Docker

What is Docker?

Docker is a platform that lets you **package and run applications in containers**. A **container** is a lightweight, standalone unit that contains everything your app needs to run — code, libraries, settings, and system tools.

What is Container

A **container** in Docker is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software—**code, runtime, system tools, libraries, and settings**.

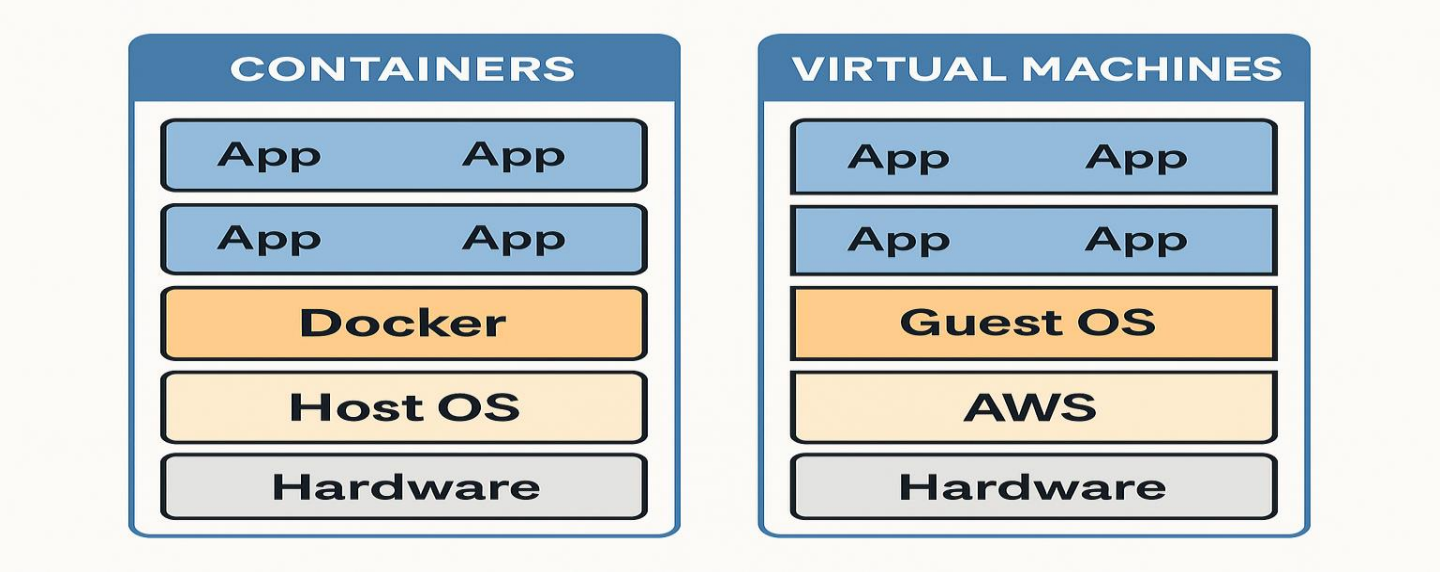
Key Points:

- **Isolated Environment**: Containers run independently and isolated from the host system and other containers.
- **Based on Images**: Containers are created from Docker images (blueprints containing the application and its dependencies).
- **Portability**: Containers can run consistently across environments (e.g., development, testing, production).
- **Lightweight**: Unlike virtual machines, containers share the host OS kernel, making them more efficient and faster to start.

Docker Vs Virtual Machine :

Containers virtualize the application layer. They **don't need a full OS**, just what's required for the app to run. That makes them **fast and efficient**.

VMs virtualize the hardware and run **entire OS instances**, making them **heavier and slower**, but offering **stronger isolation**.



In the diagram:

- On the **Containers** side, apps run on **Docker**, which uses the **Host OS**, making it smaller and more efficient.
- On the **VMs** side, apps run on **Guest OS** (inside AWS EC2 or other virtual platforms), each with their **own OS layer**, making it bulkier.

▯ Docker vs Virtual Machine – Comparison Table

Feature	Docker Containers	Virtual Machines
Architecture	Shares host OS kernel	Includes full OS (Guest OS on Host OS)
Startup Time	Very fast (seconds)	Slower (minutes)
Size	Lightweight (MBs)	Heavy (GBs)
Performance	Near-native performance	Slower due to OS overhead
Resource Usage	Low (efficient use of CPU, RAM)	High (each VM consumes more resources)
Isolation	Process-level isolation	OS-level isolation
Portability	Highly portable (same across environments)	Less portable due to OS dependencies
Boot Layer	Runs on container engine (Docker Daemon)	Runs on hypervisor (e.g., VMware, VirtualBox)
Use Case	Microservices, CI/CD pipelines, DevOps	Running multiple OS, legacy app support

🏢 Hotel vs Apartment Building Analogy

Concept	Docker Containers	Virtual Machines
Analogy	**Hotel rooms**	**Apartments in a building**
OS Sharing	**Share the hotel's central services (host OS)**	**Each apartment has its own facilities (OS)**
Startup Time	**Ready quickly (just check-in)**	**Takes time to set up (rent, furnish, etc.)**
Resources	**Use shared services efficiently**	**Duplicate many things—more space and cost**
Flexibility	**Easily move between hotels**	**Harder to move apartments (migration)**
Isolation	**Rooms are separate but share the same building**	**Fully isolated living spaces**

✅ Left Side – CONTAINERS

Structure (Top to Bottom):

1. **App (multiple)** – These are your applications running inside containers.
2. **Docker** – The **container engine** that runs and manages containers.
3. **Host OS** – The **single operating system** shared across all containers.
4. **Hardware** – The **physical or virtual server** the host OS is running on.

