Manufacturing Operational Insights Report

IAC Provisioning for Telecomm System

Course Name: DevOps

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
01	**Aayushi Solanki**	**EN24CA5030003**
02	**Ayushi Pal**	**EN24CA5030033**
03	**Avdhesh Bhadoriya**	**EN22IT301026**
04	**Kanak Kumari**	**EN24CA5030076**
05	**Ramji Soni**	**EN22IT301078**

Group Name: G01D12

Project Number: DO-39

Industry Mentor Name: Vaibhav

University Mentor Name: Akshay Saxena

Academic Year: 2025-2026

1. Introduction

1.1. Scope of the document

This document defines the high-level architecture and design for automating infrastructure provisioning in a Telecomm system. It focuses on the use of Infrastructure as Code (IaC) tools to create a repeatable and scalable environment.

1.2. Intended Audience

- **Project Supervisors**: To evaluate the technical feasibility and design accuracy.
- **DevOps Engineers**: For understanding the automation workflow.
- **System Architects**: To review the integration of K8s, Docker, and CI/CD pipelines.

1.3. System overview

The system automates the deployment of local Docker environments and Kubernetes clusters using Terraform and Jenkins. By treating infrastructure as code, the system eliminates manual configuration errors, ensuring that the telecom microservices are deployed consistently across different environments.

System Design

1.1. Application Design

The application is based on a **Microservices Architecture** where telecom-specific services are encapsulated within Docker containers. These containers are orchestrated by Kubernetes nodes to ensure high availability.

1.2. Process Flow

The process begins at the Developer's Machine where IaC scripts are authored. The code is pushed to a Git Repository, which triggers a CI/CD Pipeline (Jenkins) to execute Terraform plans and apply changes to the local Docker host or cloud instances.

