VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

AAT PROJECT REPORT
on

Web app containerising and deploying

Submitted by

Arka Sinha(1BM19CS024)
B.V Vineeth(1BM19CS033)
Derek Stanley Kannathe(1BM19CS045)

Under the Guidance of

Lakshmi Neelima
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Oct-2022 to Feb-2023

B. M. S. College of Engineering

Bull Temple Road, Bangalore 560019

(Affiliated to Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled "**Disease Prediction**" carried out by **Arka Sinha (1BM19CS024), B.V Vineeth (1BM19CS033), Derek Stanley Kannathe (1BM19CS045)** who are bonafide students of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraya Technological University, Belgaum during the year 2022-2023. The project report has been approved as it satisfies the academic requirements in respect of the **AAT Cloud Computing (21CS7PECCT)** work prescribed for the said degree.

Signature of the Guide
Lakshmi Neelima
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr Jyothi S Nayak
Associate Prof. & Head, Dept. of CSE

Table of Contents

1. Introduction

This is a cloud computing project carried out under the Cloud Computing AAT with a team of three students to build and deploy the project. This project is based on the technologies of cloud computing, with the help of which we will be carrying out this project.
We have selected a MERN stack blog application where users can register, log in and be able to write articles based on their interests. This application is going to be containerised with a free application called Docker and then be deployed on a platform which supports docker and host it.

1.1 Introduction to Cloud Computing

It allows users to access and use computing resources, such as data storage, processing power, and applications, over the internet on an as-needed basis. This means that users don't have to manage and maintain these resources themselves, and can instead access them from a cloud provider. Cloud computing can be a cost-effective and flexible solution for businesses, as it allows them to access the resources they need without having to invest in and maintain their own infrastructure. It can also be convenient for individuals, as it allows them to access and use resources and applications from any device with an internet connection. Cloud computing systems are often distributed across multiple data centers, which are facilities that house and maintain large numbers of servers and other computing resources.

Due to these features of cloud computing, it has been widely accepted by organisations of different levels and thus justifying our use of this technology for the problem statement provided.

1.2 Problem Statement

1. Make a group of 3 students.
2. Each group must implement the following.

a. Create a web application with a backend of your choice.
b. Containarise the application and host it on the cloud.

3. At least two clients should access the application from the cloud.

2. Dockers and Kubernetes

Docker:

Docker is a platform that allows developers to package and distribute their applications as lightweight, standalone containers that can be easily run on any device. It uses OS-level virtualization to create these containers, which are self-contained environments that include everything the application needs to run, such as code, libraries, and system tools. Docker offers both free and paid versions of its platform, and the software that runs the containers is called Docker Engine. Docker was first released in 2013 and is developed by Docker, Inc.

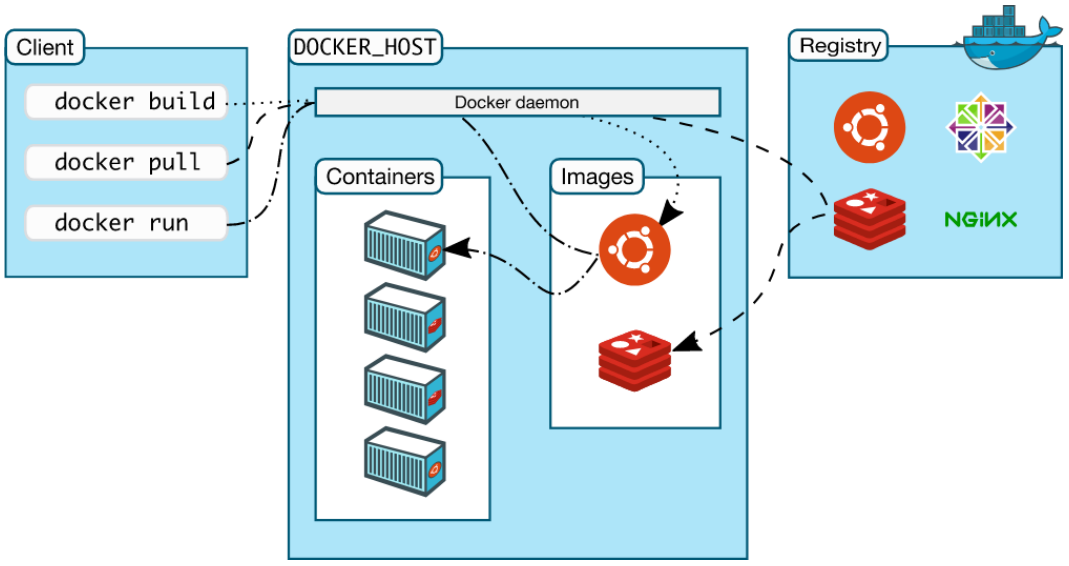


Fig 2.1

- The Docker daemon is a background process that listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. It can also communicate with other daemons to manage Docker services.
- The Docker client is the primary tool that users interact with to manage Docker. It sends commands to the Docker daemon using the Docker API, which carries out the requested actions. The Docker client can communicate with multiple daemons. Docker Desktop is an application that includes the Docker daemon, client, Compose,

Content Trust, Kubernetes, and Credential Helper, and allows users to build and share containerized applications and microservices on their local machine.

