AutoScaleGuard

Production-Grade Highly Available Web Application with Auto Scaling, Monitoring & Alerting on AWS

NAME: NADELLA SRILEKYA

BATCH: 06

Project Overview

AutoScaleGuard is a production-grade AWS project that demonstrates how to build a **highly available, self-healing web application** using **Auto Scaling**, **load balancing**, **monitoring**, and **alerting**.

The application automatically:

- Scales out during high CPU load
- Scales in when load decreases
- Monitors system health in real time
- Sends alert notifications to humans

This project follows **real production best practices** and is suitable for **DevOps / Cloud Engineer roles**.

Project Objectives

- Deploy a **highly available web application** on AWS
- Implement **automatic scaling** based on CPU utilization
- Monitor infrastructure using **CloudWatch**
- Send **email alerts** using SNS
- Validate scaling behavior using **stress testing**
- Visualize system behavior via **CloudWatch Dashboard**

Why This Project? (Problem It Solves)

Problems in Traditional Systems

- Fixed number of servers → cannot handle traffic spikes

- Manual scaling → slow and error-prone
- No visibility → hard to detect performance issues
- No alerts → failures go unnoticed

What AutoScaleGuard Solves

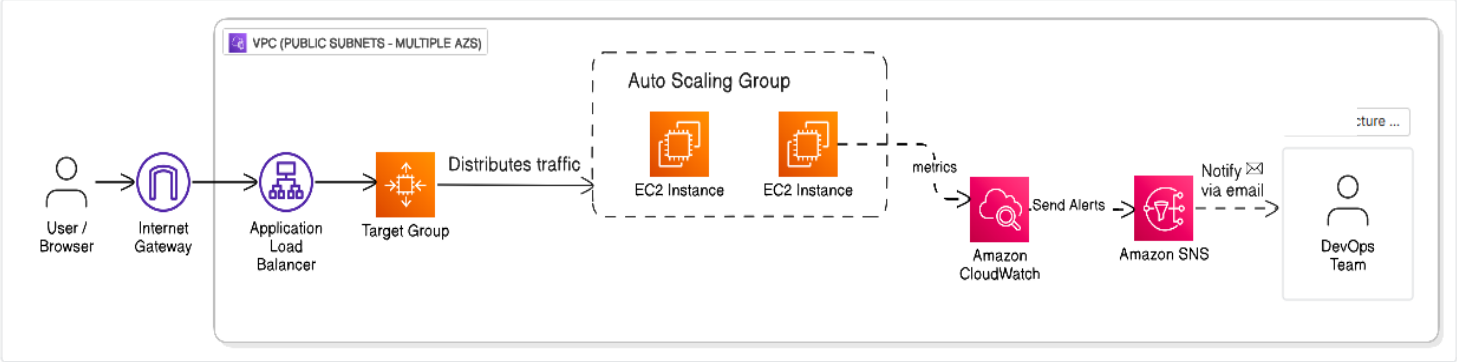
- Handles traffic spikes automatically
- Reduces cost by scaling in when idle
- Provides real-time monitoring
- Sends alerts to engineers
- Ensures high availability across AZs

This is exactly how **modern production systems** are designed.

System Architecture

Architecture Pattern:

Stateless web application with Auto Scaling, Load Balancing, Monitoring & Alerting



Core Components

- Application Load Balancer
- Auto Scaling Group
- EC2 instances
- CloudWatch alarms & dashboard
- SNS email notifications

Request Flow (How the Application Works)

User / Browser

↓

Internet Gateway

↓

Application Load Balancer (HTTP :80)

↓

Target Group

↓

Auto Scaling Group

↓

EC2 Instances

↓

CloudWatch Metrics

├— Scale Out / Scale In (Auto Scaling)

├— Dashboard Visualization

└— SNS Email Alerts

Flow Explanation

1. User sends an HTTP request
2. ALB distributes traffic to healthy EC2 instances
3. EC2 serves the web page
4. CPU metrics are sent to CloudWatch
5. Auto Scaling reacts to CPU thresholds
6. Alerts are sent to email via SNS

AWS Services Used & Their Purpose

1. Amazon EC2

- Hosts the web application
- Uses Amazon Linux 2
- Runs Apache HTTP server
- Stress tool used for testing load

2. Application Load Balancer (ALB)

- Distributes traffic across instances
- Ensures high availability
- Health checks unhealthy instances

3. Auto Scaling Group (ASG)

- Maintains desired number of EC2 instances
- Automatically scales based on CPU utilization
- Ensures fault tolerance

4. Launch Template

- Defines EC2 configuration
- Ensures consistency across instances
- Automates application setup via user data

5. Amazon CloudWatch

- Monitors CPU utilization
- Triggers scaling alarms
- Displays metrics via dashboard

6. Amazon SNS

- Sends email notifications
- Alerts humans when thresholds are breached

Step-by-Step Implementation

Step 1: Create Security Groups

Security Groups act as virtual firewalls controlling inbound and outbound traffic to AWS resources.

1.1 Create Application Load Balancer Security Group (alb-sg)

Purpose:
Allows public HTTP traffic to reach the Application Load Balancer.

Navigation:
EC2 → Security Groups → Create Security Group

Configuration:

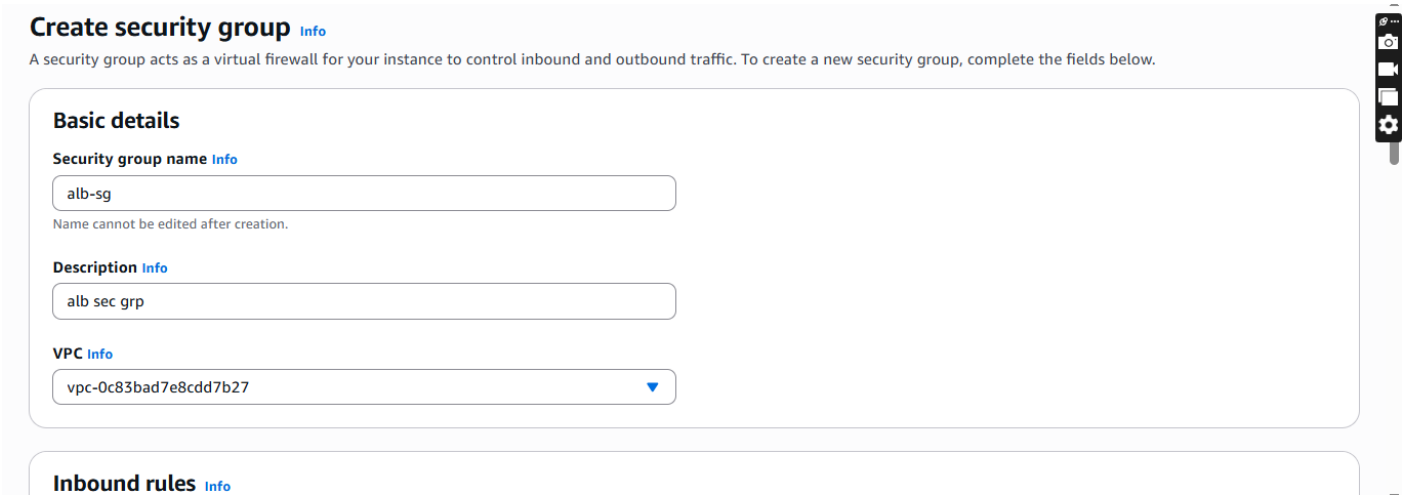
- **Name:** alb-sg
- **Description:** Security group for Application Load Balancer
- **VPC:** Default VPC

