CSE3904-01-293 ASSESSMENT 4

Table of Contents

0.0 Overview

Task 1 is about running the docker containers separately. Our group have successfully completed the task.
Task 2 is about running the docker container together using docker compose. Our group have successfully completed the task.
Task 3 is about Kubernetes and running the containers through clusters. Our group have successfully created the cluster and we have got the frontend to run with https on port 8443.

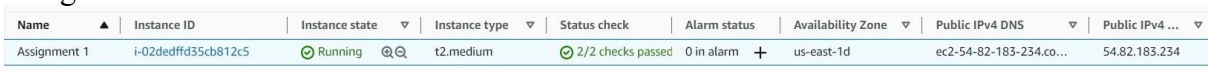
We tried to get the mongo express on https in the nginx.conf file, but there is a problem of css and it does not appear properly, so using http for mongoexpress.

We have used the same instance for all 3 tasks.

1.0 Task 1

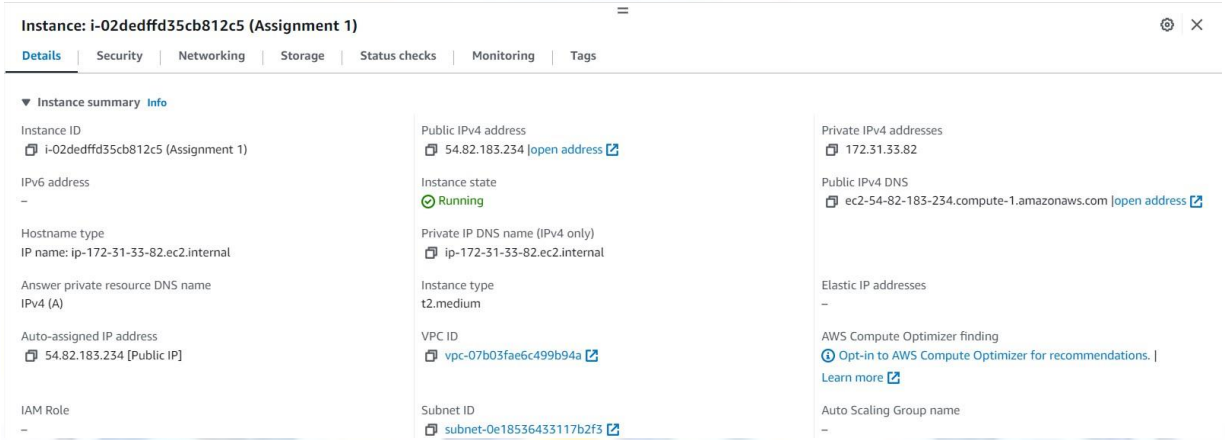
Some details about the instance:

Assignment 1 instance created:



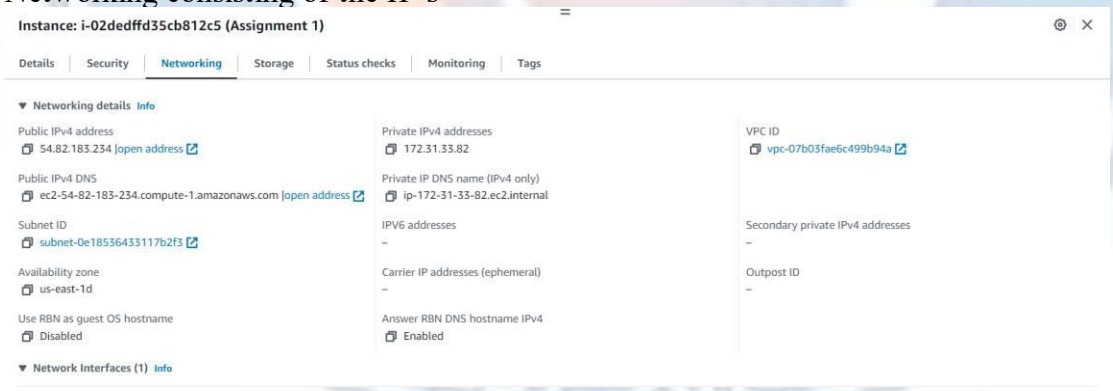
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
Assignment 1	i-02dedffd35cb812c5	Running	t2.medium	2/2 checks passed	0 in alarm	us-east-1d	ec2-54-82-183-234.co...	54.82.183.234

Details of the instance:

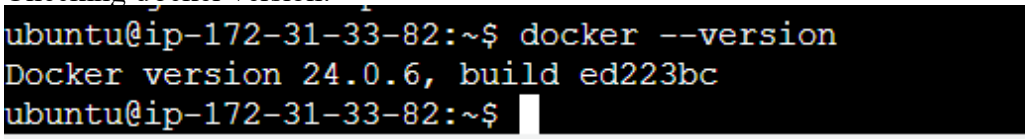


Amazon Machine Image: Ubuntu Server 18.04 LTS
Instance type: t2.medium
Storage: 30 G

Networking consisting of the IP's



Checking docker version:

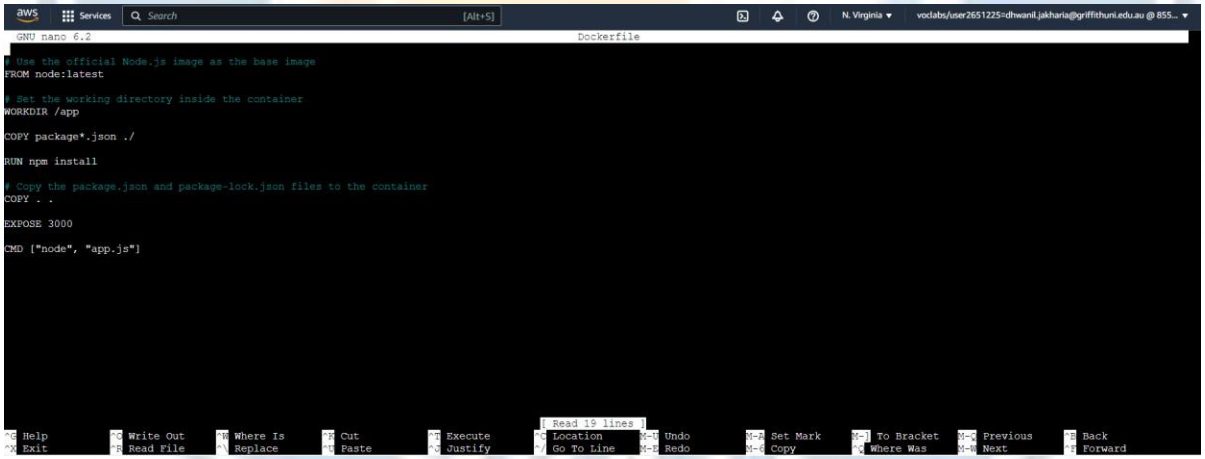


Downloading and unzipping the file:
$ wget -O assignment.zip http://formal-analysis.com/tmp/nodejs-shop.zip
$ unzip assignment.zip

```
ubuntu@ip-172-31-33-82:~$ ls -la
total 248
-rw-rw-r--  1 ubuntu ubuntu      0 Sep 13 23:24 -v
drwxr-x---  9 ubuntu ubuntu   4096 Sep 14 03:03 .
drwxr-xr-x  3 root   root     4096 Sep 11 08:12 ..
-rw-------  1 ubuntu ubuntu  25983 Sep 14 02:55 .bash_history
-rw-r--r--  1 ubuntu ubuntu    220 Jan  6  2022 .bash_logout
-rw-r--r--  1 ubuntu ubuntu   3771 Jan  6  2022 .bashrc
drwx------  2 ubuntu ubuntu   4096 Sep 11 08:12 .cache
drwx------  3 ubuntu ubuntu   4096 Sep 11 08:20 .docker
drwxrwxr-x  3 ubuntu ubuntu   4096 Sep 11 08:17 .local
-rw-r--r--  1 ubuntu ubuntu    807 Jan  6  2022 .profile
drwx------  2 ubuntu ubuntu   4096 Sep 11 08:12 .ssh
-rw-r--r--  1 ubuntu ubuntu      0 Sep 11 08:13 .sudo_as_admin_successful
drwx------  2 ubuntu ubuntu   4096 Sep 13 11:30 .w3m
drwxrwxr-x  3 ubuntu ubuntu   4096 Sep 11 08:16 __MACOSX
-rw-rw-r--  1 ubuntu ubuntu 169507 Sep 12 02:44 assignment.zip
drwxrwxr-x 11 ubuntu ubuntu   4096 Sep 14 02:55 nodejs-shop-assignment1
ubuntu@ip-172-31-33-82:~$
```

1.1 Customize Images

Content of Dockerfile for frontend:



FROM node:latest: Specifies the base image to build upon, in this case, the latest Node.js image.

WORKDIR /app: Sets the working directory inside the container to /app, where subsequent commands will be executed.

COPY package*.json ./ :Copies all the json files with name starting with package i.e., package.json and package-lock.json into the working directory.

RUN npm install: Installs Node.js application dependencies during the image building process.

COPY . . : Copies the application files from the host machine into the container's current working directory.

EXPOSE 3000: Informs Docker that the container will listen on port 3000 when it runs.

CMD ["node", "app.js"]: Defines the default command to execute when the container starts, running the Node.js application specified in app.js.

Building the frontend-image using docker build command:

$ docker build -t frontend-image

It was completed successfully with no errors.

