Task 14

Deploying Node.js Application On AWS EC2 Instances Using Terraform and GitHub Actions

Umar Satti

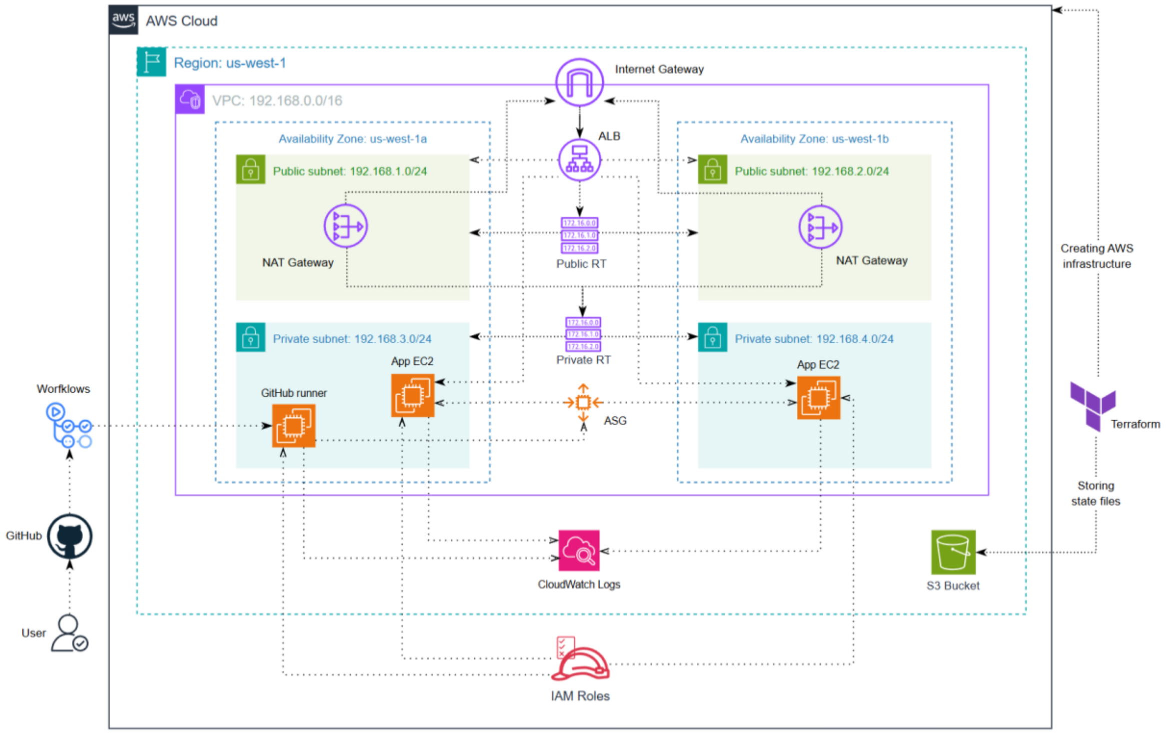
Table of Contents

1. Task Description

This project implements a CI/CD pipeline using GitHub Actions, Terraform, and AWS services. All infrastructure including VPC networking, IAM roles, Application EC2, Load Balancer, Auto Scaling Groups, GitHub Runner EC2, and CloudWatch logs are provisioned using Terraform.

GitHub self-hosted runner is set up on a private EC2 instance without public IPs and is accessed securely via AWS Systems Manager (SSM) Session Manager. The Node.js application runs using PM2 process manager and Nginx. The application code is built, pushed, and deployed to application EC2 instances that are attached to Application load balancer using GitHub workflows YAML script.

2. Architecture Diagram



3. Project Structure

This project is organized into application source files and Terraform infrastructure code. The structure separates application logic, CI/CD configuration, and infrastructure provisioning for clarity and maintainability.

3.1 Application Files

These files define the Node.js application that is deployed to Public EC2 instance.

3.1.1 app.js

- Main application entry point.
- Uses Express to serve static files from the public directory.
- Listens on port 3000 (or environment variable PORT).
- Includes a /health endpoint for basic health checks.

3.1.2 package.json

- Defines application metadata and dependencies.
- Specifies Node.js runtime version.
- Includes scripts for starting the application.
- Lists express as the primary dependency.

3.1.3 public/index.html

- Static HTML file served by the Node.js application.
- Displays a simple UI confirming successful deployment.
- Used to verify application availability after EC2 deployment.

3.2 Terraform Infrastructure Code

This directory contains Terraform configuration used to provision and manage all AWS infrastructure required for GitHub Actions, EC2, VPC, IAM, and CloudWatch. The setup follows a modular design to improve reusability and consistency.

3.2.1 Root Terraform Files

These files act as the entry point for Terraform and orchestrate all infrastructure modules.

1. terraform.tf

This file defines Terraform and AWS provider requirements.

- Locks the AWS provider version to 6.27.0 for consistency.
- Uses HTTP provider version 3.5.0 for fetching IP.

- Configures **remote state storage** using an S3 backend.
- Terraform state is stored in:
  - **S3 Bucket:** umarsatti-terraform-state-file-s3-bucket-sandbox
  - **File Path:** Task-14-GutHub-Actions/terraform.tfstate
  - **Region:** us-west-1
- Enables state encryption and locking to prevent concurrent modifications.
- Configures the AWS provider region as **us-west-1**.

2. main.tf

Serves as the central orchestration file and calls all required modules.

- Includes VPC, IAM, EC2, and ALB.
- Passes shared outputs (such as subnets, security groups, and IAM roles) between modules.
- Ensures proper dependency flow across infrastructure components.

3. variables.tf

Declares all input variables required by the root module. Groups variables logically by module. Enables flexible configuration without modifying core Terraform code.

4. terraform.tfvars

Supplies concrete values for all declared variables.

- Defines environment-specific settings such as:
  - VPC CIDR and naming
  - EC2 AMI and instance sizes
  - IAM role and policy names
  - ALB listener and target group configuration
- Allows easy reuse of the same Terraform code across environments.

5. outputs.tf

Exposes key infrastructure outputs after Terraform execution. Provides the following outputs for display after infrastructure creation:

- Private IP address of Runner EC2 instance.
- Application Load balancer DNS name.

3.2.2 Terraform Modules Directory

1. VPC Module

The VPC module provisions the complete networking layer, including subnets, routing, Internet gateway, NAT gateway, and security groups.

vpc/main.tf

- Defines a custom VPC with DNS support and hostname enabled.
- Creates a public and private subnet in **us-west-1a** AZ.
- Provisions Internet gateway and NAT gateway for public subnet.
- Configures public route table with internet access and Private route table with outbound internet access via NAT gateway.
- Implements security groups for Application EC2 (port 22, 3000, and 80) and Runner EC2 (no inbound traffic)
- Uses local variables to simplify subnet and routing mappings.

vpc/variables.tf

Declares input variables required to configure the VPC:

- CIDR blocks for VPC, public subnet, and private subnet.
- Naming for VPC, IGW, NAT, Subnets, Route tables, and Security groups.
- Internet and route parameters.

vpc/outputs.tf

Exposes key networking resources to other modules including VPC ID, public and private subnet IDs, and security group IDs for EC2 instances.

2. IAM Module

The IAM module provisions roles and policies required by application and GitHub runner EC2 instances. These are attached to the instances as IAM Instance profiles.

iam/main.tf

Creates an **IAM role** with trust relationship for EC2 service as well as an Instance Profile for attaching the role to EC2 instances. Uses AWS Managed policies as well as external JSON document for inline policies. Include the following policy permissions:

- **AmazonSSMManagedInstanceCore** (for Session Manager access)
- **CloudWatchAgentServerPolicy** (for CloudWatch agent)

iam/variables.tf

Declares input variables required to configure the IAM:

- EC2 role and policy names
- Instance profile name

iam/outputs.tf

Exposes the IAM instance profile name to be used by EC2 module.

3. EC2 Module

The EC2 module provisions the Application instances and GitHub self-hosted runner instance in private subnets. The Application EC2 instances are created via launch template and autoscaling group.

ec2/main.tf

Uses a local map to define **application EC2** and **runner EC2** instance. Both instances use the same AMI, instance type, IAM instance profile, and storage.

Application EC2:

- Placed in the private subnets.
- Attached to a security group allowing port 22, 80, and 3000 traffic.
- Bootstrapped using **app_ec2.sh** script file.

GitHub Runner EC2:

- Placed in the private subnet
- Uses a restricted security group by allowing no inbound traffic.
- Bootstrapped using **runner_ec2.sh** script file.

ec2/variables.tf

Declares input variables required to configure the EC2:

- EC2 names and AMI IDs.
- Instance IDs and Availability zones.
- EBS volume type and size.

Declares input variables that are referenced from other modules:

- Public and private subnets from VPC module.
- EC2 security groups from VPC module.
- IAM Instance profile from IAM module.

ec2/outputs.tf

Outputs private IPv4 address of runner EC2 instance.

3.2.3 Supporting Terraform Files

This section describes user data scripts and policy documents used by Terraform to bootstrap EC2 instances and define IAM permissions.

1. Application EC2 Bootstrap Script (app_ec2.sh)

This script is executed automatically when the Application EC2 instance is created to prepare the runtime environment for a Node.js application.

- Waits for full internet connectivity to ensure reliable package installation.
- Installs core system utilities including curl, Git, Nginx, and unzip.
- Installs Node.js 20 system-wide along with npm.
- Installs PM2 and configures it to start on boot using systemd.
- Creates structured application directories for rollback.
- Configures Nginx as a reverse proxy to forward HTTP traffic to Node.js app.
- Installs and configures the Amazon CloudWatch Agent.

This ensures the application server is production-ready immediately after provisioning, with process management, reverse proxying, and centralized logging in place.

2. GitHub Self-hosted Runner EC2 Bootstrap Script (runner_ec2.sh)

This script provisions the GitHub Self-hosted Runner EC2 instance used for CI/CD pipeline execution.

- Waits for internet availability before starting installation steps.
- Installs basic dependencies such as unzip, curl, and AWS CLI.
- Installs nodejs and npm packages to run application.
- Fetches the runner token from SSM Parameter Store.
- Downloads runner packages and starts the service.
- Installs and configures the Amazon CloudWatch Agent.

This guarantees that the GitHub self-hosted runner is fully operational and accessible as soon as the EC2 instance is launched.

3. CloudWatch Agent configuration on App EC2 (app_cw.json)

The app_cw.json file configures the CloudWatch Agent on the **Application EC2 instance** to centralize application and web server logs.

- The agent runs as root and collects logs every **60 seconds**.
- PM2-managed Node.js application logs are collected from /var/www/app/logs/app.log and sent to the nodejs-app log group.
- Nginx access and error logs are collected from /var/log/nginx/access.log and /var/log/nginx/error.log, each published to separate log streams within the same nodejs-app log group.

4. CloudWatch Agent configuration on Runner EC2 (runner_cw.json)

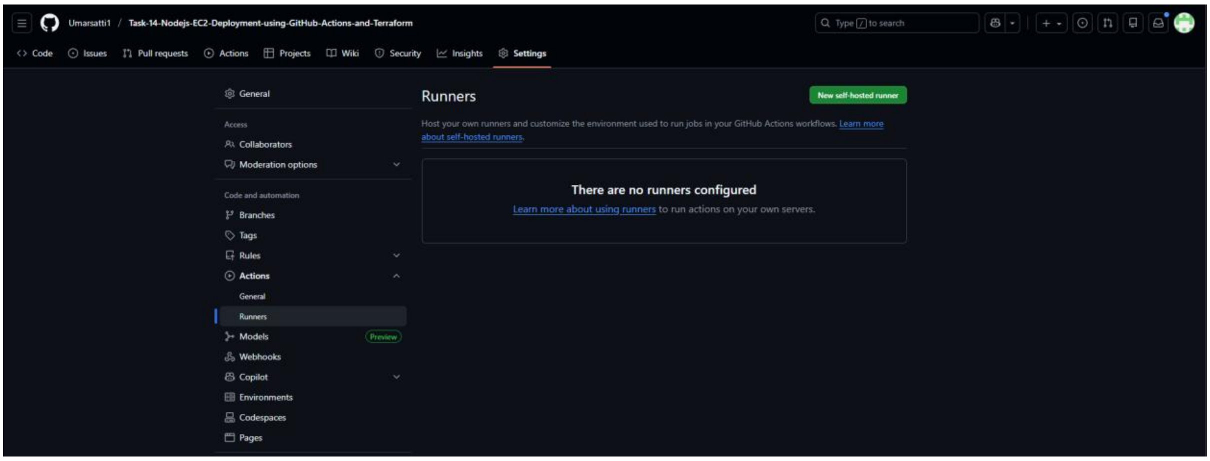
The **runner_cw.json** file defines log collection for the **GitHub self-hosted runner EC2 instance** to support CI/CD observability.

- The CloudWatch Agent runs as root and collects logs at **60 second intervals**.
- GitHub Actions runner diagnostic logs are collected from /home/ubuntu/actions-runner/_diag/*.log and sent to the github-runner log group.
- EC2 bootstrap logs from /var/log/cloud-init.log are also collected, enabling troubleshooting of user-data and provisioning failures.

3.3 GitHub Self-hosted Runner Configuration

To enable the EC2 instance to register as a GitHub self-hosted runner, a registration token must be generated in GitHub and securely stored in AWS Systems Manager Parameter Store.

1. Generate the Runner Registration Token
Navigate to the GitHub repository containing the application source code. Go to **Settings → Actions → Runners**, then click **New self-hosted runner** (as shown in the figure below). GitHub displays a set of configuration commands along with a **temporary registration token**.



Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. Learn more about self-hosted runners.

There are no runners configured

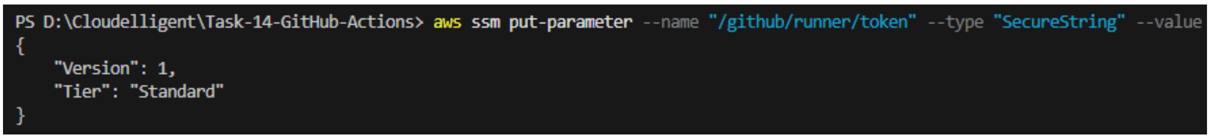
Learn more about using runners to run actions on your own servers.

Note: The required configuration commands are already embedded in the **runner_ec2.sh** user data script. Only the token value is needed at this stage.

