ECS Fargate Nginx Server Deployment with ALB And EFS using CodePipeline - Terraform (Task 8)

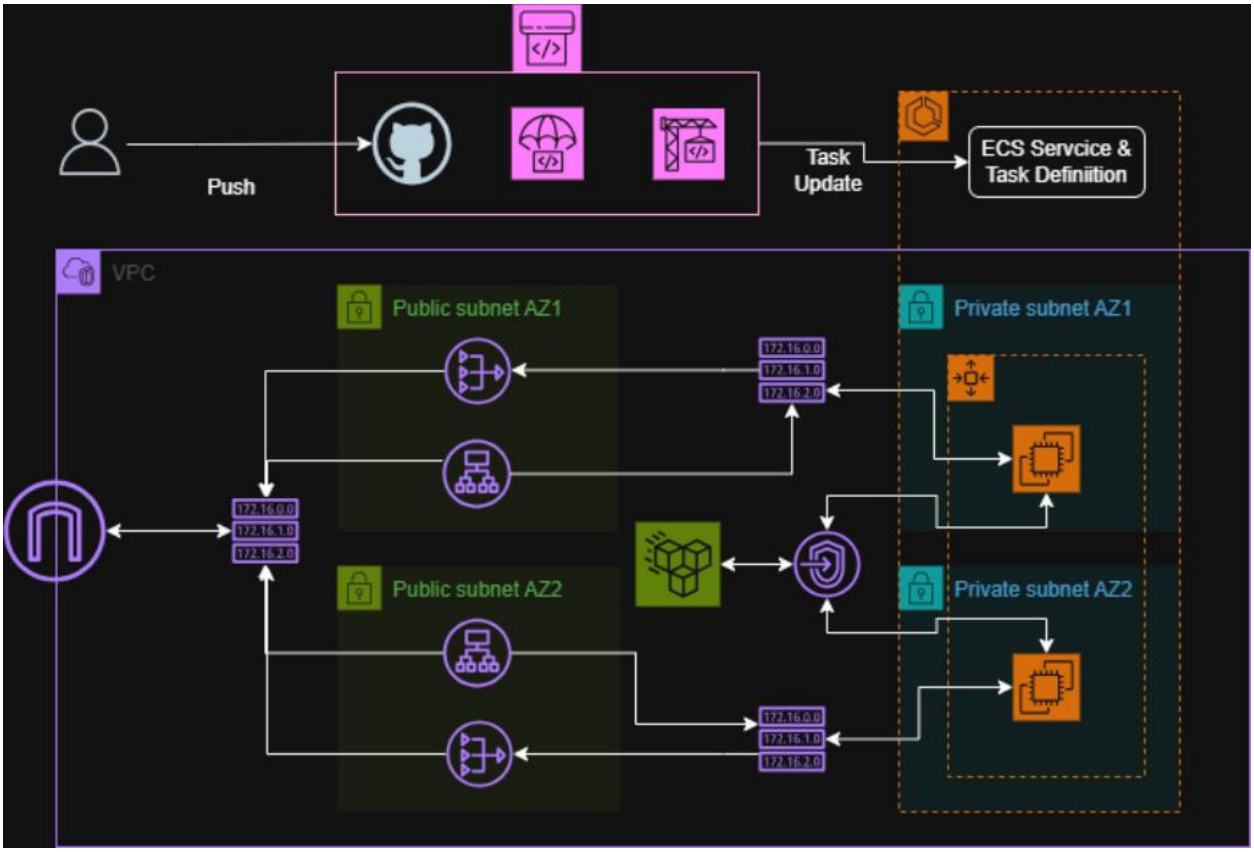
Zaeem Attique Ashar

Cloud Intern

Task Description:

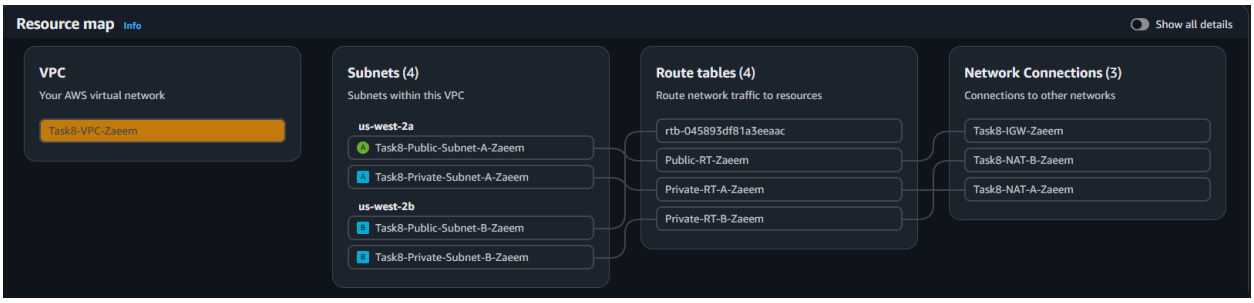
This task will be a guide on setting up a Node Application server with a highly available architecture on AWS ECS Cluster Spread over multiple availability zones. An internet facing Load Balancer will be set up in front of the ECS cluster to listen to traffic and balance it upon tasks. AWS EFS will be used for persistence in shared storage to prevent data loss. The application will automatically be deployed using the AWS CodePipeline. The complete infrastructure will be coded in Terraform modules.

Architecture Diagram:



Task8.1: Create basic networking infrastructure

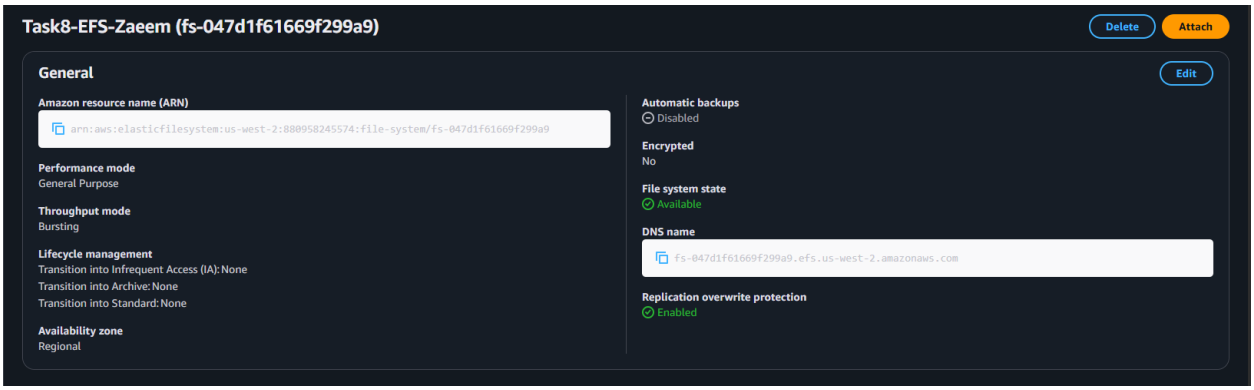
- Create and configure a VPC
    - CIDR Block: 10.0.0.0/16
- Create and configure Subnets
    - Public Subnet A (us-west-2a), CIDR: 10.0.1.0/24
    - Private Subnet A (us-west-2a), CIDR: 10.0.2.0/24
    - Public Subnet B (us-west-2b), CIDR: 10.0.3.0/24
    - Private Subnet A (us-west-2a), CIDR: 10.0.4.0/24
- Create and configure NAT Gateways
    - NAT Gateway A in Public Subnet A
    - NAT Gateway B in Public Subnet B
- Create and configure Internet Gateway
    - Create and attach to the project's VPC
- Create and configure Route Tables
    - Public Route Table, Outbound rule: 0.0.0.0/0 -> IGW, attach to Public SN A&B
    - Private Route Table A, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN A
    - Private Route Table B, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN B



