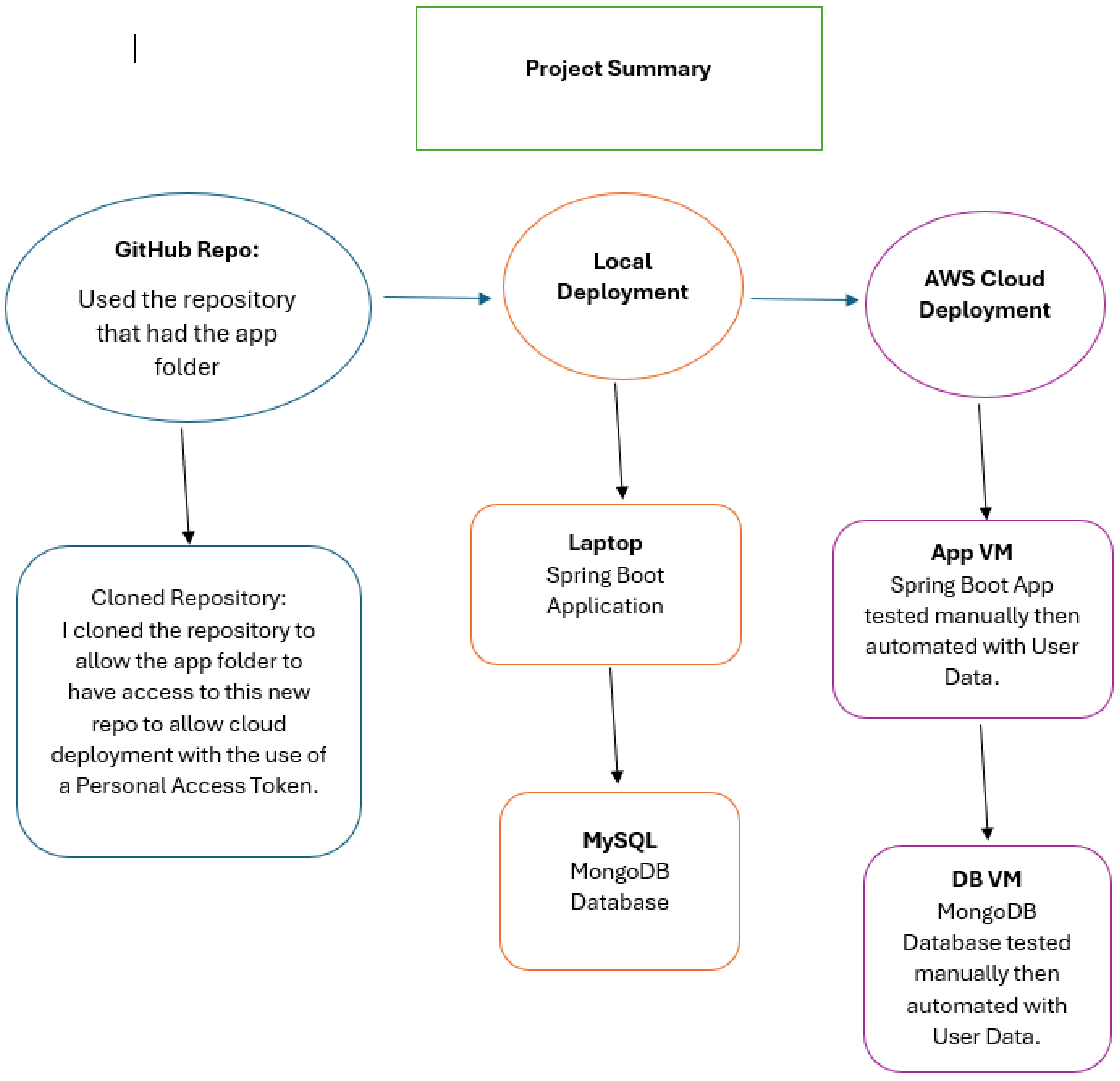
Java Springboot app Notes

Project Architecture


Project Summary
GitHub Repo: Used the repository that had the app folder
Local Deployment
AWS Cloud Deployment
Cloned Repository: I cloned the repository to allow the app folder to have access to this new repo to allow cloud deployment with the use of a Personal Access Token.
Laptop Spring Boot Application
MySQL MongoDB Database
App VM Spring Boot App tested manually then automated with User Data.
DB VM MongoDB Database tested manually then automated with User Data.

This shows the overall project architecture and the steps I have take to get the app deployed using the cloud with applications and platforms such as Kubernetes and Docker.

It begins with a GitHub repository containing the application folder. That repository is then cloned into a new repo through a PAT (Personal Access Token) to ensure that the application folder is accessible for later cloud deployment steps. From there, the workflow proceeds to Local Deployment, where the Spring Boot application runs on a laptop and connects to a local MySQL/MongoDB database for initial testing. Once the application is verified locally, the process moves to AWS Cloud Deployment, where two virtual machines (VMs) are used: one for running the Spring Boot application (App VM) and another for the database (DB VM). Both VMs are tested manually at first and then automated using User Data scripts, enabling consistent, repeatable deployment in the cloud.

Step 1: Create a Git repo

- Create a Git repo and clone it into Git bash.
- Follow the steps from the README and make sure the repo is private.
- Commands to follow:
  - `git init`
  - `git add .`
  - `git commit -m "Initial commit of file"`
  - `git remote add origin "https://github.com/<USERNAME>/repo-name"`
  - `git branch -M main`
  - `git push -u origin main`

Step 2: Deploy the app locally

- In order the get the app working locally, we need to install 3 things:
  - Java 17
  - Maven
  - MySQL
- After installing the right dependencies and verifying by checking the version in the terminal, login to MySQL:
  - Connect manually to root user: `\connect root@localhost`
  - Make sure you're in MySQL Shell by typing `\sql`
  - `mysql -u root -p`
- Set password to `Burntwood1!`
- We would need to create the database first as we haven't made it before nor has it been made for us.
  - CREATE DATABASE library;
  - USE library;
- Run this code in the terminal after making sure that your in the app's repo where the file `library.sql` is. This is what will seed the data in the database.
- In the terminal, run: `mysql -u root -p library < library.sql`
- Verify the tables and data by entering this in the MySQL Command line:
  - `SHOW TABLES;
  - SELECT * FROM authors;
- Once you can see the data, we need to enter the environmental variables.

- The app connects to MySQL using `DB_HOST`, `DB_USER` and `DB_PASS`. I set this in my local machine using the "Windows" button to "edit the system environment variables".
- `DB_HOST: jdbc:mysql://localhost:3306/library`
- `DB_USER: root`
- `DB_PASS: Burntwood1!`
- After setting the variables, I had to restart the terminal again to make sure that the variables apply.
- After entering the terminal, I made sure to get my file path to the app's directory. Once I'm in the repo I did `cd LibraryProject2`.
- Then I used Maven to start the app: `mvn spring-boot:run`
- After the process runs, look out for these lines: `Tomcat started on port(s): 5000 (http) Started SpringapiApplication in 3.234 seconds`
- To test the app, in the browser go to: `http://localhost:500/web/authors`. This will show the list of the 4 authors from the database.
- To see all authors: `http://localhost:5000/authors`.
- Single author (ID 3): `http://localhost:5000/author/3`
- To stop the app: `mvn spring-boot:stop`

Blockers/Errors:

- I had the longest time trying to fix the app using Java V17 even after installing the correct one.
- The app was continously using Java V1 and I couldn't understand why.
- I just opened a new terminal and checked the version that the app was to use after doing `java -version` and when it said 17, I understood that it now works.
- Also, I created a new environmental variable named "JAVA_HOME" and set it to JDK's path.
- Then under "System variables", I found "Path" and created a new variable: `%JAVA_HOME%\bin`.

Step 3: Deploying app using AWS VM

- I created 2 VM. One for DB and one for the app.

DB VM:

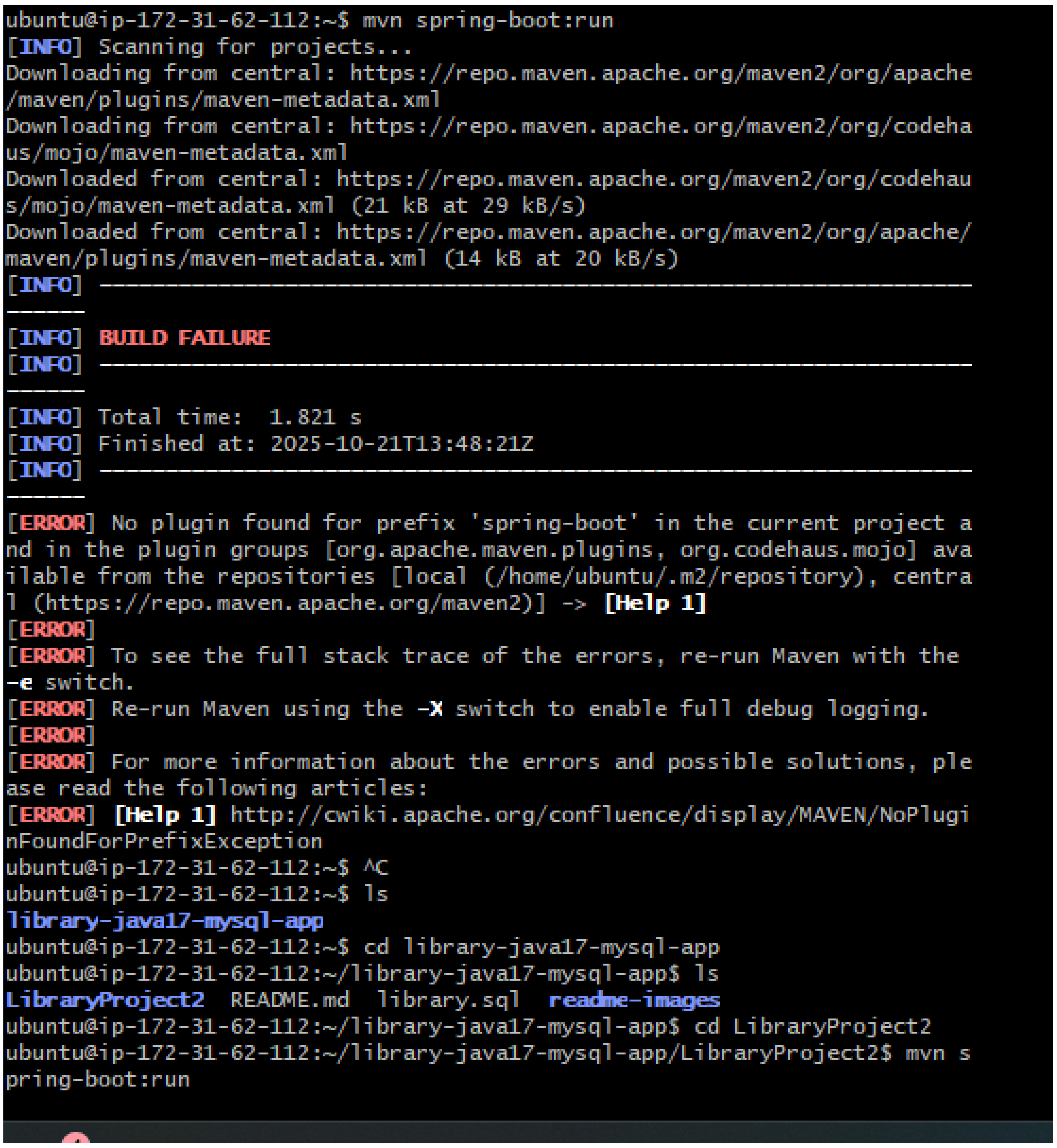
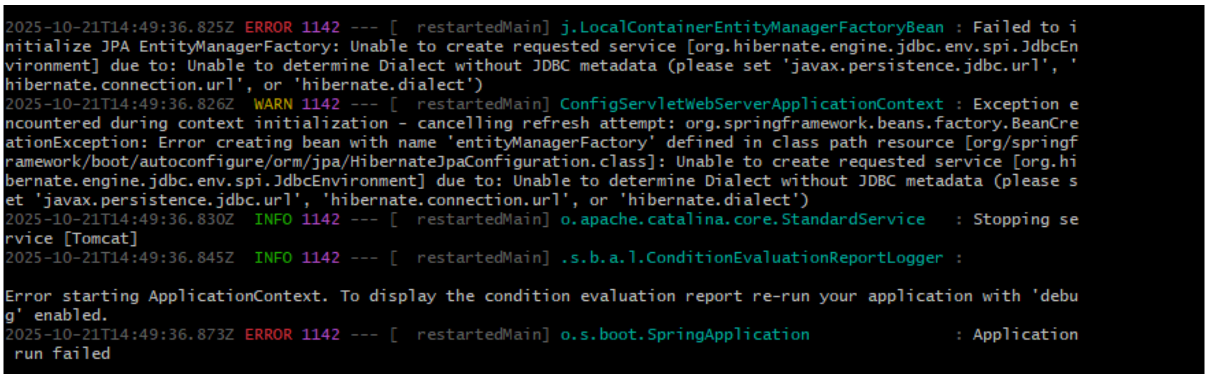
- Ubuntu 22.04 EC2 instance with t3.micro image (free tier).
- Key pair: tech511-afsheen-aws
- Security Groups:
  - Allow SSH (port 22)
  - Add Security Group rule:
    - Type: MYSQL/Aurora
    - Port: 3306
    - Source: Custom
    - Enter: 0.0.0.0/0 (for testing)
    - Add User Data
- After launching the instance, I SSH'd into a new terminal and typed in `sudo systemctl status mysql`.
- This showed "active" which told me that everything was working well.

