Project - High Level Design

On

IaC Provisioning for Telecom 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	PRATIKSHA PANCHOLI	EN22ME304071
02	VIPIN PATEL	EN22CS3011087
03	KHUSHI RAM	EN22ME304053
04	ANUBRATA MALAKAR	EN22ME304012
05	YOGESH PATIL	EN22CS3011121

Group Name: 06D10

Project Number: DO-19

Industry Mentor Name:

University Mentor Name: Anvesh oshi

Academic Year:2022-2026

Table of content

1. Introduction.

This project demonstrates Infrastructure as Code (IaC) based provisioning of a Telecom application using Docker, Terraform, GitHub Actions, and Kubernetes. The system automates containerized application deployment and ensures repeatable, scalable, and error-free infrastructure setup.

1.1 Scope of the Document

This document describes:

- Infrastructure automation using Terraform
- Containerization using Docker
- CI/CD pipeline using GitHub Actions
- Kubernetes-based deployment configuration
- System architecture and execution flow

1.2 Intended Audience

This document is intended for:

- Project evaluators
- Faculty members
- DevOps learners
- System architects
- Developers

1.3 System Overview

The system demonstrates automated infrastructure provisioning for a telecom service using Infrastructure as Code (IaC).

The project includes:

- Git for version control
- Docker for containerization
- Terraform for infrastructure provisioning
- GitHub Actions for CI automation
- Kubernetes for scalable deployment configuration
- Docker Desktop as local runtime environment

The telecom application is a Flask-based service that runs inside a Docker container. Infrastructure provisioning and deployment are fully automated using Terraform and CI/CD workflows.

2. System Design.

2.1 Application Design

The system follows a layered architecture:

- Development Layer – Source code and infrastructure code are stored in a GitHub repository.
- CI/CD Layer – GitHub Actions automatically builds the Docker image on every code push.
- Infrastructure Layer – Terraform provisions Docker resources using Infrastructure as Code.
- Container Layer – Docker packages the telecom application and its dependencies into an image.
- Orchestration Layer – Kubernetes manages container replicas for scalable deployment.

2.2 Process Flow

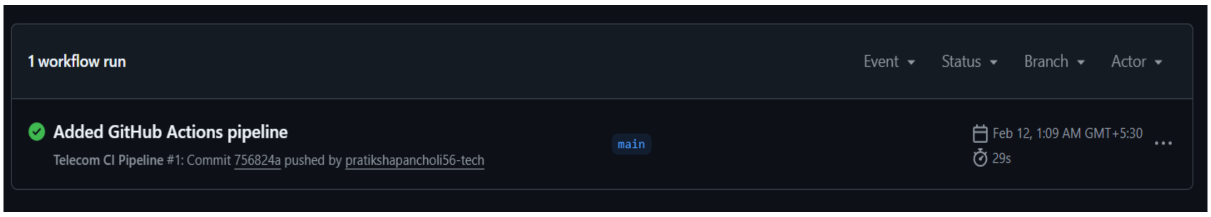
The high-level workflow is:

Developer → Git Push → GitHub Actions → Docker Image Build → Terraform Provisioning → Docker Container Created → Telecom Application Running

If Kubernetes is enabled:

Developer → Git Push → CI → Docker Image → Kubernetes Deployment → Telecom Pods Running

This pipeline eliminates manual server setup and ensures consistent deployments.



