**Final Capstone Project: AWS Multi-Region, Multi-Tier App Deployment**

**Table of Contents**

1. Project Overview
2. Project Phases
   * PHASE 1: EKS Multi-Tier Deployment
   * PHASE 2: CI/CD Automation with CodePipeline
   * PHASE 3: Multi-Region Infrastructure (CloudFormation + Terraform)
   * PHASE 4: Disaster Recovery with Route 53 Failover
   * PHASE 5: Monitoring with CloudWatch, Prometheus, Grafana
3. Architecture Diagram
4. Security Measures
5. Next Steps & Enhancements
6. Maintainer

**Project Overview**

This capstone project demonstrates a robust and fault-tolerant multi-tier web application deployment across multiple AWS regions, using Infrastructure as Code (IaC) and modern DevOps practices.

Application Stack:

* Backend: Spring Boot
* Frontend: Thymeleaf
* Database: MySQL (Amazon RDS)

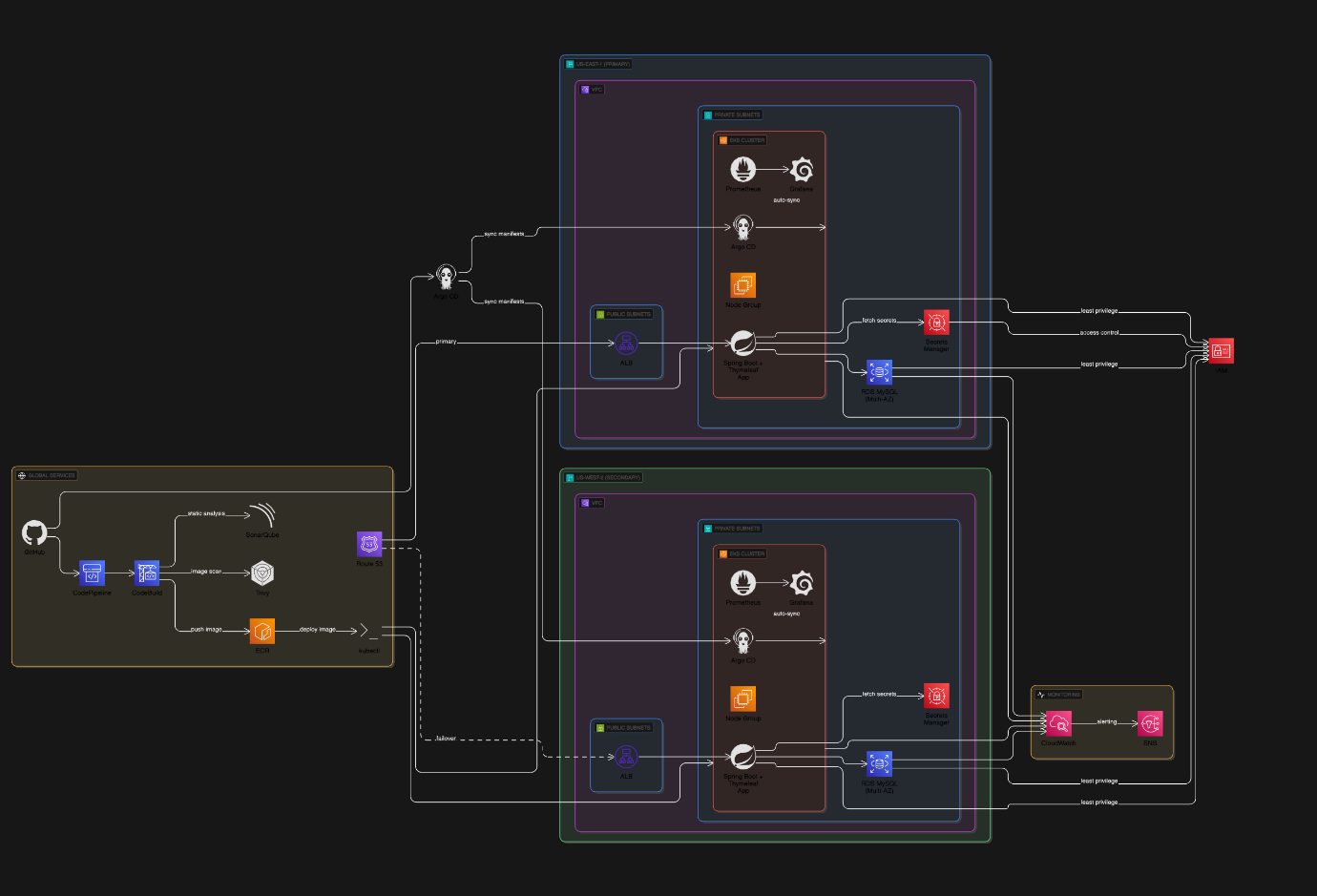
Objectives:

* Achieve high availability and fault tolerance via multi-region deployment
* Implement CI/CD pipelines with code quality and security gates
* Ensure observability and automation via GitOps and monitoring tools

Tools & Technologies:

* EKS, RDS, Route 53, CloudFormation, Terraform, CodePipeline, Prometheus, Grafana, Trivy, SonarQube, Argo CD

**Architecture Diagram:**

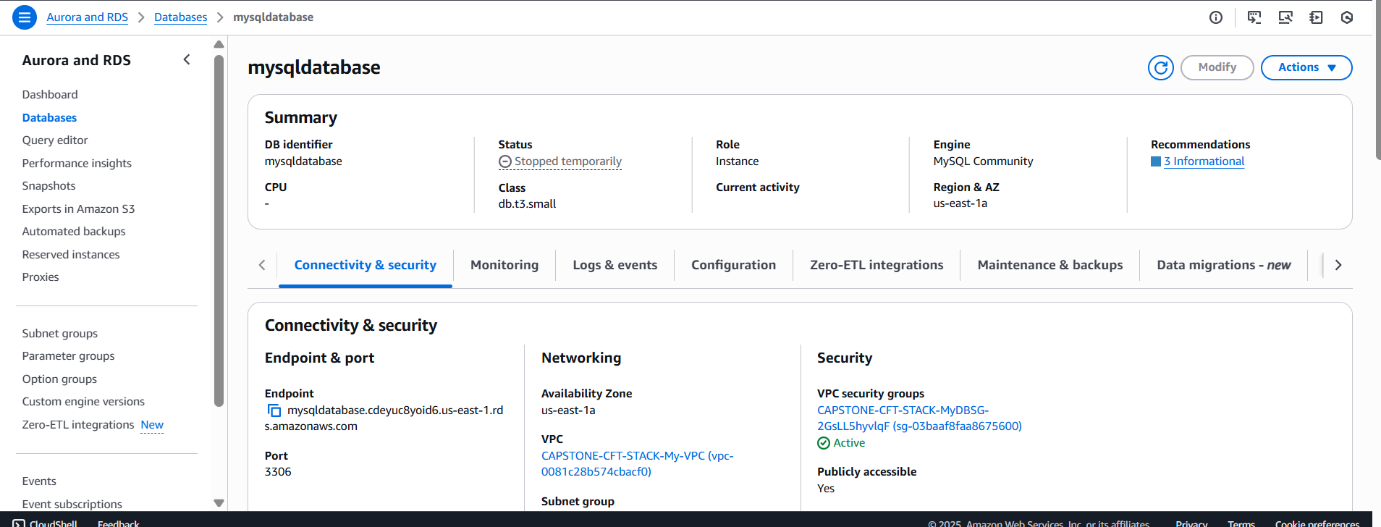


**PHASE 1: EKS Multi-Tier Deployment**

* **Application:** Spring Boot + Thymeleaf
* **Infrastructure:**
  + VPC with public/private subnets
  + Internet Gateway, NAT Gateway
  + EKS Cluster with Managed Node Groups
  + RDS MySQL (Multi-AZ, private subnets)
* **Deployment:**
  + Dockerize Spring Boot app
  + Push image to ECR
  + Deploy to EKS using deployment.yaml and service.yaml

Screenshots:

Mydatabase:

