CAPSTONE PROJECT – AWS

TASKFLOW APP

INTRODUCTION

This document presents a comprehensive overview of the three-tier application,design and implement in a scalable, resilient, high available and automated infrastructure on AWS.

1. Infrastructure as Code (IaC): CloudFormation & Terraform
2. CI/CD Automation: Deploying the Application
3. Application Demo: The Task Flow App in Action
4. High Availability: Route 53 DNS Failover
5. Monitoring: CloudWatch for EKS
6. Automated Notifications: Lambda & Event Bridge

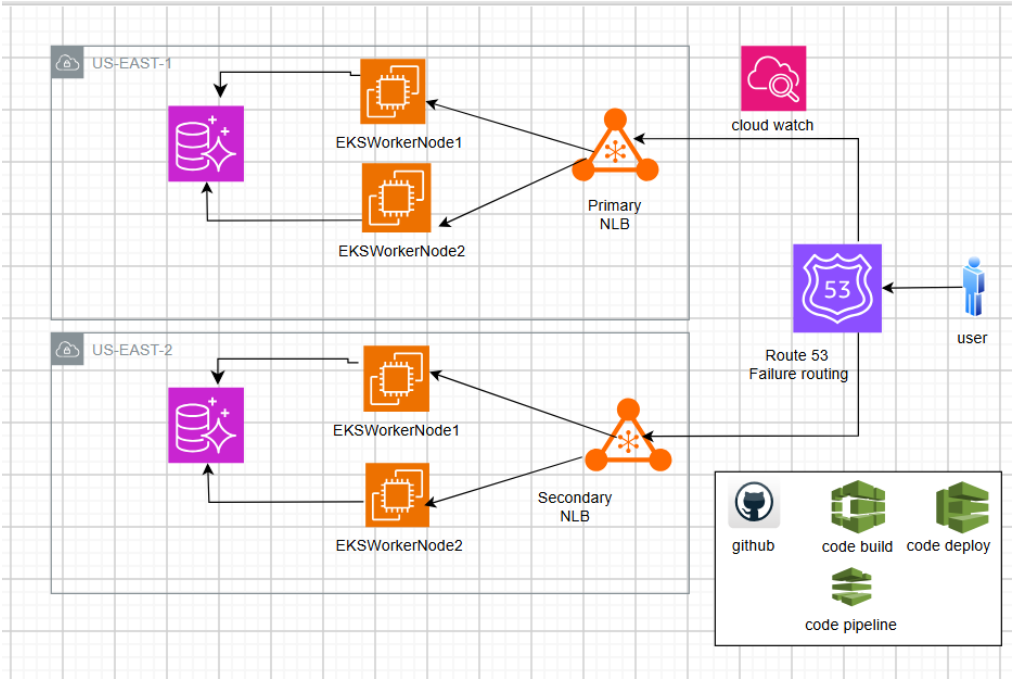
Three tier Architecture Overview :-

The application is deployed on AWS, leveraging Amazon Elastic Kubernetes Service (EKS) for container orchestration, Amazon RDS for the relational database, and various networking and IAM components to ensure secure and scalable operations.

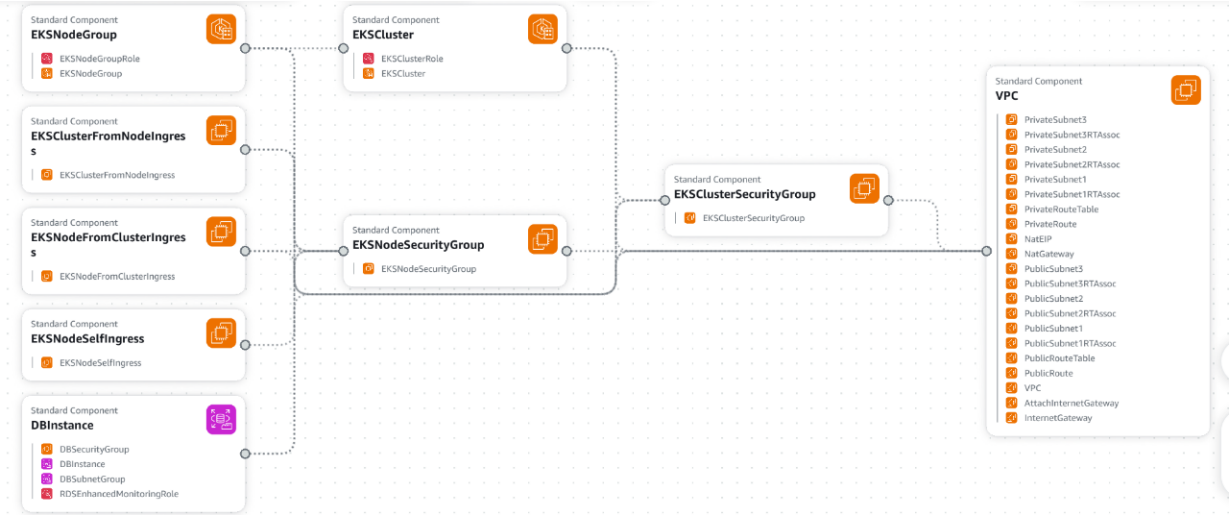
- Frontend: React application
- Backend: Node.js/Express API
- Database: AWS RDS PostgreSQL

ARCHITECHTURE

High level architecture of infrastructure with route53 failover routing



Infrastructure resources created through cloud formation and terraform (infracstructure seen in cloud formation infrastructure composer)



Git repo containing the whole code

https://github.com/SelmiNazeeb/FinalProject-Devops.git

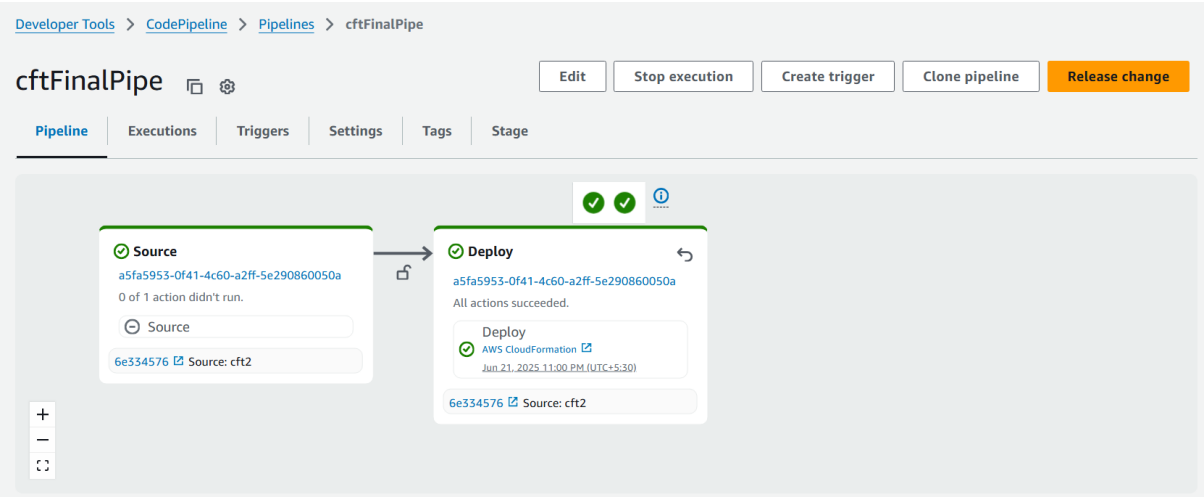
TASK1 :

CREATING INFRASTRUCTURE THROUGH CLOUDFORMATION IN REGION1 (US-EAST-1)

