TASK 10

Deploying a Simple Node.js Application on AWS Elastic Beanstalk using Terraform

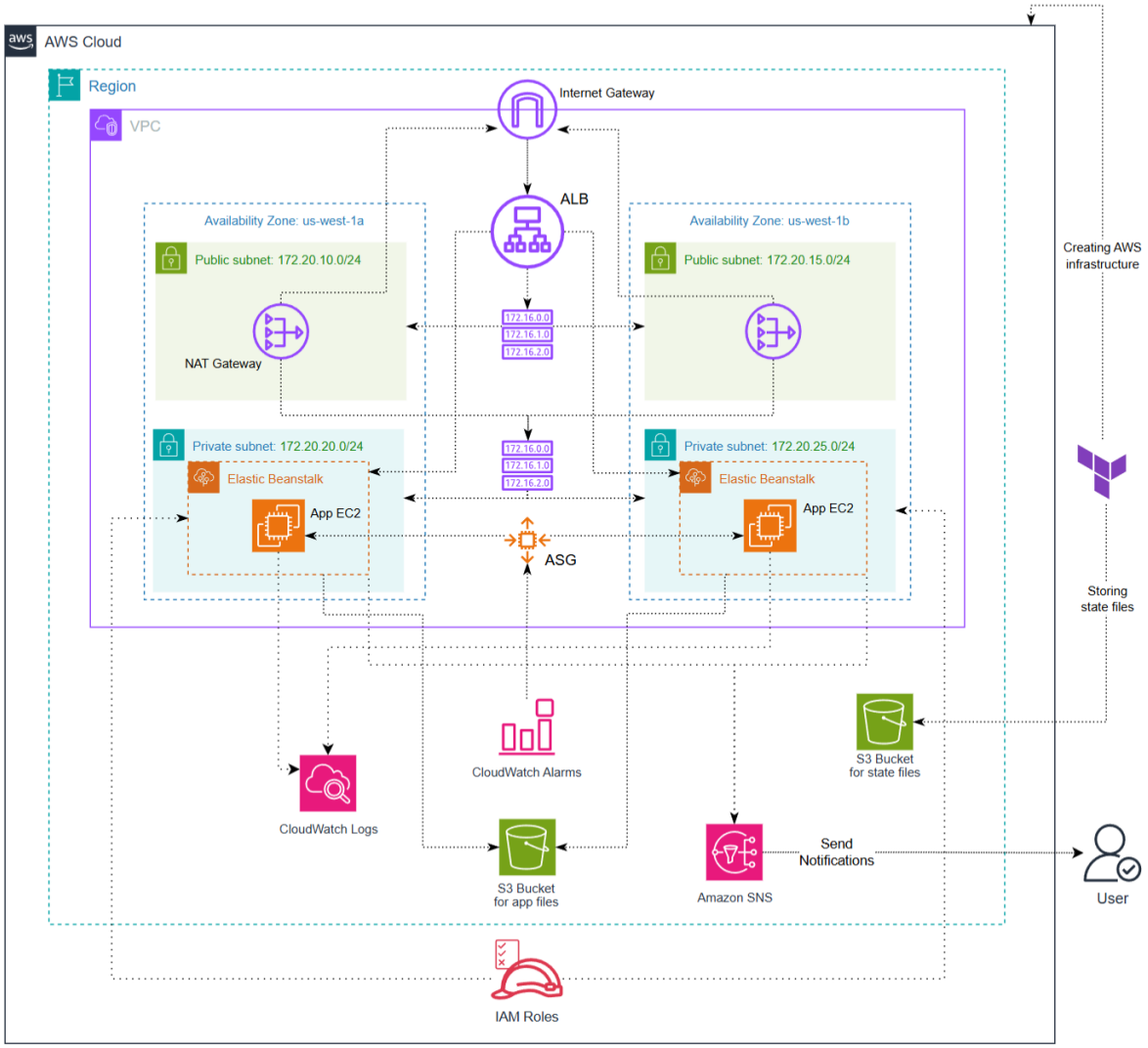
Umar Satti

Table of Contents

Task Description

This task involves deploying a Node.js application to AWS Elastic Beanstalk using Terraform. It includes creating the application and environment, uploading and deploying application code to S3, configuring platform and scaling settings, enabling monitoring and health checks, and validating the deployment through Elastic Beanstalk endpoints and CloudWatch metrics.

Architecture Diagram



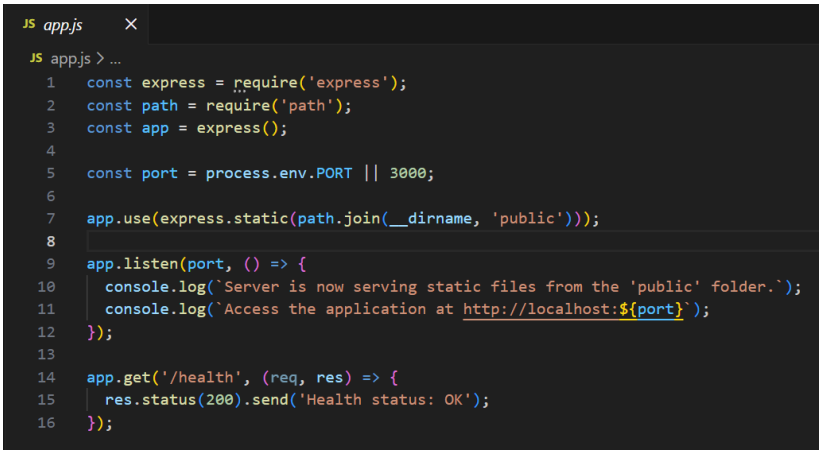
Task 1.1: Test Application Locally

Before deploying to AWS, we test the Node.js application locally to ensure all components function correctly.

1.1.1: Files Overview

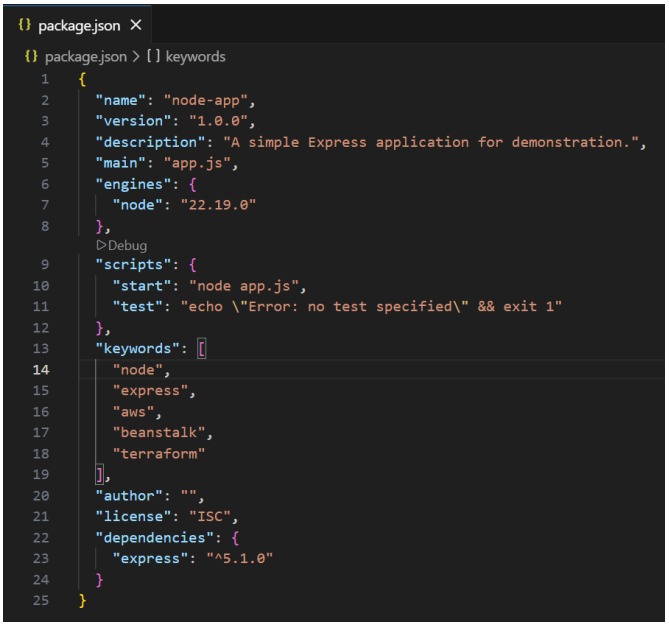
1. app.js

- Main application file.
- Serves static files from the public folder.
- Provides a /health endpoint for status checking.
- Starts the Express server on PORT (default 3000).



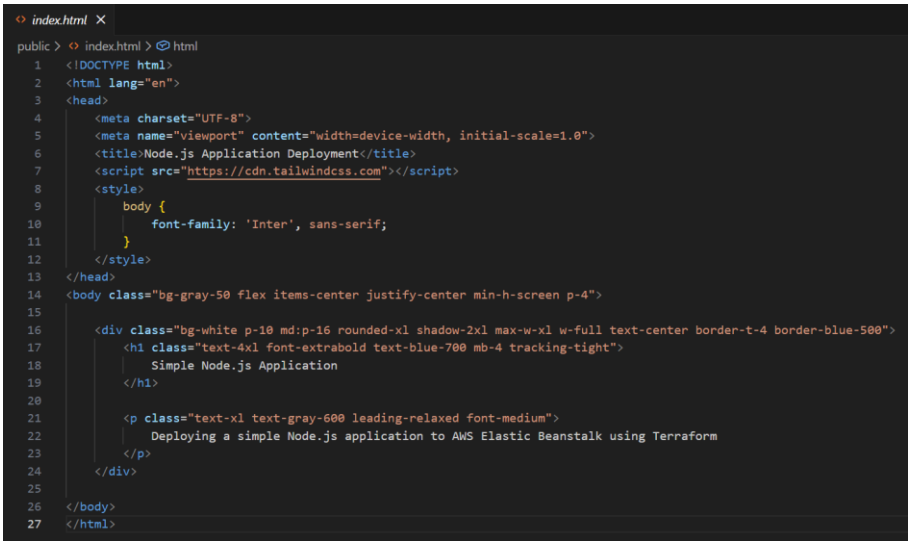
2. package.json

- Defines project metadata, dependencies, and scripts.
- Ensures the correct Node.js version (22.19.0).
- Declares **Express** as the only dependency.



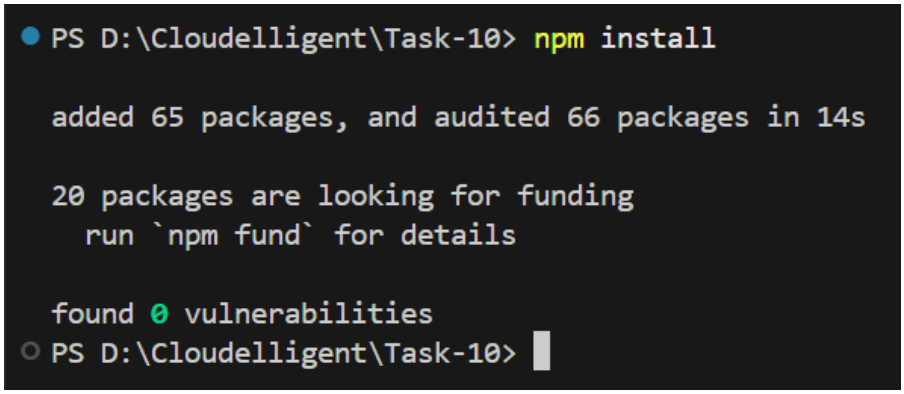
3. public/index.html

- Main HTML page displayed to users.
- Uses Tailwind CSS for styling.
- Contains a simple heading and description for the Node.js app.

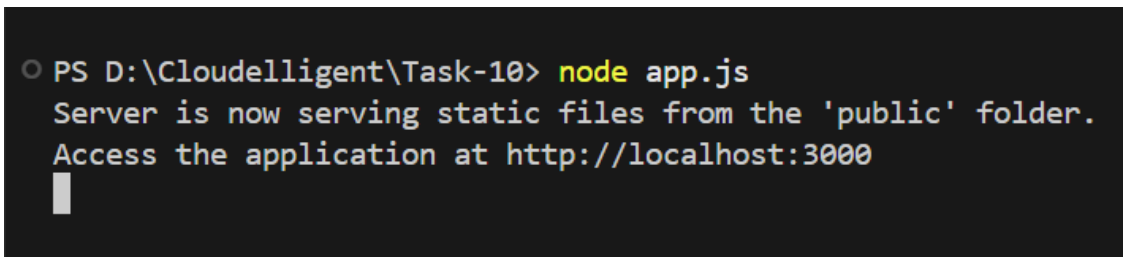


1.1.2: Test Steps

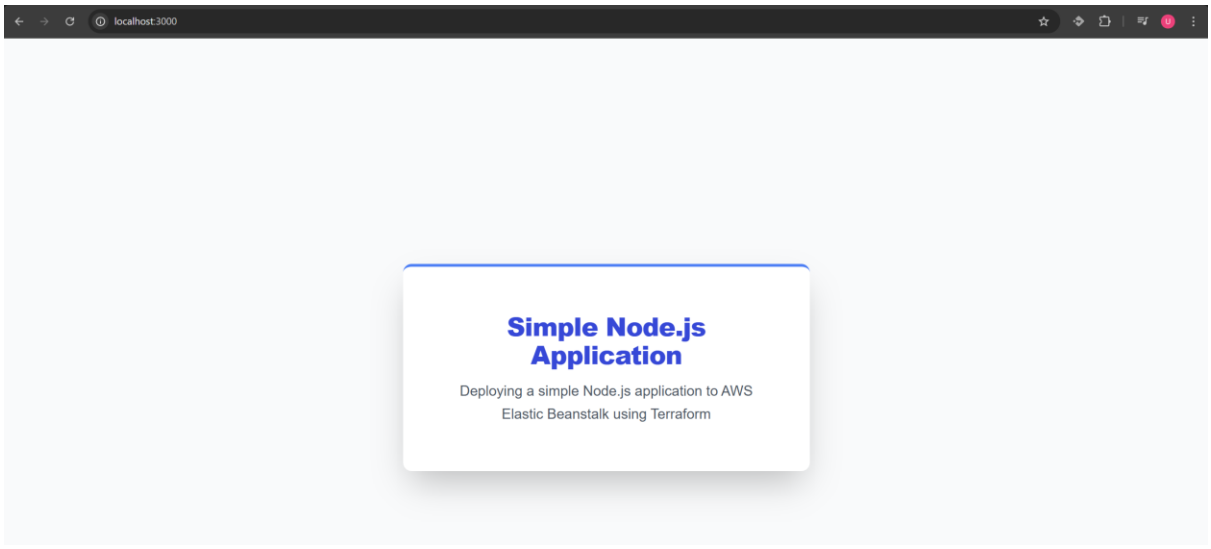
1. Install dependencies using the following command ***npm install***
   - This installs **package-lock.json** file and **node_modules** folder



2. Run the application using the command ***node app.js***
   - This runs the application on **Port 3000** as defined in the file



3. Verify the app is accessible at ***http://localhost:3000***



Task 1.2: 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-10/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-10 > terraform.tfstate

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

Task 1.3: Project Structure

The project root directory contains the following components:

- **.ebextensions/** – used by Elastic Beanstalk to configure the CloudWatch Agent on EC2 instances.
- **app.zip** – the application bundle containing app.js, package.json, public/index.html, and the .ebextensions folder.
- **terraform/** – the complete Terraform IaC setup, organized into modules (VPC, IAM, Beanstalk) and root-level configuration files.

