Write shell scripts to automate Docker container management tasks

Table of Contents

Description

This guide explains how to use **shell scripts** to automate Docker container management tasks for the **TodoApp**. Managing containers manually can be repetitive and error-prone, especially when you need to frequently start, stop, or clean up containers. Automating these tasks through shell scripts can save time and reduce errors.

Problem Statement

Manually managing Docker containers can be tedious, especially for tasks such as:

- Building Docker images.
- Starting and stopping containers.
- Cleaning up unused containers, images, and volumes.

This document provides a set of scripts to automate these common Docker tasks, making it easier to manage the **TodoApp** containerized environment.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-04) is required before proceeding with Lab Guide-05.

Software Required

- **Docker Desktop**: Installed on your Windows machine.
- **Bash (for Windows)**: Git Bash or WSL (Windows Subsystem for Linux) to run shell scripts.
- **TodoAPP_MYSQl**: To download the source folder **click here**

Hardware Requirement

- Minimum of 4 GB RAM
- At least 2 cores in the processor
- 5 GB of free storage space for Docker images and containers

Implementation Steps

Step-1: Create a Custom Docker Network and MySQL Container

1. **Create the Docker Network**:

   First, we'll create a custom network named **todoapp_network**.

   ```
   docker network create todoapp_network
   ```

2. **Run the MySQL Container**: Use the following command to create a MySQL container connected to the custom network:

   ```
   docker run -d -p3306:3306 --network=todoapp_network -e
   MYSQL_ROOT_PASSWORD=P@ssw0rd -e MYSQL_DATABASE=tododb --name=mysqldb mysql
   ```

Step-2: Automate Building and Running Containers

Create a script that builds the Docker image for the **TodoApp** starts the containers and connects it to the custom network `todoapp_network`.

> Note: Ensure you navigate to the todoapp directory using cd todoapp before executing the program, as this sets the correct working directory for running scripts or commands.

cdTodoapp

1.1 `run_todoapp.sh` - Script to Build and Run the TodoApp

Create a shell script named `run_todoapp.sh` with the following content:

```
#!/bin/bash

# Set variables
IMAGE_NAME="todoapp_image"
CONTAINER_NAME="todoapp_container"

# Step 1: Build the Docker image
echo "Building the Docker image..."
docker build -t $IMAGE_NAME .
```

```
# Step 2: Run the Docker container
echo "Starting the Docker container..."
# docker run -d --name $CONTAINER_NAME -p 8080:8080 $IMAGE_NAME
docker run -d -p8081:8081 --name todoapp_container --network=todoapp_network -e
MYSQL_HOST=mysqldb todoapp_image

# Step 3: Check if the container is running
if [ $(docker inspect -f '{{.State.Running}}' $CONTAINER_NAME) == "true" ]; then
  echo "TodoApp is running on http://localhost:8081"
else
  echo "Failed to start the TodoApp container."
fi
```

- **Build the Docker image**: This uses `docker build` to create an image from the `Dockerfile`.
- **Run the container**: This runs the container in detached mode (`-d`) and maps port `8081` on the host to port `8081` in the container.
- **Check container status**: The script verifies if the container is running and provides a link to the application.

Usage:

1. Make the script executable:

```
chmod +x run_todoapp.sh
```

2. Run the script:

```
./run_todoapp.sh
```

---

chmodRun

runSH

runSH1

The command `chmod +x run_todoapp.sh` is used to make the file `run_todoapp.sh` executable. Here's a breakdown of its components:

1. **`chmod`**: This is the command to change file permissions in Unix-like operating systems (e.g., Linux, macOS).

2. **`+x`**: This flag is used to add execution permission to the file for the user. It allows the user to run the file as a program or script.

3. **`run_todoapp.sh`**: This is the name of the file to which you are applying the execution permission. In this case, it's a script file, typically written in shell script (because of the `.sh` extension).

Function of `chmod +x run_todoapp.sh`:

By executing this command, you make the script `run_todoapp.sh` executable, which means you can run it directly from the command line like this:

```
./run_todoapp.sh
```

Without this permission, you would not be able to run the script directly, and you might get a "Permission denied" error when trying to execute it.

Step-3: Automate Stopping and Removing Containers

Create a script that stops and removes running containers, especially for **TodoApp**.

2.1 `stop_todoapp.sh` - Script to Stop and Remove Containers

Create a shell script named `stop_todoapp.sh` with the following content:

```bash
#!/bin/bash

# Set variables
CONTAINER_NAME="todoapp_container"

# Step 1: Stop the running container
echo "Stopping the container..."
docker stop $CONTAINER_NAME

# Step 2: Remove the container
echo "Removing the container..."
docker rm $CONTAINER_NAME

# Step 3: Confirm the container is removed
if [ $(docker ps -a | grep $CONTAINER_NAME) ]; then
  echo "Failed to remove the container."
else
  echo "Container removed successfully."
fi
```

- **Stop the container**: This uses `docker stop` to stop the running container.
- **Remove the container**: This uses `docker rm` to delete the stopped container.
- **Verify removal**: The script checks if the container still exists after the removal process.

Usage:

1. Make the script executable:

```
chmod +x stop_todoapp.sh
```

2. Run the script:

```
./stop_todoapp.sh
```

---

stopSH

Step-4: Automate Cleaning Up Unused Docker Resources

Create a script that cleans up unused Docker containers, images, and volumes to free up space.

3.1 `cleanup_docker.sh` - Script to Clean Up Docker Resources

Create a shell script named `cleanup_docker.sh` with the following content:

```bash
#!/bin/bash

# Step 1: Remove stopped containers
echo "Removing stopped containers..."
docker container prune -f

# Step 2: Remove unused images
echo "Removing unused images..."
docker image prune -f

# Step 3: Remove unused volumes
echo "Removing unused volumes..."
docker volume prune -f

# Step 4: Remove unused networks
echo "Removing unused networks..."
docker network prune -f

echo "Docker cleanup completed!"
```

- **Remove stopped containers**: The `docker container prune` command removes all stopped containers.
- **Remove unused images**: The `docker image prune` command removes all dangling (unused) images.
- **Remove unused volumes**: The `docker volume prune` command removes all unused volumes.
- **Remove unused networks**: The `docker network prune` command removes all unused networks.

Usage:

1. Make the script executable:

```
chmod +x cleanup_docker.sh
```

2. Run the script:

```
./cleanup_docker.sh
```

---

cleanSH

References

- Docker CLI Commands: https://docs.docker.com/engine/reference/commandline/docker/
- Automating Docker Workflows: https://docs.docker.com/engine/admin/