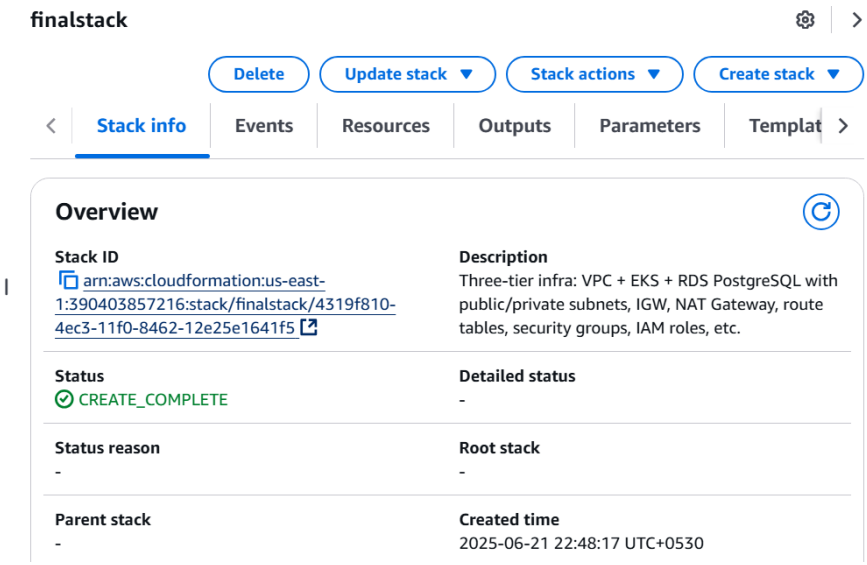
Permissions needed

- Cloud formation role – full admin access
- Pipeline role - AmazonEKSClusterPolicy, AWSCloudFormationFullAccess

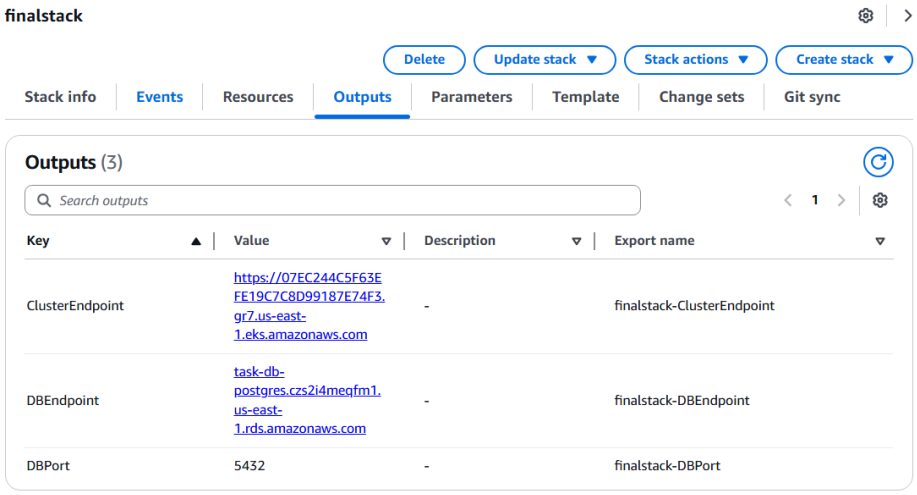
Pipeline of infrastructure creation : success



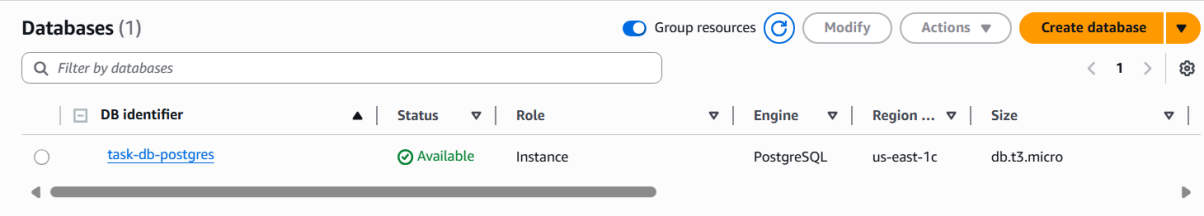
Cloud formation stack created through pipeline



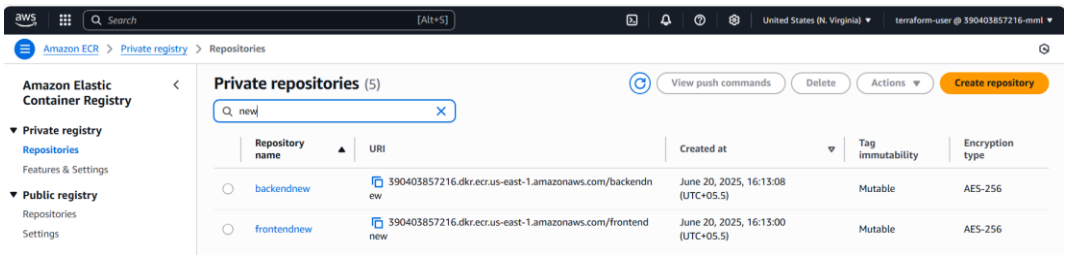
Output in cloud formation stack



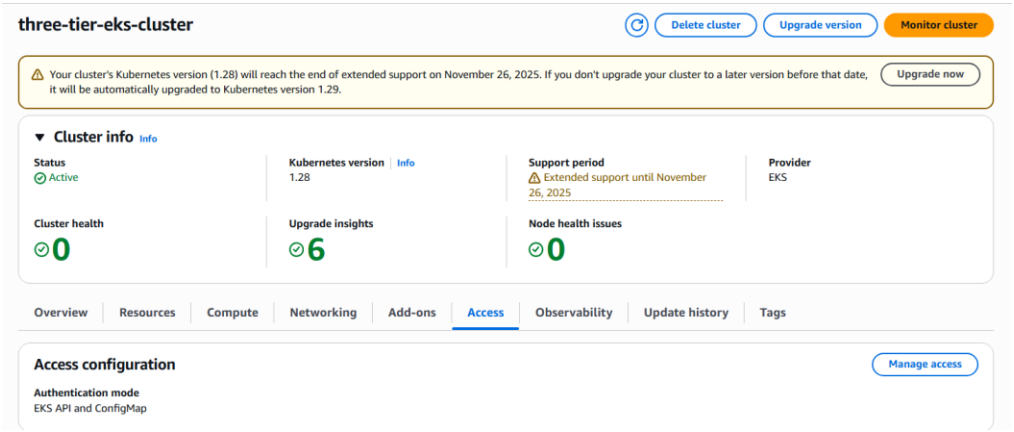
Postgresql Database



ECR



EKS cluster



Permission to user to access EKS cluster

Select EKS cluster -> Access -> Add access entry -> give user arn -> next -> add “AmazonEKSClusterAdminPolicy “ -> save

arn:aws:iam::390403857216:user/terraform-user	Standard	arn:aws:iam::390403857216:user/terraform-user	-	AmazonEKSClusterAdminPolicy

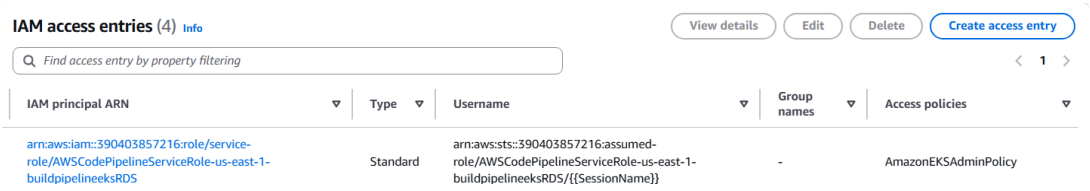
TASK 2:

DEPLOYING APPLICATION ON THE INFRASTUCTURE CREATED BY CLOUD FORMATION

Permissions needed

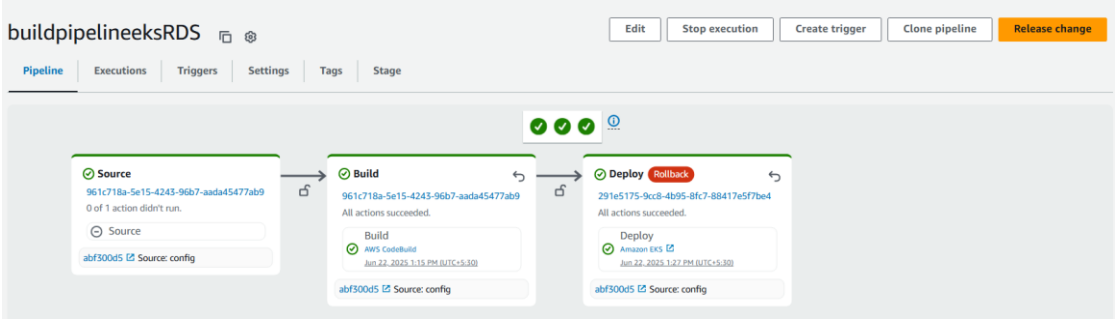
- pipeline role – Access entry in EKS cluster

Select EKS cluster -> Access -> Add access entry -> give pipeline role arn -> next -> add “AmazonEKSAdminPolicy “ -> save



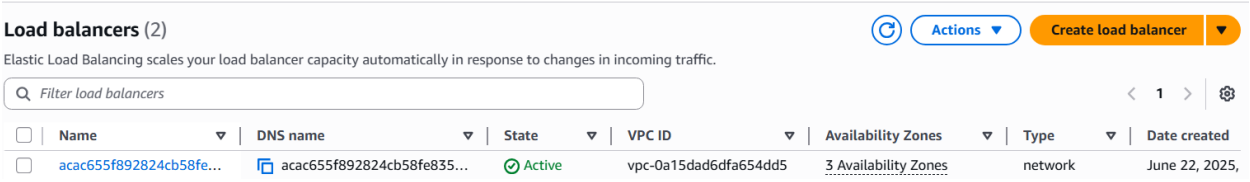
- Code build role - AmazonEC2ContainerRegistryPowerUser

Success pipeline of application deployment



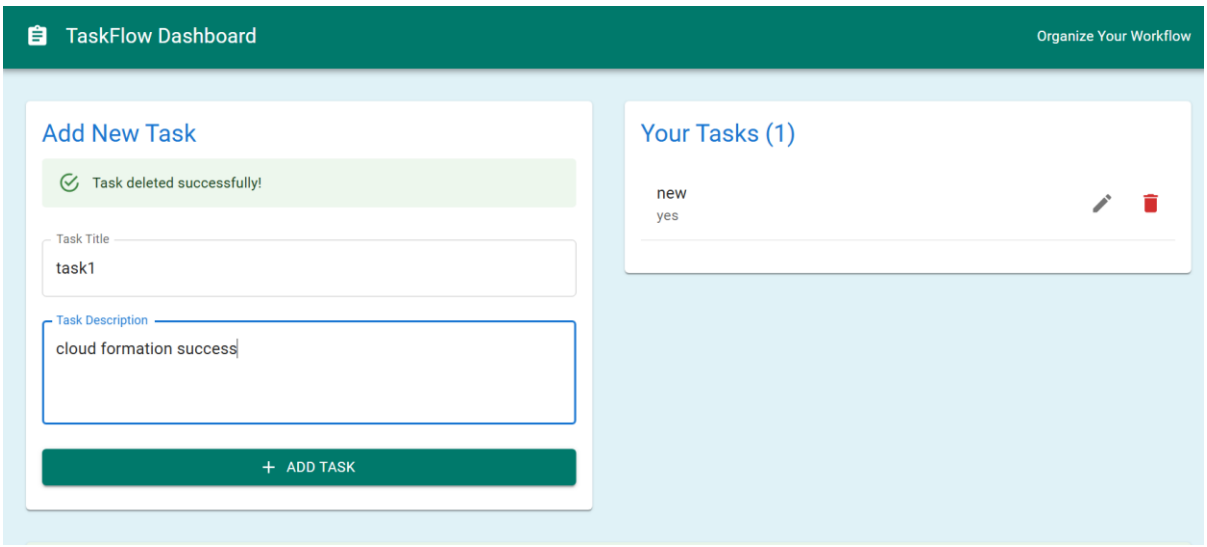
>>> Now all containers, pods, deployment and services are running

Load Balancer created for the application

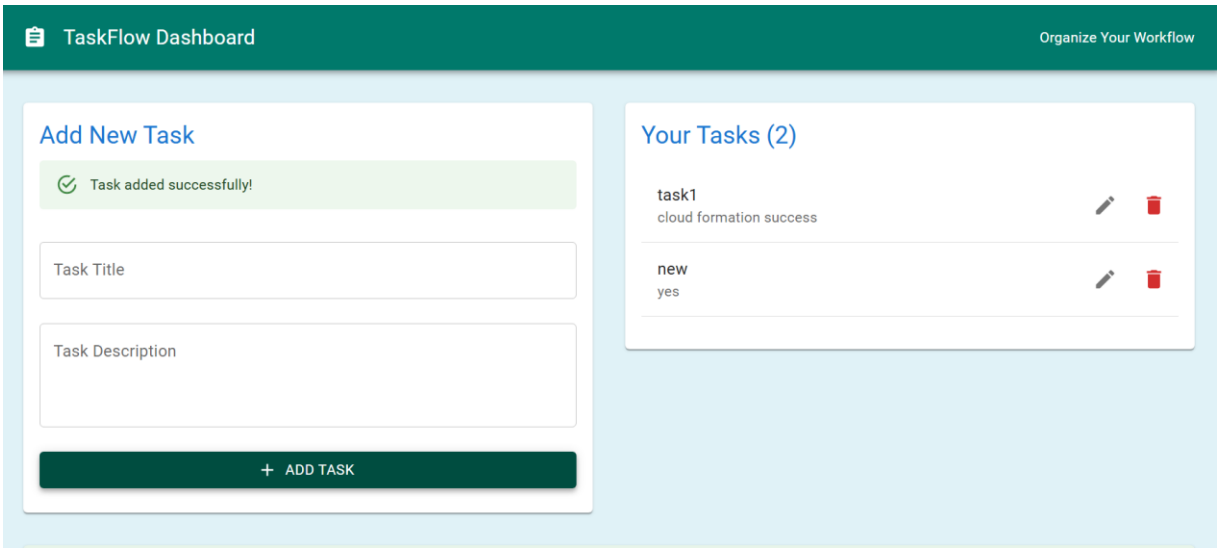


In Browser – paste load balancer arn **(Application is working)**

Frontend of Task management APP – Adding a new task (task1 – give title and description -> add task)



Added task and stored in Database



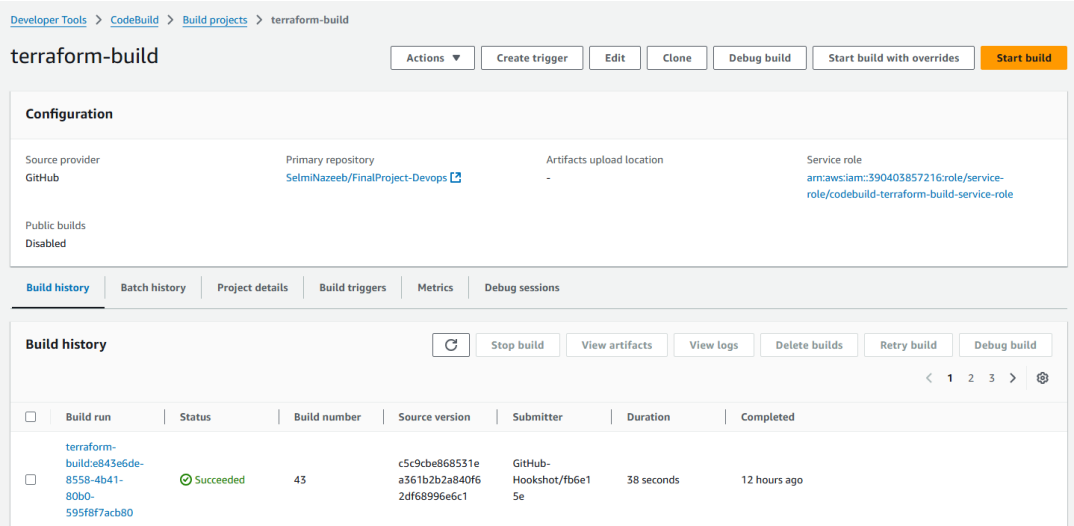
TASK3 :