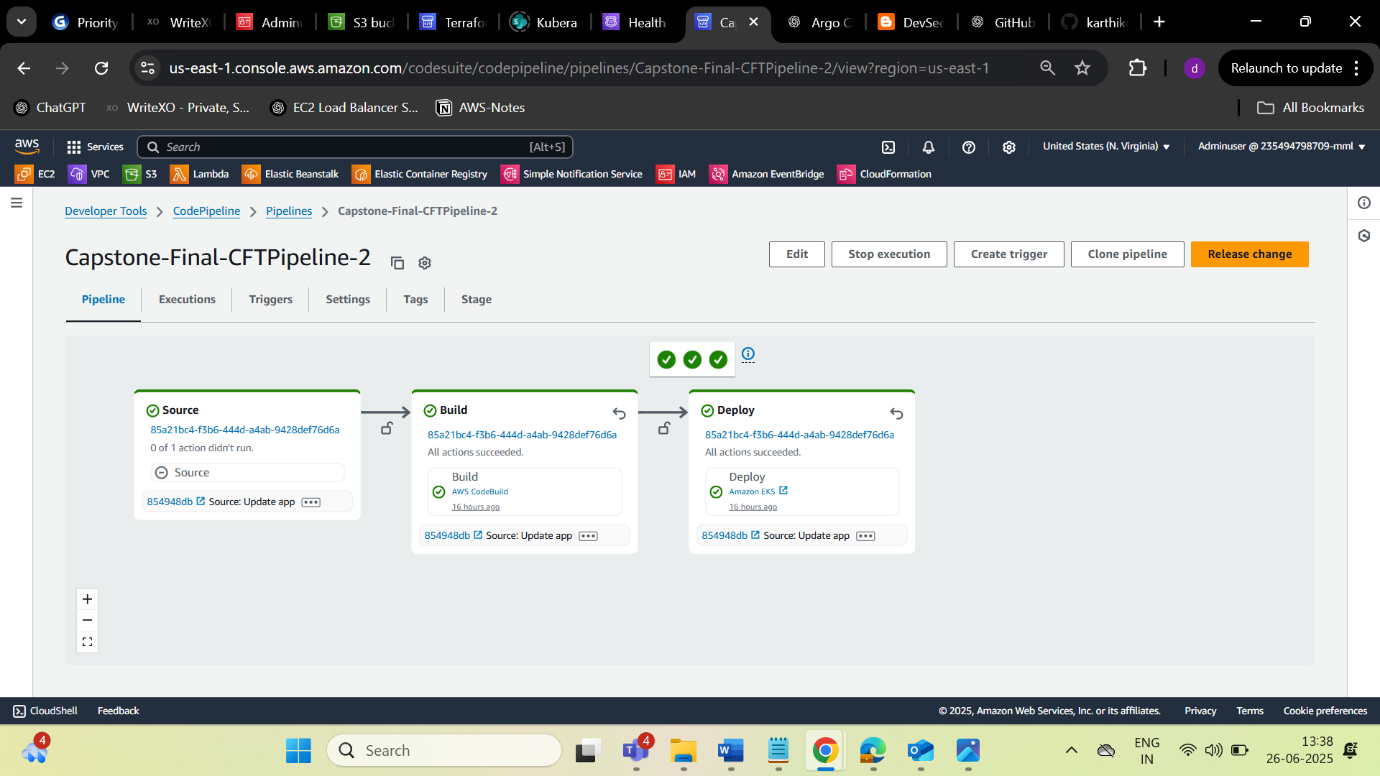


**PHASE 2: CI/CD Automation with CodePipeline**

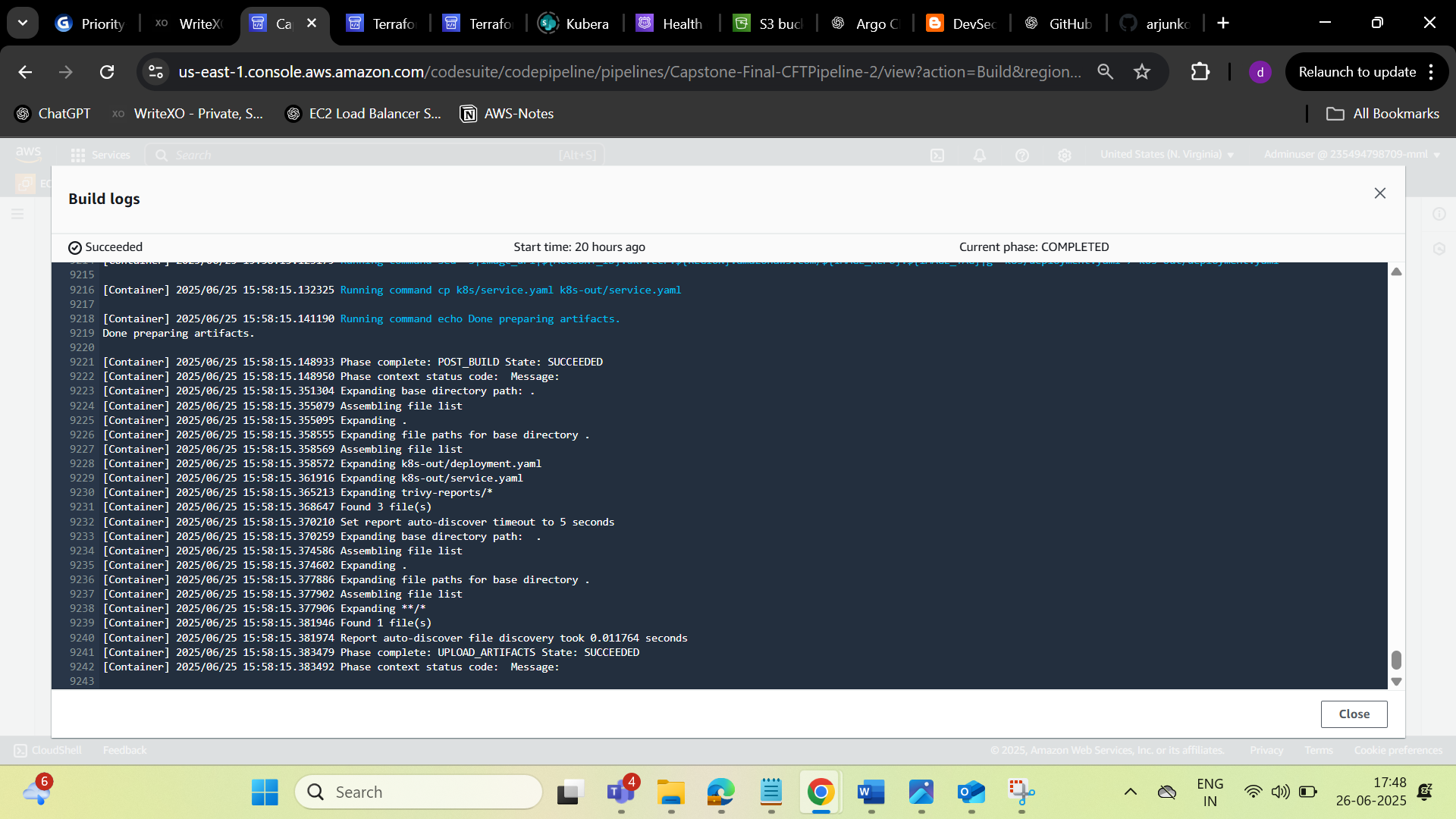
* **Tools Used:**
  + **CodePipeline** (Pipeline Orchestration)
  + **CodeBuild** (Build & kubectl apply)
  + **ECR** (Docker image store, optional)
  + **S3** (Artifact storage) - '\*\*/\*'

**Screenshots:**

* CodePipeline run success
* CodeBuild logs



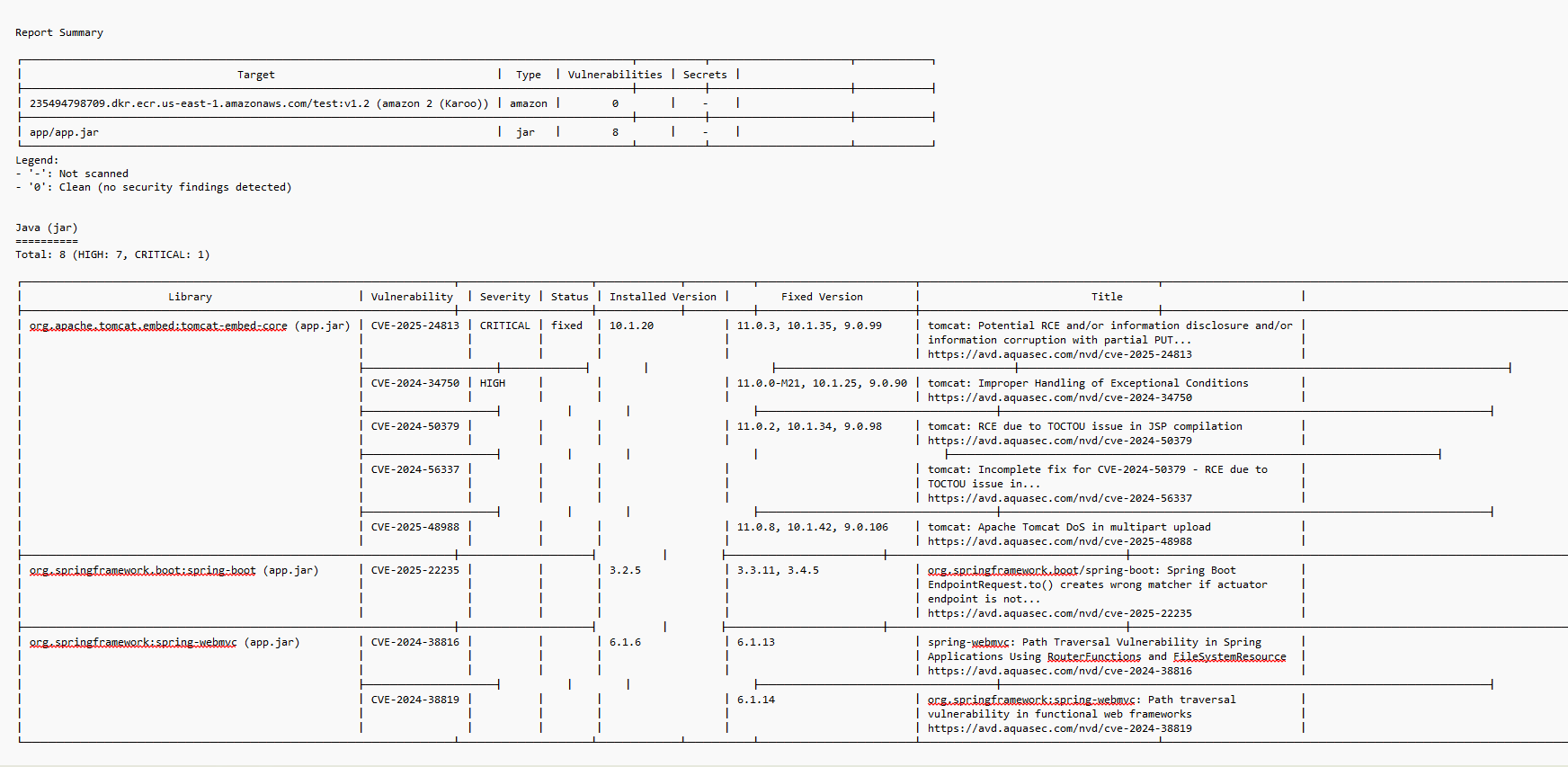
Build Logs



**SONARQUBE:**



**Trivy:**

****

**Integrations:**

* ✔ **SonarQube**: Static Code Analysis (via CodeBuild)
* ✔ **Trivy**: Docker image scanning before pushing to ECR or deploying

**PHASE 3: Multi-Region Infrastructure (CloudFormation + Terraform)**

* **Region A (us-east-1):** CloudFormation Stack
  + Contains all VPC, NAT, EKS, RDS, IAM components
* **Region B (us-west-2):** Terraform
  + main.tf, variables.tf
  + Uses modules for reusable infrastructure

