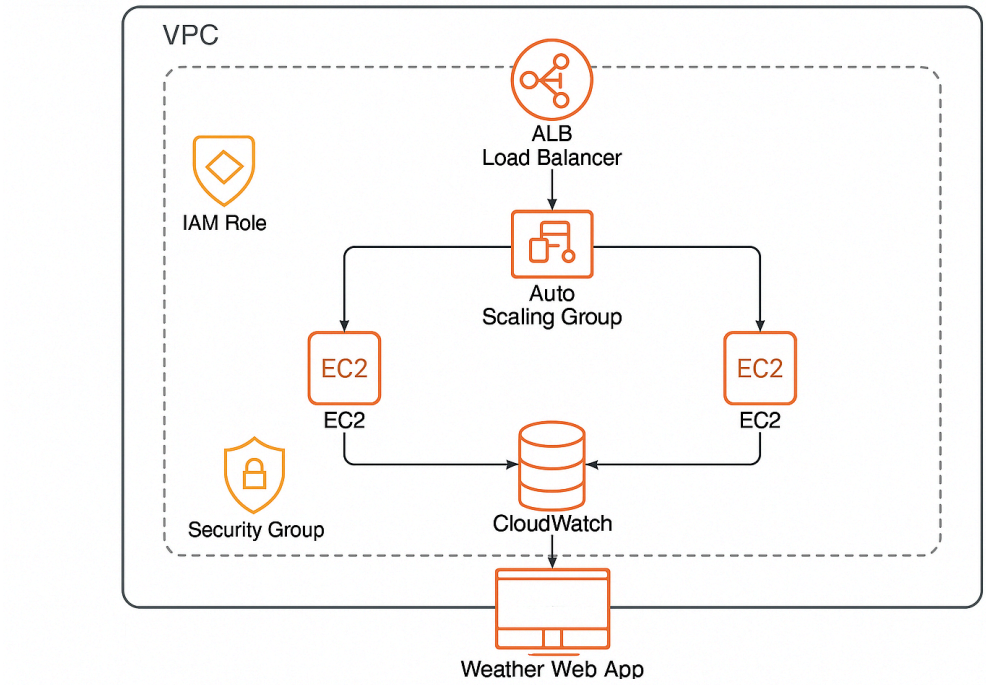
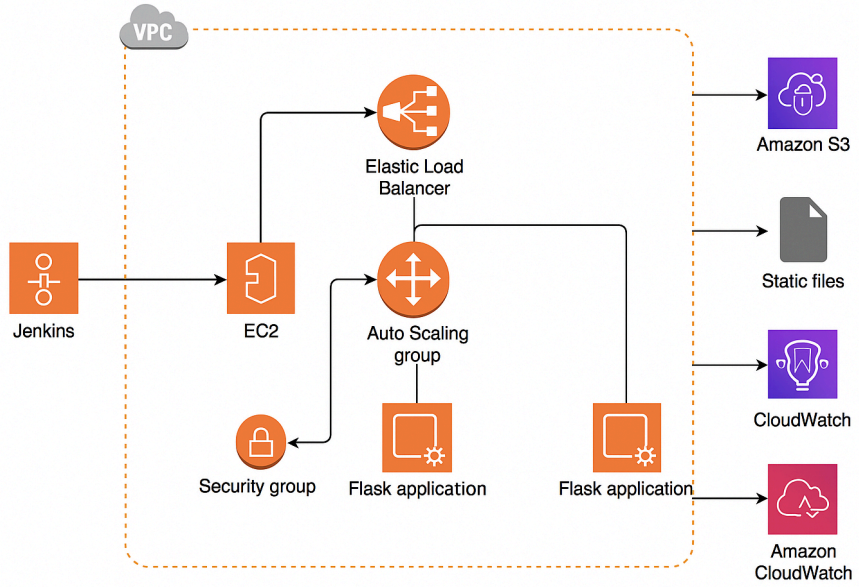
Cloud-Based Web App Deployment & Resilience Simulation

Cloud Infrastructure Project - Angeline CSE 'A'

1. Architecture Diagram

A visual representation of the deployed architecture, including:





2. Code & Scripts

Terraform Scripts

- `main.tf`: Defines EC2 instance, security group, IAM role, and user data.

```
provider "aws" {
  region = "eu-north-1"
}

resource "aws_instance" "weather_ec2" {
  ami                         = "ami-006b4a3ad5f56fbd6"
  instance_type               = "t3.micro"
  key_name                    = "WeatherAppKey"
vpc_security_group_ids      = ["sg-09328a13b7916ba38"]
  subnet_id                   = "subnet-05f4980e60d37121e"
  tags = {
    Name = "WEATHERMAN"
  }

  user_data = <<-EOF

              yum update -y
              yum install -y python3 git
              cd /home/ec2-user
              git clone https://github.com/23f-3004447/WeatherApp.git
              cd WeatherApp
              pip3 install -r requirements.txt
              nohup python3 app.py > flask.log 2>&1 &
              EOF
}
```

Jenkinsfile

- Pipeline script to automate deployment using CI/CD.

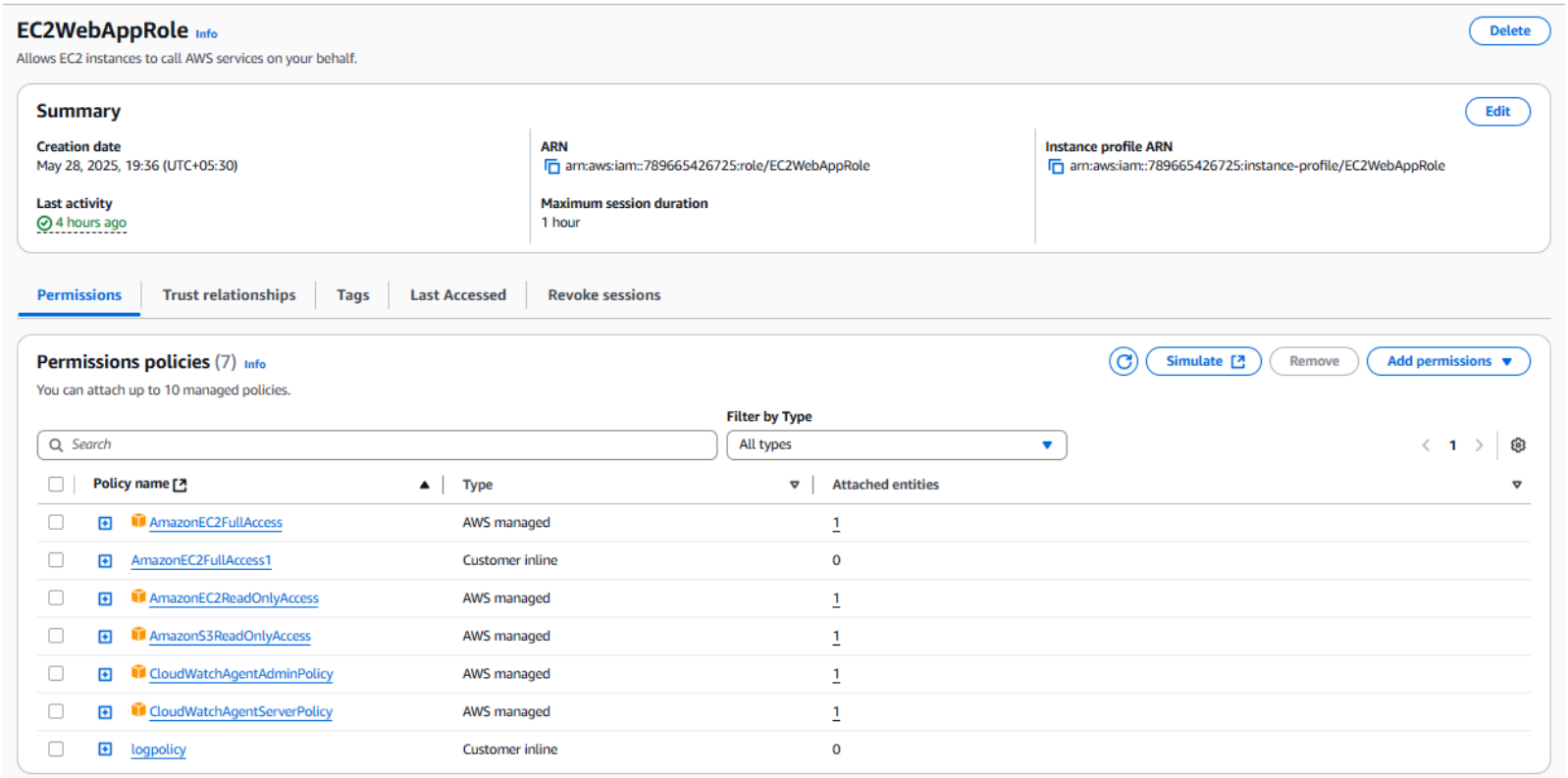
```
pipeline {
    agent any

    stages {
        stage('Clone Repo') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/23f-3004447/WeatherApp.git',
                    credentialsId: 'a5308c41-1e1b-4839-af30-fe85ccdef2fa'
            }
        }

        stage('Install Dependencies') {
            steps {
                sh 'pip3 install -r requirements.txt'
            }
        }

        stage('Run Flask App') {
            steps {
                sh '''
                    fuser -k 5000/tcp || true
                    nohup python3 app.py > flask.log 2>&1 &
                '''
            }
        }
    }
}
```

