Project 4 - Containerizing an application

06 June 2024 21:48

INTRODUCTION
Containerizing an application involves packaging it along with its dependencies and configurations into a container, which can run consistently across different computing environments.

- Containers ensure your application runs the same way on any computer, like using your game console anywhere.
- They keep your application in its own separate space, making it safer and more reliable, like having your own room.
- Containers can be easily scaled to handle more users, similar to adding more lanes on a highway during rush hour.
- They allow you to update parts of your application without affecting the whole system, like changing the battery in a remote.
- Containers help developers write and test code faster, like using a recipe to quickly make the same cake every time.

PROJECT
In this project I containerize a Java application hosted within a Tomcat server. The dependencies are mysql, rabbitmq, memcached, and nginx. These will be containerized.

Containerization Tools employed

- Docker (Container runtime)
    - Docker-compose
- Java stack
    - Application code
    - Vprofile app
        - Nginx, Tomcat, MySQL, Memcached, RabbitMQ
- Vagrant vm image from vagrant cloud (ec2 instance could very much be used)

Note: prior to this project, I had already installed the following:

- VirtualBox
- Vagrant
- Git was already setup
- VScode

<u>Steps to setup our stack</u>

- Setup stack services
- Find right Base image from dockerhub
- Write Dockerfile to customize images
- Write docker-compose.yml to run multi-containers
- Test and send images to Dockerhub

The following images will be taken from dockerhub without customization - rabbitmq and memcached. Tomcat and mysql images were customized to suit the project's use-case. Tomcat image is customized because our java application will live here; no docker images has our application. The mysql image was indeed customized to create the database with the right schema for the project. The vanilla memcached image from dockerhub meets our requirements.

Step 1: download and initialize ubuntu image from vagrant cloud

- I searched for vagrant cloud on google and clicked on the link to the website
- Searched for ubuntu 22 on the vagrant file website, clicked on one of the options and copied the box name. the option I went with is bento/ubuntu-22.04.
- Other versions (20+) could be used as well.

bento / ubuntu-22.04 Vagrant box

How to use this box with Vagrant:

Vagrantfile New

```
Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-22.04"
end
```

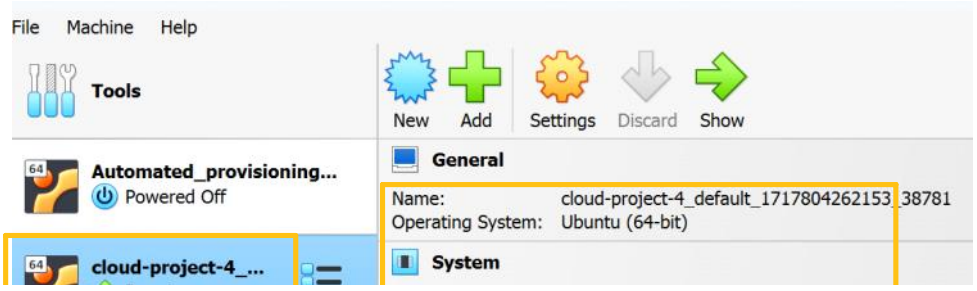
- I initialized the ubuntu vagrantfile and edited it

