EMPLOYEE MANAGEMENT APP

CloudFormation Stack Description

This stack provisions a full AWS infrastructure to deploy a containerized React + Spring Boot full-stack application using Amazon EKS, Amazon RDS, Amazon Route 53, CodePipeline, CloudWatch, and other supporting services.

Main Infrastructure Components

---

1. Networking – Amazon VPC

- Custom VPC with:
    - 2 Public Subnets for:
        - ALB (Ingress Controller)
        - NAT Gateway
    - 2 Private Subnets for:
        - EKS node group (frontend & backend pods)
        - RDS MySQL instance
- Associated Route Tables, Internet Gateway, and NAT Gateway
- Elastic IP for NAT Gateway

---

2. Amazon EKS Cluster

- EKS Cluster (AWS::EKS::Cluster) with control plane and networking config
- Node Group (AWS::EKS::Nodegroup) using private subnets (for workloads)
- OIDC Provider enabled for IAM Roles for Service Accounts (IRSA)

---

3. IAM Roles and Policies

- EKS Cluster Role with:
    - AmazonEKSClusterPolicy
- EKS Nodegroup Role with:
    - AmazonEKSWorkerNodePolicy
    - AmazonEC2ContainerRegistryReadOnly

    - AmazonEKS_CNI_Policy
- ALB Ingress Controller Role (IRSA) with custom policy to manage ALB
- CloudWatch Agent Role (IRSA) with:
    - CloudWatchAgentServerPolicy
- CodeBuild Role with:
    - AmazonEC2ContainerRegistryPowerUser
    - AmazonEKSClusterPolicy
    - AmazonS3ReadOnlyAccess
    - CloudWatchLogsFullAccess

---

4. Container Registry – Amazon ECR

- ECR repo for frontend (kaushal-frontend-ecr)
- ECR repo for backend (kaushal-backend-ecr)

---

5. CI/CD – CodePipeline + CodeBuild

- Source: GitHub repository
- CodePipeline with:
    - Source stage (GitHub)
    - Build stage (CodeBuild)
- CodeBuild uses:
    - buildspec.yml to:
        - Build Docker images
        - Push to ECR
        - Deploy to EKS using kubectl apply
- Required IAM Role and permissions included

---

6. Ingress – ALB Ingress Controller

- Installed via Helm or manifest
- Service Account with IRSA Role
- Handles routing based on domain and path

- ALB created dynamically

---

7. Frontend

- Dockerized ReactJS app
- Kubernetes Deployment
- Kubernetes Service (type: LoadBalancer)
- Traffic routed via ALB Ingress Controller
- REST API calls to backend

---

8. Backend

- Dockerized Spring Boot app
- Kubernetes Deployment
- Kubernetes Service (ClusterIP or LoadBalancer)
- Connects to RDS via JDBC
- DB credentials provided via Kubernetes Secrets

---

9. Database – Amazon RDS MySQL

- MySQL instance deployed in private subnets
- DBSubnetGroup created
- Credentials managed using SecretsManager or Kubernetes Secrets
- Backend connects using JDBC

---

10. DNS – Amazon Route 53

- Public Hosted Zone: 2118solutions.info
- A (Alias) Record: frontend.2118solutions.info → ALB DNS
- Automatically resolves to Ingress controller

---

11. Monitoring – Amazon CloudWatch

- CloudWatch Agent deployed as DaemonSet on all EKS nodes
- Collects:

    - Container logs
    - Node-level metrics (CPU, memory, etc.)
- Metrics and logs sent to CloudWatch Log Groups
- Alarms created:
    - High memory/cpu usage
    - Backend unavailability
    - Pod restarts or failures

Optional/Supporting Resources

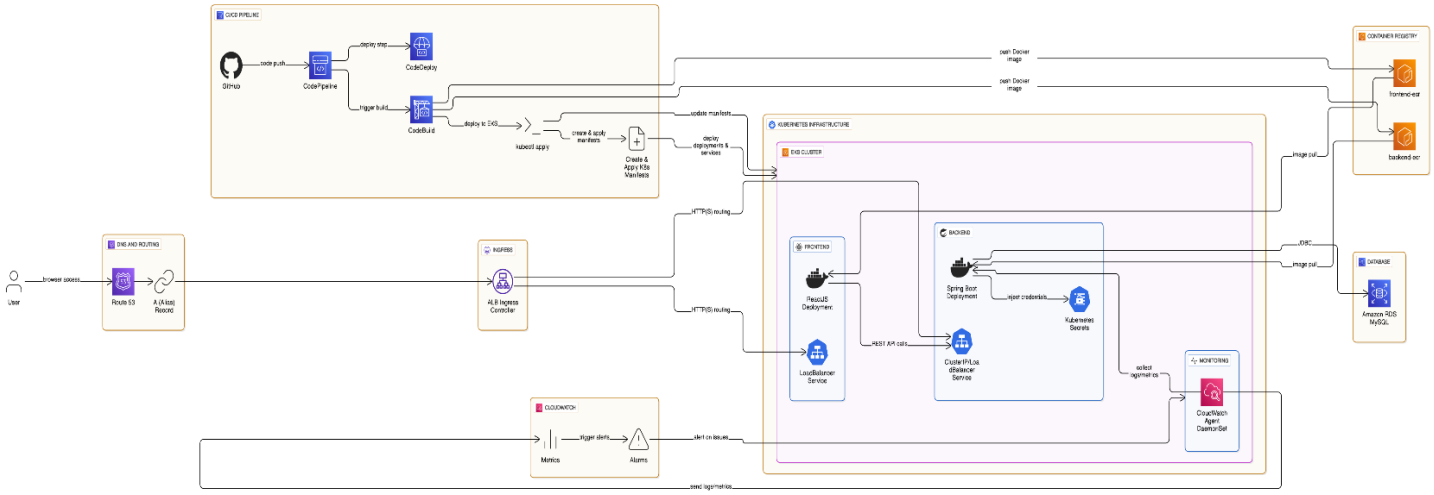
Security Groups

- ALB Security Group
    - Inbound: HTTP (80), HTTPS (443)
    - Outbound: All
- EKS Node Group Security Group
    - Inbound: Pods/ALB as needed
    - Outbound: RDS, Internet (via NAT)
- RDS Security Group
    - Inbound: From EKS Node Group on port 3306 (MySQL)
    - Outbound: All

CloudWatch Log Groups

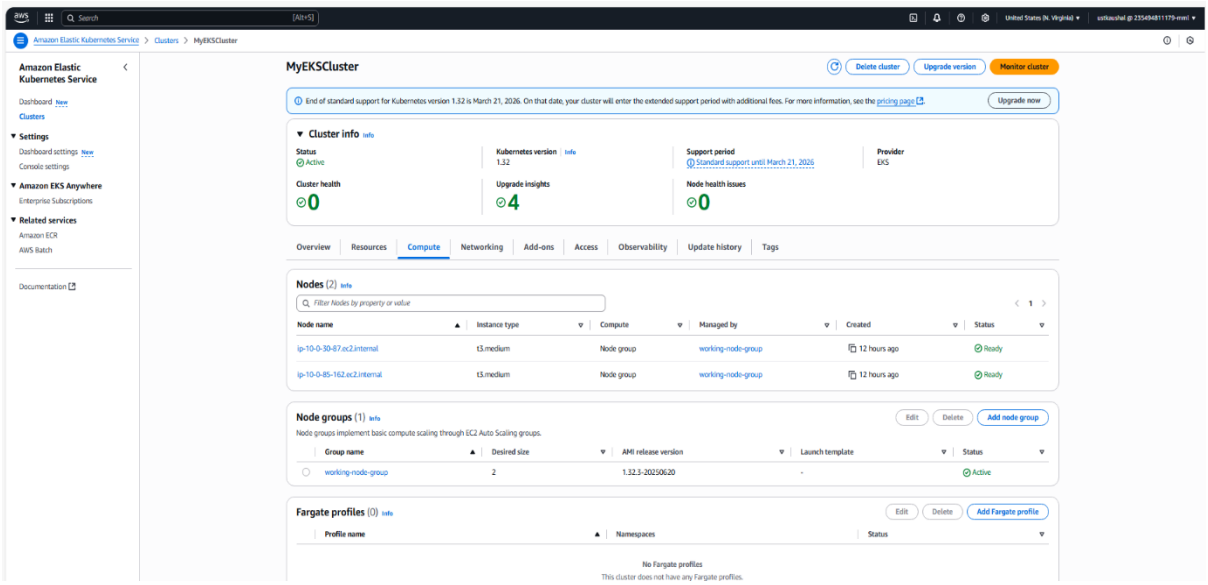
- Log group for:
    - Frontend pods
    - Backend pods
    - Node logs (via CloudWatch Agent)
    - CodeBuild logs (auto-created)

