Deployment of a demo app in AWS.

Step 1: Pull mongo and mongo-express.

Commands:

$ sudo docker pull mongo

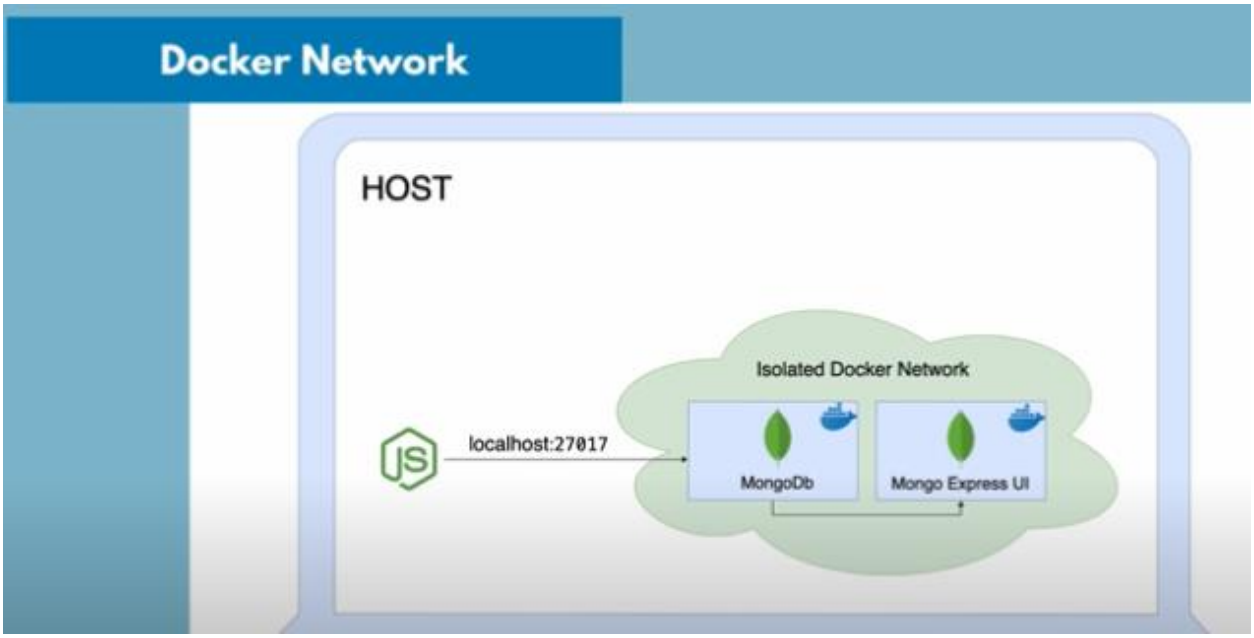
$ sudo docker pull mongo-express

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
ea362f368469: Downloading  14.12MB/28.57MB
ea362f368469: Downloading  14.72MB/28.57MB
ea362f368469: Pull complete
ecab26900ceb: Pull complete
1847fcb70562: Pull complete
a7de23811c0d: Pull complete
29dd51833fb9: Pull complete
5eccd2be8afb: Pull complete
cd8a8cd6879f: Pull complete
e6ca3abc397d: Downloading  131.8MB/210.4MB
```

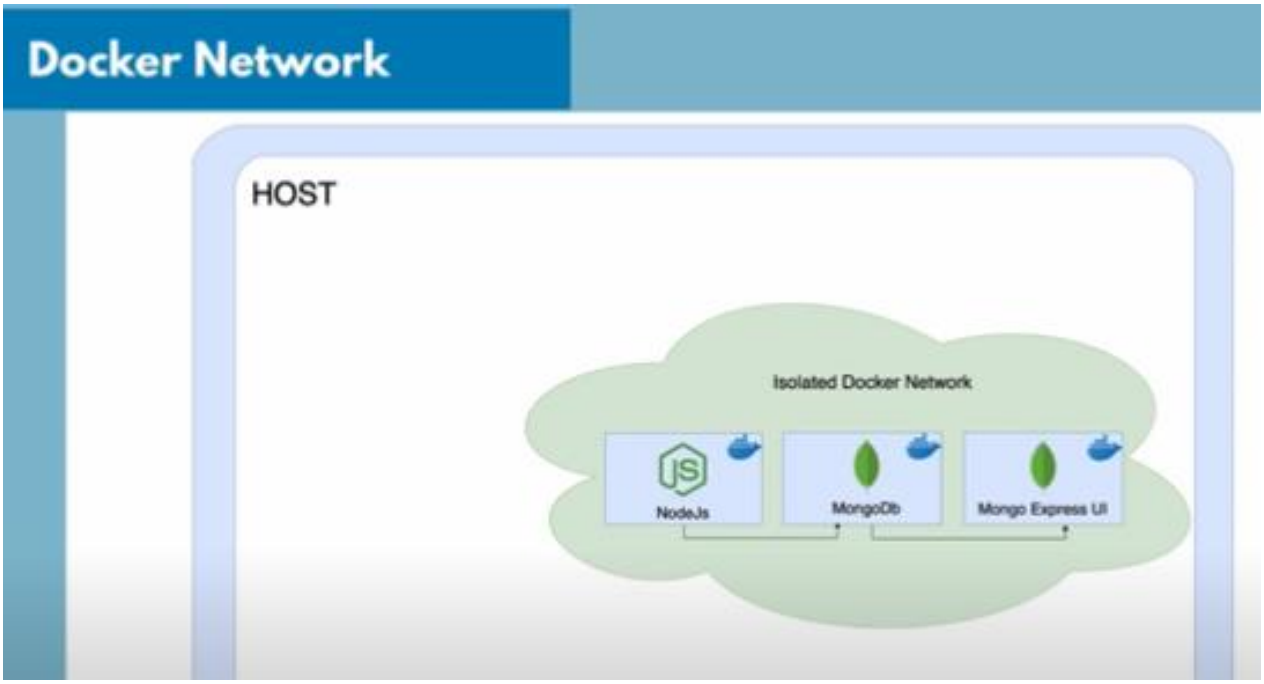
```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker images
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
mongo           latest    ee13a1eacac9   2 weeks ago    696MB
redis           latest    7614ae9453d1   4 weeks ago    113MB
mongo-express   latest    2d2fb2cabc8f   3 months ago   136MB
hello-world     latest    feb5d9fea6a5   4 months ago   13.3kB
```

Step 2: Setup a network.

Default setup.



what we need:



Create a network for mongo, mongo-express.

$ sudo docker network create mongo-network

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
a15d71d51cab   bridge    bridge    local
7f7d8b23870d   host      host      local
3e7e6bc2ccef   none      null      local
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker network create mongo-network
3c9c3ba09ff7d108dc5d78cde1f5276e13f5c3c2a38cf04484f290767824a12b
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
a15d71d51cab   bridge          bridge    local
7f7d8b23870d   host            host      local
3c9c3ba09ff7   mongo-network   bridge    local
3e7e6bc2ccef   none            null      local
```

