Udemy Project 8

Goal of the project

- To ultimately containerise the deployment of the Sparta test app and database using Docker and a cloud provider

GitHub repo

Available here

Installing Docker Desktop

- I installed Docker Desktop by following the instructions on the Docker site

---

Research

Virtualisation vs containerisation

- **Virtualisation**: a technology allowing users to run several isolated VMs on one physical server/host machine
- Downsides:
    - heavier than containerisation because it requires a separate, full OS on each VM
    - uses a lot of resources
    - start-up and provisioning can be slow
- **Containerisation**: bundling (or **encapsulating**) code and all its dependencies into an **image**
- An image is then run in instances called a **container**: a lightweight, standalone, and consistent executable package that includes everything needed to run a piece of software and can be reliably deployed across most systems
- An alternative to full virtualisation, containers use an abstracted OS so that they can run consistently on any machine
- So unlike VMs, containers don't include OSes
- these containers operate in isolated environments, so they run independently from the infrastructure they're deployed into
- **Images**: pre-built packages containing everything needed to run an application, including the code, dependencies, and configuration

Benefits of virtualisation vs containerisation

Virtualisation	Containerisation
Compared to traditional architecture, virtualisation **reduces resource (hardware) costs** because VMs can run on one physical machine and can be assigned only the computing power they actually need	Containers are **lighter** than VMs because they share the host machine's OS kernel, so they minimise resource overhead (e.g. they don't use the hypervisors that VMs do) because they don't run separate OS tasks
VMs provide **high availability** via monitoring and scale sets (so they are also very helpful in disaster situations)	Though VMs are portable between machines, containers are more so because they are also **OS-agnostic** (so long as the OSes support the containerisation tool, e.g. Docker) — *write once, run anywhere*
VMs are very **scalable**	The microservices approach means images are dense and often have relatively small file sizes, so they can be **quicker to deploy, share, migrate, and move**
Compared to containerisation, VMs are **more secure** than containers because they are fully isolated (i.e. vulnerabilities in one host kernel won't affect all the others thanks to using different OS kernels in each VM)	The compartmentalisation approach of microservices makes **maintenance** and **changes** to an app **easier to implement**

Microservices

- **Microservices**: splitting large apps into loosely coupled services that run in their own process and can be deployed independently
- Each microservice should be small, as independent as possible, and specialised, with a minimal set of features
- They are a lighter solution to monolith containers, in which the whole app is built into one package and deployed all or nothing
    - Once apps reach a certain size/complexity in monolith architectures, development and start-up slow down
- They communicate using lightweight mechanisms like APIs
- They enable faster feature delivery and scaling for large apps

Benefits

- They allow DevOps teams to introduce new components without causing downtime because microservices are independent
- Improve fault isolation
- Accelerate deployment
- Optimise resource allocation within organisations because teams work on small, well-defined services
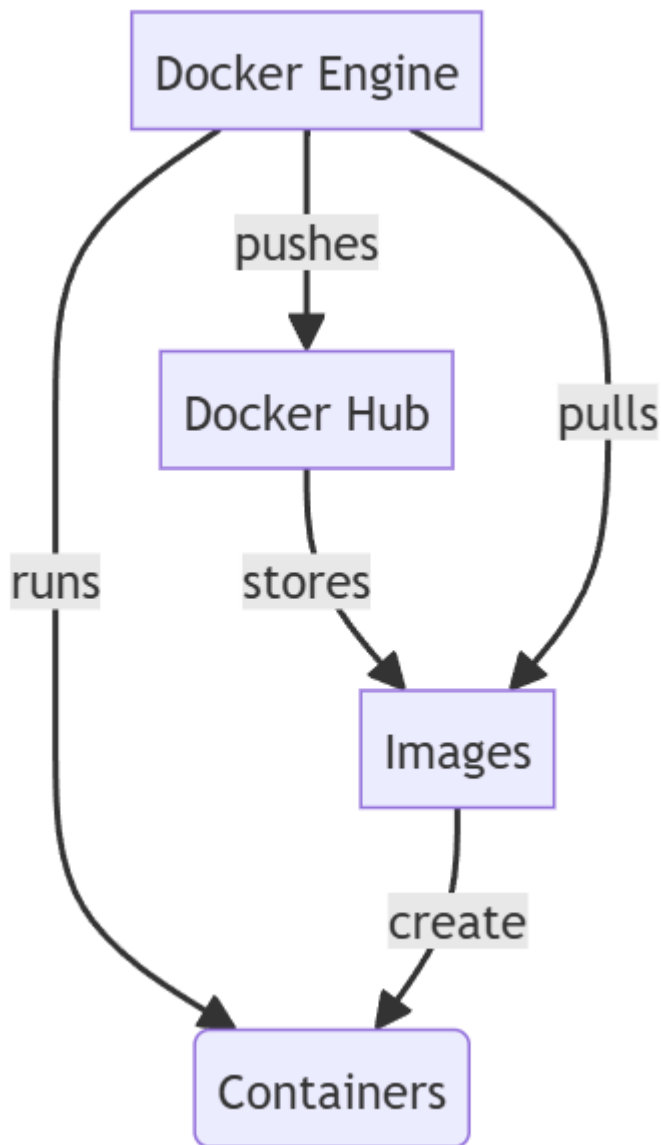
Docker

