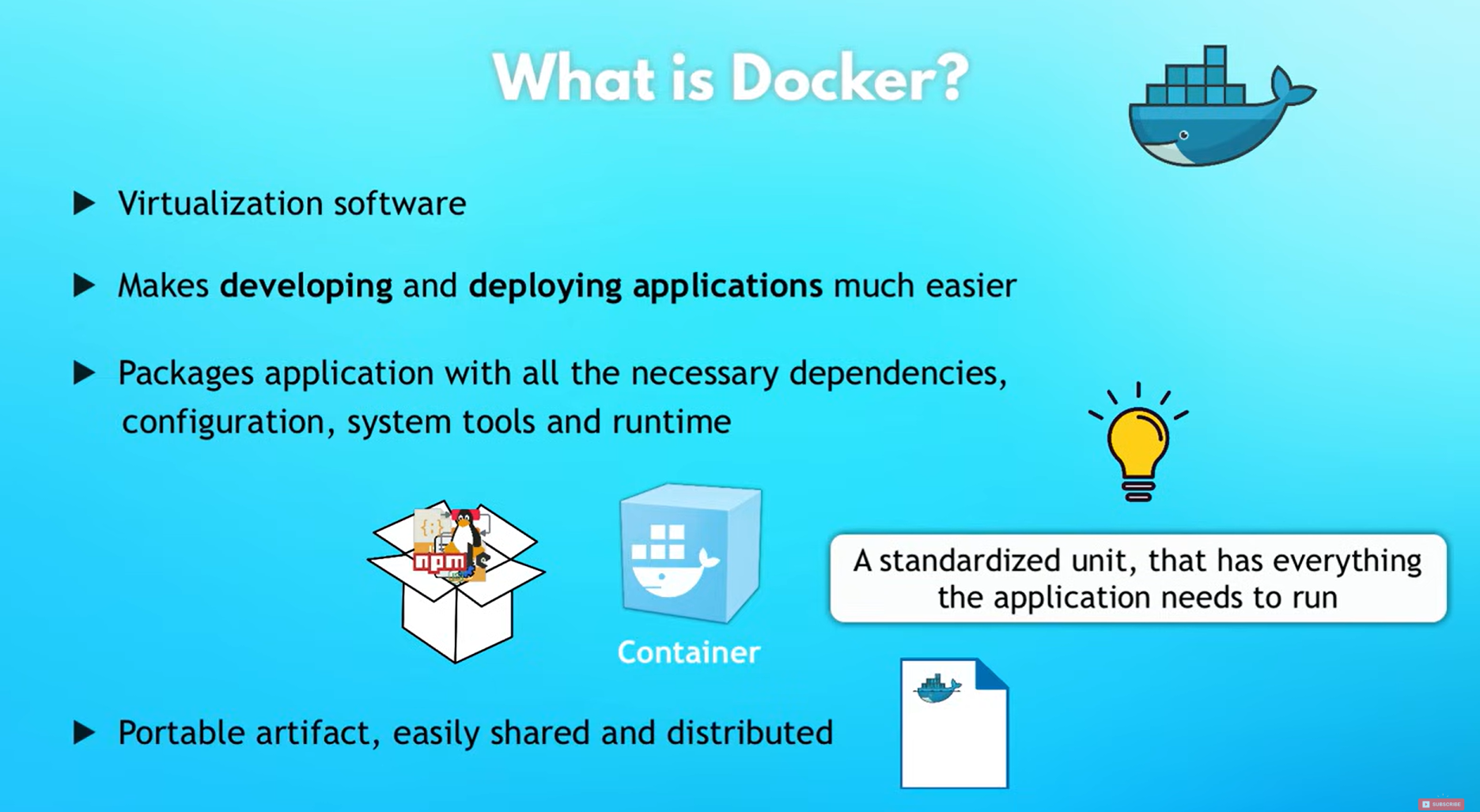
Docker Playlist : <https://videoportal.epam.com/playlist/va8EzVY1>

Docker git repo: <https://githyd.epam.com/Rakesh_Budugu/docker-test-automation.git>

Docker certified associate Course - [(1) New Messages! (pluralsight.com)](https://www.pluralsight.com/cloud-guru/courses/docker-certified-associate-dca)

