IaC Provisioning for Non-profit System using Terraform, Docker, Kubernetes, and CI/CD

Course Name: DevOps

Institution Name: Medicaps University – Datagami Skill Based Course

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

SR NO	STUDENT NAME	ENROLMENT NUMBER
1	KRATIKA SINGHAL	EN22CS303029
2	SUJOY CHAKRAVARTY	EN22CS304063
3	SAMARTH RAO	EN22CS303049
4	UJJAWAL SISODIYA	EN22CS303057
5	PARTH REVARAM	EN22CS304044
6	PIYUSH BARKHADE	EN22CS304045

Group Name:09D11

Project Number:DO-35

Industry Mentor Name: AASHRUTI SHAH

University Mentor Name: PROF. SHYAM PATEL

Academic Year: 2022-2026

Problem Statement & Objectives

1. Problem Statement

2. Project Objectives
3. Scope of the Project

Proposed Solution

1. Key features *(Just mention key features here no need to go into details)*
2. Overall Architecture / Workflow
3. Tools & Technologies Used (*If applicable)*

Results & Output

Add the below details here:

1. *Screenshots / outputs*
2. *Reports / dashboards / models*
3. *Key outcomes*

Conclusion

Mention a brief conclusion about your project summing up everything you have worked on and the key learning.

Future Scope & Enhancements

1. Introduction

The *IaC Provisioning for Non-profit System* is designed to automate infrastructure deployment for agriculture-based applications using Infrastructure as Code (IaC) principles. Instead of manually configuring servers, networks, and deployment environments, the system uses configuration files to define infrastructure components.

Modern Non-profit applications such as crop monitoring systems, weather dashboards, soil analytics platforms, and market price trackers require scalable, reliable, and secure infrastructure. Manual setup of such systems leads to inconsistencies, errors, and scalability issues.

This project implements a cloud-native architecture using:

- Terraform for infrastructure provisioning
- Docker for containerization
- Kubernetes for orchestration
- CI/CD pipelines for automated deployment

The system follows DevOps best practices and ensures automation, repeatability, and scalability.

2. Problem Statement and Objectives

2.1 Problem Statement

Non-profit organizations increasingly rely on digital platforms such as donation management systems, volunteer registration portals, beneficiary tracking applications, fundraising dashboards, and impact reporting tools. These applications require reliable, scalable, and secure infrastructure to handle donor data, financial transactions, user interactions, and program records. As digital engagement and online fundraising grow, the demand for stable and secure infrastructure becomes critical.

Traditionally, infrastructure for such non-profit platforms is configured manually. System administrators manually create servers, configure networking, install software dependencies, and deploy applications. This manual approach leads to several challenges:

1. Configuration Inconsistency

Manual setup creates differences between development, testing, and production environments, leading to unexpected application failures.

2. Human Errors

Manual provisioning increases the risk of mistakes such as incorrect security rules or misconfigured servers, which can cause downtime or security issues.

3. Poor Scalability

Traditional infrastructure cannot easily handle sudden traffic spikes during peak agricultural seasons.

4. Time-Consuming Deployment

Manual infrastructure setup is slow and delays application updates and feature releases.

5. Limited Automation

Repetitive manual configuration increases operational effort and reduces efficiency.

6. Security Risks

Improper network and access configurations may expose sensitive Non-profit data to vulnerabilities.

7. Lack of Repeatability

Manual infrastructure cannot be easily replicated across environments, making scaling and disaster recovery difficult.

2.2 Project Objectives

The main objective of this project is to design and implement an automated, scalable, and secure infrastructure provisioning system for Non-profit based applications using Infrastructure as Code (IaC) principles.

The specific objectives are as follows:

1. Implement Infrastructure as Code (IaC)

To define and manage cloud infrastructure using Terraform configuration files instead of manual setup, ensuring consistency and repeatability.

2. Automate Cloud Resource Provisioning

To provision AWS resources such as EC2 instances, VPC, subnets, security groups, and IAM roles automatically using Terraform.

3. Containerize the Application

To package the Non-profit web application and its dependencies into Docker containers for consistent deployment across environments.