- An open-source platform designed to help build, share, and run container apps

- **Official images**: images curated by Docker; typically well-maintained and documented so a good choice for beginners
- **Docker Hub**: a cloud-based registry service **serving as a central repository** where Docker users and orgs can find, store, and share their Docker images; like a GitHub for Docker images -**Docker Engine**: The core technology that runs and manages containers on a machine
- **Tags**: aliases that point to versions of an image (e.g. *latest*, *2.1*, or *Linux*); when you don't specify a tag in your commands, Docker assumes you want the "*latest*" tag
- Benefits of tags:
  - Useful for **version control** because you can tag images with version numbers (e.g. v1, v2)
  - **Environment separation**: you might tag images for different environments (e.g. dev, staging, prod)
  - **Readability**: custom tags can make it clearer what an image is for
- **Pull command**: On each image's page, there is a "pull command"; this is what you'd use to manually download the image without running a container; e.g. `docker pull hello-world`

```
farah@Farah-laptop MINGW64 ~
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
e6590344b1a5: Download complete
Digest: sha256:e0b569a5163a5e6be84e210a2587e7d447e08f87a0e90798363fa44a0464a1e8
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

- Docker images are built using a **layered filesystem**, with each layer representing a command in the Dockerfile used to build the image
  - this layered approach allows Docker to be efficient with storage and network usage
  - each layer is represented by a long string (called a SHA256 hash)
  - they are cached, speeding up builds of similar images
  - when pushing or pulling images, only changed layers are transferred
- I also researched Docker commands here

Dockerfile

- **Dockerfile**: a text file that contains all the commands a user should call on the CLI to assemble an image; can be helpful to understand how the image was created
- these are named with a capital D and have no extension
- each container directory **can only have one** Dockerfile
- A Dockerfile includes the **base image** to be used, **commands** telling Docker how to set up the image, and the **start up** command
- must begin with a `FROM` instruction, which specifies the base image to be built from
- other common instructions include `RUN` e.g. to install dependencies, `COPY` to copy local files into the image, `EXPOSE` which tells Docker that the container listens on given ports, and `CMD` which defines the default command to run on start-up
- an **ENTRYPOINT** instruction can also reference a script file to specify the commands to be run when the container starts, e.g.

```
FROM nginx
ENV NGINX_PORT 9100
COPY start.sh /start.sh
```

```
RUN chmod +x /start.sh
ENTRYPOINT ["/start.sh"]
```

- you must then also create the file referenced in the ENTRYPOINT instruction (i.e. */start.sh* here)

Docker Compose

- **Docker Compose**: a YAML file that allows you to run multi-container apps and manage each container simultaneously
- allows these linked containers to be spun up or torn down with `docker compose up` and `docker compose down` (there are also optional flags that can be used with these commands; see here)
- standard name is `compose.yml` or `compose.yaml`
- example syntax:

```
services:
  app:
    image: <imagename:optional tag>
    container_name: <optional container name>
    environment:
      - <optional environment variable>
    ports:
      - <ports to be used by service>
    depends_on:
      - <another service defined in the Compose file that is required for this
service>
```

- other than `services`, blocks could be `networks`, `volumes`, `configs`, or `secrets`

Alternatives to Docker

- **Podman**:
  - developed by RedHat
  - daemon-less, so doesn't need root privileges and thus reduces attack surfaces
  - fully open-source (whereas Docker requires licenses for orgs with 150+ employees or $10+ million revenue)
  - but less support for non-Linux OSes
- **Rancher Desktop**:
  - fully open-source
  - Kubernetes-focused (which can be a pro and a con)
  - has a GUI for Kubernetes, so is more beginner-friendly

Success story using Docker

- Uber uses Docker for microservices for passenger management, billing, driver management, trip management, and notifications

Task 1: Learning to manage Docker containers locally

Run and pull my first image

1. Get help from a Docker command

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/t
ench-udemy-pathways/Project 8 (main)
$ docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run          Create and run a new container from an image
  exec         Execute a command in a running container
  ps           List containers
  build        Build an image from a Dockerfile
  pull         Download an image from a registry
```

2. Run a Docker command to show all the images I have on my local machine

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-
ench-udemy-pathways/Project 8 (main)
$ docker images
REPOSITORY                 TAG         IMAGE ID        CREATED         SIZE
project8-app-db-method2     latest      1745ccad5954    3 hours ago     1.7GB
farahc123/sparta-app        v1          95c36fe3ca41    26 hours ago    1.66GB
farahc123/nginx-dreamteam   latest      9845584b7449    30 hours ago    323MB
mongo                       latest      961312ca9515    3 days ago      1.19GB
nginx                       latest      9d6b58feebd2    2 weeks ago     279MB
hello-world                 latest      e0b569a5163a    4 weeks ago     20.4kB
daraymonsta/nginx-257       dreamteam   aca9f1774d81    10 months ago   313MB
mongo                       7.0         048559287a51    55 years ago    270MB
```

3. Run my first Docker container using the `hello-world` image and then re-run the image to see that it doesn't need to be downloaded again

```
farah@Farah-laptop MINGW64 ~
$ docker images
REPOSITORY    TAG        IMAGE ID    CREATED    SIZE

farah@Farah-laptop MINGW64 ~
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Download complete
Digest: sha256:e0b569a5163a5e6be84e210a2587e7d447e08f87a0e90798363fa44a0464a1e8
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

farah@Farah-laptop MINGW64 ~
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Run Nginx web server in a Docker container

1. Download the latest Nginx Docker image using a `docker pull` command, run it so that it exposes the running container on port 80 on my local machine, and run a `docker ps` command to check if it's

running

```
farah@Farah-laptop MINGW64 ~
$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
84cade77a831: Download complete
6c78b0ba1a32: Download complete
c558df217949: Download complete
24ff42a0d907: Download complete
c29f5b76f736: Download complete
e19db8451adb: Download complete
976e8f6b25dd: Download complete
Digest: sha256:91734281c0ebfc6f1aea979cffeed5079cfe786228a71cc6f1f46a228cde6e34
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

farah@Farah-laptop MINGW64 ~
$ docker run -d -p 80:80 nginx:latest
2ff9f5014b6fb6de85cb137c1d37a76fd23d6a06fb3ff571af0b124e78afecda

farah@Farah-laptop MINGW64 ~
$ docker ps
CONTAINER ID    IMAGE           COMMAND                 CREATED         STATUS
       PORTS               NAMES
2ff9f5014b6f    nginx:latest    "/docker-entrypoint.…"  9 seconds ago   Up 8 seco
nds    0.0.0.0:80->80/tcp    nervous_mcclintock
```