**Commands:**

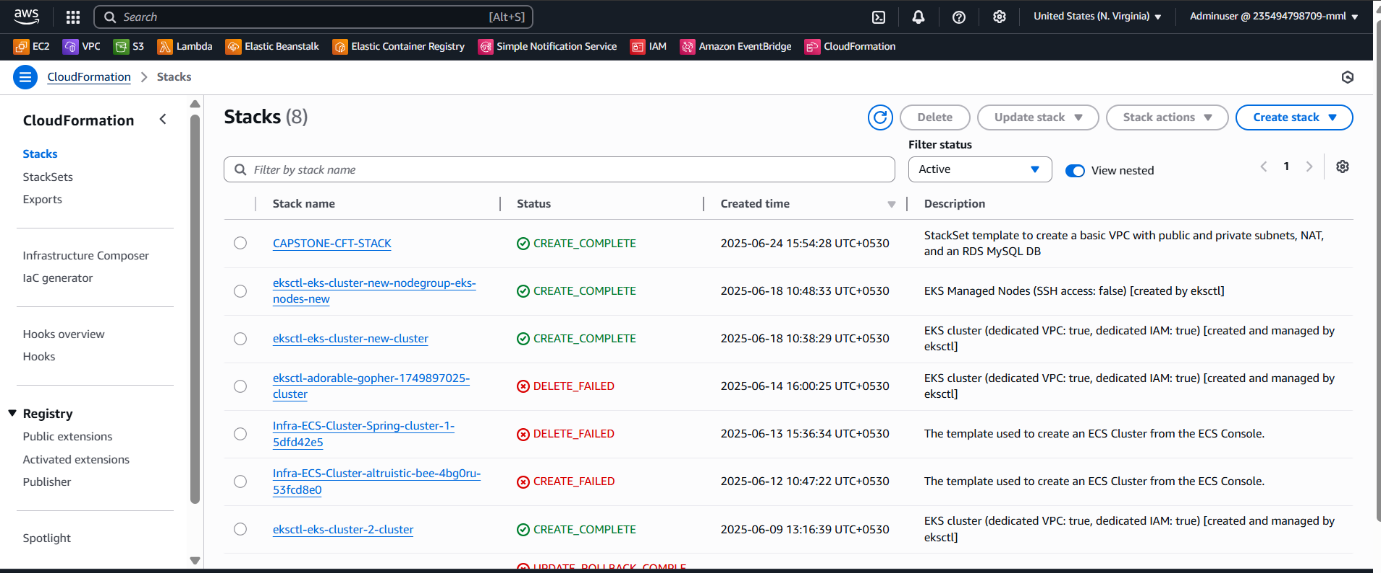
cd region-2-terraform

terraform init

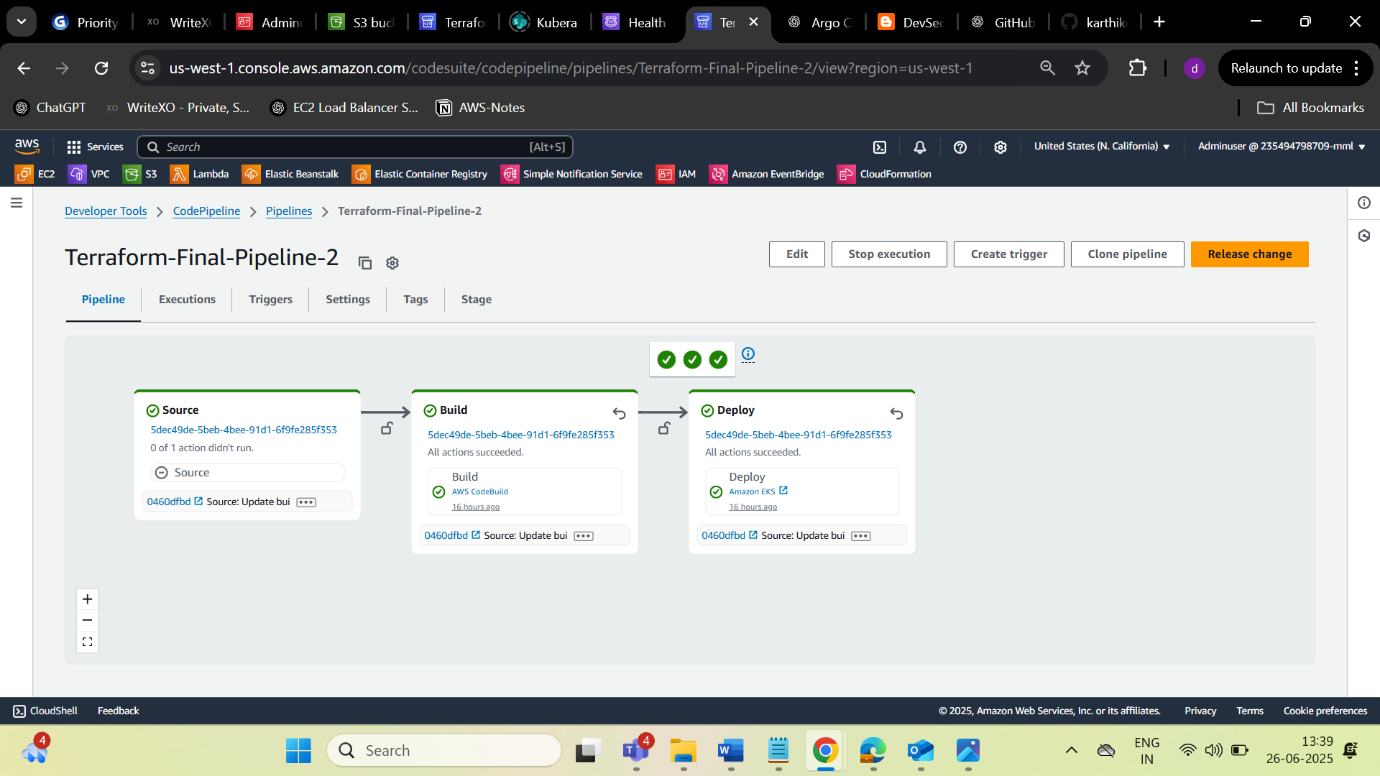
terraform apply

**Screenshots:**

* CloudFormation Stack in Console



Terraform pipeline



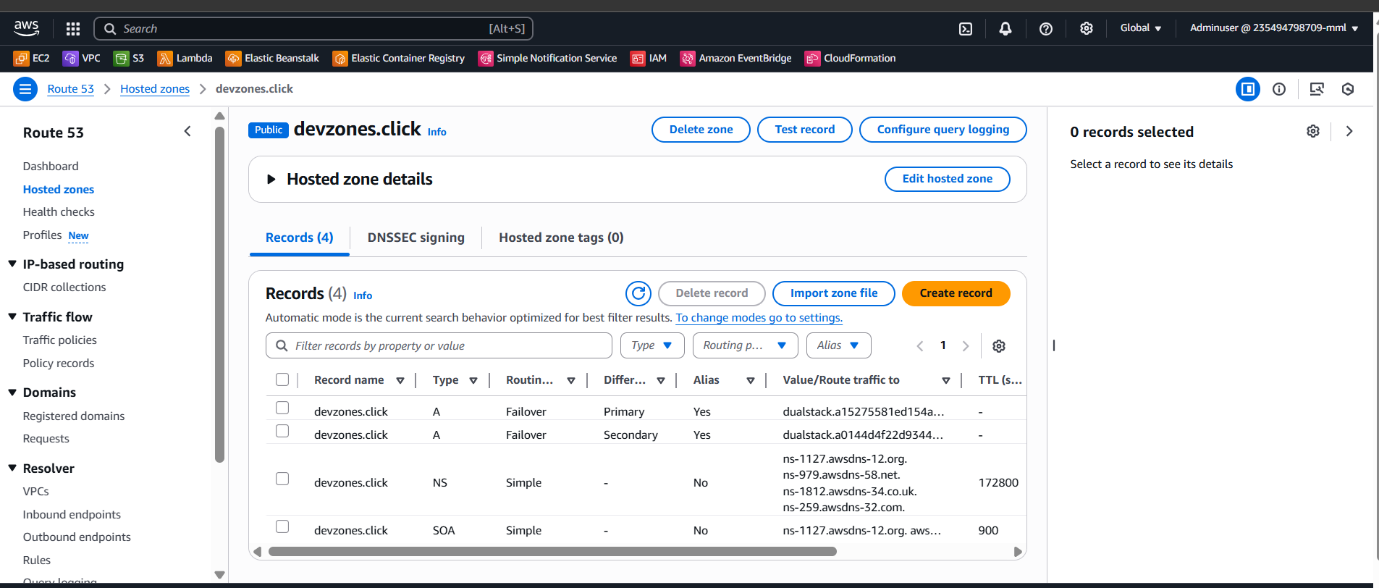
**PHASE 4: Disaster Recovery with Route 53 Failover**

* **Route 53 Records:**
  + Primary A record -> ALB in us-east-1
  + Secondary A record -> ALB in us-west-2
  + Health checks enabled on primary ALB
  + Failover Routing policy

**Behavior:** If the primary ALB fails health check, Route 53 routes traffic to the secondary ALB.

**Screenshots:**

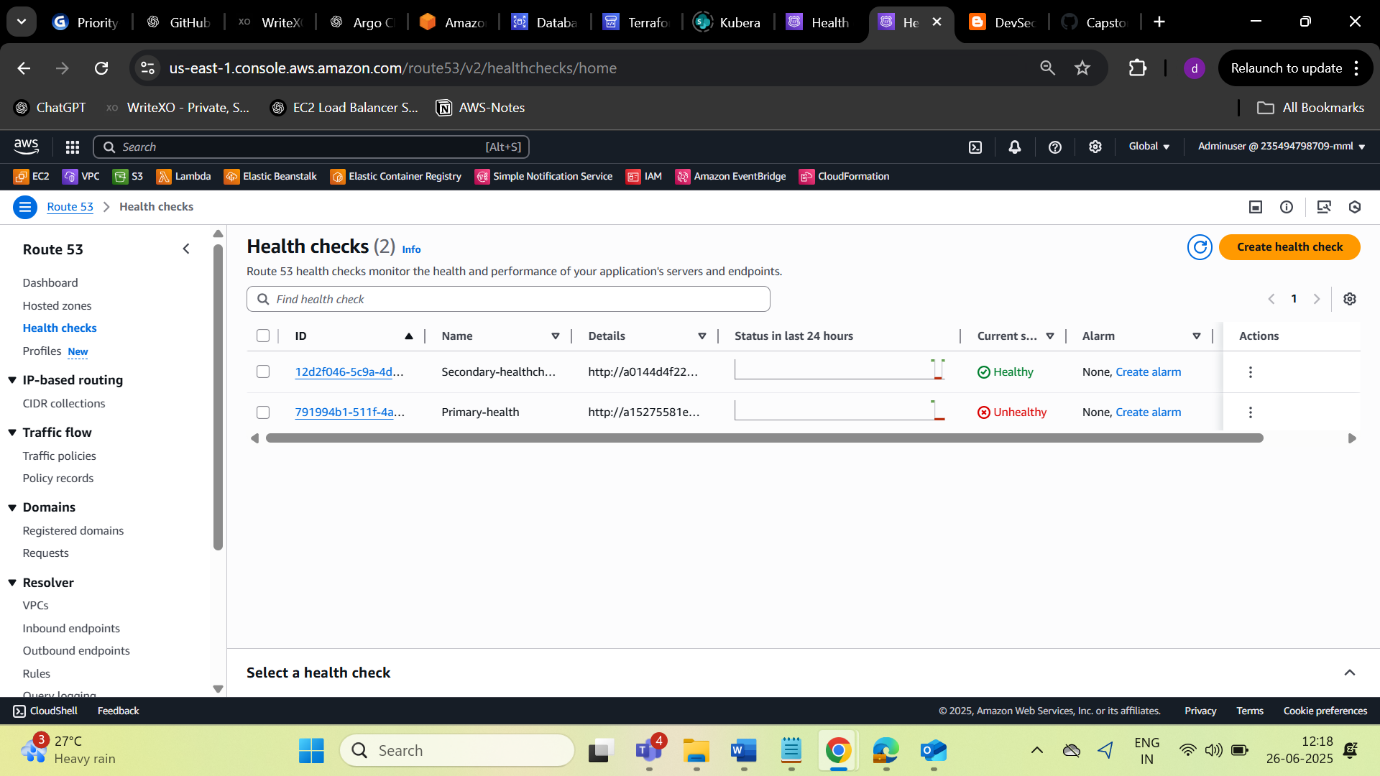
* Route 53 record set
* Health check config
* Traffic shift during failure



