National College of Ireland

Configuration Manual

MSc Research Project
MSc In Cloud Computing

Prasanna Shinde
Student ID: X23278480

School of Computing
National College of Ireland

Supervisor: Sai Emani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Prasanna Shinde

Student ID: X23278480

Programme: MSc in Cloud Computing **Year:** 2024

Module: MSc Research Project

Lecturer: Sai Emani

Submission Due Date: 11th August 2025

Project Title: Self-Healing CI/CD Pipelines: Automating Kubernetes Deployments with Terraform and Ansible.

Word Count:1468 **Page Count:** 20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Prasanna Shinde

Date: 10th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Prasanna Shinde
Student ID: X23278480

1 Introduction

This configuration guide is a point-by-point tutorial on architecting and deploying a self-healing CI/CD pipeline based on infrastructure provisioning by Terraform, a service layer by Flask web application and PostgreSQL, CI/CD orchestration by Jenkins, monitoring by Prometheus and policy enforcement by Open Policy Agent (OPA) Gatekeeper. The custom Python modules used in the system includes failure classification, trust-aware deployment, dynamic policy deployment, deployment health scoring and explainable rollbacks. It addresses failure of deployment, violations of policies, crashes of applications, and automatically recovers based on predefined workflows that are initiated by Ansible and Kubernetes. Each configuration is Infrastructure as Code (IaC) based to provide repeatability, scalability, and compliance of each deployment.

2 System Requirements

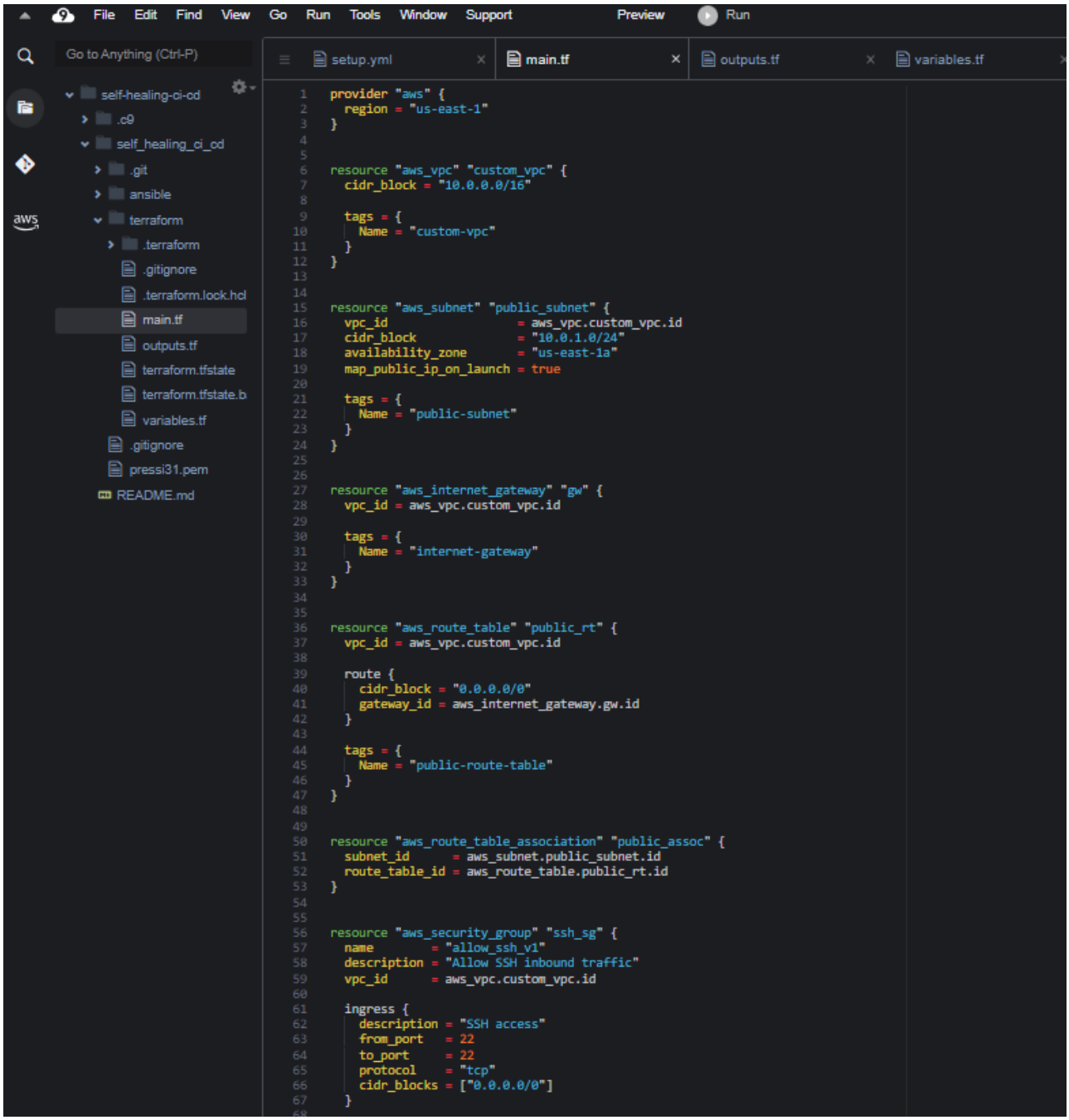
This section provides the minimum hardware, software, and network conditions to be used to install and operate the experimental environment of the Self-Healing CI/CD Pipelines: Automating Kubernetes Deployments with Terraform and Ansible.

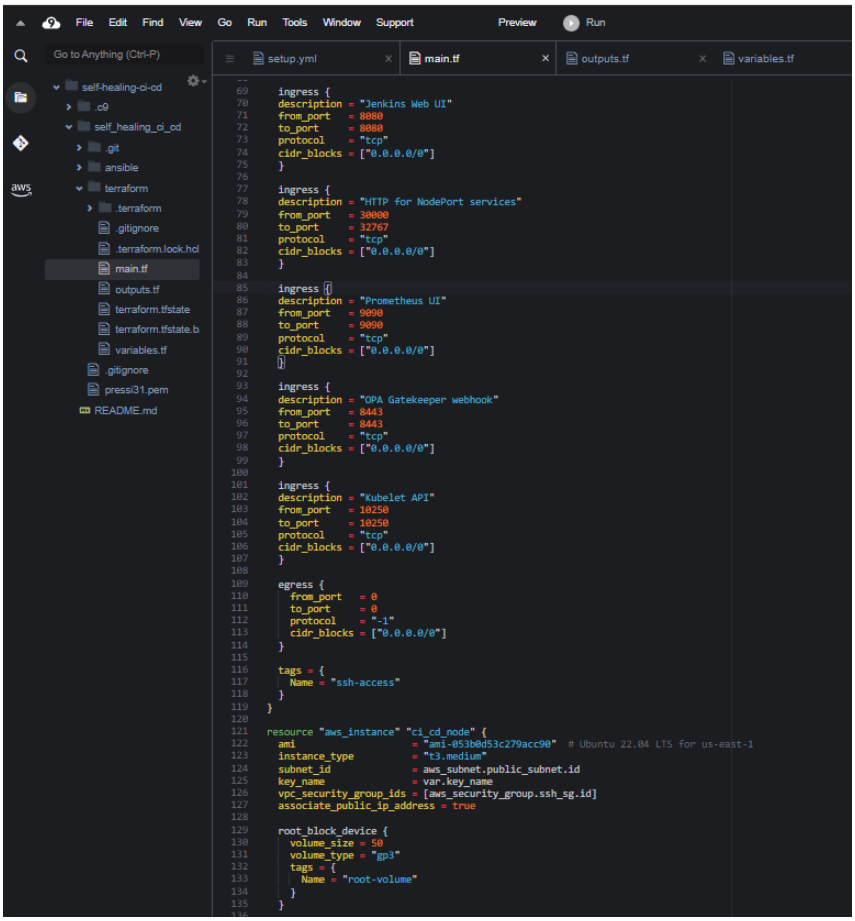
- Operating System: Ubuntu 22.04 LTS
- Terraform: v1.7+
- Ansible: v2.15
- Jenkins: Latest
- Docker: v25+
- cri-dockerd: v0.3.4+
- Kubernetes: v1.30+
- Helm: v3.14+
- Prometheus: v2.53+
- OPA Gatekeeper: v3.16+
- Python: v3.10+
- Cloud Platform: AWS EC2 Instance

