**Phase-2 Submission Template Student Name:** Vishal S

**Register Number:** 511823205043

**Institution:** Podhigai College of Engineering And Technology

**Department:** B.tech-Information Technology

**Date of Submission: 19-05-2025**

# Problem Statement

* Manual cloud provisioning is slow and error-prone.
* Inconsistent configurations across environments cause reliability issues.
* Lack of automation reduces deployment speed and scalability.
* Infrastructure changes are hard to track without version control.
* A combined IaC approach using Terraform and Ansible is needed for full automation.
* Automation improves consistency, repeatability, and operational efficiency.

# Project Objectives

#  Automate cloud infrastructure provisioning using Terraform.

#  Configure provisioned servers using Ansible.

#  Ensure consistency across development, staging, and production environments.

#  Enable version control for infrastructure and configurations.

#  Reduce manual errors and improve deployment speed.

#  Create a scalable and repeatable Infrastructure as Code (IaC) workflow.

#  Integrate Terraform and Ansible for end-to-end automation.

# 

# Flowchart of the Project Workflow

Requirement Analysis

Environment Setup

Terraform Implementation

Inventory Generation

Configuration with Ansible

Test & Automate

# Infrastructure Setup

 **Cloud Infrastructure Provisioning with Terraform**  
 Terraform is used to create cloud resources like VPCs, subnets, EC2 instances, security groups, and load balancers automatically.

 **Compute Instance Setup**  
Terraform provisions virtual machines (e.g., AWS EC2) with necessary configurations like SSH access, instance types, and networking.

 **Configuration Management with Ansible**  
Ansible configures the provisioned servers by installing software, deploying applications, and setting up services like Nginx or Docker.

 **Code Organization and Version Control**  
All Terraform and Ansible code is stored in a structured Git repository to support collaboration, reusability, and version control.

# 5.CI/CD Pipeline Implementation

 **Version Control with Git**  
All Terraform and Ansible code is maintained in a Git repository, triggering pipeline actions on code changes (e.g., via GitHub/GitLab).

 **CI/CD Tool Integration (e.g., Jenkins, GitLab CI)**  
A CI/CD tool runs automated jobs for linting, testing, Terraform plan/apply, and Ansible playbook execution.

 **Terraform Automation**  
The pipeline runs terraform init, terraform plan, and terraform apply to provision infrastructure automatically on code push.

 **Ansible Automation**  
After infrastructure provisioning, the pipeline executes Ansible playbooks to configure servers and deploy applications.

 **Environment Segregation**  
Separate stages for dev, test, and production environments using different branches, workspaces, or variable files.

# 6.Security and Compliance Integration

 **IAM Role and Policy Management**  
Terraform defines strict IAM roles and policies to grant least-privilege access to resources.

 **Secrets Management**  
Sensitive data like passwords and API keys are stored securely using tools like AWS Secrets Manager or Ansible Vault.

 **Security Group and Firewall Rules**  
Only required ports (e.g., 22, 80, 443) are allowed via Terraform-managed security groups, blocking unnecessary access.

 **Code Scanning and Policy Checks**  
Use tools like TFLint, Checkov, or Terraform Cloud to scan IaC code for misconfigurations and compliance violations.

 **OS and Package Hardening**  
Ansible playbooks apply security patches, disable root login, enforce SSH key authentication, and install firewalls (e.g., UFW).

# 7.Monitoring, Logging, and Alerts

# CloudWatch is used to monitor infrastructure metrics like CPU, memory, and network.

* Logs are centralized using CloudWatch Logs or ELK Stack for easier debugging.
* Ansible installs monitoring agents on all provisioned cloud instances.
* Alerts are configured for failures or threshold breaches via email or Slack.

# 8.Testing and Validation Strategy

 **Terraform Plan Review**  
Use terraform plan to preview infrastructure changes and validate configurations before applying them.

 **Linting and Static Code Analysis**  
Tools like tflint, terraform validate, and ansible-lint are used to catch syntax errors and enforce best practices.

 **Infrastructure Testing with Terratest**  
Automated tests written in Go using Terratest validate that infrastructure behaves as expected after deployment.

 **Ansible Playbook Dry Runs**  
Use --check and --diff flags in Ansible to simulate playbook execution and verify intended changes.

 **Environment-Based Testing**  
Apply changes in dev/test environments first, validate functionality, then promote to staging and production.

# 9.Tools and Technologies Used

1. **Terraform** – For provisioning and managing cloud infrastructure using Infrastructure as Code.
2. **Ansible** – For configuration management, software installation, and post-deployment automation.
3. **AWS / Azure / GCP** – Cloud provider platform for hosting infrastructure and services.
4. **Git** – Version control system to track changes in Terraform and Ansible code.
5. **Jenkins / GitLab CI / GitHub Actions** – CI/CD tools for automating deployment pipelines.
6. **Terraform Cloud / TFLint / Checkov** – For code scanning, validation, and policy enforcement.
7. **CloudWatch / ELK Stack / Prometheus** – For monitoring, logging, and performance tracking.
8. **Ansible Vault / AWS Secrets Manager** – To securely manage sensitive information and credentials.
9. **Visual Studio Code / IntelliJ / Terminal** – Common development environments and CLI tools.

# 10.Team Members and Contributions

* **Vishal S** – Worked on cloud infrastructure setup using Terraform.
* **Prasanth A** – Handled server configuration and automation using Ansible.
* **Prakash** S – Managed CI/CD pipeline integration for automated deployments.
* **Kalaiarsan** K – Took care of security setup and monitoring configuration.