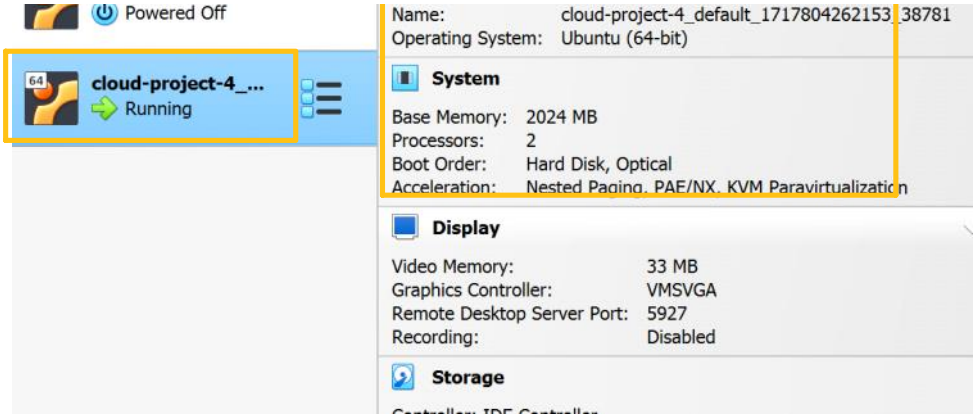
```
$ vagrant init bento/ubuntu-22.04
==> vagrant: A new version of Vagrant is available: 2.4.1 (installed version:
3.4)!
==> vagrant: To upgrade visit: https://www.vagrantup.com/downloads.html

A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

- Next was to edit the Vagrantfile
  - Uncommented the network settings (public and private networks) because the containers will be accessed from the internet.
  - Increased the RAM size of the vm

```
# Create a private network, which allows host-only access to the machine
# using a specific IP.
  config.vm.network "private_network", ip: "192.168.56.38"

# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
  config.vm.network "public_network"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
   vb.memory = "2024"
end
```

- Save and exit from the Vagrantfile.
- Bring up the vm with the `vagrant up` command
- Open the VirtualBox to see the running VM





Step 2: install docker engine on VM.

- Ssh into the vm using `vagrant ssh` command. I switched to the root user using `sudo -i`
- I visited the docker installation documentation for the steps to install docker engine on ubuntu
- I followed the steps to install docker engine on the ubuntu vm
    - It is advised to visit the documentation to use the latest commands to install docker as these commands change regularly.

Installation steps (from docker docs)

Set up Docker's `apt` repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc
  $(. /etc/os-release && echo "$VERSION_CODENAME") stabl
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/nu
sudo apt-get update
```