2. Store the Token in AWS Parameter Store
Open a terminal with AWS CLI credentials configured and execute the following command to securely store the token:

```
aws ssm put-parameter --name "/github/runner/token" --type "SecureString"
--value <TOKEN>
```

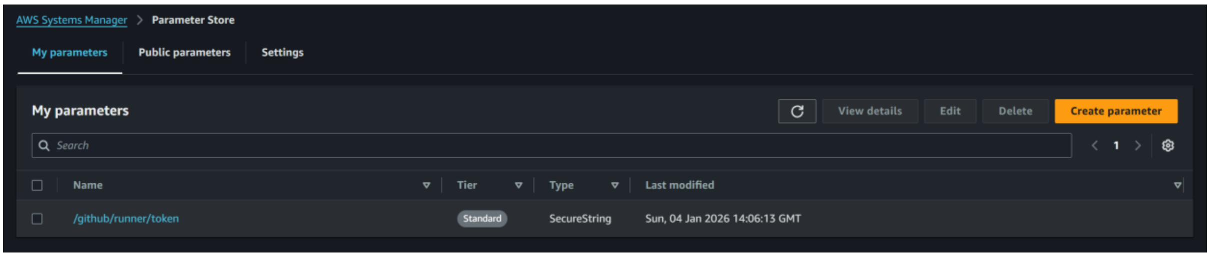
This ensures the token is encrypted and securely retrievable by the runner EC2 instance at launch.



```
PS D:\Cloudelligent\Task-14-GitHub-Actions> aws ssm put-parameter --name "/github/runner/token" --type "SecureString" --value
{
    "Version": 1,
    "Tier": "Standard"
}
```

3. Verification

In the AWS Management Console, navigate to **AWS Systems Manager → Parameter Store → My parameters**. Confirm that the parameter /github/runner/token exists and is stored as a **SecureString**.



My parameters

□	Name	Tier	Type	Last modified
□	/github/runner/token	Standard	SecureString	Sun, 04 Jan 2026 14:06:13 GMT

This approach prevents hardcoding sensitive credentials in Terraform or user data scripts while enabling automated and secure runner registration during EC2 provisioning.

3.4 Terraform Commands

1. Terraform init

Initializes the Terraform working directory by downloading required providers and modules, configuring the remote backend, and preparing Terraform.

2. Terraform validate

Checks the Terraform configuration for syntax and logical errors without creating resources, ensuring the configuration is valid and consistent.

3. Terraform plan

Creates an execution plan that previews the AWS resources Terraform will create and any changes that will be applied, allowing review before deployment.

4. Terraform apply

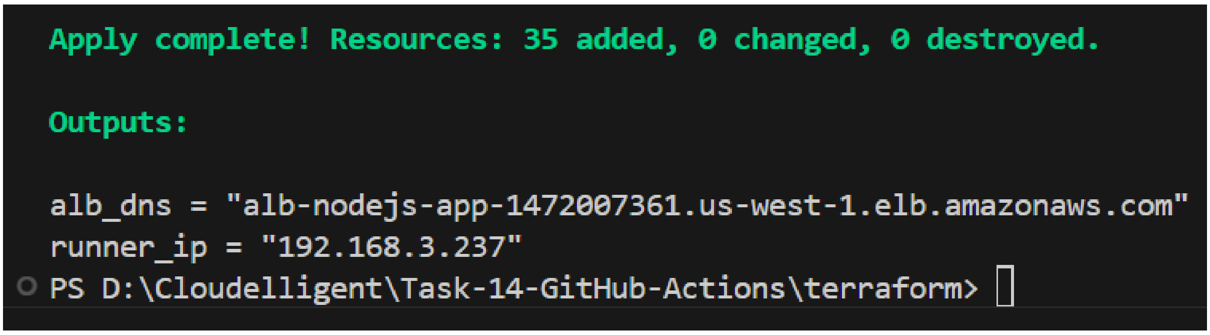
The terraform apply command provisions the AWS infrastructure as defined in the Terraform configuration. Upon confirmation, Terraform creates all required resources defined in VPC, IAM, and EC2 modules.

Terraform also outputs key infrastructure details for later use:

- **ALB DNS:** alb-nodejs-app-1472007361.us-west-1.elb.amazonaws.com
- **Runner EC2 Private IP:** 192.168.3.237

These outputs are used for:

- Connecting to GitHub self-hosted runner using its private IP.
- Accessing the deployed application through the ALB.
- Verifying successful infrastructure provisioning.



```
Apply complete! Resources: 35 added, 0 changed, 0 destroyed.

Outputs:

alb_dns = "alb-nodejs-app-1472007361.us-west-1.elb.amazonaws.com"
runner_ip = "192.168.3.237"
PS D:\Cloudelligent\Task-14-GitHub-Actions\terraform>
```

4. Validate Infrastructure in AWS

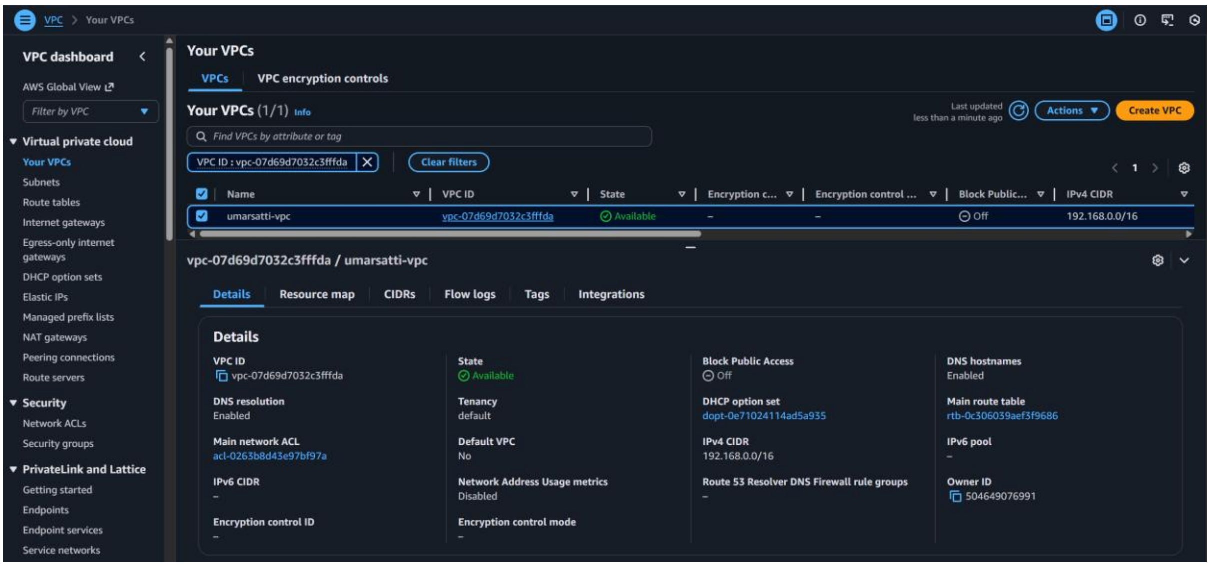
This section validates that all AWS resources created using Terraform are provisioned correctly and functioning as expected.

4.1 VPC and Networking Validation

1. Verify VPC

In the AWS Console, navigate to **VPC** service. Select **Your VPCs** and verify that the VPC create by Terraform exists with the correct configuration. The following shows the VPC configuration:

- **Name:** umarsatti-vpc
- **VPC ID:** vpc-07d69d7032c3fffda
- **IPv4 CIDR:** 192.168.0.0/16
- **DNS Resolution and Hostname:** Enabled



Your VPCs (1/1) Info

VPC ID : vpc-07d69d7032c3fffda

☑	Name	VPC ID	State	Encryption c...	Encryption control ...	Block Public...	IPv4 CIDR
☑	umarsatti-vpc	vpc-07d69d7032c3fffda	Available	–	–	Off	192.168.0.0/16

vpc-07d69d7032c3fffda / umarsatti-vpc

Details

VPC ID vpc-07d69d7032c3fffda	State Available	Block Public Access Off	DNS hostnames Enabled
DNS resolution Enabled	Tenancy default	DHCP option set dopt-0e71024114ad5a935	Main route table rtb-0c306039aef3f9686
Main network ACL acl-0263b8d43e97bf97a	Default VPC No	IPv4 CIDR 192.168.0.0/16	IPv6 pool –
IPv6 CIDR –	Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups –	Owner ID 504649076991
Encryption control ID –	Encryption control mode –		

2. Verify Subnets

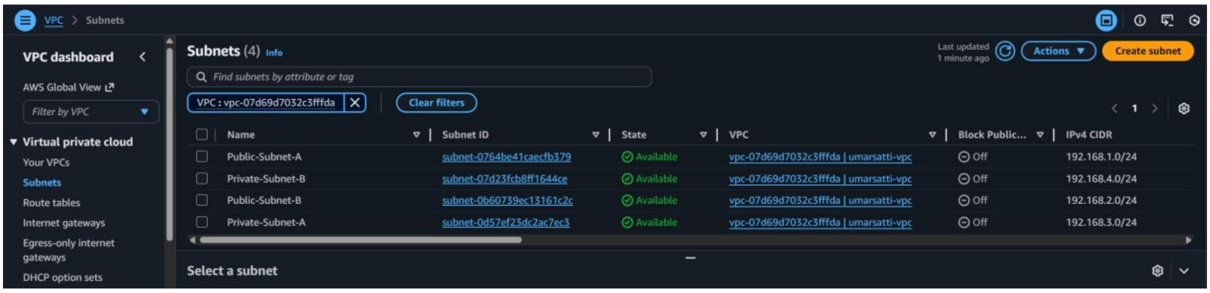
Navigate to **Subnets** section in the VPC console and confirm creation of public and private subnets. The following shows the subnets configuration:

Public Subnets

- **Names:** Public-Subnet-A and Public-Subnet-B
- **IPv4 CIDRs:** 192.168.1.0/16 and 192.168.2.0/16
- **Subnet IDs:** subnet-0764be41caecfb379 and subnet-0b60739ec13161c2c

Private Subnets

- **Names:** Public-Subnet-A and Public-Subnet-B
- **IPv4 CIDRs:** 192.168.3.0/16 and 192.168.4.0/16
- **Subnet IDs:** subnet-0d57ef23dc2ac7ec3 and subnet-07d23fcb8ff1644ce



Subnets (4) Info

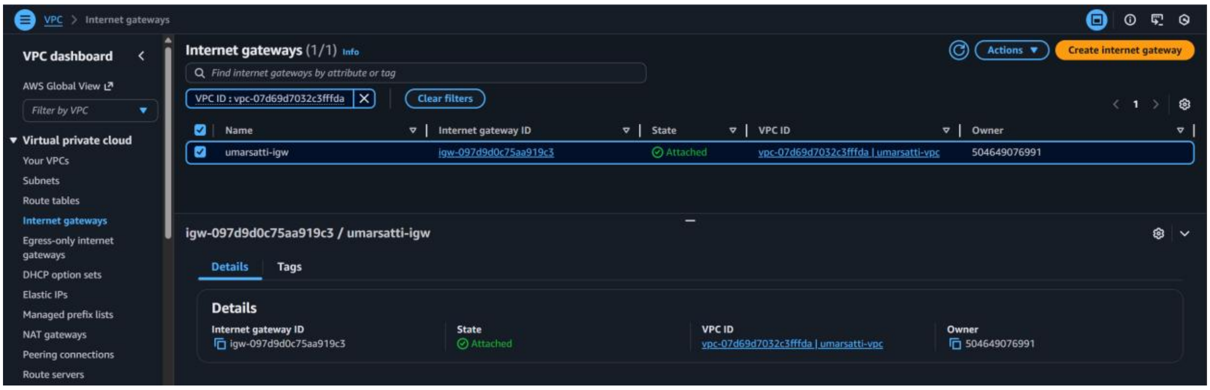
VPC : vpc-07d69d7032c3fffda

□	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
□	Public-Subnet-A	subnet-0764be41caecfb379	✓ Available	vpc-07d69d7032c3fffda | umarsatti-vpc	⊖ Off	192.168.1.0/24
□	Private-Subnet-B	subnet-07d23fcb8ff1644ce	✓ Available	vpc-07d69d7032c3fffda | umarsatti-vpc	⊖ Off	192.168.4.0/24
□	Public-Subnet-B	subnet-0b60739ec13161c2c	✓ Available	vpc-07d69d7032c3fffda | umarsatti-vpc	⊖ Off	192.168.2.0/24
□	Private-Subnet-A	subnet-0d57ef23dc2ac7ec3	✓ Available	vpc-07d69d7032c3fffda | umarsatti-vpc	⊖ Off	192.168.3.0/24

3. Internet Gateway

Navigate to **Internet Gateways** in the VPC console.

- Confirm that Internet gateway exists and **attached** to VPC.
- The following shows the IGW configuration:
  - **Name:** umarsatti-igw
  - **Internet gateway ID:** igw-097d9d0c75aa919c3
  - **State:** Attached



Internet gateways (1/1) Info

VPC ID : vpc-07d69d7032c3fffda

☑	Name	Internet gateway ID	State	VPC ID	Owner
☑	umarsatti-igw	igw-097d9d0c75aa919c3	✓ Attached	vpc-07d69d7032c3fffda | umarsatti-vpc	504649076991

igw-097d9d0c75aa919c3 / umarsatti-igw

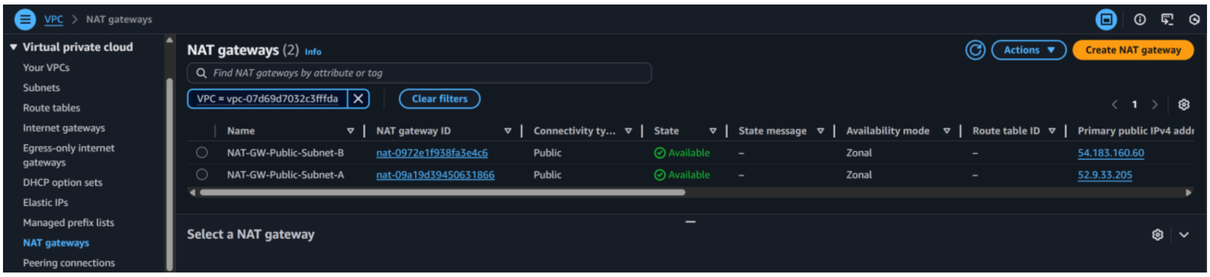
Details

Internet gateway ID	State	VPC ID	Owner
igw-097d9d0c75aa919c3	✓ Attached	vpc-07d69d7032c3fffda | umarsatti-vpc	504649076991

4. NAT Gateway

Navigate to **NAT gateways** in the VPC console.

