what is docker engine vs docker daemon vs docker client

Docker Engine encompasses both the Docker Daemon and the Docker Client. The Docker Daemon is the background service responsible for managing containers, while the Docker Client is the command-line tool used by users to interact with the Docker Daemon and perform container-related operations.
Docker Engine is the core component of the Docker platform responsible for building, running, and managing Docker containers.
The Docker Daemon is a background service (process) that runs on the host operating system and manages Docker containers.
The Docker Client is a command-line interface (CLI) tool used by users to interact with the Docker Daemon

what is docker api?

The Docker API is a RESTful HTTP API provided by the Docker daemon (**`dockerd`**) that allows external clients to interact with Docker and perform various container-related operations programmatically. It enables users to manage Docker containers, images, networks, volumes, and other resources through HTTP requests.

what is docker.sock?

The **`docker.sock`** file is a Unix socket used by the Docker daemon to communicate with client programs. It provides a communication channel between Docker client commands and the Docker daemon. When the Docker daemon (**`dockerd`**) starts up, it creates a Unix socket file named **`docker.sock`** by default.

When you run Docker commands such as **`docker ps`**, **`docker run`**, or **`docker build`**, the Docker client communicates with the Docker daemon through the **`docker.sock`** Unix socket.

The **`docker.sock`** file is typically located at **`/var/run/docker.sock`**. It is created when the Docker daemon starts up and allows Docker client commands to interact with the Docker API provided by the daemon.

unix socket- allows processes on the same host to communicate with each other using inter-process communication (IPC). Unix sockets are represented as special files within the Unix filesystem. They are typically created in the **`/tmp`** directory or in a system-specific directory such as **`/var/run`**.

Dockerfile to install tomcat on alpine

```
FROM alpine:latest

RUN apk update && \

apk add openjdk11

RUN mkdir /opt/tomcat

WORKDIR /opt/tomcat

ADD https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.87/bin/apache-tomcat-9.0.87.tar.gz .

RUN tar -xvzf apache-tomcat-9.0.87.tar.gz

RUN mv apache-tomcat-9.0.87/* /opt/tomcat

EXPOSE 8080

CMD ["/opt/tomcat/bin/catalina.sh","run"]
```

Dockerfile to install python on alpine

```
From alpine:latest

RUN apk update && apk add --no-cache bash && \

apk add --no-cache python3 py3-pip && \

apk python3 --version && \

apk pip3 --version


WORKDIR /workspace1

COPY . /workspace1

CMD ["bash"]
```

CMD- The **`CMD`** instruction in a Dockerfile specifies the command that should be executed by default when a container based on that image is started. It is not executed during the build process of the image.

the **`CMD`** instruction defines the default command that will be run when you start a container from that image, unless overridden by specifying a different command at runtime,you can override the cmd command specified in dockerfile while executing run command.

docker run my_image python another_script.py

ENTRYPOINT:

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

ENTRYPOINT [“yum”,”install”,”-y”,”git”]

docker run image_name #installs the git

docker run image_name http #installs git and httpd

Here argument specified in run command appended to the entrypoint command.whereas in cmd the arguments passed in run command overwrites the command in CMD

Using CMD and ENTRYPOINT:

ENTRYPOINT [“yum”,”install”,”-y”,”git”] #executes compulsory

CMD [“git’] #optional ,executes only when no args are specified in run command

docker run image_name httpd #installs httpd

docker run image_name #installs git as no arg specified

If there are multiple entrypoints in dockerfile,it will consider latest entry only.same for cmd also.

Package managers:

A package manager is a software tool used to automate the process of installing, updating, configuring, and removing software packages on a computer system.

Each Docker image is typically based on a specific operating system (OS) distribution, and the package manager available within the Docker image depends on the OS distribution used as its base.

For Ubuntu: apt-advanced package tool

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y nginx
```

For alpine:

```
FROM alpine:latest
```

RUN apk update && apk add nginx

For Centos

Starting from CentOS 8, the default package manager is **`dnf`**, although **`yum`** is still available. If you're using CentOS 8 or later, you can substitute **`yum`** with **`dnf`** in the examples provided above

FROM centos:latest

RUN yum update -y && yum install -y nginx

Additionally, starting from CentOS 8, the default package manager is **`dnf`**, although **`yum`** is still available. If you're using CentOS 8 or later, you can substitute **`yum`** with **`dnf`** in the examples provided above

CURL:

`curl` is a command-line tool and library for transferring data with URLs. **`curl`** is primarily used for fetching content from a URL and displaying it in the terminal, or for downloading files from the internet.

curl -O https://example.com/file.txt

- **`curl`**: Invokes the **`curl`** command.
- **`-O`**: (uppercase O) Tells **`curl`** to write the output to a local file with the same name as the remote file. In this case, **`file.txt`** will be saved to the current directory.
- `https://example.com/file.txt`: Specifies the URL of the file you want to download.

NETWORKS IN DOCKER:

By default, containers have their own IP addresses separate from the host system. This is one of the key features of containerization, which allows containers to run isolated from each other and from the host system.

1)Bridge network: When you start a container without specifying a network, it is attached to the default bridge network.

- **Virtual Bridge:** Docker creates a virtual Ethernet bridge on the host system called **`docker0`**. This bridge acts like a virtual switch that connects the host system's physical network interface to the containers' virtual network interfaces.
- **Container Interface:** When you start a container, Docker creates a virtual Ethernet interface (**`veth`**) pair. One end of the pair is attached to the container's network namespace, and the other end is attached to the **`docker0`** bridge on the host system.

To create a custom bridge network in Docker, you can use the **`docker network create`** command.

**docker network create **

**--driver bridge **

**--subnet 172.20.0.0/16 **

**--gateway 172.20.0.1 **

my_custom_bridge

- **`--driver bridge`**: Specifies the network driver to use. In this case, we're using the built-in bridge driver, which creates a bridge network.
- **`--subnet`**: Specifies the subnet range for the network. In this example, we're using the **`172.20.0.0/16`** subnet.
- **`--gateway`**: Specifies the gateway IP address for the network. Here, we're using **`172.20.0.1`** as the gateway.
- **`my_custom_bridge`**: Specifies the name of the custom bridge network.

docker run -d --name container1 nginx:latest

In Docker, when you specify an image without specifying the operating system explicitly, it defaults to using the base image specified by the Dockerfile or the image repository.

The base image **`nginx:latest`** typically uses a Linux distribution as its base operating system. In the case of the Nginx official image, it's often based on Debian or Alpine Linux

Here each container assigned with unique ip address within the same subnet.

To install ping in debain:

apt update

apt-get install putils ping

ping ip_address_of_host

Docker containers can be created on physical servers or vm's.containers have light weight operating system,if they required any resources ,they get it from host operating system(i.e,resources of host os are shared with containers)

Docker Volumes

When you create a volume,a logical partition or folder is created on host system.

docker volume create vol_name

docker volume ls

docker volume inspect vol_name #gives the details of volume

docker volume rm vol_name1 vol_name #can remove multiple volumes at a time

Through volume we can create from cloud storage too.To delete the volume,stop the containers that uses the volume

docker container prune #removes all the containers that are not running

Bridge network

docker run –d –name demo1 –p 8080:80 httpd

`httpd`: This is the name of the Docker image used to create the container. In this case, it's the official Apache HTTP Server image.

`-p [host_port]:[container_port]`

- **`[host_port]`**: This is the port on the host system that you want to map to the container's port.
- **`[container_port]`**: This is the port inside the Docker container that you want to expose to the host system

`Here,-p 8080:80` means that port 8080 on the host system will be mapped to port 80 inside the Docker container. Any traffic sent to port 8080 on the host will be forwarded to port 80 inside the container. now we can access the apache server through ip address of host followed by port [192.89.78.45:8080] For every container unique port should be given.Every container is accessed through docker host ip address only.

Docker engine assigns unique ip address to each docker container. These IP addresses used for communication docker containers within the docker systems.These are not directly accessible from outside the host system.To make services running inside Docker containers accessible from outside the host system, you typically use port mapping (port forwarding) or expose specific ports from the container to the host system. This allows incoming network traffic on specific ports of the host system to be forwarded to the corresponding ports of the containers.

docker network inspect bridge

Host network:

docker run –d –name demo1 --net host httpd

docker run –d –name demo1 --network host httpd

Through host network apache server can be directly accessed through host ip address without port mapping

None network:

docker run –d –name demo1 --net none httpd

This container won't communicate with outside the host and internally too.

Sample dockerfile to create ubuntu image and run bash shell

From ubuntu:latest

RUN apt update

CMD ["bash"]