Key Idea:

- Containers **share the Host OS** using Docker.
- **No separate OS per app**, making it lightweight and fast.
- Efficient for microservices and DevOps.

✅ Right Side – VIRTUAL MACHINES

Structure (Top to Bottom):

1. **App (multiple)** – Applications inside each virtual machine.
2. **Guest OS** – Each VM has its **own full operating system** (Windows/Linux).
3. **AWS** – The **hypervisor or cloud provider** running the VMs (e.g., AWS EC2).
4. **Hardware** – The **underlying physical machine**.

Key Idea:

- Each app runs on a **separate OS** inside a VM.
- Heavier, more isolated, but **less resource-efficient**.
- Suitable for apps needing complete OS isolation or legacy support.

📊 Containers vs Virtual Machines – Extended Comparison with Sizes

Feature	Docker Containers	Virtual Machines
OS Layer	Shared Host OS	Each VM has its own Guest OS
Size	⚡ **Lightweight** (10s to 100s of MBs)	🐘 **Heavy** (several GBs per VM)
Startup Time	Seconds	Minutes
Resource Usage	Low (efficient CPU, RAM, disk)	High (due to OS overhead)
Isolation	Process-level (less secure but faster)	Full OS-level (more secure, heavier)
Use Case	Microservices, CI/CD, cloud-native apps	Legacy apps, multi-OS setups, deep isolation
Portability	High – runs anywhere Docker runs	Moderate – VM image migration is heavier

⚖️ Size Example:

- **Docker Container Image (e.g., Spring Boot app)**:
  ~100 MB - 300 MB
- **Virtual Machine (e.g., Ubuntu + App)**:
  ~2 GB - 10+ GB depending on the OS and software stack

How to Install Docker:

Open ubantu terminal.

shipconsole@B4XSGX3:~$ sudo apt install docker.io

shipconsole@B4XSGX3:~$ password : Ship$1234

shipconsole@B4XSGX3:~$ sudo docker –version

Post-installation steps for Linux :

🔧 **Purpose: Run Docker Without sudo**

By default, **Docker requires sudo** to run commands because the Docker daemon (dockerd) has **root-level privileges**. These commands allow you to run Docker as a **non-root user** safely.

🔹 Command-by-Command Explanation:

1. ✅ Create Docker Group

sudo groupadd docker

What it does: Creates a new Linux group called docker.

Why: Docker will allow anyone in this group to run Docker commands without using sudo.

2. ✅ Add Current User to Docker Group

sudo usermod -aG docker $USER

What it does: Adds your current user (via $USER) to the docker group.

Flags:

- -a: Append the group (don't remove from other groups).
- -G: Specify the group to add to.

Why: This gives your user permission to communicate with the Docker daemon.

3. ✅ Apply Group Changes in Current Session

newgrp docker

What it does: Starts a new shell with updated group permissions **without needing to log out and log back in**.

Why: Ensures your current terminal session applies the group change immediately.

4. ✅ Test Docker Access Without sudo

docker run hello-world

What it does: Runs a simple test container that prints a success message.

Why: Confirms that Docker is installed correctly and **your user can run it without sudo**.

Main Docker Commands :

Existing Docker images: https://hub.docker.com

docker pull hello-world

▼ Download (pull) the hello-world Docker image from **Docker Hub** (the default public image registry).

docker run hello-world

is used to verify that your Docker installation is working correctly.

docker pull hello-world = Download the image.

docker run hello-world = Start a container using the image.

What is hello-world?

- It's a **small test image** provided by Docker.
- Designed to verify that Docker is installed and working properly on your system.
- When run, it prints a confirmation message and exits.

What Happens Behind the Scenes:

1. Docker looks for the hello-world image **locally**.
2. If it doesn't exist locally, it **pulls it from Docker Hub**.
3. The image gets stored in your **local Docker image cache**.
4. You can then run it with:

docker images or **docker image ls**

What It Does:

It lists all the Docker images stored locally on your system.

```
shipconsole@B4XSGX3:~$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED        SIZE
ubuntu                  latest    a0e45e2ce6e6   7 days ago     78.1MB
hello-world             latest    74cc54e27dc4   3 months ago   10.1kB
atlassian/hello-world   latest    ee301c921b8a   2 years ago    9.14kB
```

docker ps

- > Shows only running containers.

```
shipconsole@B4XSGX3:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
shipconsole@B4XSGX3:~$
```

docker ps -a

Shows all containers:

- ▯ Running
- 🛑 Stopped
- ❌ Exited
- ⚠️ Crashed

```
shipconsole@B4XSGX3:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND       CREATED          STATUS                      PORTS     NAMES
2e3a39e615c0   hello-world              "/hello"      19 minutes ago   Exited (0) 19 minutes ago             wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"      20 minutes ago   Exited (1) 20 minutes ago             beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"   8 hours ago      Exited (0) 8 hours ago                wizardly_cartwright
3e6bcf5d1d74   hello-world              "/hello"      9 hours ago      Exited (0) 9 hours ago                nice_noether
83ebfaa62acd   hello-world              "/hello"      9 hours ago      Exited (0) 9 hours ago                affectionate_gauss
shipconsole@B4XSGX3:~$
```

docker run -it ubuntu

is used to start an interactive Ubuntu container.

🔍 Breakdown:

Part	Meaning
docker run	Create and start a new container
-i	Keep STDIN open (interactive mode)
-t	Allocate a pseudo-TTY (terminal)
ubuntu	The image to run (pulls from Docker Hub if not present)

Together, -it lets you **interact with the container via the terminal**, like you're using a Linux shell.

apt update

🔄 Update the local package index — a list of all available packages and their versions.

▯ What it does:

- Connects to the configured package repositories (e.g., Ubuntu mirrors).
- Downloads the latest **package metadata**.
- Ensures your system knows about the **latest versions** of software and dependencies.

```
shipconsole@B4XSGX3:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND       CREATED             STATUS                          PORTS     NAMES
620dc85eee8d   ubuntu                   "/bin/bash"   About a minute ago  Up About a minute                         hopeful_shamir
30747c569424   alpine                   "/bin/sh"     18 minutes ago      Up 8 minutes                              condescending_shockley
2de43af94585   ubuntu                   "/bin/bash"   About an hour ago   Exited (129) About an hour ago            charming_noyce
2e3a39e615c0   hello-world              "/hello"      2 hours ago         Exited (0) 6 minutes ago                  wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"      2 hours ago         Exited (1) 9 minutes ago                  beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"   10 hours ago        Exited (0) 11 minutes ago                 wizardly_cartwright
shipconsole@B4XSGX3:~$ docker stop 620dc85eee8d
```

docker stop 2de43af94585

is used to gracefully stop the running container with ID **2de43af94585**.

```
shipconsole@B4XSGX3:~$ docker stop 620dc85eee8d
620dc85eee8d
shipconsole@B4XSGX3:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND       CREATED             STATUS                          PORTS     NAMES
620dc85eee8d   ubuntu                   "/bin/bash"   2 minutes ago       Exited (127) 9 seconds ago                hopeful_shamir
30747c569424   alpine                   "/bin/sh"     18 minutes ago      Up 9 minutes                              condescending_shockley
2de43af94585   ubuntu                   "/bin/bash"   About an hour ago   Exited (129) About an hour ago            charming_noyce
2e3a39e615c0   hello-world              "/hello"      2 hours ago         Exited (0) 7 minutes ago                  wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"      2 hours ago         Exited (1) 10 minutes ago                 beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"   10 hours ago        Exited (0) 12 minutes ago                 wizardly_cartwright
```

🛑 What It Does:

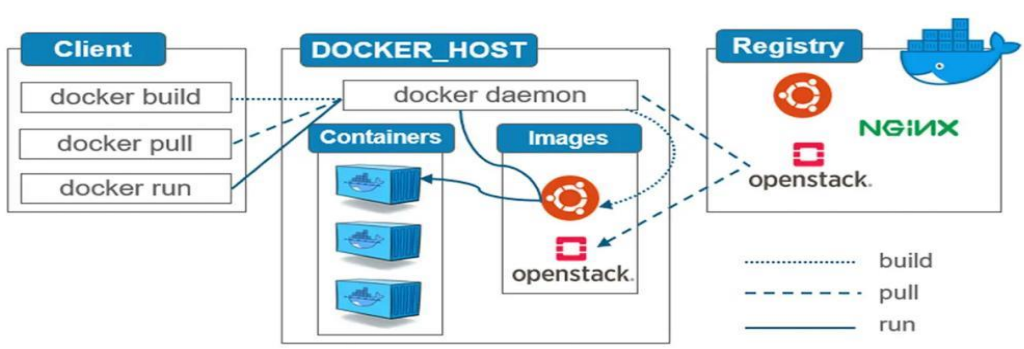
- Sends a **SIGTERM** signal to the container's main process to ask it to stop.
- If the process doesn't stop within **10 seconds**, Docker sends a **SIGKILL** to force-stop it.