4. Deploy Using Kubernetes

To use Kubernetes for container orchestration, enabling high availability, fault tolerance, and scalable deployment.

5. Integrate CI/CD Pipeline

To implement CI/CD automation using Jenkins or GitHub Actions for continuous integration and automated deployment.

6. Ensure High Availability

To deploy multiple replicas of the application using Kubernetes to avoid downtime and improve system reliability.

7. Improve Security

To implement secure infrastructure practices such as IAM role-based access control, VPC isolation, encryption, and secret management.

8. Achieve Scalability

To design the system in a way that allows easy horizontal scaling based on user demand or workload.

9. Maintain Infrastructure State Management

To manage Terraform state remotely using AWS S3 and DynamoDB for consistency and safe collaboration.

10. Follow DevOps Best Practices

To implement automation, monitoring, logging, and version control to align with modern DevOps methodologies.

2.3 Scope of the Project

This project focuses on automating the deployment of an Non-profit web application using Infrastructure as Code (IaC).

The scope includes:

- Provisioning cloud infrastructure on AWS using Terraform.
- Containerizing the application using Docker.
- Deploying and managing containers using Kubernetes.
- Implementing CI/CD for automated deployment.
- Ensuring basic security, scalability, and high availability.

This project does not include real IoT hardware implementation or advanced AI-based Non-profit models.

3. Proposed Solution

The proposed solution follows a DevOps-based automated deployment model where infrastructure is defined using configuration files instead of manual commands.

High-Level Workflow

1. Developer writes infrastructure code.
2. Code is pushed to GitHub repository.
3. CI/CD pipeline triggers automatically.
4. Terraform provisions AWS infrastructure.
5. Docker builds container image.
6. Kubernetes deploys application pods.
7. Application becomes accessible to users.

This approach ensures:

- Automated deployment
- Reduced human errors
- Scalable infrastructure
- Repeatable environments

3.1 Key Features

1. **Infrastructure as Code (IaC):** Uses Terraform to automatically create and manage cloud infrastructure.

2. **Containerization:** Uses Docker to package the Non-profit application for consistent deployment.

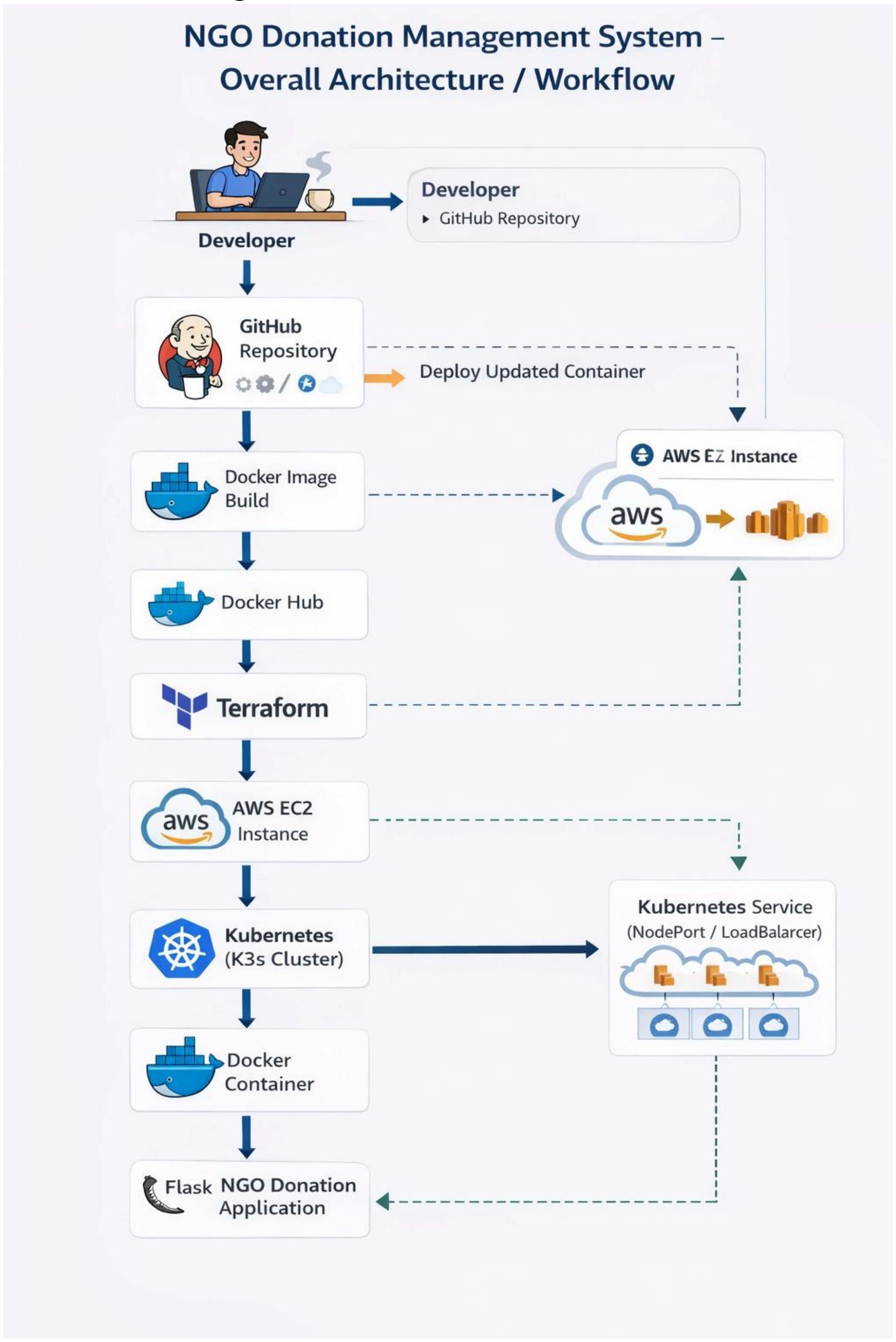
3. **Kubernetes Deployment:** Manages containers with multiple replicas for high availability.