Step 3: Run the container for mongodb.

$ sudo docker run -p 27017:27017 -d -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INIT_ROOT_PASSWORD=password --name mongodb --net mongo-network mongo

Check the container.

$sudo docker log <container-id>

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker run -d \
> -p 27017:27017 \
> -e MONGO_INITDB_ROOT_USERNAME=user \
> -e MONGO_INITDB_ROOT_PASSWORD=user \
> --name mongodb2 \
> --net mongo-network \
> mongo
9099cd58c77d624af89f67e39d9f188d4bf975be1bc8836629643b1bc6c53e9c
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker logs 9099cd58c77d624af89f67e39d9f188d4bf975be1bc883662964
3b1bc6c53e9c
about to fork child process, waiting until server is ready for co
nnections.
forked process: 31

{"t":{"$date":"2022-01-23T16:07:00.072+00:00"},"s":"I",  "c":"CON
TROL",  "id":20698,   "ctx":"-","msg":"***** SERVER RESTARTED ***
**"}
{"t":{"$date":"2022-01-23T16:07:00.086+00:00"},"s":"I",  "c":"NET
WORK",  "id":4915701, "ctx":"-","msg":"Initialized wire specifica
tion","attr":{"spec":{"incomingExternalClient":{"minWireVersion":
```

Step 4: Run container for mongo-express.

$sudo docker run -d \

> -p 8081:8081 \

> -e ME_CONFIG_MONGODB_ADMINUSERNAME=user \

> -e ME_CONFIG_MONGODB_ADMINPASSWORD=user \

> --net mongo-network \

> --name mongo-express \

> -e ME_CONFIG_MONGODB_SERVER=mongodb2 \

> mongo-express

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker run -d \
> -p 8081:8081 \
> -e ME_CONFIG_MONGODB_ADMINUSERNAME=user \
> -e ME_CONFIG_MONGODB_ADMINPASSWORD=user \
> --net mongo-network \
> --name mongo-express \
> -e ME_CONFIG_MONGODB_SERVER=mongodb2 \
> mongo-express
```

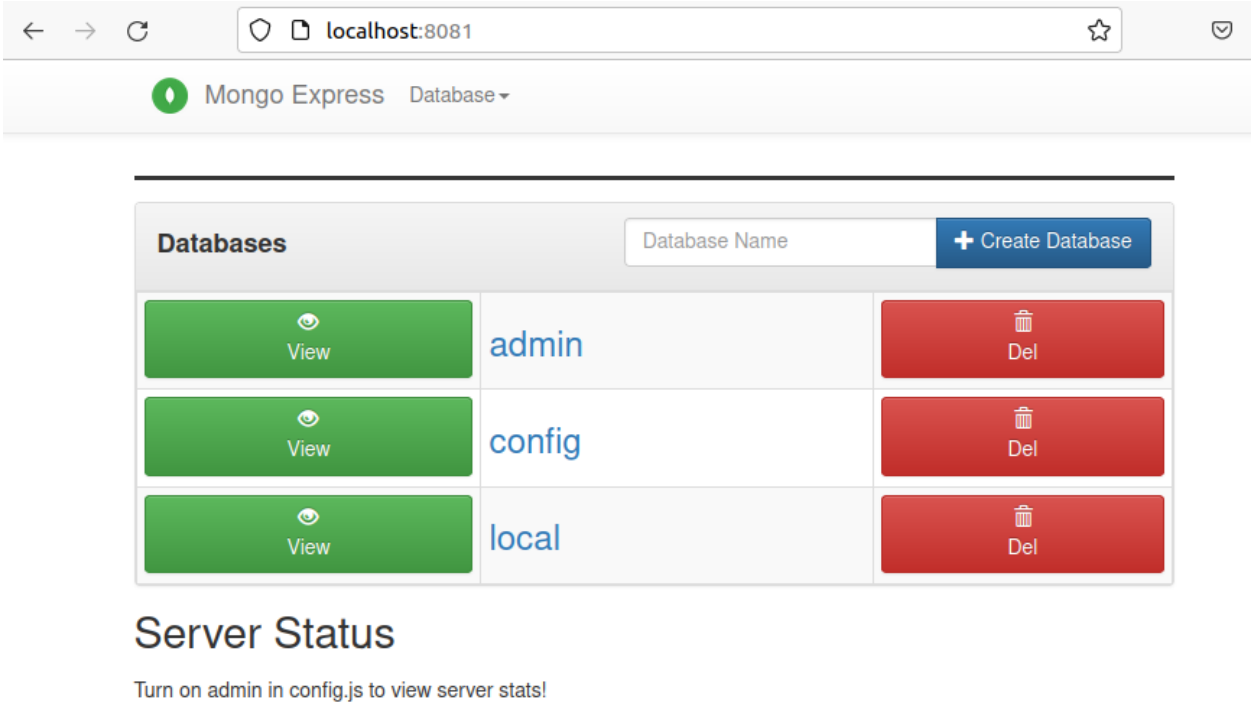
Check the container.

