

# SPRING BOOT DOCKER

## BASICS



**ROHAN THAPA**

**THAPAROHAN2019@GMAIL.COM**

# What is a Docker?

**Docker** is a platform for **developing, shipping,** and **running** applications inside containers.

**Containers** are isolated environments that encapsulate everything an application needs to run, including the code, runtime, libraries, and system tools, ensuring that the application runs consistently across different computing environments.

- Containerization isolates applications, avoiding conflicts between libraries and dependencies.
- Portability is one of Docker's primary advantages, meaning you can "build once, run anywhere."

# Why Docker?

- **Consistency Across Environments:** Applications in Docker containers will run the same regardless of the environment (development, testing, production).
- **Efficiency:** Containers use fewer resources than virtual machines because they don't bundle a full operating system.
- **Fast Deployment:** Containers can be quickly created, deployed, and scaled, facilitating continuous integration and delivery (CI/CD).
- **Isolation:** Each Docker container runs in isolation from others, avoiding dependency conflicts.
- **Scalability:** Containers can be easily scaled across clusters of machines.

# Key Docker Concepts

## Docker Containers

- Containers are lightweight, stand-alone, executable packages of software that include everything needed to run a piece of software: code, runtime, system tools, libraries, and settings.
- Difference from VMs: Containers share the same OS kernel, unlike VMs, which virtualize entire hardware, making containers faster and more lightweight.

# Key Docker Concepts


## Docker Images


- Images are the blueprints of containers. They include the application and its dependencies but are immutable and read-only. Containers are instances of images.
- Layers: Images are built in layers, with each command in a Dockerfile creating a new layer. This layering allows for efficient storage and reuse of common layers.

# Key Docker Concepts

## Dockerfile

- A Dockerfile is a script containing a set of instructions that Docker uses to create a container image. Each instruction in a Dockerfile creates a new layer in the image.

 Dockerfile X

 Dockerfile > ...

```
1 FROM openjdk:22
2 WORKDIR /usr/src/app
3 COPY . /usr/src/app
4 EXPOSE 9595
5 CMD ["java" , "-jar" , "ConsumingRestAPI-0.0.1-SNAPSHOT.jar"]
```

# Common Dockerfile Instruction

## 1. FROM :

Specifies the base image for the Docker image.

```
FROM node:14-alpine
```

## 2.LABEL

Adds metadata to an image in the form of key-value pairs.

```
LABEL version="1.0"
```

## 3.WORKDIR

Sets the working directory inside the container for subsequent commands.

```
WORKDIR /app
```

## 4.COPY

Copies files or directories from the host filesystem to the container.

```
COPY . /app
```

# Common Dockerfile Instruction

## 5.ADD

Similar to COPY, but also supports extracting tar files and downloading URLs.

```
ADD https://example.com/file.tar.gz /app
```

## 6.RUN

Executes commands inside the container during the build process.

```
RUN apt-get update && apt-get install -y curl
```

## 7.CMD

Specifies the default command to run when a container starts (overridden by docker run command).

```
CMD ["node", "server.js"]
```

## 8.ENTRYPOINT

Configures the container to run as an executable with a command that can't be overridden.

```
ENTRYPOINT ["python", "app.py"]
```



# Common Dockerfile Instruction

## 9.ENV

Sets environment variables inside the container.

```
ENV NODE_ENV=production
```

## 10.ARG

Defines build-time variables (accessible only during the build process).

```
ARG VERSION=1.0
```

## 11.EXPOSE

Informs Docker that the container listens on specific network ports.

```
EXPOSE 8080
```

## 12.VOLUME

Defines a mount point with a specific path in the container to persist data.

```
VOLUME /app/data
```

# Key Docker Concepts

## Docker Hub

- Docker Hub is a cloud-based repository where Docker users can push and pull container images. It's the default registry used by Docker, offering both official images and user-contributed images.

## Docker Volumes

- Volumes are used to persist data generated by a container. Volumes are stored outside the container filesystem, so they aren't tied to the lifecycle of a container.