- A Docker registry is a storage location for Docker images.
- Docker Hub is a public registry that anyone can use, and the Docker client is configured to search for images on Docker Hub by default. It is also possible to set up a private registry. Docker objects include images, containers, networks, volumes, plugins, and other elements used in the Docker ecosystem.
- When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.
    - An image is a read-only template that contains instructions for creating a Docker container.
    - A container is a runnable instance of an image, and can be created, started, stopped, moved, or deleted using the Docker API or CLI. It can be connected to one or more networks, have storage attached to it, and even be used to create a new image.

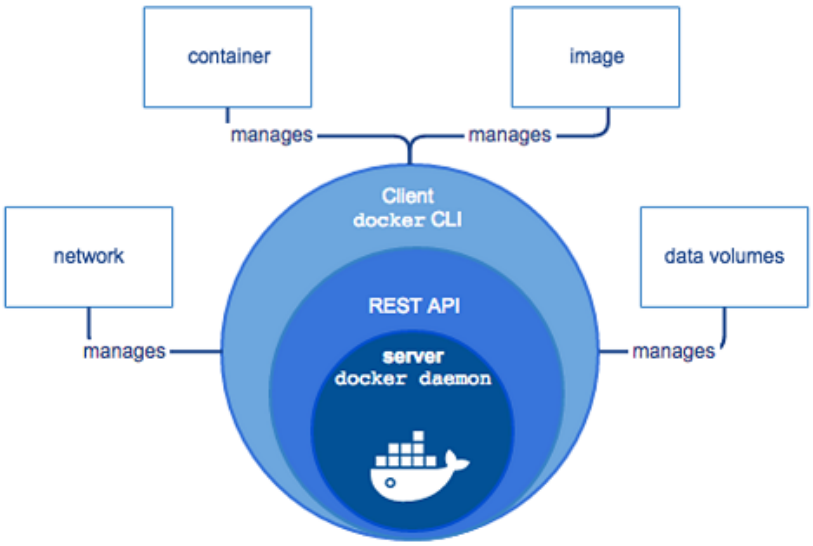


Fig 2.2

Kubernetes:

Kubernetes is a free and open-source system for automating the deployment, scaling, and management of containerized applications. It was originally developed by Google, but is now maintained by the Cloud Native Computing Foundation. Kubernetes allows developers to

deploy their applications in containers and manage them using a set of tools and APIs. The name "Kubernetes" comes from the Greek word for "helmsman" or "pilot," which reflects its role in steering and guiding the deployment and operation of containerized applications.

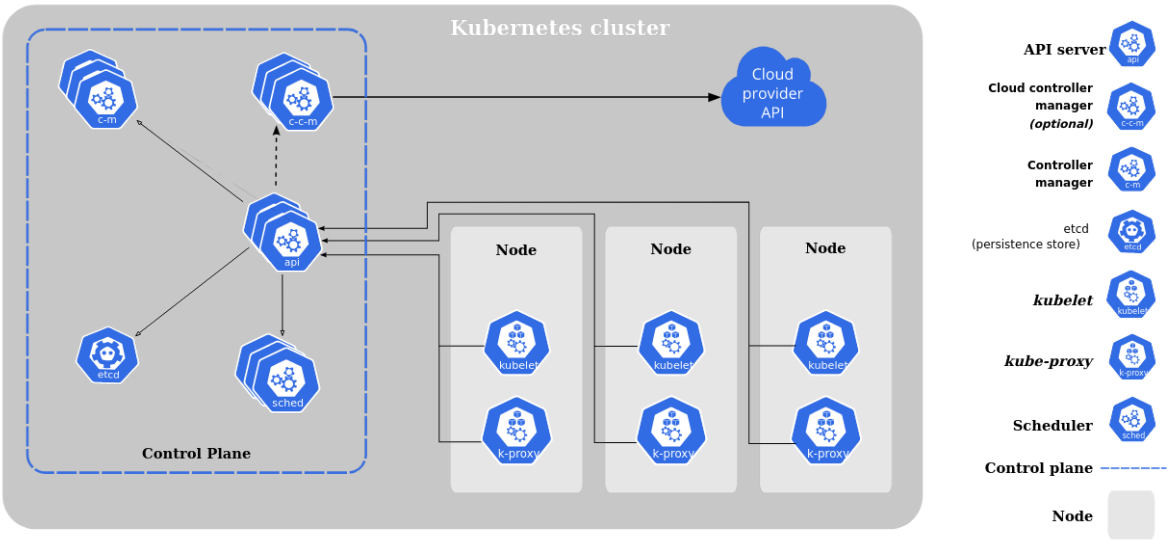


Fig 2.3

- **The control plane** hosts the components used to manage the Kubernetes cluster.
- **Worker nodes** can be virtual machines (VMs) or physical machines. A node hosts pods, which run one or more containers.
- **Pods**—pods are the smallest unit provided by Kubernetes to manage containerized workloads. A pod typically includes several containers, which together form a functional unit or microservice.
- **Persistent storage**—local storage on Kubernetes nodes is ephemeral and is deleted when a pod shuts down. This can make it difficult to run stateful applications. Kubernetes provides the Persistent Volumes (PV) mechanism, allowing containerized applications to store data beyond the lifetime of a pod or node.

