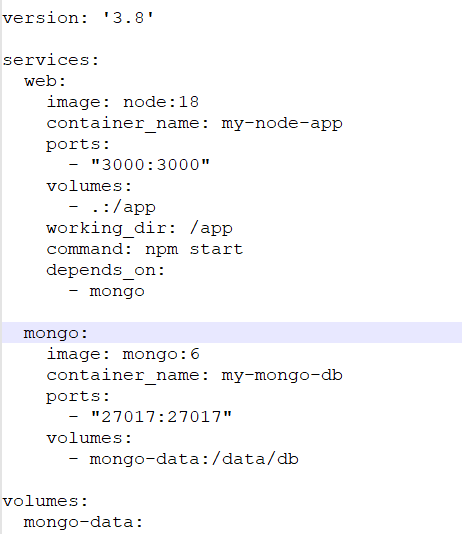
DOCKER

**1. Docker-compose :**   
with docker compose we can create a config file in yaml format and put together different services.  
we can run multiple containers with docker compose

commands: docker compose -f compose.yaml up -d  
 docker compose -f compose.yaml down  
  


**2. what is Dockefile?**  
A Dockerfile is set of instructions which are used to construct a Docker Image.  
These instructions are called directives.  
  
**Dockerfile directive:**

**FROM**: start a new build stage and sets the base image. usually must be the first directive in the Dockerfile (except ARG can be placed before FROM).

**ENV**: set environment variables. These can be refefenced in the Dockerfile itself and are visible

to the container at runtime.

**RUN**: creates a new layer on top of the previous layer by running a command inside that

new layer and commiting the changes.

**CMD**: specify a default command used to run a container at execution time.

**EXPOSE**: documents which port(s) are intended to published when running a container

**WORKDIR**: sets the current working directorHEALTHy for subsequent directives such as ADD, COPY, CMD, ENTRYPOINT, etc.  
Can be used multiple time to change the directories throught the Dockerfile.  
you can also use relative path, which sets the new working directory relative to the previous working direcotry.

**COPY**: copy files from the local machine to the image.

**ADD**: similar to COPY, but can also pull files using a URL and extract an archive into loose files in the image.

**STOPSIGNAL**: specify the signal that will be used to stop the container.

**HEALTHCHECK**: specify a command to run in order to perform a custom health check to verify that the container is working properly.

**3. Difference b/w ENTRYPOINT & CMD?**

The ENTRYPOINT specifies a command that will always be executed when the container starts.

The CMD specifies arguments that will be fed to the ENTRYPOINT.

**4. How to copy files from a stopped container?**

docker cp $container name:/path/in/container:/path/on/host------------(To copy from container to local)

docker cp /path/on/host $container name:/path/in/container----------- (from local machine to container)

**5. Multi stage docker file:**   
We can write a multi stage docker file by using multiple From statements in a single DockerFile where it will build multiple intermediate images with a single docker file , with a specific purpose.  
This helps in reducing image size by copying necessary artifacts by ignoring unwanted build dependencies.

**Ex:**

|  |  |
| --- | --- |
| Write a docker file for react JS application  # Stage 1: Build the React app  FROM node:18 AS build    WORKDIR /app    COPY package\*.json ./  RUN npm install    COPY . .    RUN npm run build    # Stage 2: Serve the app with nginx  FROM nginx:alpine    # Copy built app from previous stage  COPY --from=build /app/build /usr/share/nginx/html    # Copy custom nginx config (optional)  # COPY nginx.conf /etc/nginx/nginx.conf    EXPOSE 80    CMD ["nginx", "-g", "daemon off;"] | # Stage 1: Build the Go application FROM golang:1.19 AS builder # Set the working directory WORKDIR /app # Copy the Go module files and download dependencies COPY go.mod go.sum ./ RUN go mod download # Copy the source code COPY . . # Build the application RUN CGO\_ENABLED=0 GOOS=linux go build -o myapp . # Stage 2: Create the final image FROM alpine:latest # Set the working directory WORKDIR /root/ # Copy the built binary from the builder stage COPY --from=builder /app/myapp . # Expose the application port EXPOSE 8080 # Command to run the application CMD ["./myapp"] |

What is an entry point in docker ?

