SOFTWARE ENGINEERING

UNIT - 4

TOPIC – 4

CONTAINERIZATION USING DOCKER AND DOCKER COMPOSE

Definition of Containerization:

Containerization is a way to package software and everything it needs (like code, libraries, and settings) into a single "container" so it can run anywhere—on your computer, in the cloud, or on a server—without any problems.

Description:

Imagine you have a lunchbox that keeps your food separate (rice, curry, snacks) and ready to eat anywhere. A container in DevOps works like this lunchbox. It packs your application and all its "ingredients" (dependencies) into one box, making sure it works the same way, no matter where you open it (run it).

Before containerization, moving software between environments (like from your laptop to a server) often caused issues because the environments were different. Containers solve this by providing a consistent environment inside the "box."

Example:

You create an app that needs a specific version of Python and some libraries. Normally, if you run it on a friend's computer that doesn’t have the same setup, it won’t work. But if you put your app in a container, the container includes Python, libraries, and everything else your app needs. Now, it will work perfectly on your friend’s computer, your laptop, or even on a big server.

Types of Containerization Tools:

1. Docker - Most popular tool for creating and managing containers.
2. Kubernetes - Manages and orchestrates containers at scale.
3. Podman - Similar to Docker but daemonless.
4. LXC (Linux Containers) - Lightweight container system.
5. OpenShift - Built on Kubernetes, for enterprise use.
6. Containerd - A core container runtime tool.
7. Rkt (Rocket) - Simple alternative to Docker.

Docker is popular because it makes creating, sharing, and running containers super easy!

Difference Between Jenkins and Docker

Jenkins and Docker are both tools used in software development, but they do very different jobs. Jenkins is a tool that helps automate repetitive tasks in software development. For example, when developers write code, it needs to be built (converted into a working program), tested (to make sure there are no bugs), and deployed (put into use). Instead of doing these steps manually, Jenkins can handle them automatically, saving time and reducing mistakes.

Docker is more focused on making sure your application runs smoothly anywhere. It does this by packing the application and everything it needs—like files, libraries, and settings—into something called a **container**. A container ensures that your application behaves the same way no matter where you run it, whether it's on a developer's laptop, a testing server, or in live production.

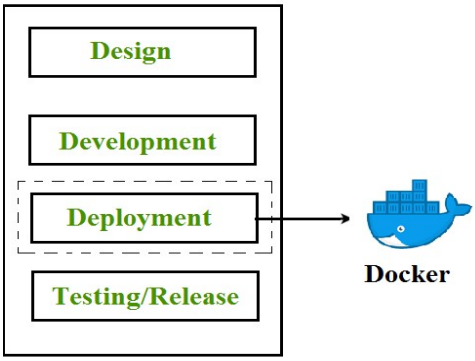
In short, Jenkins automates the process of preparing and delivering your application, while Docker ensures that the application works perfectly everywhere it is used.

Feature	Jenkins	Docker
Purpose	Automates tasks like building, testing, and deploying code.	Packages applications into containers so they run the same everywhere.

Feature	Jenkins	Docker
Focus	Streamlines the process of software development.	Ensures the app works perfectly on any computer or server.
Usage	Used for Continuous Integration/Continuous Deployment (CI/CD).	Used for creating, running, and managing containers.
What It Solves	Reduces manual work during development.	Removes compatibility issues when moving apps between computers.

Introduction to Docker

Docker is a tool that makes it easy to develop, package, and run applications. Imagine you've created an app on your laptop, and it works fine there. But when you try to run it on another computer or server, it fails because the environment is different. For example, your laptop might have a newer version of a library or software that the app needs, while the other computer has an older version.



Docker solves this problem by putting the app and all its dependencies into a **container**. This container includes everything the app needs to run, such as libraries, tools, and system settings. Once the app is in a container, it will run exactly the same way anywhere.

This is why Docker is so important in modern software development. It removes the headache of compatibility issues and makes deploying apps much easier and faster.