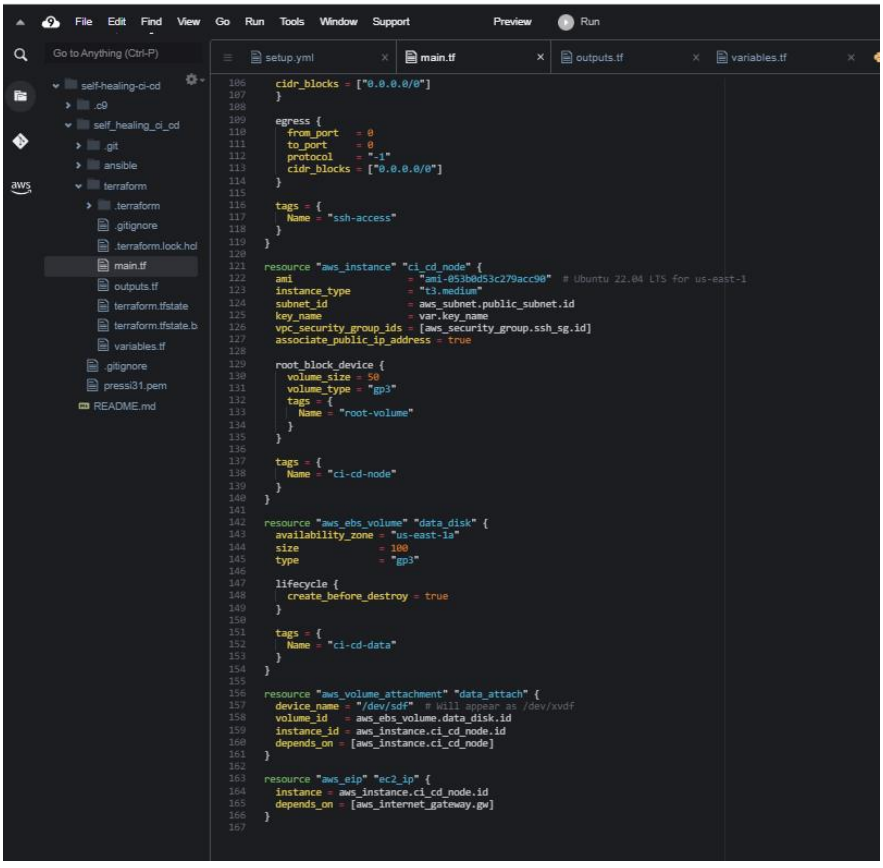
3 Setup Guide

3.1 Terraform

The Terraform configuration specifies details on the cloud infrastructure of the experiment. The main.tf file is used to provision the AWS EC2 instance with all resources needed, including security groups, IAM roles, and networking aspects of self-healing CI/CD pipeline.




File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (Ctrl-P)
setup.yml
main.tf
outputs.tf
variables.tf
self-healing-ci-cd
.c9
self_healing_ci_cd
.git
ansible
terraform
.terraform
.gitignore
.terraform.lock.hcl
main.tf
outputs.tf
terraform.tfstate
terraform.tfstate.b
variables.tf
.gitignore
pressi31.pem
README.md
ingress {
description = "Jenkins Web UI"
from_port = 8080
to_port = 8080
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
description = "HTTP for NodePort services"
from_port = 30000
to_port = 32767
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
description = "Prometheus UI"
from_port = 9090
to_port = 9090
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
description = "OPA Gatekeeper webhook"
from_port = 8443
to_port = 8443
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
description = "Kubelet API"
from_port = 10250
to_port = 10250
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
egress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}
tags = {
Name = "ssh-access"
}
}
resource "aws_instance" "ci_cd_node" {
ami = "ami-053b0d53c279acc90" # Ubuntu 22.04 LTS for us-east-1
instance_type = "t3.medium"
subnet_id = aws_subnet.public_subnet.id
key_name = var.key_name
vpc_security_group_ids = [aws_security_group.ssh_sg.id]
associate_public_ip_address = true
root_block_device {
volume_size = 50
volume_type = "gp3"
tags = {
Name = "root-volume"
}
}

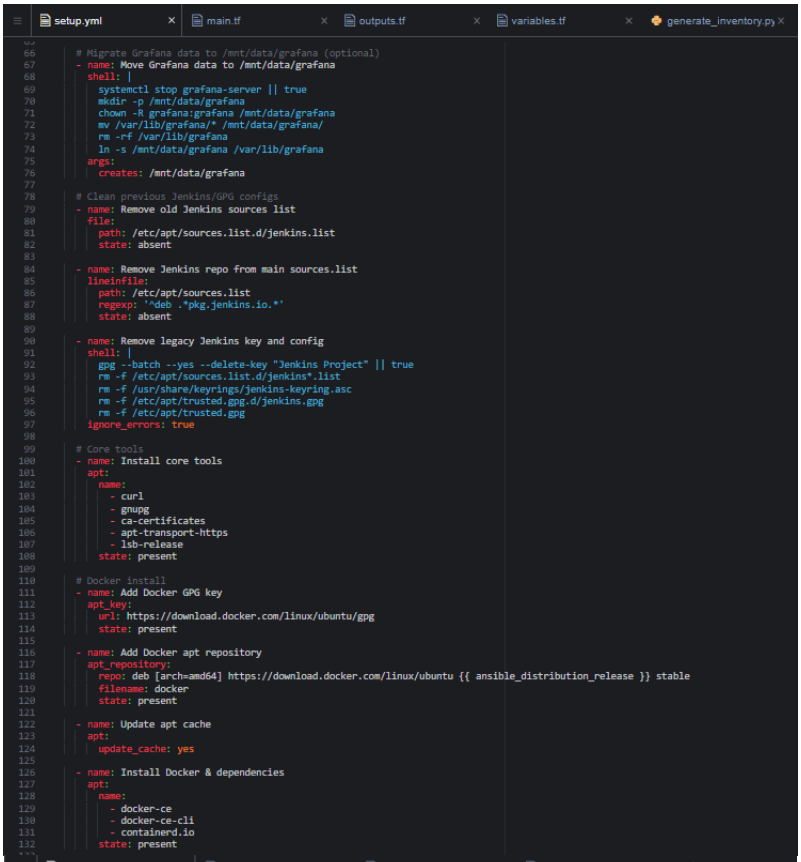

File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (Ctrl-P)
setup.yml
main.tf
outputs.tf
variables.tf
self-healing-ci-cd
.c9
self_healing_ci_cd
.git
ansible
terraform
.terraform
.gitignore
.terraform.lock.hcl
main.tf
outputs.tf
terraform.tfstate
terraform.tfstate.b
variables.tf
.gitignore
pressi31.pem
README.md
cidr_blocks = ["0.0.0.0/0"]
}
egress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}
tags = {
Name = "ssh-access"
}
}
resource "aws_instance" "ci_cd_node" {
ami = "ami-053b0d53c279acc90" # Ubuntu 22.04 LTS for us-east-1
instance_type = "t3.medium"
subnet_id = aws_subnet.public_subnet.id
key_name = var.key_name
vpc_security_group_ids = [aws_security_group.ssh_sg.id]
associate_public_ip_address = true
root_block_device {
volume_size = 50
volume_type = "gp3"
tags = {
Name = "root-volume"
}
}
tags = {
Name = "ci-cd-node"
}
}
resource "aws_ebs_volume" "data_disk" {
availability_zone = "us-east-1a"
size = 100
type = "gp3"
lifecycle {
create_before_destroy = true
}
tags = {
Name = "ci-cd-data"
}
}
resource "aws_volume_attachment" "data_attach" {
device_name = "/dev/sdf" # Will appear as /dev/xvdf
volume_id = aws_ebs_volume.data_disk.id
instance_id = aws_instance.ci_cd_node.id
depends_on = [aws_instance.ci_cd_node]
}
resource "aws_eip" "ec2_ip" {
instance = aws_instance.ci_cd_node.id
depends_on = [aws_internet_gateway.gw]
}

These commands are used to initialize the infrastructure (Terraform)

