

## Experiment No. 6

### Title:

Exploring Containerization and Application Deployment with Docker.

### Objective:

The objective of this experiment is to provide hands-on experience with Docker containerization and application deployment in a Docker container. By the end of this experiment, you will understand the basics of Docker, how to build Docker images, how to create Docker containers, and how to deploy a simple Java SpringBoot application.

### Introduction:

Containerization is a technology that has revolutionized the way applications are developed, deployed, and managed in the modern IT landscape. It provides a standardised and efficient way to package, distribute, and run software applications and their dependencies in isolated environments called containers.

Containerization technology has gained immense popularity, with Docker being one of the most well-known containerization platforms. This introduction explores the fundamental concepts of containerization, its benefits, and how it differs from traditional approaches to application deployment.

### Key Concepts of Containerization:

- **Containers:** Containers are lightweight, stand-alone executable packages that include everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers ensure that an application runs consistently and reliably across different environments, from a developer's laptop to a production server.
- **Images:** Container images are the templates for creating containers. They are read-only and contain all the necessary files and configurations to run an application. Images are typically built from a set of instructions defined in a Dockerfile.
- **Docker:** Docker is a popular containerization platform that simplifies the creation, distribution, and management of containers. It provides tools and services for building, running, and orchestrating containers at scale.

- **Isolation:** Containers provide process and filesystem isolation, ensuring that applications and their dependencies do not interfere with each other. This isolation enhances security and allows multiple containers to run on the same host without conflicts.

#### **Benefits of Containerization:**

- **Consistency:** Containers ensure that applications run consistently across different environments, reducing the "it works on my machine" problem.
- **Portability:** Containers are portable and can be easily moved between different host machines and cloud providers.
- **Resource Efficiency:** Containers share the host operating system's kernel, which makes them lightweight and efficient in terms of resource utilization.
- **Scalability:** Containers can be quickly scaled up or down to meet changing application demands, making them ideal for microservices architectures.
- **Version Control:** Container images are versioned, enabling easy rollback to previous application states if issues arise.
- **DevOps and CI/CD:** Containerization is a fundamental technology in DevOps and CI/CD pipelines, allowing for automated testing, integration, and deployment.

#### **Containerization vs. Virtualization:**

- Containerization differs from traditional virtualization, where a hypervisor virtualizes an entire operating system (VM) to run multiple applications.
- In contrast: Containers share the host OS kernel, making them more lightweight and efficient.
- Containers start faster and use fewer resources than VMs.
- VMs encapsulate an entire OS, while containers package only the application and its dependencies.

#### **Prerequisites:**

A computer with Docker, Java and Maven installed.

## Experiment Steps:

- In this experiment let us containerize and deploy a Java application.
- We are going to have the following folder structure.

```
springboot-docker-app/  
├── src/  
│   └── main/  
│       └── java/  
│           └── com/  
│               └── example/  
│                   └── demo/  
│                       └── DemoApplication.java  
├── pom.xml  
└── Dockerfile
```

### Step 1: Launch and Connect to EC2 Instance

(If already running, skip to Step 2)

1. Launch an Ubuntu `t2.large` EC2 instance.
2. Allow ports **22, 8080** (used by Spring Boot).
3. SSH into the instance:

```
ssh -i your-key.pem ubuntu@<EC2-Public-IP>
```

