PROJECT REPORT

(Project Term January – May 2024)

Web-App Deployment using Minikube

Submitted by

JHADE BHUVANESWAR REG NO:12017874

Course Code : INT-334

Course Title : **ENTERPRISE APPLICATION AUTOMATION**

Submitted to

Dr. Varsha - 28384

School of Computer Science and Engineering

DECLARATION

We hereby declare that the project work entitled Web-App Deployment using Minikube is an authentic record of our own work carried out as requirements of Capstone Project for the award of B.Tech degree in Computer Science and Engineering from Lovely Professional University, Phagwara, under the guidance of Dr. Varsha, during January to May 2024. All the information furnished in this project report is based on my own intensive work and is genuine.

Name of Student : JHADE BHUVANESWAR

Registration Number : 12017874

JHADE BHUVANESWAR

DATE:24-04-2024

CERTIFICATE

This is to certify that the declaration statement made by the student is correct to the best of my knowledge and belief. He /She have completed this Project under my guidance and supervision. The present work is the result of his/her original investigation, effort, and study. No part of the work has ever been submitted for any other degree at any University. The Project is fit for the submission and partial fulfilment of the conditions for the award of B.Tech degree in Computer Science and Engineering from Lovely Professional University, Phagwara.

Dr. Varsha,

UID: 28384

School of Computer Science and Engineering,

Lovely Professional University,

Phagwara, Punjab.

Date : 24-04-2024.

TABLE OF CONTENTS

INTRODUCTION:

The project embarked upon in the domain of Git, Docker, and Minikube revolves around deploying a web application using Kubernetes orchestration within a local system environment. The endeavor encompasses several key components and processes aimed at establishing a robust and scalable infrastructure for hosting web applications.

At its core, the project involves the creation of a Kubernetes cluster utilizing Minikube, a tool designed to streamline the setup and management of Kubernetes clusters on local machines. This cluster serves as the foundation upon which the web application deployment is orchestrated.

Through the utilization of Docker containers, the project encapsulates and manages web applications within Kubernetes pods. Docker containers offer a lightweight and portable solution for packaging applications and their dependencies, ensuring consistency and efficiency across different environments.

To bolster the reliability and scalability of the infrastructure, various deployment strategies are implemented within the Kubernetes cluster. These strategies, including replication, auto-healing, and auto-scaling, are instrumental in maintaining service availability, automatically recovering from failures, and dynamically adjusting resource allocation based on workload demands.

Furthermore, meticulous service configuration is undertaken to facilitate seamless communication between pods within the Kubernetes cluster and to enable external connectivity. This configuration ensures that the deployed web applications can interact effectively with each other and with external users or services, thereby enhancing overall functionality and user experience.

In summary, the project represents a comprehensive approach to web application deployment within a Kubernetes environment using Minikube. By leveraging the capabilities of Kubernetes, Docker, and related tools, the project aims to establish a resilient and scalable infrastructure capable of supporting the hosting needs of various web applications, paving the way for enhanced efficiency and innovation in software development and deployment processes.

Tools Used in Web-App Deployment using Minikube:

Git:

Git is a distributed version control system for tracking changes in source code during software development. It facilitates collaboration among developers, allowing them to work on the same codebase concurrently and manage code branches.

Docker:

Docker is a platform for developing, shipping, and running applications in containers. Containers provide a lightweight, isolated environment for applications, ensuring consistency across different environments and simplifying deployment processes.

Minikube:

Minikube is a tool that simplifies the setup and management of Kubernetes clusters on local machines. It allows developers to create a single-node Kubernetes cluster for development and testing purposes, enabling them to experiment with Kubernetes features and deploy applications locally.

AWS EC2 Instance:

An Amazon Elastic Compute Cloud (EC2) instance provides scalable computing resources and secure networking capabilities for deploying applications and performing development tasks in the cloud. It serves as a remote server for storing code repositories, running development environments, and hosting applications.

Dockerfile:

Dockerfile is a text document that contains instructions for building a Docker image. It specifies the base image, copies application code, and configures runtime environments, enabling developers to create reproducible and consistent Docker images.

Docker Hub:

Docker Hub is a cloud-based registry service for storing and sharing Docker images. It provides versioning and access control features, allowing developers to manage and share Docker images securely.

OVERVIEW OF TOOLS:

GIT:

Git is a distributed version control system used for software development. It was created by Linus Torvalds in 2005 and has since become one of the most popular version control systems used by developers worldwide. Git is open-source software that is available for free and can be run on various platforms, including Windows, macOS, and Linux.

Git is designed to keep track of changes made to a project's source code over time, making it easy for developers to collaborate on projects, track changes, and revert to earlier versions of code. Git allows developers to work on the same codebase simultaneously, without fear of losing work or introducing conflicts. Git also provides a branching and merging mechanism that allows developers to work on separate features and then merge their changes back into the main codebase.

One of the significant benefits of Git is that it is a distributed system, which means that developers can work offline and then synchronize their changes with the central repository when they come online. Git also provides a robust set of tools for managing code, including features like commit, push, pull, branch, merge, and revert.

Git is widely used in software development and is particularly popular with open-source projects. Many hosting platforms like GitHub, GitLab, and Bitbucket provide Git-based version control systems as a service, making it easy for developers to collaborate and share their work with others.

In conclusion, Git is a powerful and widely used version control system that helps developers manage code changes, collaborate on projects, and keep track of code versions. It has become an essential tool for software development and is likely to continue to be an important part of the software development ecosystem.

DOCKER:

Docker is an open-source platform used to create, deploy, and run applications in containers. Docker was first released in 2013 and has since become one of the most widely used containerization tools in software development. Docker containers are lightweight, portable, and self-contained, making them ideal for developing, testing, and deploying applications in various environments.

A container is a standardized unit of software that contains all the dependencies, libraries, and configuration needed to run an application. Containers provide a consistent environment for developers to work in and help reduce the complexity of managing applications across different environments.

Docker provides a command-line interface and a set of APIs that allow developers to create, manage, and deploy containers. Docker images are used to create containers, and they contain all the necessary files, libraries, and configuration needed to run an application. Docker images are created using a Dockerfile, which contains instructions for building the image.

Docker containers are isolated from each other and from the host system, which makes them more secure than traditional virtual machines. Containers also use fewer resources than virtual machines, making them more efficient and cost-effective.

Docker is widely used in software development and is particularly popular with DevOps teams who need to deploy applications quickly and efficiently. Docker makes it easy to build and deploy applications in various environments, including cloud, on-premise, and hybrid environments.

In conclusion, Docker is a powerful and widely used containerization platform that helps developers create, deploy, and run applications in containers. It provides a consistent and secure environment for developers to work in and helps reduce the complexity of managing applications across different environments. Docker has become an essential tool for software development and is likely to continue to be an important part of the DevOps ecosystem.

MINIKUBE:

Minikube is a tool that facilitates the setup and management of Kubernetes clusters on local machines. It provides a single-node Kubernetes cluster that enables developers to develop, test, and experiment with Kubernetes features without requiring access to a production-like environment. Here's a comprehensive overview of Minikube and its key features:

Local Kubernetes Cluster:
Minikube allows developers to create a local Kubernetes cluster on their development machines, providing a lightweight environment for testing and development.
The local Kubernetes cluster created by Minikube consists of a single node, making it suitable for small-scale development and testing tasks.

Simplified Setup:
Minikube simplifies the setup process for Kubernetes clusters by automating the installation and configuration of all necessary components, including the Kubernetes control plane and container runtime.
Developers can set up a local Kubernetes cluster with just a few commands, eliminating the need for manual configuration and reducing the time and effort required to get started with Kubernetes development.

Cross-Platform Support:
Minikube is designed to work seamlessly across different operating systems, including Linux, macOS, and Windows.
It leverages native virtualization technologies, such as Hyper-V, VirtualBox, and KVM, to create isolated environments for running the Kubernetes cluster, ensuring compatibility and performance across various platforms.

Development and Testing:
Minikube provides developers with a platform for developing and testing Kubernetes applications in a local environment before deploying them to production clusters.
Developers can deploy and manage applications on the Minikube cluster using familiar Kubernetes APIs and tools, enabling rapid iteration and debugging of applications.

Integration with Kubernetes Ecosystem:
Minikube integrates seamlessly with the broader Kubernetes ecosystem, allowing developers to leverage Kubernetes features and extensions in their local development environment.
Developers can use tools such as kubectl, Helm, and Kubernetes Dashboard to interact with the Minikube cluster and manage applications running on it.
Isolation and Security:

Minikube creates isolated environments for running Kubernetes clusters on local machines, ensuring that applications and resources are sandboxed and secure.
Developers can experiment with different configurations and settings without affecting other applications or environments on their development machines, maintaining a high level of isolation and security.

AWS EC2 INSTANCE:

An AWS EC2 (Elastic Compute Cloud) instance is a virtual server provided by Amazon Web Services (AWS) that allows you to run applications on the cloud. Here's an overview of key aspects of EC2 instances:

1. Scalability and Flexibility:
   - EC2 instances offer scalability and flexibility, allowing you to quickly scale up or down the computing resources based on your needs.
   - You can choose from various instance types with different combinations of CPU, memory, storage, and networking capacity to match your application requirements.

2. On-Demand Provisioning:
   - With EC2, you can provision instances on-demand, meaning you can launch them when needed without any upfront commitments.
   - This allows you to scale your infrastructure according to demand, optimizing costs and resource utilization.

3. Variety of Operating Systems and Applications:
   - EC2 supports a wide range of operating systems, including Amazon Linux, Ubuntu, Windows Server, and others.
   - You can also deploy pre-configured Amazon Machine Images (AMIs) containing specific software stacks or applications, making it easy to get started with various use cases.

4. Security and Isolation:
   - EC2 instances run in a virtualized environment, providing isolation from other instances on the same physical server.
   - AWS offers various security features, such as security groups, network access control lists (ACLs), and AWS Identity and Access Management (IAM), to control access to your instances and resources.

5. Elastic IP Addresses and Elastic Block Storage (EBS):
   - EC2 instances can be associated with Elastic IP addresses, providing a static IPv4 address that remains associated with the instance until you explicitly release it.
   - You can attach Elastic Block Storage (EBS) volumes to EC2 instances to provide persistent storage for your data. EBS volumes can be resized and attached/detached from instances as needed.

6. Monitoring and Management:

- AWS provides various tools for monitoring and managing EC2 instances, such as Amazon CloudWatch, which allows you to collect and analyze metrics, set alarms, and automate actions based on predefined triggers.
- Additionally, you can use AWS Systems Manager to automate tasks like patch management, software inventory, and configuration management for EC2 instances.

Overall, AWS EC2 instances provide a flexible and scalable compute environment for running applications in the cloud, offering a wide range of features and options to meet different use cases and requirements.