2. Check it's working by going to *localhost* or *127.0.0.1* in a web browser

```
←    →    C      ⓘ   127.0.0.1

⊞    |    ▢  SpartaGlobal learni...
```

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

3. Stop the container running

```
farah@Farah-laptop MINGW64 ~
$ docker ps
CONTAINER ID    IMAGE           COMMAND                 CREATED           STATUS            PORTS               NAMES
2ff9f5014b6f    nginx:latest    "/docker-entrypoint.…"  About a minute ago  Up About a minute  0.0.0.0:80->80/tcp  nervous_mcclintock
c
farah@Farah-laptop MINGW64 ~
$ docker stop 2ff9f5014b6f
2ff9f5014b6f

farah@Farah-laptop MINGW64 ~
$ docker ps
CONTAINER ID    IMAGE       COMMAND     CREATED     STATUS     PORTS       NAMES

farah@Farah-laptop MINGW64 ~
$ |
```

Remove a container

1. Re-start the Nginx container I was previously running and, while the Nginx container is running, try to remove the container with `docker rm` – take note of the error

```
farah@Farah-laptop MINGW64 ~
$ docker run -d -p 80:80 --name my-nginx nginx
17e2e9b0abc402a328499dc87a638bcb401ae9c327675f8c33b57f3e81de9ff4

farah@Farah-laptop MINGW64 ~
$ docker rm my-nginx
Error response from daemon: cannot remove container "/my-nginx": container is ru
nning: stop the container before removing or force remove
```

2. Work out the switch/option needed to use to forcibly remove a container while it's running

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 8 (main)
$ docker rm -f my-nginx
```

3. Run the docker command to check whether the container is still there

```
farah@Farah-laptop MINGW64 ~
$ docker ps -a
CONTAINER ID    IMAGE        COMMAND      CREATED      STATUS      PORTS        NAMES

farah@Farah-laptop MINGW64 ~
```

Modify the Nginx default page in the running container

1. Re-run the Nginx container exposed on port 80 of my local machine

```
farah@Farah-laptop MINGW64 ~
$ docker run -d -p 80:80 --name my-nginx nginx
41869e9da7ade1e4e89f41480f7e81d6441dda34622dfe24aded6084d799b359
```

2. Check the default webpage of the container in my web browser and keep it open for later

127.0.0.1

SpartaGlobal learni...

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

3. Access the shell of the Nginx container that is running with with `docker exec -it my-nginx sh` (use

```
farah@Farah-laptop MINGW64 ~
$ docker exec -it my-nginx sh
#
```

sh not /bin/sh)

4. After logging into the shell of the Nginx container, try to do an update & upgrade with `sudo` — notice the problem, then install sudo to fix it - had to do `apt update` first then install sudo

```
# sudo apt update && apt upgrade -y
sh: 3: sudo: not found
# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB
]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8792 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.5 kB
]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Package
s [246 kB]
Fetched 9306 kB in 4s (2648 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
# apt install sudo -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  sudo
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 1889 kB of archives.
After this operation, 6199 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 sudo amd64 1.9.13p3-1+deb12u1 [1889 kB]
Fetched 1889 kB in 1s (2167 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package sudo.
(Reading database ... 7580 files and directories currently installed.)
Preparing to unpack .../sudo_1.9.13p3-1+deb12u1_amd64.deb ...
Unpacking sudo (1.9.13p3-1+deb12u1) ...
Setting up sudo (1.9.13p3-1+deb12u1) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for libc-bin (2.36-9+deb12u9) ...
#
```

- I could then update and upgrade as normal

```
# sudo apt update && apt upgrade -y
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian bookworm-updates InRelease
Hit:3 http://deb.debian.org/debian-security bookworm-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  libgnutls30 libtasn1-6
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 1454 kB of archives.
After this operation, 8192 B disk space will be freed.
Get:1 http://deb.debian.org/debian-security bookworm-security/main amd64 libtasn1-6 amd64 4.19.0-2+deb12u1 [48.6 kB]
Get:2 http://deb.debian.org/debian-security bookworm-security/main amd64 libgnutls30 amd64 3.7.9-2+deb12u4 [1405 kB]
Fetched 1454 kB in 0s (2990 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
(Reading database ... 7723 files and directories currently installed.)
Preparing to unpack .../libtasn1-6_4.19.0-2+deb12u1_amd64.deb ...
Unpacking libtasn1-6:amd64 (4.19.0-2+deb12u1) over (4.19.0-2) ...
Setting up libtasn1-6:amd64 (4.19.0-2+deb12u1) ...
(Reading database ... 7723 files and directories currently installed.)
Preparing to unpack .../libgnutls30_3.7.9-2+deb12u4_amd64.deb ...
Unpacking libgnutls30:amd64 (3.7.9-2+deb12u4) over (3.7.9-2+deb12u3) ...
Setting up libgnutls30:amd64 (3.7.9-2+deb12u4) ...
Processing triggers for libc-bin (2.36-9+deb12u9) ...
# |
```

5. Check my present working directory, then navigate to where the default Nginx file is kept

   ○ Use nano to edit index.html – notice the problem, then fix it so I can use nano

```
# cd /usr/share/nginx/html
# ls
50x.html  index.html
# sudo nano index.html
```

- ○ I had to install nano first