Task8.2: Create EFS File System and Mount Points

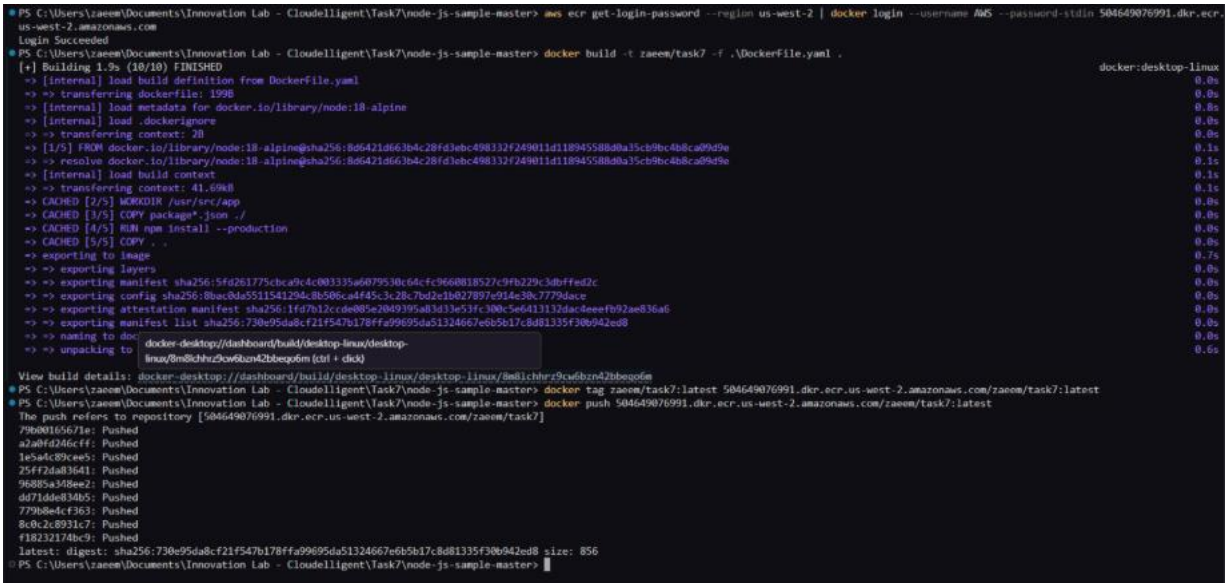
- Create File System:
    - Name: Task8-EFS-Zaeem
    - Ecryption: Yes
- Create Access Points:
    - File System: Task8-EFS-Zaeem

    - Name: Task8-EFS-AP-Zaeem
    - Root Directiory: /
    - POSIX UID 1000, GID 1000
    - Owner UID 1000, Owner GID: 1000
    - AP Permissions: 0755
- Create Mount Points:
    - AZ us-west-2a, Subnet Private, SG Task8-EFS-SG-Zaeem
    - AZ us-west-2b, Subnet Private, SG Task8-EFS-SG-Zaeem



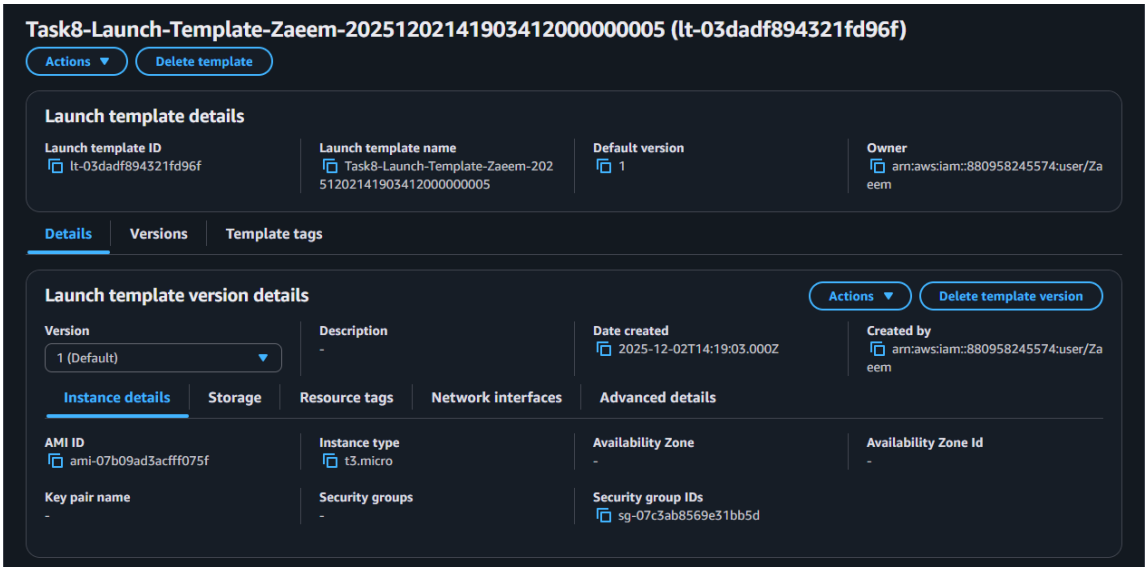
Task8.3: Build Docker Image and push to ECR