- Confirm that NAT Gateway exists in public subnet.
- Status shows **Available**.
- The following shows the NAT gateway configuration:
  - **Name:** umarsatti-nat-gw
  - **Nat gateway IDs:** nat-09a19d39450631866 / nat-0972e1f938fa3e4c6
  - **Elastic IPs:** 52.9.33.205 / 54.183.160.60



NAT gateways (2) Info

VPC = vpc-07d69d7032c3fffda

	Name	NAT gateway ID	Connectivity ty...	State	State message	Availability mode	Route table ID	Primary public IPv4 addr
○	NAT-GW-Public-Subnet-B	nat-0972e1f938fa3e4c6	Public	✓ Available	–	Zonal	–	54.183.160.60
○	NAT-GW-Public-Subnet-A	nat-09a19d39450631866	Public	✓ Available	–	Zonal	–	52.9.33.205

5. Route Tables

Navigate to **Route Tables** in the VPC console. Confirm that public and private route tables are created. The following shows the configuration:

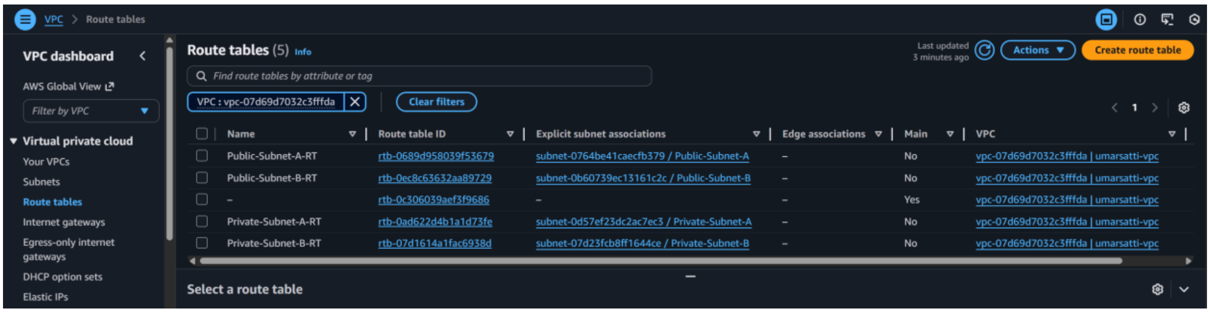
- **Public route table names**
  - Public-Subnet-A-RT and Public-Subnet-B-RT
- **Private route table names:**
  - Private-Subnet-A-RT and Private-Subnet-B-RT

Select the **Public route tables** and confirm the following.

- Explicit Association with public subnets. (in Subnet Associations tab).
- Contains route **0.0.0.0/0 → Internet Gateway** (in Routes tab).

Select the **Private route tables** and confirm the following.

- Explicit Association with private subnets. (in Subnet associations tab).
- Contains route **0.0.0.0/0 → NAT Gateway** (in Routes tab).

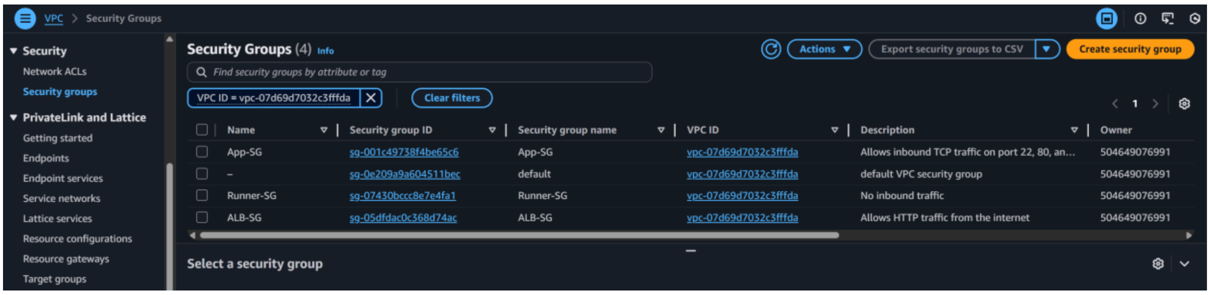


Route tables (5) Info

Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC
Public-Subnet-A-RT	rtb-0689d958039f53679	subnet-0764be41caecfb379 / Public-Subnet-A	–	No	vpc-07d69d7032c3fffda | umarsatti-vpc
Public-Subnet-B-RT	rtb-0ec8c63632aa89729	subnet-0b60739ec13161c2c / Public-Subnet-B	–	No	vpc-07d69d7032c3fffda | umarsatti-vpc
–	rtb-0c306039aef3f9686	–	–	Yes	vpc-07d69d7032c3fffda | umarsatti-vpc
Private-Subnet-A-RT	rtb-0ad622d4b1a1d73fe	subnet-0d57ef23dc2ac7ec3 / Private-Subnet-A	–	No	vpc-07d69d7032c3fffda | umarsatti-vpc
Private-Subnet-B-RT	rtb-07d1614a1fac6938d	subnet-07d23fcb8ff1644ce / Private-Subnet-B	–	No	vpc-07d69d7032c3fffda | umarsatti-vpc

6. Security Groups

Navigate to **Security Groups** in the VPC console. Confirm that both the security groups are created. The following shows the security group configuration:

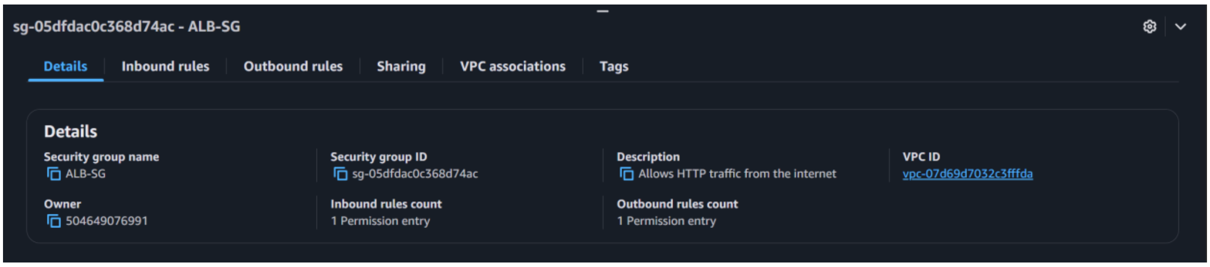


Security Groups (4) Info

Name	Security group ID	Security group name	VPC ID	Description	Owner
App-SG	sg-001c49738f4be65c6	App-SG	vpc-07d69d7032c3fffda	Allows inbound TCP traffic on port 22, 80, an...	504649076991
–	sg-0e209a9a604511bec	default	vpc-07d69d7032c3fffda	default VPC security group	504649076991
Runner-SG	sg-07430bccc8e7e4fa1	Runner-SG	vpc-07d69d7032c3fffda	No inbound traffic	504649076991
ALB-SG	sg-05dfdac0c368d74ac	ALB-SG	vpc-07d69d7032c3fffda	Allows HTTP traffic from the internet	504649076991

Application Load Balancer Security group

- **Inbound rules:** HTTP traffic from anywhere (0.0.0.0/0).
- **Outbound rules:** All traffic allowed (0.0.0.0/0)



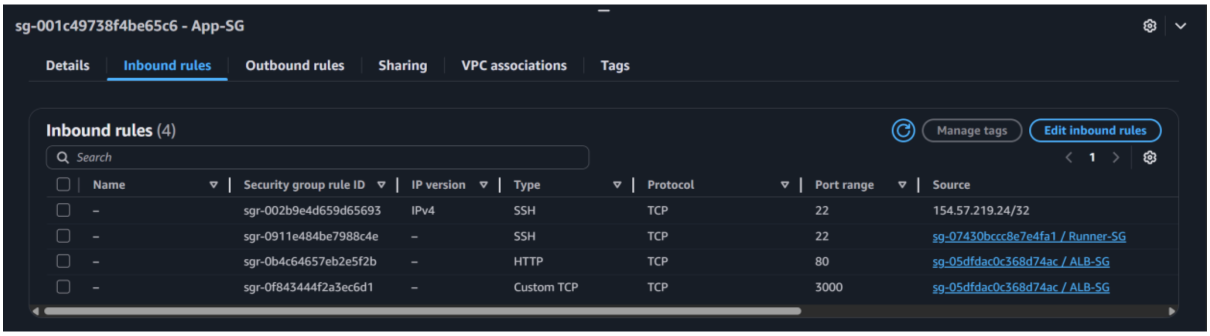
sg-05dfdac0c368d74ac - ALB-SG

Details

Security group name ALB-SG	Security group ID sg-05dfdac0c368d74ac	Description Allows HTTP traffic from the internet	VPC ID vpc-07d69d7032c3fffda
Owner 504649076991	Inbound rules count 1 Permission entry	Outbound rules count 1 Permission entry	

Application EC2 Security group

- **Inbound rules:** Traffic on Port 22, 80, and 3000.
- **Outbound rules:** All traffic allowed (0.0.0.0/0)



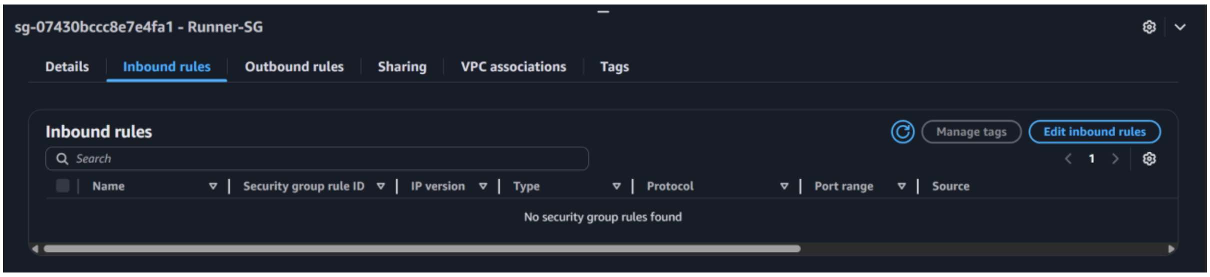
sg-001c49738f4be65c6 - App-SG

Inbound rules (4)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
□	–	sgr-002b9e4d659d65693	IPv4	SSH	TCP	22	154.57.219.24/32
□	–	sgr-0911e484be7988c4e	–	SSH	TCP	22	sg-07430bccc8e7e4fa1 / Runner-SG
□	–	sgr-0b4c64657eb2e5f2b	–	HTTP	TCP	80	sg-05dfdac0c368d74ac / ALB-SG
□	–	sgr-0f843444f2a3ec6d1	–	Custom TCP	TCP	3000	sg-05dfdac0c368d74ac / ALB-SG

Runner Security group

- **Inbound rules:** None.
- **Outbound rules:** All traffic allowed (0.0.0.0/0)



sg-07430bccc8e7e4fa1 - Runner-SG

Inbound rules

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source

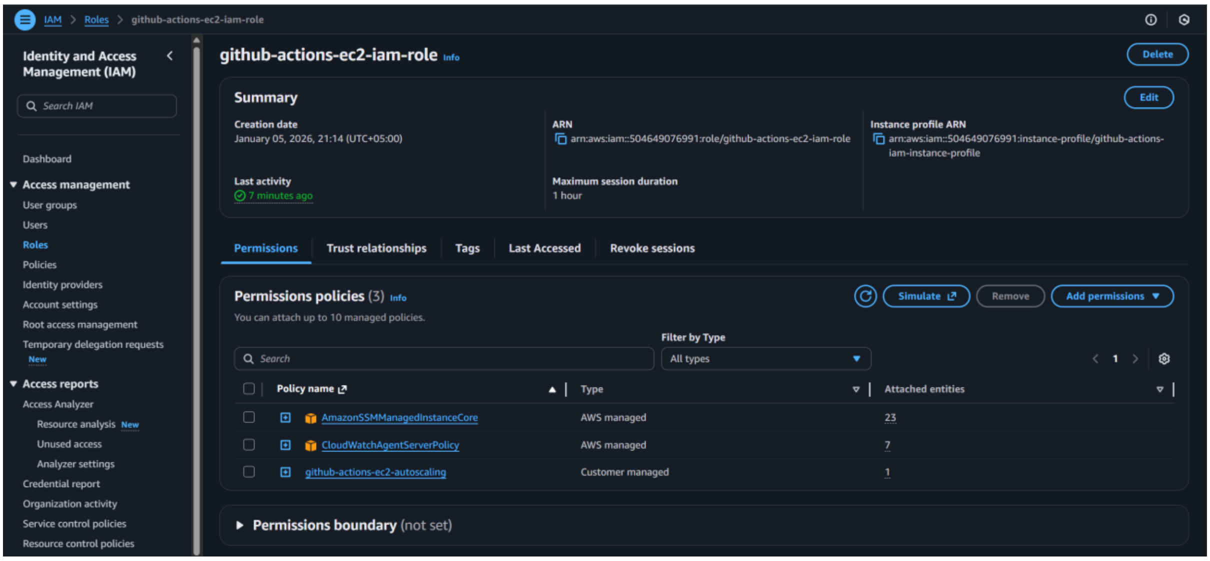
No security group rules found

4.2 IAM Validation

1. EC2 Instance Role

Navigate to **IAM** console and select **Roles**. Use the search bar to verify that the IAM role named **“github-actions-ec2-iam-role”** exists. Select this IAM role and confirm the following:

- Trust relationship and Instance profile exists.
- Inline policy matches **ec2_autoscaling_policy.json**.
- Permissions policies include AmazonSSMManageInstanceCore, CloudWatchAgentServerPolicy, and github-actions-ec2-autoscaling.



github-actions-ec2-iam-role Info

Summary

Creation date: January 05, 2026, 21:14 (UTC+05:00)

ARN: arn:aws:iam::504649076991:role/github-actions-ec2-iam-role

Instance profile ARN: arn:aws:iam::504649076991:instance-profile/github-actions-iam-instance-profile

Last activity: 7 minutes ago

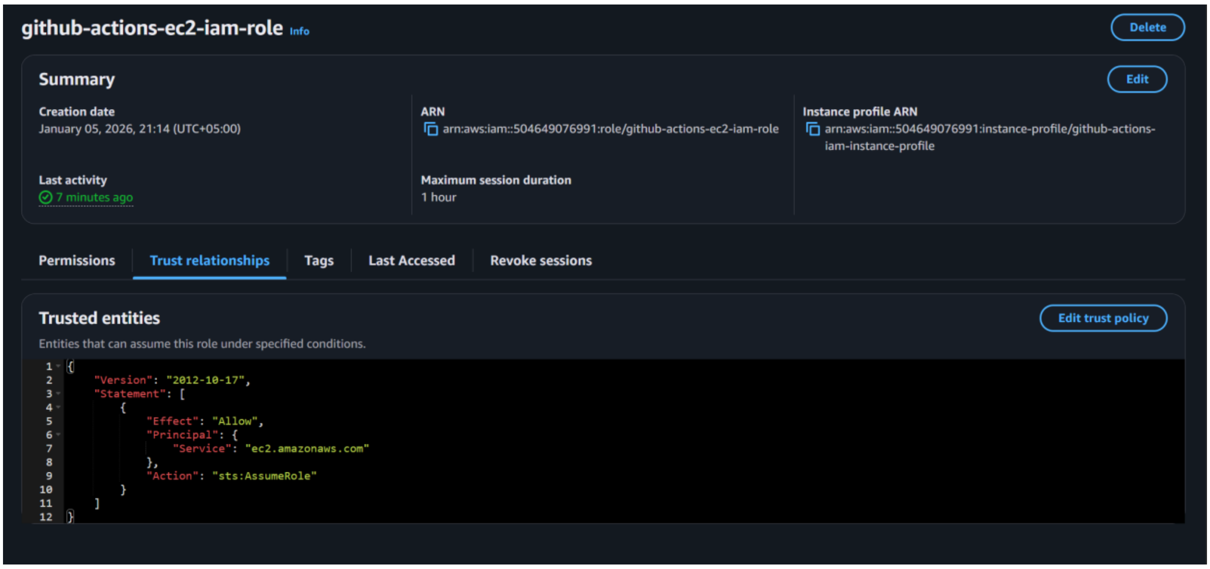
Maximum session duration: 1 hour

Permissions policies (3) Info

You can attach up to 10 managed policies.

