DOCKER

1. What is a Virtual Machine?

A Virtual Machine (VM) is an emulation of a computer system that runs on a hypervisor. It allows multiple operating systems (OS) to run on a single physical machine by virtualizing hardware components.

Key Features:

- Runs an entire OS.
- Requires a hypervisor (e.g., VMware, VirtualBox, Hyper-V).
- Provides full isolation between VMs.
- Consumes more resources (RAM, CPU, storage).

Advantages:

- Strong isolation and security.
- Ability to run different OS types on the same hardware.
- Useful for legacy applications.

Disadvantages:

- High resource consumption.
- Slower performance compared to containers.
- Complex setup and maintenance.

2. How Organizations Worked Before Container Platforms

Before containers, organizations relied heavily on Virtual Machines (VMs) for application deployment. This approach had several challenges:

Challenges:

1. **Resource Inefficiency**: Each VM required a separate OS, consuming significant system resources.
2. **Slow Deployment**: VMs had large footprints, making provisioning and scaling time-consuming.
3. **Compatibility Issues**: Applications behaved differently across environments, causing inconsistencies between development, testing, and production.
4. **High Maintenance Overhead**: Managing multiple VMs required extensive configurations, monitoring, and orchestration.

3. What is a Container?

A **container** is a lightweight, standalone, executable package that includes everything needed to run an application: code, runtime, libraries, and dependencies. Unlike VMs, containers share the host OS kernel but maintain isolation between applications.

Difference Between VM and Container

Feature	Virtual Machine	Container
OS Requirement	Each VM has a separate OS	Shares the host OS kernel
Resource Utilization	High	Lightweight
Startup Time	Slow (minutes)	Fast (seconds)
Isolation	Strong (separate OS)	Process-level isolation
Portability	Less portable	Highly portable

Advantages of Containers:

- **Faster Deployment**: Starts in seconds.
- **Less Overhead**: No need for separate OS instances.
- **Scalability**: Easily scales applications using orchestration tools.
- **Consistency**: Works the same across different environments.

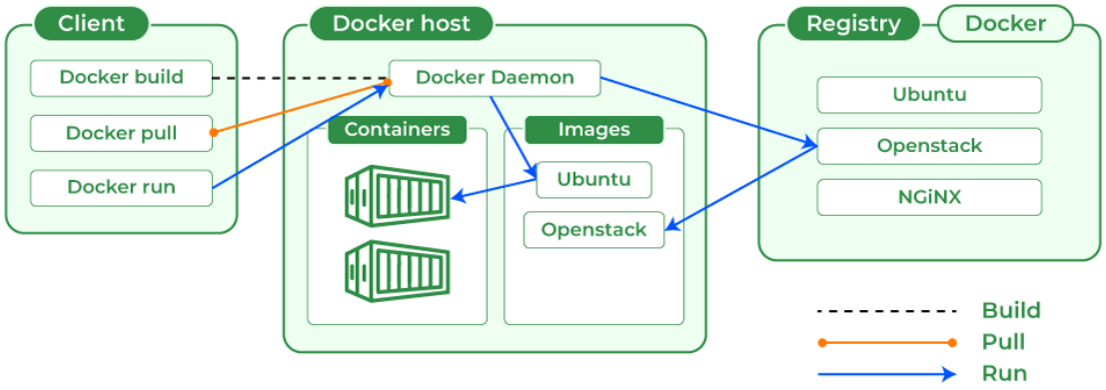
4. What is Docker?

Docker is a platform that enables developers to build, package, and distribute applications in containers.

Why Docker?

- **Lightweight and Portable**: Runs applications consistently across different environments.
- **Rapid Deployment**: Faster startup compared to VMs.
- **Automation**: Simplifies DevOps workflows.
- **Scalability**: Easily integrates with orchestration tools like Kubernetes.

5. Docker Architecture



- **Docker Client**: CLI or API to interact with Docker.
- **Docker Daemon (dockerd)**: Runs on the host machine and manages images, containers, networks, and volumes.
- **Docker Image**: A blueprint for creating containers.
- **Docker Container**: A running instance of an image.
- **Docker Registry**: Stores images (Docker Hub, private registries).

6. Docker Workflow

1. **Write a Dockerfile** to define an image.
2. **Build the Image** using `docker build`.
3. **Run the Container** using `docker run`.
4. **Push the Image** to a registry.
5. **Deploy** containers on multiple environments.

7. Features of Docker

- **Portability**: Works across different OS and cloud providers.
- **Lightweight**: Shares the host OS kernel, reducing overhead.
- **Security**: Provides process-level isolation.
- **Version Control**: Ensures consistency in application builds.
- **Networking**: Provides built-in networking support.

8. Benefits of Docker

- **Faster application deployment**.
- **Reduced conflicts between environments**.
- **Efficient resource utilization**.
- **Improved DevOps workflows**.

9. Docker Commands with Examples

Basic Commands:

```
docker exec -it <container_id_or_name> /bin/bash
```

```
docker --version   # Check Docker version
docker images      # List available images
docker ps -a       # List all containers
docker run -d -p 80:80 nginx  # Run an Nginx container
docker stop <container_id>  # Stop a container
```

```
docker rm <container_id>  # Remove a container
docker rmi <image_id>  # Remove an image
docker logs <container_id>  # View container logs
```

10. What is a Dockerfile?

A **Dockerfile** is a script containing instructions to create a Docker image.

Example Dockerfile:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y nginx
COPY index.html /var/www/html/
CMD ["nginx", "-g", "daemon off;"]
```

11. Components Inside a Dockerfile

- **FROM**: Specifies base image.
- **RUN**: Executes commands during build.
- **COPY/ADD**: Copies files into the image.
- **CMD/ENTRYPOINT**: Defines the startup command.
- **EXPOSE**: Specifies ports to expose.
- **WORKDIR**: Sets the working directory.

12. Docker Registries

- **Docker Hub** : Public registry for Docker images.
- **Private Registry**: Organizations can host their own registries.
- **AWS ECR, GCR, Azure ACR**: Cloud-based registries.

13. Docker Compose

Docker Compose is a tool to define and manage multi-container applications using a YAML file (`docker-compose.yml`).

Difference Between `docker run` and Docker Compose

Feature	`docker run`	Docker Compose
Command Type	CLI-based	YAML-based
Use Case	Single container	Multi-container
Configuration	Manual command execution	Automated with `docker-compose.yml`

14. Example `docker-compose.yml`

```
version: '3'
services:
```

```
web:
  image: nginx
  ports:
    - "8080:80"
db:
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: example
```

This file defines a web service using Nginx and a database service using MySQL.