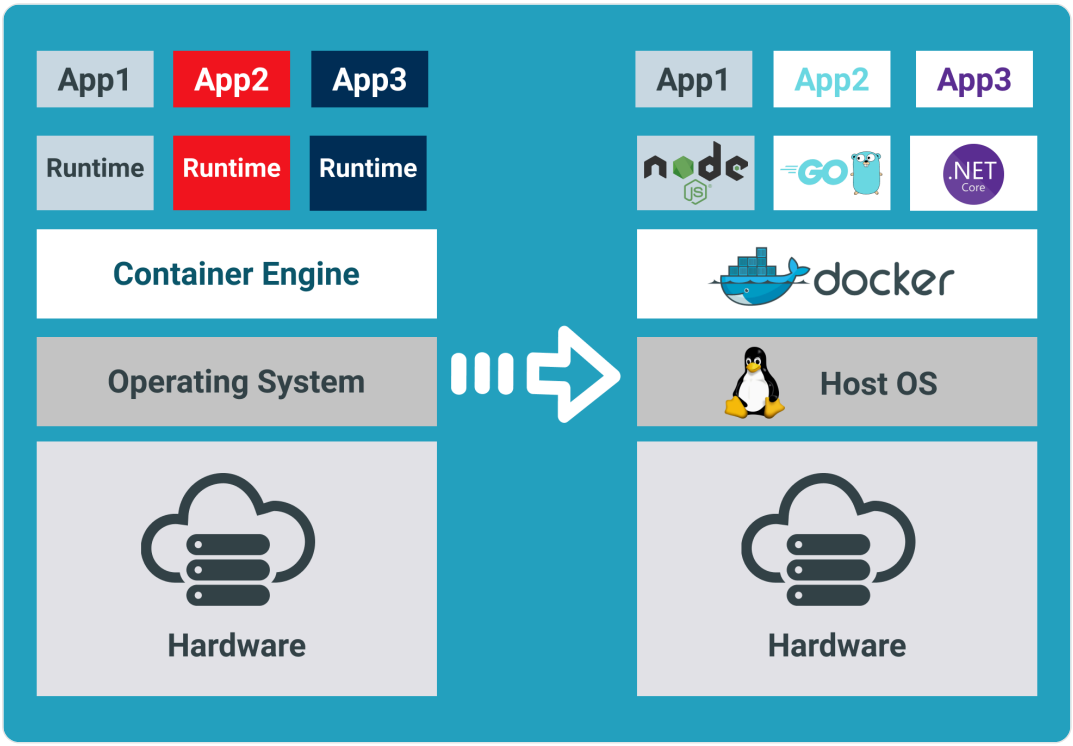
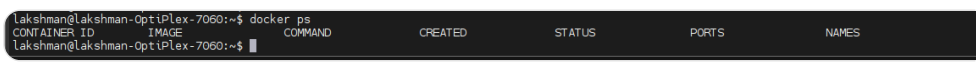
Docker commands

Docker is an open platform that helps developers build, run, and ship applications in containers.



1. Update the system	sudo apt-get -y update
2. Install docker	sudo apt-get install -y docker
3. To install docker packages we use .io	sudo apt install docker.io
4. To check version	docker --version / docker -v
5. To check first docker status	sudo systemctl status docker → o/p is like **Active: active (running)**
6. To start , enable and stop docker service	sudo systemctl start docker sudo systemctl enable docker sudo service docker stop → after stopping docker using - EX: Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled) **Active: inactive (dead)** to check docker started or not - ps aux | grep docker
7. check docker list	docker ps  fix permissions if needed

