Capstone Project Documentation

Cloud-Native Disaster Recovery and Infrastructure Automation using AWS

Author: Koppineni Harish Arjun

Table Of Contents

Project Overview

This project implements a **highly available**, **fault-tolerant**, and **disaster-resilient** infrastructure for a **production-grade web application**, built on **Amazon Web Services (AWS)**. The architecture supports **multi-region failover**, fully **automated CI/CD pipelines**, and **robust monitoring and security practices** using modern DevOps tools and principles.

The infrastructure is provisioned using **Infrastructure as Code (IaC)** with **AWS CloudFormation** and **Terraform**, and integrates tightly with AWS services for resilience, scalability, and automation.

✅ Key Features

- **Multi-Region High Availability**
  - Primary Region: us-east-1
  - Secondary Region: us-west-1
  - Disaster recovery using **Route 53 DNS Failover**
- **End-to-End CI/CD Automation**
  - Powered by **AWS CodePipeline**, **CodeBuild**, **CodeDeploy**
  - Integrated with **ECR** for container registry
- **Real-Time Monitoring and Alerts**
  - Metrics and logs via **Amazon CloudWatch**
  - Alerting through **Amazon SNS**
- **Security Hardening**
  - IAM roles with least-privilege principle
  - Fine-grained **Security Groups**, **NACLs**, and **KMS encryption**
- **DevSecOps Integrations**
  - **SonarQube** for static code analysis
  - **Trivy** for container vulnerability scanning

Application Stack

Tier	Technology
Frontend	Thymeleaf (HTML/CSS rendered)
Backend	Spring Boot (REST APIs + MVC)
Database	Amazon RDS MySQL (Multi-AZ)
Containerization	Docker
Orchestration	Amazon EKS (Kubernetes)
Load Balancing	Application Load Balancer (ALB)
Networking	VPC with public/private subnets, NAT Gateways, Route Tables

Part 1: Project Initialization and Prerequisite Setup

1.1. Objective Definition

The primary goal is to deploy a web application across two AWS regions (us-east-1 as Primary, us-west-1 as Secondary) to ensure high availability and robust disaster recovery capabilities. All infrastructure will be provisioned using code, and application deployments will be fully automated.

1.2. Required Tools and Access

Before beginning, ensure the following are installed, configured, and accessible:

- **AWS Account:** Administrator-level access to an AWS account.
- **AWS CLI:** Authenticated with aws configure.
- **Git & GitHub Account:** Git installed, and a personal GitHub account available.
- **Terraform:** Latest version installed.
- **kubectl:** Kubernetes command-line tool installed.
- **Docker Desktop:** For local container testing (Optional).

1.3. Source Code Preparation

1. **Fork the Repository:** Create a personal fork of the official application repository from https://github.com/arjunkoppineni/AWS_CapStone_Final_Project.git.

2. **Clone Locally:** Clone your forked repository to your local development machine.

   git clone <your-forked-repository-url>

Part 2: Primary Region Infrastructure Deployment (us-east-1)

This section covers provisioning the entire infrastructure stack in the us-east-1 region using AWS CloudFormation.

2.1. Review CloudFormation Template

Navigate to the region-1-cloudformation/ directory in your cloned repository. Examine the infra.yaml file to understand the resources being created:

Networking Layer

Resource	Description
VPC (Virtual Private Cloud)	Custom VPC created for the application in each region with a /16 CIDR block (e.g., 10.0.0.0/16), providing private IP addressing for all AWS resources.
Subnets (6 total)	Split across **2 Availability Zones** for high availability: - **2 Public Subnets** (1 per AZ) for ALB and NAT Gateways - **4 Private Subnets** (2 per AZ) for EKS worker nodes and RDS databases
Internet Gateway (IGW)	Attached to the VPC to allow internet access from the public subnets (e.g., for ALB).
NAT Gateways (2)	Placed in each AZ's public subnet, allowing EKS worker nodes and private RDS instances to securely reach the internet (for patching, pulling container images, etc.).
Route Tables	Custom route tables for public and private subnets: - Public subnet route table forwards traffic to the IGW. - Private subnet route tables forward outbound traffic through NAT gateways.

Compute Layer: Amazon EKS

Resource	Description
Amazon EKS Cluster	Fully managed Kubernetes control plane (1 per region) that orchestrates pods and deployments for the Spring Boot application.
EKS Node Group (Managed)	Auto-scaled group of EC2 instances (typically t3.medium or m5.large) managed by EKS for compute workloads. It spans multiple subnets for HA.
Security Groups for EKS	Controls inbound/outbound traffic: - Allow only ALB-to-node communication (e.g., ports 80/443) - Nodes can access RDS (port 3306) - EKS worker nodes have open internal cluster communication within VPC

IAM (Identity & Access Management)

Role	Usage & Permissions
EKS Cluster Role	Grants permissions for EKS control plane to interact with other AWS services (e.g., ELB, Auto Scaling, etc.). - AmazonEKSClusterPolicy, AmazonEKSServicePolicy
EKS Node IAM Role	Attached to the EC2 worker nodes to allow access to Amazon ECR and other AWS resources. - AmazonEC2ContainerRegistryReadOnly, AmazonEKSWorkerNodePolicy, AmazonEKS_CNI_Policy, CloudWatchAgentServerPolicy
CodeBuild Role	Executes pipeline build jobs. Has permissions to: - Build/push Docker images - Pull Secrets - Update EKS (via kubectl) - Needs inline policy for eks:DescribeCluster, eks:UpdateConfigMap - Needs AmazonEC2ContainerRegistryPowerUser
CodePipeline Role	Coordinates actions between GitHub (source), CodeBuild (build), and EKS (deploy). Includes permissions for: - codecommit:*, codebuild:*, eks:*, s3:*
CloudFormation Execution Role	Created during stack deployment. Allows CFN to create/modify AWS resources.