$sudo docker logs <CID>

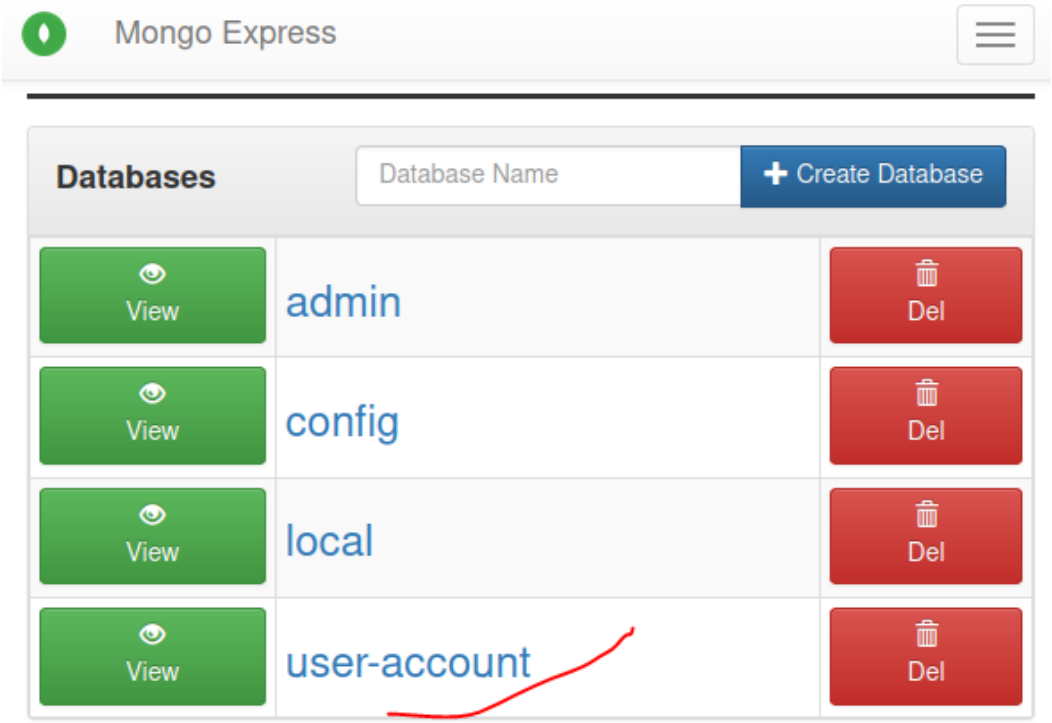
```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker logs 7041852cef7953a23f241e7e28ee4ed8d50e950bfc3fcbf7868d
f61083ed5dbb
Welcome to mongo-express
------------------------


(node:7) [MONGODB DRIVER] Warning: Current Server Discovery and M
onitoring engine is deprecated, and will be removed in a future v
ersion. To use the new Server Discover and Monitoring engine, pas
s option { useUnifiedTopology: true } to the MongoClient construc
tor.
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 cha
nge this in your config.js!
```

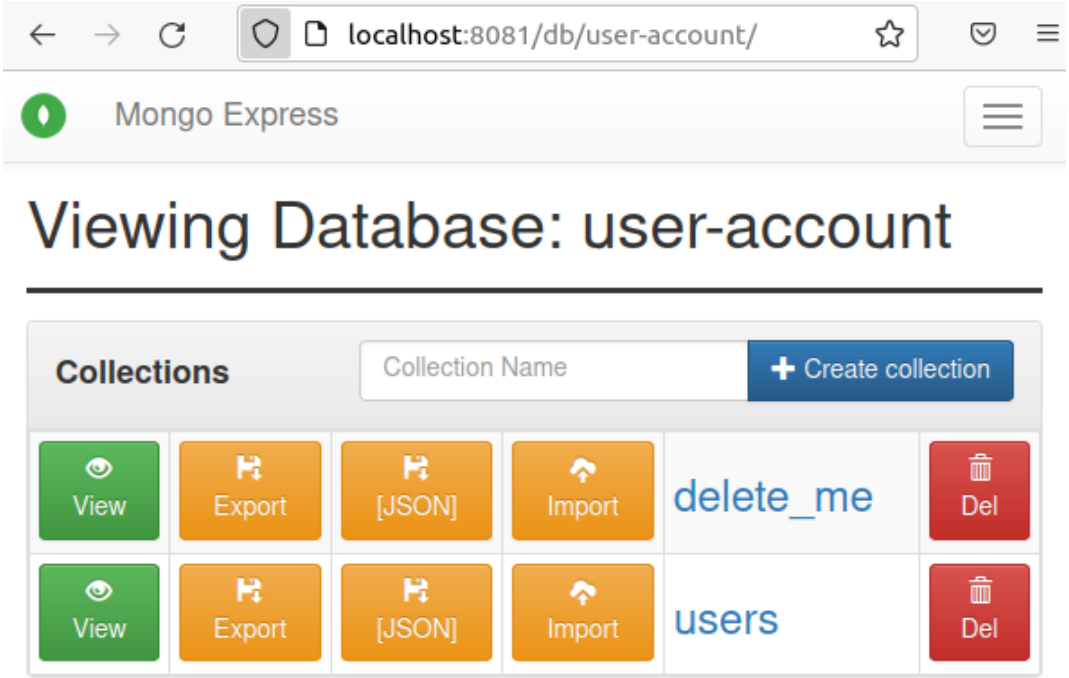
Step 5: Check and open mongo-express in browser.



Create a database in mongo-express.



Create table in database user-account named users.



