Real Estate System – DevOps Enabled Deployment using IaC

Course Name: DevOps Foundation

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1	Samruddhi Muley	EN22CS304056
2	Payal Singh	EN22CS301701
3	Aarti Bamanke	EN22EE301001
4	Akshat Joshi	EN22CS304006
5	Punit Pawar	EN22CS303038
6	Shubhi Dubey	EN22CS304061

Group Name: 01D11

Project Number: *DO-27*

Industry Mentor Name: *Mr. Vaibhav*

University Mentor Name: *Prof. Shyam Patel*

Academic Year: *2026*

Table of Contents

Problem Statement & Objectives

1. Problem Statement

In traditional application deployment, infrastructure setup and software deployment are performed manually. This process is time-consuming, error-prone, and difficult to scale. Manual configuration often leads to inconsistencies between development and production environments, causing deployment failures and system downtime.

In the real estate domain, applications must be reliable, scalable, and easily updatable to handle multiple users and frequent data changes. Therefore, there is a need for an automated, repeatable, and efficient deployment solution.

This project addresses the problem by implementing a Real Estate web application using DevOps practices and Infrastructure as Code (IaC). The goal is to automate infrastructure provisioning, containerize the application, and enable continuous integration and deployment to ensure scalability, portability, and reliability.

Additionally, the absence of automated deployment pipelines increases operational overhead and delays application updates. Managing infrastructure manually also makes it difficult to ensure consistency, security, and reliability across environments. As user demand grows, traditional deployment approaches fail to provide the flexibility required for rapid scaling and continuous delivery.

This highlights the need for a modern DevOps-based solution that integrates automation, containerization, and cloud technologies to streamline application deployment and infrastructure management.

2. Project Objectives

The main objectives of this project are:

- To design and develop a web-based Real Estate application using Flask.
- To automate infrastructure provisioning using Terraform (Infrastructure as Code).
- To containerize the application using Docker for portability and consistency.
- To deploy and manage containers using Kubernetes for scalability and high availability.
- To implement a CI/CD pipeline for automated build and deployment.
- To host the application on AWS EC2 for cloud-based accessibility.
- To demonstrate practical implementation of DevOps tools and automation techniques.

3. Scope of the Project

The scope of this project includes the design, development, containerization, and automated deployment of a Real Estate web application using modern DevOps practices and Infrastructure as Code (IaC).

The project focuses on building a scalable, repeatable, and automated deployment architecture using:

- Terraform for infrastructure provisioning
- Docker for containerization
- Kubernetes for orchestration
- Amazon Web Services (EC2) for cloud infrastructure
- GitHub Actions or Jenkins for CI/CD automation

1. Application Scope

- Development of a minimal Real Estate web application using Flask
- Basic frontend interface for property display
- Exposure of application port for containerized deployment

2. Infrastructure Scope

- Provisioning of virtual machines (EC2 instances)
- Configuration of networking components (security groups, ports)
- Creation of container runtime environment
- Infrastructure provisioning using declarative configuration files
- Repeatable environment setup across development and deployment

3. Containerization Scope

- Creation of Docker image using Dockerfile
- Application packaging with dependencies
- Standardized runtime environment across systems
- Image deployment to Kubernetes cluster

4. Orchestration Scope

- Deployment using Kubernetes manifests
- Service exposure via NodePort or LoadBalancer
- Pod lifecycle management
- Basic scalability via replica configuration

5. CI/CD Scope

- Version control integration using Git
- Automated pipeline triggered on code push
- Automatic Docker build and deployment
- Automated infrastructure provisioning (if configured)
- Continuous integration and deployment validation

6. Operational Scope

- Log verification during deployment
- Health endpoint monitoring
- Automated redeployment on failure (via Kubernetes)
- Idempotent infrastructure provisioning using Terraform

Proposed Solution

The proposed solution is to develop and deploy a Real Estate web application using modern DevOps practices and Infrastructure as Code (IaC). The system is designed to automate infrastructure provisioning, application deployment, and container orchestration to ensure scalability, reliability, and portability.

The application is developed using the Flask framework, which handles backend logic, routing, and interaction with the SQLite database. The frontend is designed using HTML, CSS, and Jinja2 templates to provide a user-friendly interface for browsing property listings and viewing detailed property information.

To ensure portability and consistency across different environments, the application is containerized using Docker. The Docker image is built and stored in Docker Hub, allowing easy deployment across cloud environments.

Infrastructure provisioning is automated using Terraform, which follows the Infrastructure as Code (IaC) approach. Terraform scripts create and configure AWS EC2 instances and necessary networking components without manual intervention. This ensures repeatable and error-free infrastructure setup.

Kubernetes (K3s) is used for container orchestration to manage deployment, scaling, and availability of the application containers. It ensures that the application remains available even if a container fails.