CREATING INFRASTRUCTURE THROUGH TERRAFORM IN REGION2 (US-EAST-2)

Permissions needed

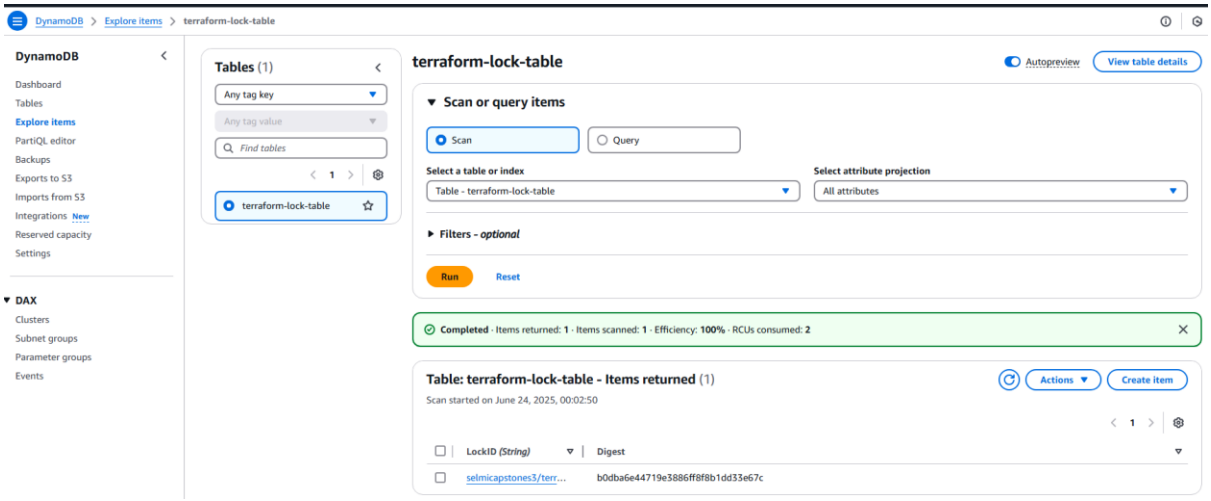
- Code build role - policy should be attached (added in github -> permissions/terraform-build-policy)
  AmazonS3FullAccess, AmazonVPCFullAccess

Infrastructure creation : success

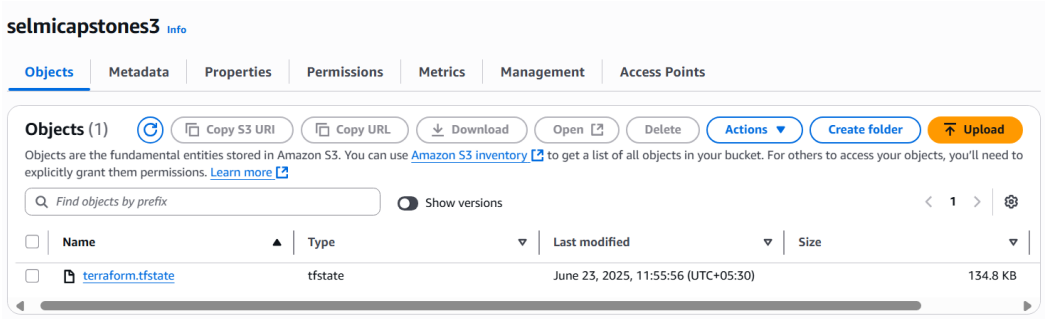


Dynamo DB table and S3 bucket for storing terraform tf state storage of code build of infrastructure

Dynamo DB



S3 bucket



Outputs

```
Outputs:

eks_cluster_endpoint = "https://7627E37233A3CF883ECD615F9D063276.gr7.us-east-2.eks.amazonaws.com"
eks_cluster_security_group_id = "sg-020b2b5f3047cfc66"
rds_endpoint = "task-db-postgres.czus4mukos81.us-east-2.rds.amazonaws.com:5432"
vpc_id = "vpc-02b566a9f4f50daec"
```

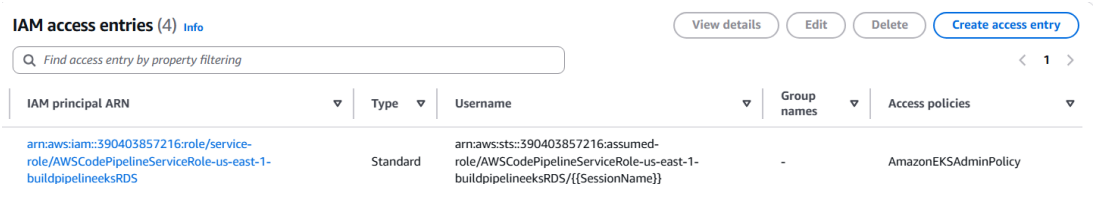
TASK 4:

DEPLOYING APPLICATION ON THE INFRASTUCTURE CREATED BY TERRAFORM

Permissions needed

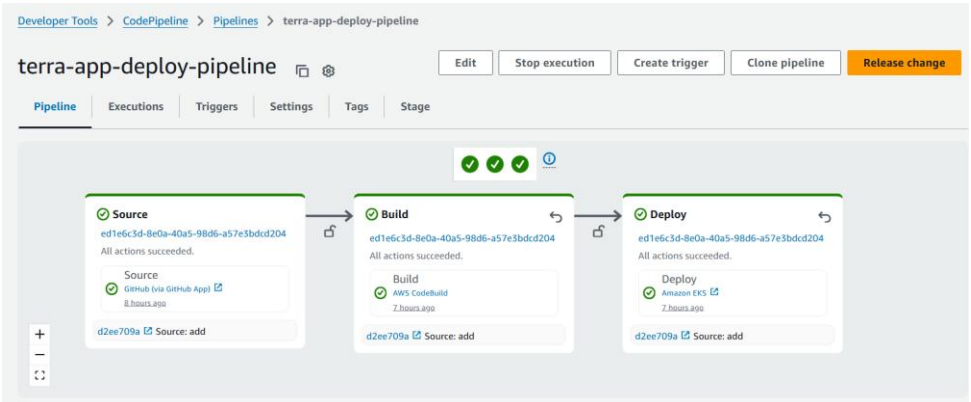
- pipeline role – Access entry in EKS cluster

Select EKS cluster -> Access -> Add access entry -> give pipeline role arn -> next -> add “AmazonEKSAdminPolicy “ -> save



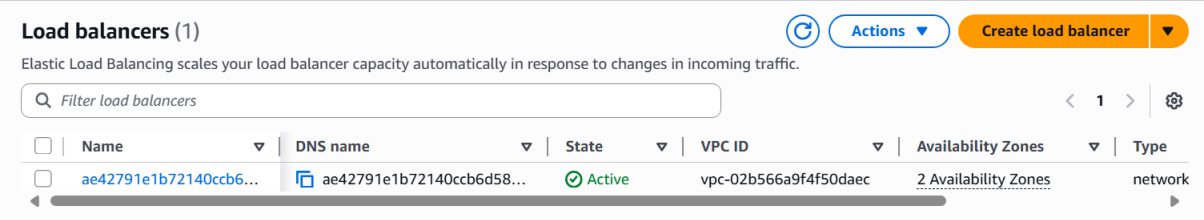
- Code build role - AmazonEC2ContainerRegistryPowerUser