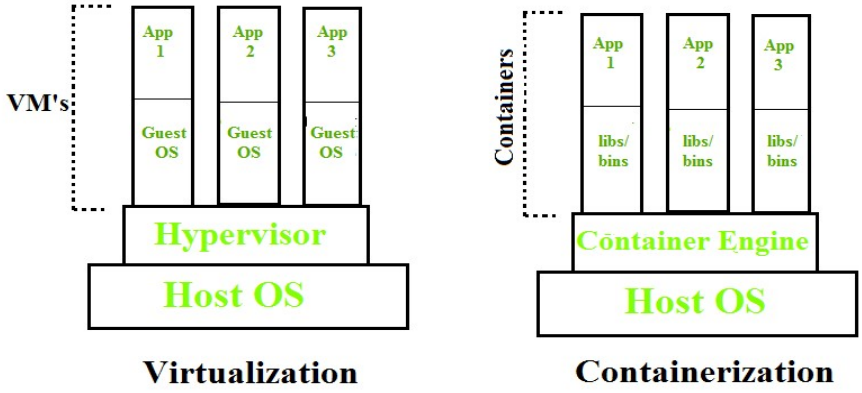
Docker and Virtual Machines

Docker is often compared to Virtual Machines (VMs) because both are used to run applications in isolated environments. But they work in very different ways.

A Docker container is a small and lightweight unit. It shares the operating system of the computer it's running on, which makes it fast and efficient. A VM, on the other hand, is heavier because it creates a separate operating system for each application.

Let's say you have three applications to run:

- With Docker, each application runs in its own container but uses the same operating system as the computer.
- With VMs, each application needs its own virtual operating system, which takes up a lot of resources.



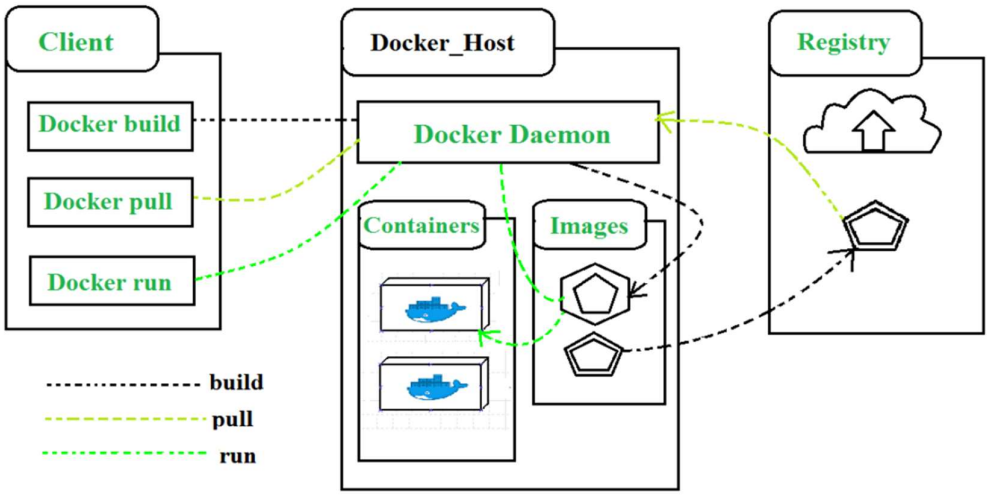
Because of this, Docker containers are faster to start, use less memory, and are easier to manage than VMs. This is why developers often prefer Docker for modern applications.

Feature	Docker (Containers)	Virtual Machines (VMs)
Size	Small and lightweight.	Heavy because each VM has its own full operating system.
Startup Speed	Very fast because it uses the host's operating system.	Slower because each VM needs to boot its own operating system.

Feature	Docker (Containers)	Virtual Machines (VMs)
Resource Use	Uses fewer resources like memory and CPU.	Uses more resources as every VM runs its own OS.
Isolation	Provides isolation but shares the host's OS.	Full isolation as each VM has its own OS.
Use Case	Good for running many lightweight applications quickly.	Useful for running applications that need very strong isolation.

Docker Architecture

Docker's architecture is how all the pieces of Docker work together. Each part has a specific role, and together they make Docker run smoothly.



