**Capstone Final Project Report**

**Project Title: Event Dash Board**

**Cloud-Native Deployment and Monitoring of a Microservices Application using AWS EKS, CI/CD, Terraform, and Route 53**

**GitHub Repository:**

🔗 <https://github.com/sujeetsrahate/Capstone-Final-z>

**Table of Contents**

1. Introduction
2. Objective
3. Tools and Technologies Used
4. Architecture Overview
5. Phase 1: EKS Cluster Deployment
6. Phase 2: CI/CD Pipeline Deployment
7. Phase 3: Multi-Region Infra Setup using CloudFormation & Terraform
8. Terraform Infra Structure
9. Phase 4: Route 53 Restructuring (Disaster Recovery)
10. Phase 5: Monitoring with CloudWatch
11. SonarQube Integration
12. Challenges Faced
13. Future Scope
14. Conclusion
15. Appendix (YAML & Terraform Snippets)

**1. Introduction**

The goal of this capstone project was to design and deploy a full-stack cloud-native microservices application using AWS managed services and Infrastructure as Code (IaC) tools. This project integrates EKS, Terraform, AWS CodePipeline, Route 53, and CloudWatch to provide a scalable, highly available, and observable solution.

**2. Objective**

To implement a complete DevOps workflow with multi-region disaster recovery, monitoring, and CI/CD capabilities by:

* Deploying Java Spring Boot & ReactJS applications.
* Automating infrastructure using CloudFormation and Terraform.
* Implementing automated pipelines using CodePipeline.
* Configuring Route 53 for disaster recovery.
* Setting up CloudWatch for monitoring and logging.

**3. Tools and Technologies Used**

* **AWS Services**: EKS, EC2, RDS, CloudWatch, Route 53, CodePipeline, S3, IAM
* **IaC**: Terraform, AWS CloudFormation
* **CI/CD**: AWS CodePipeline, CodeBuild
* **Containerization**: Docker
* **Monitoring**: CloudWatch
* **Languages**: Java (Spring Boot), ReactJS, YAML, HCL

**4. Architecture Overview**

The project implements a **three-tier microservice architecture**:

* **Frontend**: ReactJS
* **Backend**: Spring Boot + MySQL (RDS)
* **Infrastructure**: Provisioned across regions using CloudFormation and Terraform
* **Pipeline**: Source -> Build -> Deploy stages in AWS CodePipeline

**5. Phase 1: Deploy High-Level Application on EKS**

The initial phase involves deploying the application on Amazon EKS. Components:

* EKS Cluster creation (via AWS CLI)
* Kubernetes manifests for:
  + Spring Boot deployment & service
  + MySQL deployment & persistent volume
  + ReactJS frontend deployment & service
  + Ingress controller

Manifests used:

* springboot-deploy.yaml
* mysql-deploy.yaml
* react-event-deploy.yaml
* ingress.yaml

**6. Phase 2: CI/CD Pipeline with CodePipeline**

This phase integrates CI/CD via AWS CodePipeline.

**Key Files:**

* buildspec.yaml  
  Specifies how CodeBuild builds the project and pushes Docker images to ECR.

**Pipeline Flow:**

1. Code push to GitHub
2. CodeBuild builds and pushes to ECR
3. Deploys to EKS using kubectl

**7. Phase 3: Multi-Region Infra via CloudFormation & Terraform**

This phase establishes high availability and disaster recovery.

* CloudFormation stacks create VPC, subnets, security groups, etc.
* Terraform modules provision additional resources and EKS nodes.

**Repositories:**

* [CloudFormation.yaml](https://github.com/sujeetsrahate/Capstone-Final-z/blob/main/CloudFormation.yaml)
* [Infra-terraform.tf](https://github.com/sujeetsrahate/Capstone-Final-z/blob/main/Infra-terraform.tf)

**8. Terraform Infra Structure**

Additional infrastructure is provisioned using Terraform:

* VPC and subnets
* EKS cluster and worker node groups
* Route tables and gateways
* S3 and IAM roles

**GitHub Repo**: sujeetsrahate/terraform\_infra

**9. Phase 4: Route 53 Restructuring (Disaster Recovery)**

Route 53 is configured with:

* **Primary hosted zone** for active region
* **Secondary failover policy** for standby region
* Health checks on application endpoints
* Weighted routing policies for zero-downtime switch

**10. Phase 5: Monitoring with CloudWatch**

CloudWatch is configured to:

* Collect logs from EKS using FluentBit
* Visualize metrics like CPU/Memory usage
* Trigger alarms for unhealthy pods or services
* Store logs from Spring Boot backend and application frontend

**Importance:**

* Ensures observability
* Enables root cause analysis
* Supports performance tuning

**11. SonarQube Integration**

SonarQube was added as a quality gate in the CI/CD pipeline.

* Scans for bugs, vulnerabilities, and code smells
* Improves maintainability and security
* Integrated in CodeBuild using Dockerized scanner

**12. Challenges Faced**

* Managing VPC peering across regions
* IAM role policies during automated provisioning
* Synchronizing Terraform and CloudFormation resources
* Kubernetes networking in multi-region setup
* Log ingestion and visibility in CloudWatch

**13. Future Scope**

* Use AWS Backup to automate RDS backups
* Integrate Prometheus & Grafana for advanced monitoring
* Add more microservices for horizontal scaling
* Integrate with GitHub Actions or Jenkins for hybrid CI/CD