4. **CI/CD Automation:** Automatically builds and deploys the application when code changes.

5. **Scalability and Security:** Designed to scale easily and includes secure cloud configuration.

3.2 Overall Architecture/ Workflow

Architecture Diagram:



Step-by-Step Flow

1. User accesses the system through browser.
2. NodePort service exposes application externally.
3. Kubernetes Service routes traffic.
4. Deployment manages replica pods.
5. Pods run Docker containers.
6. Containers host the Non-profit web application via Nginx.

3.3 Tools and Technologies

Tool	**Purpose**
Git	Version Control
Docker	Containerization
Kubernetes	Container Orchestration
Nginx	Web Server
YAML	Infrastructure Definition
Docker Desktop	Local Kubernetes Cluster
VS Code	Development

4. Results and Outputs

The proposed IaC-based Non-profit System follows a layered containerized deployment architecture. The implementation successfully demonstrates automated infrastructure provisioning using Docker and Kubernetes.

The deployment ensures high availability, scalability, and automated recovery through replica management and container orchestration.

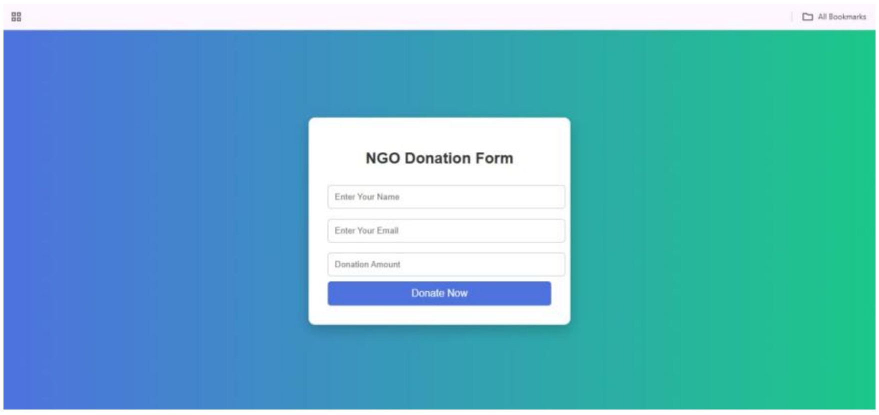
The architecture consists of the following layers:

Layer 1: User Layer

Users access the Non-profit Monitoring Application through a web browser using the NodePort service. Application URL:

http://localhost:5000/

This confirms successful external exposure of the application.