```
sudo: nano: command not found
# apt install nano
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libgpm2 libncursesw6
Suggested packages:
  gpm hunspell
The following NEW packages will be installed:
  libgpm2 libncursesw6 nano
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 838 kB of archives.
After this operation, 3339 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://deb.debian.org/debian bookworm/main amd64 libncursesw6 amd64 6.4-4
[134 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64 nano amd64 7.2-1+deb12u1
[690 kB]
Get:3 http://deb.debian.org/debian bookworm/main amd64 libgpm2 amd64 1.20.7-10+b
1 [14.2 kB]
Fetched 838 kB in 0s (1991 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libncursesw6:amd64.
(Reading database ... 7723 files and directories currently installed.)
```

6. Once nano works, modify the HTML of the file so that instead of "Welcome to nginx!" the home page will be "Welcome to the Tech xxx Dreamteam!" – save my changes

```
# cat /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to the Tech 501 Dreamteam!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to the Tech 501 Dreamteam!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

7. Check how the default web page in the browser has changed

← → C ⓘ 127.0.0.1

⊞ | ▢ SpartaGlobal learni...

Welcome to the Tech 501 Dreamteam!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Run a different container on a different port

1. Try to run another container exposed on port 80 on my local machine (connect to port 80 inside the container) – here is the endpoint for the image to use: daraymonsta/nginx-257:dreamteam

```
farah@Farah-laptop MINGW64 ~
$ docker run -d -p 80:80 daraymonsta/nginx-257:dreamteam
Unable to find image 'daraymonsta/nginx-257:dreamteam' locally
dreamteam: Pulling from daraymonsta/nginx-257
305d309a0ba5: Download complete
6fcdffcd79f0: Download complete
c9590dd9c988: Download complete
0cef83155b8b: Download complete
fbf231d461b3: Download complete
13808c22b207: Download complete
abaefc5fcbde: Download complete
04033143d859: Download complete
Digest: sha256:aca9f1774d81786850052224e68a247259ec8d1790d14694aea373feaf57c03f
Status: Downloaded newer image for daraymonsta/nginx-257:dreamteam
ccd4b481ded8d29ea87f0be63d48d31980275e13487fd22ef161f9bf18c08410
docker: Error response from daemon: driver failed programming external connectiv
ity on endpoint elastic_banach (12d6bd540db2a11f1b467b86243f7801e37b0ae7010d30f0
995039b64f33b9d8): Bind for 0.0.0.0:80 failed: port is already allocated.
```

   - occurs because Docker can't bind two containers to the same port as it's already in use by my-nginx container as seen here:

```
$ docker ps
CONTAINER ID    IMAGE      COMMAND               CREATED          STATUS
 PORTS                     NAMES
41869e9da7ad    nginx       "/docker-entrypoint.…"   28 minutes ago   Up 7 minutes
 0.0.0.0:80->80/tcp    my-nginx
```

2. Remove the container we tried to run but couldn't
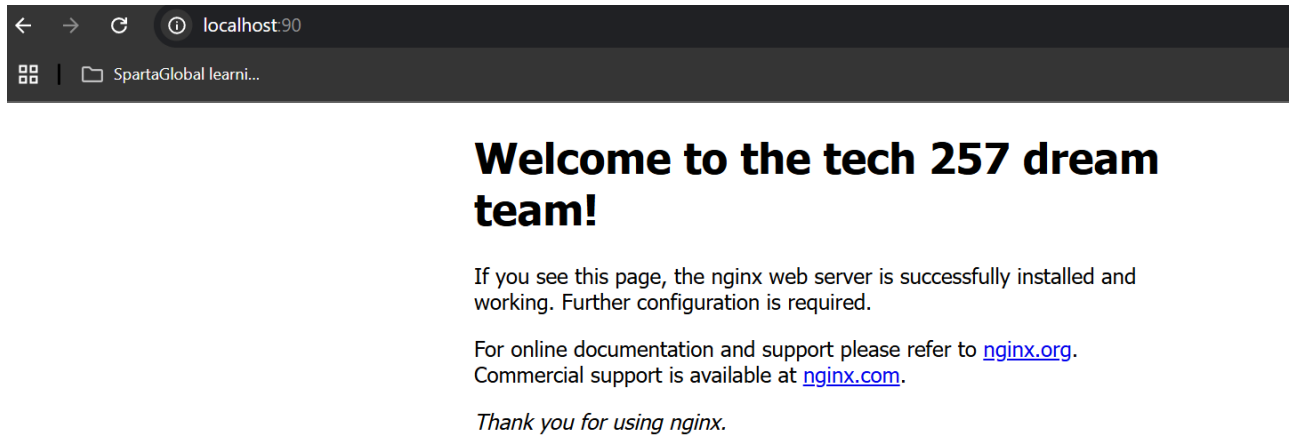
```
farah@Farah-laptop MINGW64 ~
$ docker ps -a
CONTAINER ID    IMAGE                            COMMAND               CREATE
D           STATUS          PORTS           NAMES
ccd4b481ded8    daraymonsta/nginx-257:dreamteam   "/docker-entrypoint.…"   2 minu
tes ago     Created                         elastic_banach
41869e9da7ad    nginx                            "/docker-entrypoint.…"   29 min
utes ago   Up 8 minutes    0.0.0.0:80->80/tcp    my-nginx

farah@Farah-laptop MINGW64 ~
$ docker rm ccd4b481ded8
ccd4b481ded8

farah@Farah-laptop MINGW64 ~
$ docker ps -a
CONTAINER ID    IMAGE      COMMAND               CREATED          STATUS
 PORTS                     NAMES
41869e9da7ad    nginx       "/docker-entrypoint.…"   30 minutes ago   Up 9 minutes
 0.0.0.0:80->80/tcp    my-nginx
```

3. Try to run the container from step 2 again but this time expose it on my local machine on port 90. with `docker run -d -p 90:80 daraymonsta/nginx-257:dreamteam`; check this container is running in my web browser at `localhost:90`

