Project - High Level Designon

on

NGO IAC Provisioning

Course Name: PROJECT WORK - II

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Aayush Banwadikar	EN22ME304002
2	Divyanshu Mehta	EN22EL301024
3	Aditya Kumar Sahu	EN22ME304005
4	Vinay Patidar	EN22CS3011085
5	Yatendra Verma	EN22ME304118

Group Name: 02D10

Project Number: : D0-15

Industry Mentor Name:

University Mentor Name: Prof. Avnesh Joshi

Academic Year: 2026

Table of Contents

1. Introduction.

This document provides a comprehensive overview of the **IaC Provisioning for Non-Profit System** project. The purpose of this project is to automate the deployment and configuration of infrastructure using Infrastructure as Code (IaC) tools such as Terraform and Ansible.

The system ensures consistent, repeatable, and scalable environment setup for non-profit organizations by eliminating manual configuration errors. It covers the automated provisioning of local Docker environments or cloud-based infrastructure (e.g., AWS), along with the installation of required dependencies and services.

The document outlines the project objectives, scope, tools used, architecture, workflow, and implementation strategy.

1.1 Scope of the document.

This document covers:

- The overall project objectives and problem statement
- Infrastructure provisioning using Terraform/Ansible
- Automated deployment using Docker or cloud platforms
- Configuration management and dependency installation
- Workflow and architecture explanation
- Tools and technologies used
- Key features and expected outcomes

- Security and scalability considerations

It does not cover detailed organizational policies or unrelated system modules outside the IaC provisioning framework.

1.2 Intended Audience

This document is intended for:

- Project Developers – To understand the implementation process and workflow
- DevOps Engineers – For infrastructure automation and deployment practices
- System Administrators – For managing and maintaining deployed environments
- Project Managers – To track project scope, deliverables, and objectives
- Non-Profit Organization IT Teams – To understand system setup and benefits

The document assumes basic knowledge of cloud computing, Docker, and Infrastructure as Code concepts.

1.3 System overview

The IaC Provisioning for Non-Profit System automates infrastructure deployment using Infrastructure as Code (IaC) tools such as Terraform and Ansible.
The system provisions:

- Cloud infrastructure (AWS EC2, VPC, S3, IAM, etc.) or Local Docker environments

- Application dependencies and runtime environments
- Network and security configurations
- CI/CD compatible deployment structure

The solution ensures:

- Repeatable deployments
- Reduced manual errors
- Scalability and high availability
- Environment consistency across development, testing, and production