NGO Donation Form

Layer 2: Kubernetes Service Layer

The NodePort Service exposes the Non-profit application externally and routes traffic to available pods inside the Kubernetes cluster.

The service performs:

- • Traffic routing
- • Load balancing
- • External exposure
- • Stable endpoint management

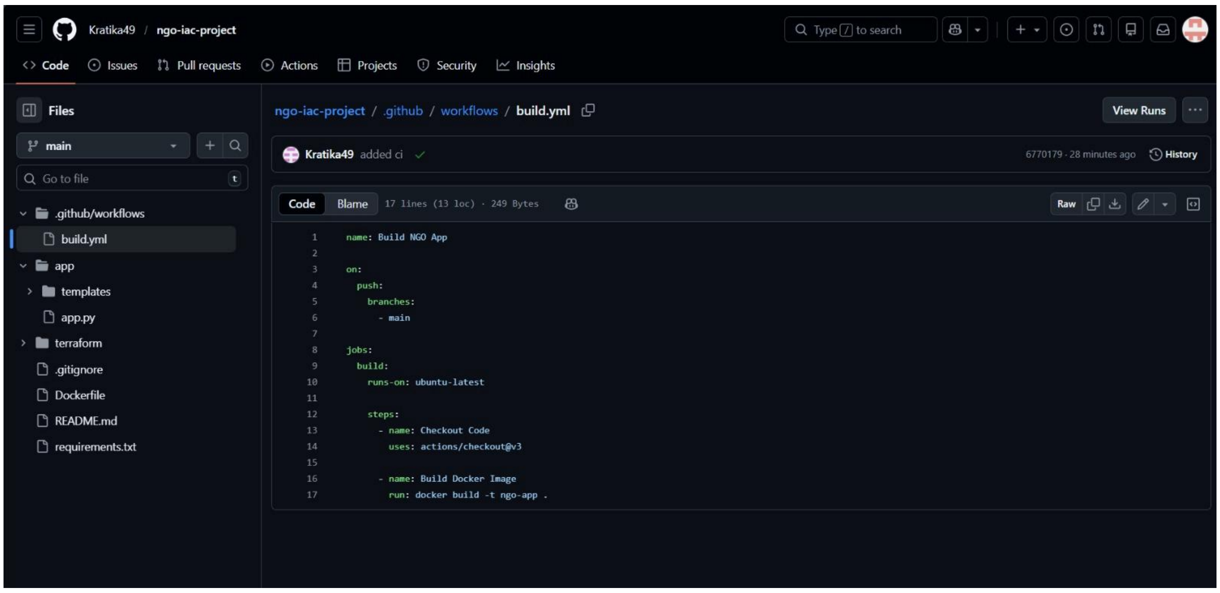
This ensures seamless communication between users and application pods.

Layer 3: Deployment Layer

The Kubernetes Deployment manages:

- • Replica count (2 replicas)
- • Desired state maintenance
- • Rolling update capability
- • Pod recreation in case of failure

The deployment configuration ensures high availability and automated lifecycle management of pods.



```yaml
name: Build NGO App

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Build Docker Image
        run: docker build -t ngo-app .
```

Layer 4: Pod Layer

Two running pods were successfully created as per the replica configuration.

Verification was performed using: kubectl

get pods

Both pods were in the Running state, confirming successful scheduling and execution.

Each pod operates independently and serves incoming requests.