Welcome to the tech 257 dream team!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Blockers

1. On the Nginx modification task, I ran into this error when running an interactive command to enter the container's shell for the first time because Git Bash has issues with TTY-based commands (i.e. it was trying to use a Windows Git Bash path instead of the correct shell inside the Nginx container)

```
farah@Farah-laptop MINGW64 ~
$ docker exec -it my-nginx /bin/sh
OCI runtime exec failed: exec failed: unable to start container process: exec: '
C:/Program Files/Git/usr/bin/sh": stat C:/Program Files/Git/usr/bin/sh: no such
file or directory: unknown
```

   - To solve this, I installed **winpty**, a Windows software package that prevents these errors when running interactive commands
   - By default, I would have had to preface every interactive command with `winpty`, so I created an

```
farah@Farah-laptop MINGW64 ~
$ alias docker="winpty docker"
```

   alias which I then added to the *.bashrc* file

2. I also had issues trying to use the `/bin/sh/` shell in my `docker exec -it` command, so I switched to

```
farah@Farah-laptop MINGW64 ~
$ docker exec -it my-nginx sh
```

   `sh` `#`

3. I had an "Unable to locate package sudo" error because the Nginx Docker image is based on Debian Slim, which doesn't include `sudo` by default

```
# sudo apt update && apt upgrade -y
sh: 3: sudo: not found
# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB
]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8792 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.5 kB
]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Package
s [246 kB]
Fetched 9306 kB in 4s (2648 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
# apt install sudo -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  sudo
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 1889 kB of archives.
After this operation, 6199 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 sudo amd64 1.9.13p3-1+deb12u1 [1889 kB]
Fetched 1889 kB in 1s (2167 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package sudo.
(Reading database ... 7580 files and directories currently installed.)
Preparing to unpack .../sudo_1.9.13p3-1+deb12u1_amd64.deb ...
Unpacking sudo (1.9.13p3-1+deb12u1) ...
Setting up sudo (1.9.13p3-1+deb12u1) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for libc-bin (2.36-9+deb12u9) ...
#
```

- I solved this by running `apt update` and then `apt install sudo`
- I was then able to `sudo apt update` and `upgrade` as normal

```
# sudo apt update && apt upgrade -y
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian bookworm-updates InRelease
Hit:3 http://deb.debian.org/debian-security bookworm-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  libgnutls30 libtasn1-6
2 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 1454 kB of archives.
After this operation, 8192 B disk space will be freed.
Get:1 http://deb.debian.org/debian-security bookworm-security/main amd64 libtasn1-6 amd64 4.19.0-2+deb12u1 [48.6 kB]
Get:2 http://deb.debian.org/debian-security bookworm-security/main amd64 libgnutls30 amd64 3.7.9-2+deb12u4 [1405 kB]
Fetched 1454 kB in 0s (2990 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
(Reading database ... 7723 files and directories currently installed.)
Preparing to unpack .../libtasn1-6_4.19.0-2+deb12u1_amd64.deb ...
Unpacking libtasn1-6:amd64 (4.19.0-2+deb12u1) over (4.19.0-2) ...
Setting up libtasn1-6:amd64 (4.19.0-2+deb12u1) ...
(Reading database ... 7723 files and directories currently installed.)
Preparing to unpack .../libgnutls30_3.7.9-2+deb12u4_amd64.deb ...
Unpacking libgnutls30:amd64 (3.7.9-2+deb12u4) over (3.7.9-2+deb12u3) ...
Setting up libgnutls30:amd64 (3.7.9-2+deb12u4) ...
Processing triggers for libc-bin (2.36-9+deb12u9) ...
#
```

4. I had the same issue with nano, which I solved by installing it

---

Task 2: Using Docker Hub to host custom images

- To easily access, store, and share images via Docker Hub's central repository

Push host-custom-static-webpage container

1. Create an image from the running Nginx container with the modified *index.html* file and push it image to my Docker Hub account

```
farah@Farah-laptop MINGW64 ~
$ docker commit my-nginx farahc123/nginx-dreamteam
sha256:9845584b74497d2d84170c513d98e173c5137c225652ccf5b85e595a08fb0f87

farah@Farah-laptop MINGW64 ~
$ docker login
Authenticating with existing credentials...
Login Succeeded

farah@Farah-laptop MINGW64 ~
$ docker push farahc123/nginx-dreamteam
Using default tag: latest
The push refers to repository [docker.io/farahc123/nginx-dreamteam]
eefef33fde88: Pushed
c558df217949: Mounted from library/nginx
24ff42a0d907: Mounted from library/nginx
6c78b0ba1a32: Mounted from library/nginx
c29f5b76f736: Mounted from library/nginx
e19db8451adb: Mounted from library/nginx
84cade77a831: Mounted from library/nginx
976e8f6b25dd: Mounted from library/nginx
latest: digest: sha256:9845584b74497d2d84170c513d98e173c5137c225652ccf5b85e595a0
8fb0f87 size: 2042
```

2. Use a Docker command to run the container using the above pushed container image, which contains my username on Docker Hub.

```
farah@Farah-laptop MINGW64 ~
$ docker images
REPOSITORY                    TAG         IMAGE ID        CREATED
farahc123/nginx-dreamteam     latest      9845584b7449    About a minute
B
nginx                         latest      91734281c0eb    2 weeks ago
B
hello-world                   latest      e0b569a5163a    4 weeks ago
kB
daraymonsta/nginx-257         dreamteam   aca9f1774d81    10 months ago
B

farah@Farah-laptop MINGW64 ~
$ docker run -d farahc123/nginx-dreamteam
81c5348ccb5196c26cc203d255d0f39a88d8182f6e76e04bc21807bc14897b1f
```