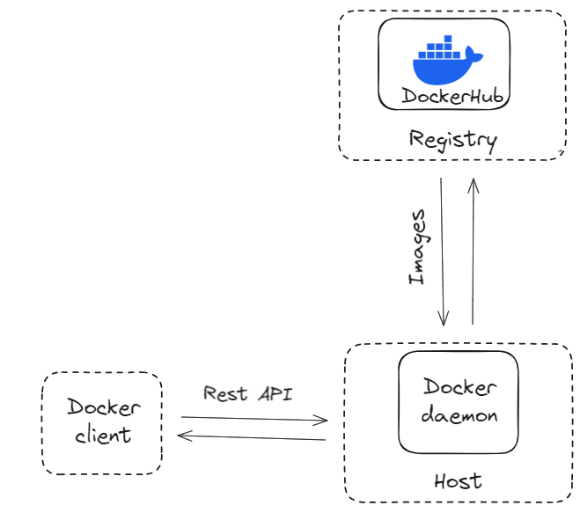


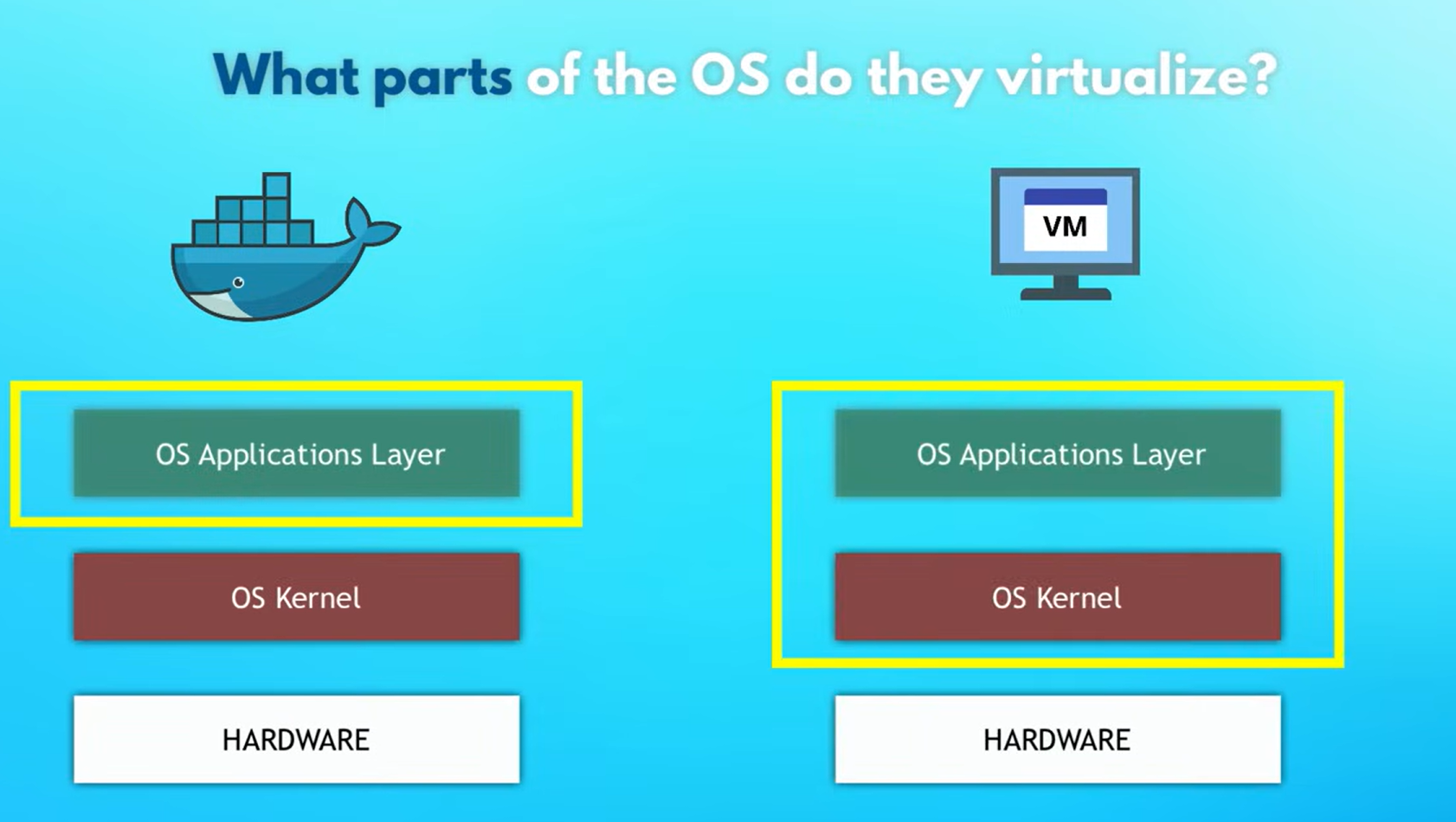


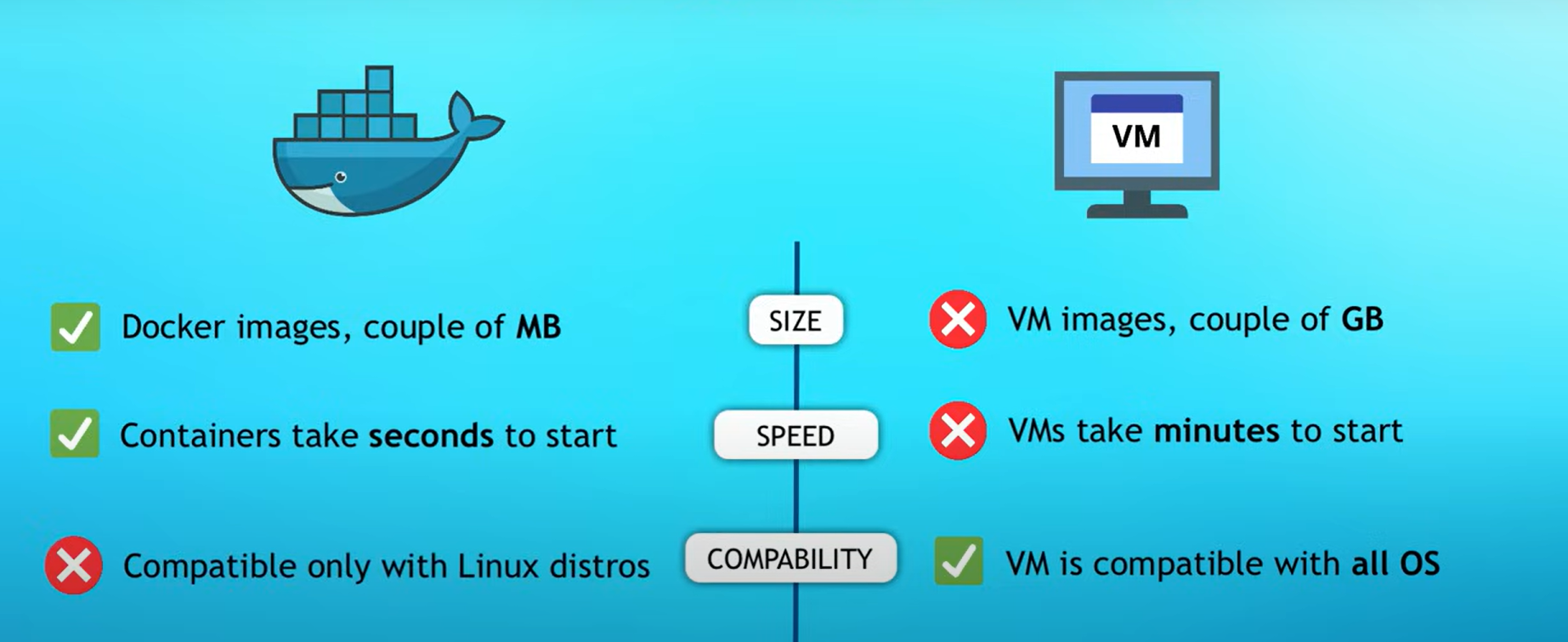
* **Docker Engine:**Docker engine is the runtime that executes containers.
* **Docker Images:**[Docker images](https://www.geeksforgeeks.org/what-is-docker-images)are lightweight, readable templates containing executable packages that include the application with its dependencies.
* **Docker Containers:**[Docker containers](https://www.geeksforgeeks.org/docker-containers-hosts) are standardized , encapsulated environments that run applications/instances of Docker images.
* **Docker Compose:**[Docker compose](https://www.geeksforgeeks.org/docker-compose)is a tool for defining and running multi-containered Docker applications
* **Docker Daemon :** The Docker daemon, often referred to as dockerd, is a persistent background process that manages Docker containers on a host.