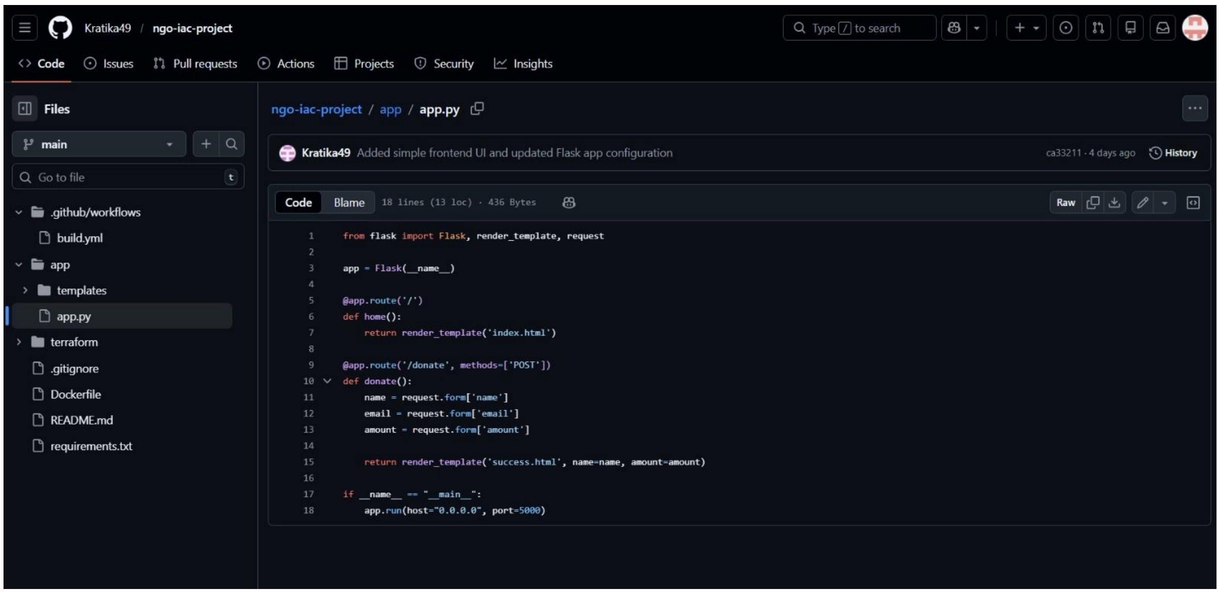
Layer 5: Container Layer

Each pod contains a Docker container built from the Non-profit application image.

The container:

- Uses nginx:alpine base image
- Hosts static Non-profit monitoring web files
- Provides lightweight execution
- Ensures consistent runtime environment

Docker successfully encapsulated the application and its dependencies.



Kratika49 Added simple frontend UI and updated Flask app configuration ca33211 · 4 days ago

