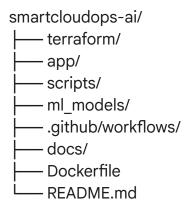
# **Smart Cloud Ops Al: Project Plan**

- PHASE 0 Foundation & Setup
- 0.1 Repo + Branching
- Create GitHub repo smartcloudops-ai
- Add .gitignore, README.md, LICENSE
- V Branches: main, dev, infra/terraform, app/chatops
- 0.2 Folder Structure



### 0.3 Tool Installations

- Terraform CLI
- Docker & Docker Compose
- AWS CLI
- Python 3.10+ + venv
- PHASE 1 Infrastructure Provisioning + Monitoring
- 1.1 Terraform Setup
- 1.1.1 Provider & Remote State
  - Configure main.tf: provider "aws" { region = "us-east-1" }
  - o Optional: Use S3 backend for tfstate
- 1.1.2 VPC + Subnets
  - o VPC: 10.0.0.0/16
  - o Public subnets x2

IGW + route table

### 1.1.3 Security Groups

 Ports: 22 (SSH), 80 (HTTP), 3000 (Grafana), 9090 (Prometheus), 9100 (Node Exporter)

### • 1.1.4 EC2 Instances

o Create:

ec2\_monitoring: Prometheus + Grafanaec2\_application: Flask ChatOps app

## • 1.2 Monitoring Stack

#### • 1.2.1 Prometheus

Install Prometheus on ec2\_monitoring

Configure prometheus.yml:

scrape\_configs:
- job\_name: 'ec2\_node'

static\_configs:

- targets: ['localhost:9100']

## • 1.2.2 Node Exporter

- o Install on all EC2s
- o Run on port 9100

### • 1.2.3 Grafana

- o Install via RPM
- o Access via http://<public-ip>:3000
- o Add Prometheus as data source
- o Create dashboards: CPU, RAM, Disk

### • 1.3 CI/CD Infra

• 1.3.1 GitHub Actions: infra.yml

on: [push] jobs:

terraform:

steps:

- run: terraform fmt

- run: terraform validate

# PHASE 2 – Flask ChatOps App + Dockerization

## 2.1 Flask App Basics

- Create app/main.py
- Endpoints: /query, /status, /logs

## 2.2 GPT Integration

- Use openai or litellm SDK
- Implement prompt template
- Sanitize user input

### 2.3 Dockerization

• Create Dockerfile:

FROM python:3.10

COPY app//app

RUN pip install -r /app/requirements.txt

CMD ["python", "/app/main.py"]

### • 2.4 CI/CD

Add ci-app.yml to auto-build, lint, and push container

# PHASE 3 – Anomaly Detection (ML Layer)

### 3.1 Data Preparation

Use Prometheus metrics → CSV via API or node\_exporter logs

## 3.2 Model Training

- Use Isolation Forest or Prophet
- Save model to ml\_models/anomaly\_model.pkl
- Validation: F1-score ≥ 0.85

## • 3.3 Inference Pipeline

- Load model in script:
- Input: live metrics
- Output: anomaly status + severity

# PHASE 4 – Auto-Remediation Logic

### 4.1 Rule Engine

• Trigger logic:

if cpu\_util > 90 and duration > 3 minutes: trigger\_remediation()

### 4.2 Scripts

- restart\_service.py
- scale\_up.py

### 4.3 Logging

- JSON logs: timestamp, action, instance, result
- Store in /logs/ folder with daily rotation

# PHASE 5 – ChatOps GPT Layer

## 5.1 NLP Queries

- Examples:
  - "What's current CPU?"
  - "Summarize last 3 anomalies"
  - "Show logs from 10 minutes ago"

#### 5.2 Context Window

- Cache last anomalies via redis or in-memory
- Use logs + ML outputs to answer intelligently

## 5.3 GPT Prompting

SYSTEM: You are a DevOps assistant. USER: Summarize recent incidents.

# PHASE 6 – Testing, Security & Documentation

# • 6.1 Unit & Integration Tests

- pytest for app
- Load tests for Flask endpoints

# 6.2 Security

- IAM: least privilege
- Secrets: store in AWS SSM
- Static scan: bandit, trivy

### 6.3 Documentation

- README.md
- docs/architecture.png
- Project Walkthrough (video or markdown)

# PHASE 7 – Production Launch & Feedback

## • 7.1 Final Deployment

- Deploy all modules in live AWS VPC
- Enable alerting via Grafana email/SNS

# 7.2 Beta Testing

- Invite 2–3 users (DevOps engineers)
- Collect feedback in Notion or GitHub Issues

# 7.3 Final Wrap-up

- Deliver:
  - Source Code
  - o Architecture Diagrams
  - o CI/CD pipelines
  - o Demo video
  - o Installation guide

## **V** FINAL OUTPUT SUMMARY

Deliverable	Phase Done
Terraform Infra	Phase 1
Monitoring Stack	Phase 1
Flask ChatOps App	Phase 2
Docker Container	Phase 2
GPT Integration	Phase 2, 5
ML Anomaly Detection	Phase 3
Auto-Healing Scripts	Phase 4
ChatOps Assistant	Phase 5
Security & Hardening	Phase 6
Docs & Demo	Phase 6
Production Rollout	Phase 7