TASK 11

Deploying a Three-tier Application on AWS Elastic Beanstalk with CodePipeline using Terraform

Umar Satti

Table of Contents

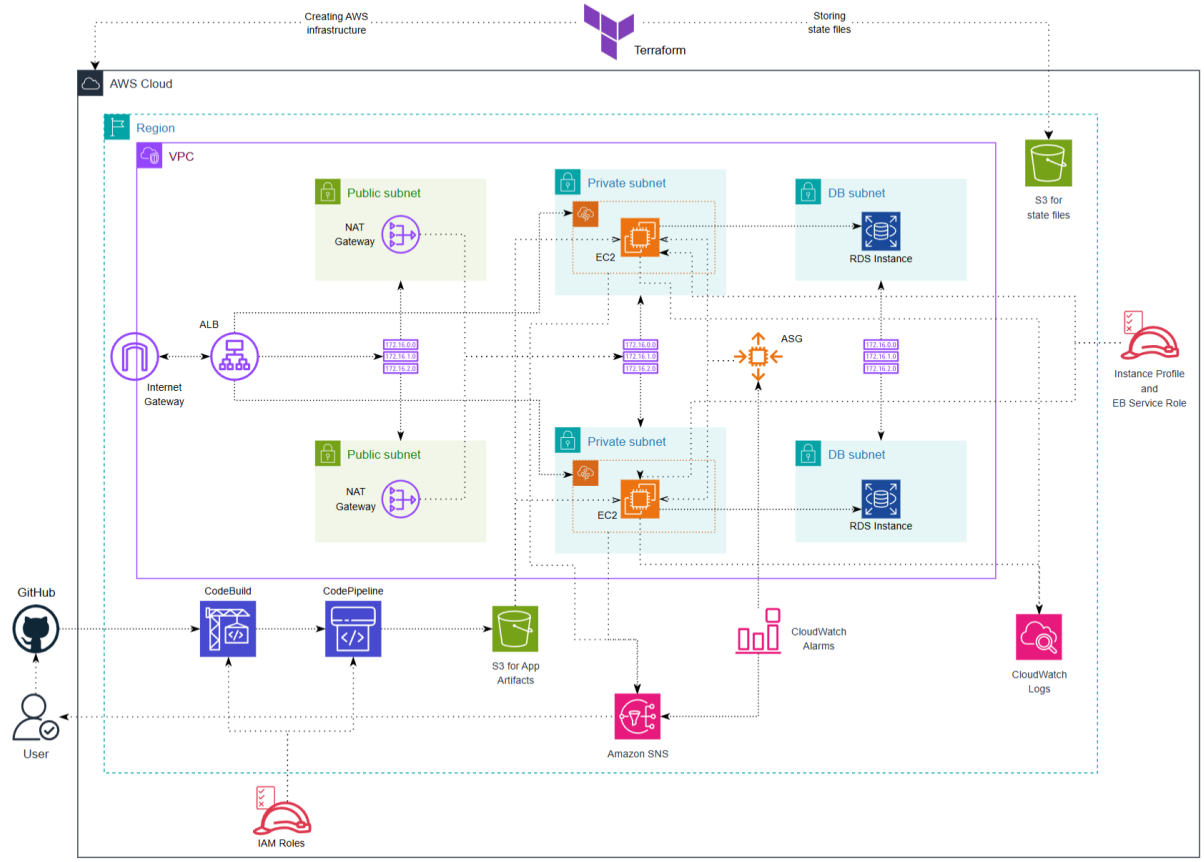
Task Description

This task involves deploying a simple three-tier application using React, Node.js, Express, and MySQL database through Terraform. The application is deployed in a single Elastic Beanstalk environment, with the frontend packaged inside the backend build folder.

Terraform provisions a secure AWS infrastructure including a VPC with public, private, and database subnets, routing, gateways, and security groups. An external RDS MySQL database is integrated, and IAM roles provide necessary permissions.

CI/CD is implemented using AWS CodePipeline and CodeBuild, with S3 storing artifacts and Terraform state files. Monitoring, logging, and notifications are enabled through CloudWatch and SNS. Deployment is validated by testing the application endpoint and confirming automated redeployments on code changes.

Architecture Diagram



1. Test Application Locally

Before deploying to AWS, the three-tier e-commerce application is tested locally to ensure all backend and database components function correctly.

1.1: Files Overview

1. app.js

- Main backend application file using Express.
- Sets up middleware for CORS and JSON parsing.
- Defines API routes for users, products, orders, cart, and token management.
- Serves the React frontend from the **build/** folder.
- Starts the server on Port 8080 (or environment-defined).

2. package.json

- Defines project metadata, dependencies, and scripts.
- Includes Express, MySQL2, bcryptjs, jsonwebtoken, dotenv, and other dependencies.
- Dev dependencies include nodemon and eslint for development and debugging.
- Scripts include start, debug, and build.

3. database/connection.js

- Configures MySQL connection using mysql2 with pooling and promise support.
- Supports both local and production (Elastic Beanstalk) environments.
- Tests database connectivity and logs status to the console.

4. Routes, Models, and Controllers

- Organized into directories by functionality: users, products, orders, and cart.
- Supports user registration, login, products, cart, and order.
- Includes role-based users (admin and normal users).

5. Database Schema

- Five tables: users, product, shoppingCart, orders, productsInOrder.
- Foreign keys enforce relationships between users, products, carts, and orders.
- Supports basic CRUD operations for all application entities.

1.2: Test Steps

1. Install backend dependencies

Open the first terminal, navigate to the backend (server/) folder, and run ***npm install***. This installs node_modules and generates a package-lock.json file.

2. Install frontend dependencies
Open the second terminal, navigate to the frontend folder, and run ***npm install***. This installs React and other frontend dependencies.

3. Start backend server
In the backend terminal, run **node app.js**. This starts the Express server on Port 8080 (or as defined in environment variables).

4. Start frontend application
In the frontend terminal, run ***npm start***. This starts the React application on **http://localhost:3000**.

5. Verify database connection

- Use MySQL Workbench to connect to the MySQL database.
- Create the database and tables using the provided schema.
- Ensure that the backend successfully connects to the database.

6. Test application functionality

- Open **http://localhost:3000** in a browser.
- Test all features including user registration/login, products, cart, and order.
- Confirm that both frontend and backend are working together correctly.

2. S3 Bucket for Terraform Remote Backend

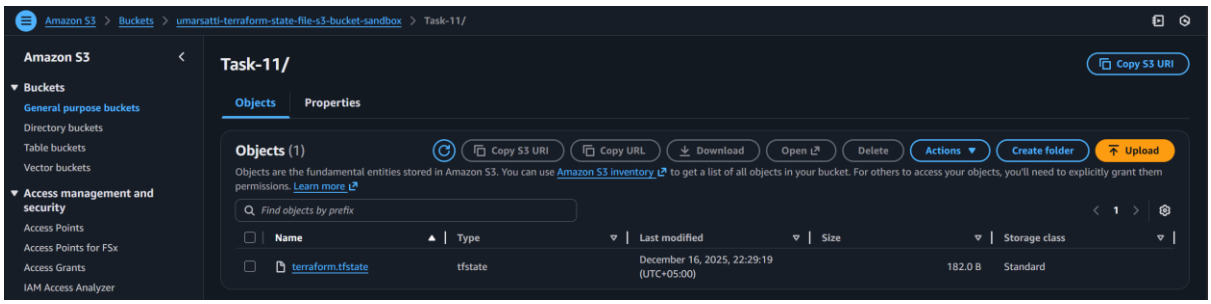
Steps:

1. Log in to the AWS Management Console and navigate to S3 service using search bar.
2. Click on **Create Bucket** button.
3. Select General Purpose, provide a globally unique bucket name, and ensure the region matches Terraform (us-west-1).
4. Click **Create Bucket**.
5. Update **terraform.tf** file to reference this bucket in the backend block.

```
  backend "s3" {
    bucket         = "umarsatti-terraform-state-file-s3-bucket-sandbox"
    key            = "Task-11/terraform.tfstate"
    region         = "us-west-1"
    encrypt        = true
    use_lockfile   = true
  }
}
```

- The S3 bucket ensures centralized and secure storage for Terraform state.
- Terraform automatically reads and updates the state file (terraform.tfstate) on every plan, apply, or destroy.
- **State locking** (use_lockfile = true) prevents concurrent modifications, reducing the risk of corruption.
- Example path in the bucket:

  S3 > Buckets > umarsatti-terraform-state-file-s3-bucket-sandbox > Task-11 > terraform.tfstate



This confirms that Terraform is ready to manage resources reliably across multiple users or machines.

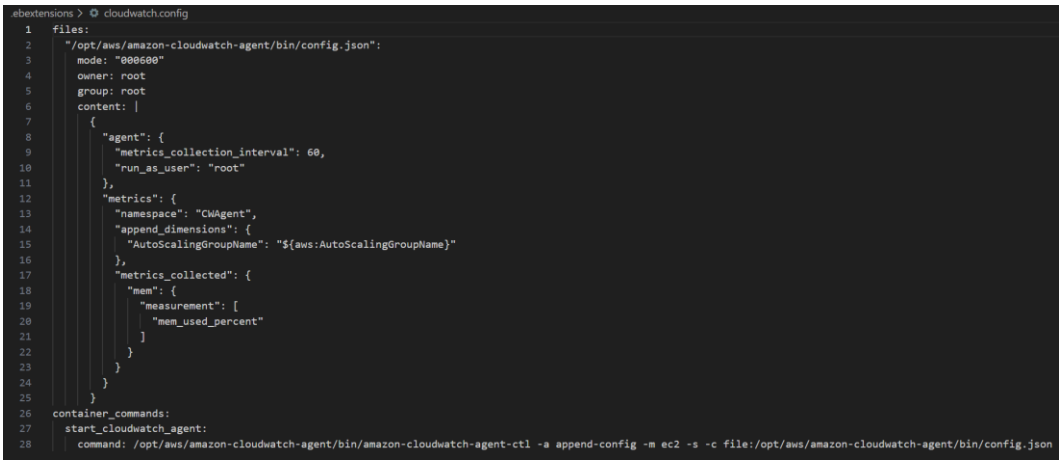
3. Project Structure

The project is organized into application, infrastructure, and configuration components to support local development, CI/CD, and AWS deployment.

3.1: .ebextensions

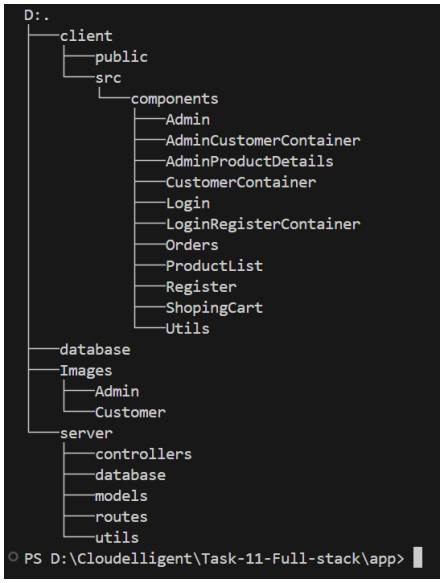
This folder contains Elastic Beanstalk platform configuration files applied during deployment.

- Contains a JSON file named **cloudwatch.config.**
- Configures the CloudWatch Agent on EC2 instances.
- Enables memory metrics and enhanced monitoring.
- Streams logs to CloudWatch.
- Used for alarms, scaling, and notifications.



3.2: Application Structure

The application is a simple three-tier e-commerce system.



Frontend (client/ folder)

- React application for UI
- Runs locally on port 3000
- Built using npm run build
- The generated build/ folder is copied into the backend for deployment

Backend (server/)

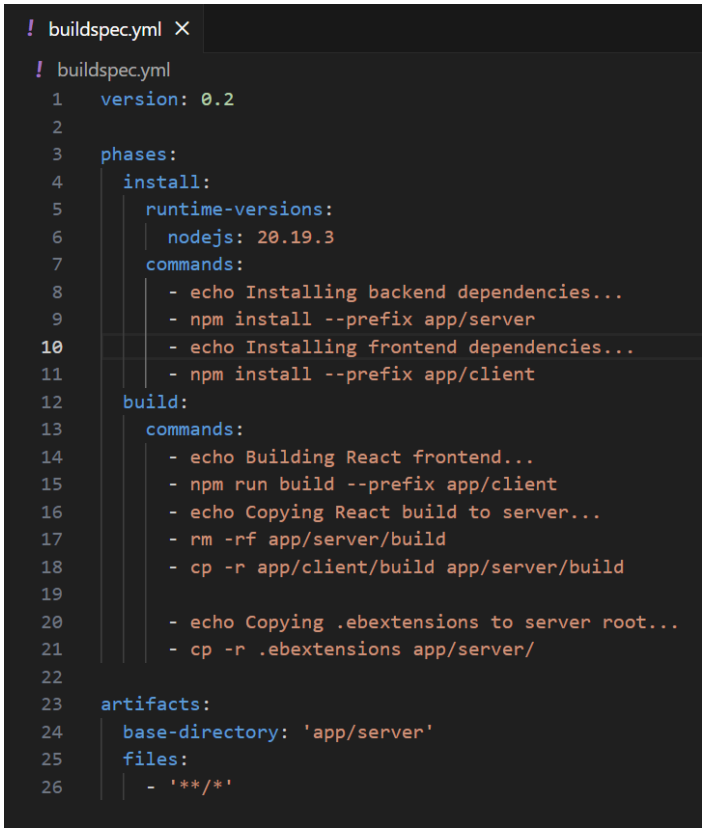
- Node.js and Express REST API
- Handles users, products, orders, cart, and authentication
- Serves the React build/ folder in production
- Connects to MySQL database using environment variables

Database (database/)

- Contains createTables.sql
- Defines schema for users, products, cart, and orders

3.3: buildspec.yml

The buildspec.yml file defines the build instructions used by AWS CodeBuild to prepare the application artifacts for deployment to Elastic Beanstalk. It installs dependencies, builds the frontend, packages the backend, and prepares the final deployable bundle.



Install Phase

- Specifies the Node.js runtime version required for the application.
- Installs backend dependencies from the server/ directory.
- Installs frontend dependencies from the client/ directory.
- Ensures both application tiers are ready before the build phase.

Build Phase

- Builds the React frontend using npm run build.
- Removes any existing frontend build directory in the backend.
- Copies the generated React build/ folder into the backend directory.
- Copies the .ebextensions folder into the backend root so Elastic Beanstalk can apply platform configurations (CW agent, logging, and monitoring) during deployment.

Artifacts

- Defines the backend (server/) directory as the root of the deployment package.
- Includes all files required for Elastic Beanstalk, such as:
  - package.json
  - app.js
  - build/ (React frontend)
  - .ebextensions/
- Ensures Elastic Beanstalk can correctly detect and run the Node.js application.

3.4: Terraform Directory Files

The Terraform configuration for this task is organized using a modular structure, where each major AWS service is defined in its own module, and the root directory coordinates the entire deployment. This approach improves readability, reusability, and maintainability of the infrastructure code. The following section describes each root-level Terraform file and its role in the deployment.

1. main.tf

This file acts as the central entry point for the infrastructure deployment. It integrates multiple Terraform modules and passes required variables and outputs between them:

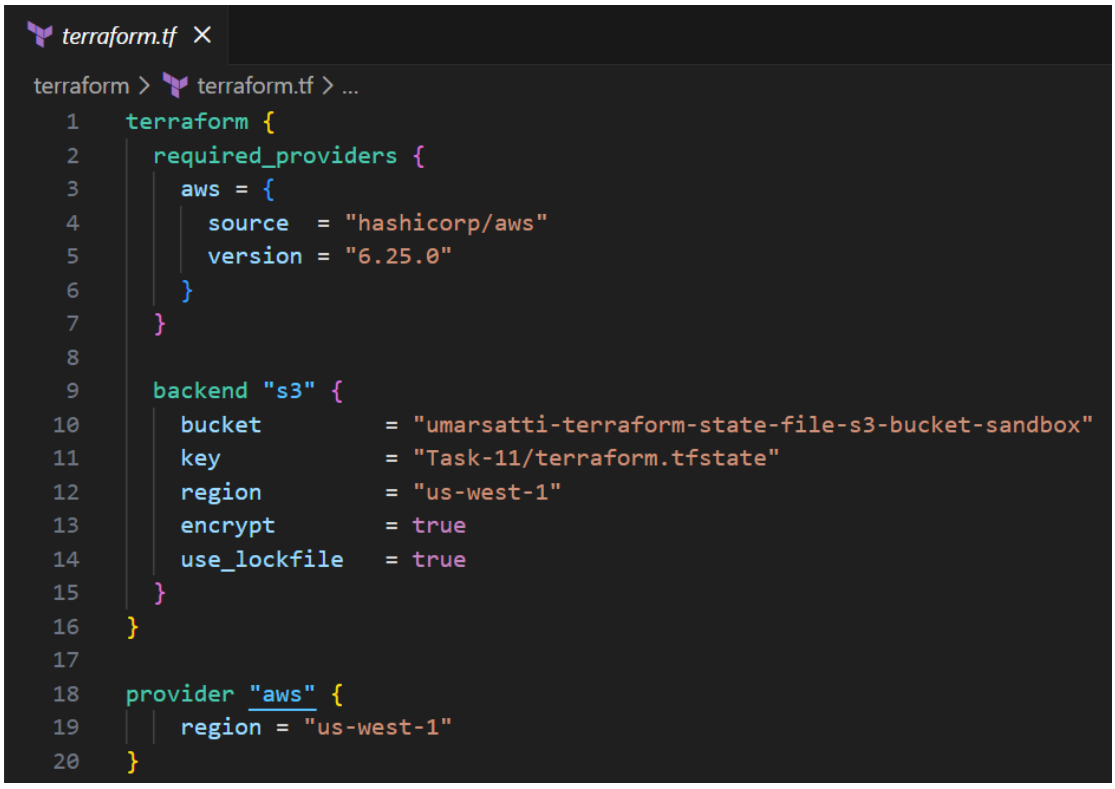
- **VPC module** – provisions the networking layer, including the VPC, public, private, and database subnets, routing, NAT gateway, internet gateway, and security groups. It outputs VPC IDs and subnet IDs used by other modules.
- **IAM module** – creates IAM service roles and instance profiles for Elastic Beanstalk, EC2, CodeBuild, and CodePipeline using external JSON policy files.
- **RDS module** – provisions an external MySQL RDS instance inside database subnets and outputs the database endpoint.