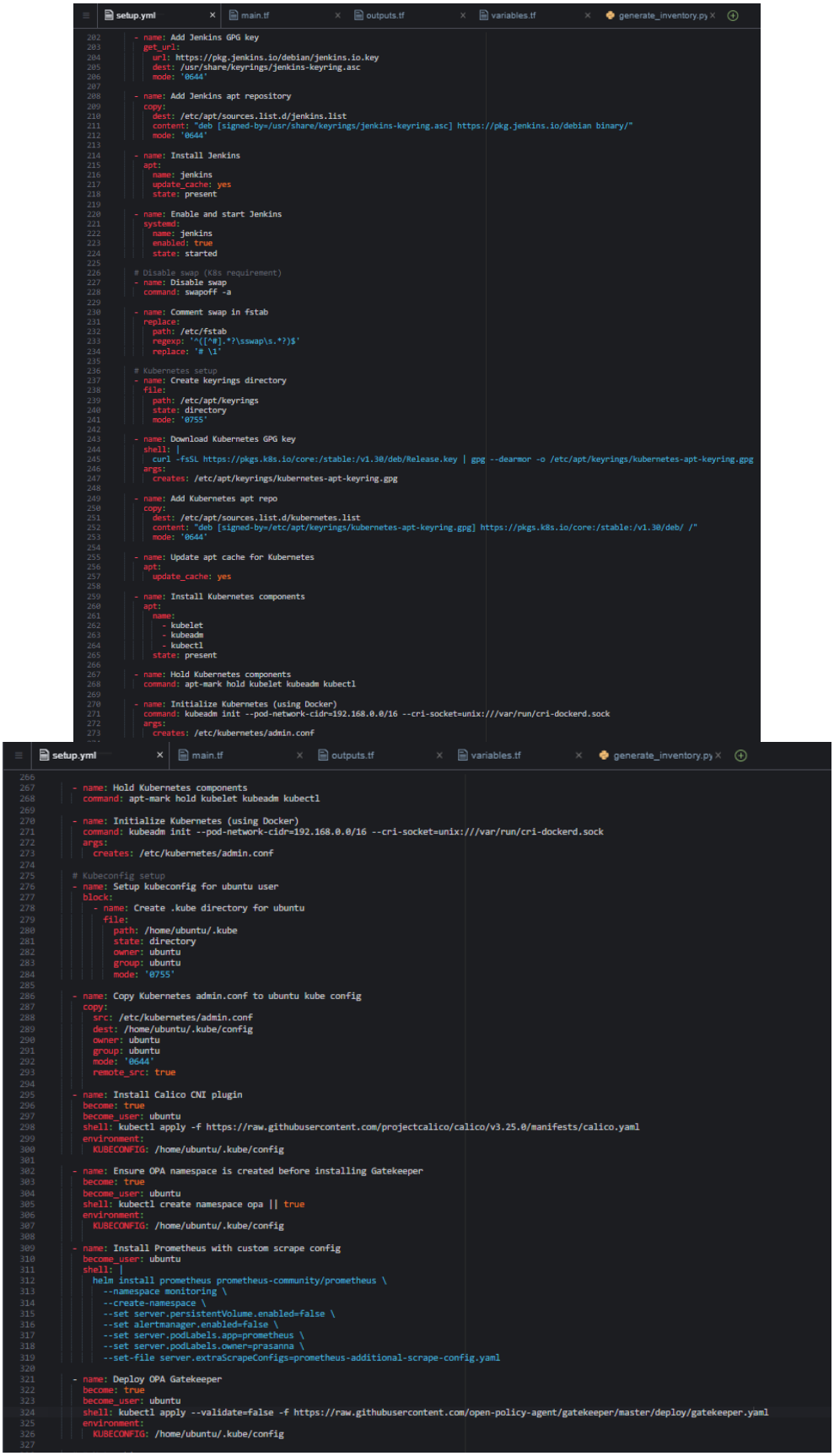
- terraform init
- terraform apply -var="key_name=pressi31" -auto-approve

3.2 Ansible

Ansible script is use for installing the require tools in the system such as Docker, Kubernetes, OPA, Prometheus, Helm and Jenkins


setup.yml
main.tf
outputs.tf
variables.tf
generate_inventory.py
Migrate Grafana data to /mnt/data/grafana (optional)
- name: Move Grafana data to /mnt/data/grafana
shell: |
systemctl stop grafana-server || true
mkdir -p /mnt/data/grafana
chown -R grafana:grafana /mnt/data/grafana
mv /var/lib/grafana/* /mnt/data/grafana/
rm -rf /var/lib/grafana
ln -s /mnt/data/grafana /var/lib/grafana
args:
creates: /mnt/data/grafana
Clean previous Jenkins/GPG configs
- name: Remove old Jenkins sources list
file:
path: /etc/apt/sources.list.d/jenkins.list
state: absent
- name: Remove Jenkins repo from main sources.list
lineinfile:
path: /etc/apt/sources.list
regexp: '^deb .*pkg.jenkins.io.*'
state: absent
- name: Remove legacy Jenkins key and config
shell: |
gpg --batch --yes --delete-key "Jenkins Project" || true
rm -f /etc/apt/sources.list.d/jenkins*.list
rm -f /usr/share/keyrings/jenkins-keyring.asc
rm -f /etc/apt/trusted.gpg.d/jenkins.gpg
rm -f /etc/apt/trusted.gpg
ignore_errors: true
Core tools
- name: Install core tools
apt:
name:
- curl
- gnupg
- ca-certificates
- apt-transport-https
- lsb-release
state: present
Docker install
- name: Add Docker GPG key
apt_key:
url: https://download.docker.com/linux/ubuntu/gpg
state: present
- name: Add Docker apt repository
apt_repository:
repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu {{ ansible_distribution_release }} stable
filename: docker
state: present
- name: Update apt cache
apt:
update_cache: yes
- name: Install Docker & dependencies
apt:
name:
- docker-ce
- docker-ce-cli
- containerd.io
state: present


setup.yml
main.tf
outputs.tf
variables.tf
- name: Enable and start Docker
systemd:
name: docker
enabled: true
state: started
- name: Add ubuntu user to docker group
user:
name: ubuntu
groups: docker
append: yes
cri-dockerd setup
- name: Install cri-dockerd dependencies
apt:
name:
- git
- golang
- make
state: present
- name: Clone cri-dockerd (stable v0.3.1)
git:
repo: https://github.com/Mirantis/cri-dockerd.git
dest: /opt/cri-dockerd
version: v0.3.1
- name: Download and install Go 1.20.13
shell: |
wget https://go.dev/dl/go1.20.13.linux-amd64.tar.gz -O /tmp/go.tar.gz
rm -rf /usr/local/go && tar -C /usr/local -xzf /tmp/go.tar.gz
echo "export PATH=/usr/local/go/bin:$PATH" >> /etc/profile
args:
creates: /usr/local/go/bin/go
- name: Build cri-dockerd with Go 1.23+
shell: |
export PATH=$PATH:/usr/local/go/bin
cd /opt/cri-dockerd
mkdir -p bin
go build -o bin/cri-dockerd
- name: Move cri-dockerd binary
copy:
src: /opt/cri-dockerd/bin/cri-dockerd
dest: /usr/local/bin/cri-dockerd
mode: '0755'
remote_src: true
- name: Configure cri-dockerd systemd service
shell: |
cd /opt/cri-dockerd
cp packaging/systemd/* /etc/systemd/system/
sed -i 's:/usr/bin/cri-dockerd:/usr/local/bin/cri-dockerd:' /etc/systemd/system/cri-docker.service
- name: Enable and start cri-dockerd service
systemd:
name: cri-docker.service
daemon_reload: yes
enabled: yes
state: started
Jenkins install
- name: Install Java 17 (Jenkins requirement)
apt:
name: openjdk-17-jdk
state: present



```
setup.yml | main.tf | outputs.tf | variables.tf | generate_inventory.py

    - name: Add Jenkins GPG key
      get_url:
        url: https://pkg.jenkins.io/debian/jenkins.io.key
        dest: /usr/share/keyrings/jenkins-keyring.asc
        mode: '0644'

    - name: Add Jenkins apt repository
      copy:
        dest: /etc/apt/sources.list.d/jenkins.list
        content: "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian binary/"
        mode: '0644'

    - name: Install Jenkins
      apt:
        name: jenkins
        update_cache: yes
        state: present

    - name: Enable and start Jenkins
      systemd:
        name: jenkins
        enabled: true
        state: started

    # Disable swap (K8s requirement)
    - name: Disable swap
      command: swapoff -a

    - name: Comment swap in fstab
      replace:
        path: /etc/fstab
        regexp: '^([^#].*?\sswap\s.*?)$'
        replace: '# \1'

    # Kubernetes setup
    - name: Create keyrings directory
      file:
        path: /etc/apt/keyrings
        state: directory
        mode: '0755'

    - name: Download Kubernetes GPG key
      shell: |
        curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
      args:
        creates: /etc/apt/keyrings/kubernetes-apt-keyring.gpg

    - name: Add Kubernetes apt repo
      copy:
        dest: /etc/apt/sources.list.d/kubernetes.list
        content: "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /"
        mode: '0644'

    - name: Update apt cache for Kubernetes
      apt:
        update_cache: yes

    - name: Install Kubernetes components
      apt:
        name:
          - kubelet
          - kubeadm
          - kubectl
        state: present

    - name: Hold Kubernetes components
      command: apt-mark hold kubelet kubeadm kubectl

    - name: Initialize Kubernetes (using Docker)
      command: kubeadm init --pod-network-cidr=192.168.0.0/16 --cri-socket=unix:///var/run/cri-dockerd.sock
      args:
        creates: /etc/kubernetes/admin.conf
```


```
setup.yml | main.tf | outputs.tf | variables.tf | generate_inventory.py

    - name: Hold Kubernetes components
      command: apt-mark hold kubelet kubeadm kubectl

    - name: Initialize Kubernetes (using Docker)
      command: kubeadm init --pod-network-cidr=192.168.0.0/16 --cri-socket=unix:///var/run/cri-dockerd.sock
      args:
        creates: /etc/kubernetes/admin.conf
```


```
    # Kubeconfig setup
    - name: Setup kubeconfig for ubuntu user
      block:
        - name: Create .kube directory for ubuntu
          file:
            path: /home/ubuntu/.kube
            state: directory
            owner: ubuntu
            group: ubuntu
            mode: '0755'

    - name: Copy Kubernetes admin.conf to ubuntu kube config
      copy:
        src: /etc/kubernetes/admin.conf
        dest: /home/ubuntu/.kube/config
        owner: ubuntu
        group: ubuntu
        mode: '0644'
        remote_src: true

    - name: Install Calico CNI plugin
      become: true
      become_user: ubuntu
      shell: kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
      environment:
        KUBECONFIG: /home/ubuntu/.kube/config

    - name: Ensure OPA namespace is created before installing Gatekeeper
      become: true
      become_user: ubuntu
      shell: kubectl create namespace opa || true
      environment:
        KUBECONFIG: /home/ubuntu/.kube/config

    - name: Install Prometheus with custom scrape config
      become_user: ubuntu
      shell: |
        helm install prometheus prometheus-community/prometheus \
          --namespace monitoring \
          --create-namespace \
          --set server.persistentVolume.enabled=false \
          --set alertmanager.enabled=false \
          --set server.podLabels.app=prometheus \
          --set server.podLabels.owner=prasanna \
          --set-file server.extraScrapeConfigs=prometheus-additional-scrape-config.yaml

    - name: Deploy OPA Gatekeeper
      become: true
      become_user: ubuntu
      shell: kubectl apply --validate=false -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
      environment:
        KUBECONFIG: /home/ubuntu/.kube/config
```