Inbound Rules:

Type	Protocol	Port	Source
HTTP	TCP	80	0.0.0.0/0

Outbound Rules:

- Allow all traffic to 0.0.0.0/0



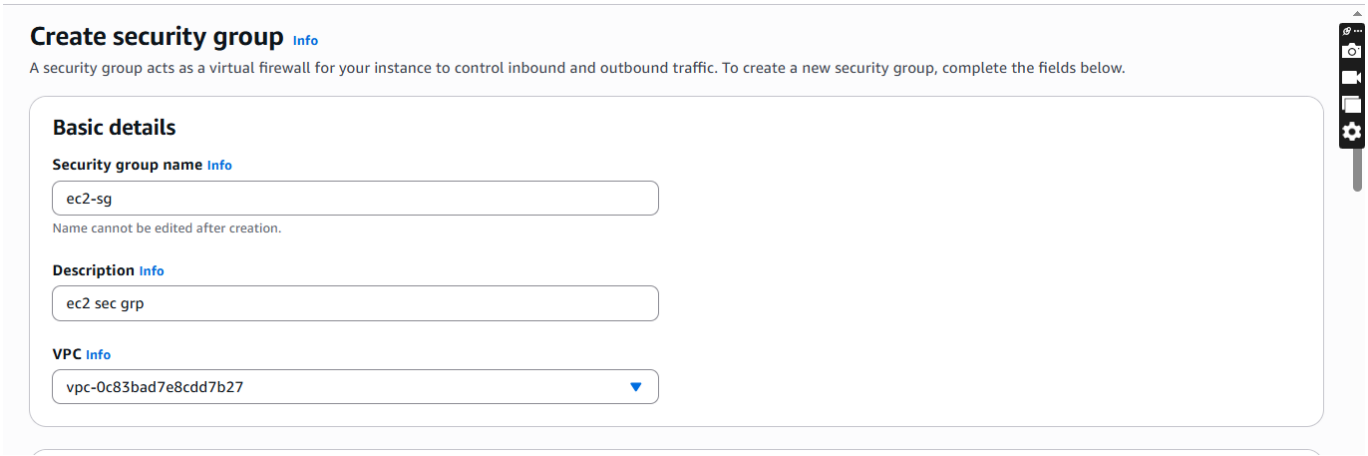
1.2 Create EC2 Instance Security Group (ec2-sg)

Purpose:
Restricts direct public access to EC2 instances and allows traffic only from the ALB.

Navigation:
EC2 → Security Groups → Create Security Group

Configuration:

- **Name:** ec2-sg
- **Description:** Security group for EC2 instances
- **VPC:** Default VPC

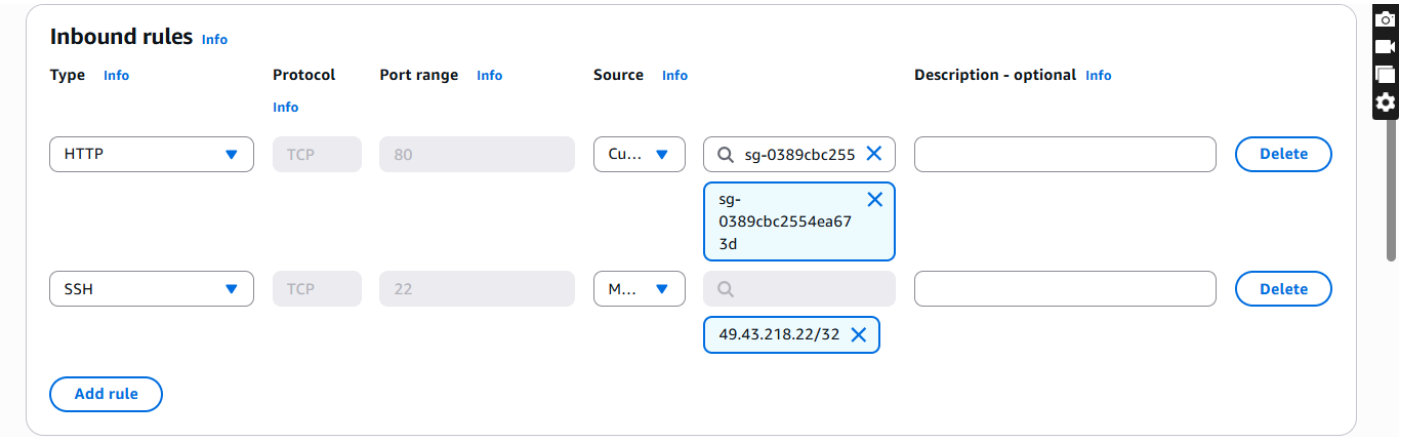


Inbound Rules:

Type	Protocol	Port	Source
HTTP	TCP	80	alb-sg
SSH	TCP	22	Your Public IP

Outbound Rules:

- Allow all traffic



EC2 instances are **not exposed to the internet** directly — all traffic flows through ALB.

Step 2: Create Launch Template

A Launch Template defines how EC2 instances are launched in the Auto Scaling Group.

Navigation:
EC2 → Launch Templates → Create Launch Template

Configuration:

- **Launch template name:** autoscaleguard-lt
- **AMI:** Amazon Linux 2
- **Instance type:** t2.micro
- **Key pair:** Select or create a key pair
- **Security group:** ec2-sg

User Data Script

```
#!/bin/bash

yum update -y

yum install -y httpd stress
```

systemctl start httpd

systemctl enable httpd

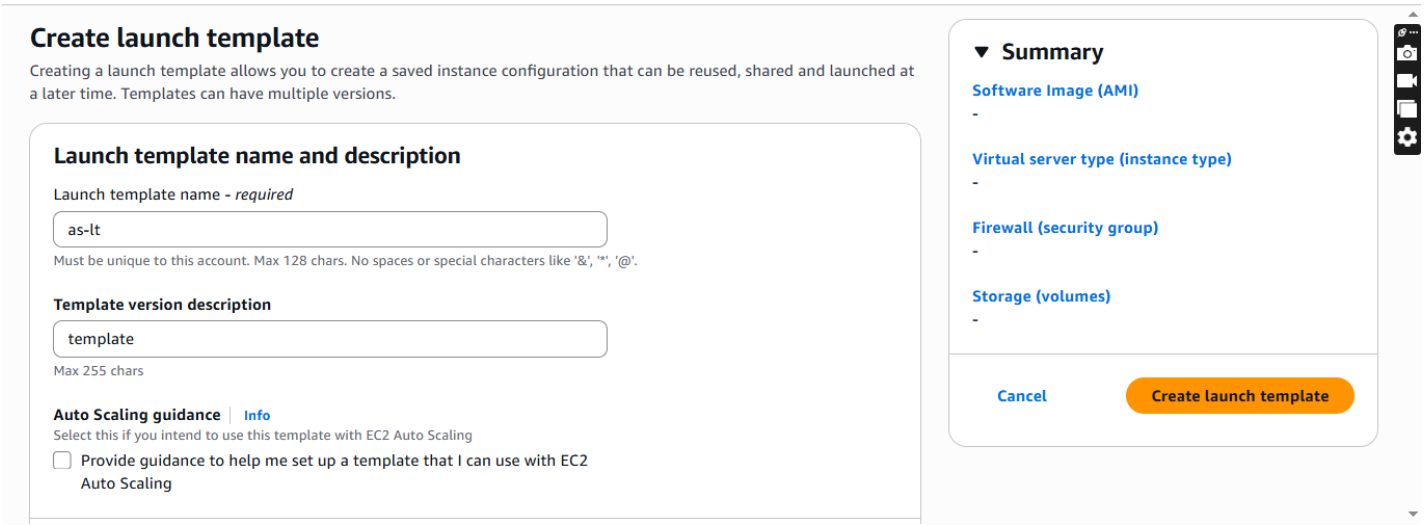
echo "<h1>AutoScaleGuard Demo</h1>" > /var/www/html/index.html