# Key Docker Concepts

## Docker Networks

- Docker allows containers to communicate with each other using different networking modes, including:
  - **Bridge:** Default, allows containers to communicate on the same host.
  - **Host:** Shares the host's network stack.
  - **Overlay:** Enables swarm services to communicate across multiple Docker daemons.

# Docker Architecture

**Docker has a client-server architecture:**

- **Docker Client:** The primary user interface for Docker. Users interact with Docker by issuing commands using the Docker CLI.
- **Docker Daemon:** Responsible for building, running, and managing containers.
- **Docker Registry:** A place where Docker images are stored. Docker Hub is a public registry.
- **Docker Objects:** Images, containers, networks, and volumes are the objects Docker uses.

# Core Docker Operations

## Basic Docker Commands

- **docker pull:** Fetches an image from a registry.
- **docker build:** Builds an image from a Dockerfile.
- **docker run:** Runs a container from an image.
- **docker ps:** Lists running containers.
- **docker stop:** Stops a running container.
- **docker rm:** Removes a stopped container.
- **docker exec:** Runs a command inside a running container

# Basic Docker Commands

check version

```
C:\Users\RohanThapa>docker -v
Docker version 27.0.3, build 7d4bcd8
```

pull image from dockerhub

```
C:\Users\RohanThapa>docker pull openjdk
Using default tag: latest
latest: Pulling from library/openjdk
197c1adcd755: Pull complete
57b698b7af4b: Pull complete
95a27dbe0150: Pull complete
Digest: sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
Status: Downloaded newer image for openjdk:latest
docker.io/library/openjdk:latest
```

list images

```
C:\Users\RohanThapa>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
notes-management-system.jar	nms	03e69a3633ec	4 weeks ago	595MB
rohanthapa02/notes-management-system	v1.0.1	03e69a3633ec	4 weeks ago	595MB
rohanthapa01/notes-management-system	v1.0	9f9da916b554	4 weeks ago	595MB
rohanthapa02/notes-management-system	v1.0	9f9da916b554	4 weeks ago	595MB
hello-world	latest	d2c94e258dcb	16 months ago	13.3kB
openjdk	latest	71260f256d19	19 months ago	470MB

# Basic Docker Commands

## search images

```
C:\Users\RohanThapa>docker search nginx
```

NAME	DESCRIPTION	STARS	OFFICIAL
nginx	Official build of Nginx.	20184	[OK]
nginx/nginx-quic-qns	NGINX QUIC interop	1	
nginx/nginx-prometheus-exporter	NGINX Prometheus Exporter for NGINX and NGIN...	43	
nginx/nginx-ingress	NGINX and NGINX Plus Ingress Controllers fo...	94	
nginx/nginx-ingress-operator	NGINX Ingress Operator for NGINX and NGINX P...	2	
nginx/unit	This repository is retired, use the Docker o...	63	
nginx/unit-preview	Unit preview features	0	
bitnami/nginx	Bitnami container image for NGINX	193	
rapidfort/nginx	RapidFort optimized, hardened image for NGINX	15	

## run the image

```
C:\Users\RohanThapa>docker run --name javaContainer1 -it -d openjdk
89cb44a266bd5e049437ab4b7738a4a567b281b7aaae1fc7dc96bc974fc9627
```

```
C:\Users\RohanThapa>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
89cb44a266bd	openjdk	"jshell"	6 seconds ago	Up 4 seconds		javaContainer1

```
C:\Users\RohanThapa>docker exec -it 89cb44a266bd jshell
| Welcome to JShell -- Version 18.0.2.1
| For an introduction type: /help intro

jshell> System.out.println("Hello Rohan");
Hello Rohan

jshell>
```



