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
1.	PRATIKSHA PANCHOLI	EN22ME304071
2.	VIPIN PATEL	EN22CS3011087
3.	KHUSHI RAM	EN22ME304053
4.	ANUBRATA MALAKAR	EN22ME304012
5.	YOGESH PATIL	EN22CS3011121

Group Name:06D10

Project Number:DO-19

Industry Mentor Name:

University Mentor Name: Avnesh Joshi

Academic Year:2025-26

Table of Contents

1. Introduction

The TelePay – Indian Telecom & Utility Payment System is a cloud-native and containerized application developed to automate telecom recharge and utility payment services using Infrastructure as Code (IaC) principles. The system supports multiple Indian telecom operators such as Airtel, Jio, Vi, and BSNL with real-time recharge processing and transaction tracking.

This project focuses on automating infrastructure provisioning using modern DevOps tools like Terraform, Docker, Kubernetes, and Ansible to eliminate manual configuration errors and ensure scalable deployment. The application is built using Python Flask for the backend and a responsive web interface using HTML, CSS, and JavaScript.

The system demonstrates how IaC, containerization, and cloud deployment can be integrated to build a production-ready telecom platform with automated setup, consistent environments, and scalable architecture. According to the project overview, the platform provides real-time recharge processing, RESTful APIs, and multi-deployment options including Docker, Kubernetes, and AWS cloud deployment

2. Problem Statement & Objectives

2.1 Problem Statement

Traditional telecom recharge systems often suffer from manual deployment, lack of scalability, environment inconsistency, and inefficient infrastructure management. Most small-scale systems require manual setup of dependencies such as Docker, Terraform, and cloud configurations, which leads to configuration errors and deployment failures.

Key Challenges:

- Manual infrastructure provisioning
- Lack of automation in deployment
- Environment inconsistency across systems
- Limited scalability of traditional recharge platforms
- No centralized transaction tracking system
- Difficulty in cloud deployment and management

2.2 Project Objectives

The main objective of this project is to design and deploy a fully automated telecom recharge system using Infrastructure as Code (IaC) with DevOps practices.

Technical Objectives:

- Implement a Flask-based RESTful web application
- Automate infrastructure provisioning using Terraform
- Containerize the application using Docker and Docker Compose
- Enable Kubernetes orchestration for scalability
- Implement CI/CD pipeline using GitHub Actions/Jenkin
- Deploy the application on AWS EC2 cloud
- Provide automated health checks and monitoring

Functional Objectives:

- Multi-operator mobile recharge (Airtel, Jio, Vi, BSNL)
- Real-time recharge processing with transaction IDs
- Transaction history tracking
- Mobile number validation (Indian format)
- User-friendly responsive web interface
- API-based recharge and plan retrieval system

2.3 Scope of the Project

The scope of this project includes full-stack application development, containerization, infrastructure automation, and cloud deployment. The system covers application development, DevOps automation, and scalable infrastructure provisioning.

Core Modules Covered:

- Telecom Recharge Module
- Utility Payment Module
- Transaction Management System
- RESTful API Services
- Infrastructure Automation (IaC)
- Cloud Deployment on AWS
- Container Orchestration using Kubernetes

3. Proposed Solution

3.1 Key Features

- Infrastructure as Code (IaC) using Terraform
- Docker containerization for portable deployment
- Kubernetes orchestration for scalability
- Multi-operator telecom recharge support
- Real-time transaction processing with unique transaction IDs
- RESTful API endpoints for recharge and plans
- Automated setup using Ansible scripts
- CI/CD pipeline integration (GitHub Actions & Jenkins)
- Health checks and monitoring support
- Responsive web-based user interface

The system also includes mobile validation, plan selection, and transaction history tracking to enhance user experience and system reliability

3.2 Overall Architecture / Workflow

The system follows a layered cloud-native architecture integrated with DevOps automation and Infrastructure as Code.