2.2. Execute CloudFormation Stack Creation

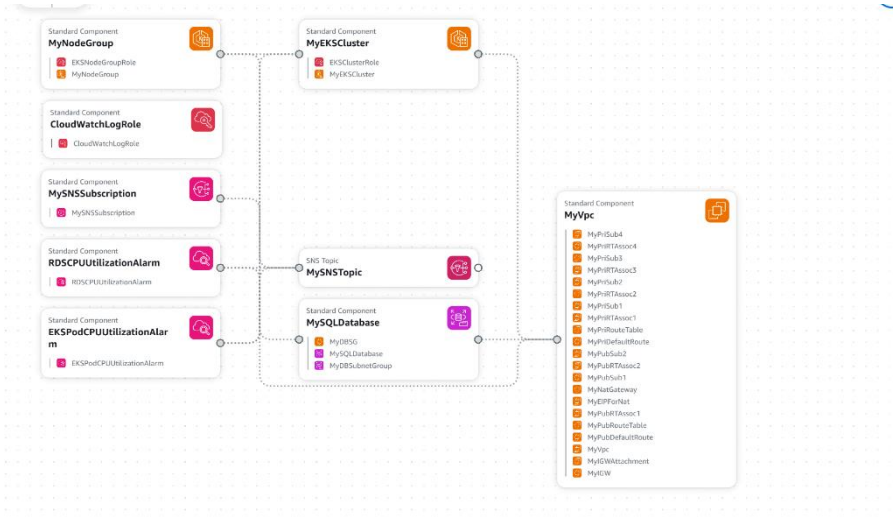
1. **Navigate to AWS Console:** Log in and go to the CloudFormation service in the **N. Virginia (us-east-1)** region.
2. **Initiate Stack Creation:** Click **Create stack** > **With new resources (standard)**.
3. **Upload Template:** Select **Upload a template file** and choose the infra.yaml file.
4. **Specify Parameters:**
    - **Stack name:** Provide a unique name (e.g., prod-east-stack).
    - **Parameters:** Carefully fill in the EKS cluster name, database name, and master credentials. **Securely store the database password.**
5. **Acknowledge and Create:** Proceed through the options, acknowledge the IAM resource creation at the final review step, and click **Create stack**. *(**Note:** This process takes approximately 20 minutes.)*

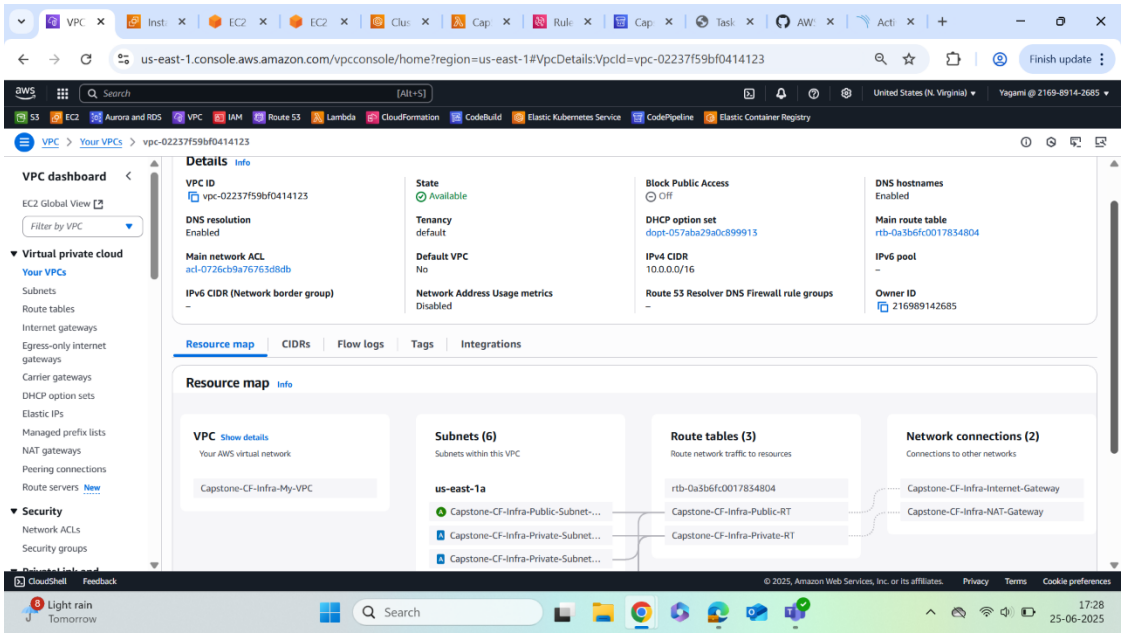
2.3. Verify EKS Cluster Connectivity

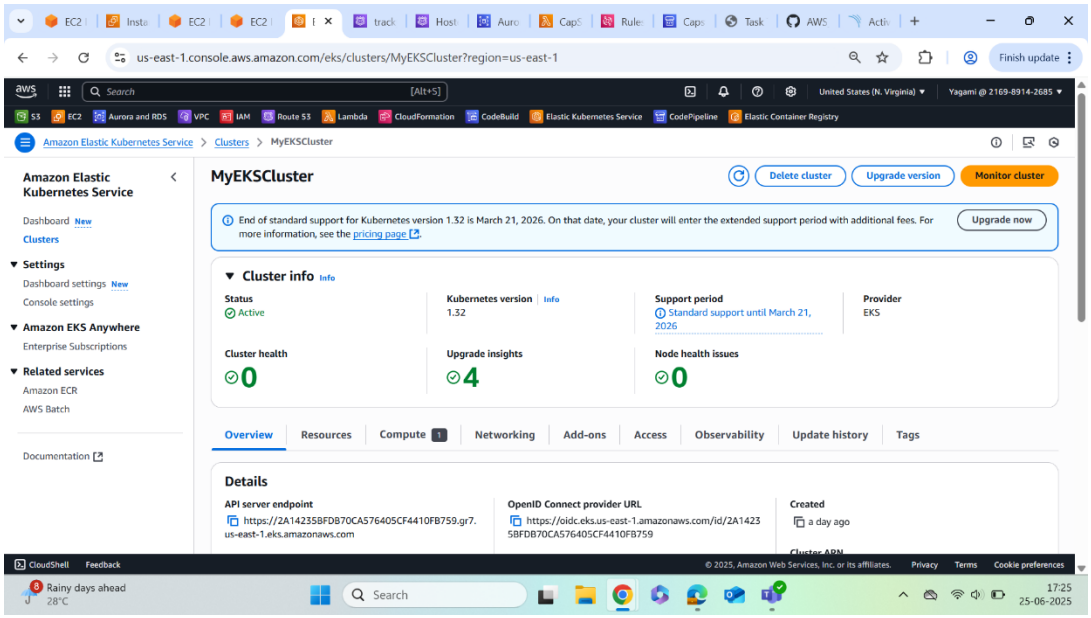
1. Once the stack status is CREATE_COMPLETE, configure kubectl to communicate with the new cluster.
2. aws eks update-kubeconfig --region us-east-1 --name <your-eks-cluster-name>
3. Confirm that worker nodes are connected and ready.
4. kubectl get nodes

The output should list two nodes with a Ready status.

This are the resources that are created from the template







Part 3: Secondary Region Infrastructure Deployment (us-west-1)

This section details the provisioning of the failover infrastructure in us-west-1 using Terraform.