Note: The root also contains app.js, package.json, and public/index.html separately. These are used for local testing (as mentioned in Task 1.1)

1.3.1: .ebextensions

This folder contains EB platform configuration files that Elastic Beanstalk automatically applies during deployment. This folder contains the **cloudwatch.config** file.

cloudwatch.config:

1. **Writes the CloudWatch Agent configuration** to /opt/aws/amazon-cloudwatch-agent/bin/config.json
2. **Defines memory metrics** to send to CloudWatch (namespace: CWAgent)
3. **Appends AutoScalingGroupName dimension** dynamically using Elastic Beanstalk metadata:
4. "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
5. **Starts the CloudWatch Agent** using container_commands.

This is essential for enabling **custom memory-based CloudWatch alarms** and **custom scaling** added later via Terraform.

1.3.2: *app.zip*

This application bundle ZIP file is uploaded to S3 (using terraform later) and used by the Elastic Beanstalk environment as the application version.

app.zip contains:

- **app.js** (Node.js application)
- **package.json**
- **public/index.html** (static files)
- **.ebextensions/** (CloudWatch Agent configuration)

Terraform references this ZIP in the Elastic Beanstalk module to deploy the actual application.

1.3.3: Terraform Directory Files

The Terraform configuration is structured to separate networking, IAM, and Elastic Beanstalk into independent modules, while the root directory orchestrates the entire deployment. Below is a breakdown of each root-level file and how it contributes to the overall infrastructure.

1. main.tf

main.tf

terraform > main.tf > ...

```hcl
# VPC Variables
module "vpc" {
    source       = "./modules/vpc"
    vpc_cidr     = var.vpc_cidr
    vpc_name     = var.vpc_name
    igw_name     = var.igw_name
    eip_domain   = var.eip_domain
    public_route = var.public_route
}

module "iam" {
    source           = "./modules/iam"
    eb_service_role  = var.eb_service_role
    eb_policy        = var.eb_policy
    ec2_service_role = var.ec2_service_role
    ec2_policy       = var.ec2_policy
}

module "beanstalk" {
    source               = "./modules/beanstalk"
    eb_service_role      = module.iam.eb_service_role
    ec2_instance_profile = module.iam.ec2_instance_profile
    vpc_id               = module.vpc.vpc_id
    public_subnets       = module.vpc.public_subnets
    private_subnets      = module.vpc.private_subnets
    ec2_sg_id            = module.vpc.ec2_sg_id
    bucket_name          = var.bucket_name
    application_name     = var.application_name
    environment_name     = var.environment_name
    platform             = var.platform
    volume_type          = var.volume_type
    volume_size          = var.volume_size
    monitoring_interval  = var.monitoring_interval
    alert_email          = var.alert_email
}
```

This is the entry point of the deployment. It pulls together the three modules:

- **VPC module** – receives CIDR ranges, IGW name, and routing inputs and returns VPC ID, public/private subnets, and security groups.
- **IAM module** – creates the Elastic Beanstalk service role and EC2 instance profile using the JSON permission files.
- **Elastic Beanstalk module** – builds the EB application and environment and depends on outputs from both the VPC and IAM modules (VPC ID, subnets, SG, IAM roles, etc.).

All dependencies are wired automatically because module outputs feed directly into other modules.

2. terraform.tf

Defines the AWS provider and configures the **S3 backend**:

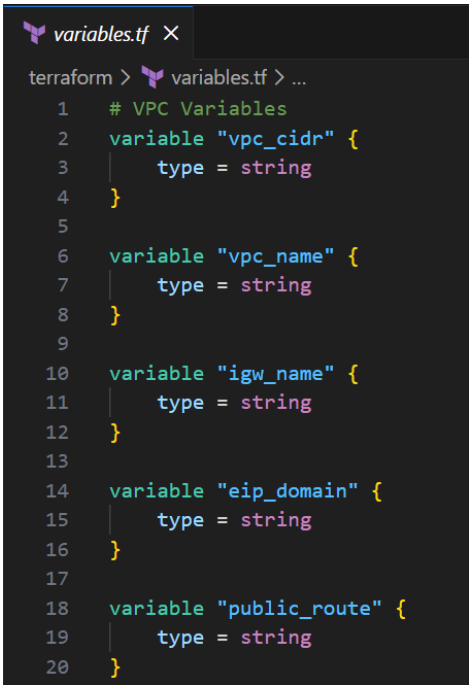
- Stores the Terraform state file in:
  - ***umarsatti-terraform-state-file-s3-bucket-sandbox/Task-10/terraform.tfstate***
- Enables encryption and locking to prevent concurrent state updates.

This ensures centralized, versioned, and secure state management.

3. variables.tf

Declares all input variables used across the modules:

- **VPC variables:** CIDR, IGW name, routing details.



- **IAM variables:** names for IAM roles & policies referenced inside the IAM module.

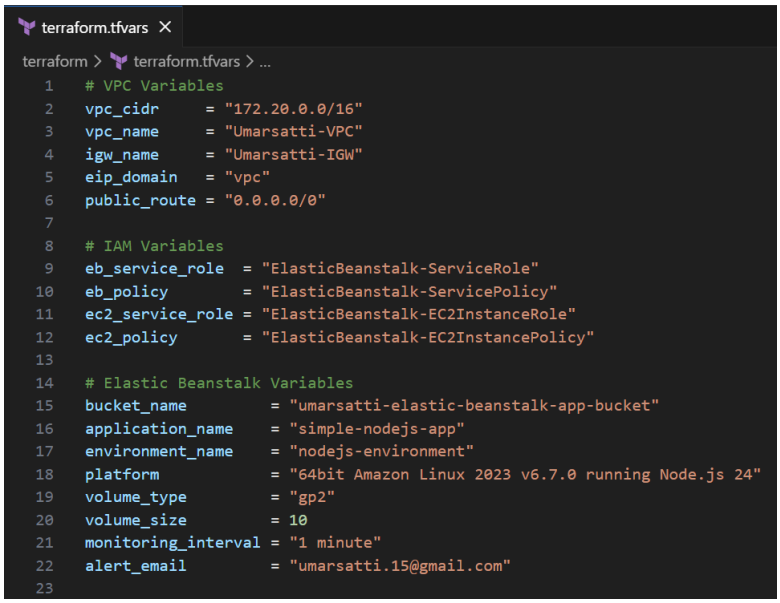
```hcl
# IAM Variables
variable "eb_service_role" {
    type = string
}

variable "eb_policy" {
    type = string
}

variable "ec2_service_role" {
    type = string
}

variable "ec2_policy" {
    type = string
}
```

- **Elastic Beanstalk variables:** bucket name, application name, environment name, platform version, EBS volume type/size, monitoring interval, CloudWatch alert email.

```hcl
# Elastic Beanstalk Variables
variable "bucket_name" {
    type = string
}

variable "application_name" {
    type = string
}

variable "environment_name" {
    type = string
}

variable "platform" {
    type = string
}

variable "volume_type" {
    type = string
}

variable "volume_size" {
    type = number
}

variable "monitoring_interval" {
    type = string
}

variable "alert_email" {
    type = string
}
```

This file defines *what can be configured* without modifying code.

4. terraform.tfvars

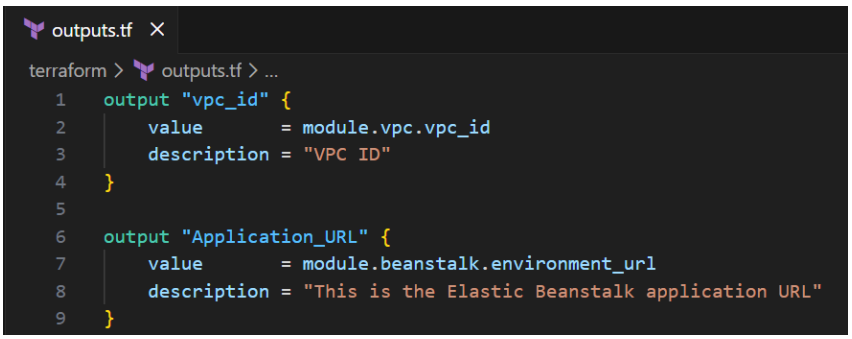


Populates the variables with actual values for this deployment:

- Full VPC configuration
- Role and policy names
- Node.js platform version
- Beanstalk environment name
- EBS instance volume configuration
- CloudWatch monitoring interval
- App deployment S3 bucket

This keeps the configuration clean and environment specific.

5. outputs.tf



Exposes key outputs:

- **vpc_id** – useful for validation and future integrations
- **Application_URL** – the Beanstalk public endpoint generated after deployment

These outputs help verify successful provisioning immediately after **terraform apply**.

6. eb-permissions.json and instance-permissions.json

These JSON files define:

- **ElasticBeanstalk service role permissions**
- **EC2 instance profile permissions**

The IAM module reads these files and attaches their policies to the appropriate roles, keeping permission logic clean and externalized.

1.3.2 VPC Module

The **VPC module** provisions the full networking layer for the application—including subnets, routing, NAT, and security groups used by the ALB and EC2/ECS instances.

1. main.tf

Defines all networking resources:

- **VPC**: Creates a VPC with DNS support/hostnames.
- **Subnets**
  - Two **public** subnets (A/B) mapped to specific AZs
  - Two **private** subnets (A/B) mapped to the same AZs
  - Subnet definitions driven by locals maps.
- **Internet Gateway**: Enables outbound internet access for public subnets.
- **NAT Gateways and Elastic IPs**
  - One NAT per public subnet.
  - Private subnets route outbound internet traffic through their corresponding NAT.
- **Route Tables & Associations**
  - Public RTs route 0.0.0.0/0 → IGW
  - Private RTs route 0.0.0.0/0 → NAT
  - Each subnet is associated with its own RT.
- **Security Groups**
  - **ALB-SG**: Allows inbound HTTP (80) from anywhere; full outbound.
  - **EC2-SG**: Allows inbound port **3000** only from ALB-SG; full outbound.

This file builds the entire networking topology.

variables.tf

Defines required input variables for flexibility:

- **vpc_cidr** – CIDR of the VPC
- **vpc_name** – Name tag

- **igw_name** – Internet Gateway name
- **eip_domain** – EIP domain ("vpc")
- **public_route** – 0.0.0.0/0

outputs.tf

Exposes key IDs for other modules:

- vpc_id
- public_subnets[]
- private_subnets[]
- alb_sg_id
- ec2_sg_id

These outputs are consumed by other modules for reference or root directory outputs.tf.

1.3.3: IAM Module

This module provisions all IAM roles and policies required for Elastic Beanstalk and the EC2 instances it launches.

main.tf

Creates three core IAM components:

1. Elastic Beanstalk Service Role

Used by Beanstalk itself to manage deployments and AWS resources.

- **Assume-role policy** allowing elasticbeanstalk.amazonaws.com
- **IAM role** (eb_service_role)
- **Custom policy** loaded from eb-permissions.json
- **Policy attachment** to the role

2. Elastic Beanstalk EC2 Instance Role

Assigned to EC2 instances created by Beanstalk.

- **Assume-role policy** allowing ec2.amazonaws.com
- **IAM role** (eb_ec2_role)
- **Custom policy** loaded from instance-permissions.json
- **Policy attachment** to the role

3. IAM Instance Profile

Required for EC2 to use the IAM role.

- Instance profile using **eb_ec2_role**

variables.tf

Defines the input values for IAM role and policy names:

- eb_service_role
- eb_policy
- ec2_service_role
- ec2_policy

outputs.tf

Exposes key IAM identifiers to the Beanstalk module:

- **eb_service_role** – Name of the Beanstalk service role
- **ec2_instance_profile** – Instance profile assigned to EC2 instances

1.3.4: Elastic Beanstalk Module

This module packages the application, uploads it to S3, and provisions the full Elastic Beanstalk application and environment, including scaling policies, monitoring, networking, and custom alarms.

main.tf

1. S3 for Application Storage

- Creates an S3 bucket for Elastic Beanstalk application versions.
- Uploads **app.zip** (in project root directory) to **beanstalk/app.zip** (in S3).
- Used by the application version resource.

2. Elastic Beanstalk Components

- **Application** – Logical container for all versions.
- **Application version** – Points to the uploaded ZIP in S3.
- **Environment** – Full EB environment (compute, load balancer, and scaling).

The environment includes:

- **Service access**
    - Sets the Beanstalk **service role** and **EC2 instance profile** from IAM module.
- **Networking**
    - Uses the shared VPC.
    - Places instances in **private subnets** and ALB in **public subnets**.
    - Disables public IP assignment.
- **Instance configuration**
    - Root volume type/size

    - IMDSv1 disabled
    - Enhanced monitoring interval
    - EC2 security group from VPC module
- **Scaling & capacity**
    - Load-balanced environment
    - Min/max size (1–3)
    - Instance types (t3.micro/small)
    - CPU-based auto scaling thresholds
- **Load Balancer**
    - Uses ALB
    - Enables default listener
    - Configures LB subnets
- **Logging & health**
    - Enhanced health monitoring
    - CloudWatch Logs streaming and cleanup
- **Notifications**
    - Email alerts via Elastic Beanstalk SNS settings.
- **Deployment**
    - All-at-once deployment policy.

3. Additional Custom Monitoring

Adds resources outside EB to enhance visibility and scaling:

- **SNS topic and subscription –** custom alerts sent to email
- **Auto Scaling policy** – scale out EC2 instance by 1
- **CloudWatch Alarm** (memory > 60%)
    - Sends SNS alert
    - Triggers scale-up policy

variables.tf

Defines inputs for:

- Application names
- Platform
- Storage & monitoring settings
- IAM role references
- VPC subnet IDs & security groups

outputs.tf

- Exposes the **environment URL** for easy access after deployment.
- Directly access the website using this URL instead of searching through the console.

Task 1.4: Execute Terraform Commands

This task deploys the entire **Elastic Beanstalk Node.js environment** using Terraform. Make sure you are inside the **root Terraform 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: 38 added, 0 changed, 0 destroyed.

