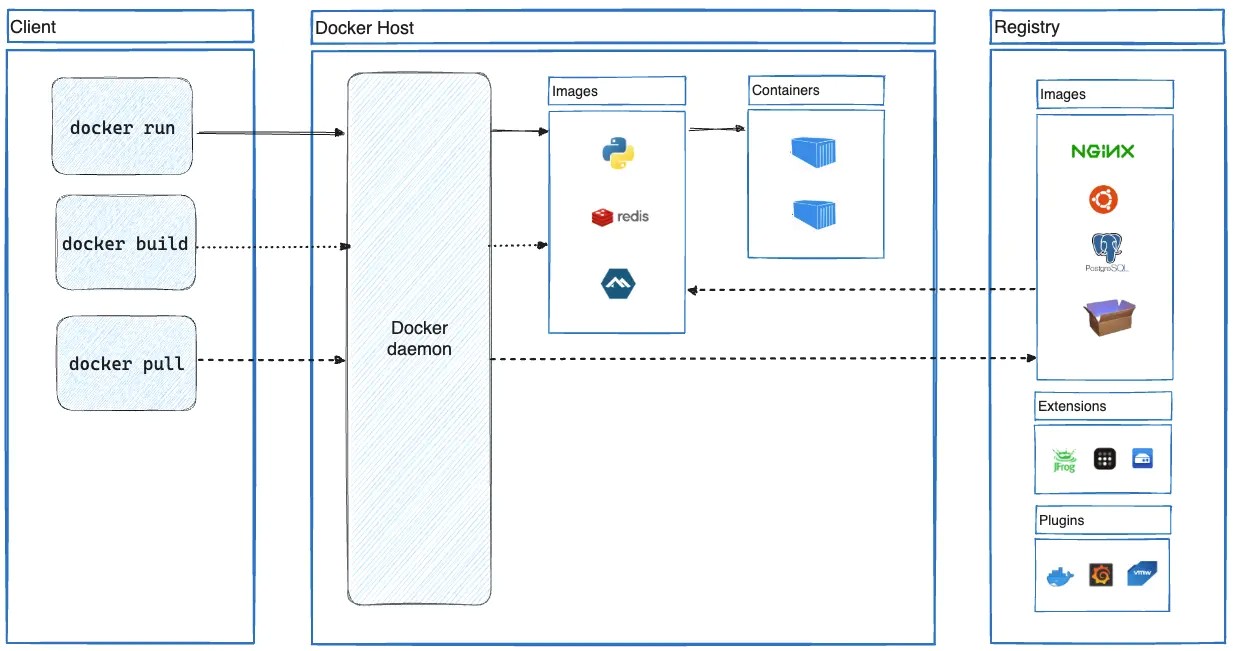
Docker image – oka application ni run cheseki kavalsina requirements (dependency versions) anni oka dockerfile config file rastham , ee file ni base cheskuni oka docker image ni generate chestham.

Ee imge ni host system lo run chesinapdu daane container antam



Docker is a open platform for developing, shipping and running applications.

Enables you to separate your applications from your infrastructure so you can deliver s/w quickly.

Significantly reduce delay time b/w writing code and running it in production.

Provides the ability to package and run an application in a loosely isolated environment called a container.

Container is a light-weight and contain everything needed to run the application, so you don’t need to rely on what’s installed on host.

Images are immutable(cant change)

Containers are created from docker images which are templates defining the application’s components and dependencies.

Key features: Containers

Run independently, ensuring one appln env does not interfere with another appln env.

Can run on any machine that supports docker, irrespective of what underlying hardware/OS

Unlike VMs, can host the OS kernel

Can package an appln and its dependencies into container n move it between dev,test n production env

Docker Registry

ECR Registry – docker registry

Isolating services are IMP

High availability achieved by multiple instances/VMs

Portability matters or eases the deployment

All this raises capEx, opEx

Install Docker engine

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "${UBUNTU\_CODENAME:-$VERSION\_CODENAME}") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Receiving errors when trying to run without root?

The docker user group exists but contains no users, which is why you’re required to use sudo to run Docker commands. Continue to [Linux postinstall](https://docs.docker.com/engine/install/linux-postinstall) to allow non-privileged users to run Docker commands and for other optional configuration steps.

ECR -- ECR registry is docker registry from AWS and ACR registry is docker reg from Azure where we need to store docker images

Docker Daemon listens for docker API requests n manages docker images,voulmes,containers, network, can also communicate with other daemons to manage docker services.

Docker client primary way that user interacts with docker.

When you use docker run, the client sends the requests to daemon(dockerd) which carries them out.

Docker cmd uses DOCKER pai

Docker client can communicate with more than 1 daemon

Docker Desktop is easy to install application for your environment.

