

**Submitted By:**

**Name: Nayan Tank**

**Reg No: O23MCA110844**

# **DECLARATION**

I, Nayan Tank, bearing Registration Number O23MCA110844, a student of the Master of Computer Applications at the Centre for Distance & Online Education, Chandigarh University, hereby declare that the Major Project report entitled "End-to-End CI/CD Deployment for MERN Application on GKE using Terraform, Jenkins, Helm, Prometheus & Grafana" is an original work done by me.

The information and data given in the report are authentic to the best of my knowledge and belief. This work has not been submitted in part or full to any other university or institution for the award of any degree or diploma.

**Nayan Tank**

**Reg. No.: O23MCA110844**

Date: 25/05/2025

# **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to everyone who supported me throughout the development of this project. I extend my thanks to Chandigarh University and the Centre for Distance & Online Education for providing the opportunity and resources to undertake this major project. I am thankful to my project guide and faculty members for their guidance, feedback, and encouragement throughout this journey.

I also thank my peers for their constructive discussions and my family for their constant support and motivation. This project would not have been possible without the contributions and encouragement of all the above individuals.

**Nayan Tank**

# **ABSTRACT**

This project demonstrates the implementation of a complete DevOps CI/CD pipeline to deploy a containerized MERN (MongoDB, Express, React, Node.js) application on Google Kubernetes Engine (GKE). The pipeline leverages modern DevOps tools and practices including Terraform for infrastructure provisioning, Helm for application deployment, Jenkins for CI/CD automation, and Prometheus-Grafana for cluster monitoring.

The project flow involves automating GKE cluster creation using Terraform, deploying Jenkins and monitoring tools via Helm, configuring a Jenkins pipeline to build a Docker image of the MERN application and push it to Docker Hub, and finally deploying it on GKE. The setup ensures continuous integration and deployment, observability, and infrastructure reproducibility.

This solution demonstrates a practical and scalable approach to cloud-native application deployment and monitoring in real-world enterprise environments.

# **TABLE OF CONTENTS**

1. Introduction
   1. Background
   2. Problem Statement
   3. Objectives
   4. Scope
   5. Motivation
   6. Organization of Report
2. Software Development Life Cycle (SDLC)
   1. SDLC Model Used
   2. Phases of Development
      1. Requirement Gathering and Analysis
      2. Design
      3. Development
      4. Testing
      5. Deployment
      6. Maintenance
3. System Analysis and Design
   1. System Architecture
   2. Infrastructure Design (Terraform)
   3. Helm Chart Configuration
   4. Jenkins Pipeline Architecture
   5. Security and Access Control
4. Tools and Technologies Used
5. Implementation
   1. GKE Cluster Provisioning with Terraform
   2. Helm Installation of Jenkins and MERN app
   3. Jenkins Configuration and CI/CD Pipelines
   4. Kubernetes Deployment Strategy
   5. Screenshots & Demo
6. Testing
   1. Unit Testing
   2. Integration Testing
   3. System Testing
7. Results and Observations
8. Challenges Faced and Solutions
9. Conclusion
   1. Summary of Work
   2. Key Achievements
   3. Learning Outcomes
   4. Limitations
   5. Future Scope
10. Bibliography (APA Style)

# **Chapter 1: Introduction**

## **1.1 Background**

The rise of cloud-native architectures and DevOps practices has transformed how software is built, tested, and deployed. Organizations are moving from traditional monolithic deployments to containerized microservices hosted on Kubernetes clusters. To manage these environments efficiently, Infrastructure as Code (IaC) tools like Terraform and package managers like Helm are used to provision and configure cloud infrastructure automatically.

Simultaneously, Continuous Integration and Continuous Deployment (CI/CD) pipelines have become integral for reducing time-to-market and enhancing deployment consistency. Jenkins, a popular automation server, helps implement CI/CD pipelines for building and deploying applications. Integrating Jenkins with Kubernetes (GKE) allows dynamic deployment of workloads in an orchestrated, automated fashion and monitoring using Prometheus and Grafana.

## **1.2 Problem Statement**

Manual deployment of applications on Kubernetes can be error-prone, time-consuming, and non-repeatable. The lack of automation creates bottlenecks and increases the risk of configuration drift. Additionally, setting up scalable and secure infrastructure for deploying applications often requires in-depth knowledge of cloud services and container orchestration.

There is a need for a reliable, automated, and reproducible solution that integrates infrastructure provisioning, software deployment, and application monitoring on a managed Kubernetes platform.

## **1.3 Objectives**

* To automate GKE cluster creation using Terraform.
* To deploy MERN app and Jenkins using Helm charts.
* Real time monitoring using Prometheus and Grafana.
* To configure Jenkins with a GitHub webhook for CI/CD pipeline execution.
* To enable automatic deployment of a full-stack application to GKE upon a push event to the GitHub repository's main branch.
* To build Docker images via Jenkins and push them to Docker Hub.
* To pull Docker images from Docker Hub for deployment on GKE.
* To implement secure access controls and cloud best practices.
* To deploy MERN app, Prometheus/Grafana and Jenkins Helm charts

## **1.4 Scope**

The project focuses on:

* Provisioning cloud infrastructure using Terraform.
* Deploying Prometheus/Grafana, Jenkins and MERN app using Helm on GKE.
* Integrating GitHub with Jenkins using a webhook that triggers on push to the main branch.
* Automating the build and deployment process for a full-stack application stored in a GitHub repository.
* Building Docker images in Jenkins and pushing them to Docker Hub.
* Pulling Docker images from Docker Hub for deployment on GKE.
* Leveraging open-source tools and GCP Free Tier wherever applicable.

## **1.5 Motivation**

This project was motivated by the need to bridge the gap between software development and deployment through automation and infrastructure-as-code principles. By implementing a fully automated CI/CD pipeline on GKE, it demonstrates a complete DevOps lifecycle in a cloud-native environment.

## **1.6 Hardware Specification**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Requirement** | **Recommended** |
| **Local Machine / Dev Laptop** |  |  |
| **CPU** | Dual-core 2.0 GHz | Quad-core 2.5+ GHz |
| **RAM** | 8 GB | 16 GB or higher |
| **Storage** | 20 GB free | SSD with 50+ GB |
| **Network** | Stable internet | High-speed broadband |
| **Cloud Infrastructure (GCP)** |  |  |
| **GKE Cluster Nodes** | 2 vCPUs, 2 GB RAM per node (e2-medium) | 2–3 nodes with autoscaling enabled |
| **Persistent Disk (Jenkins PVC)** | 10 GB | 20+ GB |
| **Static IP / LoadBalancer** | Required | Required |

## **1.7 Software Specification**

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Version / Notes** |
| **Operating System** | Ubuntu / macOS / Windows (WSL) | Any recent stable version |
| **Containerization** | Docker | Docker Engine 20+ |
| **Infrastructure as Code** | Terraform | v1.3+ |
| **Kubernetes CLI** | kubectl | v1.24+ |
| **Helm** | Helm | v3.x |
| **CI/CD** | Jenkins | Latest LTS (with Docker, GitHub plugins) |
| **Frontend** | React (Vite) | React 18, Vite 4 |
| **Backend** | Node.js / Express | Node.js 18+ |
| **Database (Cloud)** | MongoDB Atlas | Free tier or paid cluster |
| **Monitoring** | Prometheus + Grafana | Latest stable charts via Helm |
| **VCS** | Git + GitHub | With webhook access to Jenkins |
| **Cloud Platform** | Google Cloud Platform | GKE, IAM, GCR, VPC, etc. |
| **DNS (optional)** | Cloud DNS / Any provider | For domain-based app access |

## **1.8 Organization of Report**

* **Chapter 1** introduces the problem, objectives, and motivation.
* **Chapter 2** discusses the SDLC model and development phases.
* **Chapter 3** details the design and architecture of the system.
* **Chapter 4** covers the tools and technologies used.
* **Chapter 5** explains implementation steps with screenshots.
* **Chapter 6** outlines testing strategies.
* **Chapter 7** summarizes the results.
* **Chapter 8** discusses challenges and solutions.
* **Chapter 9** concludes with key learnings and future scope.
* **Chapter 10** lists references in APA format.

# **Chapter 2: Software Development Life Cycle (SDLC)**

## **2.1 SDLC Model Used**

For the development of this project, the **Agile SDLC model** was adopted. Agile promotes iterative development and continuous feedback, which aligns perfectly with the dynamic nature of DevOps and CI/CD workflows. As components such as Terraform modules, Jenkins pipelines, Prometheus/Grafana and Helm charts are independently testable and deployable, Agile allowed flexibility and faster adaptation to changes.