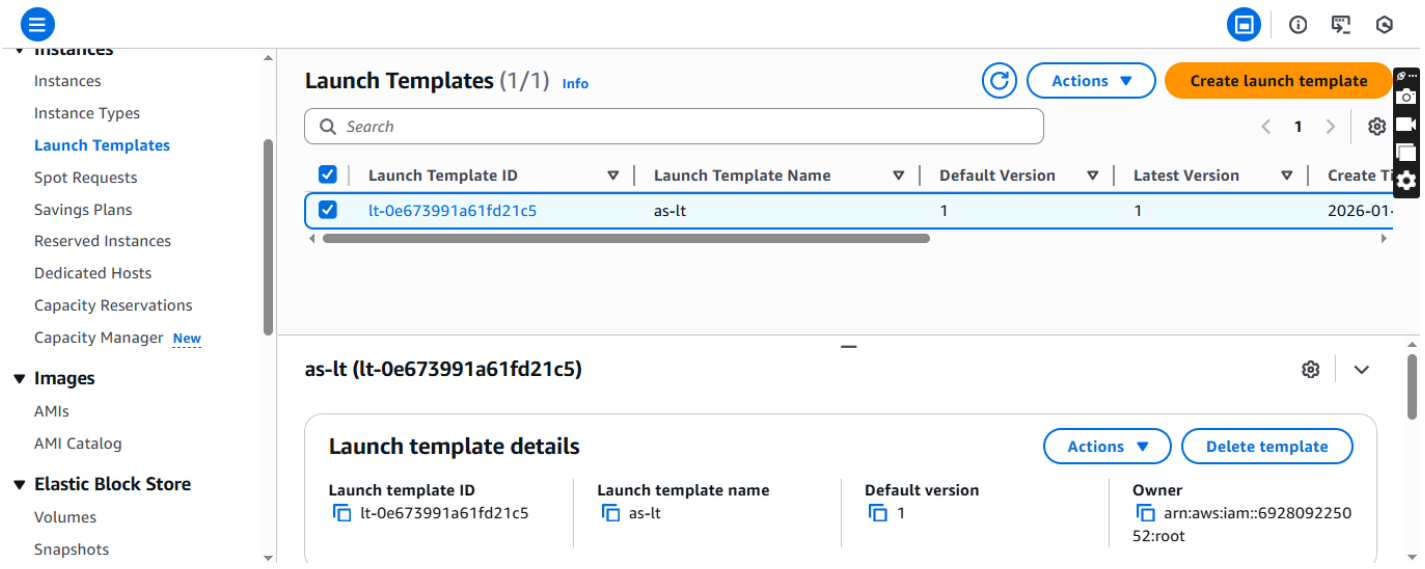
echo " Instance ID: $(curl -s http://169.254.169.254/latest/meta-data/instance-id) " >> /var/www/html/index.html

This ensures **every EC2 configures itself automatically**.

Purpose of User Data:

- Installs Apache web server
- Installs stress tool for load testing
- Automatically starts the web application on boot





STEP 3: Create Target Group

The Target Group defines where the ALB forwards traffic.

Navigation:
EC2 → Target Groups → Create Target Group

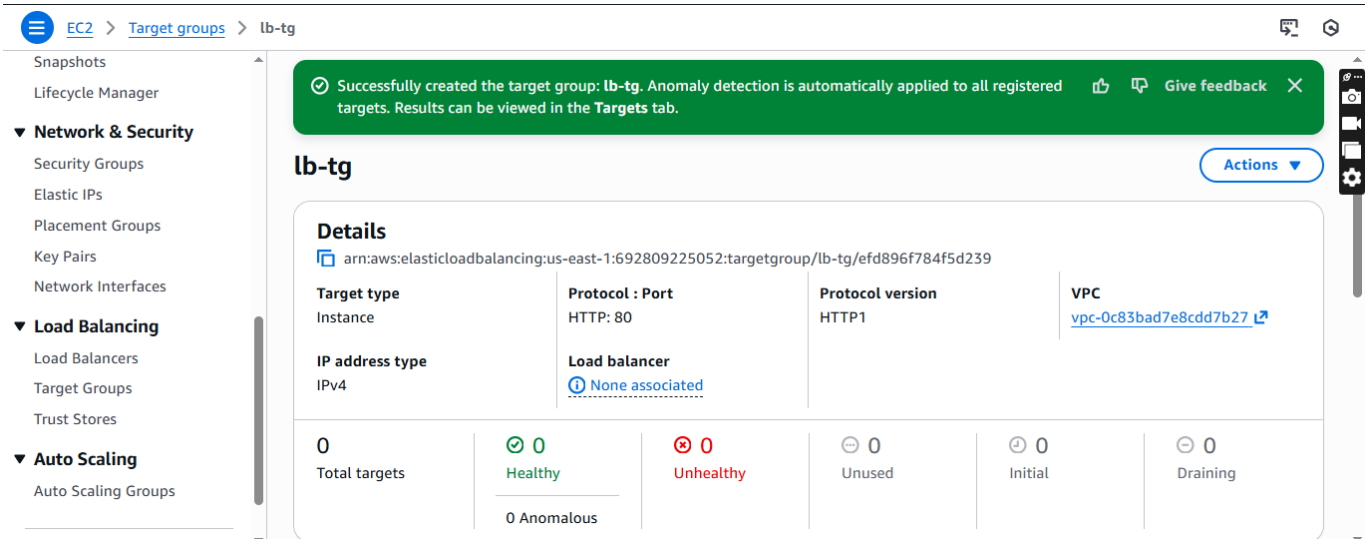
Configuration:

- **Target type:** Instance
- **Protocol:** HTTP
- **Port:** 80
- **VPC:** Default VPC

Health Check Settings:

- **Protocol:** HTTP
- **Path:** /
- **Healthy threshold:** Default

Health checks ensure traffic is routed only to healthy instances.



Step 4: Create Application Load Balancer (ALB)

The Application Load Balancer distributes incoming traffic across EC2 instances.