3.1. Prepare Terraform Configuration

Navigate to the terraform/ directory. The configuration files here mirror the CloudFormation setup using Terraform

3.2. Execute Terraform Deployment

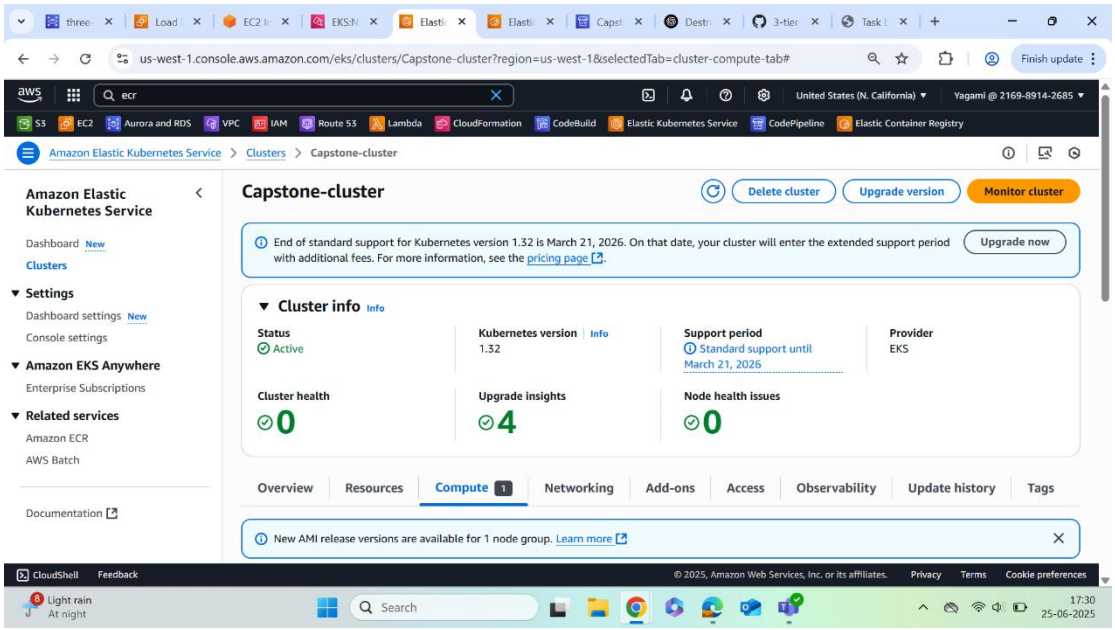
1. **Initialize Terraform:** Open a terminal in the region-2-terraform directory and run:
2. terraform init
3. **Plan Deployment:** (Optional but Recommended) Preview the resources created.

4. terraform plan
5. **Apply Configuration:** Provision the infrastructure. Type yes when prompted to
6. terraform apply *(**Note:** This process also takes approximately 20 minutes.)*

3.3. Verify Secondary EKS Cluster

1. Configure kubectl for the new us-west-1 cluster.
2. aws eks update-kubeconfig --region us-west-1 --name <your-region-2-CN>
3. kubectl get nodes

*(**Tip:** Use kubectl config get-contexts to see and kubectl config use-context <name> to switch between your primary and secondary clusters.)*



Part 4: CI/CD Pipeline Configuration and Automation

This section details the setup of an automated pipeline to build and deploy the application to the primary EKS cluster.

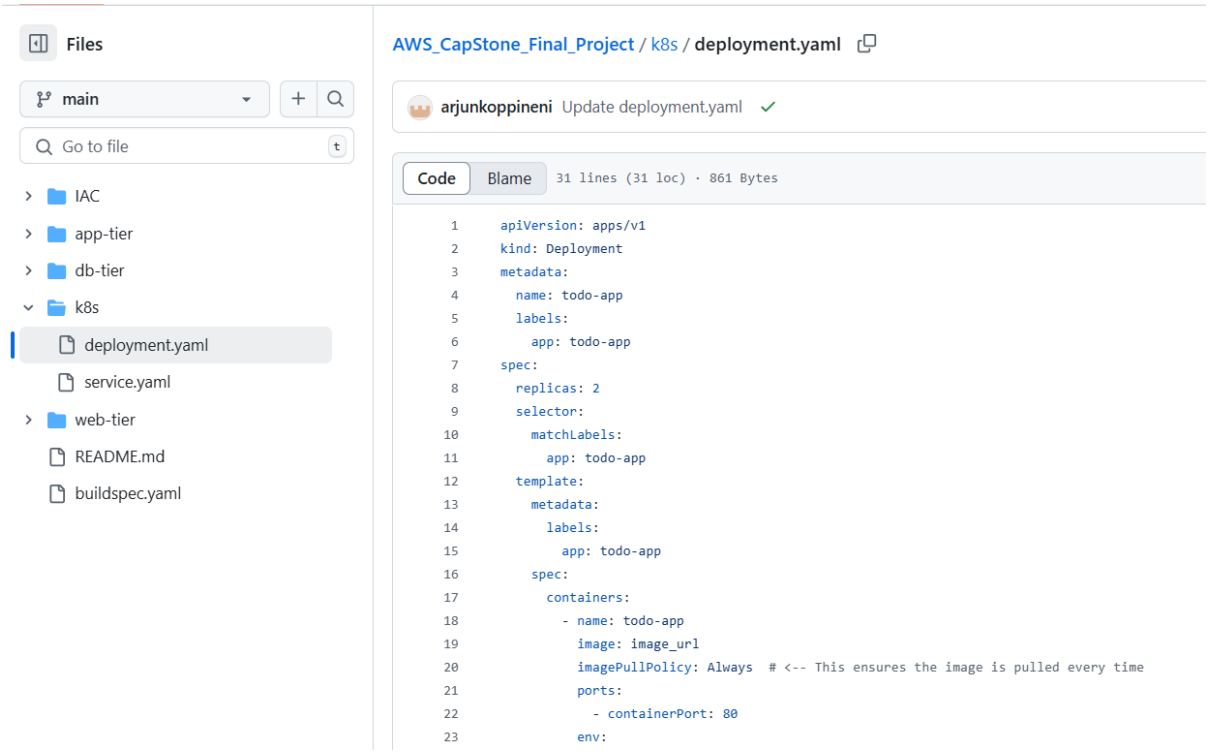
4.1. Create ECR Repository

1. In the AWS Console, navigate to **Elastic Container Registry (ECR)** in the us-east-1 region.
2. Create a new **private** repository. Name it my-app or similar.
3. Note the **repository URI** for later use.