```
veena@linux:~/images$ sudo docker run --name ubuntu1 ubuntu-bash
veena@linux:~/images$ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.45/containers/json": dia
l unix /var/run/docker.sock: connect: permission denied
veena@linux:~/images$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS                                   NAMES
4a8ee77787fb   httpd     "httpd-foreground"   22 minutes ago   Up 22 minutes                                           demo3
af4397396f42   httpd     "httpd-foreground"   2 hours ago      Up 2 hours      0.0.0.0:8081->80/tcp, :::8081->80/tcp   demo2
veena@linux:~/images$ sudo docker ps -a
CONTAINER ID   IMAGE         COMMAND              CREATED          STATUS                      PORTS                                   NAMES
cb17fc7a1503   ubuntu-bash   "bash"               20 seconds ago   Exited (0) 19 seconds ago                                           ubuntu1
4a8ee77787fb   httpd         "httpd-foreground"   22 minutes ago   Up 22 minutes                                                       demo3
af4397396f42   httpd         "httpd-foreground"   2 hours ago      Up 2 hours                  0.0.0.0:8081->80/tcp, :::8081->80/tcp   demo2
06085645d575   httpd         "httpd-foreground"   2 hours ago      Created                                                             demo1
veena@linux:~/images$ docker remove ubuntu1
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Delete "http://%2Fvar%2Frun%2Fdocker.sock/v1.45/containers/ubuntu1
": dial unix /var/run/docker.sock: connect: permission denied
veena@linux:~/images$ sudo docker remove ubuntu1
ubuntu1
veena@linux:~/images$ sudo docker run --name ubuntu1 --it ubuntu-bash
unknown flag: --it
See 'docker run --help'.
veena@linux:~/images$ sudo docker run --name ubuntu1 -it ubuntu-bash
root@7155f2d866fc:/# exit
exit
veena@linux:~/images$ vi Dockerfile
```

sudo docker build -it ubuntu-bash .

-it interactive, **.** specifies dockerfile in current directory,can specify the path if dockerfile not present in current dir,if –it not used container exited automatically after executing bash command specified in CMD

sudo docker images –q #lists all the images

docker images # lists all the images in verbose manner

docker rmi $(docker images -q) #removes all the images

docker ps –a #lists all the container in verbose manner

docker ps -aq #lists all the container

docker rm -f $(docker ps -aq) #forcely remove all the container including running conatiners

Installing jenkins on alpine:

To installl jenkins we need to add jenkins repo to package manager's configuration, we're essentially telling the package manager where it can find Jenkins packages for installation.

Why should we add jenkins repo to apk

Adding a repository is typically done when the software you want to install is not available in the default repositories provided by the distribution. In the case of Jenkins, it's often not available in the default repositories of many Linux distributions, including Alpine Linux. Therefore, you need to add the Jenkins repository to access its packages.

On the other hand, Java is often available in the default repositories of many Linux distributions, including Alpine Linux. This means you can usually install Java directly using the package manager (**apk** in the case of Alpine Linux) without needing to add an additional repository.

sudo apk update #update package index

sudo apk add openjdk11 #install java

Alpine Linux does not have Jenkins in its default repositories. You'll need to add the Jenkins repository to your package manager:

wget -q -O /etc/apk/keys/jenkins.io.key https://pkg.jenkins.io/alpine/jenkins.io.key

echo "https://pkg.jenkins.io/alpine" | sudo tee -a /etc/apk/repositories

Now that the repository is added, you can install Jenkins:

sudo apk update

sudo apk add jenkins

Dockerfile to install jenkins on ubuntu

```
# Use Ubuntu as base image

FROM ubuntu:20.04

# Update package index and install necessary packages
RUN apt-get update && \
   apt-get install -y --no-install-recommends \
   wget \
   gnupg \
   ca-certificates \
   && rm -rf /var/lib/apt/lists/*

# Install Java (OpenJDK)
RUN apt-get update && \
   apt-get install -y --no-install-recommends \
   openjdk-11-jdk \
   && rm -rf /var/lib/apt/lists/*

# Add Jenkins repository and GPG key
RUN wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | apt-key add - && \
   echo "deb https://pkg.jenkins.io/debian-stable binary/" >> /etc/apt/sources.list.d/jenkins.list

# Update package index and install Jenkins
RUN apt-get update && \
apt-get install -y --no-install-recommends \
jenkins \
   && rm -rf /var/lib/apt/lists
# Expose Jenkins port

EXPOSE 8080

# Set command to start Jenkins

CMD ["java", "-jar", "/usr/share/jenkins/jenkins.war"]
```

Docker compose

Sample yaml file:

```
version: '3'   #specifing the version of docker compose
services:                          #in services specify the containers to run
  my-redis-server:                 #container name
    image: redis                   #image used by the container
    expose:                        #port number to expose
      - '6379'
    volumes:
      - ./redis-data:/data

  visitor-app:
    build:                         #instead of using created image,you can specify the dockerfile here
      dockerfile: Dockerfile       #give the name of dockerfile (dockerfile  can be given with diff name too)
      context: ./                  #specify the path of dockerfile
    ports:
      - '9999:8888'
    depends_on:                    #specify the container that needs to be run before this
      - my-redis-server
    volumes:
      - ./src:/usr/visitorapp/src
```

environment:

  SERVERPORT: 8888