It handles tasks such as:

* Pulling container images from registries (e.g., Docker Hub).
* Creating new containers.
* Starting, stopping, and managing existing containers.
* Managing storage and networking configurations for containers.
* Ensuring security and isolation between containers.







Linux can not be run on Windows/mac OS kernal

But through Docker Desktop we can run linux containers

A blue whale with blocks on it

Description automatically generated

The **docker exec** command is used when you need to execute a new command in a running Docker container.  -i: attach to ubuntu -t: shell

docker exec -it ubuntu bash

Alpine by default does not come with the **bash** shell installed; it uses **ash** (Almquist Shell), a lightweight Bourne-compatible shell.

1. **-i** (or **--interactive**): This flag keeps the standard input (STDIN) open even if not attached. It ensures that the container can receive input from the user, which is important for interactive applications.
2. **-t** (or **--tty**): This allocates a pseudo-TTY (a pseudo-terminal), which provides terminal capabilities to the command line within the container. This enables users to interact with the command line interface within the container as if they were interacting with a regular terminal session.

Container is running but you don’t want to disturb container and press ctrl+P+Q

[Docker Commands - GeeksforGeeks](https://www.geeksforgeeks.org/docker-instruction-commands/)   
lot more commands are learn from google

-d – detach mode not show running commands on screen runs in backgrd

-p – publish , port mapping -p[host port]:[container port]

-P – publishing, automatically docker sees available port and publishes

rmi – remove image

--restart

inspect, logs, history, top

use --help

A screen shot of a phone

Description automatically generated

After exit it will restart again as we use –-restart always command

Id: 12 characters

docker rm -f $(docker ps -aq) – to delete all existing containers

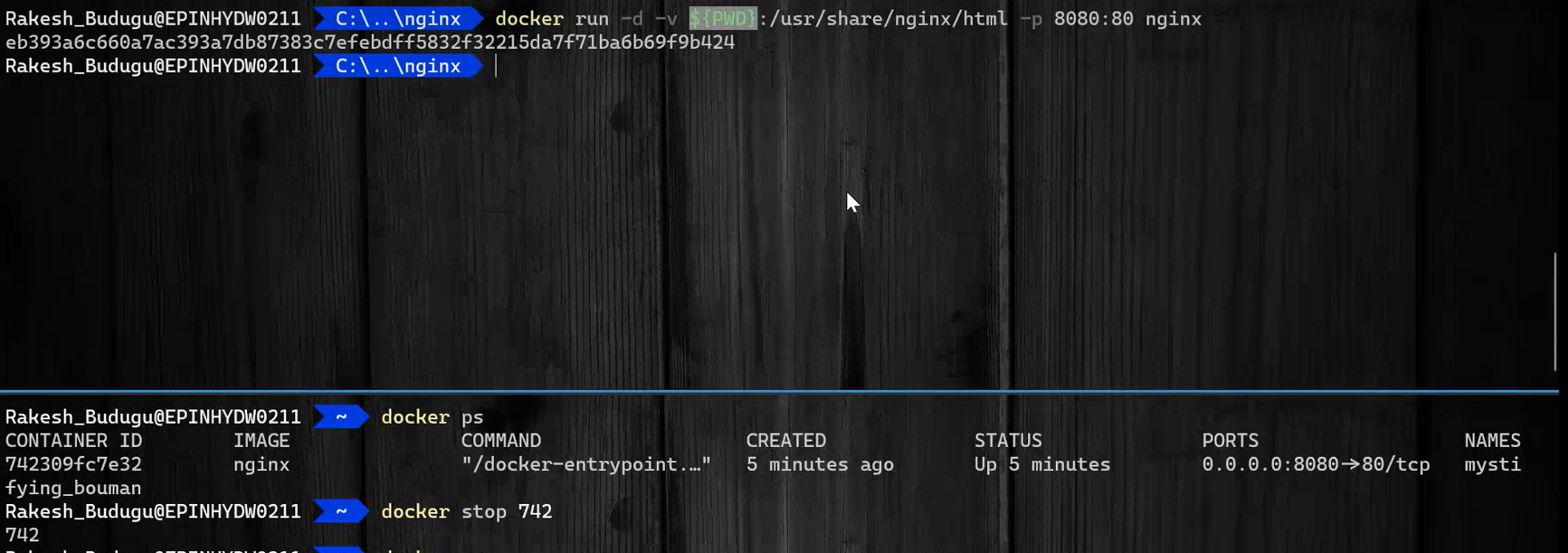
* **docker ps -aq**: This lists all containers (**-a** for all, **-q** for quiet mode, which only outputs numeric IDs).
* **docker rm -f**: The **rm** command removes the specified containers. The **-f** flag forces the deletion of running containers (force removal).