The ENTRYPOINT specifies a command that will always be executed when the container starts.  
ENTRYPOINT ["node", "app.js"]

Docker Engine Architecture  
Docker Engine consists of three primary parts that work together to build, ship, and run containers:

* **Docker Daemon (dockerd)**  
  The background service that manages images, containers, networks, and volumes on your host.
* **REST API**  
  A set of HTTP endpoints that expose the daemon’s functionality to clients and automation tools.
* **Docker CLI (docker)**  
  The command-line interface that sends commands to the REST API.

Docker Objects  
Docker Engine manages four primary resource types:

|  |  |
| --- | --- |
| Object | Description |
| **Images** | Read-only templates composed of layered filesystem snapshots and metadata. |
| **Containers** | Instances of images providing a writable layer and running processes. |
| **Networks** | Virtual networks enabling container-to-container and external communication. |
| **Volumes** | Persistent storage volumes decoupled from container lifecycles. |

**Container Creation Flow**

When you run **docker run**, Docker follows a series of steps:

1. **CLI to API**  
   The Docker CLI translates your command into a REST API call.
2. **Daemon Processing**  
   The daemon checks for the image locally or pulls it from the registry.
3. **containerd**  
   Converts the image into an OCI bundle.
4. **containerd-shim**  
   Hands off the bundle to runC and monitors the container’s lifecycle.
5. **runC**  
   Uses kernel namespaces and cgroups to spawn and isolate the container.

DOCKER COMMANDS

docker pull <image-name> ------ To pull required image

docker run nginx -----------start a container

docker run -d <image-name> ------- To run the container in detached mode (runs in background)

docker attach <containerID> -------- To bring it to foreground

docker ps ---------lists running containers

docker stop containerID ------- to stop a container

docker ps -a --------to list running and stopped containers

docker rm containerID --------to remove a container

docker rm -f <containerID> ----- To remove container forcefully

docker container prune -------to remove all stopped/exited containers

docker images ------- to list images

docker rmi <image-name> ---------to remove images

docker exec -it <containerID> /bin/bash

docker run redis:5.0 -------- to sepcify version

docker run jenkins:2.60.3

docker inspect <containerid> ------- To inspect the container

docker run -p 8080:8080 jenkins:2.60.3 ---------- To access it from outside,To expose to external port

docker run -p 8080:8080 -v /root/my-jenkins-data:/var/jenkins\_home jenkins:2.60.3

docker stop <containerID> ----- to stop a container

docker start <containerID> ------ to start a container

docker restart <containerID> ---- to restart a container

Managing Network

docker network ls

docker network create <network-name> ----- by default it create a bridge network

docker network create --driver bridge <network-name>

docker network create --driver overlay <network-name>

docker network inspect <network-name>

docekr network rm <network-name>   
  
Lets say I have a local docker desktop in my local machine I have a docker image which serve some website how do i expose it to the ouside world so that anyone can access it.

docker run -d -p 8080:80 my-website-image

DockerFiles:  
  
FROM ubuntu  
  
RUN apt-get update && apt-get -y install python  
RUN pip install flask flask-mysql  
COPY . /opt/source-code  
ENTRYPOINT ["flask", "run"]

------------------------------------------------------------------------------  
  
FROM ubuntu

RUN apt-get update

RUN apt-get install -y nginx

RUN mkdir /home/config/

RUN touch /home/config/db.props

COPY . /var/www/html/

Expose 80

CMD /usr/sbin/nginx -g "daemon off;"

------------------------------------------------------------------------------

**Cgroups:**  
Containers by default can use as much resource as they require, which may lead to resource exhaustion on the host. Docker utilizes ***Linux control groups (cgroups)*** to restrict the hardware resources available to each container, ensuring efficient resource management.  
EX:  
docker run --cpus=0.5 ubuntu  
docker run --memory=100m ubuntu