GitHub Repository: https://github.com/23f-3004447/WeatherApp.git

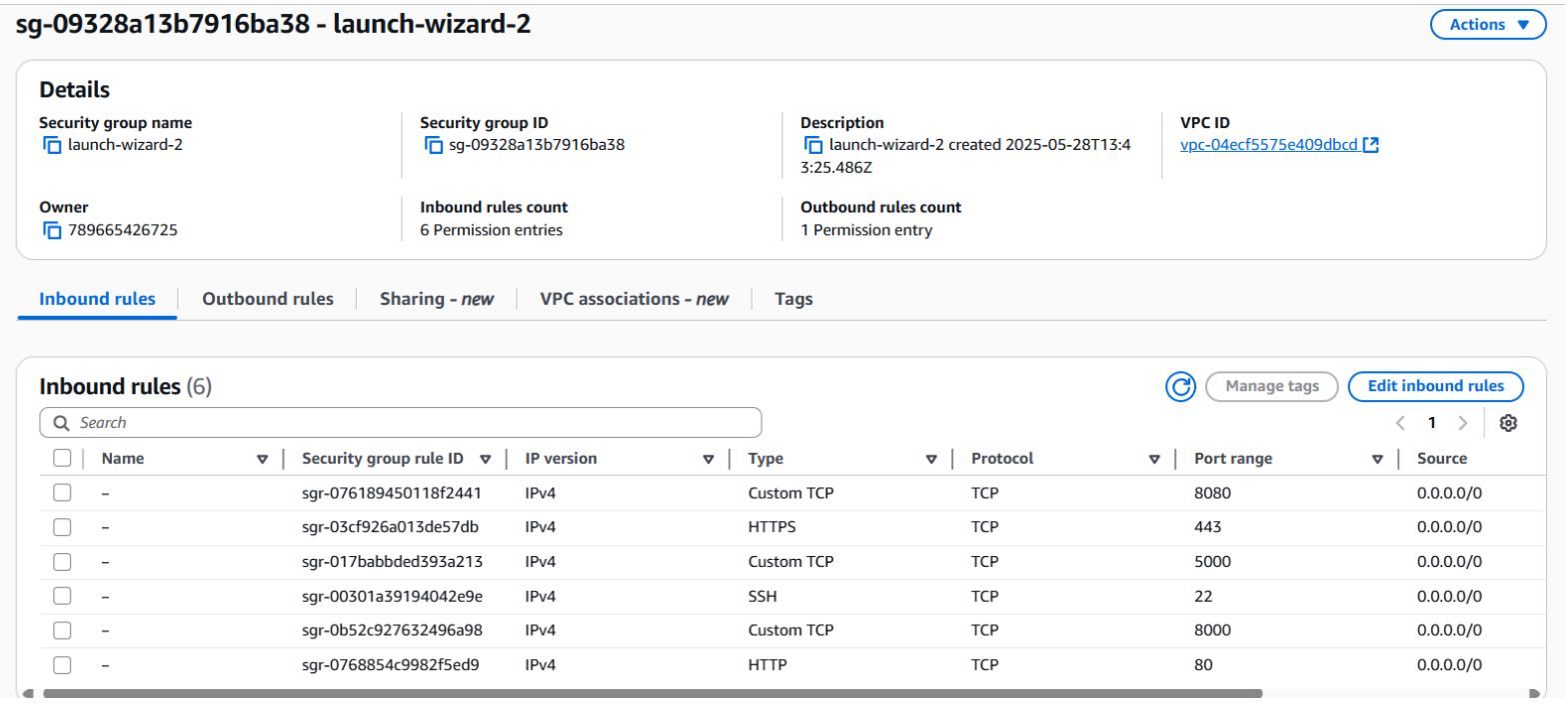
---