1. **Docker Daemon**: This is the part of Docker that does all the heavy lifting. It runs in the background and takes care of creating, running, and managing containers.
2. **Docker Client**: This is the tool you use to interact with Docker. For example, when you type commands like `docker run`, the client sends these instructions to the Docker Daemon to execute.
3. **Docker Images**: These are like templates or blueprints. They contain everything needed to create a container, such as the application code and its dependencies.

4. **Docker Containers**: These are the actual running versions of Docker images. When you start a container, it's like bringing an image to life.
5. **Docker Registry**: This is a storage area for Docker images. Docker Hub is the most popular registry where you can find and share images.

Imagine the architecture like this:

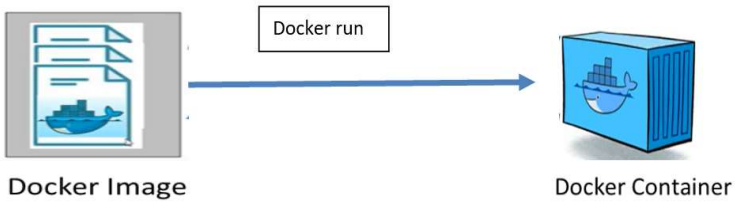
- You give a command to the **Docker Client**, which sends it to the **Docker Daemon**.
- The daemon pulls an image from the **Docker Registry** or uses a local one.
- It then uses this image to create and manage **containers** where your application runs.

This architecture makes Docker very efficient and easy to use for developers.

Key Components of Docker

To use Docker, you need to understand its main tools and features. These include:

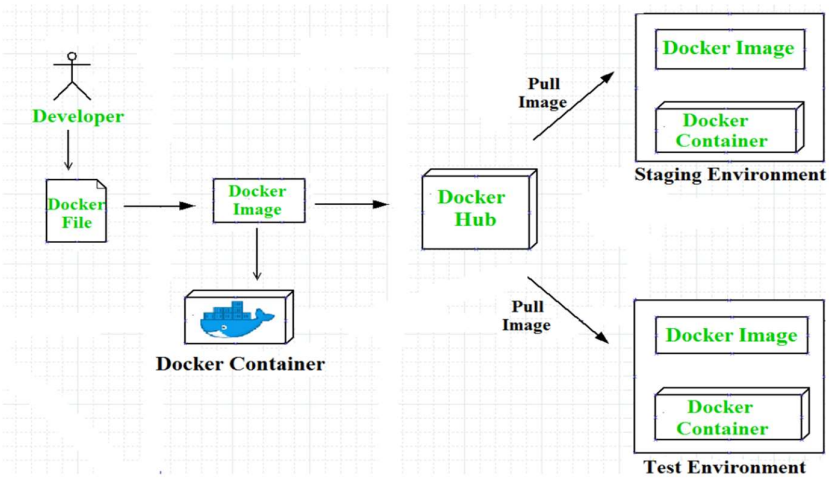
- **Docker Engine**: The main software that powers Docker. It takes care of creating and running containers.
- **Docker Image**: This is a package that has all the files and settings an application needs to run.
- **Container**: A live instance of an image where your application is running.
- **Dockerfile**: A script that lists all the steps to create a Docker image.
- **Docker Compose**: A tool that lets you manage multiple containers at the same time, making it easier to run complex applications.



How Docker Works

Using Docker typically involves three steps:

1. Create a **Dockerfile**. This file contains all the instructions Docker needs to create an image.
2. Build the image using the `docker build` command.
3. Run the image as a container using the `docker run` command.



For example, let's say you've built a web application. You would create a Dockerfile that includes the web server, your application code, and any libraries it needs. When you build the image and run it as a container, your application will be ready to use on any computer or server.