Navigation:
EC2 → Load Balancers → Create Load Balancer

Configuration:

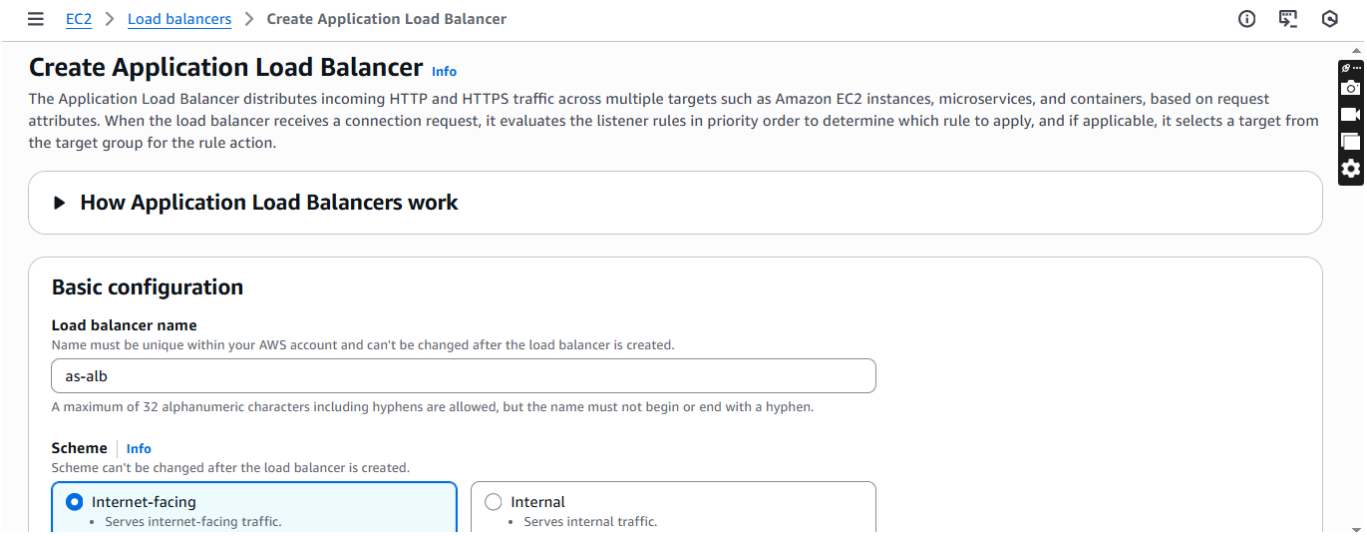
- **Load balancer type:** Application Load Balancer
- **Scheme:** Internet-facing

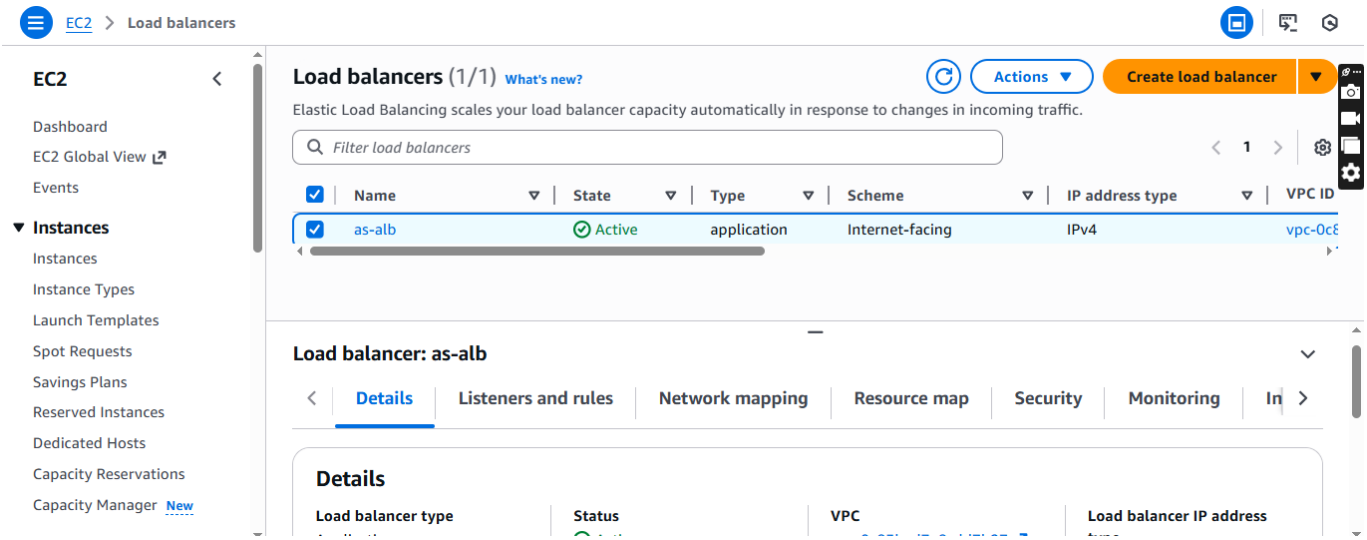
- **IP address type:** IPv4
- **VPC:** Default VPC
- **Subnets:** public-subnet-1, public-subnet-2
- **Security group:** alb-sg

Listener Configuration

- **Protocol:** HTTP
- **Port:** 80
- **Forward to:** Target Group created earlier

The ALB acts as the **single-entry point** for user traffic.





Step 5: Create Auto Scaling Group (ASG)

The Auto Scaling Group ensures the correct number of EC2 instances are always running.

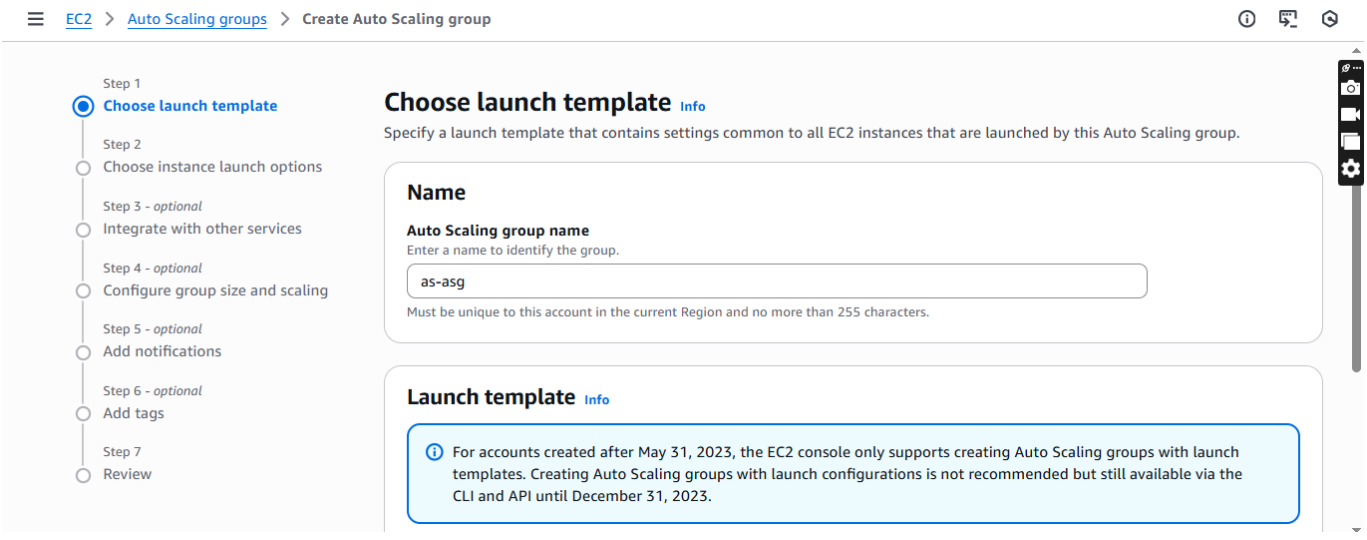
Navigation:
EC2 → Auto Scaling Groups → Create Auto Scaling Group

Configuration:

- **Name:** as-asg
- **Launch template:** as-lt
- **VPC:** Default VPC
- **Subnets:** Same subnets as ALB
- **Attach to load balancer:** Yes
- **Target group:** Previously created target group