3. Screenshots

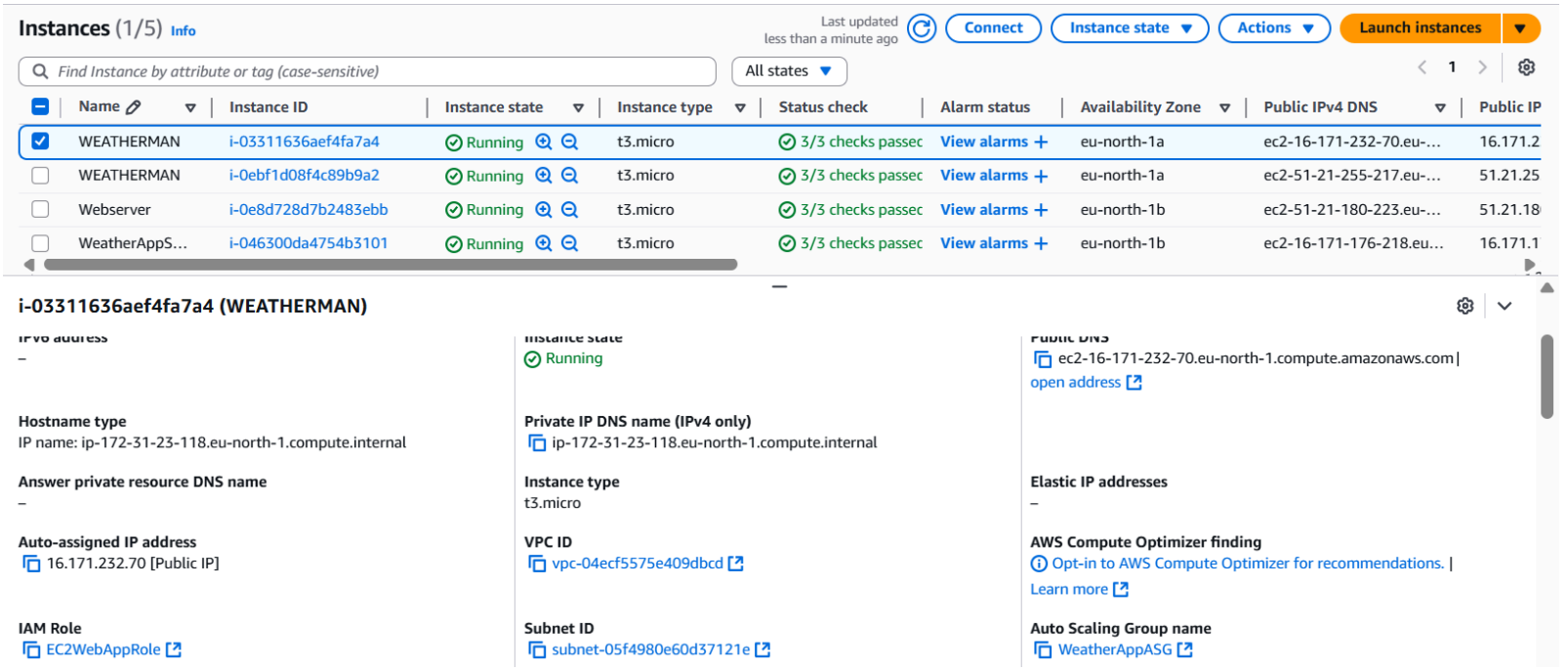
- IAM Role: `EC2WebAppRole` with permissions for EC2, CloudWatch.