End-to-End DevOps Workflow on AWS using EKS and CI/CD

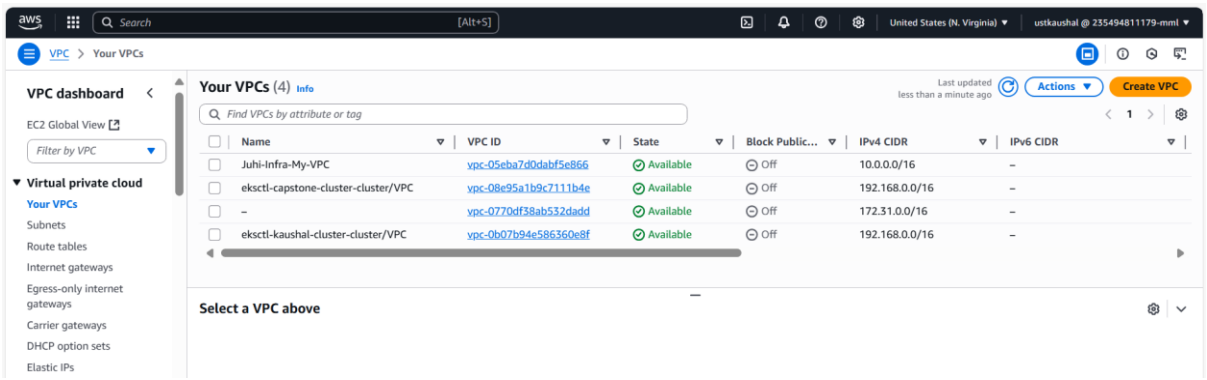


1. Infrastructure Setup using CloudFormation

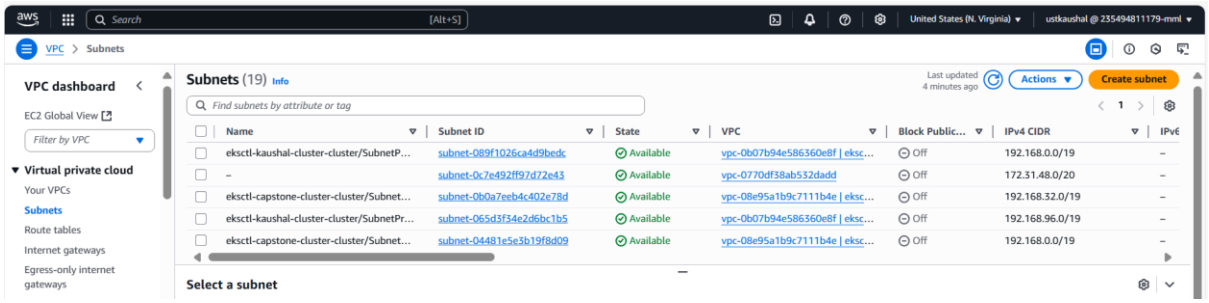
- Created an EKS Cluster with a node group consisting of 2 nodes using AWS CloudFormation



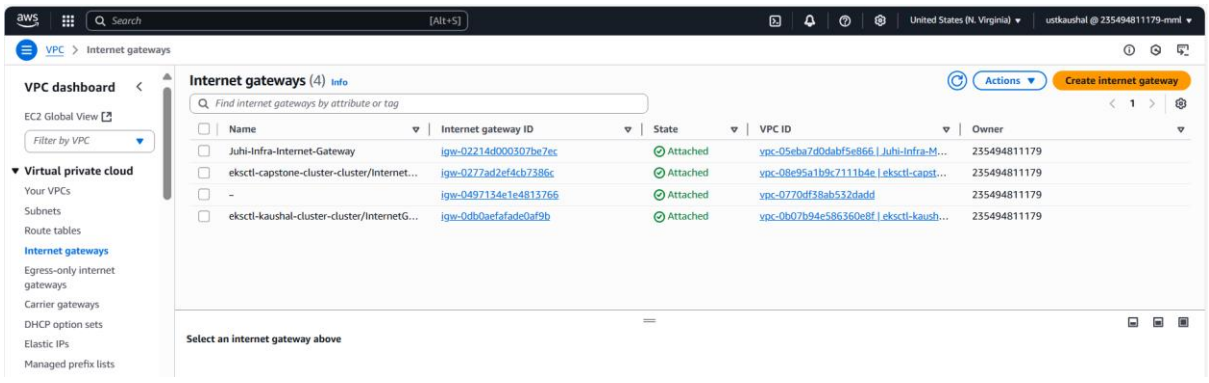
- Created a custom VPC with public and private subnets to securely host application components and control traffic flow within the AWS network.



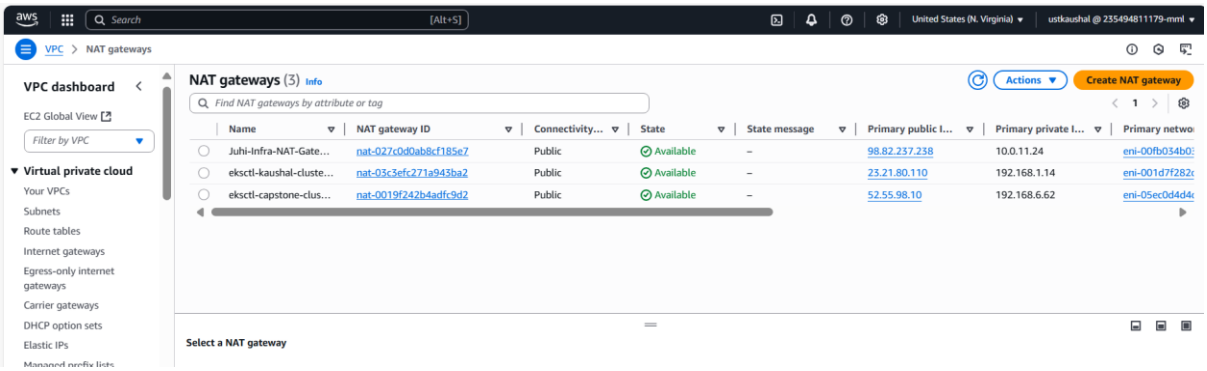
- Provisioned public and private subnets across multiple Availability Zones to ensure high availability and fault tolerance of application resources.