A CI/CD pipeline using GitHub Actions or Jenkins is implemented to automate the build and deployment process. Whenever code is pushed to the repository, the pipeline automatically builds the Docker image, pushes it to Docker Hub, provisions infrastructure if required, and deploys the updated application.

Overall, the proposed solution provides an automated, scalable, and efficient cloud-based deployment system that minimizes manual effort, reduces errors, and improves reliability.

1. Key Features

• Web-based Real Estate Property Listing System

The system provides a web-based interface that allows users to view property listings, access property details, and interact with the application through a browser. The application is built using Flask with HTML, CSS, and Jinja2 templates for rendering dynamic content.

• Infrastructure as Code (IaC) Using Terraform

All cloud infrastructure components such as virtual machines and networking configurations are defined declaratively using Terraform scripts. This enables repeatable, version-controlled, and automated provisioning of infrastructure, eliminating manual setup errors.

• Docker-Based Containerization Using Docker

The application is packaged into Docker containers along with its dependencies, ensuring consistent execution across development, testing, and production environments. Containerization improves portability and simplifies deployment.

• Kubernetes (K3s) Orchestration for Scalability

K3s is used as a lightweight Kubernetes distribution to orchestrate containerized workloads. It manages pod lifecycle, service exposure, and basic scalability, enabling efficient container deployment on cloud instances.

• CI/CD Pipeline for Automated Deployment

A Continuous Integration and Continuous Deployment pipeline is implemented using GitHub Actions or Jenkins. Any code change pushed to the GitHub repository automatically triggers build, containerization, and deployment processes, ensuring rapid and consistent application delivery.

• Cloud Hosting on Amazon Web Services EC2

The application is deployed on AWS EC2 instances, providing scalable and reliable cloud infrastructure. Terraform provisions the EC2 instances, while Kubernetes manages application deployment on top of these virtual machines.

• Secure Access Control Using AWS Security Groups and SSH Keys

Network access is restricted through AWS Security Groups, allowing only necessary ports such as SSH and application ports. SSH key-based authentication is used to securely access cloud instances, enhancing system security.

• Scalable and Automated DevOps-Based Architecture

The overall architecture follows DevOps best practices by integrating source control, CI/CD, containerization, orchestration, and IaC. This design supports scalability, automation, and rapid recovery while minimizing manual intervention.

• Modular and Extensible System Design

The project architecture is modular, allowing future enhancements such as monitoring, auto-scaling, logging, security hardening, and database upgrades without major redesign.

2. Overall Architecture / Workflow

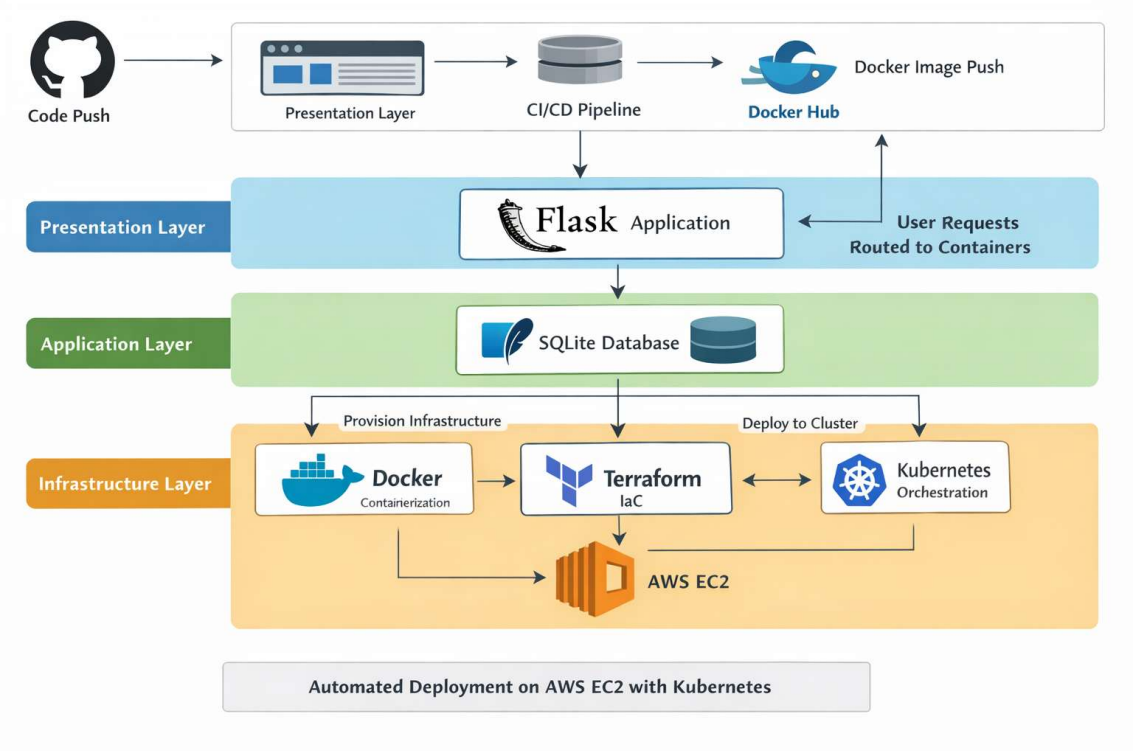
The proposed solution follows a layered and automated architecture that integrates application development with DevOps practices.