Code Blame 18 lines (13 loc) · 436 Bytes

```python
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/donate', methods=['POST'])
def donate():
    name = request.form['name']
    email = request.form['email']
    amount = request.form['amount']

    return render_template('success.html', name=name, amount=amount)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Layer 6: Application Layer

The Non-Profit Management Web Application was successfully deployed and made accessible through the NodePort service.

The application interface provides a centralized digital platform for managing donations, volunteer registrations, and program information.

The application interface displays:

• Donation Management Dashboard

• Active Fundraising Campaigns

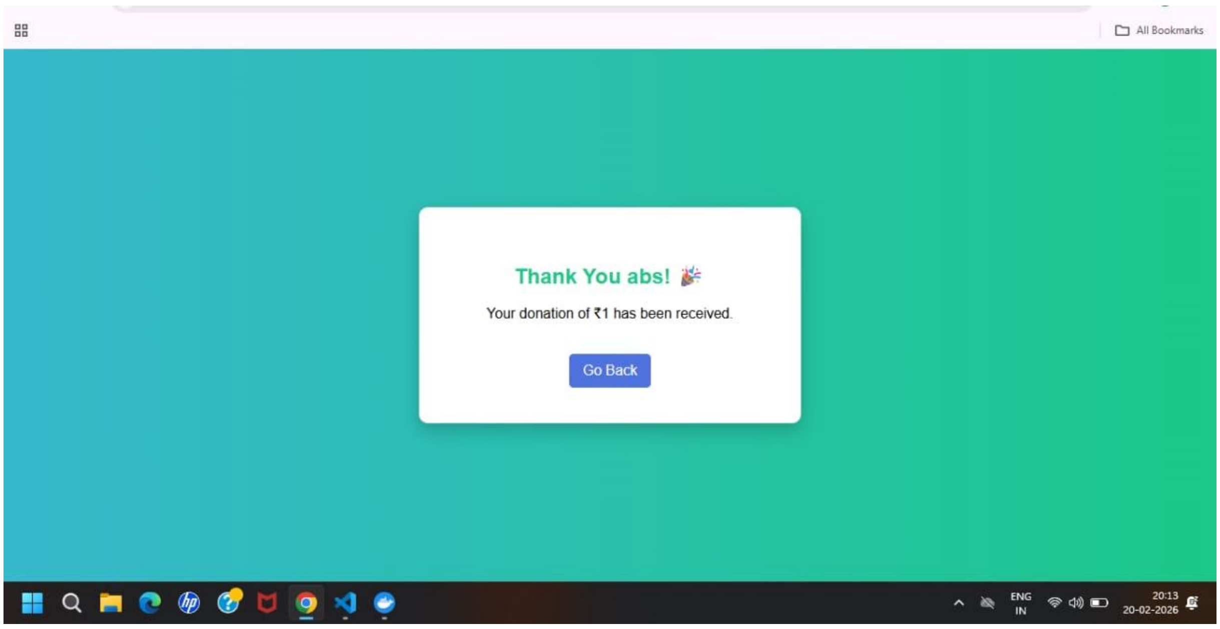
• Volunteer Registration Portal

• Beneficiary Program Details

• Real-Time Donation Summary

• System Status Indicator

This confirms successful containerized deployment and service routing within the Kubernetes cluster



Thank You abs! 🎉

Your donation of ₹1 has been received.

5. System Testing & Validation

To ensure the reliability, availability, and correct functionality of the IaC Provisioning for Non-profit System, multiple validation tests were performed. The deployed pods were verified using kubectl get pods, confirming that all replicas were in the Running state and properly scheduled within the Kubernetes cluster. The NodePort service was validated using kubectl get svc, which confirmed successful external exposure on port 30007, allowing the application to be accessed via http://localhost:30007.

The Non-profit Monitoring Dashboard loaded successfully in the browser, displaying crop information, soil health data, and weather updates, thereby confirming correct container deployment and service routing. Replica management was validated through the deployment configuration, ensuring that the desired replica count was consistently maintained. A selfhealing test was conducted by manually deleting one running pod, after which Kubernetes automatically recreated a new pod to maintain the desired state, demonstrating infrastructure resilience. Additionally, infrastructure provisioning was validated using declarative YAML files, confirming successful implementation of Infrastructure as Code (IaC). The system remained stable during all tests, verifying high availability, automated recovery, and production-ready deployment behavior.

6. Performance & Reliability

The IaC Provisioning for Non-profit System demonstrates high performance and reliability through its containerized and orchestrated architecture. By deploying multiple replicas using Kubernetes Deployment, the system ensures continuous availability even if one pod fails. The stateless container architecture allows efficient resource utilization and smooth scaling without service interruption. During testing, the application remained accessible even when a pod was manually deleted, confirming automated recovery and fault tolerance.

The NodePort service effectively distributed traffic across available pods, ensuring stable response handling. Although deployed in a local Kubernetes cluster, the architecture follows production-ready principles, providing a strong foundation for scalable and reliable cloudnative deployment.

7. DevOps Principles Applied

This project effectively applies modern DevOps practices by implementing Infrastructure as Code (IaC) through declarative YAML configuration files for deployment and service creation. Containerization using Docker ensured environment consistency, portability, and elimination of dependency conflicts. Kubernetes orchestration automated pod scheduling, replica management, service exposure, and self-healing capabilities. Version tagging of

Docker images enabled structured release management and potential rolling updates. The entire deployment process was automated using kubectl commands, reducing manual intervention and configuration errors. These practices align with enterprise-level cloud-native deployment standards and simulate real-world DevOps workflows used in modern Non-profit technology platforms.

8. Security Considerations

The system incorporates basic security measures through container isolation and controlled service exposure using NodePort. Docker ensures that the application and its dependencies are packaged securely within containers, minimizing conflicts and unauthorized modifications. Kubernetes manages internal networking and pod communication within the cluster.

However, since the project is deployed locally, advanced security mechanisms such as RoleBased Access Control (RBAC), secrets management, HTTPS encryption, and network policies were not implemented. Future enhancements may include secure image scanning, TLS configuration, and restricted external access for production-grade security compliance.