Filter by Type: All types

☐	Policy name	Type	Attached entities
☐	AmazonSSMManagedInstanceCore	AWS managed	23
☐	CloudWatchAgentServerPolicy	AWS managed	7
☐	github-actions-ec2-autoscaling	Customer managed	1



github-actions-ec2-iam-role Info

Summary

Creation date: January 05, 2026, 21:14 (UTC+05:00)

ARN: arn:aws:iam::504649076991:role/github-actions-ec2-iam-role

Instance profile ARN: arn:aws:iam::504649076991:instance-profile/github-actions-iam-instance-profile

Last activity: 7 minutes ago

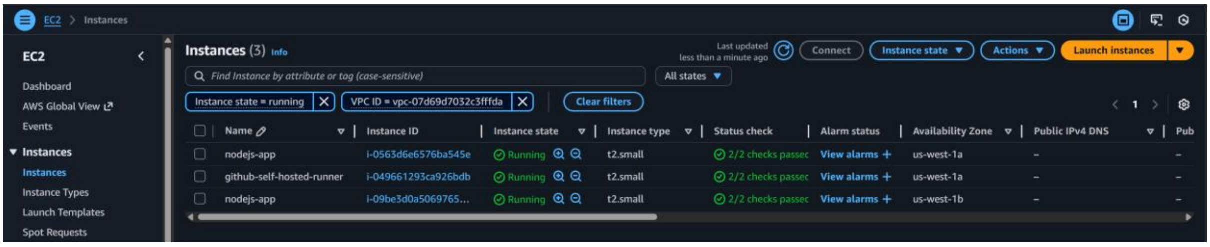
Maximum session duration: 1 hour

Trusted entities

Entities that can assume this role under specified conditions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4.3 EC2 Validation



Instances (3) Info

Last updated less than a minute ago

Instance state = running ✕ VPC ID = vpc-07d69d7032c3fffda ✕

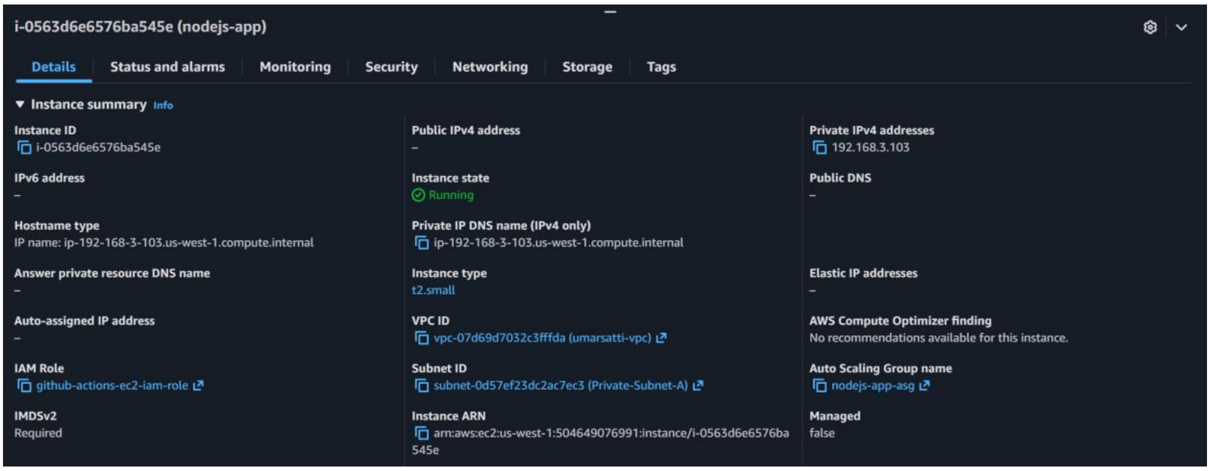
☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Pub
☐	nodejs-app	i-0563d6e6576ba545e	Running	t2.small	2/2 checks passec	View alarms +	us-west-1a	–	–
☐	github-self-hosted-runner	i-049661293ca926bdb	Running	t2.small	2/2 checks passec	View alarms +	us-west-1a	–	–
☐	nodejs-app	i-09be3d0a5069765...	Running	t2.small	2/2 checks passec	View alarms +	us-west-1b	–	–

1. Application EC2 instance

Navigate to **EC2** console and select **Instances**. Select the application EC2 instance and verify the following:

- Application EC2 **(nodejs-app-instance)** instance is created.
- Instance is running and status checks are passed.

- Located in **public subnets** and have no public IPs.
- Has private IPv4 addresses of 10.0.3.xxx and 10.0.4.xxx.
- IAM instance profile attached.
- Correct security group attached to it.
- Instance type matches Terraform configuration.



i-0563d6e6576ba545e (nodejs-app)

Instance ID
i-0563d6e6576ba545e

IPv6 address
–

Hostname type
IP name: ip-192-168-3-103.us-west-1.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
–

IAM Role
github-actions-ec2-iam-role

IMDSv2
Required

Public IPv4 address
–

Instance state
Running

Private IP DNS name (IPv4 only)
ip-192-168-3-103.us-west-1.compute.internal

Instance type
t2.small

VPC ID
vpc-07d69d7032c3fffda (umarsatti-vpc)

Subnet ID
subnet-0d57ef23dc2ac7ec3 (Private-Subnet-A)

Instance ARN
arn:aws:ec2:us-west-1:504649076991:instance/i-0563d6e6576ba545e

Private IPv4 addresses
192.168.3.103

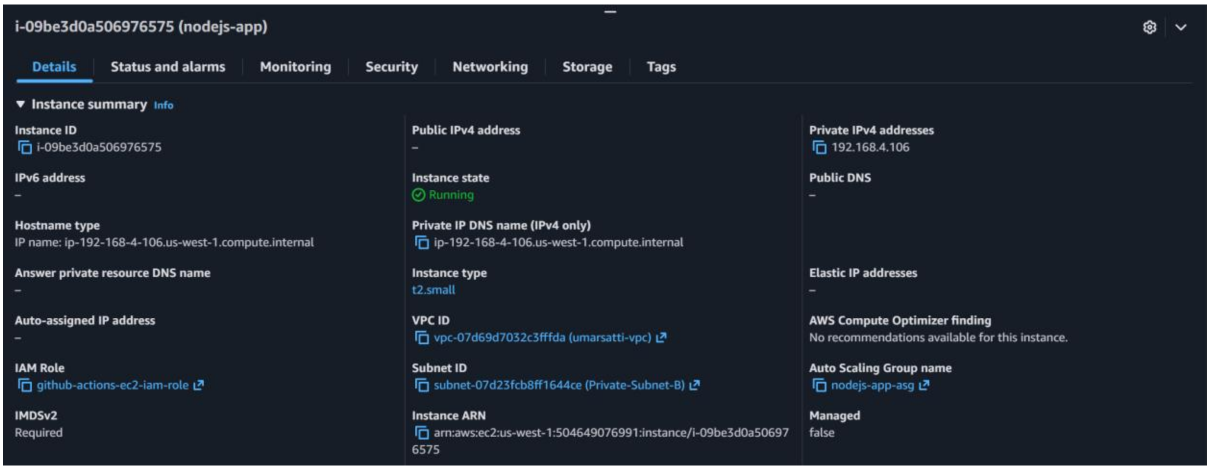
Public DNS
–

Elastic IP addresses
–

AWS Compute Optimizer finding
No recommendations available for this instance.

Auto Scaling Group name
nodejs-app-asg

Managed
false



i-09be3d0a506976575 (nodejs-app)

Instance ID
i-09be3d0a506976575

IPv6 address
–

Hostname type
IP name: ip-192-168-4-106.us-west-1.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
–

IAM Role
github-actions-ec2-iam-role

IMDSv2
Required

Public IPv4 address
–

Instance state
Running

Private IP DNS name (IPv4 only)
ip-192-168-4-106.us-west-1.compute.internal

Instance type
t2.small

VPC ID
vpc-07d69d7032c3fffda (umarsatti-vpc)

Subnet ID
subnet-07d23fcb8ff1644ce (Private-Subnet-B)

Instance ARN
arn:aws:ec2:us-west-1:504649076991:instance/i-09be3d0a506976575

Private IPv4 addresses
192.168.4.106

Public DNS
–

Elastic IP addresses
–

AWS Compute Optimizer finding
No recommendations available for this instance.

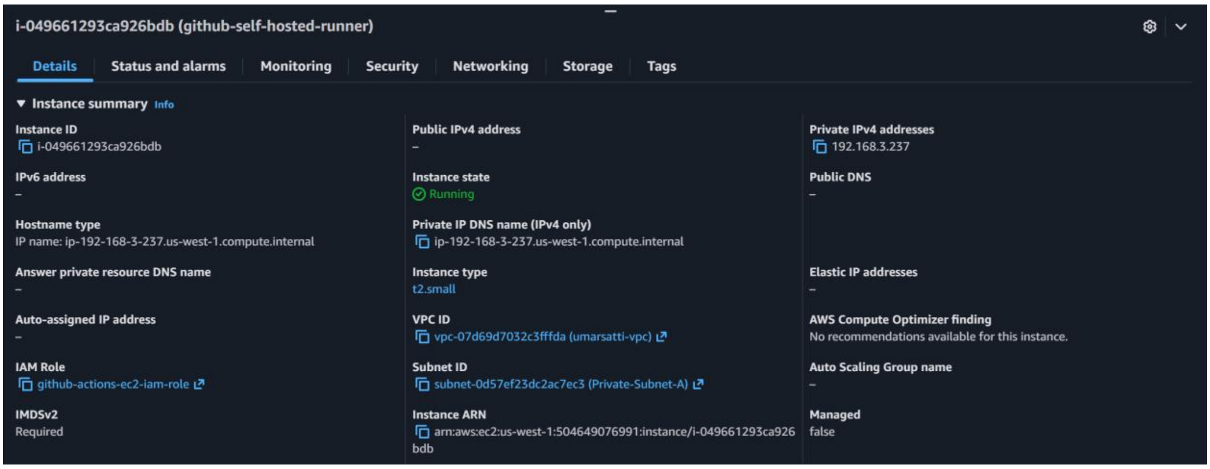
Auto Scaling Group name
nodejs-app-asg

Managed
false

2. GitHub Runner EC2 instance

Navigate to **EC2** console and select **Instances**. Select the Runner EC2 instance and verify the following:

- Runner EC2 **(github-self-hosted-runner)** instance is created.
- Instance is running and status checks are passed.
- Located in **private subnet** without a public IPv4 address.
- Has a private IPv4 address of 10.0.3.xx.
- IAM instance profile attached.
- Correct security group attached to it.
- Instance type matches Terraform configuration.



i-049661293ca926bdb (github-self-hosted-runner)

▼ Instance summary Info

Instance ID
i-049661293ca926bdb

IPv6 address
–

Hostname type
IP name: ip-192-168-3-237.us-west-1.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
–

IAM Role
github-actions-ec2-iam-role

IMDSv2
Required

Public IPv4 address
–

Instance state
Running

Private IP DNS name (IPv4 only)
ip-192-168-3-237.us-west-1.compute.internal

Instance type
t2.small

VPC ID
vpc-07d69d7032c3fffda (umarsatti-vpc)

Subnet ID
subnet-0d57ef23dc2ac7ec3 (Private-Subnet-A)

Instance ARN
arn:aws:ec2:us-west-1:504649076991:instance/i-049661293ca926bdb

Private IPv4 addresses
192.168.3.237

Public DNS
–

Elastic IP addresses
–

AWS Compute Optimizer finding
No recommendations available for this instance.

Auto Scaling Group name
–

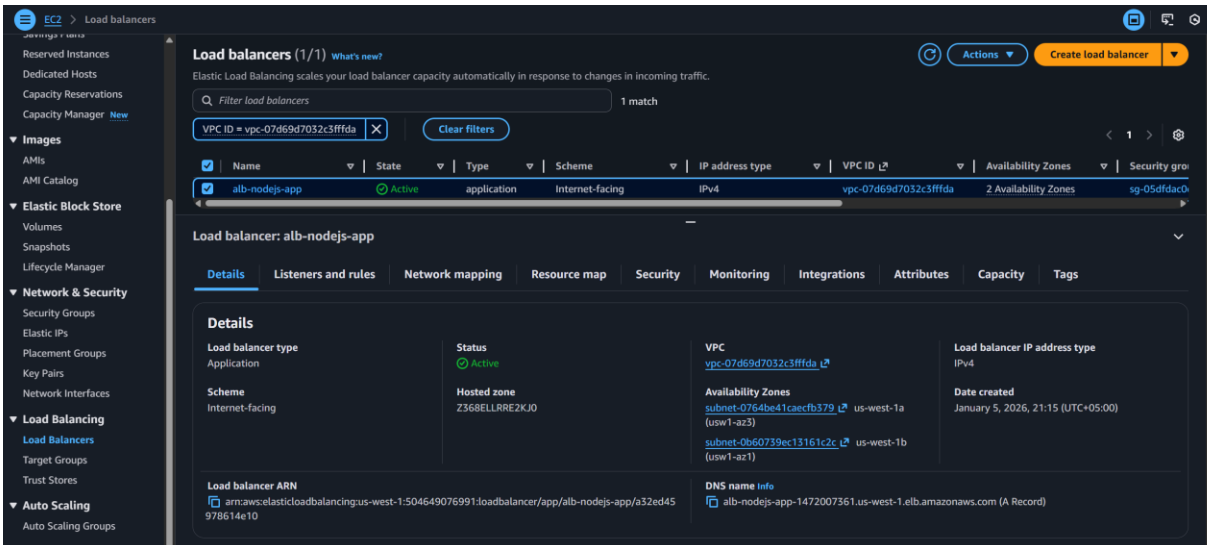
Managed
false

4.4 Load Balancer Verification

Navigate to **EC2 Console** and click **Load Balancers.** Select the Application Load Balancer named **alb-nodejs-app** and confirm the following:

- Belongs to the VPC created earlier
- **Availability zones**: us-west-1a and us-west-1b
- **Security groups**: ALB-SG
- Has a DNS name

The presence of an active ALB spanning two Availability Zones confirms high availability and correct network configuration as defined in Terraform.



Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

1 match

VPC ID = vpc-07d69d7032c3fffda

☑	Name	State	Type	Scheme	IP address type	VPC ID	Availability Zones	Security gro
☑	alb-nodejs-app	Active	application	Internet-facing	IPv4	vpc-07d69d7032c3fffda	2 Availability Zones	sg-05dfdac0

Load balancer: alb-nodejs-app

Details

Load balancer type
Application

Scheme
Internet-facing

Status
Active

Hosted zone
Z368ELLRRE2KJ0

VPC
vpc-07d69d7032c3fffda

Availability Zones
subnet-0764be41caecfb379 us-west-1a (usw1-az3)
subnet-0b60739ec13161c2c us-west-1b (usw1-az1)

Load balancer IP address type
IPv4

Date created
January 5, 2026, 21:15 (UTC+05:00)

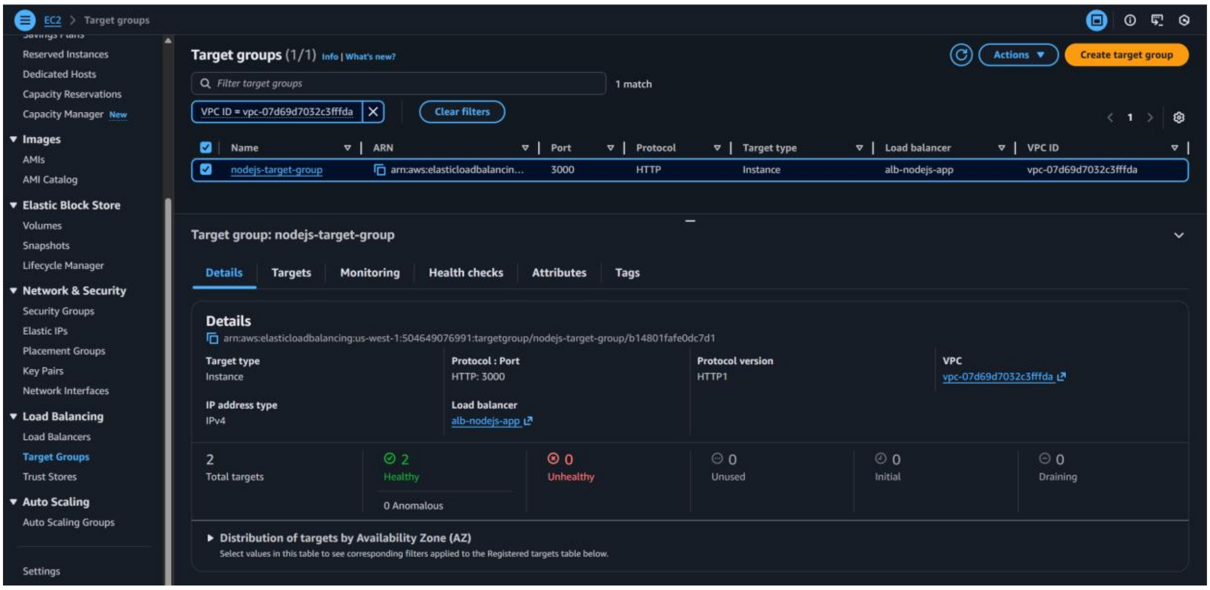
Load balancer ARN
arn:aws:elasticloadbalancing:us-west-1:504649076991:loadbalancer/app/alb-nodejs-app/a32ed45978614e10

DNS name Info
alb-nodejs-app-1472007361.us-west-1.elb.amazonaws.com (A Record)

4.5 Target Group Verification

Navigate to **EC2 Console** and click **Target Groups.** Select the target group named **nodejs-target-group** and confirm the following:

- **Target type**: Instance
- **Protocol/Port**: HTTP/3000
- **Total targets**: 2
- All targets are **healthy** and deployed in different Availability Zones.



Target groups (1/1)

1 match

VPC ID = vpc-07d69d7032c3fffda

☑	Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
☑	nodejs-target-group	arn:aws:elasticloadbalancin...	3000	HTTP	Instance	alb-nodejs-app	vpc-07d69d7032c3fffda

Target group: nodejs-target-group

Details

arn:aws:elasticloadbalancing:us-west-1:504649076991:targetgroup/nodejs-target-group/b14801fafe0dc7d1

Target type: Instance

Protocol : Port: HTTP: 3000

Protocol version: HTTP1

VPC: vpc-07d69d7032c3fffda

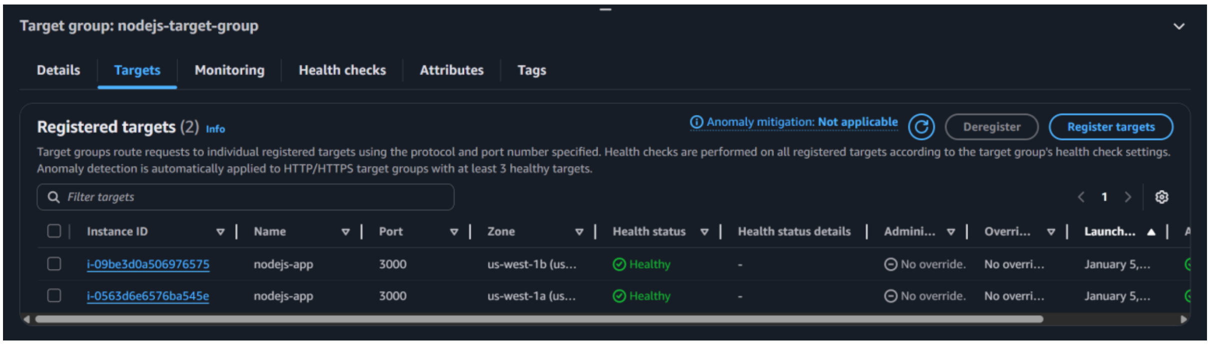
IP address type: IPv4

Load balancer: alb-nodejs-app

2 Total targets | 2 Healthy | 0 Anomalous | 0 Unhealthy | 0 Unused | 0 Initial | 0 Draining

▶ Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.



Target group: nodejs-target-group

Registered targets (2)

Anomaly mitigation: Not applicable

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

□	Instance ID	Name	Port	Zone	Health status	Health status details	Admini...	Overri...	Launch...
□	i-09be3d0a506976575	nodejs-app	3000	us-west-1b (us...	Healthy	-	No override.	No overri...	January 5,...
□	i-0563d6e6576ba545e	nodejs-app	3000	us-west-1a (us...	Healthy	-	No override.	No overri...	January 5,...

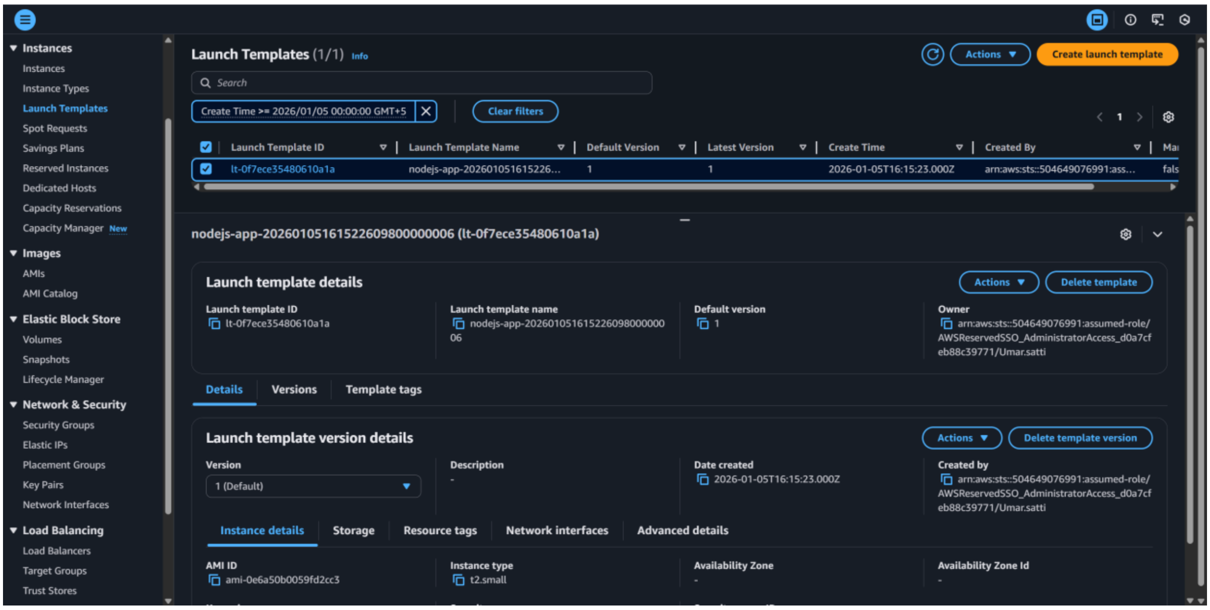
This confirms that the Node.js application instances are correctly registered and passing health checks across multiple AZs

4.6 Launch Template Verification

Navigate to **EC2 Console → Launch Templates** and select the template named **nodejs-app-**. Verify the following:

- **Launch Template ID:** lt-0f7ece35480610a1a
- **Default / Latest version:** 1
- **AMI ID:** ami-0e6a50b0059fd2cc3
- **Instance type:** t2.small
- **Security Group ID:** sg-001c49738f4be65c6
- **Key pair:** Not configured

The launch template must exactly match the Terraform configuration to ensure consistent and repeatable instance provisioning.



Launch Templates (1/1) Info

Create Time >= 2026/01/05 00:00:00 GMT+5

☑	Launch Template ID	Launch Template Name	Default Version	Latest Version	Create Time	Created By	Ma
☑	lt-0f7ece35480610a1a	nodejs-app-202601051615226...	1	1	2026-01-05T16:15:23.000Z	arn:aws:sts::504649076991:ass...	fals

nodejs-app-20260105161522609800000006 (lt-0f7ece35480610a1a)

Launch template details

Launch template ID: lt-0f7ece35480610a1a

Launch template name: nodejs-app-20260105161522609800000006

Default version: 1

Owner: arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_AdministratorAccess_d0a7cfeb88c39771/Umar.satti

Launch template version details

Version: 1 (Default)

Description: -

Date created: 2026-01-05T16:15:23.000Z

Created by: arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_AdministratorAccess_d0a7cfeb88c39771/Umar.satti

AMI ID: ami-0e6a50b0059fd2cc3

Instance type: t2.small

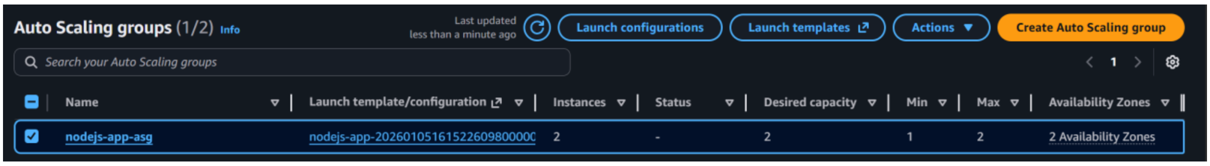
Availability Zone: -

Availability Zone Id: -

4.7 Auto Scaling Group Verification

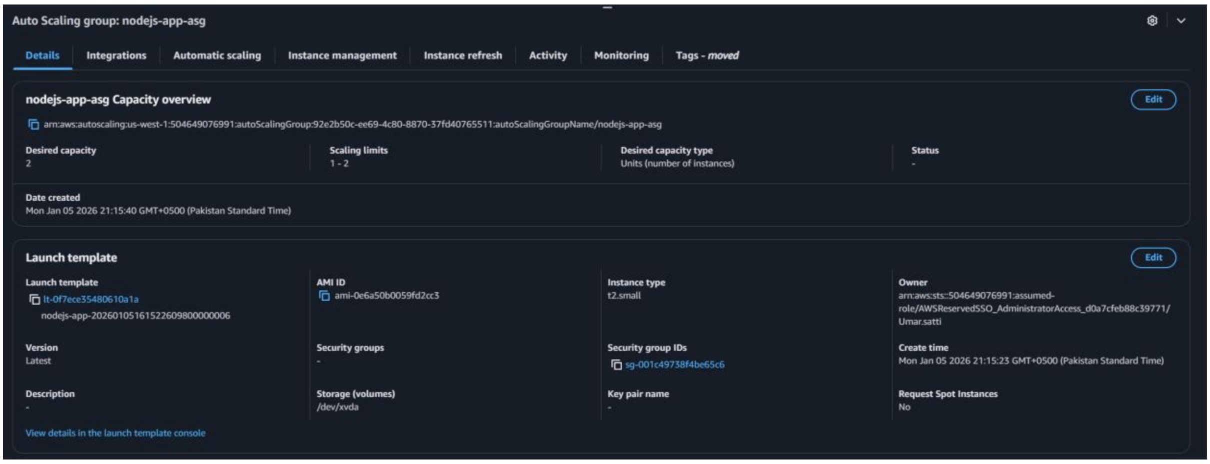
Navigate to **EC2 Console → Auto Scaling Groups** and select **nodejs-app-asg**. Confirm the following settings:

- **Desired capacity:** 2
- **Minimum capacity:** 1
- **Maximum capacity:** 2
- **Launch template:** Match the one created earlier
- **Instance type:** t2.small
- **Availability Zones:** us-west-1a and us-west-1b
- **Subnet distribution:** One instance per AZ (balanced)



Auto Scaling groups (1/2) Info

	Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Availability Zones
☑	nodejs-app-asg	nodejs-app-2026010516152260980000C	2	-	2	1	2	2 Availability Zones



Auto Scaling group: nodejs-app-asg

nodejs-app-asg Capacity overview

arn:aws:autoscaling:us-west-1:504649076991:autoScalingGroup:92e2b50c-ee69-4c80-8870-37fd40765511:autoScalingGroupName/nodejs-app-asg