- Security Group: `sg-09328a13b7916ba38` with necessary inbound/outbound rules.

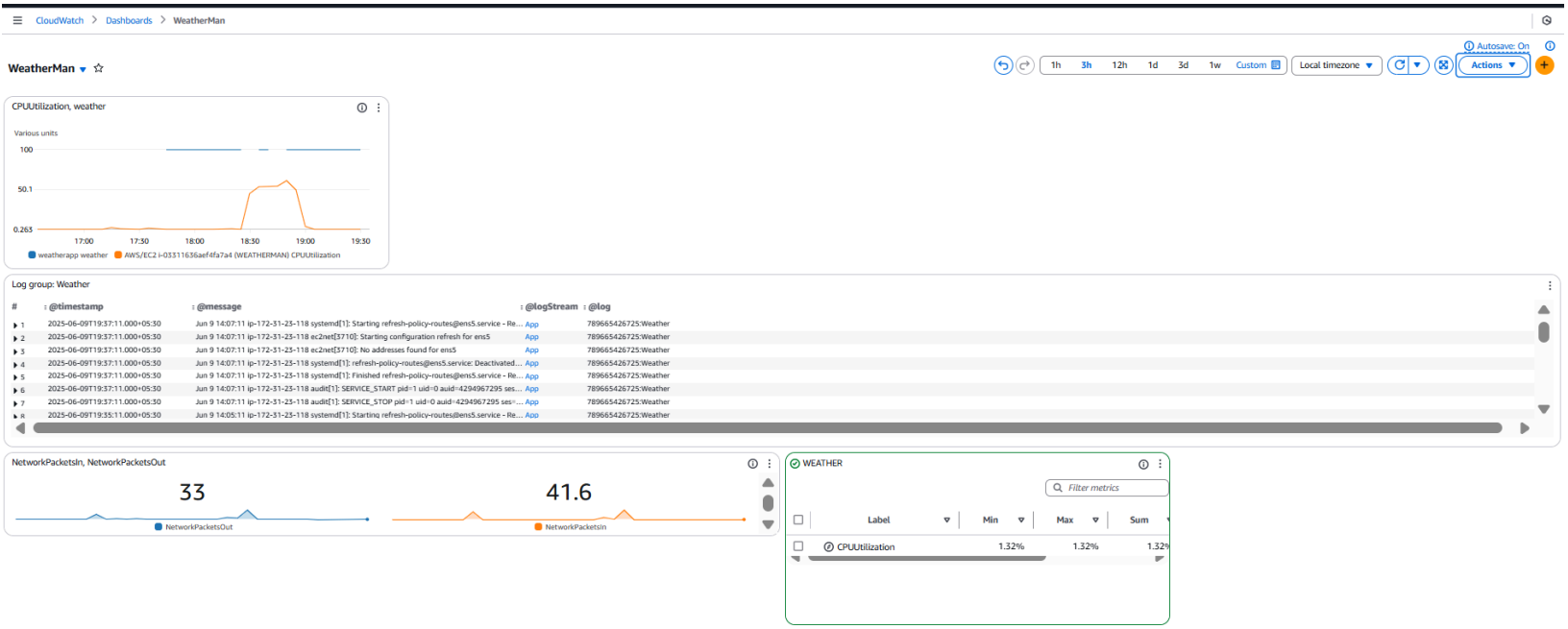


- EC2 Instance Tags: `Name=WeatherAppServer`, `Role=WebApp`
- CloudWatch Agent Configuration & Logs