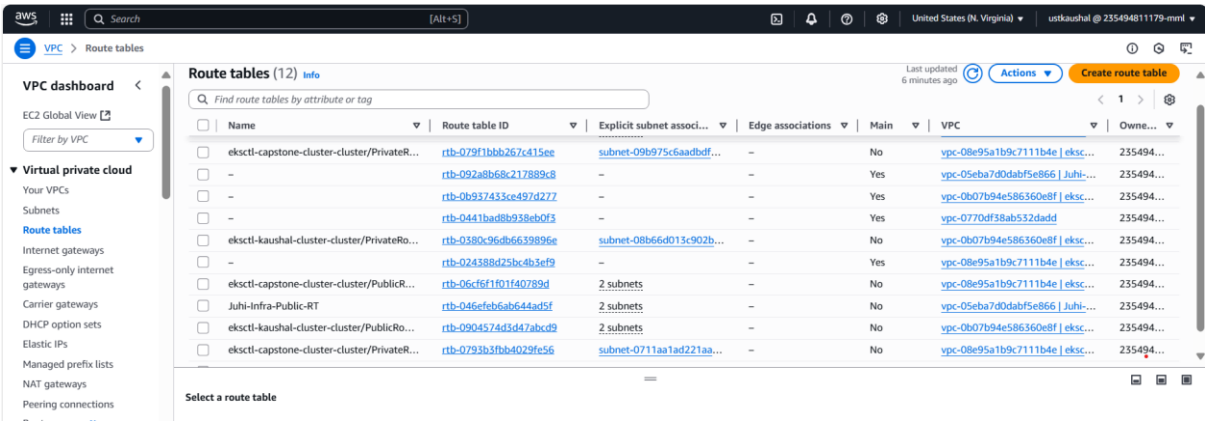
- Attached an Internet Gateway to the VPC to enable internet connectivity for resources within the public subnets.



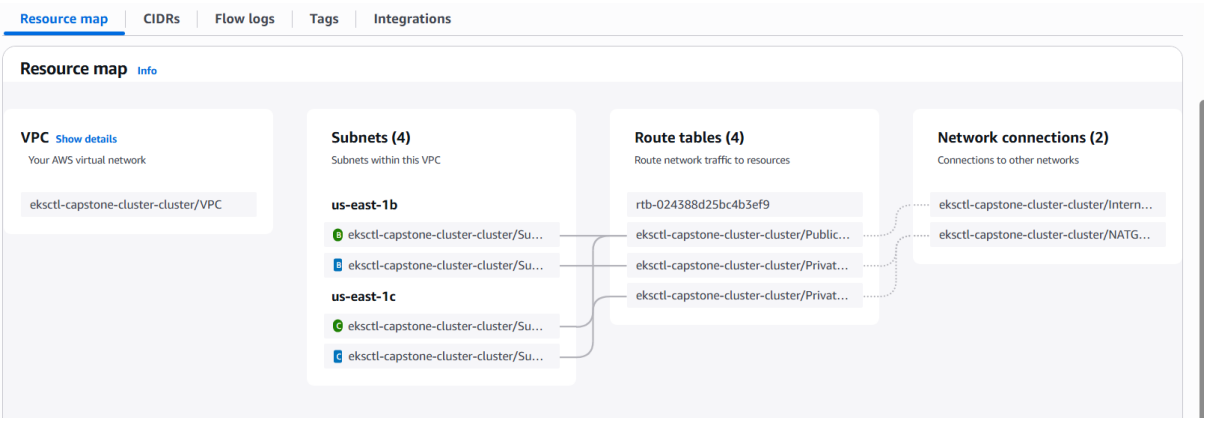
- Provisioned a NAT Gateway in the public subnet to allow instances in private subnets to securely access the internet without exposing them to inbound traffic.



- Configured route tables to manage network traffic flow between public and private subnets, enabling internet access via the Internet Gateway and secure outbound access from private subnets through the NAT Gateway.

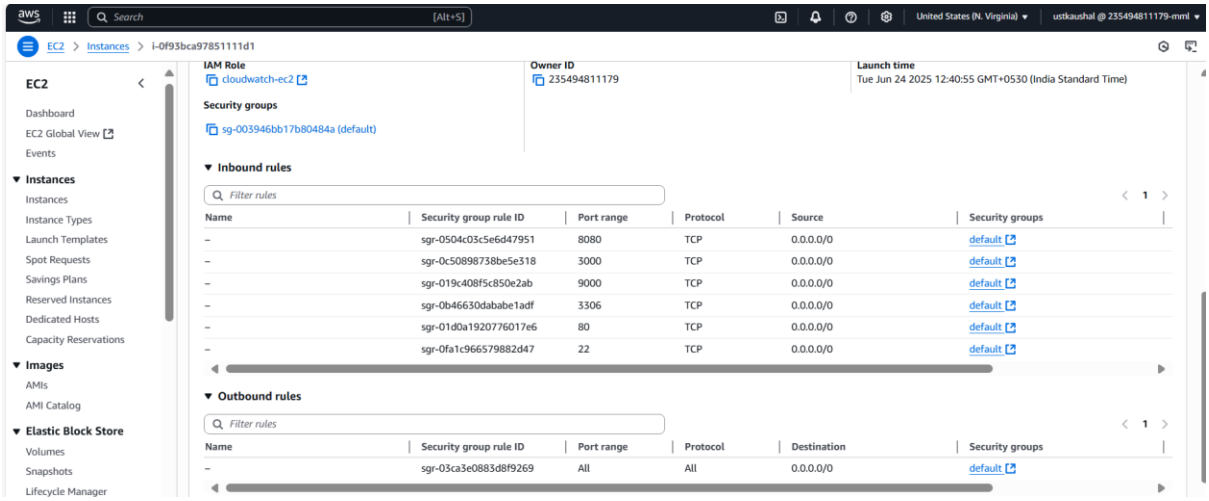


- Designed a resource map within the VPC to logically organize and allocate resources such as subnets, route tables, NAT gateways, and security groups for efficient traffic management and isolation.



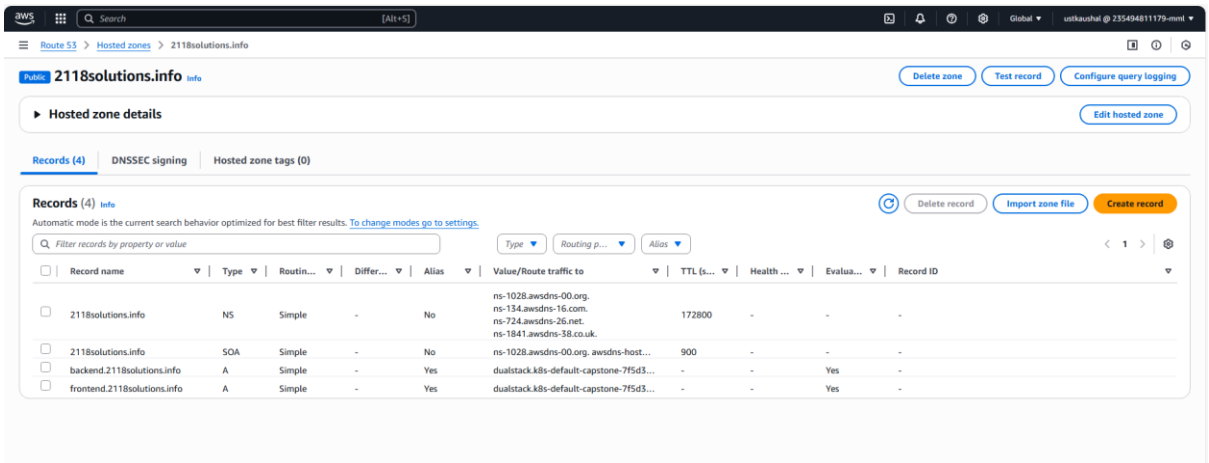
2. Network and Security Configuration

- Configured security groups to control inbound and outbound traffic:
  - **Frontend (React)**: Allowed inbound traffic on port **3000** for IP-based access during development, and on port **80/443** for domain-based access in production.
  - **Backend (Spring Boot)**: Allowed inbound traffic on port **8080** from the frontend security group.
  - **RDS (MySQL)**: Allowed inbound traffic on port **3306** only from the backend security group for secure database connectivity.
  - **SonarQube**: Allowed inbound traffic on port **9000** to enable access to the code quality dashboard.