- Build Docker Image:
    - Got to source code directory.
    - Create DockerFile.yaml and write instructions
    - Use command to build image: *docker image –t nodejs:latest –f DockerFile.yaml .*
    - Use to tag image: *docker tag nodejs:lastest <ecr uri>/zaeem/Task8:latest*
    - Use command to push to ECR: *docker push <ecr uri>/zaeem/Task8:latest*

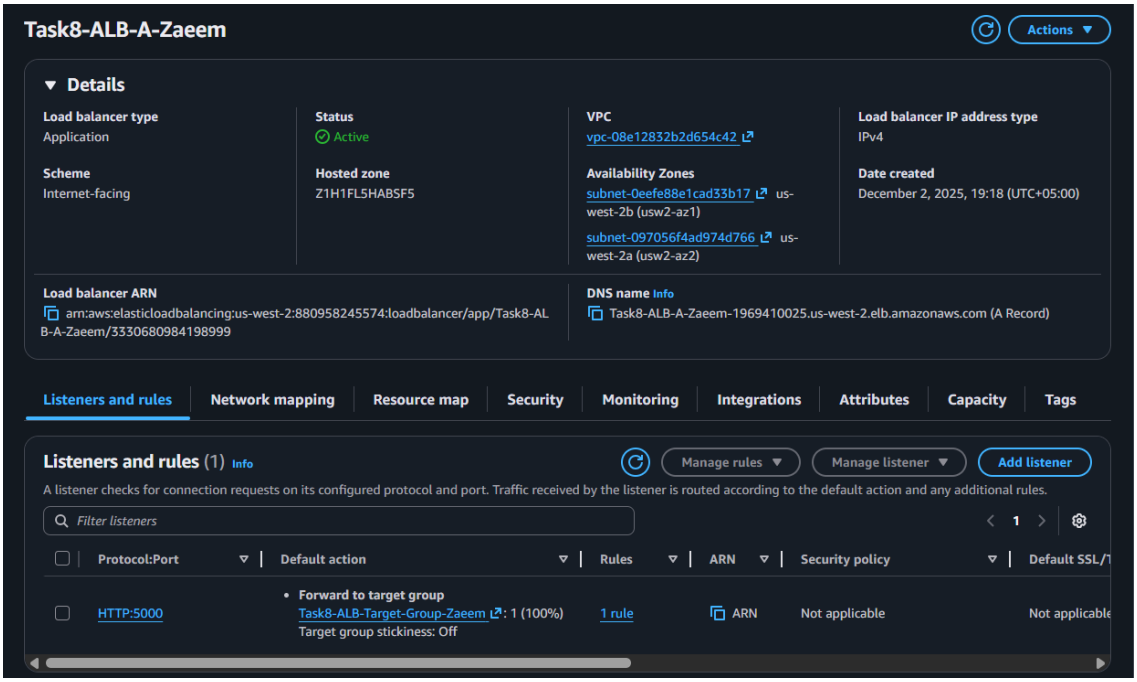


Task8.4: Create an ECS + EC2 Cluster and dependencies

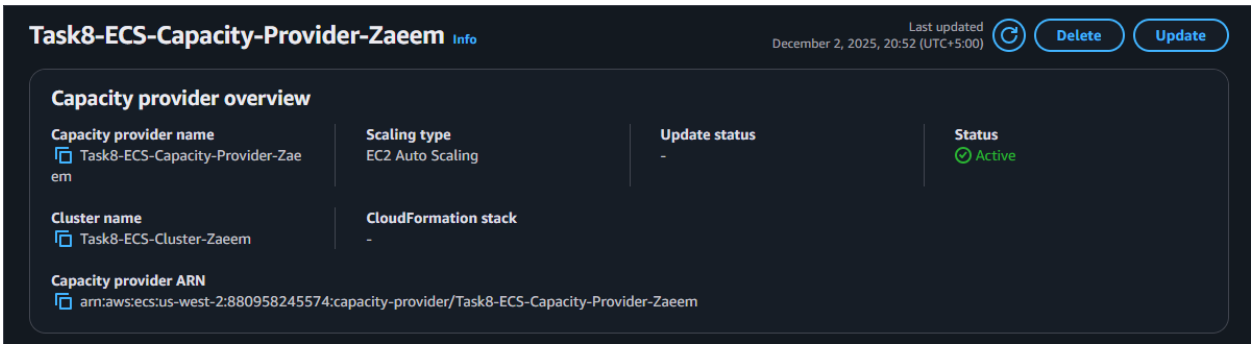
- Create and Configure a Launch Template
    - Name: Task8-EC2-LT-Zaeem
    - Container instance AMI: Amazon Linux 2023