Each sprint was focused on building and validating one component of the system—such as provisioning infrastructure, configuring CI/CD, or securing access to the pipeline.

## **2.2 Phases of Development**

### **Requirement Gathering and Analysis**

* Identified the need for automated deployment of containerized full-stack applications.
* Defined the project scope: Provision GKE cluster, deploy Jenkins, Prometheus/Grafana and MERN app, trigger CI/CD pipeline from GitHub webhook.
* Evaluated tools and cloud services to be used: Terraform, Helm, Jenkins, Docker Hub, GCP (GKE, IAM).

### **Design**

* Created architecture diagram showing Terraform provisioning GKE, Jenkins executing CI/CD, and Docker images hosted on Docker Hub.
* Designed GitHub-to-Jenkins integration through a webhook.
* Outlined pipeline stages: checkout, build, Docker image push, deploy to GKE.
* Defined Helm configurations for deploying Jenkins, Prometheus/Grafana and MERN app.

### **Development**

* Wrote Terraform code to create VPC, GKE cluster, and IAM roles.
* Created Helm values files and installed Jenkins and Prometheus/Grafana and MERN app using helmfile.
* Developed the Jenkins pipeline (Jenkinsfile) to build Docker images and push to Docker Hub.
* Configured GitHub webhook for triggering Jenkins on main branch push events.

### **Testing**

* Unit tested individual Terraform and Jenkins modules.
* Performed integration testing for GitHub-Jenkins and Jenkins-GKE connectivity.
* Verified Docker image build and push functionality.
* Ran test deployments from Jenkins and validated application behavior on GKE.

### **Deployment**

* Final setup deployed a working CI/CD pipeline:
  + Push to GitHub main branch triggers Jenkins job.
  + Jenkins builds Docker image and pushes to Docker Hub.
  + Jenkins deploys the application using kubectl or Helm to GKE.
  + Prometheus/Grafana for monitoring the cluster
* Helm was also used to manage upgrades.

### **Maintenance**

* Implemented proper IAM permissions for secure access.
* Documented Helm values and Terraform variables for future updates.
* Maintained Docker Hub repository hygiene with version tagging.
* Monitored GCP billing to stay within free tier usage.

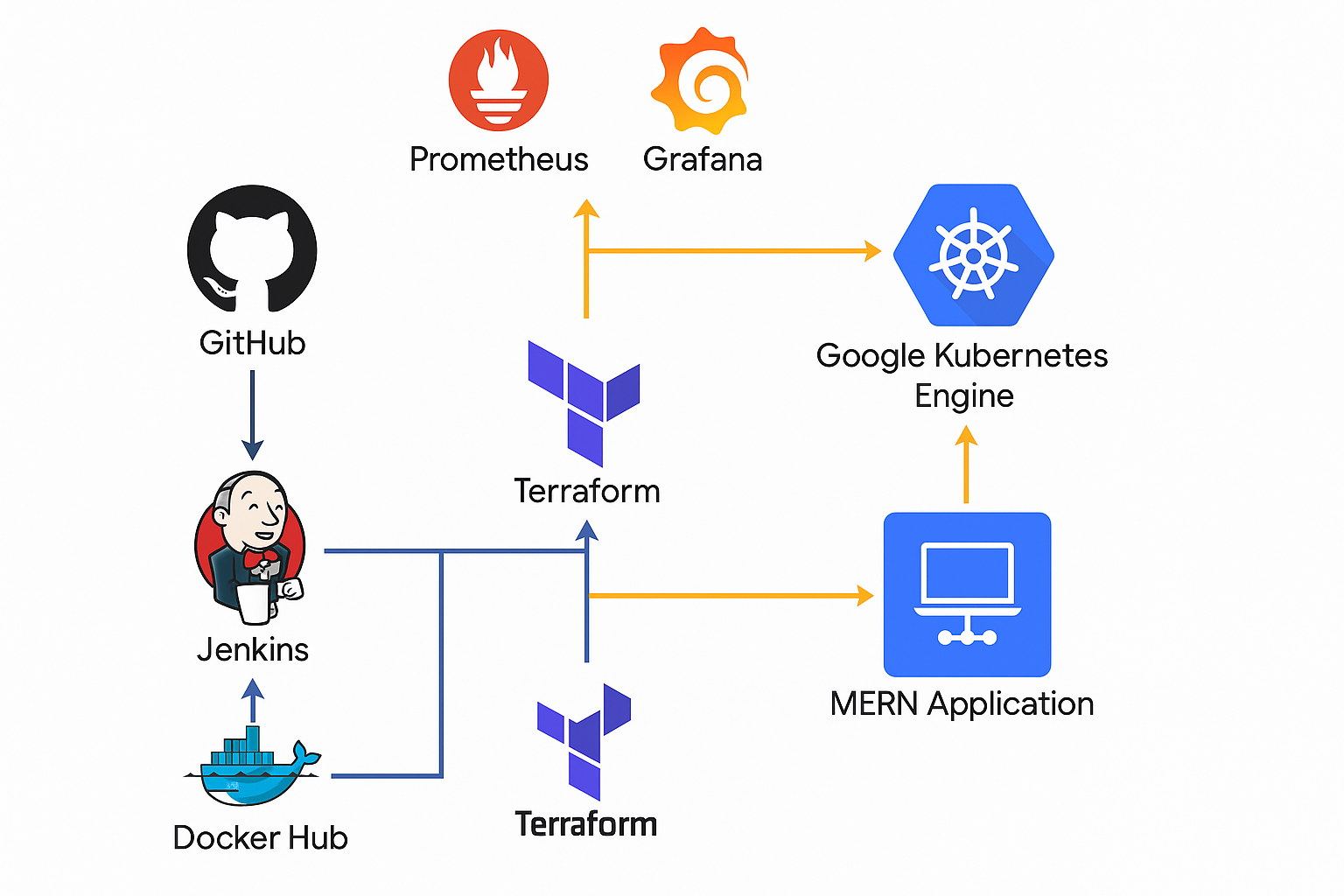
This SDLC approach enabled iterative delivery with reliable validation at each stage, ensuring a robust and automated DevOps project outcome.

# **Chapter 3: System Analysis and Design**

## **3.1 System Architecture**

The system architecture is designed to support a cloud-native CI/CD pipeline that automates infrastructure provisioning, continuous integration, and continuous deployment. Below are the key components and their interactions:

* **Terraform**: Provisions the Google Kubernetes Engine (GKE) cluster, VPC, and IAM roles.
* **Helm**: Manages the installation and configuration of Jenkins and Prometheus/Grafana on the GKE cluster.
* **Prometheus & Grafana:** Real time cluster monitoring
* **Jenkins**: Acts as the CI/CD orchestrator. It pulls code from GitHub, builds Docker images, pushes them to Docker Hub, and deploys to GKE.
* **GitHub**: Hosts the full-stack application source code and triggers Jenkins jobs via webhook on main branch push.
* **Docker Hub**: Stores Docker images created by Jenkins, which are then pulled by GKE for deployment.
* **GKE**: Hosts the Kubernetes workloads, providing container orchestration, scaling, and management.



The architecture is modular and follows best practices for separation of concerns, automation, and scalability.

## **3.2 Infrastructure Design (Terraform)**

Terraform is used to define the entire infrastructure as code. Key resources provisioned include:

* VPC with subnet configuration
* GKE cluster with node pools
* IAM service accounts and roles

Below is the tree structure of my terraform code with some code snippets or you can visit <https://github.com/nayan-tank/cu-project> for more details.

.