- **Elastic Beanstalk backend module** – deploys the Node.js application, configures the environment, auto scaling, load balancer, monitoring, logging, and SNS notifications. It consumes outputs from the VPC, IAM, and RDS modules.
- **CodeBuild module** – creates the build project used in the CI/CD pipeline, configured to run inside the VPC with appropriate networking permissions.
- **CodePipeline module** – defines the CI/CD pipeline stages (source, build, deploy) and connects GitHub, CodeBuild, and Elastic Beanstalk.

All dependencies are resolved automatically through module outputs, ensuring correct resource creation order.

2. terraform.tf

This file defines the AWS provider and configures the remote backend for Terraform state management.



- The Terraform state file is stored in an S3 bucket:
    - umarsatti-terraform-state-file-s3-bucket-sandbox/Task-11/terraform.tfstate
- State encryption is enabled, and locking is enforced to prevent concurrent modifications.

This setup ensures secure, centralized, and consistent state management across deployments.

3. variables.tf

This file declares all configurable input variables used throughout the infrastructure.

- **VPC variables** – CIDR block, VPC name, internet gateway name, routing configuration.
- **IAM variables** – names for service roles and permission policies for Elastic Beanstalk, EC2, CodeBuild, and CodePipeline.
- **Elastic Beanstalk variables** – application name, environment name, platform version, EBS volume configuration, monitoring interval, and alert email.
- **CodeBuild variables** – project name, logging configuration, AWS account ID, and region.
- **CodePipeline variables** – pipeline name, artifact bucket, GitHub repository, and connection ARN.

This file defines what can be customized without modifying the core Terraform code.

4. terraform.tfvars

This file provides concrete values for all declared variables and represents the actual deployment configuration:

- Full VPC networking configuration
- IAM role and policy names
- Elastic Beanstalk application and environment details
- Node.js platform version and EC2 storage settings
- CloudWatch monitoring interval and SNS notification email
- CodeBuild and CodePipeline configuration details
- GitHub repository and CodeStar connection ARN

Separating values from logic keeps the deployment environment-specific and clean.

5. outputs.tf

This file exposes important outputs generated during the deployment:

- **vpc_id** – used for validation and future integrations.
- **Application_URL** – the public Elastic Beanstalk endpoint used to access the deployed application.

These outputs allow quick verification after running terraform apply.

6. IAM Permission JSON Files

Multiple JSON files define granular permissions for AWS services:

- **eb-permissions.json** – Elastic Beanstalk service role permissions.
- **instance-permissions.json** – EC2 instance profile permissions.
- **codebuild-permissions.json** – permissions for VPC networking, logging, S3 access, CodeBuild reports, and GitHub connections.
- **codepipeline-permissions.json** – permissions for S3 artifacts, CodeBuild execution, Elastic Beanstalk deployments, VPC networking, and CodeStar connections.

These files are referenced by the IAM module, keeping permission logic externalized and easier to manage.

3.5: VPC Module

The VPC module provisions all networking resources required for the three-tier application, including subnet segmentation, routing, NAT gateways, and security groups for Elastic Beanstalk, EC2, ALB, and RDS.

1. main.tf

Defines the networking stack:

- VPC
    - Creates a VPC with DNS support and hostnames enabled.
- Subnets
    - Two public subnets for the Application Load Balancer.
    - Two private subnets for Elastic Beanstalk EC2 instances.
    - Two database subnets dedicated to RDS.
    - Subnets are distributed across multiple Availability Zones.
- Internet Gateway & NAT Gateways
    - Internet Gateway enables public subnet internet access.
    - One NAT Gateway per public subnet allows outbound access for private subnets.
- Route Tables
    - Public subnets route traffic to the Internet Gateway.
    - Private subnets route traffic through NAT Gateways.

    - Database subnets use isolated route tables.
- Security Groups
    - ALB-SG: Allows inbound HTTP (port 80) from the internet.
    - EC2-SG: Allows Node.js traffic on port 3000 from ALB only.
    - RDS-SG: Allows MySQL traffic on port 3306 from EC2 only.

2. variables.tf

Defines configurable networking inputs:

- VPC CIDR and name
- Internet Gateway name
- Elastic IP domain
- Default public route (0.0.0.0/0)

3. outputs.tf

Exports key networking identifiers:

- VPC ID
- Public, private, and database subnet IDs
- ALB, EC2, and RDS security group IDs

These outputs are consumed by the Beanstalk, RDS, CodeBuild, and root modules.

3.6: IAM Module

This module provisions all IAM roles, policies, and instance profiles required by Elastic Beanstalk, EC2, CodeBuild, and CodePipeline.

1. main.tf

Creates the following roles:

- Elastic Beanstalk Service Role
    - Used by Elastic Beanstalk to manage environments.
    - Permissions loaded from eb-permissions.json.
- Elastic Beanstalk EC2 Role
    - Assigned to EC2 instances launched by Beanstalk.

    - Uses instance-permissions.json.
    - Includes an IAM instance profile.
- CodeBuild Service Role
    - Used during the CI/CD build stage.
    - Permissions loaded from codebuild-permissions.json.
- CodePipeline Service Role
    - Orchestrates source, build, and deploy stages.
    - Permissions loaded from codepipeline-permissions.json.
    - Includes AWS-managed Elastic Beanstalk administrative access.

2. variables.tf

Defines IAM role and policy names:

- Beanstalk service and EC2 roles
- CodeBuild and CodePipeline roles
- Custom policy names

3. outputs.tf

Exposes IAM references:

- Beanstalk service role name
- EC2 instance profile name
- CodeBuild role ARN
- CodePipeline role ARN

These outputs are referenced by CI/CD and Elastic Beanstalk modules.

3.7: Elastic Beanstalk Module

This module provisions the Elastic Beanstalk application and environment, configures networking, scaling, monitoring, environment variables, and integrates external services such as RDS and CloudWatch.

1. main.tf

- S3 Application Bucket

    - Creates an S3 bucket for Elastic Beanstalk artifacts.
- Elastic Beanstalk Application & Environment
    - Defines the application and environment using a Node.js platform.
    - Uses a load-balanced environment with an Application Load Balancer.
    - EC2 instances run in private subnets. ALB runs in public subnets.
    - Associates required IAM roles and instance profile.
- Instance & Scaling Configuration
    - Configures root volume type and size.
    - Enables enhanced monitoring and disables IMDSv1.
    - Auto Scaling Group configured with min/max capacity (1 to 3).
    - CPU-based scaling triggers configured.
- Logging, Monitoring & Notifications
    - Enables CloudWatch log streaming and enhanced health checks.
    - Configures email notifications using SNS.
- Environment Variables
    - Injects database connection details and JWT secrets.
    - Connects the application to the external RDS database.
- Custom CloudWatch Monitoring
    - Creates SNS alerts for memory utilization.
    - Adds Auto Scaling policy based on CloudWatch Agent memory metrics.

2. variables.tf

Defines inputs for:

- Application and environment names
- Platform version
- Storage and monitoring settings
- IAM role references
- VPC subnet IDs
- Database endpoint

3. outputs.tf

Exposes key environment details:

- Elastic Beanstalk environment URL
- Application and environment names
- SNS topic ARN
- Memory alarm ARN
- Backend API endpoint URL

3.8: RDS Module

This module provisions a MySQL RDS instance used by the backend application running in Elastic Beanstalk.

1. main.tf

- DB Subnet Group
    - Uses isolated database subnets from the VPC module.
- RDS MySQL Instance
    - MySQL 8.0 engine
    - Private, non-publicly accessible instance
    - Secured using the RDS security group
    - Used as the persistent data store for the application

2. variables.tf

Accepts references to:

- Database subnet IDs
- RDS security group ID

3. outputs.tf

- Exposes the RDS endpoint address used by the Elastic Beanstalk environment.

3.9: CodeBuild Module

The CodeBuild module configures the build stage of the CI/CD pipeline, responsible for installing dependencies and preparing application artifacts for deployment to Elastic Beanstalk.

1. main.tf

- **CodeBuild Project**
    - Creates a CodeBuild project using a managed Amazon Linux image.
    - Uses a small compute type suitable for Node.js builds.
    - Integrates directly with CodePipeline for source input and build artifacts.
    - Runs inside private subnets within the VPC for secure network access.
    - Associates a dedicated security group and IAM service role.
- **Environment Configuration**
    - Enables privileged mode to support advanced build steps.
    - Injects AWS account ID and region as environment variables.
    - Sends build logs to a specified CloudWatch log group.

2. variables.tf

Accepts configuration values for:

- VPC and private subnet IDs
- Security group ID
- CodeBuild project name
- IAM service role
- AWS account ID and region
- CloudWatch log group name

3. outputs.tf

Exports:

- CodeBuild project ARN
- CodeBuild project name

These outputs are referenced by the CodePipeline module.

3.10: CodePipeline Module

The CodePipeline module defines the end-to-end CI/CD workflow for automatically building and deploying the application to Elastic Beanstalk.

1. main.tf

- **Artifact Store**
    - Creates a private S3 bucket to store pipeline artifacts.
    - Blocks all public access for security.
- **Source Stage**
    - Uses an existing GitHub CodeStar connection.
    - Monitors the main branch for code changes.
    - Automatically triggers the pipeline on commits.
- **Build Stage**
    - Invokes the CodeBuild project to install dependencies and build the application.
    - Consumes source artifacts and produces build artifacts.
- **Deploy Stage**
    - Deploys both backend and frontend artifacts to the Elastic Beanstalk environment.
    - Uses the Elastic Beanstalk application and environment created by Terraform.

2. variables.tf

Defines inputs for:

- Pipeline name and artifact bucket
- GitHub repository and CodeStar connection ARN
- CodeBuild project references
- Elastic Beanstalk application and environment names
- CodePipeline IAM service role

4. Execute Terraform Commands

This task deploys the entire **Elastic Beanstalk full-stack environment** using Terraform. The user has to be inside the **Terraform root directory** before running the commands.

1. ***terraform init***

Initializes Terraform, downloads providers, and configures the **S3 remote backend** for the state file.

2. ***terraform validate***

Validates the Terraform configuration for syntax, structure, and module correctness.

3. ***terraform plan***

Generates an execution plan and shows all resources that will be created, including the VPC, IAM roles, S3 bucket, and Beanstalk environment.

4. ***terraform apply --auto-approve***

Deploys the full infrastructure (38 resources in total), including:

```
Apply complete! Resources: 56 added, 0 changed, 0 destroyed.

Outputs:

Application_URL = "awseb--AWSEB-uDBFz8W46vqe-2013849991.us-west-1.elb.amazonaws.com"
vpc_id = "vpc-09ef99b05826a2f15"
PS D:\Cloudelligent\Task-11-Full-stack\terraform>
```

- VPC, Public/Private/Db subnets, Route tables, IGW, NAT gateways.
- Security groups (ALB-SG, EC2-SG, RDS-SG).
- IAM roles, policies, and EC2 instance profile.
- S3 bucket for artifact upload.
- Elastic Beanstalk application, version, environment, platform, and env.
- ALB creation and attachment to public subnets.
- EC2 Auto scaling configuration.
- SNS topic and email subscription.
- CloudWatch alarms and scaling policy.
- DB subnets and RDS MySQL database instance.
- CodeBuild and CodePipeline for CI/CD.

5. Validate Infrastructure in AWS Console

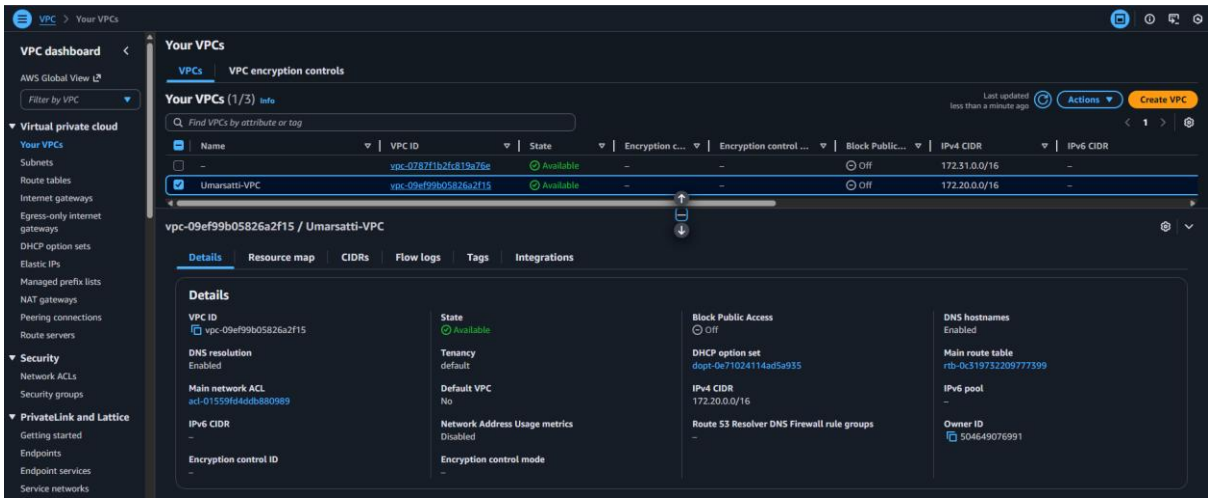
After running **terraform apply**, the next step is to verify that all AWS resources for the **Elastic Beanstalk full-stack environment** were successfully provisioned. This task walks through the AWS Console to confirm networking, IAM, compute, storage, database, and monitoring components.

Note: *Screenshots have been attached as evidence.*

5.1: VPC and Networking Validation

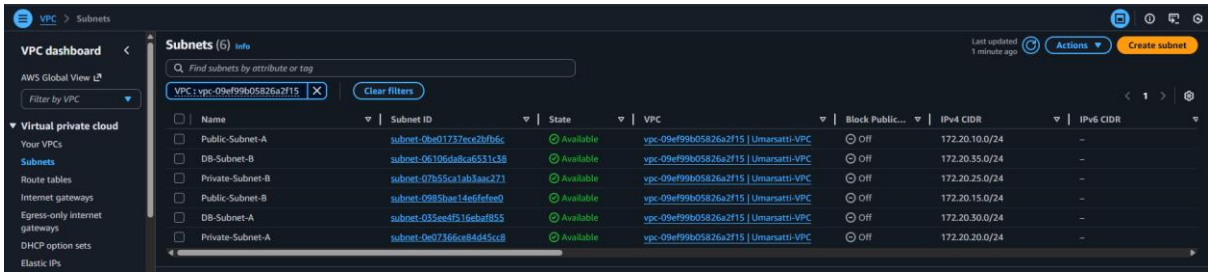
1. Verify VPC

- Open the **AWS Management Console.**
- Navigate to **VPC** using the search bar at the top.
- Click **Your VPCs** and check that the VPC created by Terraform exists:
    - Correct **CIDR block** (172.20.0.0/16) and **Name** (Umarsatti-VPC).
    - **DNS hostnames** and **DNS resolution** are enabled.



2. Verify Subnets

- Navigate to **Subnets** section located in the left navigation bar.
- Confirm that the subnets are created.



Public Subnets (*Used by Application Load balancer)*

- Public-Subnet-A (us-west-1a): 172.20.10.0/24
- Public-Subnet-B (us-west-1b): 172.20.15.0/24

Private Subnets (*Used by EC2 instances)*

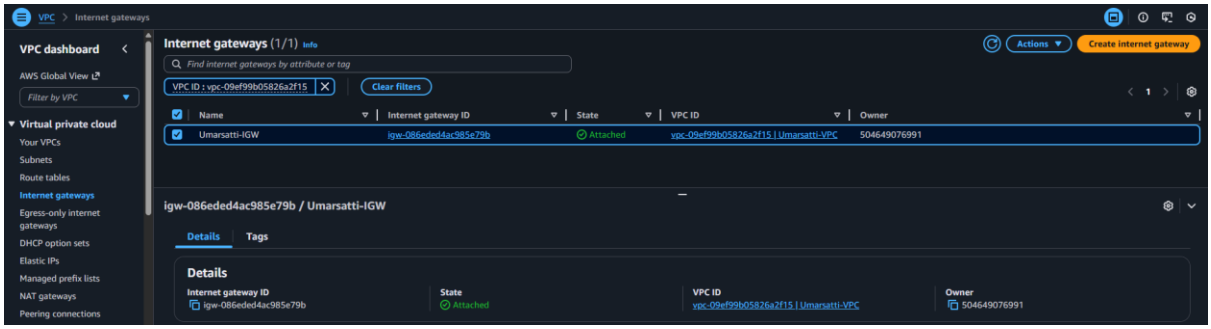
- Private-Subnet-A (us-west-1a): 172.20.20.0/24
- Private-Subnet-B (us-west-1b): 172.20.25.0/24

Database Subnets *(Used by RDS database instances)*

- DB-Subnet-A (us-west-1a): 172.20.35.0/24
- DB-Subnet-B (us-west-1b): 172.20.30.0/24

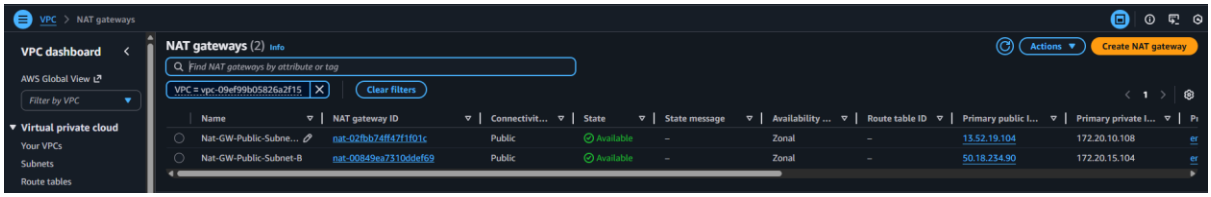
3. Internet Gateway and NAT Gateways

Internet Gateway



- Navigate to **Internet Gateway** section located in the left navigation bar.
- Confirm that the Internet Gateway has been created and attached to the VPC.
    - **Status:** Attached
    - **Name:** Umarsatti-IGW

NAT Gateways

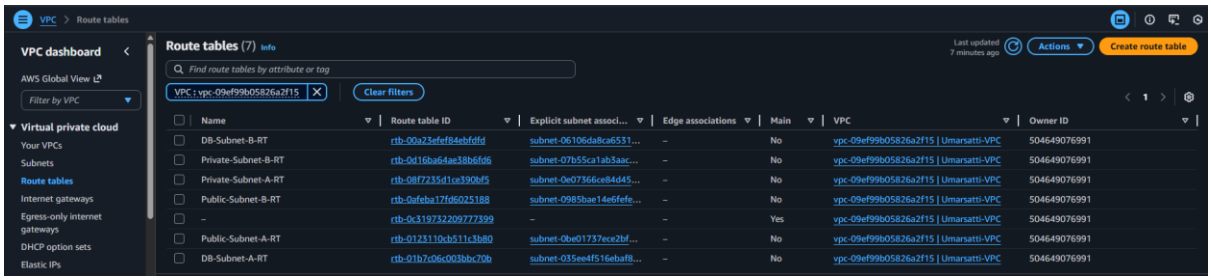


- Navigate to **NAT Gateway** section located in the left navigation bar.
- Confirm that two NAT Gateways have been created.
    - One NAT gateway in each public subnet.
    - Status: **Available**.
    - Elastic IP allocated automatically.