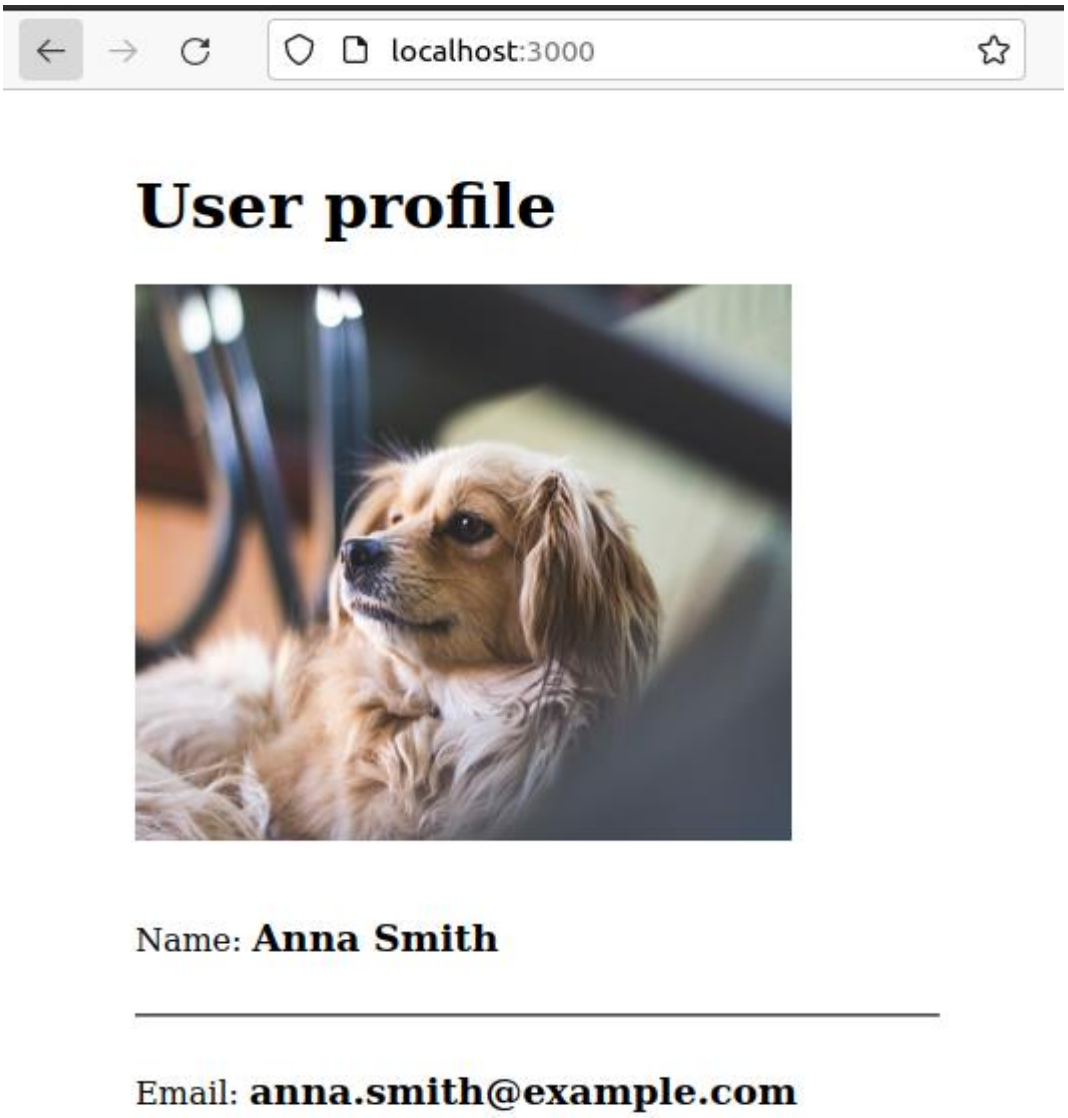
Step 6: Load application.