To run the Ansible script

- ansible-playbook -i inventory/hosts.ini setup.yml

3.3 Kubernetes

Kubernetes and its components including kubeadm and kubectl is configured using Ansible on the AWS EC2 instance to orchestrate containerized applications with a self-healing CI/CD pipeline. After running and configuring the EC2 instance, log in to it with your local system using Bash terminal or one of the tools like PuTTY. Upon successful installation, the Kubernetes cluster could be accessed using kubectl command at the terminal to manage our deployments, service and other Kubernetes cluster resources.

```
ubuntu@ip-10-0-1-62:~$ kubectl get pods
NAME                                    READY   STATUS             RESTARTS         AGE
crashpod                                0/1     CrashLoopBackOff   88 (47s ago)     23h
failpod                                 0/1     CrashLoopBackOff   86 (3m8s ago)    23h
failure-classification-29244765-nlwd6   0/1     Completed          0                6m16s
failure-classification-29244770-wjr8n   0/1     Completed          0                76s
failure-test-l69z5                      0/1     Completed          0                8m21s
flask-app-67c44584cb-8rskh              1/1     Running            0                58m
flask-app-67c44584cb-ff955              1/1     Running            0                58m
flask-app-6b4d8f789b-2szsp              0/1     ImagePullBackOff   0                52m
health-scoring-29244765-kz4lh           0/1     Completed          0                6m16s
health-scoring-29244768-4gfps           0/1     Completed          0                3m16s
health-scoring-29244771-h7vgb           0/1     Completed          0                16s
postgres-59c4578695-hbm8z               0/1     Running            2 (3h4m ago)     23h
postgres-954f854b9-ntvfk                0/1     Running            2 (3h4m ago)     23h
trust-score-check-29244760-ql8zh        0/1     Completed          0                11m
trust-score-check-29244770-gl2cb        0/1     Completed          0                76s
trust-test-68h4v                        0/1     Completed          0                8m55s
ubuntu@ip-10-0-1-62:~$
```

- kubectl get pods
- kubectl apply -f <yaml>

3.4 Jenkins

Jenkins server is deployed on an AWS EC2 instance. After instance is launched and configured completely connect with your local system via Bash terminal or any other tool like PuTTY. Installation of Jenkins in done by Ansible and initiate the service. The web console of Jenkins can then be visited in through the browser using the Public IPv4 of the EC2 instance and appended with port 8080

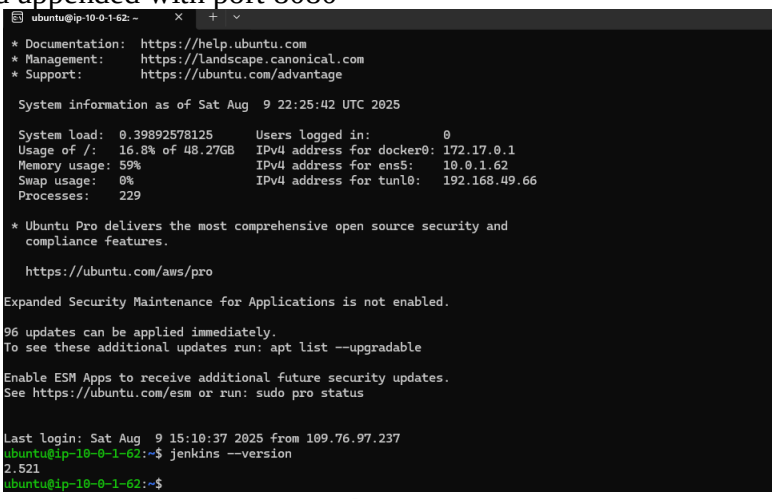


Fig1: Jenkins on EC2

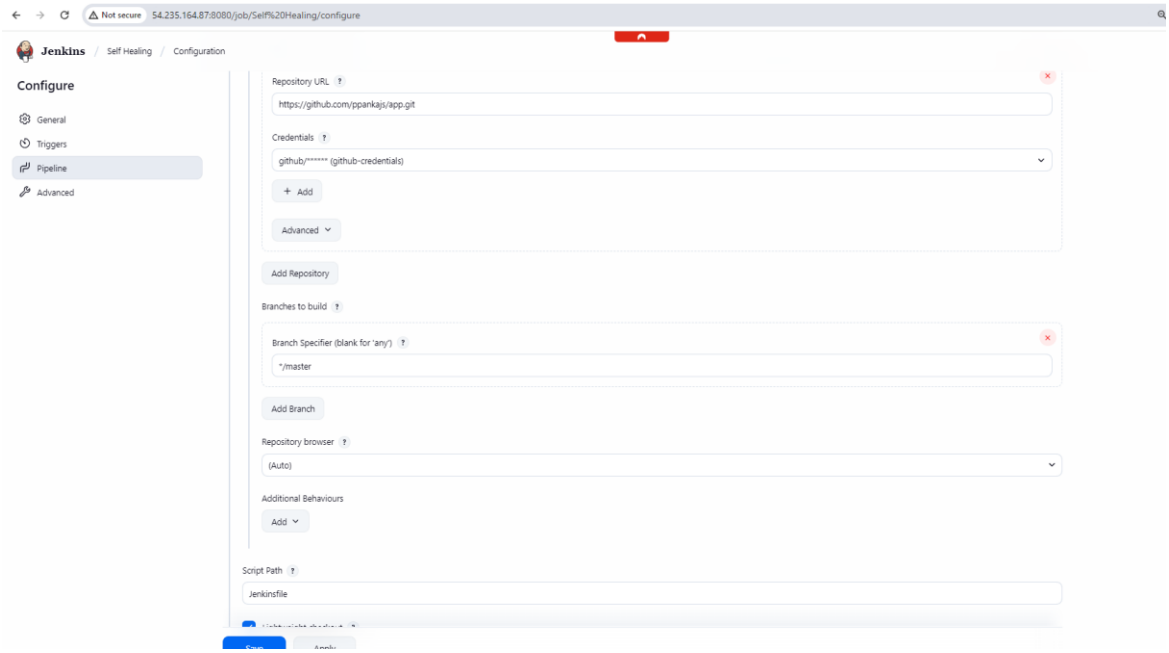
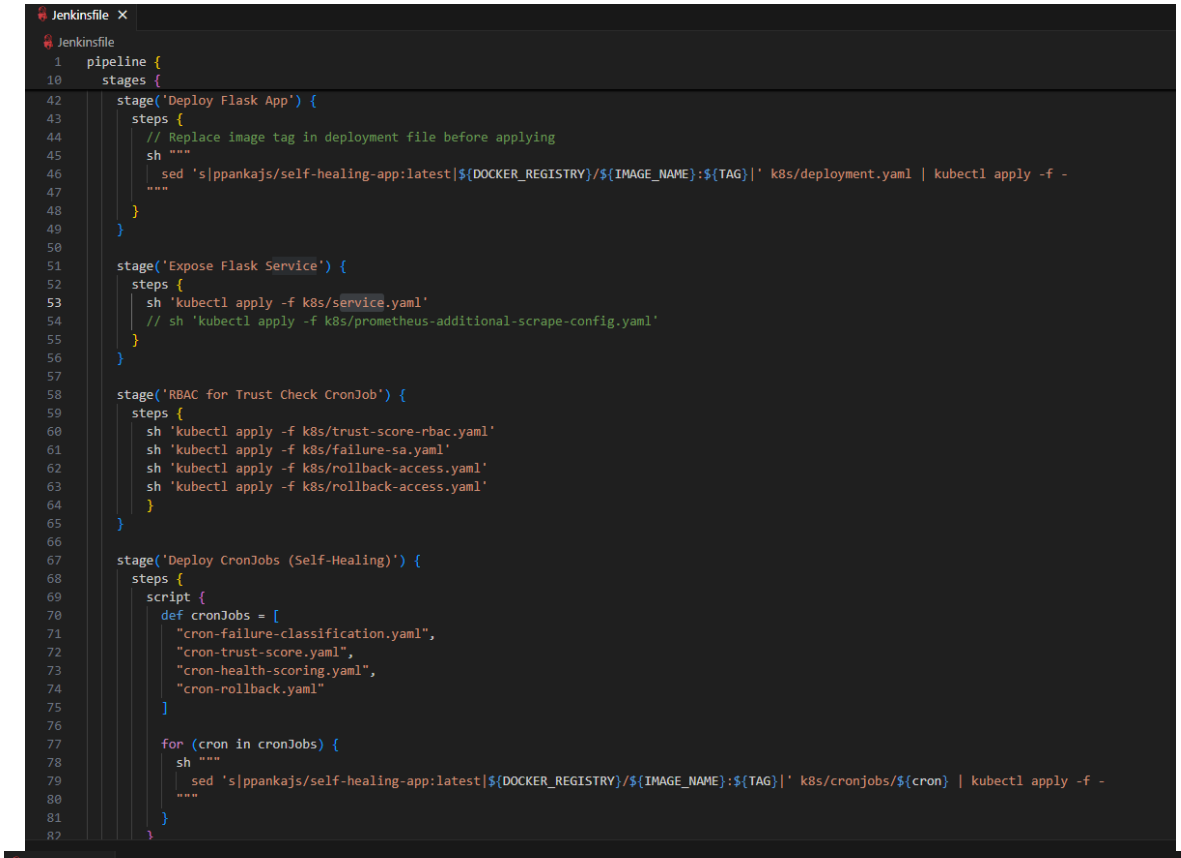