docker start 620dc85eee8d

This starts the container.

```
shipconsole@B4XSGX3:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND       CREATED             STATUS                          PORTS     NAMES
620dc85eee8d   ubuntu                   "/bin/bash"   8 minutes ago       Exited (127) 44 seconds ago               hopeful_shamir
30747c569424   alpine                   "/bin/sh"     25 minutes ago      Up 15 minutes                             condescending_shockley
2de43af94585   ubuntu                   "/bin/bash"   About an hour ago   Exited (129) About an hour ago            charming_noyce
2e3a39e615c0   hello-world              "/hello"      2 hours ago         Exited (0) 14 minutes ago                 wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"      2 hours ago         Exited (1) 16 minutes ago                 beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"   10 hours ago        Exited (0) 18 minutes ago                 wizardly_cartwright
shipconsole@B4XSGX3:~$ docker start 620dc85eee8d
620dc85eee8d
shipconsole@B4XSGX3:~$ docker ps -a
CONTAINER ID   IMAGE                    COMMAND       CREATED             STATUS                          PORTS     NAMES
620dc85eee8d   ubuntu                   "/bin/bash"   9 minutes ago       Up 2 seconds                              hopeful_shamir
30747c569424   alpine                   "/bin/sh"     26 minutes ago      Up 16 minutes                             condescending_shockley
2de43af94585   ubuntu                   "/bin/bash"   About an hour ago   Exited (129) About an hour ago            charming_noyce
2e3a39e615c0   hello-world              "/hello"      2 hours ago         Exited (0) 14 minutes ago                 wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"      2 hours ago         Exited (1) 17 minutes ago                 beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"   10 hours ago        Exited (0) 19 minutes ago                 wizardly_cartwright
shipconsole@B4XSGX3:~$
```

Architecture of Docker :



docker run -it -p 8999:80 nginx

```
shipconsole@B4XSGX3: $ docker ps -a
CONTAINER ID   IMAGE                    COMMAND                  CREATED             STATUS                      PORTS                                     NAMES
a1df9a4b7f14   nginx                    "/docker-entrypoint.…"   5 seconds ago       Up 4 seconds                0.0.0.0:8999->80/tcp, :::8999->80/tcp     agitated_pascal
9386495bbc1a   nginx                    "/docker-entrypoint.…"   2 minutes ago       Exited (0) 14 seconds ago                                             focused_merkle
606b0cade474   nginx                    "/docker-entrypoint.…"   5 minutes ago       Exited (0) 2 minutes ago                                              zealous_brattain
2bf2a7129a8e   nginx                    "/docker-entrypoint.…"   8 minutes ago       Exited (0) 8 minutes ago                                              naughty_feistel
620dc85eee8d   ubuntu                   "/bin/bash"              50 minutes ago      Exited (137) 39 minutes ago                                           hopeful_shamir
30747c569424   alpine                   "/bin/sh"                About an hour ago   Up 57 minutes                                                         condescending_shockley
2de43af94585   ubuntu                   "/bin/bash"              2 hours ago         Exited (129) 2 hours ago                                              charming_noyce
2e3a39e615c0   hello-world              "/hello"                 2 hours ago         Exited (0) 56 minutes ago                                             wizardly_goldstine
0309837ffb22   atlassian/hello-world    "/hello"                 2 hours ago         Exited (1) 58 minutes ago                                             beautiful_dirac
42a80bcf5638   ubuntu                   "/bin/bash"              11 hours ago        Exited (0) About an hour ago                                          wizardly_cartwright
shipconsole@B4XSGX3: $
```

Now we can access http://localhost:8999/

It tells Docker:

"Take traffic coming to host_port on my computer (or server) and send it to container_port inside the container."

-P = port

📌 **Why is it needed?**

Because **containers are isolated**, they don't expose ports to your host by default. Port mapping lets you **access the app running inside the container** from your browser or tools like Postman.

Inside the container: NGINX listens on port **80**

On your machine: You access it at http://localhost:8999

1. docker logs <container_id>

- 📌 **Use:** Shows the logs/output of a running container.
- ✅ **Example:** Useful for debugging errors in your app.
- □ **Example:** docker logs my-app

```
shipconsole@B4XSGX3:~$ docker logs 0a01d4181b92
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
```

2. docker exec -it <container_id> <command>

- 📌 **Use:** Runs a command inside a running container.
- ✅ **Example:** You can open a terminal inside the container.
- □ **Example:** docker exec -it my-app bash → opens shell

3. docker inspect <container_id>

- 📌 **Use:** Shows detailed info about a container or image in JSON.
- ✅ **Example:** Useful to check IP, mounted volumes, env variables, etc.
- □ **Example:** docker inspect my-app

```
shipconsole@B4XSGX3:~$ docker inspect 1fd17d784693
[
    {
        "Id": "1fd17d7846930647016512877 9d1671f121a7739b6e15b2f625fe7c2100cce75",
        "Created": "2025-05-06T16:10:44.777250892Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
```