Pulling the mongo image from library:

$ docker pull mongo

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
Digest: sha256:31bf43c4959c283733a004b0a3a9c4ddc52efafb4cf9a38e42d2da93e9a72546
Status: Image is up to date for mongo:latest
docker.io/library/mongo:latest
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Pulling the mongo-express image from library:

$ docker pull mongo-expres

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker pull mongo-express
Using default tag: latest
latest: Pulling from library/mongo-express
Digest: sha256:dcfcf89bf91238ff129469a5a94523b3025913dcc41597d72d4d5f4a0339cc7d
Status: Image is up to date for mongo-express:latest
docker.io/library/mongo-express:latest
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Pulling the nginx image from library:

$ docker pull nginx

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:6926dd802f40e5e7257fded83e0d8030039642e4e10c4a98a6478e9c6fe06153
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

1.2 Run Individual Containers

Created network shop

$ docker network create shop

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker network create shop
a4b06d45ebdf21a5db1942fd0ffdf50d050276047f504d16ccbec75d1a0108a7
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

i-02dedffd35cb812c5 (Assignment 1)

PublicIPs: 54.82.183.234 PrivateIPs: 172.31.33.82

Running mongo using docker run command:

$ docker run -d -p 27017:27017 --name mongo --network shop -v /data/db mongo

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker run -d -p 27017:27017 --name mongo --network shop -v /data/db  mongo
f619d8649482d2057b40d5fb163b60fff4bfbd827e458cabalec70af3aee7a3
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                           NAMES
f619d8649482   mongo     "docker-entrypoint.s…"   3 seconds ago   Up 2 seconds   0.0.0.0:27017->27017/tcp, :::27017->27017/tcp   mongo
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Running frontend using docker run command:

$ docker run -d -p 3000:3000 --name frontend-container --network shop -e MONGODB_URI=mongo://username:password@mongodb:27017/mydatabasse -e SECRET=your_secret_key frontend-image

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker run -d -p 3000:3000 --name frontend-container --network shop  -e MONGODB_URI=mongo://username:password@mongodb:27017/mydatabase -e SE
CRET=your_secret_key frontend-image
fe86466c6aebb894fc3b172da8d29e54fbcf064895e0ac69e6501716b1ffb378
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                           NAMES
fe86466c6aeb   frontend-image  "docker-entrypoint.s…"   13 seconds ago   Up 12 seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp       frontend-container
f619d8649482   mongo           "docker-entrypoint.s…"   45 seconds ago   Up 44 seconds   0.0.0.0:27017->27017/tcp, :::27017->27017/tcp   mongo
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Running mongo express using docker run command:

$ docker run -d -p 8081:8081 --name mongo-express-container --network shop --link mongodb:mongo mongo-express

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker run -d -p 8081:8081 --name mongo-express-container --network shop --link mongo:mongo mongo-express
9cb77e0db8e1b6553f9a2353359a50c48be0846e1bc5bdd3debe533d3e02e216
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                           NAMES
9cb77e0db8e1   mongo-express   "tini -- /docker-ent…"   4 seconds ago    Up 3 seconds    0.0.0.0:8081->8081/tcp, :::8081->8081/tcp       mongo-express-container
fe86466c6aeb   frontend-image  "docker-entrypoint.s…"   4 minutes ago    Up 4 minutes    0.0.0.0:3000->3000/tcp, :::3000->3000/tcp       frontend-container
f619d8649482   mongo           "docker-entrypoint.s…"   5 minutes ago    Up 5 minutes    0.0.0.0:27017->27017/tcp, :::27017->27017/tcp   mongo
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Contents of nginx.conf file

```
worker_processes 1;
events {
    worker_connections 1024;
}

http {
    sendfile on;
    large_client_header_buffers 4 32k;

    upstream nodejs-shop {
        server frontend-container:3000; # Replace with your Node.js application's IP and port
    }

    upstream mongoexpress-server  {
        server mongo-express-container:8081;
    }
```

```
    server {
        listen 80;
        server_name localhost; # Replace with your server's public IP address

        location / {
            return 301 https://$host$request_uri;
        }
    }

    server {
        listen 443 ssl;
        server_name localhost; # Replace with your server's public IP address

        ssl_certificate /etc/ssl/certs/localhost.crt;  # Update with your SSL certificate path
        ssl_certificate_key /etc/ssl/private/localhost.key;  # Update with your SSL certificate key
path

        location / {
            proxy_pass http://nodejs-shop;
            proxy_redirect off;
            proxy_http_version 1.1;
            proxy_cache_bypass $http_upgrade;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection keep-alive;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header X-Forwarded-Host $server_name;
            proxy_buffer_size 128k;
            proxy_buffers 4 256k;
            proxy_busy_buffers_size 256k;
        }
        location /mongoexpress/  {
            proxy_pass http://mongoexpress-server/;
            proxy_redirect off;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection keep-alive;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header X-Forwarded-Host $server_name;
            proxy_buffer_size 128k;
            proxy_buffers 4 256k;
            proxy_busy_buffers_size 256k;
        }
    }
}
```

Generate SSL certificate using private key and signature for HTTPs:

Generate the root CA private key (key) and privacy enhanced mail (pem) file.

$ openssl req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048 -keyout RootCA.key -out RootCA.pem -subj "/C=US/CN=My-Root-CA"

Run the following to generate the root certificate (crt) from the pem file.

$ openssl x509 -outform pem -in RootCA.pem -out RootCA.crt

Create a file domains.ext that lists all your local domains:

$ nano domains.ext

authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
DNS.2 = fake1.local

Generate Keys:

$ openssl req -new -nodes -newkey rsa:2048 -keyout localhost.key -out localhost.csr -subj "/C=US/ST=YourState/L=YourCity/O=Example-Certificates/CN=localhost.local"
Return Certificate

$ openssl x509 -req -sha256 -days 1024 -in localhost.csr -CA RootCA.pem -CAkey RootCA.key -CAcreateserial -extfile domains.ext -out localhost.crt

Move files
$ mkdir ssl
$ mv localhost.* ssl/
$ mv RootCA.* ssl/

Running nginx using docker run command:
$ docker run -d --network shop --name nginx-proxy -p 80:80 -p 443:443 \
-v /home/ubuntu/nodejs-shop-assignment1/nginx.conf:/etc/nginx/nginx.conf:ro \
-v /home/ubuntu/nodejs-shop-assignment1/ssl/localhost.crt:/etc/ssl/certs/localhost.crt:ro \
-v /home/ubuntu/nodejs-shop-assignment1/ssl/localhost.key:/etc/ssl/private/localhost.key:ro \
nginx

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker run -d --network shop --name nginx-proxy -p 80:80 -p 443:443 \
    -v /home/ubuntu/nodejs-shop-assignment1/nginx.conf:/etc/nginx/nginx.conf:ro \
    -v /home/ubuntu/nodejs-shop-assignment1/ssl/localhost.crt:/etc/ssl/certs/localhost.crt:ro \
    -v /home/ubuntu/nodejs-shop-assignment1/ssl/localhost.key:/etc/ssl/private/localhost.key:ro \
    nginx
493b4030e54c0efc996a2cbac59cf7abb2afc98a69268dbb90c99ad20b80c26d
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                                                                  NAMES
493b4030e54c   nginx           "/docker-entrypoint.…"   3 seconds ago    Up 2 seconds    0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp               nginx-proxy
9cb77e0db8e1   mongo-express   "tini -- /docker-ent…"   5 minutes ago    Up 5 minutes    0.0.0.0:8081->8081/tcp, :::8081->8081/tcp                                              mongo-express-container
fe86466c6aeb   frontend-image  "docker-entrypoint.s…"   10 minutes ago   Up 10 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp                                              frontend-container
f619d8649482   mongo           "docker-entrypoint.s…"   10 minutes ago   Up 10 minutes   0.0.0.0:27017->27017/tcp, :::27017->27017/tcp                                          mongo
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Logs:

Frontend:

$ docker logs frontend-container

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker logs frontend-container
(node:1) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:1) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'Schema' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Mongo:

$ docker logs mongo

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker logs mongo
{"t":{"$date":"2023-09-14T03:10:27.915+00:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"main","msg":"Initialized wire specification","attr":{"spec":{"incomingExternalClient":{"minWireV
ersion":0,"maxWireVersion":21},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":21},"outgoing":{"minWireVersion":6,"maxWireVersion":21},"isInternalClient":true}}}
{"t":{"$date":"2023-09-14T03:10:27.920+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProto
cols 'none'"}
{"t":{"$date":"2023-09-14T03:10:27.923+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServ
er, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2023-09-14T03:10:27.933+00:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"main","msg":"Successfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonor
Service","namespace":"config.tenantMigrationDonors"}}
{"t":{"$date":"2023-09-14T03:10:27.933+00:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"main","msg":"Successfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecip
ientService","namespace":"config.tenantMigrationRecipients"}}
{"t":{"$date":"2023-09-14T03:10:27.933+00:00"},"s":"I",  "c":"CONTROL",  "id":5945603, "ctx":"main","msg":"Multi threading initialized"}
{"t":{"$date":"2023-09-14T03:10:27.933+00:00"},"s":"I",  "c":"TENANT_M", "id":7091600, "ctx":"main","msg":"Starting TenantMigrationAccessBlockerRegistry"}
{"t":{"$date":"2023-09-14T03:10:27.934+00:00"},"s":"I",  "c":"CONTROL",  "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","archit
ecture":"64-bit","host":"40aa8df570c5"}}
{"t":{"$date":"2023-09-14T03:10:27.934+00:00"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"7.0.1","gitVersion":"425a0454d
12f2664f9e31002bbe4a386a25345b5","openSSLVersion":"OpenSSL 3.0.2 15 Mar 2022","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu2204","distarch":"x86_64","target_arch":"x86
_64"}}}}
{"t":{"$date":"2023-09-14T03:10:27.934+00:00"},"s":"I",  "c":"CONTROL",  "id":51765,   "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"22.04"}}}
{"t":{"$date":"2023-09-14T03:10:27.934+00:00"},"s":"I",  "c":"CONTROL",  "id":21951,   "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{"net":{"bindIp":"*"}}}}
{"t":{"$date":"2023-09-14T03:10:27.936+00:00"},"s":"I",  "c":"STORAGE",  "id":22297,   "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger stora
ge engine. See http://dochub.mongodb.org/core/prodnotes-filesystem","tags":["startupWarnings"]}
{"t":{"$date":"2023-09-14T03:10:27.936+00:00"},"s":"I",  "c":"STORAGE",  "id":22315,   "ctx":"initandlisten","msg":"Opening WiredTiger","attr":{"config":"create,cache_size=256M,session_max=3
3000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,remove=true,path=journal,compressor=snappy),builtin_extension_config=(zstd=(compression_leve
l=6)),file_manager=(close_idle_time=600,close_scan_interval=10,close_handle_minimum=2000),statistics_log=(wait=0),json_output=(error,message),verbose=[recovery_progress:1,checkpoint_progress
:1,compact_progress:1,backup:0,checkpoint:0,compact:0,evict:0,history_store:0,recovery:0,rts:0,salvage:0,tiered:0,timestamp:0,transaction:0,verify:0,log:0],"}}
{"t":{"$date":"2023-09-14T03:10:28.755+00:00"},"s":"I",  "c":"STORAGE",  "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":{"durationMillis":818}}
{"t":{"$date":"2023-09-14T03:10:28.755+00:00"},"s":"I",  "c":"RECOVERY", "id":23987,   "ctx":"initandlisten","msg":"WiredTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$timestamp":{"
t":0,"i":0}}}}
{"t":{"$date":"2023-09-14T03:10:28.813+00:00"},"s":"W",  "c":"CONTROL",  "id":22120,   "ctx":"initandlisten","msg":"Access control is not enabled for the database. Read and write access to d
ata and configuration is unrestricted","tags":["startupWarnings"]}
{"t":{"$date":"2023-09-14T03:10:28.814+00:00"},"s":"W",  "c":"CONTROL",  "id":5123300, "ctx":"initandlisten","msg":"vm.max_map_count is too low","attr":{"currentValue":65530,"recommendedMini
mum":1677720,"maxConns":838860},"tags":["startupWarnings"]}
{"t":{"$date":"2023-09-14T03:10:28.815+00:00"},"s":"I",  "c":"STORAGE",  "id":20320,   "ctx":"initandlisten","msg":"createCollection","attr":{"namespace":"admin.system.version","uuidDisposit
```

Mongo Express:

$ docker logs mongo-express-container

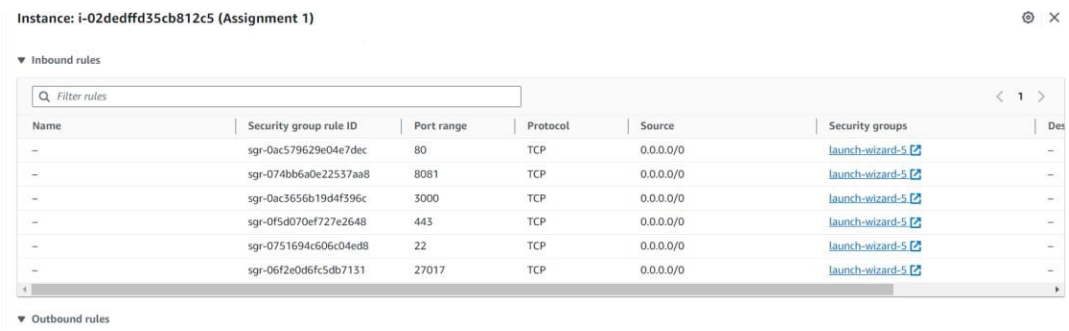
```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker logs mongo-express-container
Welcome to mongo-express
------------------------