Step 1: User Interaction Layer
Users access the application through a web browser using the URL (localhost or cloud IP). The user enters a mobile number, selects an operator, chooses a plan, and proceeds with recharge. The system validates the input and sends HTTP requests to the backend server.

Step 2: Application Layer (Flask Backend)
The backend is developed using Python Flask, which handles all business logic, API routing, recharge processing, and transaction management. The backend provides RESTful APIs such as:
• GET /api/plans/{operator}
• POST /api/recharge
• GET /api/transactions

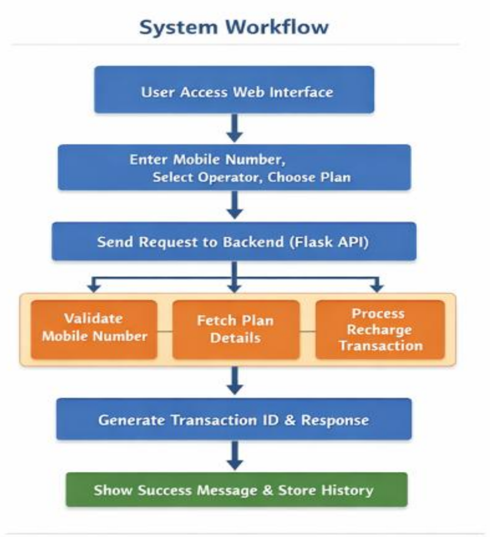
These APIs return JSON responses and process recharge requests in real time.

Step 3: Container Layer (Docker & Docker Compose)
The application is containerized using Docker to ensure consistency across development and production environments. Docker Compose is used to automate multi-container deployment and simplify the execution process using a single command (docker-compose up -d)

Step 4: Infrastructure Layer (IaC & Cloud)
Infrastructure provisioning is automated using Terraform, which can deploy the system locally or on AWS EC2 instances. Kubernetes manifests are also provided for scalable production deployment with multiple replicas and load balancing.

Step 5: Automation & CI/CD
The project includes CI/CD workflows that automatically build Docker images, run tests, and deploy the application, ensuring continuous integration and reliable deployment

Workflow Diagram


System Workflow
User Access Web Interface
Enter Mobile Number, Select Operator, Choose Plan
Send Request to Backend (Flask API)
Validate Mobile Number
Fetch Plan Details
Process Recharge Transaction
Generate Transaction ID & Response
Show Success Message & Store History

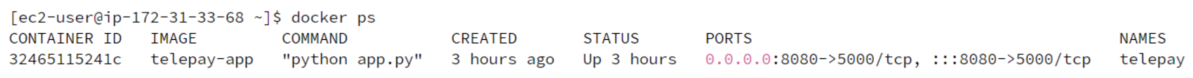
3.3 Tools & Technologies Used

Component:	**Technologies:**
Frontend	HTML5, CSS3, JavaScript
Backend	Python 3.9, Flask REST API
Containerization	Docker, Docker Compose
Orchestration	Kubernetes
Infrastructure as Code	Terraform
Configuration Management	Ansible
CI/CD	GitHub Actions, Jenkins
Cloud Platform	AWS EC2
Version Control	Git & GitHub

4. Results & Output

4.1 Screenshots / Outputs

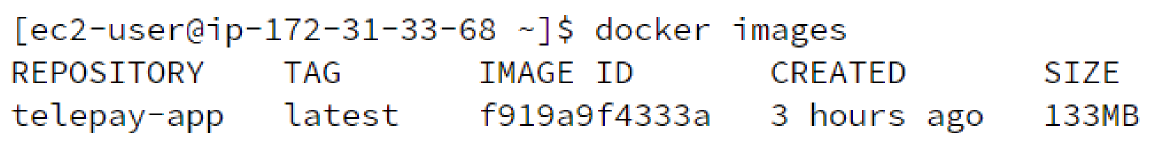
Docker Container Status (docker ps):



```
[ec2-user@ip-172-31-33-68 ~]$ docker ps
CONTAINER ID   IMAGE         COMMAND           CREATED       STATUS      PORTS                                         NAMES
32465115241c   telepay-app   "python app.py"   3 hours ago   Up 3 hours  0.0.0.0:8080->5000/tcp, :::8080->5000/tcp     telepay
```

This above screenshot shows the running Docker container for the TelePay application on the AWS EC2 instance. The container named "telepay" is in an active state (Up 3 hours), confirming that the Flask-based telecom recharge system is successfully deployed and continuously running. The port mapping 8080 → 5000 indicates that the application is accessible externally through port 8080 while internally running on port 5000.