Capacity Settings

Setting	Value
Minimum capacity	1
Desired capacity	1
Maximum capacity	3



Step 6: Configure Auto Scaling Policy

Scaling Policy Type: Target Tracking

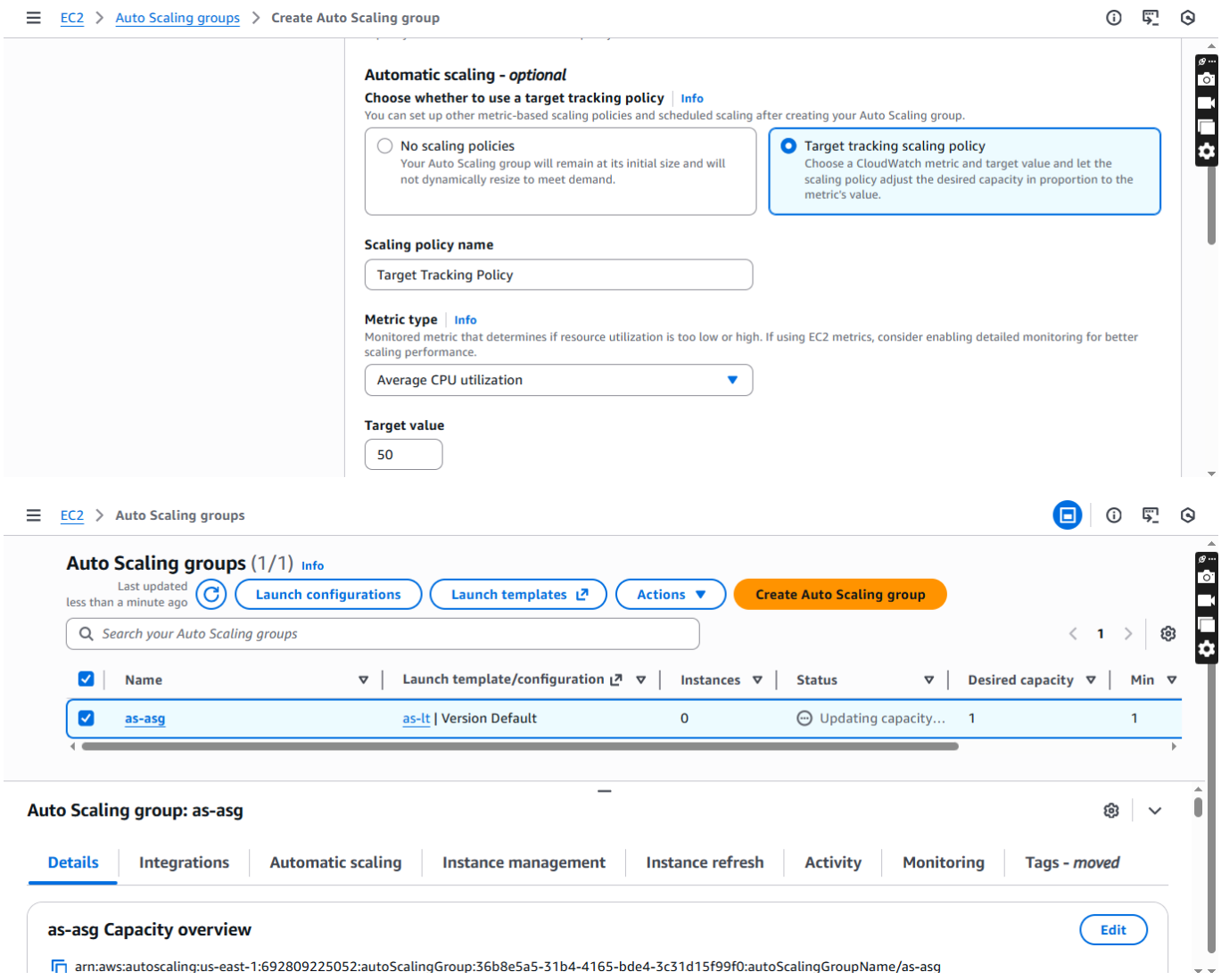
Configuration:

- **Metric:** Average CPU Utilization
- **Target value:** 50%

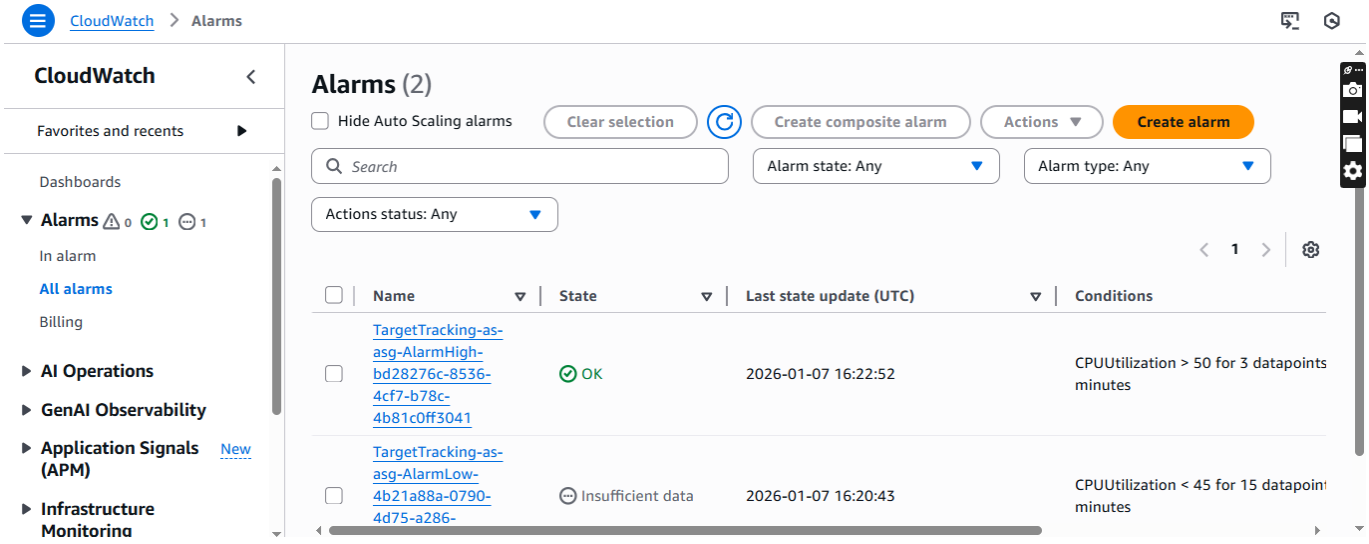
AWS automatically creates:

- Scale-out alarm
- Scale-in alarm

These alarms **must not be deleted**.