4. Route Tables

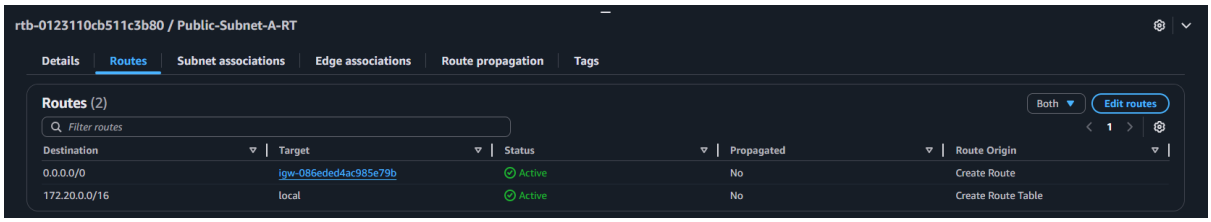
- Navigate to **Route tables** section located in the left navigation bar.

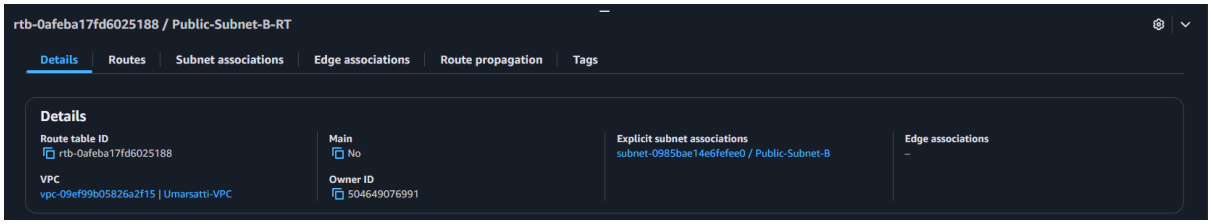
- Confirm that both public and private route tables have been created.



Public Route Tables

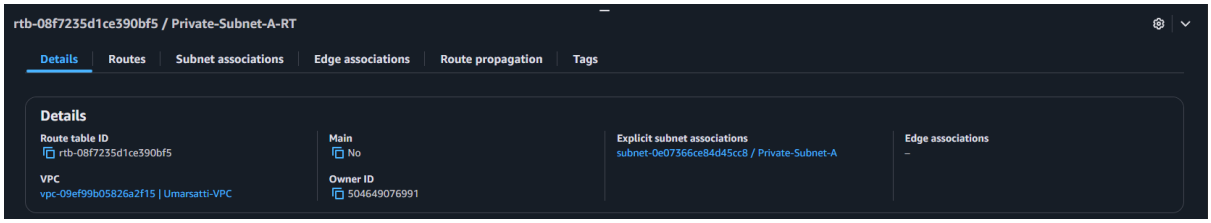
- Associated with public subnets.
- Should contain route 0.0.0.0/0 → Internet Gateway.

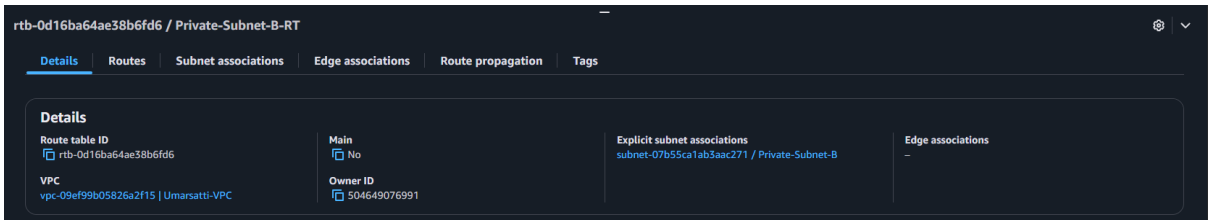




Private Route Tables

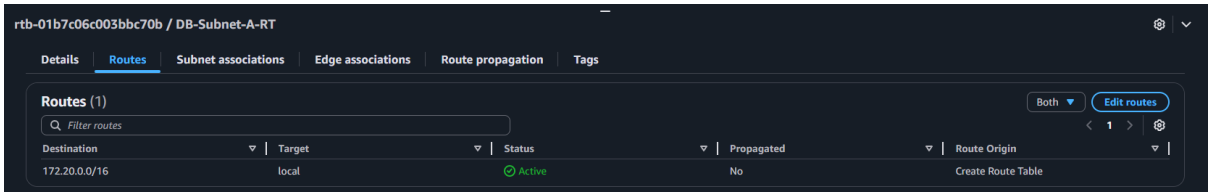
- Associated with private subnets
- Should contain route 0.0.0.0/0 → NAT Gateway

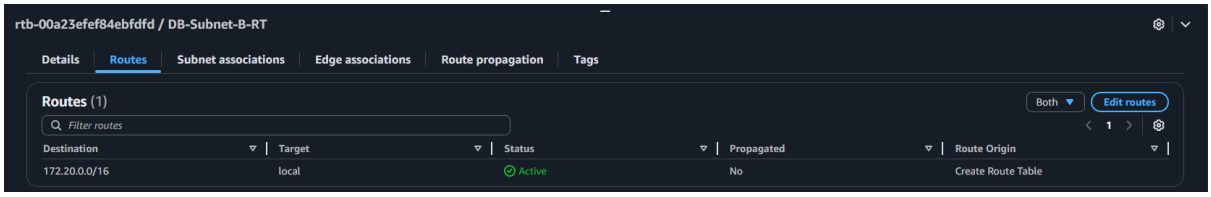




Database Route Tables

- Associated with DB subnets.
- It should contain the local route only (for security).



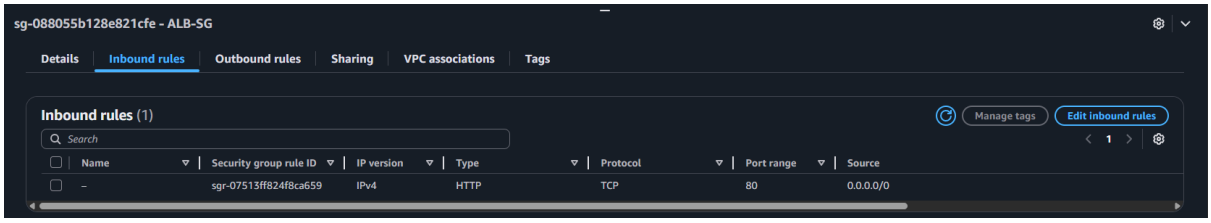


5. Security Groups

- In VPC console, navigate to **Security groups** located in the left navigation bar.
- Confirm that both the load balancer and EC2 security groups have been created.

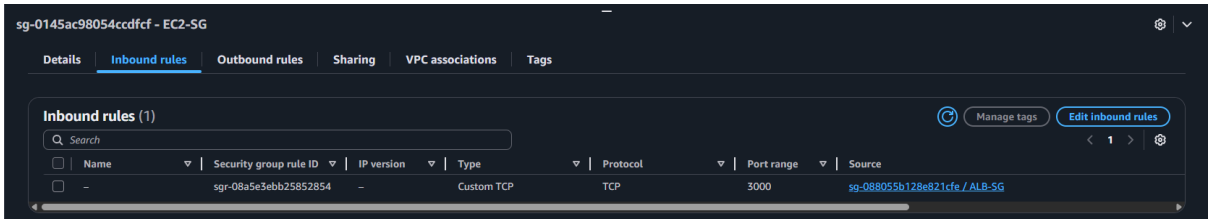
ALB-SG

- Allows inbound HTTP (80) from anywhere
- Egress open to 0.0.0.0/0



EC2-SG

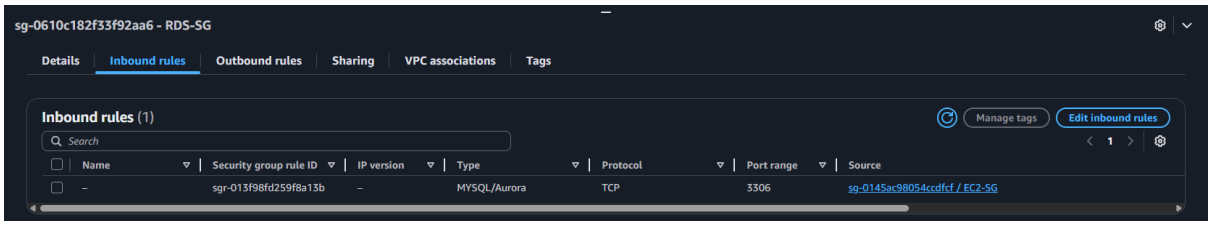
- Allows inbound TCP 3000 *only* from ALB-SG
- Egress open to 0.0.0.0/0



RDS-SG

- Allows inbound TCP 3000 *only* from ALB-SG

- Egress open to 0.0.0.0/0



Note: Elastic Beanstalk creates security groups by itself and as indicated by the name ***nodejs-environment***.

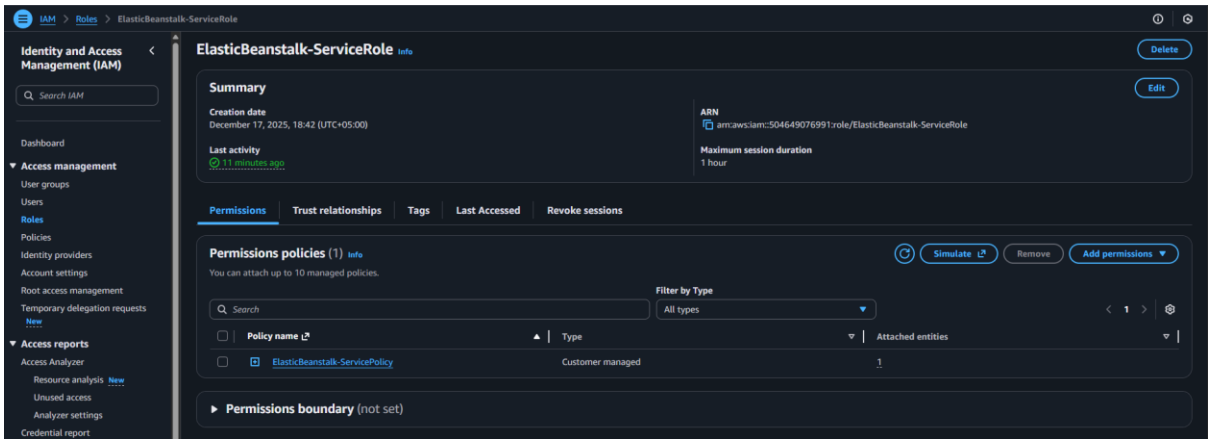
5.2: IAM Validation

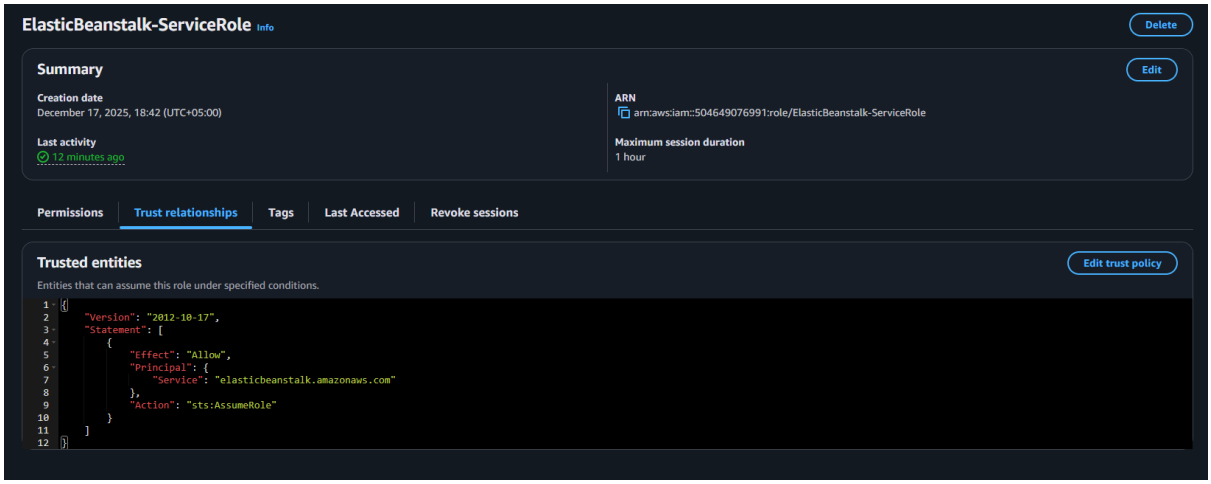
1. Elastic Beanstalk Service Role

- Navigate to the **IAM Console** using the search bar.
- Search for the Elastic Beanstalk service role.

Check:

- Role name: ElasticBeanstalk-ServiceRole
- Trust policy: elasticbeanstalk.amazonaws.com
- Attached policy matches JSON from eb-permissions.json
    - *ElasticBeanstalk-ServicePolicy*



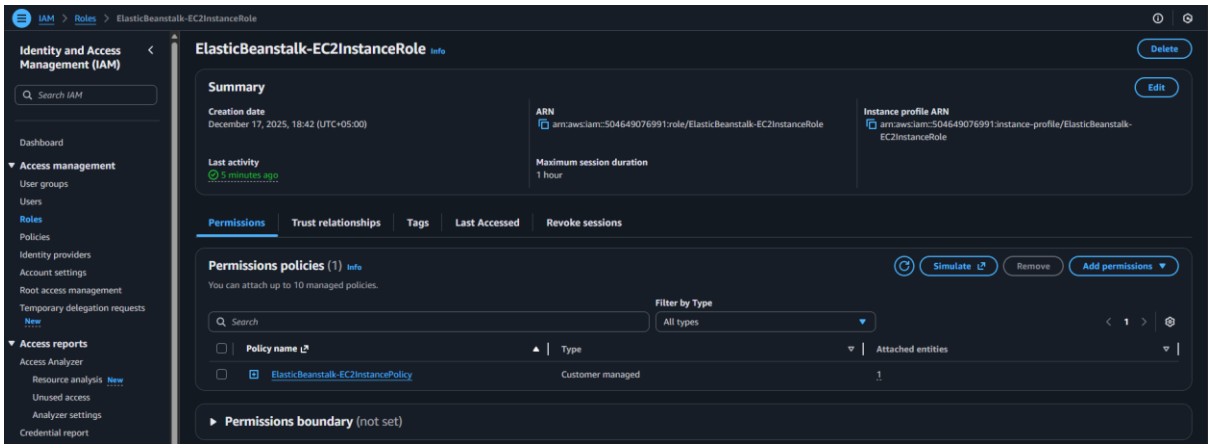


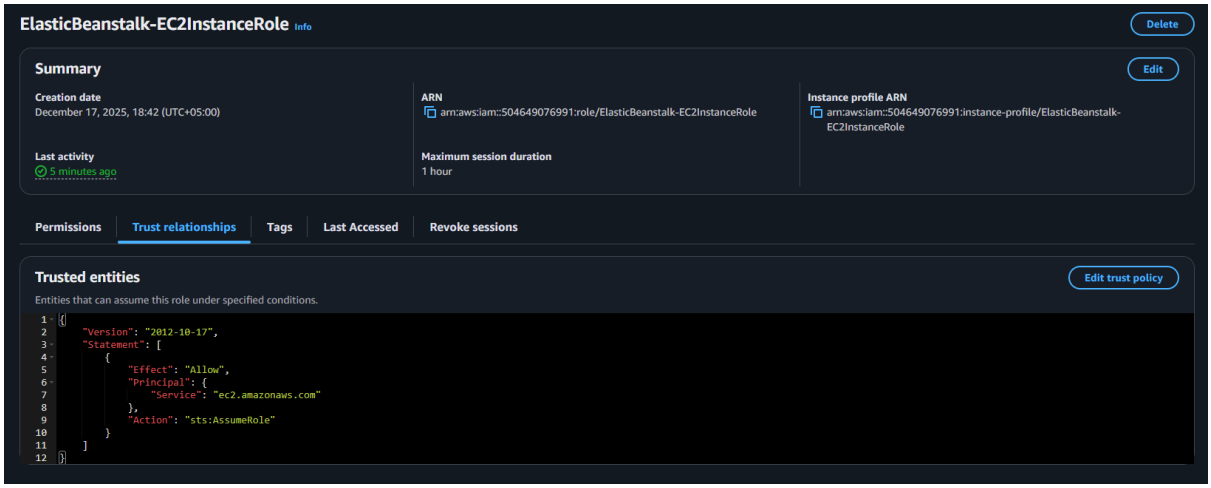
2. EC2 Instance Profile

- Navigate to the **IAM Console** using the search bar.
- Search for the EC2 instance role.

