docker -v

docker images

docker pull hello-world

docker search mysql

docker ps # show only running contianers

docker ps -a # show all contianers including stopped ones

docker run hello-world - create container using iamge hello-world

docker run hello-world:latest - careate new continer using latest image tag

Hello from Docker! - docker run to crate container using image hello-world image

docker run --name hello-c1 hello-world

**#create new container using hello-world image and name of container hello-c1 – we use docker run to create new conatianer**

docker run --name python-c1 python:3.9-slim

docker run --name python-c2 -d python:3.9-slim

0aada851820ecfb28eaf8d90596484a310b56aa1176a4a6e81cde483dcf779d3

**#run coantienr in bg and print coantienr id**

docker run --name python-c3 -d python:3.9-slim

docker run --name python-c3 -it -d python:3.9-slim

**# create and run a container in interactive and detach mode , up rahega kyonki -it ka use kara**

docker exec -it python-c3 python - # wite now we are in python terminal

docker inspect python-c3

docker run --name python-c4 -it python:3.9-slim **-d use backg main run karega aur exec se terminal milega apakoo , -it in interactive mode main run karega container ko**

docker run --name python-c5 -it --rm python:3.9-slim

**# run for specific interval till you use it and then automatically stop and remove**

docker stop python-c3

docker start python-c3

docker restart python-c3

docker rm python-c4

docker rmi hello-world - to remove images

docker container prune

WARNING! This will remove all stopped containers.

docker system prune

**WARNING! This will remove:**

**- all stopped containers**

**- all networks not used by at least one container**

- all dangling images

docker logs python-c3

docker pull mysql

docker run --name mysql-c1 -e MYSQL\_ROOT\_PASSWORD=deep -d mysql

**# e for env variable , d for run in background , mysql-c1 coantiner name and mysql is image return coantiner id**

**Docker run to run conainer to create container , -d bg main run karne ka liye**

**Docker exec to run in shell**

docker exec -it mysql-c1 mysql -u root -p

**docker run --name nginx-c1 -d -p 8080:80 nginx**

# hsot machine ka 8080 port ko hit karunga toh ham 80 port ko hit kar rha hai

# create nginx container and expose it port

docker pull httpd:alpine

#use ubuntu base image

FROM ubuntu

#setup working dirctory inside coatiner

WORKDIR /app

# copy script file to the container in app directory

COPY script.sh .

# make the script executable

RUN chmod+x script.xh

# run the script using this cmd when container starts

CMD ["./script.sh"]

Dockerfile

**now build docker images and give iamge name**

docker build -t mydockerimage1 .

**# start building docker image in current directory**

docker run myimage1

Hello from Deepanshu's image # create coatner with our image

docker run --name my-c1 myimage1

Hello from Deepanshu's image - **crating coaniner form docker iamges**

docker build --no-cache -t myimage . 3

[+] Building 6.1s (10/10) FINISHED - image build without cache , thoda slow hoga , but without cache build hoga

**.dockerignore – ignore file while build the imgae**

.dockerignore - .env , node-modules

**# To push image inside docker hub**

C:\Windows\system32>docker tag 3652fda97c30 deepanshu12353/mydockerimage

**# mydockerimage ko tag kara usename ka sath , image id di jo push karni hai**

C:\Windows\system32>docker push deepanshu12353/mydockerimage

**Docker Volume**

#host to container

**This is bind volume as we specify path and it can be located anywhere**

docker run --name python-c1 -itd -v "C:\Users\my pc\Desktop\docker geekyshows\DockerVolume\volume:/myapp" python - bind volume fill path mount with vol

mounting fplder with my app

docker exec -it python-c1 bash

**# Between container mount data volume – named volume**

docker volume create myvol

docker volume ls

docker volume inspect myvol

* docker run --name c1 -itd -v myvol:/myapp python

**# myvol mount to myapp for container c1 – terminal 2**

**docker run --name c2 -itd -v myvol:/myapp1 python – named volume –**

myvol is mount to myapp1 mount to container directory – named volueme\

docker exec -it c1 bash

**# dono terminal pa data share hoga aapka , between container data share karenge , toh volume create kara pahle phir mount kar diya dono continaer se**