$npm install

$node server.js

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app/app$
sudo node server.js
app listening on port 3000!
```

Application is running successfully.



Edit the information.

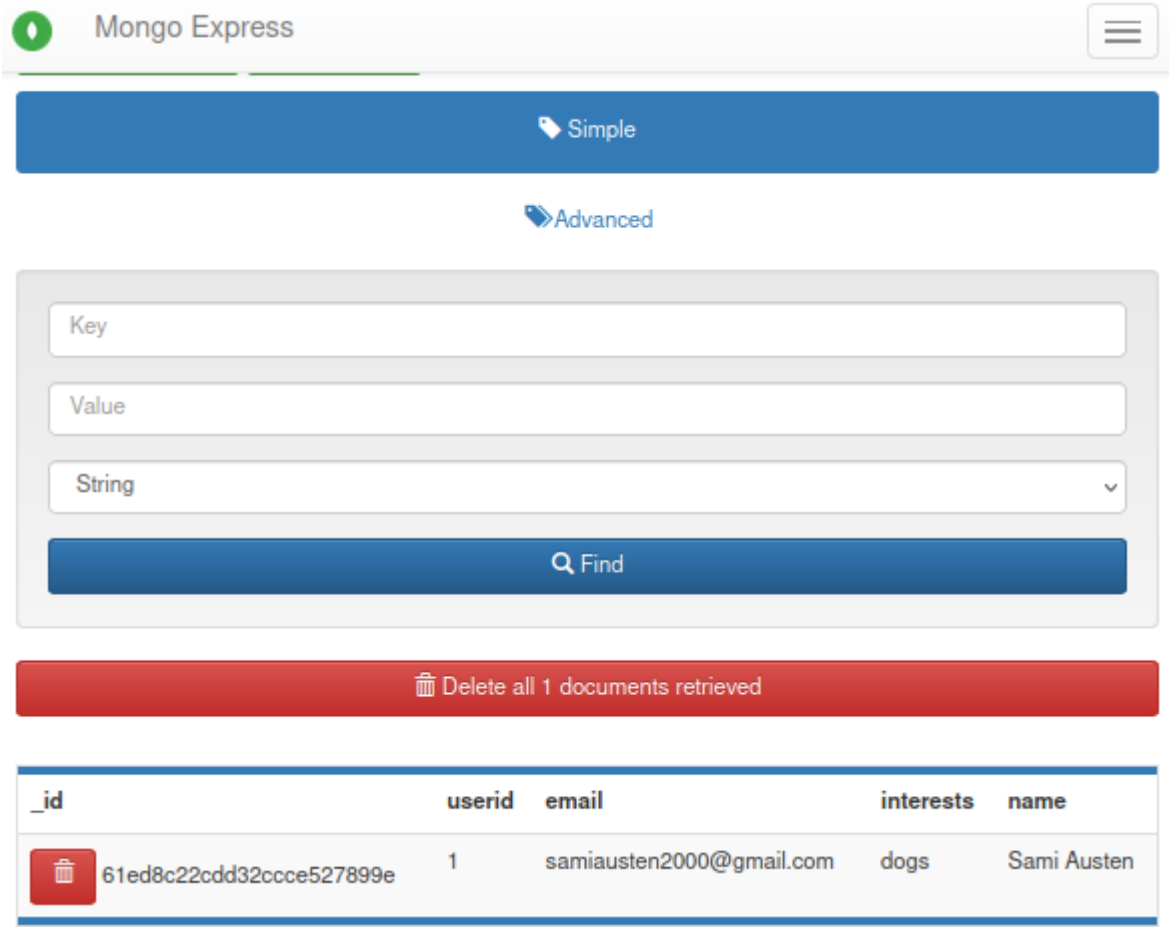
User profile

Name: **Sami Austen**

Email: **samiausten2000@gmail.com**

Interests: **dogs**

Observe the mongo-express, the table is updated there.

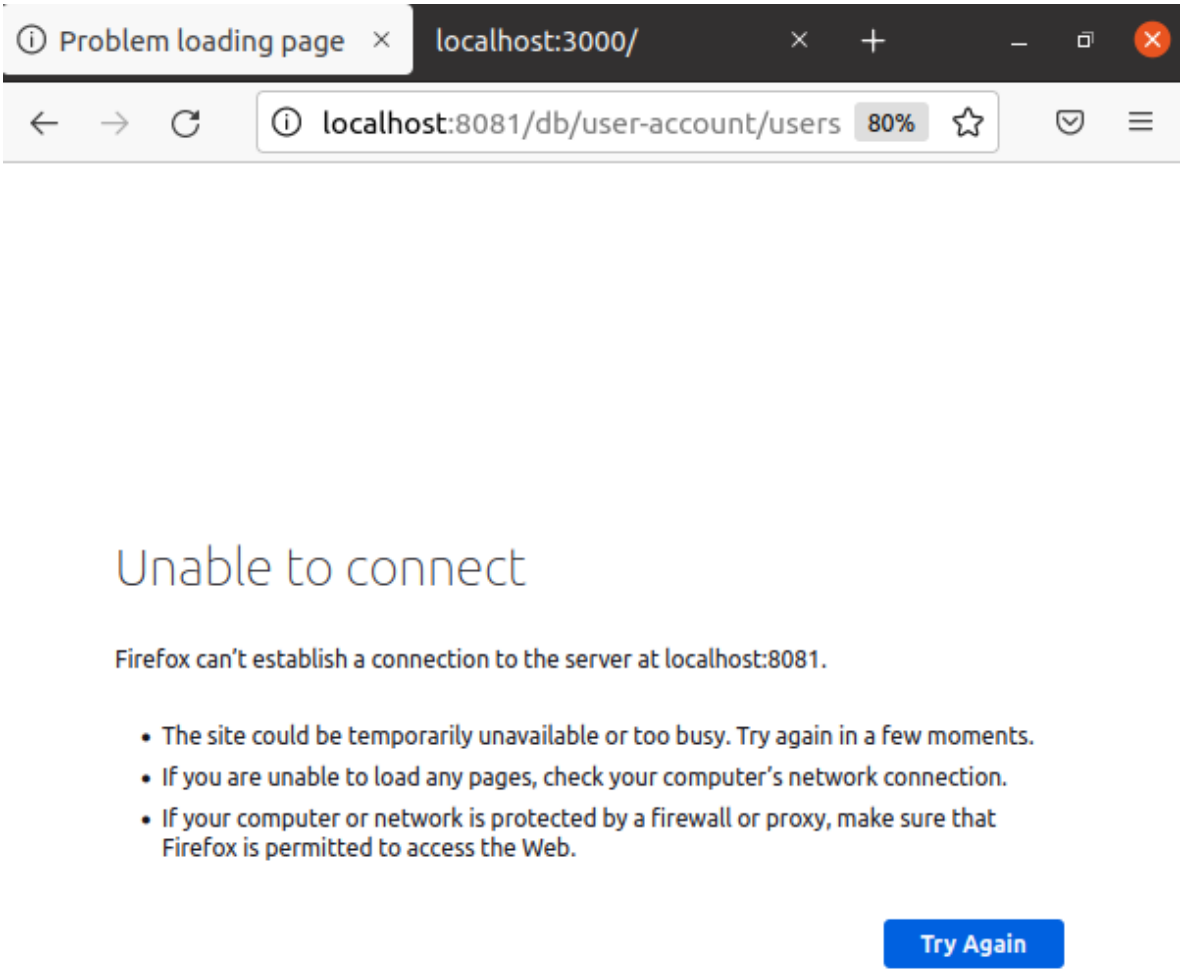


This proves that the application is connected to database and functioning correctly on deployment.

Now, deploying application with docker-compose file.

Step 1: Check for running containers. Here, no containers are up currently.

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
NAMES
```



Step 2: Create a docker compose file.

```
  GNU nano 4.8            docker-compose.yaml
version: '3'
services:
  # my-app:
  # image: ${docker-registry}/my-app:1.0
  # ports:
  # - 3000:3000
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=user
      - MONGO_INITDB_ROOT_PASSWORD=user
    volumes:
      - mongo-data:/data/db
  mongo-express:
    image: mongo-express
    restart: always # fixes MongoNetworkError when mongodb is no>
    ports:
      - 8080:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=user
      - ME_CONFIG_MONGODB_ADMINPASSWORD=user
      - ME_CONFIG_MONGODB_SERVER=mongodb
volumes:
```

Step 3: Run docker-compose command.