Failover routing

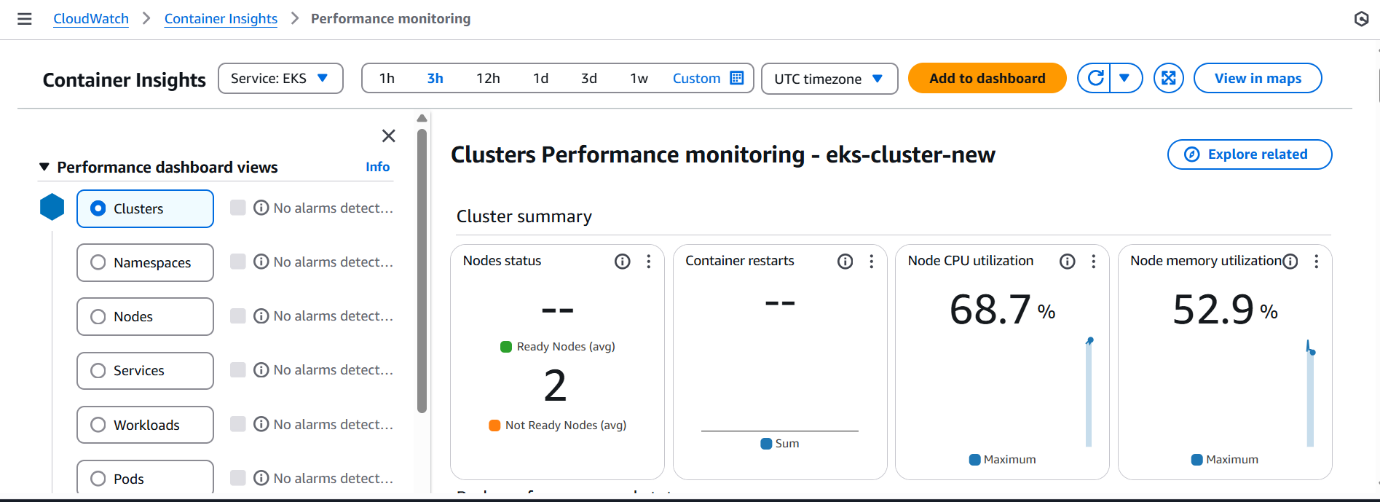
**Mechanism:**

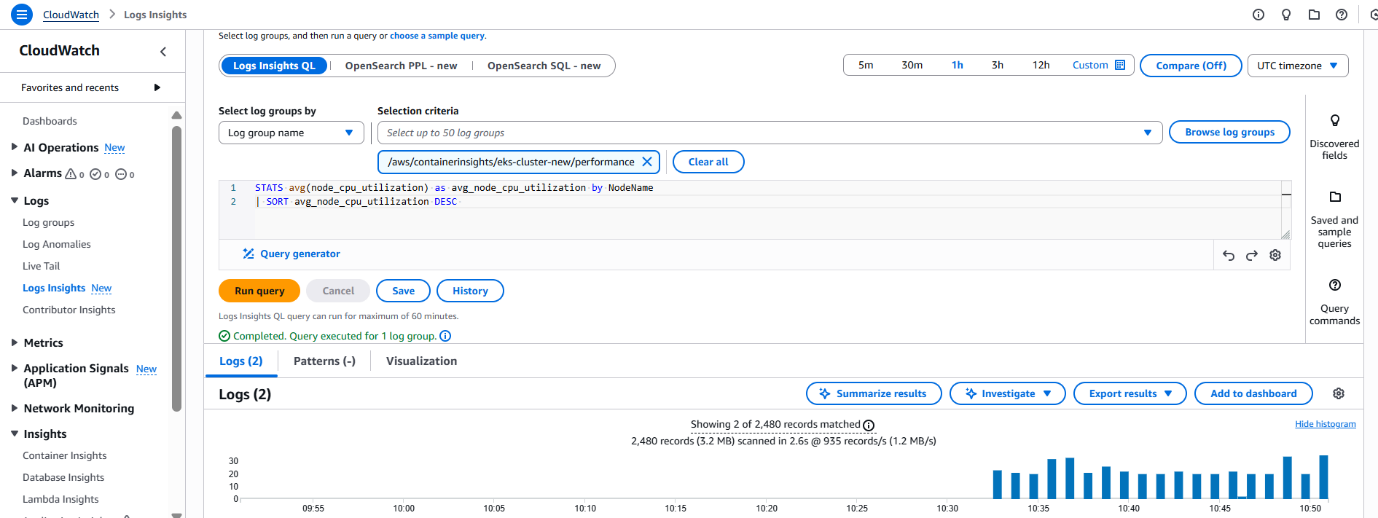
* If ALB in Region A fails health check, traffic is redirected to Region B

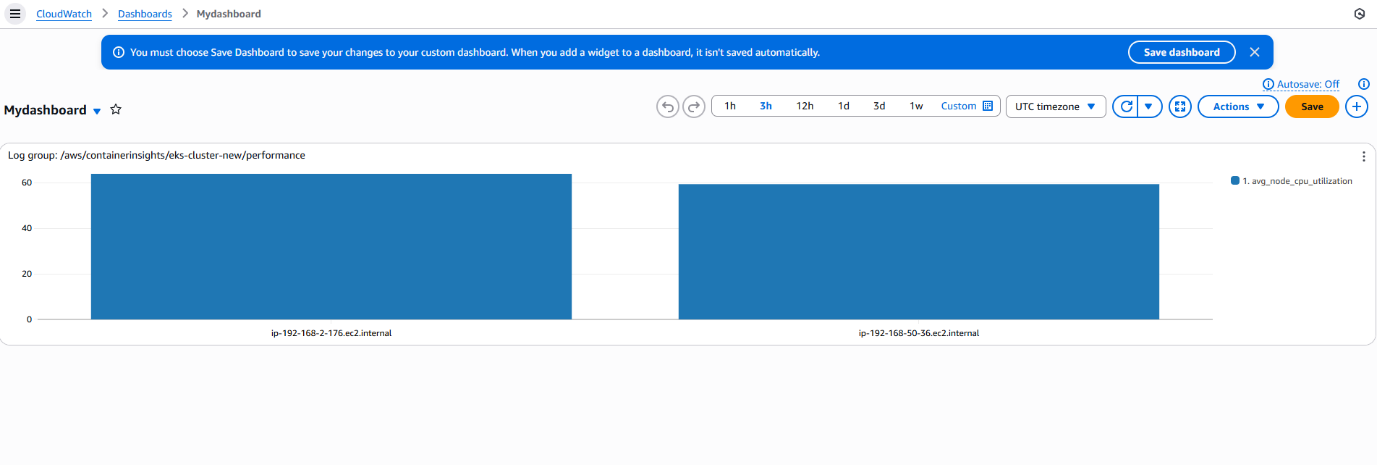


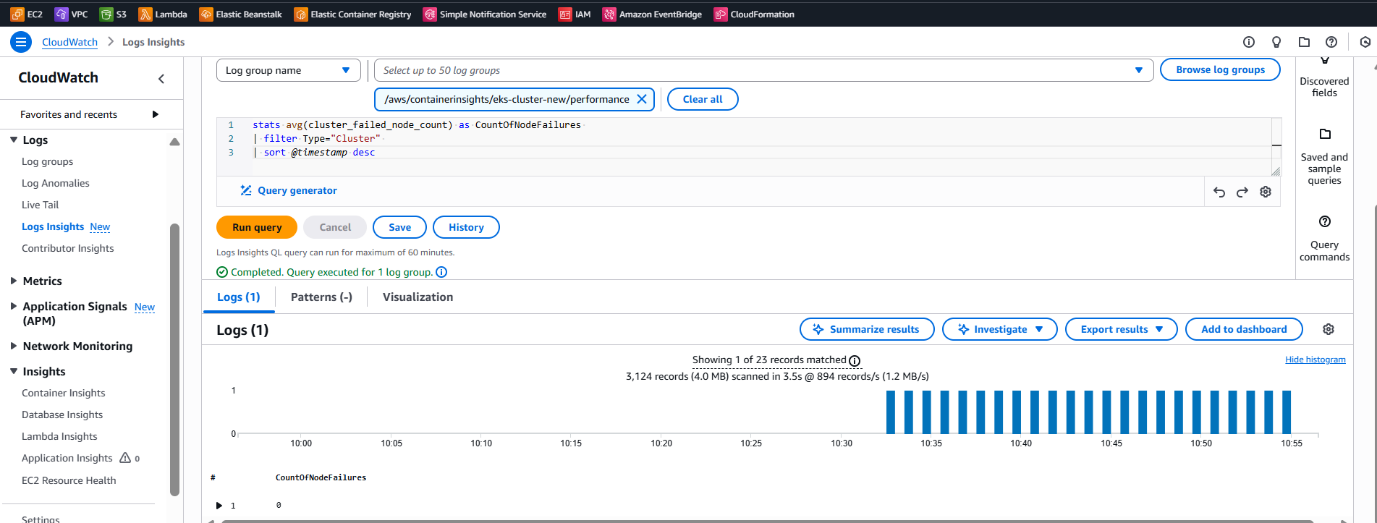
**PHASE 5: Monitoring with CloudWatch, Prometheus, Grafana**