(node:6) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine
, pass option { useUnifiedTopology: true } to the MongoClient constructor.
Mongo Express server listening at http://0.0.0.0:8081
Server is open to allow connections from anyone (0.0.0.0)
basicAuth credentials are "admin:pass", it is recommended you change this in your config.js!
GET / 200 41.536 ms - 7956
GET /public/css/bootstrap.min.css 200 11.374 ms - 121457
GET /public/css/bootstrap-theme.min.css 200 11.887 ms - 23411
GET /public/css/style.css 200 3.873 ms - 1883
GET /public/index-6145173d12f8f196322e.min.js 200 4.714 ms - 965
GET /public/vendor-d1b820f8a9cf3d5a8c6a.min.js 200 7.197 ms - 130761
GET /public/img/mongo-express-logo.png 200 6.202 ms - 17847
GET /public/img/gears.gif 200 5.550 ms - 50281
GET /public/fonts/glyphicons-halflings-regular.woff2 200 8.468 ms - 18028
GET / 304 20.352 ms - -
GET /public/css/bootstrap.min.css 304 1.854 ms - -
GET /public/css/bootstrap-theme.min.css 304 2.018 ms - -
GET /public/css/style.css 304 2.189 ms - -
GET /public/vendor-d1b820f8a9cf3d5a8c6a.min.js 304 0.534 ms - -
GET /public/index-6145173d12f8f196322e.min.js 304 0.429 ms - -
GET /public/img/mongo-express-logo.png 304 0.537 ms - -
GET /public/img/gears.gif 304 0.350 ms - -
GET /public/fonts/glyphicons-halflings-regular.woff2 304 0.424 ms - -
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Nginx:

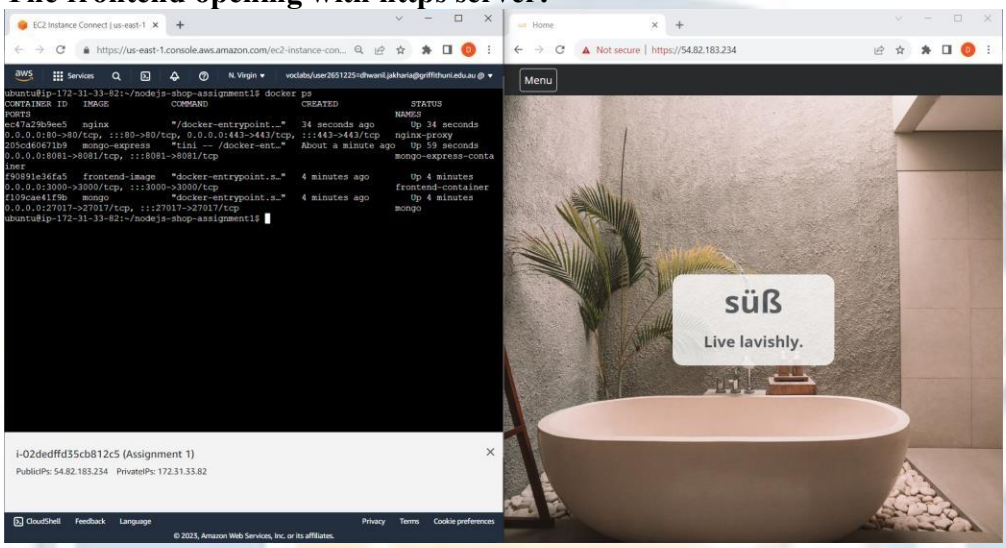
$ docker logs nginx-proxy

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ docker logs nginx-proxy
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
110.145.166.154 - - [14/Sep/2023:03:12:43 +0000] "GET / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36
"
110.145.166.154 - - [14/Sep/2023:03:13:07 +0000] "GET / HTTP/1.1" 200 563 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36
"
110.145.166.154 - - [14/Sep/2023:03:13:07 +0000] "GET /css/homepage.css HTTP/1.1" 200 773 "https://18.234.151.190/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/116.0.0.0 Safari/537.36"
110.145.166.154 - - [14/Sep/2023:03:13:07 +0000] "GET /css/main.css HTTP/1.1" 200 1778 "https://18.234.151.190/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/116.0.0.0 Safari/537.36"
110.145.166.154 - - [14/Sep/2023:03:13:07 +0000] "GET /js/main.js HTTP/1.1" 200 519 "https://18.234.151.190/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko
) Chrome/116.0.0.0 Safari/537.36"
110.145.166.154 - - [14/Sep/2023:03:13:09 +0000] "GET /favicon.ico HTTP/1.1" 200 1847 "https://18.234.151.190/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
ko) Chrome/116.0.0.0 Safari/537.36"
```

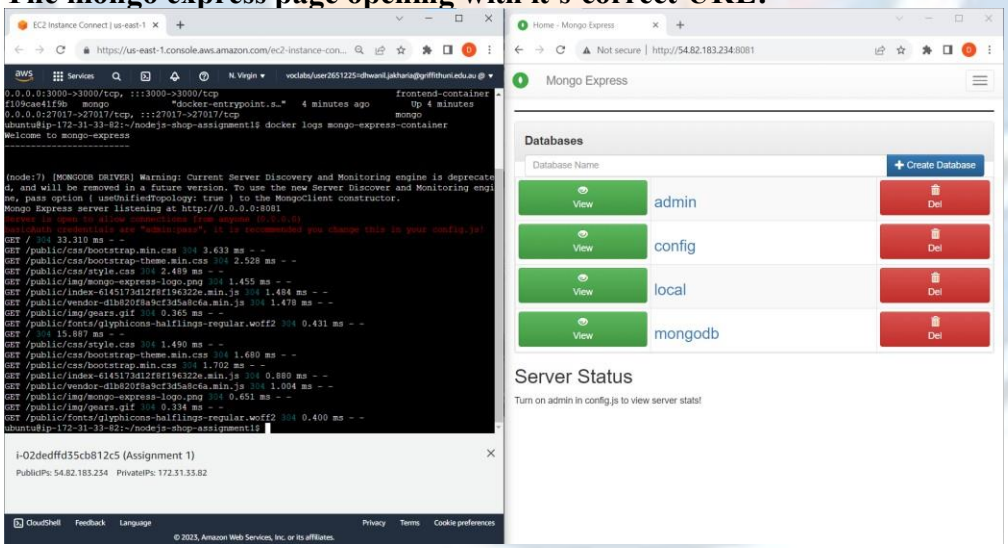
Changing Inbound Rules:



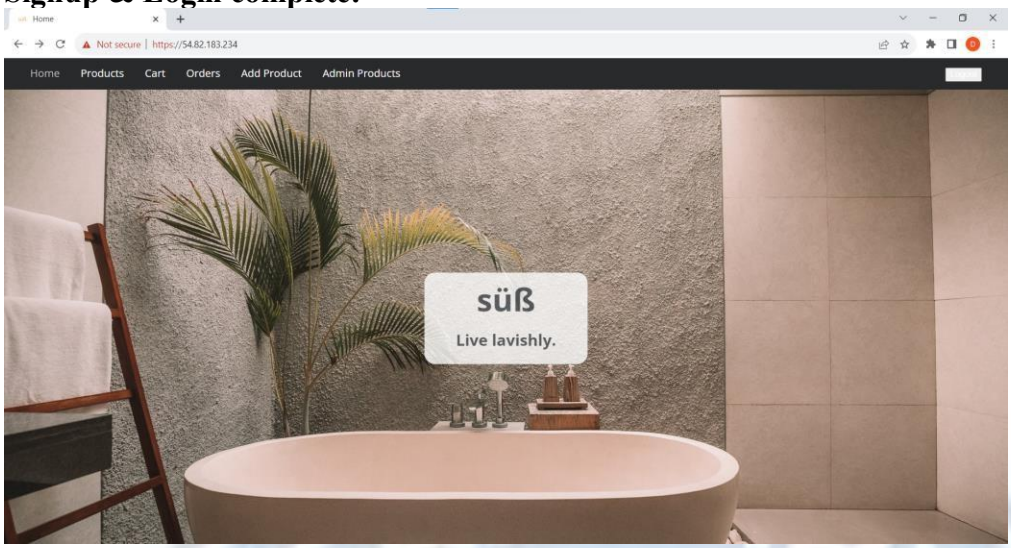
The frontend opening with https server:



The mongo express page opening with it's correct URL:



Signup & Login complete:



2.0 Task 2

2.1 Docker Compose

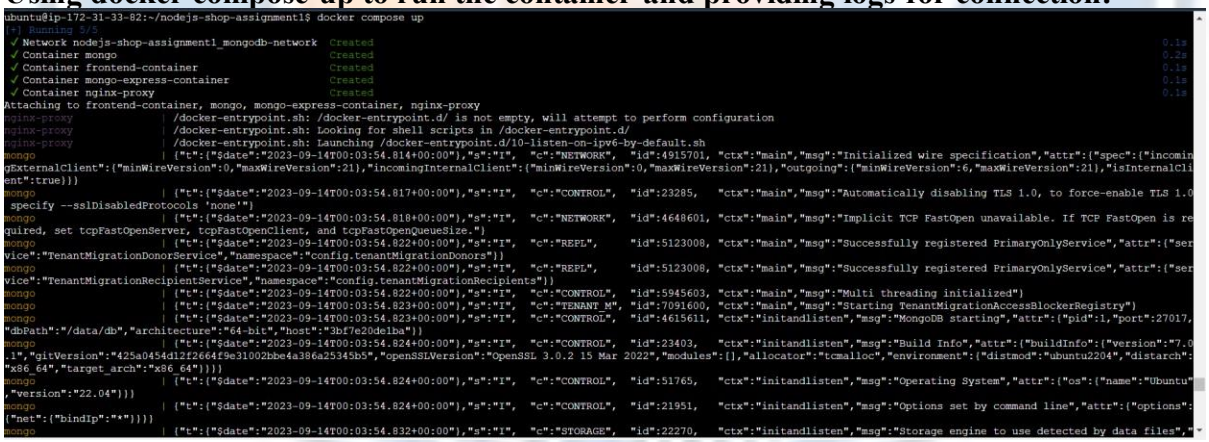
Content of Docker-compose.yml

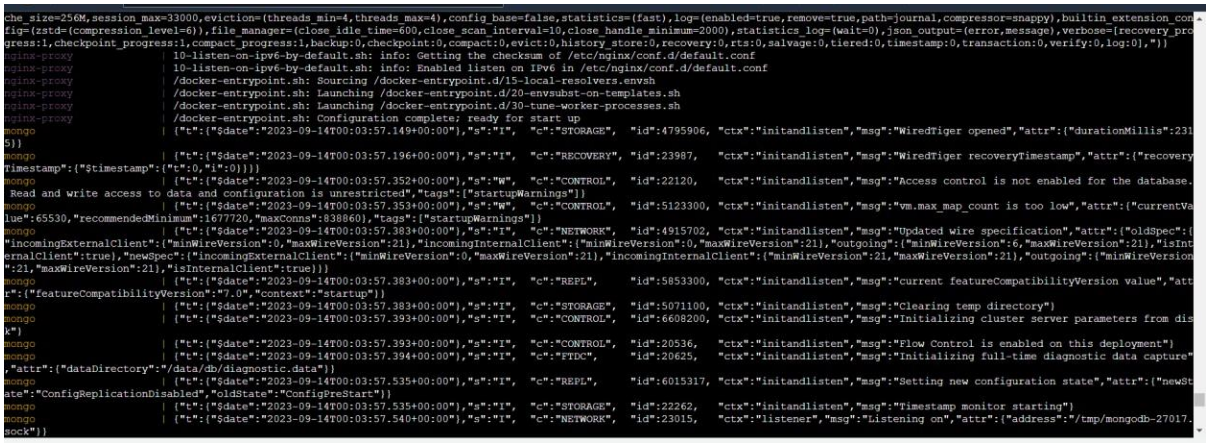
```
version: '3.9'
services:                                          //defining services
  frontend:                                        //defining frontend service
    container_name: frontend-container             //name of the container
    image: frontend-image:latest                   //image used to run the container (built in task 1)
    ports:
      - "3000:3000"                                //using ports 3000 for frontend
    environment:                                   //defining environments URI, port and secret
      MONGODB_URI: 'mongodb://mongo:27017'
      PORT: 3000
      SECRET: 'abcd'
    depends_on:
      - mongo                                      // depends on mongo service
    networks:                                      //defining networks used in this service
      - frontend
      - backend

  mongo:                                           //defining mongo service
    container_name: mongo                          //name of the container
    image: mongo                                   //image used to run (from library)
    volumes:                                       // defining database
      - mongodb_data:/data/db
    ports:
      - "27017:27017"                              //using ports 27017
    networks:                                      //defining networks used in this service
      - backend

  mongoexpress:                                    // defining mongoexpress service
    container_name: mongo-express-container        //name of the container
    image: mongo-express                           // image used to run (from library)
    ports:
      - "8081:8081"                                //using port 8081
    environment:
      ME_CONFIG_MONGODB_PORT:27017
      ME_CONFIG_MONGODB_SERVER: 'mongo'
    depends_on:                                    // depends on mongo service
      - mongo
    networks:                                      //defining networks used in this service
      - frontend
      - backend

  proxy:                                           //defining proxy service
    container_name: nginx-proxy                    //name of the container
```

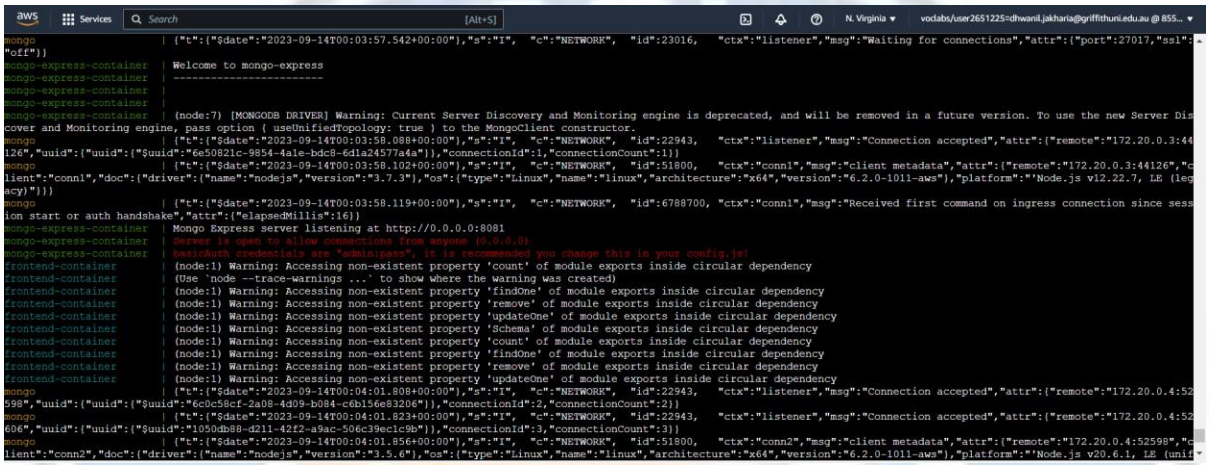
```
    image: nginx                                    // image used to run (from library)
    depends_on:
      - mongo                                       //depends on mongo service
    volumes:                                        //volumes used (nginx.conf, certificate and key)
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl/localhost.crt:/etc/ssl/certs/localhost.crt
      - ./ssl/localhost.key:/etc/ssl/private/localhost.key
    ports:                                          //defining ports for http and https
      - 80:80
      - 443:443
    networks:                                       //defining networks
      - frontend


networks:                                           //defining networks used in all the services
  frontend:
  backend:
    internal: true