4. Monitoring Dashboard Snapshots

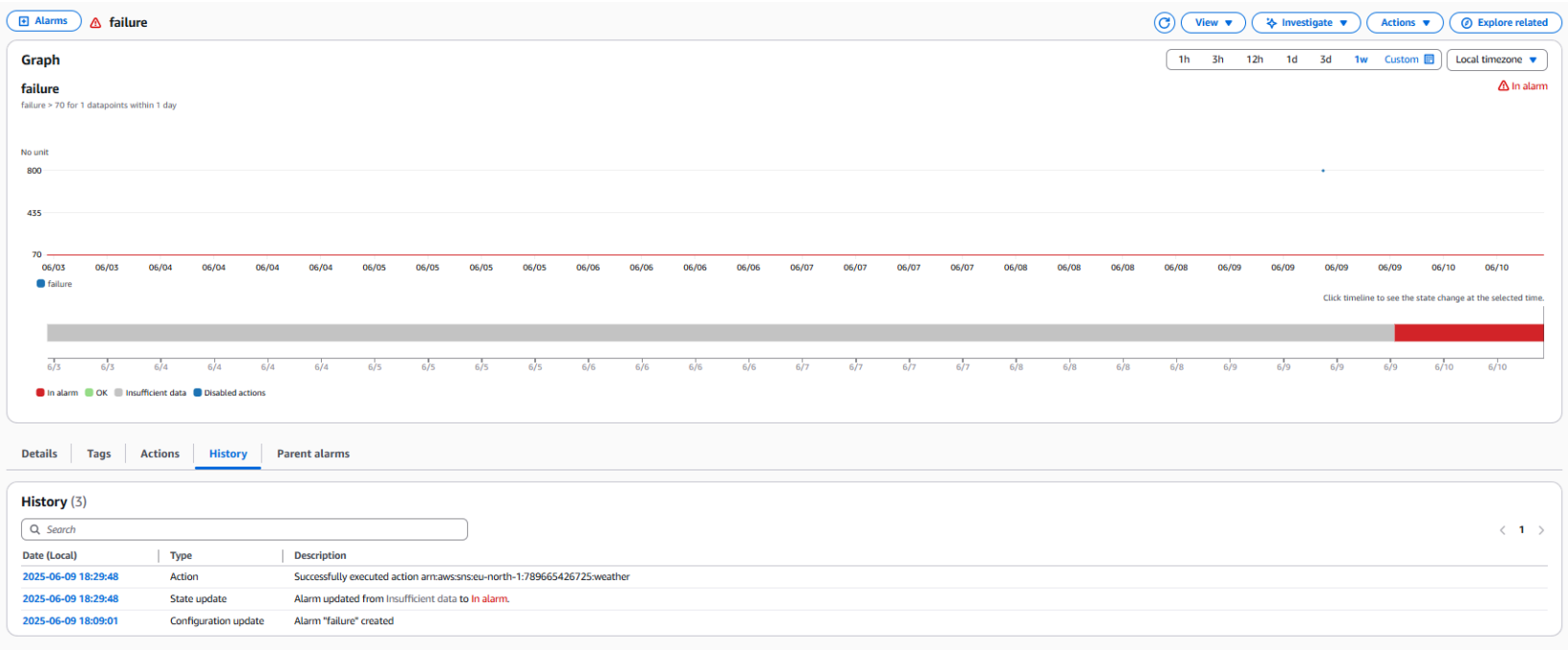
- **Service**: Amazon CloudWatch
- Dashboards monitoring:
  - CPU Utilization
  - Network In/Out
  - Custom Logs (e.g., application logs)



5. Simulation Report

System Failure – Custom Alarm (`failure`)

- **Alarm Name:** `failure`
- **Metric Tracked:** Custom metric – `failure`
- **Condition:** `failure > 70` for `1 datapoint` within `1 day`
- **State:** `In alarm`
- **Trigger Time:** `2025-06-09 18:29:48` (Local Time)
- **Alarm History:**
  - `2025-06-09 18:09:01` – Alarm created.
  - `2025-06-09 18:29:48` – Alarm transitioned to `In alarm`.
  - SNS Action executed: `arn:aws:sns:eu-north-1:789665426725:weather`.



- **Response:**
  - Notification sent via SNS to the monitoring topic.

- No automatic remediation was configured for this custom metric.
- Manual inspection initiated.
- Admin verified application logs for anomalies.
- Instance showed abnormally high values; restarted Flask service.

---

6. Final Presentation / Video

Capstone.mp4

http://weatherman-alb-787651582.eu-north-1.elb.amazonaws.com/

---

7. Conclusion

This project showcases end-to-end deployment of a Flask app using Terraform, Jenkins, and AWS services.

It demonstrates cloud-native practices like infrastructure as code, scalable architecture, proper security, real-time monitoring, CI/CD automation, and a successful resilience simulation.