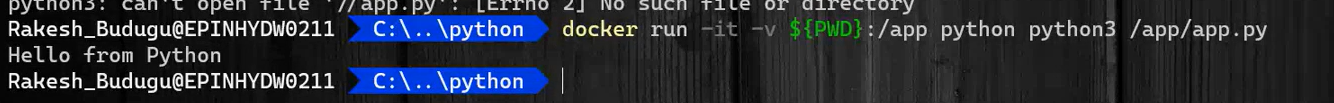
A white paper with black text

Description automatically generated

A screen shot of a computer

Description automatically generated





nginx, python, node, express

docker container run --name CustomJSContainer -it -w /app -v ${PWD}:/app node node index.js

docker run -it node – interactive mode

Express:  
docker container run --name ExpressJsInit -it -w /app -v ${PWD}:/app node npm init

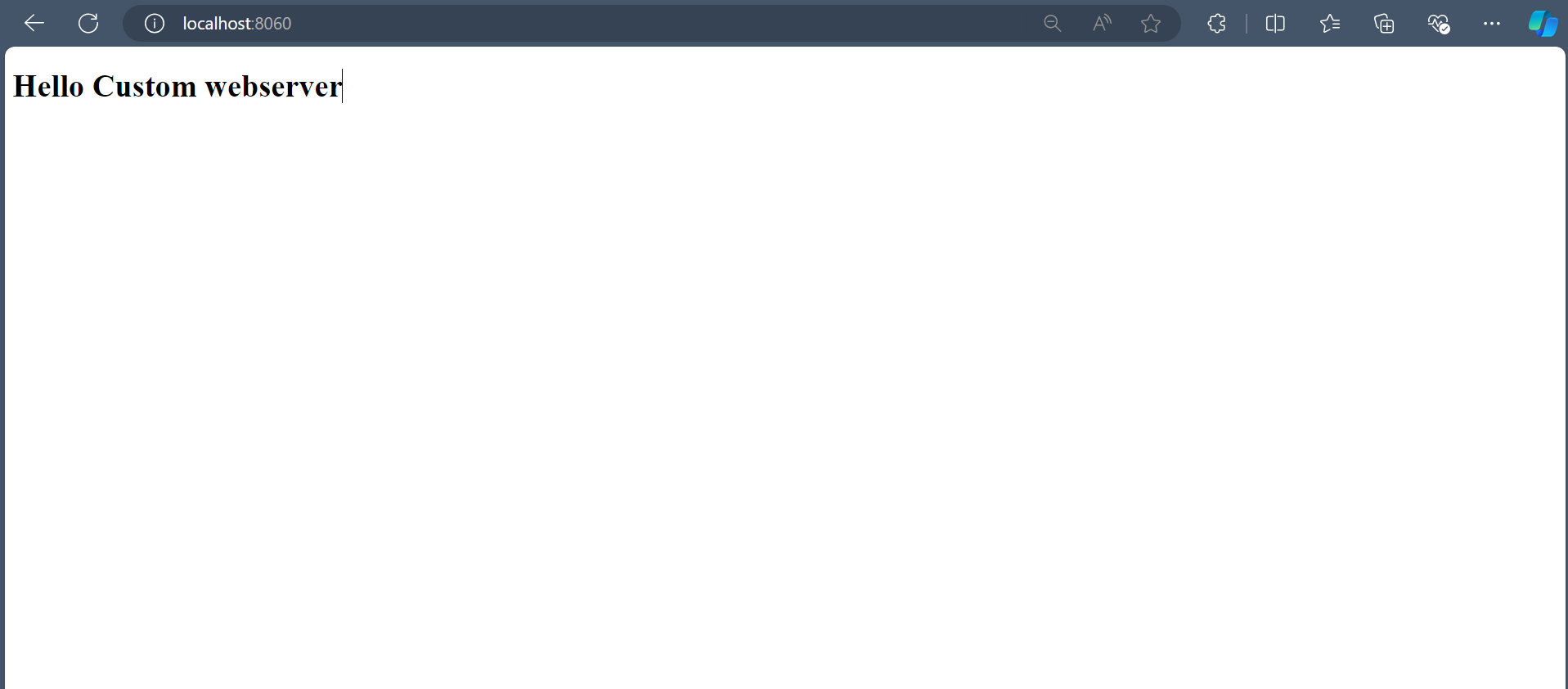
docker container run --name ExpressJsInstall -it -w /app -v ${PWD}:/app node npm install express