**Namespaces:**  
Docker utilizes namespaces to isolate various resources like

* Workspaces
* Process IDs
* Network interfaces
* Inter-process communication (IPC)
* Filesystem mounts
* Unix time-sharing systems

This isolation provides containers with the appearance of independent systems while sharing hardware resources with the host.  
This means that each container has its own isolated namespace for processes, network, mount points, and more.  
  
network type:

* The ***bridge network*** is a private, internal network created by Docker on the host. Containers connected to this network receive an internal IP address—typically in the 172.17.x.x range—and can communicate with each other using these addresses. To allow external access to a container, map its ports to ports on the Docker host.
* The ***host network*** uses the host’s network stack directly, eliminating network isolation between the container and the Docker host. For example, running a web server container on port 5000 will make the server immediately accessible on the host’s port 5000 without any additional port mapping. However, this also means that multiple containers cannot simultaneously use the same port on the host.
* The ***none network*** disconnects the container from any networking, ensuring complete isolation from external networks and other containers.

Docker networking uses network namespaces to create separate network environments for each container   
  
  
Docker Swarm?

Docker Swarm allows you to combine multiple Docker hosts into a single cluster. Within the swarm cluster, the manager node orchestrates the distribution of your services (or application instances) across different hosts, ensuring high availability and effective load balancing. To set up Docker Swarm, ensure you have several hosts with Docker installed. Designate one host as the manager (sometimes known as the master or swarm manager) and the others as worker nodes.  
  
  
**You have a Node.js app on port 8081 — how would you write a Dockerfile for it?**

**Answer:**

FROM node:18  
WORKDIR /app  
COPY . .  
RUN npm install  
EXPOSE 8081  
CMD ["node", "index.js"]

**Write a Dockerfile for a ReactJS application.**

**Answer:**

FROM node:18 as build  
WORKDIR /app  
COPY . .  
RUN npm install  
RUN npm run build  
  
FROM nginx:alpine  
COPY --from=build /app/build /usr/share/nginx/html  
EXPOSE 80  
CMD ["nginx", "-g", "daemon off;"]  
  
-g stands for "global directive"  
  
**Why Docker?**  
With Docker, every component runs inside its own container with dedicated dependencies and libraries, all on the same virtual machine and operating system while remaining isolated from each other.

**What Are Containers?**

Containers are actual running instances of the docker images with isolated environments that operate independently from one another. Each container maintains its own processes, network interfaces, and mounts—similar to virtual machines—but all containers share the host OS kernel.

**Difference b/w containers vs Virtual machine?**  
**Containers:**   
containers get isolation at process level , They have shared OS   
Containers are lightweight   
Can start in seconds   
minimal overhead   
  
**Virtual machine:**  
Complete isolation with full OS  
Higher resource usage   
Boot time in minutes   
Needs higher disk space due to multiple OS   
  
**Docker image ?**  
Docker image is a pre-built package that has all the dependencies, libraries that are required to build an application and run a container out of it.  
  
Dockerfile:

|  |
| --- |
| FROM ubuntu  RUN apt-get update && apt-get install –y python python-pip  RUN pip install flask flask-mysql COPY . /opt/source-code  ENTRYPOINT [“sh”, “-c”, “FLASK\_APP=/opt/source -code/app.py flask run” |

Write a docker file to run nginx

|  |
| --- |
| FROM ubuntu:latest  RUN apt-get update -y && \  apt-get install -y nginx && \  apt-get clean  COPY nginx.conf /etc/nginx/nginx.conf  EXPOSE 80 CMD ["nginx", "-g", "daemon off;"] |

**Docker file with node18 , set workfdir , copy package.json file to containers and install npm , expose it to 3000 and start the container?**

|  |
| --- |
| FROM node:18  WORKDIR /app  COPY package\*.json /opt/source-code  RUN npm install  COPY . .  EXPOSE 3000  CMD ["npm", "start"] |

FROM ubi8

RUN dnf install -y httpd

COPY index.html /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]