Outputs:

Application_URL = "awseb--AWSEB-XtYzmaFBvjzK-925857975.us-west-1.elb.amazonaws.com"
vpc_id = "vpc-0054e10bfed067aed"
PS D:\Cloudelligent\Task-10\terraform>
```

- VPC, public/private subnets, route tables, IGW, NAT gateways
- Security groups (ALB-SG, EC2-SG)
- IAM roles, policies, and EC2 instance profile
- S3 bucket and app.zip upload
- Elastic Beanstalk application, application version, and environment
- ALB creation and attachment to public subnets
- Auto scaling configuration
- SNS topic and email subscription
- CloudWatch alarms and scale-up policy

This completes the infrastructure deployment for the Node.js Elastic Beanstalk application.

Task 1.5: Validate Infrastructure in AWS Console

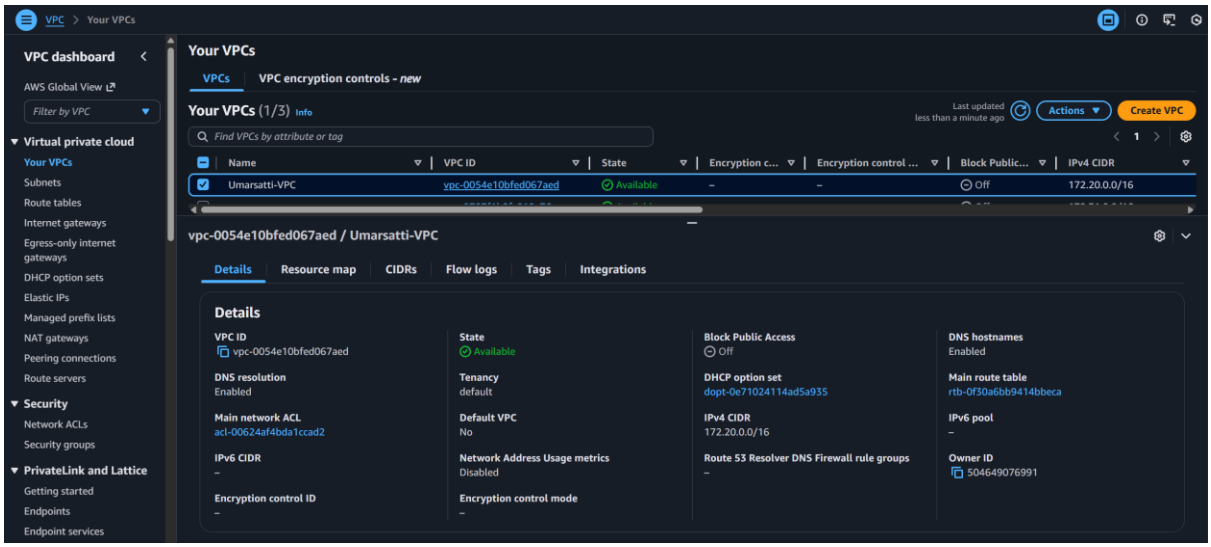
After running **terraform apply**, the next step is to verify that all AWS resources for the **Elastic Beanstalk Node.js environment** were successfully provisioned. This task walks through the AWS Console to confirm networking, IAM, compute, storage, and monitoring components.

Note: *Screenshots have been attached as evidence.*

1.5.1 VPC and Networking Validation

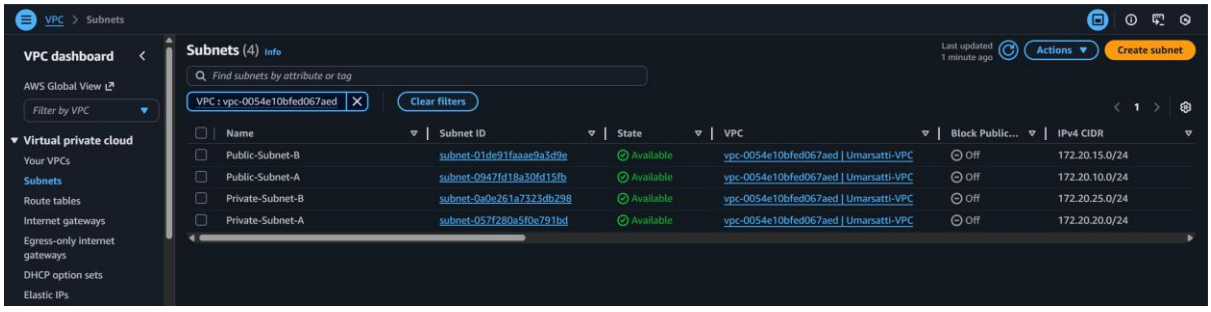
1. Verify VPC

- 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 the Application load balancer)*

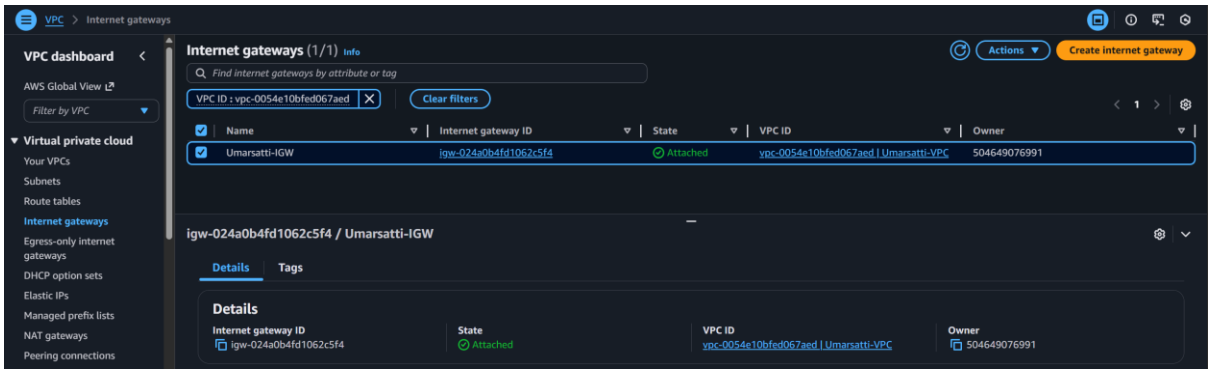
- Public-Subnet-A (us-west-1a)
- Public-Subnet-B (us-west-1b)

Private Subnets (*Used by the Elastic Beanstalk EC2 instances)*

- Private-Subnet-A (us-west-1a)
- Private-Subnet-B (us-west-1b)

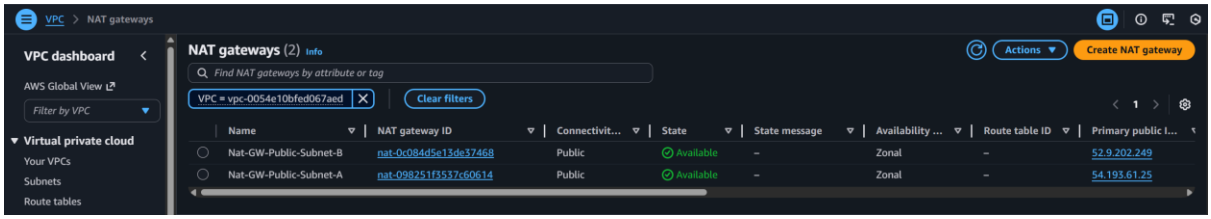
3. Internet Gateway and NAT Gateways

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

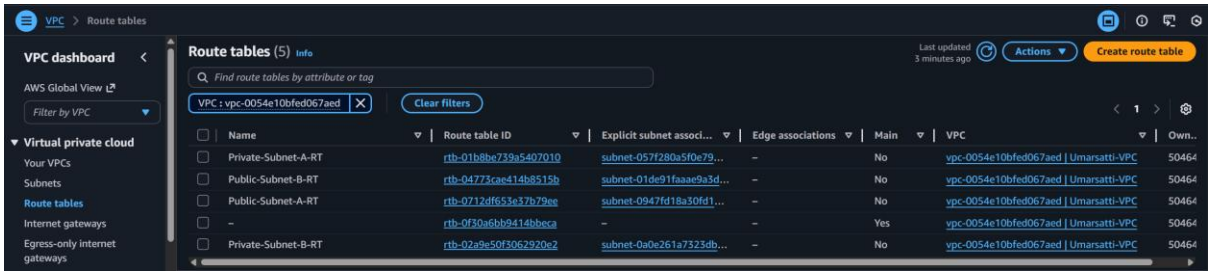
NAT Gateways



- 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**
    - EIP allocated automatically

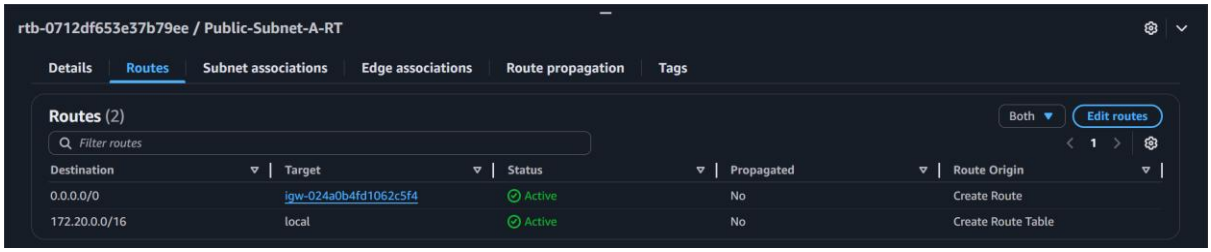
4. Route Tables

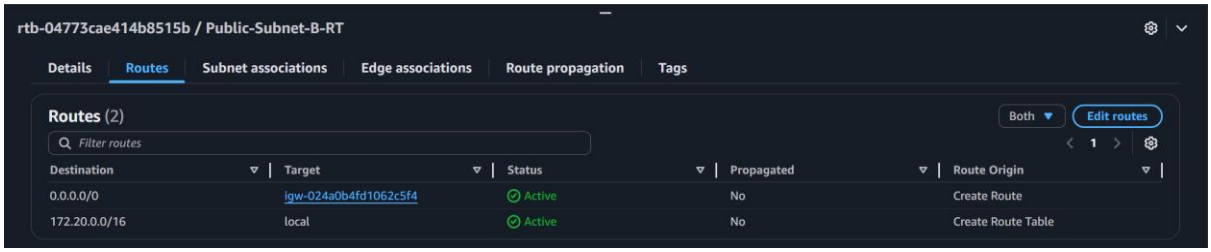
- Navigate to **Route tables** section located in the left navigation bar.
- Confirm that both public and private route tables have been created.



Public Route Tables