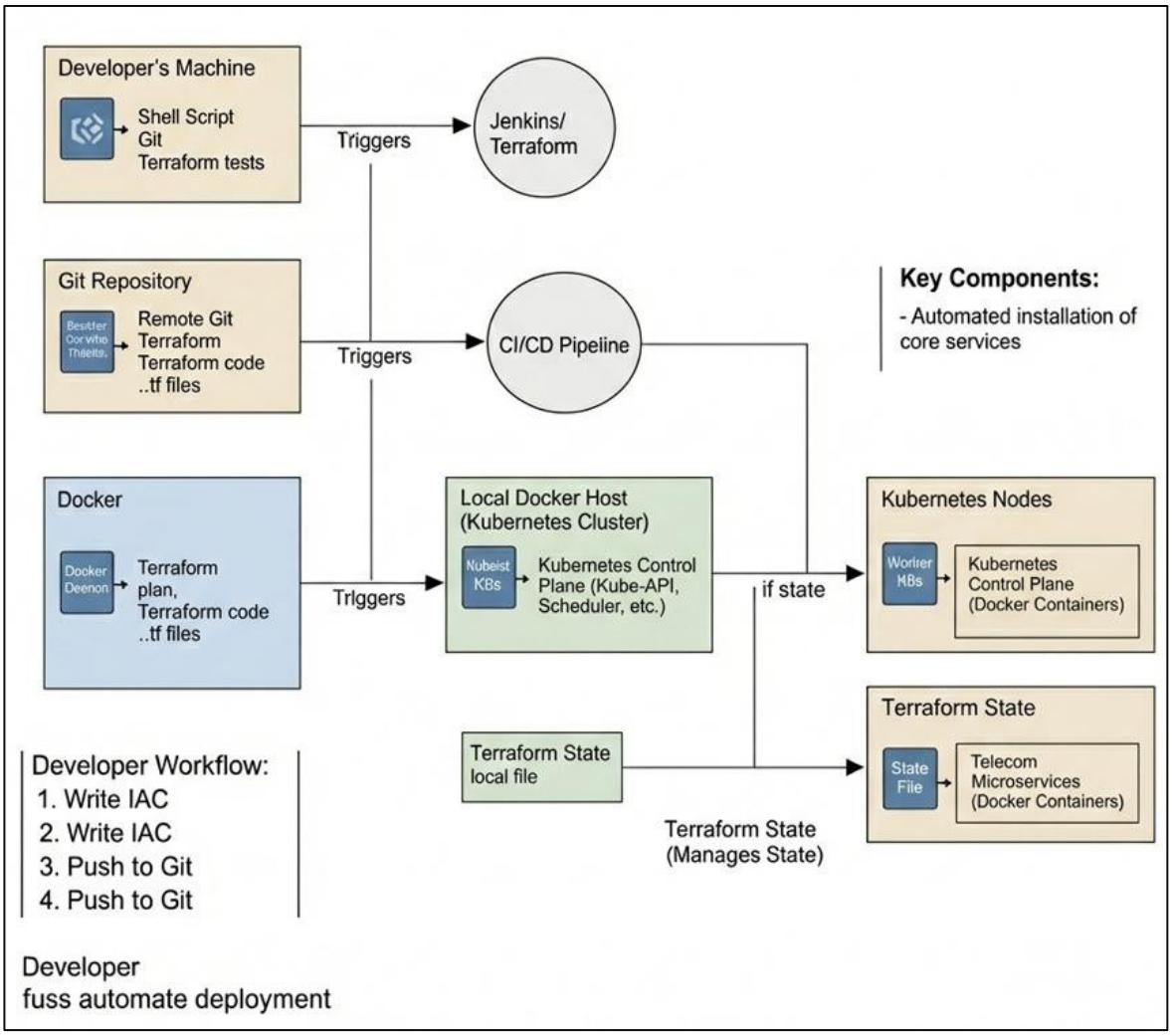

Figure 1: High-Level System Architecture

1.3. Information Flow

Information flows from the source (Git) through the automation engine (Jenkins/Terraform) to the target infrastructure (K8s/Docker). Terraform maintains a "State File" to track the mapping between the code and the real-world resources.

1.4. Components Design

- **Git Repository**: Serves as the single source of truth for infrastructure code.
- **Jenkins/CI-CD**: Acts as the automation controller that triggers deployments upon code changes.
- **Terraform**: The core IaC tool used to provision and manage resources.
- **Docker & Kubernetes**: The runtime environment for hosting telecom microservices.

1.5. Key Design Considerations

- **Modularity**: IaC code is divided into modules (e.g., Network, Compute, Apps) for better maintainability.
- **Scalability**: Using K8s allows the telecom system to scale resources dynamically based on demand.

1.6. API Catalogue

- **Kubernetes API**: Used by the CI/CD pipeline to communicate with the cluster and manage container lifecycles.

2. Data Design

2.1. Data Model

The project utilizes an **Infrastructure Meta-Model** to track entities such as IaC_Modules, Deployment_Jobs, Infrastructure_Resources, and Telecom_Services.

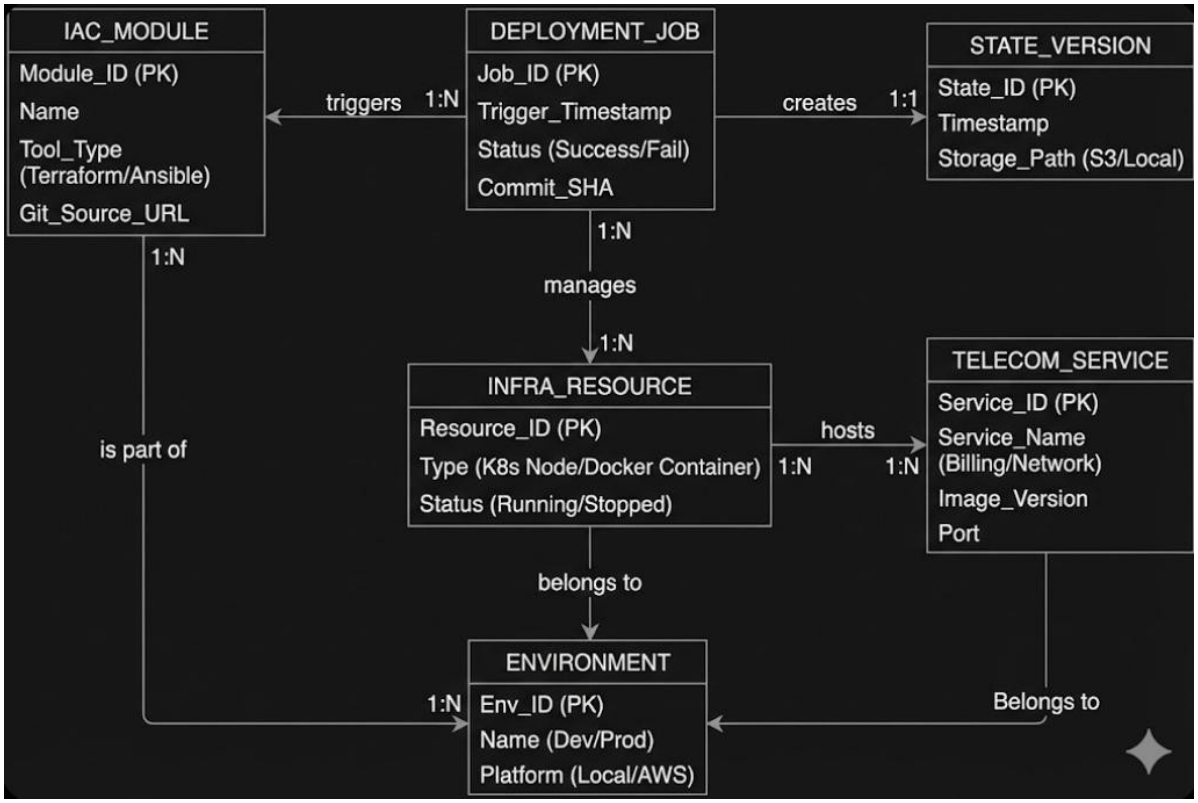

Figure 2: Infrastructure ER Diagram

Logical Relationships (The "Lines")

1. Job Triggers Module (1:N): One Jenkins Job can execute one or more IaC Modules.
2. Job Creates State (1:1): Every successful Deployment Job generates exactly one State Version.
3. Module Defines Resource (1:N): One IaC Module (code file) defines multiple Infrastructure Resources.
4. Resource Hosts Service (1:N): One Resource (like a Kubernetes Node) can host multiple Telecom Services (microservices).
5. Resource Belongs to Environment (N:1): Multiple Resources are grouped under one Environment (e.g., all Dev containers in the "Dev" environment).