3. Cloud Solutions for Applications Hosting

1. Google Cloud Platform: Google offers a free tier with a limited amount of resources, including Compute Engine, Cloud Functions, and Firebase.

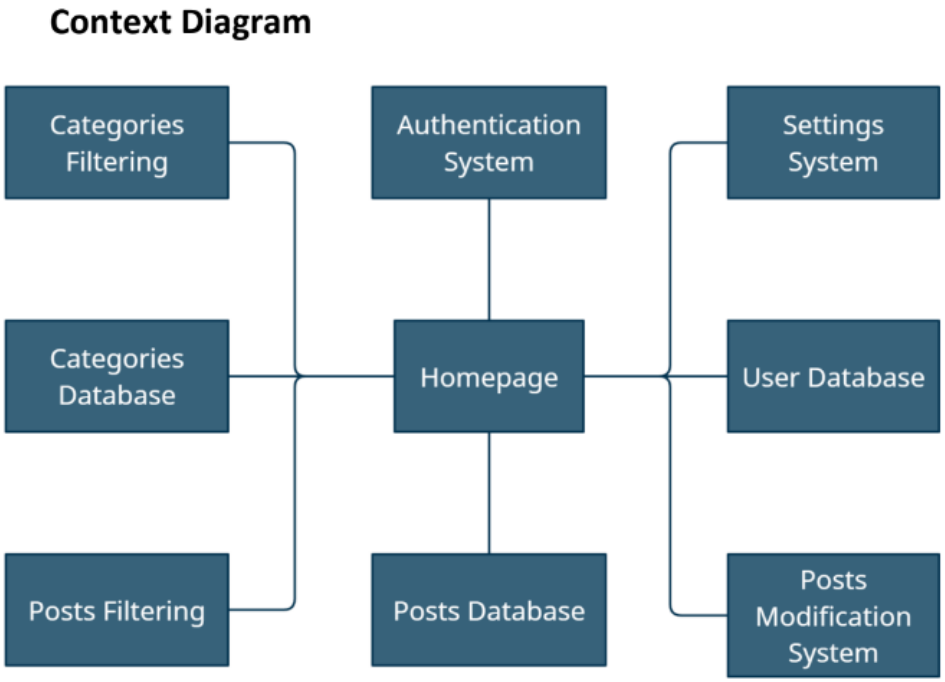
2. Amazon Web Services (AWS): AWS offers a free tier with a limited amount of resources, including EC2, S3, and Lambda.
3. Microsoft Azure: Microsoft Azure offers a free tier with a limited amount of resources, including Virtual Machines, Web Apps, and Functions.
4. IBM Cloud: IBM Cloud offers a free tier with a limited amount of resources, including Virtual Servers, Cloud Functions, and Object Storage.

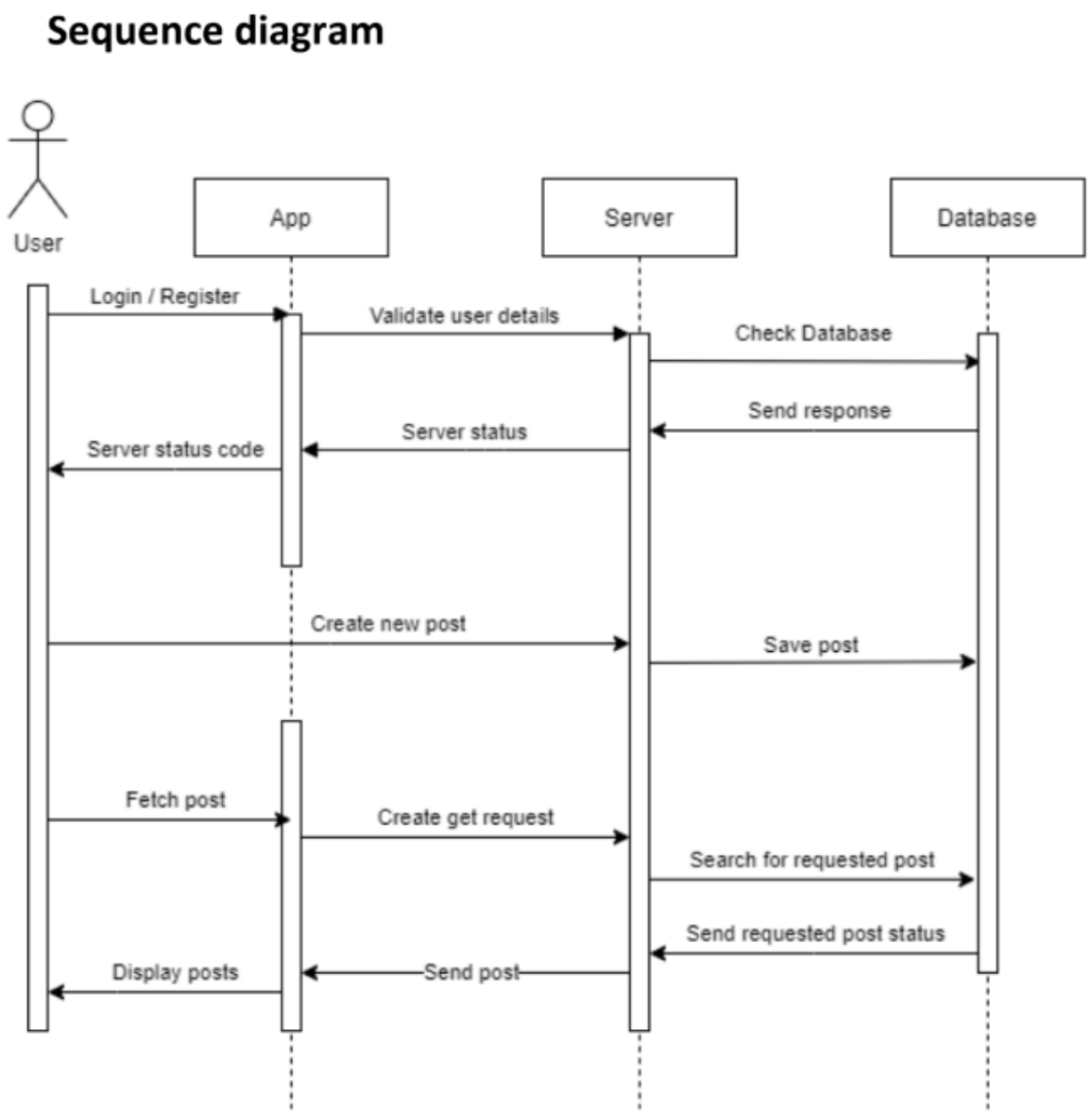
Suitable with docker containers:

1. DigitalOcean: With a student account one can free access to $200 credits for a period of 2 months which can be used to set up droplets for application.
2. Netlify: has a free pricing plan with limited features but can only be used for serverless applications deployment.
3. Heroku: student account with the GitHub student developer pack provides you with $13 credits which you are able to use and deploy your containers.
4. AWS: With free credits, in a student account, one can deploy docker on AWS ECS service

4. Detailed Design

4.1 Application Design





4.2 Steps to Host Application

First, check the application if the localhost is working with a proper connection to the database.

Dockerizing (Containerising):

1. Create a docker file for both the client and server side in the local repository with their respective environments, ports and environment variables.
2. Now after setting up the docker files we can either go by containerising both differently or use a docker-compose command to set up both. We have set up both differently.

3. Now run the build command `docker 'build [OPTIONS] PATH | URL | -'` for both the client and server with their tags. This will build the docker images for both.
4. After getting the images, we then run the command '`docker run -p port:port -e ENV_VAR='VAR_VAL' image_name:tag'`
5. This helps us create the containers for both client and server and run them simultaneously.
6. The Docker Desktop gives the URL for the client side to view the localhost client.
7. Now for easier access to the images we upload the images to Docker Hub, a docker registry from where if the repository is public then we can directly pull the images to any system with docker installed.

Deploy (DigitalOcean):

1. We log in with the student account and get $200 credits free for 60 days from registering if the card used is acceptable.
2. We then create a project on the platform. There then we create a droplet which will serve as the hosting machine for our application. One needs to decide on the base image(i.e OS and apps) and the resources to be allocated to the platform and then create the droplet with the help of an ssh key or password for protection.
3. Now one can access the droplet with the console provided or with the ssh key in their terminal by using the command '`ssh -i /path/to/keys username@ipaddress'`.
4. Now one can directly download docker from the marketplace or add through docker documentation commands, we did through the marketplace.
5. Now pull the images from the docker hub with '`docker pull <image_name>:tag'`.
6. Now set the nginx conf files such that it may redirect the HTTP port requests to the specified port, and run the images. With:
    a. Client: '` docker run -p 8080:80 nexus-api `'
    b. Server: '` docker run -p 5000:5000 -e MONGO_URL='URL_LINK' nexus-client `'
7. Here we tried to run it but due to some error the client side was not accessible but the server side was running thus we had to shift to the AWS EC2 instance.

Deploy (AWS EC2):

1. With the free tier, we get a few hours of computing for free on EC2.
2. We first create an instance of EC2 which gives us the required resources to run the application. We enable HTTP and HTTPS connection for instance for connecting the app to the outside internet.
3. Now we open the instance and install docker '`sudo yum install docker'` and check the installation with '`docker –version'`.
4. Now use the command '`docker pull <image_name>:tag'` to pull the image in the EC2 instance.
5. Now with the instance we run the docker run command.
    a. Client: '` docker run -p 8080:80 nexus-api `'