APP VM:

- Ubuntu 22.04 EC2 instance with t3.micro image (free tier)
- Key pair: tech511-afsheen-aws
- Security Groups:
- Allow SSH (port 22)
- Custom TCP (port 5000) (for app web access)
- Add User Data but make sure that you type in the Private IPv4 address from the DB VM.
- From the User Data, enter your github username and repo name.
- To verify everything works, got to the app VM and be in the java app repo where `pom.xml` is. Click on the public IPv4 address from the app VM: `http://<APP_PUBLIC_IP>:5000/web/authors`
- Make sure you have cloned the repo in the DB VM. I had not done this and had to do a `git clone` using a new SSH key token since the repo is private and I couldn't remember the username and password it was asking for.
- In the App VM, to build the app, I did this: `./mvnw clean package -DskipTests`

Blockers/Errors:

- I had problem with SQL rejecting permissions for the app to access the database even though it could. It only gave access for 'root' to access the db.
- I had to change the environmental variables in the bash scripts/user data that changed the permissions to allow the app to access the DB.

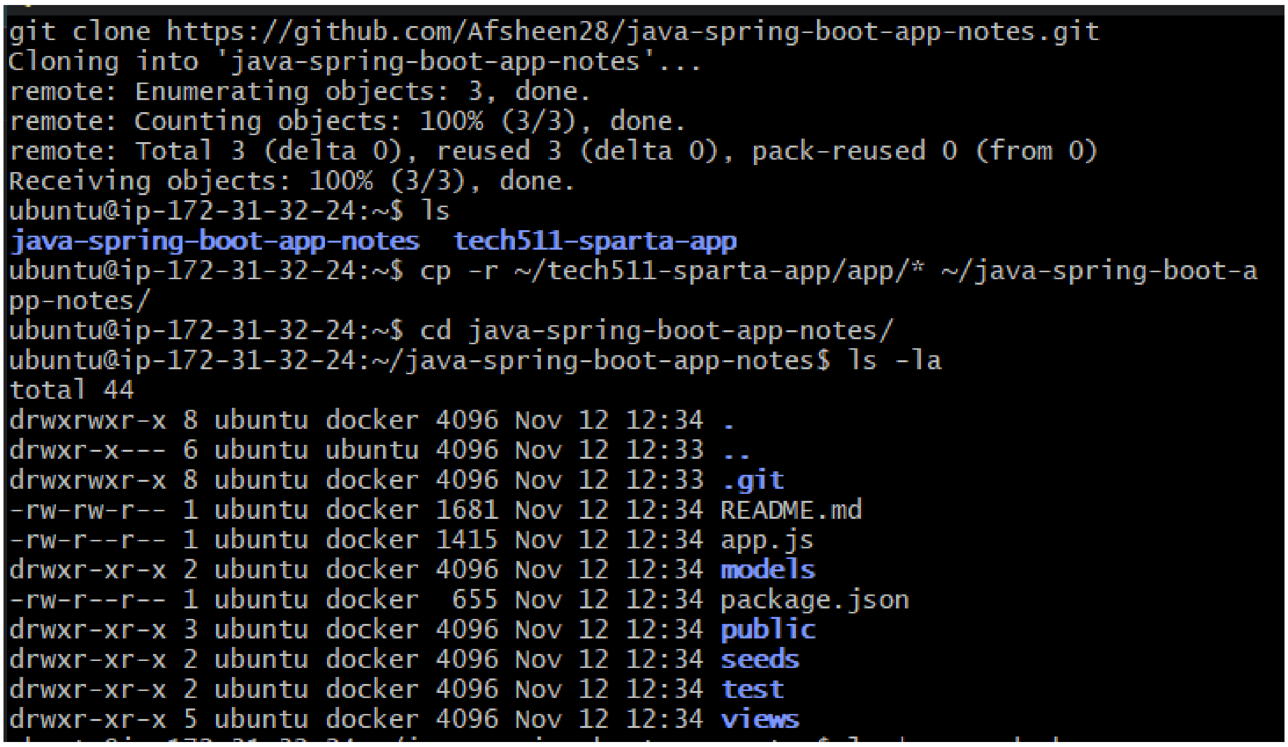
- Also, I kept on forgetting to get to the correct file path that wasn't getting the app deployed.
  
  ```
  ubuntu@ip-172-31-62-112:~$ mvn spring-boot:run
  [INFO] Scanning for projects...
  Downloading from central: https://repo.maven.apache.org/maven2/org/apache
  /maven/plugins/maven-metadata.xml
  Downloading from central: https://repo.maven.apache.org/maven2/org/codeha
  us/mojo/maven-metadata.xml
  Downloaded from central: https://repo.maven.apache.org/maven2/org/codehau
  s/mojo/maven-metadata.xml (21 kB at 29 kB/s)
  Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/
  maven/plugins/maven-metadata.xml (14 kB at 20 kB/s)
  [INFO] ------------------------------------------------------------------
  ------
  [INFO] BUILD FAILURE
  [INFO] ------------------------------------------------------------------
  ------
  [INFO] Total time:  1.821 s
  [INFO] Finished at: 2025-10-21T13:48:21Z
  [INFO] ------------------------------------------------------------------
  ------
  [ERROR] No plugin found for prefix 'spring-boot' in the current project a
  nd in the plugin groups [org.apache.maven.plugins, org.codehaus.mojo] ava
  ilable from the repositories [local (/home/ubuntu/.m2/repository), centra
  l (https://repo.maven.apache.org/maven2)] -> [Help 1]
  [ERROR]
  [ERROR] To see the full stack trace of the errors, re-run Maven with the
  -e switch.
  [ERROR] Re-run Maven using the -X switch to enable full debug logging.
  [ERROR]
  [ERROR] For more information about the errors and possible solutions, ple
  ase read the following articles:
  [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/NoPlugi
  nFoundForPrefixException
  ubuntu@ip-172-31-62-112:~$ ^C
  ubuntu@ip-172-31-62-112:~$ ls
  library-java17-mysql-app
  ubuntu@ip-172-31-62-112:~$ cd library-java17-mysql-app
  ubuntu@ip-172-31-62-112:~/library-java17-mysql-app$ ls
  LibraryProject2  README.md  library.sql  readme-images
  ubuntu@ip-172-31-62-112:~/library-java17-mysql-app$ cd LibraryProject2
  ubuntu@ip-172-31-62-112:~/library-java17-mysql-app/LibraryProject2$ mvn s
  pring-boot:run
  ```
- One error I got was that MySQL was not recognising the environment variables I set and constantly asked for variables to start with JBDC. To fix this I had to set the environmental variables again:
  - `export SPRING_DATASOURCE_URL=jdbc:mysql://172.31.25.143:3306/library`
  - `export SPRING_DATASOURCE_USERNAME=root`
  - `export SPRING_DATASOURCE_PASSWORD=Burntwood1!`
    
    ```
    2025-10-21T14:49:36.825Z ERROR 1142 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Failed to i
    nitialize JPA EntityManagerFactory: Unable to create requested service [org.hibernate.engine.jdbc.env.spi.JdbcEn
    vironment] due to: Unable to determine Dialect without JDBC metadata (please set 'javax.persistence.jdbc.url', '
    hibernate.connection.url', or 'hibernate.dialect')
    2025-10-21T14:49:36.826Z  WARN 1142 --- [  restartedMain] ConfigServletWebServerApplicationContext : Exception e
    ncountered during context initialization - cancelling refresh attempt: org.springframework.beans.factory.BeanCre
    ationException: Error creating bean with name 'entityManagerFactory' defined in class path resource [org/springf
    ramework/boot/autoconfigure/orm/jpa/HibernateJpaConfiguration.class]: Unable to create requested service [org.hi
    bernate.engine.jdbc.env.spi.JdbcEnvironment] due to: Unable to determine Dialect without JDBC metadata (please s
    et 'javax.persistence.jdbc.url', 'hibernate.connection.url', or 'hibernate.dialect')
    2025-10-21T14:49:36.830Z  INFO 1142 --- [  restartedMain] o.apache.catalina.core.StandardService   : Stopping se
    rvice [Tomcat]
    2025-10-21T14:49:36.845Z  INFO 1142 --- [  restartedMain] .s.b.a.l.ConditionEvaluationReportLogger :

    Error starting ApplicationContext. To display the condition evaluation report re-run your application with 'debu
    g' enabled.
    2025-10-21T14:49:36.873Z ERROR 1142 --- [  restartedMain] o.s.boot.SpringApplication               : Application
     run failed
    ```

Step 4: Deploy app and database on AWS using Docker

- To start of this task, I went on to create an EC2 instance on AWS using an `22.04 LTS AMI`, a `t3.small` instance type and inbound ports 22, 80, 3000, and 27017.
- I then added user data:

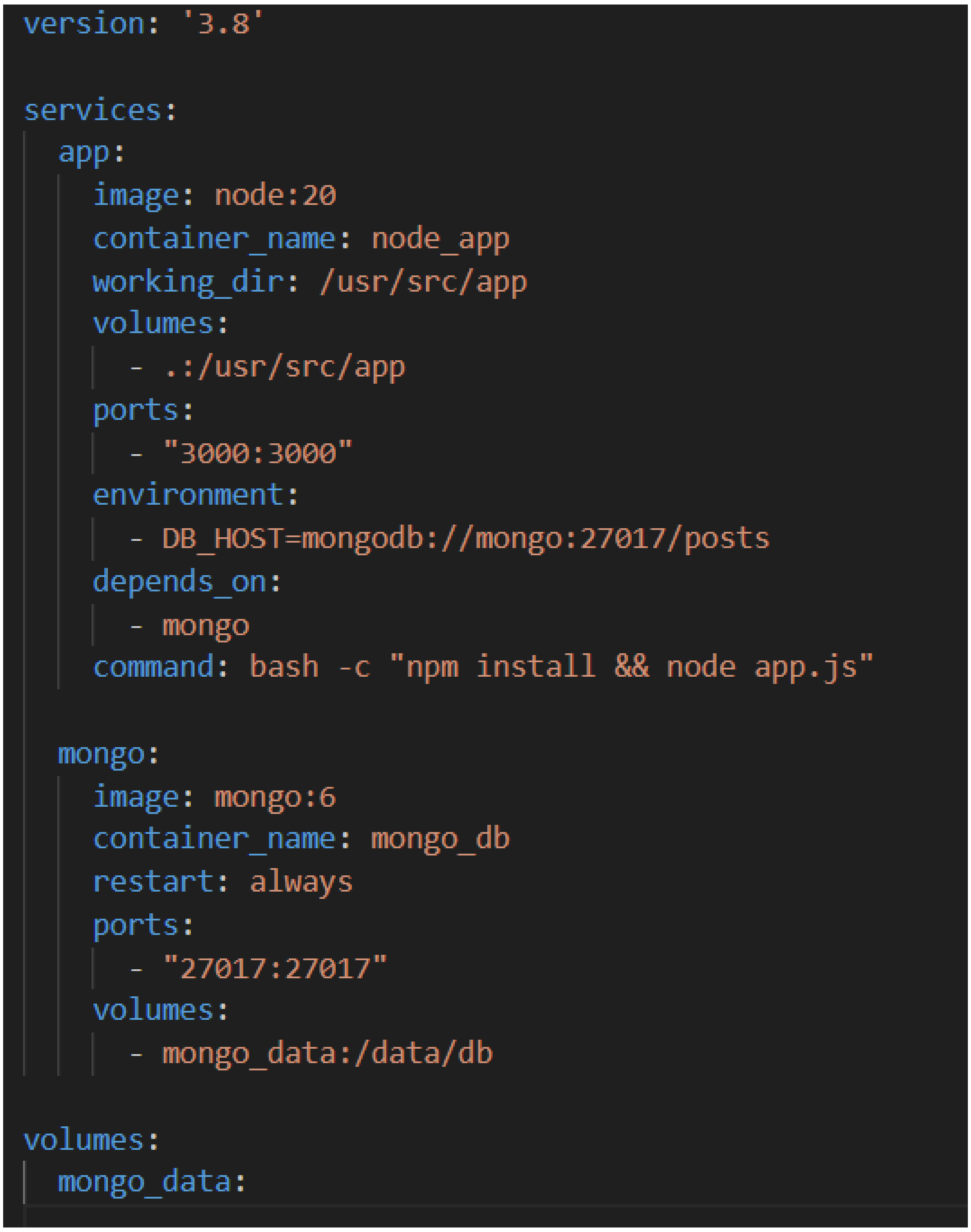
```bash
#!/bin/bash
apt-get update -y
apt-get install -y git docker.io docker-compose -y
systemctl enable docker
systemctl start docker

# Deploy app automatically
cd /home/ubuntu
git clone https://github.com/Afsheen28/tech511-sparta-app.git
cd tech511-sparta-app
docker-compose up -d
```

- Once the instance had launched, I SSH'd into the VM.
- I then cloned my `app` folder from my `tech511-sparta-app` to this `java-spring-boot-app-notes` repository. This is because the app folder was not in this repo.

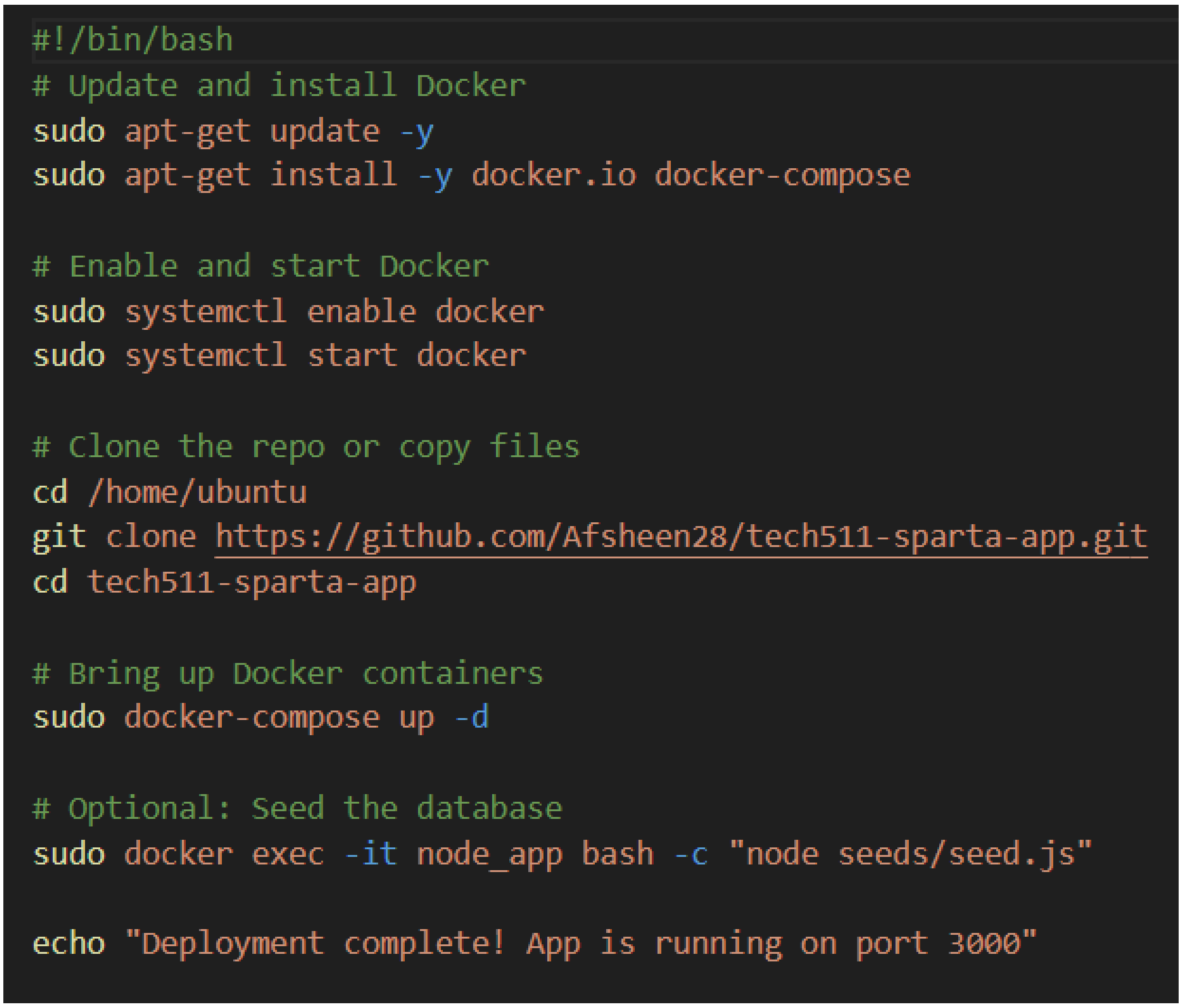


```
git clone https://github.com/Afsheen28/java-spring-boot-app-notes.git
Cloning into 'java-spring-boot-app-notes'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
ubuntu@ip-172-31-32-24:~$ ls
java-spring-boot-app-notes  tech511-sparta-app
ubuntu@ip-172-31-32-24:~$ cp -r ~/tech511-sparta-app/app/* ~/java-spring-boot-a
pp-notes/
ubuntu@ip-172-31-32-24:~$ cd java-spring-boot-app-notes/
ubuntu@ip-172-31-32-24:~/java-spring-boot-app-notes$ ls -la
total 44
drwxrwxr-x 8 ubuntu docker 4096 Nov 12 12:34 .
drwxr-x--- 6 ubuntu ubuntu 4096 Nov 12 12:33 ..
drwxrwxr-x 8 ubuntu docker 4096 Nov 12 12:33 .git
-rw-rw-r-- 1 ubuntu docker 1681 Nov 12 12:34 README.md
-rw-r--r-- 1 ubuntu docker 1415 Nov 12 12:34 app.js
drwxr-xr-x 2 ubuntu docker 4096 Nov 12 12:34 models
-rw-r--r-- 1 ubuntu docker  655 Nov 12 12:34 package.json
drwxr-xr-x 3 ubuntu docker 4096 Nov 12 12:34 public
drwxr-xr-x 2 ubuntu docker 4096 Nov 12 12:34 seeds
drwxr-xr-x 2 ubuntu docker 4096 Nov 12 12:34 test
drwxr-xr-x 5 ubuntu docker 4096 Nov 12 12:34 views
```

- Once the app folder was in the repo, I created a `docker-compose.yml` and `provision.sh` file.

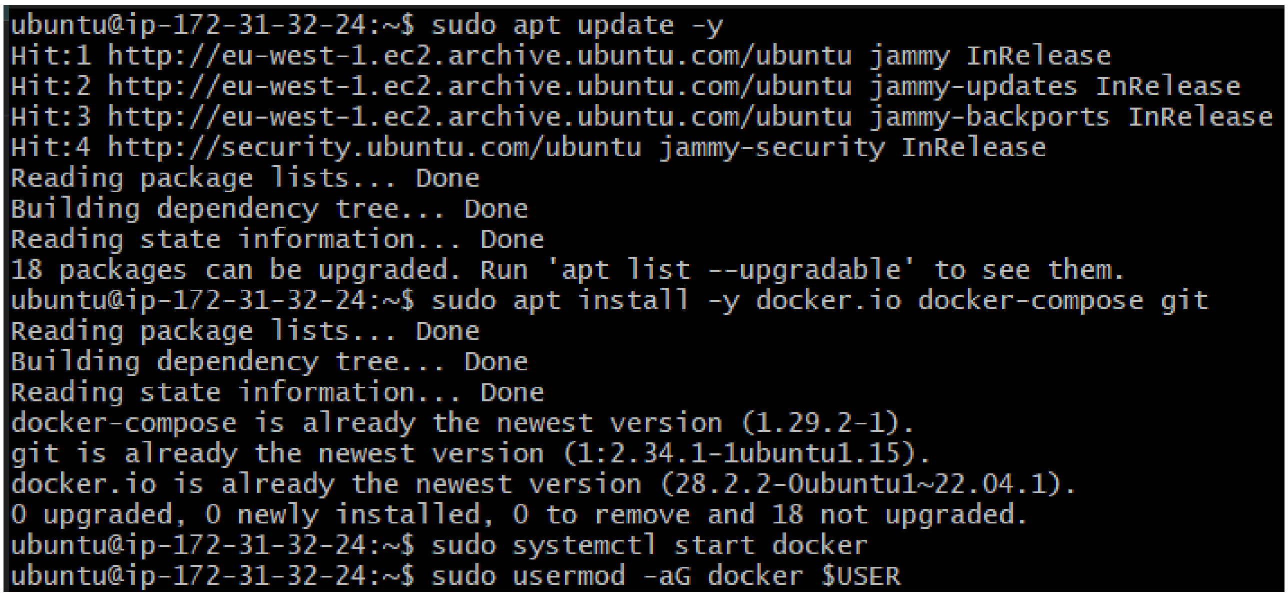


```yaml
version: '3.8'

services:
  app:
    image: node:20
    container_name: node_app
    working_dir: /usr/src/app
    volumes:
      - .:/usr/src/app
    ports:
      - "3000:3000"
    environment:
      - DB_HOST=mongodb://mongo:27017/posts
    depends_on:
      - mongo
    command: bash -c "npm install && node app.js"

  mongo:
    image: mongo:6
    container_name: mongo_db
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
```