2.2. Data Access Mechanism

Access to infrastructure state and code is controlled via Git permissions and Terraform backend configurations (e.g., S3 buckets with IAM roles).

2.3. Data Retention Policies

- **State History**: Previous versions of the Terraform state are retained for disaster recovery.
- **Audit Logs**: Jenkins build logs are kept to track who triggered which infrastructure change.

2.4. Data Migration

Infrastructure migration is handled via Terraform by updating the "Provider" configuration to move resources between local Docker hosts and cloud providers like AWS.

3. Interfaces

- **User Interface**: Jenkins Web Dashboard for monitoring pipeline status.
- **Command Line Interface (CLI)**: Terraform and Git CLI for developers.

4. State and Session Management

State management is handled through the **Terraform State File** (.tfstate). This file acts as a database that records the current state of the infrastructure, ensuring that Terraform knows which resources to create, update, or delete during a run.

5. Caching

- **Docker Layers**: Jenkins uses Docker layer caching to speed up the building of telecom service images.

- **Terraform Plugins**: Provider plugins are cached locally to reduce deployment time.

6. Functional Requirements

The functional requirements define the specific behaviors and services the Management Portal must provide to facilitate the IaC provisioning of the Telecom system.

6.1. User Authentication & Authorization

- **Registration & Login**: The system must allow authorized administrators to create accounts and log in securely.
- **Password Hashing:** The system must use bcrypt to securely hash and salt passwords before storing them in the database.
- **Session Management:** Secure user sessions must be maintained using jsonwebtoken (JWT) and cookie-parser to ensure only authenticated users can trigger infrastructure deployments.

6.2. Infrastructure Configuration Management

- **Environment Configuration:** The system must support the dynamic loading of environment-specific variables (e.g., AWS keys, Docker host IPs) using dotenv.
- **Script Uploads:** The system must provide an interface to upload infrastructure scripts or configuration files using multer.
- **Database Integration:** All metadata regarding deployments, resource statuses, and user activity must be stored and retrieved from a MongoDB database using mongoose.

6.3. Provisioning Control & Monitoring

- **Deployment Trigger:** Users must be able to trigger Terraform or Ansible scripts via the Express backend to provision Docker containers or K8s nodes.
- **Real-time Dashboard:** The system must render a dynamic web interface using EJS (Embedded JavaScript templates) to display the current status of the Telecom infrastructure.
- **Automated Installation:** Upon request, the system should trigger the automated installation of core dependencies on remote or local instances.

6.4. Telecom Service Orchestration

- **Microservice Control:** The portal must provide functionality to start, stop, or restart specific Telecom microservices (Billing, Network Management) hosted on the infrastructure.
- **Resource Mapping:** The system must map specific Infrastructure_Resources to their respective Telecom_Services for easy identification within the dashboard.

6.5. Dockerfile (Application Containerization)

- The Dockerfile is used to package the Telecom Management Portal into a portable image. This ensures that the environment is identical during development and production.

6.6. GitHub Actions (Workflow Configuration)

- GitHub Actions automates the CI/CD process. Whenever code is pushed to the main branch, it automatically builds the Docker image and runs Terraform tests.

6.7. CI/CD Strategy (Continuous Integration & Deployment)

This section describes the automated pipeline that connects your code to the live Telecom infrastructure.

A. Continuous Integration (CI)

- **Automated Testing:** Every push triggers a build to check for syntax errors in both Node.js (JavaScript) and Terraform (.tf) files.
- **Security Scanning:** Uses tools to check for hardcoded secrets or vulnerabilities in the npm packages (like bcrypt or jsonwebtoken).
- **Artifact Creation:** Upon successful testing, a Docker image is created, acting as a "Single Immutable Artifact."

B. Continuous Deployment (CD)

- **Infrastructure Provisioning:** Terraform automatically updates the state of the local Docker host or AWS instances to match the latest code configuration.
- **Container Orchestration:** The updated Docker image is pushed to a registry and pulled by the Kubernetes (K8s) nodes.
- **Zero Downtime:** The pipeline ensures that the "Telecom Billing" and "Network" services remain active while new updates are being rolled out.

7. Non-Functional Requirements

7.1. Security Aspects

- **Automated Installation**: Core dependencies are installed via scripts to ensure no manual security misconfigurations.
- **Secret Masking**: Sensitive data (passwords/API keys) is managed through Jenkins credentials or secret managers.

7.2. Performance Aspects

- **Efficiency**: Automation reduces environment setup time from hours to minutes.
- **Consistency**: Eliminates "Environment Drift" where different servers have different configurations.

8. References

- Problem Statement: **DO-39 IaC Provisioning for Telecomm System**.
- Project Group: **D12 - Group 01**.

9. GitHub - https://github.com/its-avdhesh/Telecom-infra-automation.git