Enables you to build, share containerized applns n microservices

Includes daemon, client, compose, content trust, Kubernetes n credential helper

Docker registries store docker images

Dockerhub is a public default registry that anyone can use. docker looks for images in dockerhub by default.

We can have our own private docker registries such cloud based registries which are Azure ACR, AWS ECR, dockerhub, GCR(google container registry) and Inhouse /local registries – Nexus, Jfrog artifactory , DTR(docker trusted registry)

When you use docker pull/docker run docker will pulls the required images from the configured registry

When you use docker push docker pushes your image to configured registry.

Docker Objects:

Docker Images read-only template with instructions for creating docker container.

Image is based on another img

Ex: you build img based on ubuntu img,but installs apache web server n your application as well as the config req to make appln run

You can create your own images or make use of existing images created by others n published in a registry.

To build your own image, need to write dockefiles with instructions and steps needed to create image and run.

Each instruction in the dockerfile creates a layer in the image.

When you change the dockerfile n rebuild the img,only those layers which have changed

are rebuilt.

This is why its light-weight,small n fast

Docker Containers

Runnable instance of an image. docker images become when they run on docker engine

Ex :: docker run -i -t ubuntu /bin/bash

Docker cmds:

Docker images => lists images locally

Docker run => creates new container

Docker ps => lists running containers

Docker ps -a => lists all containers

Docker exec => Executes cmds on containers

Docker start/stop/restart/rm

Docker rmi => docker remove images

Docker inspect => details of container & image

The **docker inspect** command provides JSON information about Docker containers or images, but it is not recommended to directly edit the JSON data. Instead, you can create a new Dockerfile, make the desired changes, build a new image using **docker build**, and then use the updated image for running containers.

* Docker run –name myweb -d nginx

-d if you don’t want this cmd to run in a shell but in background

-name \*\*\* giving name to docker image “nginx”

Bydefault it will running on port 80, but from the document, we need to check processor is running on what port in container.

If this container is running on private network and if you want to access it from outside network you have to map host port.

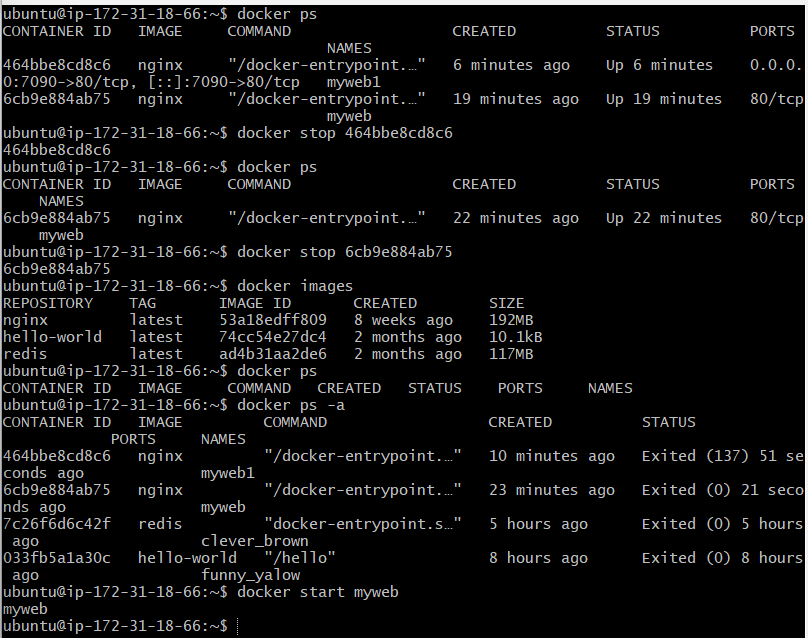
docker run --name myweb1 -p 7090:80 -d nginx

7090(host port) mapping with 80 (container port)