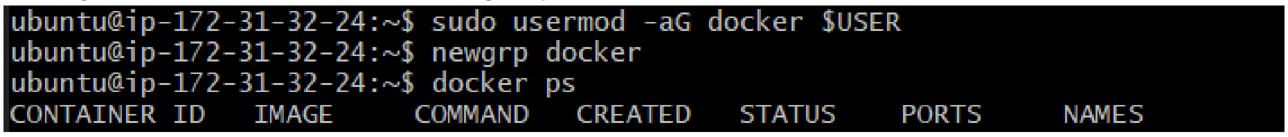


```
#!/bin/bash
# Update and install Docker
sudo apt-get update -y
sudo apt-get install -y docker.io docker-compose

# Enable and start Docker
sudo systemctl enable docker
sudo systemctl start docker

# Clone the repo or copy files
cd /home/ubuntu
git clone https://github.com/Afsheen28/tech511-sparta-app.git
cd tech511-sparta-app

# Bring up Docker containers
sudo docker-compose up -d

# Optional: Seed the database
sudo docker exec -it node_app bash -c "node seeds/seed.js"

echo "Deployment complete! App is running on port 3000"
```

- I then updated the system and installed the required packages (like docker):



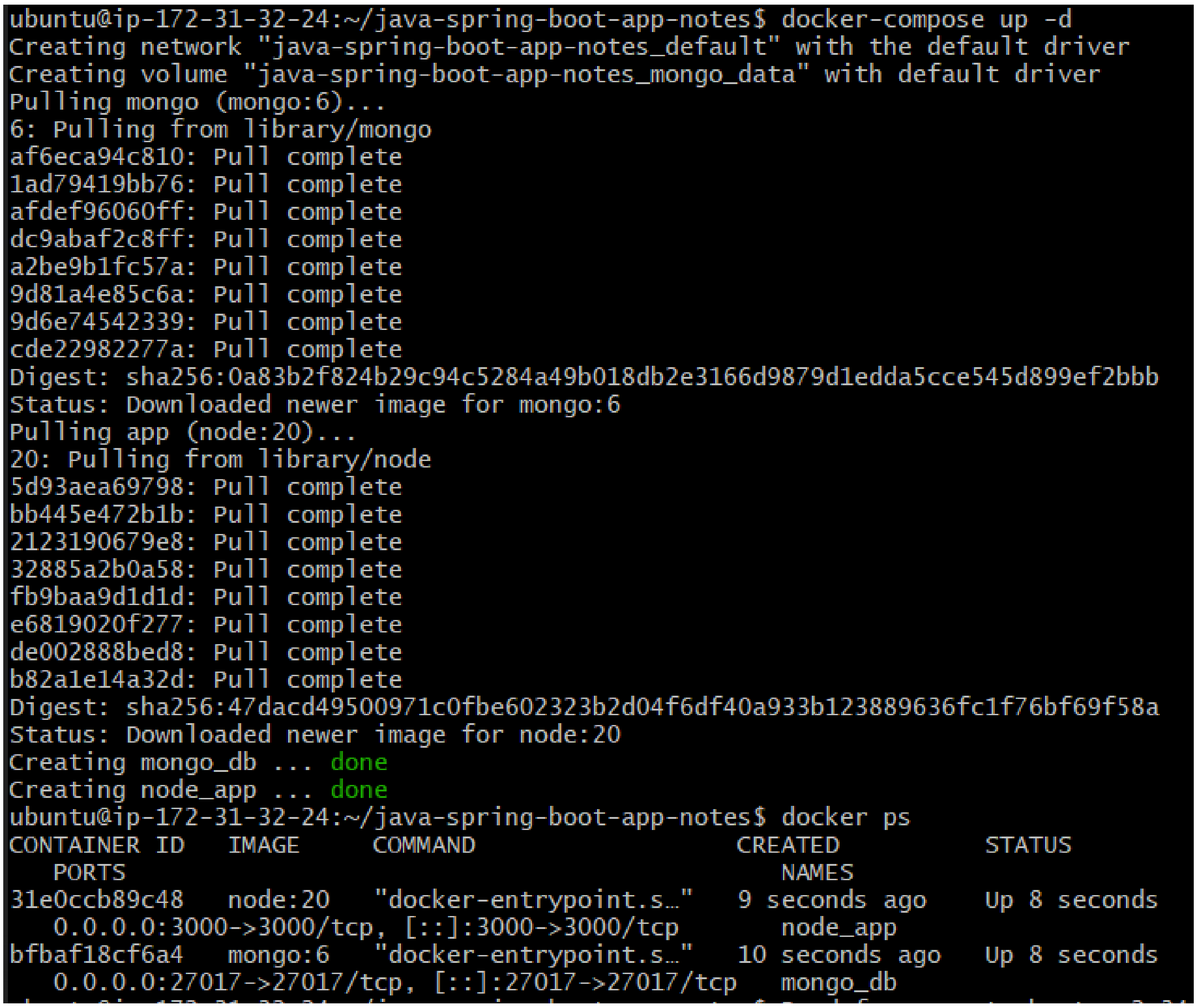
```
ubuntu@ip-172-31-32-24:~$ sudo apt update -y
Hit:1 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-west-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
18 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-32-24:~$ sudo apt install -y docker.io docker-compose git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-compose is already the newest version (1.29.2-1).
git is already the newest version (1:2.34.1-1ubuntu1.15).
docker.io is already the newest version (28.2.2-0ubuntu1~22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 18 not upgraded.
ubuntu@ip-172-31-32-24:~$ sudo systemctl start docker
ubuntu@ip-172-31-32-24:~$ sudo usermod -aG docker $USER
```