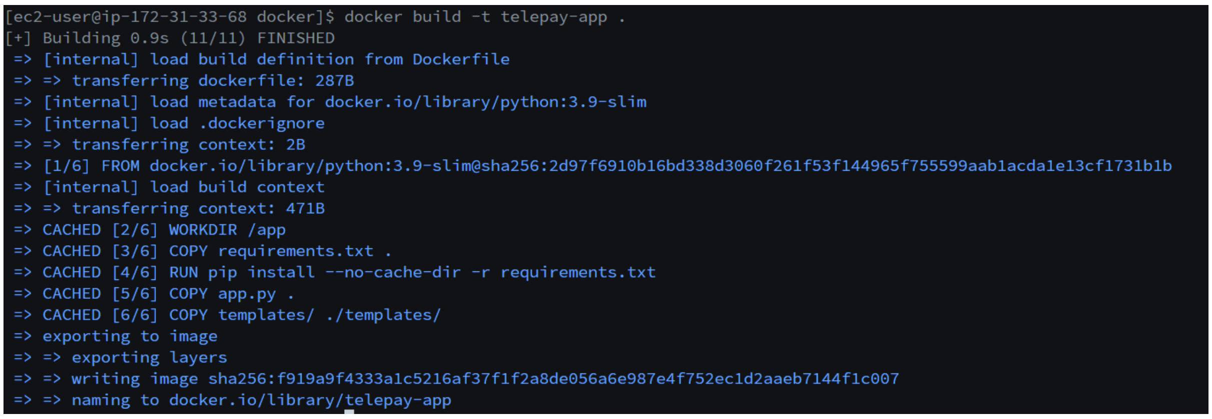
Docker Image Build Process (docker build):



```
[ec2-user@ip-172-31-33-68 ~]$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
telepay-app   latest    f919a9f4333a   3 hours ago   133MB
```

This image displays the successful build process of the Docker image using the Dockerfile. It shows that all layers such as base Python image, requirements installation, and application files (app.py and templates) were processed correctly. The "FINISHED" status confirms that the telepay-app image was created without errors and is ready for deployment.

Docker Images List (docker images):