Scale-out and Scale-in alarms, which are automatically created by AWS by configuring Auto Scaling Policy



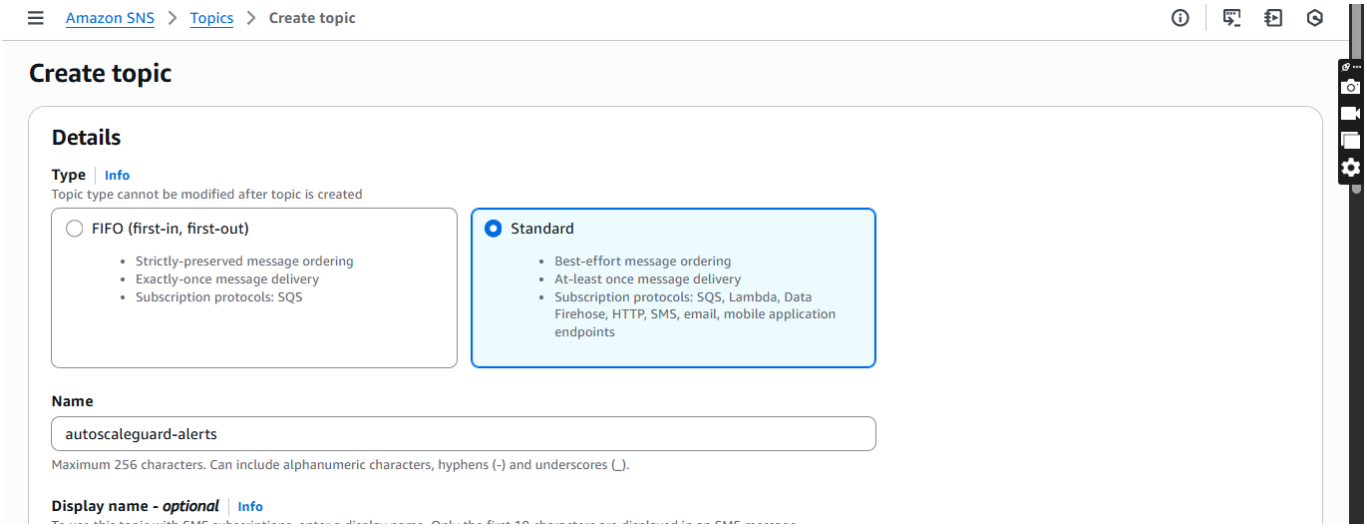
Step 7: Create SNS Topic for Alerts

SNS is used to notify DevOps teams via email when alerts are triggered.

Navigation:
SNS → Topics → Create Topic

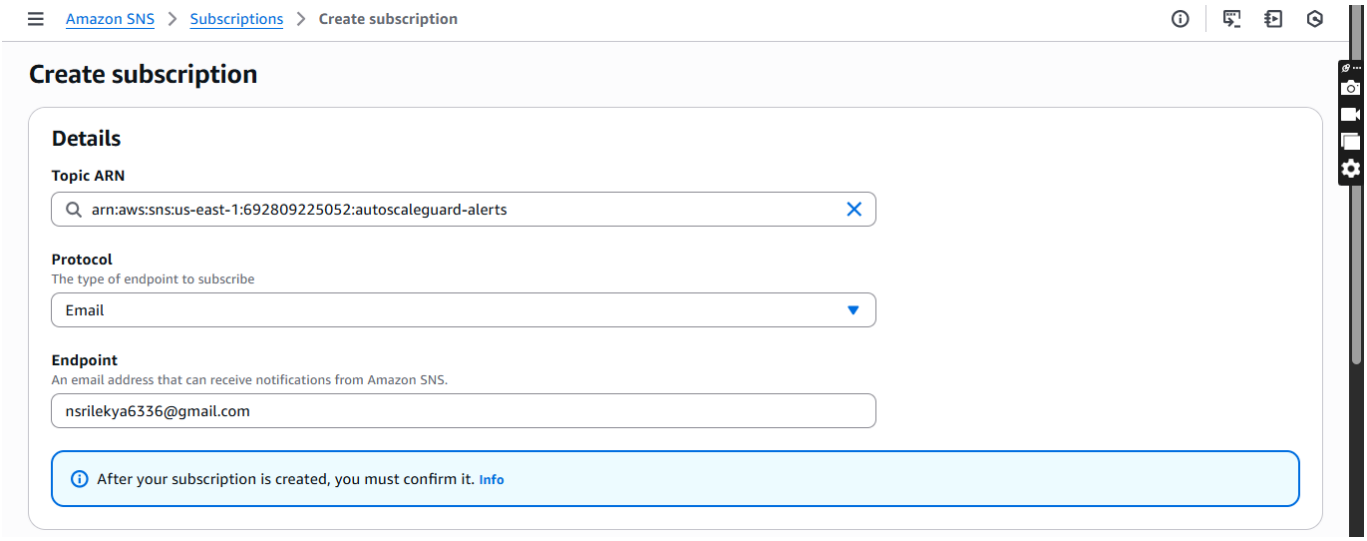
Configuration:

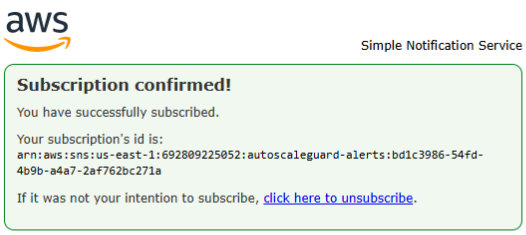
- **Type:** Standard
- **Name:** autoscaleguard-alerts



Create Subscription

- **Protocol:** Email
- **Endpoint:** Your email address
- Confirm the subscription from email





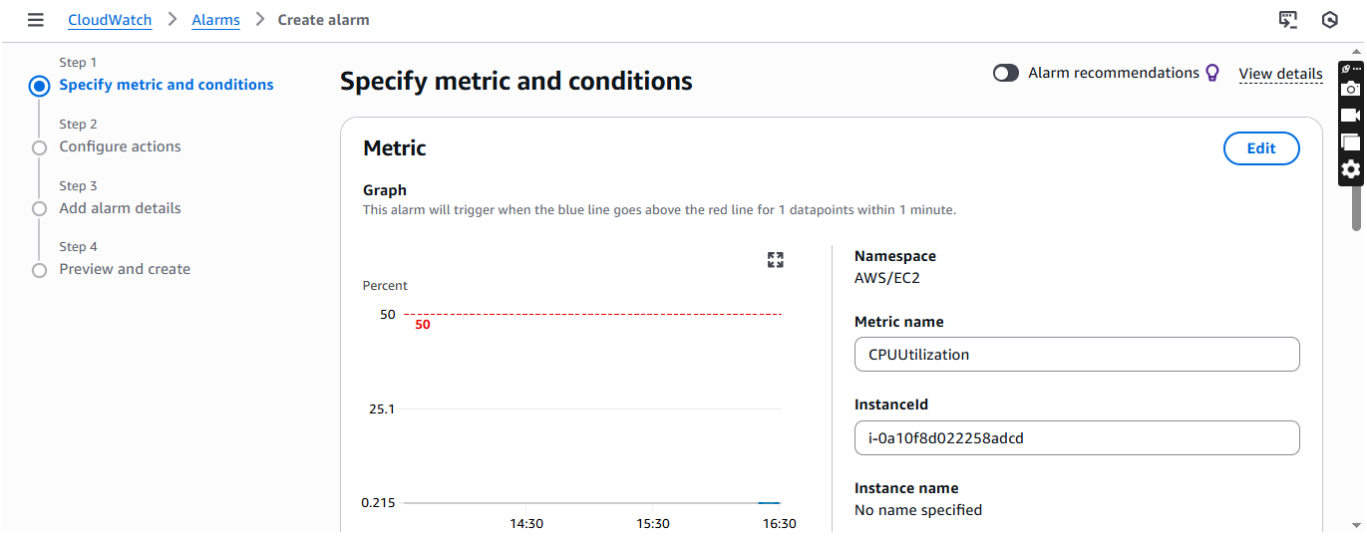
Step 8: Create CloudWatch Alarm (Human Alert)

This alarm is for **human awareness**, not auto scaling.