- I then gave user permissions to run Docker without `sudo`. By default only root can access Docker. Adding the `ubuntu` user to the docker group allows Docker commands without `sudo`.
- 

```
ubuntu@ip-172-31-32-24:~$ sudo usermod -aG docker $USER
ubuntu@ip-172-31-32-24:~$ newgrp docker
ubuntu@ip-172-31-32-24:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

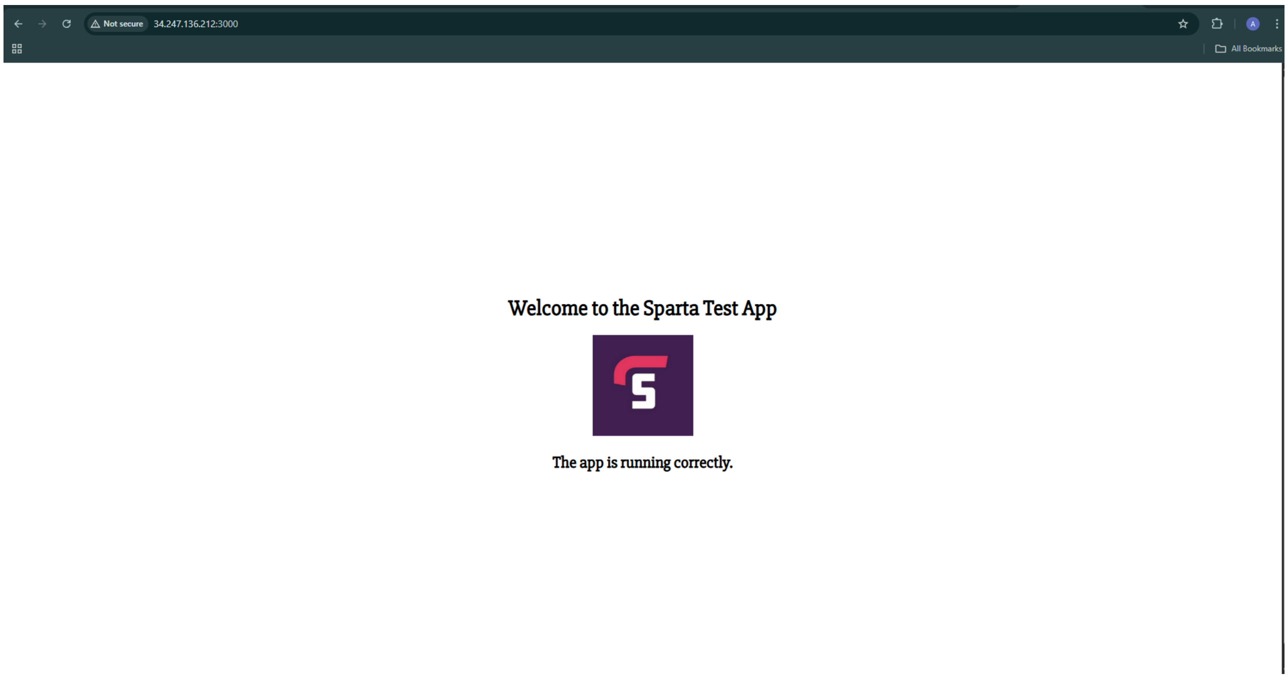
- I then verified docker access by running `docker ps`. This confirmed that Docker was working and accessible to the user. An empty table was shown meaning that no containers were running yet which was true.

- I then deployed the containers by running `docker-compose up -d`. This launched both the Node.js and MongoDB containers in detached mode. Compose automatically created an internal network linking the app to the database.
- After that, I ran `docker ps` to verify both servics were running and mapped to their ports.



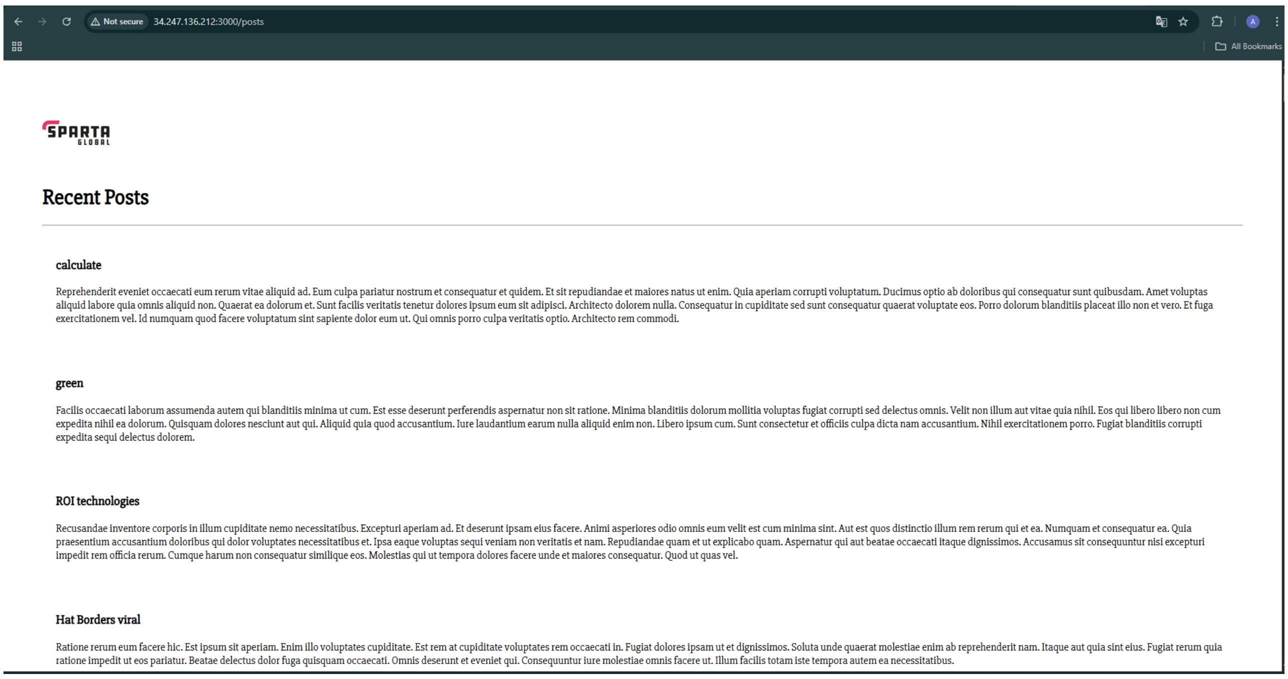
```
ubuntu@ip-172-31-32-24:~/java-spring-boot-app-notes$ docker-compose up -d
Creating network "java-spring-boot-app-notes_default" with the default driver
Creating volume "java-spring-boot-app-notes_mongo_data" with default driver
Pulling mongo (mongo:6)...
6: Pulling from library/mongo
af6eca94c810: Pull complete
1ad79419bb76: Pull complete
afdef96060ff: Pull complete
dc9abaf2c8ff: Pull complete
a2be9b1fc57a: Pull complete
9d81a4e85c6a: Pull complete
9d6e74542339: Pull complete
cde22982277a: Pull complete
Digest: sha256:0a83b2f824b29c94c5284a49b018db2e3166d9879d1edda5cce545d899ef2bbb
Status: Downloaded newer image for mongo:6
Pulling app (node:20)...
20: Pulling from library/node
5d93aea69798: Pull complete
bb445e472b1b: Pull complete
2123190679e8: Pull complete
32885a2b0a58: Pull complete
fb9baa9d1d1d: Pull complete
e6819020f277: Pull complete
de002888bed8: Pull complete
b82a1e14a32d: Pull complete
Digest: sha256:47dacd49500971c0fbe602323b2d04f6df40a933b123889636fc1f76bf69f58a
Status: Downloaded newer image for node:20
Creating mongo_db ... done
Creating node_app ... done
ubuntu@ip-172-31-32-24:~/java-spring-boot-app-notes$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
   PORTS                                          NAMES
31e0ccb89c48   node:20   "docker-entrypoint.s…"   9 seconds ago    Up 8 seconds
   0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp       node_app
bfbaf18cf6a4   mongo:6   "docker-entrypoint.s…"   10 seconds ago   Up 8 seconds
   0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp   mongo_db
```

- I then tested the application by going on `http://<EC2_PUBLIC_ID>:3000` and `http://<EC2_PUBLIC_ID>:3000/posts`



Welcome to the Sparta Test App

The app is running correctly.



Recent Posts

calculate

Reprehenderit eveniet occaecati eum rerum vitae aliquid ad. Eum culpa pariatur nostrum et consequatur et quidem. Et sit repudiandae et maiores natus ut enim. Quia aperiam corrupti voluptatum. Ducimus optio ab doloribus qui consequatur sunt quibusdam. Amet voluptas aliquid labore quia omnis aliquid non. Quaerat ea dolorum et. Sunt facilis veritatis tenetur dolores ipsum eum sit adipisci. Architecto dolorem nulla. Consequatur in cupiditate sed sunt consequatur quaerat voluptate eos. Porro dolorum blanditiis placeat illo non et vero. Et fuga exercitationem vel. Id numquam quod facere voluptatum sint sapiente dolor eum ut. Qui omnis porro culpa veritatis optio. Architecto rem commodi.

green

Facilis occaecati laborum assumenda autem qui blanditiis minima ut cum. Est esse deserunt perferendis aspernatur non sit ratione. Minima blanditiis dolorum mollitia voluptas fugiat corrupti sed delectus omnis. Velit non illum aut vitae quia nihil. Eos qui libero libero non cum expedita nihil ea dolorum. Quisquam dolores nesciunt aut qui. Aliquid quia quod accusantium. Iure laudantium earum nulla aliquid enim non. Libero ipsum cum. Sunt consectetur et officiis culpa dicta nam accusantium. Nihil exercitationem porro. Fugiat blanditiis corrupti expedita sequi delectus dolorem.

ROI technologies

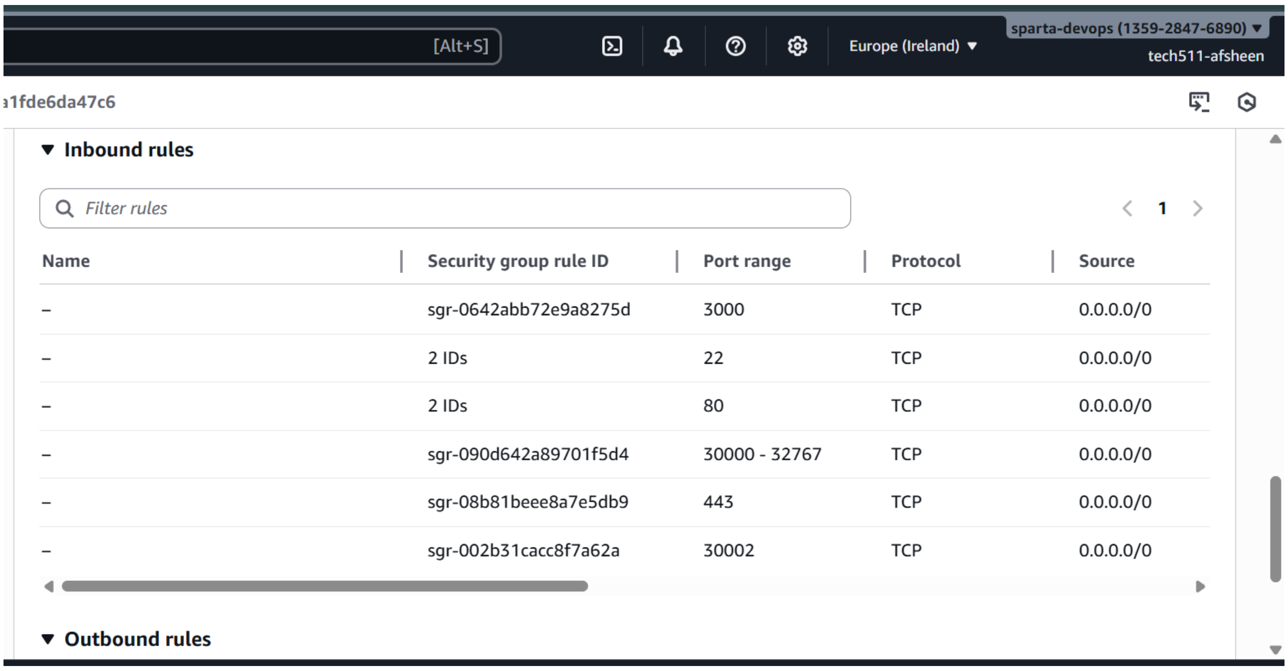
Recusandae inventore corporis in illum cupiditate nemo necessitatibus. Excepturi aperiam ad. Et deserunt ipsam eius facere. Animi asperiores odio omnis eum velit est cum minima sint. Aut est quos distinctio illum rem rerum qui et ea. Numquam et consequatur ea. Quia praesentium accusantium doloribus qui dolor voluptates necessitatibus et. Ipsa eaque voluptas sequi veniam non veritatis et nam. Repudiandae quam et ut explicabo quam. Aspernatur qui aut beatae occaecati itaque dignissimos. Accusamus sit consequuntur nisi excepturi impedit rem officia rerum. Cumque harum non consequatur similique eos. Molestias qui ut tempora dolores facere unde et maiores consequatur. Quod ut quas vel.

Hat Borders viral

Ratione rerum eum facere hic. Est ipsum sit aperiam. Enim illo voluptates cupiditate. Est rem at cupiditate voluptates rem occaecati in. Fugiat dolores ipsam ut et dignissimos. Soluta unde quaerat molestiae enim ab reprehenderit nam. Itaque aut quia sint eius. Fugiat rerum quia ratione impedit ut eos pariatur. Beatae delectus dolor fuga quisquam occaecati. Omnis deserunt et eveniet qui. Consequuntur iure molestiae omnis facere ut. Illum facilis totam iste tempora autem ea necessitatibus.

Step 5: Deploy app and database on AWS using Kubernetes (Minikube)

- Firstly, I created an EC2 instance with an instance type of t3.medium due to the RAM size I needed. I also added the following inbound rules in the security group.

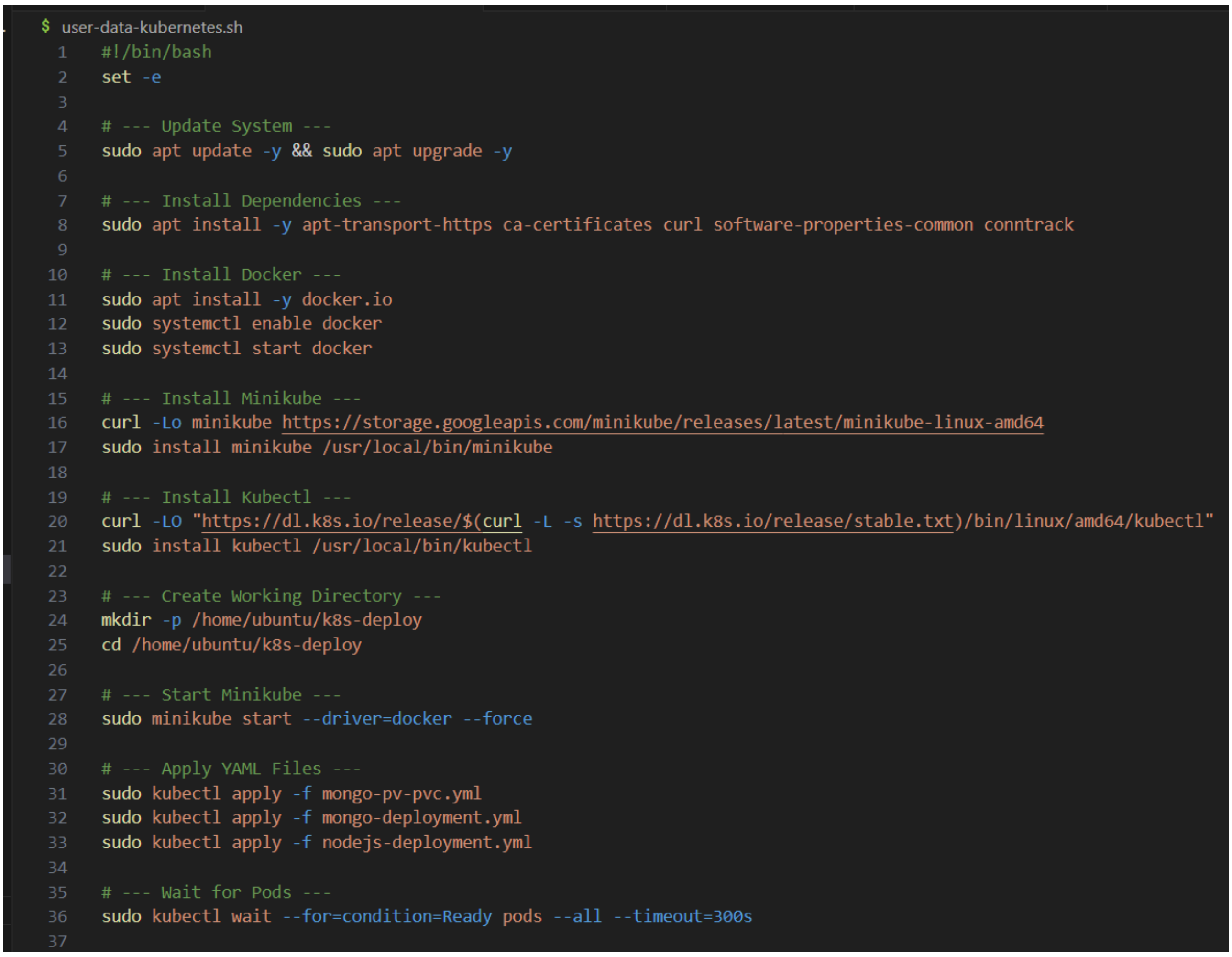

[Alt+S]
Europe (Ireland)
sparta-devops (1359-2847-6890)
tech511-afsheen
a1fde6da47c6

Inbound rules

Name	Security group rule ID	Port range	Protocol	Source
–	sgr-0642abb72e9a8275d	3000	TCP	0.0.0.0/0
–	2 IDs	22	TCP	0.0.0.0/0
–	2 IDs	80	TCP	0.0.0.0/0
–	sgr-090d642a89701f5d4	30000 - 32767	TCP	0.0.0.0/0
–	sgr-08b81beee8a7e5db9	443	TCP	0.0.0.0/0
–	sgr-002b31cacc8f7a62a	30002	TCP	0.0.0.0/0

Outbound rules

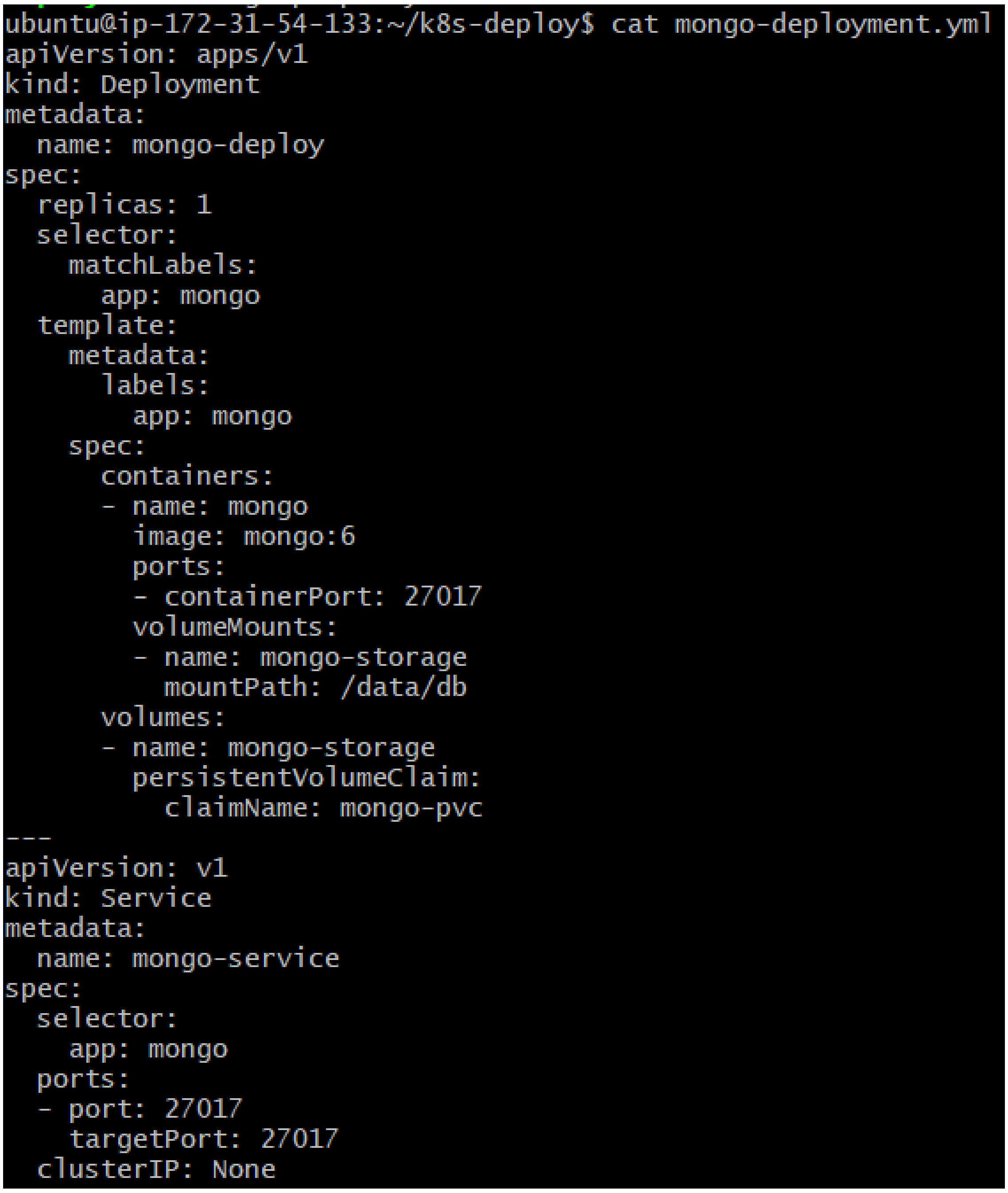
- I then added user data:


user-data-kubernetes.sh

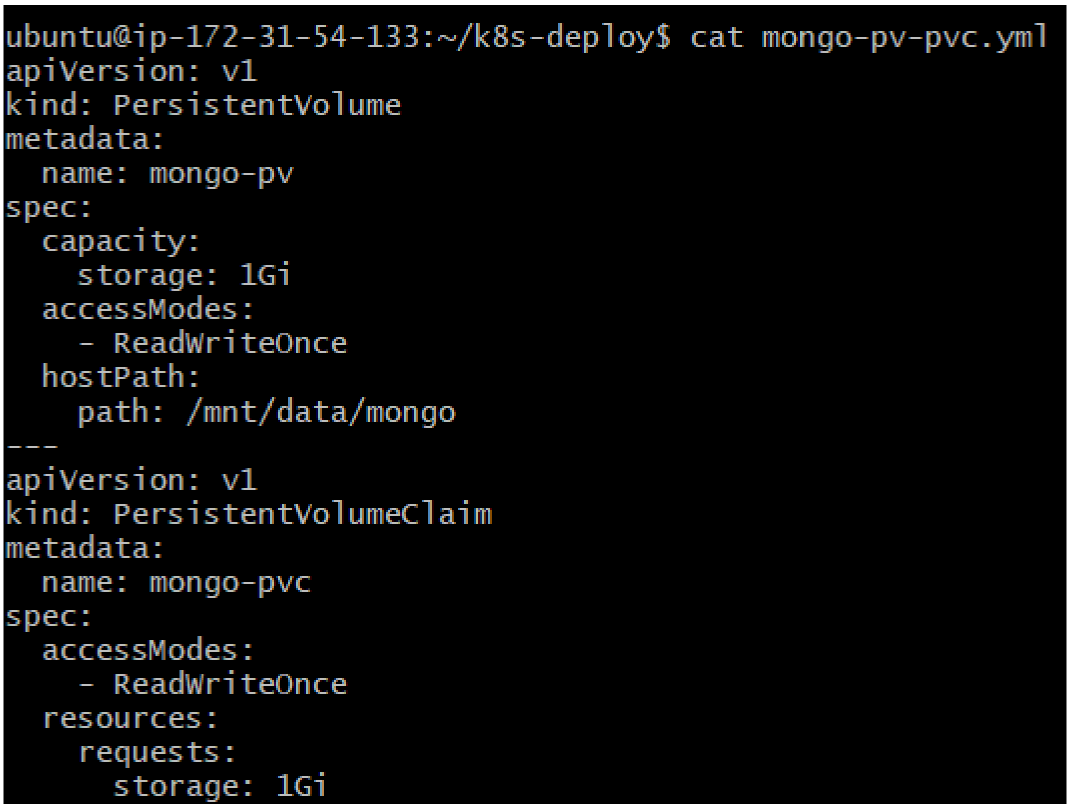
```bash
#!/bin/bash
set -e