DOCKERFILE:

A Dockerfile is a text file that contains instructions for building a Docker image. It serves as a blueprint for creating containerized applications within the Docker ecosystem. Here's an overview of the key components and concepts of a Dockerfile:

1. Base Image:
   - Every Dockerfile begins with a base image, which forms the foundation of the container. It typically contains a minimal operating system or runtime environment, such as Alpine Linux, Ubuntu, or a specific programming language runtime.
   - You can specify the base image using the `FROM` instruction, followed by the name of the image and optionally its version or tag.
   - Example: `FROM alpine:3.14`

2. Environment Configuration:
   - Dockerfiles allow you to configure the environment within the container by setting environment variables using the `ENV` instruction.
   - This can include defining variables for paths, ports, database connection strings, or any other configuration settings your application requires.
   - Example: `ENV APP_PORT=8080`

3. Working Directory:
   - You can set the working directory inside the container using the `WORKDIR` instruction.
   - This is the directory where subsequent commands in the Dockerfile will be executed.
   - Example: `WORKDIR /app`

4. Copying Files:
   - Dockerfiles enable you to copy files from the host machine into the container using the `COPY` or `ADD` instructions.
   - This is useful for transferring application code, configuration files, and other resources into the container.
   - Example: `COPY . /app`

5. Executing Commands:
   - Dockerfiles allow you to execute commands inside the container using the `RUN` instruction.
   - This can include installing dependencies, running build scripts, or performing any other actions necessary to prepare the environment.

- Example: `RUN npm install`

6. Exposing Ports:
   - If your containerized application listens for incoming network connections, you can expose ports using the `EXPOSE` instruction.
   - This informs Docker that the specified ports should be accessible from outside the container.
   - Example: `EXPOSE 8080`

7. Defining Entrypoint or Command:
   - You can specify the default command to run when the container starts using the `CMD` or `ENTRYPOINT` instruction.
   - This can be the primary process of your application or a shell script that initializes the environment.
   - Example: `CMD ["node", "app.js"]`

8. Building the Image:
   - Once you've written the Dockerfile, you can build the Docker image using the `docker build` command.
   - This command reads the Dockerfile and executes each instruction to create a standalone image.
   - Example: `docker build -t myapp .`

9. Using Docker Compose (Optional):
   - For more complex applications composed of multiple services, you can use Docker Compose to define and manage multi-container Docker applications using a YAML file.
   - Docker Compose allows you to specify services, networks, and volumes, making it easier to orchestrate containers.
   - Example: `docker-compose up`

By understanding and utilizing Dockerfiles effectively, you can create reproducible and portable containerized environments for your applications, facilitating seamless development, testing, and deployment workflows within the Docker ecosystem.

DOCKER-HUB:

Docker Hub is a cloud-based repository service provided by Docker that allows developers to store, manage, and distribute Docker container images. Here's an overview of the key features and functionalities of Docker Hub:

1. Image Hosting:
   - Docker Hub serves as a centralized registry for hosting Docker images. It provides a secure and reliable platform for storing container images, ensuring availability and accessibility to users.

2. Public and Private Repositories:
   - Docker Hub allows users to create both public and private repositories for storing Docker images.
   - Public repositories are accessible to anyone and can be used to share open-source projects and community-contributed images.
   - Private repositories are restricted to authorized users and organizations, providing a secure environment for storing proprietary or sensitive images.

3. Collaboration and Sharing:
   - Docker Hub facilitates collaboration among developers by enabling them to share Docker images with team members, collaborators, and the broader community.
   - Users can publish images to public repositories, making them available for others to use and contribute to.
   - Additionally, Docker Hub supports organization accounts, allowing teams to manage and share images within a centralized environment.

4. Automated Builds:
   - Docker Hub provides automated build functionality, allowing users to configure automated workflows for building Docker images from source code repositories.
   - Users can connect their GitHub, Bitbucket, or GitLab repositories to Docker Hub and trigger automated builds whenever changes are pushed to the repository.
   - This streamlines the process of building and updating Docker images, ensuring consistency and reliability in the deployment pipeline.

5. Web Interface and CLI Access:
   - Docker Hub offers a web-based user interface that allows users to browse, search, and manage Docker images and repositories through a web browser.
   - Additionally, Docker Hub provides a command-line interface (CLI) that allows users to

interact with Docker Hub from the terminal, enabling tasks such as image management, authentication, and repository manipulation.

6. Integration with Docker Desktop and Docker CLI:
   - Docker Hub seamlessly integrates with Docker Desktop and the Docker CLI, providing a streamlined experience for pushing, pulling, and managing Docker images directly from the development environment.
   - Users can authenticate Docker Desktop and CLI sessions with Docker Hub credentials, enabling seamless access to private repositories and automated workflows.

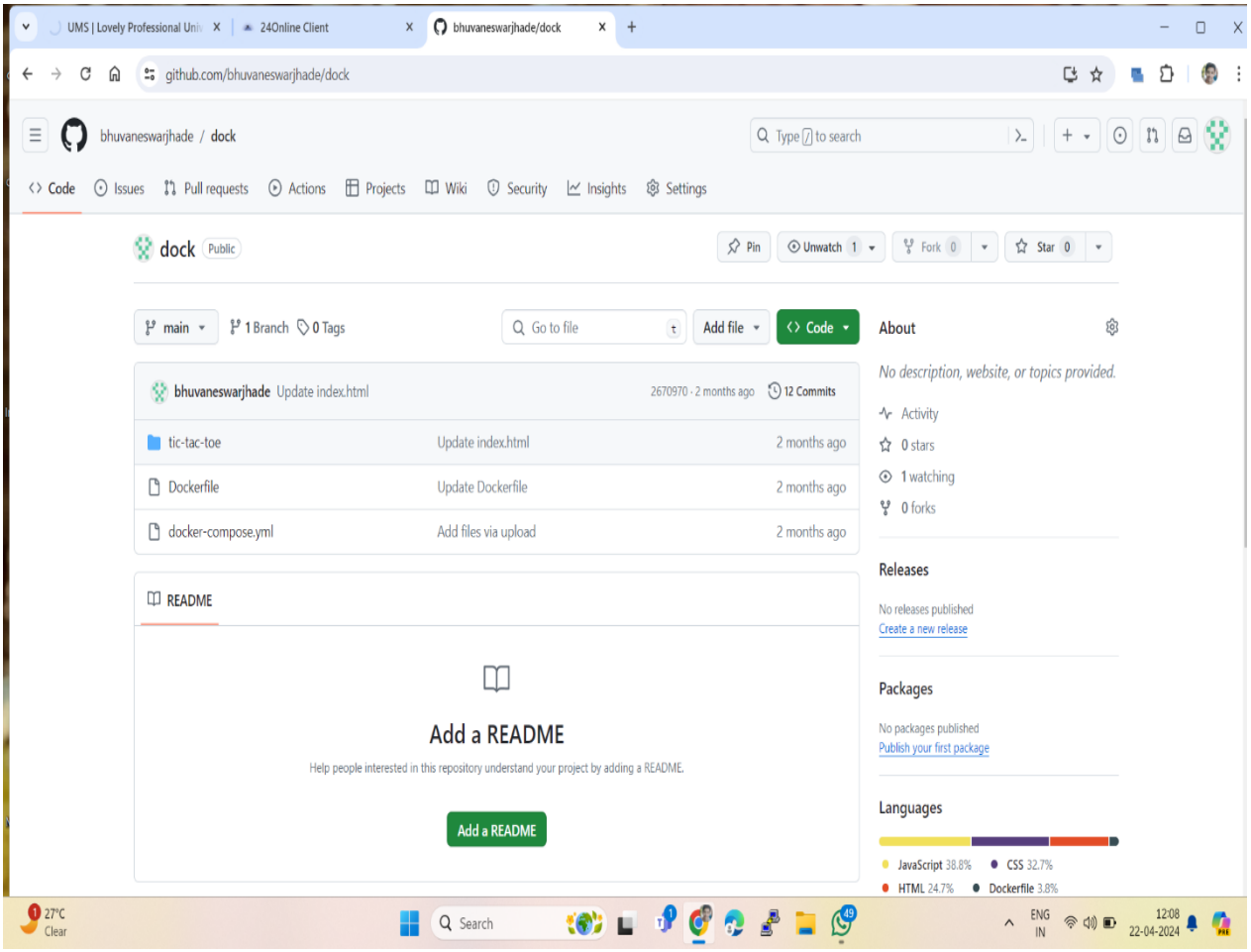
7. Security and Vulnerability Scanning:
   - Docker Hub offers security scanning capabilities that analyze Docker images for known security vulnerabilities and issues.
   - Users can leverage these scanning features to identify and remediate security vulnerabilities in their container images, ensuring compliance with security best practices and standards.

Overall, Docker Hub plays a pivotal role in the Docker ecosystem, providing developers and organizations with a centralized platform for managing Docker images, collaborating on containerized projects, and streamlining the deployment workflow.

STEPS TO CREATE A PROJECT

Firstly I have created a Tic-Tac-Toe Game using HTML, CSS and JavaScript, I have pushed my code on Git Hub.

Link: https://github.com/bhuvaneswarjhade/dock.git



1. Creating an EC2 Instance on AWS:
   - AWS EC2 (Elastic Compute Cloud) is a web service that provides resizable compute capacity in the cloud. It allows users to rent virtual servers on which they can run their applications.
   - To create an EC2 instance, you need to log in to the AWS Management Console, navigate to the EC2 dashboard, and follow the steps mentioned earlier in creating an EC2 instance.

2. Installing Docker on the EC2 Instance:
   - After launching the EC2 instance, you need to connect to it using SSH, as explained earlier.
   - Once connected to the instance, you can install Docker using the package manager available on your instance's operating system (typically either yum for Amazon Linux or apt for Ubuntu).

- Docker enables you to create, deploy, and manage containers, which are lightweight, standalone, and executable packages that contain everything needed to run an application, including code, runtime, libraries, and dependencies.