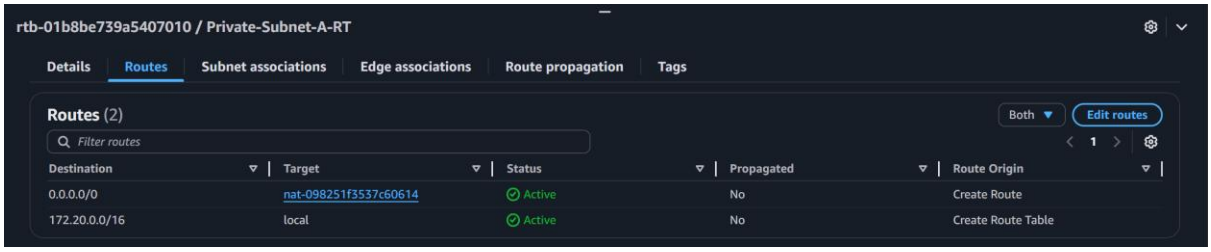
- Associated with public subnets
- Should contain route 0.0.0.0/0 → Internet Gateway

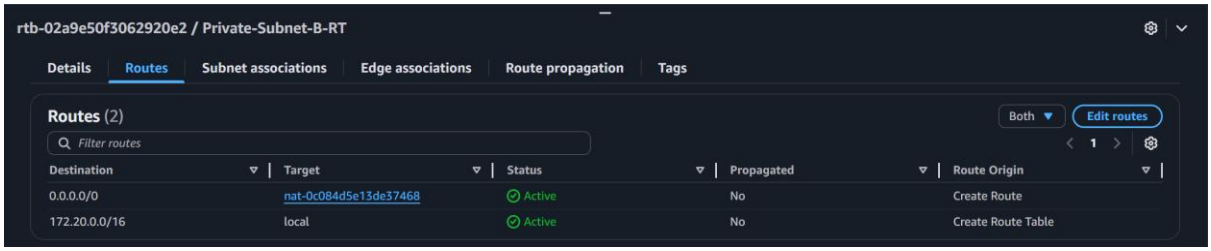




Private Route Tables

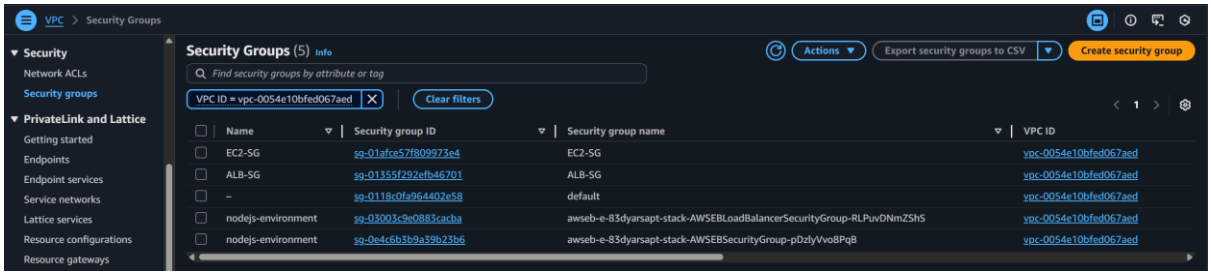
- Associated with private subnets
- Should contain route 0.0.0.0/0 → NAT Gateway





1.5 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.

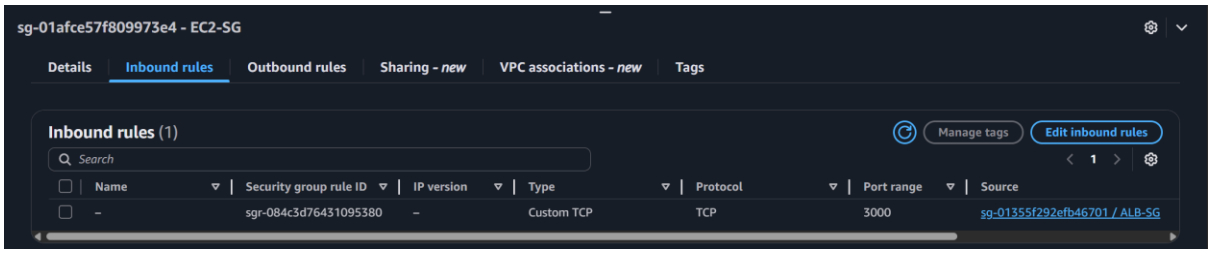


ALB-SG

- 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



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

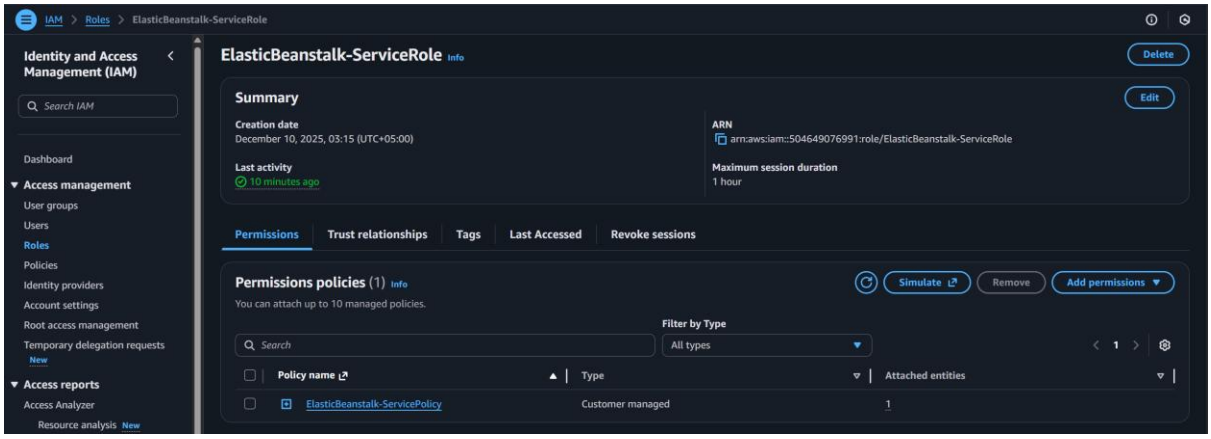
1.5.2 IAM Validation

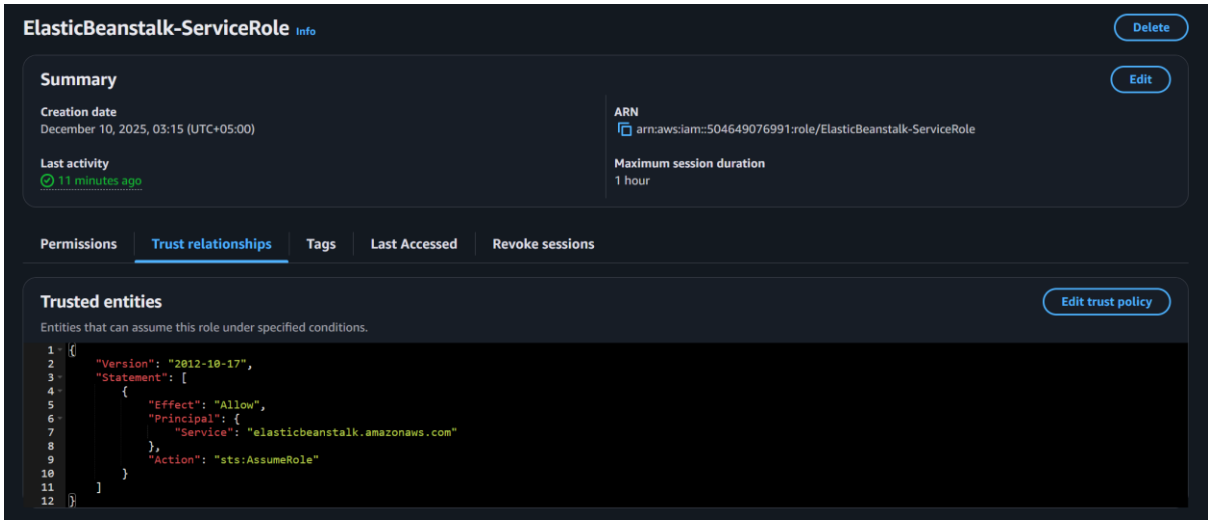
1. Elastic Beanstalk Service Role

- 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*





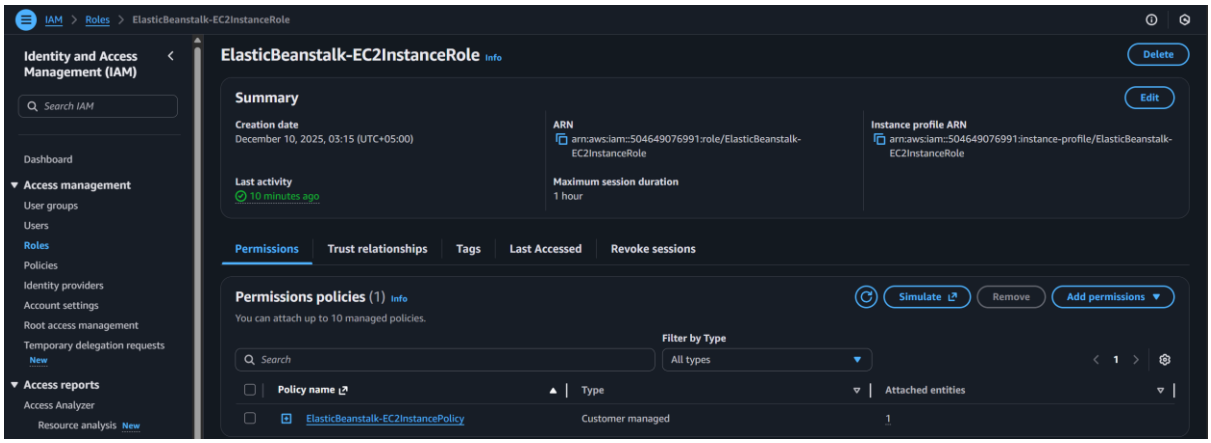
2. EC2 Instance Profile

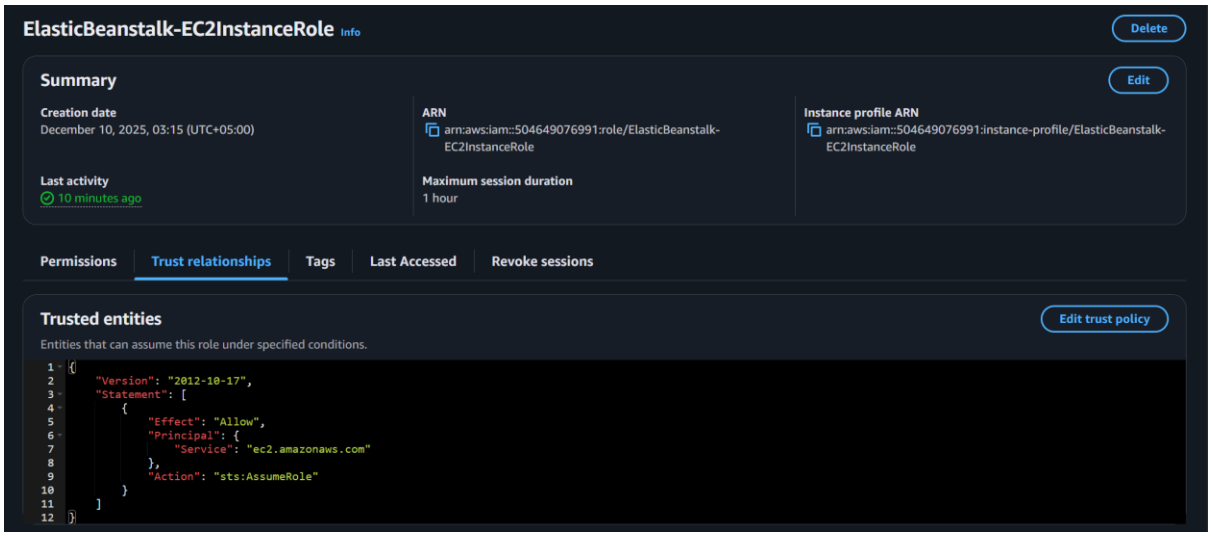
- Navigate to the **IAM Console** using the search bar.
- Search for the Elastic Beanstalk service role.

Check:

- IAM Role: ElasticBeanstalk-EC2InstanceRole

- Instance Profile created
- Attached policy from instance-permissions.json
    - *ElasticBeanstalk-EC2InstancePolicy*

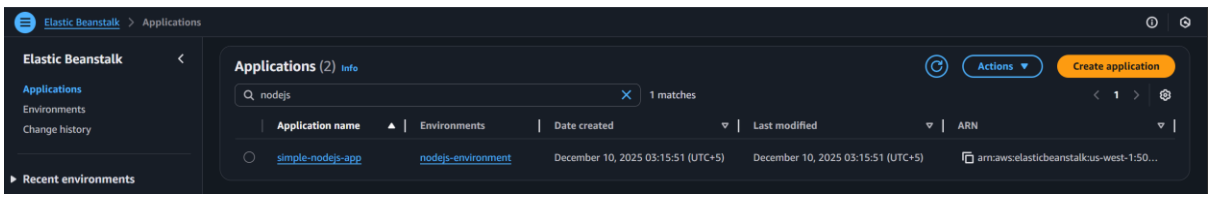




1.5.3: Elastic Beanstalk Validation

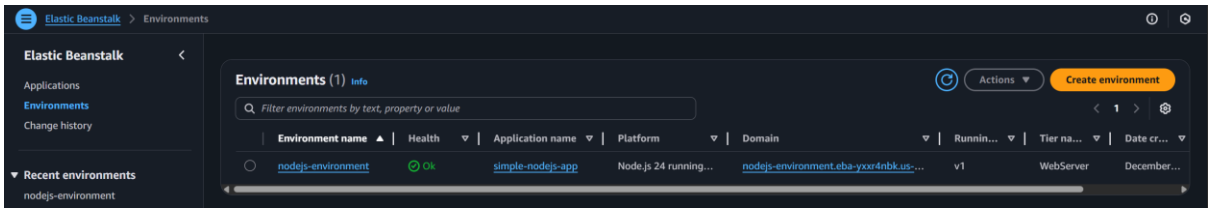
1. Application

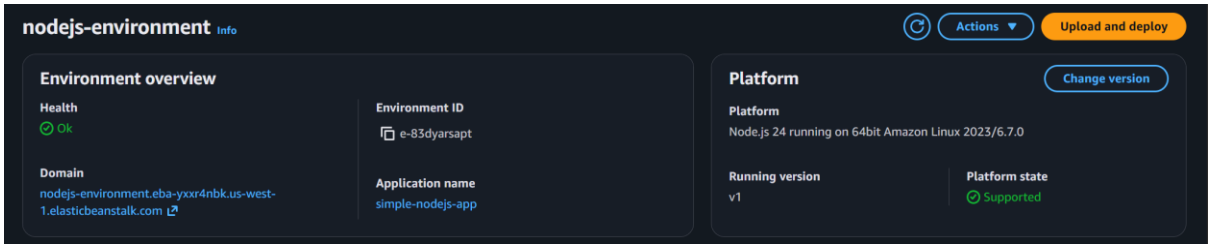
- Navigate to **Elastic Beanstalk** console using the search bar.
- Click **Application** option located in the left navigation bar.
- Verify the creation of the application



2. Environment

- Within the **Elastic Beanstalk**, click the Environments section located in the left navigation bar.
- Verify the creation of:
    - Environment name: nodejs-environment
    - Health Status: OK
    - Valid Domain: nodejs-environment.…

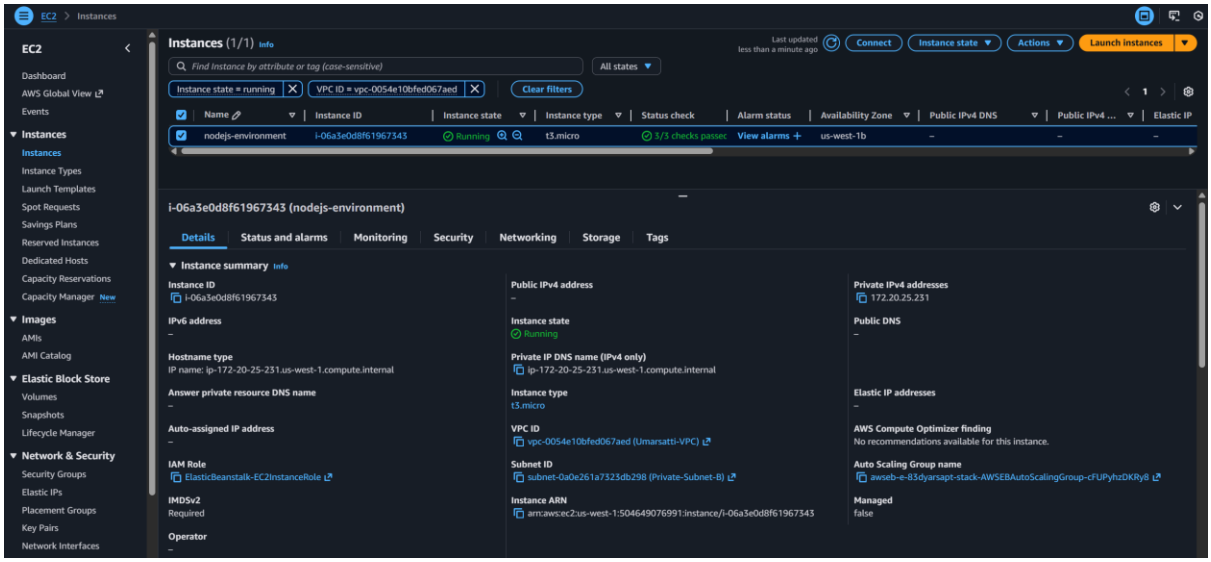




1.5.4: EC2, ALB, Target groups, and ASG Validation

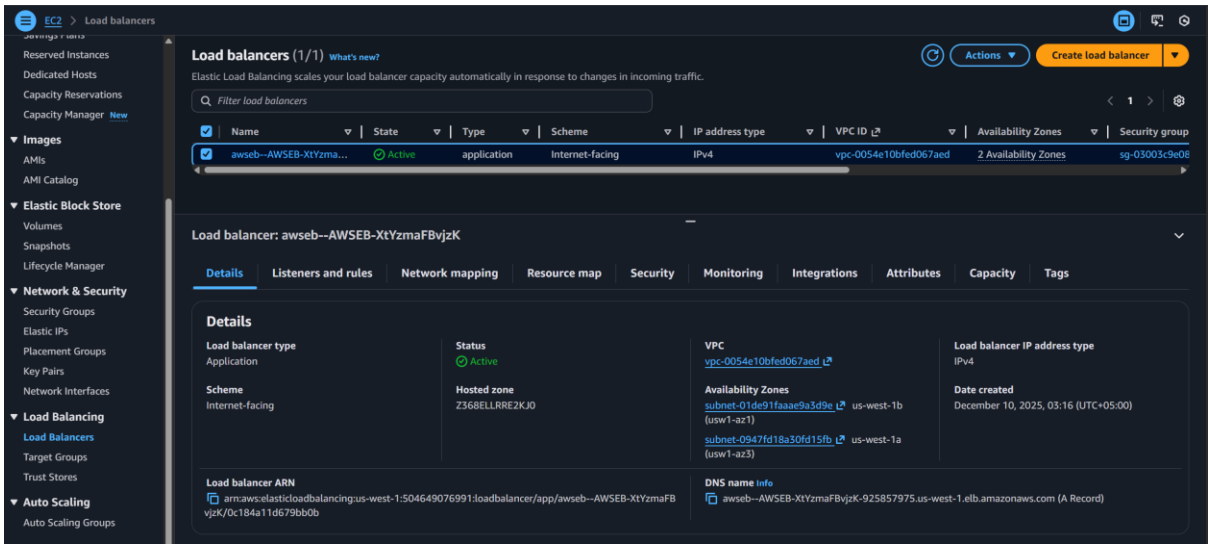
1. EC2 Instances

- 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.micro



2. Load Balancer

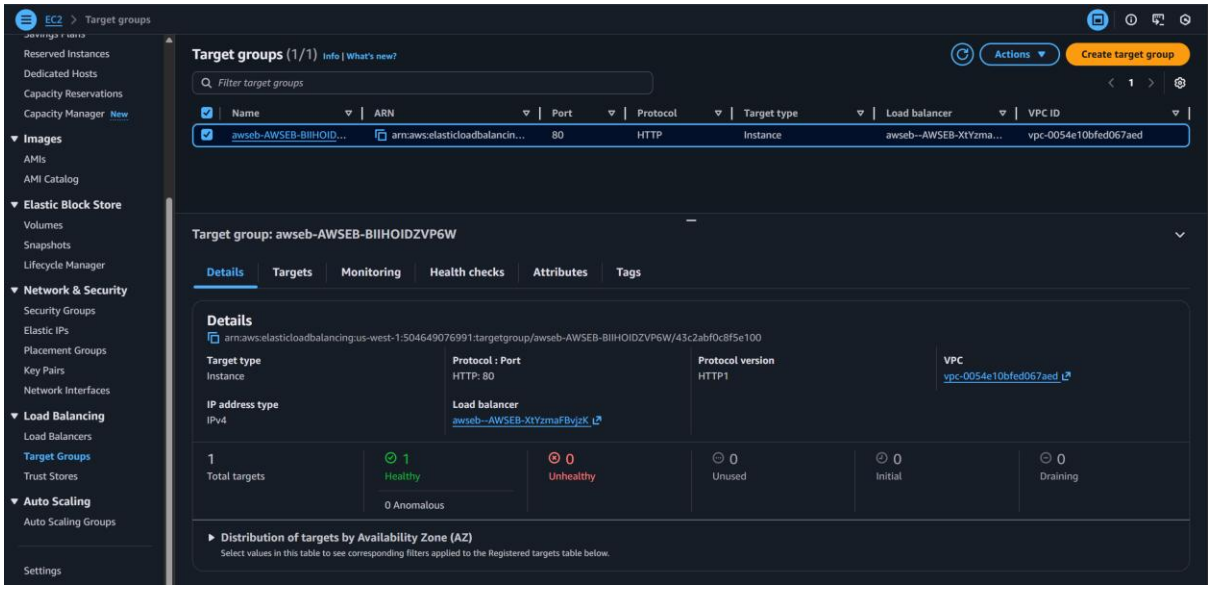
- Within the EC2 console, navigate to the **Load Balancer** section.
