how to crearte login page java Web project using docker volumes and how to manage docker volumes with code and command and example on ubuntu 22.04 ?

ChatGPT

Creating a login page in a Java web project using Docker involves several steps. You'll need to:

1. Set up a Java web application (using Spring Boot for simplicity).
2. Create a Dockerfile to containerize the application.
3. Use Docker volumes to manage data persistence.
4. Write commands to manage Docker volumes.
5. Run the project on Ubuntu 22.04.

**Step 1: Set Up the Java Web Application**

We'll use Spring Boot to create a simple web application with a login page.

**1.1. Create a Spring Boot Application**

Use Spring Initializr to bootstrap your project or set it up manually.

**pom.xml**

xml

Copy code

<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>loginapp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**1.2. Create the Login Page and Security Configuration**

Create a simple login page using Thymeleaf and configure Spring Security.

**src/main/resources/templates/login.html**

html

Copy code

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Login</title>

</head>

<body>

<h1>Login</h1>

<form th:action="@{/login}" method="post">

<div>

<label>Username: <input type="text" name="username"/></label>

</div>

<div>

<label>Password: <input type="password" name="password"/></label>

</div>

<div>

<button type="submit">Sign in</button>

</div>

</form>

</body>

</html>

**src/main/java/com/example/loginapp/SecurityConfig.java**

java

Copy code

package com.example.loginapp;

import org.springframework.context.annotation.Bean;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.web.SecurityFilterChain;

@EnableWebSecurity

public class SecurityConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.authorizeRequests(authorizeRequests ->

authorizeRequests.anyRequest().authenticated()

)

.formLogin(formLogin ->

formLogin.loginPage("/login").permitAll()

);

return http.build();

}

}

**src/main/java/com/example/loginapp/LoginAppApplication.java**

java

Copy code

package com.example.loginapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LoginAppApplication {

public static void main(String[] args) {

SpringApplication.run(LoginAppApplication.class, args);

}

}

**Step 2: Create a Dockerfile**

Create a Dockerfile in the root of your project to containerize the Spring Boot application.

**Dockerfile**

dockerfile

Copy code

# Use an official OpenJDK runtime as a parent image

FROM openjdk:17-jdk-slim

# Set the working directory in the container

WORKDIR /app

# Copy the project's JAR file into the container

COPY target/loginapp-0.0.1-SNAPSHOT.jar app.jar

# Make port 8080 available to the world outside this container

EXPOSE 8080

# Run the JAR file

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

**Step 3: Use Docker Volumes**

Docker volumes are used to persist data generated by and used by Docker containers.

**3.1. Build the Docker Image**

sh

Copy code

docker build -t loginapp:latest .

**3.2. Create and Run the Docker Container with a Volume**

sh

Copy code

docker volume create loginapp\_data

docker run -d -p 8080:8080 -v loginapp\_data:/app/data --name loginapp\_container loginapp:latest

**Step 4: Manage Docker Volumes**

**4.1. List Volumes**

sh

Copy code

docker volume ls

**4.2. Inspect a Volume**

sh

Copy code

docker volume inspect loginapp\_data

**4.3. Remove a Volume**

To remove a volume, make sure no containers are using it.

sh

Copy code

docker volume rm loginapp\_data

**Step 5: Run the Project on Ubuntu 22.04**

1. Install Docker on Ubuntu 22.04:

sh

Copy code

sudo apt update

sudo apt install -y docker.io

sudo systemctl start docker

sudo systemctl enable docker

1. Build and run your Dockerized application:

sh

Copy code

cd /path/to/your/project

docker build -t loginapp:latest .

docker volume create loginapp\_data

docker run -d -p 8080:8080 -v loginapp\_data:/app/data --name loginapp\_container loginapp:latest

Your application should now be running and accessible on http://localhost:8080. You can manage Docker volumes as described in step 4 to ensure data persistence and proper volume handling.