2. Install the Docker packages.

Latest Specific version

To install the latest version, run:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin doc
```

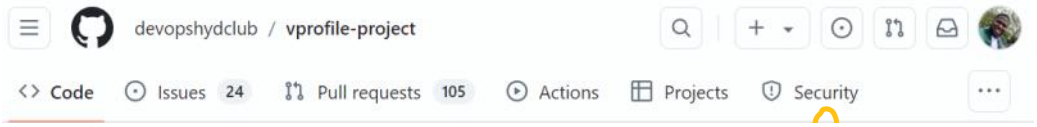
- After docker installation, because I intended to run docker commands using the vagrant user, I had to add the vagrant user to the docker group using this command
-

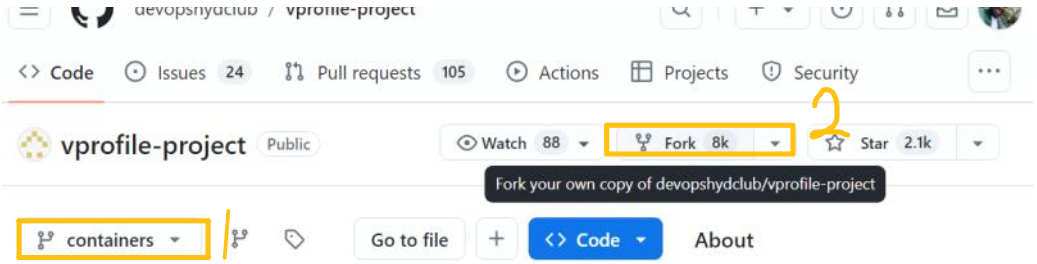
```
root@vagrant:~# usermod -aG docker vagrant
root@vagrant:~# id vagrant
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant),4(adm),24(cdrom),27(sud
o),30(dip),46(plugdev),110(lxd),997(docker)
root@vagrant:~#
```

- Logout and relogin

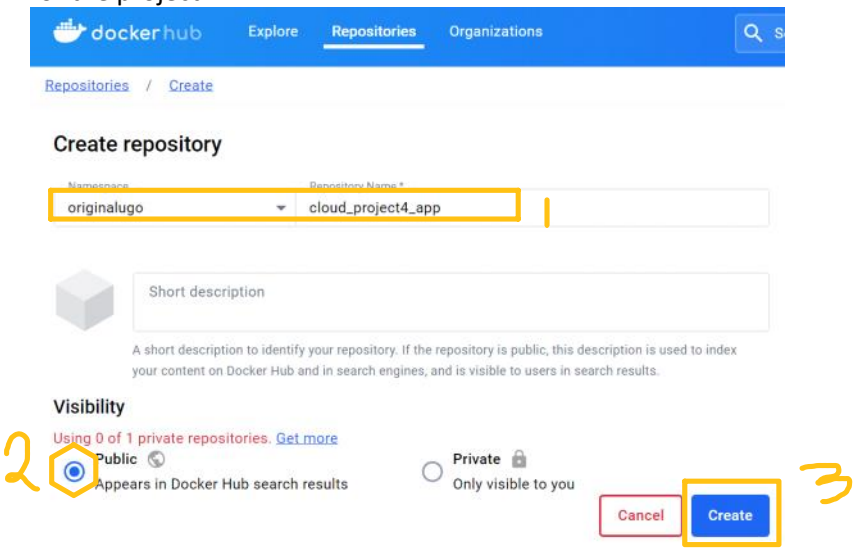
Step 3: obtain source code of application

- The application source code is in the containers branch of this github repo: `https:github.com/devopshydclub/vprofile-project`. I had to fork it to my own github account to be able to make changes.





- I cloned the repository on my pc in the same directory where I have the Vagrantfile saved. This is to make it easy to access the source code from the VM.
- I switched to the containers branch.
- Inspecting the Dockerfiles in the Dockerfiles directory show that they are empty. I will write the contents of these files to create the docker images.
- I signed in to my dockerhub account and created repositories for the container images I will use for the project



Step 4: Write Dockerfiles to create the images for the project

Tomcat image

- My application is a java application and as such I need the following (in reality, the image version is given by the developers or gotten by trial and error, in this instance)
  - Openjdk-11, tomcat, maven
- The information needed to build the Dockerfile is found in the docker hub website.
- I built a multi-stage docker file to keep the size of the container image small. Without using this strategy, the docker image size will be large because we will have lots of dependencies downloaded into the image. Using a multi-stage docker file, downloads the dependencies separately from the docker image to be used in the container

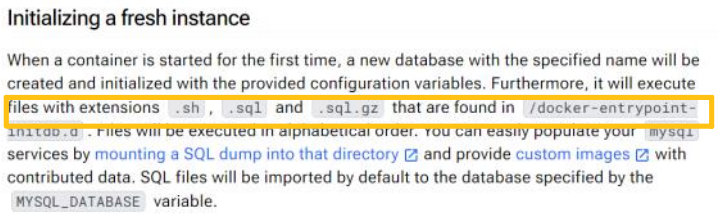
```
Docker-files > app > Dockerfile > ...
1   FROM openjdk:11 AS BUILD_IMAGE
2   LABEL "Project"="Vprofile"
3   LABEL "Author"="Samson"
4
5   RUN apt update && apt install maven -y