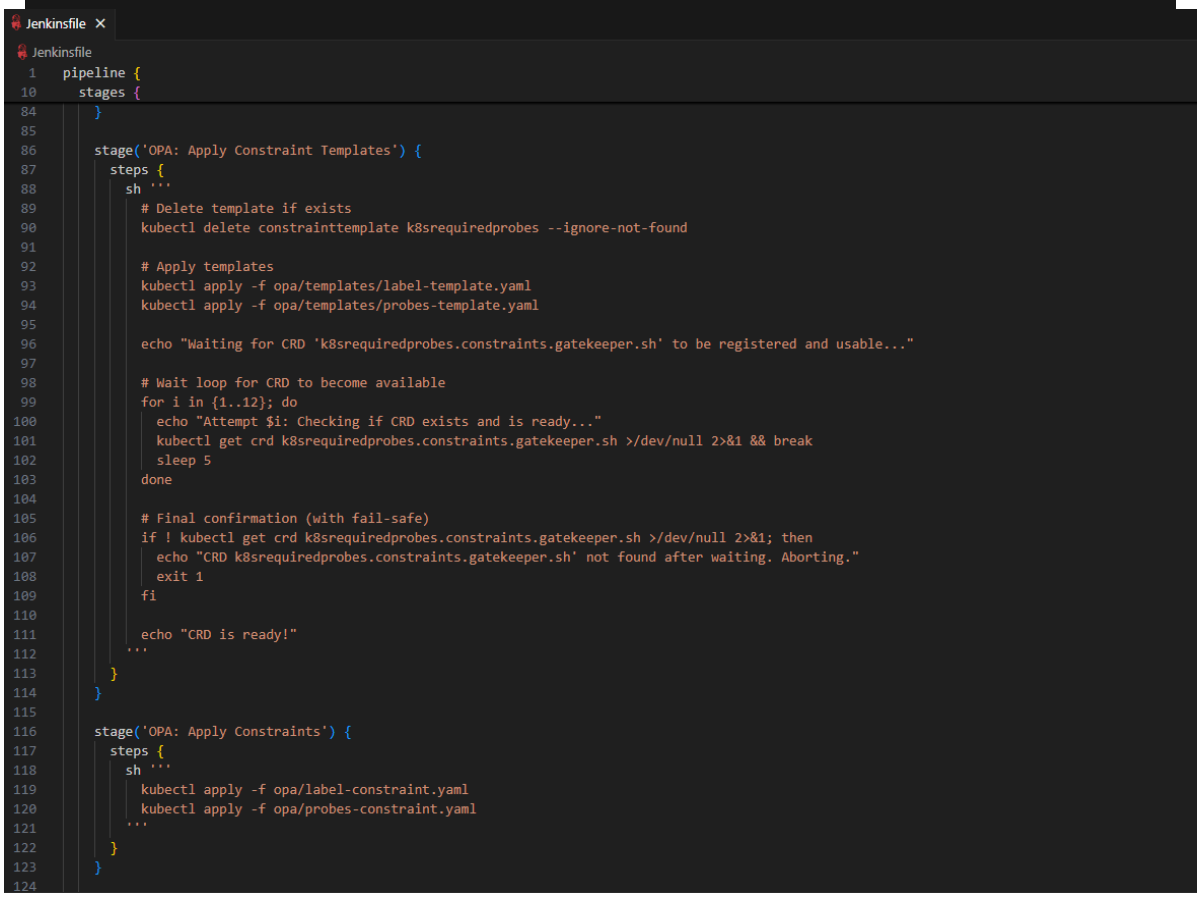
Fig2: Jenkins Pipeline Configuration

Jenkins File

```groovy
pipeline {
  agent any

  environment {
    IMAGE_NAME = "self-healing-app"
    DOCKER_REGISTRY = "ppankajs"
    TAG = "v${BUILD_NUMBER}"
  }

  stages {

    stage('Clone Repo') {
      steps {
        git credentialsId: 'github-credentials', url: 'https://github.com/ppankajs/app.git'
      }
    }

    stage('Build Docker Image') {
      steps {
        script {
          docker.build("${DOCKER_REGISTRY}/${IMAGE_NAME}:${TAG}")
        }
      }
    }

    stage('Push Docker Image') {
      steps {
        script {
          docker.withRegistry('', 'docker-credentials') {
            docker.image("${DOCKER_REGISTRY}/${IMAGE_NAME}:${TAG}").push()
          }
        }
      }
    }

    stage('Deploy PostgreSQL') {
      steps {
        sh 'kubectl apply -f k8s/postgres.yaml'
      }
    }
```



```
Jenkinsfile
pipeline {
  stages {
    stage('Deploy Flask App') {
      steps {
        // Replace image tag in deployment file before applying
        sh """
          sed 's|ppankajs/self-healing-app:latest|${DOCKER_REGISTRY}/${IMAGE_NAME}:${TAG}|' k8s/deployment.yaml | kubectl apply -f -
        """
      }
    }

    stage('Expose Flask Service') {
      steps {
        sh 'kubectl apply -f k8s/service.yaml'
        // sh 'kubectl apply -f k8s/prometheus-additional-scrape-config.yaml'
      }
    }

    stage('RBAC for Trust Check CronJob') {
      steps {
        sh 'kubectl apply -f k8s/trust-score-rbac.yaml'
        sh 'kubectl apply -f k8s/failure-sa.yaml'
        sh 'kubectl apply -f k8s/rollback-access.yaml'
        sh 'kubectl apply -f k8s/rollback-access.yaml'
        }
    }

    stage('Deploy CronJobs (Self-Healing)') {
      steps {
        script {
          def cronJobs = [
            "cron-failure-classification.yaml",
            "cron-trust-score.yaml",
            "cron-health-scoring.yaml",
            "cron-rollback.yaml"
          ]

          for (cron in cronJobs) {
            sh """
              sed 's|ppankajs/self-healing-app:latest|${DOCKER_REGISTRY}/${IMAGE_NAME}:${TAG}|' k8s/cronjobs/${cron} | kubectl apply -f -
            """
          }
        }
```



```
Jenkinsfile
pipeline {
  stages {
    }

    stage('OPA: Apply Constraint Templates') {
      steps {
        sh '''
          # Delete template if exists
          kubectl delete constrainttemplate k8srequiredprobes --ignore-not-found

          # Apply templates
          kubectl apply -f opa/templates/label-template.yaml
          kubectl apply -f opa/templates/probes-template.yaml

          echo "Waiting for CRD 'k8srequiredprobes.constraints.gatekeeper.sh' to be registered and usable..."

          # Wait loop for CRD to become available
          for i in {1..12}; do
            echo "Attempt $i: Checking if CRD exists and is ready..."
            kubectl get crd k8srequiredprobes.constraints.gatekeeper.sh >/dev/null 2>&1 && break
            sleep 5
          done

          # Final confirmation (with fail-safe)
          if ! kubectl get crd k8srequiredprobes.constraints.gatekeeper.sh >/dev/null 2>&1; then
            echo "CRD k8srequiredprobes.constraints.gatekeeper.sh' not found after waiting. Aborting."
            exit 1
          fi

          echo "CRD is ready!"
        '''
      }
    }

    stage('OPA: Apply Constraints') {
      steps {
        sh '''
          kubectl apply -f opa/label-constraint.yaml
          kubectl apply -f opa/probes-constraint.yaml
        '''
      }
    }
```

This is the JenkinsFile where my pipeline is written from fetching the code from github, building docker image, deploying my application as well as my cron jobs on the kubernetes.