Automate Docker image creation using a Dockerfile

1. To create a Dockerfile, in the repo (that won't get published), run `nano Dockerfile` (with no extensions)
   - my Dockerfile in tech501-mod-nginx-dockerfile

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ ls
Dockerfile  index.html
```

2. Create an *index.html* file to use instead of the Nginx default page

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ cat index.html
<!DOCTYPE html><html><head><title>Welcome to the Tech 501 Dreamteam!</title></he
ad><body><h1>Welcome to the Tech 501 Dreamteam!</h1></body></html>
```

3. Create a Dockerfile to use the Nginx base image, copy the *index.html* to the location of the Nginx default page in the container, use a `docker build` command to build the custom image, and tag it as *tech501-nginx-auto:v1*

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ echo '<!DOCTYPE html><html><head><title>Welcome to the Tech 501 Dreamteam!</ti
tle></head><body><h1>Welcome to the Tech 501 Dreamteam!</h1></body></html>' > in
dex.html

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ ls
index.html

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ nano Dockerfile

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ ls
Dockerfile   index.html

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ docker build -t tech501-nginx-auto:v1
ERROR: "docker buildx build" requires exactly 1 argument.
See 'docker buildx build --help'.

Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ docker build -t tech501-nginx-auto:v1 .
[+] Building 1.5s (8/8) FINISHED                             docker:desktop-linux
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 277B                                        0.0s
 => [internal] load metadata for docker.io/library/nginx:latest            0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 186B                                          0.0s
 => [1/2] FROM docker.io/library/nginx:latest@sha256:91734281c0ebfc6f1aea  1.1s
 => => resolve docker.io/library/nginx:latest@sha256:91734281c0ebfc6f1aea  1.0s
 => [auth] library/nginx:pull token for registry-1.docker.io              0.0s
 => [2/2] COPY index.html /usr/share/nginx/html/index.html                0.0s
 => exporting to image                                                     0.2s
 => => exporting layers                                                    0.1s
 => => exporting manifest sha256:522646e0da62c0114dc21f0704ff8940ae3eb617  0.0s
 => => exporting config sha256:46e49307544dd7ec825d0ac76fe290280c56d925d4  0.0s
 => => exporting attestation manifest sha256:426b5b6584a5a4f86af696b86116  0.0s
 => => exporting manifest list sha256:08cbda84cc3c04fb733f42580c148aa64e8  0.0s
 => => naming to docker.io/library/tech501-nginx-auto:v1                   0.0s
```

- tagging image before pushing

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngi
x-dockerfile
$ docker tag tech501-nginx-auto:v1 farahc123/tech501-nginx-auto:v1
```
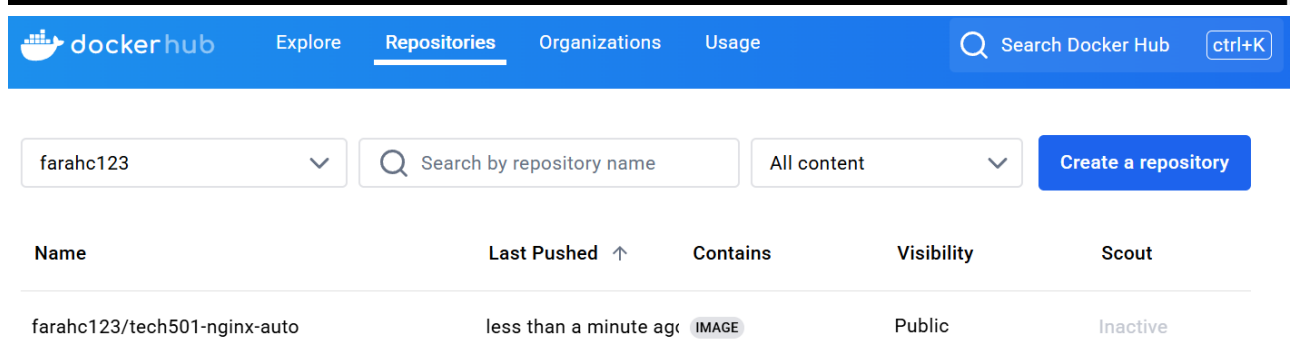
4. Push the custom image to Docker Hub

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ docker push farahc123/tech501-nginx-auto:v1
The push refers to repository [docker.io/farahc123/tech501-nginx-auto]
c558df217949: Mounted from farahc123/nginx-dreamteam
c29f5b76f736: Mounted from farahc123/nginx-dreamteam
e19db8451adb: Mounted from farahc123/nginx-dreamteam
976e8f6b25dd: Mounted from farahc123/nginx-dreamteam
84cade77a831: Mounted from farahc123/nginx-dreamteam
4e8fc9e2aea8: Pushed
55b46b75c58d: Pushed
6c78b0ba1a32: Mounted from farahc123/nginx-dreamteam
24ff42a0d907: Mounted from farahc123/nginx-dreamteam
v1: digest: sha256:08cbda84cc3c04fb733f42580c148aa64e8b61c3b6aaa83d20b9f6a80aea2
ff7 size: 856
```

| dockerhub | Explore | **Repositories** | Organizations | Usage | 🔍 Search Docker Hub | ctrl+K |

| farahc123 ▾ | 🔍 Search by repository name | All content ▾ | **Create a repository** |

Name	Last Pushed ↑	Contains	Visibility	Scout
farahc123/tech501-nginx-auto	less than a minute ago IMAGE		Public	Inactive

5. Use a Docker command to run the container which uses the pushed custom image

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ docker run -d -p 80:80 farahc123/tech501-nginx-auto:v1
3335733be63dd8e9dbb95fe7b639a757bd2526b366c628f2379bbd3a3ce05a67
```