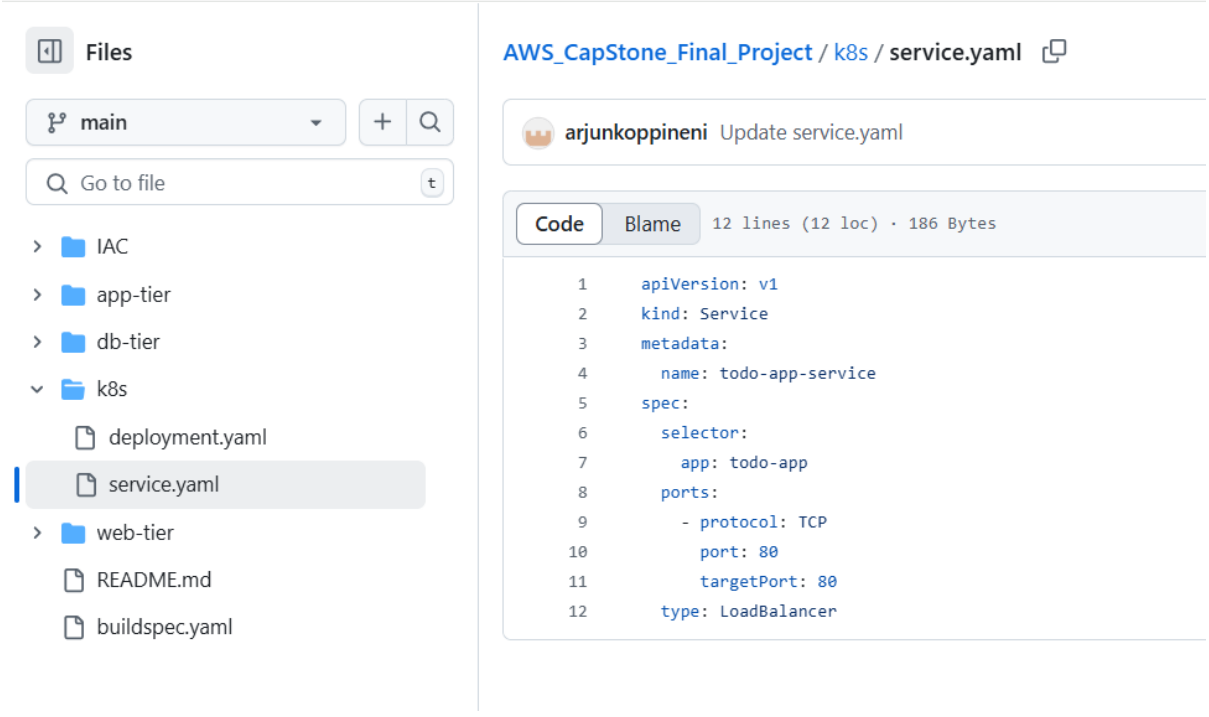
4.2. Create Kubernetes Manifests

In the root of your repository, create a k8s/ directory and add two files:

- **k8s/deployment.yaml:**



k8s/service.yaml:



4.3. Configure the buildspec.yml

Update the buildspec.yml file in your repository to contain the complete build, push, and deploy logic. Ensure the placeholder DOCKER_IMAGE_PLACEHOLDER is used.

4.4. Build the AWS CodePipeline

1. In the AWS Console, navigate to **CodePipeline** and start the creation wizard.
2. **Stage 1: Source:** Connect to your forked GitHub repository (main branch).
3. **Stage 2: Build:**
    - Select **AWS CodeBuild** as the provider.
    - Click **Create project** and configure it:
        - **Environment:** Use a managed **Amazon Linux 2** image.
        - **Privileged: Enable** this flag for building Docker images.
        - **Service Role:** Create a new service role. **You must edit this role later.**
        - **Buildspec:** Use the buildspec.yml from your source code.
    - Add the **ECR_REPOSITORY_URI** and **EKS_CLUSTER_NAME** as environment variables in the CodeBuild project configuration.

Static Code Analysis & Container Security

To ensure high code quality and secure container images before deploying to production, the CI/CD pipeline integrates:

SonarQube – Static Code Quality Checks

- **Purpose**: Analyzes codebase for code smells, bugs, vulnerabilities, and maintainability.
- **Integration**:
    - SonarQube is triggered during the build stage of CodeBuild.
    - SonarQube scanner is configured via environment variables: SONAR_TOKEN, SONAR_HOST_URL.
    - Fails the build if quality gates are not met.

- **SonarQube Scan** : Make sure SonarQube server is publicly reachable (or hosted in VPC) and credentials are securely stored in AWS Secrets Manager or CodeBuild environment variables.

Trivy – Container Vulnerability Scanning

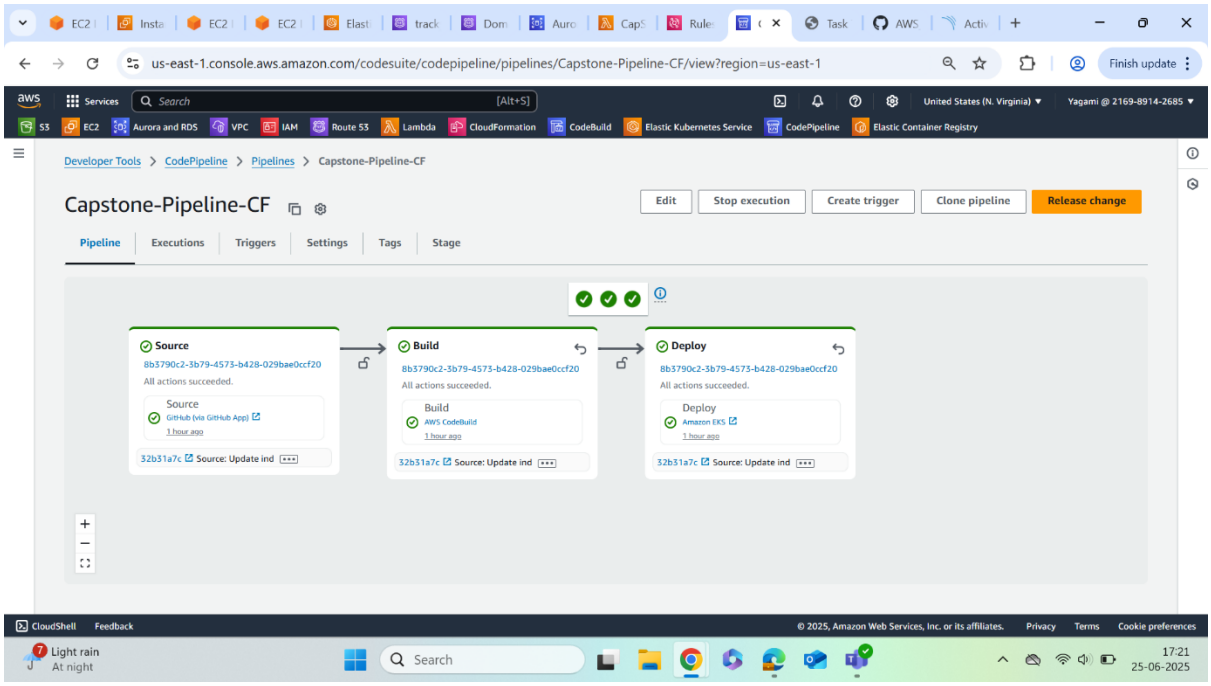
- **Purpose**: Scans container images for vulnerabilities (CVEs), misconfigurations, and secrets before pushing to ECR.
- **Integration**:
    - Used in CodeBuild phase after image build, before pushing to ECR.
    - Fails the build if critical vulnerabilities are found.
- **Trivy Scan**