- Click **Load balancers** option located in the left navigation bar.
- Verify the creation of:
    - ALB created
    - Assigned to both public subnets
    - Listener: HTTP:80 → default target group
    - Security group: ALB-SG

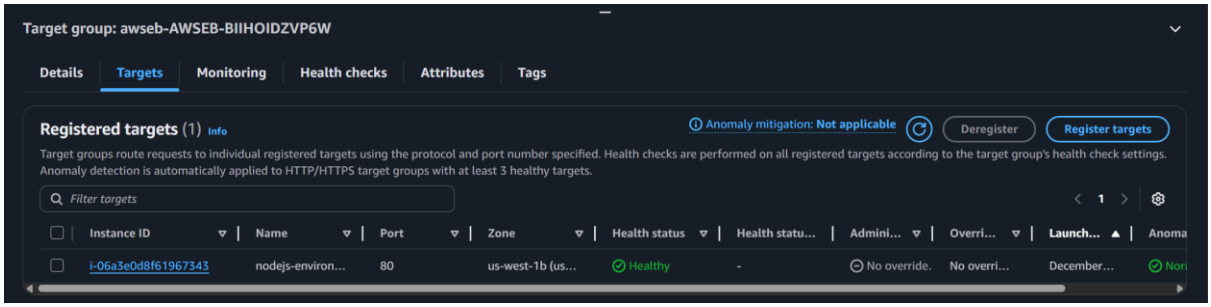


3. Target Group

- 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 EB auto configuration)

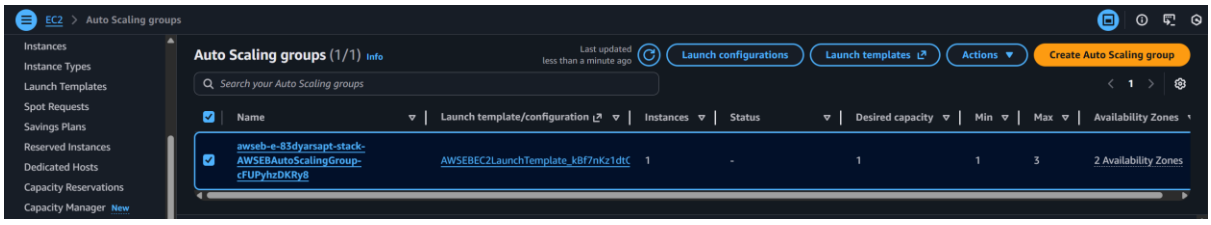
- Target type: Instance
- Attached to load balancer and VPC

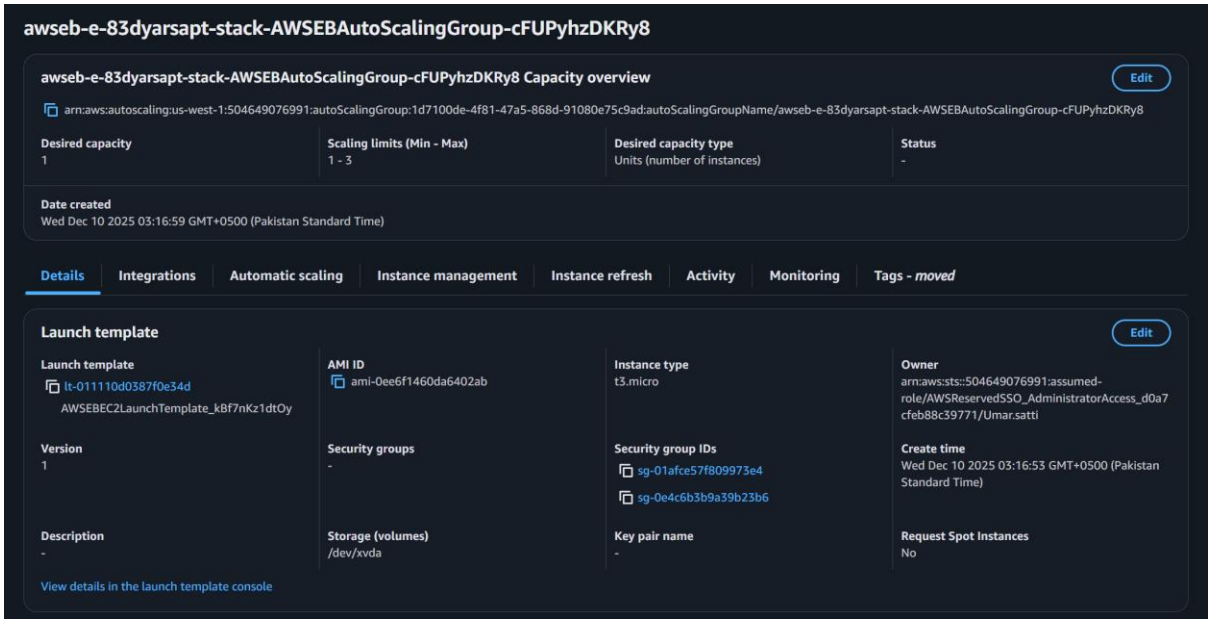




4. Auto Scaling Group

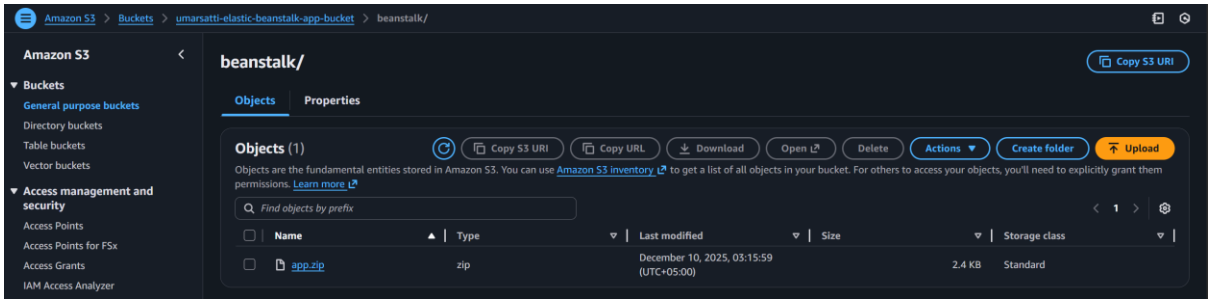
- Within the EC2 console, navigate to the **Target groups** section.
- Click **Target groups** option located in the left navigation bar.
- Verify the creation of:
  - ASG created by Elastic Beanstalk
  - Minimum instances = 1, Maximum instance = 3
  - Linked to Launch Template





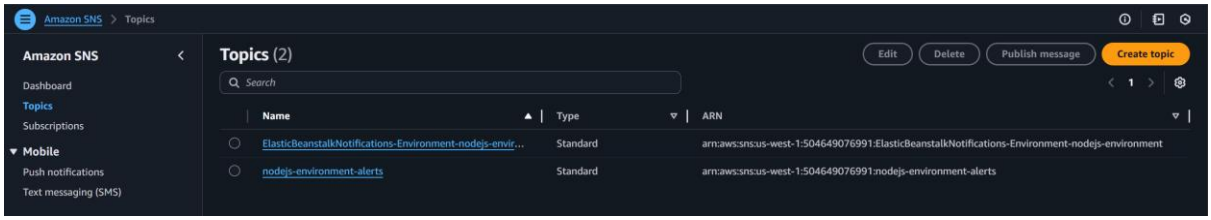
1.5.5: S3 Bucket Validation

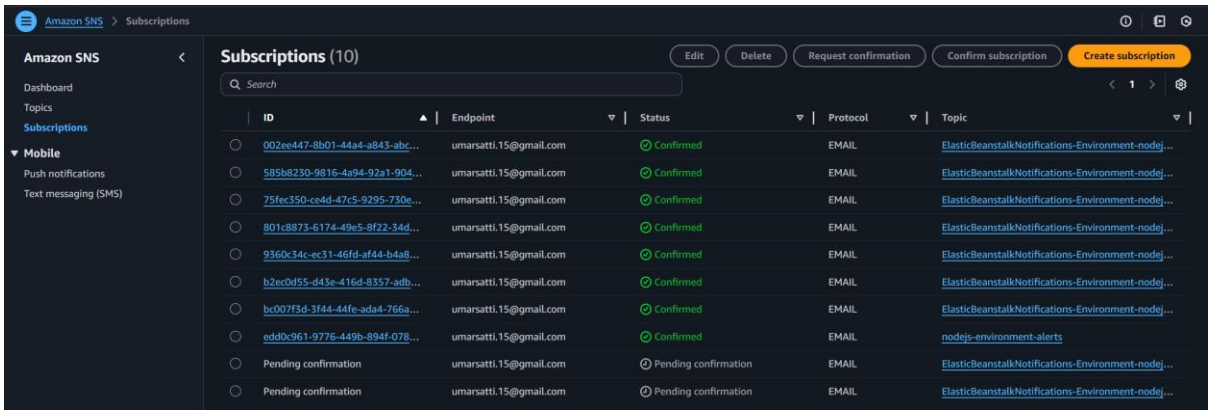
- Navigate to the **S3 console** using the search bar.
- Search for the bucket name (*umarsatti-elastic-beanstalk-app-bucket* in this case)
- Verify the creation of
    - Bucket: umarsatti-elastic-beanstalk-app-bucket
    - File uploaded: beanstalk/app.zip
    - Optionally download the zip file and verify the application code exists



1.5.6: 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)

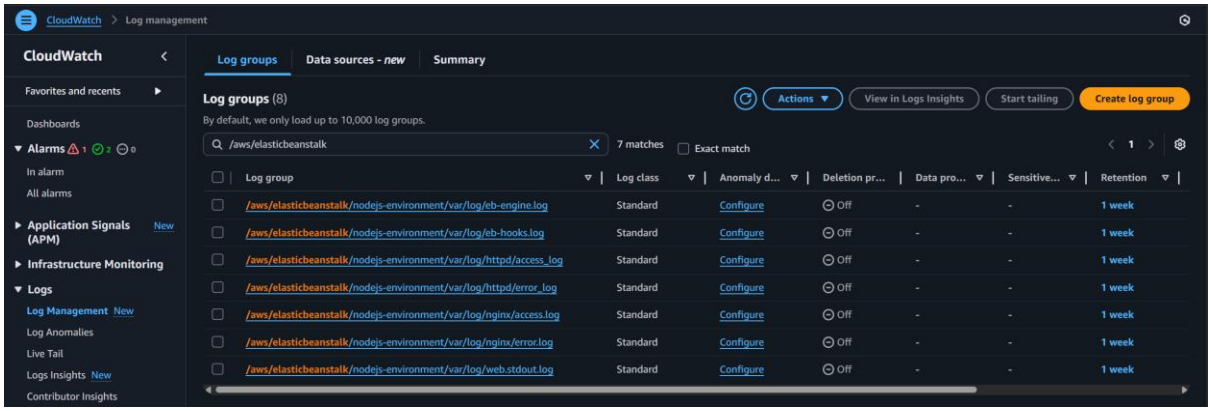




1.5.7: CloudWatch Validation

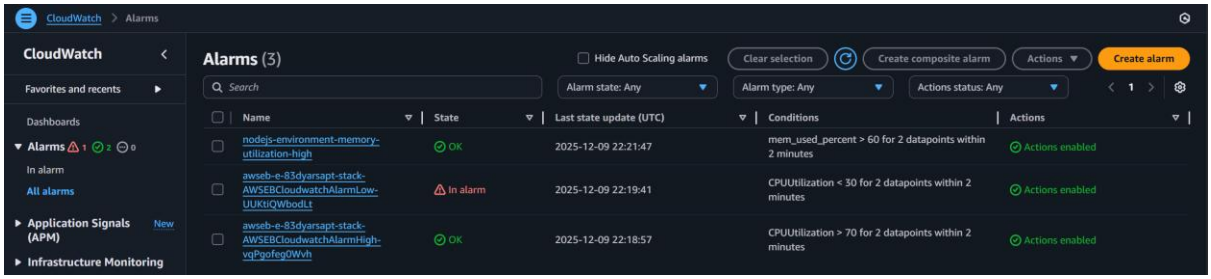
1. Log Groups

- 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/*beanstalk* in the search bar
- Verify the creation of these logs



2. CloudWatch Alarms

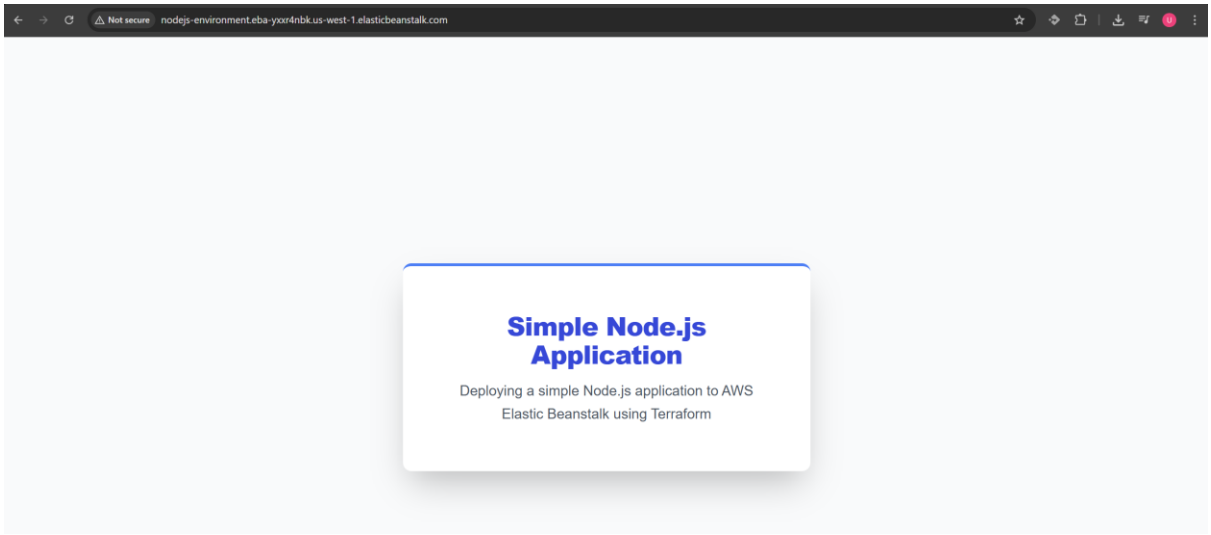
- Within the **CloudWatch console**, locate the **Alarms** section.
- Under **Alarms** section, select All alarms to view the **CPU** and **Memory** alarms.
- Verify creation of:
    - Alarms: 1 nodejs-memory-utilization-high / 2 CPU alarms
    - Status: OK / In ALARM


CloudWatch
Alarms
CloudWatch
Favorites and recents
Dashboards
Alarms 1 2 0