4. docker network ls / docker network inspect <network_name>

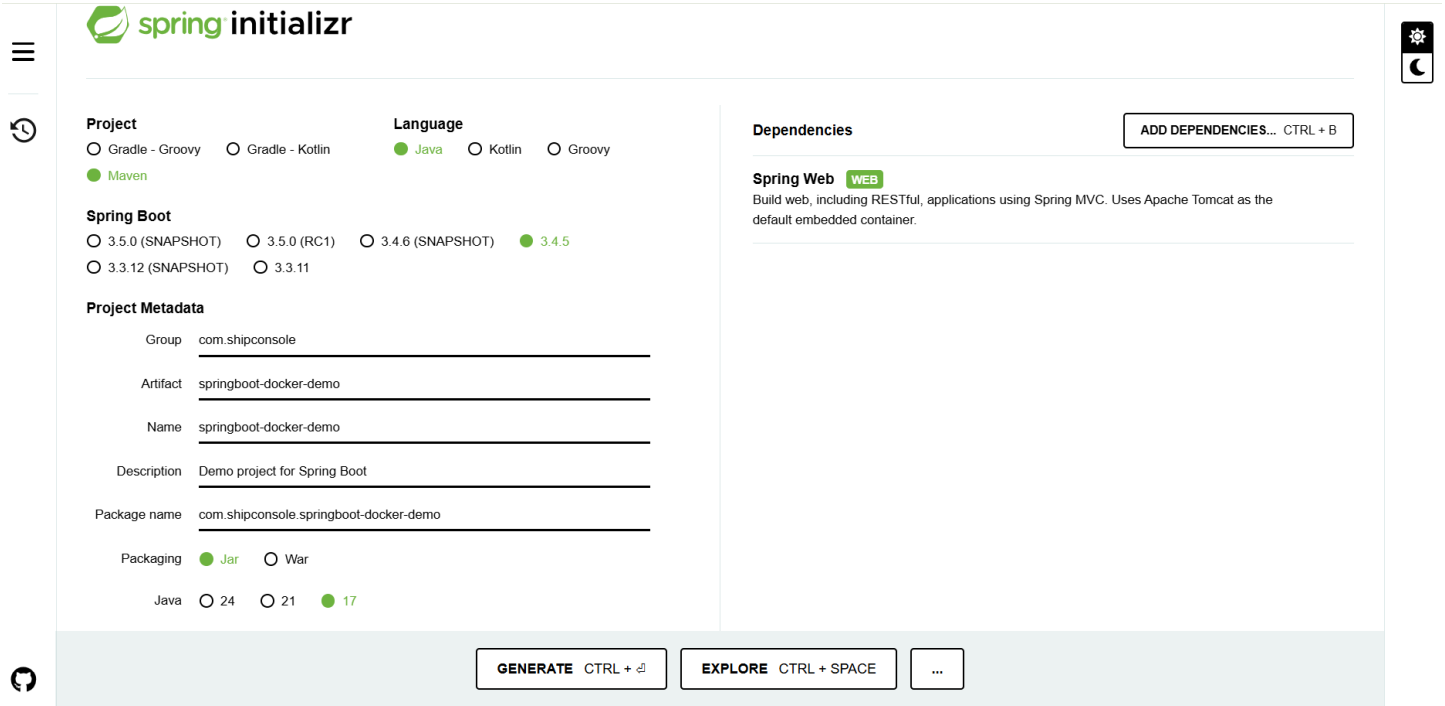
- 📌 **Use:** Lists and shows details about Docker networks.
- ✅ **Example:** Helps in understanding how containers communicate.
- □ **Example:**
  - docker network ls → list networks
  - docker network inspect bridge → details of 'bridge' network

📘 How to Build and Run a Docker Image for a Spring Boot Application

✅ 1. Create a Spring Boot Application

- Use **Spring Initializr** or your IDE.
- Add dependencies like: Spring Web, Spring Boot DevTools.

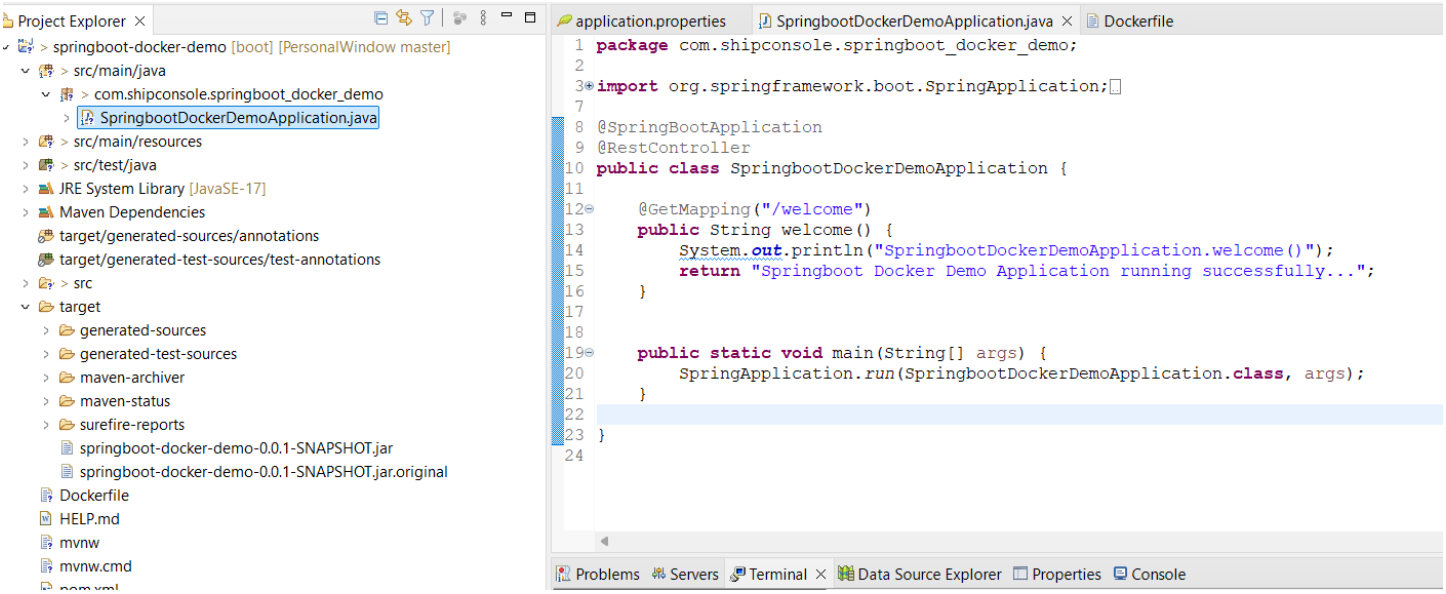
Open : **Spring Initializr** and select the Spring Boot Version and Maven, Java version , dependency, group name and artifact name.