sudo chmod 666 /var/run/docker.sock

you get permission denied

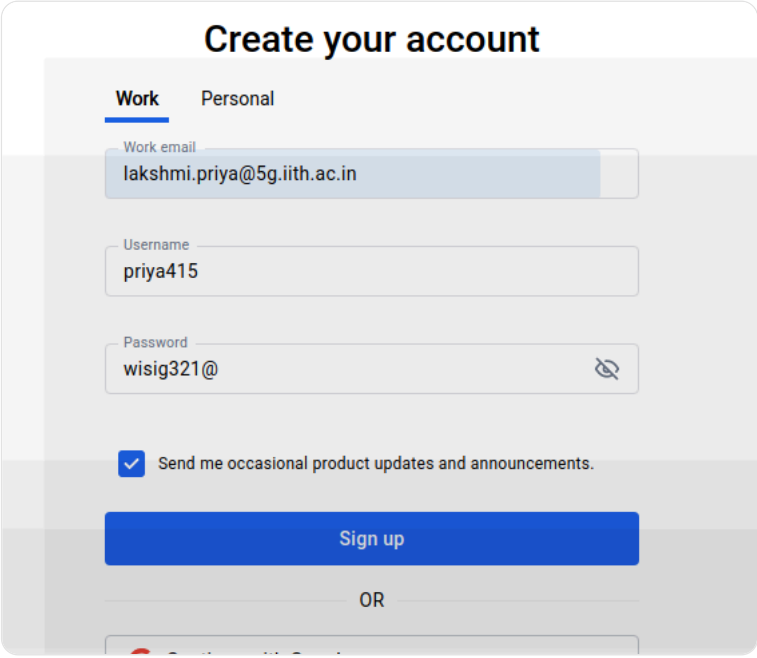
- By default, **only the root user** (or someone using `sudo`) can run Docker.
- Your user **does not have permission** to use Docker yet.

8. Add Your User to the Docker Group

sudo usermod -aG docker $USER
lakshman in user place

9. basics

→ docker info //to get the information
→ docker --help
→ docker login
to login create account in docker hub - go to signup option



to check succesfully login or not
docker info | grep Username

10. project python simple example

docker file is used to build an image
syntax : INSTRUCTION arguments

- Instructions like `FROM`, `COPY`, `RUN`, and `CMD` are case-sensitive and written in uppercase.

Step 1 : Create a file named Dockerfile
Step 2 : Add instructions in Dockerfile
Step 3 : Build dockerfile to create image
Step 4 : Run image to create container

```
1.create a python script.
   vi app.py
   print("Hello from Docker!")

2. create a docker file
 # Use an official Python image
 FROM python:3.9
```

- we use `python:3.9`, which is a pre-built image containing Python 3.9.

```
# Copy the Python script into the container
COPY app.py /app.py
```

- Copies `app.py` from the **local system** to the container's file system at `/app.py`

```
# Command to run the script
CMD ["python", "/app.py"]
```

- command to run inside container

```
3. build a docker file  [ use dot at last]
    docker build -t my-python-app .
       -t my-python-app ->  [tags the image with the name]
       . -> means current directory (Dockerfile location)
4. run the docker container
     1st way - docker run my-python-app
     2nd way -sudo docker run -d --name tomcat-sample -p 8091:8080 sample-tomcat-app --restart alway
```

- `-d` → run in background (detached mode)
- `--name tomcat-sample` → gives your container a name
- `-p 8091:8080` → maps port **8091 (your system)** → **8080 (inside container)**
- `sample-tomcat-app` → name of your image

`--restart always` → ensures the container **auto-restarts** if it stops or system reboots

3rd way - sudo docker run --name tomcat-container1 -p 8091:8080 mytomcatapp1 [hostport:containerport]

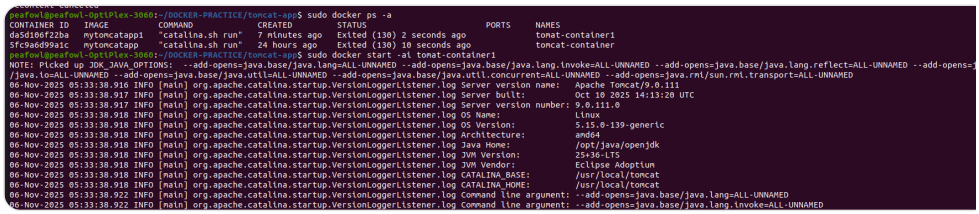
- `--name tomcat-container1` → container name
- `-p 8091:8080` → maps local port 8091 to Tomcat's 8080 port inside the container
- `mytomcatapp1` → your image name