In alarm
All alarms
Application Signals (APM)
New
Infrastructure Monitoring
Alarms (3)
Hide Auto Scaling alarms
Clear selection
Create composite alarm
Actions
Create alarm
Search
Alarm state: Any
Alarm type: Any
Actions status: Any
1
Name
State
Last state update (UTC)
Conditions
Actions
nodejs-environment-memory-utilization-high
OK
2025-12-09 22:21:47
mem_used_percent > 60 for 2 datapoints within 2 minutes
Actions enabled
awseb-e-83dyarsapt-stack-AWSEBCloudwatchAlarmLow-UUKtiQWbodLt
In alarm
2025-12-09 22:19:41
CPUUtilization < 30 for 2 datapoints within 2 minutes
Actions enabled
awseb-e-83dyarsapt-stack-AWSEBCloudwatchAlarmHigh-vqPgofeg0Wvh
OK
2025-12-09 22:18:57
CPUUtilization > 70 for 2 datapoints within 2 minutes
Actions enabled

Task 1.6: Application Testing, Alarm Verification, and Auto Scaling Validation

1.6.1: 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 above displays the Node.js application load successfully.
- No client-side (4xx) or server-side (5xx) errors.



URL: ***http://nodejs-environment.eba-yxxr4nbk.us-west-1.elasticbeanstalk.com/***

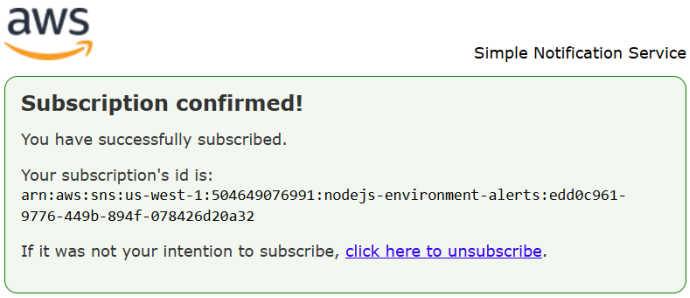
1.6.2: Validate Email Subscriptions for CloudWatch Alarms

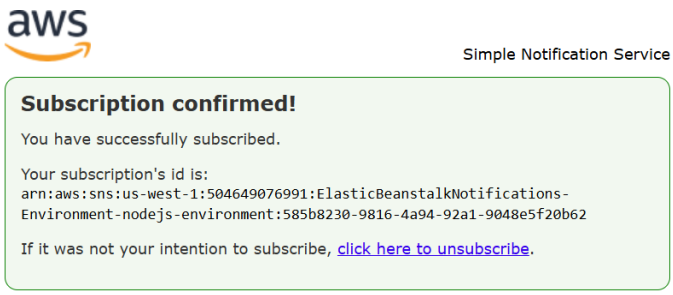
Two SNS topics were created for:

- CPU Utilization alarm notifications
- Memory Utilization alarm notifications

Steps:

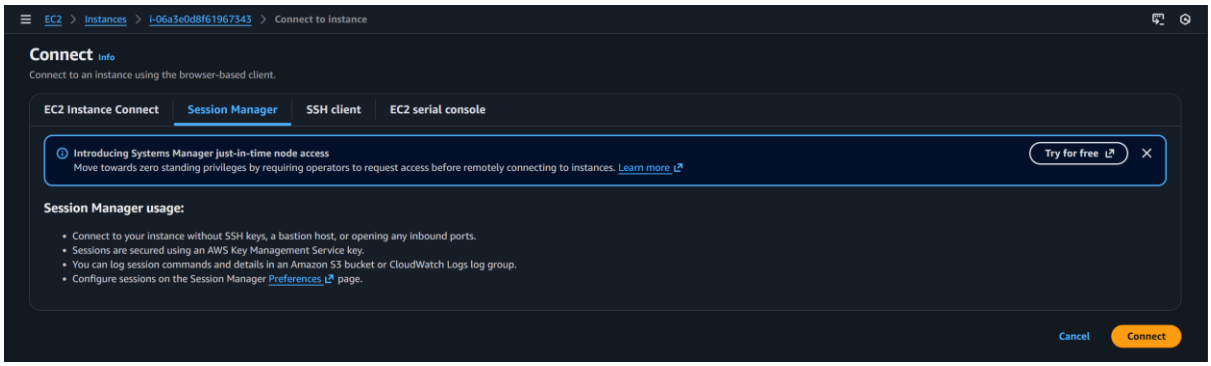
1. Check email inbox.
2. Confirm the two received **subscription confirmation emails**.
3. Accept both subscriptions.





1.6.3: 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.



1.6.4: Verify CloudWatch Agent on the 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": "2025-12-09T22:19:36+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***

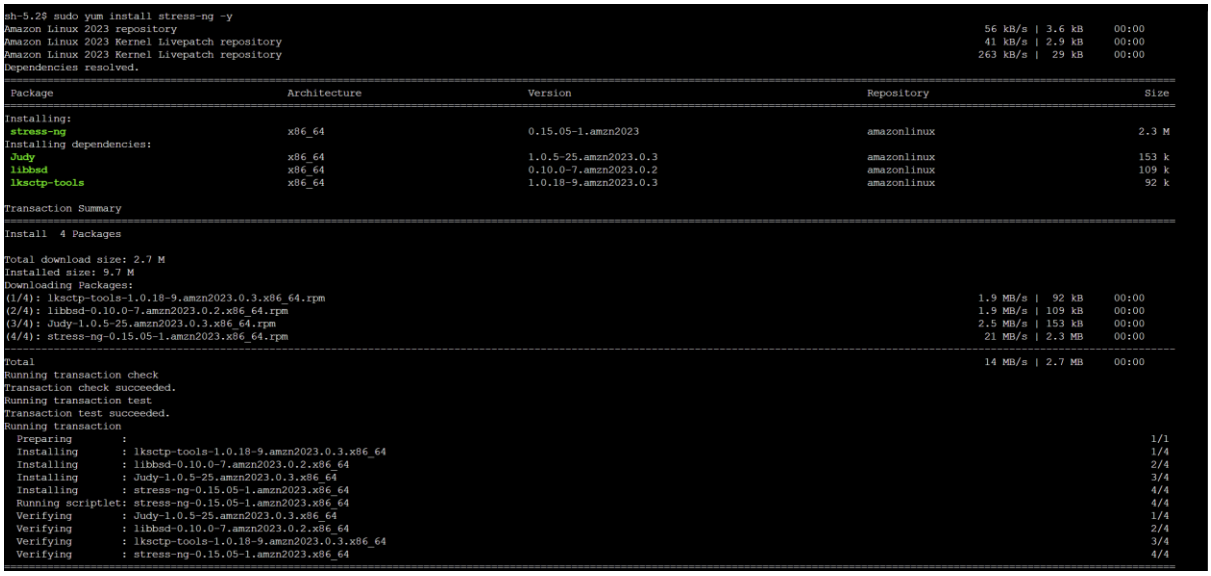
```
sh-5.2$ sudo cat /opt/aws/amazon-cloudwatch-agent/bin/config.json
{
  "agent": {
    "metrics_collection_interval": 60,
    "run_as_user": "root"
  },
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "metrics_collected":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