Success pipeline of application deployment



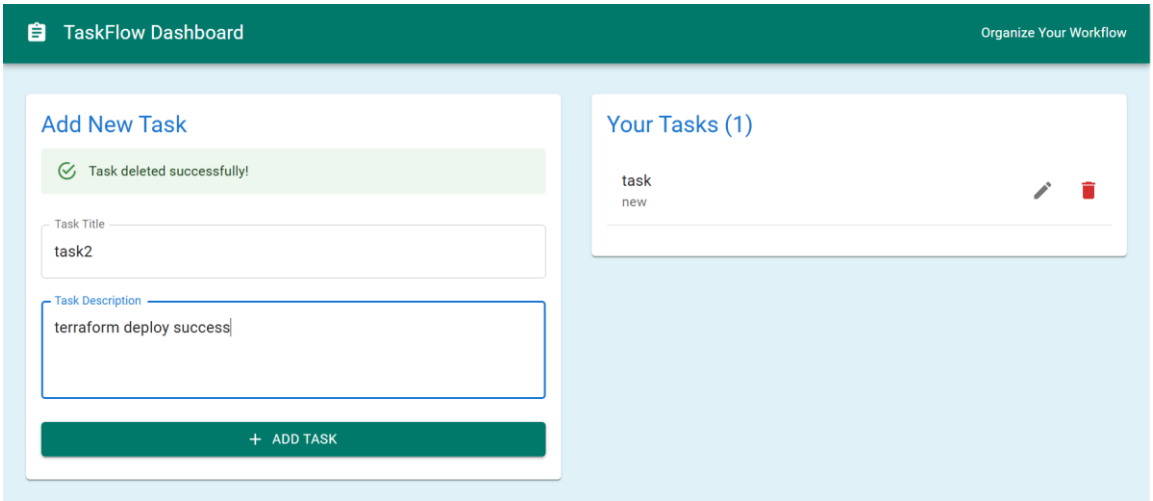
>>> Now all containers, pods, deployment and services are running

Load Balancer created for the application

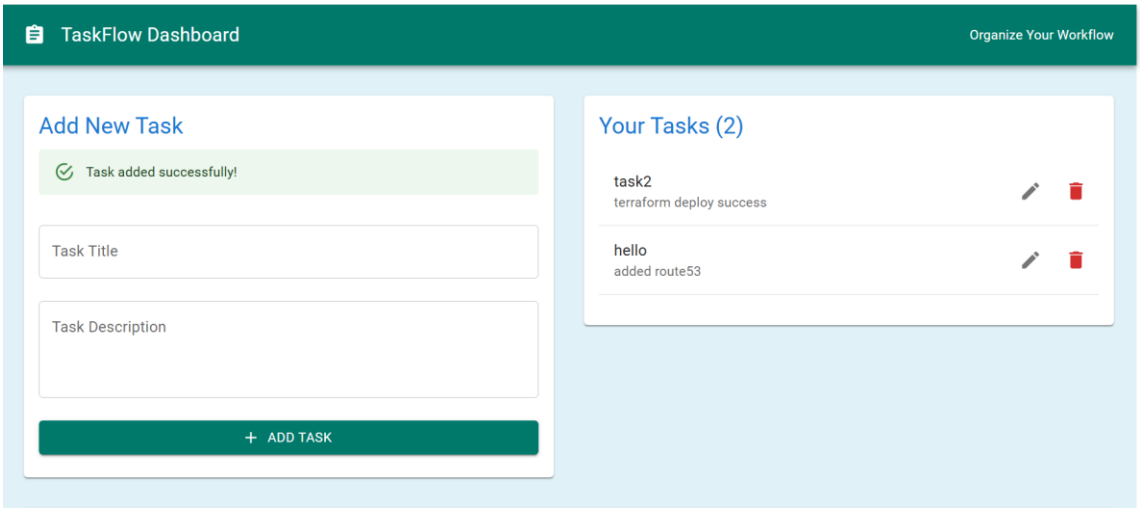


In Browser – paste load balancer arn **(Application is working)**

Frontend of Task management APP – Adding a new task (task2 – give title and description -> add task)



Added task and stored in Database



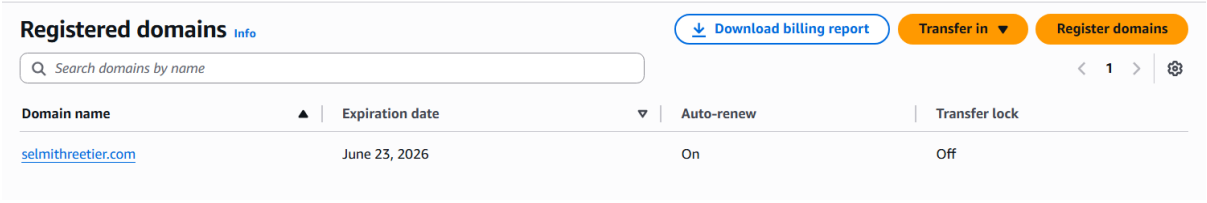
TASK 5 :

ROUTE 53 FOR FAILURE ROUTING

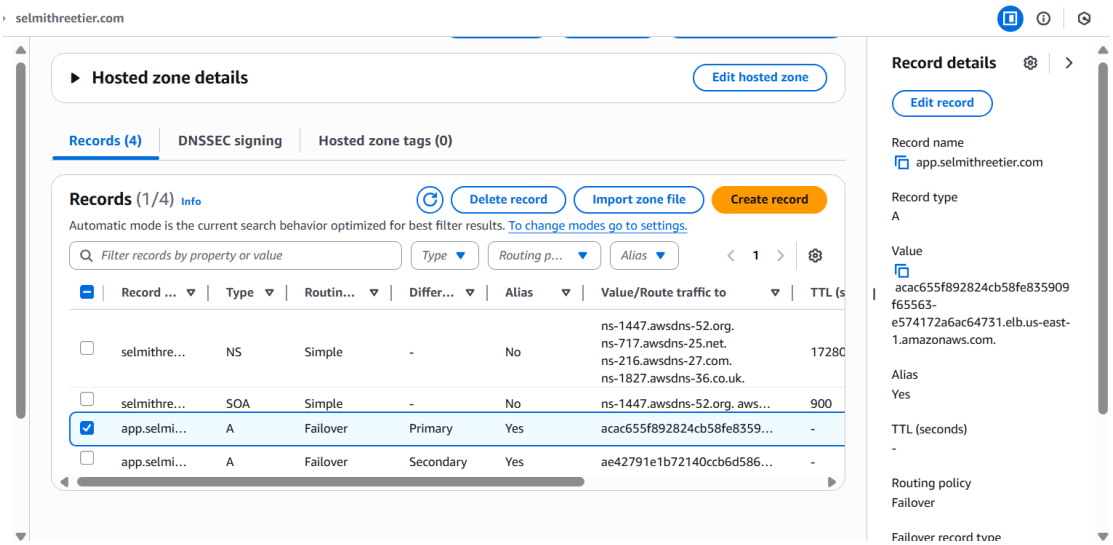
To ensure high availability of application, implemented a DNS failover strategy using Amazon Route 53.

- Configuration: An 'A' record was created with a Failover routing policy.
- Primary Target: The Network Load Balancer in us-east-1
- Secondary Target: The Network Load Balancer in us-east-2
- Function: If the primary NLB becomes unhealthy, Route 53 automatically reroutes all traffic to the secondary region.