volumes:                                            //defining volumes used in all the services
  mongodb_data:
```

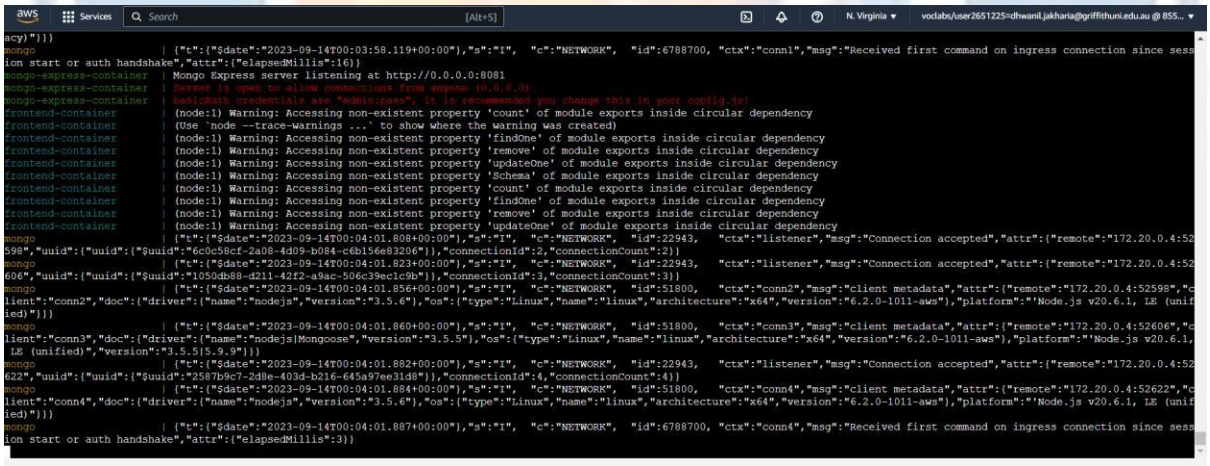
2.2 Run Containers with Docker Compose

Using docker compose up to run the container and providing logs for connection:

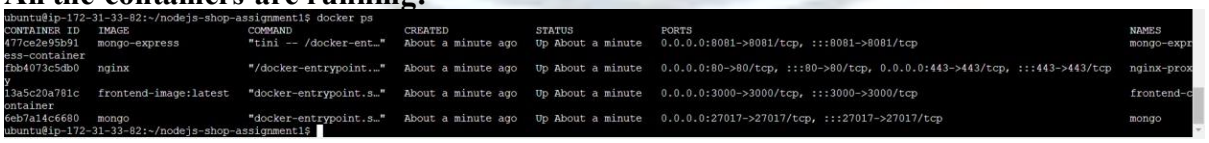




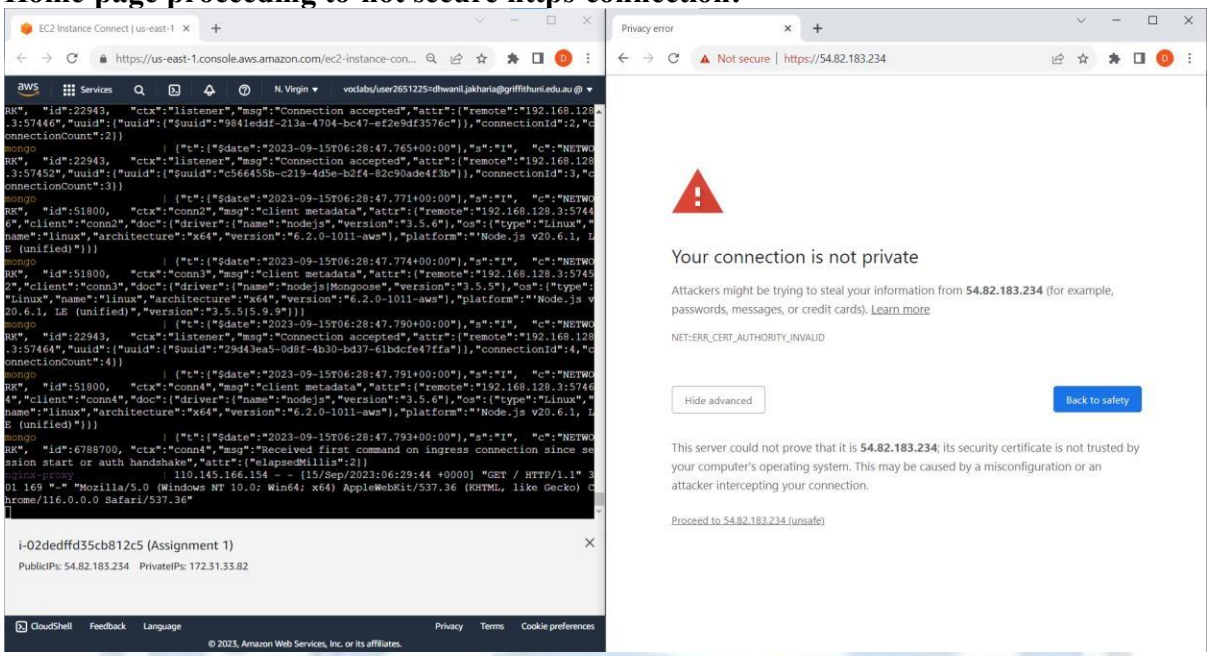




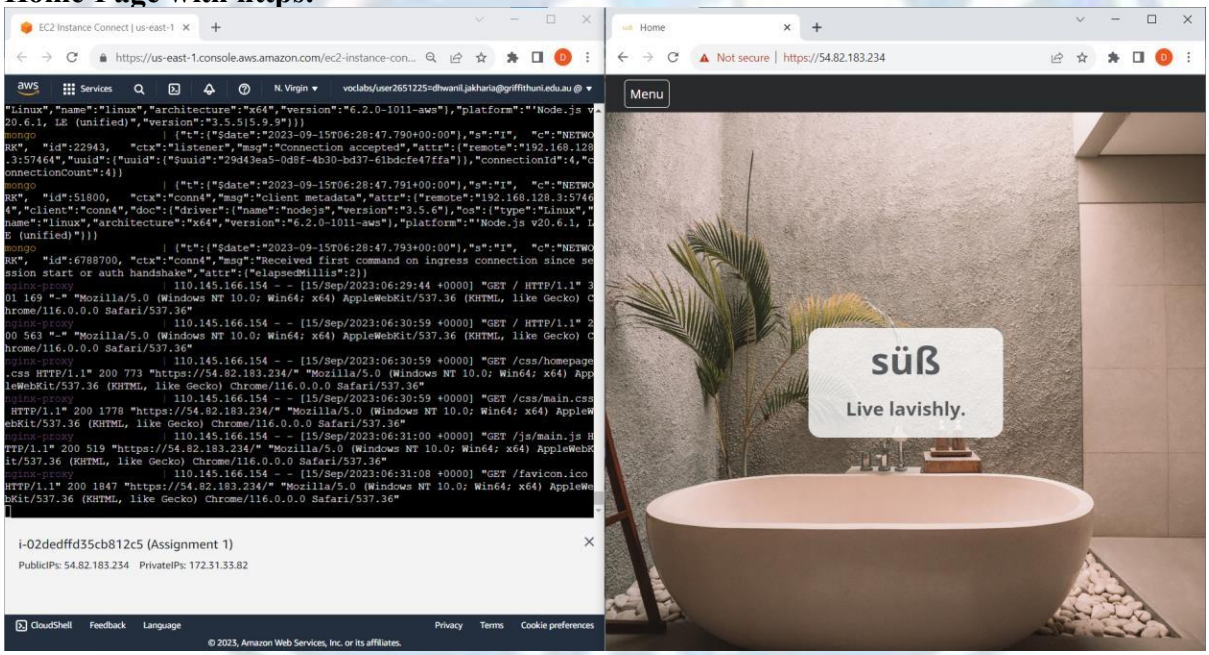
All the containers are running:



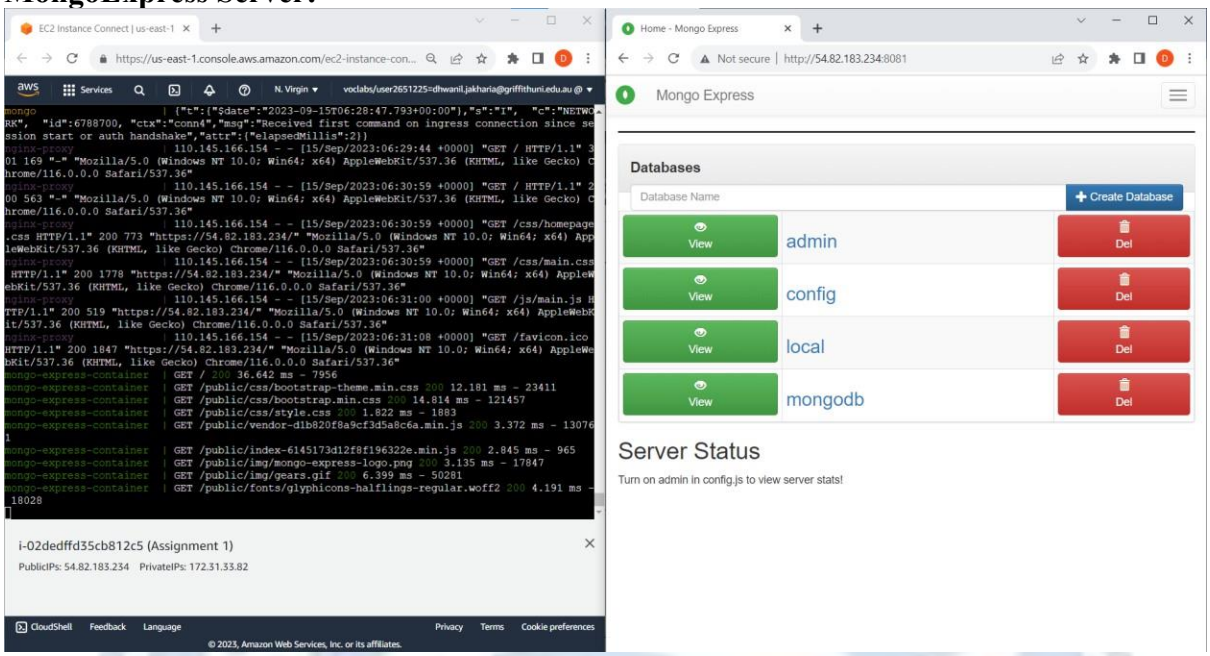
Home page proceeding to not secure https connection:



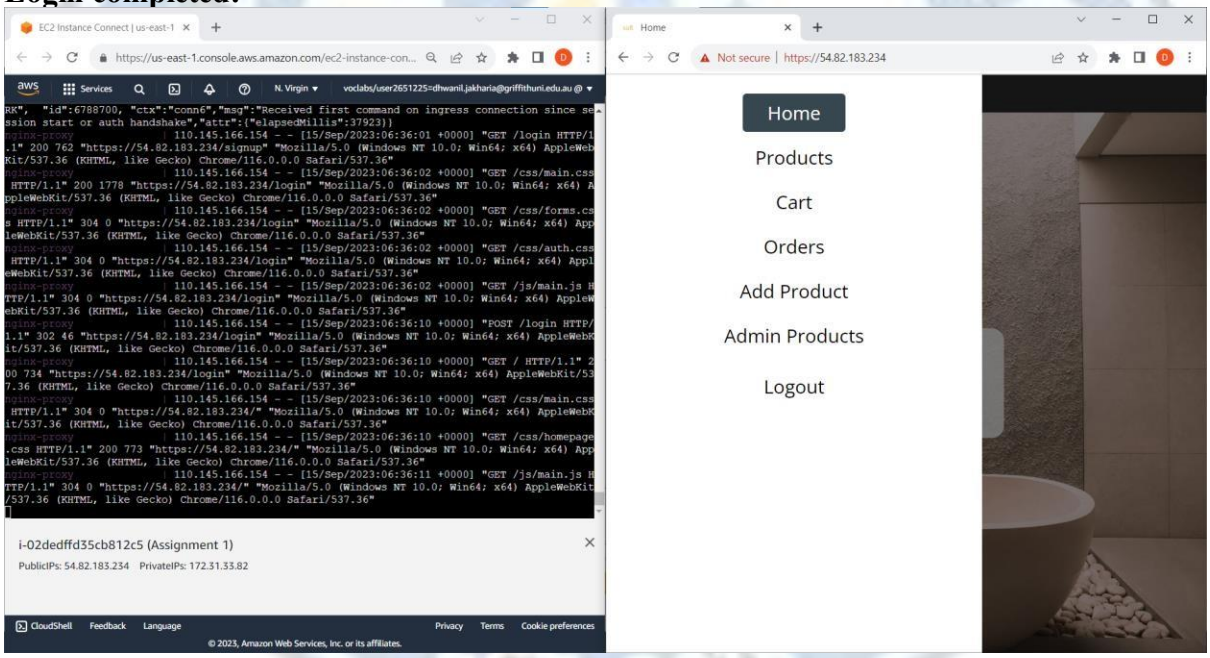
Home Page with https:



MongoExpress Server:



Login completed:



3.0 Task 3

3.1 Create Pods

Using the same Dockerfile to build image as in task 1 inside Kubernetes
Making changes in the nginx.conf file changing the container names to service names:

```
    upstream nodejs-shop {
        server webapp-service:3000; # Replace with your Node.js application's IP and port
    }

    upstream mongoexpress-server {
        server mongoexpress-service:8081;
    }
```

Create pod.yml

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: webapp-pod
  labels:
    app: webapp
spec:
  containers:
    - name: webapp
      imagePullPolicy: Never
      image: frontend-image
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create mongo-deployment.yml

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat mongo-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-deployment
  labels:
    app: mongo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
      - name: mongodb
        image: mongo
        ports:
        - containerPort: 27017
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create mongo-service.yml

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat mongo-service.yml
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  selector:
    app: mongo
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

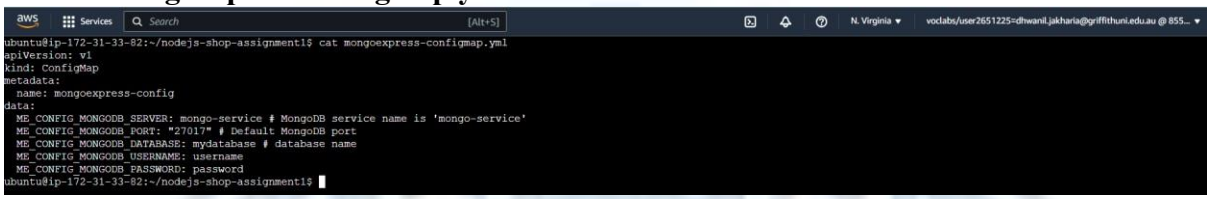
Create webapp-deployment.yml

```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat webapp-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  labels:
    app: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: frontend-image
        imagePullPolicy: Never
        ports:
        - containerPort: 3000
        env:
          - name: MONGODB_URI
            value: "mongodb://username:password@mongo-service:27017/mydatabase"

ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create webapp-service.yml

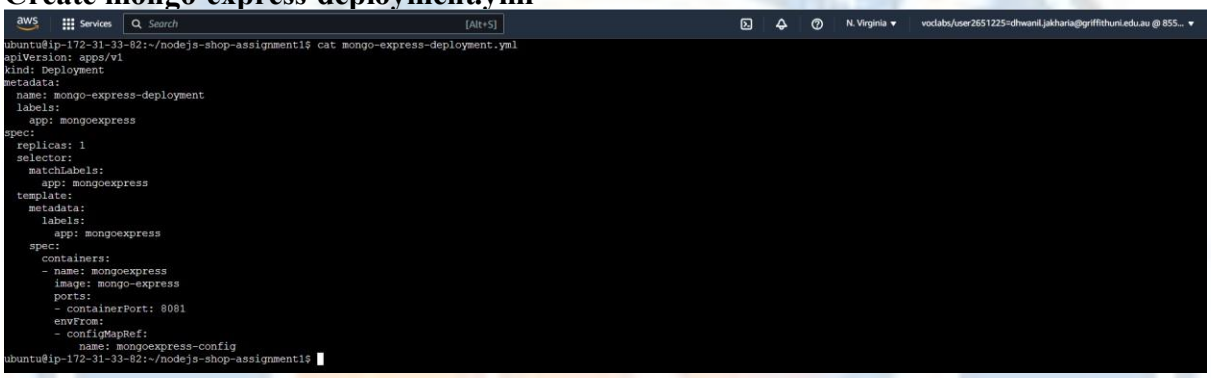
```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat webapp-service.yml
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  type: LoadBalancer
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 32000
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create mongoexpress-configmap.yml



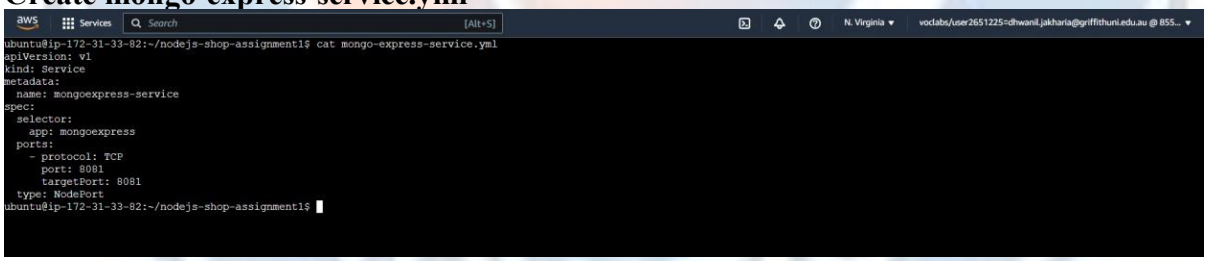
```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat mongoexpress-configmap.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongoexpress-config
data:
  ME_CONFIG_MONGODB_SERVER: mongo-service # MongoDB service name is 'mongo-service'
  ME_CONFIG_MONGODB_PORT: "27017" # Default MongoDB port
  ME_CONFIG_MONGODB_DATABASE: mydatabase # database name
  ME_CONFIG_MONGODB_USERNAME: username
  ME_CONFIG_MONGODB_PASSWORD: password
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create mongo-express-deployment.yml



```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat mongo-express-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express-deployment
  labels:
    app: mongoexpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongoexpress
  template:
    metadata:
      labels:
        app: mongoexpress
    spec:
      containers:
      - name: mongoexpress
        image: mongo-express
        ports:
        - containerPort: 8081
        envFrom:
        - configMapRef:
            name: mongoexpress-config
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create mongo-express-service.yml



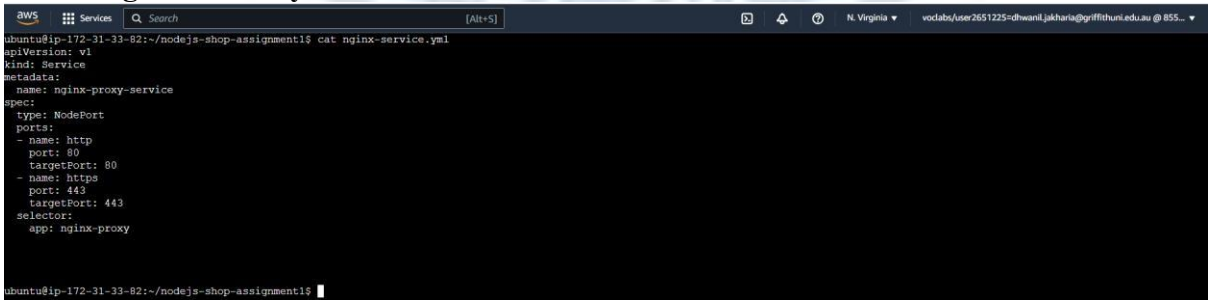
```
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$ cat mongo-express-service.yml
apiVersion: v1
kind: Service
metadata:
  name: mongoexpress-service
spec:
  selector:
    app: mongoexpress
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: NodePort
ubuntu@ip-172-31-33-82:~/nodejs-shop-assignment1$
```

Create nginx-deployment.yml (too big to fit in a screenshot so added text)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy
spec:
  replicas: 1
  selector:
  matchLabels:
      app: nginx-proxy
```

```
  template:
    metadata:
      labels:
        app: nginx-proxy
    spec:
      containers:
      - name:  nginx
        image:  nginx
        ports:
        - containerPort: 80
        - containerPort: 443
        volumeMounts:
        - name: nginx-config-volume
          mountPath: /etc/nginx/nginx.conf
          subPath: nginx.conf
        - name: nginx-certs
          mountPath: /etc/ssl/certs
          readOnly: true
        - name: nginx-keys
          mountPath: /etc/ssl/private
          readOnly: true
      volumes:
      - name: nginx-config-volume
        configMap:
          name: nginx-config
      - name: nginx-certs
        secret:
          secretName: nginx-certs
          items:
          - key: cert
            path: localhost.crt
      - name: nginx-keys
        secret:
          secretName: nginx-certs
          items:
          - key: key
            path: localhost.key
```

Create nginx-service.yml



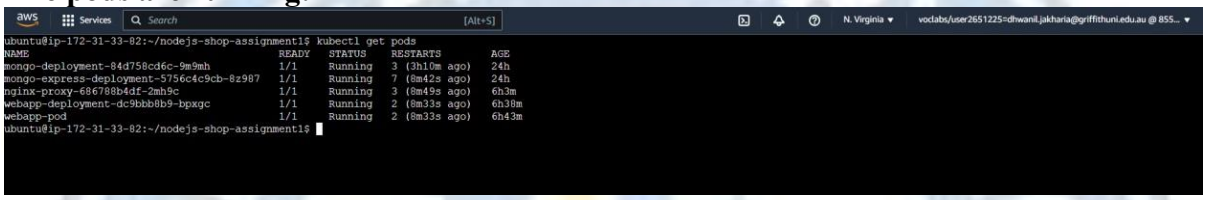
3.2 Run Cluster with Minikube

Running pods, deployment, services, configmap using:
$ kubectl apply -f (filename.yml)