4. **Stage 3: Deploy: Deploy to EKS,** It Deploys the new Docker image to the EKS cluster by applying updated Kubernetes manifests.
5. **Create** the pipeline.

4.5. Grant Permissions to CodeBuild Role

1. Navigate to **IAM** > **Roles**.
2. Find the service role created for your CodeBuild project.
3. Attach the necessary permissions:
    - AmazonEC2ContainerRegistryPowerUser (to push to ECR).
    - An inline policy allowing eks:DescribeCluster.
4. **Crucially, authorize the role in EKS:** Add the CodeBuild role's ARN to the aws-auth ConfigMap in your us-east-1 EKS cluster to grant it kubectl permissions.

After commit to GitHub → Pipeline triggers → CodeBuild builds, pushes, deploys.



These steps should be replicated in Region 2 (us-west-2) to ensure consistent infrastructure and deployment configuration

Part 5: Route 53 Global DNS Failover

5.1 Create Hosted Zone

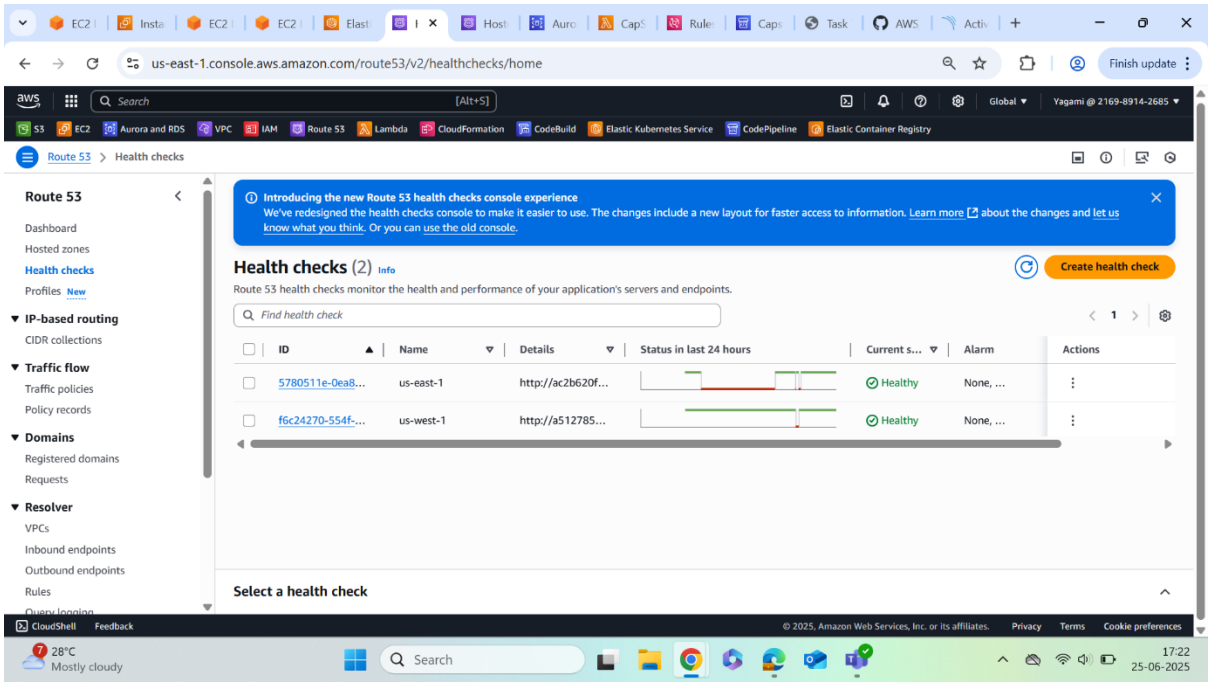
1. Go to Route 53 → Hosted zones → Create hosted zone
2. Enter domain (use registered one or subdomain for test)