Docker ps

Docker stop name/containerid

Docker start name/containerid



du -sh container

docker exec -it myweb /bin/bash

shows you to execute specific cmds inside a running docker container

i interactive mode allow input from your terminal to container

t allocates pseudo-TTY for the session

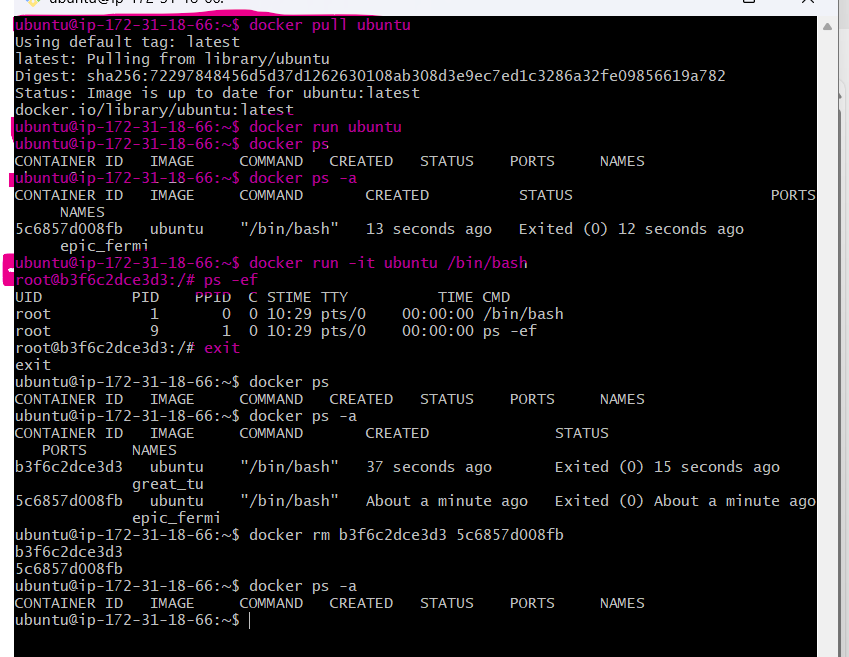
/bin/bash it opens a bash shell, allows you to interact with the container’s filesystem n environment

If you want to remove the image, 1st stop the container n remove the container

docker stop 6cb9e884ab75

docker rm 6cb9e884ab75

docker rmi nginx



Docker pull ubuntu 🡪 you r pulling the img from public registry

Docker run ubuntu 🡪 you r running container based on img

Docker ps and docker ps -a 🡪 you see container is in Exited state

So you have to run docker run -it ubuntu /bin/bash

i interactive mode allow input from your terminal to container

t allocates pseudo-TTY for the session

ps -ef

1. **ps**: This is the command to display information about active processes.
2. **-e**: This option tells ps to show all processes running on the system, not just those related to the current user or session.
3. **-f**: This option provides a full-format listing, which includes additional details about each process.

When you run ps -ef, you'll see columns of information such as:

* **UID**: The user ID of the process owner.
* **PID**: The process ID.
* **PPID**: The parent process ID.
* **C**: CPU utilization percentage.
* **STIME**: Start time of the process.
* **TTY**: Terminal associated with the process (if any).
* **TIME**: CPU time used by the process.
* **CMD**: The command that initiated the process

Docker Logs

The docker logs command is used to view the logs of a running or stopped Docker container. These logs provide useful information, such as application output, error messages, or debugging data from within the container

**Key Options:**

* -f **(follow):** Continuously stream logs in real-time, similar to tail -f.

docker logs -f <container\_name\_or\_id>

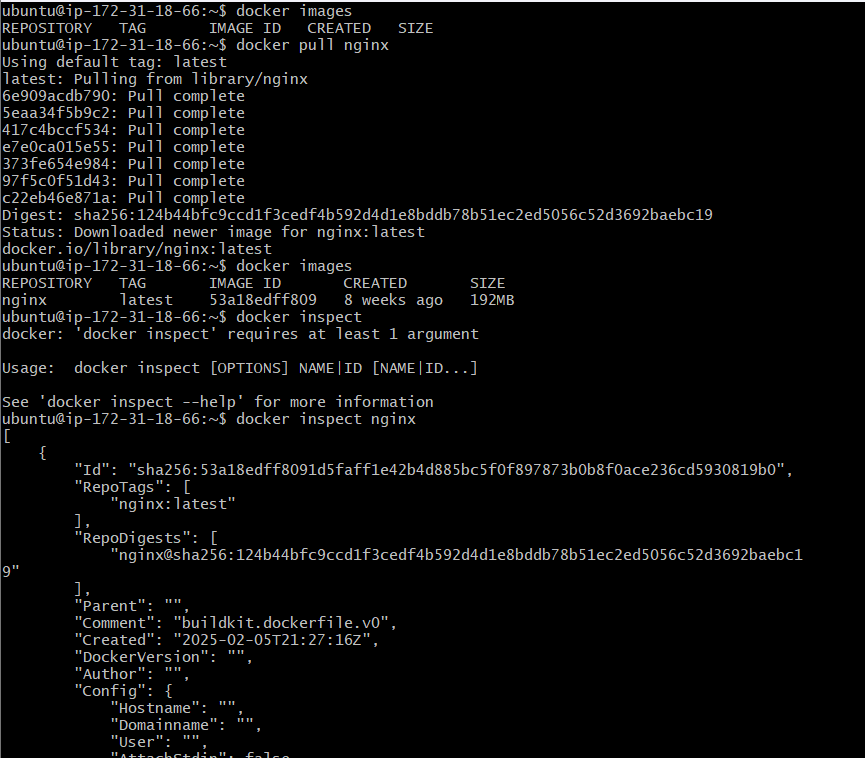
* --since **or** --until**:** Show logs from a specific time frame.