b. Server: ` docker run -p 5000:5000 -e MONGO_URL='URL_LINK' nexus-client `

6. This will run the client and server sides.

5. Implementation Details

5.1 Application code and Database Connectivity snapshot

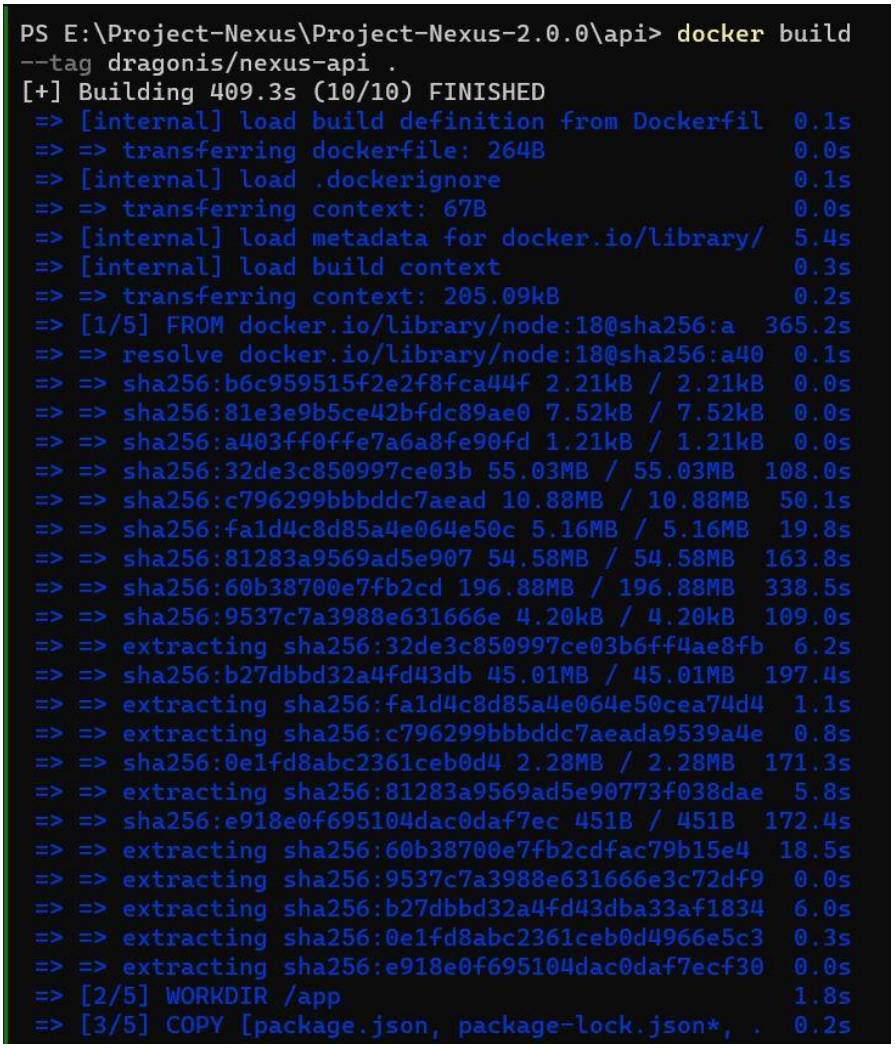
```
const express = require("express");
const app = express();
const dotenv = require("dotenv");
const mongoose = require("mongoose");
const authRoute = require("./routes/auth");
const userRoute = require("./routes/users");
const postRoute = require("./routes/posts");
const categoryRoute = require("./routes/categories");
const multer = require("multer");
const cors = require("cors");
const path = require("path");

const port = process.env.PORT || 5000;

dotenv.config();
app.use(express.json());
app.use(cors());

mongoose.set("strictQuery", false);
mongoose
    .connect(
        process.env.MONGO_URL,
        {
            useNewUrlParser: true,
            useUnifiedTopology: true
        }
    )
    .then(
        console.log("Connected to MongoDB")
    )
    .catch((err) => {
        console.log(err + "\nError brrr :(")
    });
```