Domain Name : selmithreetier.com

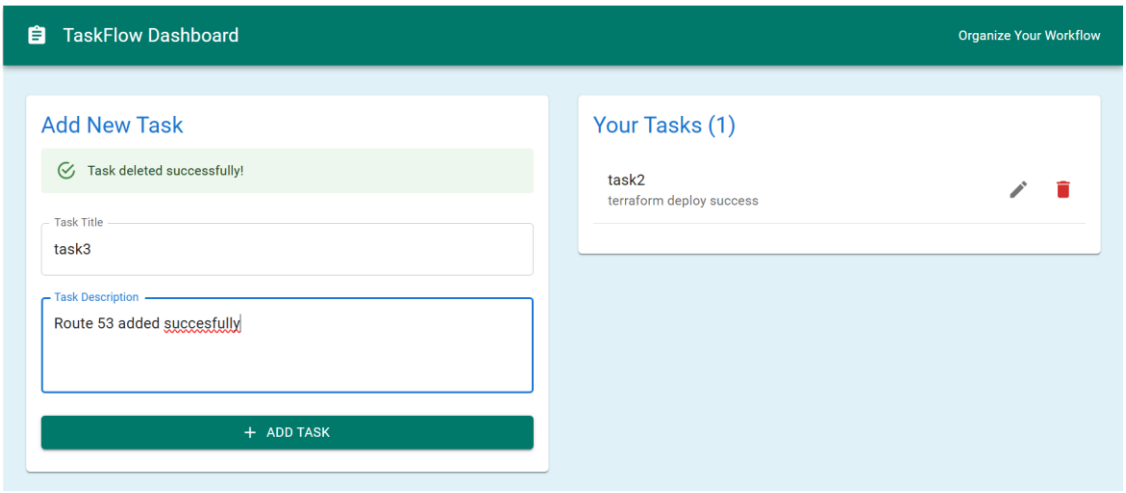


Creating a failure routing A record to public hosted zone

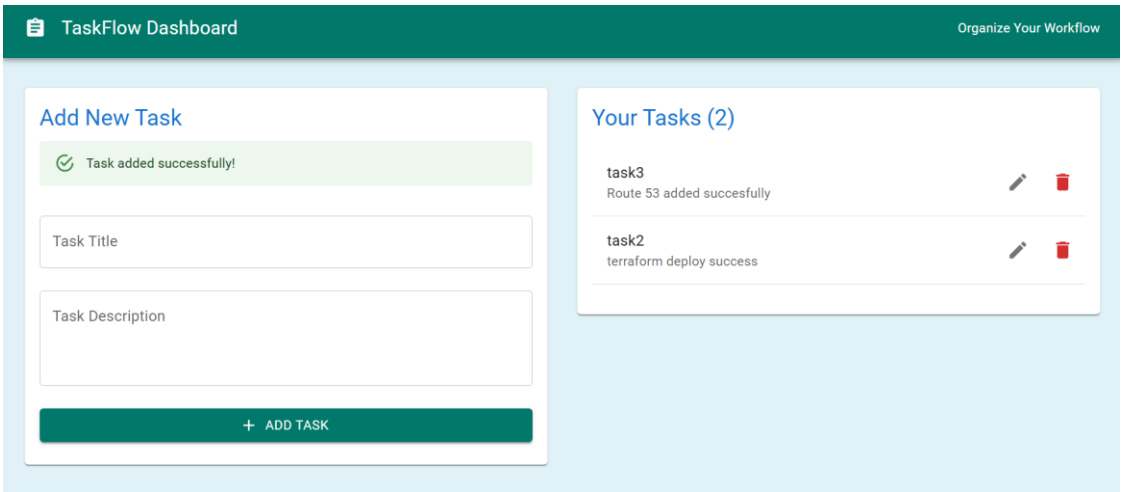


Browse – “https://app.selmithreetier.com”

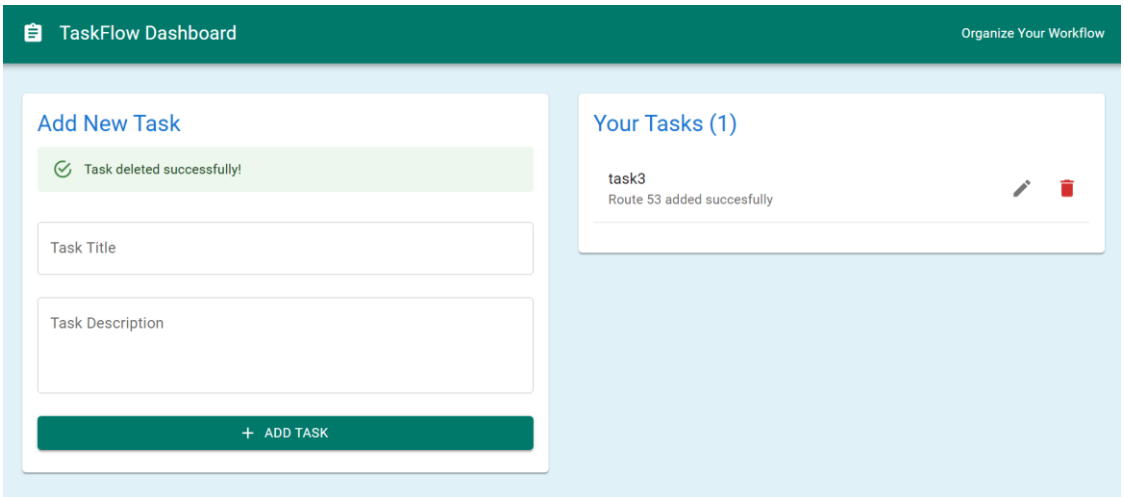
Frontend of Task management APP – Adding a new task (task3 – give title and description -> add task)



Added task and stored in Database



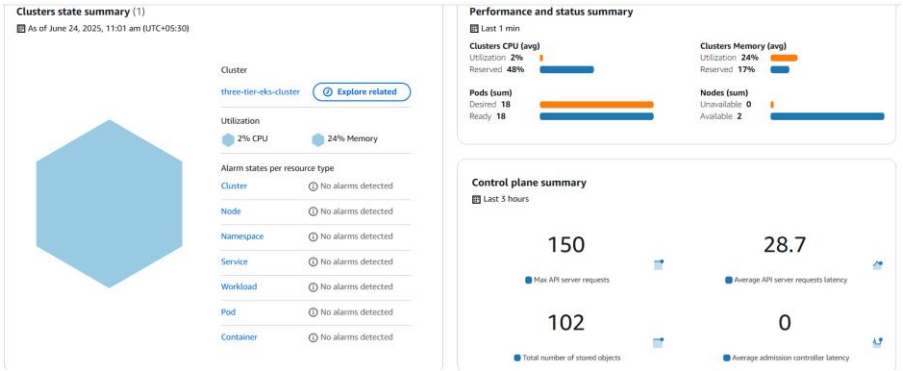
Deleted one stored value



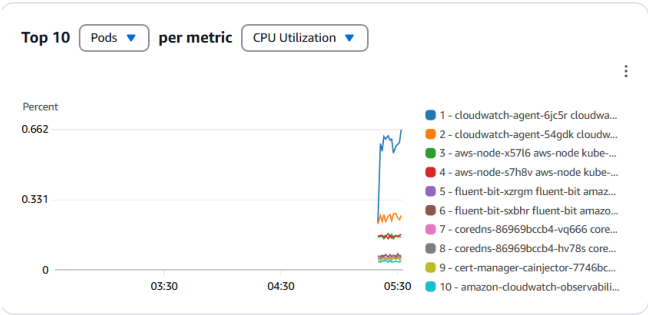
TASK 6:

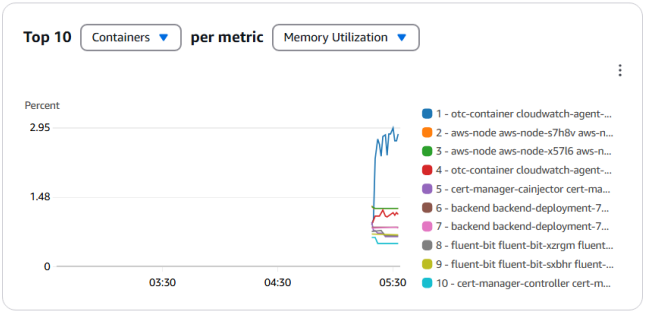
CLOUD WATCH – MONITORING EKS CLUSTER THROUGH CLOUD WATCH