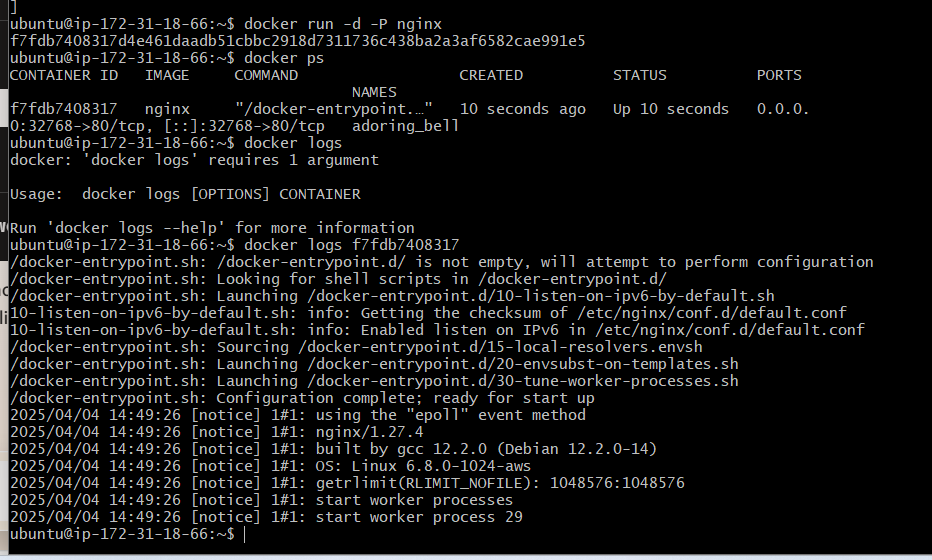
docker logs --since="1h" <container\_name\_or\_id>

This shows logs from the last hour.

* --tail**:** Display only the last *N* lines of logs.

docker logs --tail 50 <container\_name\_or\_id>





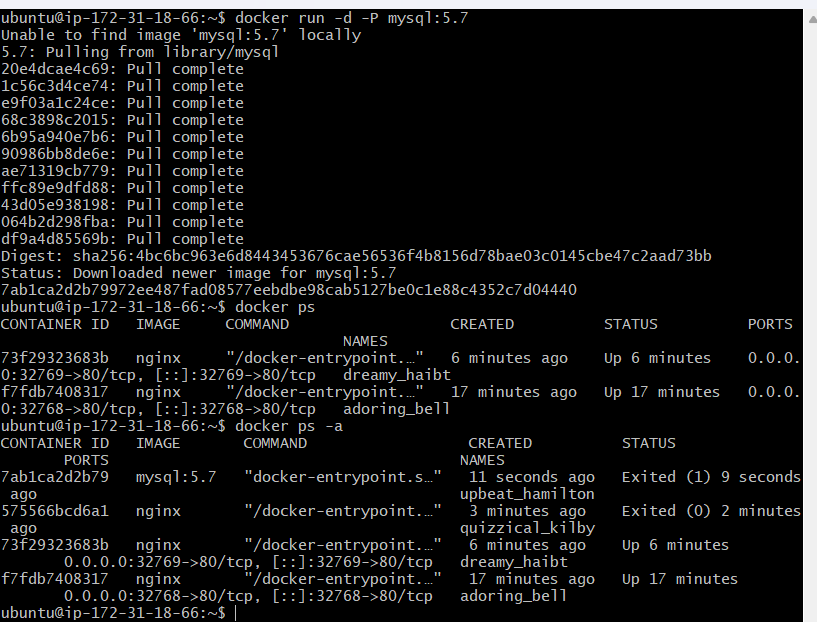
* docker run -d -P nginx

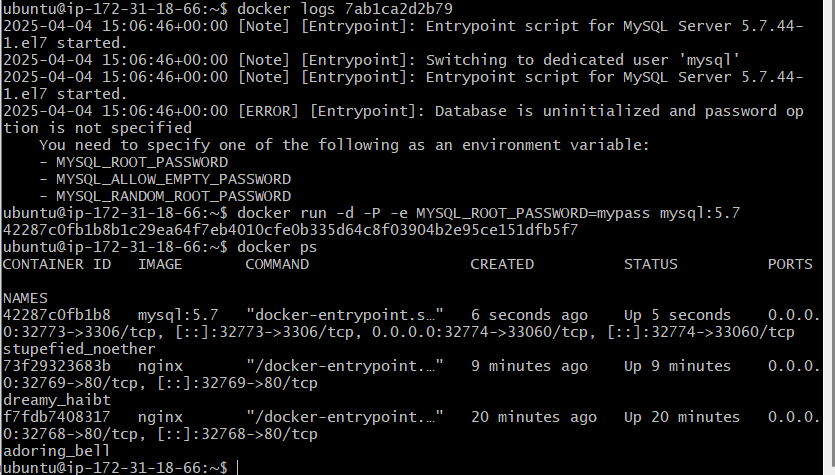
-d (**detached mode)**: This option ensures the container runs in the background, so it doesn’t occupy your terminal session.

