Project-High Level Design

on

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

Course Name:

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Ujjawal Sisodiya	EN22CS303057
2	Kratika Singhal	EN22CS303029
3	Sujoy Chakravarty	EN22CS304063
4	Samarth Rao	EN22CS303049
5	Parth Revaram	EN22CS304044

Group Name:Group 09D11

Project Number:DO-35

Industry Mentor Name:

University Mentor Name: Prof. Shyam Patel

Academic Year:2026

Table of Contents

1. Introduction

The IaC Provisioning for Non-profitSystem is designed to automate the deployment and provisioning of infrastructure required for agriculture-based applications. This system uses Infrastructure as Code (IaC) tools such as Terraform and Ansible to provision cloud infrastructure and configure environments automatically.

The system enables automated creation of:

- Cloud infrastructure (AWS EC2, Networking)
- Containerized application environment using Docker
- Kubernetes cluster deployment
- Automated CI/CD pipeline using GitHub Actions or Jenkins

This eliminates manual infrastructure setup, reduces human errors, and ensures scalability, repeatability, and reliability.
The system is designed following DevOps principles and cloud-native architecture.

1.1 Scope of the Document

This document provides a high-level architectural design of the IaC Provisioning Non-profitSystem.
It includes:

- Infrastructure design
- Deployment workflow
- Application architecture
- Data handling mechanisms
- Security and performance design
- Integration between system components

This document does NOT include:

- Source code implementation
- Low-level configuration details.

1.2 Intended Audience

This document is intended for:

- DevOps Engineers
- Cloud Engineers
- System Architects
- Developers
- Project Evaluators
- Technical Reviewers

Required knowledge:

- Cloud computing basics
- Docker and Kubernetes
- Terraform and DevOps concepts

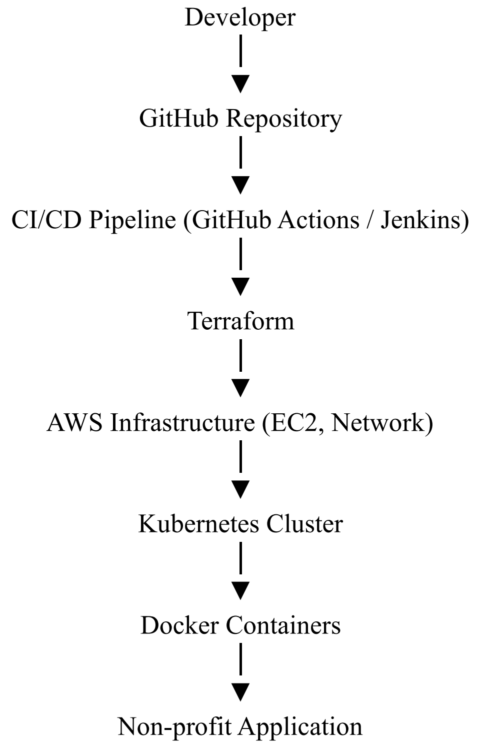
1.3 System Overview

The system automates infrastructure provisioning and application deployment using Terraform, Docker, Kubernetes, and CI/CD pipelines.

Major Components:

Component	Purpose
Terraform	Provisions AWS cloud infrastructure (EC2, EKS, VPC) for Non-profitworkloads
Docker	Containerizes Non-profitapplications (crop monitoring, sensor processing)
Kubernetes	Orchestrates container scaling and high availability
GitHub	Centralized code repository for IaC scripts and app source
Jenkins/GitHub Actions	Automates CI/CD pipelines from commit to deployment
AWS EC2	Hosts containerized Non-profitservices and databases

Architecture Diagram



Developer

▼

GitHub Repository

▼

CI/CD Pipeline (GitHub Actions / Jenkins)

▼

Terraform

▼

AWS Infrastructure (EC2, Network)

▼

Kubernetes Cluster

▼

Docker Containers

▼

Non-profit Application

2. System Design

2.1 Application Design

The application follows a layered architecture:

Layer	Description
Presentation Layer	User interface
Application Layer	Non-profitapplication logic
Container Layer	Docker containers
Orchestration Layer	Kubernetes
Infrastructure Layer	AWS EC2

2.2 Process Flow

Infrastructure Provisioning Flow:

1. Developer writes Terraform code
2. Code pushed to GitHub
3. CI/CD pipeline triggers
4. Terraform provisions AWS infrastructure
5. Docker builds container
6. Kubernetes deploys container

Deployment Flow Table

Step	Tool	Action
1	GitHub	Store code
2	Jenkins	Trigger pipeline
3	Terraform	Create infrastructure
4	Docker	Build container
5	Kubernetes	Deploy application

2.3 Information Flow

User Request

|

▼

Load Balancer

|

▼

Kubernetes Service

|

▼

Docker Container

|

▼

Application Response

2.4 Components Design

Component	Technology	Purpose
Code Repo	Git/GitHub	Source control
CI/CD	Jenkins/GitHub Actions	Pipeline automation
IaC Local	Terraform + Docker	Dev environments
IaC Cloud	Terraform + Ansible	AWS production
Orchestration	Kubernetes	Container management
Monitoring	Prometheus/Grafana	Infrastructure health

2.5 Key Design Considerations

- **Idempotency:** Terraform ensures repeatable deployments □ **State Management:** Remote backend (S3 + DynamoDB)
- **Zero-Downtime:** Blue-green deployments via Kubernetes
- **Cost Optimization:** Spot instances for non-critical workloads
- **Security:** IAM roles, Secrets Manager integration

2.6 API Catalogue

API	Method	Purpose	Endpoint
/provision/local	POST	Local Docker env	/api/v1/provision/local
/provision/cloud	POST	AWS infrastructure	/api/v1/provision/cloud
/status/{env-id}	GET	Deployment status	/api/v1/status/{env-id}
/teardown/{env-id}	DELETE	Destroy resources	/api/v1/teardown/{env-id}

3. Data Design

3.1 Data Model

Field	Description
Instance ID	Cloud instance
Container ID	Docker container
Deployment status	Status

.

3.2 Data Access Mechanism

- Terraform State: S3 bucket with DynamoDB locking
- Secrets: AWS Secrets Manager
- Config: Kubernetes ConfigMaps/Secrets
- Logs: CloudWatch Logs + ELK Stack

3.3 Data Retention Policies

Data Type	Retention	Storage
Terraform State	90 days	S3 (versioned)
Build Artifacts	30 days	S3
Application Logs	7 days	CloudWatch
Metrics	15 months	CloudWatch

