# **Docker Complete Notes**

## **1. Introduction to Docker**

### **What is Docker?**

Docker is an open-source platform that allows developers to automate the deployment of applications inside lightweight, portable containers.

### **Why Use Docker?**

* Simplifies software deployment
* Works across different environments (dev, test, prod)
* Reduces dependency conflicts
* Efficient resource utilization

### **Docker vs Virtual Machines**

| **Feature** | **Docker (Containers)** | **Virtual Machines (VMs)** |
| --- | --- | --- |
| Boot Time | Seconds | Minutes |
| Performance | High (No Hypervisor) | Low (Due to Hypervisor) |
| Size | Lightweight (~MBs) | Heavy (~GBs) |
| Isolation | Process-level | Full OS-level |

### **Docker Architecture**

* **Docker Client**: CLI/GUI to interact with Docker
* **Docker Daemon**: Runs in the background to manage containers
* **Docker Images**: Read-only template for containers
* **Docker Containers**: Running instances of images
* **Docker Registry**: Stores images (Docker Hub, private registries)

### **Installing Docker**

#### **Linux (Ubuntu/Debian)**

sudo apt update

sudo apt install docker.io -y

sudo systemctl start docker

sudo systemctl enable docker

#### **Windows & macOS**

Download and install Docker Desktop from [Docker’s official website](https://www.docker.com/get-started).

## **2. Docker Basics**

### **Common Docker Commands**

#### **Check Docker Version**

docker --version

#### **Run a Container**

docker run hello-world

#### **List Running Containers**

docker ps

#### **List All Containers (Including Stopped)**

docker ps -a

#### **Stop a Container**

docker stop <container\_id>

#### **Remove a Container**

docker rm <container\_id>

## **3. Docker Images**

### **Pulling an Image**

docker pull ubuntu

### **Listing Images**

docker images

### **Removing an Image**

docker rmi ubuntu

### **Building a Custom Image**

#### **Create a** Dockerfile

FROM ubuntu:latest

RUN apt update && apt install -y nginx

CMD ["nginx", "-g", "daemon off;"]

#### **Build the Image**

docker build -t my-nginx .

#### **Run a Container from Custom Image**

docker run -d -p 80:80 my-nginx

## **4. Docker Networking**

### **Types of Docker Networks**

* **Bridge** (Default) – Used for container-to-container communication
* **Host** – Removes isolation, directly uses host network
* **None** – No networking
* **Overlay** – Used in Swarm mode
* **Macvlan** – Assigns MAC addresses to containers

### **Creating a Network**

docker network create my\_network

### **Connecting a Container to a Network**

docker network connect my\_network my\_container

### **Exposing a Port**

EXPOSE 8080

### **Publishing a Port**

docker run -d -p 8080:80 nginx

## **5. Docker Volumes and Storage**

### **Types of Storage**

* **Bind Mounts** – Direct mapping to host directory
* **Volumes** – Managed by Docker
* **tmpfs** – In-memory storage

### **Creating a Volume**

docker volume create my\_volume

### **Mounting a Volume**

docker run -v my\_volume:/app -d nginx

## **6. Docker Compose**

### **What is Docker Compose?**

Docker Compose allows running multi-container applications using a docker-compose.yml file.

### **Installing Docker Compose**

sudo apt install docker-compose -y

### **Example** docker-compose.yml

version: '3'

services:

web:

image: nginx

ports:

- "8080:80"

### **Running Compose**

docker-compose up -d

## **7. Docker Security**

### **Best Practices**

* Run containers as non-root users
* Scan images for vulnerabilities (docker scan <image>)
* Use docker secrets for sensitive data

## **8. Docker Logging and Monitoring**

### **Viewing Logs**

docker logs <container\_id>

### **Monitoring Containers**

docker stats

## **9. Docker Swarm (Orchestration)**

### **Initializing Swarm Mode**

docker swarm init

### **Deploying a Service**

docker service create --name web -p 80:80 nginx

### **Scaling a Service**

docker service scale web=3

## **10. Docker in Kubernetes**

### **Deploying a Container in Kubernetes**

apiVersion: v1

kind: Pod

metadata:

name: my-nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

kubectl apply -f my-nginx.yaml

## **11. Docker in CI/CD Pipelines**

### **Using Docker in Jenkins**

pipeline {

agent any

stages {

stage('Build') {

steps {

sh 'docker build -t my-app .'

}

}

stage('Push') {

steps {

sh 'docker push my-app'

}

}

}

}

## **Conclusion**

Docker simplifies development, deployment, and scalability by using containerized applications. It’s widely used in modern DevOps workflows, microservices, and cloud-native applications.

This guide provides a **solid foundation** in Docker with **practical examples**. 🚀