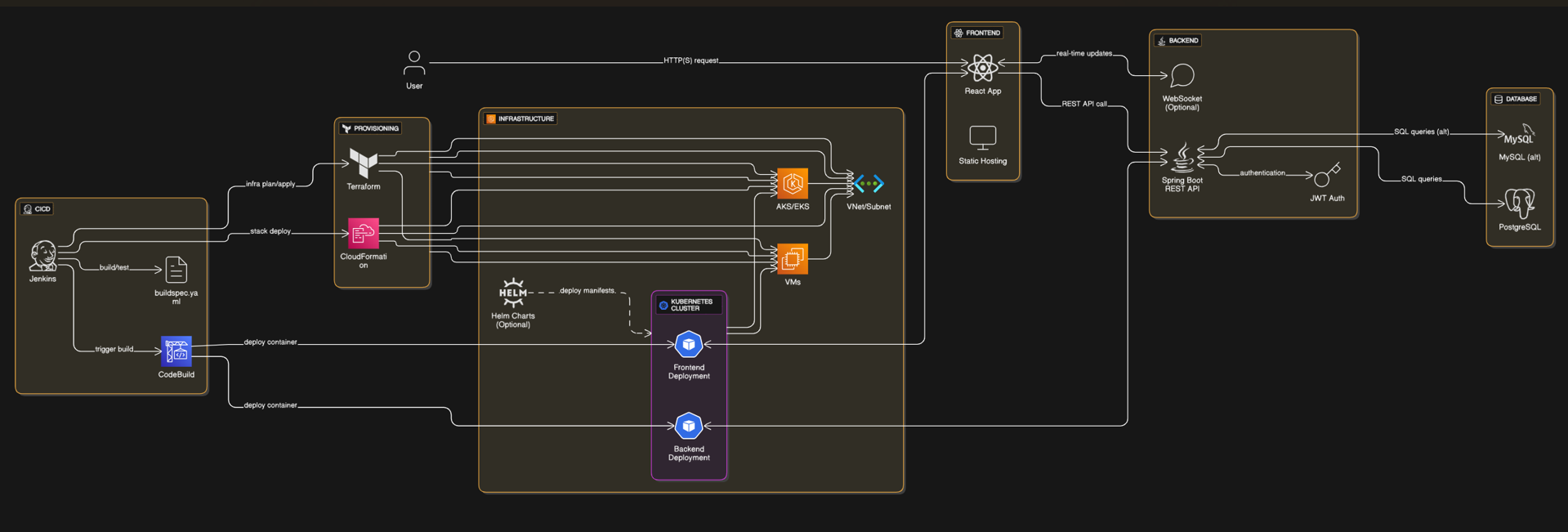
**14. Conclusion**

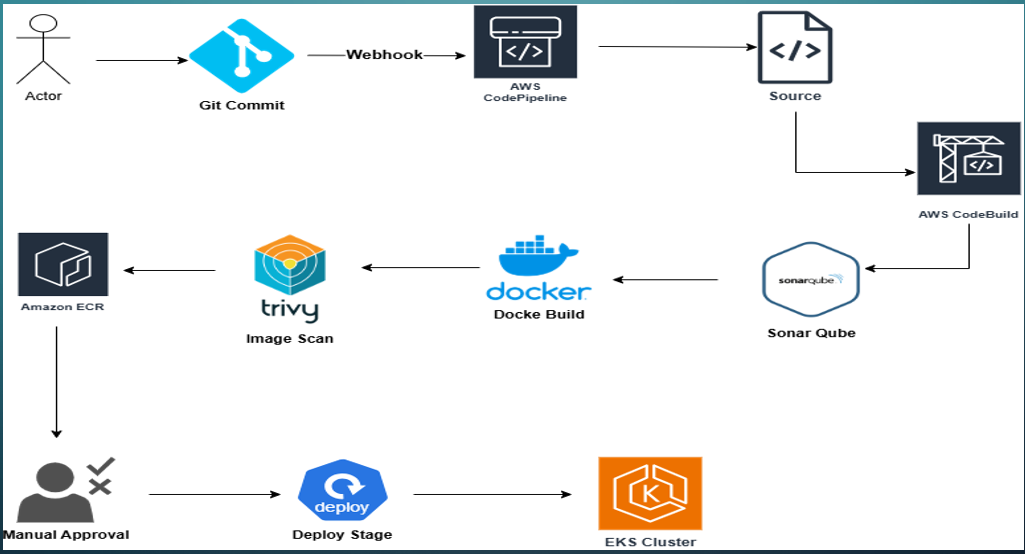
This capstone project successfully delivered a multi-region, production-grade cloud-native application using industry-standard DevOps tools and AWS services. All core objectives including CI/CD, container orchestration, disaster recovery, and monitoring were achieved.

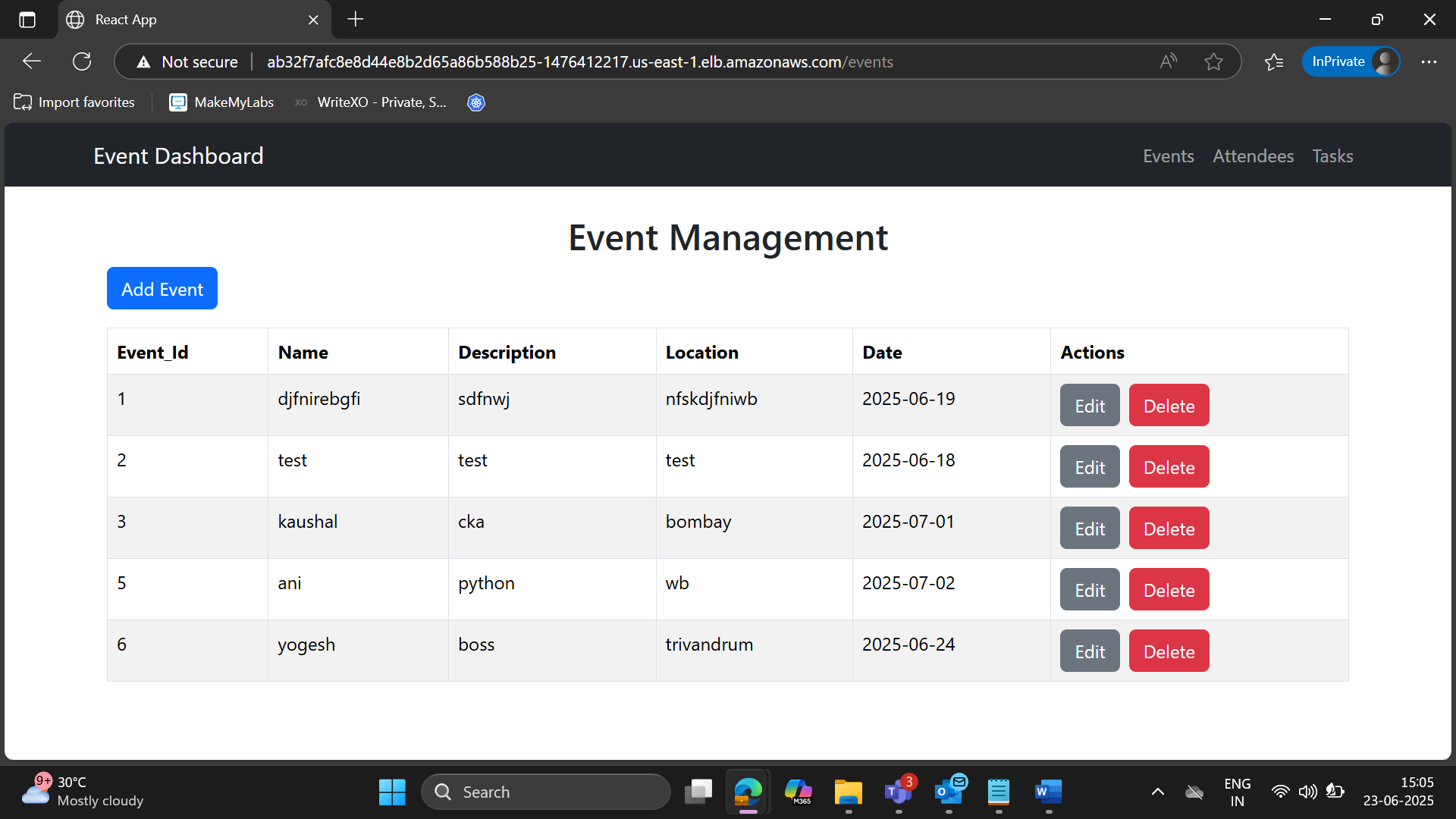
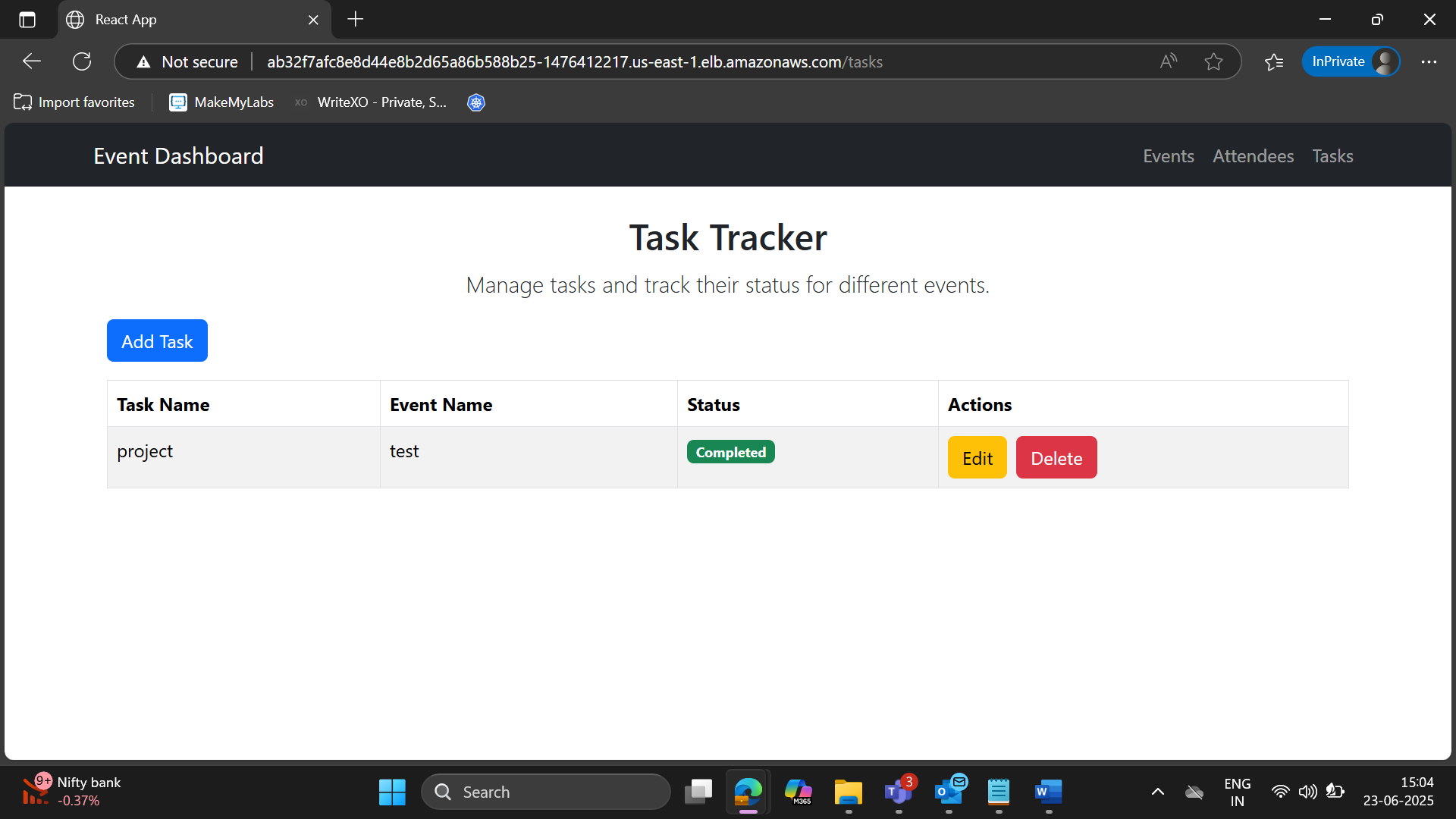
Capstone Project:

GitHub Repository: [sujeetsrahate/Capstone-Final-z](https://github.com/sujeetsrahate/Capstone-Final-z)

Phase 1: Deploy high level application on EKS Cluster

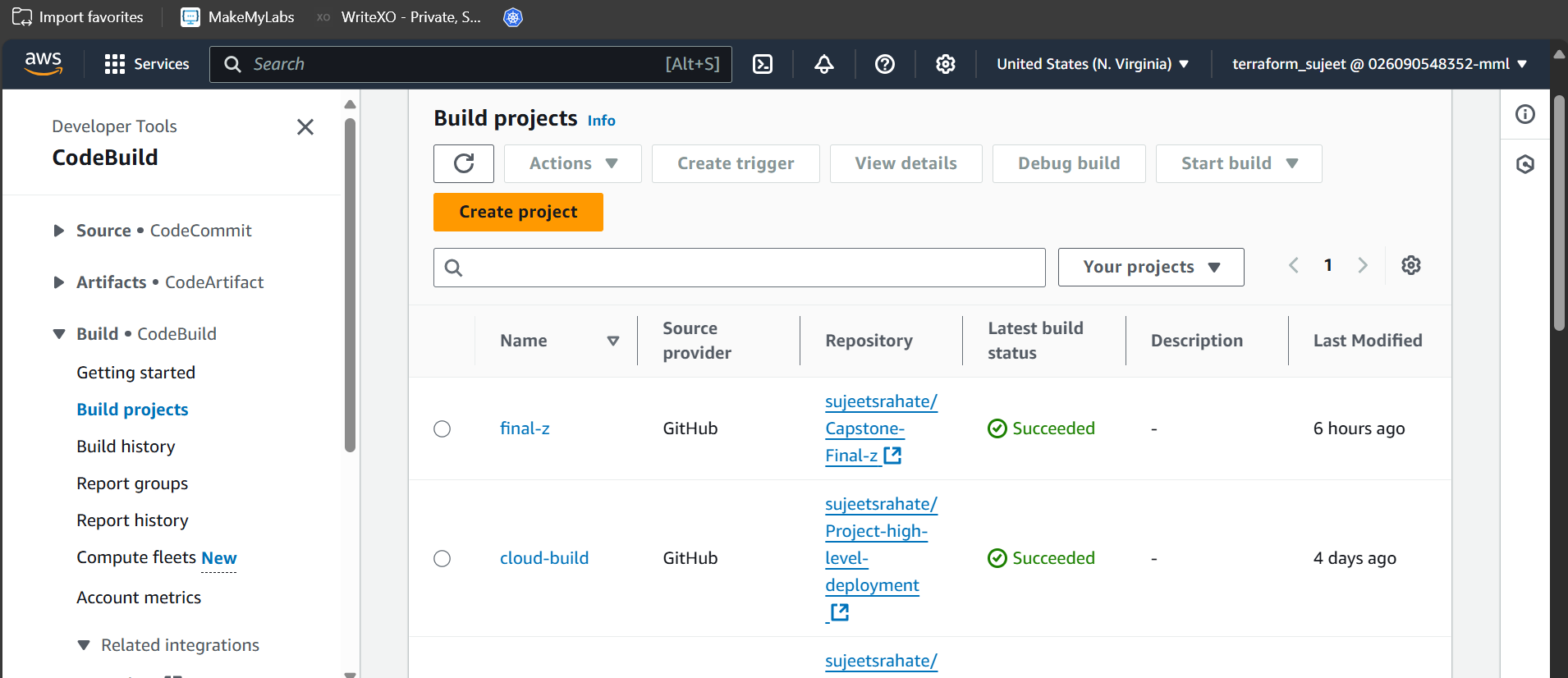


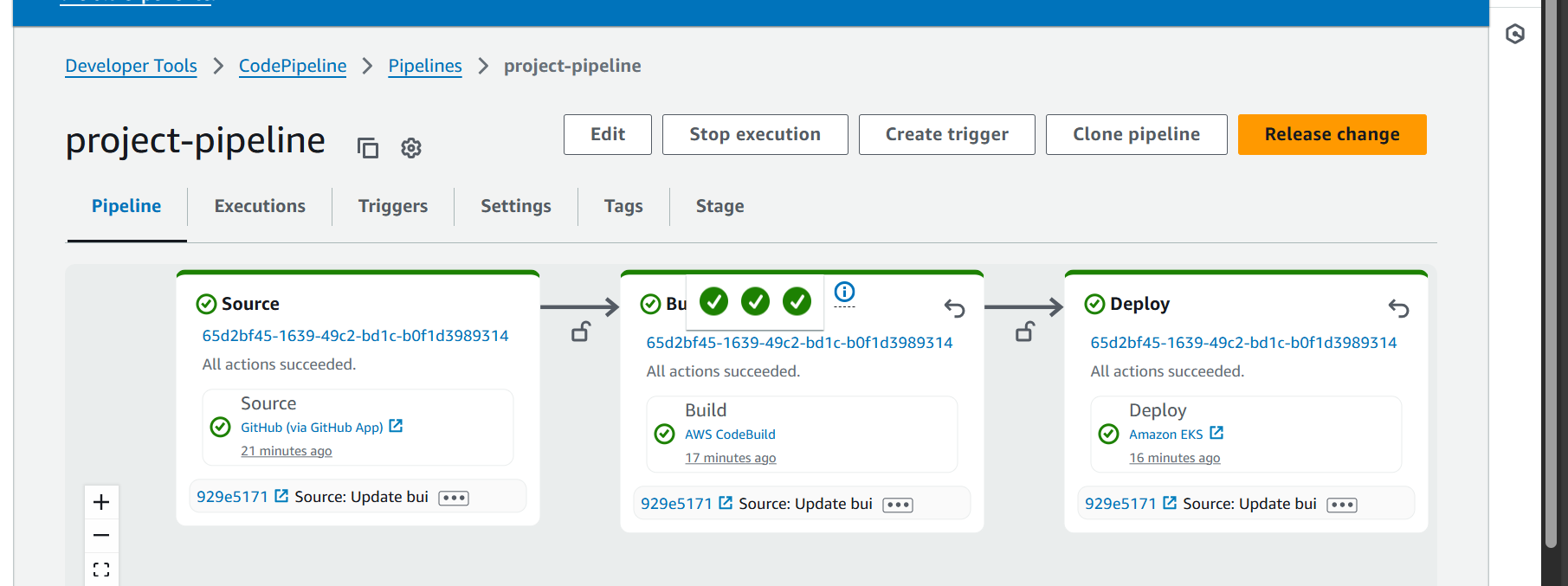




Phase 2: deploy Application Through pipeline

Git Repo(buildspec.yaml): [Capstone-Final-z/buildspec.yaml at main · sujeetsrahate/Capstone-Final-z](https://github.com/sujeetsrahate/Capstone-Final-z/blob/main/buildspec.yaml)