Writing a Dockerfile

A Dockerfile is a simple text file with instructions for Docker. Here's an example of a Dockerfile and what each line does:

```
FROM python:3.9
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

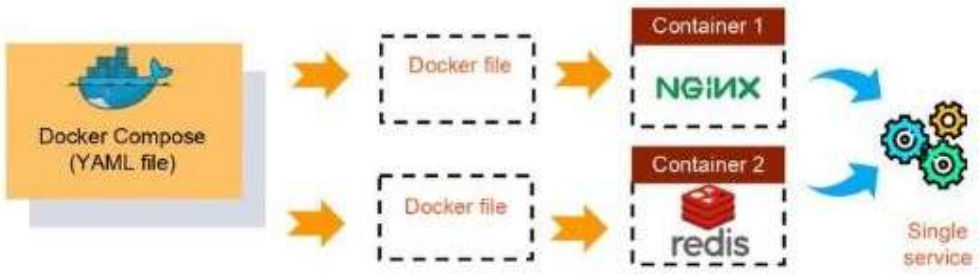
- **`FROM python:3.9`:** This sets the base image. It tells Docker to start with Python version 3.9.
- **`WORKDIR /app`:** This creates a folder called `/app` inside the container where all actions will happen.

- **`COPY . /app:`** This copies all files from your computer into the `/app` folder in the container.
- **`RUN pip install -r requirements.txt:`** This installs the Python libraries your application needs.
- **`EXPOSE 5000:`** This opens port 5000 so the application can communicate with the outside world.
- **`CMD ["python", "app.py"]:`** This starts the application by running the file `app.py`.

This Dockerfile ensures that your application has everything it needs to run properly.

Docker Compose

When your application needs more than one container, managing them manually can be difficult. For example, a web application might need one container for the web server and another for the database.



Docker Compose makes this process easier. It lets you define all the containers in a single file called **`docker-compose.yml`**. With one command, you can start or stop all the containers at once.

Here's an example **`docker-compose.yml`** file for running WordPress with a MySQL database:

```
version: '3'
services:
  wordpress:
    image: wordpress:latest
    ports:
```

```
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_PASSWORD: example
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
```

This file defines two services:

- **WordPress**: This container runs the WordPress application, accessible on port 8080.
- **Database (db)**: This container runs a MySQL database that WordPress uses to store its data.

To start both containers, you just run:

```
docker-compose up
```

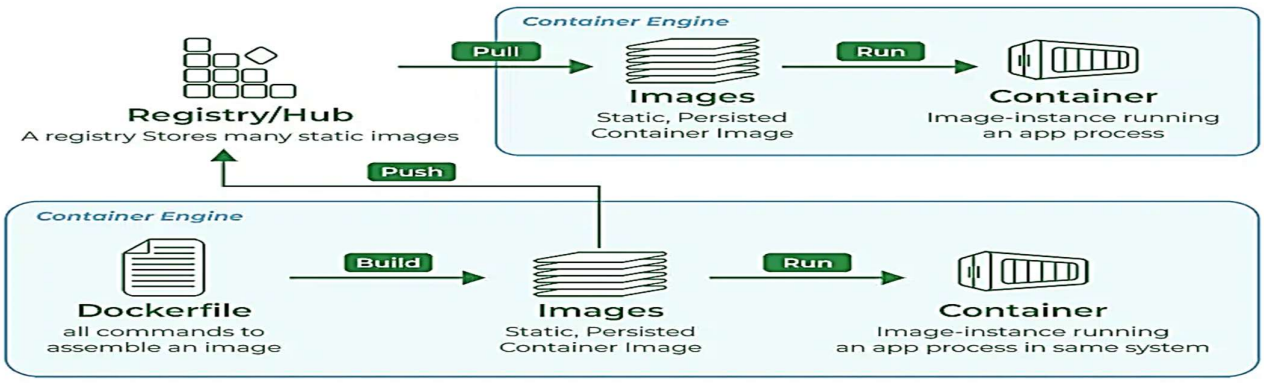
After running this command, your WordPress site will be available at `http://localhost:8080.`

Docker Hub

Docker Hub is an online library where developers can share and download Docker images. For example, if you need a basic setup like Ubuntu or MySQL, you can pull it from Docker Hub using:

```
docker pull ubuntu
```

You can also upload your own images to Docker Hub. This is useful for sharing your work with others or reusing setups in different projects.


Container Engine
Pull
Run
Registry/Hub
A registry Stores many static images
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process
Push
Container Engine
Build
Run
Dockerfile
all commands to
assemble an image
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process in same system