CloudWatch dashboard showing the "Clusters state summary" and graphs for CPU and Memory Utilization

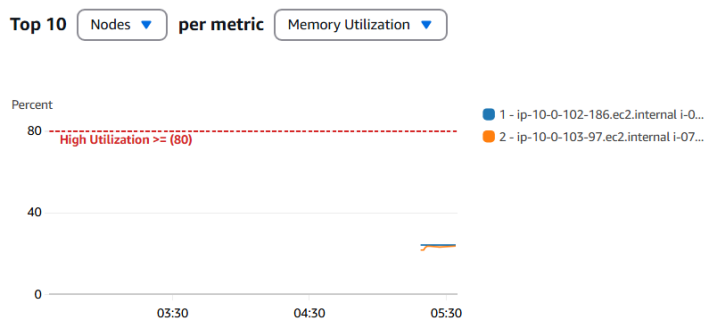


Pod cpu utilization monitoring





nodes memory utilization monitoring



TASK 7:

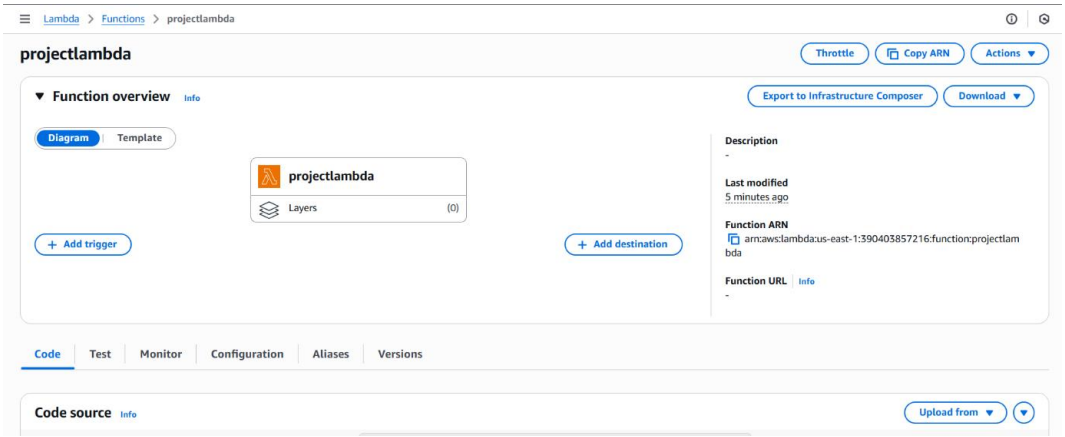
LAMBDA AND EVENT BRIDGE

This solution ensures that whenever a pipeline execution fails, an automated email notification is sent using Amazon SNS, triggered by Amazon EventBridge and handled via an AWS Lambda function.

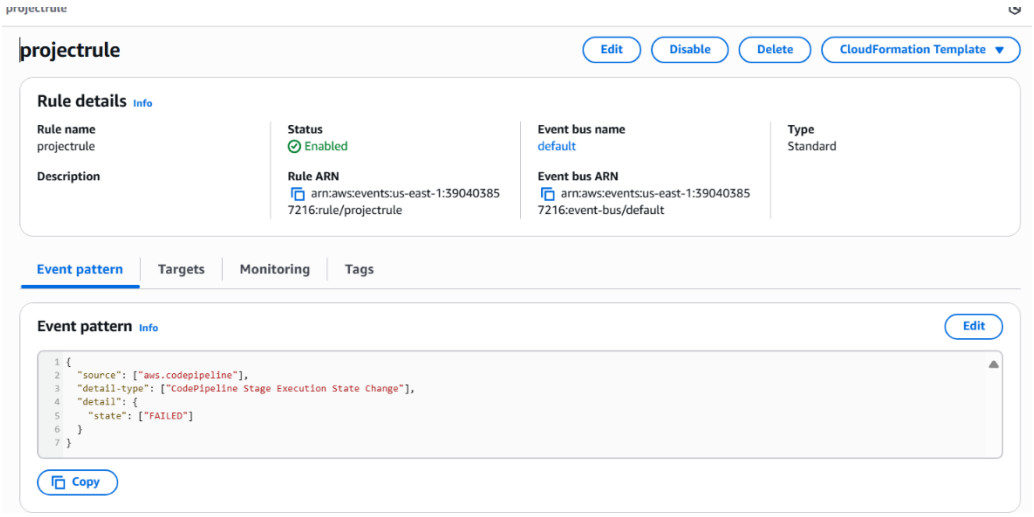
Workflow:

1. A Code Pipeline execution fails.
2. Amazon Event Bridge detects the state change.
3. An AWS Lambda function is triggered.
4. The Lambda function sends a message to an SNS topic.
5. SNS sends a formatted email notification to subscribers.