* **CloudWatch Dashboards:**
  + EKS Pod CPU, Memory
  + RDS CPU usage
  + Alarms triggering SNS notifications

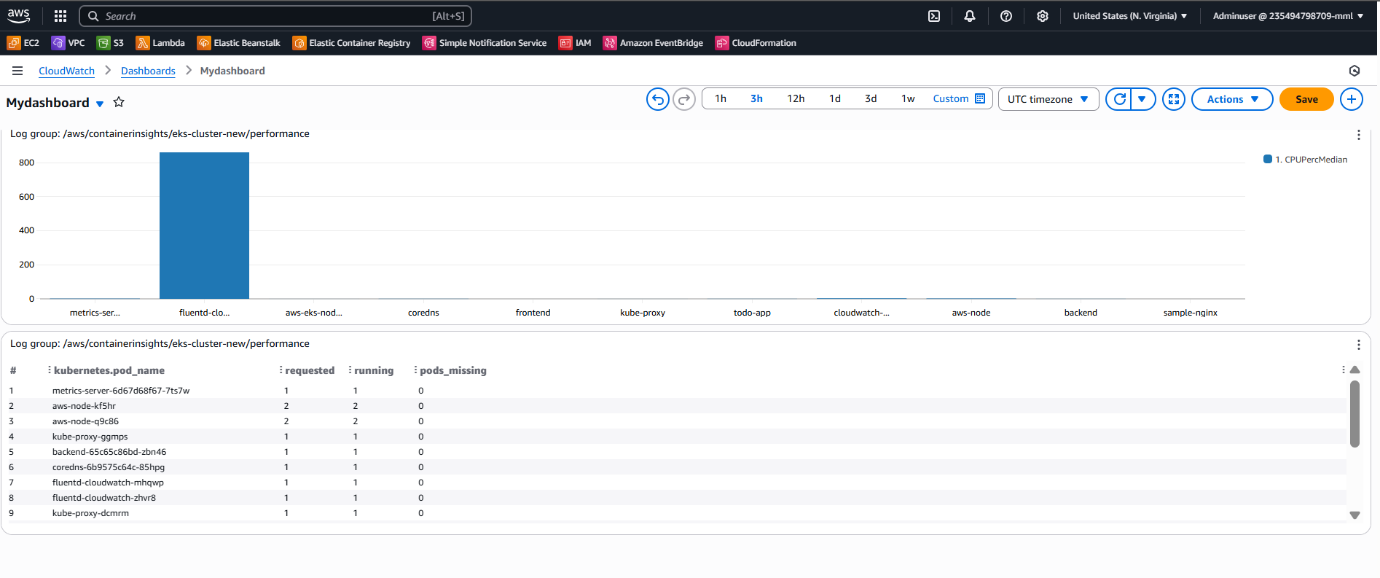




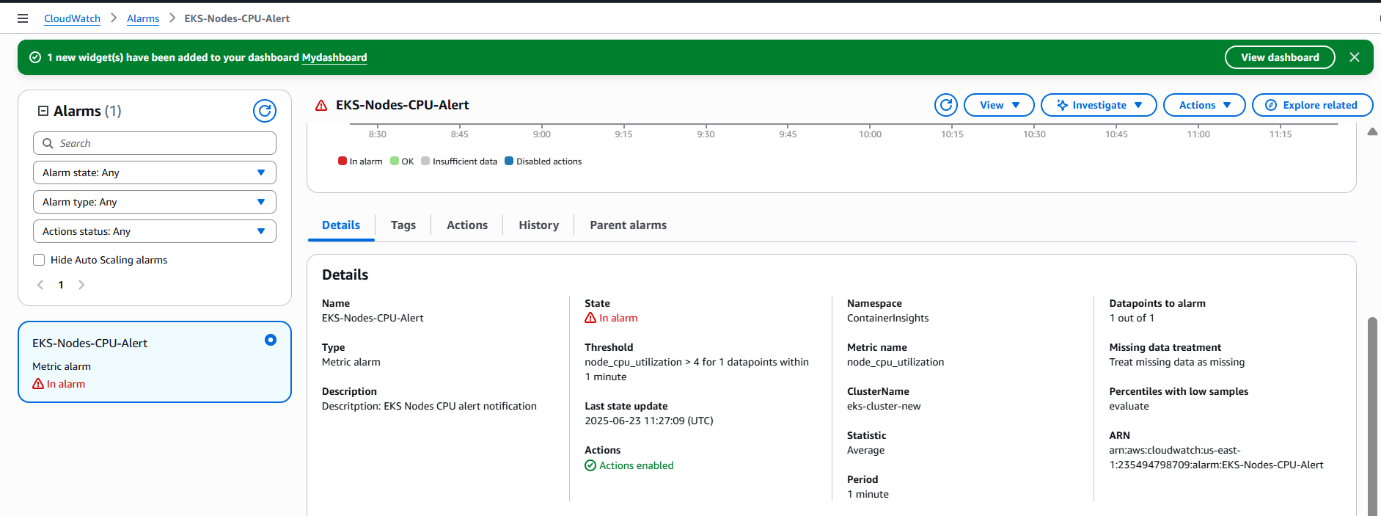




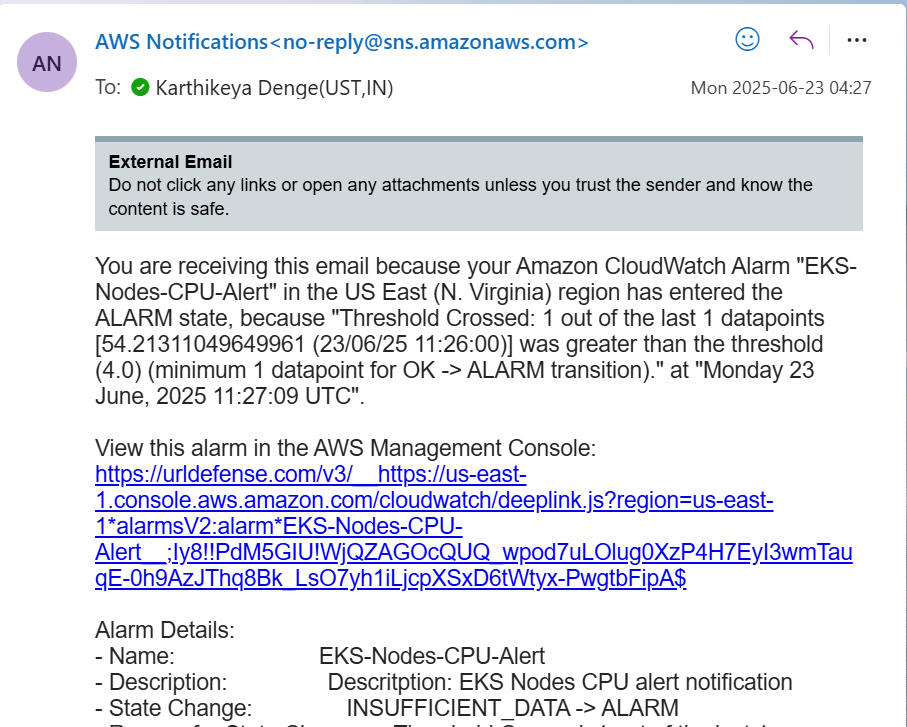
DASHBOARD



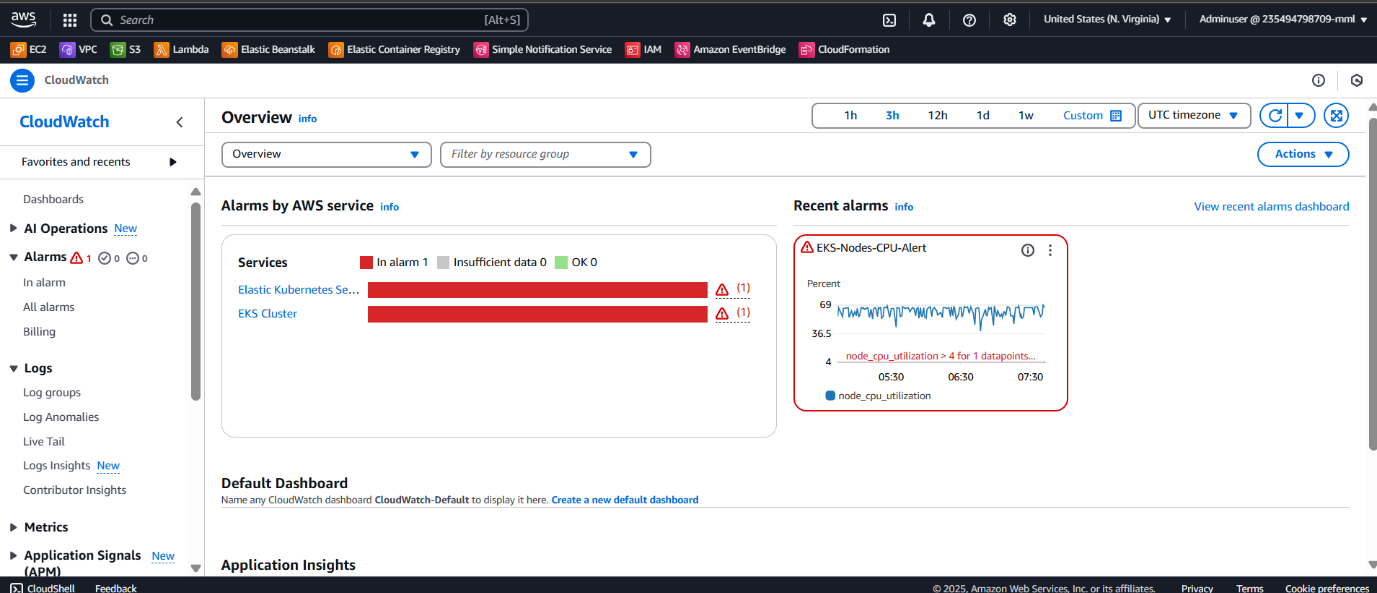
Alarm:

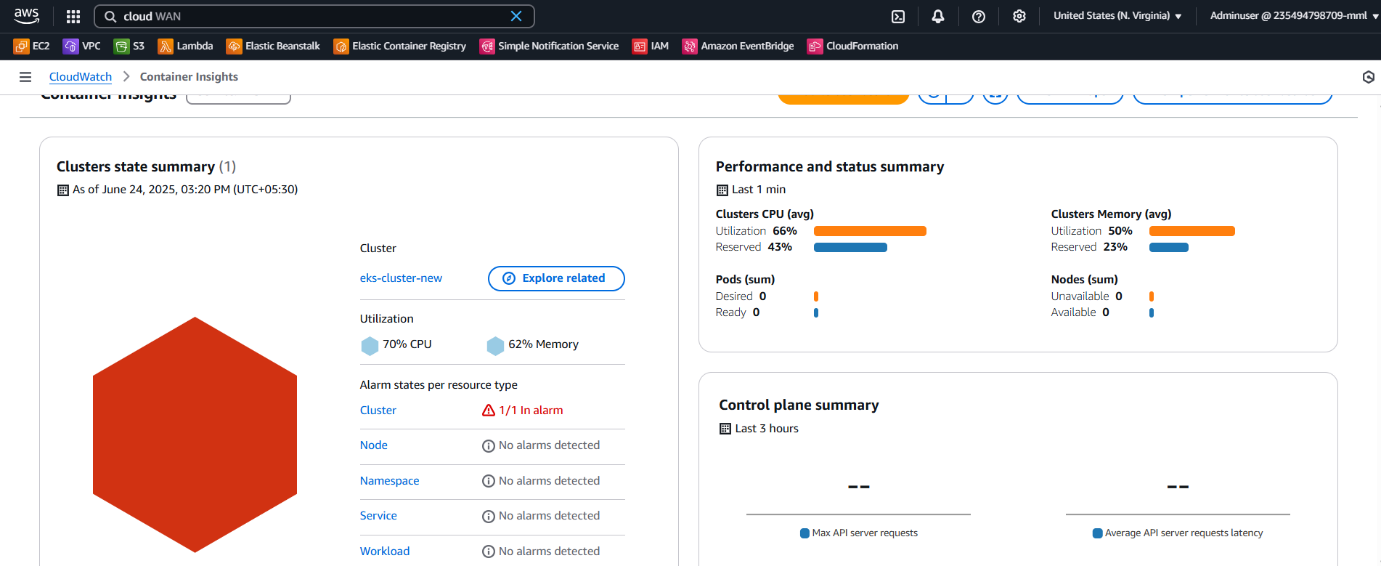


SNS Notification



Highusage:





**Prometheus + Grafana:**

* Installed via Helm:
* helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
* helm repo update

helm install kube-prom prometheus-community/kube-prometheus-stack

* Access Grafana via LoadBalancer
* Add Kubernetes and custom dashboards

📊 **Prometheus**:

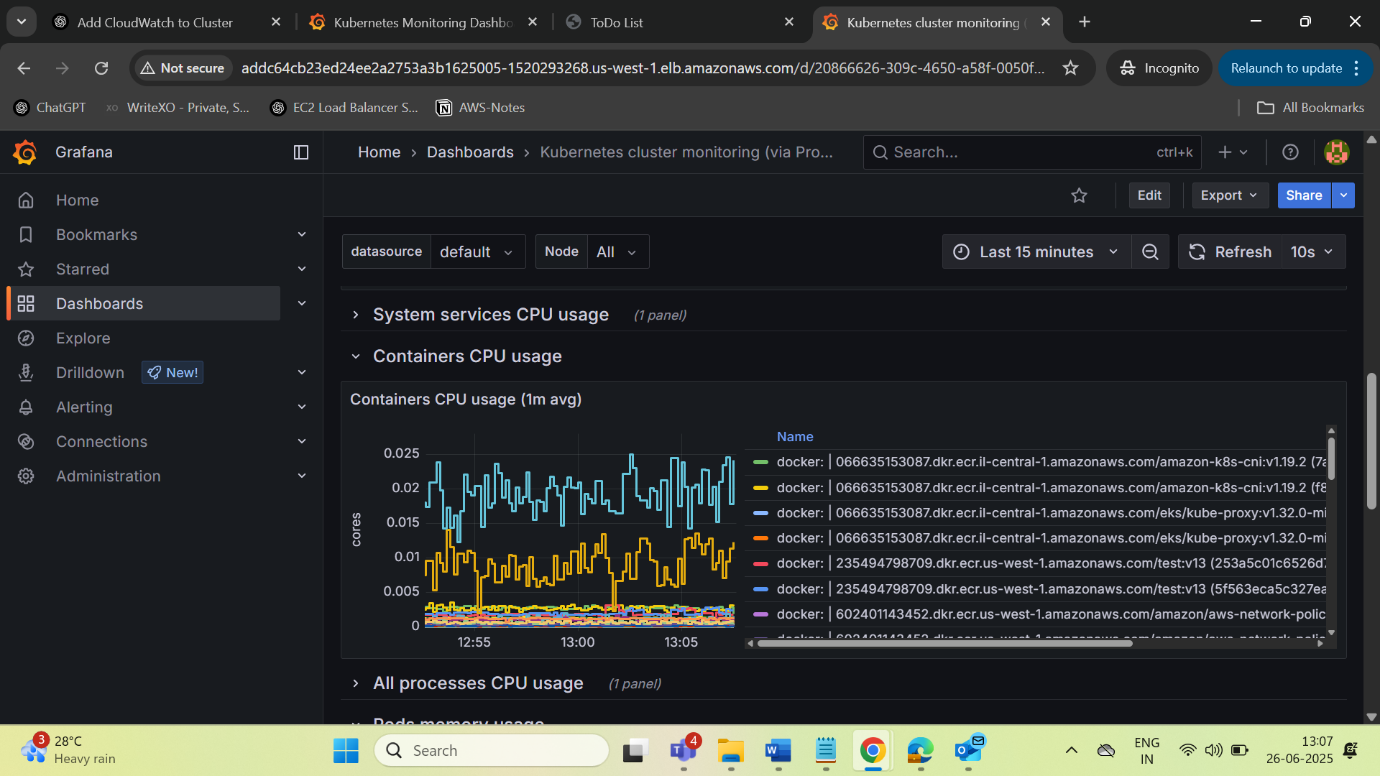
* Scrapes metrics from K8s, app pods

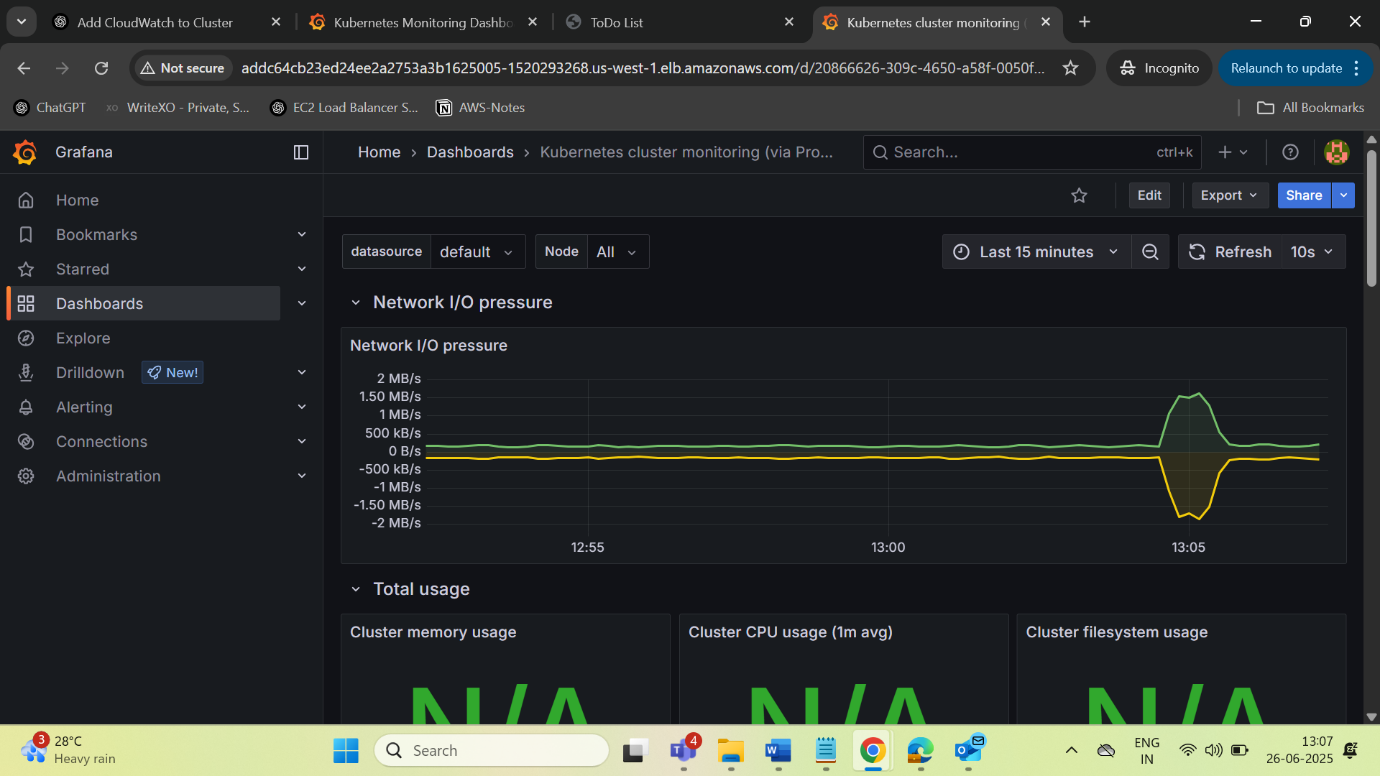
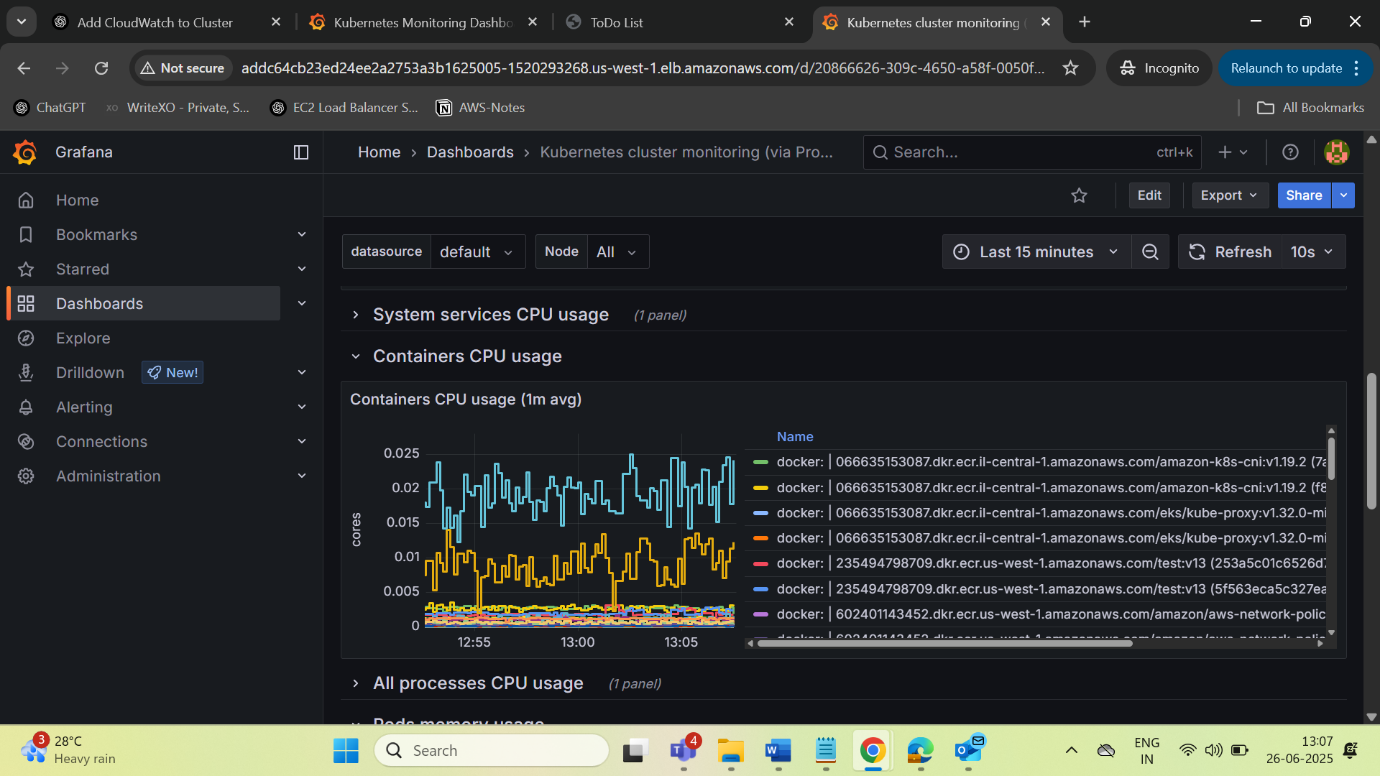
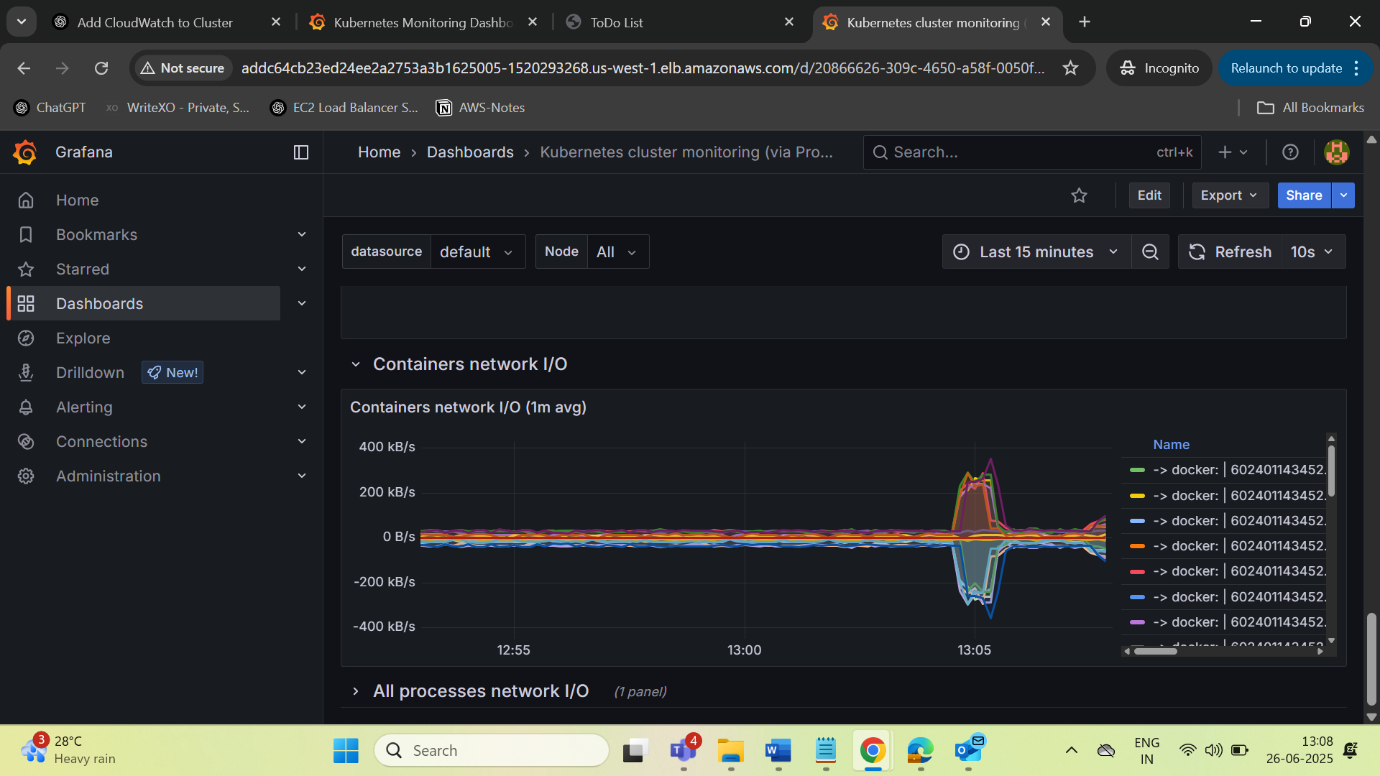
📈 **Grafana**:

* Dashboards for CPU, memory, latency, error rates

📡 **CloudWatch Alarms**:

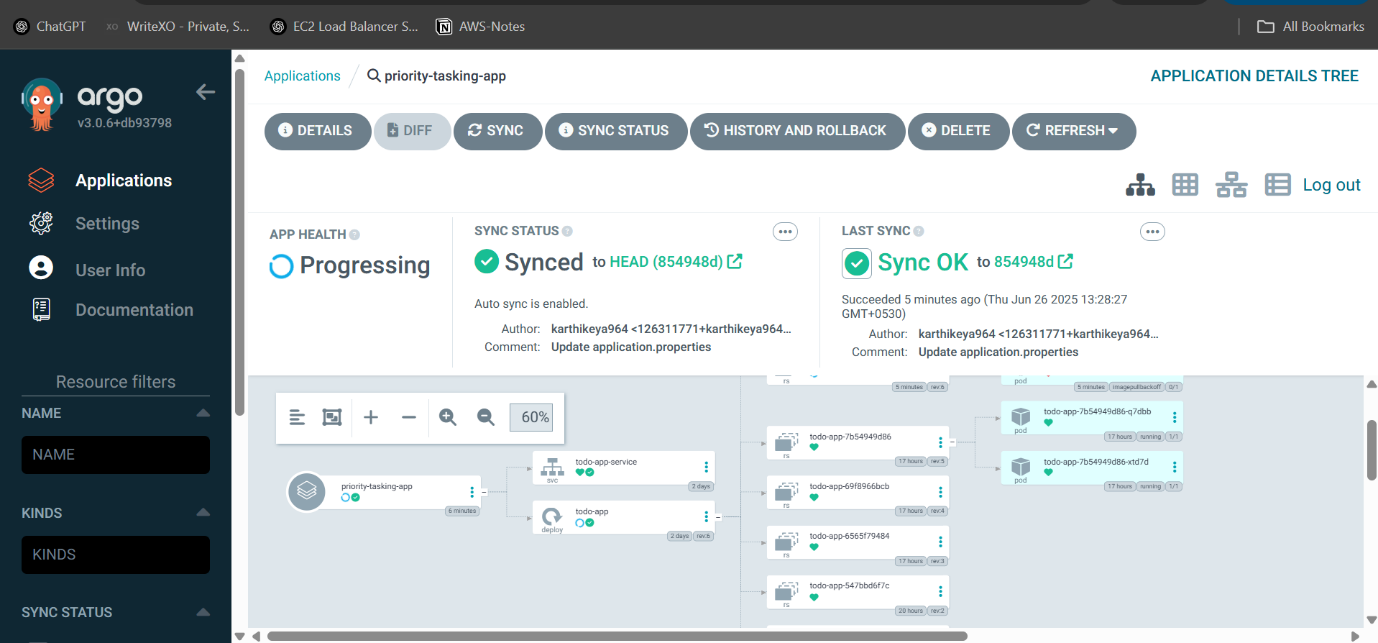
* RDS CPU > 70%
* EKS Pod CPU > 80%
* SNS Email Notifications for alerts