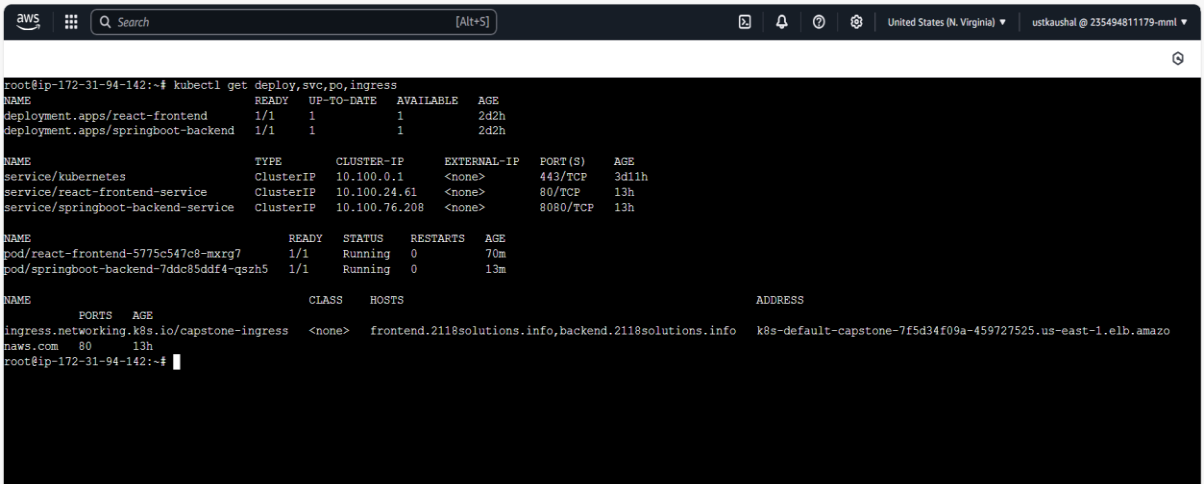
3. DNS Management

- Created a Route 53 hosted zone and configured alias records pointing to the ALB DNS to enable domain-based access for frontend and backend services.

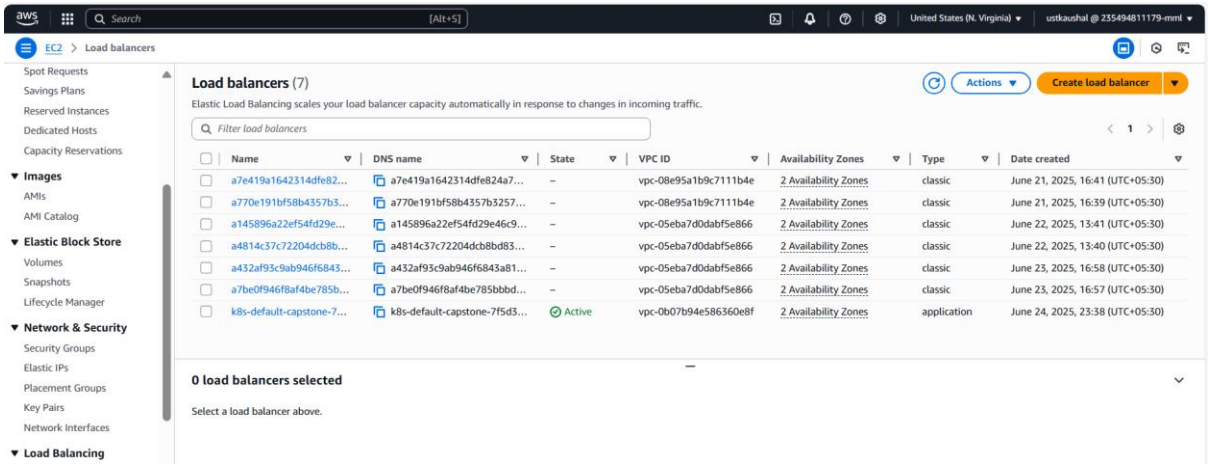


4. Application Deployment on EKS

- Created Kubernetes Deployment, Service, and Ingress resources for frontend and backend services.

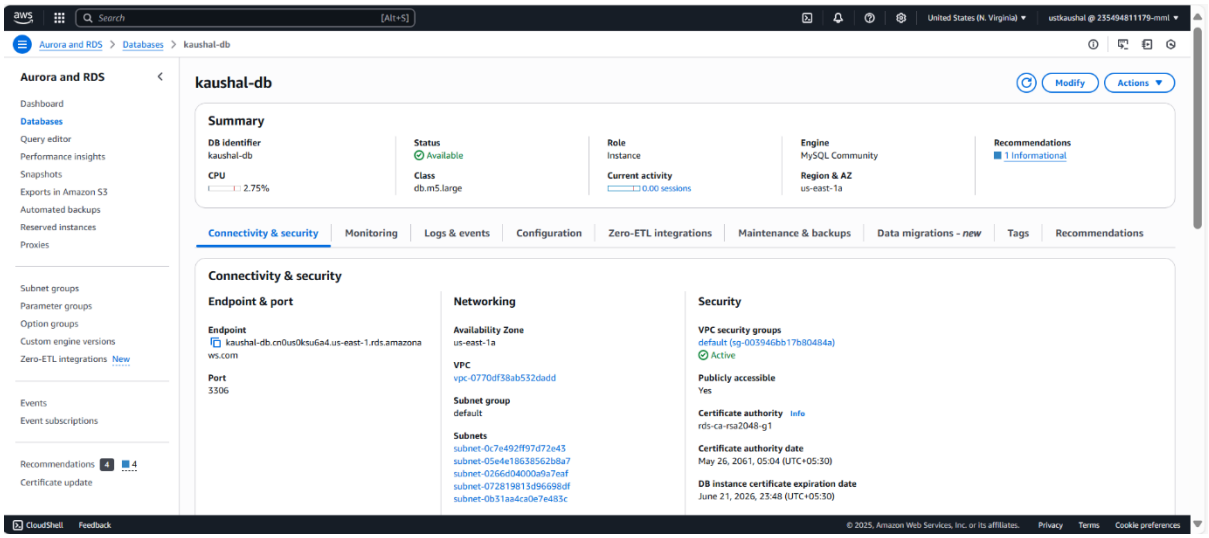


- Configured a single ALB via Kubernetes Ingress to route traffic to multiple services using host-based rules for frontend.2118solutions.info and backend.2118solutions.info, ensuring centralized load balancing and efficient traffic management.