# continaer delte kar do par volume delte nahi hoga ,same rahega data voleume ka persisit reahge

docker volume rm myvol

>docker run --name c1 -itd -v /myvol python

docker run --name c2 -itd --volumes-from c1 python

Dockerfile

FROM ubuntu

VOLUME [ "/data" ]

VOLUME[“/data”]

**# conainer ka andar mount point create karo iss wala ka andar , data directory main jo likhoge iske data outside volume pa kahi hoga data iske , volume nahi mount point create kar rha coantiner ka liye**

# volume carete nahi karta , is container ko use karke mount point banagega container

> docker build -t mynewimage .

**docker run --name c1 -itd -v myvol:/data mynewimage /bin/bash** - named volume and data is container directory that is present inside the containers

> docker exec -it c1 bash

> docker inspect c1

**DOCKER Networking**

docker network ls

docker pull nginx

docker network inpsect bridge

docker run --name c1 -itd alpine ash

docker network create mynet

docker run --name c1 --network mynet -it ubuntu

docker attach c1 = # login into c1

/ # ip addr show

/ # ping -c 5 google.com

/ # ping -c 5 172.17.0.3/16

ping: bad address '172.17.0.3/16'

/ # ping -c 5 172.17.0.3 $ ping c2 from c1 sucdcefully

docker network create --driver bridge net1

>docker network inspect net1 $ **createing customer network attach container to c3 and c4**

docker run --name c3 -itd --network net1 alpine ash

**#create c3 coantienr c3 and attach to network net1**

docker container ls

**docker inspect network net1**

docker run --name c4 -itd --network net1 alpine ash

**$ two container are present inside the network net1 such as c3 and c4**

docker inspect bridge

docker attach c1

/ # ip addr show

**$ from container c1 sending reques to c2 and successfully getting response**

docker network inspect net1

docker attach c3

/ # ping -c 5 ^C

/ # ping -c 5 172.18.0.3

**# container can only communicate when they are in same network like in a bridge or in net1 custome network – when they both are at same network**

docker network inspect host

docker run -itd --network host --name nginx1 nginx

**# is container se hsot se sattach kar diya hamnee**

**Overlay network** – alag host pa diff container hai who aapas main communicate kar paye toh hsot akr use karenge

**Closeted setup ka use kar sakte hai , kubertens main aache se semajh payega**

Why Do We Need Overlay Networks?

By default, Docker containers can only communicate on the same host using bridge or host networks.

But what if your app runs like this?

pgsql

Copy

Edit

Frontend --> Host A

Backend --> Host B

Database --> Host C

They need to communicate across physical/virtual machines.

👉 That’s where overlay networks come in!

**What is the Docker Daemon?**

The **Docker Daemon** is the **background service** that runs on your system and does **all the heavy lifting** for Docker.

**🧠 Think of it like:**

**"The brain of Docker"** — it listens for Docker commands and manages:

* Containers
* Images
* Volumes
* Networks
* And everything else Docker-related

**DOCKER Compose**

#multi container appn ka liye , jusme alag component chaiye application ko run ka liye ,

jaise different iamge , contaienrs chaiye hamko toh docker compose use karo

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

**Docker Compose** is a tool that allows you to **define and run multi-container Docker applications** using a single YAML file.

Instead of running multiple docker run commands for each container (e.g., web server, DB, Redis), you can use **one file** and **one command** to run them **all together**.

