COMPOSING ENVIRONMENTS

Overview

The goal of this lab is to introduce you to Docker Compose environment. You will learn how to construct YAML files and use docker-compose.yaml files to create and deploy multi-container applications.

Procedure

Installing Docker Compose CLI

We will first configure our Docker environment for Docker Compose.

- Download the binary to /usr/local/bin with the following command in your Terminal:

  sudo curl -SL https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose

- Make the downloaded binary executable with the following command:

  sudo chmod +x /usr/local/bin/docker-compose

- Test the CLI and installation with the following command in the Terminal on all operating systems:

  docker-compose version

If it is installed correctly, you will see the versions of the CLI and its dependencies. For instance, you might see the output, the docker-compose CLI has version 2.16.0.

Getting Started with Docker Compose

- Create a folder named server-with-compose and navigate into it using the cd command:

  mkdir server-with-compose && cd server-with-compose

- Create a folder with the name init and navigate into it using the cd command:

  mkdir init && cd init

- Create a Bash script file with the following content and save it as prepare.sh:

```
#!/usr/bin/env sh
rm /data/index.html
echo "<h1>Welcome to Humber Institute of Technology and Advanced Learning</h1>" >>
/data/index.html
echo "<img src='https://ccgc5500.s3.amazonaws.com/Humber_BCTI_new.jpg' />" >>
/data/index.html
```

This script generates a sample HTML page with the echo commands.

- Create a Dockerfile with the name Dockerfile and the following content:

```
FROM busybox
ADD prepare.sh /usr/bin/prepare.sh
RUN chmod +x /usr/bin/prepare.sh
ENTRYPOINT ["sh", "/usr/bin/prepare.sh"]
```

This Dockerfile is based on busybox, which is a tiny operating system for space-efficient containers, and it adds the prepare.sh script into the filesystem. In addition, it makes the file executable and set it as the ENTRYPOINT command. The ENTRYPOINT command, in our case, the prepare.sh script is initialized with the start of the Docker container.

- Change the directory to the parent folder with the cd .. command and create a docker-compose.yaml file with the following content:

```
version: "3"
services:
  init:
    build:
      context: ./init
    volumes:
      - static:/data
  server:
    image: nginx
    volumes:
      - static:/usr/share/nginx/html
    ports:
      - "8080:80"
volumes:
  static:
```

This docker-compose file creates one volume named static, and two services with the names init and server. The volume is mounted to both containers. In addition, the server has published port 8080, connecting to container port 80.

- Start the application with the following command in detach mode to continue using the Terminal:

```
docker-compose up --detach
```

The preceding command creates and starts the containers in detached mode. It starts by creating the server-with-compose_default network and the server-with-compose_static volume. Then, it builds the init container using the Dockerfile, downloads the nginx Docker image for the server, and starts the containers. Finally, it prints the names of the containers and makes them run in the background.

- Check the status of the application with docker-compose ps command.

```
docker-compose ps
```

This output lists two containers. The init container exited successfully with code 0, while the server container is Up and its port is available. This is the expected output since the init container is designed to prepare the index.html file and complete its operations, whereas the server container should always be up and running.

- Open http://Docker_Host_IP_Address:8080 in the browser to verify your application is running.

In this exercise, a multi-container application was created and configured by docker-compose. Networking and volume options were stored in the docker-compose.yaml file. In addition, CLI commands were demonstrated in action for creating applications and checking the status.

Creating Applications with Docker Compose

Now that we know how to work with Docker Compose. Let us compose a more practical application using Docker Compose.

- Create a folder named wordpress and navigate into it using the cd command:

  mkdir wordpress && cd wordpress

- Create a docker-compose.yaml file with the following content:
  (Note: You will have to format the YAML file with appropriate indentation and structures)

```
version: "3.9"

services:
db:
image: mysql:5.7
volumes:
- db_data:/var/lib/mysql
restart: always
environment:
MYSQL_ROOT_PASSWORD: somewordpress
MYSQL_DATABASE: wordpress
MYSQL_USER: wordpress
MYSQL_PASSWORD: wordpress

wordpress:
depends_on:
- db
image: wordpress:latest
volumes:
- wordpress_data:/var/www/html
ports:
- "8000:80"
restart: always
environment:
WORDPRESS_DB_HOST: db:3306
```

```
WORDPRESS_DB_USER: wordpress
WORDPRESS_DB_PASSWORD: wordpress
WORDPRESS_DB_NAME: wordpress
volumes:
db_data: {}
wordpress_data: {}
```

- Start the application with the docker-compose up command:

  docker-compose up --detach

- Check for the running containers with the ps command.

  docker-compose ps

- Open http://Docker_Host_IP_Address:8000 in your browser to check the WordPress setup screen.
- Stop and remove all the resources with the following command if you do not need the application up and running:

  docker-compose down

In this activity, you have created a deployment for a real-life application using Docker Compose. The application consists of a database container and a WordPress container. Both container services are configured using environment variables, connected via Docker Compose networking and volumes.

Clean up your Docker environment.