# --- Update System ---
sudo apt update -y && sudo apt upgrade -y

# --- Install Dependencies ---
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common conntrack

# --- Install Docker ---
sudo apt install -y docker.io
sudo systemctl enable docker
sudo systemctl start docker

# --- Install Minikube ---
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube /usr/local/bin/minikube

# --- Install Kubectl ---
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install kubectl /usr/local/bin/kubectl

# --- Create Working Directory ---
mkdir -p /home/ubuntu/k8s-deploy
cd /home/ubuntu/k8s-deploy

# --- Start Minikube ---
sudo minikube start --driver=docker --force

# --- Apply YAML Files ---
sudo kubectl apply -f mongo-pv-pvc.yml
sudo kubectl apply -f mongo-deployment.yml
sudo kubectl apply -f nodejs-deployment.yml

# --- Wait for Pods ---
sudo kubectl wait --for=condition=Ready pods --all --timeout=300s

```

```
# --- Wait for Pods ---
sudo kubectl wait --for=condition=Ready pods --all --timeout=300s

# --- Display Deployment Info ---
sudo kubectl get pods -o wide
sudo kubectl get svc -o wide
```

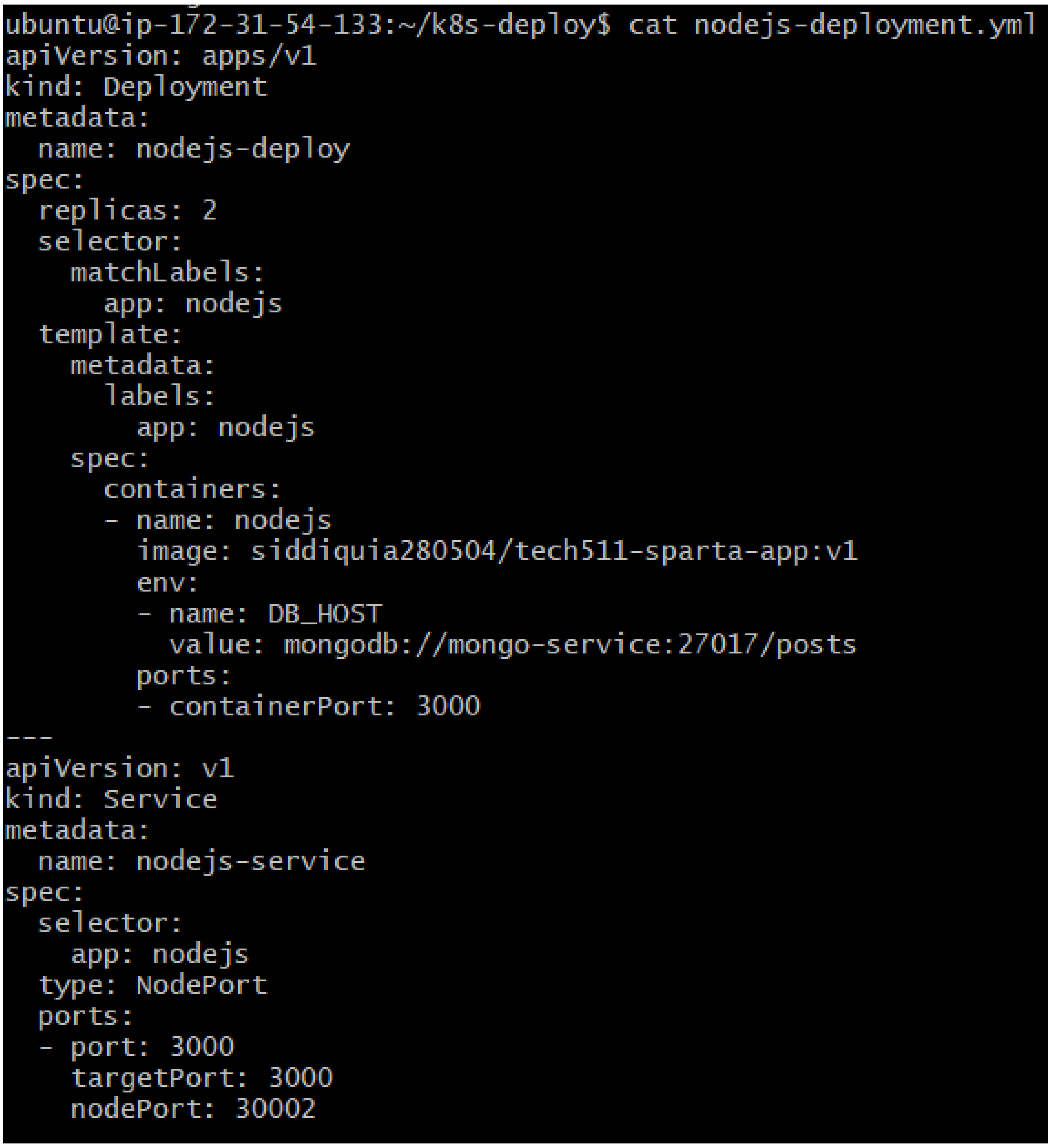
- Once I launched the instance, and SSH'd into a new Gitbash terminal, I created a repository named `k8s-deploy` and created 3 YAML files.