Desired capacity	Scaling limits	Desired capacity type	Status
2	1 - 2	Units (number of instances)	-

Date created
Mon Jan 05 2026 21:15:40 GMT+0500 (Pakistan Standard Time)

Launch template

Launch template lt-0f7ece35480610a1a nodejs-app-20260105161522609800000006	**AMI ID** ami-0e6a50b0059fd2cc3	**Instance type** t2.small	**Owner** arn:aws:sts::504649076991:assumed-role/AWSReservedSSO_AdministratorAccess_d0a7cfeb88c39771/Umar.satti
Version Latest	**Security groups** -	**Security group IDs** sg-001c49738f4be65c6	**Create time** Mon Jan 05 2026 21:15:23 GMT+0500 (Pakistan Standard Time)
Description -	**Storage (volumes)** /dev/xvda	**Key pair name** -	**Request Spot Instances** No

View details in the launch template console

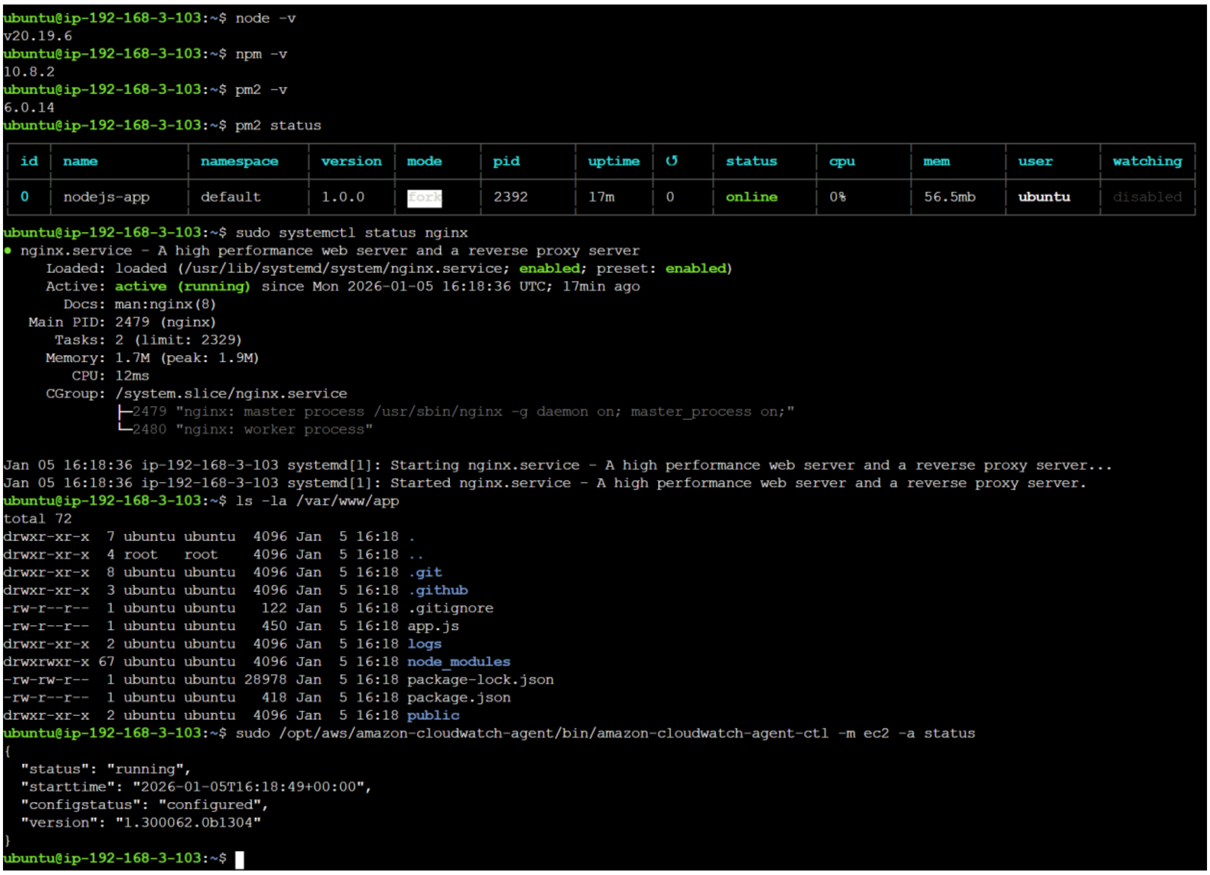
5. Application EC2 Deployment Verification

This section validates that the Application EC2 instances were provisioned correctly and that the deployed Node.js application is accessible through the load balancer.

5.1 Verifying Application EC2 Bootstrap via AWS Session Manager

AWS Session Manager is used to securely access the Application EC2 instance without SSH keys. Once connected, the following commands are executed to confirm that the user data script completed successfully:

- Verify Node.js and npm installation:
  - ***node -v*** and ***npm -v***
- Confirm PM2 is installed and managing the application:
  - ***pm2 -v*** and ***pm2 status***
- Ensure Nginx is running as a reverse proxy
  - ***sudo systemctl status nginx***
- Validate application files and directory structure:
  - ***ls -la /var/www/app***
- Confirm CloudWatch Agent is active
  - ***sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status***

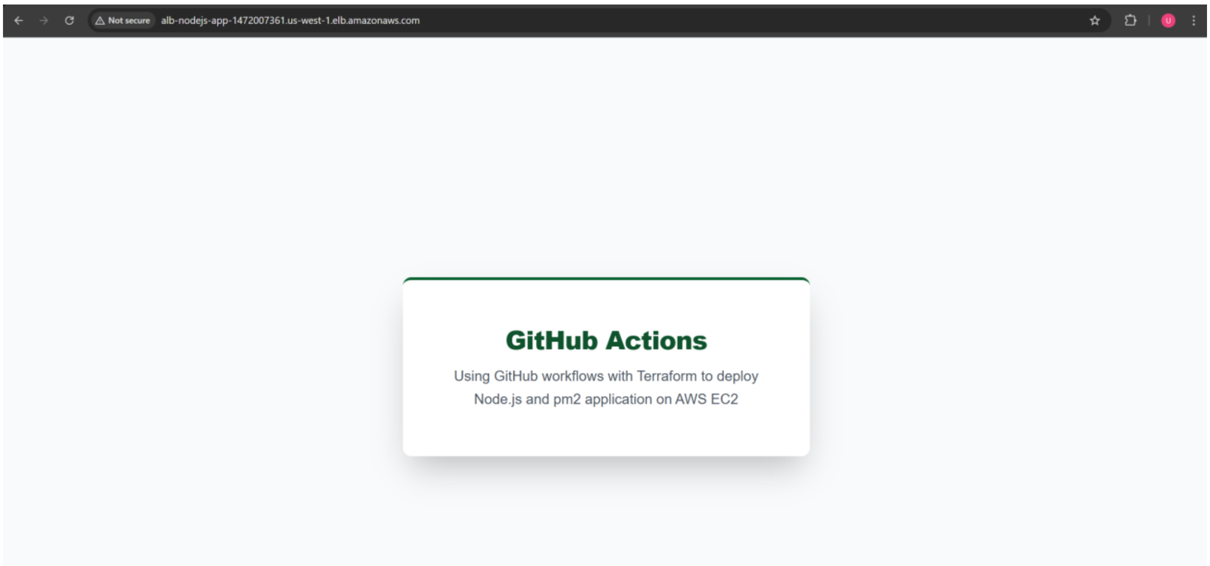


```
ubuntu@ip-192-168-3-103:~$ node -v
v20.19.6
ubuntu@ip-192-168-3-103:~$ npm -v
10.8.2
ubuntu@ip-192-168-3-103:~$ pm2 -v
6.0.14
ubuntu@ip-192-168-3-103:~$ pm2 status
┌────┬───────────────┬─────────────┬─────────┬─────────┬──────────┬────────┬──────┬───────────┬──────────┬──────────┬──────────┬──────────┐
│ id │ name          │ namespace   │ version │ mode    │ pid      │ uptime │ ↺    │ status    │ cpu      │ mem      │ user     │ watching │
├────┼───────────────┼─────────────┼─────────┼─────────┼──────────┼────────┼──────┼───────────┼──────────┼──────────┼──────────┼──────────┤
│ 0  │ nodejs-app    │ default     │ 1.0.0   │ fork    │ 2392     │ 17m    │ 0    │ online    │ 0%       │ 56.5mb   │ ubuntu   │ disabled │
└────┴───────────────┴─────────────┴─────────┴─────────┴──────────┴────────┴──────┴───────────┴──────────┴──────────┴──────────┴──────────┘
ubuntu@ip-192-168-3-103:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Mon 2026-01-05 16:18:36 UTC; 17min ago
       Docs: man:nginx(8)
   Main PID: 2479 (nginx)
      Tasks: 2 (limit: 2329)
     Memory: 1.7M (peak: 1.9M)
        CPU: 12ms
     CGroup: /system.slice/nginx.service
             ├─2479 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             └─2480 "nginx: worker process"

Jan 05 16:18:36 ip-192-168-3-103 systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server...
Jan 05 16:18:36 ip-192-168-3-103 systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.
ubuntu@ip-192-168-3-103:~$ ls -la /var/www/app
total 72
drwxr-xr-x  7 ubuntu ubuntu  4096 Jan  5 16:18 .
drwxr-xr-x  4 root   root    4096 Jan  5 16:18 ..
drwxr-xr-x  8 ubuntu ubuntu  4096 Jan  5 16:18 .git
drwxr-xr-x  3 ubuntu ubuntu  4096 Jan  5 16:18 .github
-rw-r--r--  1 ubuntu ubuntu   122 Jan  5 16:18 .gitignore
-rw-r--r--  1 ubuntu ubuntu   450 Jan  5 16:18 app.js
drwxr-xr-x  2 ubuntu ubuntu  4096 Jan  5 16:18 logs
drwxrwxr-x 67 ubuntu ubuntu  4096 Jan  5 16:18 node_modules
-rw-rw-r--  1 ubuntu ubuntu 28978 Jan  5 16:18 package-lock.json
-rw-r--r--  1 ubuntu ubuntu   418 Jan  5 16:18 package.json
drwxr-xr-x  2 ubuntu ubuntu  4096 Jan  5 16:18 public
ubuntu@ip-192-168-3-103:~$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
{
  "status": "running",
  "starttime": "2026-01-05T16:18:49+00:00",
  "configstatus": "configured",
  "version": "1.300062.0b1304"
}
ubuntu@ip-192-168-3-103:~$
```

5.2 Application Accessibility via Application Load Balancer

The Application Load Balancer (ALB) DNS name **alb-nodejs-app-1472007361.us-west-1.elb.amazonaws.com** is accessed through a web browser to verify end-to-end connectivity. The successful rendering of the Node.js application confirms that:

- EC2 instances are healthy
- Nginx reverse proxy is functioning
- Traffic routing through the ALB is correctly configured



GitHub Actions

Using GitHub workflows with Terraform to deploy Node.js and pm2 application on AWS EC2

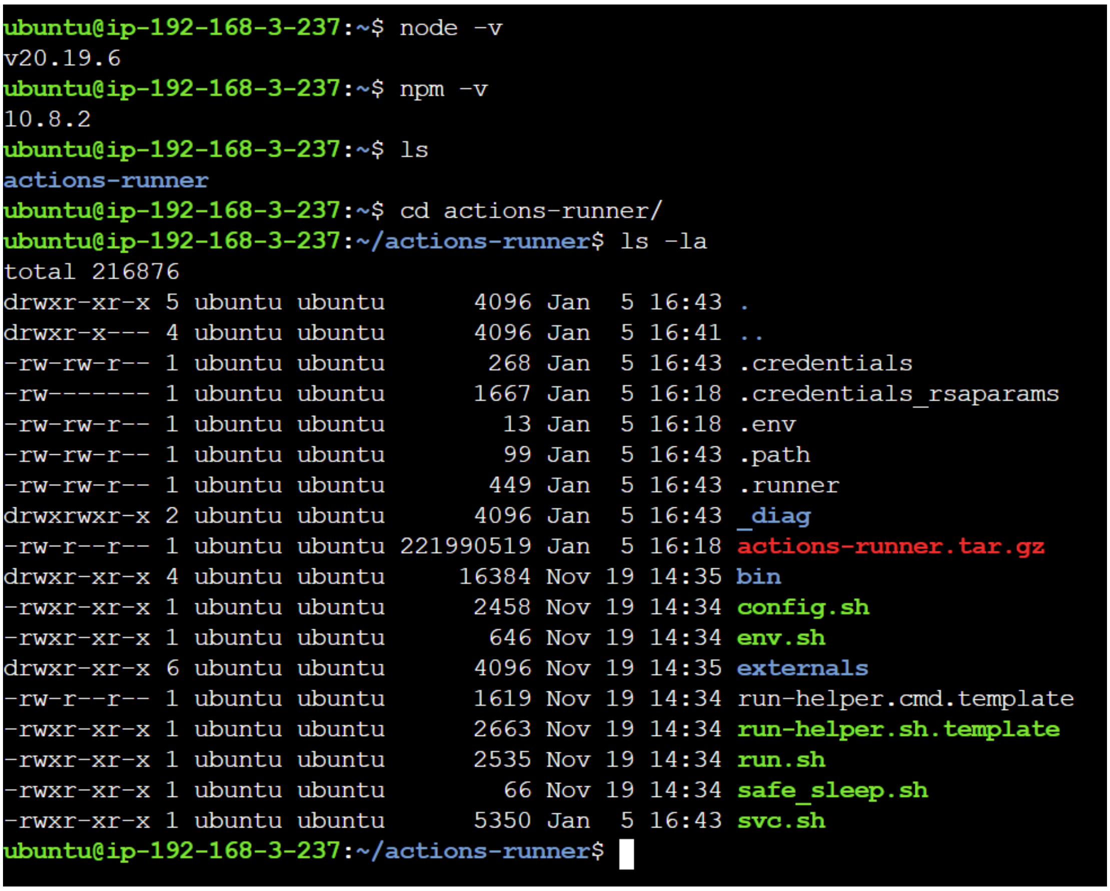
6. GitHub Workflows and CI/CD Pipeline Setup

This section validates the self-hosted GitHub Actions runner and explains the CI/CD pipeline used for automated deployments.