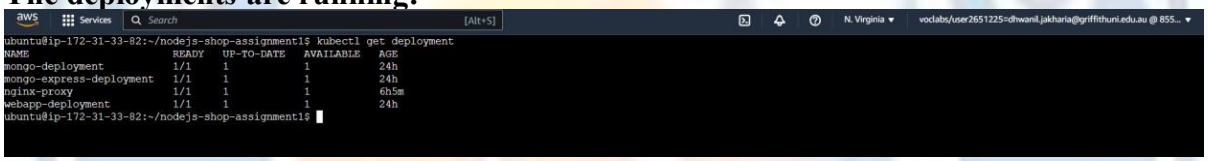
Running nginx.config from nginx.conf using:
$ kubectl create configmap nginx-config --from-file=nginx.conf=./nginx.conf

All the ssl are downloaded from task 1
Creating nginx-certs secret:
$ kubectl create secret generic nginx-certs \
--from-file=cert=./ssl/localhost.crt \
--from-file=key=./ssl/localhost.key

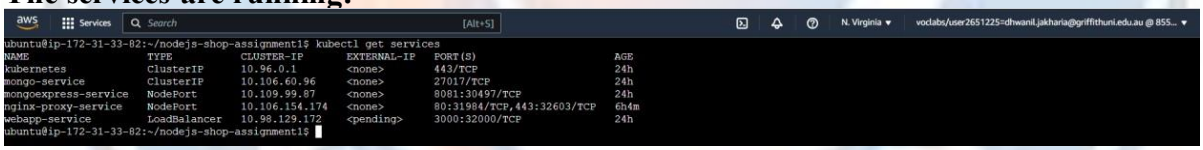
The pods are running:



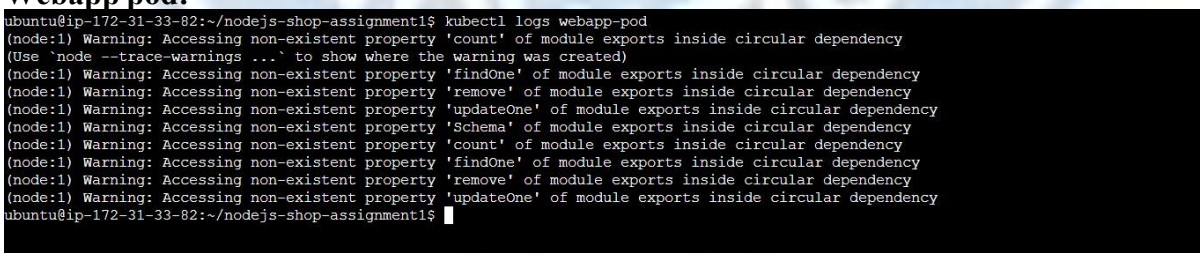
The deployments are running:



The services are running:



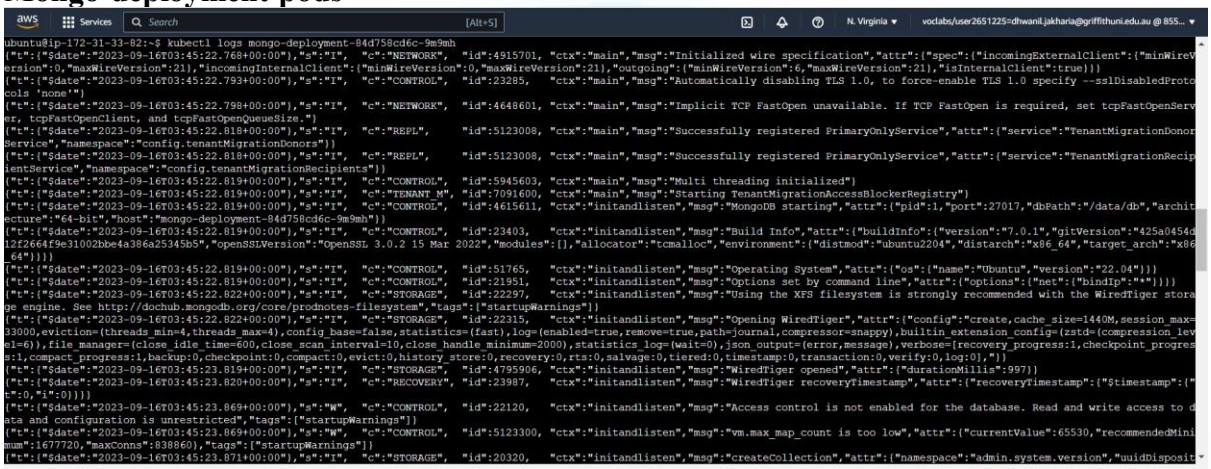
Logs:
Webapp pod:



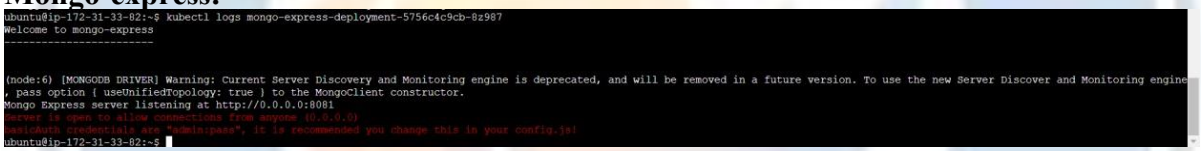
Webapp deployment:

```
ubuntu@ip-172-31-33-82:~$ kubectl logs webapp-deployment-dc9bbb8b9-bpxgc
(node:1) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:1) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'Schema' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:1) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
ubuntu@ip-172-31-33-82:~$
```

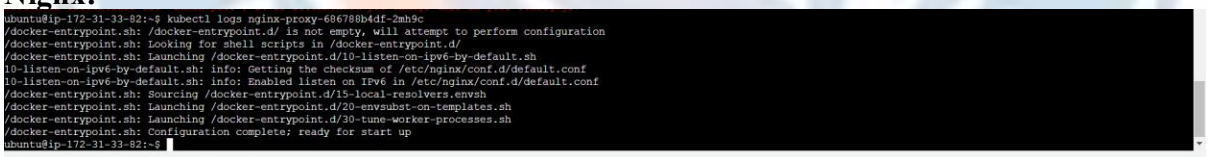
Mongo-deployment pods



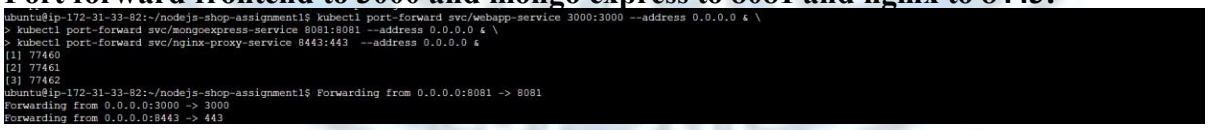
Mongo-express:



Nignx:



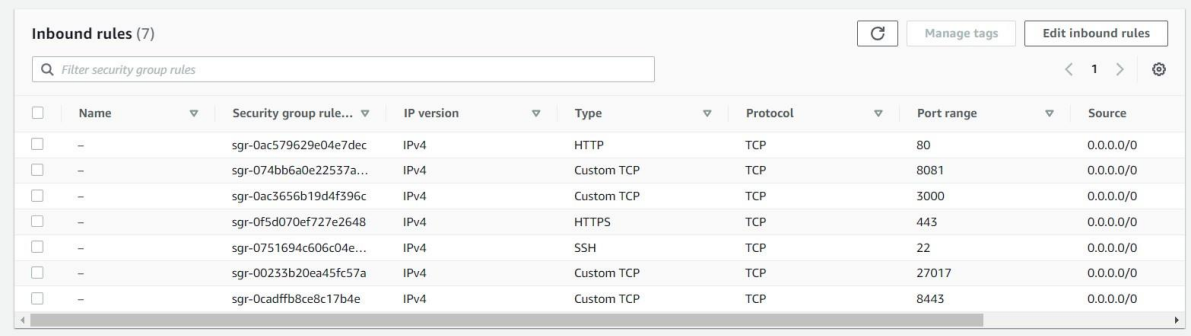
Port forward frontend to 3000 and mongo express to 8081 and nginx to 8443:



We have restarted the aws learner lab so the public IP has now changed but the private IP is the same. We have added security group 8443 to run on https.

We had some error in connecting to https, we had to forward it to a port, which we did not do at first, because of which it was not working, then when we port-forward.

Added port 8443:



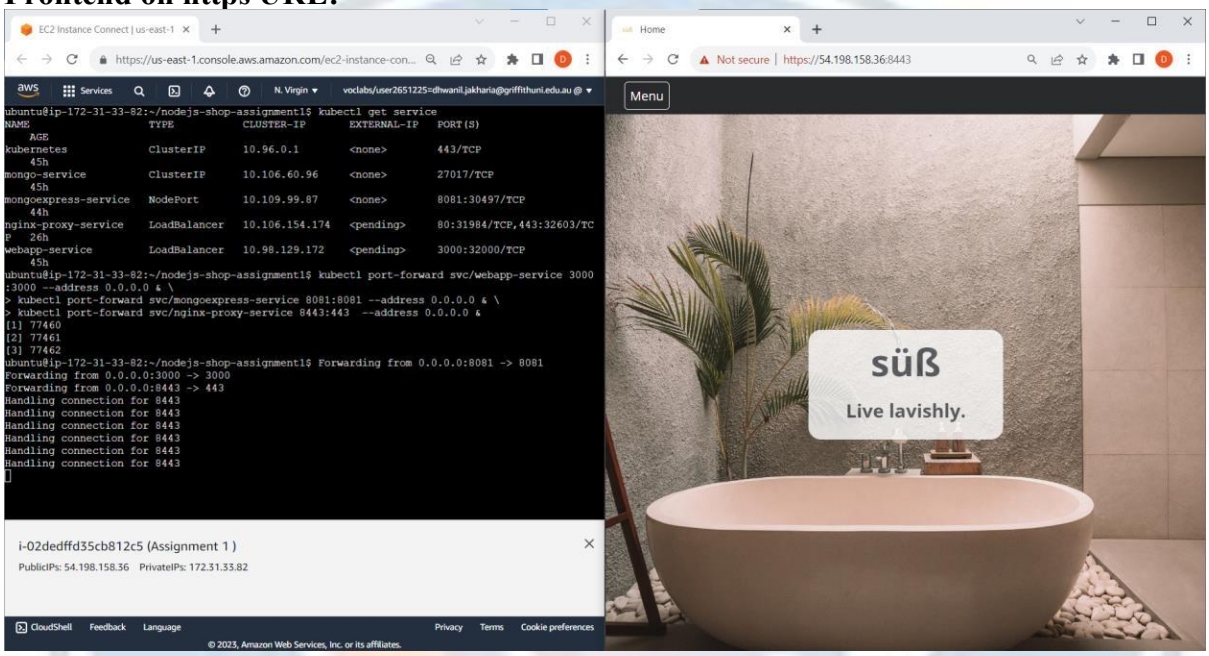
Inbound rules (7)