```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ cat mongo-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
      - name: mongo
        image: mongo:6
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: mongo-storage
          mountPath: /data/db
      volumes:
      - name: mongo-storage
        persistentVolumeClaim:
          claimName: mongo-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  selector:
    app: mongo
  ports:
  - port: 27017
    targetPort: 27017
  clusterIP: None
```

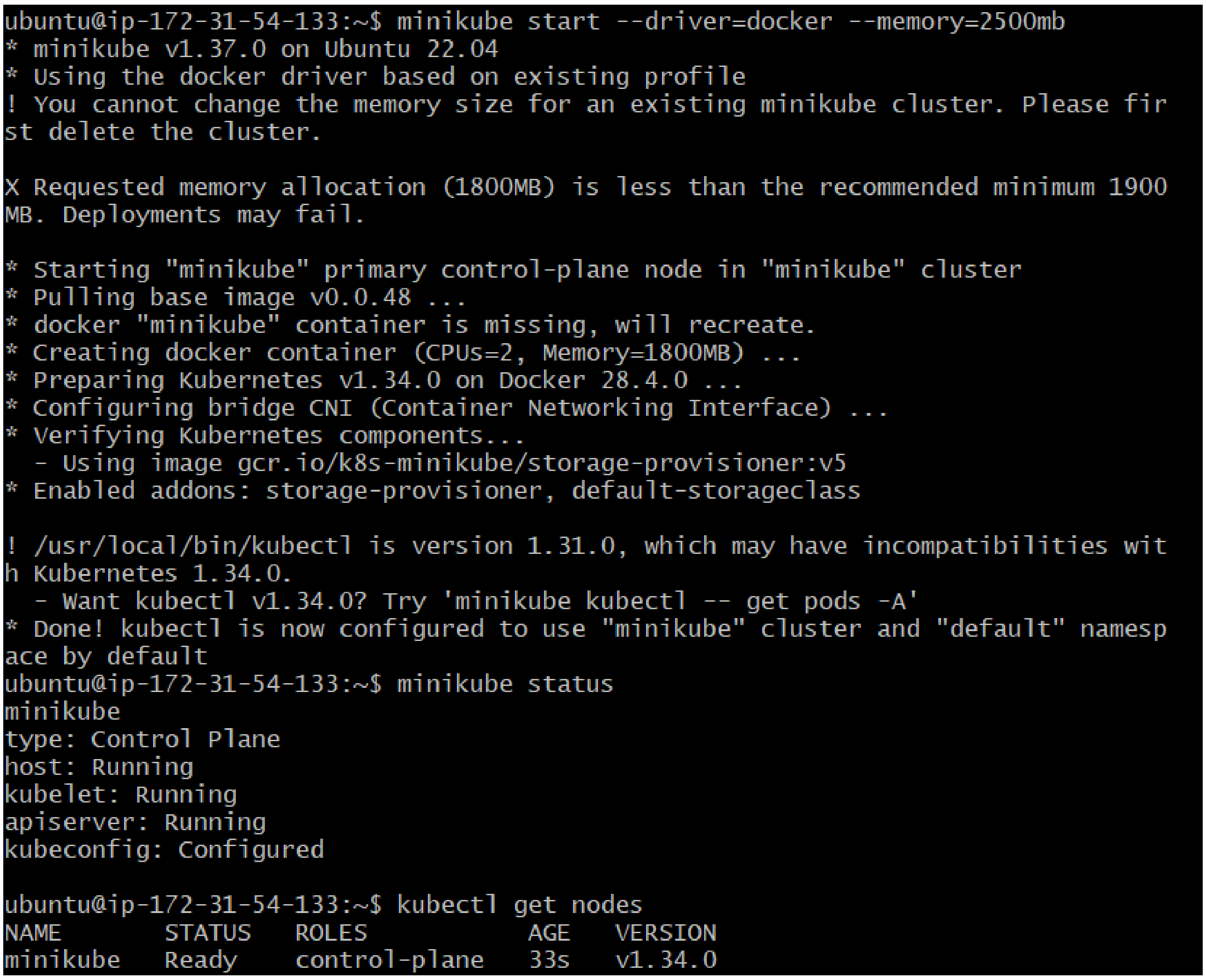


```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ cat mongo-pv-pvc.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongo-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data/mongo
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```



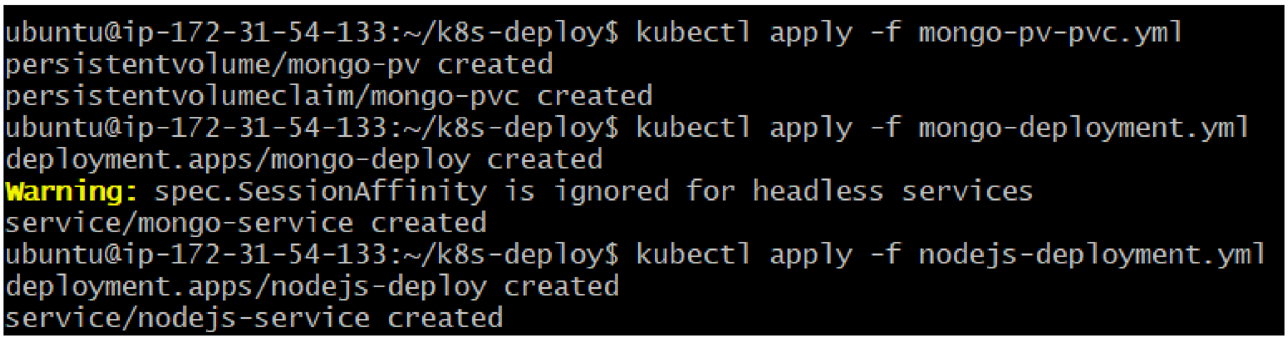
```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ cat nodejs-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodejs
  template:
    metadata:
      labels:
        app: nodejs
    spec:
      containers:
      - name: nodejs
        image: siddiquia280504/tech511-sparta-app:v1
        env:
        - name: DB_HOST
          value: mongodb://mongo-service:27017/posts
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: nodejs-service
spec:
  selector:
    app: nodejs
  type: NodePort
  ports:
  - port: 3000
    targetPort: 3000
    nodePort: 30002
```

- I then went to start minikube by running `minikube start --driver=docker --memory=2500mb`



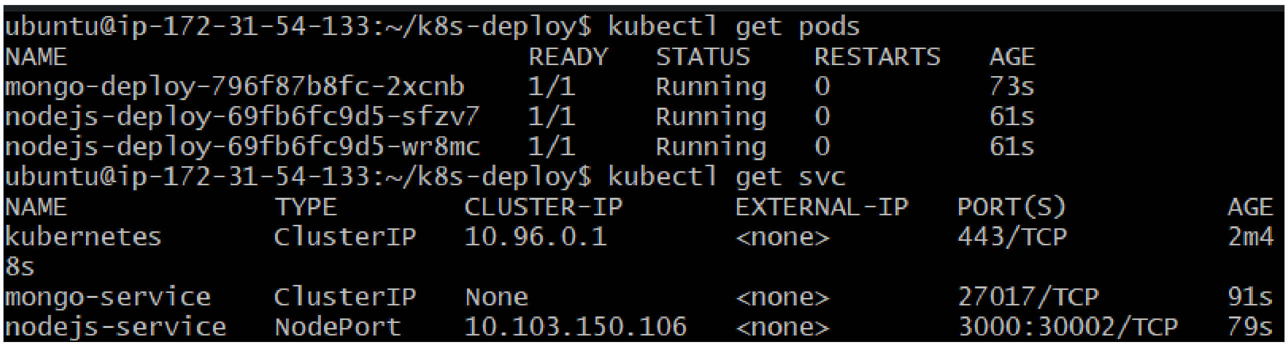
```
ubuntu@ip-172-31-54-133:~$ minikube start --driver=docker --memory=2500mb
* minikube v1.37.0 on Ubuntu 22.04
* Using the docker driver based on existing profile
! You cannot change the memory size for an existing minikube cluster. Please fir
st delete the cluster.

X Requested memory allocation (1800MB) is less than the recommended minimum 1900
MB. Deployments may fail.

* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.48 ...
* docker "minikube" container is missing, will recreate.
* Creating docker container (CPUs=2, Memory=1800MB) ...
* Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass

! /usr/local/bin/kubectl is version 1.31.0, which may have incompatibilities wit
h Kubernetes 1.34.0.
  - Want kubectl v1.34.0? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namesp
ace by default
ubuntu@ip-172-31-54-133:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

ubuntu@ip-172-31-54-133:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   33s   v1.34.0
```

- Minikube had initially warned that the memory allocation (1800MB) was below the recommended limit (1900MB). I then recreated the Minikube cluster with the sufficient memory (2.5GB) and confirmed the status as shown in the picture above.
- I then went ahead and applied the YAML files by running `kubectl apply -f fileName.yml`.



```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl apply -f mongo-pv-pvc.yml
persistentvolume/mongo-pv created
persistentvolumeclaim/mongo-pvc created
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl apply -f mongo-deployment.yml
deployment.apps/mongo-deploy created
Warning: spec.SessionAffinity is ignored for headless services
service/mongo-service created
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl apply -f nodejs-deployment.yml
deployment.apps/nodejs-deploy created
service/nodejs-service created
```

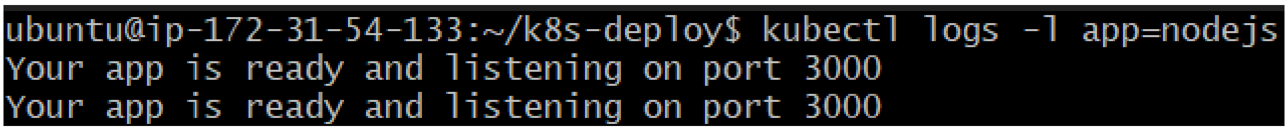
-
- I then ran `kubectl get pods` and `kubect get svc` to check everything was running.



```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
mongo-deploy-796f87b8fc-2xcnb    1/1     Running   0          73s
nodejs-deploy-69fb6fc9d5-sfzv7   1/1     Running   0          61s
nodejs-deploy-69fb6fc9d5-wr8mc   1/1     Running   0          61s
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl get svc
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP          2m4
8s
mongo-service    ClusterIP   None             <none>        27017/TCP        91s
nodejs-service   NodePort    10.103.150.106   <none>        3000:30002/TCP   79s
```

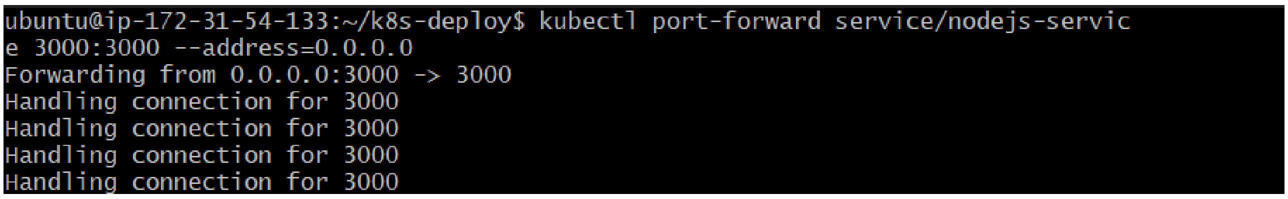
- Both MongoDB and Node.js pods were healthy and responding.