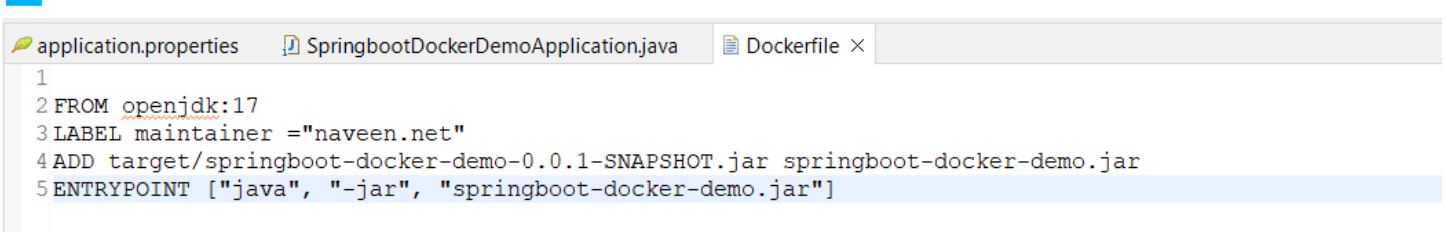
Click on **GENERATE** CTRL button

Import into Eclipse.

✅ 2. Build a Simple REST API



✅ 3. Create a Dockerfile in the root folder:



✅ 4. Build the JAR & Docker Image

```
E:\PersonalWindow\DockerPOC\springboot-docker-demo>mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------< com.shipconsole:springboot-docker-demo >---------------
[INFO] Building springboot-docker-demo 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.4.1:clean (default-clean) @ springboot-docker-demo ---
[INFO] Deleting E:\PersonalWindow\DockerPOC\springboot-docker-demo\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ springboot-docker-demo ---
[INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ springboot-docker-demo ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 1 source file with javac [debug parameters release 17] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ springboot-docker-demo ---
[INFO] skip non existing resourceDirectory E:\PersonalWindow\DockerPOC\springboot-docker-demo\src\test\resources
```

The jar will be generated in your project's target folder.

- target
  - generated-sources
  - generated-test-sources
  - maven-archiver
  - maven-status
  - surefire-reports
  - springboot-docker-demo-0.0.1-SNAPSHOT.jar
  - springboot-docker-demo-0.0.1-SNAPSHOT.jar.original
- Dockerfile

Then, open the **Ubuntu terminal** and navigate to your project directory.

```
shipconsole@B4XSGX3:/mnt/e/PersonalWindow/DockerPOC/springboot-docker-demo$
```

✅ 5. 🐳 Build the Docker image

docker build -t springboot-docker-demo:latest .

✅ 6. Run Docker Container

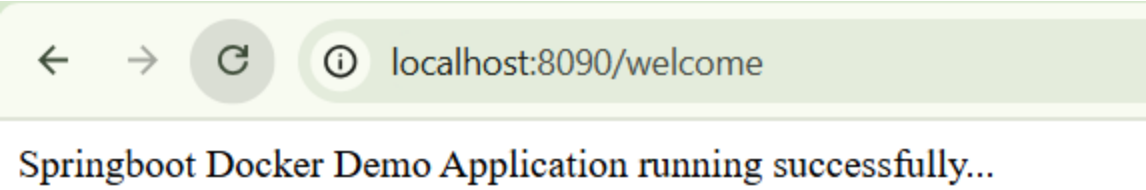
▶ **Foreground (show logs):**

docker run -p 8090:8060 springboot-docker-demo

🕶 **Background (detached mode):**

docker run -d -p 8090:8060 springboot-docker-demo

Now you can access your **API at:** http://localhost:8090/hello



✅ 6. Demo & Testing

🔍 **Check container:**

docker ps – Displays only running containers.