Manage tags | Edit inbound rules

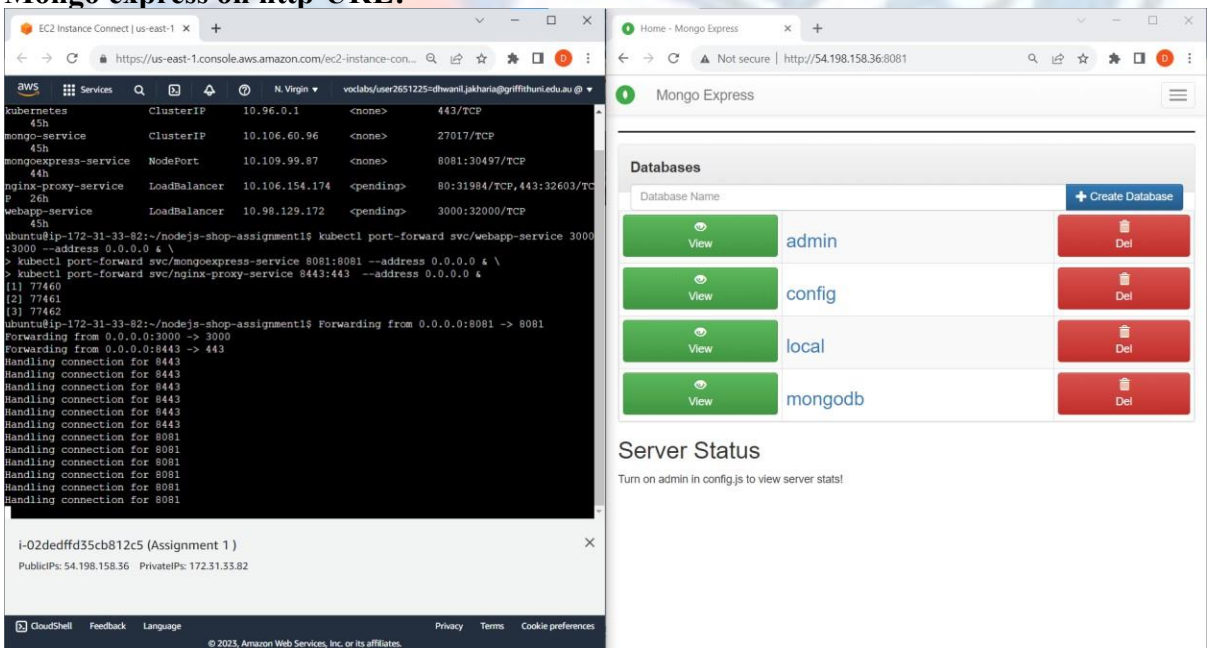
Filter security group rules

Name	Security group rule...	IP version	Type	Protocol	Port range	Source
–	sgr-0ac579629e04e7dec	IPv4	HTTP	TCP	80	0.0.0.0/0
–	sgr-074bb6a0e22537a...	IPv4	Custom TCP	TCP	8081	0.0.0.0/0
–	sgr-0ac3656b19d4f396c	IPv4	Custom TCP	TCP	3000	0.0.0.0/0
–	sgr-0f5d070ef727e2648	IPv4	HTTPS	TCP	443	0.0.0.0/0
–	sgr-0751694c606c04e...	IPv4	SSH	TCP	22	0.0.0.0/0
–	sgr-00233b20ea45fc57a	IPv4	Custom TCP	TCP	27017	0.0.0.0/0
–	sgr-0cadffb8ce8c17b4e	IPv4	Custom TCP	TCP	8443	0.0.0.0/0

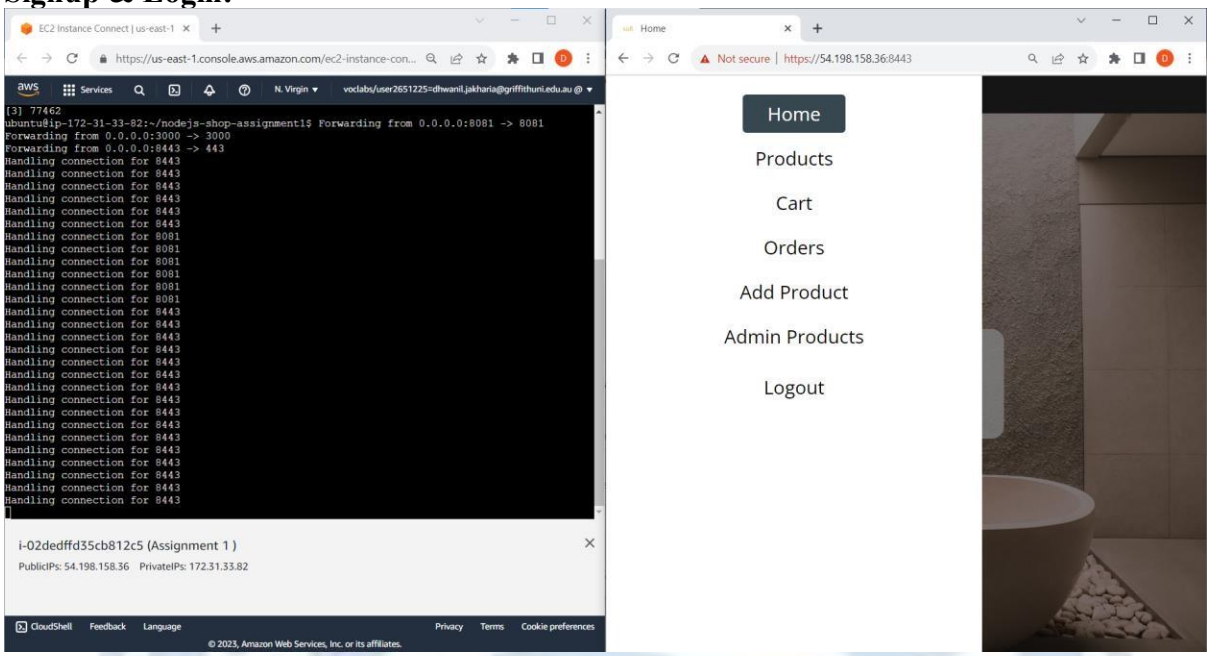
Frontend on https URL:



Mongo express on http URL:



Signup & Login:



4.0 Benefits of Container Technologies

Ans:

Technologies Used:

Undoubtedly, containers along with related technologies have become important in current DevOps practises, offering many advantages over standard approaches to software development and operations. I'll give a brief overview of the technologies utilised in the assignment, explain how they fit into the DevOps workflow, and list their key benefits in this discussion.

Docker: An application and its dependencies can be wrapped into a single, portable container using the popular containerization technology Docker. Applications may be built, deployed, and managed more easily since it provides a standardised format for them.

Kubernetes: An open-source technology called Kubernetes is used to automatically deploy, scale, and manage containerized applications. It offers capabilities for scaling, load balancing, and self-healing container deployment.

Integration into DevOps:

The lifecycle of software development and deployment involves several stages, and container technologies are a critical component of the DevOps toolchain.

Development: Containers are used by developers to establish a reliable and consistent environment for building and testing programmes. This makes ensuring that code functions the same in development and production.

Testing: By making the process of setting up testing environments less complicated, containers make automated testing and integration possible. If there are conflicts between several versions of dependencies, testing can be carried out in separate containers.

Continuous Integration: Pipelines for CI/CD are frequently set up to create container images from code repositories. Following that, these images can be automatically reviewed and delivered to various environments (such as development, staging, and production) based on predetermined triggers.

Deployment: The deployment of containerized apps is orchestrated by Kubernetes, ensuring that they function well, scale as necessary, and recover from errors automatically. High availability and reliability in production contexts depend on this.

Benefits Compared to Traditional Methods:

Portability: Containers make it simple to run the same application on several platforms and environments, which eliminates "it works on my machine" problems. They encapsulate an application and its dependencies.

Isolation: Process and file system isolation given by containers enables different applications function freely on the same host without interfering with one another. It is safer and more stable because of its isolation.

Scalability: Kubernetes has strong scaling features that let applications scale horizontally to cope with a rise in load. Compared to vertical scaling in conventional settings, this is more effective and economical.

Version control: Container images may easily be rolled back to earlier versions or updated because they are versioned. Version control and simple administration of programme releases are encouraged by this.

Resource Efficiency: Compared to virtual machines, resource overhead is reduced by containers' lightweight design and shared host OS kernel. Cost reductions and effective resource use result from this.

Automation: Automation is a key component of DevOps practises. Tools for containerization and orchestration automate deployment, scaling, and management processes, minimising the need for manual labour and error-prone human judgement.

Consistency: Containers ensure consistency throughout the development, testing, and production environments, which helps to prevent environment-related problems.

5.0 Self-Learning

- We have used the course materials for learning the basics of building image using Dockerfile, pulling from library and running containers.
- We also found some details from the articles provided in the lab and lecture documents.
- We have had some errors in running and stuck when something was added and had to remove those, we used ChatGPT for the errors, and asked how to remove the extra things added such as containers those were exited and not running.
- We have watched some YouTube videos about docker containers and it's working.

References

- https://www.youtube.com/watch?v=cdqbPfGkUu4&t=468s
- https://docs.docker.com/engine/reference/commandline/run/
- https://www.howtogeek.com/devops/how-to-run-commands-inside-kubernetes-pod-containers/
- https://chat.openai.com/
- https://www.netapp.com/devops-solutions/what-are-containers/
- https://circleci.com/blog/benefits-of-containerization/