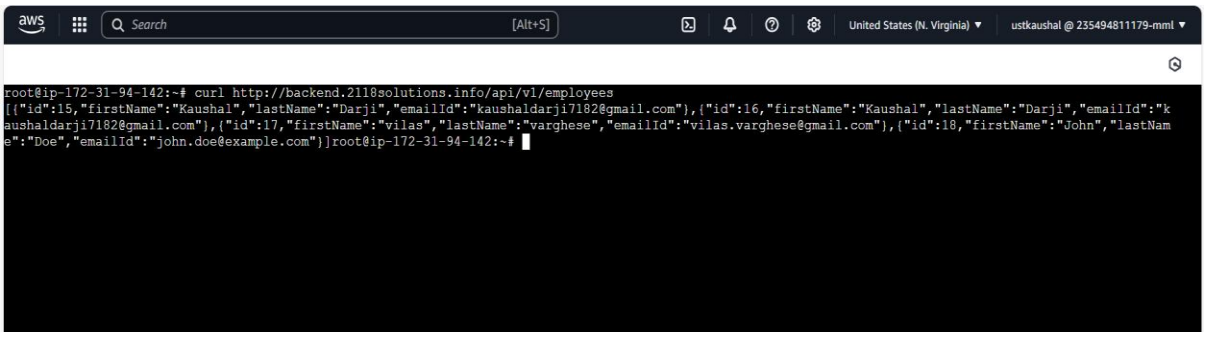


5. Database Integration

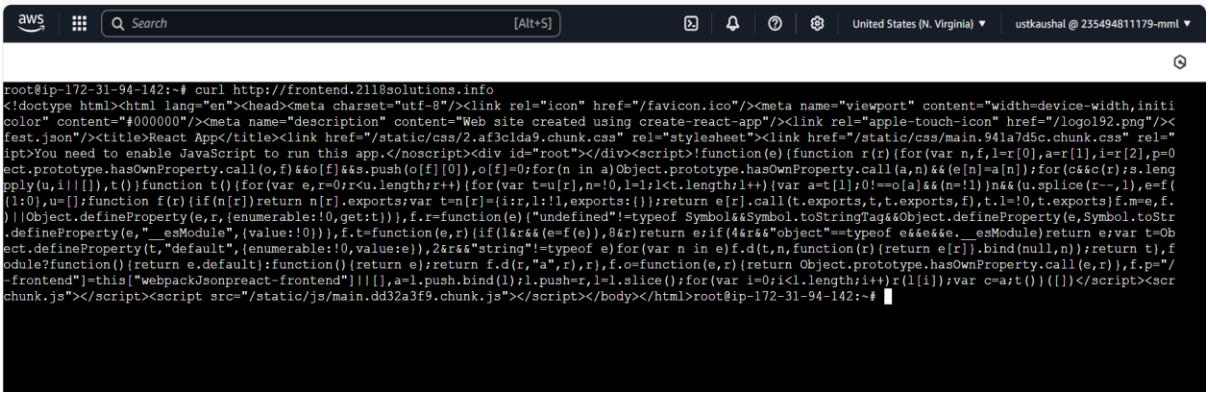
- Integrated Amazon RDS (MySQL) for scalable, managed relational database services with high availability and automated backups.



- Executed curl command internally on the backend to verify API availability and confirm database connectivity.

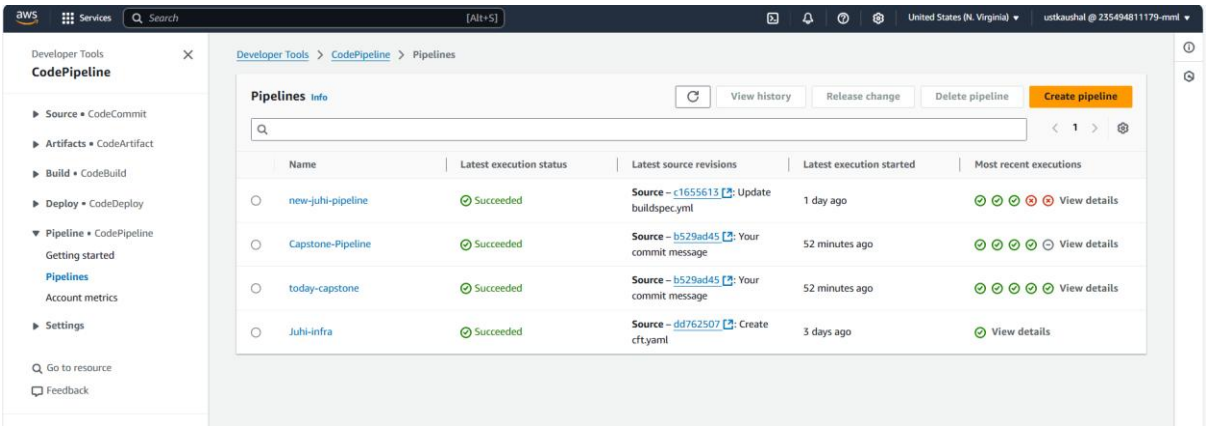


- Executed curl command internally on the frontend to verify service responsiveness and endpoint health.

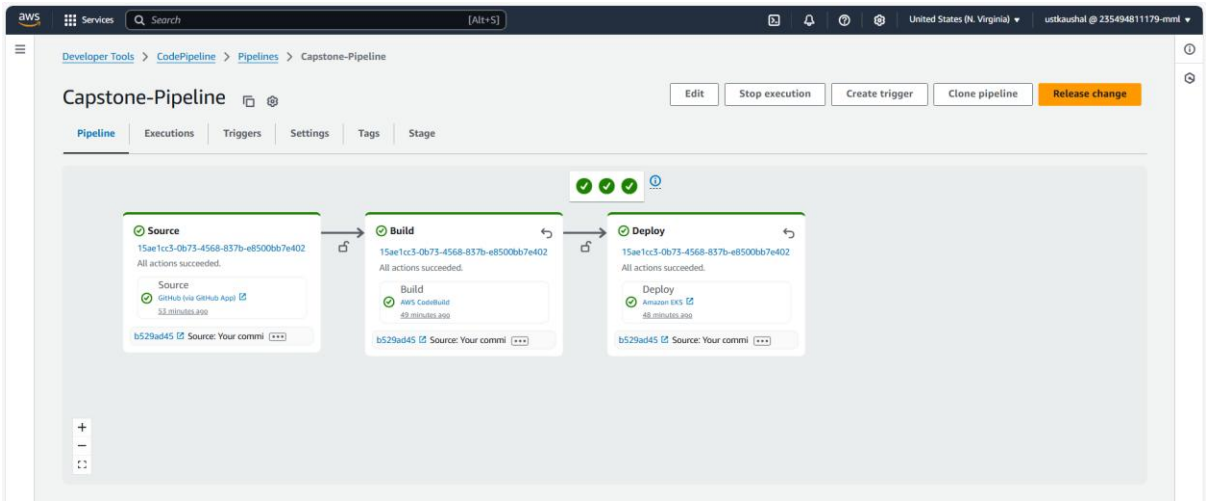


6. CI/CD Automation

- Implemented a CI/CD pipeline using AWS CodePipeline and CodeBuild to automate Docker image builds, push to Amazon ECR, and deploy to EKS using Kubernetes manifest

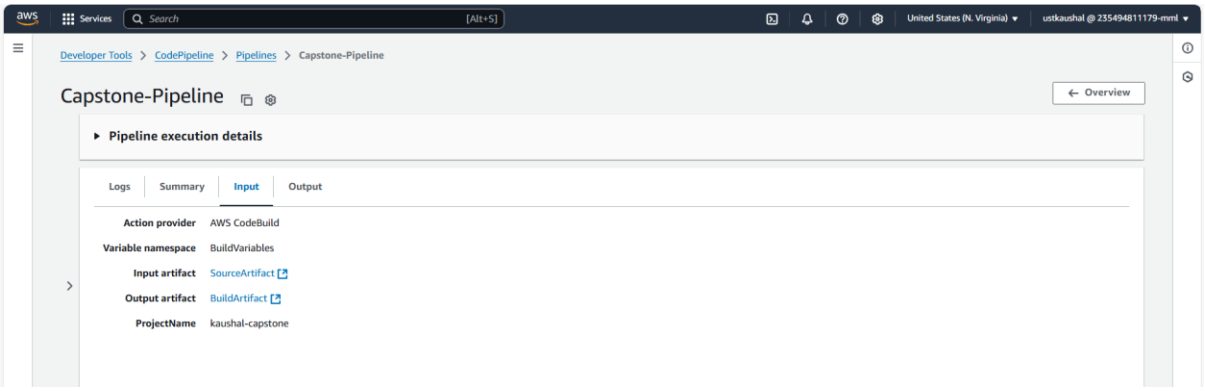


- Successfully executed the CI/CD pipeline, automating the end-to-end build, image deployment to ECR, and application rollout to the EKS cluster.