├── Infra

│ ├── main.tf

│ ├── output.tf

│ ├── provider.tf

│ ├── variables.tf

│ └── variables.tfvars

├── modules

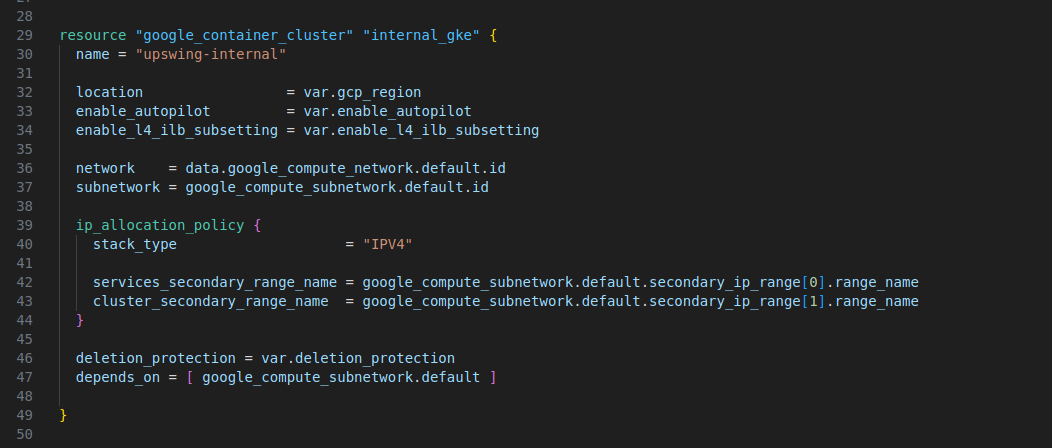
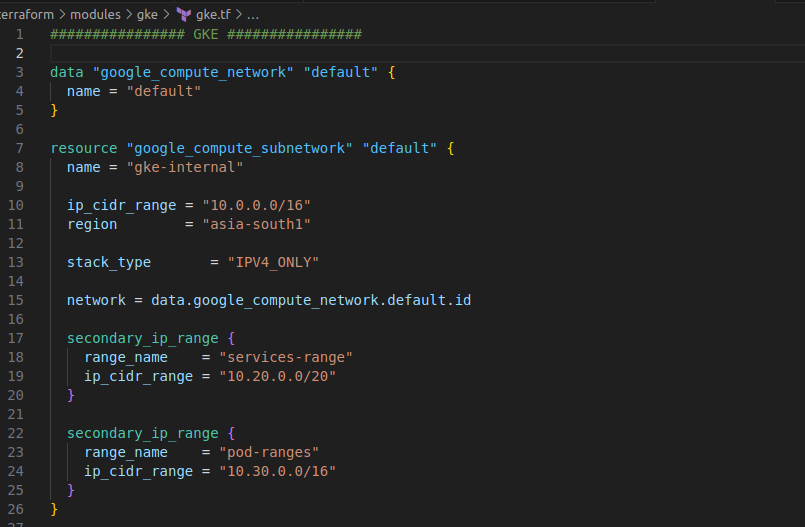
│ └── gke

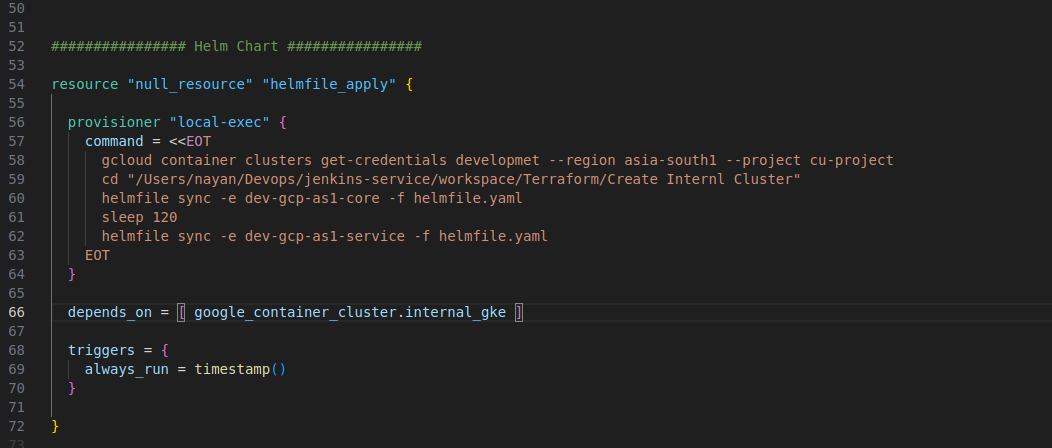
│ ├── gke.tf

│ ├── output.tf

│ └── variables.tf

└── README.md

**gke.tf** 

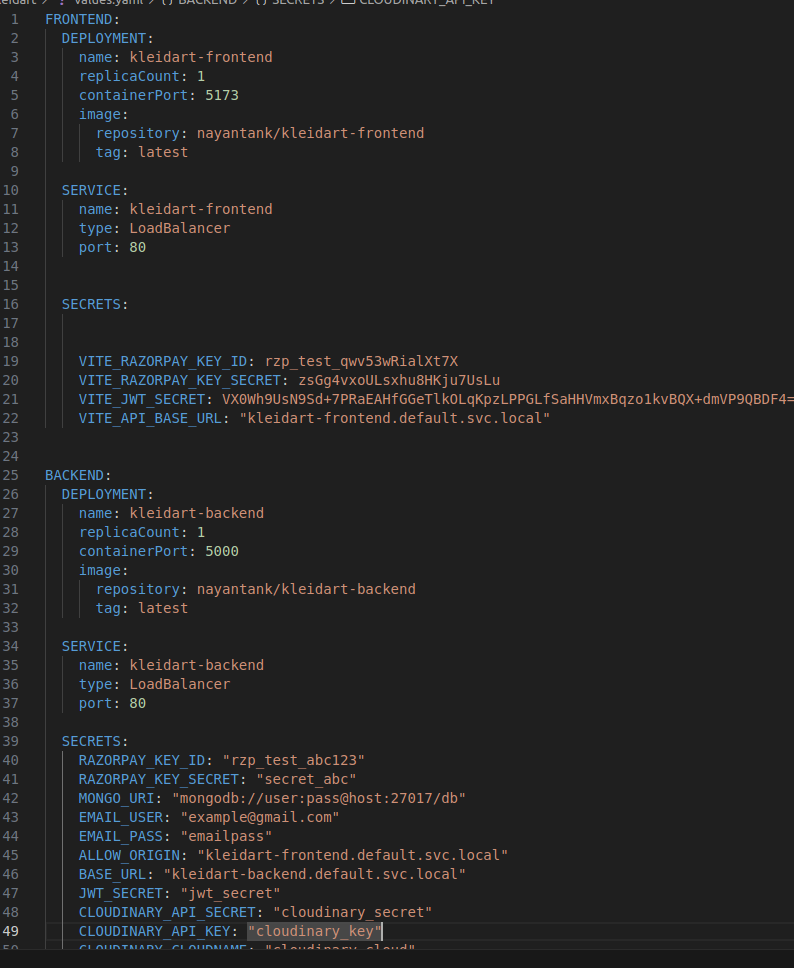


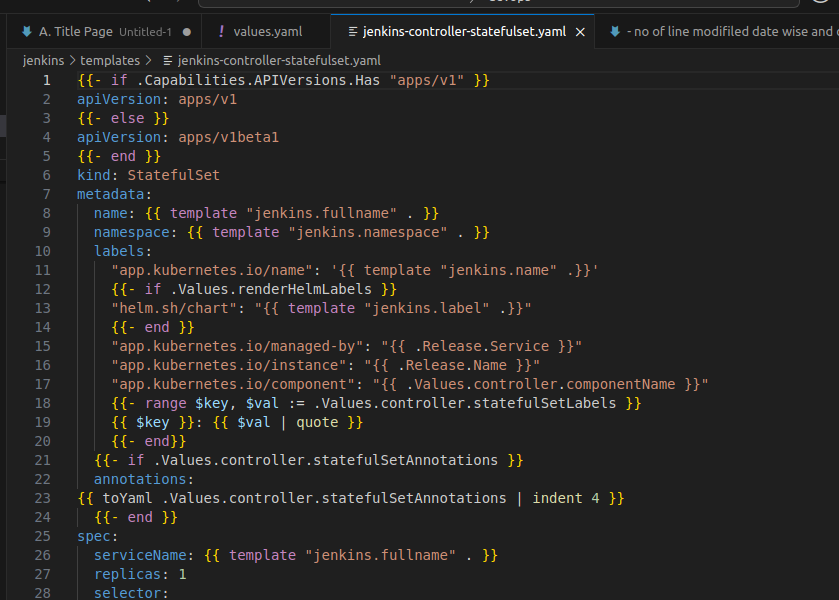
Terraform ensures the infrastructure is reproducible, version-controlled, and scalable.

## **3.3 Helm Chart Configuration**

Helm is used to deploy:

* **Jenkins**: Installed via the official Helm chart, with persistent storage and service configuration.
* **Prometheus & Grafana**: Deployed as a monitoring solution, useful for monitoring GKE cluster.
* **MERN App:** Installed via custom helm chart.