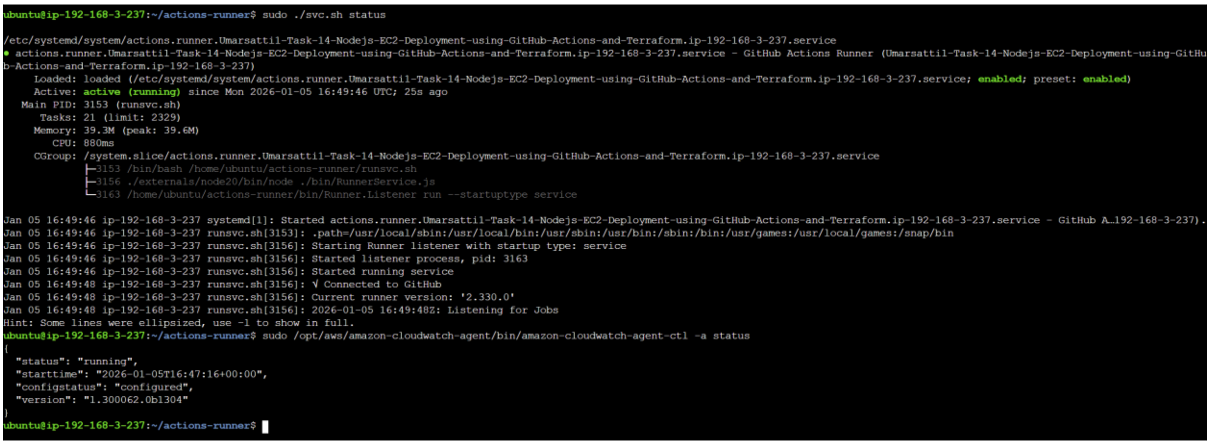
6.1 Verifying Runner EC2 Bootstrap via AWS Session Manager

AWS Session Manager is used to connect to the Runner EC2 instance and confirm successful initialization. The following checks are performed:

- Switch to runner use:
  - ***sudo su - ubuntu***
- Verify AWS CLI installation:
  - ***aws --version***
- Confirm Node.js runtime availability
  - ***node -v*** and ***npm -v***
- Validate GitHub runner installation
  - ***cd actions-runner && ls -la***
- Ensure the runner service is running (inside actions-runner/ directory)
  - ***sudo ./svc.sh status***
- Confirm CloudWatch Agent is active
  - ***sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status***



```
ubuntu@ip-192-168-3-237:~$ node -v
v20.19.6
ubuntu@ip-192-168-3-237:~$ npm -v
10.8.2
ubuntu@ip-192-168-3-237:~$ ls
actions-runner
ubuntu@ip-192-168-3-237:~$ cd actions-runner/
ubuntu@ip-192-168-3-237:~/actions-runner$ ls -la
total 216876
drwxr-xr-x 5 ubuntu ubuntu      4096 Jan  5 16:43 .
drwxr-x--- 4 ubuntu ubuntu      4096 Jan  5 16:41 ..
-rw-rw-r-- 1 ubuntu ubuntu       268 Jan  5 16:43 .credentials
-rw------- 1 ubuntu ubuntu      1667 Jan  5 16:18 .credentials_rsaparams
-rw-rw-r-- 1 ubuntu ubuntu        13 Jan  5 16:18 .env
-rw-rw-r-- 1 ubuntu ubuntu        99 Jan  5 16:43 .path
-rw-rw-r-- 1 ubuntu ubuntu       449 Jan  5 16:43 .runner
drwxrwxr-x 2 ubuntu ubuntu      4096 Jan  5 16:43 _diag
-rw-r--r-- 1 ubuntu ubuntu 221990519 Jan  5 16:18 actions-runner.tar.gz
drwxr-xr-x 4 ubuntu ubuntu     16384 Nov 19 14:35 bin
-rwxr-xr-x 1 ubuntu ubuntu      2458 Nov 19 14:34 config.sh
-rwxr-xr-x 1 ubuntu ubuntu       646 Nov 19 14:34 env.sh
drwxr-xr-x 6 ubuntu ubuntu      4096 Nov 19 14:35 externals
-rw-r--r-- 1 ubuntu ubuntu      1619 Nov 19 14:34 run-helper.cmd.template
-rwxr-xr-x 1 ubuntu ubuntu      2663 Nov 19 14:34 run-helper.sh.template
-rwxr-xr-x 1 ubuntu ubuntu      2535 Nov 19 14:34 run.sh
-rwxr-xr-x 1 ubuntu ubuntu        66 Nov 19 14:34 safe_sleep.sh
-rwxr-xr-x 1 ubuntu ubuntu      5350 Jan  5 16:43 svc.sh
ubuntu@ip-192-168-3-237:~/actions-runner$ 
```



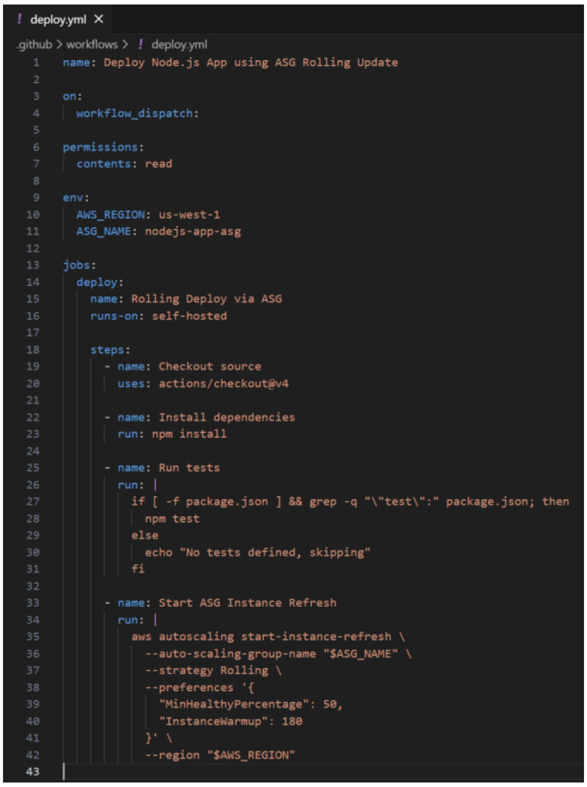
```
ubuntu@ip-192-168-3-237:~/actions-runner$ sudo ./svc.sh status

/etc/systemd/system/actions.runner.Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHub-Actions-and-Terraform.ip-192-168-3-237.service
● actions.runner.Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHub-Actions-and-Terraform.ip-192-168-3-237.service - GitHub Actions Runner (Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHu
b-Actions-and-Terraform.ip-192-168-3-237)
     Loaded: loaded (/etc/systemd/system/actions.runner.Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHub-Actions-and-Terraform.ip-192-168-3-237.service; enabled; preset: enabled)
     Active: active (running) since Mon 2026-01-05 16:49:46 UTC; 25s ago
   Main PID: 3153 (runsvc.sh)
      Tasks: 21 (limit: 2329)
     Memory: 39.3M (peak: 39.6M)
        CPU: 880ms
     CGroup: /system.slice/actions.runner.Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHub-Actions-and-Terraform.ip-192-168-3-237.service
             ├─3153 /bin/bash /home/ubuntu/actions-runner/runsvc.sh
             ├─3156 ./externals/node20/bin/node ./bin/RunnerService.js
             └─3163 /home/ubuntu/actions-runner/bin/Runner.Listener run --startuptype service

Jan 05 16:49:46 ip-192-168-3-237 systemd[1]: Started actions.runner.Umarsattil-Task-14-Nodejs-EC2-Deployment-using-GitHub-Actions-and-Terraform.ip-192-168-3-237.service - GitHub A…192-168-3-237).
Jan 05 16:49:46 ip-192-168-3-237 runsvc.sh[3153]: .path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
Jan 05 16:49:46 ip-192-168-3-237 runsvc.sh[3156]: Starting Runner listener with startup type: service
Jan 05 16:49:46 ip-192-168-3-237 runsvc.sh[3156]: Started listener process, pid: 3163
Jan 05 16:49:46 ip-192-168-3-237 runsvc.sh[3156]: Started running service
Jan 05 16:49:48 ip-192-168-3-237 runsvc.sh[3156]: √ Connected to GitHub
Jan 05 16:49:48 ip-192-168-3-237 runsvc.sh[3156]: Current runner version: '2.330.0'
Jan 05 16:49:48 ip-192-168-3-237 runsvc.sh[3156]: 2026-01-05 16:49:48Z: Listening for Jobs
Hint: Some lines were ellipsized, use -l to show in full.
ubuntu@ip-192-168-3-237:~/actions-runner$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
{
  "status": "running",
  "starttime": "2026-01-05T16:47:16+00:00",
  "configstatus": "configured",
  "version": "1.300062.0b1304"
}
ubuntu@ip-192-168-3-237:~/actions-runner$
```

6.2 GitHub Actions Workflow Configuration

The GitHub Actions pipeline is defined using a YAML workflow and is triggered manually via **workflow_dispatch**. Key pipeline functions include:

- Checking out the application source code
- Installing Node.js dependencies
- Running tests if defined
- Triggering an **Auto Scaling Group rolling instance refresh** using AWS CLI for rolling updates.

This approach enables zero-downtime deployments by gradually replacing EC2 instances with updated application versions.



```yaml
name: Deploy Node.js App using ASG Rolling Update

on:
  workflow_dispatch:

permissions:
  contents: read

env:
  AWS_REGION: us-west-1
  ASG_NAME: nodejs-app-asg

jobs:
  deploy:
    name: Rolling Deploy via ASG
    runs-on: self-hosted

    steps:
      - name: Checkout source
        uses: actions/checkout@v4

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: |
          if [ -f package.json ] && grep -q "\"test\":" package.json; then
            npm test
          else
            echo "No tests defined, skipping"
          fi

      - name: Start ASG Instance Refresh
        run: |
          aws autoscaling start-instance-refresh \
            --auto-scaling-group-name "$ASG_NAME" \
            --strategy Rolling \
            --preferences '{
              "MinHealthyPercentage": 50,
              "InstanceWarmup": 180
            }' \
            --region "$AWS_REGION"

```

6.3 CI/CD Pipeline Execution Verification

The GitHub Actions workflow is executed manually from the Actions console. Screenshots demonstrate:

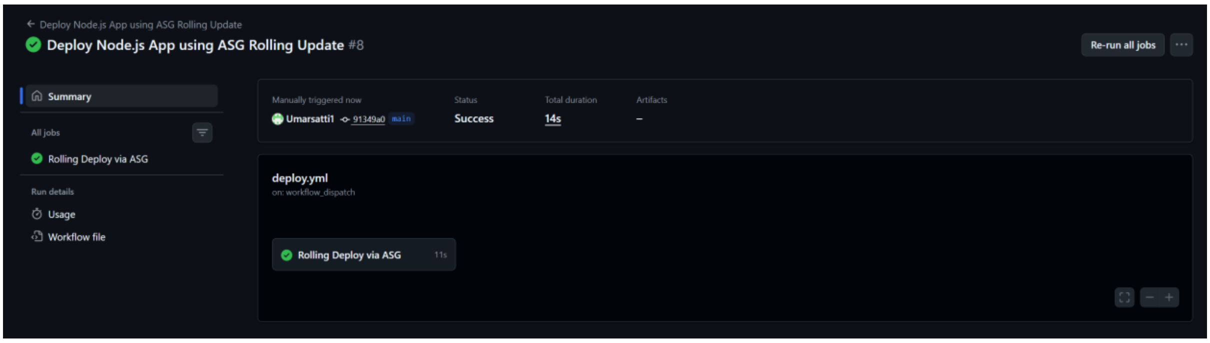
- Successful job execution indicated by a green checkmark
- All pipeline steps completing without errors
- An Auto Scaling Group **rolling instance refresh** being initiated

This confirms that the self-hosted runner, GitHub Actions workflow, and AWS infrastructure are fully integrated and operating as designed.

Key Observations:

- New EC2 instances are launched using the updated launch template
- Existing instances are gracefully deregistered from the target group and placed into a draining state
- Newly launched instances transition from **Unhealthy** to **Healthy** after passing load balancer health checks

This behavior validates a zero-downtime deployment strategy.



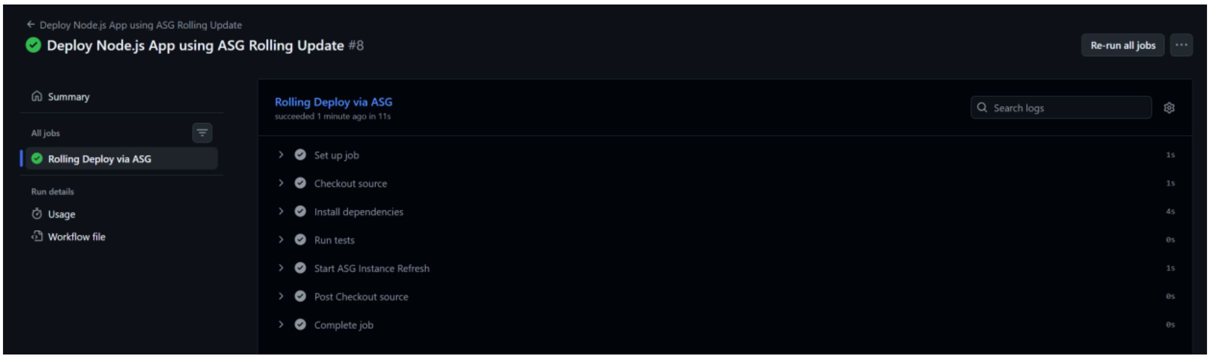
← Deploy Node.js App using ASG Rolling Update

✓ Deploy Node.js App using ASG Rolling Update #8

Manually triggered now	Status	Total duration	Artifacts
Umarsatti1 -○- 91349a0 main	Success	14s	–

deploy.yml
on: workflow_dispatch

✓ Rolling Deploy via ASG 11s



← Deploy Node.js App using ASG Rolling Update

✓ Deploy Node.js App using ASG Rolling Update #8

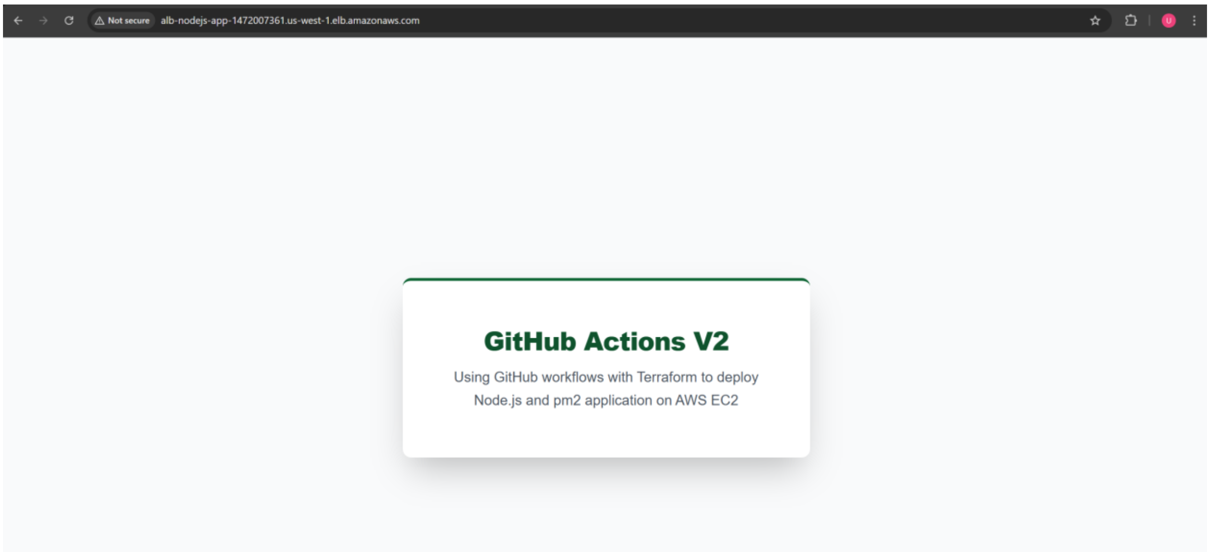
Rolling Deploy via ASG
succeeded 1 minute ago in 11s

```
> ✓ Set up job                      1s
> ✓ Checkout source                 1s
> ✓ Install dependencies            4s
> ✓ Run tests                       0s
> ✓ Start ASG Instance Refresh      1s
> ✓ Post Checkout source            0s
> ✓ Complete job                    0s
```

This confirms that the self-hosted runner, GitHub Actions workflow, and AWS infrastructure are fully integrated and functioning as intended.

