1.What is the difference between Physical Servers, Virtual Machines and Containers and Oschestrations?

A Physical server is a hardware server with a motherboard, CPU, memory, and IO controllers that we can touch and feel. It is a bare-metal server as its hardware is used directly by a single OS instead of a virtualization platform.

A Virtual machine is a virtual version of a physical server that works like an actual computer. It has its own CPU, RAM, and storage, and emulates the functionalities of a dedicated server. One physical server can host multiple virtual servers, with its own virtual infrastructure.

A container is an isolated environment for our code. This means that a container has no knowledge of our operating system or files. It runs on the environment provided to us by Docker Desktop. This is why a container usually has everything your code needs, like libraries and dependencies, to run.

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux containers and hosts can benefit from container orchestration. Container orchestration can be used in any environment where you use containers. It can help you deploy the same application across different environments without redesigning it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

2. What is hypervisor?

A hypervisor, also known as a virtual machine monitor (VMM), is a software or firmware layer that enables the creation and management of virtual machines (VMs). Its primary function is to abstract the underlying.
physical hardware and provide an environment for multiple VMs to run concurrently on a single physical machine.

How it works:
Hypervisor sits between physical hardware and VMs. It abstracts and manages hardware resources for VMs. Intercepts and maps VMs' hardware access requests. Provides isolation between VMs. Manages resource allocation and optimization.

Advantages and disadvantages of hypervisor?
Advantages:
Server consolidation and resource utilization.
Isolation for security and stability.
Hardware independence.
Live migration for flexibility.

Disadvantages:
Overhead and potential impact on performance.
Resource allocation challenges.
Complexity in setup and management.
Single point of failure if the hypervisor crashes.

What is Docker ? and why we need Docker ?

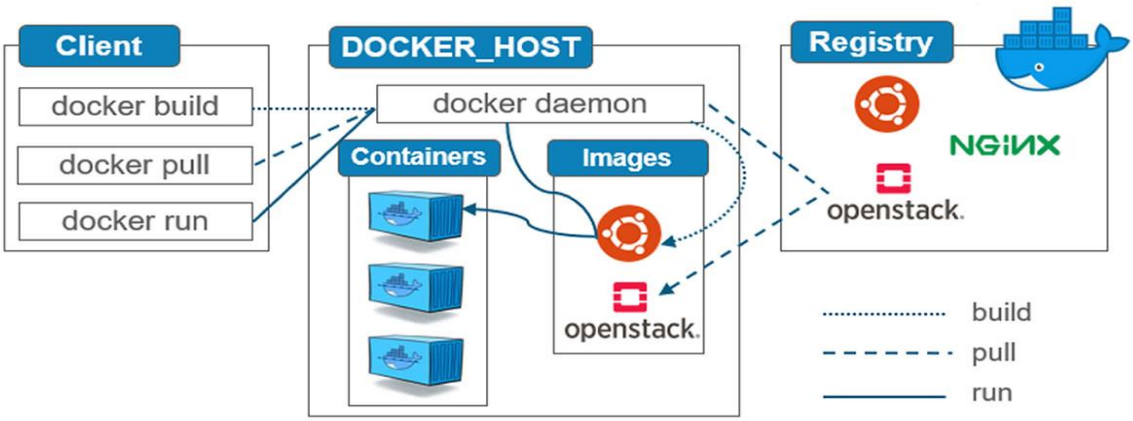
Docker is a tool that helps in developing, shipping, and running your applications on containers and enables you to separate your applications from your infrastructure so you can deliver software quickly. It provides the ability to package and run an application in an isolated environment i.e; containers. The isolation and security allows you to run many containers simultaneously on a given host.

Why we need Docker :

Docker takes care of all the dependencies and configuration of any software and packages them into a Docker run container. This makes your software portable and provides the same environment to your software as it was when

you created it and which ensures that it will run on any computer despite their OS types, system configuration, and anything.