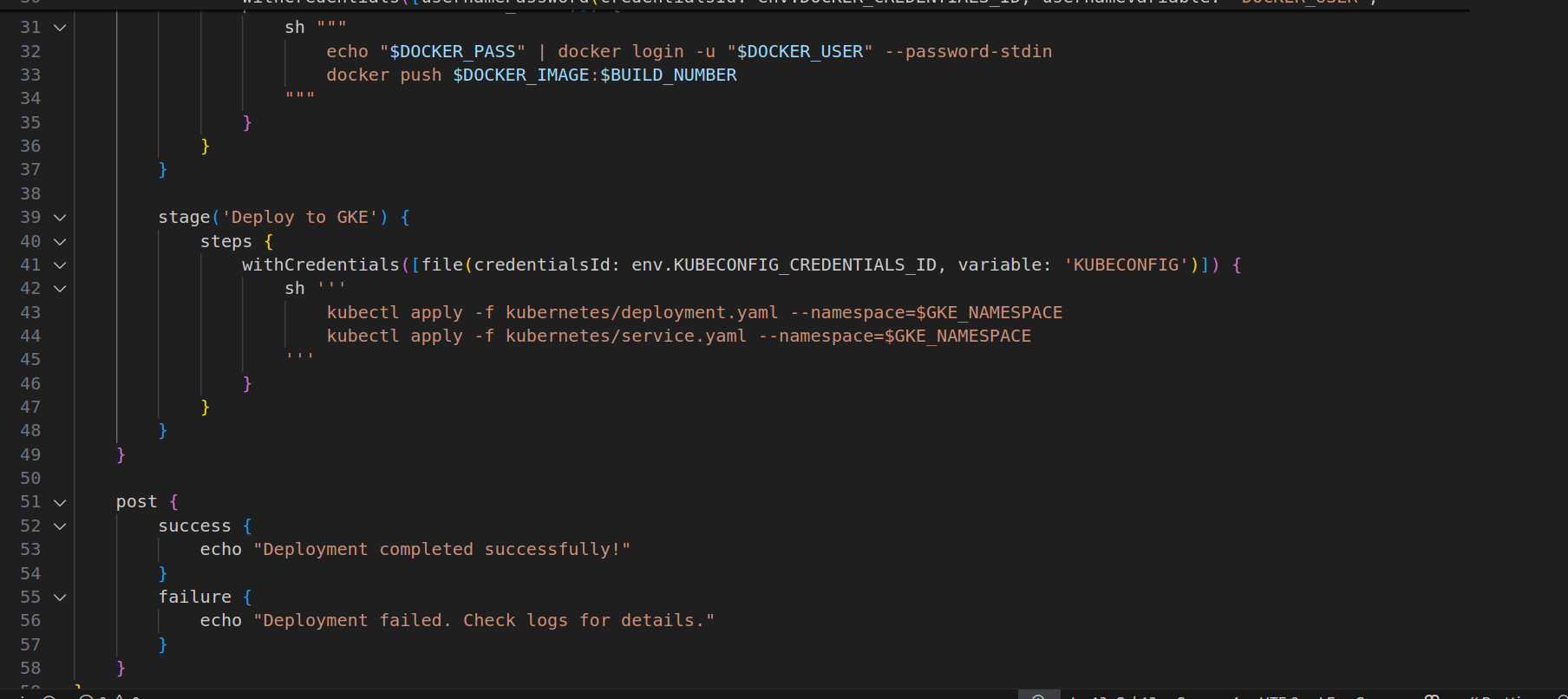
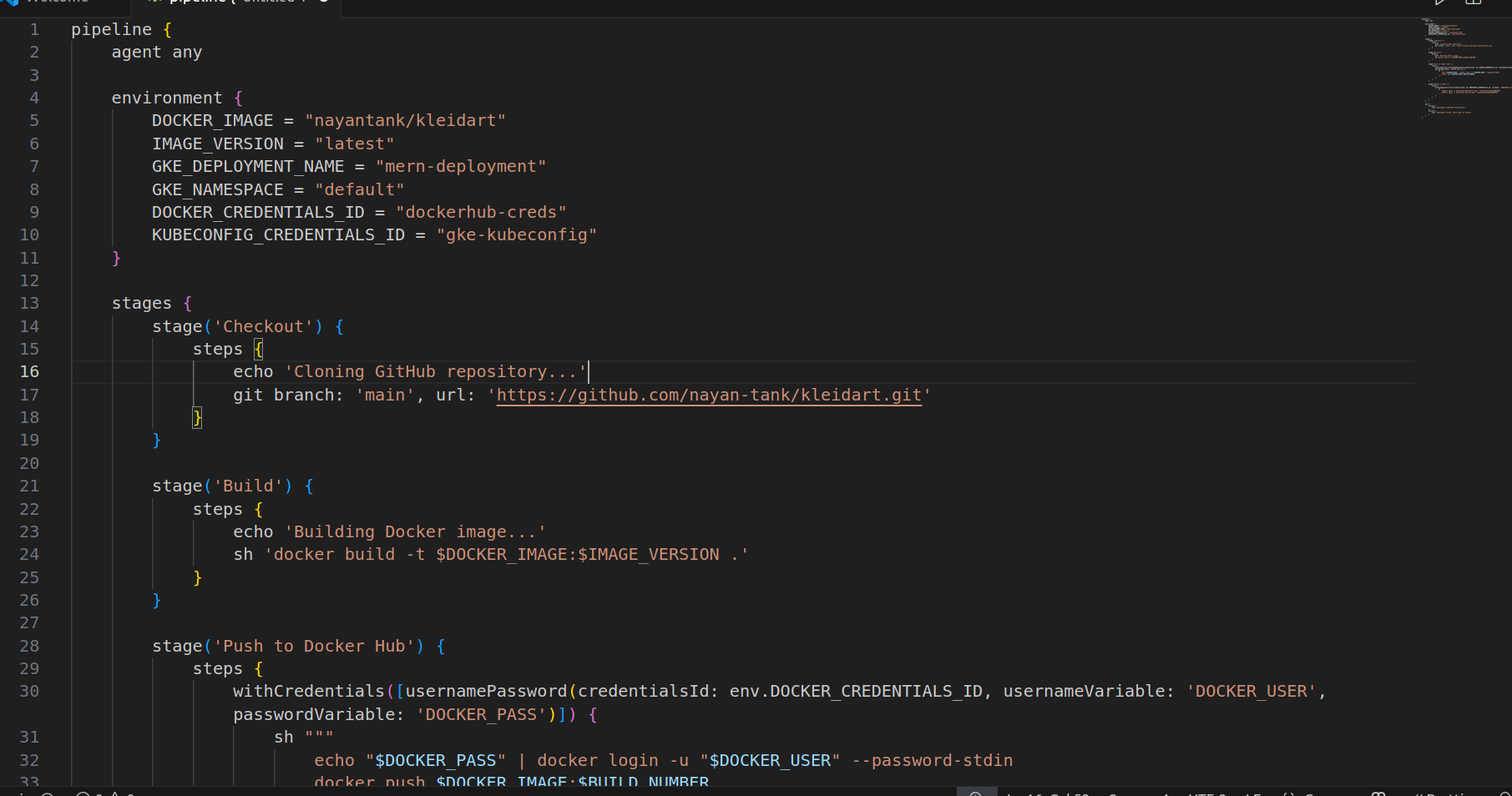


Values files are created for custom configurations. Helm makes upgrades and rollbacks easy through versioned releases.

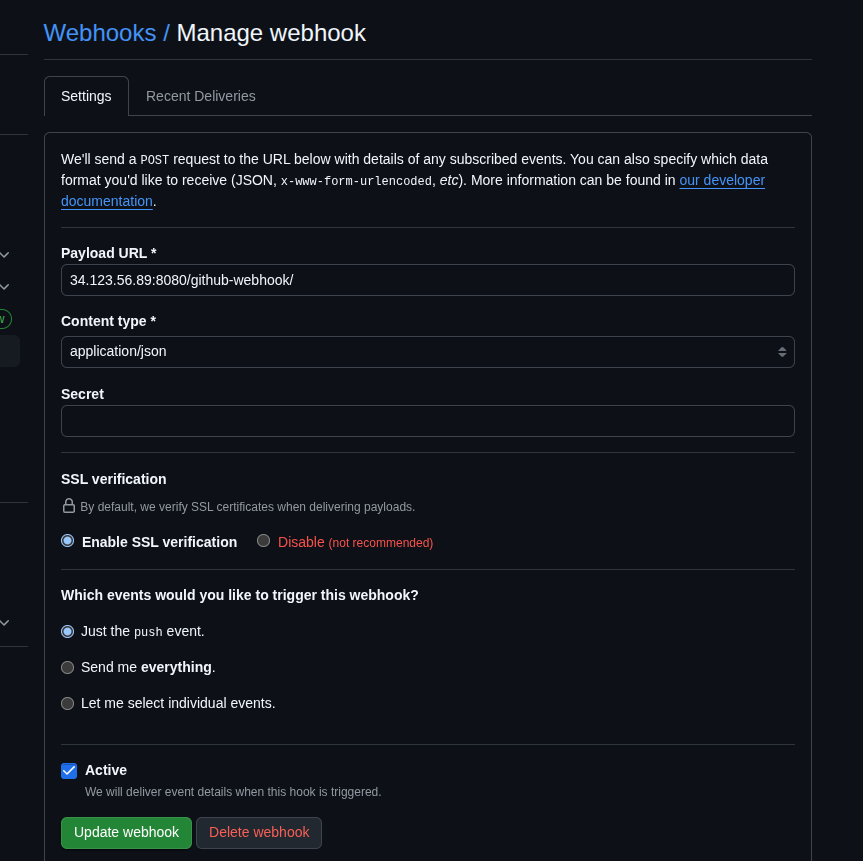
## **3.4 Jenkins Pipeline Architecture**