2. System Design

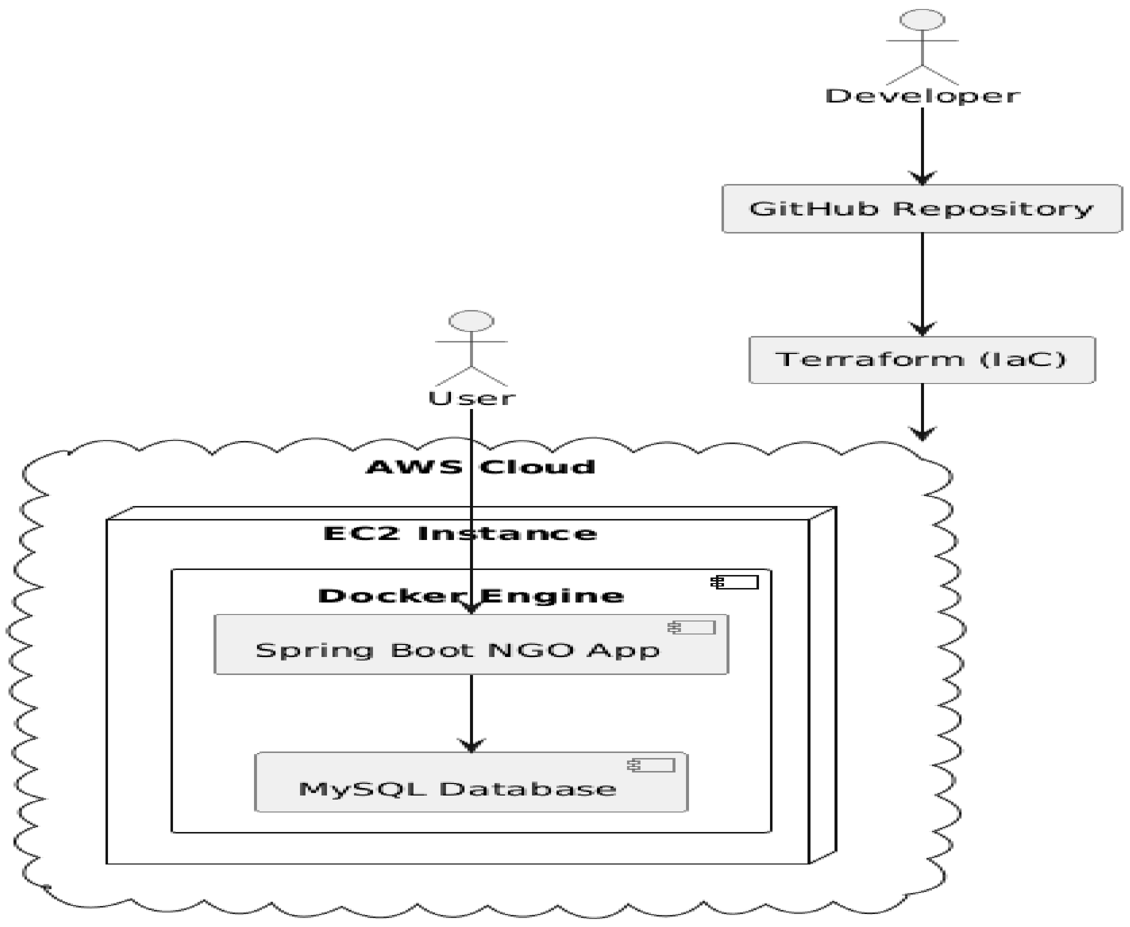
2.1 Application Design

The application follows a **modular and layered architecture**:
Layers:

- Infrastructure Layer (Terraform Scripts)
- Configuration Layer (Ansible Playbooks)
- Container Layer (Docker)
- Version Control Layer (Git)
- Cloud/Local Execution Layer (AWS / Local System)

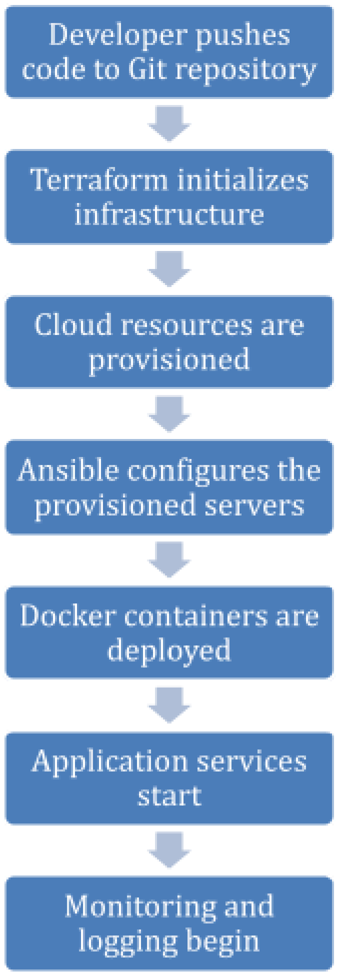
Design Principles:

- Infrastructure as Code
- Immutable Infrastructure
- Idempotent Configuration
- Modular Terraform Modules
- Reusable Ansible Roles