# Basic Docker Commands

## view container and stop container

```
C:\Users\RohanThapa>docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
aeccc6147b05   httpd     "httpd-foreground"      36 seconds ago Up 35 seconds  0.0.0.0:8081->80/tcp               apacheServer
338516341b2e   nginx     "/docker-entrypoint...." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               nginxServer1
60b91e94fd3a   mysql     "docker-entrypoint.s..." 9 minutes ago  Up 9 minutes  3306/tcp, 33060/tcp               mysqlDb
3f7e6cc03c53   python   "python3"               15 minutes ago Up 15 minutes  pythonContainer
89cb44a266bd   openjdk   "jshell"                24 minutes ago Up 24 minutes  javaContainer1

C:\Users\RohanThapa>docker stop pythonContainer
pythonContainer

C:\Users\RohanThapa>docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
aeccc6147b05   httpd     "httpd-foreground"      56 seconds ago Up 55 seconds  0.0.0.0:8081->80/tcp               apacheServer
338516341b2e   nginx     "/docker-entrypoint...." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               nginxServer1
60b91e94fd3a   mysql     "docker-entrypoint.s..." 10 minutes ago Up 10 minutes  3306/tcp, 33060/tcp               mysqlDb
89cb44a266bd   openjdk   "jshell"                25 minutes ago Up 25 minutes  javaContainer1
```

## remove the container

```
C:\Users\RohanThapa>docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
49e8f308c210   openjdk   "jshell"                30 minutes ago Exited (0) 30 minutes ago         openjdkContainer
ce667cc1727c   openjdk   "jshell"                33 minutes ago Exited (0) 33 minutes ago         lucid_jackson
90f6bc35a787   hello-world "/hello"               51 minutes ago Exited (0) 51 minutes ago         peaceful_black
f98a3242a019   9f9da916b554 "java -jar /app.jar"    4 weeks ago   Exited (130) 4 weeks ago         strange_robinson

C:\Users\RohanThapa>docker rm 49e8f308c210 ce667cc1727c 90f6bc35a787 f98a3242a019
49e8f308c210
ce667cc1727c
90f6bc35a787
f98a3242a019

C:\Users\RohanThapa>docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
```



# Basic Docker Commands

remove the image

```
C:\Users\RohanThapa>docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
python              latest       ea2ebd905ab2     3 days ago     1.01GB
nginx               latest       39286ab8a5e1     4 weeks ago     188MB
notes-management-system.jar    nms         03e69a3633ec     4 weeks ago     595MB
rohanthapa02/notes-management-system  v1.0.1     03e69a3633ec     4 weeks ago     595MB
rohanthapa01/notes-management-system  v1.0       9f9da916b554     4 weeks ago     595MB
rohanthapa02/notes-management-system  v1.0       9f9da916b554     4 weeks ago     595MB
mysql               latest       680b8c60dce6     7 weeks ago     586MB
httpd               latest       9cb0a2315602     8 weeks ago     148MB
hello-world         latest       d2c94e258dcb     16 months ago   13.3kB
openjdk             latest       71260f256d19     19 months ago   470MB

C:\Users\RohanThapa>docker rmi python
Untagged: python:latest
Untagged: python@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe
Deleted: sha256:ea2ebd905ab246ece277be25520ca0cfe82758b3d2b369e2fd69b374c1d6c7fa
Deleted: sha256:9658a1108394ea7cf7259db602cbd992c854cf619bb59e2bfb214cb1dcc2beb4
Deleted: sha256:51ecd3d6a46387e7db55cecc99f35fdfcc5c979bfff3ec945b1ccd91cb605b9
Deleted: sha256:de416e01626049422823c0ec98214e6ed47cb0d37807fd61a07b43787ff00a01
Deleted: sha256:c5142b72a47981685608282fe9119b4bd9fdcee17f5e43d167535b5a5c29d650
Deleted: sha256:2f95d34d0b116436dc087782907bb0cc2b70149cfc6c1aec59ad8c46e9cb9ce6
Deleted: sha256:a4f266ef9aeadd75cff7caa4c6e879fcbf4304749c1534ef2a97cbb694597b8c3
Deleted: sha256:6abe10f2f60150ae9768e117986e4f8af5546137221553228eb5d21066f596b4
```

# Basic Docker Commands

## **docker login :**

Logs into a Docker registry (usually Docker Hub). You will be prompted to enter your credentials.

