Course code: DV1566

Introduction to Cloud Computing

Practice with containers

Group - 9

Sagar Premchand Yatam: 000306T058

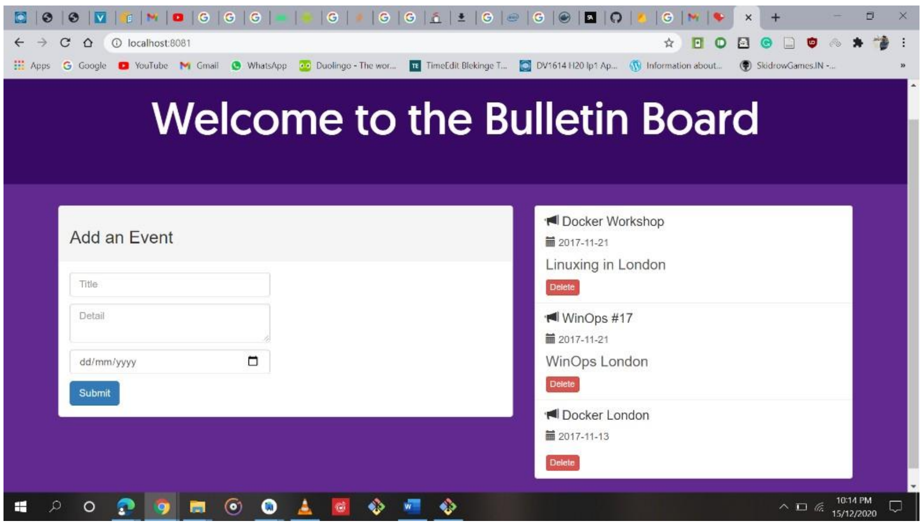
Assessment

We have selected an online application called Chipfelton/bulletinboard from the Docker hub. We have made this report on the tasks performed, and the used commands and problems faces during our implementation.

Step-1

The application which we have selected is online Bulletin Board. Basically, bulletin board is a client-server application which is used for exchanging messages with other clients through public message boards. Bulletin board is similar to notice boards or pin boards used in schools and colleges.

The main component of Bulletin board is web application server where the web server fulfils the client requests, and the application server is used to run applications. The main function of this application is to display the events (title of the event, details of the event and date) which are specified by the user.



Webpage of our Bulletin board application

Step-2

We selected a simple client/server application, which is pulled from the Docker hub using the command,

```
$docker pull chipfelton/bulletinboard:1.0
```

To change the pulled application to our own repository the command used is:

```
$docker tag chipfelton/bulletinboard:1.0 prudhvi: 1.0
```

Dockerfile: describes how to assemble a private filesystem for a container and some metadata containing about how to run a container based on the image.

Dockerfile :

```
# Use the official image as a parent image.
FROM node:current-slim

# Setting the working directory.
 WORKDIR /usr/src/app

# Copy file from host to current location.
COPY package.json. .

# Run the command inside your file system.
 RUN npm install

# local host number
EXPOSE 8080

#run command within container
CMD [ "npm", "start"]
```

The run command is used to run the image to activate the container. Here we used port '8080' as network port for the container to listen during the run time. And port '8081' which was specified below in the command is a localhost port, through which we connect to the container through web browser.

```
$ docker run -it -d -p 8081:8080 prudhvi:1.0
```

The command is used to inspect the image/container.

```
$ docker inspect prudhvi:1.0
```

To execute the application:

```
$ docker exec -it id of the image container bash
```

Volumes: Volumes are the used to manage data generated and used by Docker containers. Volumes can be shared among multiple containers. The command we used is,

$ docker run -it -d -p 8081:8080 -v volume1:/usr/src/app prudhvi:1.0

Here, we used Volume 'volume1' to store the data of the container. This volume remains, even we terminate the container.

Bind mount: In general, bind mount means memory, a file or directory on host machine in a container. They have limited functionality compared to volumes. The command we used is,

$ docker run -it -d -p 8081:8080 -v mount:/usr/src/app prudhvi:1.0

Here, we will assign the directory in the local drive to store the data of the container into. The bind mount remains safe even we terminate the container.

Networking method: Bridge is the default network driver. Which is used as the software bridge which allows the containers connected in the same network to communicate.

Command: $ *docker network ls* (this command is used to list all the networks)

The command mentioned below is used to find the IP address of the containers by inspecting the bridge network.

Command: $ *docker network inspect bridge*

Step-3

Building of docker compose fie:

Docker-compose.yml: We built a yaml file called docker-compose.yml, which included the container deployment by considering two replicas.

```
services:

    node:
        image: prudhvi:1.0
            ports:
             - "8081:8080"
           deploy:
              restart_policy:
                condition: onfailure
               resources:
               limits:
                 cpus: '0.01'
                 memory: 25M
  networks:
      default:
         driver: bridge
```

Command we use to run and terminate services using docker-compose file are,

docker-compose up -d

docker-compose down

To scale the services, we can use’–scale’ flag:

$ *docker-compose up -d –scale web=4*

Docker Swarm: Swarm is an orchestration feature which is present in the docker engine. Swarm consists of multiple docker hosts, docker hosts can be managers, worker, or perform both roles. To create a swarm manager node we used the below command,

```
$ docker swarm init –advertise-addr <ip address>
```

We used the below command to create the workers node.

```
$ docker swarm join –token <token id>
```

We used below command to create and scale the services of application.

```
$ docker service create –replicas 2 -p 8081:8080

chipfelton/bulletinboard:1.0
```

Motivation: Initially it was difficult while setting up docker and understanding the terminology. As we are new to this, we did not understand the connection between cloud and docker after a lot of research we came to a conclusion. Firstly, we did not understand which application to choose after looking at a lot of different applications we decided to take the example file you mentioned. The implementation and running of the docker file was pretty simple. We face difficulty in creating volumes and orchestration it ate most of our time but we have learn a lot of new things and finally accomplished.