Database document snapshot

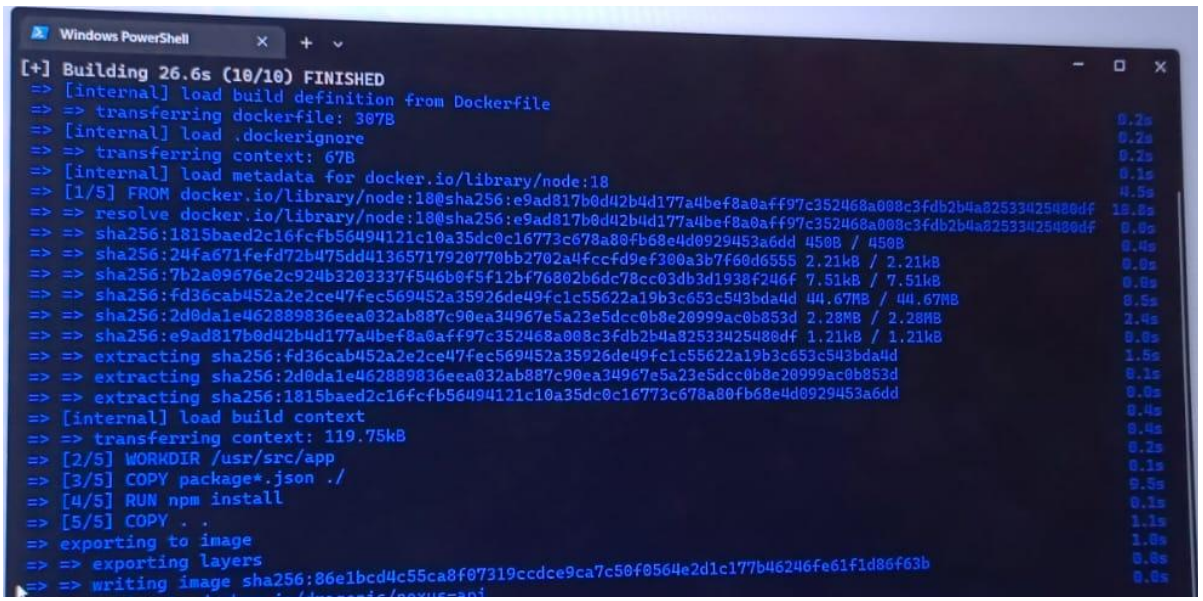
```
_id: ObjectId('63bb8ee84e354cf63f66b09f')
title: "The cc aat"
desc: "Hi there cc aat"
photo: "https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fw…"
username: "Dragonis"
› categories: Array
createdAt: 2023-01-09T03:50:00.271+00:00
updatedAt: 2023-01-09T03:50:00.271+00:00
__v: 0
```

5.2 Commands used to host application snapshots

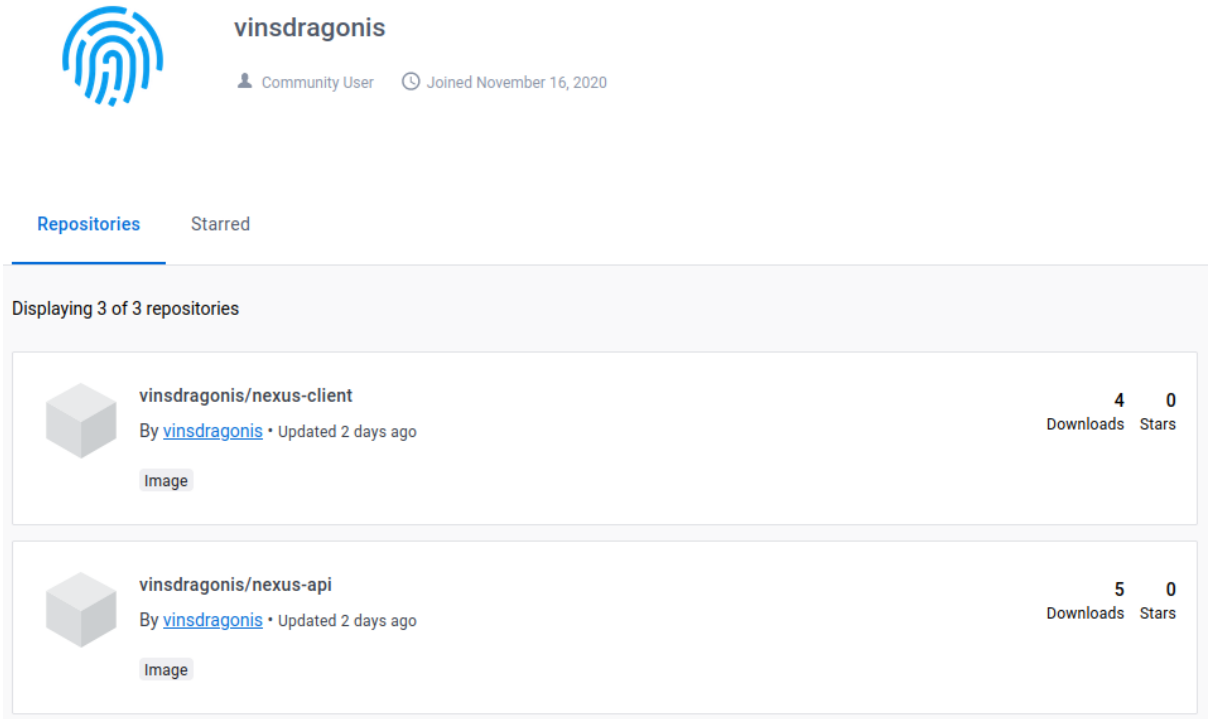
Containerisig:



Containerise server: `docker build -tag dragonis/nexus-api`



Build client: `docker build -tag dragonis/nexus-client`



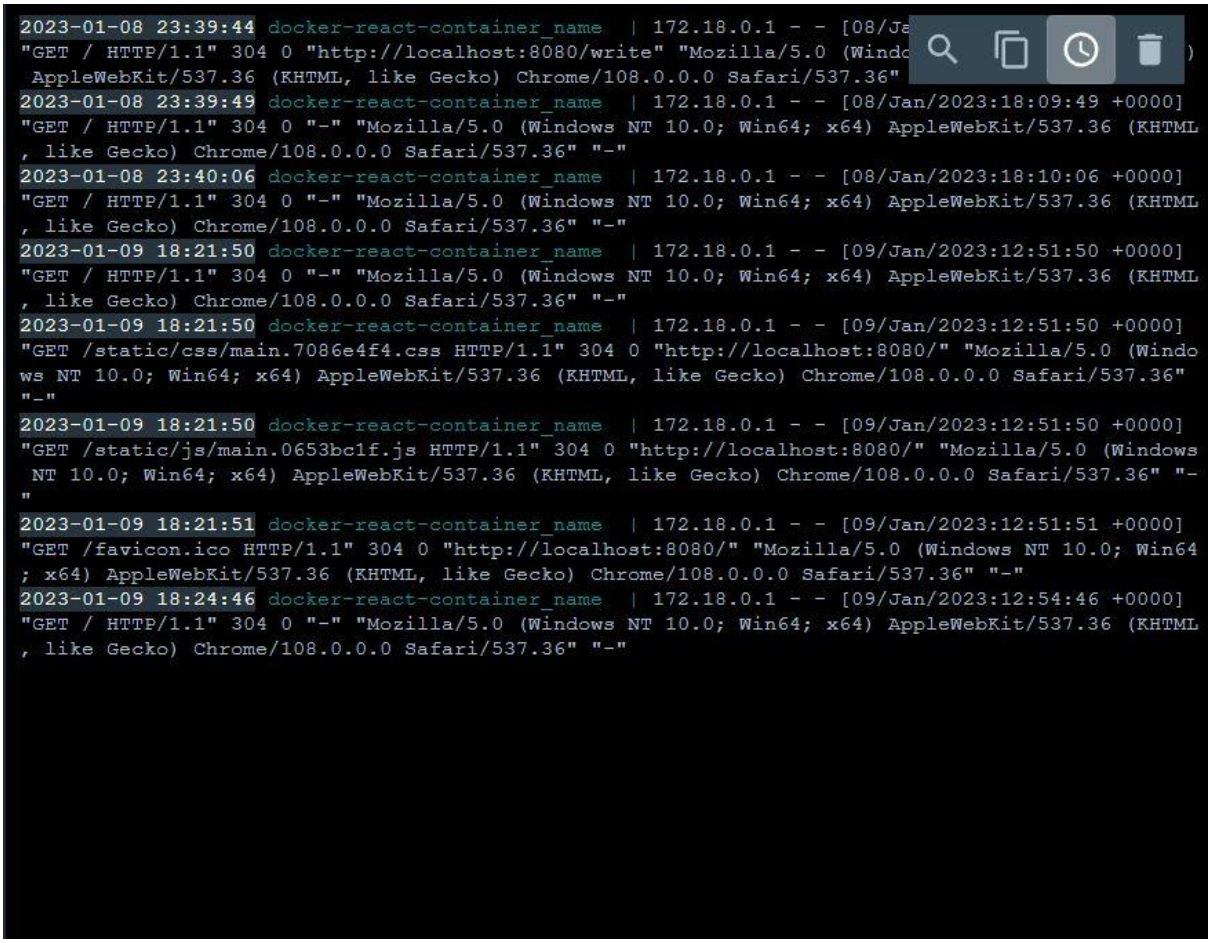
Docker Hub

For push:

```
docker push vinsdragonis/nexus-api
docker push vinsdragonis/nexus-client
```

For pull:

```
docker pull vinsdragonis/nexus-api
docker pull vinsdragonis/nexus-client
```



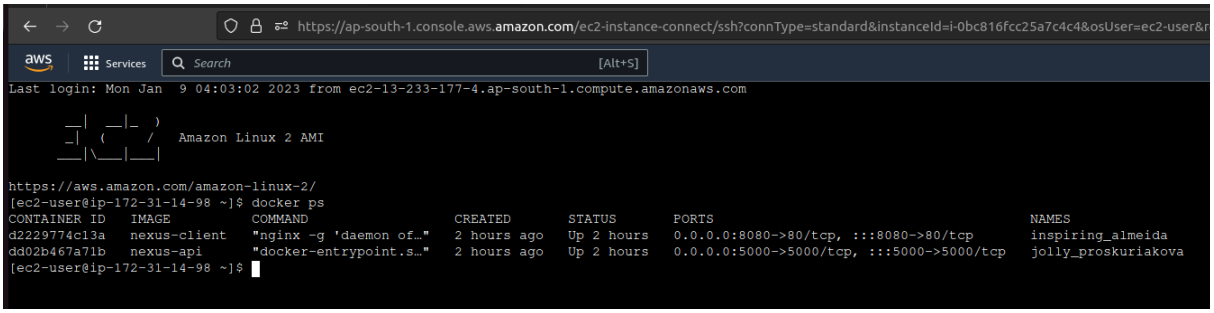
Docker

Run Server: `docker run -p 5000:5000 -e MONGO_URL='MONGO_CONNECT_URL' dragonis/nexus-api`

