Docker

By Eng: Abdelrahman Ayman Lotfy

DevOps track

Table of contents

Spring Boot Applications build tools

1. Maven

- **Style:** XML-based (pom.xml).
- **Strengths:**
  - Standardized, widely used.
  - Huge ecosystem of plugins and libraries.
  - Convention over configuration → predictable structure.
- **Weaknesses:**
  - XML can be verbose.
  - Slower compared to Gradle.
- **Dockerfile Example:**

```
FROM maven:3.9.6-eclipse-temurin-21 AS builder
COPY . .
RUN mvn clean package -DskipTests
```

2. Gradle

- **Style:** Groovy/Kotlin DSL (build.gradle).
- **Strengths:**
  - Faster builds (incremental + caching).
  - More flexible/customizable than Maven.
  - Popular for modern projects.
- **Weaknesses:**
  - Less predictable than Maven.
  - Slightly steeper learning curve.
- **Dockerfile Example:**

```
FROM gradle:8.10-jdk21 AS builder
COPY . .
RUN gradle build -x test
```

3. Ant

- **Style:** XML build scripts (build.xml).
- **Strengths:**
  - Very customizable.
  - Good for legacy projects.
- **Weaknesses:**
  - No built-in dependency management (you need Ivy).
  - Verbose and outdated compared to Maven/Gradle.
- **Dockerfile Example:**

```
FROM openjdk:21-jdk AS builder

COPY . .

RUN ant build
```

4. SBT (Scala Build Tool)

- **Style:** Scala DSL (build.sbt).
- **Strengths:**
  - Designed for Scala, but supports Java too.
  - Powerful dependency management.
- **Weaknesses:**
  - Heavier than Maven/Gradle for pure Java.

- **Dockerfile Example:**

```
FROM hseeberger/scala-sbt:21.0.2_1.10.0_3.5.0 AS builder

WORKDIR /app

COPY . .

RUN sbt package
```

5. Bazel (by Google)

- **Style:** Declarative (BUILD files).
- **Strengths:**
  - Extremely fast builds.
  - Supports multiple languages (not only Java).
  - Scales very well for large codebases.
- **Weaknesses:**
  - Steeper learning curve.
  - Smaller Java community compared to Maven/Gradle.
- **Dockerfile Example:**

```
FROM openjdk:21-jdk AS builder

COPY . .

RUN bazel build //src/main:myapp
```

Spring pet clinic

What is Spring PetClinic?

Spring PetClinic is a **sample application** created by the Spring team to demonstrate how to build a real-world web application using the **Spring Framework**.

It simulates a small veterinary clinic management system where you can manage:

- **Owners** ➔ people who own pets.
- **Pets** ➔ animals with details like type and birth date.
- **Vets** ➔ veterinarians who work in the clinic.
- **Visits** ➔ medical appointments for pets.

Main Technologies Used

- **Spring Boot** ➔ for running the application as a standalone service.
- **Spring MVC** ➔ for handling web requests (controllers, routing, etc.).
- **Spring Data JPA** ➔ for database access and repository management.
- **Thymeleaf** ➔ as the template engine for rendering web pages.
- **H2 Database** (by default) ➔ an in-memory database, but you can switch to MySQL/PostgreSQL.

Project Structure

- model/ ➔ domain classes (Owner, Pet, Vet, Visit).
- repository/ ➔ interfaces that handle database operations.

- service/ ➔ contains business logic (often thin, thanks to Spring Data JPA).
- controller/ ➔ Spring MVC controllers that handle web requests.
- resources/templates/ ➔ Thymeleaf HTML pages.
- application.properties ➔ configuration file.

Key Features

- Search for owners and view their pets.
- Add new owners and pets.
- View and manage veterinarians.
- Record visits for pets.