The system consists of three main layers: Presentation Layer, Application Layer, and Infrastructure Layer. The Presentation Layer includes HTML, CSS, and Jinja2 templates that interact with users through a web browser. The Application Layer is built using Flask, which processes requests, manages routing, and interacts with the SQLite database.

The Infrastructure Layer includes Docker for containerizing the application, Kubernetes for container orchestration, and AWS EC2 for cloud hosting. Infrastructure provisioning is automated using Terraform scripts.

The workflow begins when code is pushed to the GitHub repository. The CI/CD pipeline is triggered automatically, which builds a Docker image and pushes it to Docker Hub. Terraform provisions the required AWS infrastructure, and Kubernetes deploys the updated container. Users access the application through the EC2 public IP, and requests are routed through Kubernetes services to the running container.

This architecture ensures automation, scalability, portability, and reliability.



2.1 Process Flow

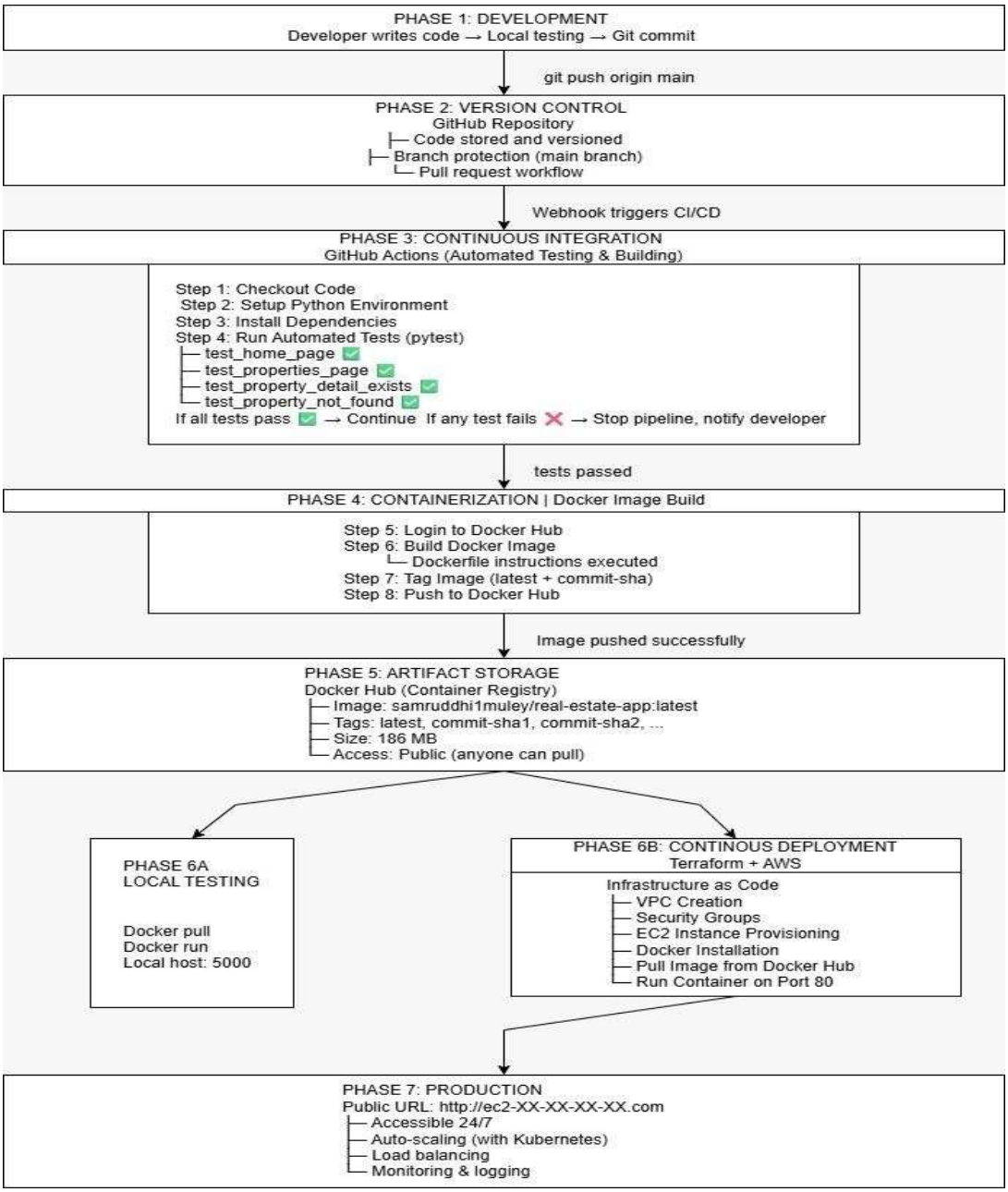
The deployment pipeline ensures zero manual intervention after code push. Automated testing and image building reduce the risk of broken deployments.