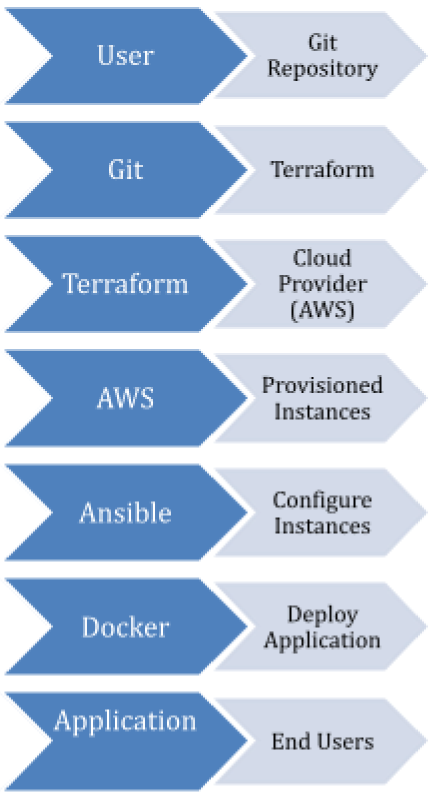


2.2 Process Flow

Step-by-step Process Flow:


Developer pushes code to Git repository
Terraform initializes infrastructure
Cloud resources are provisioned
Ansible configures the provisioned servers
Docker containers are deployed
Application services start
Monitoring and logging begin

2.3 Information Flow



2.4 Components Design

Core Components:

1. Version Control System (Git)
2. Terraform Modules
3. Ansible Playbooks
4. Docker Engine
5. Cloud Provider (AWS)
6. Monitoring & Logging Tools

Each component operates independently but integrates through automated scripts.

2.5 Key Design Considerations

1. Scalability
2. Security (IAM roles, SSH key management)
3. High Availability

4. Infrastructure Modularity
5. Environment Isolation
6. Cost Optimization
7. Disaster Recovery Readiness
8. Idempotent Configuration

2.6 API Catalogue

S No.	API Name	Purpose	Method	Description
1	AWS EC2 API	Instance Creation	POST	Creates virtual machines
2	AWS S3 API	Storage Management	PUT/GET	Stores files & backups
3	Docker API	Container Control	REST	Manages containers
4	Terraform CLI	Infrastructure Execution	CLI	Applies infrastructure code
5	Ansible API	Configuration Mgmt	SSH	Configures remote machines

3. Data Design

3.1 Data Model

The system primarily handles:

1. Infrastructure State Files (Terraform state)
2. Configuration Files (YAML, HCL)
3. Logs and Monitoring Data
4. Application Database Data (if applicable)
    a. Data Entities:
5. Infrastructure Resources
6. Deployment Configurations
7. User Credentials
8. Logs

3.2 Data Access Mechanism

1. Secure SSH access
2. IAM-based access control
3. Role-based authentication
4. Encrypted S3 backend for Terraform state
5. REST APIs for application interaction

3.3 Data Retention Policies

1. Terraform state stored securely (S3 with versioning)
2. Logs retained for defined time (30–90 days)
3. Backup retention policy (Weekly/Monthly backups)
4. Encrypted storage of sensitive data

3.4 Data Migration

1. Database backup and restore strategy
2. Version-controlled schema migrations
3. Automated migration scripts
4. Zero-downtime deployment strategy

4. Interfaces

1. CLI Interface (Terraform & Ansible)
2. Cloud Console Interface (AWS)
3. Web Application Interface (if deployed)
4. REST APIs
5. Monitoring Dashboard

5. State and Session Management

1. Terraform Remote Backend (S3 + Locking)
2. Stateless Application Containers
3. Session stored in:

- Database
- Redis Cache (optional)

4. IAM session management

6. Caching

1) Redis (Optional)
2) Docker image layer caching
3) Terraform plugin caching
4) CDN caching (if cloud-based web app)

7. Non-Functional Requirements

7.1 Security Aspects

1. IAM Role-Based Access Control
2. Encrypted Storage (S3 Encryption)
3. HTTPS Communication
4. SSH Key Authentication
5. Secrets Management
6. Firewall & Security Groups

7.2 Performance Aspects

1. Auto Scaling Groups
2. Load Balancers
3. Optimized Docker images
4. Efficient Terraform Modules
5. Parallel resource provisioning
6. Monitoring with alerts

8. References

1. Terraform Documentation
2. Ansible Documentation
3. Docker Documentation
4. AWS Documentation
5. DevOps Best Practices
6. Infrastructure as Code Principles