1 workflow run

Event ▾ Status ▾ Branch ▾ Actor ▾

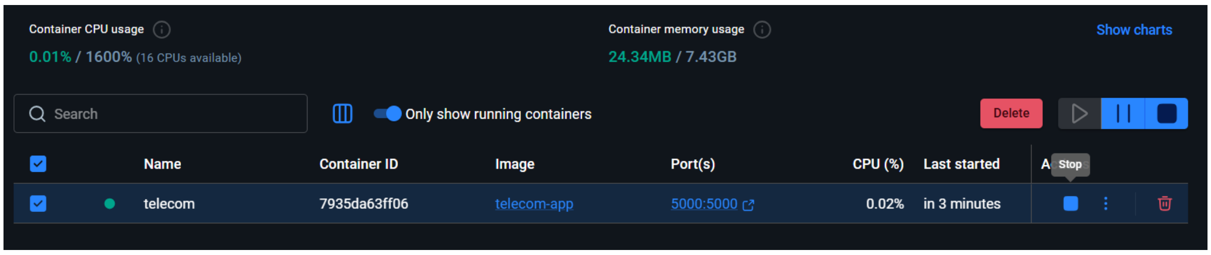
✓ Added GitHub Actions pipeline
Telecom CI Pipeline #1: Commit 756824a pushed by pratikshapancholi56-tech
main
Feb 12, 1:09 AM GMT+5:30
29s

Figure 1: GitHub Actions CI Pipeline Successful Execution

The screenshot shows the successful execution of the CI pipeline triggered automatically on code push. The pipeline builds the Docker image, validating the automated deployment process and eliminating manual build steps.

2.3 Information Flow

User → Browser → http://localhost:5000 → Docker Container → Flask Application → Response
The application processes HTTP requests and returns a simple telecom service response.
No persistent database is used in the current implementation.

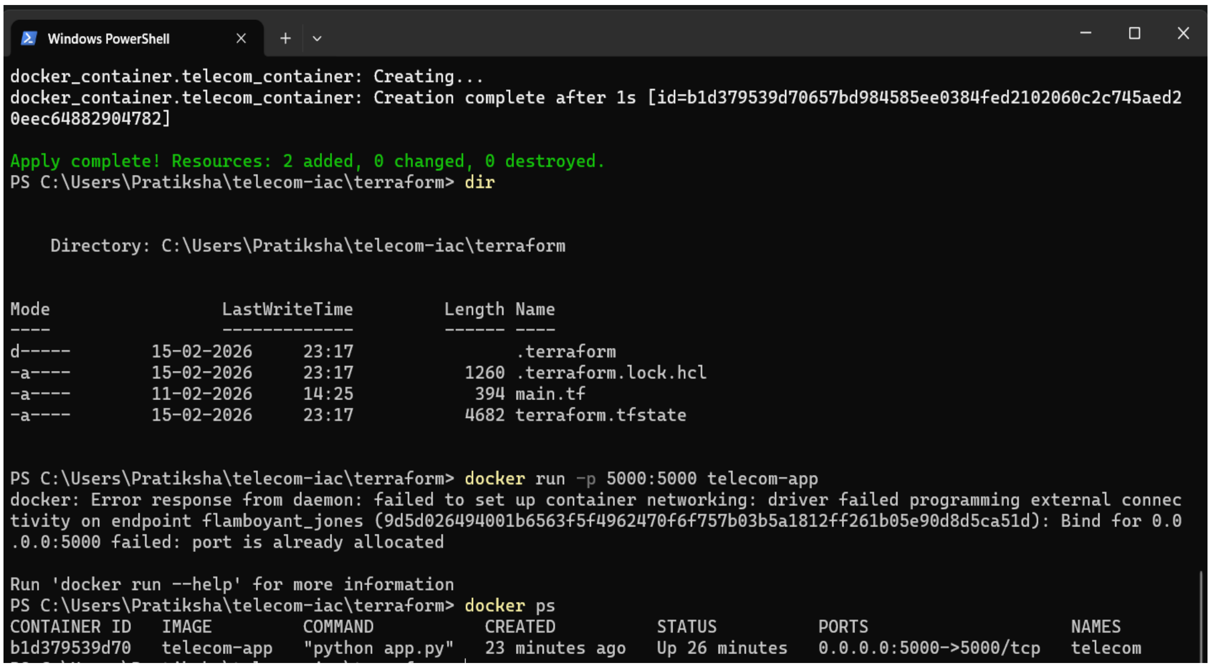


Container CPU usage
0.01% / 1600% (16 CPUs available)
Container memory usage
24.34MB / 7.43GB

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	A
✓ ●	telecom	7935da63ff06	telecom-app	5000:5000	0.02%	in 3 minutes	

