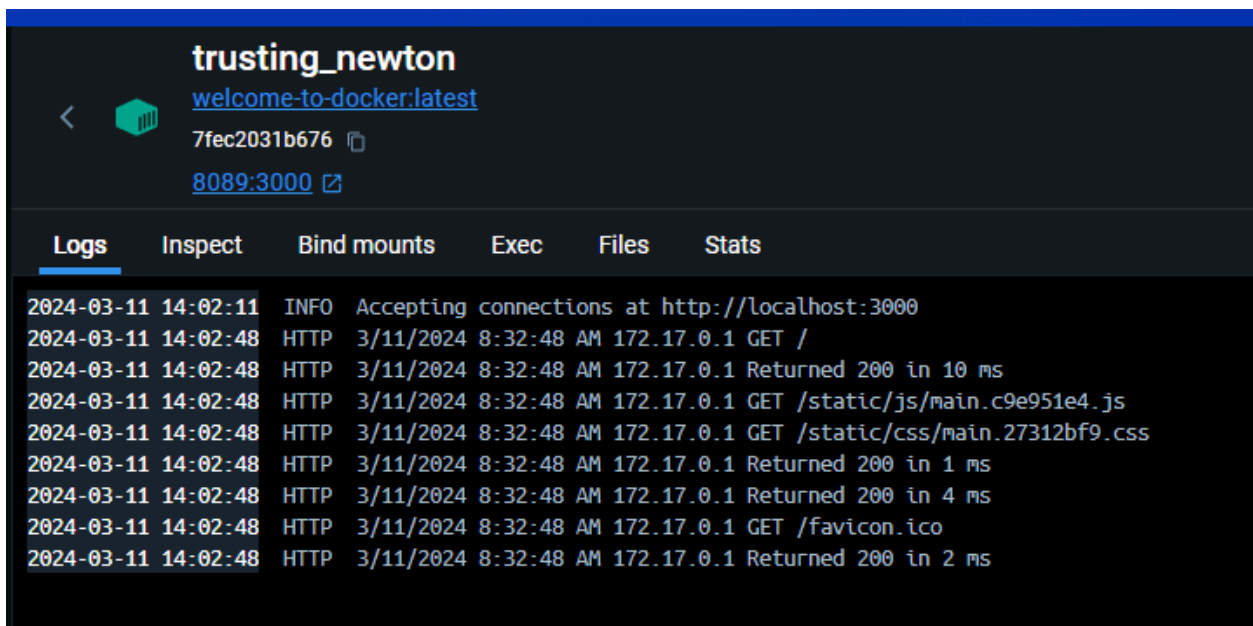
=> => naming to docker.io/library/welcome-to-docker

What's Next?
1. Sign in to your Docker account → docker login
2. View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\15L\Desktop\welcome-to-docker> docker scout quickview
i New version 1.5.1 available (installed version is 1.4.1) at https://github.com/docker/scout-cli
v SBOM of image already cached, 307 packages indexed

Target          local://welcome-to-docker:latest  0C  0H  1M  0L
digest          0e7372b5f723
Base image       node:18-alpine                   0C  0H  1M  0L
Updated base image node:18.19.0-alpine              0C  0H  1M  0L

What's Next?
View vulnerabilities → docker scout cves local://welcome-to-docker:latest
View base image update recommendations → docker scout recommendations local://welcome-to-docker:latest
Include policy results in your quickview by supplying an organization → docker scout quickview local://welcome-to-docker:latest --organization

```



The screenshot shows the Docker Scout quickview interface for the image `trusting_newton/welcome-to-docker:latest`. The interface includes a header with the image name and a back arrow. Below the header, there is a section for the image ID `7fec2031b676` and a link to the image details page `8089:3000`. The main section is a table with tabs for `Logs`, `Inspect`, `Bind mounts`, `Exec`, `Files`, and `Stats`. The `Logs` tab is selected, showing a list of log entries with timestamps, log levels, and messages.

Timestamp	Level	Message
2024-03-11 14:02:11	INFO	Accepting connections at http://localhost:3000
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 GET /
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 Returned 200 in 10 ms
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 GET /static/js/main.c9e951e4.js
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 GET /static/css/main.27312bf9.css
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 Returned 200 in 1 ms
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 Returned 200 in 4 ms
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 GET /favicon.ico
2024-03-11 14:02:48	HTTP	3/11/2024 8:32:48 AM 172.17.0.1 Returned 200 in 2 ms

Conclusion:

Comprehending Docker architecture and the lifecycles of containers is pivotal for leveraging Docker proficiently in software development and deployment. Armed with a profound understanding of these concepts and adeptness with Docker commands, one can seamlessly manage Docker images and containers, streamline developmental workflows, and deploy applications with utmost efficacy.