docker container run --name CustomExpressJsPortMap -it -p 3000:3000 -w /app -v ${PWD}:/app node node index.js

docker exec -it CustomExpressJs bash – same container for installing express

A screenshot of a computer

Description automatically generated



**Dockerfile**:

docker build -t figlet .

[Dockerfile reference | Docker Docs](https://docs.docker.com/reference/dockerfile/)

<https://youtu.be/1ymi24PeF3M?si=_Pa-VatUm0L-MFLz>

docker image build -t csharpdockerimage .

docker image tag csharpdockerimage vishnu855/csharpdockerimage

docker run --rm --name CSharpDemo csharpdockerimage

docker image push vishnu855/csharpdockerimage

**Docker networking:**

A screenshot of a computer

Description automatically generated

Docker networking allows containers to communicate with each other and with the outside world.

**Types of Docker Networks:**

1. **Bridge Network**: This is the default network type when you run a container without specifying a network. Containers on a single host can communicate with each other using this network type. Each container connected to a bridge network is assigned an IP address. Most user-defined networks are of this type.
2. **Host Network**: If you use the host network mode for a container, that container’s network stack is not isolated from the Docker host (the container shares the host’s networking namespace), and the container does not get its own IP-address allocated. This is often used for services where performance is critical, like high-traffic web servers.
3. **Overlay Network**: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons.
4. **None Network**: This mode disables all networking for the container. It is usually used for containers where network isolation is important or where the container doesn’t network to communicate with other containers or outside entities.
5. **Macvlan Network**: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The container has its own IP address and can communicate with the rest of the network directly.

**Common Docker Networking Commands:**

1. Create a network:

*docker network create --driver <driver\_name> <network\_name>*

Example to create a bridge network:

*docker network create --driver bridge my-bridge-network*

2. List networks:

*docker network ls*

3. Inspect a network (to see which containers are connected to it, among other details):

*docker network inspect <network\_name>*

4. Connect a container to a network:

*docker network connect <network\_name> <container\_name>*

This is useful when your container is already running, and you need to connect it to a network.

docker network connect --ip 192.168.10.5 my-network my-container

5. Disconnect a container from a network:

*docker network disconnect <network\_name> <container\_name>*

6. Remove a network:

*docker network rm <network\_name>*

Note that you cannot remove a network that has containers attached to it.

*docker run --net my-network --name my-container -d nginx*

**Docker Volume**: even after container deleted or stopped data will persists

Docker volumes are a mechanism for persisting data generated by and used by Docker containers. They are especially useful for when you need to keep data around even when a container is deleted or updated, or you need to share data between containers. Volumes are stored in a part of the host filesystem which is managed by Docker

1. **Create a volume**:

*docker volume create my-volume*

This creates a new volume named **my-volume** that containers can utilize to store data.

1. **List volumes**:

*docker volume ls*

This lists all the volumes known to Docker on the host machine.

1. **Inspect a volume**:

*docker volume inspect my-volume*

This provides detailed information about a specific volume, such as where it is stored on the host.

1. **Remove a volume**:

*docker volume rm my-volume*

This deletes the specified volume if no containers are currently using it.