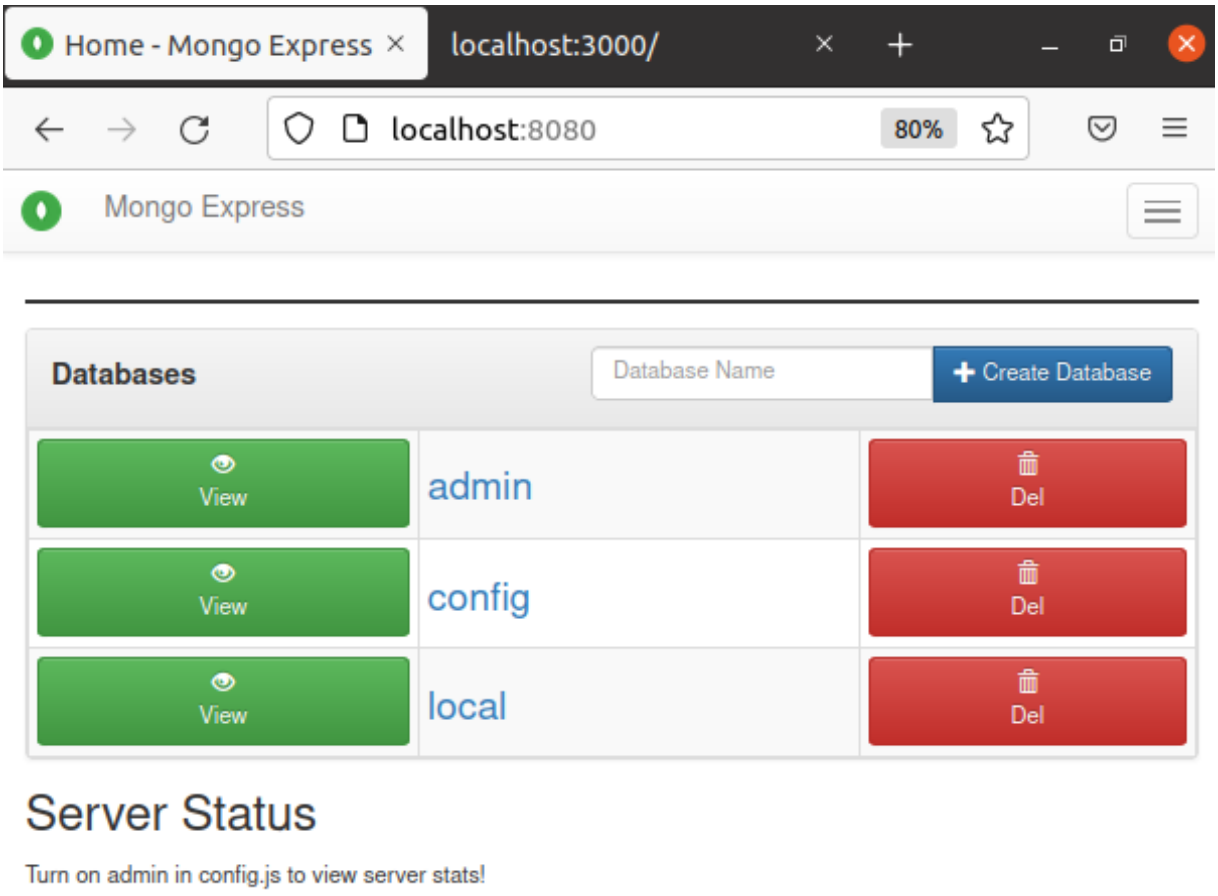
$sudo docker-compose -f <file.yaml> up

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker-compose -f docker-compose.yaml up
Creating network "techworld-js-docker-demo-app_default" with the
default driver
Creating volume "techworld-js-docker-demo-app_mongo-data" with lo
cal driver
Creating techworld-js-docker-demo-app_mongo-express_1 ... done
Creating techworld-js-docker-demo-app_mongodb_1       ... done
Attaching to techworld-js-docker-demo-app_mongo-express_1, techwo
rld-js-docker-demo-app_mongodb_1
mongodb_1        | about to fork child process, waiting until ser
ver is ready for connections.
mongodb_1        | forked process: 29
mongodb_1        |
mongodb_1        | {"t":{"$date":"2022-01-23T18:04:02.498+00:00"}
,"s":"I",  "c":"CONTROL",  "id":20698,   "ctx":"-","msg":"***** S
ERVER RESTARTED *****"}
```

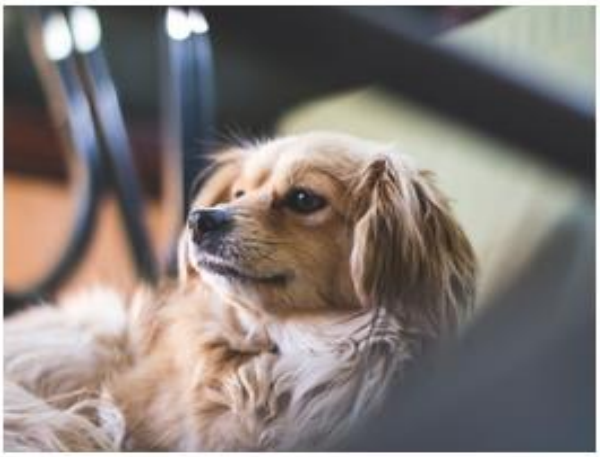
Step 4: Check for running containers. Two containers for mongo and mongo-express are created. Docker created default network for these 2 containers.

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED
       STATUS         PORTS
     NAMES
c387c28ed96b   mongo-express   "tini -- /docker-ent…"   2 minutes
 ago   Up 2 minutes   0.0.0.0:8080->8081/tcp, :::8080->8081/tcp
     techworld-js-docker-demo-app_mongo-express_1
f674ee3a10fd   mongo           "docker-entrypoint.s…"   2 minutes
 ago   Up 2 minutes   0.0.0.0:27017->27017/tcp, :::27017->27017/t
cp   techworld-js-docker-demo-app_mongodb_1
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker network ls
NETWORK ID     NAME                                      DRIVER    S
COPE
a15d71d51cab   bridge                                    bridge    l
ocal
7f7d8b23870d   host                                      host      l
ocal
3c9c3ba09ff7   mongo-network                             bridge    l
ocal
3e7e6bc2ccef   none                                      null      l
ocal
fe85f0e94eb9   techworld-js-docker-demo-app_default      bridge    l
ocal
```

Step 5: Check if mongo-express is running properly.