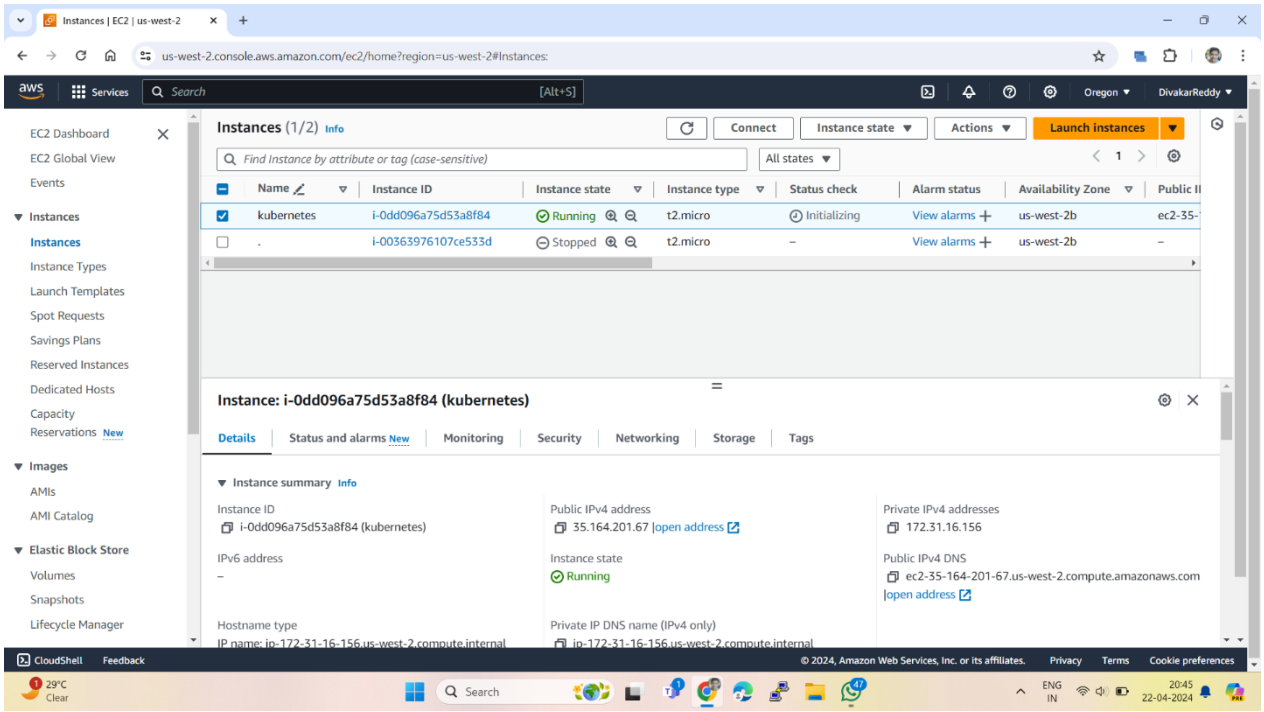
3. Building a Docker Image using a Dockerfile:
   - A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using a Dockerfile, you can specify the environment and dependencies required for your application.
   - Create a Dockerfile in your project directory, specifying the base image, adding dependencies, setting environment variables, copying files, and running commands to set up your application environment.
   - Once the Dockerfile is ready, you can build a Docker image by running the `docker build` command and providing the path to the directory containing the Dockerfile.

docker build -t myapp-image .

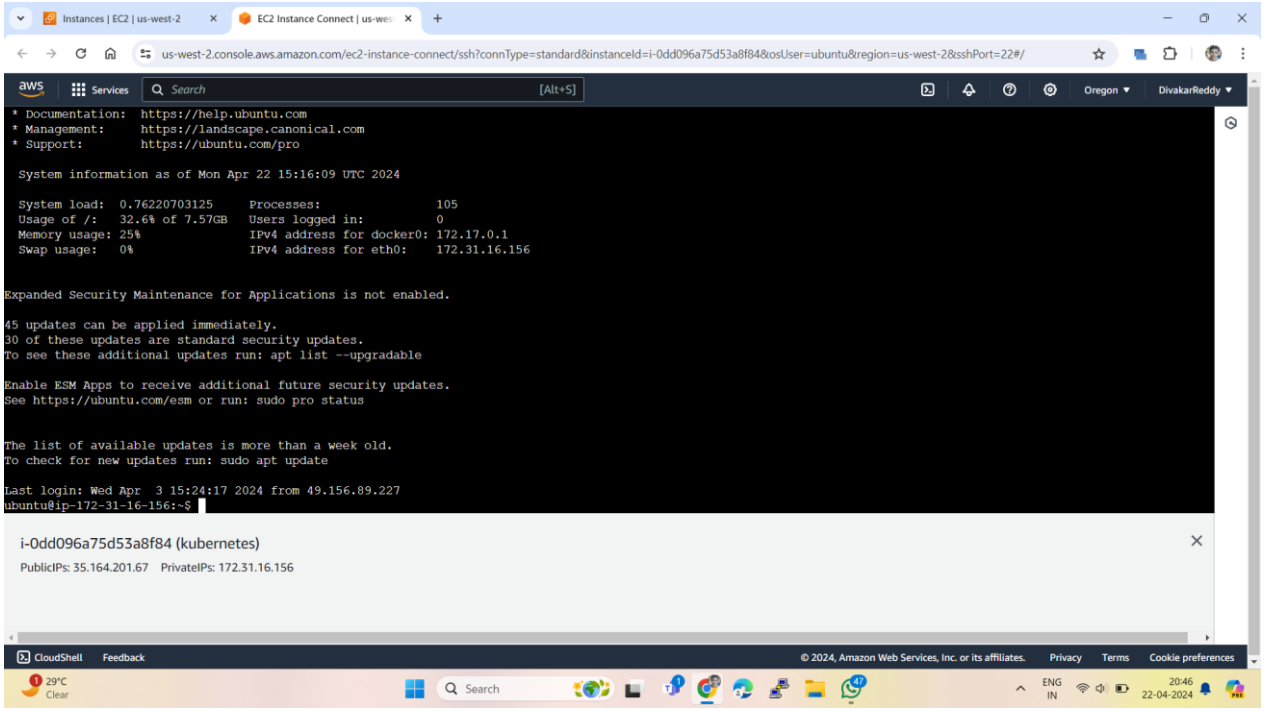
This command will build a Docker image named `myapp-image` using the Dockerfile in the current directory (`.`).

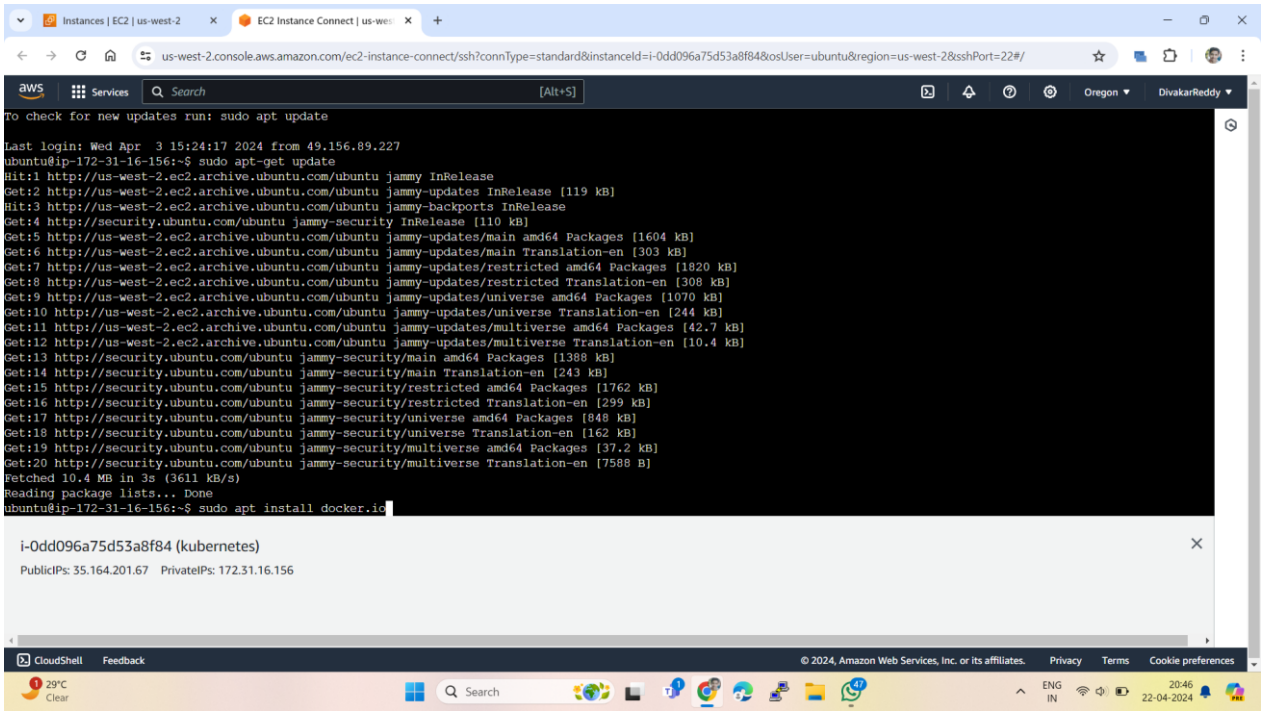
4. Pushing the Docker Image to Docker Hub:
   - Docker Hub is a cloud-based registry service that allows you to store and share Docker images.
   - To push your Docker image to Docker Hub, you need to first log in to Docker Hub using the `docker login` command.
   - After logging in, you can tag your local Docker image with your Docker Hub username and the repository name using the `docker tag` command.
   - Finally, you can push the tagged image to Docker Hub using the `docker push` command.

EC2 Instance:

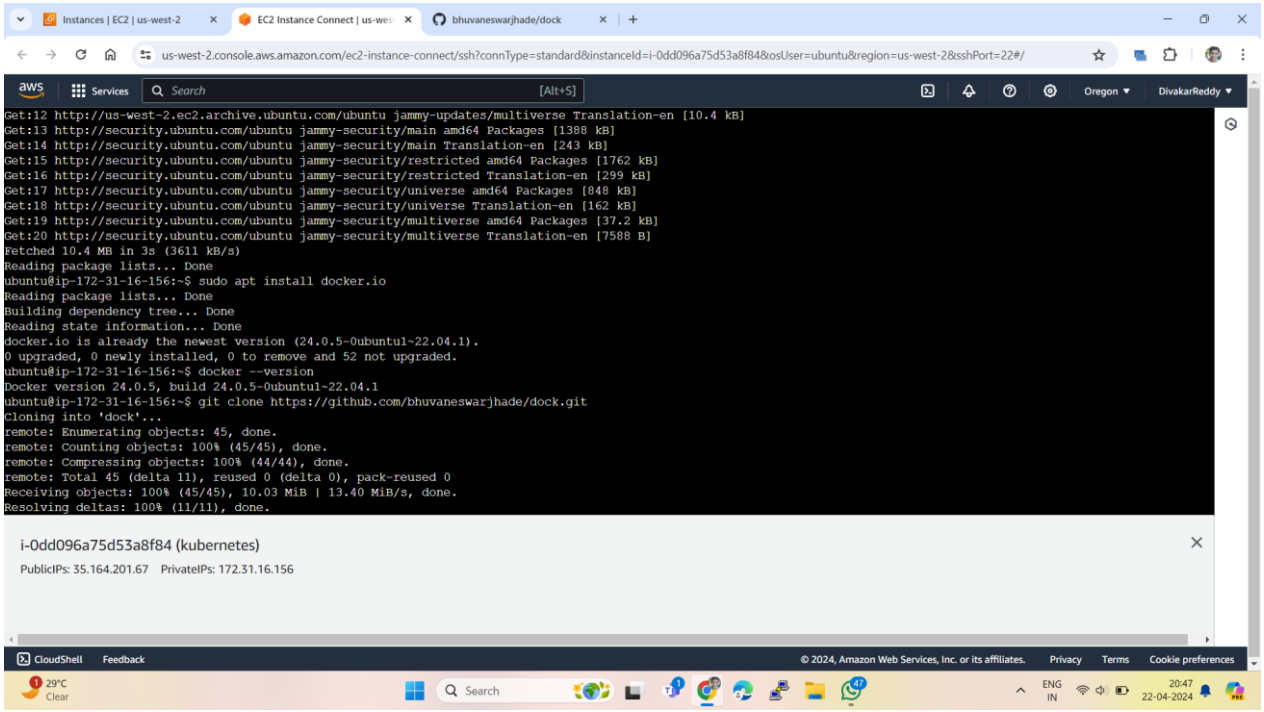


Installing Docker in EC2 Instance:

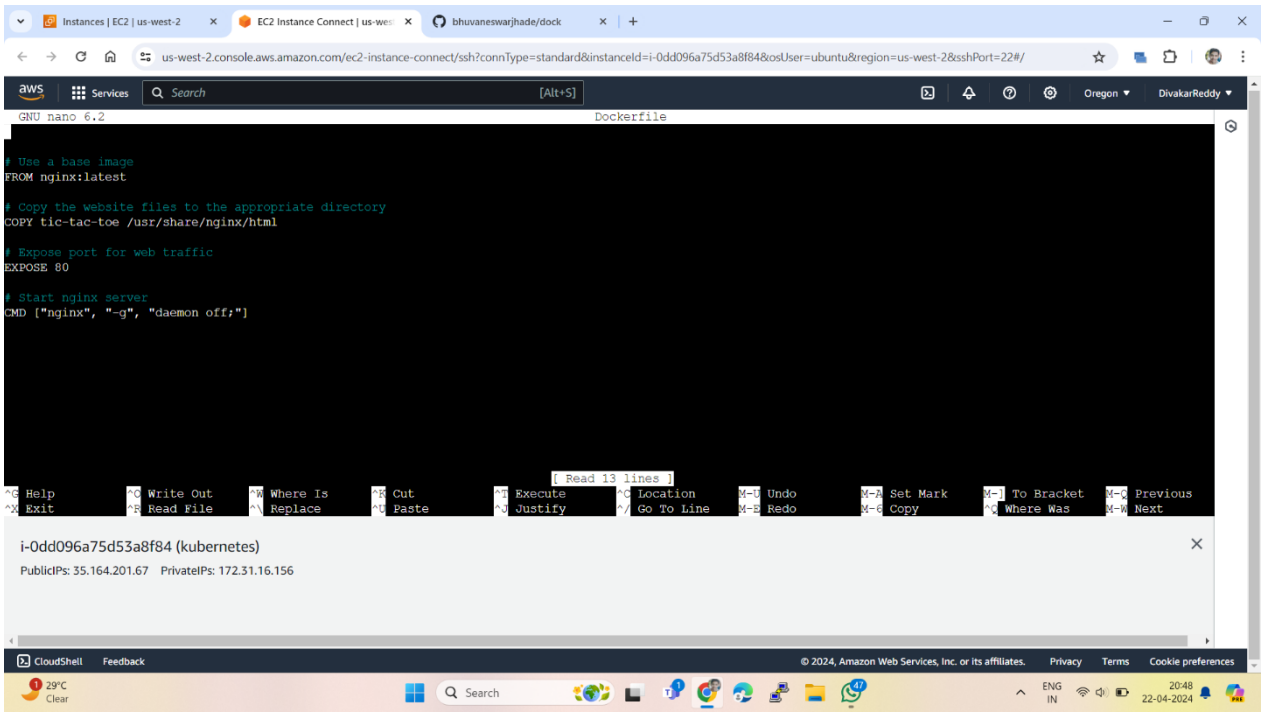




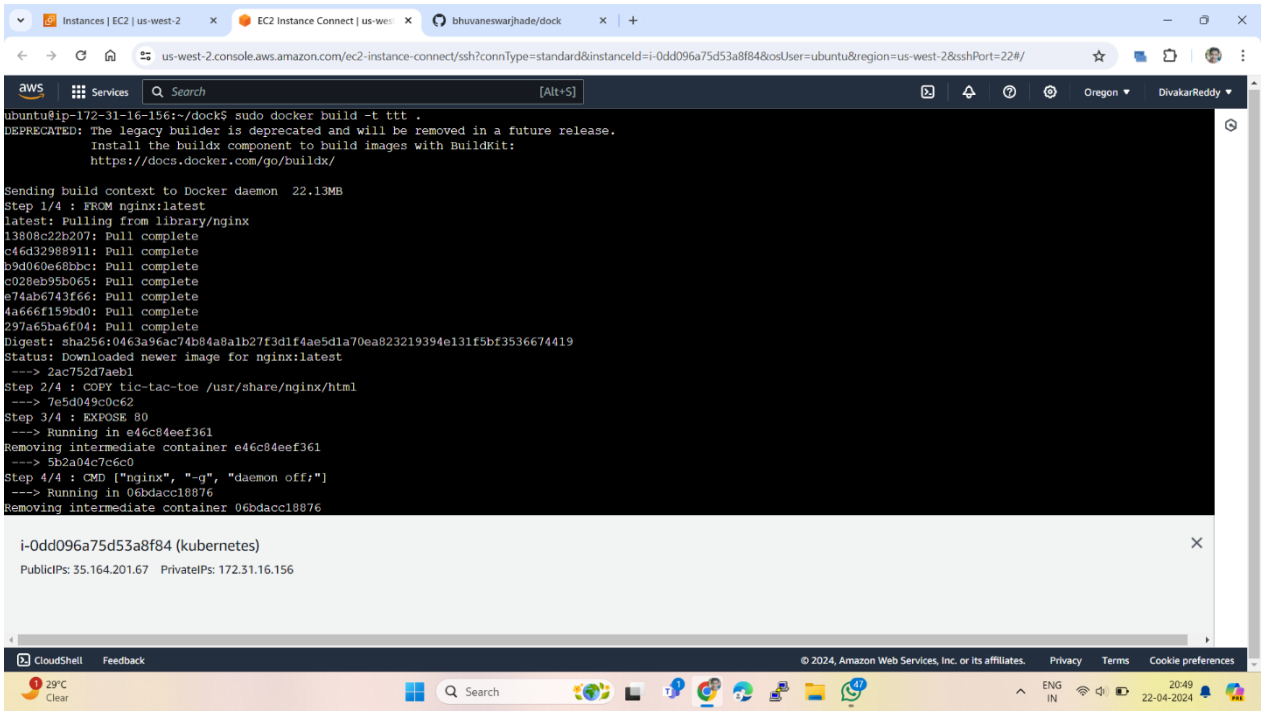
Cloning the code form Git hub:



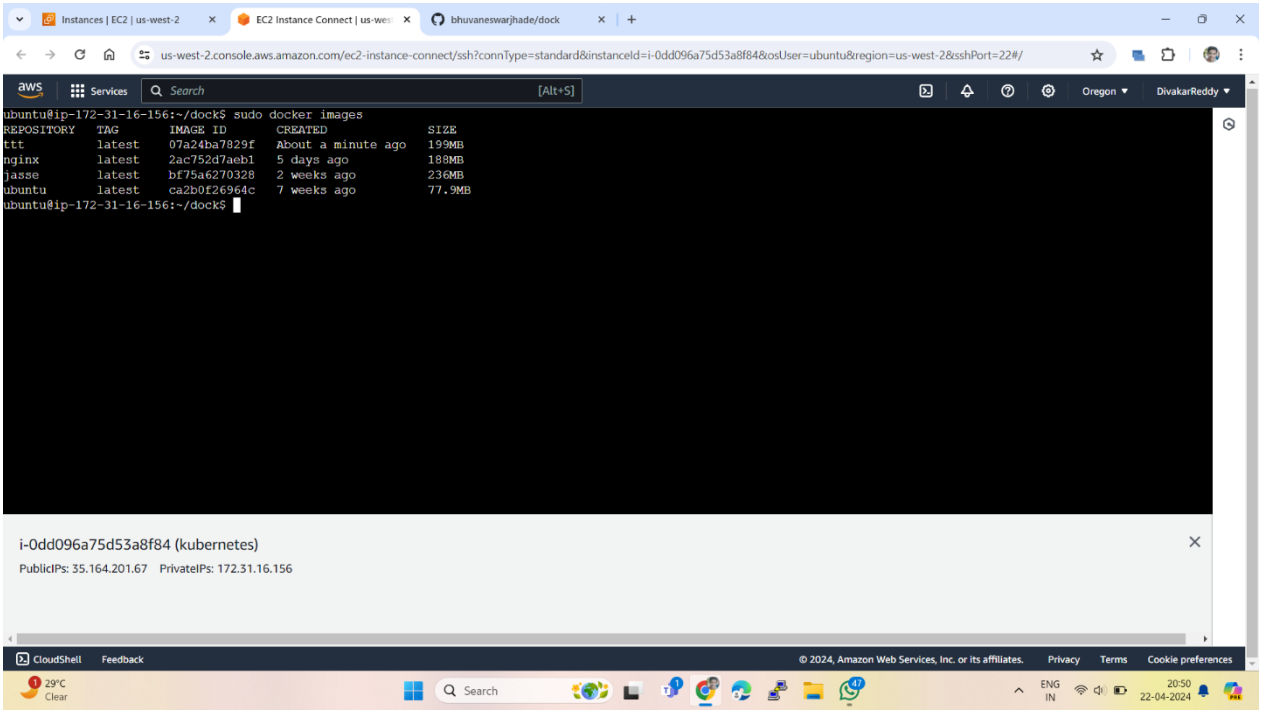
Created a Docker File in this folder:



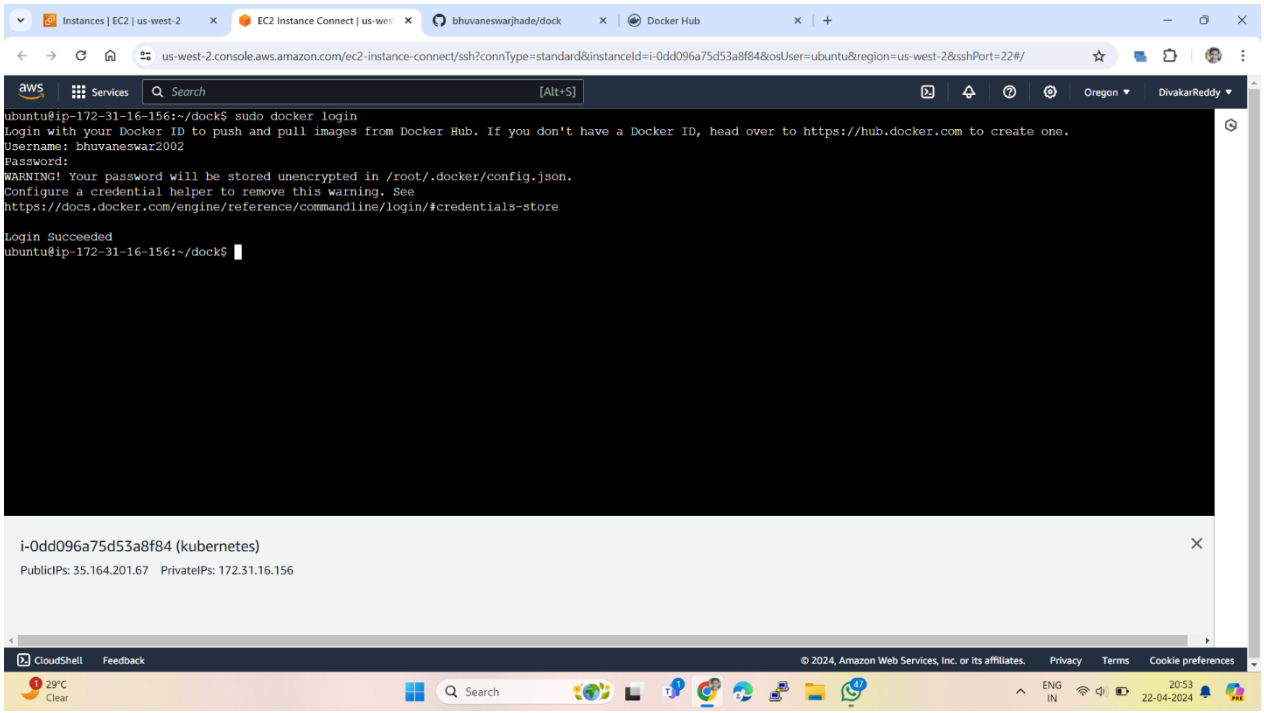
Built the Docker Image using Docker build Command:



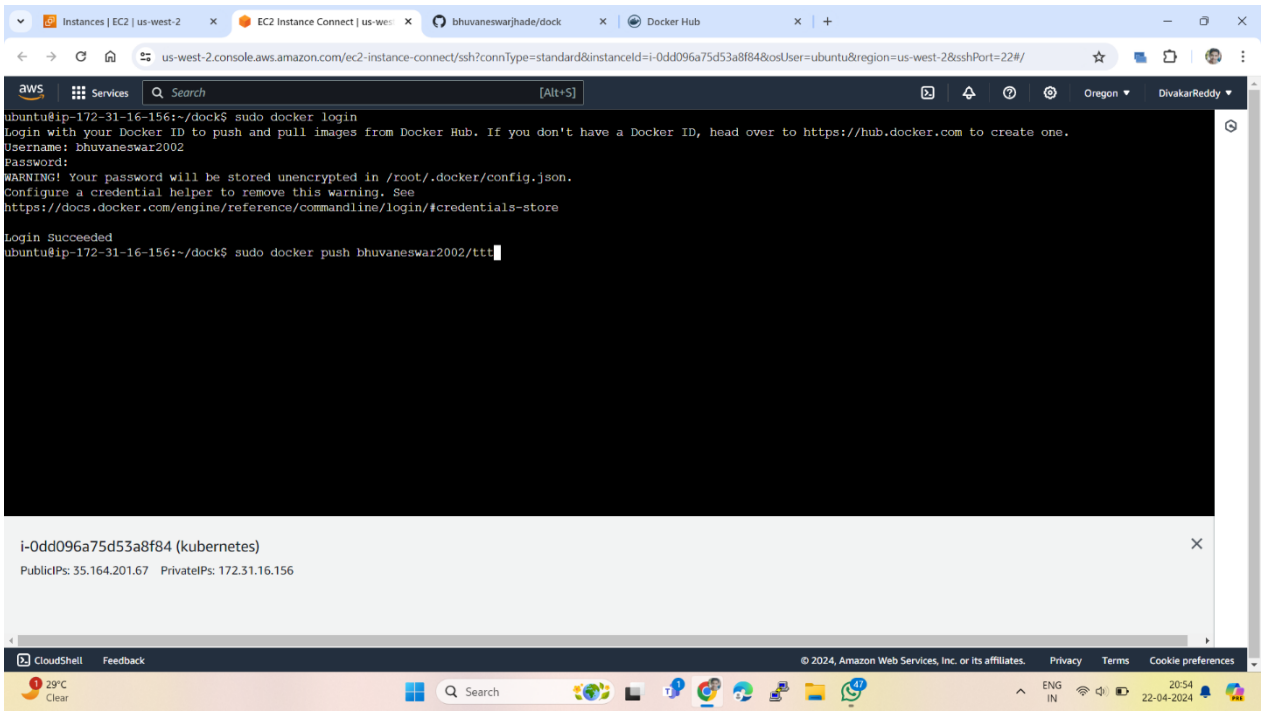
•



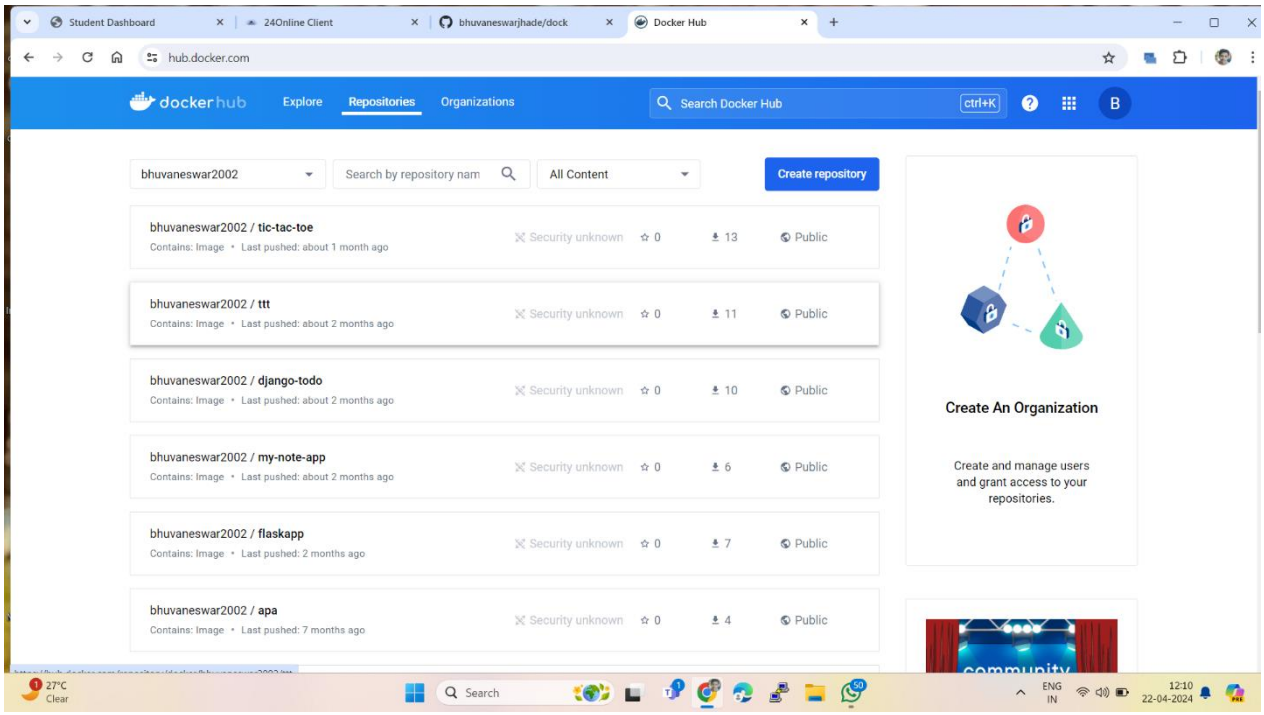
Logging into Docker Hub account using Docker login Command:



Pushing the Image to docker hub:



Docker Image has been pushed to Docker Hub:



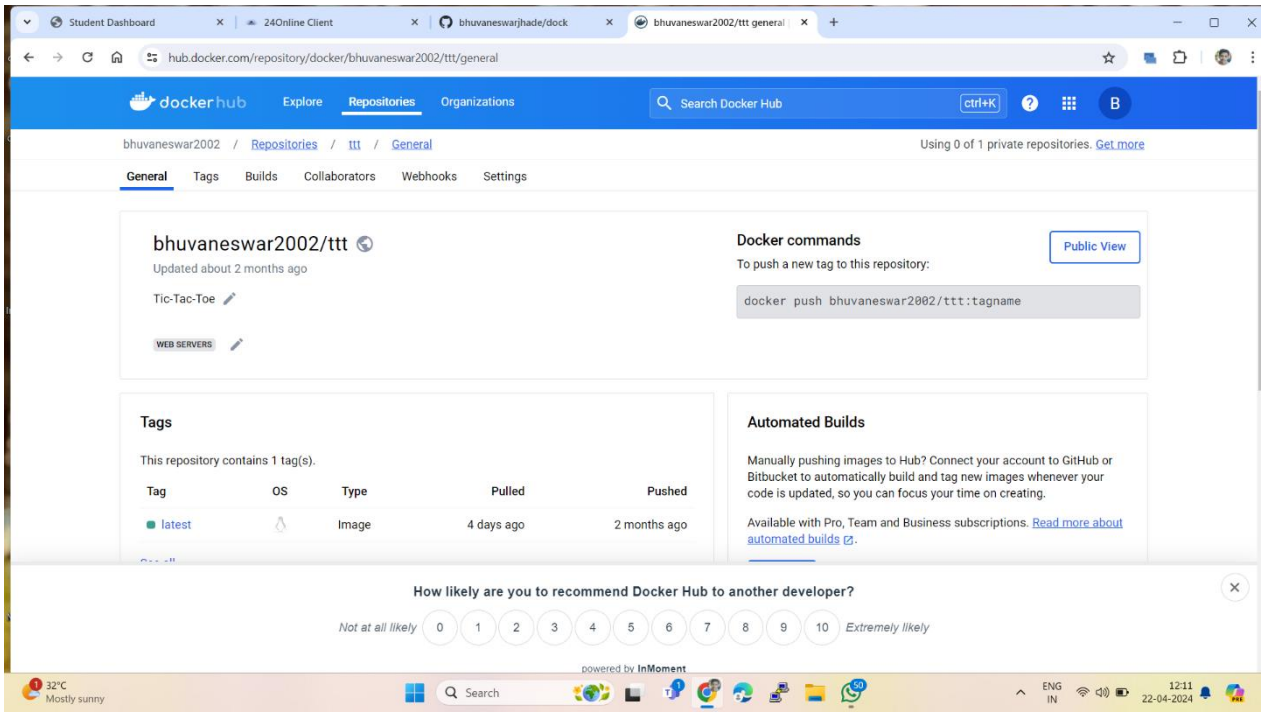
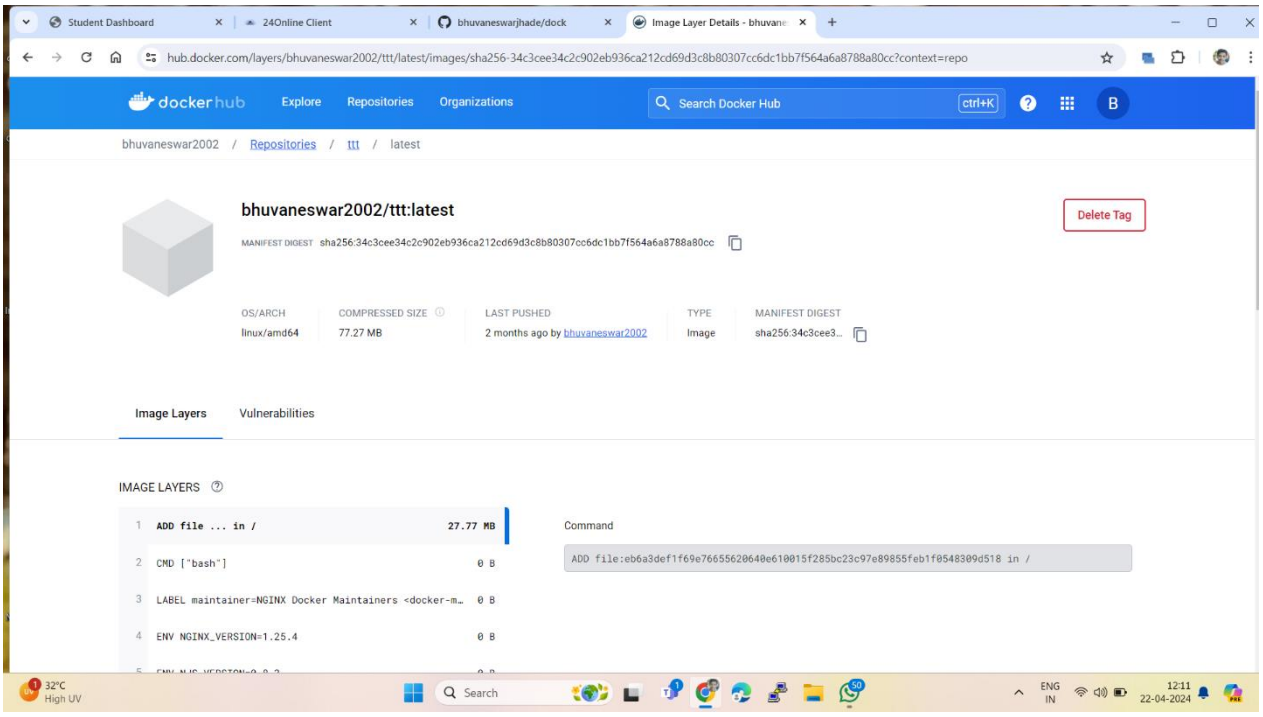


Image Present in Docker Hub:



1. Winget Overview:

- Winget is a package manager for Windows that allows you to search, install, and manage software packages from the command line.
- It simplifies the process of installing and updating software on Windows machines.

2. Installing Minikube with Winget:

- First, ensure that you have Winget installed on your Windows machine. If you haven't installed it yet, you can download and install it from the Microsoft Store or through other methods.
- Once Winget is installed, open a command prompt or PowerShell window with administrative privileges.
- To install Minikube using Winget, you can use the following command:

2. Starting Minikube:

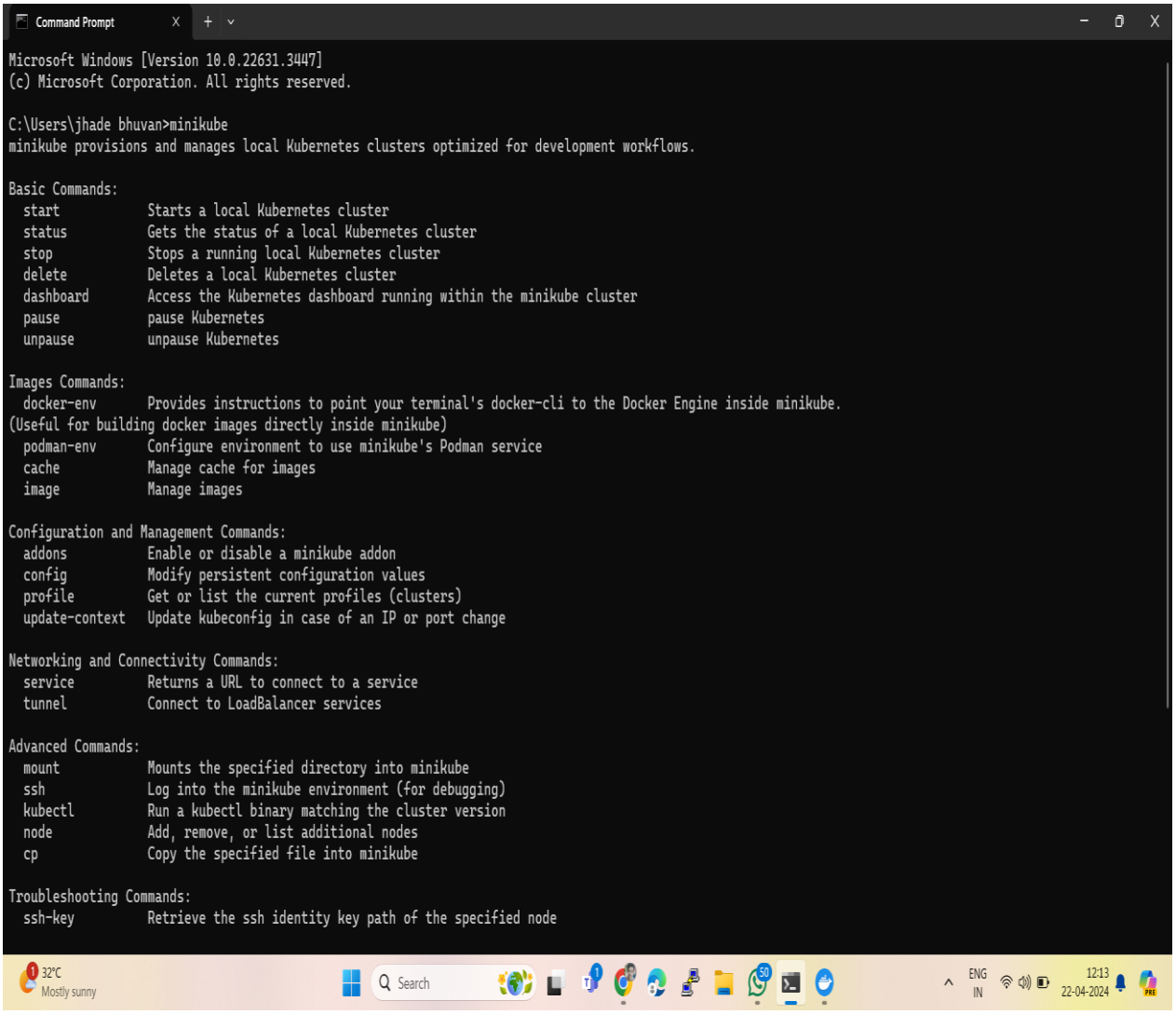
- After installing Minikube, you can start a local Kubernetes cluster by running the following command:

minikube start

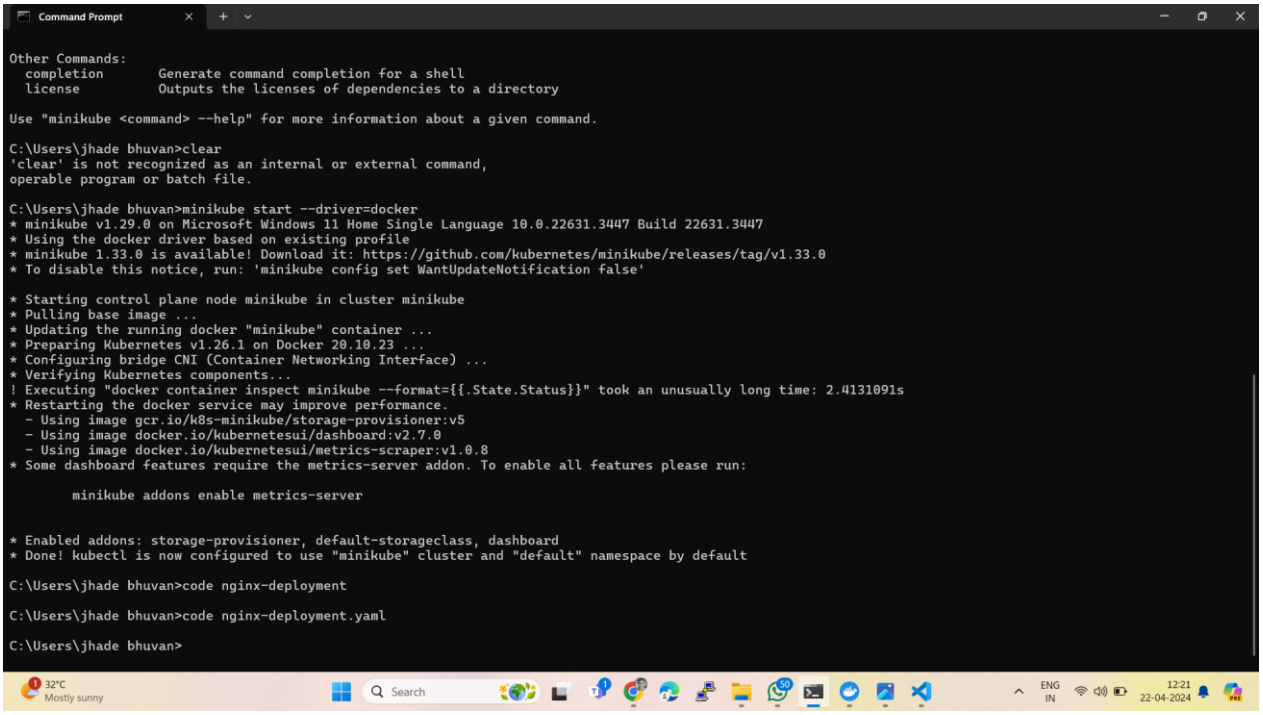
This command initializes and starts a single-node Kubernetes cluster using Minikube on your local machine.