6   RUN git clone https://github.com/hashtagsam/vprofile-project.git
7   RUN cd vprofile-project && git checkout docker && mvn install
8
9   FROM tomcat:9-jre11
10  RUN rm -rf /usr/local/tomcat/webapps/*
11  COPY --from=BUILD_IMAGE vprofile-project/target/vprofile-v2.war /usr/local/tomcat/webapps/ROOT.war
12
13  EXPOSE 8080
14  CMD ["catalina.sh", "run"]
```

The steps in the tomcat Dockerfile is explained below:

- Got the openjdk:11 image from dockerhub
- Gave few labels
- Used the ubuntu package manager, apt to update the repository and installed maven
- Cloned the repository where the source code lives.
- Entered the project directory (vprofile-project), chose the right branch and built the artifact using mvn install. The artifact is saved in 'vprofile-project/target' directory
- Downloaded the tomcat image, removed the template server page
- Copied the artifact from the target directory to the default location for tomcat applications (saved with ROOT.war)
- Exposed the port 8080 and ran the catalina.sh command. This step will be carried out anyways without being explicit.

Mysql image

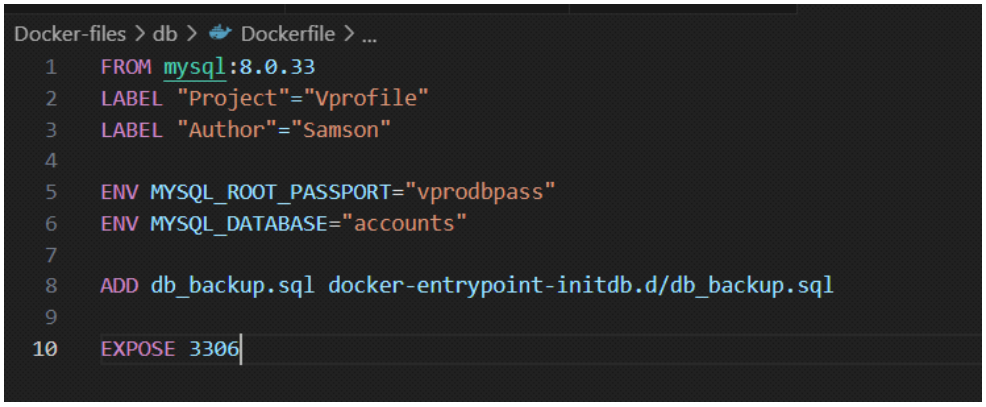
- Again, the information used to create this Dockerfile originates from the image documentation on docker hub
    - I am creating the image from mysql version 8.0.33.
    - The environment variables are mandatory
    - Pushing the db_back.sql file (containing the mysql db creation) to the docker-entrypoint-initdb.d directory runs and executes the db_backup.sql in the container. According to the documentation the file must end in one of the following extensions: .sh, .sql or .sql.gz



Initializing a fresh instance

When a container is started for the first time, a new database with the specified name will be created and initialized with the provided configuration variables. Furthermore, it will execute files with extensions `.sh`, `.sql` and `.sql.gz` that are found in `/docker-entrypoint-initdb.d`. Files will be executed in alphabetical order. You can easily populate your `mysql` services by mounting a SQL dump into that directory and provide custom images with contributed data. SQL files will be imported by default to the database specified by the `MYSQL_DATABASE` variable.

- The Dockerfile



Docker-files > db > Dockerfile > ...