Step 6: Now, run the application and edit profile again.



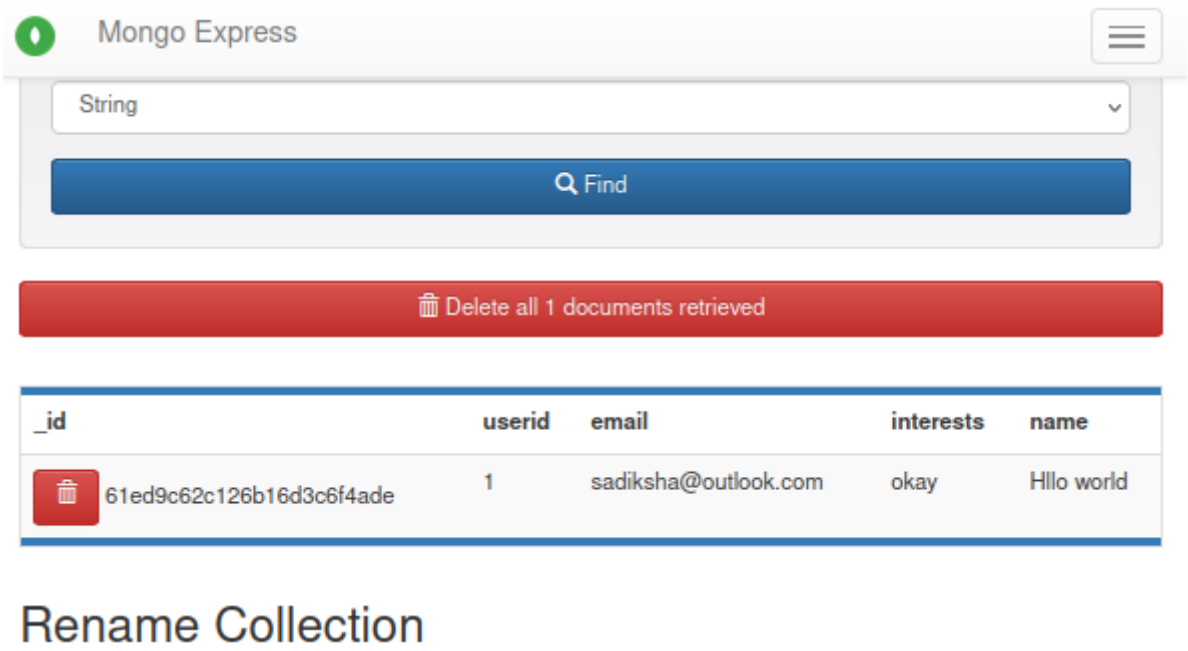
Name: **Hllo world**

Email: **sadiksha@outlook.com**

Interests: **okay**

Edit Profile

The application is running properly.



Step 7: To stop all the containers at once using docker-compose.

$ sudo docker-compose -f docker-compose.yaml down

It stops all containers at once specified in yaml file and also removes the created docker network.

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker-compose -f docker-compose.yaml down
Stopping techworld-js-docker-demo-app_mongo-express_1 ... done
Stopping techworld-js-docker-demo-app_mongodb_1       ... done
Removing techworld-js-docker-demo-app_mongo-express_1 ... done
Removing techworld-js-docker-demo-app_mongodb_1       ... done
Removing network techworld-js-docker-demo-app_default
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
NAMES
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
a15d71d51cab   bridge          bridge    local
7f7d8b23870d   host            host      local
3c9c3ba09ff7   mongo-network   bridge    local
3e7e6bc2ccef   none            null      local
```

Creating own docker image using dockerfile.

Step 1: Create a docker file.

```
  GNU nano 4.8                    Dockerfile
FROM node:13-alpine

ENV MONGO_DB_USERNAME=user \
    MONGO_DB_PWD=user

RUN mkdir -p /home/app

COPY ./app /home/app

# set default dir so that next commands executes in /home/app dir
WORKDIR /home/app

# will execute npm install in /home/app because of WORKDIR
RUN npm install

# no need for /home/app/server.js because of WORKDIR
CMD ["node", "server.js"]
```

Step 2: Now, build the image from dockerfile.

$ sudo docker build -t sadiksha-app:1.0

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker build -t sadiksha-app:1.0 .
[sudo] password for sadiksha:
Sending build context to Docker daemon  17.42MB
Step 1/7 : FROM node:13-alpine
13-alpine: Pulling from library/node
cbdbe7a5bc2a: Pull complete
780514bed1ad: Pull complete
5d74fb112a7d: Pull complete
4b9536424fa1: Pull complete
Digest: sha256:527c70f74817f6f6b5853588c28de33459178ab72421f1fb7b
63a281ab670258
Status: Downloaded newer image for node:13-alpine
 ---> 8216bf4583a5
Step 2/7 : ENV MONGO_DB_USERNAME=user     MONGO_DB_PWD=user
 ---> Running in 25dbe352c38d
Removing intermediate container 25dbe352c38d
 ---> fcfc5b726ecd
Step 3/7 : RUN mkdir -p /home/app
 ---> Running in 85efa5c401fa
```

Step 3: Check docker images.