How to Run It

```
git clone https://github.com/spring-projects/spring-petclinic.git

cd spring-petclinic

./mvnw package

java -jar target/*.jar
```

You can write localhost:8080 to see the application

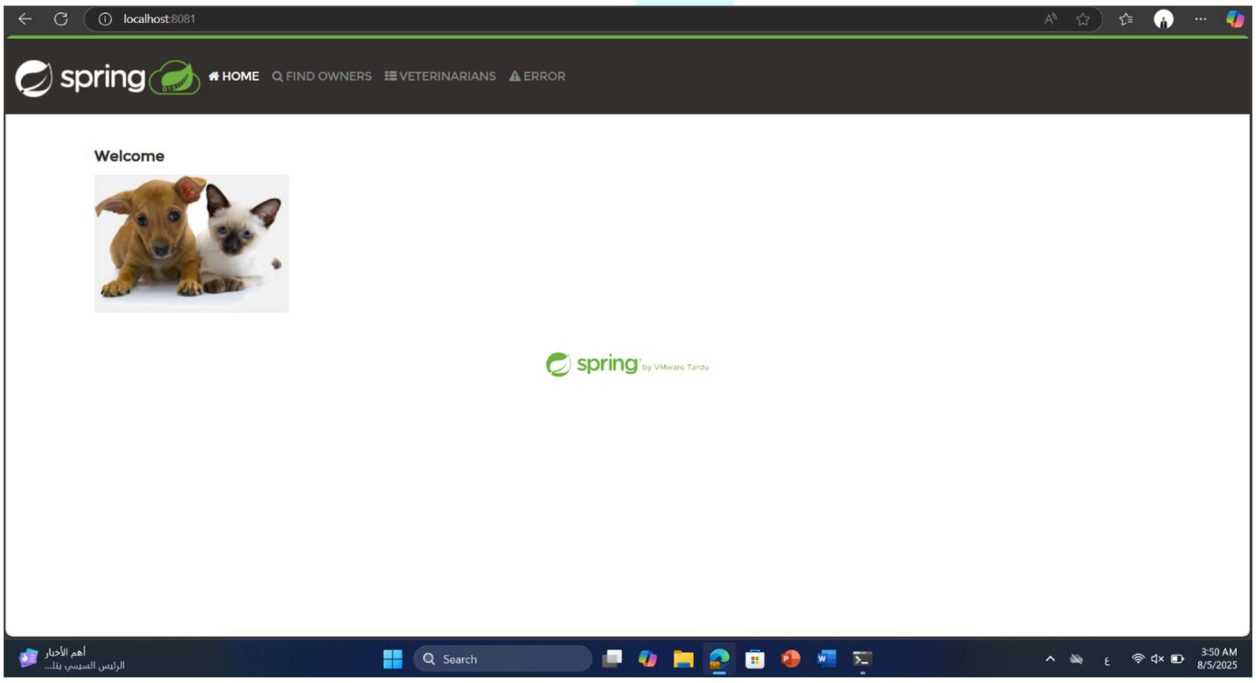
Method two pull an existing image

docker pull {image name and its tag} get it from dockerhub

docker run -d -P {image} ➔ will run it on a random port

docker run -d -p 8081:8080 {image} ➔will run it on 8081 port

You will see:



Now we need to make a dockerfile for it

1. **Goal ➔** Production-ready, small, secure, and efficient.
2. **Base Image ➔** Use a JDK only for building, then a JRE (or distroless) for running.
3. **Layers ➔** Multi-stage build (dependencies → build → runtime).
4. **Security ➔** Run as non-root, avoid unnecessary files.

5. **Entrypoint** ➔ Run the jar with java -jar.

And my final Dockerfile

```
# Build stage
FROM maven:3.9.9-eclipse-temurin-17 AS build
WORKDIR /app
#to improve caching
COPY pom.xml .
# Cache dependencies
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests


# Run stage with jre not jdk
FROM eclipse-temurin:17-jre-alpine
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar

EXPOSE 9966

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Breakdown of Best Practices Used

- **Multi-stage build** ➔ Maven image for build, slim JRE for runtime ➔ smaller, cleaner final image.
- **Dependency caching** ➔ COPY pom.xml + mvn dependency:go-offline before copying src ➔ faster rebuilds.
- **Security** ➔ non-root user.
- **ENTRYPOINT** ➔ Entrypoint is fixed (java -jar app.jar) .

Note

- I modified the default port of the application from 8080 to 9966 by editing the application.yml file and add server.port=9966
- Also to limit the size of the image I used dockerignore inside it I put

.git
.gitignore
target
*.md
.idea
*.iml
src/test

Building & Runnig

Docker build -t petclinic:v1 .
Docker run-d -p 9999:9966 petclinic:v1

Conclusion

Spring PetClinic is a simple yet practical sample application that demonstrates how to build a real-world web app with Spring Boot, Spring MVC, and Spring Data JPA. It's widely used for learning, training, and experimenting with modern software development practices such as CI/CD, containerization, and cloud deployment.