The Jenkins pipeline is defined using a Jenkinsfile and comprises the following stages:

1. **Checkout**: Pulls the latest code from the GitHub repository.
2. **Build**: Builds a Docker image for the full-stack application.
3. **Push to Docker Hub**: Tags and pushes the image to Docker Hub.
4. **Deploy to GKE**: Pulls the image from Docker Hub and deploys it to GKE using kubectl or Helm.



The pipeline is triggered by a webhook integrated with GitHub.



Visit <https://github.com/nayan-tank/cu-project> for more details.

## **3.5 Security and Access Control**

Security considerations include:

* **IAM Roles**: Assigned least privilege to Jenkins and Terraform service accounts.
* **Docker Hub Authentication**: Jenkins securely uses credentials to push images.
* **Kubernetes RBAC**: Configured for Jenkins to access only necessary Kubernetes resources.
* **GitHub Webhook Secret**: Ensures only legitimate triggers start the pipeline.

This system design ensures secure, automated, and scalable deployment of containerized applications to a managed Kubernetes platform.

# **Chapter 4: Tools and Technologies Used**

This project utilizes a wide range of DevOps and cloud-native tools to ensure seamless infrastructure provisioning, application deployment, and automation. Below is a categorized breakdown:

## **4.1 Infrastructure as Code**

* **Terraform**: Used for provisioning the entire cloud infrastructure, including GKE clusters, IAM roles, and networking resources on GCP.

## **4.2 Container Orchestration**

* **Google Kubernetes Engine (GKE)**: Hosts the full-stack application and supports scaling, load balancing, and orchestration of containerized services.

## **4.3 Package Management**

* **Helm**: Facilitates deployment and configuration of Jenkins and Prometheus/Grafana onto the GKE cluster using predefined Helm charts.

## **4.4 Continuous Integration / Continuous Deployment**

* **Jenkins**: Orchestrates the CI/CD pipeline triggered by GitHub webhooks. Automates building, testing, and deploying Docker images.
* **GitHub**: Hosts the full-stack application source code and triggers Jenkins builds through webhooks on push to the main branch.

## **4.5 Containerization**

* **Docker**: Used to containerize the full-stack application for portability and ease of deployment.
* **Docker Hub**: Stores the Docker images built by Jenkins, making them accessible for deployment on GKE.

## **4.6 Programming Languages & Scripting**

* **Shell Scripts**: Utilized in Terraform and Jenkins for automation and command-line tasks.
* **YAML**: Used for defining Kubernetes manifests and Helm values.

## **4.7 Security & Access Management**

* **GCP IAM**: Manages secure role-based access to infrastructure resources.
* **GitHub Secrets / Jenkins Credentials**: Securely stores API tokens, Docker Hub credentials, and webhook secrets.

## **4.8 Development & Monitoring Tools**

* **Visual Studio Code (VS Code)**: Used for development and configuration.
* **Google Cloud Console**: For resource monitoring, billing alerts, and visual management of GKE clusters.
* **Grafana & Prometheus:** For resource monitoring

These tools together form the backbone of a robust DevOps pipeline, ensuring automation, reproducibility, and scalability throughout the software delivery lifecycle.

# **Chapter 5: Implementation**

## **5.1 GKE Cluster Provisioning with Terraform**

Terraform was used to provision the infrastructure required for the project on Google Cloud Platform. The key resources included:

* A Virtual Private Cloud (VPC) network
* Subnets and firewall rules
* A Google Kubernetes Engine (GKE) cluster
* IAM roles for Jenkins and Kubernetes access

Terraform modules were organized and parameterized using variables to allow reuse and scalability.

## **5.2 Helm Installation of Jenkins, Prometheus and Grafana**

Once the cluster was provisioned, Helm was used to install the following:

* **Jenkins**: Deployed using the official Helm chart. Configuration included persistent volume claims, service exposure (NodePort or Ingress), and credentials setup.
* **Prometheus & Grafana**: Installed via Helm for real time monitoring of GKE cluster

Helm's values.yaml file was customized for both deployments to set appropriate configurations.

## **5.3 Jenkins Configuration and CI/CD Pipelines**

Jenkins was configured with:

* Required plugins: Kubernetes, Docker Pipeline, GitHub, and Blue Ocean
* Credentials: Docker Hub, GitHub token, GCP service account key
* A multibranch pipeline project connected to the GitHub repository

The Jenkinsfile defined the CI/CD stages:

1. **Checkout** - Clones the repository on webhook trigger
2. **Build Docker Image** - Builds the application container
3. **Push to Docker Hub** - Tags and uploads the image
4. **Deploy to GKE** - Uses kubectl or Helm to deploy using the latest image

The GitHub webhook was configured to automatically trigger this pipeline on push events to the main branch.

## **5.4 Kubernetes Deployment Strategy**

Deployment to GKE involved:

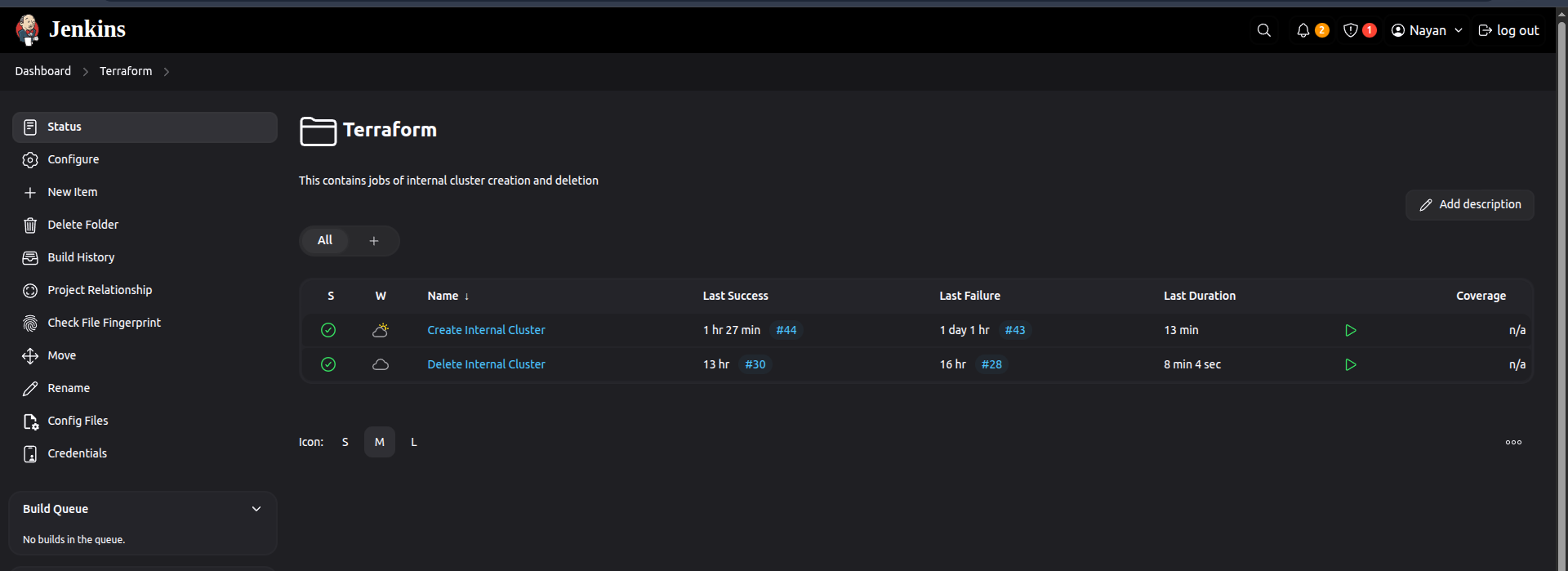
* Creating Kubernetes manifests or Helm chart for the app
* Referencing the latest Docker image from Docker Hub
* Applying deployments, services using Jenkins

kubectl apply or helm upgrade commands were executed within Jenkins using credentials stored securely as secrets.