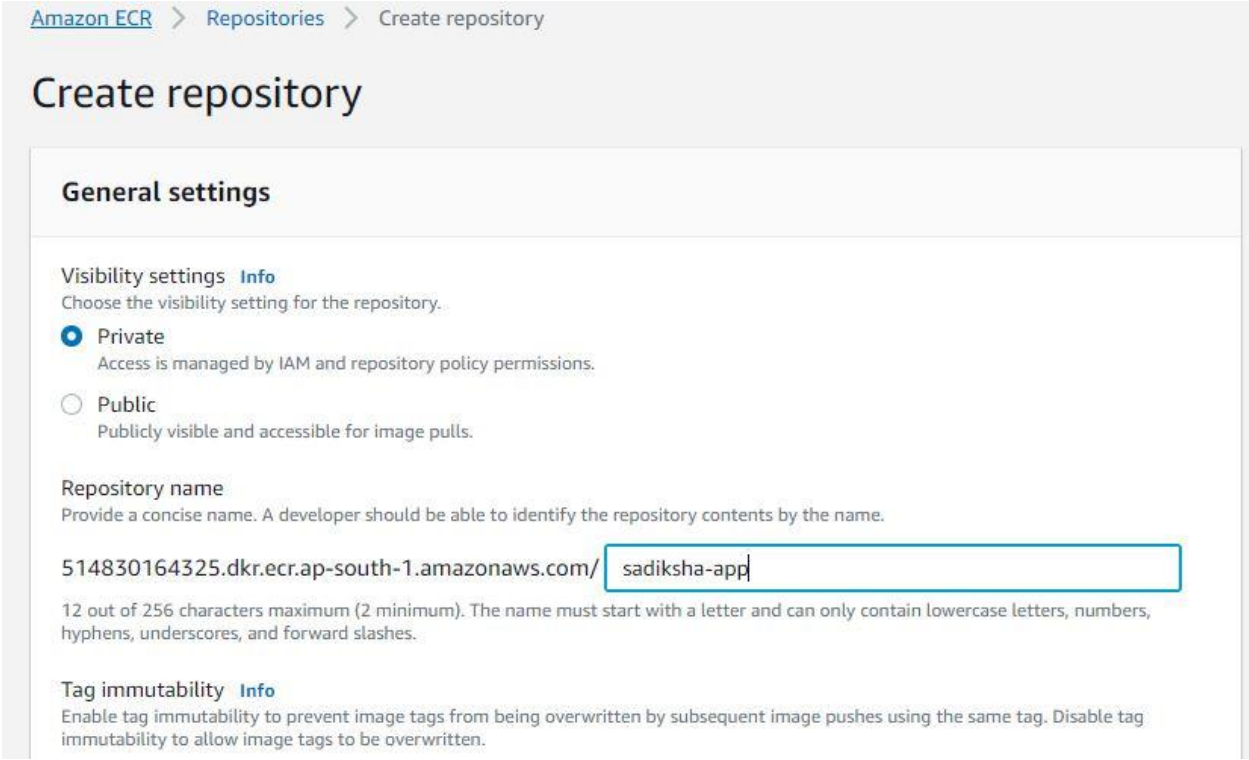
```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app$ sudo
 docker images
REPOSITORY      TAG         IMAGE ID       CREATED              S
IZE
sadiksha-app    1.0         69404aa37383   About a minute ago   1
31MB
mongo           latest      ee13a1eacac9   2 weeks ago          6
96MB
redis           latest      7614ae9453d1   4 weeks ago          1
13MB
mongo-express   latest      2d2fb2cabc8f   3 months ago         1
36MB
hello-world     latest      feb5d9fea6a5   4 months ago         1
3.3kB
node            13-alpine   8216bf4583a5   21 months ago        1
14MB
```

Creating a private repository in AWS and pushing docker images.

Step 1: Go to AWS account and log in. Go to Services>ECR.

Create a repository on AWS.

We create repository per image in AWS. We save different tags/versions of same image in AWS repository.



Step 2: Install AWS CLI and configure credentials on it.

$ sudo apt-get install unzip -y

$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

$ unzip awscliv2.zip

$ sudo ./aws/install

```
sadiksha@sadiksha-VirtualBox:~$ aws --version
aws-cli/2.4.13 Python/3.8.8 Linux/5.11.0-46-generic exe/x86_64.ub
untu.20 prompt/off
```

Step 3: Configure the credentials.

```
sadiksha@sadiksha-VirtualBox:~$ sudo aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
C/sPH
Default region name [None]: Mumbai
Default output format [None]:
```

```
sadiksha@sadiksha-VirtualBox:~$ sudo aws configure
AWS Access Key ID [****************HD7A]:
AWS Secret Access Key [****************/sPH]:
Default region name [Mumbai]: ap-south-1
Default output format [None]:
sadiksha@sadiksha-VirtualBox:~$ sudo aws ec2 describe-instances
{
    "Reservations": []
}
```

Step 4: Log into the private repository.

Select the repository and click 'view push commands'.

Push commands for sadiksha-app

macOS / Linux | Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry.

   Use the AWS CLI:

```
aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin
```

   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

```
docker build -t sadiksha-app .
```

Close

Step 5: Copy and paste that command on your local machine's terminal.

$ aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 514830164325.dkr.ecr.ap-south-1.amazonaws.com

```
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app/app$ sudo aws configure
[sudo] password for sadiksha:
AWS Access Key ID [****************HD7A]: AKIAXPXSE4FS2T7UHD7A
AWS Secret Access Key [***************/sPH]: 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
Default region name [ap-south-1]: ap-south-1
Default output format [None]:
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app/app$ sudo aws configure
AWS Access Key ID [****************HD7A]: ^Z
[1]+  Stopped                 sudo aws configure
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app/app$ aws ecr get-login-password --region ap-south-1 | docker login --username AWS
--password-stdin 514830164325.dkr.ecr.ap-south-1.amazonaws.com

Unable to locate credentials. You can configure credentials by running "aws configure".
Error: Cannot perform an interactive login from a non TTY device
sadiksha@sadiksha-VirtualBox:~/techworld-js-docker-demo-app/app$ sudo aws ecr get-login-password --region ap-south-1 | docker login --username
 AWS --password-stdin 514830164325.dkr.ecr.ap-south-1.amazonaws.com
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.s
ock/v1.24/auth": dial unix /var/run/docker.sock: connect: permission denied
```

Error.