Check:

- IAM Role: ElasticBeanstalk-EC2InstanceRole
- Instance Profile created
- Attached policy from instance-permissions.json
    - *ElasticBeanstalk-EC2InstancePolicy*



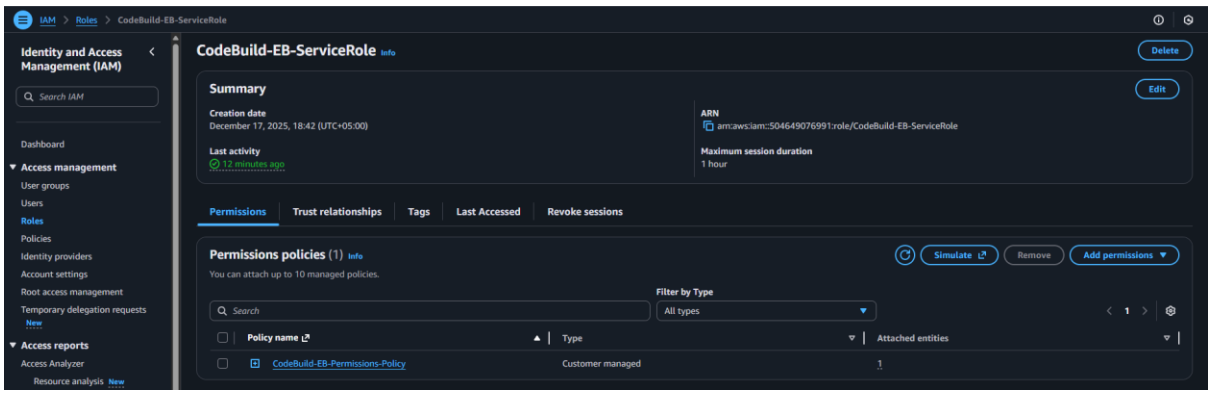


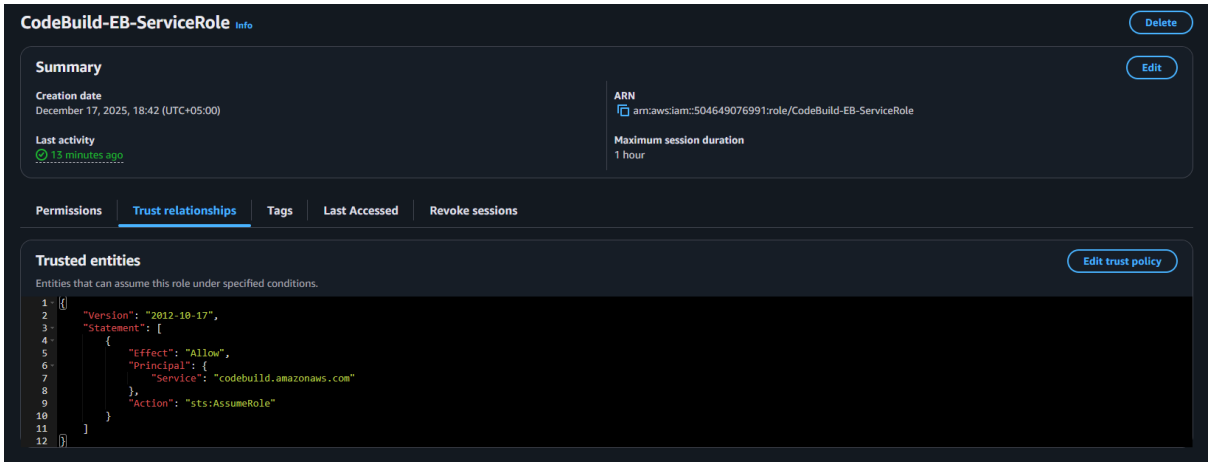
3. CodeBuild Role

- Navigate to the **IAM Console** using the search bar.
- Search for the CodeBuild IAM role.

Check:

- Role name: **CodeBuild-EB-ServiceRole**
- Trust policy: elasticbeanstalk.amazonaws.com
- Attached policy matches JSON from **codebuild-permissions.json**
    - *CodeBuild-EB-Permissions-Policy*



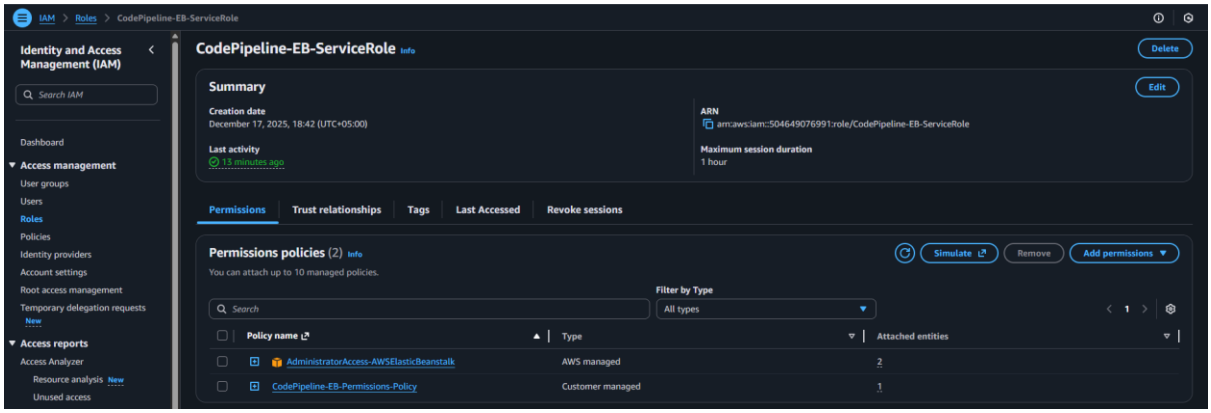


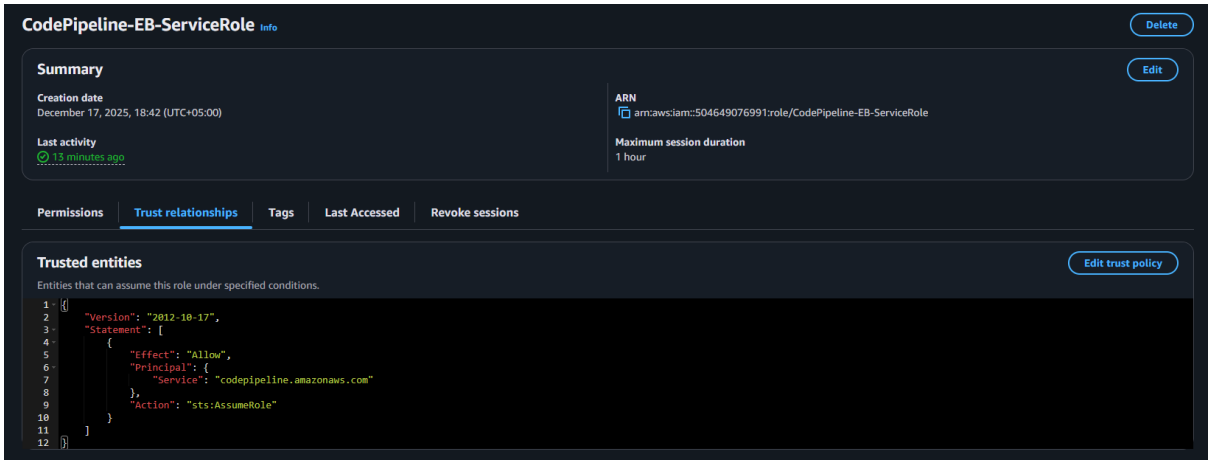
4. CodePipeline Role

- Navigate to the **IAM Console** using the search bar.
- Search for the CodePipeline IAM role.

Check:

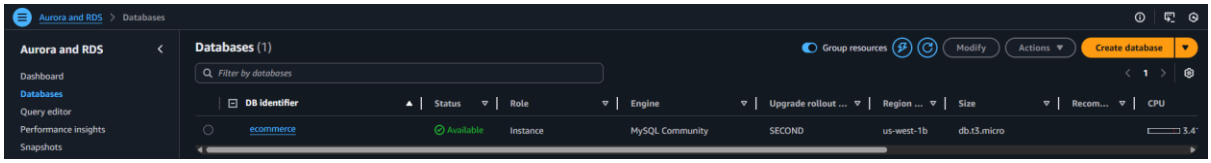
- Role name: **CodePipeline-EB-ServiceRole**
- Trust policy: elasticbeanstalk.amazonaws.com
- Attached policy matches JSON from **codepipeline-permissions.json**
    - *CodePipeline-EB-Permissions-Policy*
    - *AdministratorAccess-AWSElasticBeanstalk*

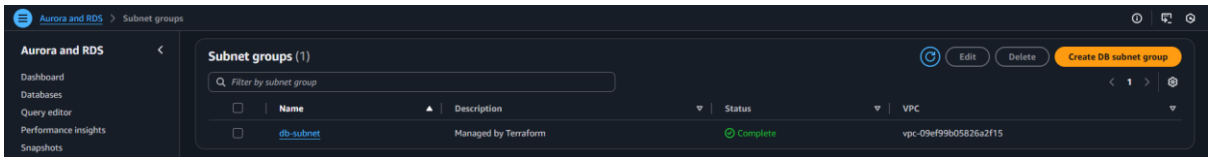




5.3. RDS Validation

- Navigate to the **Aurora and RDS** console using the search bar.
- Select Databases located on the left navigation bar. This should display MySQL database instance running.
- Select Subnet groups on the left navigation bar. This should display a DB subnet group attached to the instance.

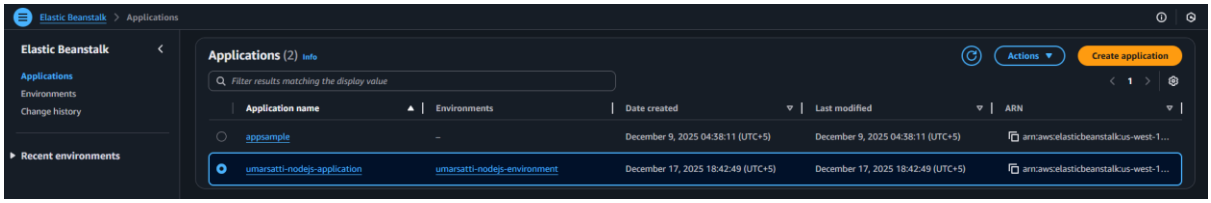




5.4: Elastic Beanstalk Validation

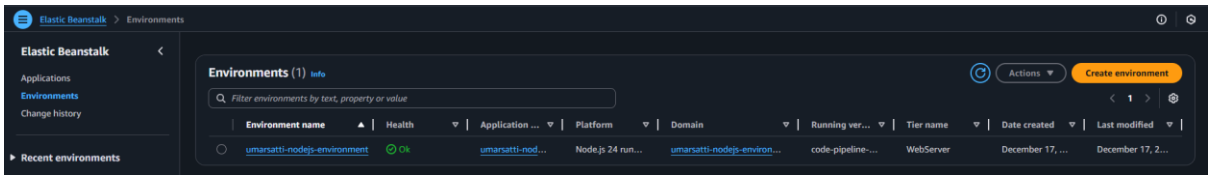
1. Application

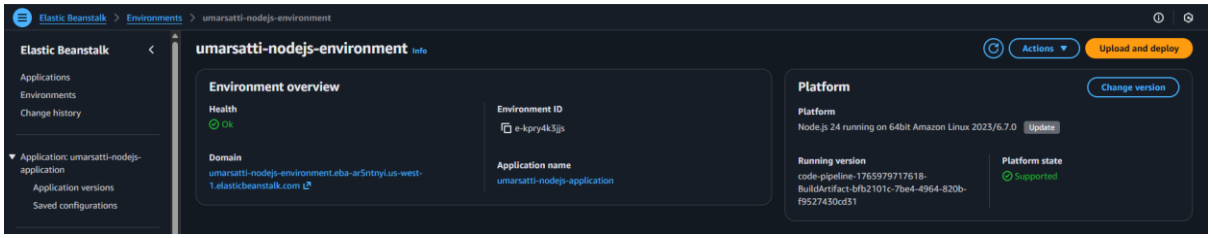
- Navigate to **Elastic Beanstalk** console using the search bar.
- Click **Application** option located in the left navigation bar.
- Verify the creation of the application named **umarsatti-nodejs-application**.



2. Environment

- Within the **Elastic Beanstalk**, click the Environments section located in the left navigation bar.
- Verify the creation of:
    - Environment name: umarsatti-nodejs-environment
    - Health Status: OK
    - Has a domain attached and runs on Node.js platform.

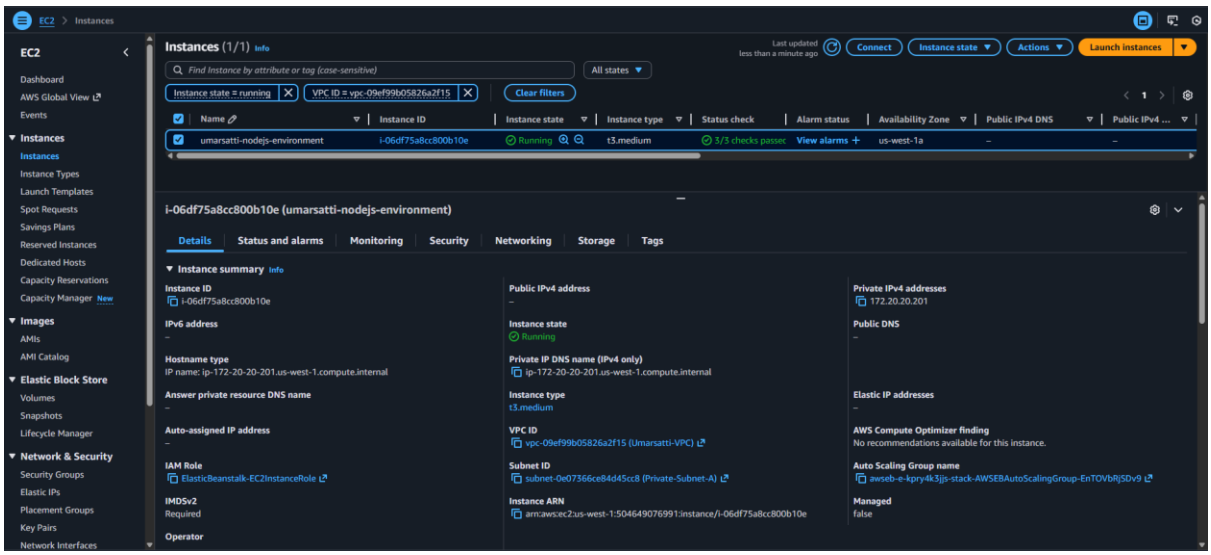




5.5: EC2, ALB, Target groups, and ASG Validation

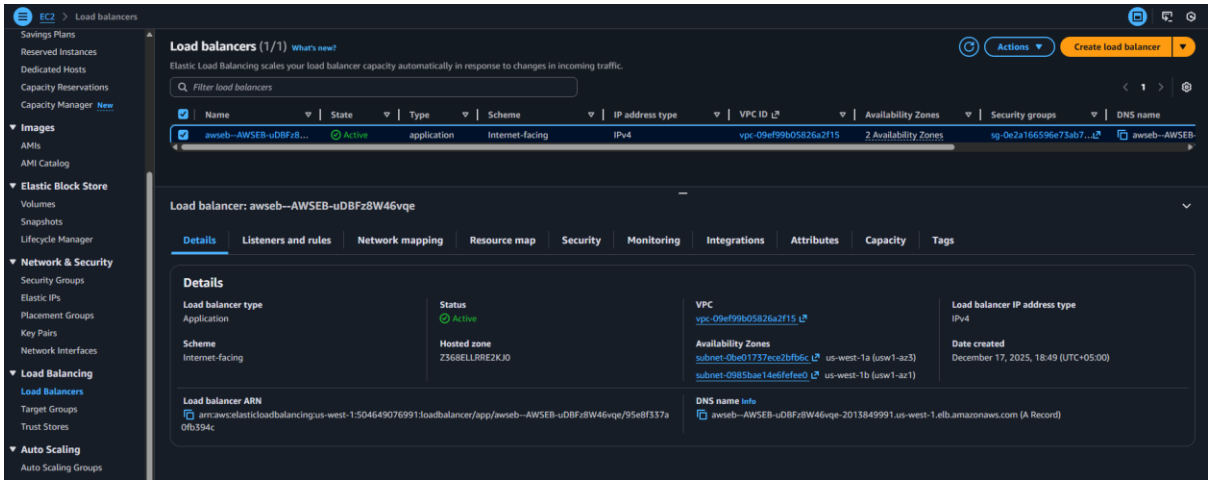
1. EC2 Instances

- Navigate to the **EC2 console** using the search bar.
- Click **Instances** option located in the left navigation bar.
- Verify creation of:
    - EC2 instance(s) created by Elastic Beanstalk
    - Instance profile attached
    - Instance in private subnet
    - Security group: EC2-SG
    - VPC: Umarsatti-VPC (created earlier)
    - Subnet: Private Subnet A or Private Subnet B
    - Private IPv4 address: 172.20.20.xxx or 172.20.25.xxx
    - Attached to the Beanstalk Auto Scaling Group
    - Instance type: t3.medium