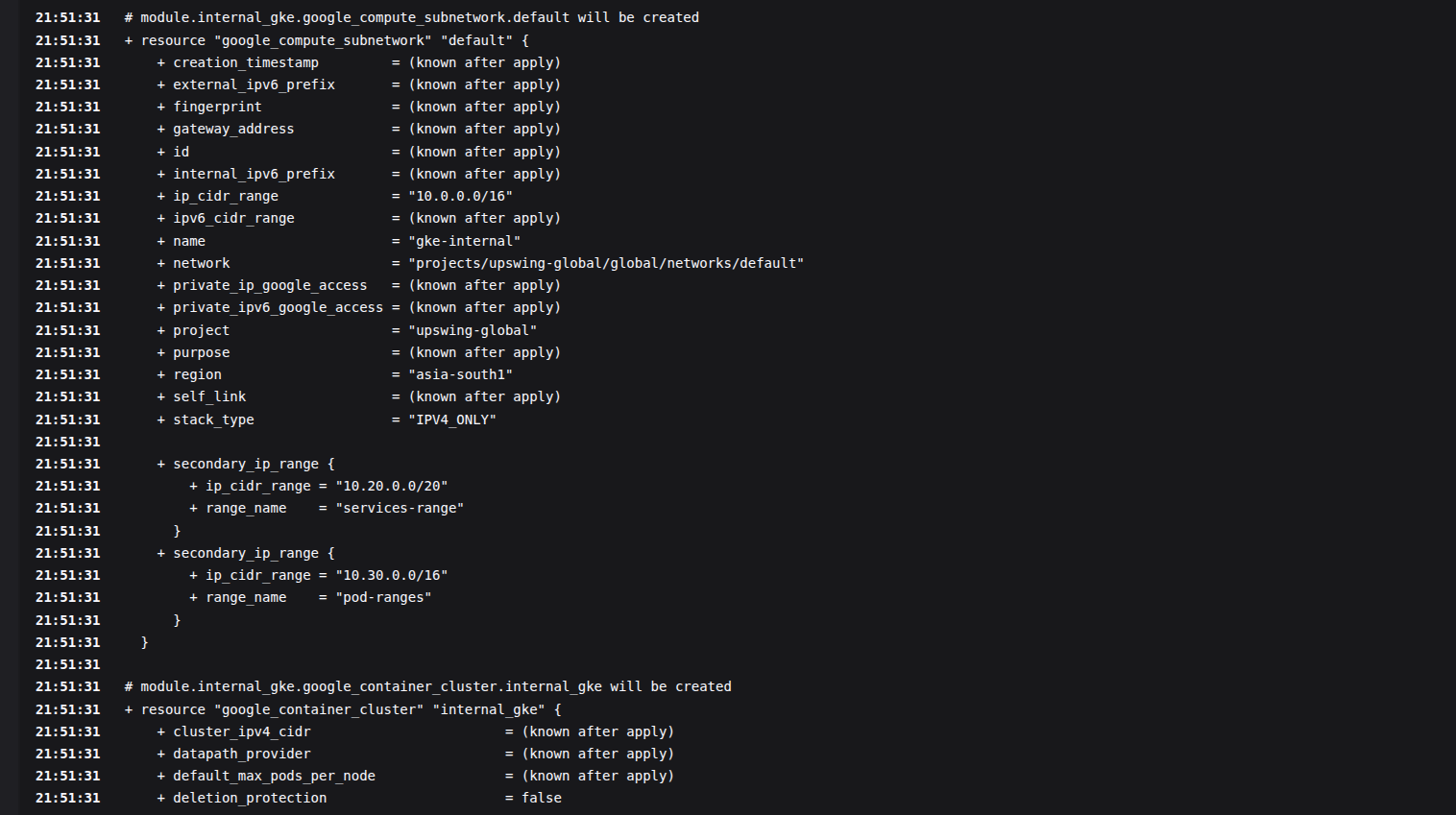
## **5.5 Screenshots & Demo**

Screenshots captured during implementation include:

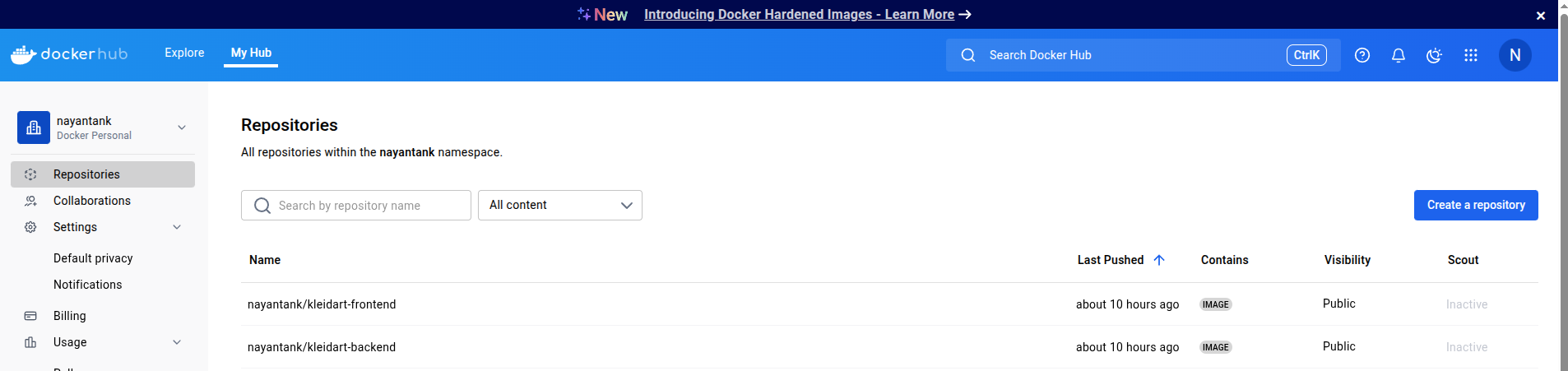
* Terraform GKE provisioning output



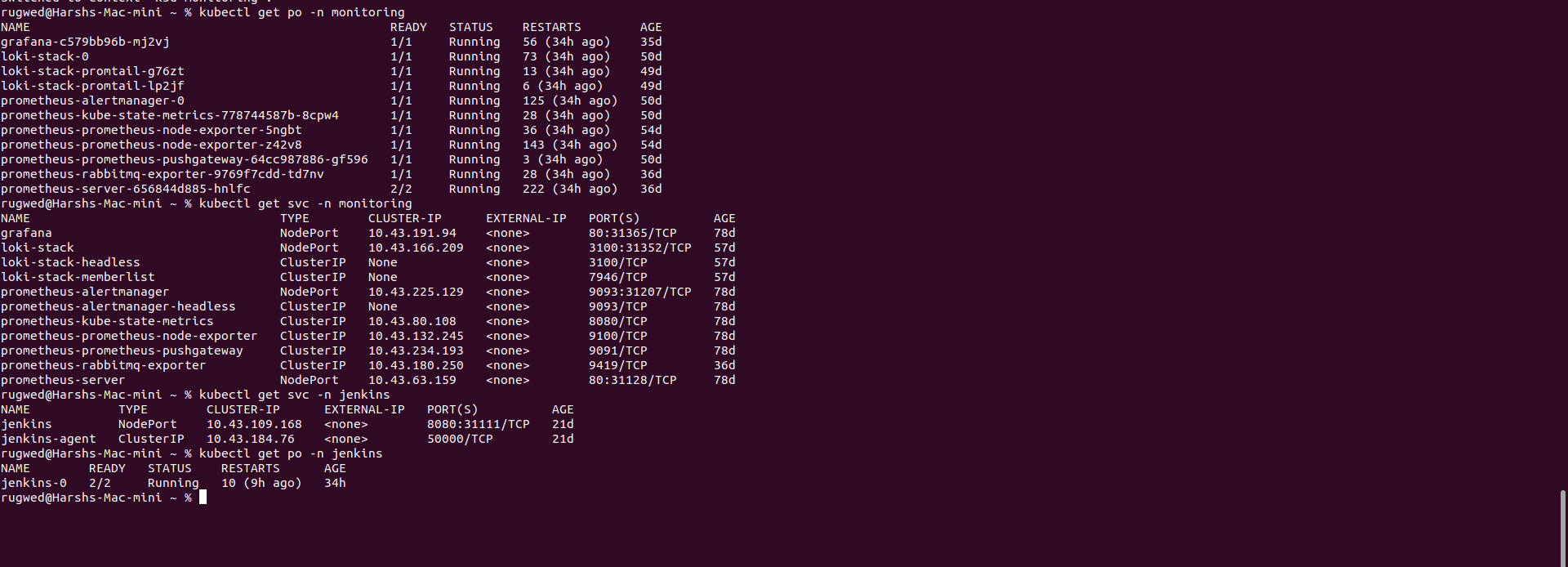
* Jenkins dashboard with successful pipeline runs