### Step 2: Install Java, Maven, and Docker

```
sudo apt update
```

```
sudo apt install -y openjdk-17-jdk maven docker.io
```

```
sudo systemctl status docker
```

```

ubuntu@ip-172-31-46-88:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Tue 2025-06-24 06:36:15 UTC; 16s ago
     TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 3907 (dockerd)
      Tasks: 9
     Memory: 20.7M (peak: 21.1M)
        CPU: 355ms
    CGroup: /system.slice/docker.service
            └─3907 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jun 24 06:36:15 ip-172-31-46-88 systemd[1]: Starting docker.service - Docker Application Container Engine...
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.076605047Z" level=info msg="Starting up"
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.080768483Z" level=info msg="OTEL tracing is not configured,
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.080986168Z" level=info msg="detected 127.0.0.53 nameserver,
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.170805289Z" level=info msg="Loading containers: start."
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.507664560Z" level=info msg="Loading containers: done."
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.528834473Z" level=info msg="Docker daemon" commit="27.5.1-0u
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.529015352Z" level=info msg="Daemon has completed initializat
Jun 24 06:36:15 ip-172-31-46-88 dockerd[3907]: time="2025-06-24T06:36:15.583422139Z" level=info msg="API listen on /run/docker.sock"
Jun 24 06:36:15 ip-172-31-46-88 systemd[1]: Started docker.service - Docker Application Container Engine.

ubuntu@ip-172-31-46-88:~$

```

Press Ctrl+C

`sudo usermod -aG docker ubuntu`

Logout and SSH again to apply group permissions:

`exit`

`ssh -i your-key.pem ubuntu@<EC2-Public-IP>`

```

ubuntu@ip-172-31-46-88:~$ sudo usermod -aG docker ubuntu ✓
ubuntu@ip-172-31-46-88:~$ exit ✓
logout ✓
Connection to ec2-65-2-149-159.ap-south-1.compute.amazonaws.com closed.
user@lenovo MINGW64 ~/Desktop/DevOps_Workshop
$ ssh -i "demo.pem" ubuntu@ec2-65-2-149-159.ap-south-1.compute.amazonaws.com
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Tue Jun 24 06:38:35 UTC 2025

System load:  0.09           Processes:            119
Usage of /:   42.2% of 6.71GB Users logged in:          0
Memory usage: 4%            IPv4 address for enx0: 172.31.46.88
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

32 updates can be applied immediately.
22 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Tue Jun 24 06:33:46 2025 from 1.7.21.45
ubuntu@ip-172-31-46-88:~$ |

```

### Step 3: Create Spring Boot App from Scratch

```
mkdir springboot-docker-app && cd springboot-docker-app
```

```

ubuntu@ip-172-31-46-88:~$ mkdir springboot-docker-app && cd springboot-docker-app
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

```
mkdir -p src/main/java/com/example/demo
```

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ mkdir -p src/main/java/com/example/demo
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

```
cd src/main/java/com/example/demo
```

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ cd src/main/java/com/example/demo
ubuntu@ip-172-31-46-88:~/springboot-docker-app/src/main/java/com/example/demo$ |

```

Create the main class:

```
vi DemoApplication.java
```

Paste the following:

```

package com.example.demo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.*;

@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/")
    public String home() {
        return "Hello from Dockerized Spring Boot App!";
    }
}

```

```

package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.*;

@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/")
    public String home() {
        return "Hello from Dockerized Spring Boot App!";
    }
}

```

#### Step 4: Create pom.xml

Go back to project root:

```
cd ~/springboot-docker-app
```

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app/src/main/java/com/example/demo$ cd ~/springboot-docker-app
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

Create the pom.xml file:

```
vi pom.xml
```

Paste the following:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>springboot-docker-app</name>
  <description>Simple Spring Boot Docker App</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

## Step 5: Build the JAR File

```
ls -la
```

```
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ ls -la
total 16
drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 24 06:43 .
drwxr-x--- 5 ubuntu ubuntu 4096 Jun 24 06:43 ..
-rw-rw-r-- 1 ubuntu ubuntu 1160 Jun 24 06:43 pom.xml
drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 24 06:41 src
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |
```

mvn clean package

```
[INFO] Replacing main artifact /home/ubuntu/springboot-docker-app/target/demo-0.0.1-SNAPSHOT.jar with
d dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to /home/ubuntu/springboot-docker-app/target/demo-0.0.1-SNAPSHOT.jar.original
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.154 s
[INFO] Finished at: 2025-06-24T06:45:01Z
[INFO] -----
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |
```