-P **(publish all exposed ports)**: This flag automatically maps all the exposed ports in the container to available ports on the host. The port mapping is done randomly unless explicitly specified.

Here if you run a container and see statis with docker ps, it does not show the container but when you check with docker ps -a,it says status as exited

We use docker logs to check what happened and then we make changes according to logs





Docker Volumes

are storage locations created, managed by docker and mounted into containers to store data outside of the container’s filesystem.

Persistent storage for volatile containers

Data doesn’t persist when that container no longer exists, and it can be difficult to get the data out of the data put of the container if another process needs it.

Container’s writable layer is tightly coupled to the host machine where the container is running, do you cant easily move the data somewhere else.

Docker has 2 options to store files in the host machine.

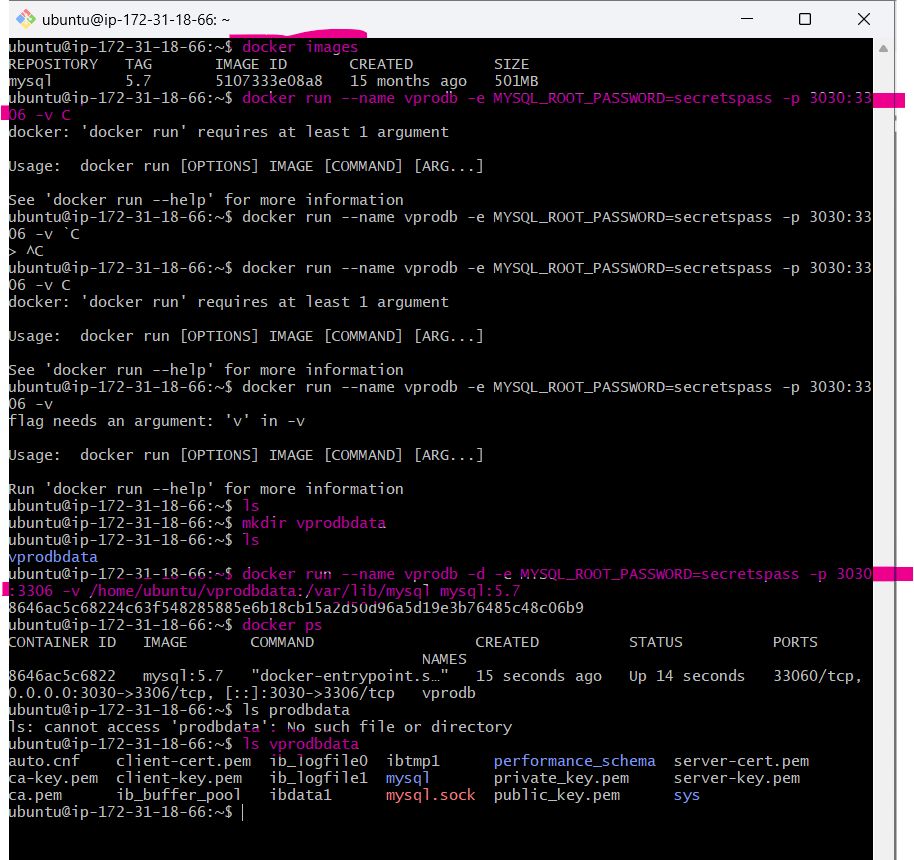
Volumes /var/lib/docker/volumes on Linux

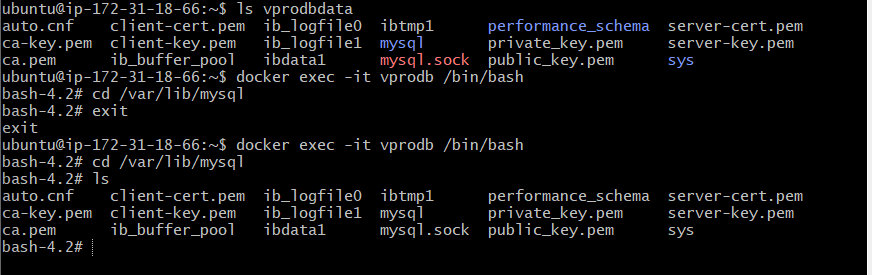
Bind mounts stored anywhere on the host system

Anonymous volumes created automatically by docker for temp storage

Named volumes created, managed by users

Bind Mount

created a directory and attached volume to it 



Even if we del container, data persists

Bind Mount is most likely to inject data from host to container,

Dev write code in host and it reflects in container

For preserving, volumes are needed.