3.5 Python Application

The Python Flask app is dockerized, and linked to a Postgres database. Flask app has health check, add users and list user endpoints. Kubernetes manifests will be generated over the application and database including Deployments, Services, ConfigMaps and Secrets. PostgreSQL accesses its data storage via PersistentVolumeClaim, both components are in the same cluster. After being deployed they can be accessed through the Kubernetes service, and all database queries are now handled by the internal PostgreSQL service.

app.py

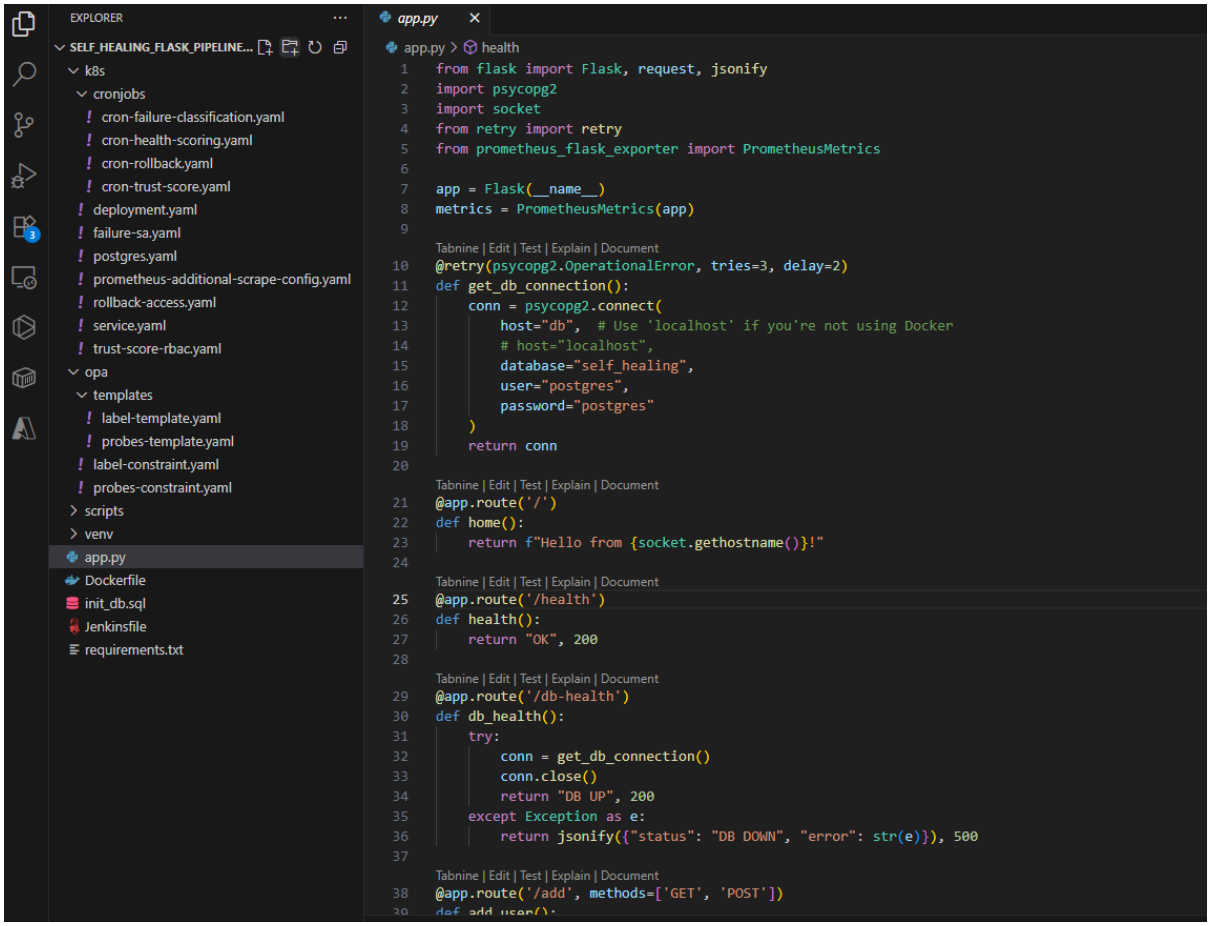
app.py > health

```python
from flask import Flask, request, jsonify
import psycopg2
import socket
from retry import retry
from prometheus_flask_exporter import PrometheusMetrics

app = Flask(__name__)
metrics = PrometheusMetrics(app)

Tabnine | Edit | Test | Explain | Document
@retry(psycopg2.OperationalError, tries=3, delay=2)
def get_db_connection():
    conn = psycopg2.connect(
        host="db",  # Use 'localhost' if you're not using Docker
        # host="localhost",
        database="self_healing",
        user="postgres",
        password="postgres"
    )
    return conn

Tabnine | Edit | Test | Explain | Document
@app.route('/')
def home():
    return f"Hello from {socket.gethostname()}!"

Tabnine | Edit | Test | Explain | Document
@app.route('/health')
def health():
    return "OK", 200

Tabnine | Edit | Test | Explain | Document
@app.route('/db-health')
def db_health():
    try:
        conn = get_db_connection()
        conn.close()
        return "DB UP", 200
    except Exception as e:
        return jsonify({"status": "DB DOWN", "error": str(e)}), 500

Tabnine | Edit | Test | Explain | Document
@app.route('/add', methods=['GET', 'POST'])
```

app.py

app.py > health

```python
from flask import Flask, request, jsonify
import psycopg2
import socket
from retry import retry
from prometheus_flask_exporter import PrometheusMetrics

app = Flask(__name__)
metrics = PrometheusMetrics(app)

Tabnine | Edit | Test | Explain | Document
@retry(psycopg2.OperationalError, tries=3, delay=2)
def get_db_connection():
    conn = psycopg2.connect(
        host="db",  # Use 'localhost' if you're not using Docker
        # host="localhost",
        database="self_healing",
        user="postgres",
        password="postgres"
    )
    return conn

Tabnine | Edit | Test | Explain | Document
@app.route('/')
def home():
    return f"Hello from {socket.gethostname()}!"

Tabnine | Edit | Test | Explain | Document
@app.route('/health')
def health():
    return "OK", 200

Tabnine | Edit | Test | Explain | Document
@app.route('/db-health')
def db_health():
    try:
        conn = get_db_connection()
        conn.close()
        return "DB UP", 200
    except Exception as e:
        return jsonify({"status": "DB DOWN", "error": str(e)}), 500

Tabnine | Edit | Test | Explain | Document
@app.route('/add', methods=['GET', 'POST'])
```



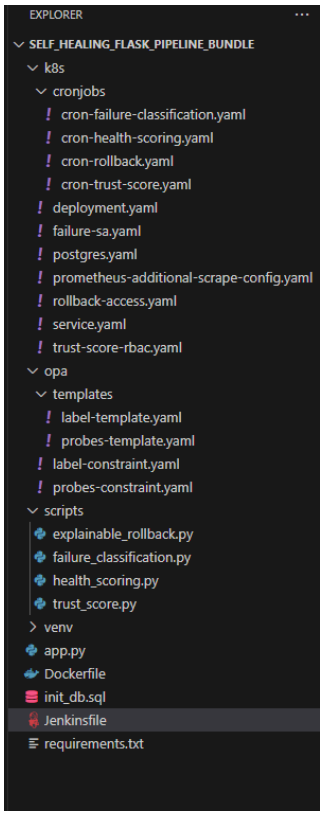


Fig4: Application Structure

3.6 Docker

Docker was deployed on the AWS EC2 instance via Ansible and Kubernetes was customized to manage containerized applications into the self-healing CI/CD process.

```
ubuntu@ip-10-0-1-62:~$ docker --version
Docker version 28.3.3, build 980b856
ubuntu@ip-10-0-1-62:~$
```

3.7 Scripts

Five Python scripts are Failure Classification, Trust-Score, Dynamic Policy Enforcement, Health Scoring, and Explainable Rollbacks that run in Kubernetes to detect issues, enforce policies, and trigger automated recovery. Each scripts have its own cron job.