The runtime process ensures stateless behaviour to support horizontal scaling in Kubernetes environments.

User Flow:

1. User opens browser
2. Sends HTTP request to EC2 Public IP
3. Kubernetes service forwards request to container
4. Flask application processes request
5. Response displayed in browser CI/CD Flow:
    1. Code pushed to Git repository
    2. Pipeline triggers automatically
    3. Docker image built

4. Image pushed to registry
5. Terraform provisions infrastructure
6. Kubernetes deploys updated container



2.2 Information Flow

Information flow is unidirectional in request-response format, ensuring clarity in data processing.

User Browser

↓

AWS EC2 Public IP

↓

Kubernetes Service

↓

Kubernetes Pod

↓

Docker Container

↓

Flask Application

↓

HTML Response

2.4 Components Design

Major Components:

- **GitHub Repository**
  Stores source code
  Stores Terraform configuration
- **GitHub Actions**
  CI/CD pipeline
  Builds Docker image
  Pushes image to Docker Hub
- **Docker**
  Containerizes application

- **Docker Hub**

  Stores Docker image

- **Terraform**

  Provisions infrastructure

  Creates EC2 instance

  Configures networking

- **AWS EC2**

  Hosts Kubernetes cluster

- **Kubernetes (K3s)** Orchestrates containers

  Manages pods and services

2.5 Key Design Considerations

- Infrastructure automation using Terraform
- Container portability using Docker
- Scalability using Kubernetes
- Automated deployment using CI/CD
- Cloud-based hosting for high availability
- Security via controlled acce

3. Tools & Technologies Used

- **Programming Language:** Python
- **Backend Framework:** Flask
- **Frontend:** HTML, CSS, Jinja2
- **Database:** SQLite
- **Containerization:** Docker
- **Container Registry:** Docker Hub
- **Infrastructure as Code:** Terraform
- **Cloud Platform:** Amazon Web Services (AWS EC2)
- **Orchestration:** Kubernetes (K3s)
- **Version Control:** Git & GitHub
- **CI/CD:** GitHub Actions / Jenkins

Results & Output

The Real Estate System was successfully developed and deployed using DevOps practices and Infrastructure as Code. The application was built using Flask and was containerized using Docker to ensure portability across environments.

Terraform was used to provision AWS EC2 infrastructure automatically, eliminating the need for manual server setup. The EC2 instance was successfully created and configured with the required networking and security settings. Kubernetes (K3s) was installed and configured on the EC2 instance to manage and orchestrate the Docker containers.

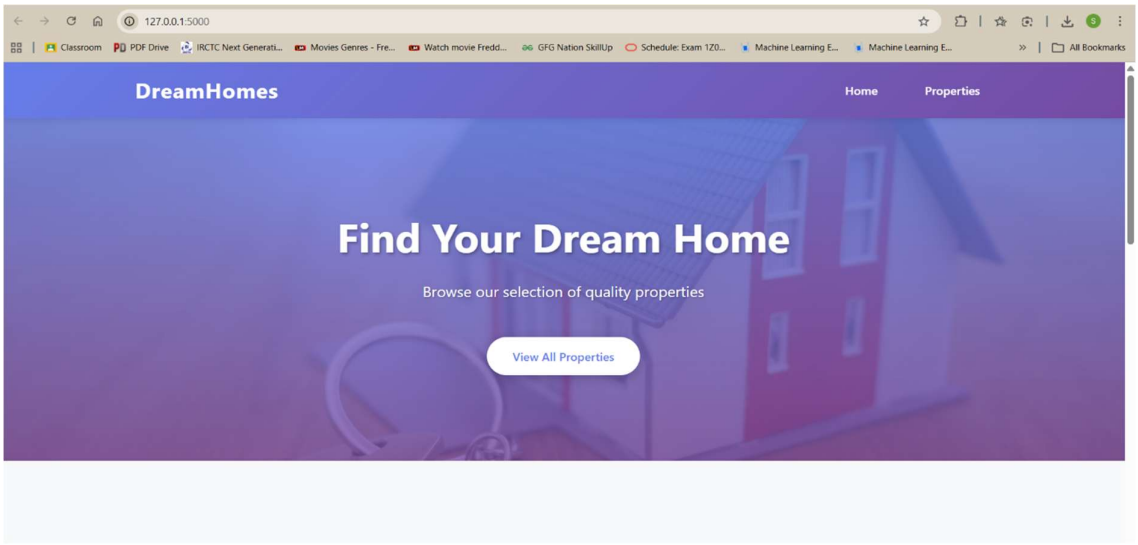
The CI/CD pipeline was successfully implemented using GitHub Actions or Jenkins. Whenever code was pushed to the repository, the pipeline automatically built the Docker image, pushed it to Docker Hub, and deployed the updated application to the Kubernetes cluster.