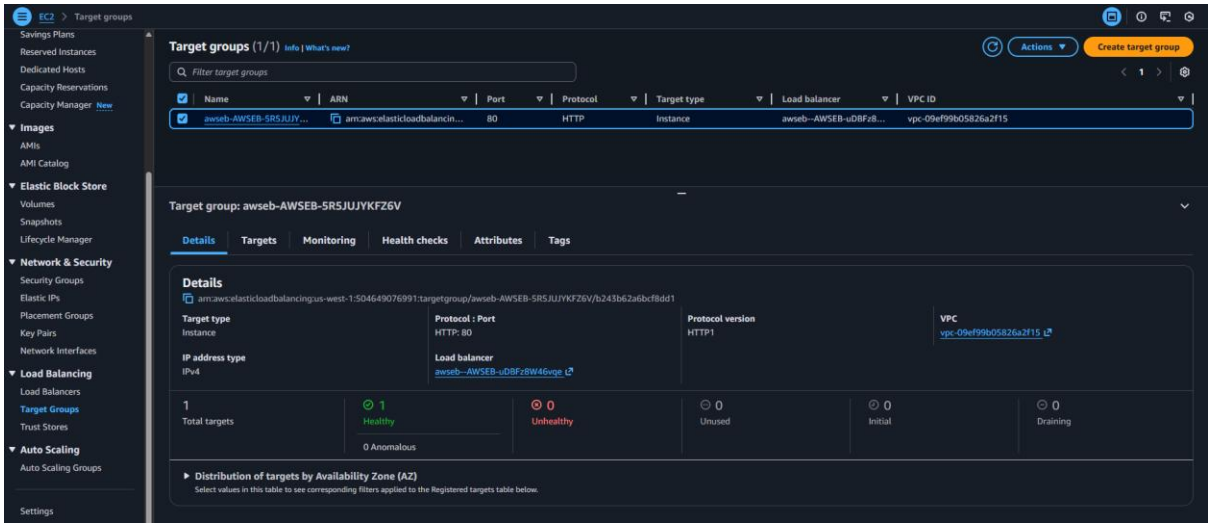
2. Load Balancer

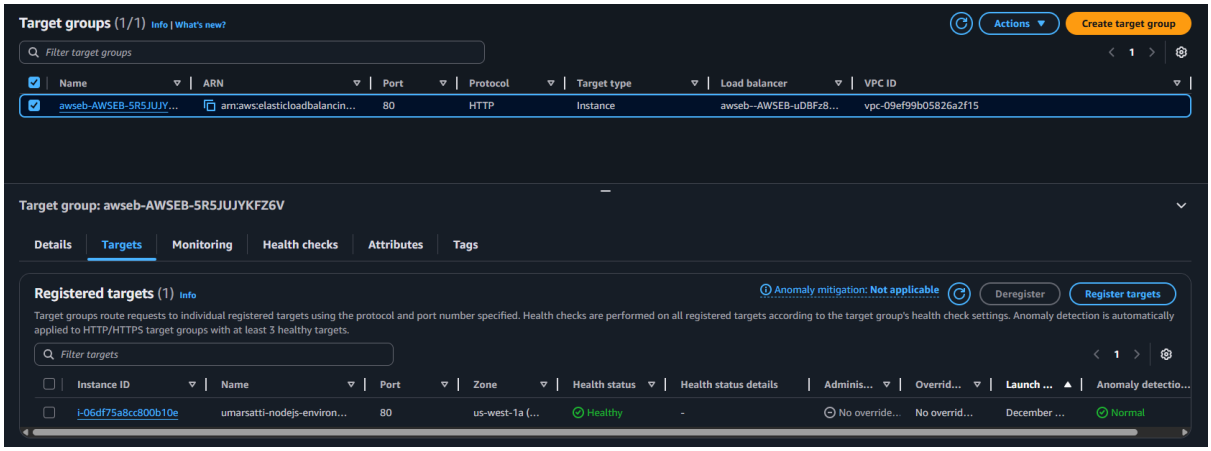
- Within the EC2 console, navigate to the **Load Balancer** section.
- Click **Load balancers** option located in the left navigation bar.
- Verify the creation of:
    - ALB created
    - Assigned to both public subnets
    - Listener: HTTP:80 → default target group
    - Security group: ALB-SG



3. Target Group

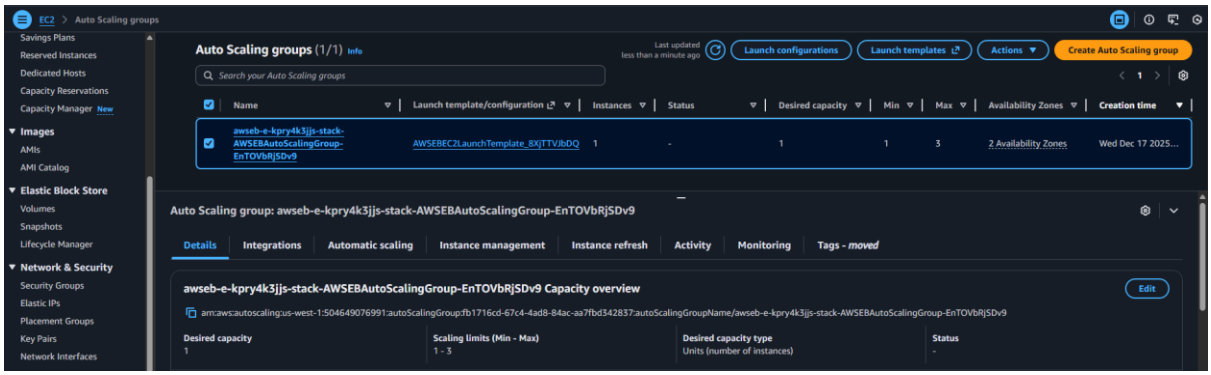
- Within the EC2 console, navigate to the **Target groups** section.
- Click **Target groups** option located in the left navigation bar.
- Verify the creation of:
    - Target group exists
    - Registered EC2 instance(s)
    - Port 3000 (or Port 80 due to Beanstalk auto configuration)
    - Target type: Instance
    - Attached to load balancer and VPC





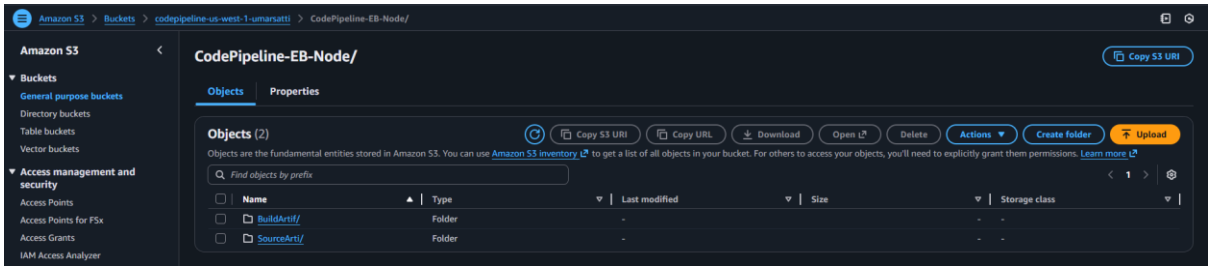
4. Auto Scaling Group

- Within the EC2 console, navigate to the **Target groups** section.
- Click **Target groups** option located in the left navigation bar.
- Verify the creation of:
    - ASG created by Elastic Beanstalk
    - Minimum instances = 1, Maximum instance = 3
    - Linked to Launch Template



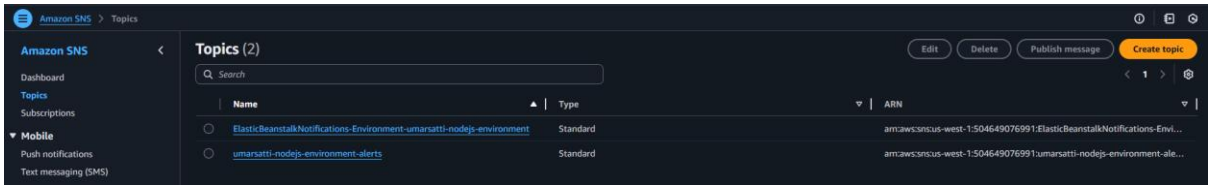
5.6: S3 Bucket Validation

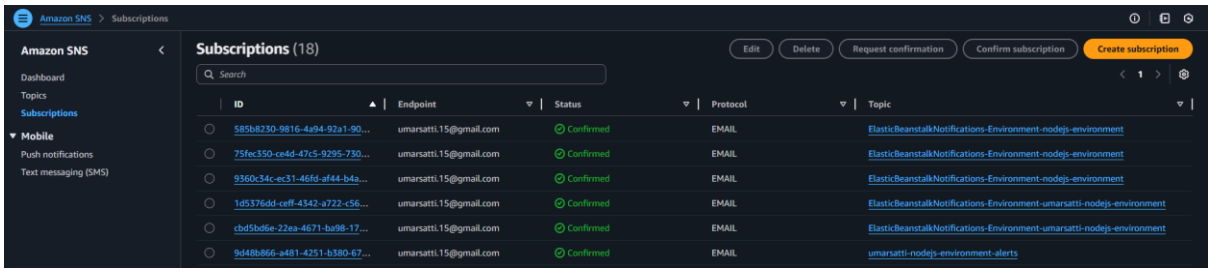
- Navigate to the **S3 console** using the search bar.
- Search for the bucket name (*codepipeline-us-west-1-umarsatti* in this case)
- Verify the creation of
    - Bucket: *codepipeline-us-west-1-umarsatti*
    - Objects: BuildArtifact/ and SourceArtifact/



5.7: SNS Validation

- Navigate to the **SNS console** using the search bar.
- Verify the creation of:
    - Topic created: <environment>-alerts
    - Email subscription to: umarsatti.15@gmail.com
    - Status: “Pending confirmation” (until email verified)

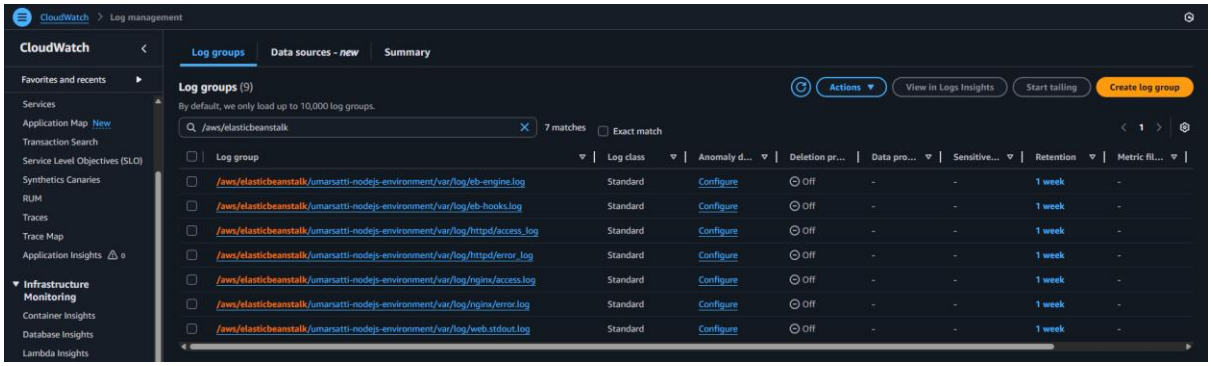




5.8: CloudWatch Validation

1. Log Groups

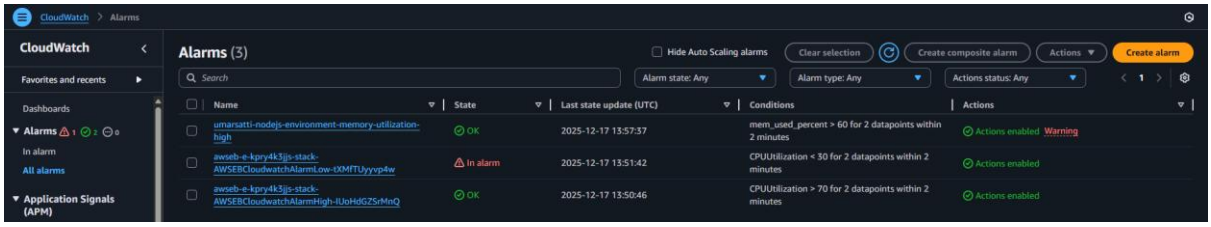
- Navigate to the **CloudWatch console** using the search bar.
- Under Logs section, select Log Management
- Search for the relevant Log groups that were created by Elastic Beanstalk.
    - Use **/aws/elasticbeanstalk** in the search bar
- Verify the creation of these logs



2. CloudWatch Alarms

- Within the **CloudWatch console**, locate the **Alarms** section.
- Under **Alarms** section, select All alarms to view the **CPU** and **Memory** alarms.

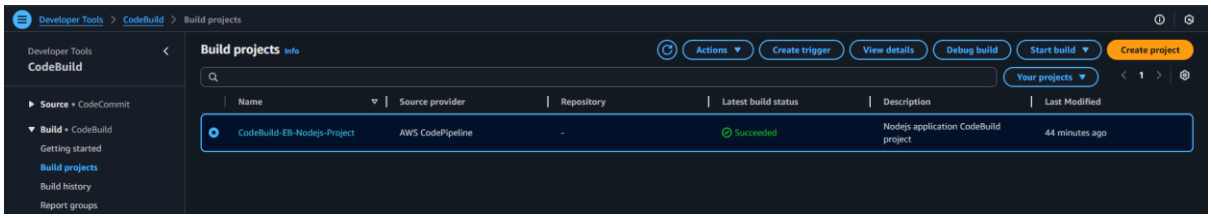
- Verify creation of:
    - Alarms: 1 nodejs-memory-utilization-high / 2 CPU alarms
    - Status: OK / In ALARM



5.9: CodeBuild

Steps:

- Navigate to the **CodeBuild** console using the search bar.
- Under **Build** section, select **Build projects**.
- Click on the project named **CodeBuild-EB-Nodejs-Project** to view details.



5.10: CodePipeline

Steps:

- Navigate to the **CodePipeline** console using the search bar.
- Under **Pipeline** section, select **Pipelines**.
- Click on the pipeline named **CodePipeline-EB-Nodejs** to view details.
- Verify that the pipeline is running without any issues. It should automatically run after Terraform infrastructure creation.


Developer Tools
CodePipeline
Pipelines
CodePipeline-EB-Nodejs
CodePipeline-EB-Nodejs
Edit
Stop execution
Create trigger
Clone pipeline
Release change
Pipeline
Executions
Triggers
Settings
Tags
Stage
Horizontal
Vertical
Source
148381ab-c9e8-4e55-b8e4-ce7f0bebbada
All actions succeeded.
Source
GitHub (via GitHub App)
37 minutes ago
bea147d2
Source: Applicatio
Build
148381ab-c9e8-4e55-b8e4-ce7f0bebbada
All actions succeeded.
Build
AWS CodeBuild
34 minutes ago
bea147d2
Source: Applicatio
DeployBackendAndFro
ntend
148381ab-c9e8-4e55-b8e4-ce7f0bebbada
All actions succeeded.
DeployBackendAndFrontend
AWS Elastic Beanstalk
33 minutes ago
bea147d2
Source: Applicatio

6. Application Testing, Alarm Verification, and Auto Scaling Validation

6.1: Setup Database using EC2

Steps:

- Navigate to **AWS Console → E2 Console → Instances** using left navigation bar.
- Select the EC2 named " " and click **Connect**. Use Sessions Manager to SSH into the instance since SSM agent is already installed.
- Connect to the instance using Sessions Manager.
- Install MySQL client using the following commands
    - sudo dnf install mariadb1011-server -y
    - mysql --version
- Navigate to the RDS console and select the database to get the DB Endpoint.
- Run the following commands
    - mysql -h ***<DB_ENDPOINT>*** -u ***<DB_USER>*** -p ***<DB_NAME>***
    - Type the password.

```
sh-5.2$ mysql -h ecommerce.ccaxt36ppifj.us-west-1.rds.amazonaws.com -u umarsatti -p ecommerce
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 48
Server version: 8.0.43 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [ecommerce]>
```

- Create tables using **createTables.sql** and mysql client.
    - createTables.sql is stored in app/database directory.
- Verify that the tables have been created
    - SHOW TABLES;
    - It should display 5 different tables.

```
MySQL [ecommerce]>
MySQL [ecommerce]> SHOW TABLES;
+---------------------+
| Tables_in_ecommerce |
+---------------------+
| orders              |
| product             |
| productsInOrder     |
| shopingCart         |
| users               |
+---------------------+
5 rows in set (0.002 sec)
```

- The RDS MySQL database is now set up. The application will function properly now.

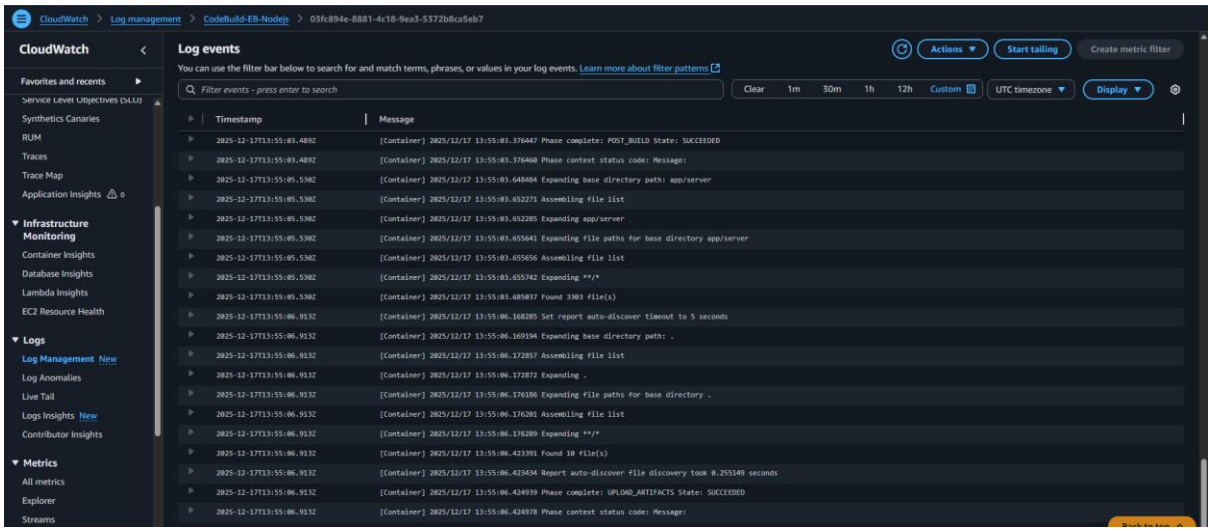
6.2: Application Logs using CloudWatch

Use CloudWatch logs to confirm that the database is set up and the application has no underlying issues or errors.