6. Remove the local copy of the custom image with `docker rmi farahc123/tech501-nginx-auto:v1`

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngi
x-dockerfile
$ docker rmi farahc123/tech501-nginx-auto:v1
Untagged: farahc123/tech501-nginx-auto:v1
```

7. Re-running the container and forcing Docker to pull the custom image from Docker Hub

```
$ docker images
REPOSITORY                TAG          IMAGE ID       CREATED             SIZE
farahc123/nginx-dreamteam latest       9845584b7449   About an hour ago   323MB
nginx                     latest       91734281c0eb   2 weeks ago         279MB
hello-world               latest       e0b569a5163a   4 weeks ago         20.4k
B
daraymonsta/nginx-257     dreamteam    aca9f1774d81   10 months ago       313MB

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/tech501-mod-ngin
x-dockerfile
$ docker run -d -p 80:80 farahc123/tech501-nginx-auto:v1
Unable to find image 'farahc123/tech501-nginx-auto:v1' locally
v1: Pulling from farahc123/tech501-nginx-auto
55b46b75c58d: Already exists
Digest: sha256:08cbda84cc3c04fb733f42580c148aa64e8b61c3b6aaa83d20b9f6a80aea2ff7
Status: Downloaded newer image for farahc123/tech501-nginx-auto:v1
ce337c228a81e95781801dadad70de98548421b8e9977b7a118f62dc3d189f42
```

Blockers

- I originally didn't tag my image, so I needed to run a `docker tag` before I could push it

Task 3: Running Sparta test app in a container using Dockerfile and Docker Compose

- The goal of this project is to containerise the deployment of the Sparta test app and the database using Docker

Initial steps and manual method

1. Created a folder for this task (because you can only have one Dockerfile per folder)

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 8 (main)
$ ls
'Docker repo'/   Notes.md   README.md   image.png   images/
```

2. I created a Dockerfile and a Docker Compose file
3. The posts hadn't been seeded on *localhost:3000/posts* (this was expected)

```
←  →  C   ⓘ  localhost:3000/posts
⊞  |  🗀  SpartaGlobal learni...
```

SPARTA
GLOBAL

Recent Posts

4. To remedy this, I first seeded the posts page manually with `docker exec -it sparta-app-container node seeds/seed.js`

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech
Project 8/Docker repo (main)
$ docker exec -it farah-sparta-app-container node seeds/seed.js
Connected to database
Database cleared
Database seeded with 100 records
Database connection closed
```

- This worked:

Automatic method 1

- Automatically seeding the database using one command in the Docker Compose file (all files here):

  - **Compose file**:

◦ **Dockerfile**:

```
Dockerfile > ...
1    # Using NodeJS version 20
2    FROM node:20
3
4    # Custom label adding metadata to the image
5    LABEL description="Farah's image of the Sparta test app"
6
7    # Set the default working directory
8    WORKDIR /usr/src/app
9
10   # Copy the application first (including seeds directory)
11   COPY app/ .
12
13   # Expose port 3000
14   EXPOSE 3000
15
16   # setting startup command
17   CMD ["npm", "start"]
```

◦ **Results**:

Automatic method 2

- Automatically seeding the database using a health-check on the MongoDB container in my Docker Compose file (all files here):

○ **Compose file:**

```yaml
compose.yaml
  ▷ Run All Services
1  services:
     ▷ Run Service
2    app:
3      build:
4        context: .
5        args:
6          - DB_HOSTS=mongodb://mongo_db:27017/posts  # Define DB_HOSTS before build
7      container_name: sparta-test-app
8      ports:
9        - "3000:3000"  # Expose Node.js directly (for debugging)
10     depends_on:
11       mongo_db:
12         condition: service_healthy  # Ensure MongoDB is fully ready before app starts
13     environment:
14       - DB_HOST=mongodb://mongo_db:27017/posts
15
     ▷ Run Service
16   mongo_db:
17     image: mongo:latest
18     container_name: mongodb
19     restart: always
20     ports:
21       - "27017:27017"  # Optional, only needed if you want to connect to MongoDB from your local machine
22     volumes:
23       - mongo_db_data:/data/db  # Persist MongoDB data
24     healthcheck:  # Add health check to ensure MongoDB is fully ready before app starts
25       test: ["CMD", "mongosh", "--eval", "db.runCommand({ ping: 1 })"]
26       interval: 10s
27       retries: 5
28       start_period: 20s
29
30 volumes:
31   mongo_db_data:
```

○ **Dockerfile:**

```dockerfile
Dockerfile ×    ⓘ README.md    compose.yaml    store.md    nginx.co

Dockerfile > ...
1    FROM node:latest
2
3    # Set working directory
4    WORKDIR /usr/src/app
5
6    # Copy your Node.js application into the container
7    COPY ./app/ /usr/src/app/
8
9    # Expose the port (assuming your Node.js app listens on 3000)
10   EXPOSE 3000
11
12   # Install dependencies & start app (ensures MongoDB is ready)
13   CMD npm install && npm start
```

- **Results** after running `docker compose up -d`:
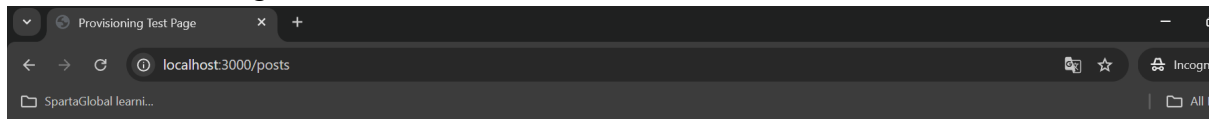
- I later edited my files to add Nginx in order to configure a reverse proxy:
  - **Dockerfile**:

- **Compose file**:

```
services:
  ▷ Run Service
  nginx:
    image: nginx:latest
    ports:
      - "80:80"  # Forward external traffic to nginx
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro  # Use custom nginx config
    depends_on:
      - app
  ▷ Run Service
  app:
    build:
      context: .
      args:
        - DB_HOSTS=mongodb://mongo_db:27017/posts  # Define DB_HOSTS before build
    container_name: sparta-test-app
    ports:
      - "3000:3000"  # Expose Node.js directly (for debugging)
    depends_on:
      mongo_db:
        condition: service_healthy  # Ensure MongoDB is fully ready before app starts
    environment:
      - DB_HOST=mongodb://mongo_db:27017/posts

  ▷ Run Service
  mongo_db:
    image: mongo:latest
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"  # Optional, only needed if you want to connect to MongoDB from your local machine
    volumes:
      - mongo_db_data:/data/db  # Persist MongoDB data
    healthcheck:  # Add health check to ensure MongoDB is fully ready before app starts
      test: ["CMD", "mongosh", "--eval", "db.runCommand({ ping: 1 })"]
      interval: 10s
      retries: 5
      start_period: 20s

volumes:
  mongo_db_data:
```

- **Results** (note the lack of *:3000* in the URL):

Blockers

- I ran into some issues getting the automatic methods to work, which turned out to be syntax issues, so these were easily fixable

Extension task: Running the containers on an EC2 instance

- The goal of this project is to containerise the deployment of the Sparta test app and the database using Docker and AWS
- This replicates a possible production task (deploying the app onto a VM)

Steps

1. After creating an EC2 instance running on Ubuntu 22.04, I had to install Docker on Ubuntu 22.04 via the CLI using these steps
2. I then had to add myself to the Docker group on my EC2 to avoid having to use `sudo` for every `docker` command following these steps
3. On my local machine, I created a **tar** file with `docker save -o sparta-app-image.tar project8-app-db-method2` — this saved the Sparta test app image to a file that I could then transfer to my EC2 instance
4. I then SCPed this tar file to the EC2 instance using `scp -i ~/.ssh/tech501-farah-aws-key.pem sparta-app-image.tar ubuntu@ec2-52-215-19-68.eu-west-1.compute.amazonaws.com:~`

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 8/automatic method 1 docker (main)
$ scp -i ~/.ssh/tech501-farah-aws-key.pem sparta-app-image.tar ubuntu@ec2-52-215
-19-68.eu-west-1.compute.amazonaws.com:~
sparta-app-image.tar                               0%  255KB 248.5KB/s   27:19 ETA
```

5. Once this tar file was on the EC2 instance, I loaded the image from it with `docker load -i sparta-app-image.tar`
6. I then modified my original Dockerfile to make it work with the tar file (i.e. by removing the `COPY app/ .` command; see below for explanation)

```
# Using NodeJS version 20
FROM node:20

# Adding metadata to the image
LABEL description="Farah's image of the Sparta test app"

# Sets the default working directory
WORKDIR /usr/src/app

# Exposes port 3000
EXPOSE 3000

# Sets startup command
CMD ["npm", "start"]
```

7. I replicated my original Docker Compose file

```
services:
  app:
    image: farahc123/sparta-app:v1
    container_name: farah-sparta-app-container
    environment:
      - DB_HOST=mongodb://db:27017/posts    # MongoDB connection string
    ports:
      - "3000:3000"  # Exposes port 3000 for Sparta test app
    depends_on:
      - db
    command: ["/bin/sh", "-c", "npm install && node seeds/seed.js && npm start"]

  db:
    image: mongo:latest
    container_name: mongodb-container
    ports:
      - "27017:27017"  # Exposes port 27017 for MongoDB container
    volumes:
      - mongo_data:/data/db  # Persists MongoDB data in a volume

volumes:
  mongo_data:  # Defines a volume for MongoDB data
```

8. I then built the image with `docker build -t sparta-app .`

9. I started it with `docker compose up -d`

10. Results: working *posts* page

Recent Posts

Credit Card Account Creative Gorgeous

Culpa qui velit ducimus aliquam nihil. Est pariatur ab id ex. Vitae qui velit dolorem omnis aperiam nemo qui. Voluptates odio odit praesentium accusantium. Nobis animi ut provident quos doloremque est dolores. Quisquam beatae non reprehenderit eaque aliquid enim culpa earum eos. Sit repellendus placeat eaque quis quae. Consequatur ipsam quia et excepturi autem. Quia est fugiat. Libero autem est consectetur et dolore velit quis nesciunt. Corrupti similique velit inventore recusandae. Aliquam quasi voluptatum enim et. Exercitationem autem at tenetur hic quam porro aut non et. Eveniet impedit consequuntur ut in sint consequuntur aperiam mollitia.

Blockers

- I realised I had to add myself to the Docker group to give myself the correct permissions so that I didn't need to preface every `docker` command with `sudo`
- I also realised I couldn't just reuse my original Dockerfile because I no longer needed to copy the Sparta test app folder (this was referenced in my `COPY app/ .` command) as I already had this in the tar file, so I had to edit my Compose file

What I learnt from the project

- How Docker containers and images work
- That containers don't always come with the packages I expect (e.g. `sudo` and `nano`)
- The `docker compose up` and `docker compose down` commands work similarly to `terraform apply` and `terraform destroy` in that they build containers (and default networks) from a Docker compose file and then stop and remove everything that the `docker compose up` built

- About saving images to tar files and how this is helpful when transferring images to a system without internet access, sharing images outside of a registry, or for backing up
- That, on Linux OSes, you have to add yourself to the Docker group in order to avoid having to preface every `docker` command with `sudo`

Benefits I personally saw from the project

- After the learning curve, using Docker seems like a much easier process than the previous methods for deployment that we used
- On the Sparta container task, I preferred my first automatic method command as it was simpler, so I used this as the basis for the EC2 extension task