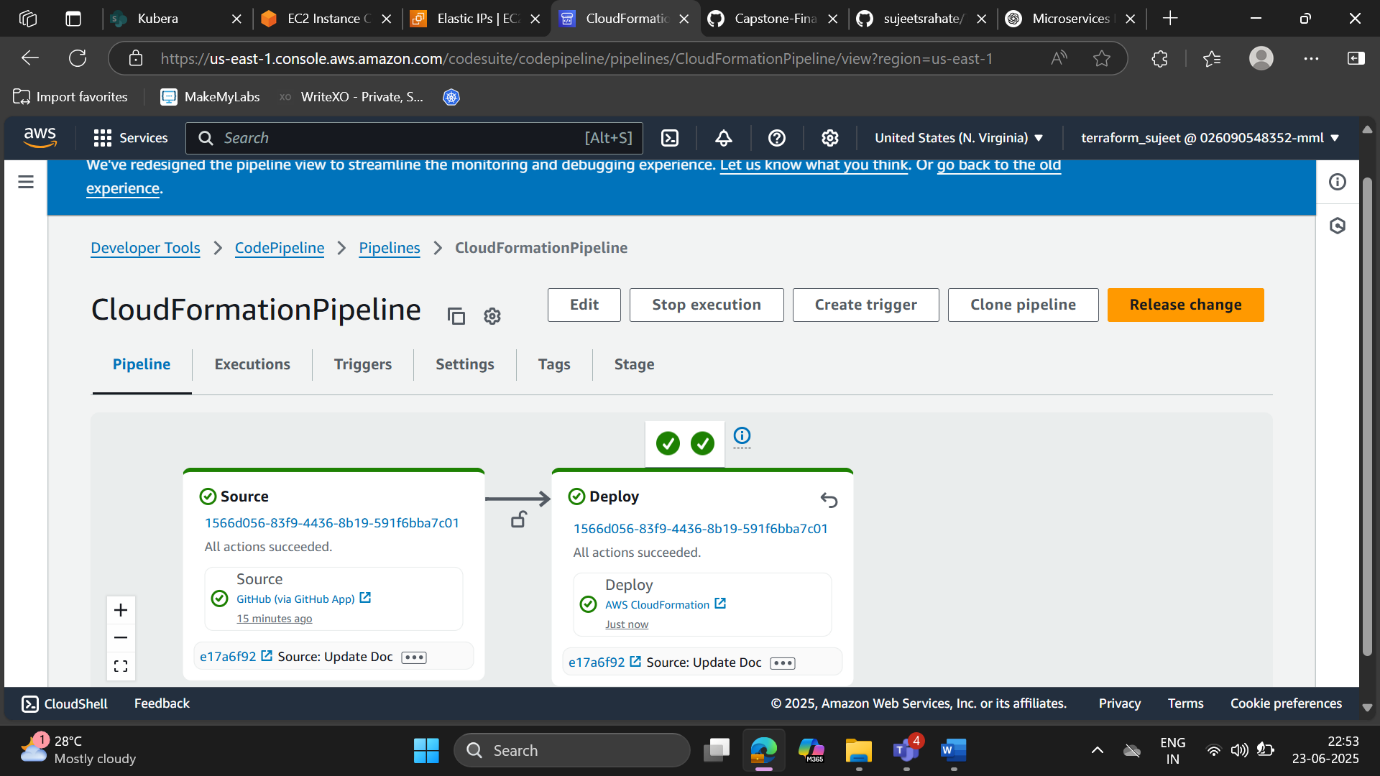


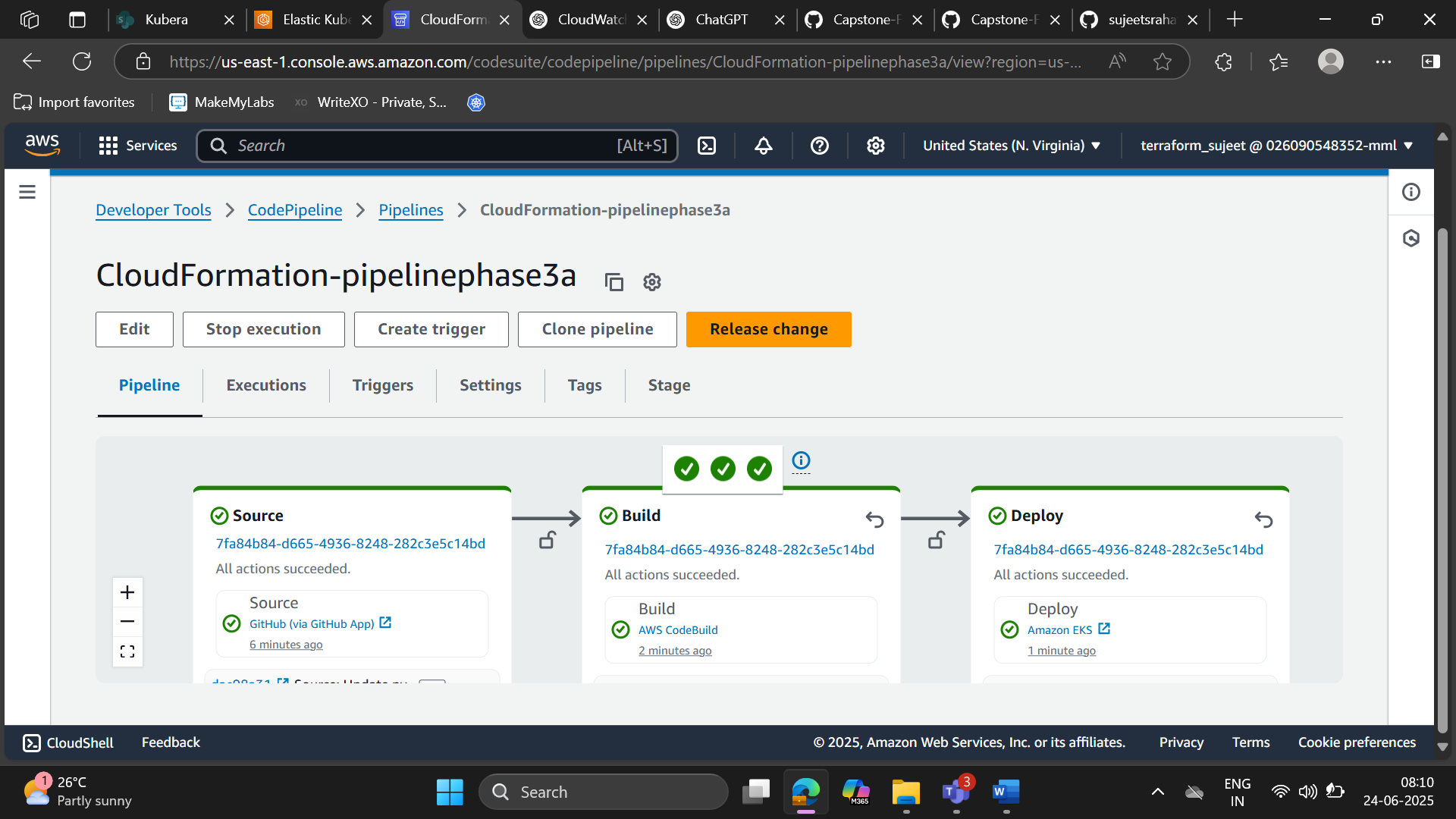


Phase 3: Use AWS Auto pipeline Multi Region of infrastructure with CFT and Terraform.

Git Hub Repo: [Capstone-Final-z/CloudFormation.yaml at main · sujeetsrahate/Capstone-Final-z](https://github.com/sujeetsrahate/Capstone-Final-z/blob/main/CloudFormation.yaml)

Terraform GitHubRepo: [Capstone-Final-z/Infra-terraform.tf at main · sujeetsrahate/Capstone-Final-z](https://github.com/sujeetsrahate/Capstone-Final-z/blob/main/Infra-terraform.tf)