Figure 2:Running Telecom Container in Docker Desktop

The above screenshot shows the running Docker container named "telecom," which was provisioned automatically using Terraform. The container uses the telecom-app image and exposes port 5000, making the telecom service accessible via browser. The green indicator confirms successful runtime deployment.



```
docker_container.telecom_container: Creating...
docker_container.telecom_container: Creation complete after 1s [id=b1d379539d70657bd984585ee0384fed2102060c2c745aed2
0eec64882904782]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\Pratiksha\telecom-iac\terraform> dir


    Directory: C:\Users\Pratiksha\telecom-iac\terraform


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        15-02-2026     23:17                .terraform
-a----        15-02-2026     23:17           1260 .terraform.lock.hcl
-a----        11-02-2026     14:25            394 main.tf
-a----        15-02-2026     23:17           4682 terraform.tfstate


PS C:\Users\Pratiksha\telecom-iac\terraform> docker run -p 5000:5000 telecom-app
docker: Error response from daemon: failed to set up container networking: driver failed programming external connec
tivity on endpoint flamboyant_jones (9d5d026494001b6563f5f4962470f6f757b03b5a1812ff261b05e90d8d5ca51d): Bind for 0.0
.0.0:5000 failed: port is already allocated

Run 'docker run --help' for more information
PS C:\Users\Pratiksha\telecom-iac\terraform> docker ps
CONTAINER ID   IMAGE         COMMAND           CREATED          STATUS          PORTS                    NAMES
b1d379539d70   telecom-app   "python app.py"   23 minutes ago   Up 26 minutes   0.0.0.0:5000->5000/tcp   telecom
```

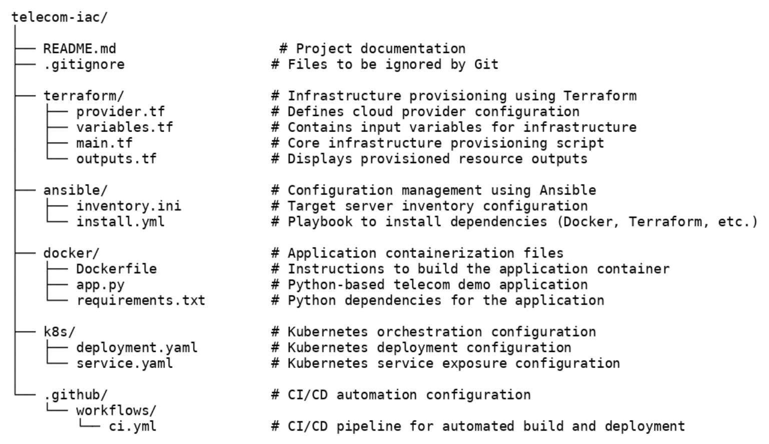
Figure 3:Terraform Infrastructure Provisioning Success Output

The above screenshot shows that Terraform successfully provisioned Docker resources using Infrastructure as Code. The output confirms automatic creation of two resources: the Docker image and the container.

2.4 Component Design

The system consists of the following components:

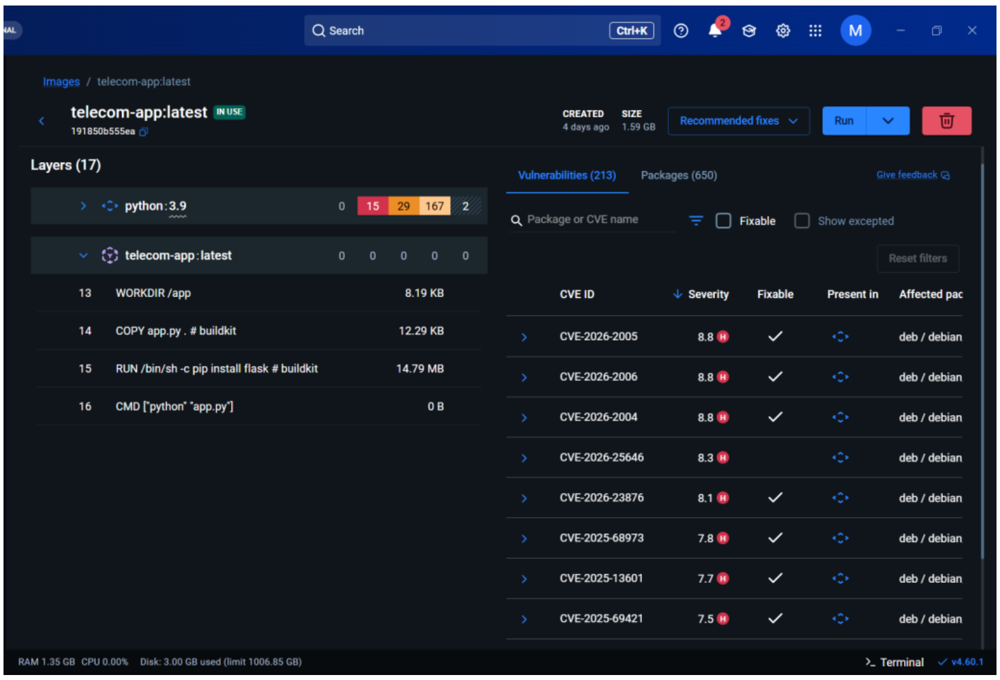
- GitHub Repository– Stores all infrastructure and application code.
- Docker– Builds the telecom application image from Dockerfile.
- Terraform – Provisions Docker image and container using Infrastructure as Code.
- GitHub Actions – Automates Docker image build on code push.
- Kubernetes – Manages container replicas for scalable deployment
- Docker Desktop – Acts as the local container runtime and Kubernetes cluster provider.



```
telecom-iac/
│
├── README.md                  # Project documentation
├── .gitignore                # Files to be ignored by Git
│
├── terraform/                # Infrastructure provisioning using Terraform
│   ├── provider.tf           # Defines cloud provider configuration
│   ├── variables.tf          # Contains input variables for infrastructure
│   ├── main.tf               # Core infrastructure provisioning script
│   └── outputs.tf            # Displays provisioned resource outputs
│
├── ansible/                  # Configuration management using Ansible
│   ├── inventory.ini         # Target server inventory configuration
│   └── install.yml           # Playbook to install dependencies (Docker, Terraform, etc.)
│
├── docker/                   # Application containerization files
│   ├── Dockerfile            # Instructions to build the application container
│   ├── app.py                # Python-based telecom demo application
│   └── requirements.txt      # Python dependencies for the application
│
├── k8s/                      # Kubernetes orchestration configuration
│   ├── deployment.yaml       # Kubernetes deployment configuration
│   └── service.yaml          # Kubernetes service exposure configuration
│
└── .github/                  # CI/CD automation configuration
    └── workflows/
        └── ci.yml            # CI/CD pipeline for automated build and deployment
```

Figure4: Add GitHub repository screenshot there

Detailed Folder Structure of Telecom IaC System with Component Descriptions
The image shows the organized repository layout including Terraform (IaC), Ansible (Configuration Management), Docker (Containerization), Kubernetes (Orchestration), and GitHub CI/CD workflow.



telecom-app:latest
191850b555ea

CREATED 4 days ago SIZE 1.59 GB

Layers (17)

python:3.9	0	15	29	167	2	
telecom-app:latest	0	0	0	0	0	

13	WORKDIR /app	8.19 KB
14	COPY app.py . # buildkit	12.29 KB
15	RUN /bin/sh -c pip install flask # buildkit	14.79 MB
16	CMD ["python" "app.py"]	0 B

Vulnerabilities (213) Packages (650)

CVE ID	Severity	Fixable	Present in	Affected pac
CVE-2026-2005	8.8 H	✓		deb / debian
CVE-2026-2006	8.8 H	✓		deb / debian
CVE-2026-2004	8.8 H	✓		deb / debian
CVE-2026-25646	8.3 H			deb / debian
CVE-2026-23876	8.1 H	✓		deb / debian
CVE-2025-68973	7.8 H	✓		deb / debian
CVE-2025-13601	7.7 H	✓		deb / debian
CVE-2025-69421	7.5 H	✓		deb / debian

RAM 1.35 GB CPU 0.00% Disk: 3.00 GB used (limit 1006.85 GB)

Figure 5:Docker Image telecom-app Built from Dockerfile (In Use)

The above screenshot shows the Docker image telecom-app built from the Dockerfile.
It contains the application code, dependencies, and runtime configuration. The image is currently in use by the running container.