- I then tested the app's internal status



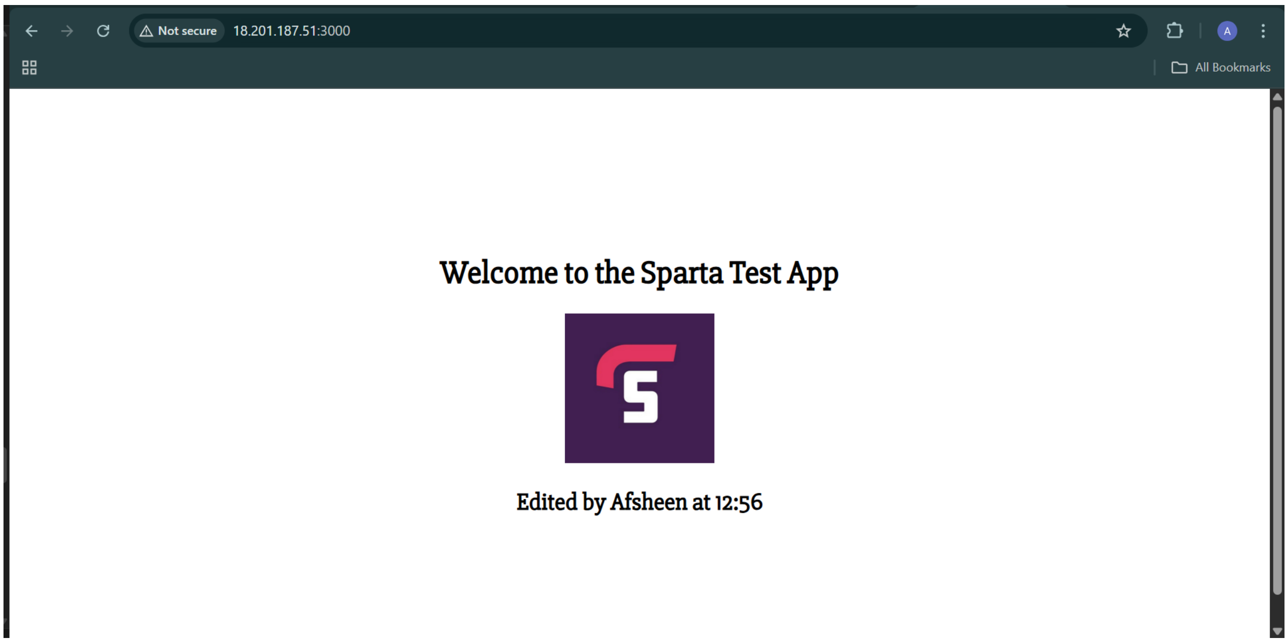
```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl logs -l app=nodejs
Your app is ready and listening on port 3000
Your app is ready and listening on port 3000
```

- Next, I went on to forward the port to 3000 to allow us to see the app running.



```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl port-forward service/nodejs-servic
e 3000:3000 --address=0.0.0.0
Forwarding from 0.0.0.0:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

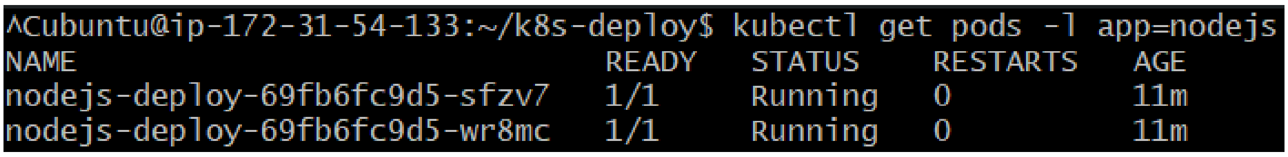
- I then went to the public IP + port 3000 of the instance and found the app working.



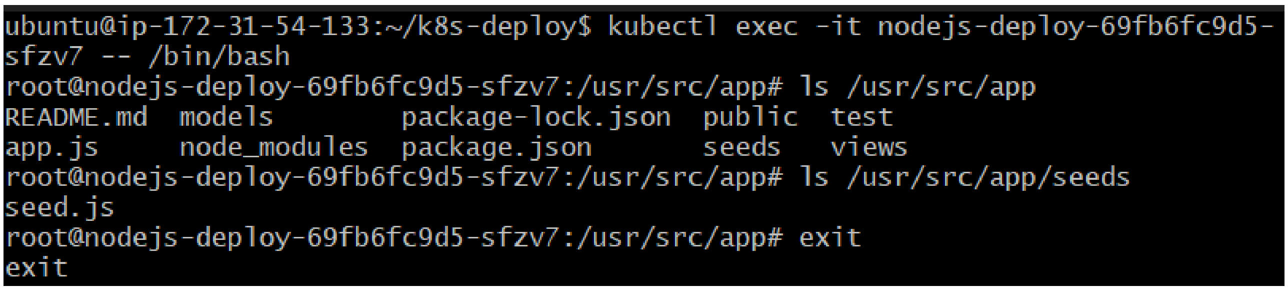
Welcome to the Sparta Test App

Edited by Afsheen at 12:56

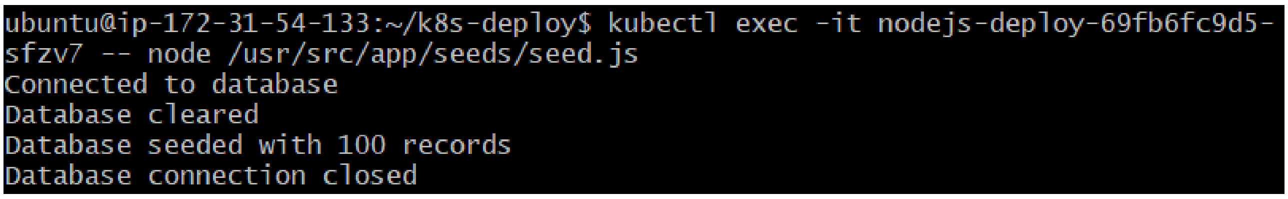
- I then had to seed the database but was not sure what the file path of the `seed.js` was so asked ChatGPT to give me a solution and asked me to run a few commands using one of the Node.js pods where I was then able to seed the database.



```
^Cubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl get pods -l app=nodejs
NAME                          READY   STATUS    RESTARTS   AGE
nodejs-deploy-69fb6fc9d5-sfzv7   1/1     Running   0          11m
nodejs-deploy-69fb6fc9d5-wr8mc   1/1     Running   0          11m
```

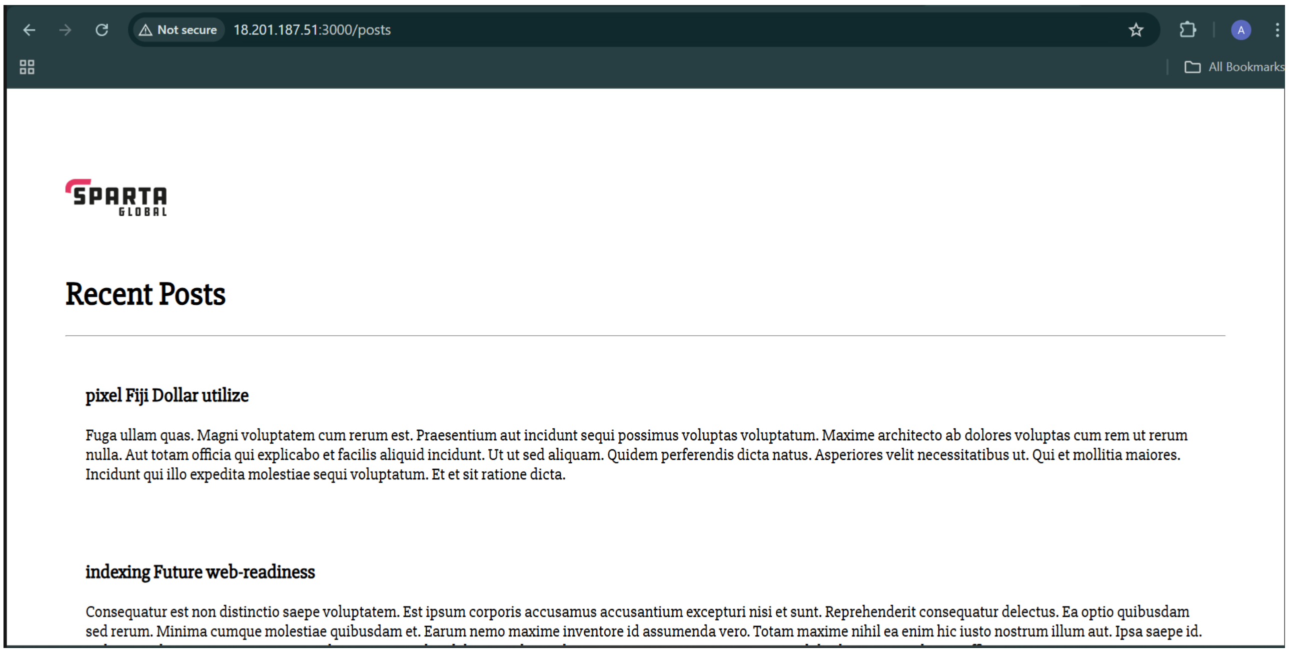


```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl exec -it nodejs-deploy-69fb6fc9d5-
sfzv7 -- /bin/bash
root@nodejs-deploy-69fb6fc9d5-sfzv7:/usr/src/app# ls /usr/src/app
README.md  models        package-lock.json  public  test
app.js     node_modules  package.json       seeds   views
root@nodejs-deploy-69fb6fc9d5-sfzv7:/usr/src/app# ls /usr/src/app/seeds
seed.js
root@nodejs-deploy-69fb6fc9d5-sfzv7:/usr/src/app# exit
exit
```



```
ubuntu@ip-172-31-54-133:~/k8s-deploy$ kubectl exec -it nodejs-deploy-69fb6fc9d5-
sfzv7 -- node /usr/src/app/seeds/seed.js
Connected to database
Database cleared
Database seeded with 100 records
Database connection closed
```

- I then went to the browser and tried the /posts page to check if the database had worked.



Recent Posts

pixel Fiji Dollar utilize

Fuga ullam quas. Magni voluptatem cum rerum est. Praesentium aut incidunt sequi possimus voluptas voluptatum. Maxime architecto ab dolores voluptas cum rem ut rerum nulla. Aut totam officia qui explicabo et facilis aliquid incidunt. Ut ut sed aliquam. Quidem perferendis dicta natus. Asperiores velit necessitatibus ut. Qui et mollitia maiores. Incidunt qui illo expedita molestiae sequi voluptatum. Et et sit ratione dicta.

indexing Future web-readiness

Consequatur est non distinctio saepe voluptatem. Est ipsum corporis accusamus accusantium excepturi nisi et sunt. Reprehenderit consequatur delectus. Ea optio quibusdam sed rerum. Minima cumque molestiae quibusdam et. Earum nemo maxime inventore id assumenda vero. Totam maxime nihil ea enim hic iusto nostrum illum aut. Ipsa saepe id.

What I learnt from this project

During this project, I was able to learn how to deploy an application from end to end using a GitHub repository and SQL seed scripts. Once I started using virtual machines, I was able to using Bash scripts that run through cloud user data, allowing the app and database VMs to configure themselves on startup. I then learned how to containerise my application using Docker, write Dockerfiles, create docker-compose configurations, build and push images to Docker Hub, and deploy them through a single VM provisioning script. I advanced to use Kubernetes to create YAML files to orchestrate the app and database on one machine, with another automated script setting up Kubernetes and applying all configurations. This made me develop my hands-on experience for platforms I had never used before and made me confident on using them as I learnt more about why we use them and what they do along the way.