To stop container ctrl+c and re run

4th way - sudo docker start -ai tomcat-sample

(`-a` = attach, `-i` = interactive → show logs live)

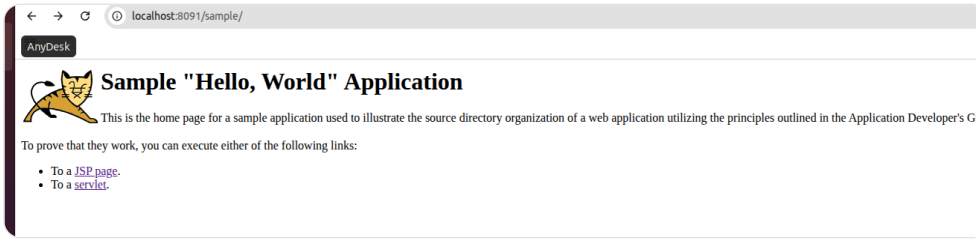
before running container check EXITED STATUS , PORTS , CREATED
EXITED means completed



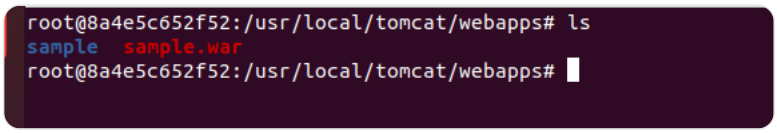
When container is in running state , will see the ports and status is up

You can open the browser and check
http://localhost:8091/sample

Output



sudo docker exec -it tomcat-container2 /bin/bash



```
a0d0081f02c05   twoscripts   "./hello1.sh"   About a minute ago   Exited (0) About a minute ago
peafowl@peafowl-OptiPlex-3060:~/DOCKER-PRACTICE/two-scripts$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
twoscripts    latest    27ce09cb29b0   17 hours ago    8.32MB
image1        latest    d9b9c9d2fc7e   18 hours ago    8.32MB
peafowl@peafowl-OptiPlex-3060:~/DOCKER-PRACTICE/two-scripts$ sudo docker run -it image1 sh
^Cpeafowl@peafowl-OptiPlex-3060:~/DOCKER-PRACTICE/two-scripts$ sudo docker run -it twoscripts  sh
/app #
/app #
/app # ls
hello1.sh  hello2.sh
/app #
```

11. CONTAINERS

Git cheat sheet commands: https://docs.docker.com/get-started/docker_cheatsheet.pdf

DOCKER CONTAINER PATH: /var/lib/docker/containers/**ID's**

→ docker ps //to get list of currently running container

→ docker ps -a //to get list of all containers [old ones already finsihing runnig]

→ sudo docker container prune [remove all stopped containers (those with “Exited”)]

→ sudo docker system prune [container & images will be removed]

```
starling@wisig:~/my-docker-python-project$ docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS                      PORT
acedd0d48bbe   my-python-app      "python3 main.py"        11 minutes ago   Exited (0) 11 minutes ago
hoover
6fc069a6e055   hello-world        "/hello"                 4 days ago       Exited (0) 4 days ago
t_franklin
bf60b59a74bf   hello-docker-python   "python app.py"       3 months ago     Exited (0) 3 months ago
_ride
37671d5c47e6   hello-docker-python   "python app.py"       3 months ago     Exited (0) 3 months ago
d_jepsen
faa376d4ccfd   hello-docker       "docker-entrypoint.s…"   3 months ago     Exited (0) 3 months ago
rley
eda77484f959   hello-docker       "docker-entrypoint.s…"   3 months ago     Exited (0) 3 months ago
goodall
de391e5edded   my-app2            "python3 app2.py"        3 months ago     Exited (0) 3 months ago
newton
starling@wisig:~/my-docker-python-project$ docker run my-python-app
```

→ docker run <image> //to run the image

ex: docker run my-python-app

→ docker start <container-id/name>

→ docker stop <container-id/name>

→ docker inspect -f '`.State.Status`' <container-id>

This will return one of the following:

- **running** → The container is successfully started.
- **exited** → The container stopped due to an issue.
- **paused** → The container is paused.