2.5 Key Design Considerations

- Automation over manual setup
- Infrastructure defined in code
- Container-based deployment
- Modular architecture
- Scalability via Kubernetes
- Reproducibility across environments

2.6 API Catalogue

The telecom application exposes a REST endpoint:
GET /
Response:
"Telecom System Running!"
The application is implemented using Flask and listens on port 5000.
No external APIs or cloud APIs are used in the current implementation.

3. Data Design.

3.1 Data Model

The current implementation does not use a persistent database.

The application is stateless and processes HTTP request-response interactions.

3.2 Data Access Mechanism

The Flask application handles user requests directly. No database layer is integrated.

3.3 Data Retention Policies

Not applicable, as no persistent user data is stored.

3.4 Data Migration

Not applicable in the current implementation. Infrastructure updates are handled via Terraform, and container updates are managed via Kubernetes rolling updates.

4. Interfaces

System interfaces include:

- Git ↔ GitHub
- GitHub ↔ GitHub Actions
- GitHub Actions ↔ Docker
- Terraform ↔ Docker Provider
- Kubernetes ↔ Docker Engine
- User ↔ Browser ↔ Telecom Container

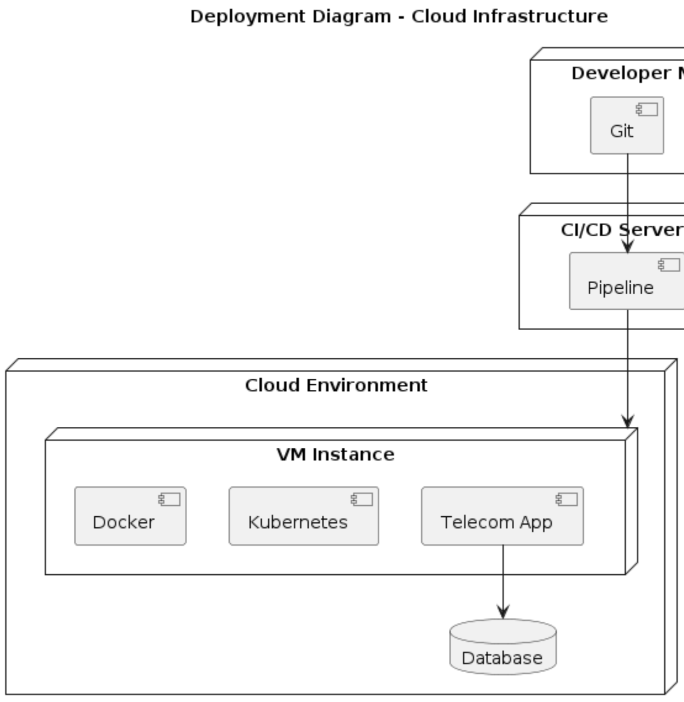


Figure 7

5. State and Session Management

The telecom application is stateless.
Each request is independently processed, enabling horizontal scaling in Kubernetes without session dependency issues.

6. Caching

Caching is not implemented in the current scope of the project. The architecture allows future integration of caching mechanisms such as Redis if required.

7. Non-Functional Requirements

7.1 Security Aspects

- Container isolation via Docker
- Controlled IaC provisioning via Terraform
- Version-controlled infrastructure
- Secure GitHub repository management
- Kubernetes RBAC (if cluster enabled)

7.2 Performance Aspects

- Containerized lightweight runtime
- Automated CI/CD reduces deployment time
- Kubernetes supports horizontal scaling
- Repeatable deployment ensures reliability

8. References

- Terraform Documentation – https://developer.hashicorp.com/terraform/docs
- Docker Documentation – https://docs.docker.com
- Kubernetes Documentation – https://kubernetes.io/docs
- GitHub Actions Documentation – https://docs.github.com/actions