9. Comparative Analysis

Compared to traditional infrastructure deployment methods, the containerized and orchestrated approach used in the IaC Provisioning for Non-profit System offers significant advantages. Traditional deployment requires manual server setup, configuration management, dependency handling, and system monitoring, which increases the risk of errors and downtime. In contrast, the Docker and Kubernetes-based deployment ensures automated provisioning, consistent runtime environments, simplified version control, and automated recovery mechanisms. Scalability is significantly improved, as replica count can be adjusted dynamically without affecting system availability. The Infrastructure as Code approach eliminates configuration inconsistencies and reduces manual intervention. Overall, the Kubernetes-based deployment model enhances reliability, scalability, operational efficiency, and maintainability compared to conventional deployment strategies.

10. Key Outcomes

- We successfully created and deployed the Non-profit Monitoring application using Docker and Kubernetes.
- The application runs inside containers, which makes it easy to manage and deploy.
- We automated the infrastructure setup using YAML files (Infrastructure as Code).
- The system runs with multiple pods, so it stays available even if one pod fails.
- The application is accessible from the browser using NodePort (Port 30007).
- Kubernetes automatically recreates a pod if it crashes (self-healing).
- We can increase or decrease the number of pods easily (scalability).
- The deployment process became faster and more organized.
- We gained practical knowledge of Docker, Kubernetes, and DevOps concepts.
- The project simulates how real-world cloud applications are deployed.

11. Limitations

- The project is deployed only on a local system, not on a cloud platform.
- The application does not use a database.
- There is no automatic CI/CD pipeline for continuous deployment.

• Advanced monitoring tools are not used.

• Security features like HTTPS and RBAC are not implemented.

• Autoscaling is not configured.

• There is no permanent storage setup.

• It is a basic demo-level project, not a full production system.

12. Conclusion

• **Successful Containerization:**
The Non-profit Monitoring Application was successfully containerized using Docker, ensuring consistent and portable deployment.

• **Implementation of Infrastructure as Code (IaC):**
The infrastructure was defined using YAML files, which automated the deployment process and reduced manual configuration errors.

• **Kubernetes-Based Deployment:**
The application was deployed using Kubernetes, which managed pods, replicas, and service routing efficiently.

• **High Availability Achieved:**
Multiple replicas were configured so that the application remains available even if one pod fails.

• **Self-Healing Capability Demonstrated:**
Kubernetes automatically recreated pods when one was deleted, ensuring system reliability and resilience.

• **External Access via NodePort:**
The application was successfully exposed using NodePort, allowing users to access it through a web browser.

• **Scalability Supported:**
The system allows easy scaling by increasing or decreasing the number of replicas as needed.

• **Practical DevOps Learning:**

The project provided hands-on experience in Docker, Kubernetes, YAML configuration, and modern DevOps deployment practices.

13. Future Scope & Enhancements

• **Cloud Deployment:**
The system can be deployed on cloud platforms like AWS, Azure, or Google Cloud to make it production-ready and accessible globally.

• **Horizontal Pod Autoscaling (HPA):**
Autoscaling can be implemented to automatically increase or decrease the number of pods based on traffic or CPU usage.

• **CI/CD Pipeline Integration:**
Tools like GitHub Actions or Jenkins can be used to automate the build, testing, and deployment process.

• **Database Integration:**
A database like MySQL or MongoDB can be added to store real-time Non-profit data.

• **Persistent Storage:**
Kubernetes Persistent Volumes can be configured to store data permanently even if pods are restarted.

• **Security Enhancements:**
Security features like HTTPS, RBAC, and secrets management can be added to protect the system.

• **Ingress Controller Implementation:**
Instead of NodePort, an Ingress Controller can be used for better traffic routing and domainbased access.

• **Monitoring & Logging:**
Tools like Prometheus, Grafana, and ELK Stack can be integrated to monitor system performance and logs.

• **IoT Integration:**
IoT devices can be connected to collect Non-profit data.