```
1  FROM mysql:8.0.33
2  LABEL "Project"="Vprofile"
3  LABEL "Author"="Samson"
4
5  ENV MYSQL_ROOT_PASSPORT="vprodbpass"
6  ENV MYSQL_DATABASE="accounts"
7
8  ADD db_backup.sql docker-entrypoint-initdb.d/db_backup.sql
9
10 EXPOSE 3306
```

Mysql Dockerfile steps

- Got the mysql image from dockerhub
- Gave few labels
- Gave few environment variables as specified by the documentation
- Used the 'ADD' command to push and initialize the sql file containing the database/schema creation commands in the container.
- Exposed the port 3306 (implied anyway)

Nginx

- create the nginx configuration file

Nginx Dockerfile	Nginx configuration

Docker-files > web > Dockerfile > ...

```
FROM nginx
LABEL "Project"="Vprofile"
LABEL "Author"="Samson"

RUN rm -rf /etc/nginx/conf.d/default.conf
COPY nginvproapp.conf /etc/nginx/conf.d/
```

Steps

- Got the nginx image from dockerhub
- Gave few labels
- Removed the default configuration and pushed mine to the default location

Docker-files > web > nginvproapp.conf

```
upstream vproapp {
 server vproapp:8080;
}
server {
  listen 80;
location / {
  proxy_pass http://vproapp;
}
}
```

The nginx server listens on port 80 for a container
named vproapp. This means we must name the application container, vproapp

- The container we run will be run in the name - vproapp.

Step 5: Build and test

- Write doocker-compose.yml file to build multiple containers at the same time

docker-compose.yml

```
version: '3.8'
services:
  vprodb:
    build:
      context: ./Docker-files/db # location of the Dockerfile for building the db image
    image: originalugo/cloud_project4_db # should be same name as the dockerhub repo
    container_name: vprodb
    ports:
      - "3306:3306" # port mapping
    volumes:
      - vprodbdata: /var/lib/mysql # mapping the container volumne
    environment:
      - MYSQL_ROOT_PASSWORD=vprodbnpass

  vprocache01:
    image: memcached
    ports:
      - "11211:11211"

  vprormq01:
    image: rabbitmq
    ports:
      - "15672:15672"
    environment:
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS= guest

  vproapp:
    build:
      context: ./Docker-files/app # location of the Dockerfile for building the tomcat image
    image: originalugo/cloud_project4_app # should be same name as the dockerhub repo
    container_name: vproapp
    ports:
      - "8080:8080" # port mapping
    volumes:
      - vprodbdata: /var/local/tomcat/webapps # mapping the container volumne

  vproweb:
    build:
      context: ./Docker-files/web # location of the Dockerfile for building the nginx image
    image: originalugo/cloud_project4_web # should be same name as the dockerhub repo
    container_name: vproweb
    ports:
      - "80:80" # port mapping

volumes:
  vprodbdata: {}
  vproappdata: {}
```

Step 6:

- Logout (if still logged in) from the VM and do a vagrant reload
- Relog in and ssh to the vagrant vm.
- The cloned vprofile repo should be seen in the same directory as the Vagrantfile.
- I cd into the vagrantfile

- Next, I built the images using **`docker compose build`**. This command builds just the custom images we configured using their respective Dockerfiles. These are the images seen when one runs the **`docker images`** command

```
vagrant@vagrant:/vagrant/vprofile-project$ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
originalugo/cloud_project4_app      latest    deb7ed084325   13 seconds ago   327MB
originalugo/cloud_project4_db       latest    efd46cb518ac   4 minutes ago    565MB
originalugo/cloud_project4_web      latest    0a115f70a851   4 minutes ago    188MB
```

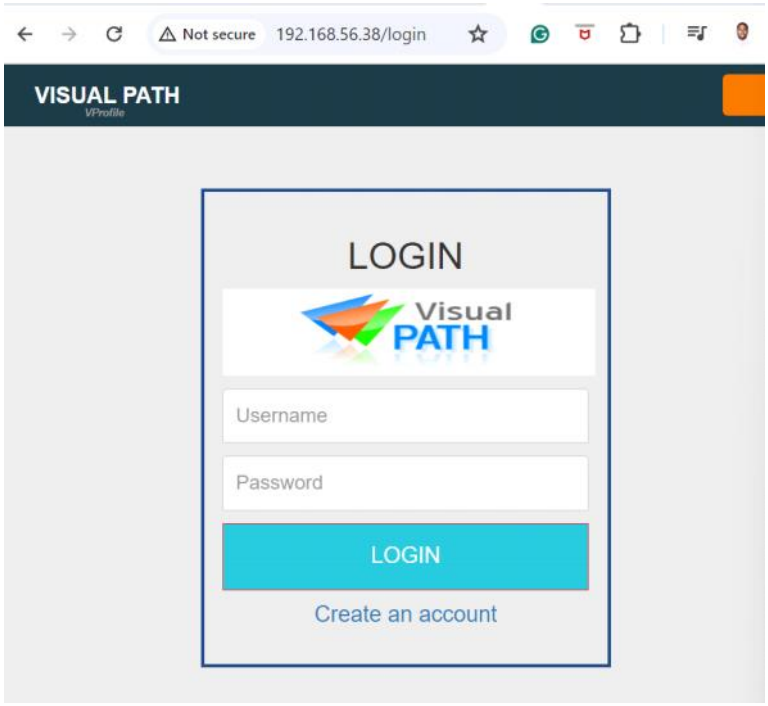
- Next was to bring up the containers from the images using **`docker compose up -d`** ('-d' makes the container run in detached mode). This also builds and run the non-customized images/containers - rabbitmq and memcached