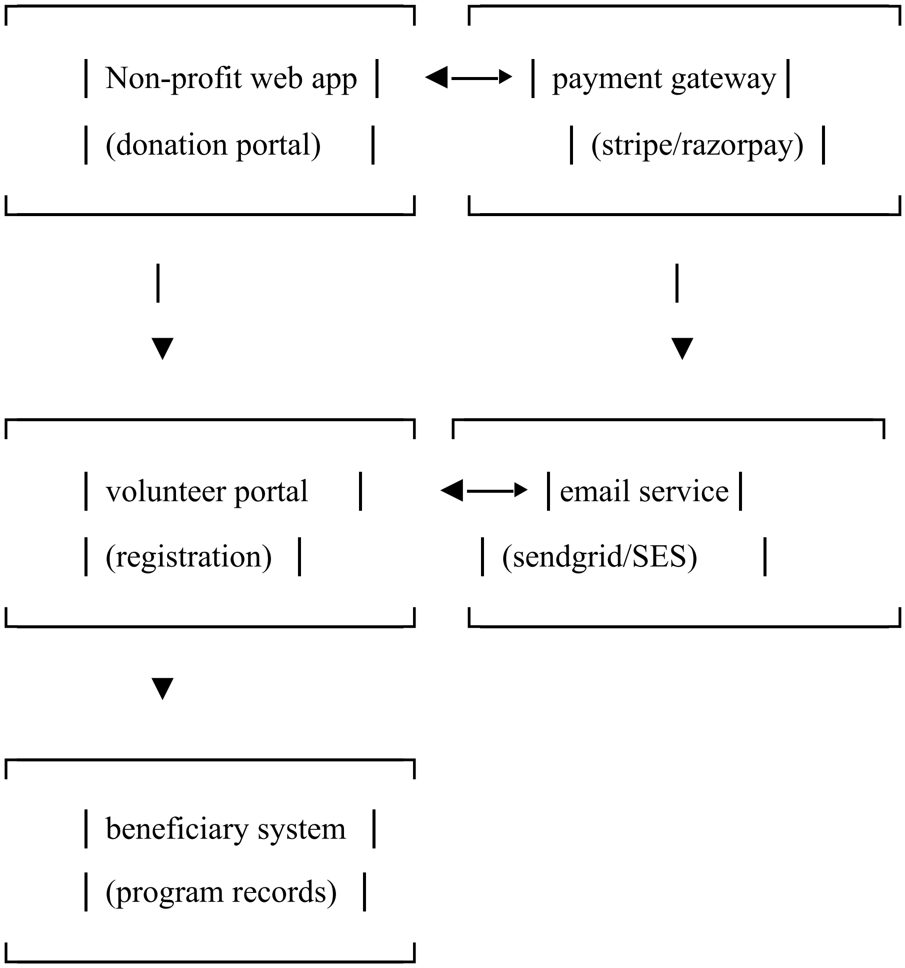
.

3.4 Data Migration

- State Import: terraform import aws_instance.agri_app i-123456789
- Config Migration: Ansible playbooks for app config
- Database Migration: Custom scripts for Non-profitdata

4. Interfaces

- External Interfaces:



- Internal Interfaces:

Interface	Description
Kubernetes to Docker	Container control
Terraform to AWS	Infrastructure provisioning

5. State and Session Management

- Terraform State: AWS S3 backend (agri-iac-state-bucket) with DynamoDB locking prevents concurrent apply operations across CI/CD pipelines.
- Kubernetes StatefulSets: Manages Non-profitdatabases (PostgreSQL for sensor data) with stable pod identities (agri-db-0, agri-db-1) and ordered scaling.
- CI/CD Sessions: Jenkins workspace persistence + environment-specific S3 state keys (dev/prod).

Type	Storage	Locking
IaC	S3	DynamoDB
DB	PVCs	StatefulSet
Pipeline	Workspace	Mutex

6. Caching

Multi-Level Caching Strategy

Layer	Technology	TTL	Purpose
Docker	Docker Layer Cache	Session	Accelerates image rebuilds during CI/CD piplines
Terraform	.terraform cache	24h	Caches provider plugins and modules for faster provisioning
Kubernetes	EmptyDir volumes	Pod lifecycle	Temorary storage for request processing and report generation
Application	Redis	1h	Caches donation summaries,campaign stats,and frequently accessed program data

7. Non-Functional Requirements

7.1 Security Aspects

Control	Implementation	Compliance
IAM	Least privilege roles	AWS Well-Architected
Secrets	AWS Secrets Manager	NIST 800-53
Network	VPC + Security Groups	CIS Benchmarks
Encryption	KMS + TLS 1.3	GDPR
Audit	CloudTrail + GuardDuty	SOC 2

7.2 Performance Aspects

Metric	Target	Measurement
Provision Time	<10 min	Terraform apply
App Startup	<2 min	Kubernetes readiness
API Latency	P95 <200ms	CloudWatch
Throughput	1000 req/s	Load testing

8. References

- Terraform Documentation: https://www.terraform.io/docs
- AWS Well-Architected Framework:

  https://aws.amazon.com/architecture/wellarchitected/
- Kubernetes Best Practices: https://kubernetes.io/docs/concepts/
- Ansible Automation: https://docs.ansible.com/
- Docker Security: https://docs.docker.com/engine/security/