**Phase-3 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**

**GitHub Repository link:**

https://github.com/vichuvishal206/Vishal.project.oracle.git

# Problem Statement

# Modern software applications demand fast, scalable, and consistent infrastructure. Traditionally, provisioning cloud infrastructure and configuring systems has been done manually, leading to significant challenges in reliability, scalability, and efficiency. This manual approach is often time-consuming, error-prone, and difficult to reproduce across development, testing, and production environments.

As organizations increasingly adopt cloud platforms such as AWS, Azure, or Google Cloud, the need for automated, repeatable, and version-controlled infrastructure becomes critical. The absence of automation results in:

* Inconsistent environments due to manual provisioning and configuration.
* Increased risk of configuration drift across servers and environments.
* Lack of visibility and traceability in infrastructure changes.
* Delays in deployment due to human dependency and errors.

# 

# 2.Abstract

# 

# This project demonstrates the implementation of Infrastructure as Code (IaC) using Terraform and Ansible for automated cloud provisioning and configuration. Terraform is used to provision cloud resources such as virtual machines, networks, and security groups, while Ansible handles the configuration of software and services on these resources. By automating the infrastructure lifecycle, this approach ensures consistency, scalability, and reduces manual errors. The project showcases how IaC can streamline cloud deployments and support DevOps practices through version-controlled, repeatable infrastructure setups.

# 3.System Requirements

### 

### **Hardware Requirements:**

* **Processor**: Intel i5 or equivalent.
* **RAM**: 8 GB or more.
* **Storage**: 50 GB free disk space.
* **Network**: Stable internet connection.

### **Software Requirements:**

* **Operating System**:
  + Linux (Ubuntu 20.04+), Windows (with WSL 2), or macOS.
* **Core Software**:
  + **Terraform**: Version 1.0+.
  + **Ansible**: Version 2.10+.
  + **Python**: Version 3.8+ (for Ansible).
  + **Cloud CLI**: AWS CLI, Azure CLI, or gcloud CLI.
  + **Git**: For version control.
  + **Text Editor/IDE**: e.g., Visual Studio Code.

### **Cloud Platform Requirements:**

* **AWS**: AWS Account with IAM user permissions.
* **Azure**: Azure Subscription with Service Principal.
* **GCP**: Google Cloud Project with Service Account permissions.

### **Optional Tools:**

* **Docker**: For containerizing Terraform/Ansible.
* **CI/CD Tools**: GitHub Actions, Jenkins for automation.

# 4.Objectives

#  Automate cloud infrastructure setup using Terraform.

#  Configure servers and software using Ansible.

#  Implement Infrastructure as Code (IaC) for consistency and repeatability.

#  Reduce manual errors and human intervention.

#  Ensure consistent environments across dev, test, and production.

#  Speed up deployment and scaling of infrastructure.

#  Enable version control for infrastructure changes.

#  Demonstrate end-to-end deployment of a cloud-based application.

# 5.Workflow / Architecture Diagram

 **Start**  
→ Developer triggers the deployment (locally or via CI/CD).

 **Terraform Initialization**  
→ Terraform initializes and loads required providers (e.g., AWS, Azure, GCP).

 **Terraform Plan & Apply**  
→ Provisions cloud infrastructure (e.g., VPC, EC2, Subnets, Security Groups).

 **Infrastructure Output**  
→ Terraform outputs instance IPs or resource IDs.

 **Ansible Inventory Setup**  
→ Use Terraform outputs to generate Ansible inventory (hosts list).

 **Ansible Playbook Execution**  
→ Ansible SSHs into provisioned servers to install and configure software (e.g., NGINX, MySQL).

 **Application Ready**  
→ Complete environment is provisioned and configured automatically.

# 6.Application / System Description

# 

# This project uses Terraform and Ansible to automate the provisioning and configuration of cloud infrastructure following the Infrastructure as Code (IaC) approach. Terraform is used to create cloud resources such as virtual machines, networks, and security groups, while Ansible is used to install and configure software like web servers and databases on those machines. The system ensures fast, consistent, and repeatable deployments across cloud environments (e.g., AWS, Azure, GCP), reducing manual work and errors.

# 7.Application / System Description

# Terraform is responsible for provisioning cloud resources such as virtual machines, networks (VPCs/subnets), and security groups on platforms like AWS, Azure, or GCP. Once the infrastructure is provisioned, Ansible connects to the virtual machines and installs necessary software, configures services, and manages system settings (e.g., Nginx, databases, users).

# The system enables consistent, repeatable, and scalable deployments by defining all infrastructure and configuration as code, stored in version control. This eliminates manual setup, reduces errors, and aligns with modern DevOps practices.

# 8.Configuration & Setup

 **Install Required Tools:**

* Terraform
* Ansible
* Python (required for Ansible)
* Cloud CLI (e.g., AWS CLI, Azure CLI, or gcloud)

 **Configure Cloud Access:**

* Set up credentials for the chosen cloud provider (e.g., AWS Access Key & Secret Key).
* Verify access by using CLI commands like aws configure.

 **Write Terraform Configuration Files:**

* Define infrastructure components such as VPC, subnets, virtual machines, and security groups.

 **Run Terraform Commands:**

* terraform init – Initialize the working directory.
* terraform plan – Preview the infrastructure changes.
* terraform apply – Provision the defined infrastructure.

 **Prepare Ansible Inventory:**

* Use Terraform outputs (e.g., IP addresses) to create a dynamic or static Ansible inventory.