## 📦 Why Use Docker Compose?

| **✅ Feature** | **💬 What It Means** |
| --- | --- |
| Multi-container setup | Run web + DB + Redis etc. together |
| YAML configuration | Declare everything in docker-compose.yml |
| Easy startup/shutdown | Just up or down the stack |
| Reusable environments | Great for dev, test, staging environments |
| Built-in networking | Containers can talk via service names |

**Redis mysql python sab chij ek sath use hori toh ham docker compose ka use karenge**

We write yml file and define all services inside container

# appn ko run kanrne ka liye multiple cheej chahiye toh docker compose use karo , ek cmd main sare kam kar skate hai

version: '3.8'

services:

web:

image: nginx

ports:

- "8080:80"

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: mypassword

docker compose version

Docker-compose.yml

services:

  web:

    image: "nginx"

  rcache:

    image: "redis:6.0-alpine3.18"

version: '3.8'

services:

web:

image: nginx

ports:

- "8080:80"

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: mypassword

**docker compose config**

docker volume ls

docker compose up –d #**create and start containers**

docker compose ps

docker compose exec web bash

docker compose stop

docker compose down --volume

$ Remove all the containers that were started from docker compose

Using of tag in

.env

TAG = 6.0-alpine3.18

Docker-compose.yml

Docker-compose.yml

services:

  web:

    image: "nginx"

  rcache:

    image: "redis:${TAG}"

docker compose up -d

docker container ls

docker compose config

docker compose down --volume

Docker-compose.yml

services:

  web:

    image: "nginx"

  rcache:

    image: "redis:${TAG}"

  db:

    image: mysql

    environment:

      - MYSQL\_ROOT\_PASSWORD=deep

docker compose up -d

docker compose ls

services:

  web:

    image: "nginx"

  rcache:

    image: "redis:${TAG}"

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    env\_file:

      - mysqlconfig.env

mysqlconfig.env

MYSQL\_ROOT\_PASSWORD=deep

docker compose config

services:

  web:

    image: "nginx"

  rcache:

    image: "redis:${TAG}"

    profiles:

      - rediscache

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    env\_file:

      - mysqlconfig.env

$ toh ab redis execute or show nahi hoga , hame batna padega tab lega redis kpo , profile manualay diya hai redis ka toh batana padega

docker compose config - It will show yaml files

docker compose --profile rediscache config

# we need to manully need to tell then it will show inside config files

docker compose down --volumes

✔ Container compose-db-1 Removed

docker compose --profile rediscache up -d

# it will start all containers including rcahce first , we need to specify profile inside cmd to run rcache container

docker compose --profile rediscache down –volumes - # if you are using profiles then we need to specify profiles then it will remove only

| **Command** | **Action** |
| --- | --- |
| docker-compose up | Start all services |
| docker-compose up -d | Start in detached mode |
| docker-compose down | Stop and remove all containers/networks |
| docker-compose logs | View logs from all services |
| docker-compose exec <svc> | Run command in a service (like e |

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

  rcache:

    image: "redis:${TAG}"

    profiles:

      - rediscache

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    env\_file:

      - mysqlconfig.env

$ PORT EXPOSE 8000 host machine ka port hamari machine ka port , or conainer ka port hai 80hamara , 8000 ko bind kara hamne 80 ka port se , 80 port container ka port hai

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

  rcache:

    image: "redis:${TAG}"

    # profiles:

    #   - rediscache

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    env\_file:

      - mysqlconfig.env

docker compose up -d

docker network ls

docker compose down --volume

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

  rcache:

    image: "redis:${TAG}"

    # profiles:

    #   - rediscache

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    env\_file:

      - mysqlconfig.env

networks:

  my\_network:

    driver: bridge

**# ye network in service main checiye specify akro**

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

    networks:

      - my\_network

  rcache:

    image: "redis:${TAG}"

    # profiles:

    #   - rediscache

    networks:

      - my\_network

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    networks:

      - my\_network

    env\_file:

      - mysqlconfig.env

networks:

  my\_network:

    driver: bridge

**#service main network specify kara**

**$ depends upon service pahle ye execute ho baad main ye ,service depend karta hia aapas main**

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

    networks:

      - my\_network

    depends\_on:

      - db

      - rcache

  rcache:

    image: "redis:${TAG}"

    # profiles:

    #   - rediscache

    networks:

      - my\_network

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    networks:

      - my\_network

    env\_file:

      - mysqlconfig.env

networks:

  my\_network:

    driver: bridge

**$ toh web service db or rcahce pa depend hogi hamari , pahle db or rcahce banegi phir hamari web service create hote hai**

Docker-compose-dev.yml – sabhi env ka liye alag file hogi , staging ka liye alag hogi , or prodn ka liye alag file hogi for both

services:

  web:

    image: "nginx"

    ports:

      - "8000:80"

    networks:

      - my\_network

    depends\_on:

      - db

      - rcache

  rcache:

    image: "redis:${TAG}"

    # profiles:

    #   - rediscache

    networks:

      - my\_network

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

    networks:

      - my\_network

    env\_file:

      - mysqlconfig.env

networks:

  my\_network:

    driver: bridge

docker-compose-dev.yml

services:

  web:

    image: "nginx"

  db:

    image: mysql

    # environment:

    #   - MYSQL\_ROOT\_PASSWORD=deep

Dev wali yml file hai hamari – docker-compose.dev.yml – isse run kasie karoge

services:

  web:

    image: "nginx"

  db:

    image: mysql

    env\_file:

      - mysqlconfig.env

docker compose -f docker-compose.dev.yml up –d

$ -f flag ka use karo , -f –f multiple file run kar sakte ho ,dev wali run kar paoge ,web or nignx cconater create hoge

docker compose -f docker-compose.dev.yml down

Compose 2

FROM alpine

WORKDIR /app

Dockerfile

services:

  web:

    image: "nginx"

  db:

    image: mysql

    environment:

      - MYSQL\_ROOT\_PASSWORD=deep

**Docker compose main docker file use karenge toh build ka use karenge , Build likhenge**

services:

  app:

    build:

      dockerfile: Dockerfile

  web:

    image: "nginx"

  db:

    image: mysql

    environment:

      - MYSQL\_ROOT\_PASSWORD=deep

C:\Users\my pc\Desktop\docker geekyshows\compose>cd compose2

$docker file se docker compose ka container create kar rha hai

docker compose up -d

docker compose exec db mysql -u root -p

Let me know if you want to try it with a small Python or Node server running inside one of them for a cooler demo 😎

4o

##### You said:

what is overlay network please

###### ChatGPT said:

Great question, Deepanshu! You're leveling up into **multi-host Docker networking** now 🔥

## 🌐 What is an ****Overlay Network**** in Docker?

An **overlay network** is a **Docker network that spans across multiple Docker hosts (machines)**. It's mainly used in:

### 🔧 Docker Swarm (or other orchestrators like Kubernetes)

## 📦 Why Do We Need Overlay Networks?

By default, Docker containers can **only communicate on the same host** using bridge or host networks.

But what if your app runs like this?

pgsql

CopyEdit

Frontend --> Host A

Backend --> Host B

Database --> Host C

They need to communicate **across physical/virtual machines**.

👉 That’s where overlay networks come in!

## 🧠 How It Works (Simplified)

Docker uses:

* **VXLAN (Virtual Extensible LAN)** under the hood
* Each node (host) runs a Docker daemon part of the Swarm
* Docker handles **routing, encryption**, and **DNS resolution**

You define a **virtual network** that spans across hosts, and Docker ensures packets are routed to the right container on the right machine.

## 🧪 How to Create an Overlay Network (Docker Swarm)

### Step 1: Initialize Swarm (on your first machine)

bash

CopyEdit

docker swarm init

### Step 2: Create an overlay network

bash

CopyEdit

docker network create -d overlay my\_overlay\_net

### Step 3: Run services using that network

bash

CopyEdit

docker service create --name web --network my\_overlay\_net nginx

docker service create --name api --network my\_overlay\_net python:3.9-slim

Now web can talk to api by name, even if they’re on **different machines**.