The application was accessible through the EC2 public IP address, and users were able to browse property listings and view detailed property information without any deployment issues.

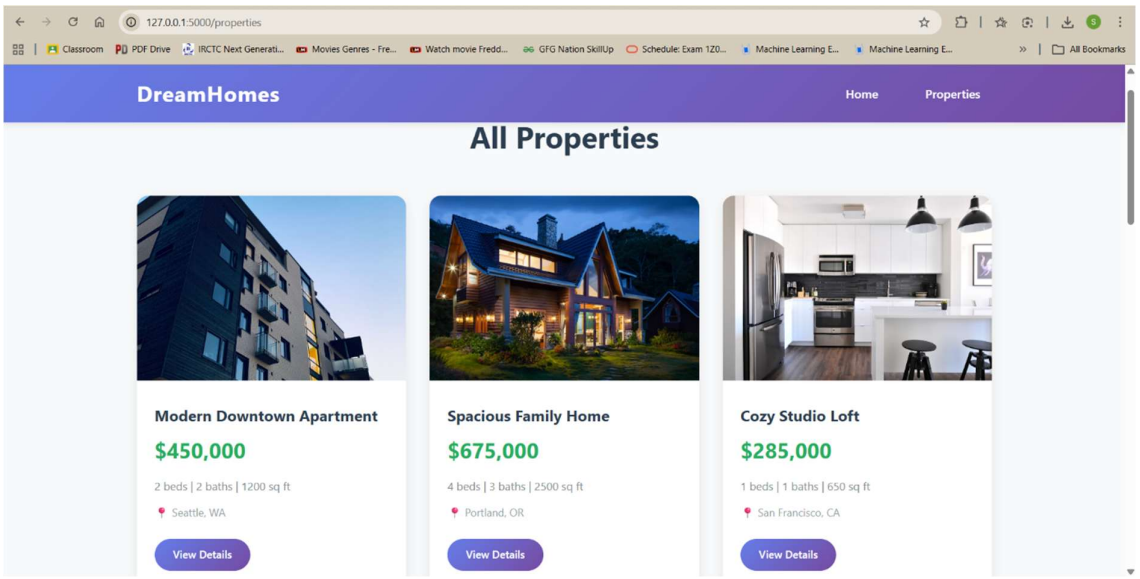
Overall, the output confirms that the system achieved automated infrastructure provisioning, containerized deployment, scalability, and reliable cloud-based performance.

1. Screenshots / Outputs

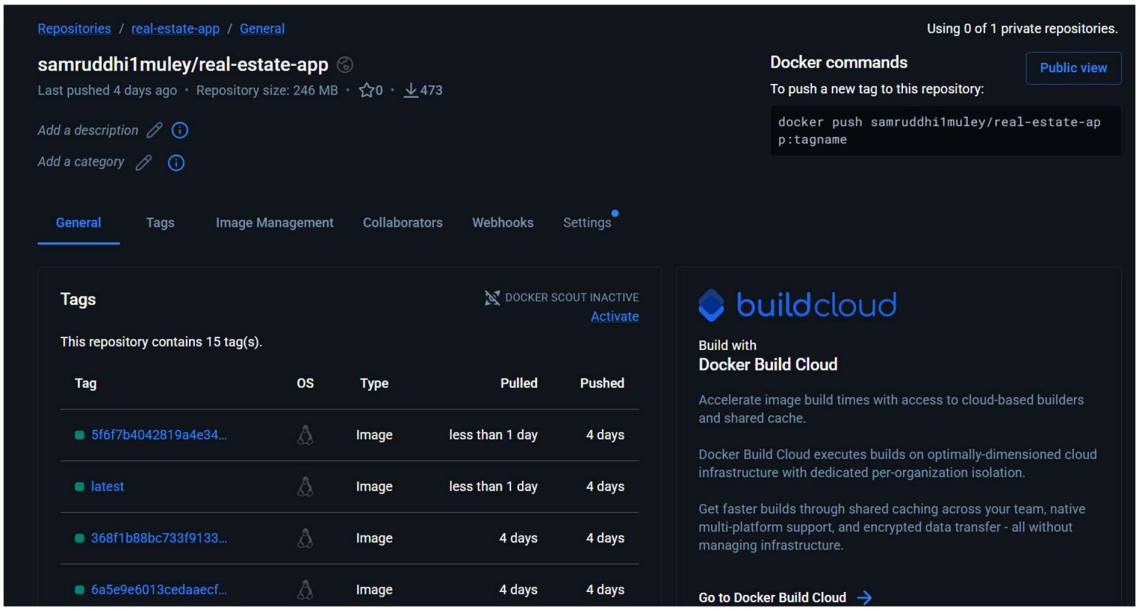
- **Application Output**
  - Home page of Real Estate website