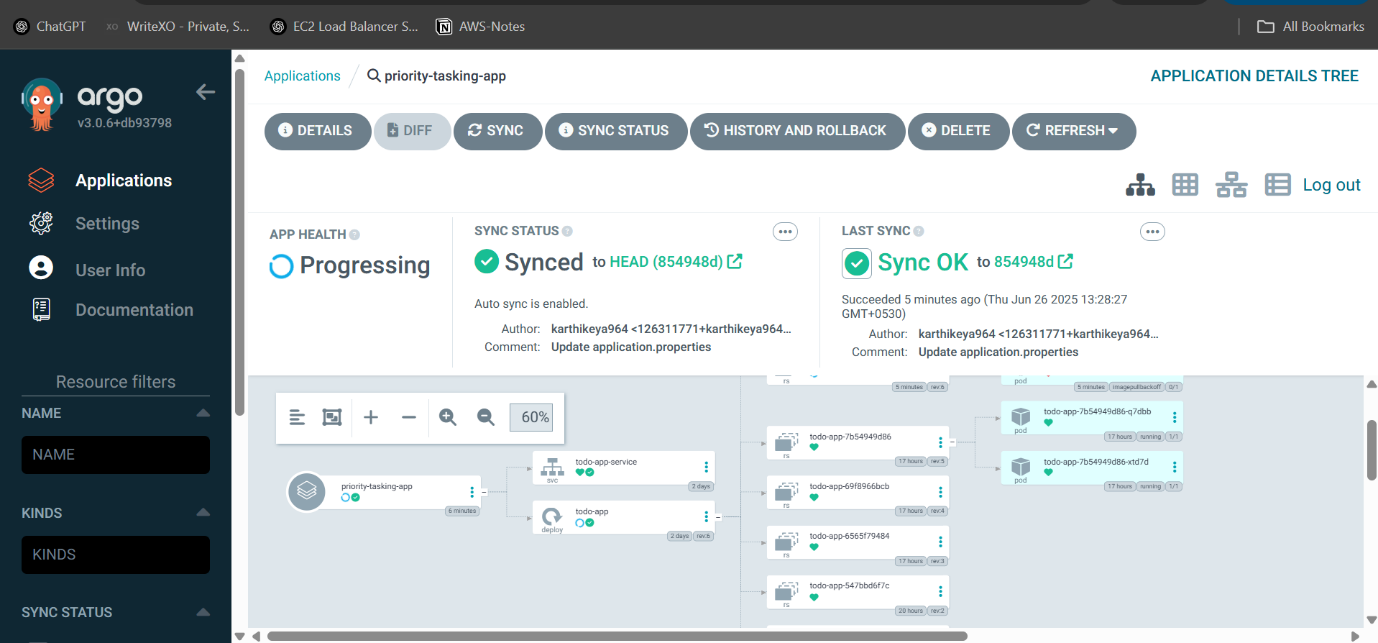
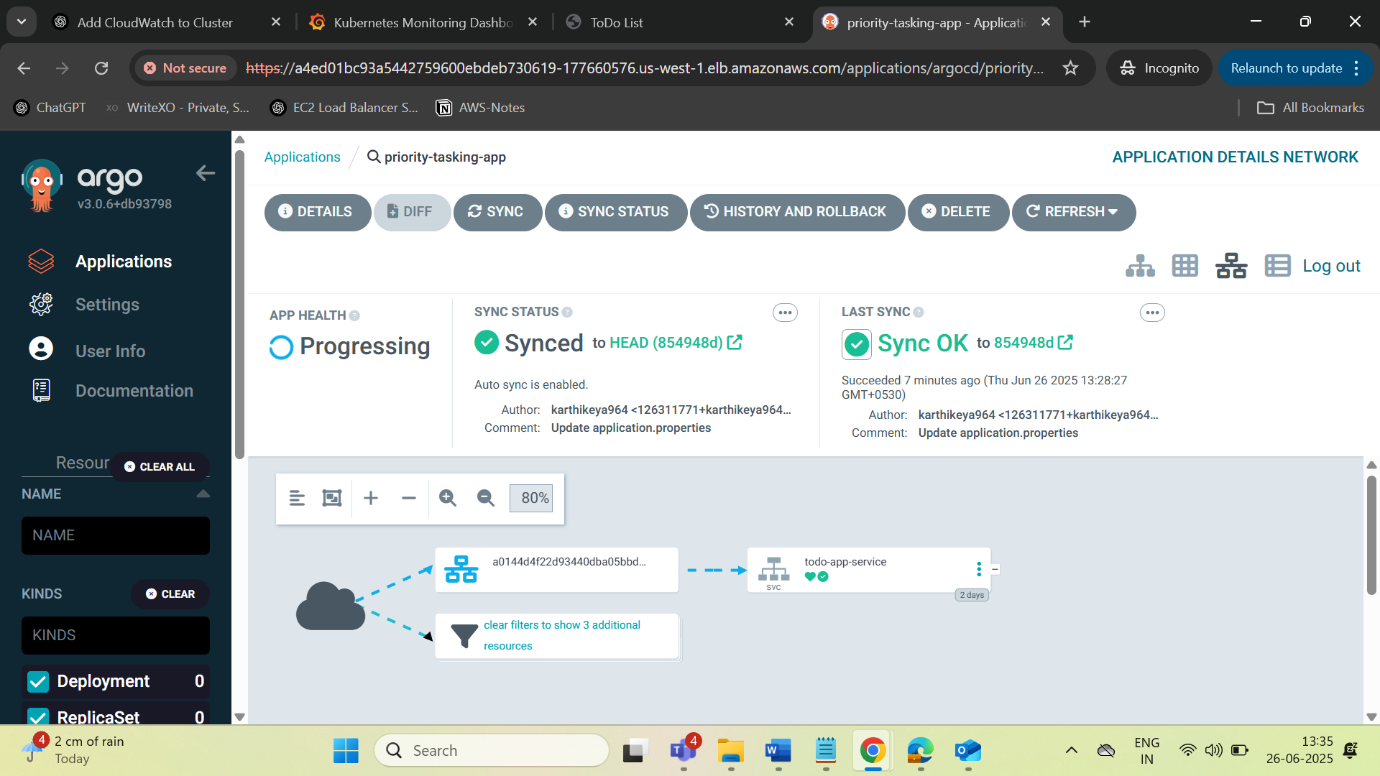


**PHASE 6: GitOps with Argo CD**

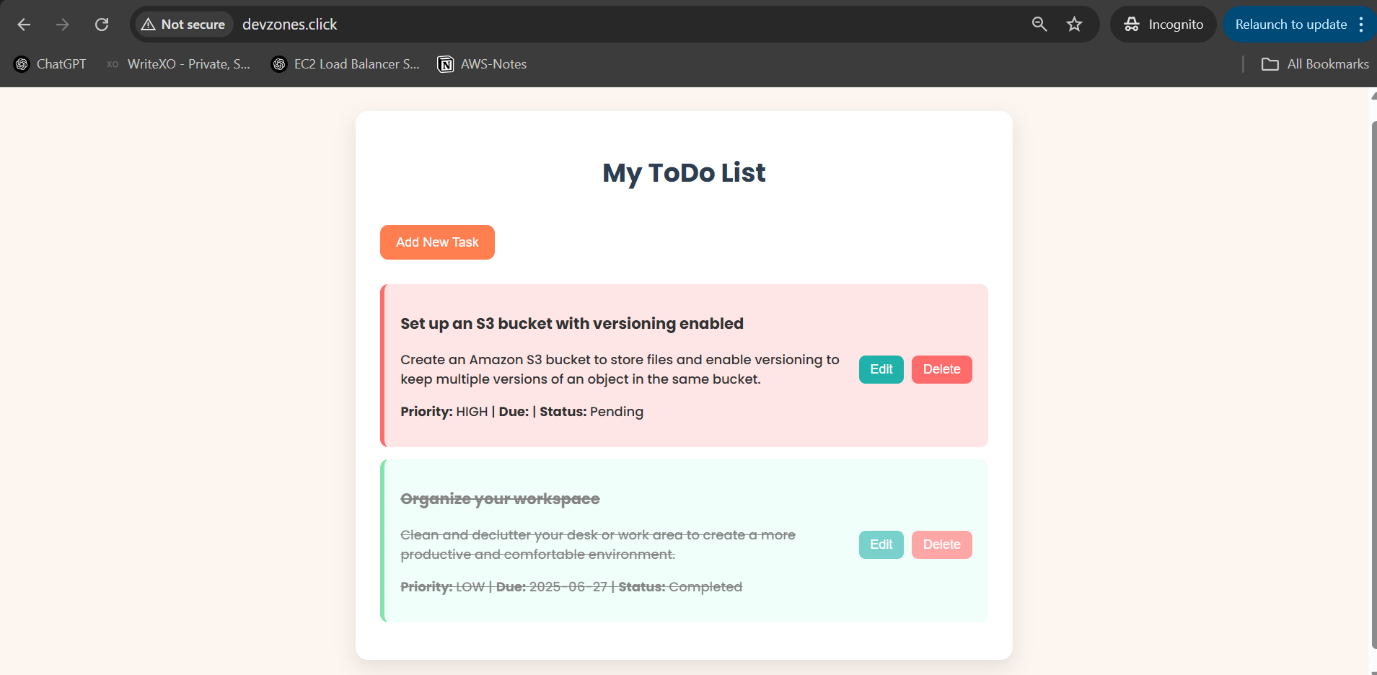
* **Argo CD setup on separate namespace**
* **Connect to GitHub repo**
* **Watch and auto-sync deployments**
* **Screenshots:**
  + **Argo CD Web UI**
  + **GitOps pipeline sync view**







**My APPLICATION:**

****

**Security Best Practices**

| **Category** | **Best Practice** |
| --- | --- |
| **RDS** | **Private Subnet, no public IP** |
| **IAM** | **Least privilege roles for EKS and CI/CD** |
| **Secrets** | **Stored securely using AWS Secrets Manager or SSM Parameter Store** |
| **Image Security** | **Trivy used to detect CVEs in Docker images before deployment** |
| **EKS Access** | **Controlled via aws-auth and RBAC** |
| **Network** | **Ingress restricted via Security Groups & NACLs** |

**Cost Optimization Measures**

| **Strategy** | **Details** |
| --- | --- |
| **Spot Instances** | **Optional for non-critical workloads** |
| **Auto-scaling** | **Enabled on EKS Node Groups** |
| **Delete Resources** | **Terraform destroy or stack deletion after demo** |
| **Budget Monitoring** | **AWS Budgets + Cost Explorer** |
| **Log Retention** | **Limited log retention in CloudWatch** |

**--**

**Next Steps & Enhancements Enable WAF and Shield for ALB Integrate Prometheus + Grafana for observability Extend pipeline with automated tests and canary deployments Add cost optimization monitoring (e.g., via AWS Cost Explorer)**

1. **Maintainer 👨‍💻 D. Karthikeya GitHub: @karthikeya964 Project: Capstone Final HV2 - AWS Multi-Region App**