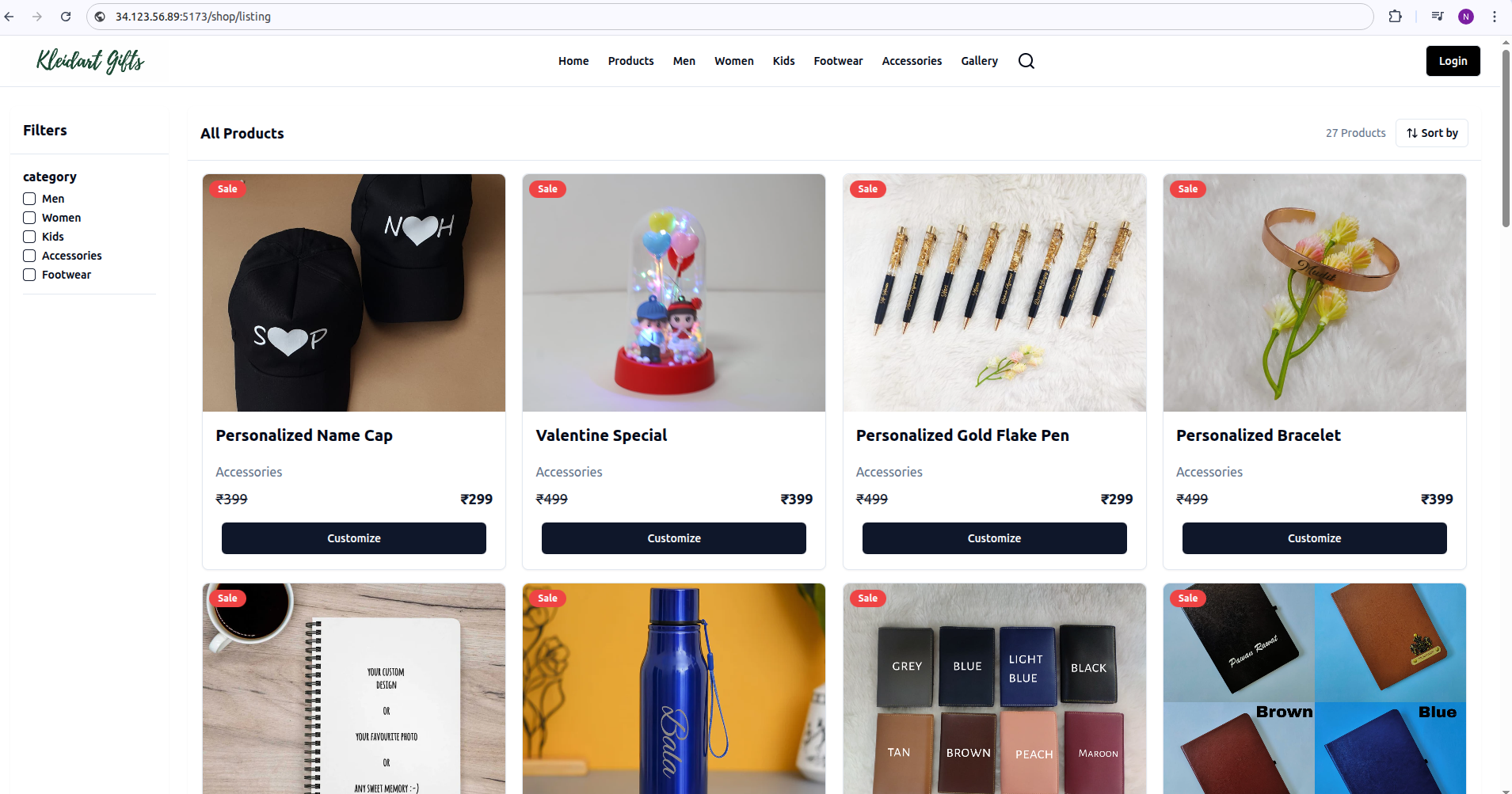
* Docker Hub with pushed image tags

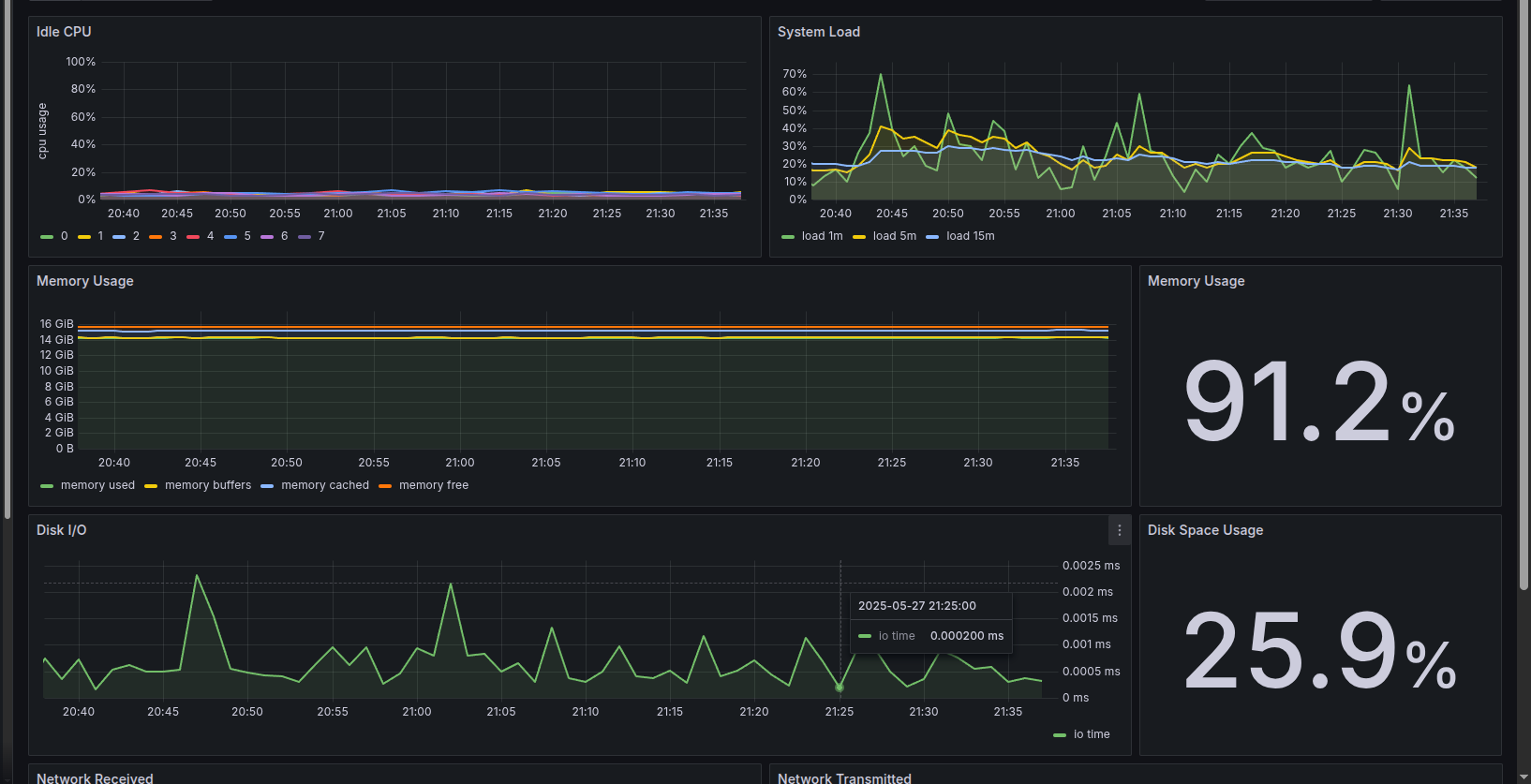


* GKE dashboard showing deployed pods and services



* Application running in the browser via external IP or Ingress



* Monitoring cluster with Grafana 

These visuals validate the successful automation of deployment from GitHub to GKE through Jenkins CI/CD pipeline.