- Property listing page



- Docker Execution

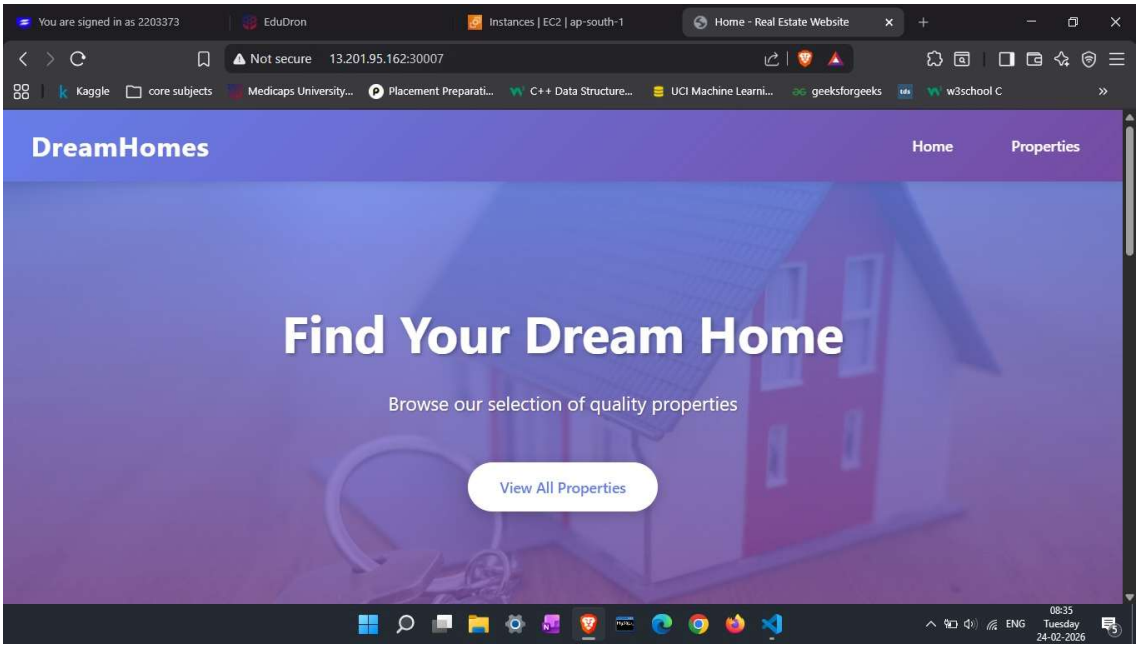


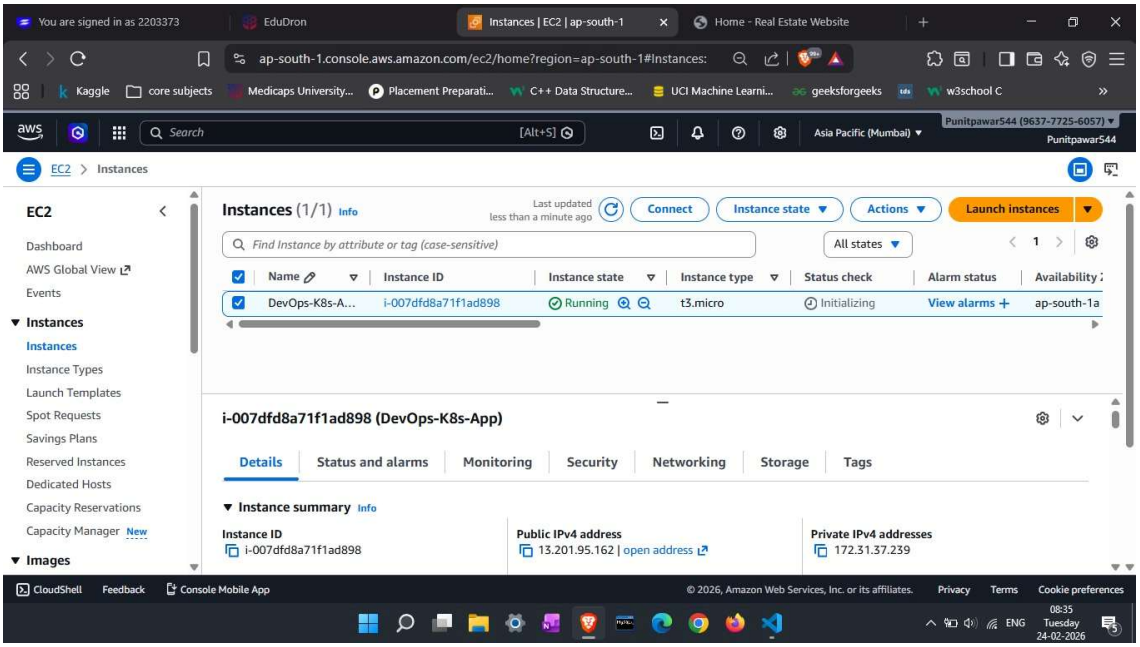
Docker image created successfully.