6.4 Updated Application

The updated application version becomes accessible automatically through the Application Load Balancer once the new EC2 instances are marked healthy. Screenshots confirm that the latest application changes are successfully deployed and served without service interruption.



GitHub Actions V2

Using GitHub workflows with Terraform to deploy Node.js and pm2 application on AWS EC2

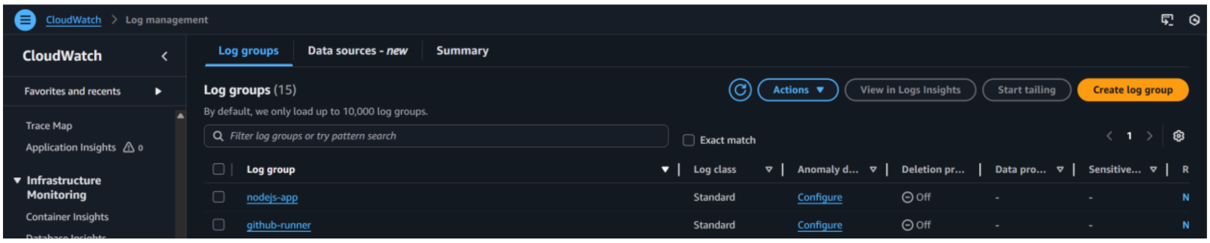
7. CloudWatch Logs Verification and Observability

This section verifies that application and CI/CD logs are successfully published to Amazon CloudWatch Logs from all EC2 instances.

7.1 CloudWatch Log Groups Verification

Navigate to the **CloudWatch Console → Logs → Log groups**.
Confirm the presence of the following log groups created by the CloudWatch Agent:

- **nodejs-app** – Logs from Application EC2 instances
- **github-runner** – Logs from GitHub Actions runner EC2 instances



Log groups (15)

By default, we only load up to 10,000 log groups.

□	Log group	Log class	Anomaly d...	Deletion pr...	Data pro...	Sensitive...	R
□	nodejs-app	Standard	Configure	⊖ Off	-	-	N
□	github-runner	Standard	Configure	⊖ Off	-	-	N

The existence of these log groups confirms that the CloudWatch Agent is installed, configured, and running successfully on both instance types.

7.2 Application EC2 Log Group (nodejs-app)

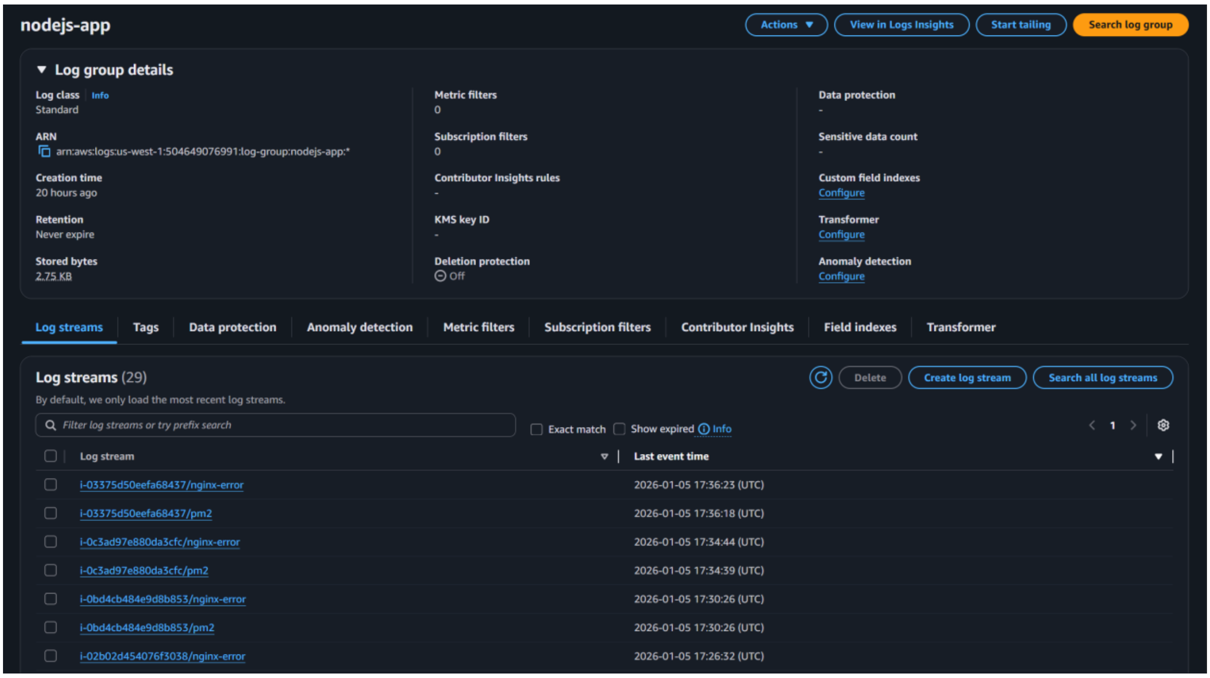
Select the **nodejs-app** log group and review the log streams.

Verification checks:

- Multiple log streams exist, each prefixed with an EC2 **instance ID**
- Recent timestamps indicate active log ingestion
- Log streams are created dynamically as instances are added or replaced

Log stream types:

- **{instance_id}/pm2**: Node.js application runtime logs managed by PM2
- **{instance_id}/nginx-access**: Incoming HTTP request logs from Nginx
- **{instance_id}/nginx-error**: Nginx error and proxy failure logs



nodejs-app

Log group details

Log class: Standard

ARN: arn:aws:logs:us-west-1:504649076991:log-group:nodejs-app:*

Creation time: 20 hours ago

Retention: Never expire

Stored bytes: 2.75 KB

Metric filters: 0

Subscription filters: 0

Contributor Insights rules: -

KMS key ID: -

Deletion protection: Off

Data protection: -

Sensitive data count: -

Custom field indexes: Configure

Transformer: Configure

Anomaly detection: Configure

Log streams (29)

By default, we only load the most recent log streams.

□ Exact match □ Show expired

□	Log stream	Last event time
□	i-03375d50eefa68437/nginx-error	2026-01-05 17:36:23 (UTC)
□	i-03375d50eefa68437/pm2	2026-01-05 17:36:18 (UTC)
□	i-0c3ad97e880da3cfc/nginx-error	2026-01-05 17:34:44 (UTC)
□	i-0c3ad97e880da3cfc/pm2	2026-01-05 17:34:39 (UTC)
□	i-0bd4cb484e9d8b853/nginx-error	2026-01-05 17:30:26 (UTC)
□	i-0bd4cb484e9d8b853/pm2	2026-01-05 17:30:26 (UTC)
□	i-02b02d454076f3038/nginx-error	2026-01-05 17:26:32 (UTC)

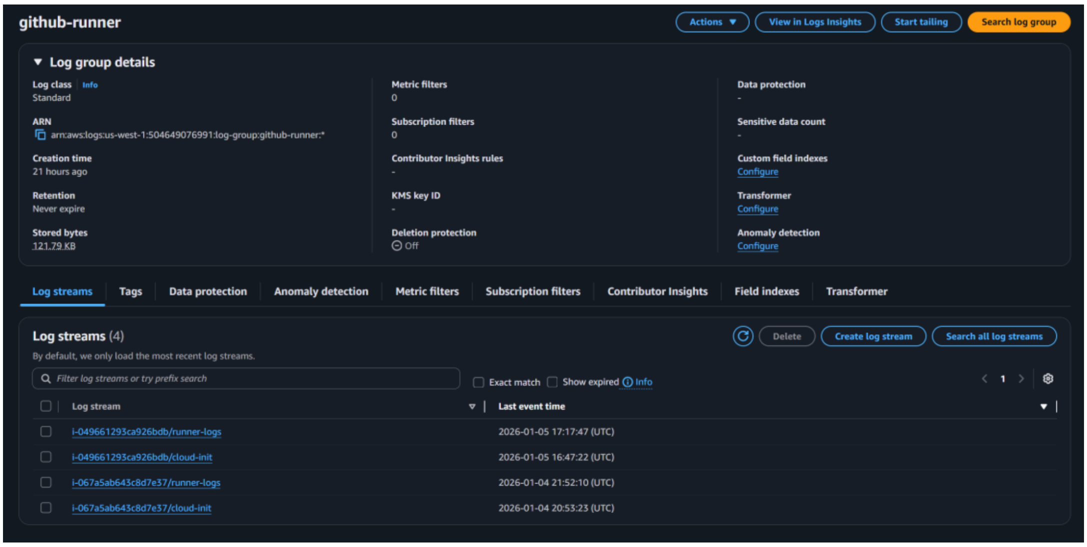
This confirms centralized logging across all application instances, including those launched during Auto Scaling or rolling deployments.

7.3 GitHub Runner Log Group (github-runner)

Select the **github-runner** log group and review the log streams.

Log stream types:

- **{instance_id}/runner-logs**: GitHub Actions runner job execution logs
- **{instance_id}/cloud-init**: EC2 user data and bootstrap execution logs



github-runner

Log group details

Log class: Standard

ARN: arn:aws:logs:us-west-1:504649076991:log-group:github-runner:*

Creation time: 21 hours ago

Retention: Never expire

Stored bytes: 121.79 KB

Metric filters: 0

Subscription filters: 0

Contributor Insights rules: -

KMS key ID: -

Deletion protection: Off

Data protection: -

Sensitive data count: -

Custom field indexes: Configure

Transformer: Configure

Anomaly detection: Configure

Log streams (4)

By default, we only load the most recent log streams.

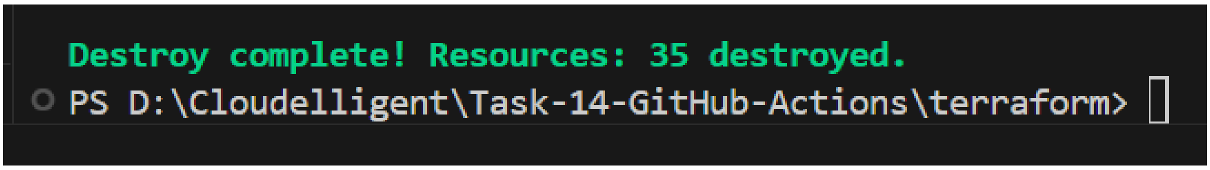
□ Exact match □ Show expired

□	Log stream	Last event time
□	i-049661293ca926bdb/runner-logs	2026-01-05 17:17:47 (UTC)
□	i-049661293ca926bdb/cloud-init	2026-01-05 16:47:22 (UTC)
□	i-067a5ab643c8d7e37/runner-logs	2026-01-04 21:52:10 (UTC)
□	i-067a5ab643c8d7e37/cloud-init	2026-01-04 20:53:23 (UTC)

This validates that CI/CD execution and instance initialization events are centrally captured for troubleshooting and audit purposes.

8. Clean Up

The ***terraform destroy -auto-approve*** command was executed to remove all AWS resources created by Terraform. This ensures that no infrastructure is left running, helping prevent unnecessary costs. The successful completion message confirms that all **35** resources were **destroyed**, as shown in the screenshot.



```
Destroy complete! Resources: 35 destroyed.
PS D:\Cloudelligent\Task-14-GitHub-Actions\terraform>
```

9. Troubleshooting

This section documents issues encountered during Terraform infrastructure provisioning and GitHub Actions CI/CD execution, along with their root causes and applied resolutions.

Issue 1: Invalid BASE64 Encoding of User Data in Launch Template

Problem

Terraform failed while creating the EC2 launch template with the error:

InvalidUserData.Malformed: Invalid BASE64 encoding of user data

Root Cause

AWS Launch Templates require the user_data field to be BASE64-encoded. Plain text user data, even when generated using templatefile(), is not accepted for launch templates.

Solution

Wrapped the user data in base64encode() when defining the aws_launch_template resource.
Plain text user data remains valid for aws_instance resources only.

Issue 2: 504 Gateway Timeout When Accessing Application via ALB

Problem

Accessing the application through the ALB DNS returned a 504 Gateway Timeout error.

Root Cause

Errors in the application EC2 user data script caused the repository clone and AWS CLI installation steps to fail, preventing the application from starting successfully.

Solution

Corrected the user data script syntax and commands, ensuring the repository is cloned properly and required dependencies are installed during instance initialization.

Issue 3: GitHub Self-hosted Runner Not Active

Problem

The GitHub self-hosted runner was not running and did not appear as an active process on the runner EC2 instance.

Root Cause
The actions-runner directory was owned by the SSM user, preventing the ubuntu user from executing runner services.

Solution
Updated directory ownership using the following command:

```
sudo chown -R ubuntu:ubuntu /home/ubuntu/actions-runner
```

This allowed the runner service to start and register successfully.

Issue 4: CI/CD Pipeline Failed with npm: command not found

Problem
The GitHub Actions workflow failed during execution with:

```
npm: command not found
```

Root Cause
Node.js and npm were not installed on the self-hosted runner EC2 instance.

Solution
Installed Node.js and npm via the runner EC2 user data script, ensuring the runner environment supports Node.js-based workflows.