4) Docker Architecture:



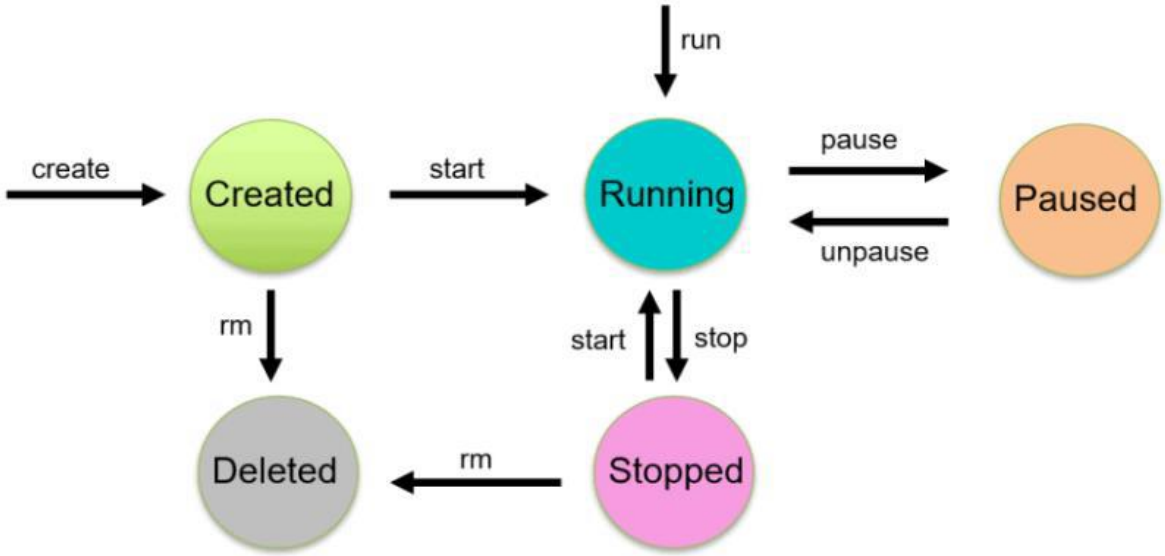
Docker Daemon: Background service managing Docker objects.
Docker Client: CLI tool for interacting with the Docker daemon.
Docker Images: Read-only templates containing application and dependencies.
Docker Containers: Runnable instances of Docker images.
Docker Registry: Repository storing Docker images.

How it works :

Docker image is created using a Dockerfile.
Docker client requests container creation from the Docker daemon.
Docker daemon pulls image from a registry if needed.
Isolated environment is set up for the container.
Container starts and runs the application.
Multiple containers can run concurrently on a host.
Containers communicate through networks and share data using volumes.
Docker daemon manages containers for easy control and scaling.

5) what is Docker Lifecycle ? Container lifecycle?

Docker Container Lifecycle:
Create: Container is created from an image using docker run.
Start: Container is started with the application running.
Execute: Application or services within the container perform their tasks.
Stop: Container is gracefully stopped using docker stop.
Restart: Stopped container can be restarted with docker start.
Remove: Stopped container is permanently deleted with docker rm.
Throughout the lifecycle, containers can be managed, monitored, and orchestrated using Docker commands and tools.



6). what is Docker file? what are the layers in docker file ?

A Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image, commands to install dependencies, copy files, set environment variables, expose ports, and define the entry point or command to run when a container is started. Docker files allow developers to automate the creation of containerized applications and ensure consistent builds.

Layers in a Dockerfile:
Base Image Layer: Specifies the base image for the Docker image.
Package Installation Layer: Installs packages and dependencies.
Application Configuration Layer: Copies configuration files or sets environment variables.
Source Code or Application Layer: Includes the application code or binaries.
Each layer represents a step in the image creation process and allows for efficient storage and sharing of Docker images.

7). what are the most comman instruction used in dockerfile?

FROM: Specifies the base image for the Docker image.
RUN: Executes commands during the image build process.
COPY: Copies files and directories from the build context to the image.
ADD: Similar to COPY, but allows additional features like extracting
archives from URLs.
ENV: Sets environment variables in the image.
WORKDIR: Sets the working directory for subsequent instructions.
EXPOSE: Documents the network ports the container listens on at runtime.
CMD: Specifies the default command to run when a container is started.
ENTRYPOINT: Configures the primary executable for the container.
VOLUME: Creates a mount point for external volumes.

8) what is Docker Compose? What is the difference between compose and Docker file?

Dockerfile is used to build Docker images.
It defines the steps and instructions to create a single Docker image.

Docker Compose:
Docker Compose is used to manage multi-container applications. It defines a collection of services, networks, volumes, and configurations for an application stack.
It is written in a YAML file format and provides a higher-level abstraction.

9) what is Multistage docker file?
A multistage Docker file is a technique used to create more efficient and smaller Docker images by using multiple stages or phases during the build process. It allows you to build intermediate images to compile, build, or package dependencies and then copy only the necessary artifacts into the final image.

10) what is difference between alphine image and multistage docker build ?