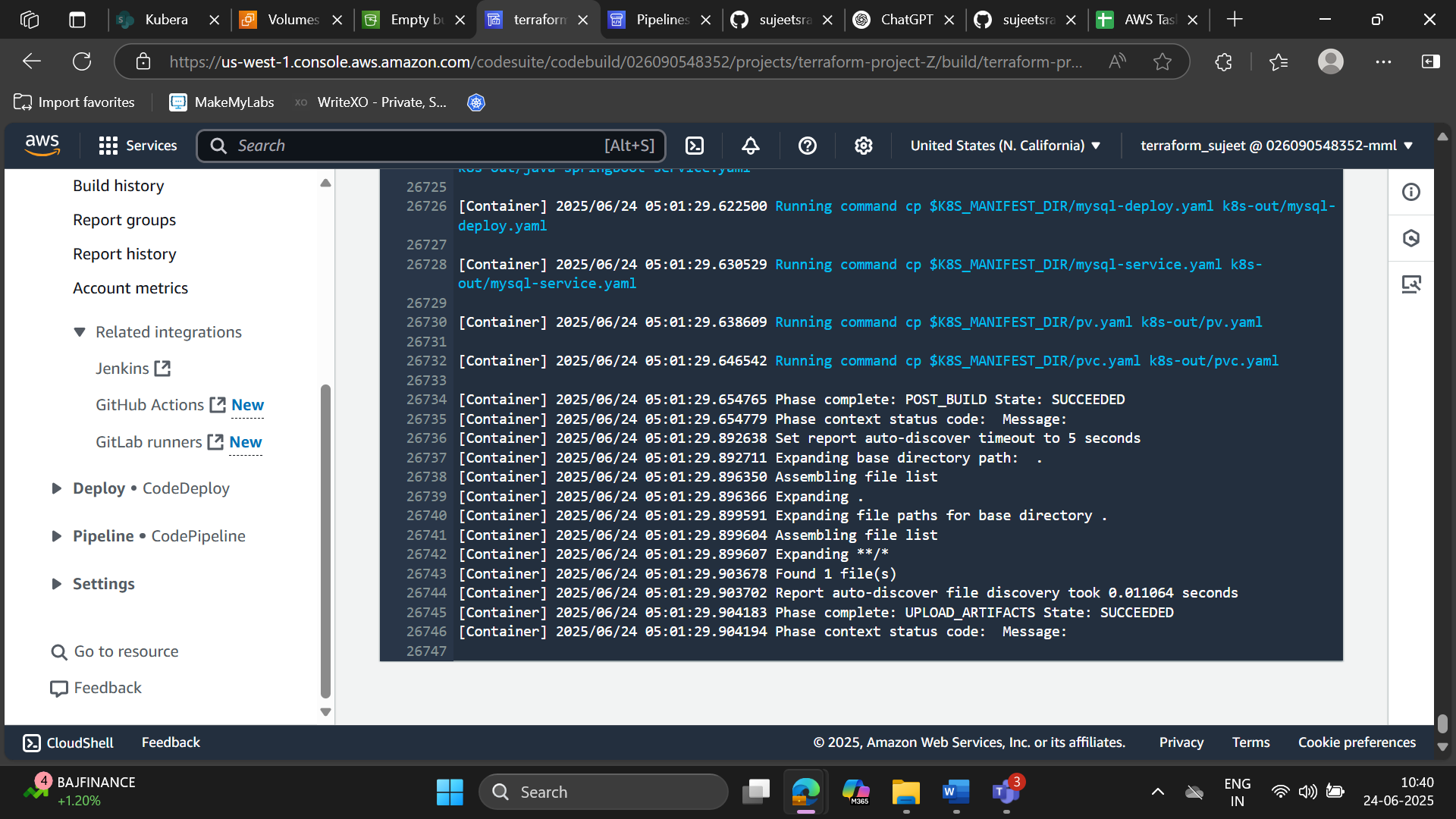


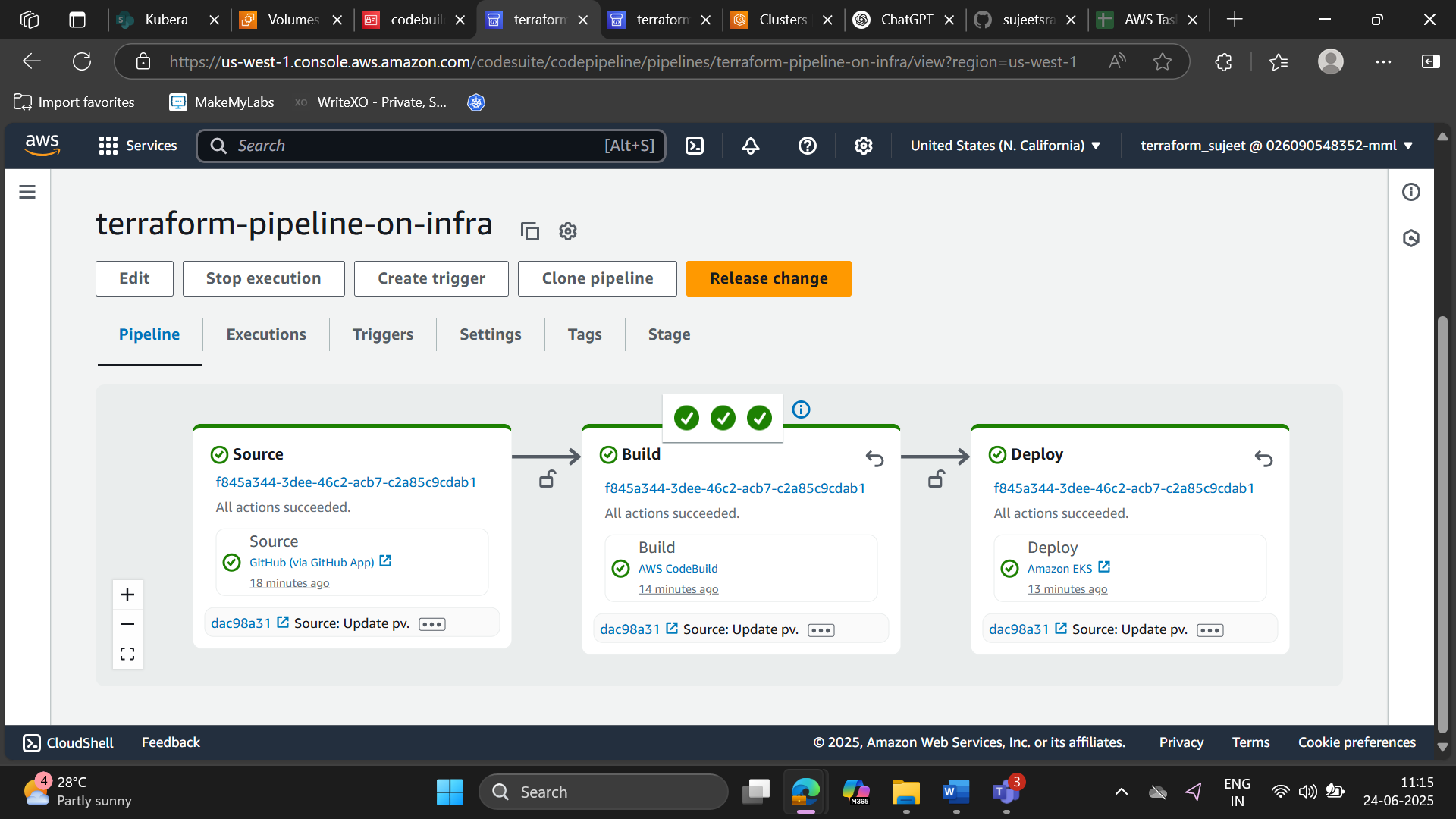


CloudFormation working proper and make the vpc and everything

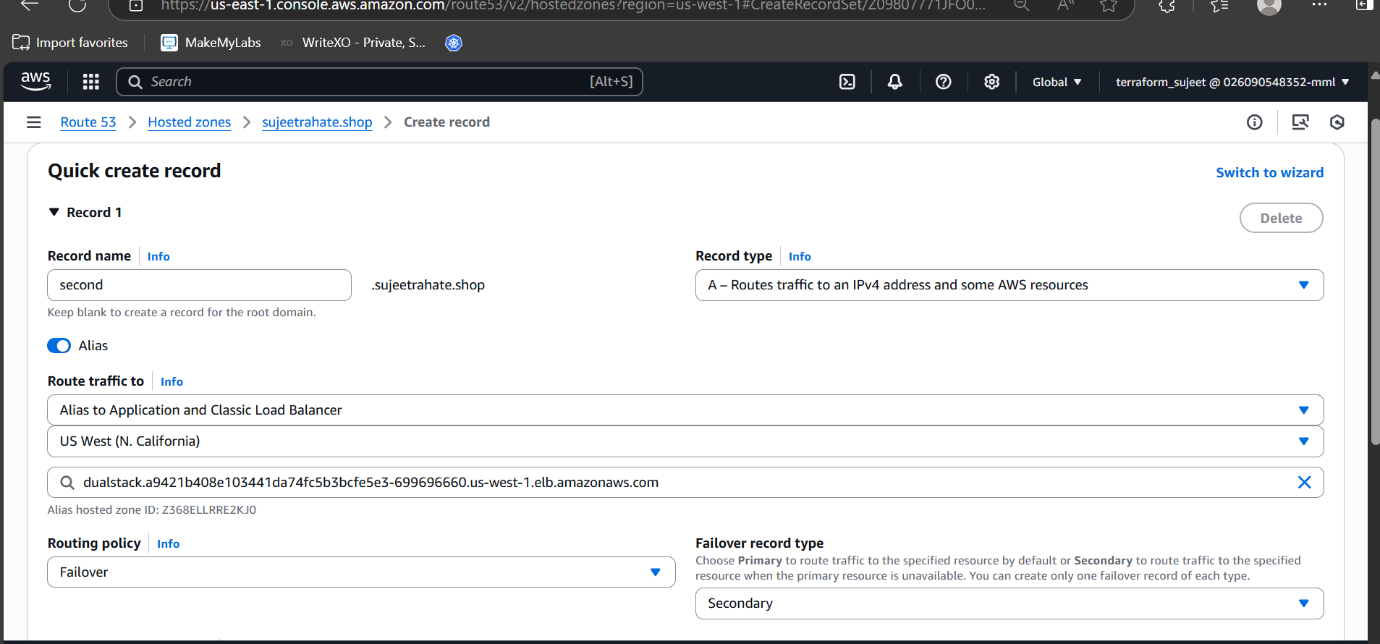
3B: creating Infra Structure through Terraform