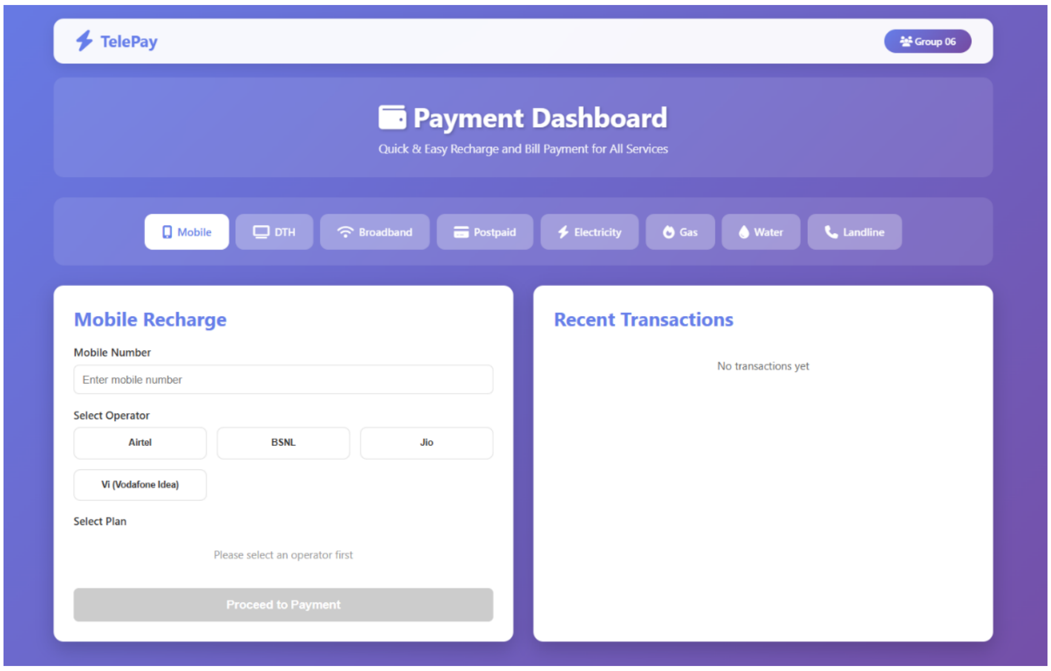


```
[ec2-user@ip-172-31-33-68 docker]$ docker build -t telepay-app .
[+] Building 0.9s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 287B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/6] FROM docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f755599aab1acda1e13cf1731b1b
 => [internal] load build context
 => => transferring context: 471B
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] COPY requirements.txt .
 => CACHED [4/6] RUN pip install --no-cache-dir -r requirements.txt
 => CACHED [5/6] COPY app.py .
 => CACHED [6/6] COPY templates/ ./templates/
 => exporting to image
 => => exporting layers
 => => writing image sha256:f919a9f4333a1c5216af37f1f2a8de056a6e987e4f752ec1d2aaeb7144f1c007
 => => naming to docker.io/library/telepay-app
```

This screenshot presents the list of available Docker images on the system. It confirms that the telepay-app image with the latest tag has been successfully created with a size of 133MB. This verifies that the application image is properly stored in the local Docker repository and ready to be used for container deployment.

4. Application User Interface

Dashboard



4.2 Reports

Application Reports:
The system generates transaction history reports displaying recharge details such as mobile number, operator, plan, transaction ID, and timestamp.

API Performance Reports:
• Successful recharge processing
• Valid operator plan retrieval
• Real-time transaction tracking

System Monitoring Reports:
• Container health status
• Logs and performance monitoring
• Automated test execution results

4.3 Key Outcomes

Technical Achievements:

• Fully automated IaC deployment using Terraform
• Successful Docker containerization
• RESTful API implementation
• Kubernetes-ready scalable architecture
• Automated CI/CD pipeline integration
• Health checks and monitoring enabled

Operational Benefits:

• Automated telecom recharge processing
• Reduced manual deployment errors
• Scalable and cloud-ready infrastructure
• Improved system reliability and performance
• Centralized transaction tracking system

The project status confirms that the application, APIs, UI, and container deployment are fully operational and successfully tested

5. Conclusion

The TelePay – Infrastructure as Code (IaC) Provisioning for Telecom System project successfully demonstrates the integration of DevOps practices, containerization, and cloud infrastructure automation in a real-world telecom application. By implementing Docker, Terraform, Kubernetes, and CI/CD pipelines, the system achieves high scalability, automation, and deployment consistency.

The project enhanced practical knowledge in Infrastructure as Code, cloud deployment, container orchestration, REST API development, and DevOps automation. It provides a production-ready telecom

recharge platform capable of real-time processing, automated deployment, and scalable infrastructure management.

Overall, the project fulfills all the requirements of an IaC-based telecom system and proves the effectiveness of automation in modern cloud-native applications.

6. Future Scope & Enhancements

- Integration of secure payment gateway (UPI/Card)
- User authentication with JWT and role-based access control
- Database integration (PostgreSQL/MongoDB)
- Redis caching for faster performance
- Prometheus and Grafana for advanced monitoring
- Multi-region cloud deployment
- SMS and Email notification system
- Wallet and offer management system
- Auto-scaling using Kubernetes HPA