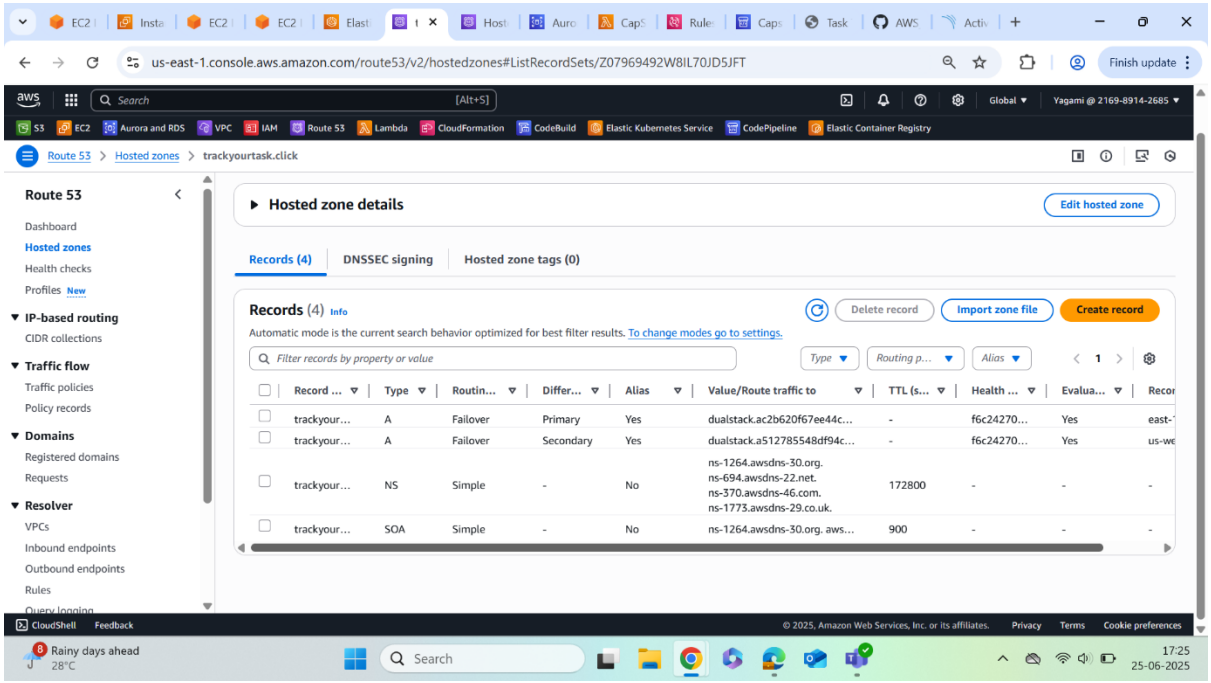
5.2 Configure DNS Failover Records

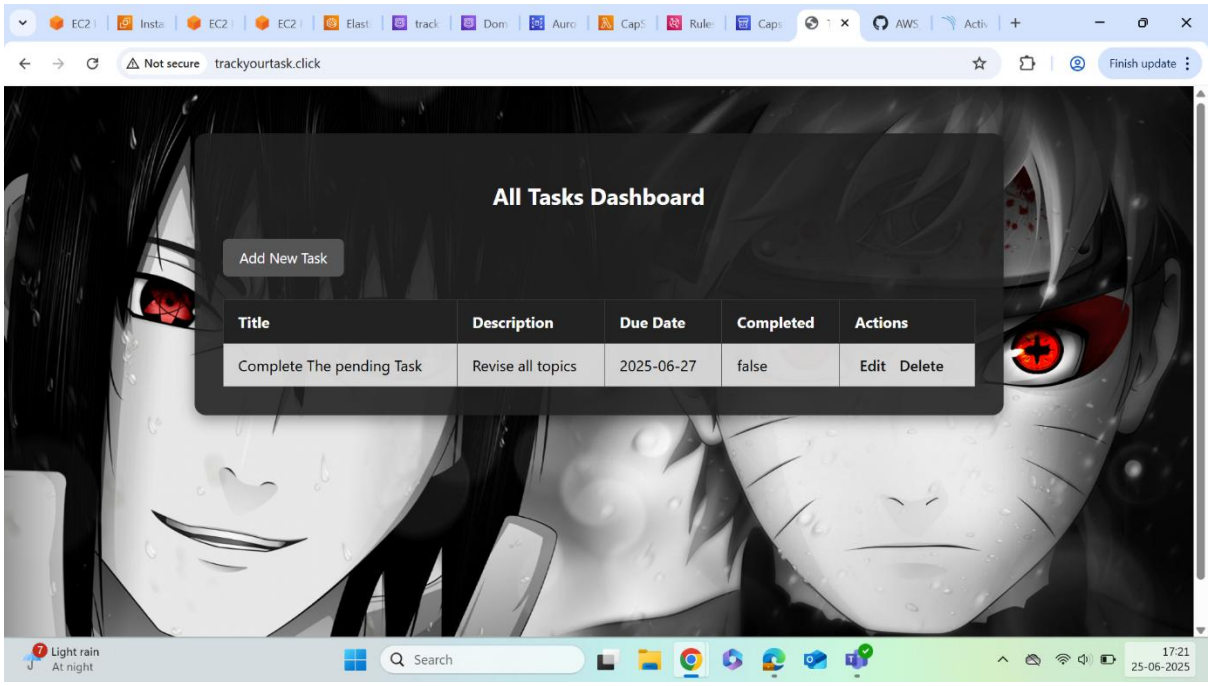
- **Record A (us-east-1)**
  - Alias to ALB DNS in Primary Region
  - Failover: Primary
  - Attach Health Check for ALB
- **Record A (us-west-1)**
  - Alias to ALB DNS in Secondary Region
  - Failover: Secondary

5.3 Health Check

- Target: ALB public endpoint
- Protocol: HTTP, Port 80 or 443
- If health check fails → Route 53 redirects to Secondary ALB