ls -la

```
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ ls -la
total 20
drwxrwxr-x 4 ubuntu ubuntu 4096 Jun 24 06:44 .
drwxr-x--- 6 ubuntu ubuntu 4096 Jun 24 06:44 ..
-rw-rw-r-- 1 ubuntu ubuntu 1160 Jun 24 06:43 pom.xml
drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 24 06:41 src
drwxrwxr-x 6 ubuntu ubuntu 4096 Jun 24 06:45 target
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |
```

ls -la ./target/

```
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ ls -la ./target/
total 19332
drwxrwxr-x 6 ubuntu ubuntu      4096 Jun 24 06:45 .
drwxrwxr-x 4 ubuntu ubuntu      4096 Jun 24 06:44 ..
drwxrwxr-x 3 ubuntu ubuntu      4096 Jun 24 06:44 classes
-rw-rw-r-- 1 ubuntu ubuntu 19763318 Jun 24 06:45 demo-0.0.1-SNAPSHOT.jar
-rw-rw-r-- 1 ubuntu ubuntu    2579 Jun 24 06:45 demo-0.0.1-SNAPSHOT.jar.original
drwxrwxr-x 3 ubuntu ubuntu      4096 Jun 24 06:44 generated-sources
drwxrwxr-x 2 ubuntu ubuntu      4096 Jun 24 06:45 maven-archiver
drwxrwxr-x 3 ubuntu ubuntu      4096 Jun 24 06:44 maven-status
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |
```

**Expected Output:**

target/demo-0.0.1-SNAPSHOT.jar



## Step 6: Create Dockerfile

vi Dockerfile

Paste the following:

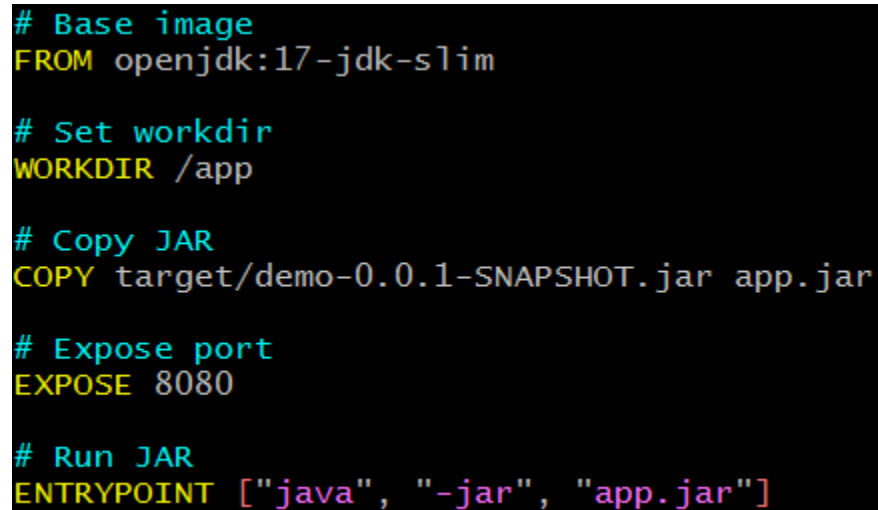
```
# Base image
FROM openjdk:17-jdk-slim

# Set workdir
WORKDIR /app

# Copy JAR
COPY target/demo-0.0.1-SNAPSHOT.jar app.jar

# Expose port
EXPOSE 8080

# Run JAR
ENTRYPOINT ["java", "-jar", "app.jar"]
```



```
# Base image
FROM openjdk:17-jdk-slim

# Set workdir
WORKDIR /app

# Copy JAR
COPY target/demo-0.0.1-SNAPSHOT.jar app.jar

# Expose port
EXPOSE 8080

# Run JAR
ENTRYPOINT ["java", "-jar", "app.jar"]
```