explainable_rollback.py

```python
import json
import os
import subprocess

SCORE_THRESHOLD = 70
DATA_DIR = "/data"
ROLLBACK_TO_TAG = "v96"

def read_score(file_name):
    try:
        with open(os.path.join(DATA_DIR, file_name), "r") as f:
            data = json.load(f)
            return list(data.values())[0]
    except Exception as e:
        print(f"[ERROR] Failed to read {file_name}: {e}")
        return 0

def get_latest_scores():
    scores = {
        "failure": read_score("failure_score.json"),
        "trust": read_score("trust_score.json"),
        "health": read_score("health_score.json"),
    }
    print(f"[SCORES] {scores}")
    return scores

def trigger_rollback():
    print(f"[ROLLBACK] Deployment score too low. Rolling back to {ROLLBACK_TO_TAG}")
    subprocess.run([
        "kubectl", "set", "image", "deployment/flask-app",
        f"flask-container=ppankajs/self-healing-app:{ROLLBACK_TO_TAG}"
    ])

def main():
    scores = get_latest_scores()
    avg_score = sum(scores.values()) // len(scores)
    print(f"[EVALUATION] Deployment Score: {avg_score}")
```

failure_classification.py

```python
import subprocess
import json
import os

output_file = "/data/failure_score.json"

def classify_crashloopbackoff():
    print("[INFO] Starting Failure Classification...")
    result = subprocess.run(
        ["kubectl", "get", "pods", "-A", "-o", "json"],
        capture_output=True, text=True
    )

    found_crash = False
    try:
        pods = json.loads(result.stdout)
        for pod in pods["items"]:
            statuses = pod.get("status", {}).get("containerStatuses", [])
            for container in statuses:
                waiting = container.get("state", {}).get("waiting", {})
                # if waiting.get("reason") == "CrashLoopBackOff":
                if waiting.get("reason") in ["CrashLoopBackOff", "ImagePullBackOff"]:
                    print(f"[FAILURE] CrashLoopBackOff in {pod['metadata']['name']}")
                    found_crash = True
    except json.JSONDecodeError:
        print("[ERROR] Failed to parse kubectl output.")

    score = 30 if found_crash else 100
    print(f"[FAILURE SCORE] {score}")
    with open(output_file, "w") as f:
        json.dump({"failure": score}, f)

if __name__ == "__main__":
    classify_crashloopbackoff()
```

scripts > trust_score.py > evaluate_trust

```python
import subprocess
import json
import sys
import os

TRUSTED_TAGS = ["v78", "v77", "v76", "v89"]
output_file = "/data/trust_score.json"

Tabnine | Edit | Test | Explain | Document
def evaluate_trust():
    print("[INFO] Checking image trust score...")
    result = subprocess.run(
        ["kubectl", "get", "deployment", "flask-app", "-o", "json"],
        capture_output=True, text=True
    )

    try:
        deployment = json.loads(result.stdout)
        image = deployment['spec']['template']['spec']['containers'][0]['image']
        tag = image.split(":")[-1]
        trust_score = 100 if tag in TRUSTED_TAGS else 40
        print(f"[IMAGE] {image}")
        print(f"[TRUST SCORE] {trust_score}")
    except Exception as e:
        print(f"[ERROR] Failed to evaluate trust: {e}")
        trust_score = 40

    with open(output_file, "w") as f:
        json.dump({"trust": trust_score}, f)

if __name__ == "__main__":
    evaluate_trust()

```

explainable_rollback.py | failure_classification.py | health_scoring.py | trust_score.py

scripts > health_scoring.py > ...

```python
import random
import json
import os

output_file = "/data/health_score.json"

Tabnine | Edit | Test | Explain | Document
def get_mock_metrics():
    cpu = random.randint(20, 95)
    memory = random.randint(30, 95)
    return cpu, memory

Tabnine | Edit | Test | Explain | Document
def calculate_health_score(cpu, memory):
    return 100 - ((cpu + memory) // 2)

if __name__ == "__main__":
    print("[INFO] Evaluating resource health score...")
    cpu, mem = get_mock_metrics()
    score = calculate_health_score(cpu, mem)
    print(f"[HEALTH SCORE] {score}/100")
    with open(output_file, "w") as f:
        json.dump({"health": score}, f)

```

3.8 Cron Jobs

All the Python self-healing scripts will be scheduled as individual Kubernetes jobs running on a certain time frequency in order to measure some metrics, apply policies, and take recovery actions as needed.

cron-failure-classification.yaml | cron-health-scoring.yaml | cron-rollback.yaml | cron-trust-scor

k8s > cronjobs > cron-failure-classification.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: failure-classification
  labels:
    app: self-healing
    owner: prasanna
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: self-healing
            owner: prasanna
        spec:
          serviceAccountName: failure-classification-sa
          containers:
          - name: failure-check
            image: ppankajs/self-healing-app:latest
            command: ["python", "/app/scripts/failure_classification.py"]
            volumeMounts:
              - name: shared-data
                mountPath: /data
          restartPolicy: OnFailure
          volumes:
            - name: shared-data
              persistentVolumeClaim:
                claimName: score-storage

```

cron-failure-classification.yaml | cron-health-scoring.yaml | cron-rollback.yaml | cron-trust-scor

k8s > cronjobs > cron-rollback.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: explainable-rollback
  labels:
    app: self-healing
    owner: prasanna
spec:
  schedule: "*/15 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: self-healing
            owner: prasanna
        spec:
          serviceAccountName: rollback-sa
          containers:
          - name: rollback
            image: ppankajs/self-healing-app:latest
            command: ["python", "/app/scripts/explainable_rollback.py"]
            volumeMounts:
              - name: shared-data
                mountPath: /data
          restartPolicy: OnFailure
          volumes:
            - name: shared-data
              persistentVolumeClaim:
                claimName: score-storage

```

cron-failure-classification.yaml | cron-health-scoring.yaml × | cron-rollback.yaml | cron-trust-

k8s > cronjobs > cron-health-scoring.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: health-scoring
  labels:
    app: self-healing
    owner: prasanna
spec:
  schedule: "*/3 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: self-healing
            owner: prasanna
        spec:
          containers:
            - name: health-score
              image: ppankajs/self-healing-app:latest
              command: ["python", "/app/scripts/health_scoring.py"]
              volumeMounts:
                - name: shared-data
                  mountPath: /data
          restartPolicy: OnFailure
          volumes:
            - name: shared-data
              persistentVolumeClaim:
                claimName: score-storage

```

cron-failure-classification.yaml | cron-health-scoring.yaml | cron-rollback.yaml

k8s > cronjobs > cron-trust-score.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: trust-score-check
  labels:
    app: self-healing
    owner: prasanna
spec:
  schedule: "*/10 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: self-healing
            owner: prasanna
        spec:
          serviceAccountName: trust-check-sa
          containers:
          - name: trust-score
            image: ppankajs/self-healing-app:latest
            command: ["python", "/app/scripts/trust_score.py"]
            volumeMounts:
              - name: shared-data
                mountPath: /data
            restartPolicy: OnFailure
            volumes:
              - name: shared-data
                persistentVolumeClaim:
                  claimName: score-storage

```

- kubectl apply -f k8s/cronjobs/cron-failure-classification.yaml
- kubectl apply -f k8s/cronjobs/cron-trust-score.yaml
- kubectl apply -f k8s/cronjobs/cron-health-scoring.yaml
- kubectl apply -f k8s/cronjobs/cron-rollback.yaml

3.9 Open Policy Agent

Open Policy Agent (OPA) Gatekeeper is placed in Kubernetes cluster to have control over various policies that work, such as mandatory labels, health probes, and secure container settings, so that non-compliant workloads could be avoided to be deployed.

label-template.yaml

```yaml
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
  labels:
    app: self-healing
    owner: prasanna
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{
          "msg": msg,
          "details": {"missing_labels": missing}
        }] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {"app", "owner"}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("Missing required labels: %v", [missing])
        }
```

probes-template.yaml

```yaml
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredprobes
  labels:
    app: self-healing
    owner: prasanna
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredProbes
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredprobes

        violation[{"msg": msg}] {
          container := input.review.object.spec.template.spec.containers[_]
          not container.livenessProbe
          msg := "Liveness probe is required"
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.template.spec.containers[_]
          not container.readinessProbe
          msg := "Readiness probe is required"
        }
```

- kubectl apply -f opa/label-constraint.yaml
- kubectl apply -f opa/probes-constraint.yaml

4 Testing and Outputs

4.1 Changing the image tag of flask application which is unknown or does not exist

Jenkinsfile ! deployment.yaml ×

k8s > ! deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
  labels:
    app: self-healing
    owner: prasanna
spec:
  replicas: 2
  selector:
    matchLabels:
      app: self-healing
      owner: prasanna
  template:
    metadata:
      labels:
        app: self-healing
        owner: prasanna
        prometheus: enabled
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/path: "/metrics"
        prometheus.io/port: "5000"
    spec:
      containers:
      - name: flask-container
        image: ppankajs/self-healing-app:v99999
        ports:
        - containerPort: 5000
        livenessProbe:
          httpGet:
            path: /health
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 10
```

4.2 Getting all pods present in the Kubernetes cluster

```
ubuntu@ip-10-0-1-62:~$ kubectl get pods
NAME                                    READY   STATUS             RESTARTS         AGE
crashpod                                0/1     CrashLoopBackOff   88 (47s ago)     23h
failpod                                 0/1     CrashLoopBackOff   86 (3m8s ago)    23h
failure-classification-29244765-nlwd6   0/1     Completed          0                6m16s
failure-classification-29244770-wjr8n   0/1     Completed          0                76s
failure-test-l69z5                      0/1     Completed          0                8m21s
flask-app-67c44584cb-8rskh              1/1     Running            0                58m
flask-app-67c44584cb-ff955              1/1     Running            0                58m
flask-app-6b4d8f789b-2szsp              0/1     ImagePullBackOff   0                52m
health-scoring-29244765-kz4lh           0/1     Completed          0                6m16s
health-scoring-29244768-4gfps           0/1     Completed          0                3m16s
health-scoring-29244771-h7vgb           0/1     Completed          0                16s
postgres-59c4578695-hbm8z               0/1     Running            2 (3h4m ago)     23h
postgres-954f854b9-ntvfk                0/1     Running            2 (3h4m ago)     23h
trust-score-check-29244760-ql8zh        0/1     Completed          0                11m
trust-score-check-29244770-gl2cb        0/1     Completed          0                76s
trust-test-68h4v                        0/1     Completed          0                8m55s
ubuntu@ip-10-0-1-62:~$
```

After chaning the Image tag of the flask application it is showing me the error called ImagePullBackoff Error. Crashpod and failpod are those use for testing the my scripts sample pods. The command use for fetching all pods in Kubernetes

- kubectl get pods

4.3 Running the cron job

```
ubuntu@ip-10-0-1-62:~$ kubectl create job --from=cronjob/failure-classification failure-test
job.batch/failure-test created
ubuntu@ip-10-0-1-62:~$ kubectl create job --from=cronjob/trust-score-check trust-test
job.batch/trust-test created
ubuntu@ip-10-0-1-62:~$ kubectl create job --from=cronjob/health-scoring health-test
job.batch/health-test created
```

The command use for creating or running the cron job is

- kubectl create job --from=cronjob/failure-classification failure-test
- kubectl create job --from=cronjob/trust-score-check trust-test
- kubectl create job --from=cronjob/health-scoring health-test

The logs of created jobs are fetched from this command and here you can see the image which we have added in the flask app which is untrusted.