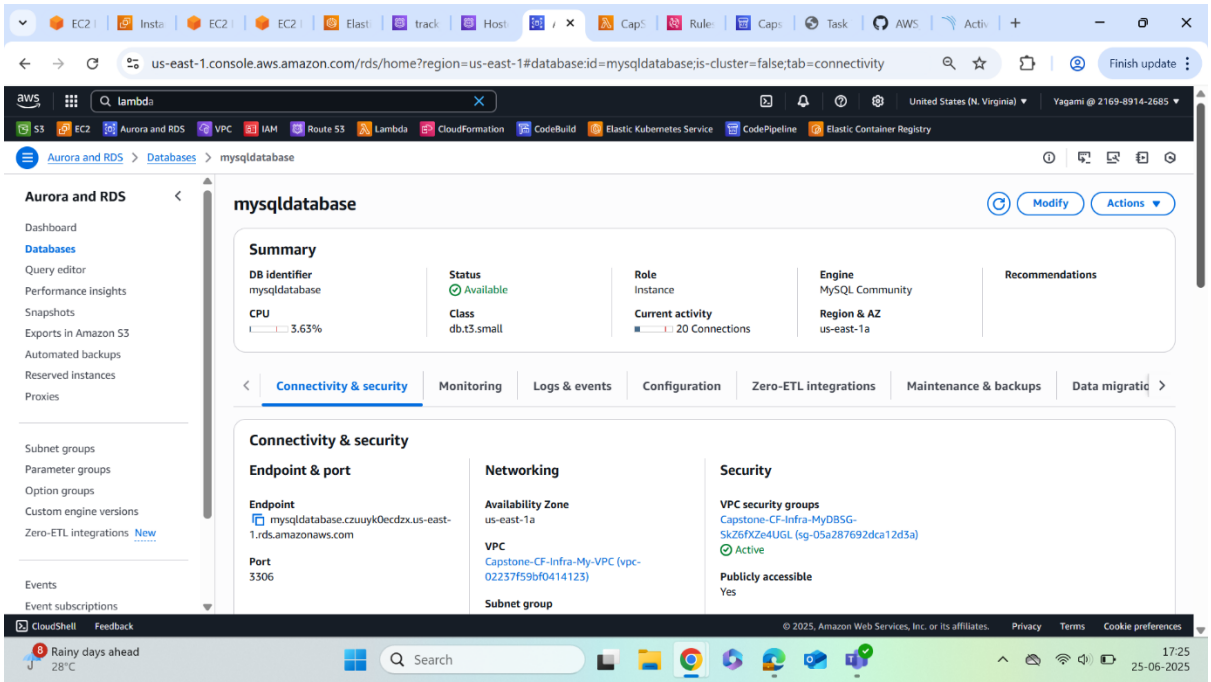


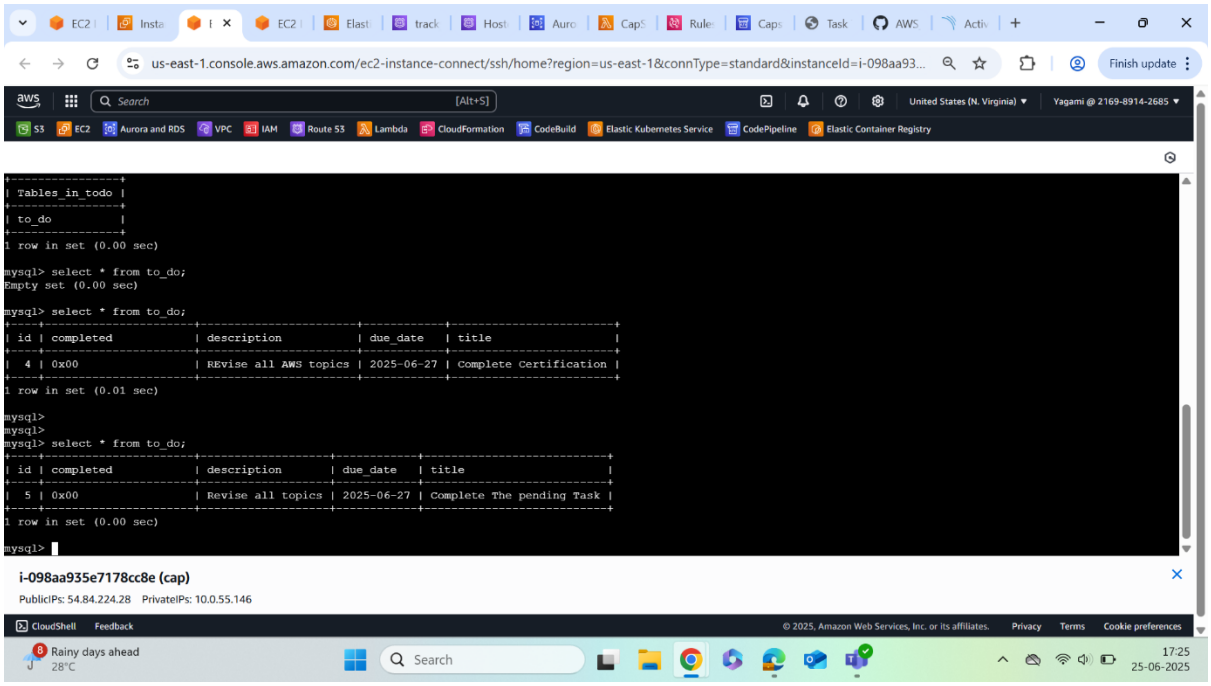




To simulate disaster recovery, you can intentionally fail the primary region's health check or temporarily disable its ALB. Route 53 will automatically shift traffic to the secondary region ALB."

After accessing the application, You can see that the data stored in the RDS.





Part 6: Monitoring and Alerting (CloudWatch + SNS)

6.0 EKS Container Insights Dashboard

AWS CloudWatch Container Insights can be enabled for EKS cluster (us-west-1) to monitor performance and operational health of pods, containers, and nodes.

Container Insights Metrics Available:

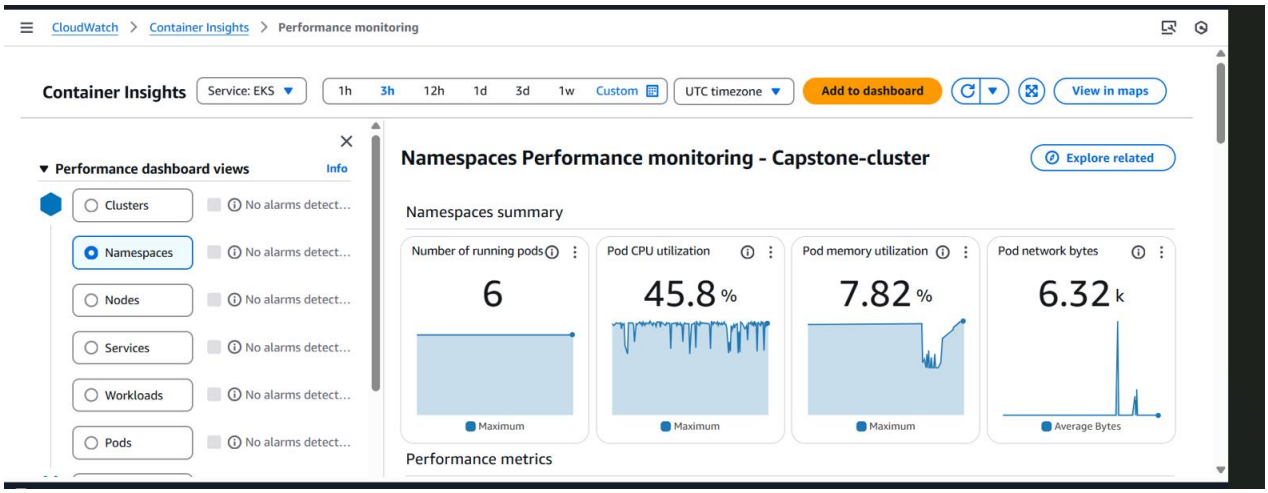
- pod_cpu_utilization
- container_memory_utilization
- node_cpu_utilization
- network_rx_bytes
- network_tx_bytes

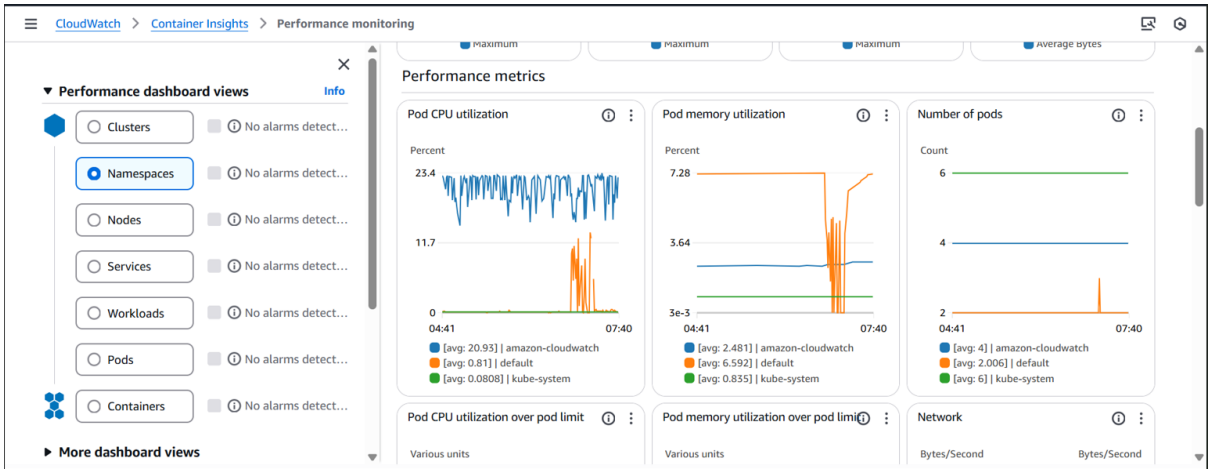
Default CloudWatch Dashboards:

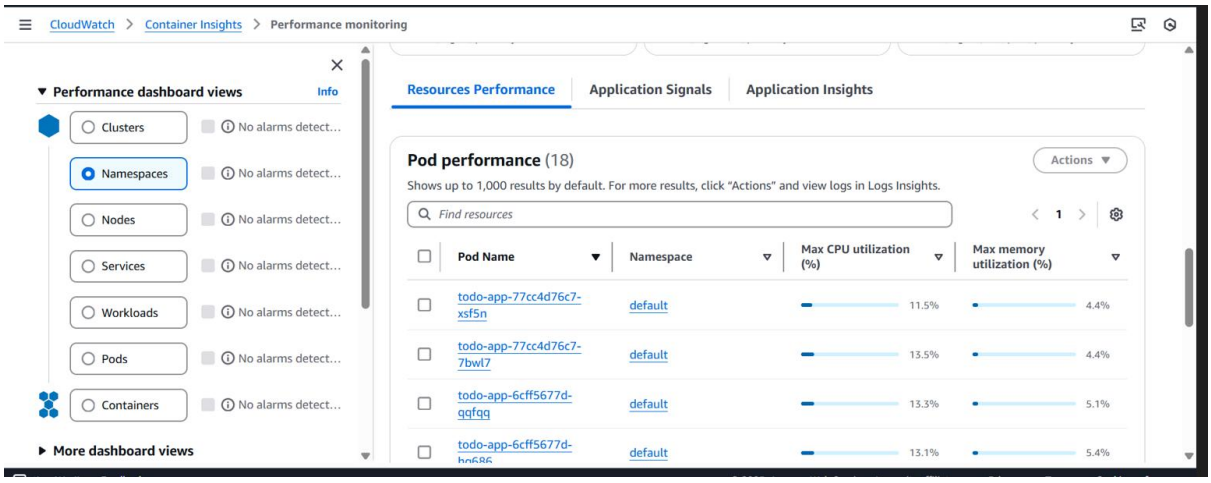
After enabling, navigate to: **CloudWatch → Dashboards → Container Insights → Performance Monitoring**

You will find:

- Node/Pod maps with CPU/Memory visualizations
- Filterable graphs for namespace, cluster, deployment, etc.
- Top N resource consumers


CloudWatch > Container Insights > Performance monitoring
Container Insights
Service: EKS
1h 3h 12h 1d 3d 1w Custom
UTC timezone
Add to dashboard
View in maps
Performance dashboard views
Info
Clusters
Namespaces
Nodes
Services
Workloads
Pods
No alarms detect...
Namespaces Performance monitoring - Capstone-cluster
Explore related
Namespaces summary
Number of running pods
6
Maximum
Pod CPU utilization
45.8 %
Maximum
Pod memory utilization
7.82 %
Maximum
Pod network bytes
6.32 k
Average Bytes
Performance metrics


CloudWatch > Container Insights > Performance monitoring
Performance dashboard views
Info
Clusters
Namespaces
Nodes
Services
Workloads
Pods
Containers
No alarms detect...
More dashboard views
Performance metrics
Pod CPU utilization
Percent
23.4
11.7
0
04:41
07:40
[avg: 20.93] | amazon-cloudwatch
[avg: 0.81] | default
[avg: 0.0808] | kube-system
Pod memory utilization
Percent
7.28
3.64
3e-3
04:41
07:40
[avg: 2.481] | amazon-cloudwatch
[avg: 6.592] | default
[avg: 0.835] | kube-system
Number of pods
Count
6
4
2
04:41
07:40
[avg: 4] | amazon-cloudwatch
[avg: 2.006] | default
[avg: 6] | kube-system
Pod CPU utilization over pod limit
Various units
Pod memory utilization over pod limit
Various units
Network
Bytes/Second
Bytes/Second


CloudWatch > Container Insights > Performance monitoring
Performance dashboard views
Info
Clusters
Namespaces
Nodes
Services
Workloads
Pods
Containers
No alarms detect...
More dashboard views
Resources Performance
Application Signals
Application Insights
Pod performance (18)
Actions
Shows up to 1,000 results by default. For more results, click "Actions" and view logs in Logs Insights.
Find resources
1
Pod Name
Namespace
Max CPU utilization (%)
Max memory utilization (%)
todo-app-77cc4d76c7-xsf5n
default
11.5%
4.4%
todo-app-77cc4d76c7-7bwl7
default
13.5%
4.4%
todo-app-6cff5677d-qqfqq
default
13.3%
5.1%
todo-app-6cff5677d-
default
13.1%
5.4%

6.2 CloudWatch Alarms

Resource	Metric	Threshold	Action
RDS	CPUUtilization	> 70%	Notify via SNS

6.2 SNS Setup

1. Go to SNS → Create Topic (e.g., infra-alerts)
2. Add email subscription
3. Attach SNS topic to CloudWatch alarms

Part 7: Security Best Practices

7.1 IAM and Role Isolation

- Principle of Least Privilege (PoLP)
- Separate roles for CodeBuild, EKS, CloudFormation

7.2 Secrets Management

- Store DB passwords, JWT secrets in **AWS Secrets Manager**
- Alternatively, use **SSM Parameter Store (SecureString)**

7.3 Network Security

- RDS in private subnets
- Use Security Groups + NACLs to restrict access
- No open ports to public except ALB (443, 80)

Final Notes

- The application runs in **two AWS regions** (us-east-1 primary, us-west-2 secondary) for **high availability** and **disaster recovery**.
- **CloudFormation** and **Terraform** are used for fully automated infrastructure setup.
- **CI/CD** is implemented using **CodePipeline**, with **SonarQube** for code analysis and **Trivy** for image scanning.
- **Route 53** provides DNS-based **failover** between regions.
- **CloudWatch** and **SNS** handle monitoring and alerts.
- **Security** is enforced using IAM roles, Secrets Manager, and private networking.
- The setup is **automated, secure, and scalable** for production use.