- Minikube will automatically download the necessary Kubernetes components and start the cluster.
- Once the cluster is up and running, you can interact with it using the `kubectl` command-line tool, which should be automatically configured to work with the Minikube cluster.

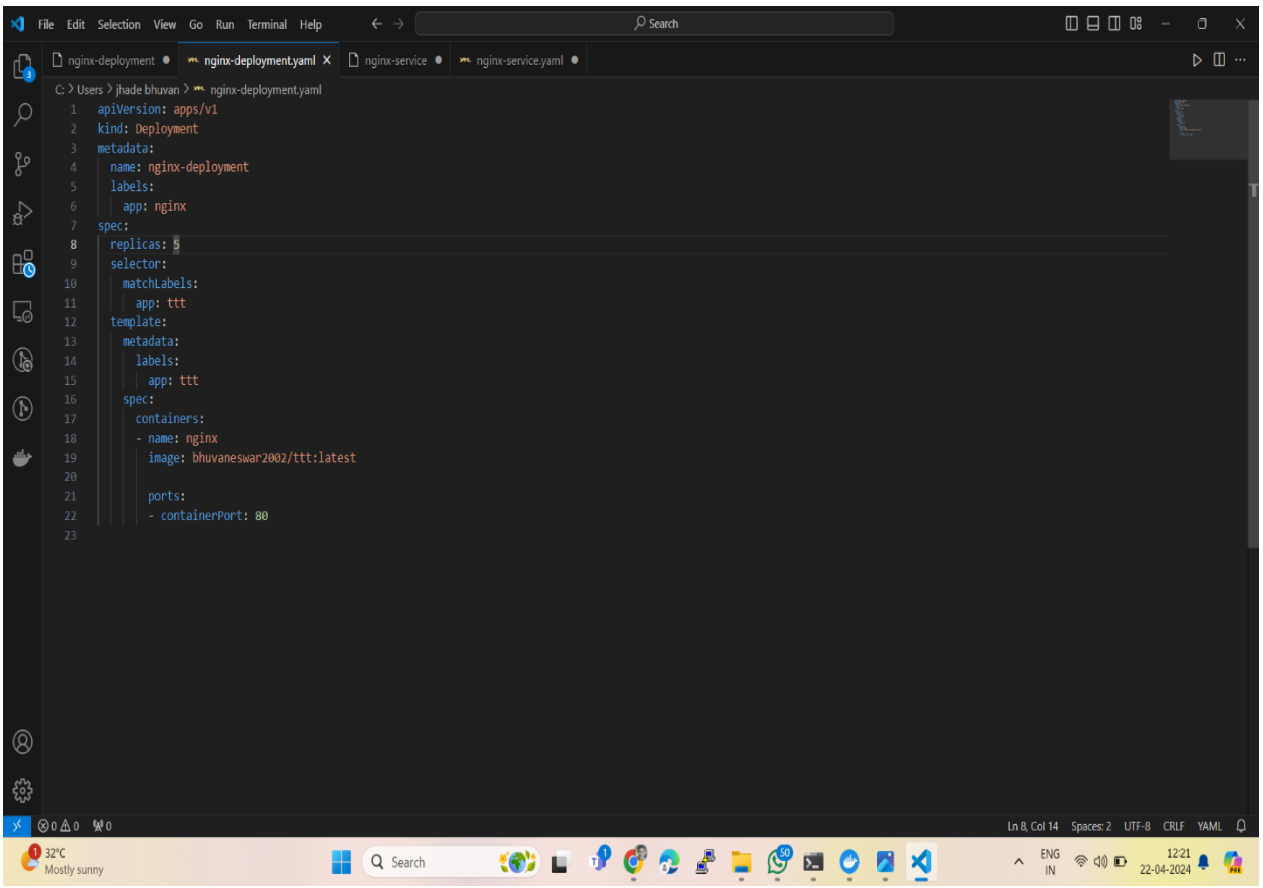
Now we Install Minikube on our Local Machine using Winget Install Minikube:



Starting Minikube using Minikube start command:

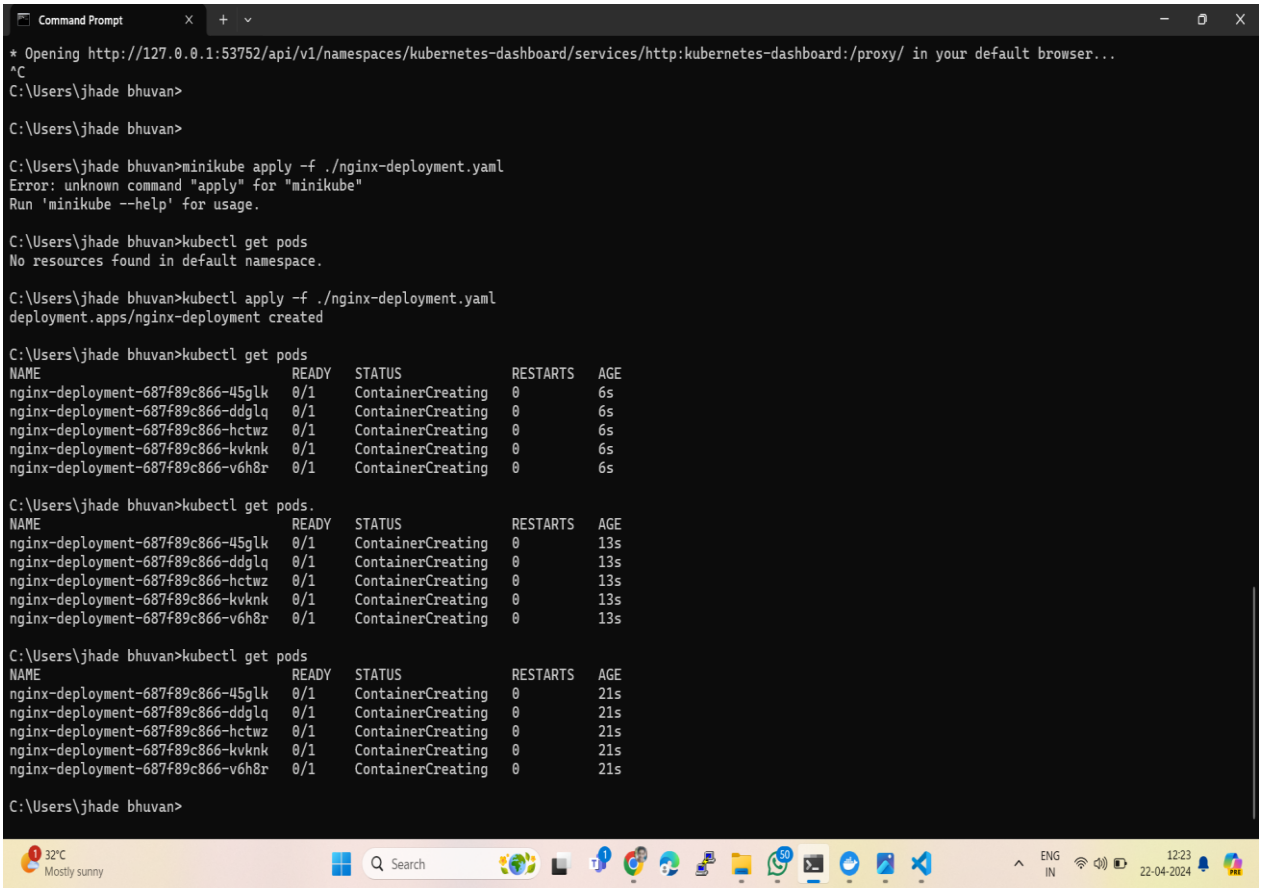


Created a Deployment file to deploy five pods :

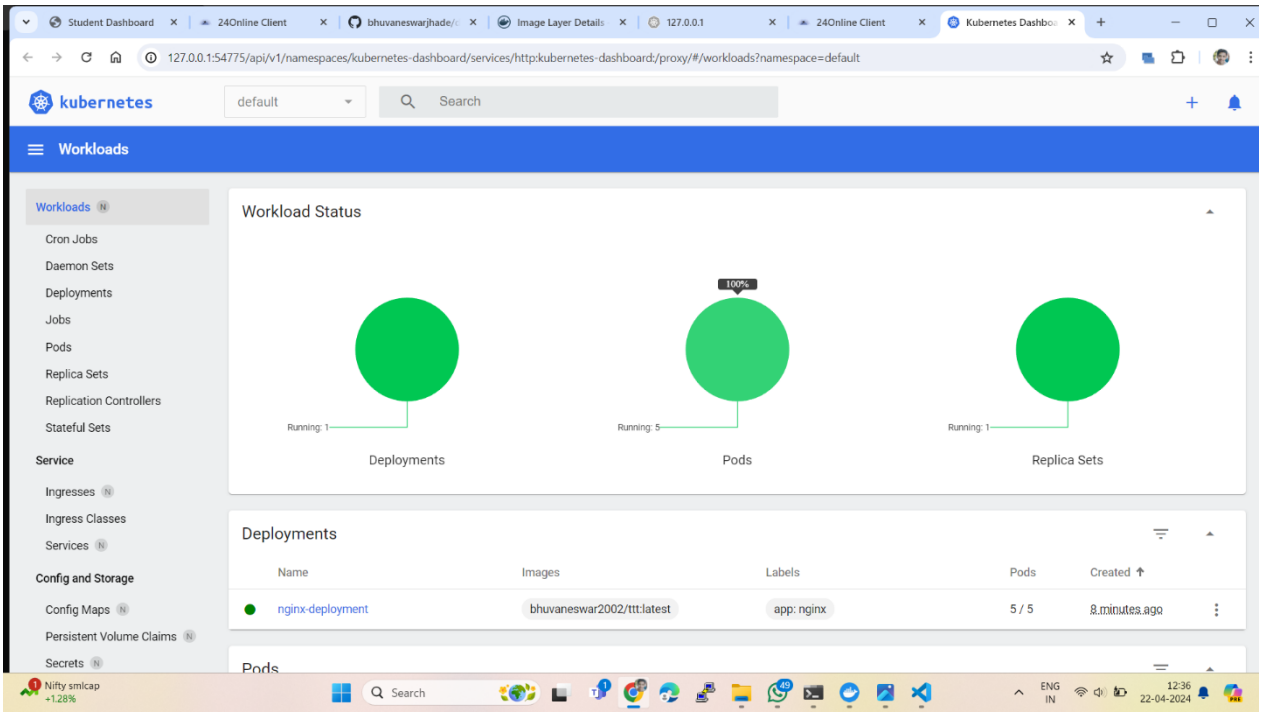


Command to run the deployment file:

Kubectl apply -f nginx-deployment

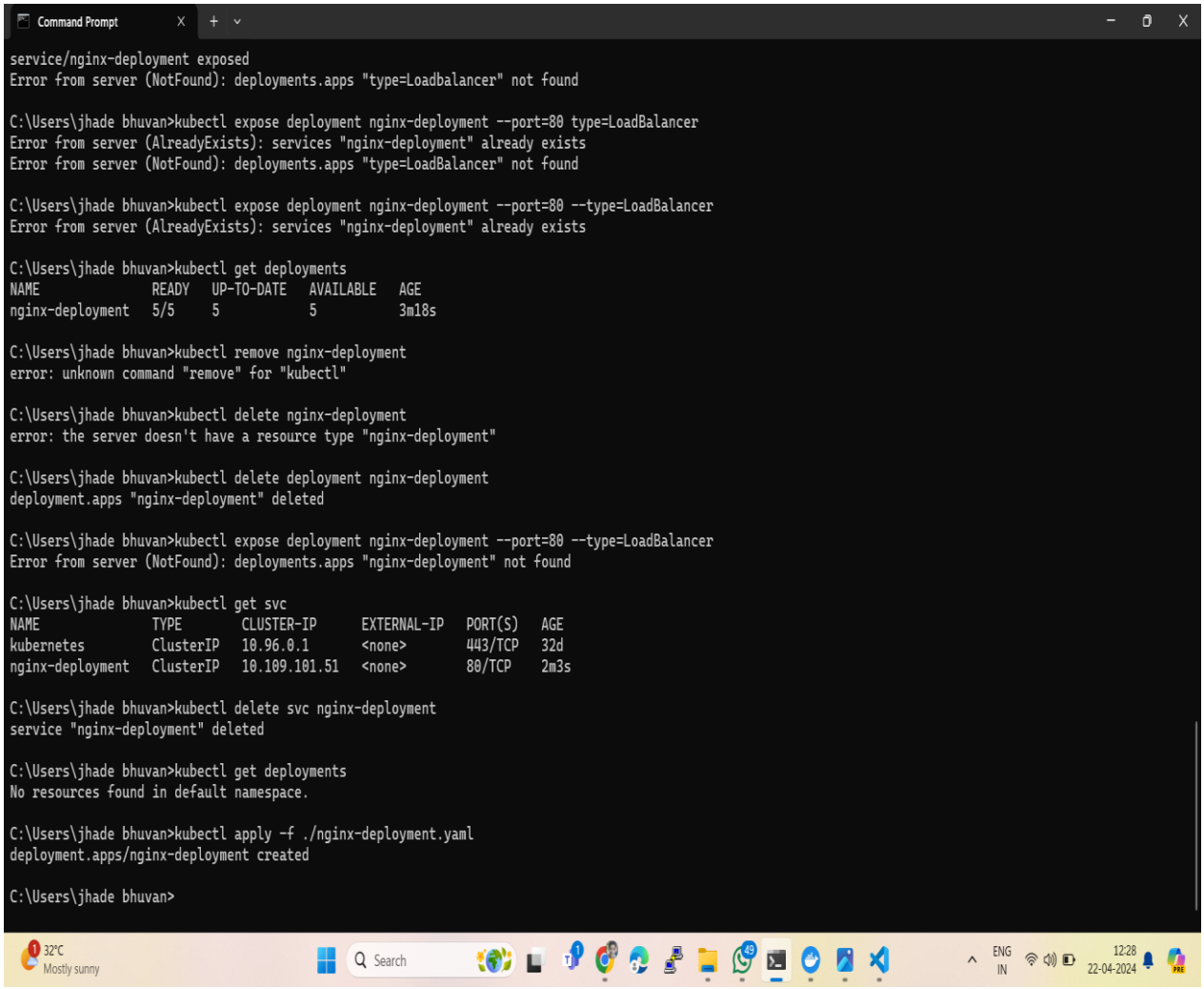


We can view our pods using Minikube dashboard:



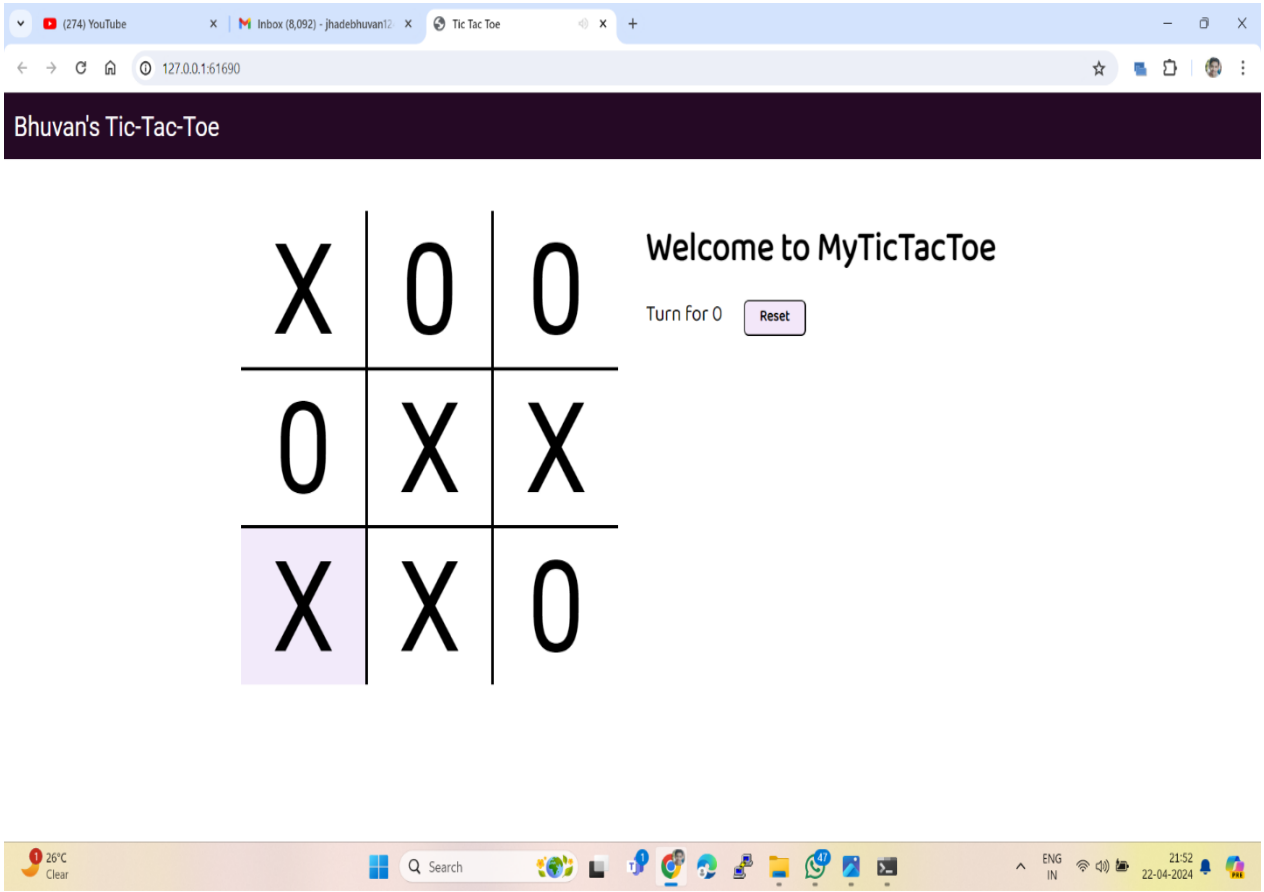
Created a service to enter into the cluster and access the application:

Kubectl expose deployment nginx-deployment –port=80 –type=LoadBalancer


Command Prompt
service/nginx-deployment exposed
Error from server (NotFound): deployments.apps "type=Loadbalancer" not found
C:\Users\jhade bhuvan>kubectl expose deployment nginx-deployment --port=80 type=LoadBalancer
Error from server (AlreadyExists): services "nginx-deployment" already exists
Error from server (NotFound): deployments.apps "type=LoadBalancer" not found
C:\Users\jhade bhuvan>kubectl expose deployment nginx-deployment --port=80 --type=LoadBalancer
Error from server (AlreadyExists): services "nginx-deployment" already exists
C:\Users\jhade bhuvan>kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
nginx-deployment 5/5 5 5 3m18s
C:\Users\jhade bhuvan>kubectl remove nginx-deployment
error: unknown command "remove" for "kubectl"
C:\Users\jhade bhuvan>kubectl delete nginx-deployment
error: the server doesn't have a resource type "nginx-deployment"
C:\Users\jhade bhuvan>kubectl delete deployment nginx-deployment
deployment.apps "nginx-deployment" deleted
C:\Users\jhade bhuvan>kubectl expose deployment nginx-deployment --port=80 --type=LoadBalancer
Error from server (NotFound): deployments.apps "nginx-deployment" not found
C:\Users\jhade bhuvan>kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 32d
nginx-deployment ClusterIP 10.109.101.51 <none> 80/TCP 2m3s
C:\Users\jhade bhuvan>kubectl delete svc nginx-deployment
service "nginx-deployment" deleted
C:\Users\jhade bhuvan>kubectl get deployments
No resources found in default namespace.
C:\Users\jhade bhuvan>kubectl apply -f ./nginx-deployment.yaml
deployment.apps/nginx-deployment created
C:\Users\jhade bhuvan>
32°C
Mostly sunny
Search
ENG
IN
12:28
22-04-2024

Finally we can view the web application using command

minikube service nginx-deployment



REFERENCES:

[1]. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[2] https://youtu.be/E2pP1MOfo3g?si=g2jI2v7ZXETq9-zt

[3] https://youtu.be/xNefZ51jHKg?si=6EOytNPSm8KUYEPX

[4] https://kubernetes.io/

[5] https://youtu.be/s_o8dwzRlu4?si=CFHZtQxK17oIzHNs

CONCLUSION:

The project focused on deploying a web application using Kubernetes orchestration within a local system environment, leveraging Git, Docker, and Minikube. It encompassed several critical components and processes aimed at establishing a robust and scalable infrastructure for hosting web applications.

The core of the project involved creating a Kubernetes cluster with Minikube, streamlining cluster setup and management on local machines. This cluster served as the foundation for orchestrating web application deployments.

Utilizing Docker containers, the project encapsulated and managed web applications within Kubernetes pods, ensuring consistency and efficiency across different environments.

To enhance reliability and scalability, various deployment strategies such as replication, auto-healing, and auto-scaling were implemented within the Kubernetes cluster. These strategies ensured service availability, automatic recovery from failures, and dynamic resource allocation based on workload demands.

Additionally, meticulous service configuration facilitated seamless communication between pods within the Kubernetes cluster and enabled external connectivity, enhancing overall functionality and user experience.

In summary, the project demonstrated a comprehensive approach to web application deployment within a Kubernetes environment using Minikube. By leveraging Kubernetes, Docker, and related tools, it aimed to establish a resilient and scalable infrastructure capable of supporting various web applications' hosting needs, fostering efficiency and innovation in software development and deployment processes.