## Step 7: Build Docker Image

```
docker build -t springboot-docker-app .
```

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ docker build -t springboot-docker-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 19.79MB
Step 1/5 : FROM openjdk:17-jdk-slim
17-jdk-slim: Pulling from library/openjdk
1fe172e4850f: Pull complete
44d3aa8d0766: Pull complete
6ce99fdf16e8: Pull complete
Digest: sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc126b41f68c3f62f9774
Status: Downloaded newer image for openjdk:17-jdk-slim
--> 37cb44321d04
Step 2/5 : WORKDIR /app
--> Running in eaf98d65a080
--> Removed intermediate container eaf98d65a080
--> 65b652e49b65
Step 3/5 : COPY target/demo-0.0.1-SNAPSHOT.jar app.jar
--> e945c3595f22
Step 4/5 : EXPOSE 8080
--> Running in 5008dae287a2
--> Removed intermediate container 5008dae287a2
--> af82ca27eb5e
Step 5/5 : ENTRYPOINT ["java", "-jar", "app.jar"]
--> Running in b1d737ff56f4
--> Removed intermediate container b1d737ff56f4
--> 648dd96c7328
Successfully built 648dd96c7328
Successfully tagged springboot-docker-app:latest
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

docker images

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ docker images
REPOSITORY          TAG          IMAGE ID      CREATED        SIZE
springboot-docker-app latest       648dd96c7328  2 minutes ago  428MB
openjdk              17-jdk-slim 37cb44321d04  3 years ago   408MB
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

## Step 8: Run Docker Container

docker run -d -p 8080:8080 springboot-docker-app

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ docker run -d -p 8080:8080 springboot-docker-app
9b1675695e406f167ee1f5f2fab323ee732cf24e60008284ad7bca81fa18c86d
ubuntu@ip-172-31-46-88:~/springboot-docker-app$ |

```

docker ps

```

ubuntu@ip-172-31-46-88:~/springboot-docker-app$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
9b1675695e40   springboot-docker-app              "java -jar app.jar"     About a minute Up About a minute 0.0.0.0:8080->8080/tcp, :::8080-
>8080/tcp      modest_kowalevski
ubuntu@ip-172-31-46-88:~/springboot-docker-app$

```

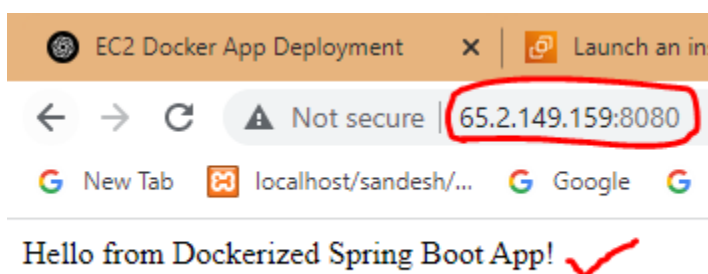
## Step 9: Test the Application

Open in browser:

`http://<EC2-Public-IP>:8080`

### Expected Output:

Hello from Dockerized Spring Boot App!



## Conclusion:

In this experiment, you explored containerization and application deployment with Docker by deploying a Java SpringBoot application in a Docker container. You learned how to create a Dockerfile, build a Docker image, run a Docker container, and access your Java SpringBoot application from your host machine. Docker's containerization capabilities make it a valuable tool for packaging and deploying applications consistently across different environments.

## Exercise/Questions:

1. Explain the concept of containerization. How does it differ from traditional virtualization methods?
2. Discuss the key components of a container. What are images and containers in the context of containerization?
3. What is Docker, and how does it contribute to containerization? Explain the role of Docker in building, running, and managing containers.
4. Describe the benefits of containerization for application deployment and management. Provide examples of scenarios where containerization is advantageous.

5. Explain the concept of isolation in containerization. How do containers provide process and filesystem isolation for applications?
6. Discuss the importance of container orchestration tools such as Kubernetes in managing containerized applications. What problems do they solve, and how do they work?
7. Compare and contrast containerization platforms like Docker, containerd, and rkt. What are their respective strengths and weaknesses?
8. Explain the process of creating a Docker image. What is a Dockerfile, and how does it help in image creation?
9. Discuss the security considerations in containerization. What measures can be taken to ensure the security of containerized applications?
10. Explore real-world use cases of containerization in software development and deployment. Provide examples of industries or companies that have benefited from containerization technologies.