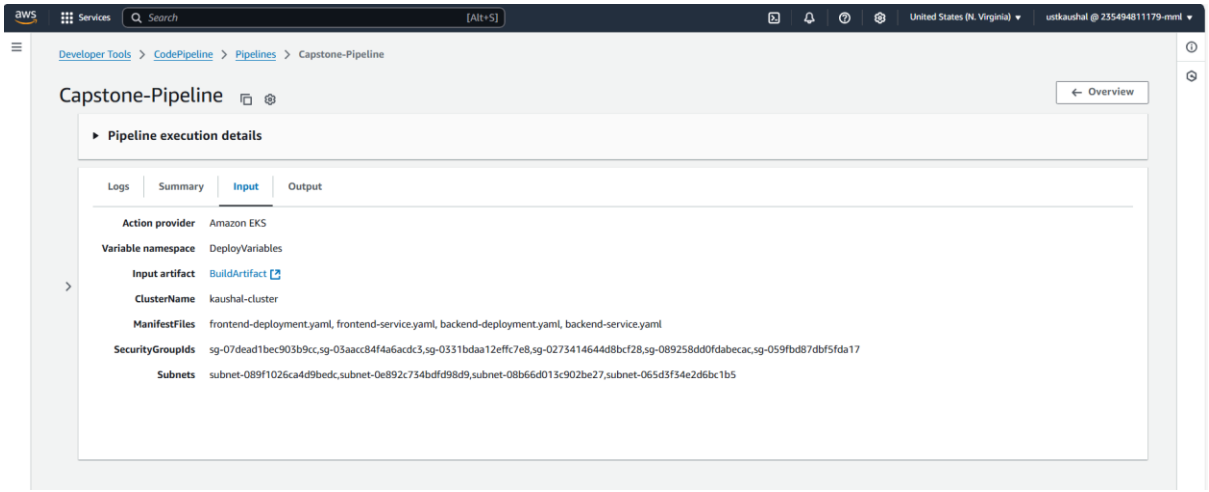


7. Build & Deployment Details

- AWS CodeBuild successfully generated source and build artifacts by building Docker images and pushing them to Amazon ECR.

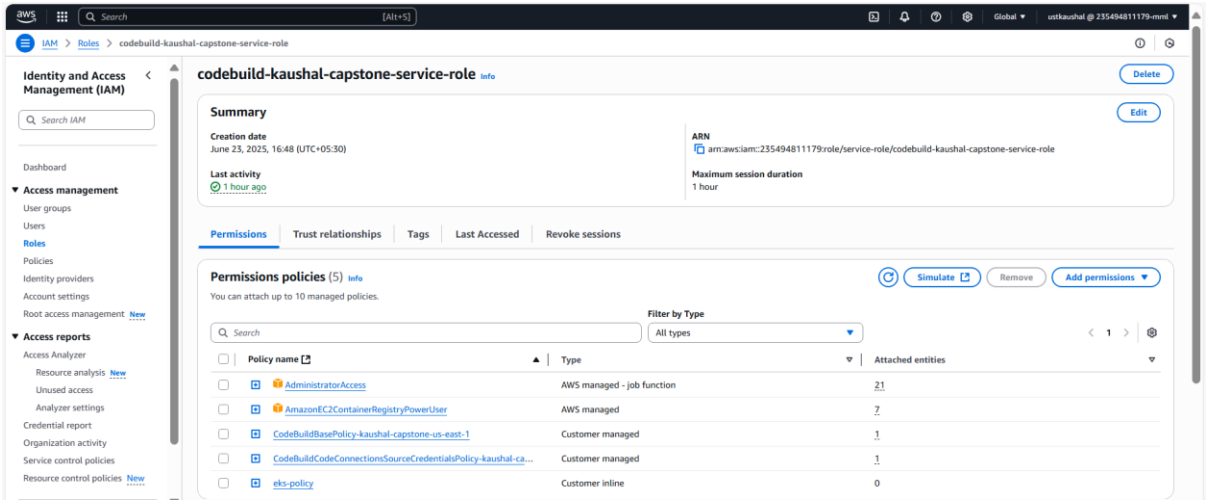


- Deployment phase used these artifacts to apply Kubernetes manifests, deploying updated containers to the EKS cluster via kubectl.