Alpine Image:

Alpine is a lightweight Linux distribution commonly used as a base image in Docker.

Alpine images are small in size and designed for efficiency and performance.

They have a minimalistic nature and reduced attack surface.

Additional steps may be required to install specific dependencies or packages.

Multistage Docker Build:

Multistage builds allow creating multiple stages within a single Dockerfile.

Each stage can have its own base image and instructions.

It separates the build environment from the final runtime environment.

The build stage compiles code and installs dependencies.

The final stage uses a lightweight base image, like Alpine, for efficient runtime.

It helps reduce the size of the final image by discarding unnecessary artifacts and dependencies.

In summary, Alpine images are lightweight base images known for their small size, while multistage builds allow you to create more efficient and optimized Docker images by separating the build environment from the final runtime environment.

11.what is the difference between docker image and docker layer ?

A) Docker Image:
A Docker image is a read-only template that contains the instructions for creating a container.
It includes the application code, dependencies, runtime, and other files needed to run the application.
Docker images are built based on a Dockerfile using the docker build command.
Images are typically stored in a registry and can be pulled and run on Docker hosts to create containers.
B) Docker layer:
A Docker layer is a component of a Docker image.
Each instruction in a Dockerfile adds a new layer to the image.
Layers are created during the image build process and represent the changes made to the image at each step.
Layers are stored in a layered file system, and each layer is stored as a separate entity.
Layers are immutable, meaning they cannot be modified once created.
Docker uses layering to optimize image building, caching, and sharing. When building a new image, if the layers already exist, they can be reused from cache, improving the build speed and reducing storage usage.

12) types of volume? Why we need volume in Docker?

Types of volume:

Host Volume: Maps a directory or file from the host machine to a container.
Anonymous Volume: Automatically created and managed by Docker.
Named Volume: Created and named by the user for persistent data.
Bind Mount: Maps a directory or file from the host machine to a container with more control over mount options.
Need of volume: Docker volumes provide a reliable and flexible way to manage and persist data in containerized applications, enabling data sharing, separation, and integration with external systems. They are essential for building scalable, reliable, and data-driven Docker deployments.

13) what are the types of networks in Docker ?

Types of networks in Docker:
A) BRIDGE Network: Default network for communication between containers on the same Docker host.
B) HOST Network: Shares the network namespace with the host machine.
C) OVERLAY Network: Enables communication between containers across multiple Docker hosts or Swarm nodes.
D) MACVLAN Network: Assigns a unique MAC address to each container for direct connectivity to the physical network.
E) NONE Network: Disables networking for a container, providing complete isolation.

14). what is the default range of CIDR range of docker container?

The default CIDR range for containers in Docker is typically 172.17.0.0/16. This means that Docker assigns IP addresses to containers from the range of 172.17.0.0 to 172.17.255.255.
Each container gets a unique IP address within this range, allowing them to communicate with each other and the host machine using the bridge network.

15. how will you assign static ip for container ?

To assign a static IP address to a container in Docker, you can make use of a user-defined network and specify a specific IP address for the container within that network.
Create a user-defined network: docker network create -- subnet=<subnet> <network-name>
Start the container with a specific IP address within the created network: docker run --network=<network-name> --ip=<desired-ip> <imagename>

16. how will you change the default subnet id of container?
ANSWER A:
To change the default subnet ID of containers in Docker, you need to
modify the Docker daemon's configuration file and restart the Docker
service. Here's a general process to change the default subnet:
Edit the Docker daemon configuration file:
For Linux, open the /etc/docker/daemon.json file.
For Windows, open the C:\ProgramData\docker\config\daemon.json file.
Add or modify the bip (Bridge IP) parameter in the configuration file to
set the desired subnet ID. For example:
json
[Copy code
{
"bip": "192.168.100.0/24"
}
Replace 192.168.100.0/24 with the desired subnet ID in CIDR notation.
Save the changes to the configuration file.
Restart the Docker service to apply the new configuration:
For Linux, use the command: sudo systemctl restart docker

For Windows, use the command: Restart-Service docker
After restarting the Docker service with the updated configuration, the default subnet ID for containers will be changed to the specified value. New containers created will use the new subnet for IP address assignment.

ANSWER B:
To change the default subnet ID of containers in Docker:
Edit the Docker daemon configuration file (/etc/docker/daemon.json on Linux, C:\ProgramData\docker\config\daemon.json on Windows).
Add or modify the "bip" parameter to set the desired subnet ID.
Save the changes and restart the Docker service.