1. **Remove all unused volumes**:

*docker volume prune*

This removes all volumes not presently used by at least one container.

docker run -d --name my-container -v my-volume:/path/in/container nginx

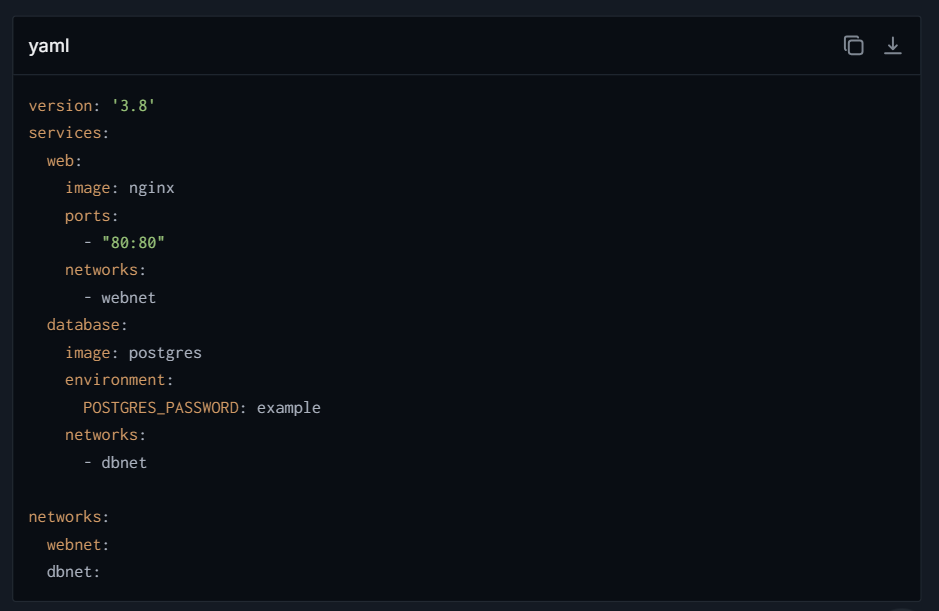
and in docker compose

**Docker Compose:**

Docker Compose is a tool for defining and managing multi-container Docker applications. With Compose, you use a YAML file to configure your application's services, networks, and volumes.

Docker Compose is particularly useful for local development, testing, and CI/CD stages of the development workflow.

**docker-compose.yml**



build : . -(if Dockerfile is present and no inbuilt image is present)

tag:[“UI”, “desktop”] in yaml -> tag:

-UI

-desktop

**Common Docker Compose Commands: start, stop, up, down, rm, logs, ps**

1. **Start Services**:

bash

docker-compose up

This command reads the **docker-compose.yml** file in the current directory, builds/creates/starts containers accordingly. If you add **-d**, it will do so in detached mode (in the background).

1. **Stop Services**:

bash

docker-compose down

This stops and removes the containers, networks, volumes, and images created by **up**.

1. **Build or Rebuild Services**:

bash

docker-compose build

This builds or rebuilds services. Necessary when you need to rebuild your containers after making changes to them.

1. **View Logs**:

bash

docker-compose logs

Displays log output from services. Particularly useful in development to debug issues.

1. **List Containers**:

bash

docker-compose ps

Lists all containers related to the **docker-compose.yml** file in the current directory.

1. **Execute a Command in a Running Container**:

bash

docker-compose exec web bash

This opens a bash shell inside the **web** service container. Replace **bash** with any command you need to run inside that container.

1. **Stop All Active Services**:

bash

docker-compose stop

Stops running containers without removing them. They can be started again with **docker-compose start**.

1. **Remove Stopped Containers**:

bash

docker-compose rm

Removes stopped service containers. By default, it does not remove volumes.

1. **Pull Services' Images**:

bash

docker-compose pull

Pulls an image associated with a service defined in **docker-compose.yml**.

1. **Scaling Services** (not typically needed in version 3+ with swarm mode):

bash

docker-compose up --scale web=3

**solenoid:**

[Aerokube Selenoid | A Selenium Testing Solution Running Browsers and Mobile Applications in Docker](https://aerokube.com/selenoid/)

Commands used:

./cm.exe selenoid start –vnc

./cm.exe selenoid’;-ui start

./cm.exe selenoid stop

./cm.exe selenoid-ui stop

docker pull selenoid/video-recorder

code

options = new ChromeOptions();

firefoxOptions = new FirefoxOptions();

firefoxOptions.AddAdditionalOption("selenoid:options", new Dictionary<string, object>

{

["enableVNC"] = true,

["enableVideo"] = true

});

**Zalenium:**

<https://youtu.be/_C5bekbxNpc?si=dQxfhIJKdqJnxFEn>

docker pull elgalu/selenium - core

docker pull dosel/zalenium – logic

docker stop zalenium

A computer screen shot of text

Description automatically generated