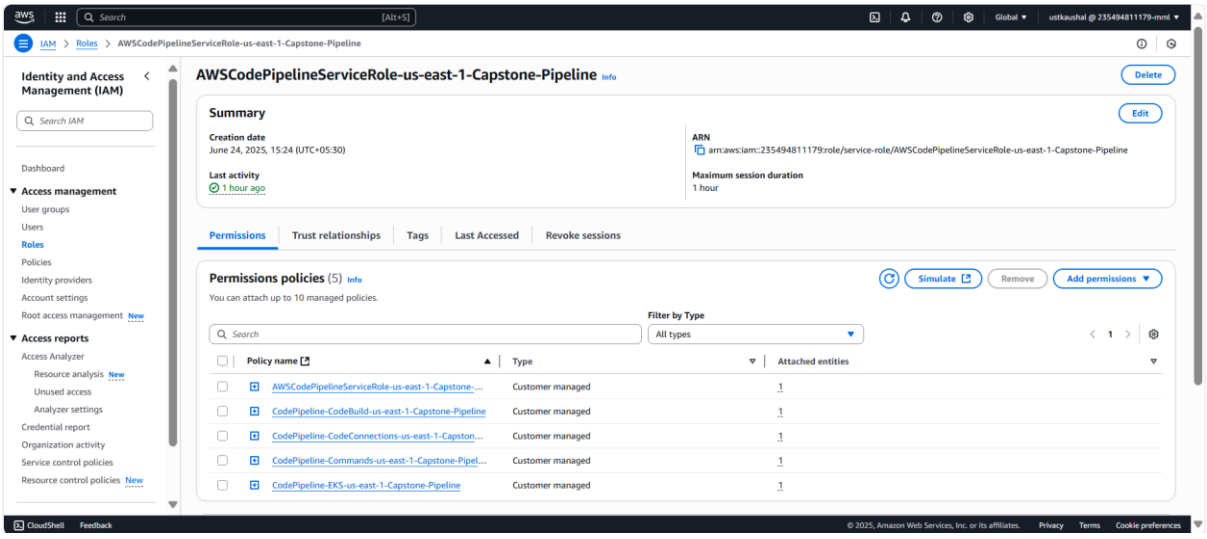


8. IAM Roles & Permissions

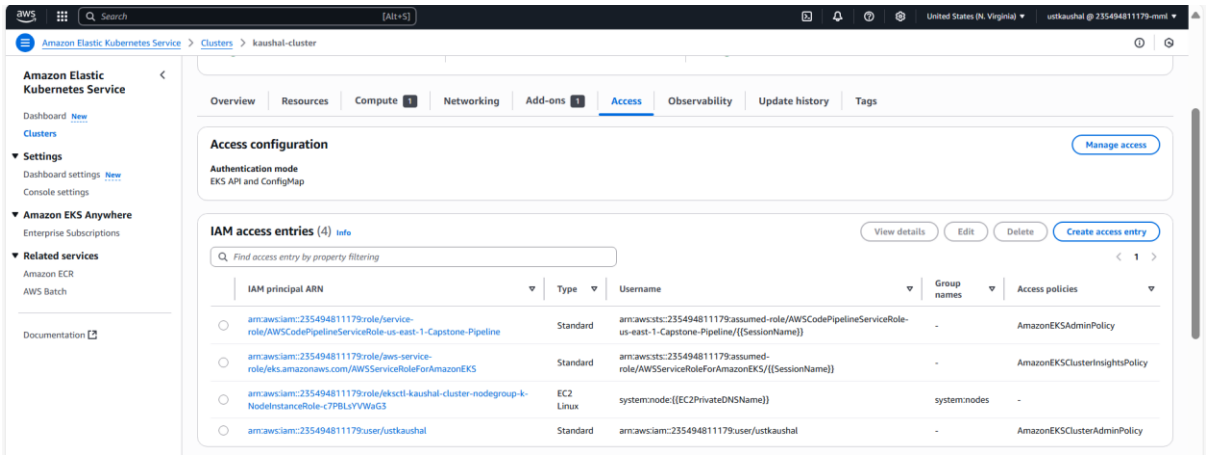
- Configured an IAM role for CodeBuild with access to ECR, S3, CodePipeline, and EKS for secure build and deployment.



- Attached a custom IAM policy to the CodeBuild role for necessary permissions (ECR push/pull, S3, etc.)

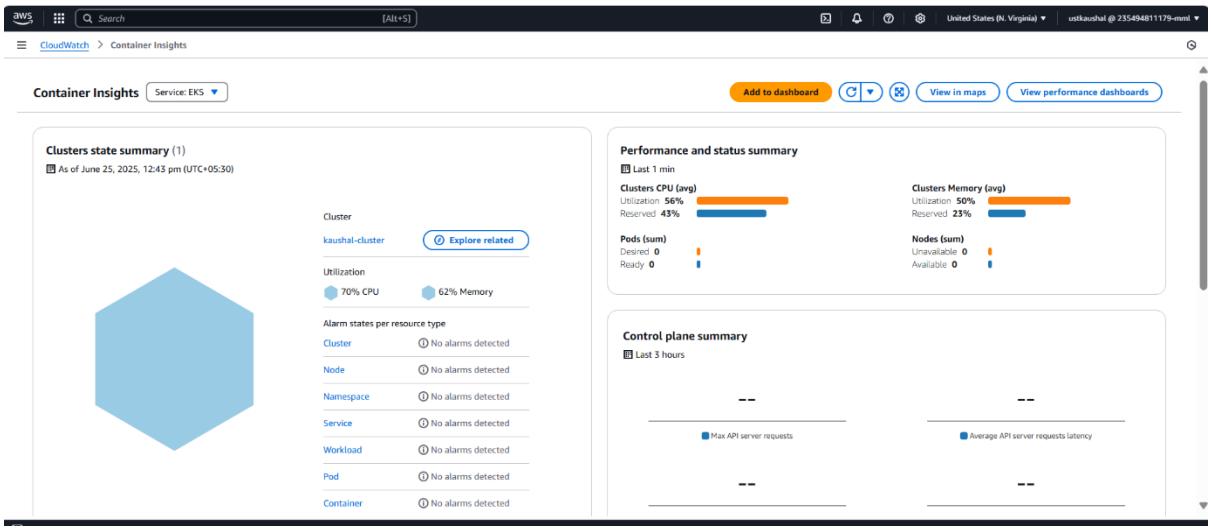


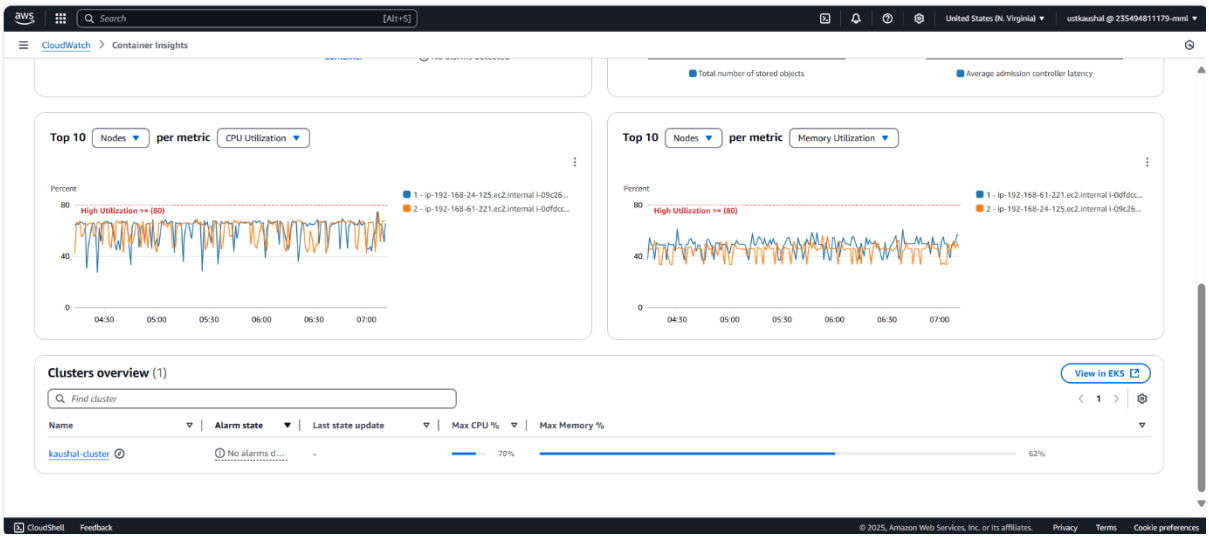
- Added the CodePipeline IAM role to the EKS cluster by updating the aws-auth ConfigMap and attached the EKS admin policy to grant full access to cluster resources.



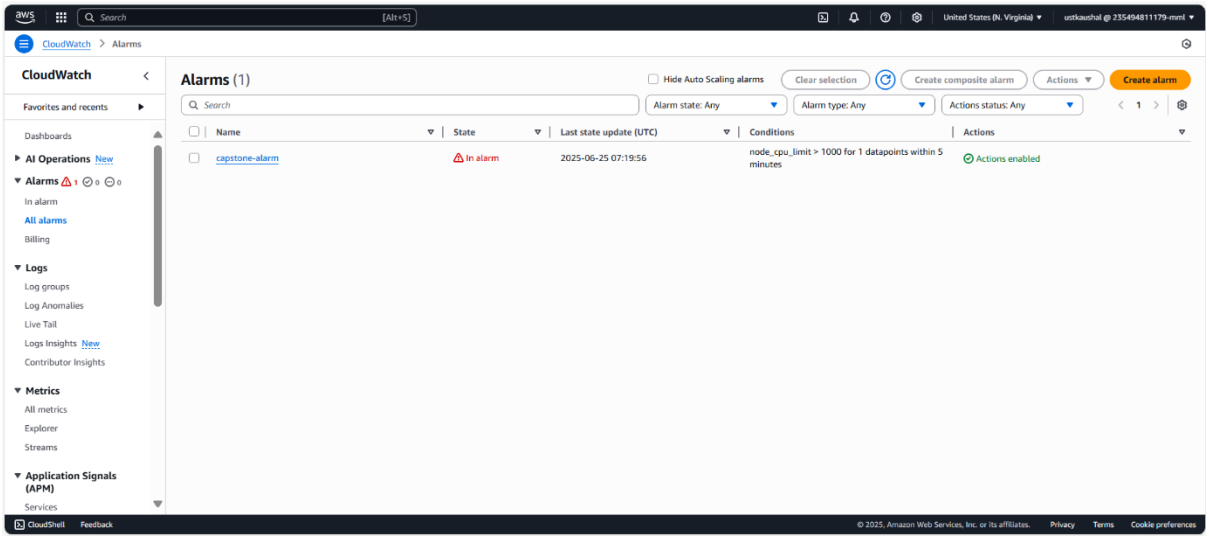
9. Monitoring and Alerting

- Created a **CloudWatch Dashboard** to monitor EKS resource usage, performance, and limits.