docker ps -a – Displays **all containers**, including **stopped ones**.

```
shipconsole@B4XSGX3:/mnt/e/PersonalWindow/DockerPOC/springboot-docker-demo$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                       NAMES
eafee9a02c34   springboot-docker-demo   "java -jar springboo…"   31 minutes ago   Up 31 minutes   0.0.0.0:8090->8060/tcp, :::8090->8060/tcp   quizzical_shtern
shipconsole@B4XSGX3:/mnt/e/PersonalWindow/DockerPOC/springboot-docker-demo$
```

Remove all containers (including stopped ones):

docker rm -f $(docker ps -aq)

q = ID's

Remove all images:

docker rmi -f $(docker images -q)

rmi = Remove images

What is Docker Compose?

Docker Compose is a tool for defining and managing multi-container Docker applications. It allows you to configure your application's services, networks, and volumes using a single YAML file (docker-compose.yml), making it easy to start and manage containers with a single command.

Key Features

- **Multi-container management:** Easily manage multiple containers that need to work together.
- **Environment setup:** Define networks, volumes, and environment variables.
- **Scaling:** Scale services up or down with a single command.
- **Simplified orchestration:** Start, stop, and manage all services with one command.

📦 Install Docker Compose on Ubuntu

✅ Step 1: Update Package Index

sudo apt update

✅ Step 2: Install Docker Compose

Install Docker Compose using the following command:

sudo apt-get install docker-compose

✅ Step 3: Verify Installation

To confirm that Docker Compose was installed successfully, run:

docker-compose –version

🐳 Docker Compose Setup for Microservices (Eureka + User + Client)

This guide explains how to set up a multi-container Spring Boot microservices project using Docker Compose with:

- **Eureka Server**
- **User Service**
- **Client Service**

🗂 Directory Structure

Your project structure should look like this:

```
project-root/
├── docker-compose.yml
├── eureka-server/
│   └── Dockerfile
├── user-service/
│   └── Dockerfile
├── client-service/
│   └── Dockerfile
```

Create a **docker-compose.yml** file to define your services.

docker-compose.yml ×

E: > PersonalWindow > DockerPOC > docker-compose.yml

```yaml
version: '3.8'

services:
  eureka-server:
    build:
      context: ./eureka-server  # Path to the folder containing the Dockerfile for Eureka Server
      dockerfile: Dockerfile  # Default Dockerfile name
    container_name: eureka-server
    ports:
      - "8762:8762"  # Map port 8762 on the container to port 8762 on the host
    networks:
      - mynetwork
```

```yaml
  user-service:
    build:
      context: ./user-service  # Path to the folder containing the Dockerfile for User Service
      dockerfile: Dockerfile  # Use the default Dockerfile name inside user-service folder
    container_name: user-service
    ports:
      - "8065:8065"  # Map port 8065 on the container to port 8065 on the host
    depends_on:
      - eureka-server  # Ensure Eureka Server is up before User Service starts
    environment:
      - SPRING_APPLICATION_NAME=user-service
      - SERVER_PORT=8065
      - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://eureka-server:8762/eureka
    networks:
      - mynetwork
```

```yaml
  client-service:
    build:
      context: ./client-service  # Path to the folder containing the Dockerfile for Client Service
      dockerfile: Dockerfile  # Default Dockerfile name
    container_name: client-service
    ports:
      - "8055:8055"  # Map port 8055 on the container to port 8055 on the host
    depends_on:
      - eureka-server  # Ensure Eureka Server is up before Client Service starts
    environment:
      - SPRING_APPLICATION_NAME=client-service
      - SERVER_PORT=8055
      - EUREKA_CLIENT_SERVICEURL_DEFAULTZONE=http://eureka-server:8762/eureka
    networks:
      - mynetwork

networks:
  mynetwork:
    driver: bridge  # Create a bridge network for the services to communicate
```

Run your application with the command:

docker-compose up -d

Stop your application with:

docker-compose down

✅ Custom Image Name Example

To build a Docker image for your user service:

docker build -t dockerpoc_user-service:1.0 .

dockerpoc_user-service → Your custom image name.

1.0 → Version tag (optional).

✅ Run a Container with Custom Name

Now run a container using this image and give the container a meaningful name too:

docker run --name user-service-container -p 8080:8080 dockerpoc_user-service:1.0

--name user-service-container → Custom container name.

-p 8080:8080 → Maps container's port 8080 to your local port 8080.

🐳 Docker

- Docker is a platform that allows you to build, run, and manage containers.
- Think of Docker as the engine or tool that manages everything related to containers.

Analogy: Like a smartphone OS that runs apps (containers).

📄 Image

- A Docker image is a **blueprint or template**.
- It contains everything needed to run an application: code, libraries, environment variables, and settings.
- Images are **read-only**.

Analogy: Like a recipe in a cookbook — it tells you how to make a dish.

📄➡🚀 Container

- A container is a **running instance** of an image.
- It's **lightweight**, isolated, and includes the application and all its dependencies.
- You can start, stop, and delete containers.