- Navigate to **CloudWatch console** using the search bar.
- Select **Log management** located in the left navigation bar under **Logs**.
- Locate the log group associated with the Elastic Beanstalk environment.
- Review the logs listed below to confirm successful execution.

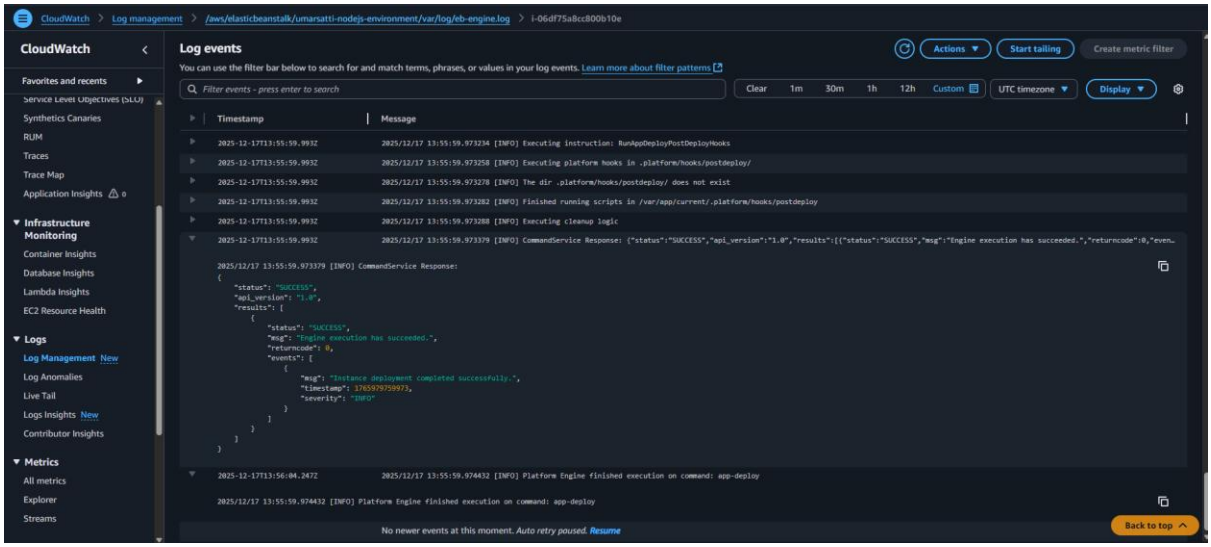
1. CodeBuild-EB-Nodejs

- Confirms that the build stage completed successfully.
- Logs show successful execution of install, build, and artifact upload phases.
- Indicates that the application package was correctly assembled and passed to CodePipeline for deployment.
- No errors during artifact upload confirms the buildspec.yml executed as expected.



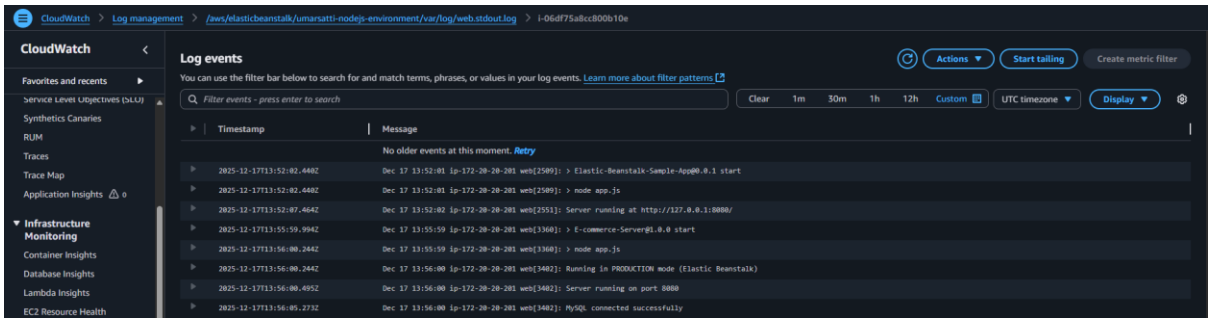
2. /var/log/eb-engine.log

- Provides detailed information about the Elastic Beanstalk deployment lifecycle.
- Confirms that the application bundle was downloaded from S3 and extracted successfully.
- Shows execution of .ebextensions configurations, CloudWatch agent setup, and environment initialization.
- Indicates successful service restarts, load balancer configuration, and health monitoring setup.
- Final log entries confirm that the deployment completed successfully without errors.



3. /var/log/web.stdout.log

- Displays application-level runtime logs from the Node.js backend.
- Confirms that the application started using **node app.js**.
- Shows the server running on port 8080, which is the default Elastic Beanstalk application port.
- Confirms successful connection to the MySQL database.
- Verifies that the application is running in PRODUCTION mode on Elastic Beanstalk.



/var/log/nginx/access.log

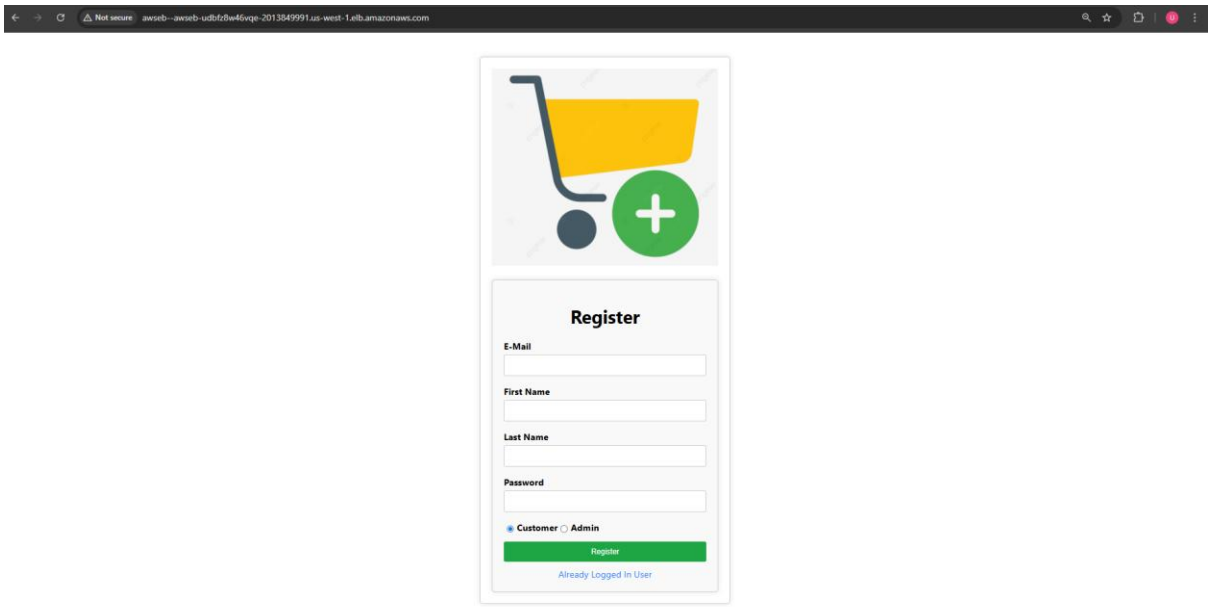
- Records incoming HTTP requests routed through the Application Load Balancer.
- Used to verify that traffic is reaching the application instances.
- Helps confirm successful client requests, response status codes, and request paths.

/var/log/nginx/error.log

- Captures errors related to NGINX, reverse proxy configuration, or request handling.
- Used to troubleshoot failed requests, misconfigurations, or connectivity issues.
- Absence of critical errors confirms stable proxy and load balancer behavior.

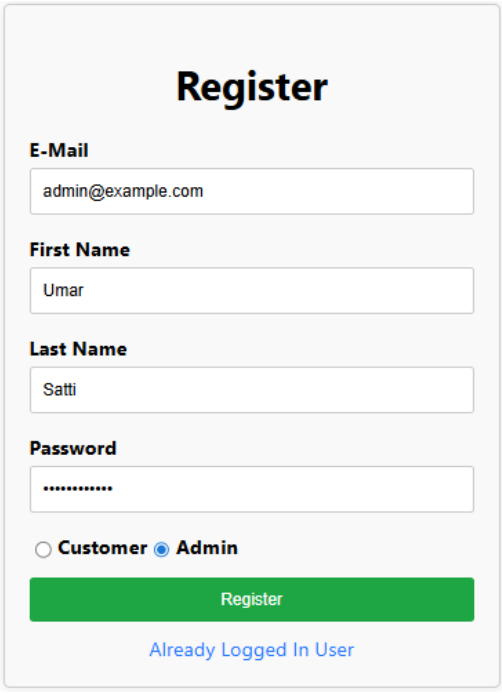
6.3: Test the Application

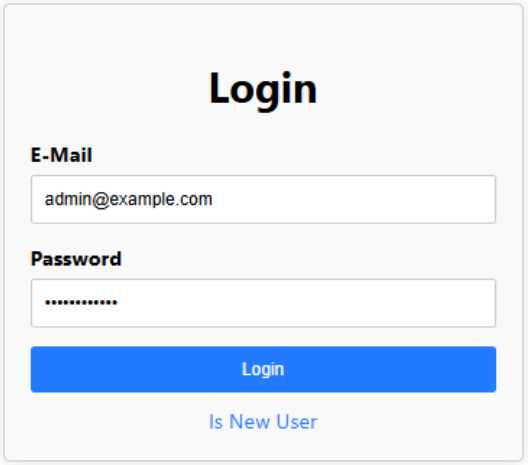
- Navigate to **AWS Console → Elastic Beanstalk → Environments**.
- Copy the **Environment URL** generated by Elastic Beanstalk.
- Open the URL in a browser. It should display the application.
- The image below displays the Node.js application load successfully.

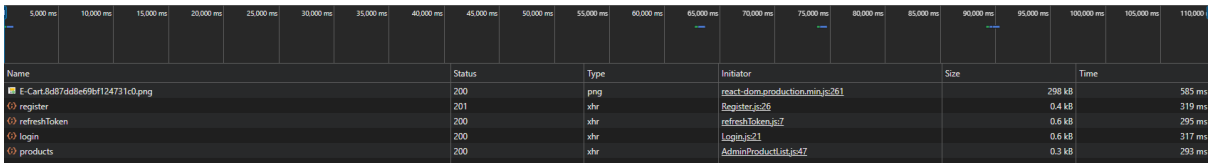


URL: ***http://awseb--awseb-udbfz8w46vqe-2013849991.us-west-1.elb.amazonaws.com/***

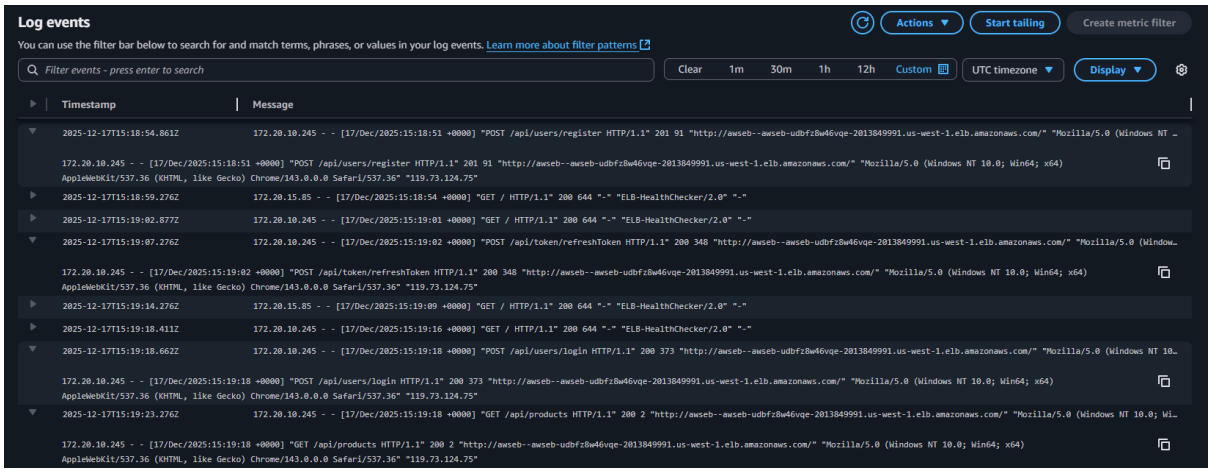
Registration, Login, and Product Logs

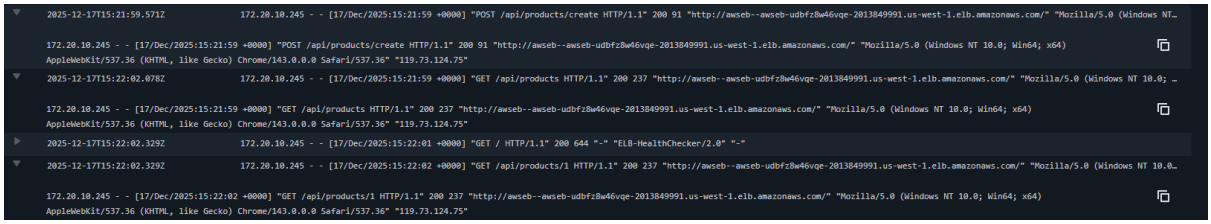






Name	Status	Type	Initiator	Size	Time
E-Cart.8d87dd8e69bf124731c0.png	200	png	react-dom.production.min.js:261	298 kB	585 ms
register	201	xhr	Register.js:26	0.4 kB	319 ms
refreshToken	200	xhr	refreshToken.js:7	0.6 kB	295 ms
login	200	xhr	Login.js:21	0.6 kB	317 ms
products	200	xhr	AdminProductList.js:47	0.3 kB	293 ms





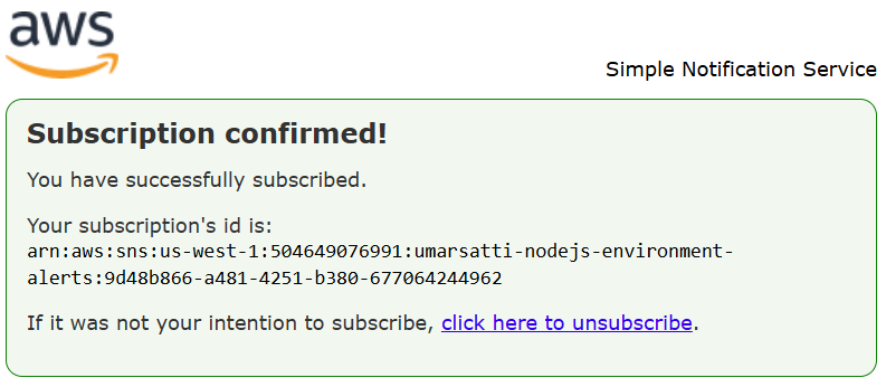
6.4: Validate Email Subscriptions for CloudWatch Alarms

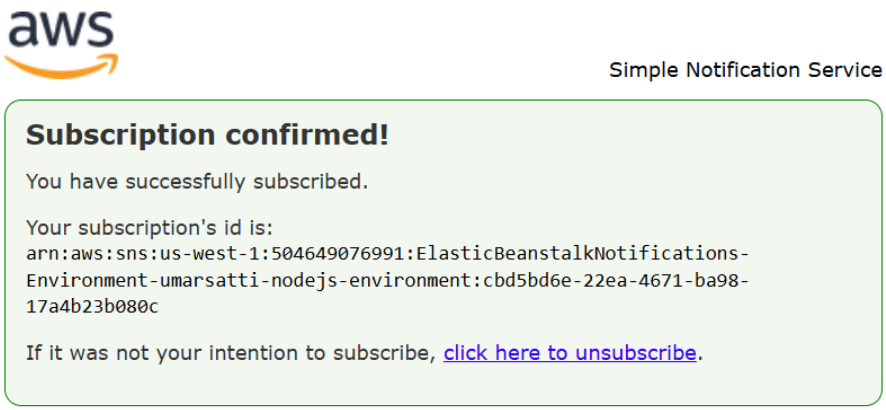
Two SNS topics were created for:

- CPU Utilization alarm notifications
- Memory Utilization alarm notifications

Steps:

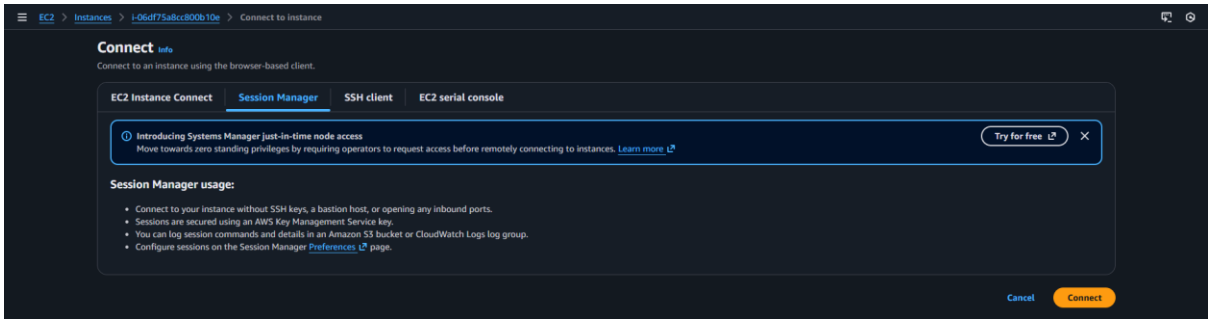
1. Check email inbox.
2. Confirm the two received **subscription confirmation emails**.
3. Accept both subscriptions.