    - Instance Type: t3.micro
    - SSH Key pair: Task8-EC2
    - Subnet: Do not include
    - Availability Zone: Do not include
    - Security Group: Task8-EC2-SG-Zaeem
    - Storage Volume: 8GiB Default EBS volume



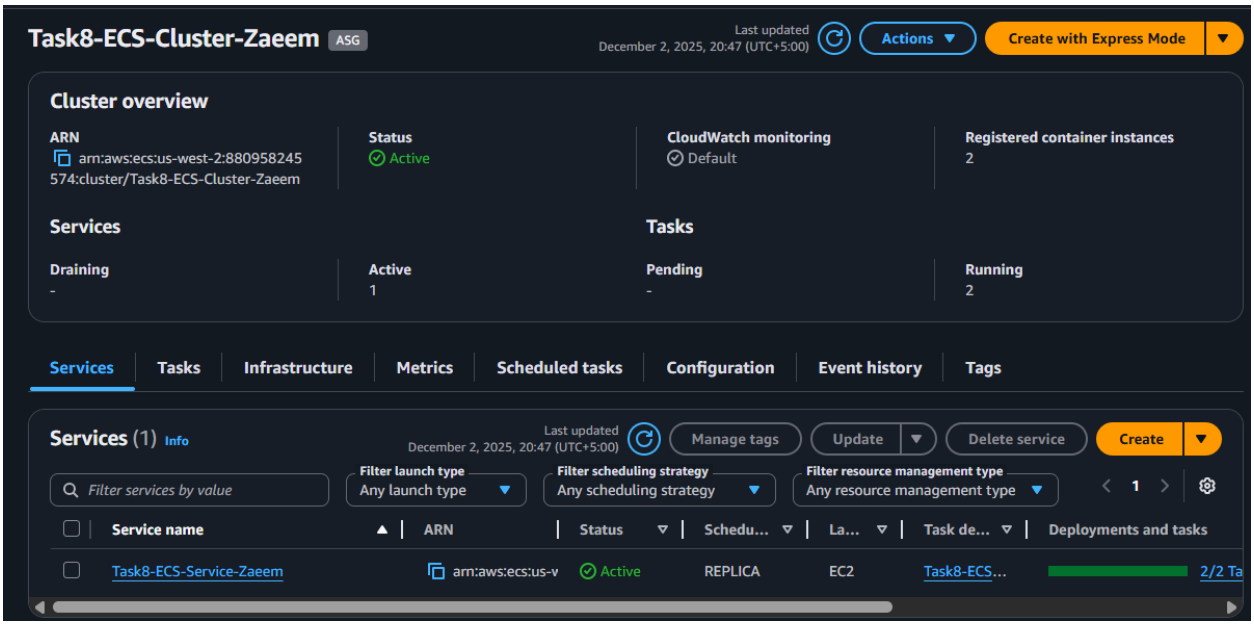
- Create and Configure Application Load Balancer
  - Name: Task8-ALB-Zaeem
  - Scheme: Internet Facing
  - Load Balancer IP: IPv4
  - VPC: Task8-VPC-Zaeem
  - AZ and Subnets: AZ1 Public SN & AZ2 Public SN
  - Security Groups: Task8-ALB-SG-Zaeem
  - Listener Protocol HTTP, Port 80
  - Routing Action: Forward to TG
  - Register Targets: None