**docker logout** : Logs out from a Docker registry.

# Core Docker Operations

## Managing Containers

- `docker logs`: Retrieves logs from a container.
- `docker stats`: Shows resource usage statistics for containers.

## Docker Compose

- Docker Compose is used to define and manage **multi-container applications**. You create a **`docker-compose.yml`** file, which defines services, networks, and volumes.

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

# Advanced Docker Concepts

## Docker Swarm

- **Docker Swarm** is Docker's native **clustering** and **orchestration** tool. It allows Docker hosts to group together into a cluster, where containers can be scaled and managed across multiple machines.

## Kubernetes

- **Kubernetes** is an open-source container orchestration platform, which is often used with Docker to automate the deployment, scaling, and management of containerized applications across clusters of machines.

# Advanced Docker Concepts

## Docker in CI/CD

- Docker plays a crucial role in Continuous Integration and Continuous Delivery pipelines. It allows teams to run automated tests, build images, and deploy the same container to any environment (dev, staging, production), ensuring consistency.

# Docker Security

Docker offers several layers of security:

- **Namespaces:** Provide isolation for a container's processes, users, file system, and network.
- **Control Groups (Cgroups):** Limit resources a container can consume (CPU, memory, etc.).
- **Seccomp Profiles:** Limit system calls that containers are allowed to make, reducing potential attack vectors.

# Example:

Here's a basic example of how to containerize a **Spring Boot application** and run it with Docker:

## 1. Create a Spring Boot Application

```
@RestController  no usages
@RequestMapping("/api")
public class HelloController {

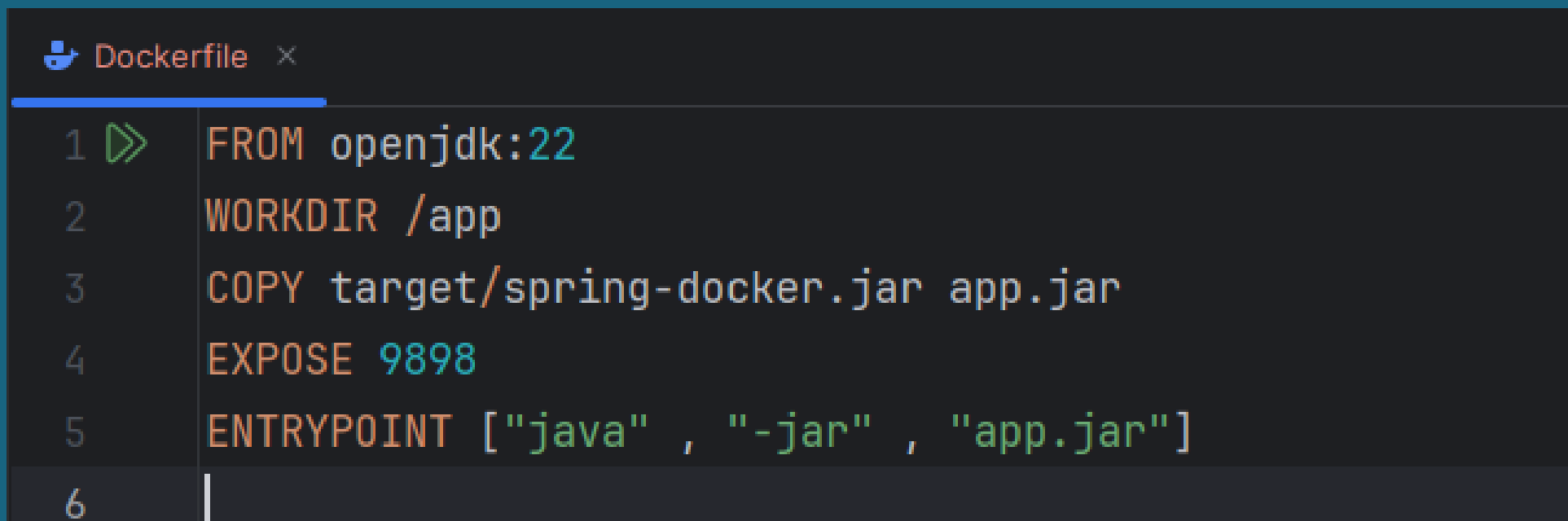
    |   @GetMapping()  no usages
    |   public String greet(){
    |       |   return "Hello this is rohan";
    |       |   }
    |   }
}
```

# Example:

## 2. Create a Dockerfile

Next, we need to create a **Dockerfile**. This file contains the instructions Docker uses to create the container.