Building docker img:

* FROM base image(ubuntu,mysql), we take base img from dockerhub

and can customize

FROM ubuntu:20.04

* LABELS like key value pairs((its like how we have tags on AWS) adds

metadata to an image.

* RUN exec cmds in a new layer and commit the results

Usually used for installing software or running scripts

RUN apt-get update && apt-get install -y python3

* ADD/COPY copy files or directories from build context into container’s

Filesystem copy . /app

adds files and folders into image, fetches files from remote

urls ADD archive.tar /app/

* CMD provides a default cmd to execute when a container is started

Runs when the container starts not during the image build.

CMD [“echo” , “hello”] echo – shell cmd, hello - argument

* ENTRYPOINT allows you to configure a container that will run as an

executable

* VOLUME create a mount point n marks it as holding externally mounted

volumes

* EXPOSE container listens to specific ports at runtime
* ENV sets the env variables
* USER sets the user name (UID)
* WORKDIR sets the working directory for any RUN/CMD?ENTRYPOINT
* ARG Defines a variable that users can pass at build-time
* ONBUILD adds to the image a trigger instruction to be executed at a

later time

Dockerfile

FROM ubuntu:latest

LABEL "author"="chandana"

RUN apt update && apt install git -y

RUN apt install apache2 -y

CMD ["/usr/sbin/apache2ctl", "-D", "FORGROUND"]

EXPOSE 80

WORKDIR /var/www/html

#ADD nano.tar.gz /var/www/html

COPY nano.tar.gz /var/www/html

🡪docker build -t nanoimg .

Build initiates process of building docker img from a dockerfile(contains instructions to create img)

-t tag used to tag the img with a name

Dot(.) build context which is the directory containing dockerfile

->docker images

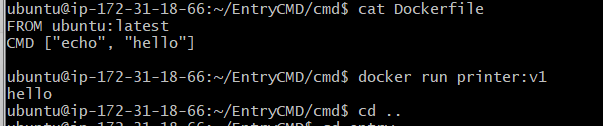
-> docker run -d --name nanowebsite -p 9080:80 nanoimg

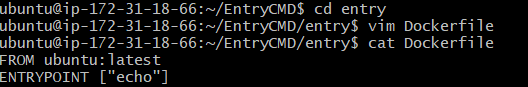
-> docker ps

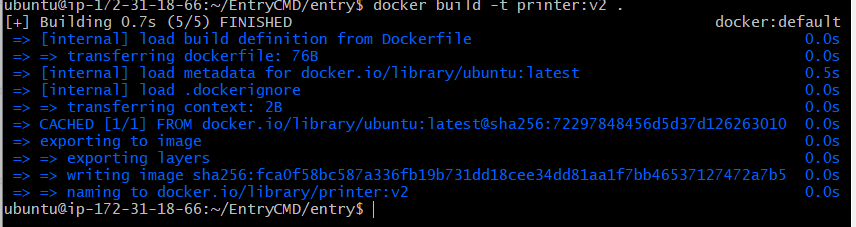
-> docker ps -a

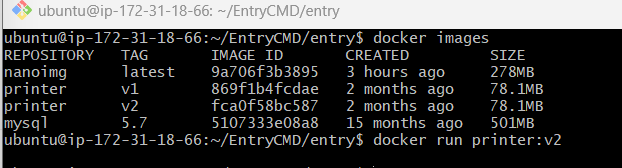
->docker login username \*\*\* password \*\*\* (login to the repository – Azure ACR/AWS ECR/Quay/dockerhub)

->Docker push chanrepo/nanoimg (repo/img name)



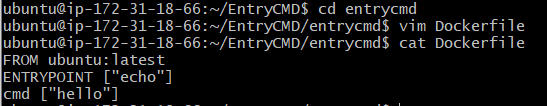






User has to pass the argument “hello”





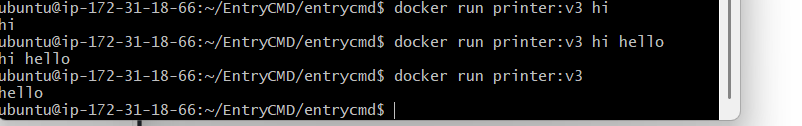
In ENTRYPOINT we have command, in cmd we have argument

You may have some script and want to initialize you want to run it first and you give it in ENTRYPOINT

Then your actual cmd starts the container process which is in CMD

If entrypoint and cmd are used together, cmd will have default argument which user can override

We can override the argument



Docker compose:

Is used for running multi-container applications.