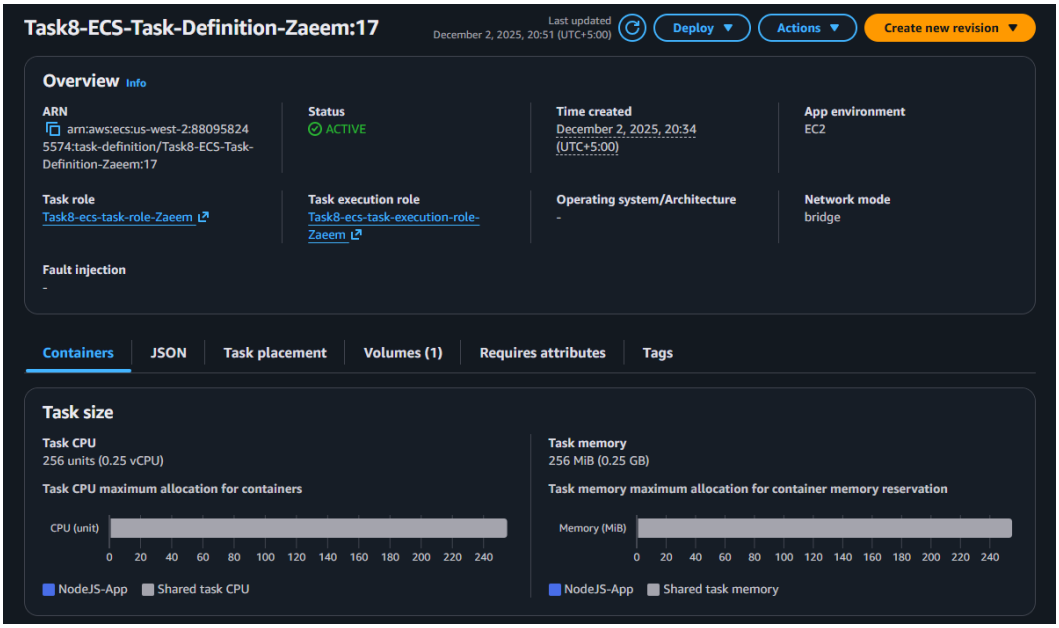
- Create and Configure Auto Scaling Group
    - Name: Task8-ASG-Zaeem
    - Launch Template: Task8-EC2-LT-Zaeem
    - VPC: Task8-VPC-Zaeem
    - AZ: Private SN A & Private SN B
    - Load Balancer: Attach to existing
    - Attach to an Existing Target Group
    - Group Size: Desired capacity 2, min 1, max 3
    - No scling policies



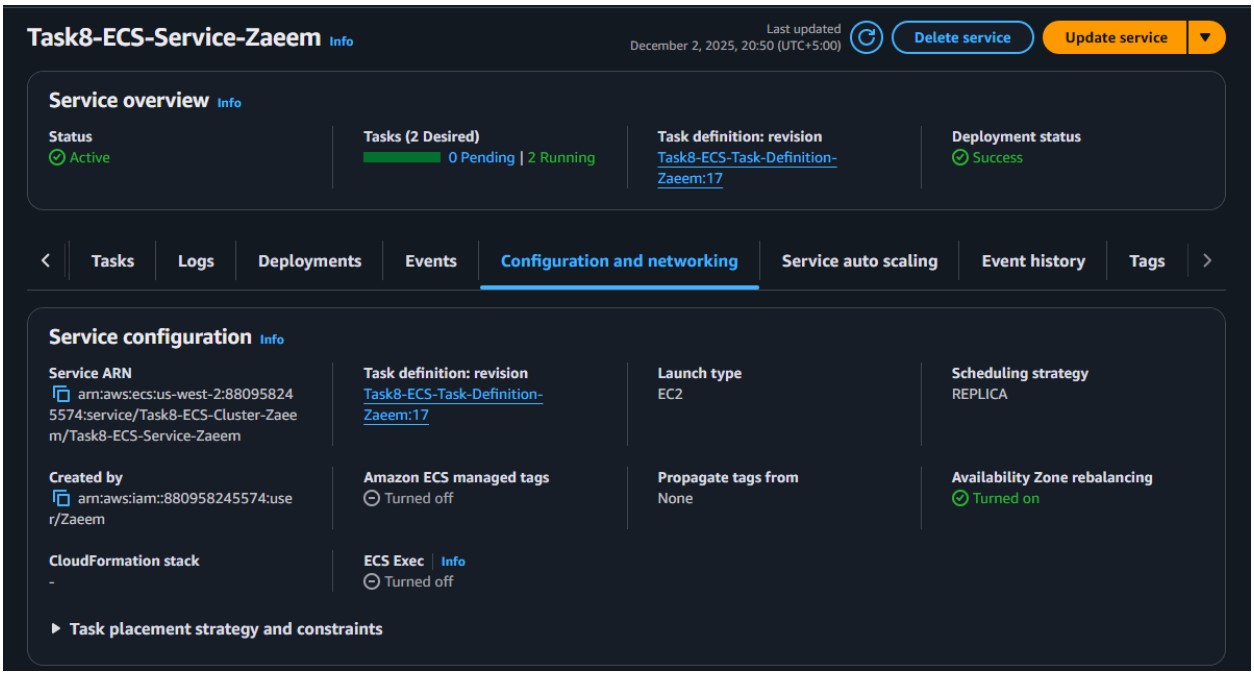
- Create and Configure ECS Cluster
    - Name: Task8-ECS-Cluster-Zaeem
    - Infrastructure: Fargate and self-managed instances
    - Select Task8-ASG-Zaeem Auto Scaling Group