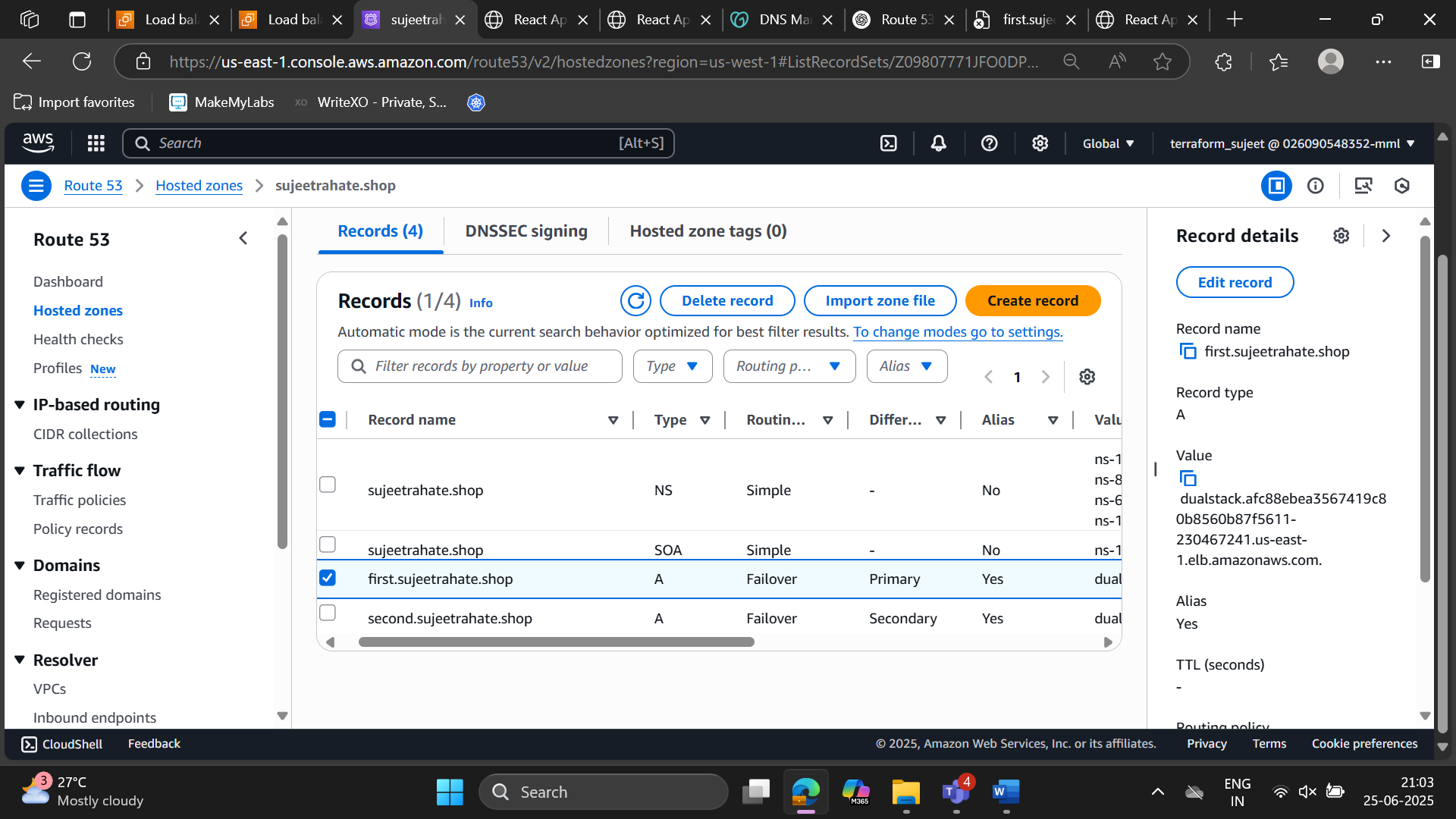
Git Hub Repo : [sujeetsrahate/terraform\_infra](https://github.com/sujeetsrahate/terraform_infra)