Key to unlock streamlined n efficient development n deployment experience.

Manages service, volumes, networks in a single YAML config file, then with a single cmd you create and start all the services from your config file.

Works in all environments – production, staging, dev,testing as well as CI workflows.

Cmds for managing whole lifecycle of your application:

Start,stop,rebuild services

View the status of running services

Stream the log output of running services

Run a one-off cmd on a service

Containerization

Deployments are done via images

Same container images can be used across environment

Reusable & repeatable

Steps:

Setup our stack services

Find right base images from dockerhub

Write dockerfile to customize images

Write docker-compose.yml file to run multi containers

Test it & host images on dockerhub

Tools used(you get this from developers in ur team who write the application code)

MySQL (Database SVC) => 8.0.33

Memcache (DB Caching SVC) => 1.6

RabbitMQ (Broker/Queue SVC) => 4.0

JDK => JDK 21

MAVEN => Maven 3.9.9

Tomcat (Application SVC) => 10, jdk21

Nginx (Web SVC) => 1.27

mysql:8.0.33

memcache:latest

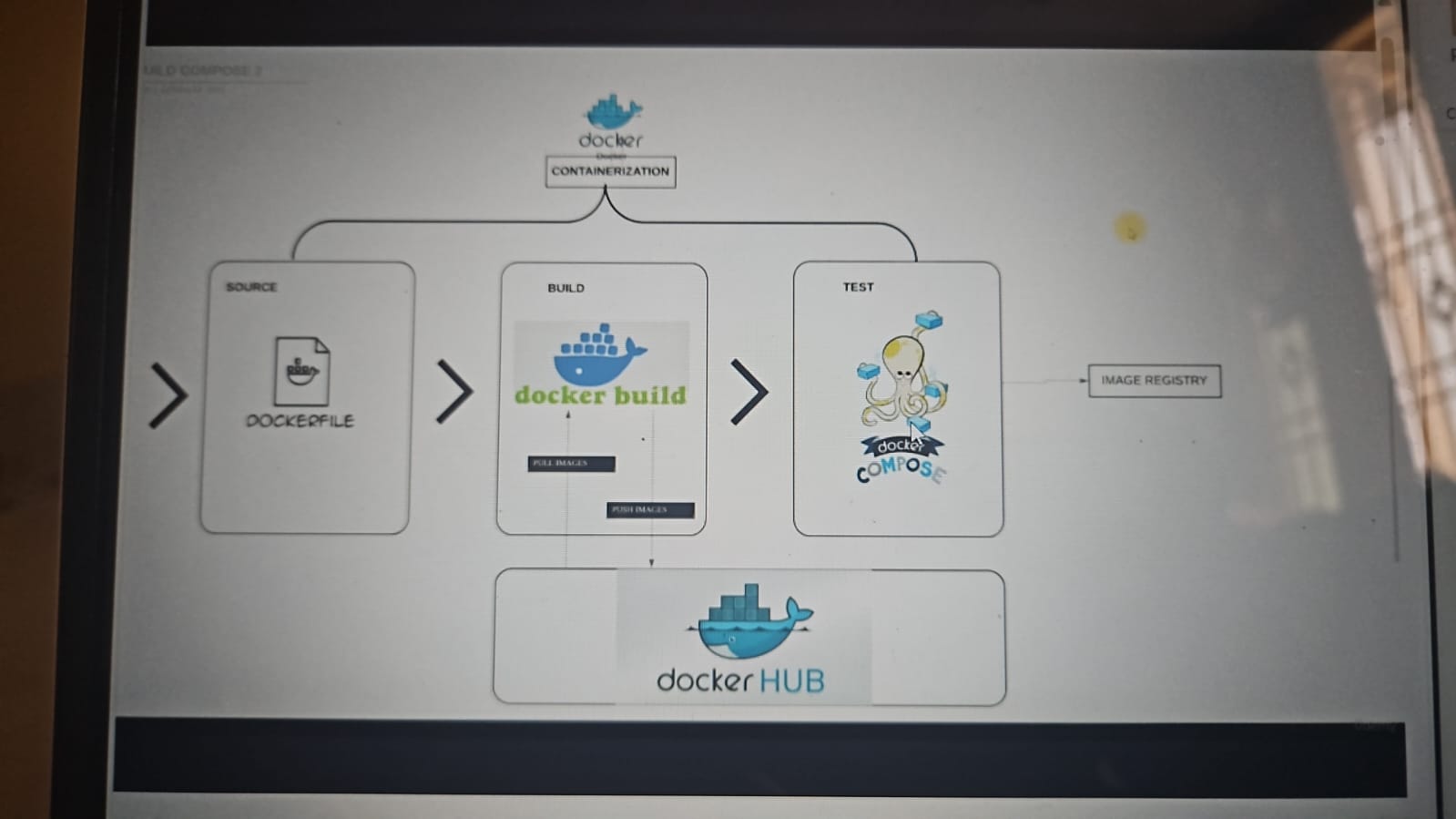
rabbitmq:latest

maven:3.9.9-eclipse-temurin-21-jammy

tomcat:10-jdk21

nginx:latest

HANSDON:



In VSCode and git clone from hkhcoder>>branch(containers)>>dockerfile

Create 3 dockerfiles for app, web, db (instructions about which img with which version all the info is taken from developer), we just write dockerfiles which instructions to build 3 docker images and run containers

We use docker-compose to write multi stage dockerfile and run multi containers

5 Containers::

app => dockerfile

db => dockerfile

Dockerfile

ContainerName: vprodb

Ports: 3306:3306 #(host port: container port)

Password: MYSQL\_ROOT\_PASSWORD

Volume: /var/lib/mysql

Web => dockerfile

Memcache => dockerhub img

Image

ContainerName: vprocache01

Ports: 11211:11211

Rabbitmq => dockerhub img

Image

ContainerName: vpromq01

Ports: 5672:5672

Variables:

User: guest

Password: guest

Steps:

Steps to setup our stack services

Find right base image from dockerhub

Write dockerfile to customize images

Write docker-compose.yml file to run multi containers

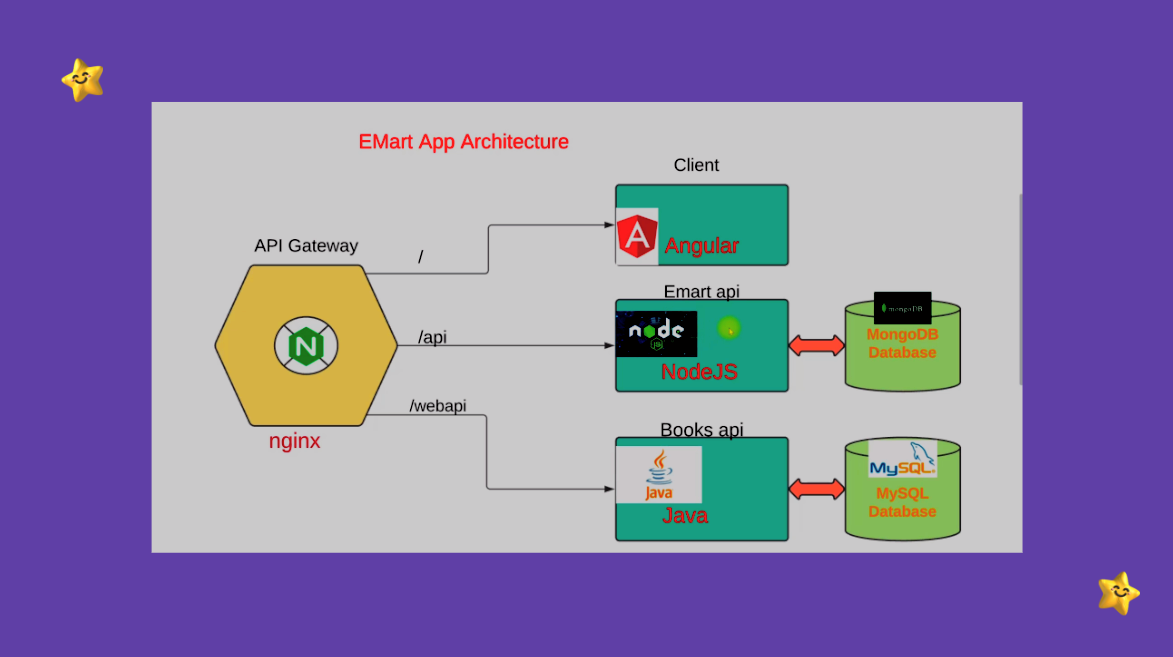
Test it & host images on dockerhub

MICROSERVICE APPLICATION

EMART APP ARCHITECTURE

Write Dockerfiles

Write Docker compose file based on dockerfile requirement



Ec2 instance and with docker build steps in advanced section

Connect to instance and

git clone https://github.com/devopshydclub/emartapp.git

cd emartapp/

ls (check if docker-compose.yml is avail from github/vscode where all dockerfiles and docker compose file is avail)

docker-compose build

docker images

docker-compose up

take public ip addr of instance and access in browser with 8080, you will see emart website>> register>>login

try booksmart

docker-compose up -d (starts all containers again in background if you give -d)

docker ps

docker stop

docker-compose down(remove all containers)

git pull (whenever developer made changes, we just pull the changes )

docker-compose build (built from modified stage)