6.5: Connect to EC2 Instance Using Session Manager

- Navigate to **EC2 console**
- Select **Instances** option located in the left navigation bar.
- Select the instance created by Elastic Beanstalk.
- Click **Connect** button on the top right.
- Choose the **Session Manager** tab.
- Start a session.



6.6: Verify CloudWatch Agent on EC2 Instance

Run the following commands inside the Session Manager terminal:

1. Check CloudWatch Agent Status

- Run the following commands inside the Session Manager terminal
- The expected output should display the agent running
- Command to use:
    - ***sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status***

```
sh-5.2$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
{
  "status": "running",
  "starttime": "2025-12-17T13:55:56+00:00",
  "configstatus": "configured",
  "version": "1.300060.1"
}
sh-5.2$
```

2. View CloudWatch Agent Configuration File

- Run the following commands inside the Session Manager terminal
- The expected output should display a custom memory metrics JSON
- Command to use:
    - ***sudo cat /opt/aws/amazon-cloudwatch-agent/bin/config.json***

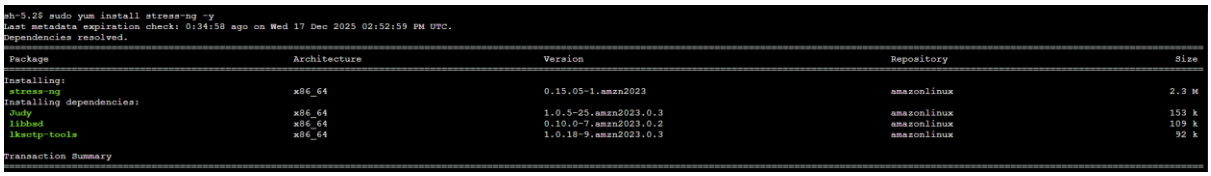
```
sh-5.2$ sudo cat /opt/aws/amazon-cloudwatch-agent/bin/config.json
{
  "agent": {
    "metrics_collection_interval": 60,
    "run_as_user": "root"
  },
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "metrics_collected": {
      "mem": {
        "measurement": [
          "mem_used_percent"
        ]
      }
    }
  }
}
sh-5.2$
```

6.7: Install and Run stress-ng to Trigger Alarms

These tests simulate high CPU and memory usage to trigger CloudWatch alarms and auto-scaling of instances.

1. Install stress-ng utility on EC2

- *sudo yum install stress-ng -y*



2. Trigger CPU Alarm (scale-out)

- *stress-ng --cpu 0 --cpu-load 90 --timeout 300*

```
sh-5.2$ sudo stress-ng --cpu 0 --cpu-load 85 --timeout 300
stress-ng: info:  [33180] setting to a 300 second (5 mins, 0.00 secs) run per stressor
stress-ng: info:  [33180] dispatching hogs: 2 cpu
stress-ng: info:  [33180] successful run completed in 300.00s (5 mins, 0.00 secs)
sh-5.2$
```

Expected output:

- CPU alarm should move from **OK → ALARM**.
- Auto Scaling Group should launch a new EC2 instance.

3. Trigger Memory Alarm (scale-out)

- *sudo stress-ng --vm 1 --vm-bytes 2500M --vm-keep --vm-method all --vm-hang 240 -t 240s*

```
sh-5.2$ sudo stress-ng --vm 1 --vm-bytes 2500M --vm-keep --vm-method all --vm-hang 240 -t 240s
stress-ng: info:  [30565] setting to a 240 second (4 mins, 0.00 secs) run per stressor
stress-ng: info:  [30565] dispatching hogs: 1 vm
stress-ng: warn:  [30565] metrics-check: all bogo-op counters are zero, data may be incorrect
stress-ng: info:  [30565] successful run completed in 240.10s (4 mins, 0.10 secs)
sh-5.2$
```

Expected output:

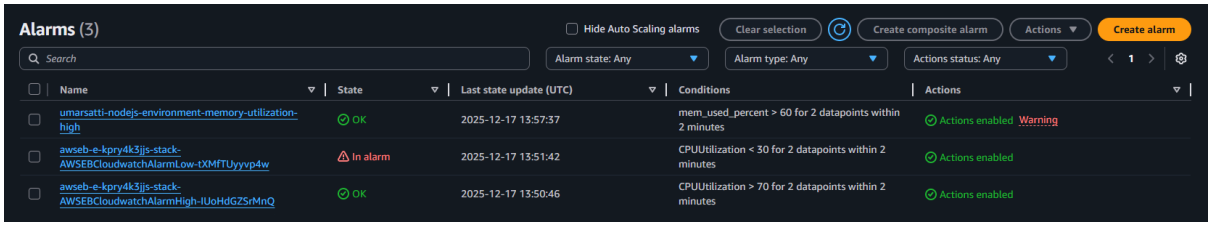
- Memory alarm should go **OK → ALARM**.
- ASG may launch additional EC2 capacity depending on scaling policies.

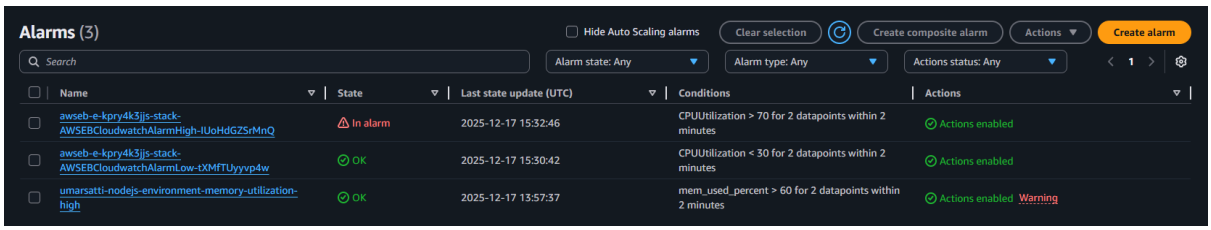
6.8: Verify Alarm State Change in CloudWatch

- Navigate to the **CloudWatch console**.
- Select the **All alarms** option under the **Alarms** section in the left navigation bar.
- Verify the state changes for both the CPU and Memory alarms.

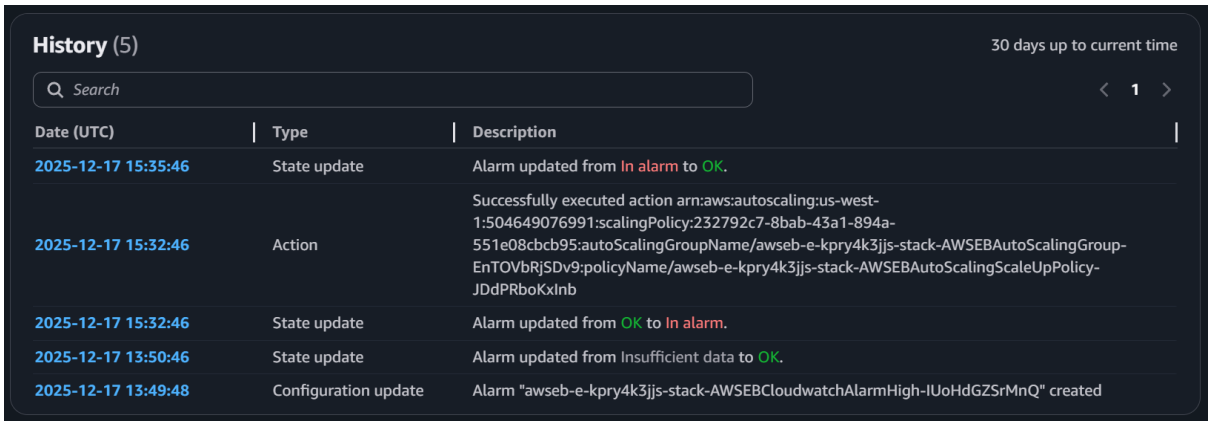
CPUUtilization Alarm

1. Transitions from **OK** to **ALARM**.

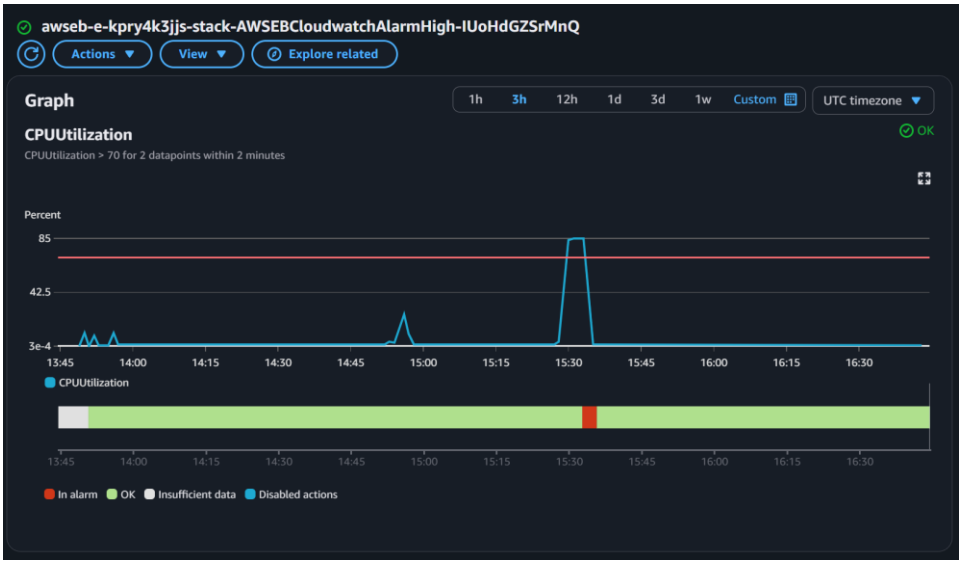




2. **CPUUtilization > 70** Alarm history shows the state change.

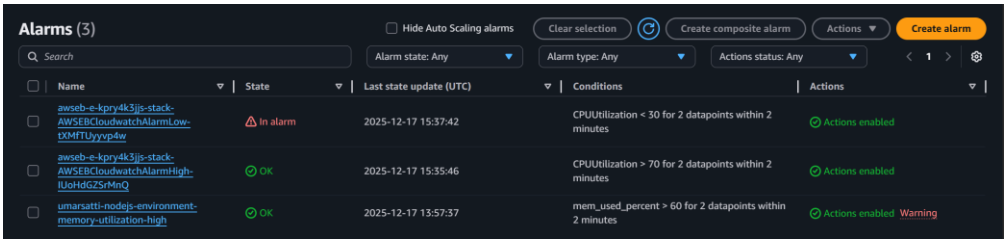


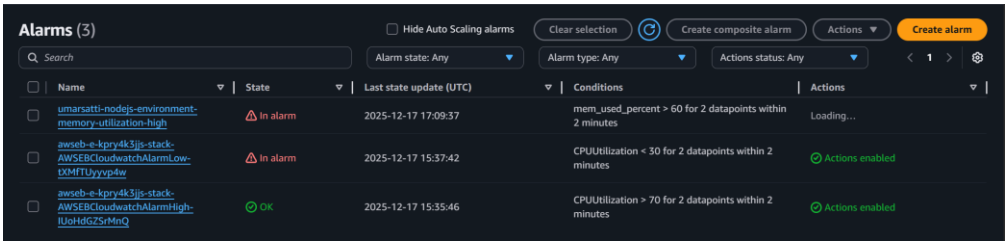
3. **CPUUtilization** graph showing CPU usage going up to 90%



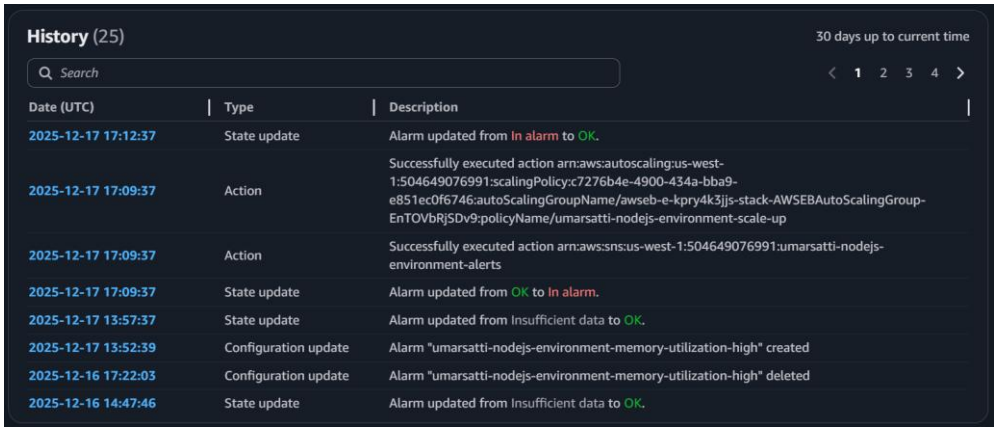
MemoryUtilization Alarm

1. Transitions from **OK** to **IN ALARM** and back to **OK** state.

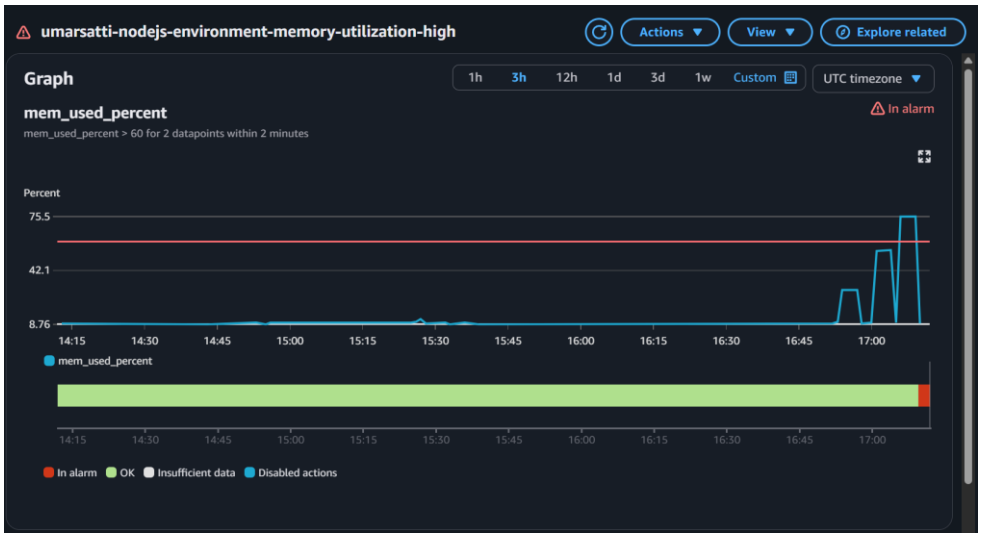




2. **Memory Utilization > 60** Alarm history shows the state change.

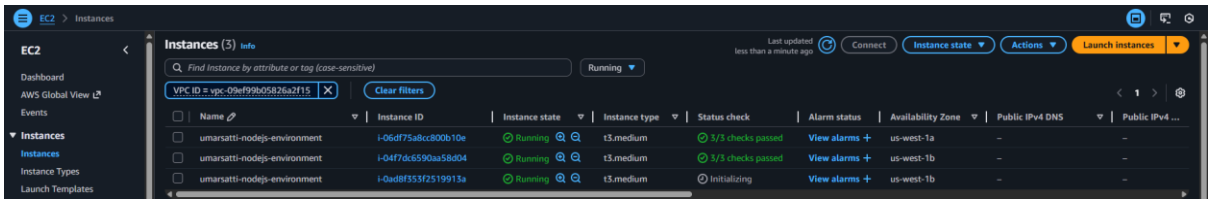


3. **Memory Utilization** graph showing CPU usage going up to 73%

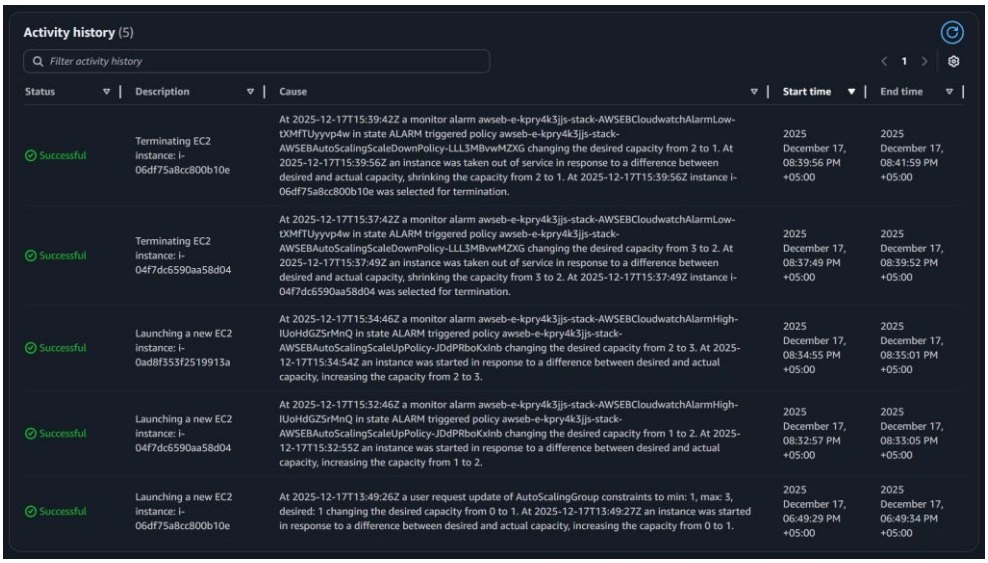


6.9: Validate Auto Scaling Activity

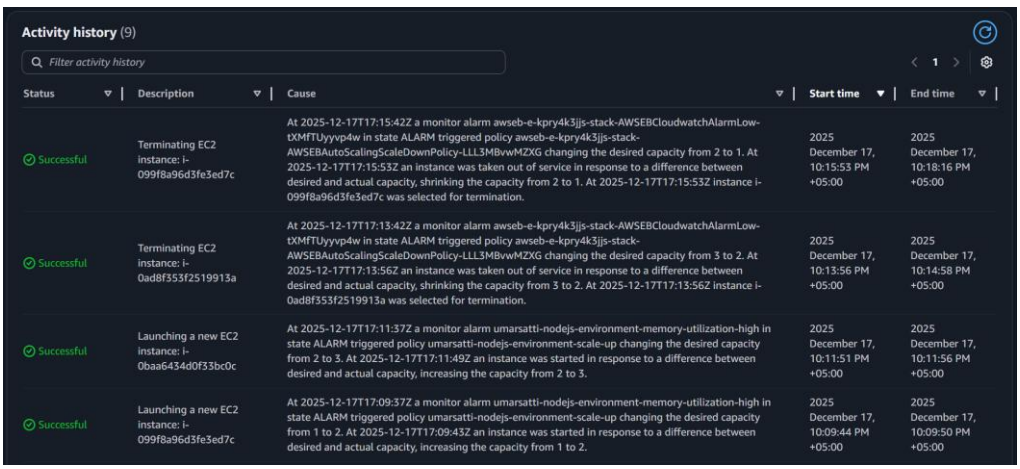
- Navigate to the EC2 console and click the **Auto Scaling Group** option.
- Verify:
    - A new EC2 instance was launched due to triggered alarms.
    - Activity history shows scale out events from 1 instance to 2 or 3 instances (as maximum was set to 3).
    - Activity history shows scale in events when CPU and Memory utilization goes back to normal.