1.6.5: 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 su -
[root@ip-172-20-25-231 ~]# stress-ng --cpu 0 --cpu-load 90 --timeout 300
stress-ng: info:  [28777] setting to a 300 second (5 mins, 0.00 secs) run per stressor
stress-ng: info:  [28777] dispatching hogs: 2 cpu
stress-ng: info:  [28777] successful run completed in 300.01s (5 mins, 0.01 secs)
[root@ip-172-20-25-231 ~]#
```

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 700M --vm-keep --vm-method all --vm-hang 180 -t 180s*

Expected output:

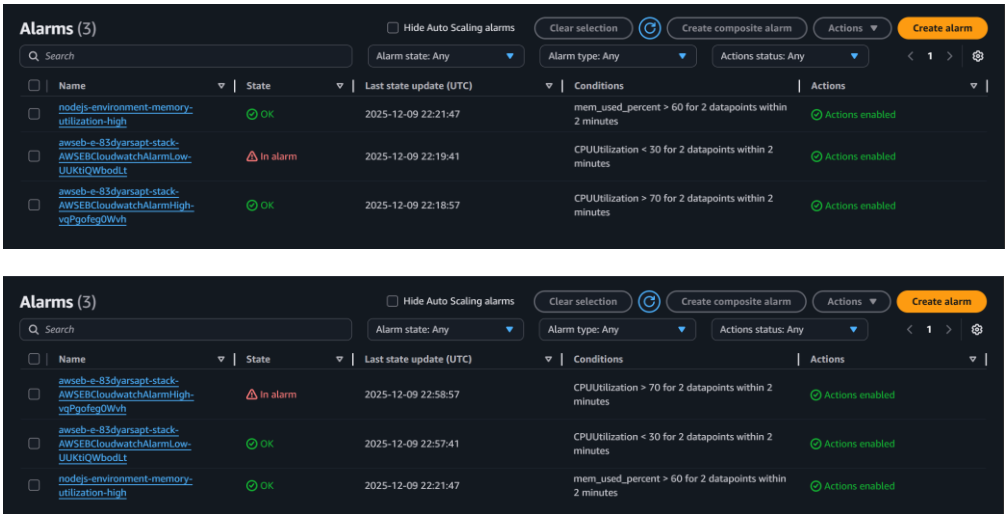
- Memory alarm should go **OK → ALARM**.
- ASG may launch additional EC2 capacity depending on scaling policies.

1.6.6: Verify Alarm State Change in CloudWatch

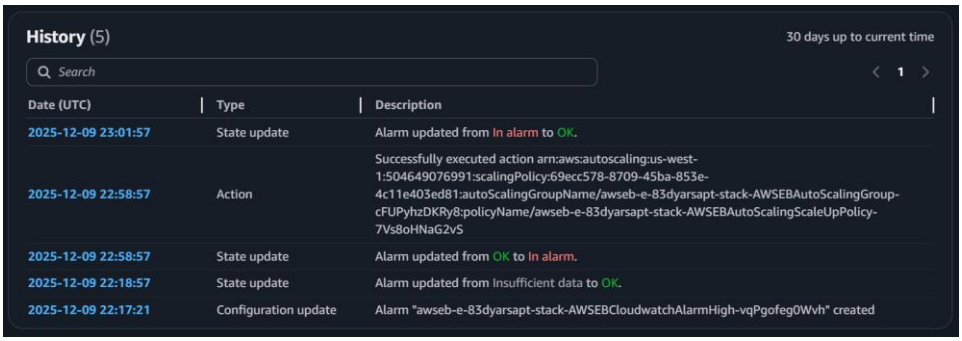
- Navigate to the **CloudWatch console**:
- Select the **All alarms** option under the **Alarms** section in the left navigation bar.
- Verify the state changes for both the CPU and Memory alarms.

CPUUtilization Alarm

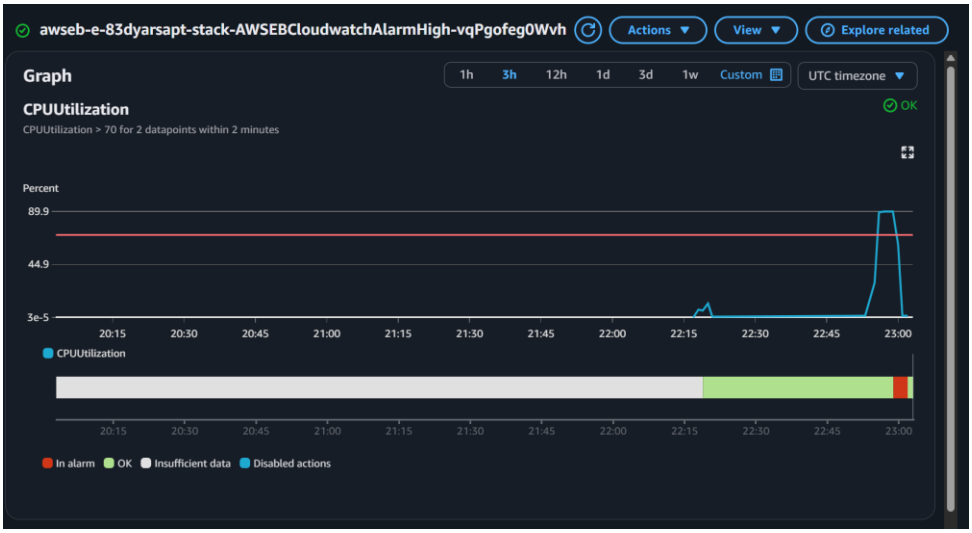
1. Transitions from **OK** to **ALARM**.



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

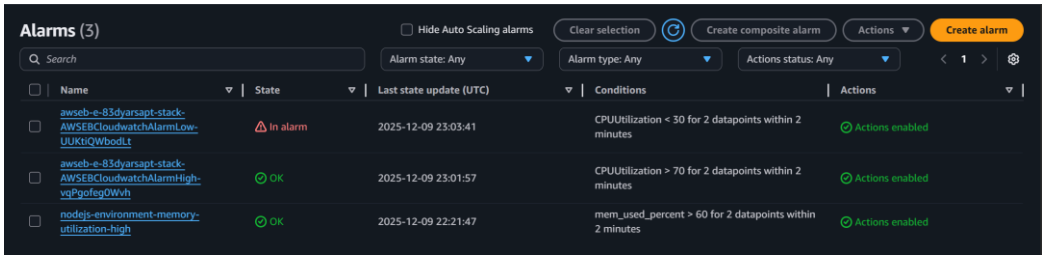


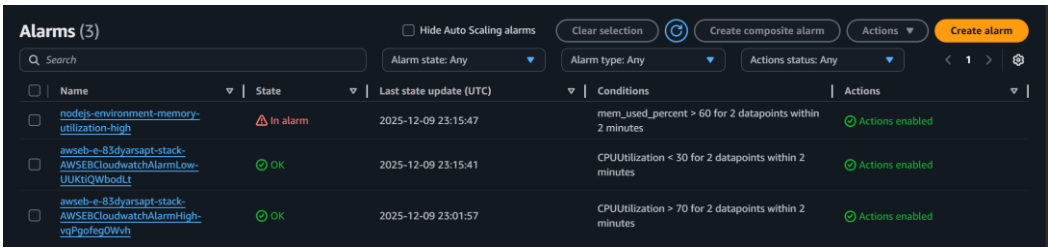
3. **CPUUtilization** graph showing CPU usage going up to 90%



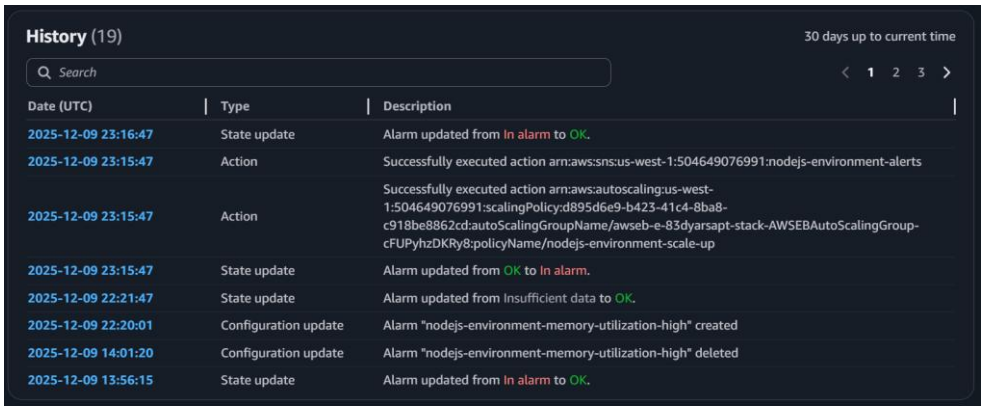
MemoryUtilization Alarm

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

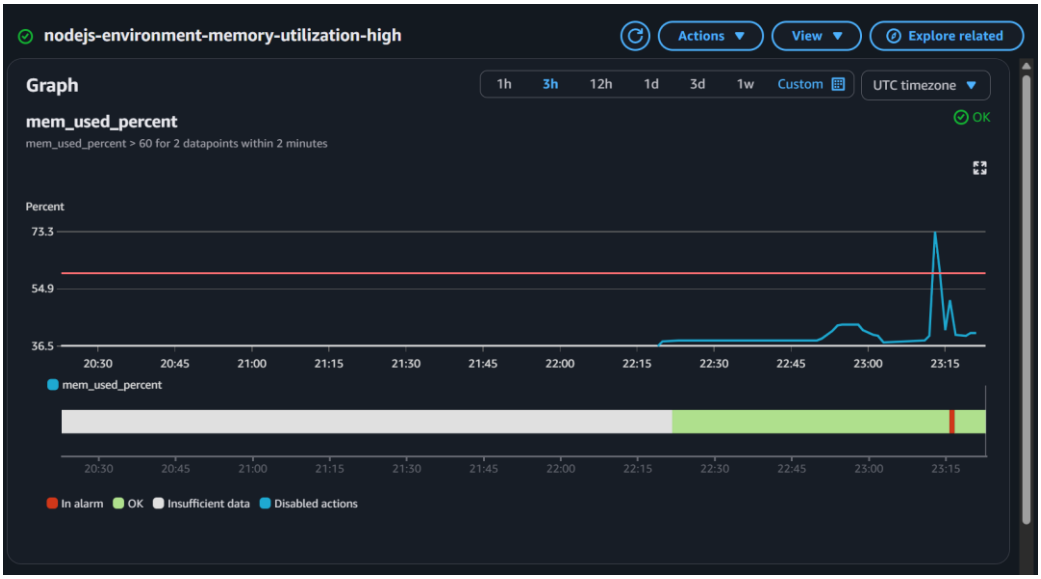




2. **Memory Utilization > 60** Alarm history shows the state change.

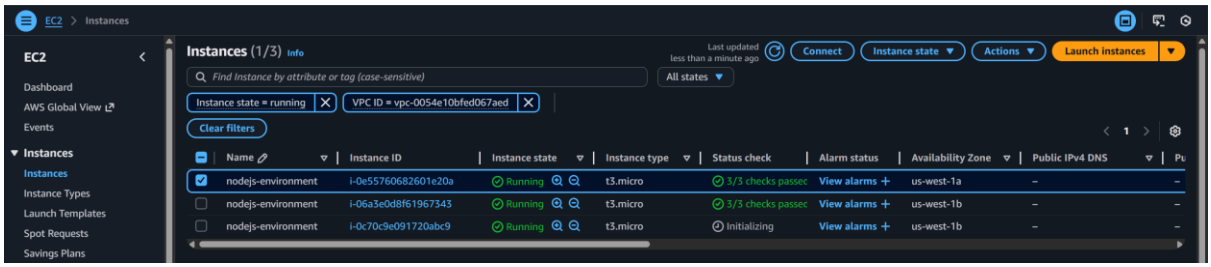


3. **Memory Utilization** graph showing CPU usage going up to 73%

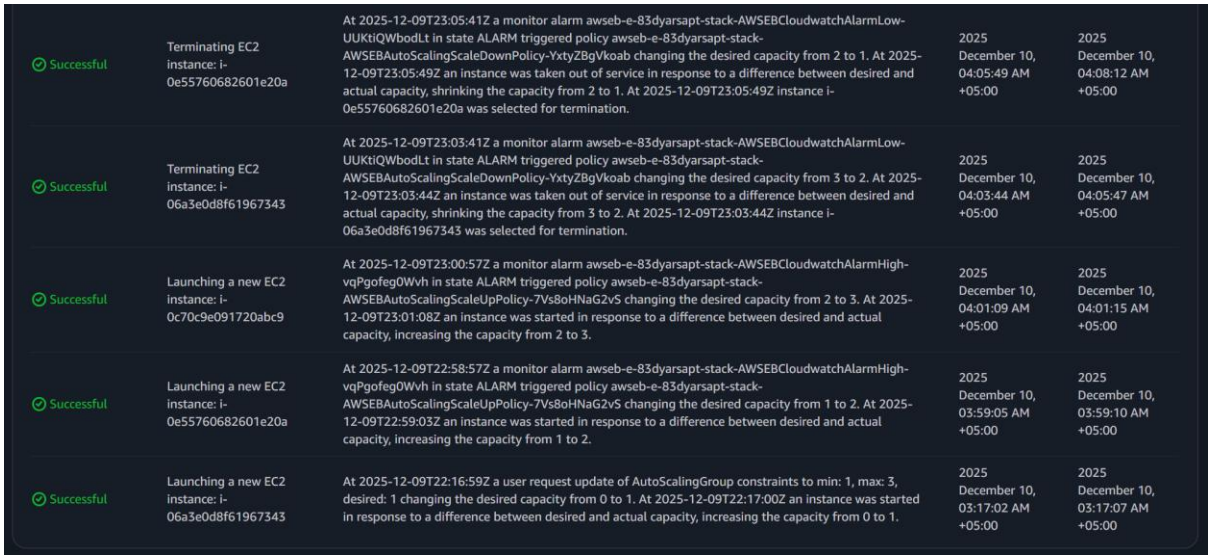


1.6.7: Validate Auto Scaling Activity

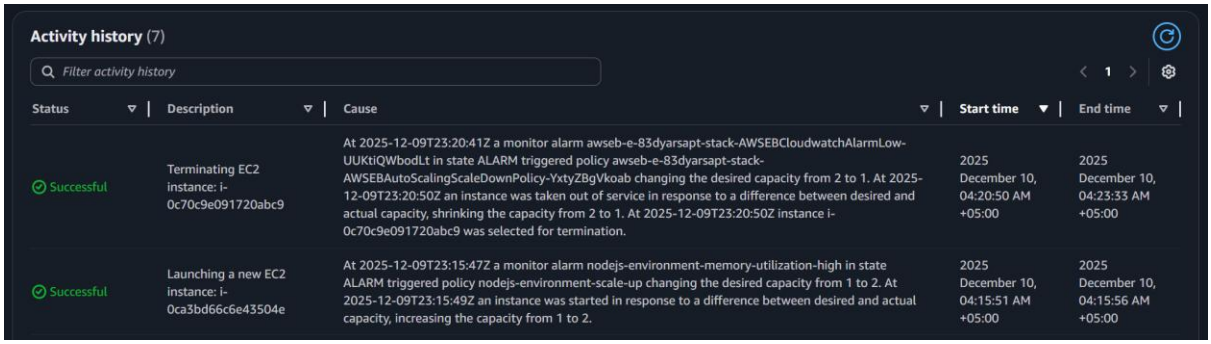
- Navigate to the EC2 console
- Click the **Auto Scaling Group** option in the left navigation bar
- Verify:
    - A new EC2 instance was launched due to triggered alarms.
    - Activity history shows scale out events from 1 to 2 or 3 instances (as maximum was set to 3).
    - Activity history shows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



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

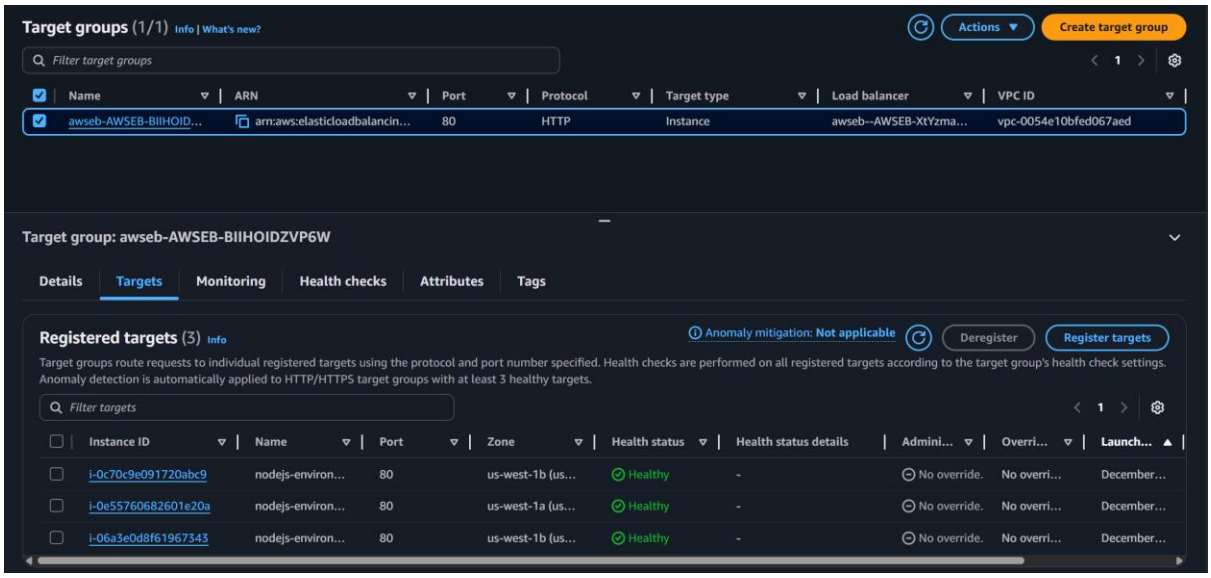


2. Scale-in and Scale-out during Memory Utilization alarm (Auto Scaling group)



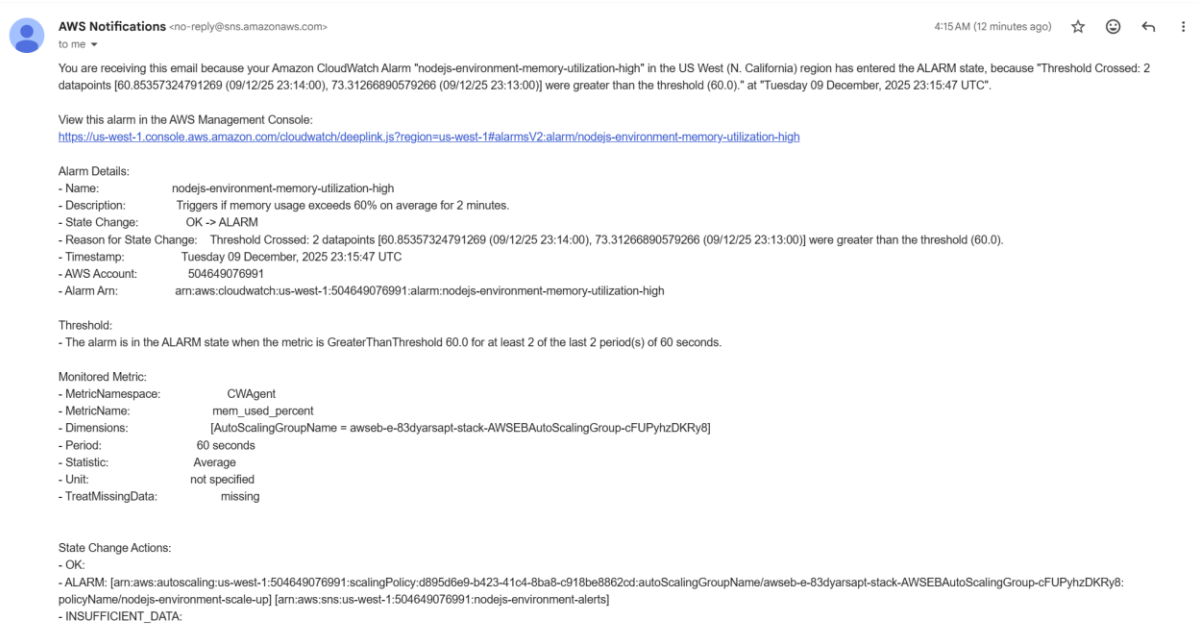
1.6.8: Verify Target Registration in Target Group

- Navigate to the EC2 console
- Click the **Auto Scaling Group** option in the left navigation bar
- Verify:
    - Newly launched EC2 instance(s) appear as **registered targets**.
    - Health status transitions from *initial* → *healthy* once the instance is ready.



1.6.9: SNS Email Notifications

1. Verification of Memory Utilization Alarm – Email Notification



AWS Notifications <no-reply@sns.amazonaws.com> 4:15 AM (12 minutes ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "nodejs-environment-memory-utilization-high" in the US West (N. California) region has entered the ALARM state, because "Threshold Crossed: 2 datapoints [60.85357324791269 (09/12/25 23:14:00), 73.31266890579266 (09/12/25 23:13:00)] were greater than the threshold (60.0)." at "Tuesday 09 December, 2025 23:15:47 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/nodejs-environment-memory-utilization-high

Alarm Details:
- Name: 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 [60.85357324791269 (09/12/25 23:14:00), 73.31266890579266 (09/12/25 23:13:00)] were greater than the threshold (60.0).
- Timestamp: Tuesday 09 December, 2025 23:15:47 UTC
- AWS Account: 504649076991
- Alarm Arn: arn:aws:cloudwatch:us-west-1:504649076991:alarm: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-83dyarsapt-stack-AWSEBAutoScalingGroup-cFUPyhzDKRy8]
- Period: 60 seconds
- Statistic: Average
- Unit: not specified
- TreatMissingData: missing

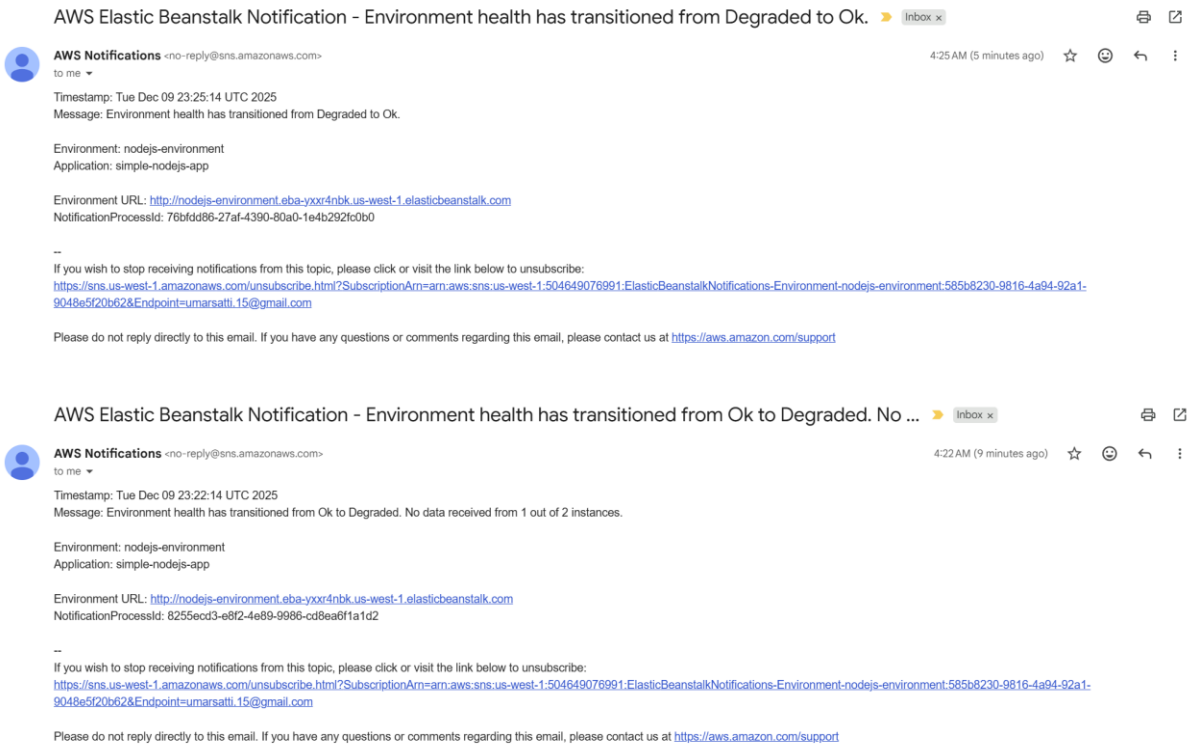
State Change Actions:
- OK:
- ALARM: [arn:aws:autoscaling:us-west-1:504649076991:scalingPolicy:d895d6e9-b423-41c4-8ba8-c918be8862cd:autoScalingGroupName/awseb-e-83dyarsapt-stack-AWSEBAutoScalingGroup-cFUPyhzDKRy8:policyName/nodejs-environment-scale-up] [arn:aws:sns:us-west-1:504649076991:nodejs-environment-alerts]
- INSUFFICIENT_DATA:

2. Elastic Beanstalk Environment Health – Email Notifications

These emails are sent via Amazon SNS to notify you of changes in the Elastic Beanstalk environment health. For example:

- Ok → Degraded: Indicates that one or more instances are not responding, potentially due to high load or application issues.
- Degraded → Ok: Shows that the environment has recovered and all instances are healthy again.

They help monitor the environment in real time and verify that the application is responding as expected.



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

AWS Notifications <no-reply@sns.amazonaws.com> 4:25 AM (5 minutes ago)
to me

Timestamp: Tue Dec 09 23:25:14 UTC 2025