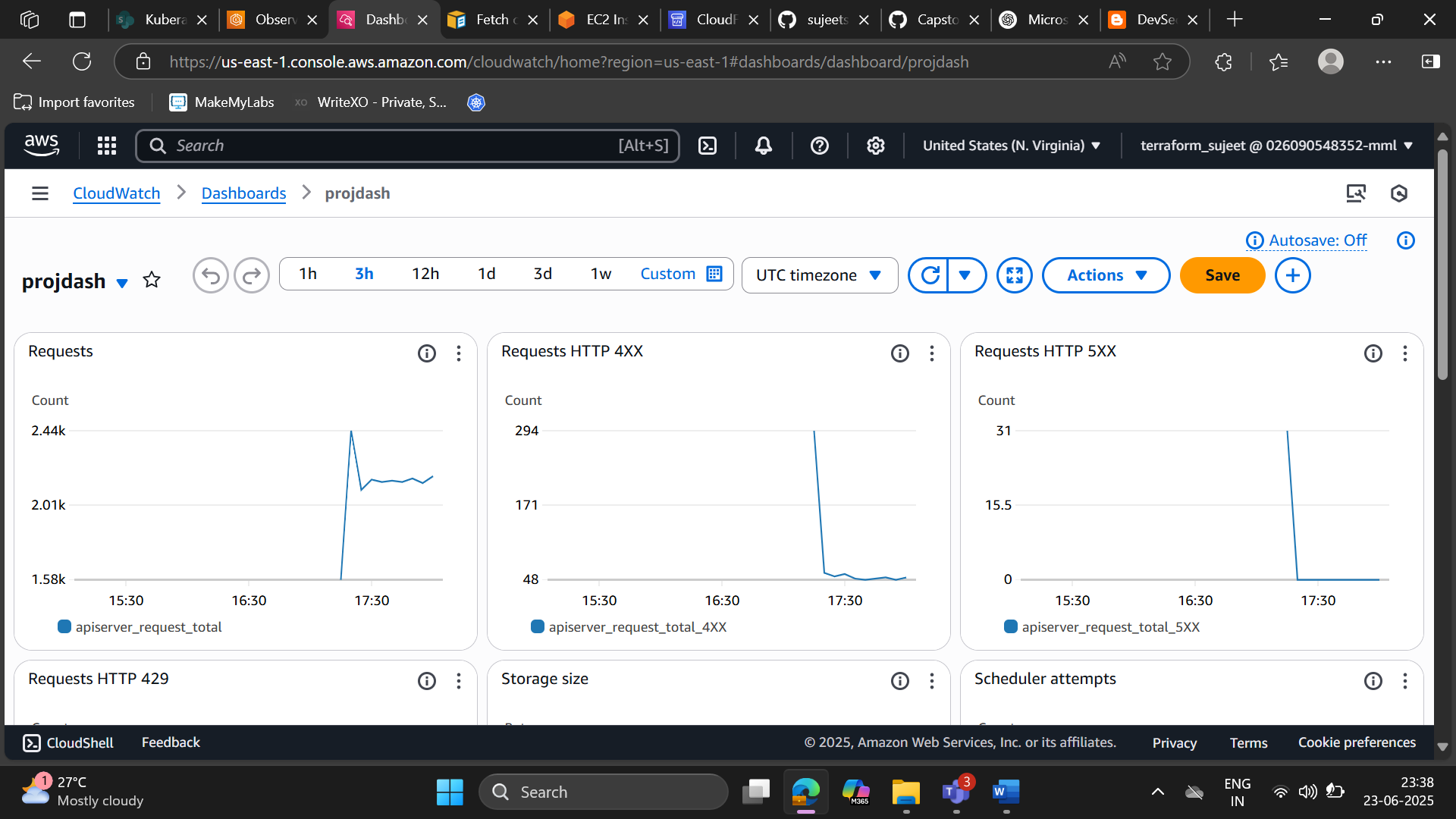


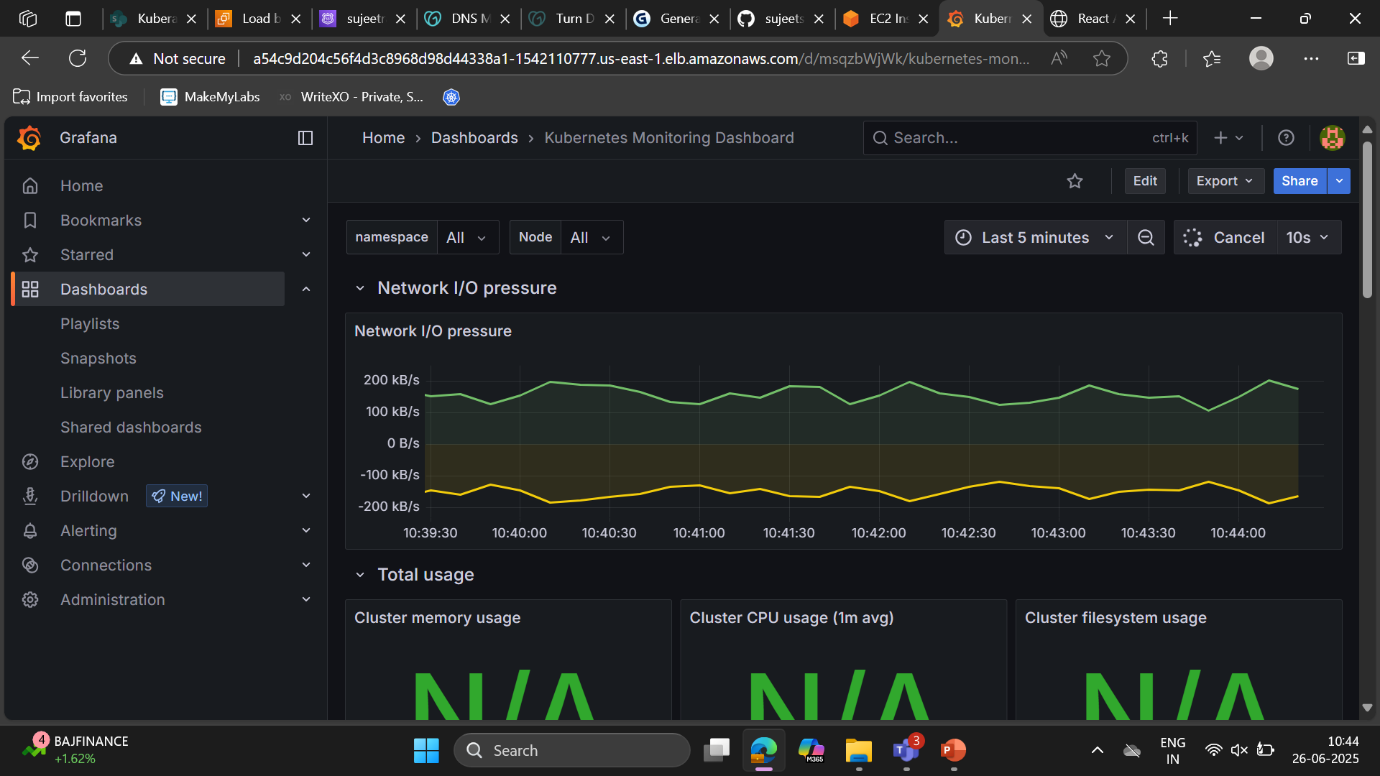
Phase 4: Make Route 53 Restructure. (make disaster strategy)