1. Scale-in and Scale-out during CPUUtilization alarm (Auto Scaling group)

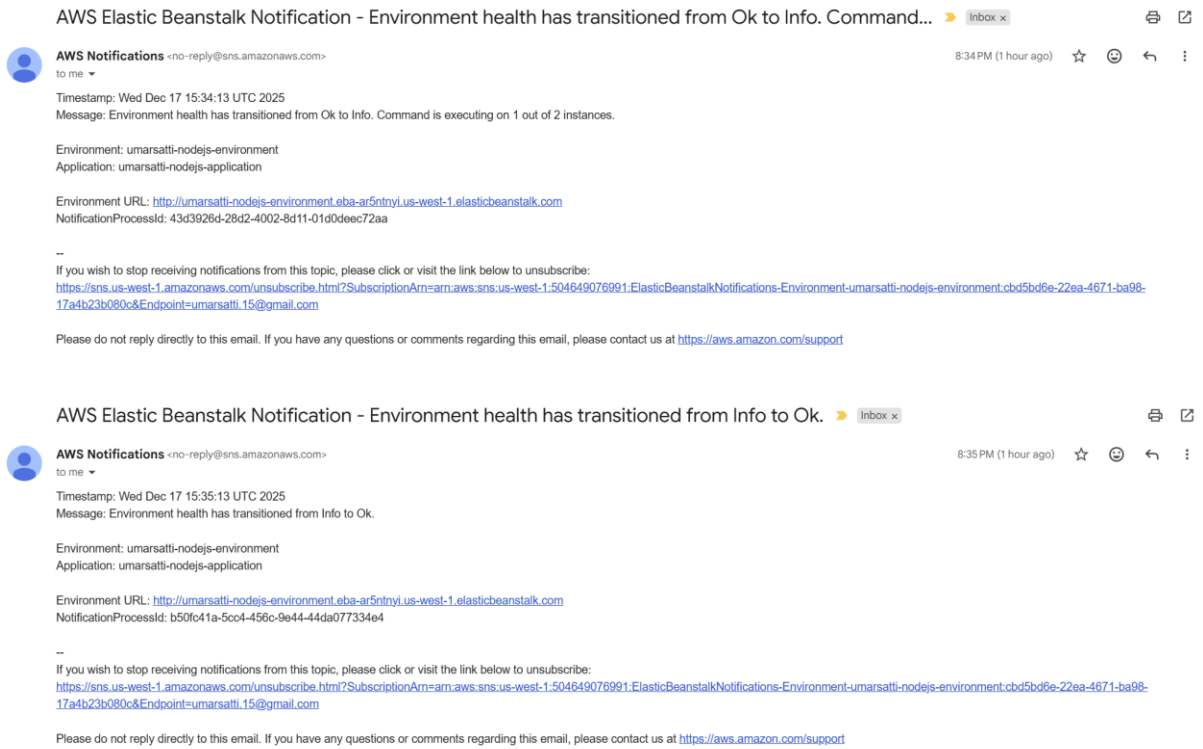


2. Scale-in and Scale-out during Memory Utilization alarm (Auto Scaling group)



6.10: SNS Email Notifications

1. Verification of CPU Utilization Alarm – Email Notification



AWS Elastic Beanstalk Notification - Environment health has transitioned from Ok to Info. Command... Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 8:34 PM (1 hour ago)
to me

Timestamp: Wed Dec 17 15:34:13 UTC 2025
Message: Environment health has transitioned from Ok to Info. Command is executing on 1 out of 2 instances.

Environment: umarsatti-nodejs-environment
Application: umarsatti-nodejs-application

Environment URL: http://umarsatti-nodejs-environment.eba-ar5ntnyi.us-west-1.elasticbeanstalk.com
NotificationProcessId: 43d3926d-28d2-4002-8d11-01d0deec72aa

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-west-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-west-1:504649076991:ElasticBeanstalkNotifications-Environment-umarsatti-nodejs-environment:cbd5bd6e-22ea-4671-ba98-17a4b23b080c&Endpoint=umarsatti.15@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

AWS Elastic Beanstalk Notification - Environment health has transitioned from Info to Ok. Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 8:35 PM (1 hour ago)
to me

Timestamp: Wed Dec 17 15:35:13 UTC 2025
Message: Environment health has transitioned from Info to Ok.

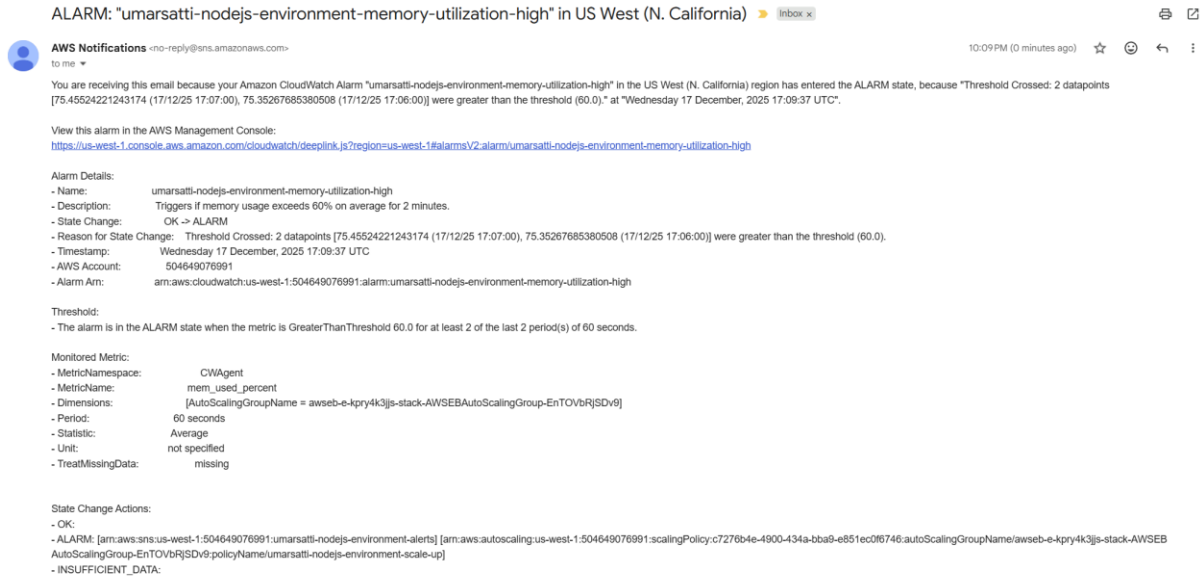
Environment: umarsatti-nodejs-environment
Application: umarsatti-nodejs-application

Environment URL: http://umarsatti-nodejs-environment.eba-ar5ntnyi.us-west-1.elasticbeanstalk.com
NotificationProcessId: b50fc41a-5cc4-456c-9e44-44da077334e4

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-west-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-west-1:504649076991:ElasticBeanstalkNotifications-Environment-umarsatti-nodejs-environment:cbd5bd6e-22ea-4671-ba98-17a4b23b080c&Endpoint=umarsatti.15@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

2. Verification of Memory Utilization Alarm – Email Notification



ALARM: "umarsatti-nodejs-environment-memory-utilization-high" in US West (N. California) Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 10:09 PM (0 minutes ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "umarsatti-nodejs-environment-memory-utilization-high" in the US West (N. California) region has entered the ALARM state, because "Threshold Crossed: 2 datapoints [75.45524221243174 (17/12/25 17:07:00), 75.35267685380508 (17/12/25 17:06:00)] were greater than the threshold (60.0)." at "Wednesday 17 December, 2025 17:09:37 UTC".

View this alarm in the AWS Management Console:
https://us-west-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-west-1#alarmsV2:alarm/umarsatti-nodejs-environment-memory-utilization-high

Alarm Details:
- Name: umarsatti-nodejs-environment-memory-utilization-high
- Description: Triggers if memory usage exceeds 60% on average for 2 minutes.
- State Change: OK -> ALARM
- Reason for State Change: Threshold Crossed: 2 datapoints [75.45524221243174 (17/12/25 17:07:00), 75.35267685380508 (17/12/25 17:06:00)] were greater than the threshold (60.0).
- Timestamp: Wednesday 17 December, 2025 17:09:37 UTC
- AWS Account: 504649076991
- Alarm Arn: arn:aws:cloudwatch:us-west-1:504649076991:alarm:umarsatti-nodejs-environment-memory-utilization-high

Threshold:
- The alarm is in the ALARM state when the metric is GreaterThanThreshold 60.0 for at least 2 of the last 2 period(s) of 60 seconds.

Monitored Metric:
- MetricNamespace: CWAgent
- MetricName: mem_used_percent
- Dimensions: [AutoScalingGroupName = awseb-e-kpry4k3jjs-stack-AWSEBAutoScalingGroup-EnTOVbRjSDv9]
- Period: 60 seconds
- Statistic: Average
- Unit: not specified
- TreatMissingData: missing

State Change Actions:
- OK:
- ALARM: [arn:aws:sns:us-west-1:504649076991:umarsatti-nodejs-environment-alerts] [arn:aws:autoscaling:us-west-1:504649076991:scalingPolicy:c7276b4e-4900-434a-bba9-e851ec0f6746:autoScalingGroupName/awseb-e-kpry4k3jjs-stack-AWSEBAutoScalingGroup-EnTOVbRjSDv9:policyName/umarsatti-nodejs-environment-scale-up]
- INSUFFICIENT_DATA:

7. Clean Up

After completing all testing and validation, the final step is to tear down the AWS infrastructure created by Terraform. This ensures that no unnecessary resources remain and prevents ongoing AWS charges.

1. Destroy the Terraform Infrastructure

Navigate to the **root** Terraform directory, then run ***terraform destroy --auto-approve***

This command performs the following actions:

- Reads the Terraform state file
- Identifies all AWS resources created during terraform apply
- Deletes them in the correct dependency order
- Automatically approves destruction without prompting

8. Troubleshooting

During the deployment of the three-tier e-commerce application using Terraform, AWS Elastic Beanstalk, CodePipeline, and CodeBuild, several issues were encountered. This section outlines each problem, explains the root cause, and describes the corrective actions taken to successfully complete the deployment.

Issue 1: CodePipeline Build Stage – Missing EC2 Network Permissions

Problem Description:
The CodePipeline build stage failed during execution, indicating that it was not authorized to perform the **DeleteNetworkInterface** action.

Root Cause:
When CodeBuild runs inside a VPC, it creates and deletes network interfaces. The CodePipeline service role did not include the required EC2 permissions to perform this operation.

Solution:
The missing permission was added to the CodePipeline IAM policy:

- ec2:DeleteNetworkInterface

After attaching the updated policy, the build stage executed successfully.

Issue 2: CodePipeline Deploy Stage – Missing Elastic Beanstalk Permissions

Problem Description:
The deploy stage failed with an error stating that the role did not have permission to create an Elastic Beanstalk application version.

Root Cause:
The CodePipeline service role lacked the elasticbeanstalk:CreateApplicationVersion permission, which is required during deployment.

Solution:
The required permission was added to the CodePipeline IAM policy, enabling CodePipeline to create and deploy new application versions.

Issue 3: Duplicate IAM Policy Attachment Resource

Problem Description:
Terraform failed with an error indicating a duplicate aws_iam_role_policy_attachment resource.

Root Cause:
The same resource name was accidentally declared twice within the IAM module, which is not allowed in Terraform.

Solution:
The duplicate resource was renamed to a unique identifier, resolving the conflict and allowing Terraform to proceed.

Issue 4: Unsupported CodeBuild Image

Problem Description:
The CodeBuild project creation failed due to an unsupported build image configuration.

Root Cause:
The image aws/codebuild/ami/amazonlinux-x86_64-base:latest is not compatible with the selected Linux container environment and compute type.

Solution:
The image was updated to a supported standard image:

- aws/codebuild/amazonlinux-x86_64-standard:5.0
  This resolved the project creation error.

Issue 5: IAM Policy Already Exists

Problem Description:
Terraform failed while creating an IAM policy, reporting that a policy with the same name already existed.

Root Cause:
The same IAM policy name was mistakenly reused for both CodeBuild and CodePipeline roles.

Solution:
The correct variable reference was applied to ensure each policy had a unique name, preventing further duplication errors.

Issue 6: Invalid CodeBuild Source Configuration

Problem Description:
CodeBuild project creation failed with an error stating that both source and artifact types must be set to CODEPIPELINE.

Root Cause:
The CodeBuild project was incorrectly configured to use GitHub as the source instead of CodePipeline.

Solution:
The source and artifact types were updated to CODEPIPELINE, allowing CodeBuild to integrate properly with the pipeline.

Issue 7: CodePipeline Cannot Start CodeBuild Project

Problem Description:
The pipeline build stage failed due to insufficient permissions to start a CodeBuild project.

Root Cause:
The CodePipeline IAM role did not include permissions to trigger CodeBuild actions.

Solution:
The following permissions were added to the CodePipeline IAM policy:

- codebuild:StartBuild
- codebuild:BatchGetBuilds
- codebuild:StopBuild

Issue 8: VPC_CLIENT_ERROR During Build

Problem Description:
The build stage failed during provisioning with a VPC_CLIENT_ERROR and an unauthorized operation message.

Root Cause:
The CodePipeline role lacked permissions required for network interface operations within the VPC.

Solution:
The permission ec2:CreateNetworkInterfacePermission was added, resolving the provisioning failure.

Issue 9: Elastic Beanstalk Deployment Failed – Missing Application Files

Problem Description:
Elastic Beanstalk deployment failed due to missing application files such as package.json, app.js, or Procfile.

Root Cause:
The CodePipeline artifact path was incorrectly defined, causing the application bundle to be packaged improperly.

Solution:
The artifact output path was corrected from output/app.zip to app.zip, ensuring the required files were present at the root of the deployment bundle.

Issue 10: Pipeline Retry Not Allowed

Problem Description:
The deploy stage could not be retried due to pipeline configuration changes after the failure.

Root Cause:
CodePipeline does not allow retries when the pipeline definition has been modified.

Solution:
A new pipeline execution was triggered instead of retrying the failed stage.

Issue 11: Elastic Beanstalk Logs Indicated Missing Source Files

Problem Description:
Elastic Beanstalk logs showed that required application files were missing after extraction.

Root Cause:
The artifact root directory structure was incorrect.

Solution:
The application packaging process was corrected to ensure all required files were placed in the root of the application bundle. For this use-case, the root is the server (backend) folder.

Issue 12: MySQL Connection Refused

Problem Description:
Application logs showed a connection refusal error when attempting to connect to MySQL.

Root Cause:
The application was attempting to connect to MySQL on localhost instead of the RDS endpoint.

Solution:
Correct RDS environment variables were configured in the Elastic Beanstalk environment.

Issue 13: Frontend API URL Defaulting to Localhost

Problem Description:
The frontend application was sending API requests to a local backend URL (localhost:3001) instead of the deployed backend.

Root Cause:
The frontend code explicitly included a fallback to http://localhost:3001 when no environment variable was provided, causing the application to default to localhost in production.

Solution:
The API base URL logic was updated to use a relative path ("/") instead of a hardcoded localhost fallback. This allows the frontend to automatically route API requests through the same domain and load balancer as the deployed backend.

Issue 14: RDS Database Tables Not Created

Problem Description:
The application failed because the required database tables did not exist.

Root Cause:
Elastic Beanstalk and CodeBuild do not automatically execute SQL schema files.

Solution:
The database schema was manually applied by connecting to the RDS instance and executing the SQL commands.

Issue 15: MySQL Promise Pool Misconfiguration

Problem Description:
API endpoints failed with 500 errors due to improper MySQL pool usage.

Root Cause:
A non-promise MySQL pool was exported while the application attempted to use async/await.

Solution:
The pool was wrapped using .promise() before exporting, enabling proper asynchronous database operations.

Issue 16: Improper Use of pool.query

Problem Description:
Database queries failed or behaved inconsistently.

Root Cause:
Callback-based queries were mixed with async/await syntax.

Solution:
All database queries within the application files were refactored to use async/await with proper error handling and logging.