- Create a Task Definition for the ECS Cluster
  - Name: Task8-NodeJS-Zaeem
  - Infrastructure Requirements: Amazon EC2 Instances
  - OS&Arch: Linux/x64, Network Mode: bridge
  - vCPU: 1, Memory: 3GB
  - Select Task role and Task Execution Roles
  - Container 1 (essential): Name NodeJS-App, ECR Image URI, Port Mapping 3000 to 80 for accessing the nodejs application
  - Storage: Configure at task definition, Vol Type: EFS, Enter EFS ID, Enter AP ID



- Create ECS Service to run Task Definition:
    - Family: Task8-NodeJS-Zaeem
    - Svc Name: Task8-NodeJS-Zaeem-Service
    - Compute option: Capacity Provider Strategy, custom: base 2, weight 1
    - Scheduling Strategy: Replica
    - Desired Task: 2
    - Load Balancer: Task8-VPC-Zaeem, Type: ALB, Container: NodeJS-App, Load Balancer: Task8-ALB-Zaeem

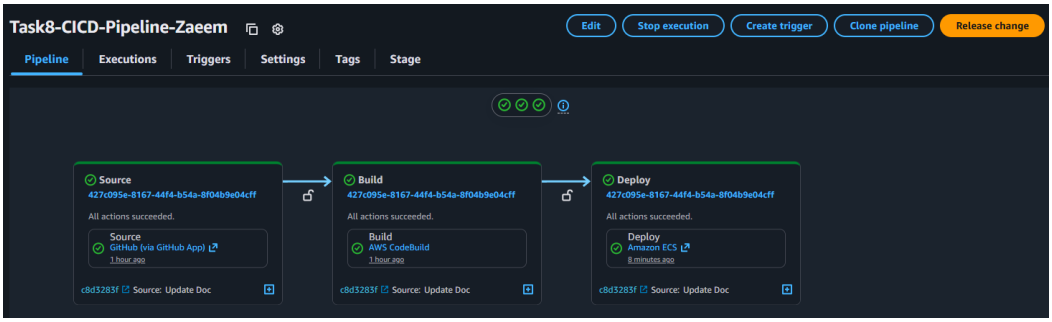


Task8.5: Creating the CodePipeline

- Basic Pipeline Configuration:
  - Use the Build Pipeline button to create a new pipeline.
  - Category: Custom Pipeline
  - Pipeline Name: Task8-Pipeline-Zaeem
  - Execution mode: Queued
  - Service Role: Create new
- Source Stage:
  - Source Provider: GitHub (OAuth via app)
  - Connect to GitHub and allow the pipeline connection
- Build Stage:
  - Build Provider: Other (AWS CodeBuild)
  - Create a new project
  - Set Environment Variables:
    - REGION="us-west-2"
    - ACCOUNT_ID="504649076991"
    - REPO_NAME="zaeem/Task8"
    - IMAGE_TAG="latest"
  - Build Type: Single Build
  - Region: United States Oregon