## ✅ Quick Summary

| **Feature** | **Overlay Network** |
| --- | --- |
| Scope | Multi-host (cross-node) communication |
| Use Case | Docker Swarm / Kubernetes clusters |
| Backed By | VXLAN tunneling |
| Created With | docker network create -d overlay ... |
| Bonus | Built-in DNS, encrypted traffic option |

Let me know if you want a simple hands-on overlay demo using VMs or Docker Swarm mode on your local machine 🚀

4o

##### You said:

what is docker daemon

###### ChatGPT said:

Awesome question, Deepanshu! Let’s break it down in simple terms. 👇

## 🐳 What is the ****Docker Daemon****?

The **Docker Daemon** is the **background service** that runs on your system and does **all the heavy lifting** for Docker.

### 🧠 Think of it like:

**"The brain of Docker"** — it listens for Docker commands and manages:

* Containers
* Images
* Volumes
* Networks
* And everything else Docker-related

## 📌 Official Definition:

The Docker **daemon (dockerd)** is a persistent background process that manages Docker containers, images, networks, and storage.

## ⚙️ How it works:

* You (the user) run a command like:

bash

CopyEdit

docker run ubuntu

* The **Docker CLI (docker)** sends that command to the **Docker Daemon (dockerd)**
* The **Docker Daemon**:
  + Pulls the image (if needed)
  + Creates and starts the container
  + Manages networking and volumes

## 🧩 Components Breakdown:

| **Component** | **Role** |
| --- | --- |
| dockerd | The Docker Daemon (server) |
| docker (CLI) | Command-line client (you interact with) |
| containerd | Low-level container runtime |

## 🖥️ Check if it's running:

### On Linux:

bash

CopyEdit

ps -ef | grep dockerd

### On Windows or Mac (via Docker Desktop):

Docker Desktop runs dockerd inside a VM.

## 🔐 Docker Daemon listens via:

* **Unix socket** (default):

arduino

CopyEdit

/var/run/docker.sock

* **TCP socket** (optional, for remote management):

nginx

CopyEdit

dockerd -H tcp://0.0.0.0:2375

## 📦 Summary:

| 🔹 Docker Daemon is: | A background process that runs Docker | | 🔹 Command | dockerd | | 🔹 Interacts with | docker CLI | | 🔹 Manages | Containers, Images, Volumes, Networks |

Let me know if you want to go deeper into how dockerd works internally or how to securely expose it for remote Docker management! 🚀

4o

##### You said:

DOCKER Compose - exxplain

###### ChatGPT said:

Absolutely, Deepanshu! Let’s simplify **Docker Compose** so well that even your future self will thank you 😄💡

## 🧱 What is ****Docker Compose****?

**Docker Compose** is a tool that allows you to **define and run multi-container Docker applications** using a single YAML file.

Instead of running multiple docker run commands for each container (e.g., web server, DB, Redis), you can use **one file** and **one command** to run them **all together**.

## 📦 Why Use Docker Compose?

| **✅ Feature** | **💬 What It Means** |
| --- | --- |
| Multi-container setup | Run web + DB + Redis etc. together |
| YAML configuration | Declare everything in docker-compose.yml |
| Easy startup/shutdown | Just up or down the stack |
| Reusable environments | Great for dev, test, staging environments |
| Built-in networking | Containers can talk via service names |

## 🧪 Basic Example

### 📄 docker-compose.yml

yaml

CopyEdit

version: '3.8'

services:

web:

image: nginx

ports:

- "8080:80"

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: mypassword

### ▶️ Run the entire app:

bash

CopyEdit

docker-compose up

🔥 That’s it! Both **nginx** and **MySQL** containers will run in a single **network** and can communicate using names like db and web.

## 🎯 Common Compose Commands

| **Command** | **Action** |
| --- | --- |
| docker-compose up | Start all services |
| docker-compose up -d | Start in detached mode |
| docker-compose down | Stop and remove all containers/networks |
| docker-compose logs | View logs from all services |
| docker-compose exec <svc> | Run command in a service (like exec) |

## 🧠 Internal Features

* **Default network**: Docker Compose creates a dedicated bridge network for your services.
* **Named volumes**: You can define volumes in the YAML to persist data.
* **Service linking**: Services can reach each other by name.