```
ubuntu@ip-10-0-1-62:~$ kubectl logs job/failure-test
[INFO] Starting Failure Classification...
[FAILURE] CrashLoopBackOff in crashpod
[FAILURE] CrashLoopBackOff in failpod
[FAILURE] CrashLoopBackOff in flask-app-6b4d8f789b-2szsp
[FAILURE SCORE] 30
ubuntu@ip-10-0-1-62:~$ kubectl logs job/trust-test
[INFO] Checking image trust score...
[IMAGE] ppankajs/self-healing-app:v99999
[TRUST SCORE] 40
ubuntu@ip-10-0-1-62:~$ kubectl logs job/health-test
[INFO] Evaluating resource health score...
[HEALTH SCORE] 48/100
ubuntu@ip-10-0-1-62:~$
```

- kubectl logs job/failure-test
- kubectl logs job/trust-test
- kubectl logs job/health-test

This the cron job for using the trusted image tag which is secure and the command use is

```
ubuntu@ip-10-0-1-62:~$ kubectl create job --from=cronjob/explainable-rollback rollback-test
job.batch/rollback-test created
```

- kubectl create job --from=cronjob/explainable-rollback rollback-test

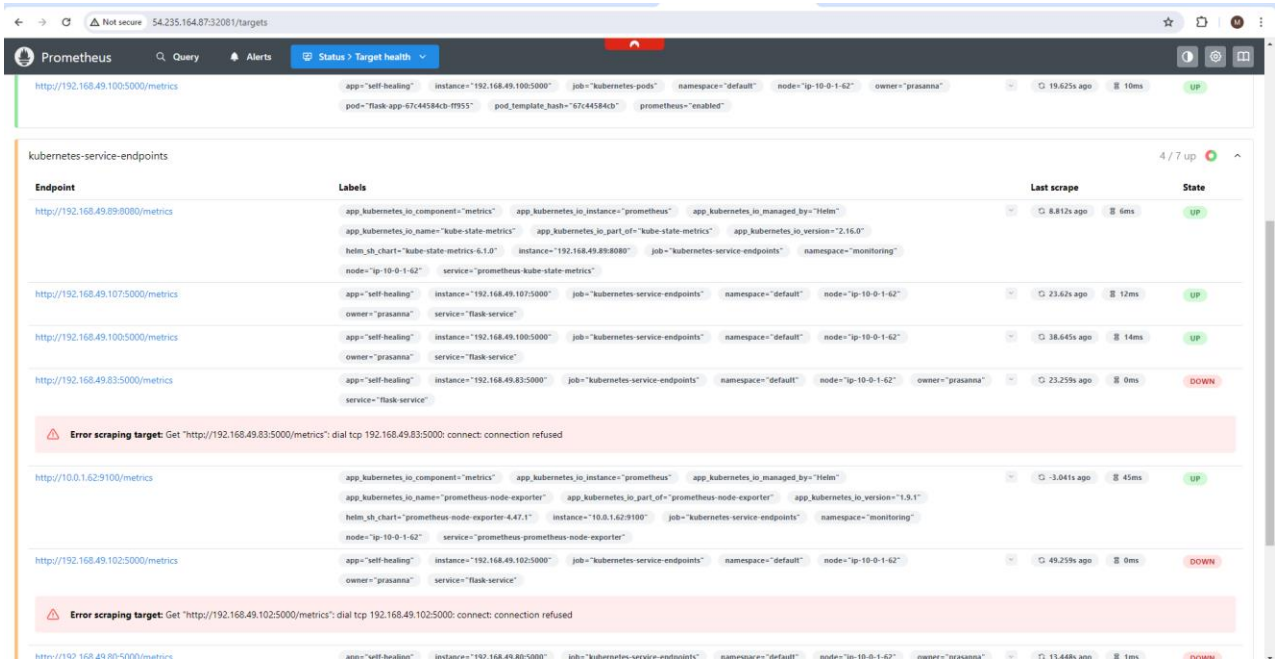
The log of created job are fetched from this command

```
ubuntu@ip-10-0-1-62:~$ kubectl logs job/rollback-test
deployment.apps/flask-app image updated
[SCORES] {'failure': 30, 'trust': 40, 'health': 51}
[EVALUATION] Deployment Score: 40
[TRIGGERED] Score unhealthy. Triggering rollback...
[ROLLBACK] Deployment score too low. Rolling back to v96
```

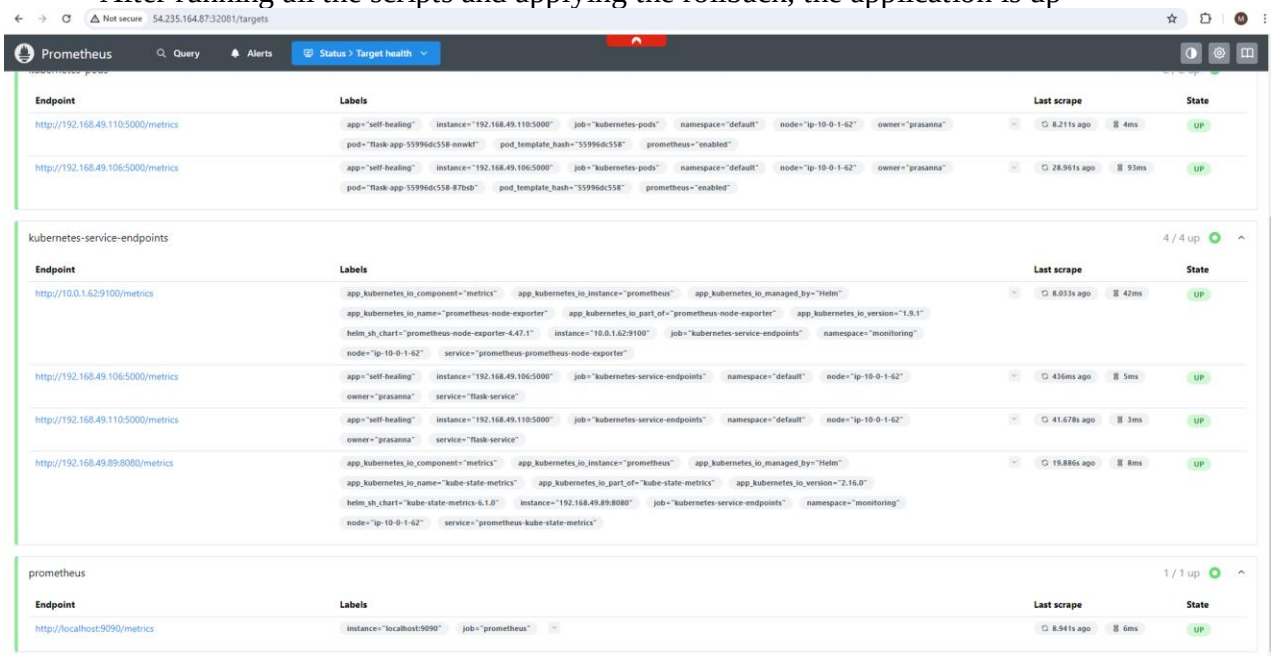
- kubectl logs job/rollback-test

4.4 Output are monitor from the Prometheus

- The output after changing the image tag which is insecure and it shows that the application is down



- After running all the scripts and applying the rollback, the application is up



5 References

HashiCorp, 2025. *Terraform Documentation*. [online] Available at: https://developer.hashicorp.com/terraform/docs

Ansible Project, 2025. *Ansible Documentation*. [online] Available at: https://docs.ansible.com/

Jenkins Project, 2025. *Jenkins User Documentation*. [online] Available at: https://www.jenkins.io/doc/

Docker Inc., 2025. *Docker Documentation*. [online] Available at: https://docs.docker.com/

Amazon Web Services, 2025. *Amazon EC2 Documentation*. [online] Available at: https://docs.aws.amazon.com/ec2/