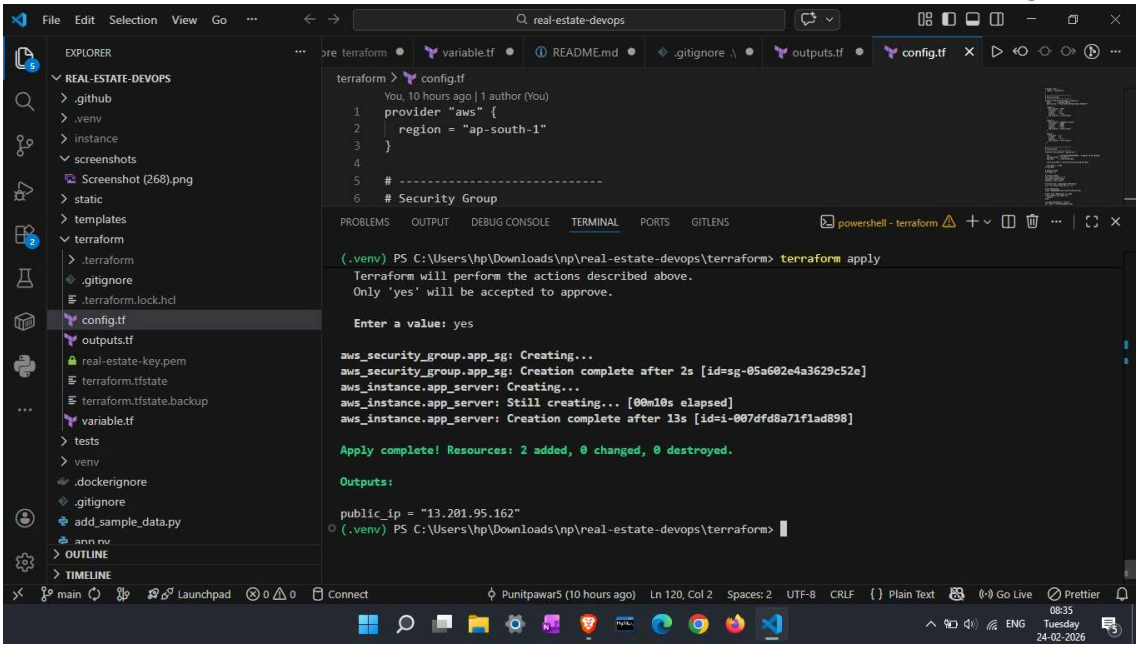
- **Kubernetes Deployment**

Screenshots of:

kubectl get pods
kubectl get services


You are signed in as 2203373
EduDron
Instances | EC2 | ap-south-1
Home - Real Estate Website
Not secure 13.201.95.162:30007
DreamHomes
Home
Properties
Find Your Dream Home
Browse our selection of quality properties
View All Properties