In the root directory of your Spring Boot application (where **pom.xml** or **build.gradle** is located), create a file called Dockerfile:

A screenshot of a code editor window titled "Dockerfile" with a close button (X) in the top right corner. The editor shows a Dockerfile with six lines of code. Line 1: FROM openjdk:22. Line 2: WORKDIR /app. Line 3: COPY target/spring-docker.jar app.jar. Line 4: EXPOSE 9898. Line 5: ENTRYPOINT ["java" , "-jar" , "app.jar"]. Line 6: A blank line with a cursor at the end. The code is color-coded: FROM, WORKDIR, COPY, and ENTRYPOINT are in orange; openjdk:22, 9898, and the array elements are in green; and the paths and arguments are in white.

```
1 >> FROM openjdk:22
2     WORKDIR /app
3     COPY target/spring-docker.jar app.jar
4     EXPOSE 9898
5     ENTRYPOINT ["java" , "-jar" , "app.jar"]
6
```



# Example:

Explanation:

- **Firstly** we add a **finalName** tag in pom.xml to set the fix name for .jar file.

```
</plugins>  
<finalName>spring-docker</finalName>  
}
```

- Then we compile the Spring Boot application and create a **.jar** file.
- Then we use a lightweight OpenJDK image to run the **.jar** file. The resulting container will run on Java 22, and expose port 9898

# Example:

## 3. Build the Docker Image

After creating the Dockerfile, you can now build the Docker image.

Open a terminal in the root directory of your Spring Boot application and run the following command:

```
C:\Users\RohanThapa\Desktop\learnjava\dockertest>docker build -t springbootimage .
[+] Building 4.3s (8/8) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile             0.1s
=> => transferring dockerfile: 162B                             0.1s
=> [internal] load metadata for docker.io/library/openjdk:22   3.7s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [1/3] FROM docker.io/library/openjdk:22@sha256:b7d44427f4622d3f6b9a60583e5218ecfa8b4e44f3e01dfd0d9b7d7abba31c9a 0.0s
=> [internal] load build context                               0.0s
=> => transferring context: 74B                                    0.0s
=> CACHED [2/3] WORKDIR /app                                    0.0s
=> CACHED [3/3] COPY target/spring-docker.jar app.jar          0.0s
=> exporting to image                                           0.1s
=> => exporting layers                                           0.0s
=> => writing image sha256:263a50496469731ac5b60667356b1c27f4dc701e51c92f5bd851d88cea5bfbd3 0.0s
=> => naming to docker.io/library/springbootimage              0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/yfdpxbfc0zjdoygbicd39dkf9
```

This command builds the Docker image from the **Dockerfile** and tags it as **springbootimage**.

```
C:\Users\RohanThapa\Desktop\learnjava\dockertest>docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
springbootimage     latest       263a50496469     8 hours ago     577MB
```

# Example:

## 4. Run the Docker Container

Now that we have the **Docker image**, we can run it in a container. Run the following command:

```
C:\Users\RohanThapa\Desktop\learnjava\dockertest>docker run --name springbootcontainer -it -p 9898:8080 springbootimage
```



:: Spring Boot :: (v3.3.3)