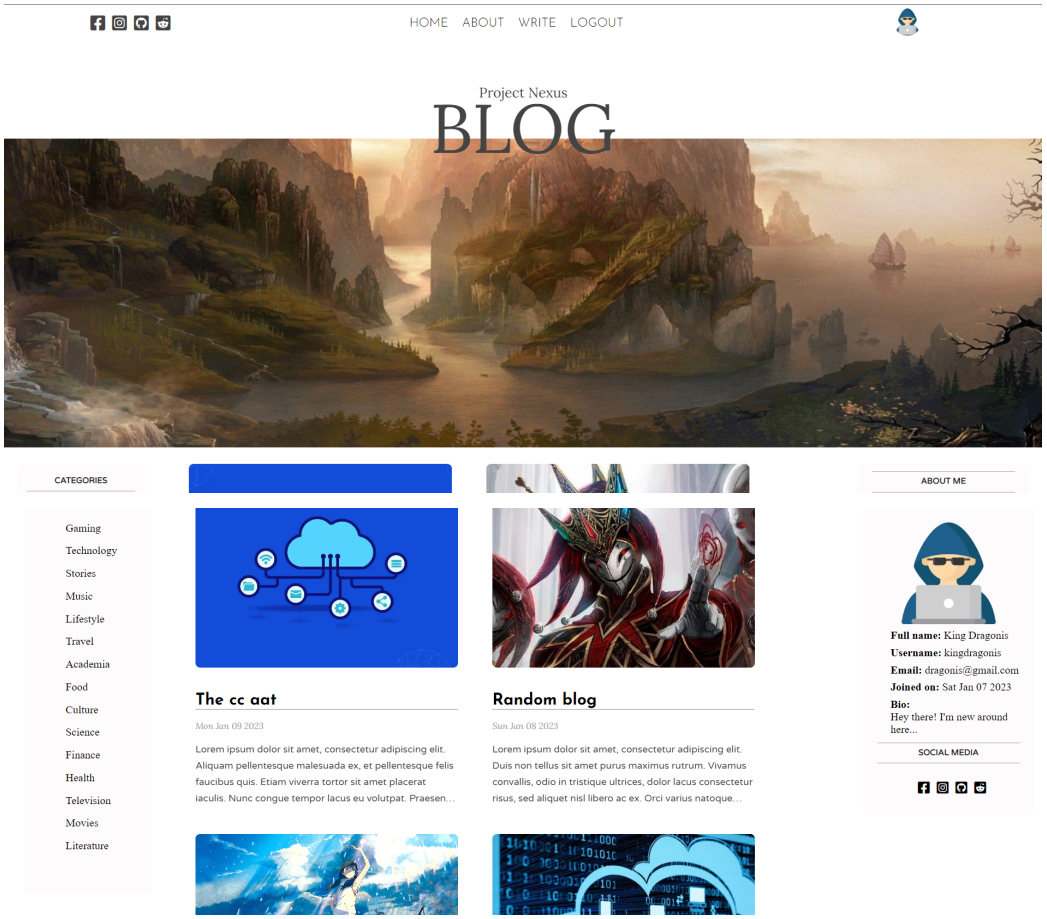
AWS EC2

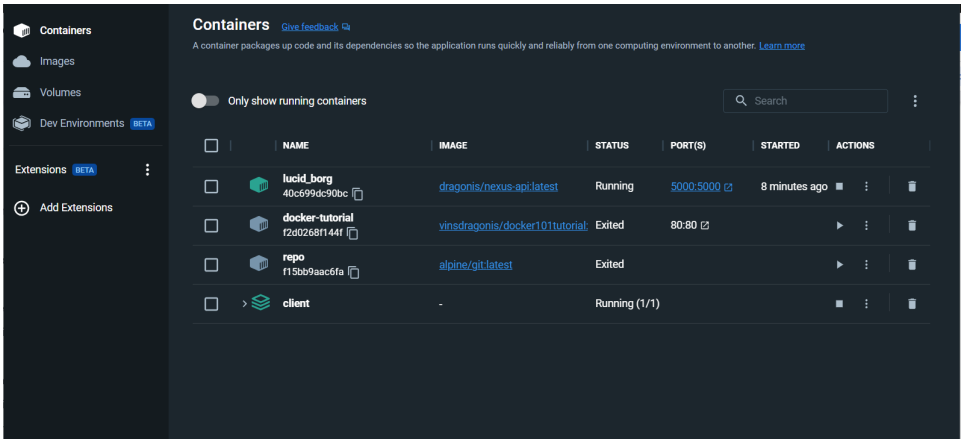
Run Server: `docker run -p 5000:5000 -e MONGO_URL='MONGO_CONNECT_URL' dragonis/nexus-api`

Run client: `docker run -p 8080:80 dragonis/nexus-client`



6. Results





7. Conclusion

We were able to dockerise and deploy a MERN stack application with the help of Docker and AWS EC2 platform successfully and were able to access the client side on various devices. By enabling the HTTP feature we were directly able to tackle the problem faced in the deployment on the DigitalOcean platform. It is easy and quick to run and deploy applications

on Cloud with better security and documentation at our disposal to be used to make our work efficient.

8. References

https://docs.docker.com/engine/install/ubuntu/#set-up-the-repository
https://hub.docker.com/
https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#SecurityGroups:
https://www.youtube.com/watch?v=RSI3v5YzPbc&t=871s