→ docker pause ContainerName/ID

→ docker unpause ContainerName/ID

→ docker top ContainerName/ID

→ docker stats ContainerName/ID

→ docker attach ContainerName/ID

→ docker kill ContainerName/ID

→ *docker rm ContainerName/ID

→ docker history ImageName/ID

→ docker restart <container-name>

12. Images

→ docker images --help

→ docker pull image

→ docker images //to get list of images present locally

→ docker images -q [q for quiet mode].

```
starling@wisig:~/my-docker-python-project$ docker images -q
533ce46f5f9d
5b9cadea6f45
511da186e2cb
aa1dbdfd73b6
aa1dbdfd73b6
908a22fcab8f
fec8bfd95b54
d2c94e258dcb
starling@wisig:~/my-docker-python-project$
```

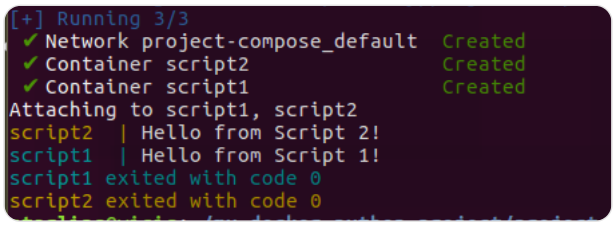
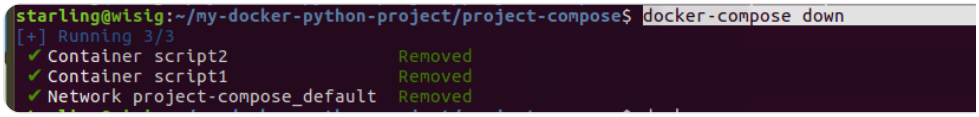
→ docker images -f “dangling=false”

→ docker images -f “dangling=false” -q

Dangling images are unused and untagged Docker images that no longer have a name but still consume disk sp
ex: specified by name none].

→ *docker run image

→ *docker exec -it <image> bash

	→ docker rmi image → docker rmi -f image → docker inspect imagename [Get full image details] → docker history imageName
13. inspect	→ docker inspect <container-id> → docker inspect <image-id> → docker inspect <volume-name> → docker inspect -f '{{.State.Status}}' <container-id> → docker inspect -f '{{.NetworkSettings.IPAddress}}' <container-id>
14. Docker Compose	`docker compose` is a tool that lets you **run multiple containers together** using a single **YAML file** — called `docke compose.yml` fist step after creating compose file **vi docker-compose.yaml** → docker-compose up --build #Build and start the containers  → docker-compose down  → docker-compose logs script1 #check logs in script1 → docker-compose config #validate and view the resolved configuration → docker-compose up -d #to start your services in the **background** (detached from your terminal). → sudo docker compose ps # show running services 🧱 **Dockerfile = how to build a container** ⚙️ **Docker Compose = how to run containers together**
15. Docker volume	→ docker volume //get information → docker volume --help → docker volume create <volume_name> → docker volume ls → docker volume inspect <volume_name> → docker volume rm <volume_name> → docker volume prune *#Remove all unused local volumes*
16. Docker networks	→ docker network create my_network → docker network rm networkID/name → docker network inspect my_network → docker network connect my_network <container_id> #allowing it to communicate with other containers → docker run -d --name web --network my_network nginx # **Attach Containers to a Network** → docker network disconnect my_*network* *<container_id>*