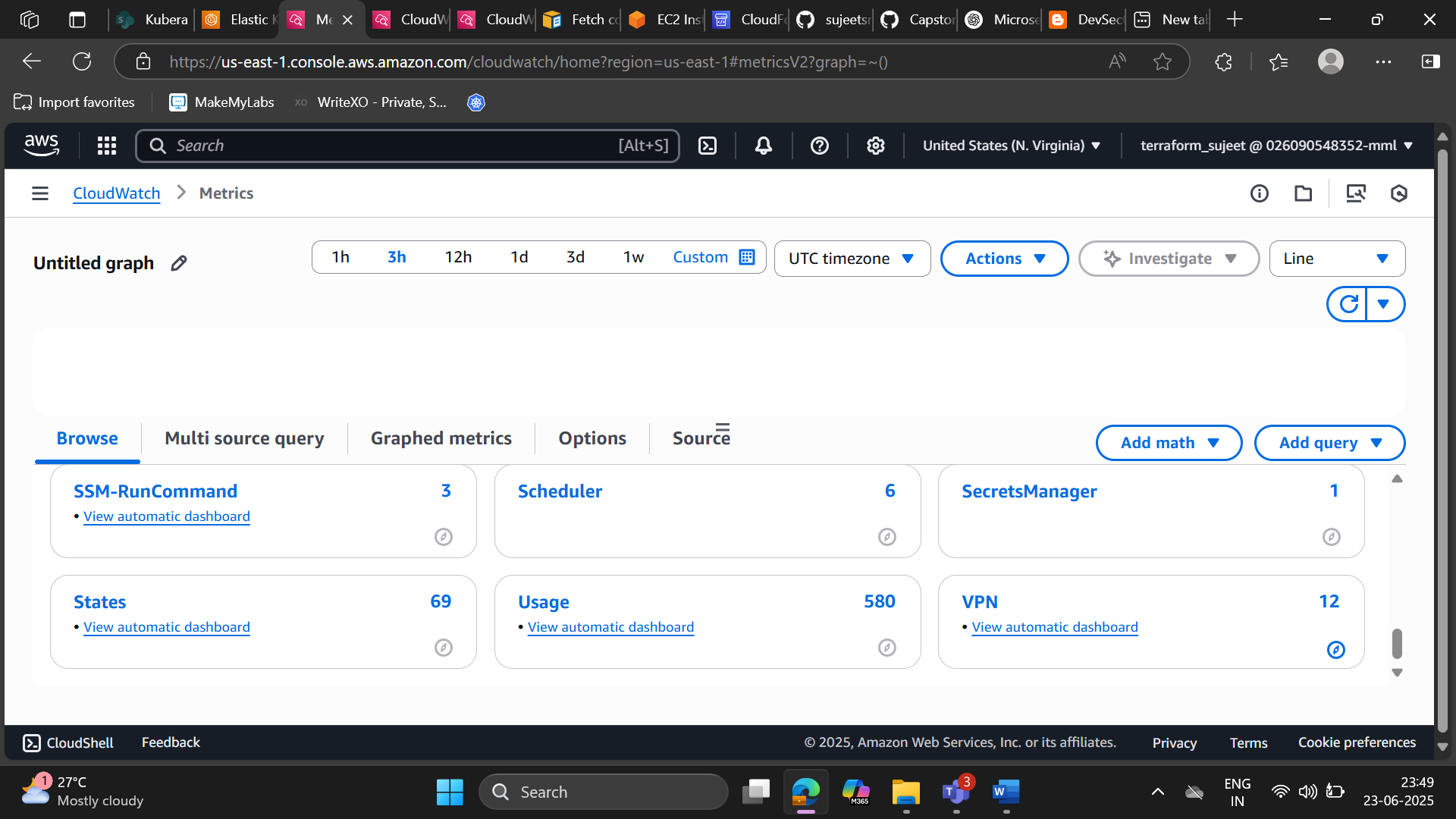


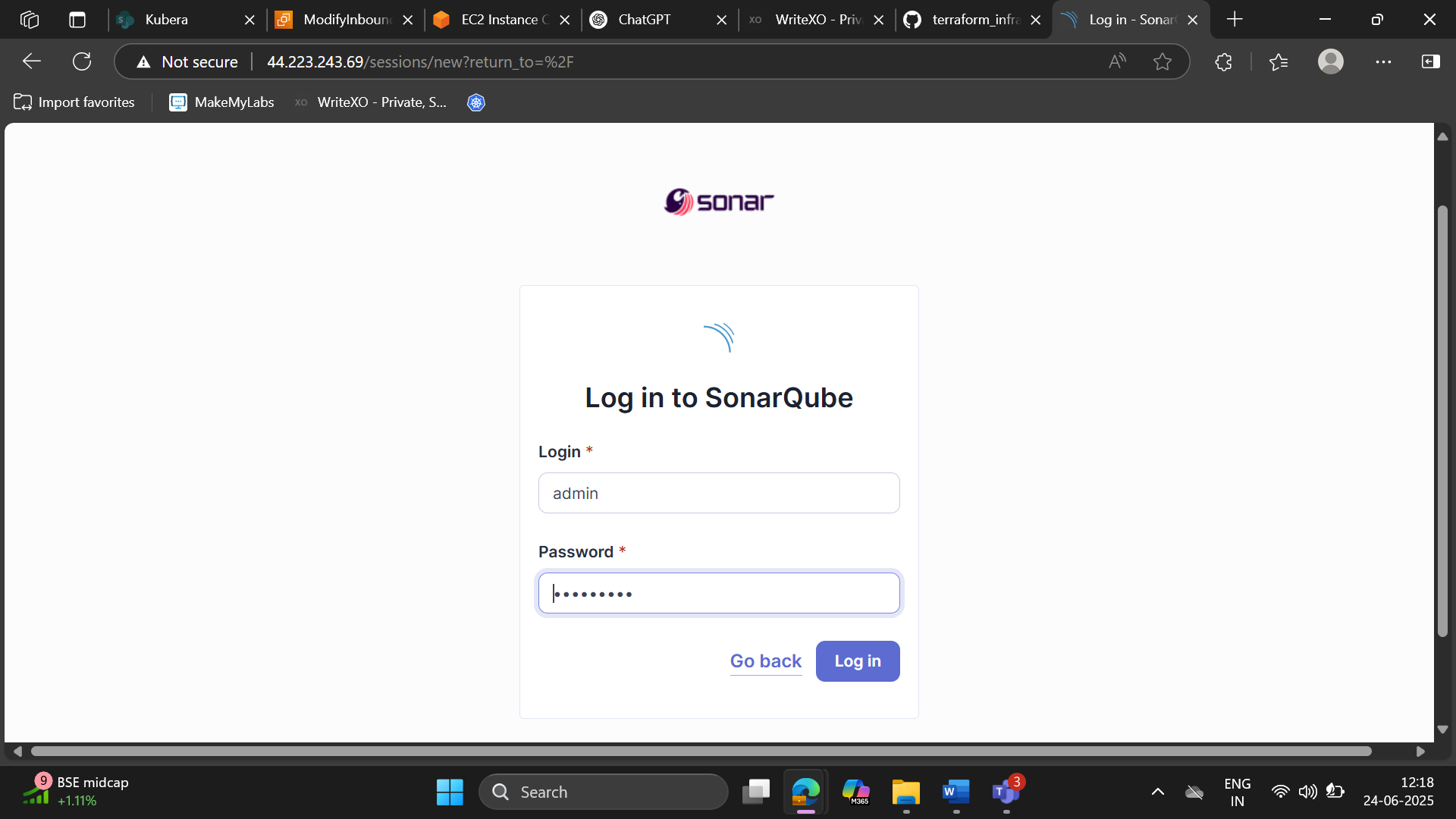


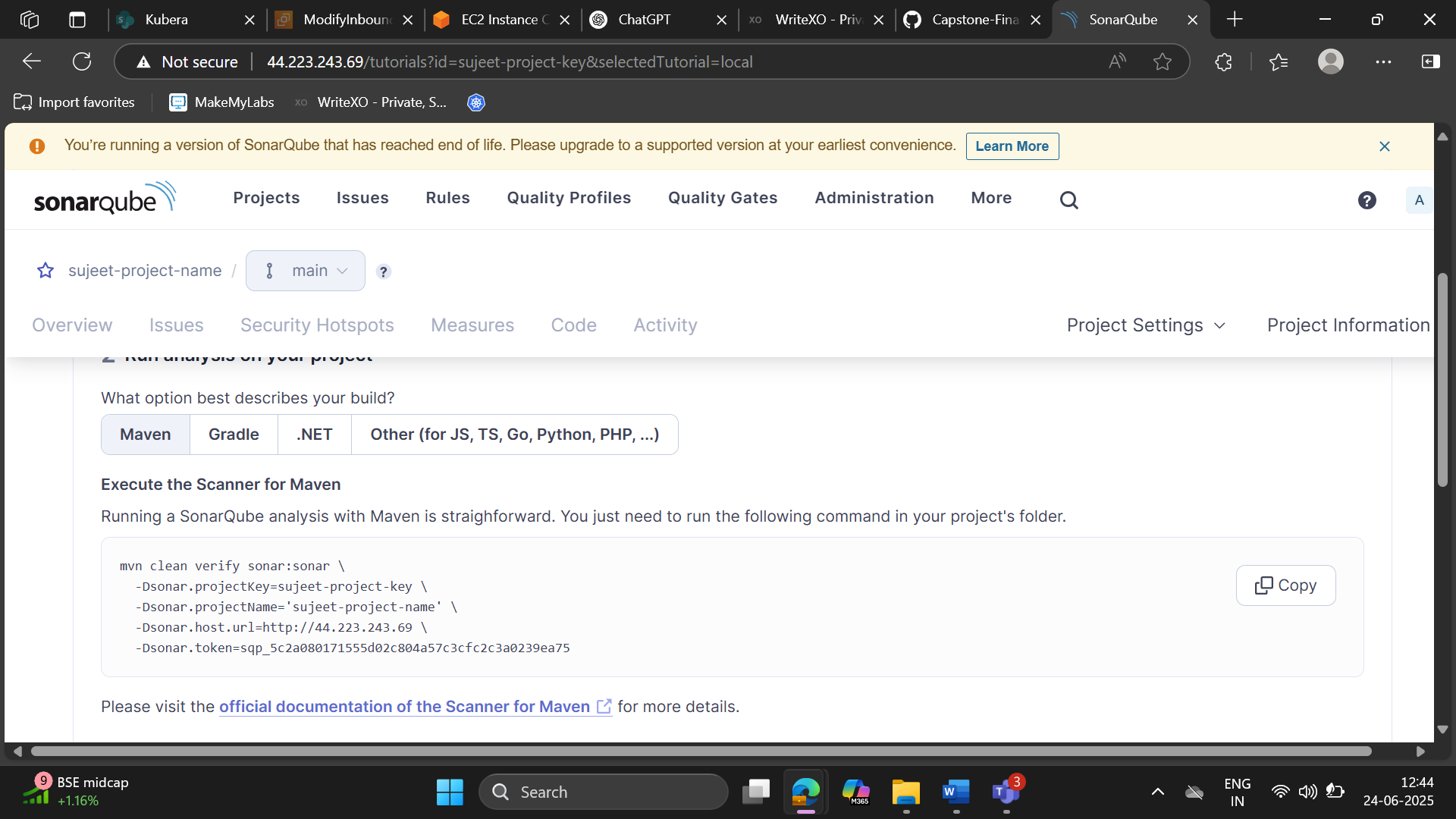
Phase 5: Cloud Watch:











Here are the **important YAML files** from your project repo that are essential for deploying your application on EKS:

**springboot-deploy.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: springboot-app

spec:

replicas: 2

selector:

matchLabels:

app: springboot

template:

metadata:

labels:

app: springboot

spec:

containers:

- name: springboot

image: <your-ecr-repo>/springboot-app:latest

ports:

- containerPort: 8080

env:

- name: SPRING\_DATASOURCE\_URL

valueFrom:

secretKeyRef:

name: db-secret

key: datasource\_url

**mysql-deploy.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql

spec:

selector:

matchLabels:

app: mysql

strategy:

type: Recreate

template:

metadata:

labels:

app: mysql

spec:

containers:

- image: mysql:5.7

name: mysql

env:

- name: MYSQL\_ROOT\_PASSWORD

value: rootpassword

- name: MYSQL\_DATABASE

value: capstone\_db

ports:

- containerPort: 3306

volumeMounts:

- name: mysql-persistent-storage

mountPath: /var/lib/mysql

volumes:

- name: mysql-persistent-storage

persistentVolumeClaim:

claimName: mysql-pvc

**react-event-deploy.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: react-frontend

spec:

replicas: 2

selector:

matchLabels:

app: react

template:

metadata:

labels:

app: react

spec:

containers:

- name: react

image: <your-ecr-repo>/react-frontend:latest

ports:

- containerPort: 3000

**ingress.yaml**

yaml

CopyEdit

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: capstone-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- http:

paths:

- path: /springboot

pathType: Prefix

backend:

service:

name: springboot-service

port:

number: 8080

- path: /react

pathType: Prefix

backend:

service:

name: react-service

port:

number: 3000

**mysql-service.yaml**

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: mysql

spec:

ports:

- port: 3306

selector:

app: mysql

**java-springboot-service.yaml**

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: springboot-service

spec:

type: ClusterIP

selector:

app: springboot

ports:

- port: 8080

targetPort: 8080

**react-event-service.yaml**

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: react-service

spec:

type: ClusterIP

selector:

app: react

ports:

- port: 3000

targetPort: 3000

**pv.yaml**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-pv

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

**pvc.yaml**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

=================Complete=========================