Write a Dockerfile to containerize a simple web application

Table of Contents

Description

This section provides instructions to create a Dockerfile for a simple **Java-based TodoApp**. It also covers the steps to build and run the Docker image. We'll explain how to run a **Java-based TodoApp** container from Docker image and manage it using common Docker commands. This includes how to start, stop, remove, and inspect running containers, as well as monitor logs.

Problem Statement

- You want to containerize a simple **Java-based TodoApp**. Using Docker will help ensure that the app runs consistently in any environment, eliminating the "works on my machine" problem. This document explains how to create the required Dockerfile, build the image, and run the container.
- Once you've built a Docker image for the Java-based TodoApp (or any other image), you need to manage the container lifecycle: starting, stopping, inspecting, and removing containers efficiently.

Prerequisites

Software Requirement

Before beginning the installation, ensure the following software is available and compatible:

1. **Docker Desktop**

2. **Windows 10 (64-bit) or higher**: Required for WSL 2 integration.
3. **Windows Subsystem for Linux 2 (WSL 2)**: This enables Linux container support on Windows systems.
4. **Python (Optional)**: If working with Python-based applications within Docker, install Python.

Docker Desktop can run either Windows containers or Linux containers; Linux containers are the default and require WSL 2 for optimal performance on Windows systems.

Hardware Requirement

Ensure your system meets the following hardware specifications to run Docker Desktop efficiently:

- **CPU**: 64-bit processor with virtualization support (Intel VT-x/AMD-V).
- **RAM**: 4 GB RAM (8 GB or higher recommended).
- **Storage**: 10 GB free disk space for Docker images and containers.
- **Virtualization**: Ensure that hardware virtualization is enabled in BIOS. This is critical for Docker to function.

Implementation Steps

Step 1: Downloading the TodoApp with H2 Database

To download the **TodoApp** with an H2 database, please follow the steps below:

1. **Click the Download Link**: Download the project from the following link: Download TodoApp with H2 Database
2. **Extract the Zip File**:
    - After downloading, go to your downloads folder.
    - Find the file named `TodoApp_H2-main.zip`.
    - Right-click on the zip file and choose **Extract All** or use any extraction tool to unzip the folder.
3. **Navigate to the Extracted Folder**:
    - Open the folder `TodoApp_H2-main` which contains the project files.
4. **Open the Project in Your IDE**:
    - Open your preferred IDE (e.g., IntelliJ IDEA, Eclipse, or VSCode).
    - Choose **File > Open** and select the extracted `TodoApp_H2-main` folder.

Now you have the **TodoApp with H2 database** downloaded and ready for setup!

Step 2: Writing the Docker File

- We are working with a simple Todo application, which is a Java-based project built using Maven. The application uses H2 as the database, along with Hibernate for ORM (Object-Relational Mapping).We will write a Dockerfile for the project and proceed to build a Docker image, which will package the entire application and its environment for deployment in any containerized setup.

```
FROM openjdk:11.0.15-jre
ADD target/*.jar app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

- **FROM openjdk:11.0.15-jre:** This sets the base image for the container, using the Java Runtime Environment (JRE) version 11.0.15. It provides the necessary environment to run Java applications.
- **ADD target/*.jar app.jar:** This copies the JAR file generated by Maven from the **target/** directory on your local machine into the Docker image, renaming it to **app.jar**.
- **ENTRYPOINT ["java", "-jar", "app.jar"]:** This defines the command that runs when the container starts. It tells the container to execute the **app.jar** file using the Java runtime.

Step 3: Building The Docker Image

- Docker builds images by reading the instructions from a Dockerfile. A Dockerfile is a text file containing instructions for building your source code.
- Open cmd prompt or cd to the source folder path.
- Docker desktop should be running in the background.

```
docker build -t todoapp .
```

The command **docker build -t todoapp .** is used to build a Docker image from a Dockerfile.

- **docker build:** This is the Docker command to build an image.
- **-t todoapp:** The **-t** option tags the image with a name (**todoapp** in this case). It helps identify the image when running or managing containers later.
- **. (dot):** This specifies the build context, which means the current directory. Docker will look for the **Dockerfile** and other necessary files in this directory to create the image.

 image

Step-4: Run a Container from an Existing Image

You can run a Docker container from the existing **Java-based TodoApp** image (**todoapp**) using the **docker run** command.

1. **Run the Container**:

   ```
   docker run -d -p 8081:8081 --name my_todoapp todoapp
   ```

   - **-d**: Runs the container in **detached mode** (in the background).
   - **-p 8081:8081**: Maps port 8081 of the host to port 8081 of the container.
   - **--name my_todoapp**: Names the container **my_todoapp** for easier management.

- **todoapp**: The name and tag of the image you built.

2. **Verify the Container is Running**:

   You can list all running containers with the following command:

   ```
   docker ps
   ```

   You should see output similar to:

3. **Access the Application**:

   Open a browser and navigate to **http://localhost:8081/swagger-ui/index.html** to see your TodoApp running inside the container.

Step-5: Managing Running Containers

Once your container is up and running, Docker provides several commands to manage the containers.

1. **List All Running Containers**:

   ```
   docker ps
   ```

2. **List All Containers (Including Stopped Containers)**:

   ```
   docker ps -a
   ```

   This will list all containers, including those that have stopped. 

Step-6: Stopping, Restarting, and Removing Containers

1. **Stop a Running Container**:

   Use the **docker stop** command to stop a running container:

   ```
   docker stop my_todoapp
   ```

   This will gracefully stop the **my_todoapp** container.

2. **Restart a Stopped Container**:

   Use the **docker start** command to restart the container:

```
docker start my_todoapp
```

   This starts the container with the previous configuration (i.e., running the same image).

3. **Remove a Stopped Container**:

   To remove a stopped container, use the **docker rm** command:

```
docker rm my_todoapp
```

   Note: A running container cannot be removed. You need to stop it first using **docker stop**.

4. **Force Remove a Running Container**:

   If you need to remove a running container, use the **-f** flag:

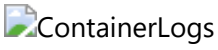
```
docker rm -f my_todoapp
```

Step-7: Accessing Logs and Container Status

1. **View Container Logs**:

   To see the logs from a running container, use the **docker logs** command:

```
docker logs my_todoapp
```



   This shows the standard output logs of the running application.

2. **Follow Logs in Real-Time**:

   If you want to stream the logs in real-time, add the **-f** flag:

```
docker logs -f my_todoapp
```

This will continue displaying logs as they are written.

3. **Inspect a Container's Details**:

   To get detailed information about the container, use the **docker inspect** command:

   ```
   docker inspect my_todoapp
   ```

   This will return JSON-formatted data that contains all the configuration and status details of the container.

4. **View the Running Container's Status**:

   Use **docker stats** to monitor resource usage like CPU, memory, network, etc.

   ```
   docker stats my_todoapp
   ```

   This command continuously updates with live statistics of the container's resource consumption.

References

For more detailed information on Dockerfile best practices and Docker command usage, refer to:

- Docker Official Documentation: click here
- Docker Command Reference: https://docs.docker.com/engine/reference/commandline/docker/