 **Write Ansible Playbooks:**

* Create YAML files to install and configure applications (e.g., Nginx, MySQL).

 **Execute Ansible Playbooks:**

* Use ansible-playbook to connect and configure the provisioned servers.

# 9.CI/CD Pipeline Stages

1. **Source Code Repository**
   * Push code (Terraform scripts, Ansible playbooks) to a Git repository (e.g., GitHub, GitLab).
2. **Build / Initialization**
   * Set up the environment (e.g., install Terraform, Ansible, dependencies).
   * Initialize Terraform configuration using terraform init.
3. **Terraform Plan Stage**
   * Run terraform plan to check for changes in the infrastructure and preview the deployment.
4. **Terraform Apply Stage**
   * Execute terraform apply to provision or update cloud infrastructure automatically.
5. **Ansible Configuration**
   * Use Terraform output (e.g., instance IPs) to generate the Ansible inventory.
   * Run Ansible playbooks to configure provisioned resources (install web servers, databases, etc.).
6. **Testing / Validation**
   * Run automated tests to verify infrastructure (e.g., check if EC2 instances are up, security groups are configured correctly).
   * Use Ansible to verify configuration (e.g., ensure services are running).
7. **Deployment / Production**
   * Once tests pass, deploy the application to production servers.
   * Ensure the environment is fully functional and accessible.
8. **Monitoring / Alerts**
   * Set up monitoring tools (e.g., CloudWatch, Datadog) to ensure the infrastructure and application are working properly.
   * Send alerts if any issues are detected.

# 10.Containerization & Orchestration

 **Containerization**:  
Use **Docker** to package Terraform and Ansible into containers for consistency across environments, enabling easy deployment and scaling.

 **Orchestration**:  
Leverage **Kubernetes** or **Docker Swarm** to manage and orchestrate containers, ensuring automated scaling, load balancing, and high availability.

# 11.Infrastructure as Code (IaC)

**Infrastructure as Code (IaC)** is a practice where infrastructure is managed and provisioned using code, instead of manual processes. In this project, **Terraform** and **Ansible** are used to define, deploy, and configure cloud resources automatically.

* **Terraform**: Defines and provisions cloud infrastructure resources (e.g., virtual machines, networks, storage) in a declarative manner.
* **Ansible**: Manages configuration and deployment of software on provisioned servers, ensuring consistency across environments.

By using IaC, the entire infrastructure is version-controlled, repeatable, and easily scalable, reducing the risk of human error and improving deployment speed.

# 12.Monitoring & Logging

 **Monitoring**:

* Use cloud-native tools like **AWS CloudWatch**, **Azure Monitor**, or **Google Cloud Operations** to track infrastructure health, resource usage, and network traffic.
* Implement **Prometheus** or **Datadog** for application performance monitoring and auto-scaling alerts.

 **Logging**:

* Centralize logs with **AWS CloudWatch Logs**, **Azure Log Analytics**, or **Google Stackdriver**.
* Aggregate logs using **ELK Stack** or **Fluentd** for detailed analysis and troubleshooting.

 **Alerting**:

* Set up alerts for resource anomalies, deployment failures, and application issues via **Slack**, **PagerDuty**, or email notifications.

# 13.Deployment & Access

 **Provision Infrastructure**:

* Use **Terraform** to define and provision cloud resources (e.g., VMs, networks, security groups).

 **Server Configuration**:

* Utilize **Ansible** to configure servers post-deployment (e.g., install software, configure services).

 **Access Control**:

* Implement **IAM roles** and **security groups** to manage access to resources.
* Use **SSH keys** for secure server access.

 **CI/CD Pipeline**:

* Automate infrastructure deployment and configuration using a CI/CD pipeline (e.g., GitHub Actions, Jenkins).

# 14.Source Code Repository

1. **Terraform Files**:
   * Store **Terraform configuration files** (main.tf, variables.tf, outputs.tf, provider.tf) for cloud infrastructure provisioning.
2. **Ansible Playbooks**:
   * Store **Ansible playbooks** (setup.yml, site.yml) for server configuration and software installation.
3. **CI/CD Integration**:
   * Automate deployment with **CI/CD pipeline configurations** (e.g., GitHub Actions, Jenkins).
4. **Version Control**:
   * Use **Git** to manage code, track changes, and enable collaboration (e.g., **GitFlow** branching).
5. **Access Control**:
   * Implement repository permissions for secure collaboration and code changes.

# 15.Future scope

 **Multi-Cloud Support**:

* Expand to support AWS, Azure, GCP with Terraform.

 **Security Enhancements**:

* Implement advanced security configurations and secret management.

 **Kubernetes Integration**:

* Integrate with Kubernetes for containerized application management.

 **Automated Testing**:

* Introduce automated infrastructure testing to validate deployment and configurations.

 **Monitoring & Self-Healing**:

* Integrate AI-based monitoring and automated resource scaling.

 **Cost Optimization**:

* Implement tools for cloud cost analysis and automatic scaling based on usage.

 **Disaster Recovery**:

* Design disaster recovery plans and high-availability architecture.

# 16.Team Members and Roles

 **Vishal** S – Oversees the project, manages timelines, and coordinates team efforts.

 **Prasanth A** – Handles cloud provisioning and resource management using **Terraform**.

 **Prakash** S– Develops and manages **Ansible** playbooks for server configuration .

 **Kalaiarasan** K– Implements **CI/CD pipelines** and ensures security best practices are followed.