## 🔁 Comparison with docker run

| **Feature** | **docker run** | **docker-compose** |
| --- | --- | --- |
| Single container | Yes | No |
| Multi-container setup | Manual | Easy (via YAML) |
| Network management | Manual | Automatic |
| Environment-specific | Difficult | Very easy (.env, YAML overrides) |

## ✅ When Should You Use Docker Compose?

* Local development with multiple containers 🧑‍💻
* Testing microservices 🧪
* Sharing environments with your team 🌍
* Automating project startup 🛠️

Let me know if you want help writing a Docker Compose file for a real project (e.g., Django + Postgres, Node.js + Redis, etc.) 💻🚀

4o

##### You said:

why docker swarm and tell why to use it

###### ChatGPT said:

Perfect question, Deepanshu — stepping into **orchestration** now! 🚀

Let’s break it down super clean:

## 🐝 What is ****Docker Swarm****?

**Docker Swarm** is Docker’s **native orchestration tool** that lets you manage a **cluster of Docker engines** (called nodes) as a **single virtual system**.

In simple terms:

It’s a **way to manage multiple Docker containers across multiple servers** — automatically.

## 🧠 Why Do We Need Docker Swarm?

Imagine you're running a production app with:

* 5 web servers 🕸️
* 2 databases 📦
* 3 Redis caches ⚡

And they run on **different machines**.

Do you want to:

* Start containers manually?
* Handle traffic manually?
* Restart failed containers yourself?

💥 **Nope!** You need **orchestration**. That's where Swarm comes in.

## ✅ What Does Docker Swarm Do?

| **Feature** | **What It Means** |
| --- | --- |
| 🔁 **High Availability** | If a container or node fails, Swarm reschedules it elsewhere |
| 📦 **Service Scaling** | Easily scale up/down: docker service scale web=10 |
| 📡 **Load Balancing** | Built-in routing and load balancing for services |
| 📜 **Declarative Service Model** | You define desired state; Swarm ensures it's always true |
| 🌐 **Overlay Networking** | Services can talk across different hosts using virtual networks |
| 🔐 **Security** | Automatic mutual TLS between nodes |

## 🛠️ How Docker Swarm Works:

### Components:

* **Manager Nodes**: Orchestrate, schedule, and manage services
* **Worker Nodes**: Just run containers as told by the manager
* **Services**: Long-running containers (e.g., nginx, mysql)
* **Tasks**: Individual container instances of a service

## 🧪 Basic Workflow Example:

bash

CopyEdit

# 1. Initialize Swarm

docker swarm init

# 2. Add worker nodes (you’ll get a join token)

# 3. Deploy a service

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

# 4. Scale it

docker service scale web=5

Swarm will deploy 5 nginx containers across available nodes and keep them running.

## 🆚 Docker Swarm vs Docker Compose vs Kubernetes

| **Feature** | **Docker Compose** | **Docker Swarm** | **Kubernetes** |
| --- | --- | --- | --- |
| Scope | Single machine | Multi-host | Multi-host |
| High Availability | ❌ | ✅ | ✅ |
| Built-in scaling | ❌ | ✅ | ✅ |
| Learning curve | 😄 Easy | 🙂 Medium | 😵‍💫 Steep |
| Use Case | Dev/Testing | Small prod apps | Enterprise-level |

## 🎯 When to Use Docker Swarm?

Use Docker Swarm when you:

* Need to **orchestrate containers** across machines
* Want **automatic failover** and **load balancing**
* Don’t want the **complexity of Kubernetes**
* Need a **lightweight, Docker-native** solution

## 🔥 Bonus: Swarm Is Built-in!

You don't need to install anything else — it's already in the Docker CLI.

Let me know if you want a mini-project to try Swarm locally or with VMs (like 1 manager + 2 workers) 💻⚙️