Message: Environment health has transitioned from Degraded to Ok.

Environment: nodejs-environment
Application: simple-nodejs-app

Environment URL: http://nodejs-environment.eba-yxxr4nbk.us-west-1.elasticbeanstalk.com
NotificationProcessId: 76bfdd86-27af-4390-80a0-1e4b292fc0b0

--
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-nodejs-environment:585b8230-9816-4a94-92a1-9048e5f20b62&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 Ok to Degraded. No ... Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 4:22 AM (9 minutes ago)
to me

Timestamp: Tue Dec 09 23:22:14 UTC 2025
Message: Environment health has transitioned from Ok to Degraded. No data received from 1 out of 2 instances.

Environment: nodejs-environment
Application: simple-nodejs-app

Environment URL: http://nodejs-environment.eba-yxxr4nbk.us-west-1.elasticbeanstalk.com
NotificationProcessId: 8255ecd3-e8f2-4e89-9986-cd8ea6f1a1d2

--
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-nodejs-environment:585b8230-9816-4a94-92a1-9048e5f20b62&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

Task 1.7: 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

Task 1.8: Troubleshooting

During the deployment of the Elastic Beanstalk environment using Terraform, several issues were encountered. This section documents each problem, the root cause, and the implemented solution.

Issue 1: Invalid ELBScheme Configuration

Problem Description:
Terraform failed to create the Elastic Beanstalk environment with the error:

- ***Invalid option specification (Namespace: 'aws:e2:vpc', OptionName: 'ELBScheme'): Unknown configuration setting.***

Elastic Beanstalk documentation showed that valid values for **ELBScheme** were "public" or "internal", but the setting itself was not recognized.

Root Cause:
The **ELBScheme** configuration option appears to be deprecated or no longer supported in modern Elastic Beanstalk environments.

Solution:
Removed the **ELBScheme** setting entirely. Elastic Beanstalk automatically defaults to a **public** load balancer when using a load-balanced environment.

Issue 2: Incorrect RootVolumeType Value

Problem Description:
Elastic Beanstalk creation failed with an error indicating an invalid volume type:

- ***'10' is not a valid volume type. Must be 'standard', 'gp2', 'gp3' or 'io1'.***

Root Cause:
The **volume_type** variable was incorrectly assigned the value of **var.volume_size** in terraform.tfvars, resulting in the number 10 being interpreted as the volume type.

Solution:
Updated **terraform.tfvars** to correctly assign the storage type to **volume_type** and the numeric value to **volume_size**. The **terraform apply** command worked successfully.

Issue 3: Missing IAM Instance Profile for EC2

Problem Description:
Environment creation stalled with repeated retries and eventually failed with the following error:

- ***couldn't find resource (21 retries)***

Elastic Beanstalk was unable to attach an EC2 Instance Profile.

Root Cause:
Only the EC2 role was created. Elastic Beanstalk requires an **IAM Instance Profile**, not just a role. The instance profile was missing from the IAM module.

Solution:
Added the missing resource:

- ***resource "aws_iam_instance_profile" "ec2_instance_profile" { ... }***

Exported it via outputs.tf, passed it into the Beanstalk module, and used it in:

- ***IamInstanceProfile = var.ec2_instance_profile***

After this, the EC2 instances successfully launched.

Issue 4: Elastic Beanstalk Cannot Use Custom CloudWatch Metrics (Memory)

Problem Description:
Elastic Beanstalk did not allow selecting CloudWatch Agent custom metrics (e.g., **Memory**) as Auto Scaling triggers.

Root Cause:
Elastic Beanstalk's built-in scaling only supports native EC2 metrics (i.e. CPUUtilization, NetworkIn, NetworkOut etc.). Custom metrics from CloudWatch Agent must be applied through external CloudWatch alarms.

Solution:
Used Terraform to:

- Directly reference the Auto Scaling Group
- Create additional SNS Topic and Subscription
- Create CloudWatch Alarm for **mem_used_percent** (from CloudWatch Agent)
- Set up alarm actions

This enabled scaling based on CloudWatch Agent memory metrics even though Elastic Beanstalk does not support them natively.

Issue 5: Insufficient Permissions for CloudWatch Agent Custom Metrics

Problem Description:
Custom metrics (Memory) were not appearing, and the CloudWatch Agent failed to publish them.

Root Cause:
The EC2 instance role lacked required permissions such as:

- ***autoscaling:DescribeAutoScalingInstances***

- ***ec2:DescribeTags***

Solution:
Added the following policy block to the Instance Profile IAM role:

```
{
    "Sid": "CWAgentASGAccess",
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeTags",
        "autoscaling:DescribeAutoScalingInstances"
    ],
    "Resource": "*"
}
```

After updating the IAM role, the CloudWatch Agent successfully published memory metrics.