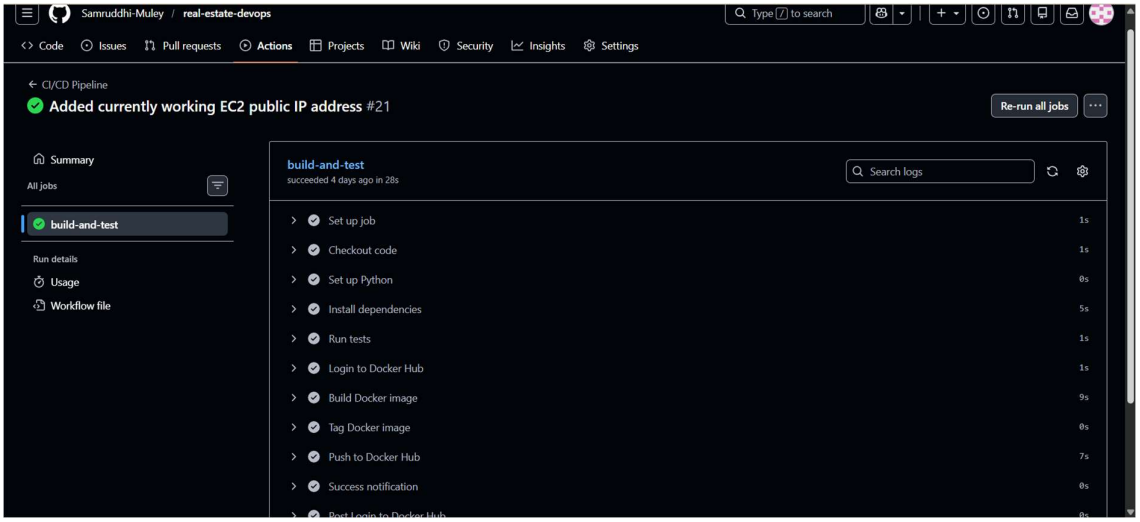

You are signed in as 2203373
EduDron
Instances | EC2 | ap-south-1
Home - Real Estate Website
ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Instances:
EC2 > Instances
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
DevOps-K8s-A...
i-007dfd8a71f1ad898
Running
t3.micro
Initializing
View alarms
ap-south-1a
i-007dfd8a71f1ad898 (DevOps-K8s-App)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Public IPv4 address
13.201.95.162 | open address
Private IPv4 addresses
172.31.37.239



Text:

Kubernetes pods were observed in running state, and services exposed the application using NodePort/LoadBalancer, validating successful orchestration.

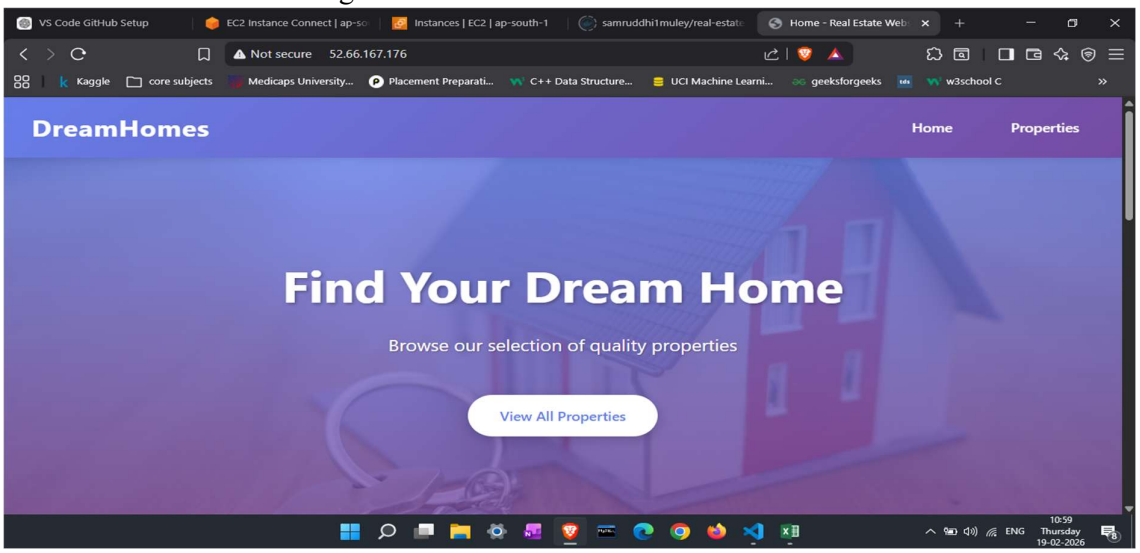
- **CI/CD Pipeline**
  - GitHub Actions



The CI/CD pipeline was triggered automatically on Git push and completed all stages including Docker image build and Kubernetes deployment.

- **Infrastructure**

- EC2 instance running on Amazon Web Services



Cloud infrastructure was provisioned successfully using Terraform, and application workloads were hosted on AWS EC2.

Conclusion

The Real Estate System project successfully demonstrated the integration of web application development with modern DevOps practices. The application was developed using Flask and deployed using an automated Infrastructure as Code approach with Terraform. Docker was used to containerize the application, ensuring portability and consistency across environments, while Kubernetes provided scalability and efficient container management.

The implementation of a CI/CD pipeline automated the build and deployment process, reducing manual effort and minimizing configuration errors. The system was successfully deployed on AWS EC2 and was accessible to users through a public IP address.

In conclusion, the project achieved its objective of creating a scalable, automated, and reliable cloud-based deployment solution. It highlights the importance of DevOps tools and automation in improving efficiency, consistency, and overall system performance in modern application development.

Future Scope & Enhancements

The Real Estate System can be further enhanced by adding advanced features and improving scalability, security, and performance. Future improvements may include implementing user authentication and role-based access control to provide secure login for administrators and users. Property management features such as adding, updating, and deleting listings through secure APIs can also be introduced.

The system can be upgraded by replacing the SQLite database with a scalable database like MySQL or PostgreSQL for handling large amounts of data. Integration of caching mechanisms such as Redis can improve performance and reduce response time. Automated backup strategies and monitoring tools can be implemented to enhance reliability.

Additionally, the application can be deployed in a multi-region cloud environment for higher availability and disaster recovery. Integration with load balancers and auto-scaling groups can further improve performance during high traffic. Advanced CI/CD pipelines with automated testing and security scanning can also be implemented.

Overall, these enhancements will make the system more secure, scalable, efficient, and production-ready for real-world deployment.