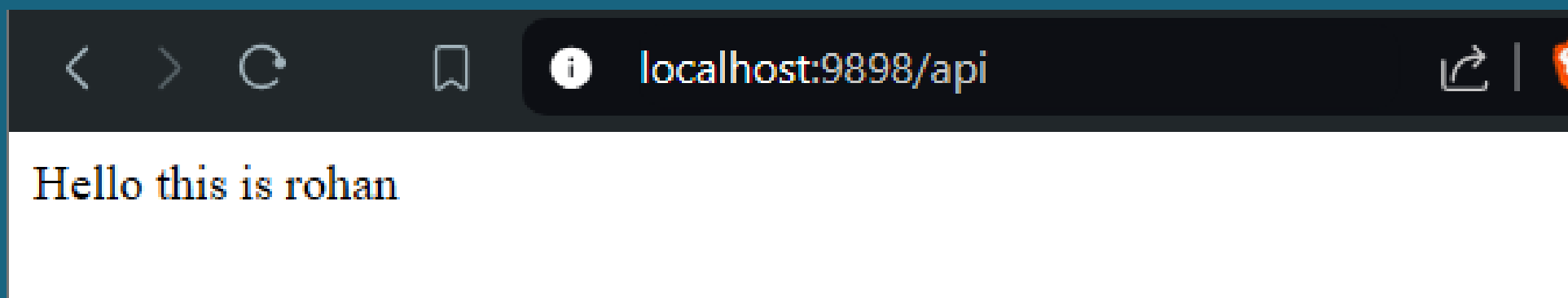
```
2024-09-14T11:57:32.365Z INFO 1 --- [dockertest] [main] c.d.dockertest.DockertestApplication : Starting DockertestApplica
tion v0.0.1-SNAPSHOT using Java 22 with PID 1 (/app/app.jar started by root in /app)
2024-09-14T11:57:32.371Z INFO 1 --- [dockertest] [main] c.d.dockertest.DockertestApplication : No active profile set, fal
ling back to 1 default profile: "default"
2024-09-14T11:57:34.092Z INFO 1 --- [dockertest] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with po
rt 8080 (http)
2024-09-14T11:57:34.120Z INFO 1 --- [dockertest] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-09-14T11:57:34.121Z INFO 1 --- [dockertest] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [
Apache Tomcat/10.1.28]
2024-09-14T11:57:34.190Z INFO 1 --- [dockertest] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedd
ed WebApplicationContext
2024-09-14T11:57:34.193Z INFO 1 --- [dockertest] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext
: initialization completed in 1696 ms
2024-09-14T11:57:34.777Z INFO 1 --- [dockertest] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 808
0 (http) with context path '/'
2024-09-14T11:57:34.825Z INFO 1 --- [dockertest] [main] c.d.dockertest.DockertestApplication : Started DockertestApplicat
ion in 3.358 seconds (process running for 4.84)
2024-09-14T11:58:04.427Z INFO 1 --- [dockertest] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring Dispat
cherServlet 'dispatcherServlet'
```

-p 9898:8080: This maps the container's port 8080 to your local machine's port 9898.

# Example:

## 5. Test the Application

Once the container is running, open your browser and go to `http://localhost:9898/api`. You should see the message:



# Docker vs Virtual Machines

Feature	Docker Containers	Virtual Machines
OS	Shares host OS kernel	Full OS inside each VM
Performance	Lightweight, fast startup	Heavyweight, slow startup
Size	MBs	GBs
Isolation	Process-level isolation	Full hardware-level isolation
Deployment	Faster deployments, more portable	Slower deployments

# Benefits of Dockerizing a Spring Boot Application

- **Consistency:** The application will behave the same in any environment because it includes all dependencies.
- **Scalability:** Docker containers can be easily replicated and deployed across multiple hosts.
- **Portability:** You can ship the application as a single image that can run anywhere Docker is installed.
- **Isolation:** Each Docker container runs in its own environment without affecting the host system or other containers.

# Conclusion

Docker has transformed the way we **develop, test, and deploy software**. It provides a **lightweight, scalable** solution for containerizing applications, ensuring consistency across environments.

Whether you're using Docker for local development, CI/CD, or managing production workloads, Docker's flexibility and speed make it an indispensable tool for modern software development.

With Docker, the phrase "**It works on my machine!**" becomes irrelevant — it works everywhere.

# Thank You

Rohan Thapa

[thaparohan2019@gmail.com](mailto:thaparohan2019@gmail.com)