Analogy: Like the actual dish made from the recipe.

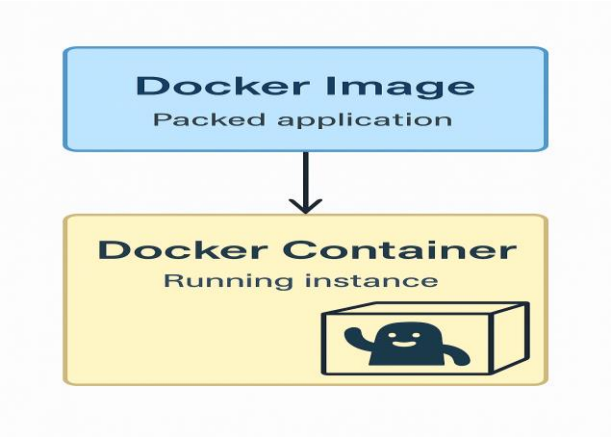
🗂 Docker Compose

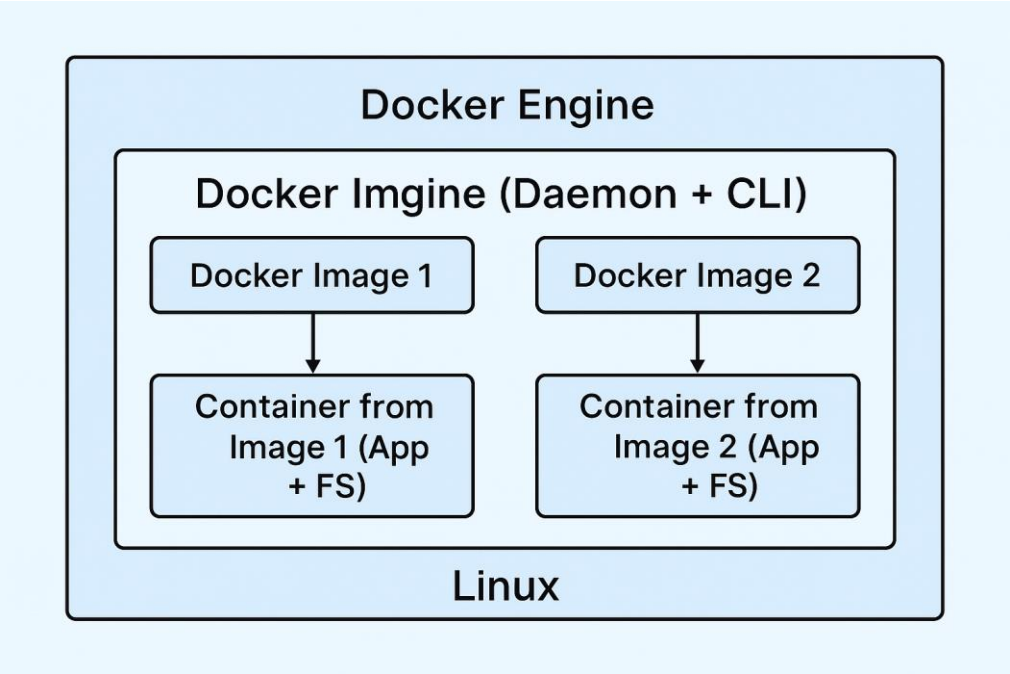
- Docker Compose is a **tool** that helps manage **multi-container** applications.
- You define all your services (e.g., app, database, cache) in one file: **docker-compose.yml.**
- With one command, you can start or stop everything together.

Analogy: Like a meal plan that uses multiple recipes together.

Container Vs Image Vs Docker Compose :

Feeture	Image	Container	Docker Compose
Definition	Template for containers	Running instance of an image	Tool to define/run multi-container apps
State	Static (read-only)	Dynamic (read/write)	Manages multi-container apps
Usage	Built once, used multiple times	Executes image to do work	Stored as docker-compose.yml
Storage	Stored in registries (e.g., Docker Hub)	Lives on host system while running	Stored as docker-compose.uml
Lifecycle	Created with `docker build`'	Created with docker run'	Uses 'docker-compose up/down'
Example	myapp:latest	myapp_container_1	Defines services: web, db, etc.





Volumes in Docker.?

Docker volumes are special storage locations managed by Docker that let containers save and access data outside their internal filesystem. This means the data stays safe even if the container is stopped, removed, or recreated.

Key Reasons to Use Volumes:

1. **Data Persistence**: Keeps data even after the container stops or is removed.
2. **Data Sharing**: Share data between containers.
3. **Separation of Concerns**: Keeps application code and data separate.
4. **Backups and Migration**: Volumes can easily be backed up, restored, or moved.
5. **Performance**: Volumes are managed by Docker and often perform better than bind mounts.

3. Good `docker-compose.yml` structure with volumes

yaml Copy Edit

```yaml
services:
  user-service:
    volumes:
      - user-service-logs:/app/logs

  client-service:
    volumes:
      - client-service-logs:/app/logs

volumes:
  user-service-logs:
  client-service-logs:
```