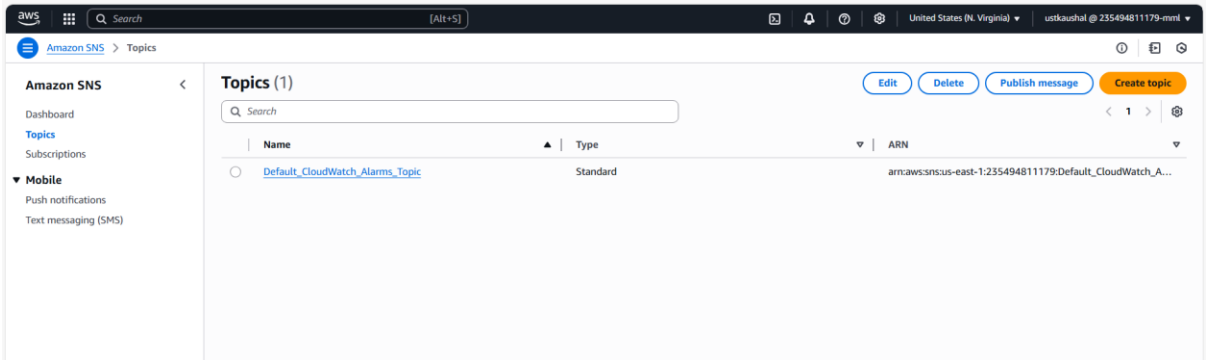




- Configured **CloudWatch Alarms** to monitor key application and infrastructure metrics and trigger alerts on anomalies.

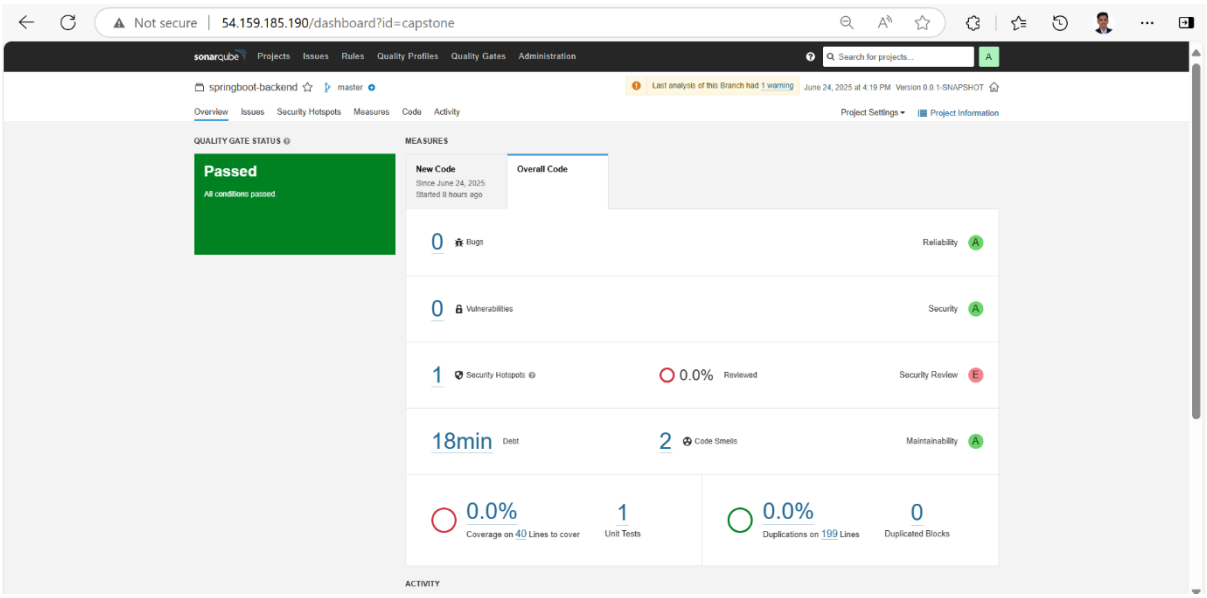


- Created an **Amazon SNS topic** and integrated it with CloudWatch Alarms to enable real-time notifications and alerts.



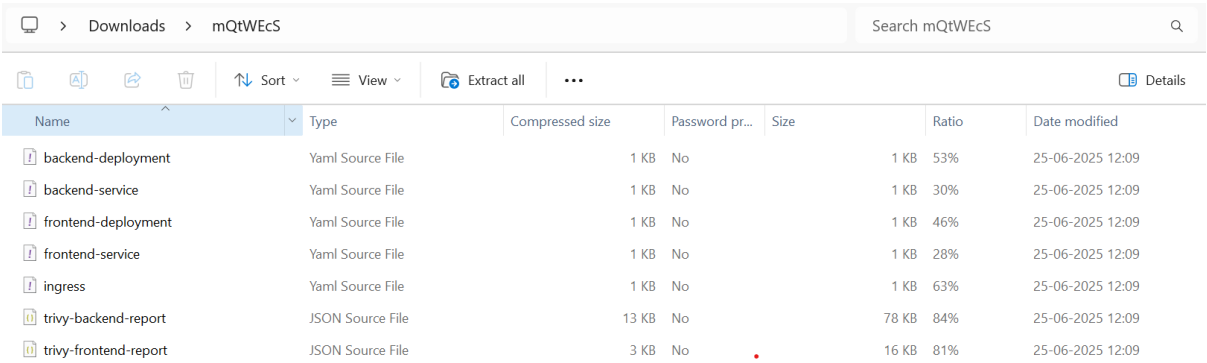
10. Quality Assurance

Integrated **SonarQube** into the CI/CD pipeline to enforce automated **code quality checks** and maintain development standards.



11. Image Vulnerability Scanning

- Integrated Trivy into the build process to scan both frontend and backend Docker images for vulnerabilities.
- Generated JSON scan reports for each image and included them in the CodeBuild build artifacts to ensure security transparency and traceability.



Name	Type	Compressed size	Password pr...	Size	Ratio	Date modified
backend-deployment	Yaml Source File	1 KB	No	1 KB	53%	25-06-2025 12:09
backend-service	Yaml Source File	1 KB	No	1 KB	30%	25-06-2025 12:09
frontend-deployment	Yaml Source File	1 KB	No	1 KB	46%	25-06-2025 12:09
frontend-service	Yaml Source File	1 KB	No	1 KB	28%	25-06-2025 12:09
ingress	Yaml Source File	1 KB	No	1 KB	63%	25-06-2025 12:09
trivy-backend-report	JSON Source File	13 KB	No	78 KB	84%	25-06-2025 12:09
trivy-frontend-report	JSON Source File	3 KB	No	16 KB	81%	25-06-2025 12:09

The screenshot displays the **Capstone-CodeCommit** repository in AWS CodeCommit, which includes the project structure and files for a React frontend and Spring Boot backend full-stack CRUD application.