- Input Artifacts: SourceArtifact
- Place the buildspec.yml file in the root of repo

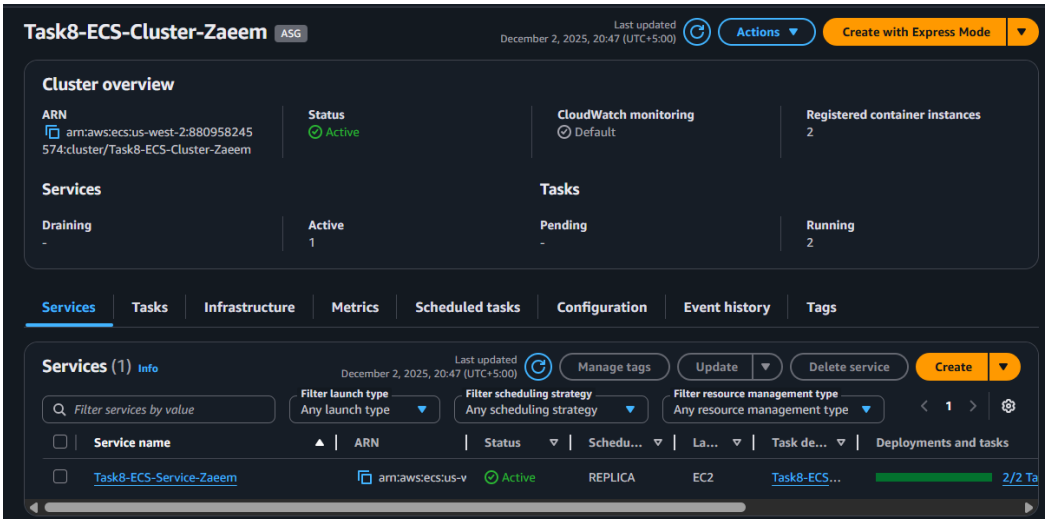
- Deploy Stage (skipping the test stage):
  - Deploy Provider: Amazon ECS
  - Region: United States (Ohio)
  - Input Artifacts: BuildArtifact
  - Place the buildspec.yml file in the root of repo
  - Cluster Name: Task8-ECS-Cluster-Zaeem
  - Service Name: Task8-NodeJS-Zaeem-Service
  - Deployment Time Out: 15 min
  - Create Pipeline



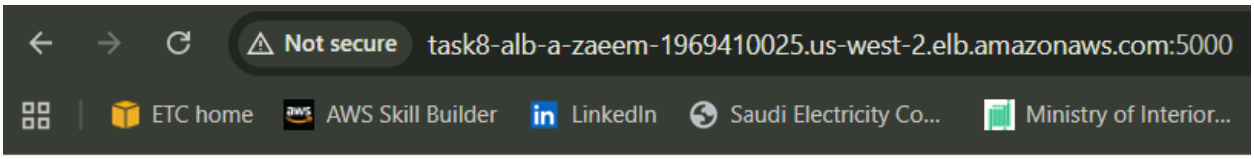
Task8.6: Verification and Testing

- Any commits to the GitHub code repository will trigger the pipeline

- The tasks deployed can be seen under the ECS Service



- The webpage of the NodeJS application can be accessed at the DNS of the ALB



Task8.7: Problems Faced

- The resources of the containers in the task definition should be less the that of the resources set in the Launch Template for the instances otherwise the containers will not be placed and will be stuck in the pending state.
- We must specify artifacts output in the buildspec.yml file to make sure that they are saved in the S3 bucket and passed onto the Deploy Stage otherwise we get an S3 permission error which is not related to permissions at all.
- The pipeline role must have the correct policies attached to carry out all the stages.
- We need to attach the proper permissions to be able to pass the Exec and Task roles to the deploy stage of the codepipeline, as well as the s3 permissions to read the artifact files.
- The name of the container in the appspec.yml file should match exactly the name of the container in the task definition otherwise the project is not deployed.