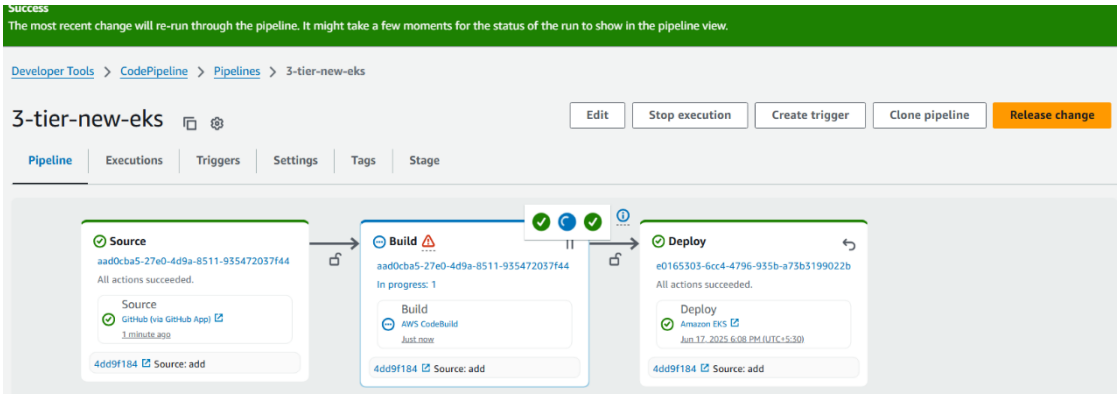
Lambda function



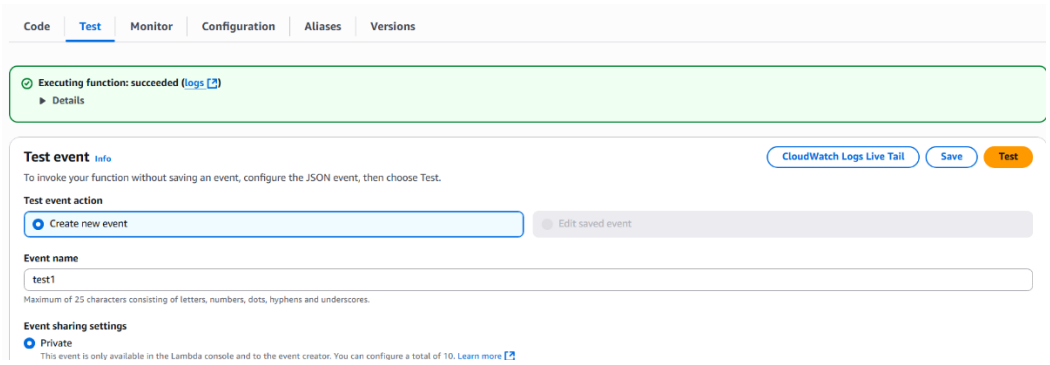
Event bridge rule



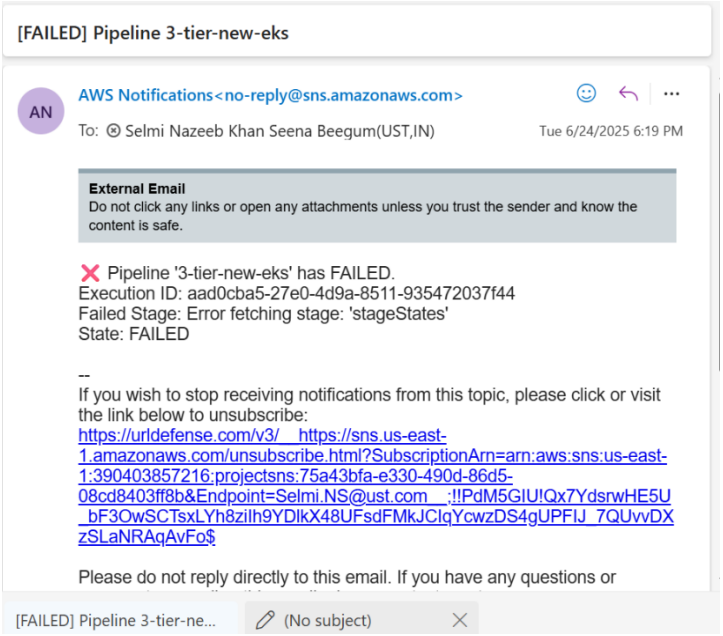
Pipeline failed



Lambda test success



Notification mail



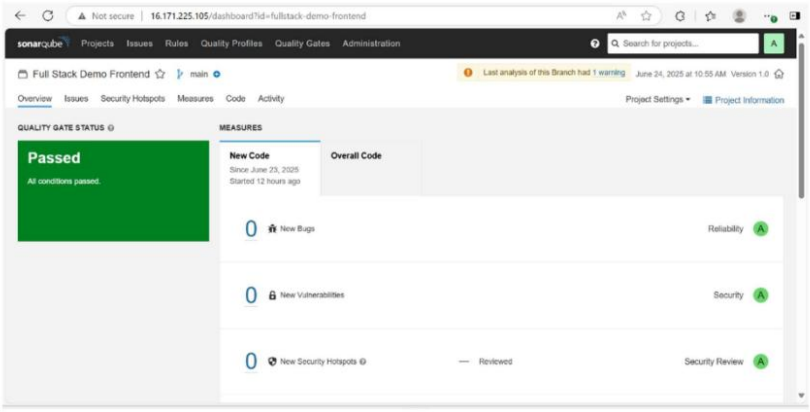
TASK 8:

SONARQUBE INTEGRATION

Create a Project in SonarQube

1. Log into the SonarQube UI
2. Go to **Projects > Create project**
3. Give it a name, like Full_Stack_backend
4. Choose **"Manually"** when asked how you'll set it up
5. Copy the generated token and save it securely — you'll use this in the buildspec.yml
6. Note down the **Project Key** and **SonarQube Server URL**

```
- sonar-scanner \
    -Dsonar.projectKey=$SONAR_PROJECT_KEY \
    -Dsonar.sources=. \
    -Dsonar.host.url=$SONAR_HOST_URL \
    -Dsonar.login=$SONAR_TOKEN
```



TASK 9:

CODE COMMIT

Code commit is service inside aws which can be used instead of github in source stage of code pipeline.

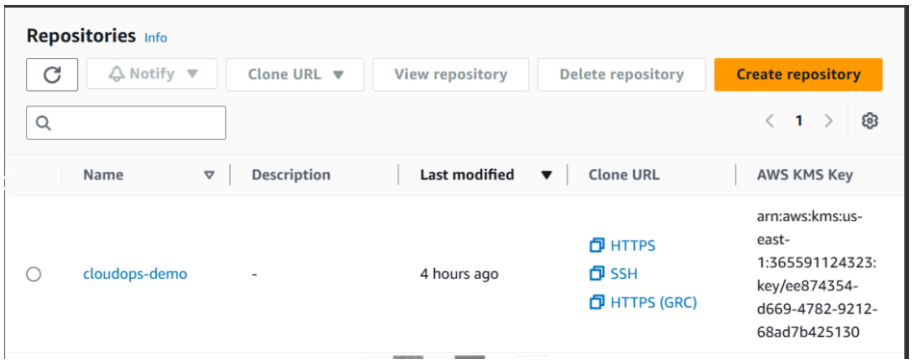
STEPS:

1. **Create a repository through CLI**

   - aws codecommit create-repository --repository-name cloudops-demo --repository-description "My first CodeCommit repo"

2. **Clone using Git:**

   - git clone ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/cloudops-demo




```
gdud9JCpfX59q3pnfULB4IdAKOBbrChiMZQZXfD/4gZ2oW7uvXLReWWjy3AAVjzzWr39m2dlLLYp7vpL2bAxqCDstHXJ8K2S8x/f8Ib//BFGYJpBflxWKO7g8NZg8V/ewToKq3iSgcfZ
TeYznmtBLgsVDG6hhTJksrijotWtGfh8fVXxaEFckdIkM7IohAOZUVYPlbILrUvsnZ1IdLWFxPU6u+FnMYFxjL0hycV9372la5s0OOZv/bMyMoNywQ007PMkOTKrmqp/8prAQSPb/PTY
nSpjIoLvabryunk88VuK4qay7hwpB7zeY70yx5tgf7oKgVpx7HGFNvuZwaVPfwc8U3lxRM/79SASQGwT1LFsr+5dZAJ0K78qWvCV6YUfG6AveNX2d8zbYguzjItvYyPEm2MRylZ0g36X
1Zf425gjZ7pDx7mXchtQEQ== codecommit-access
root@ip-172-31-86-56:~/.ssh# cd
root@ip-172-31-86-56:~# cd /home/ubuntu/
root@ip-172-31-86-56:/home/ubuntu# nano ~/.ssh/config
root@ip-172-31-86-56:/home/ubuntu# chmod 600 ~/.ssh/config
root@ip-172-31-86-56:/home/ubuntu# git clone ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/cloudops-demo
Cloning into 'cloudops-demo'...
The authenticity of host 'git-codecommit.us-east-1.amazonaws.com (52.94.226.180)' can't be established.
RSA key fingerprint is SHA256:eLMY1jODKA4uvDzcl/KgtIayZANwX6t8+8isPtotBoY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'git-codecommit.us-east-1.amazonaws.com' (RSA) to the list of known hosts.
warning: You appear to have cloned an empty repository.
root@ip-172-31-86-56:/home/ubuntu# ssh git-codecommit.us-east-1.amazonaws.com
You have successfully authenticated over SSH. You can use Git to interact with AWS CodeCommit. Interactive shells are not supported.Connecti
on to git-codecommit.us-east-1.amazonaws.com closed by remote host.
Connection to git-codecommit.us-east-1.amazonaws.com closed.
root@ip-172-31-86-56:/home/ubuntu#
```

TASK 10:

CREATE S3 BUCKET, CLONE YOUR GITHUB REPO AND UPLOAD IN S3 THROUGH TERRAFORM

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_s3_bucket" "project_bucket" {
  bucket = "selmi-bucket-123456"
  force_destroy = true

  tags = {
    Name        = "ProjectCodeBucket"
    Environment = "Dev"
  }
}

# Clone GitHub repo locally and upload to S3 using AWS CLI
resource "null_resource" "fetch_and_upload_code" {
  provisioner "local-exec" {
    command = <<EOT
      rm -rf /tmp/my-project
      git clone https://github.com/SelmiNazeeb/FinalProject-Devops.git /tmp/my-project
      aws s3 cp /tmp/my-project s3://${aws_s3_bucket.project_bucket.bucket}/ --recursive
    EOT
  }

  depends_on = [aws_s3_bucket.project_bucket]
}
```