Visit <https://github.com/nayan-tank/cu-project> for more details.

# **Chapter 6: Testing**

## **6.1 Unit Testing**

Unit testing was applied to the following components:

* **Terraform Modules**: terraform validate and terraform plan commands were used to test syntax correctness and resource dependencies.
* **Jenkins Pipeline**: Individual stages such as build, push, and deploy were tested with controlled inputs.
* **Dockerfile**: Verified locally to ensure successful image builds

## **6.2 Integration Testing**

Integration testing validated the interaction between different tools and services:

* **GitHub ↔ Jenkins**: Tested the webhook trigger mechanism on every push to the main branch.
* **Jenkins ↔ Docker Hub**: Ensured Docker images were tagged correctly and pushed without authentication errors.
* **Jenkins ↔ GKE**: Verified successful deployment of updated containers to the GKE cluster.

## **6.3 System Testing**

End-to-end testing scenarios were executed to verify the full pipeline:

1. A code change was pushed to the main branch of the GitHub repository.
2. The webhook triggered Jenkins to execute the defined pipeline.
3. Jenkins built a new Docker image and pushed it to Docker Hub.
4. The updated image was deployed to the GKE cluster.
5. The application was accessible and functional via an external IP or Ingress.

Edge cases were also tested:

* Invalid Docker credentials
* Failure to connect to GKE
* Broken pipeline steps

Each of these scenarios was logged and addressed by reviewing Jenkins console outputs and Kubernetes pod logs.

# **Chapter 7: Results and Observations**

The successful completion of this project demonstrated the feasibility and efficiency of implementing a fully automated CI/CD pipeline using modern DevOps tools and cloud-native technologies. Key observations and results are as follows:

## **7.1 Functional Results**

* The Terraform scripts reliably provisioned the GKE infrastructure on Google Cloud.
* Helm charts successfully deployed Jenkins and Grafana/Prometheus on the Kubernetes cluster.
* Jenkins was integrated seamlessly with GitHub via a webhook that triggered the pipeline on every push to the main branch.
* Docker images were built and pushed to Docker Hub with proper version tagging.
* Kubernetes deployments pulled the updated Docker images and rolled out application updates with zero downtime.

## **7.2 Performance Observations**

* Jenkins pipeline executed all stages (checkout, build, push, deploy) in under 5 minutes.
* GKE cluster spun up in less than 15 minutes using Terraform.
* Docker Hub handled image storage and retrieval without any latency issues.

## **7.3 Reliability and Scalability**

* The system maintained idempotency through Terraform, ensuring reproducibility of infrastructure.
* Jenkins pipeline was consistent and fault-tolerant; failures were logged and retried if needed.
* GKE provided auto-healing and scaling support for application pods.

## **7.4 Usability**

* Minimal manual intervention was required after the initial setup.
* Developers could focus solely on writing code while the pipeline managed builds and deployments.

These results validate the success of the project's goals to automate cloud-native application deployment using CI/CD best practices and infrastructure-as-code principles.

# **Chapter 8: Challenges Faced and Solutions**

Throughout the course of this project, several technical and operational challenges were encountered. Each challenge provided an opportunity to troubleshoot, learn, and apply best practices. Below are the key difficulties faced and how they were resolved:

## **8.1 Terraform Resource Dependencies**

**Challenge:** Managing dependencies between resources such as IAM roles, service accounts, and the GKE cluster sometimes led to provisioning errors.

**Solution:** Used depends\_on explicitly and structured Terraform modules to enforce correct execution order. Output variables were used to share resources between modules.

## **8.2 Jenkins Helm Configuration**

**Challenge:** Jenkins did not start properly due to incorrect persistent volume settings and RBAC permissions.

**Solution:** Adjusted Helm values to define proper storage classes and added necessary cluster roles and role bindings to allow Jenkins pods to run with sufficient privileges.

## **8.3 GitHub Webhook Not Triggering Jenkins**

**Challenge:** Webhook events were not reaching Jenkins due to firewall and endpoint misconfiguration.

**Solution:** Ensured Jenkins was exposed via a reachable public IP or Ingress, and verified the webhook secret, content type, and payload URL.

## **8.4 Docker Hub Push Failures**

**Challenge:** Jenkins jobs failed while pushing images to Docker Hub due to credential issues.

**Solution:** Configured Docker Hub credentials securely in Jenkins using the "Credentials" plugin and referenced them within the pipeline script.

## **8.5 Kubernetes Deployment Errors**

**Challenge:** Image pull errors occurred on GKE nodes due to tag mismatches and image unavailability.

**Solution:** Verified that Jenkins tagged the images correctly and waited for Docker Hub to make the image available before triggering deployment. Added imagePullPolicy: Always to ensure Kubernetes always fetches the latest image.

## **8.6 YAML and Helm Syntax Errors**

**Challenge:** Misconfigured YAML files or Helm values led to failed deployments.

**Solution:** Validated all configuration files using linters and dry-run commands like kubectl apply --dry-run=client and helm lint before actual deployment.

These challenges and their solutions were instrumental in strengthening the reliability and automation of the CI/CD pipeline, resulting in a resilient and production-ready workflow.

# **Chapter 9: Conclusion**

## **9.1 Summary of Work**

This project successfully demonstrated how to build an automated CI/CD pipeline for a full-stack application using modern DevOps practices and cloud-native technologies. The workflow included provisioning a GKE cluster with Terraform, deploying Jenkins, Grafana and Prometheus using Helm, and automating application deployment via Jenkins pipelines triggered by GitHub webhooks. Docker images were built and stored in Docker Hub and then deployed to GKE.

## **9.2 Key Achievements**

* Automated infrastructure provisioning using Terraform
* Jenkins, Grafana and Prometheus installation via Helm charts
* GitHub integration with Jenkins through webhooks
* CI/CD pipeline creation to build and deploy Docker containers
* Secure and scalable deployment to GKE with minimal manual intervention

## **9.3 Learning Outcomes**

* Hands-on experience with Infrastructure as Code using Terraform
* Hands-on experience with monitoring solutions like Grafana & Prometheus
* Practical skills in managing Kubernetes clusters with GKE
* Proficiency in configuring Helm charts and Jenkins pipelines
* Integration of Docker, Jenkins, GitHub, and Kubernetes in real-world DevOps workflows
* Exposure to error handling, automation, and pipeline optimization

## **9.4 Limitations**

* Role-based access control (RBAC) could be further fine-tuned
* No implementation of a staging environment for pre-production testing

## **9.5 Future Scope**

* Implement application and infrastructure monitoring using Prometheus and Grafana
* Add support for multi-environment deployment (dev, staging, production)
* Introduce a secrets management system (e.g., HashiCorp Vault, GCP Secret Manager)
* Enable Blue/Green or Canary deployments for zero-downtime updates
* Expand CI/CD pipeline to support automated testing and rollback mechanisms

# **Chapter 10: Bibliography (APA Style)**

HashiCorp. (n.d.). *Terraform by HashiCorp*. [https://www.terraform.io](https://www.terraform.io/)

Google Cloud. (n.d.). *Google Kubernetes Engine Documentation*. <https://cloud.google.com/kubernetes-engine/docs>

Docker Inc. (n.d.). *Docker Documentation*. [https://docs.docker.com](https://docs.docker.com/)

Jenkins. (n.d.). *Jenkins User Documentation*. <https://www.jenkins.io/doc/>

GitHub. (n.d.). *GitHub Docs*. <https://docs.github.com/en>

Helm. (n.d.). *Helm - The Kubernetes Package Manager*. <https://helm.sh/docs/>

Kubernetes. (n.d.). *Kubernetes Documentation*. <https://kubernetes.io/docs/>

DigitalOcean. (n.d.). *How To Set Up Continuous Integration Pipelines with Jenkins on Kubernetes*. <https://www.digitalocean.com/community/tutorials>

Docker Hub. (n.d.). *Docker Hub Documentation*. <https://docs.docker.com/docker-hub/>

HashiCorp. (n.d.). *Managing Infrastructure as Code with Terraform*. <https://learn.hashicorp.com/terraform>

Grafana. (n.d.). Visualizing Infrastructure <https://grafana.com/docs/grafana/latest/>

Prometheus. (n.d.). Collecting metrics of infrastructure <https://prometheus.io/docs/introduction/overview/>

Visit <https://github.com/nayan-tank/cu-project> for more details.

These references were instrumental in guiding the implementation and documentation of the CI/CD pipeline on GKE using Terraform, Jenkins, Helm, and other DevOps tools.