```
vagrant@vagrant:/vagrant/vprofile-project$ docker compose up -d
WARN[0000] /vagrant/vprofile-project/docker-compose.yml: `version` is obsolete
[+] Running 17/17
 ✔ vprormq01 Pulled                                                    11.6s
   ✔ 7646c8da3324 Pull complete                                         5.6s
   ✔ 04462dba3f37 Pull complete                                         7.2s
   ✔ ed4510fcf79f Pull complete                                         7.6s
   ✔ 5281fd20b957 Pull complete                                         7.6s
```

The containers

```
vagrant@vagrant:/vagrant/vprofile-project$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
                                                                NAMES
60a8aee2d9db   originalugo/cloud_project4_web   "/docker-entrypoint.…"   6 minutes ago   Up 6 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp
                                                                vproweb
6524755b44ad   originalugo/cloud_project4_db    "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp
, 33060/tcp                                                     vprodb
0b9a527141a7   rabbitmq                       "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes   4369/tcp, 5671-5672/tcp, 15691-15692/tcp,
 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp   vprofile-project-vprormq01-1
e65bbfb555fb   originalugo/cloud_project4_app   "catalina.sh run"        6 minutes ago   Up 6 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
                                                                vproapp
b7e5115fbbd7   memcached                      "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes   0.0.0.0:11211->11211/tcp, :::11211->11211
/tcp                                                            vprofile-project-vprocache01-1
```

- Get the IP address of the VM by running **`ip addr show`** command
- I then accessed the application using the ip address on my browser and voila…the containerized application



- I can log in using the username and password and validates the different microservices - rabbitmq, mysql, and memcached

Step 7: Push images to dockerhub account

This step is optional, but required if the images used for the project will be reused again.

- I logged in to my docker hub account from my bash cli using **`docker login`** command

```
root@vagrant:/vagrant/vprofile-project# docker login
Log in with your Docker ID or email address to push and pull images from Docker
Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to cre
te one.
You can log in with your password or a Personal Access Token (PAT). Using a lim
ted-scope PAT grants better security and is required for organizations using SS
. Learn more at https://docs.docker.com/go/access-tokens/

Username: originalugo
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- I then pushed the images (app, web and db) to dockerhub

```
root@vagrant:/vagrant/vprofile-project# docker push originalugo/cloud_project4_d
b
Using default tag: latest
The push refers to repository [docker.io/originalugo/cloud_project4_db]
a3a6d1daf036: Pushed
30256473ad17: Mounted from library/mysql
b30f75c501b6: Mounted from library/mysql
f5611ea49cae: Mounted from library/mysql
06af60393523: Mounted from library/mysql
6abf55c795bc: Mounted from library/mysql
5353f7b1372e: Mounted from library/mysql
f5cca8023c34: Mounted from library/mysql
c1e3f0059a6c: Mounted from library/mysql
b1a906a58dc2: Mounted from library/mysql
e19b28b0c15e: Mounted from library/mysql
32f7f5f86853: Mounted from library/mysql
latest: digest: sha256:370fa81c6a6f79bf192a674dffa923c9fe2c393ea26c06b256b9f4a7e
0005aed size: 2826
```

Step 8: clean up

- Stopped the containers using **`docker compose down`**
- Removed all the images/settings using **`docker system prune -a`**

Challenges

- Choosing the right version of docker images for your project was most challenging. The best way I think is by getting the right version from the developers. Without this information, one'll have to do a trial and error test to find the best one.
- Secondly knowing all the required information about the image you're using is important. For instance, I had to be certain of the location of where the default Tomcat application is stored in the container so I could delete and push mine there. All of this information are found from the image documentation on dockerhub.