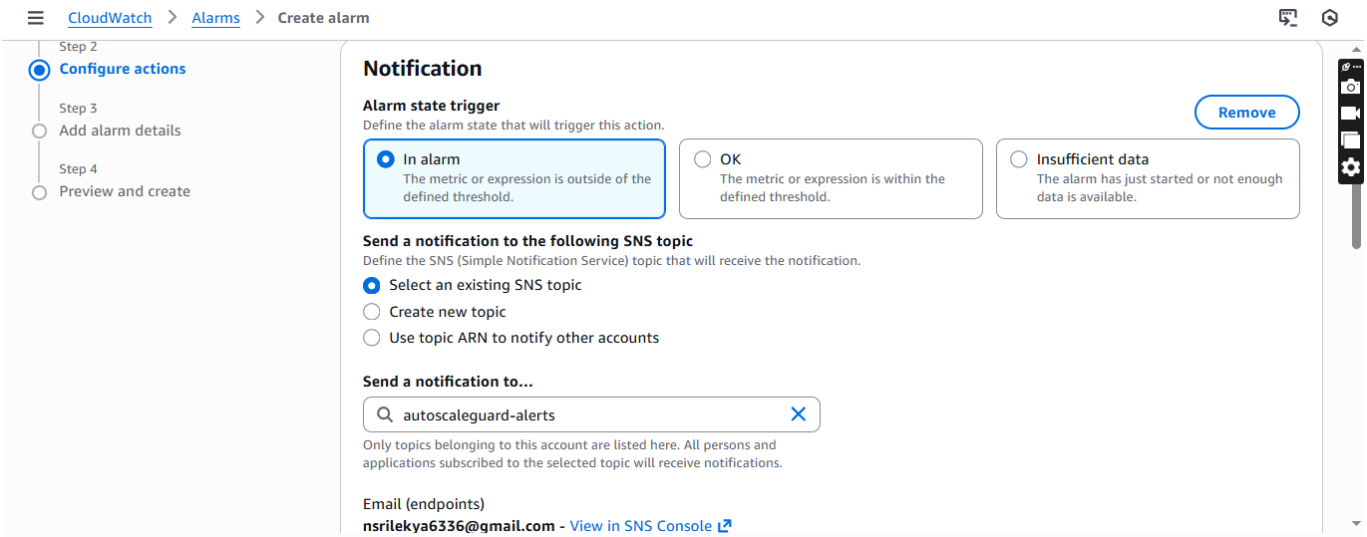
Navigation:
CloudWatch → Alarms → Create Alarm

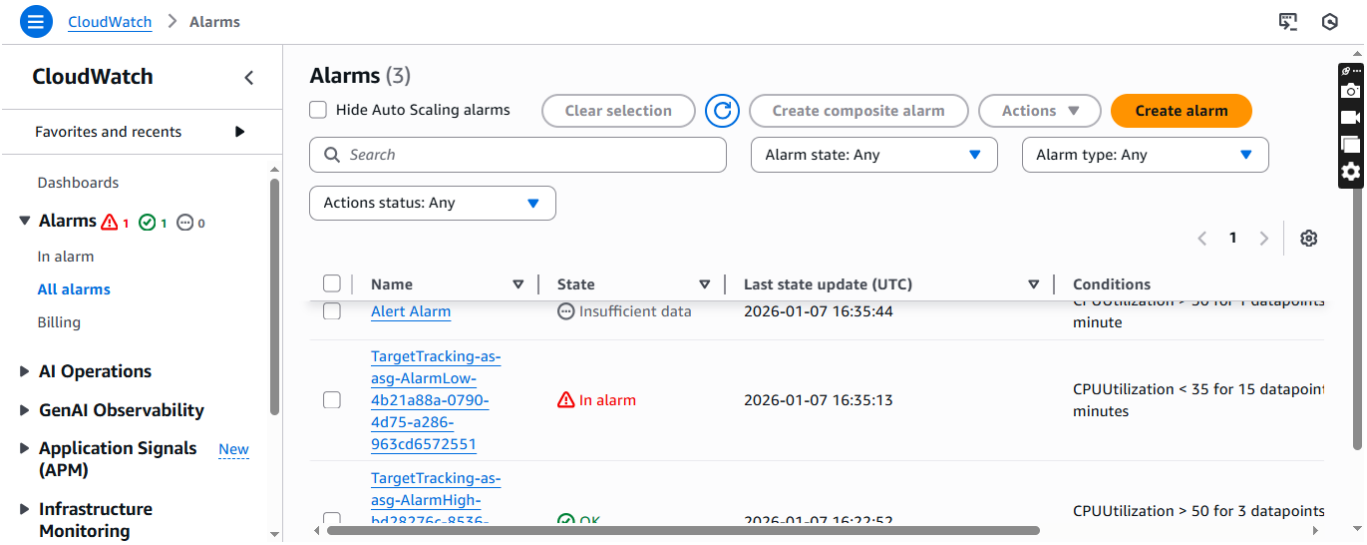
Configuration:

- **Metric:** EC2 → CPUUtilization
- **Statistic:** Average
- **Threshold:** > 70%
- **Period:** 1 minute
- **Evaluation periods:** 2



- **Action:** Notify SNS topic

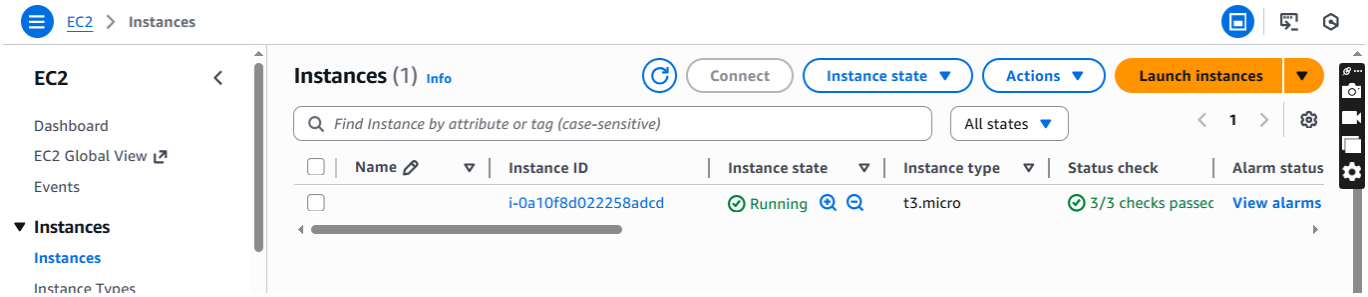




Step 9: Generate Load (Stress Test)

To validate scaling behavior, generate artificial load.

Navigation:
EC2 → Instances → Connect → EC2 Instance Connect



```
hi@DESKTOP-3C1O82Q MINGW64 /d/NSDC JR SD/DEVOPS
$ ssh -i three-tier-key.pem ec2-user@18.212.164.85
The authenticity of host '18.212.164.85 (18.212.164.85)' can't be established.
ED25519 key fingerprint is: SHA256:re9UHlxmCTZAEHK6CF1mX2xEsccvUzjf6LmFm5ueYTw
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '18.212.164.85' (ED25519) to the list of known hosts.
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-82-119 ~]$ sudo yum install stress -y
Last metadata expiration check: 0:21:30 ago on Wed Jan  7 16:21:14 2026.
Package stress-1.0.7-2.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-82-119 ~]$ stress -version
stress: FAIL: [26405] (251) unrecognized option: -version
[ec2-user@ip-172-31-82-119 ~]$ stress --version
stress 1.0.7
[ec2-user@ip-172-31-82-119 ~]$
```

Command:

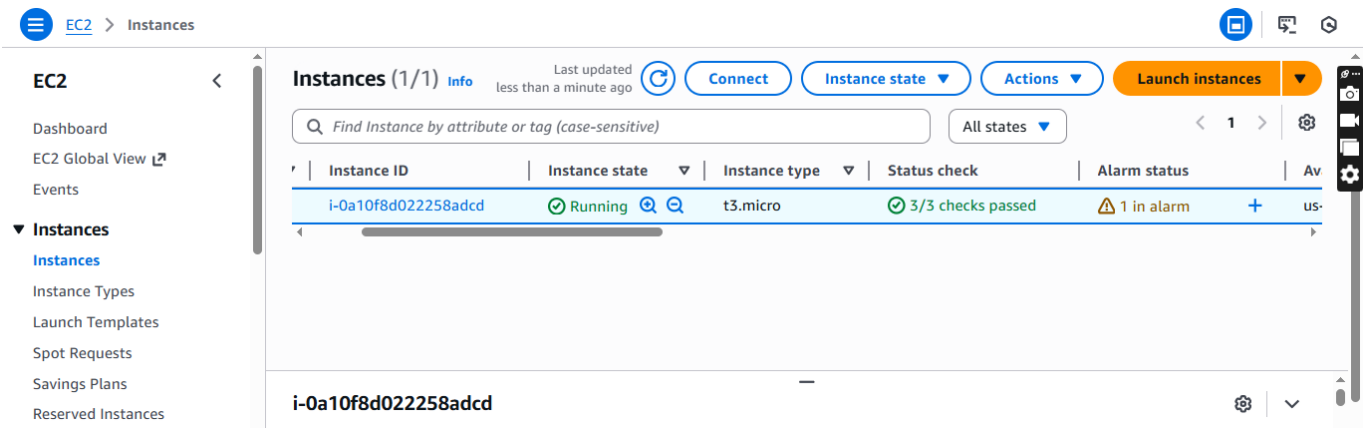
stress --cpu 2 --timeout 1000

```
ec2-user@ip-172-31-82-119:~
[ec2-user@ip-172-31-82-119 ~]$ stress --version
stress 1.0.7
[ec2-user@ip-172-31-82-119 ~]$ stress --cpu 2 --timeout 1000
stress: info: [28244] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

Step 10: Verify Auto Scaling

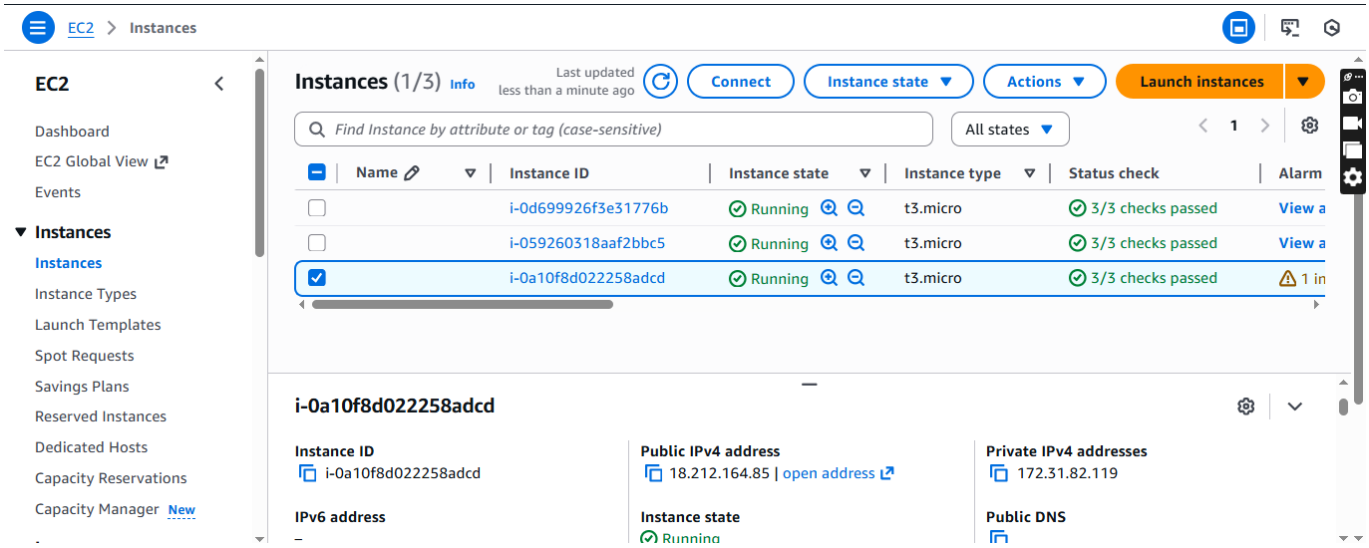
Check Auto Scaling Activity:

- **EC2 → Auto Scaling Groups → Activity**

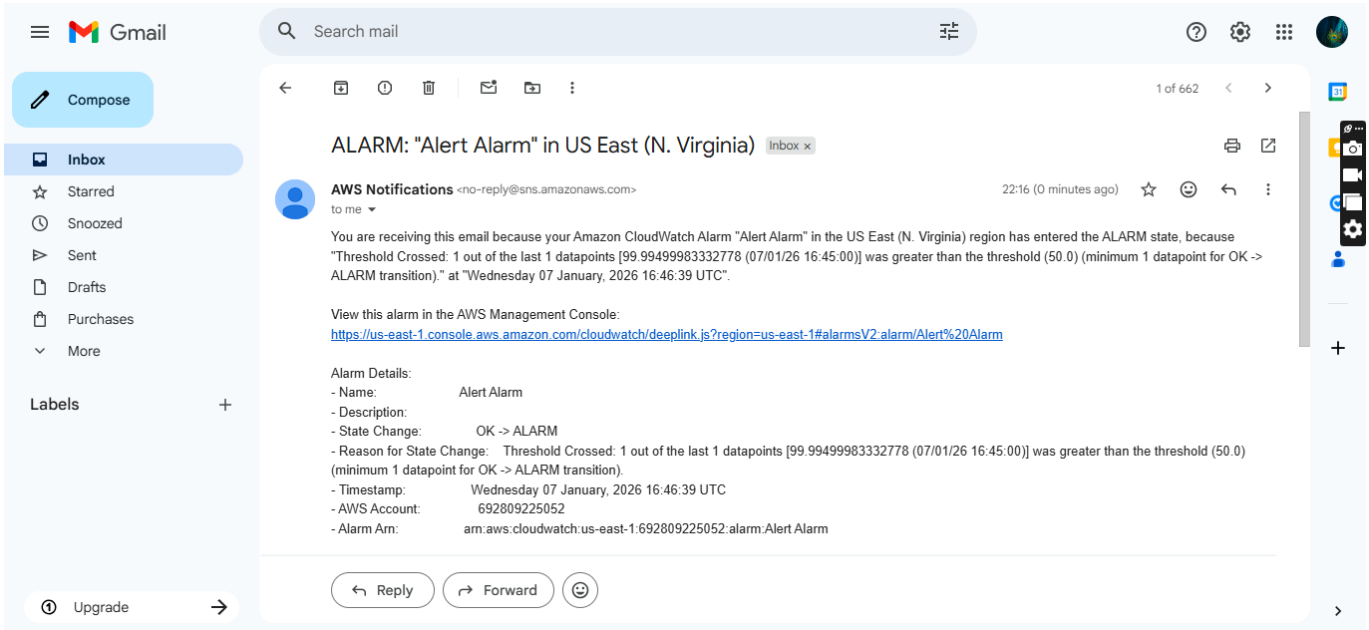


Expected behavior:

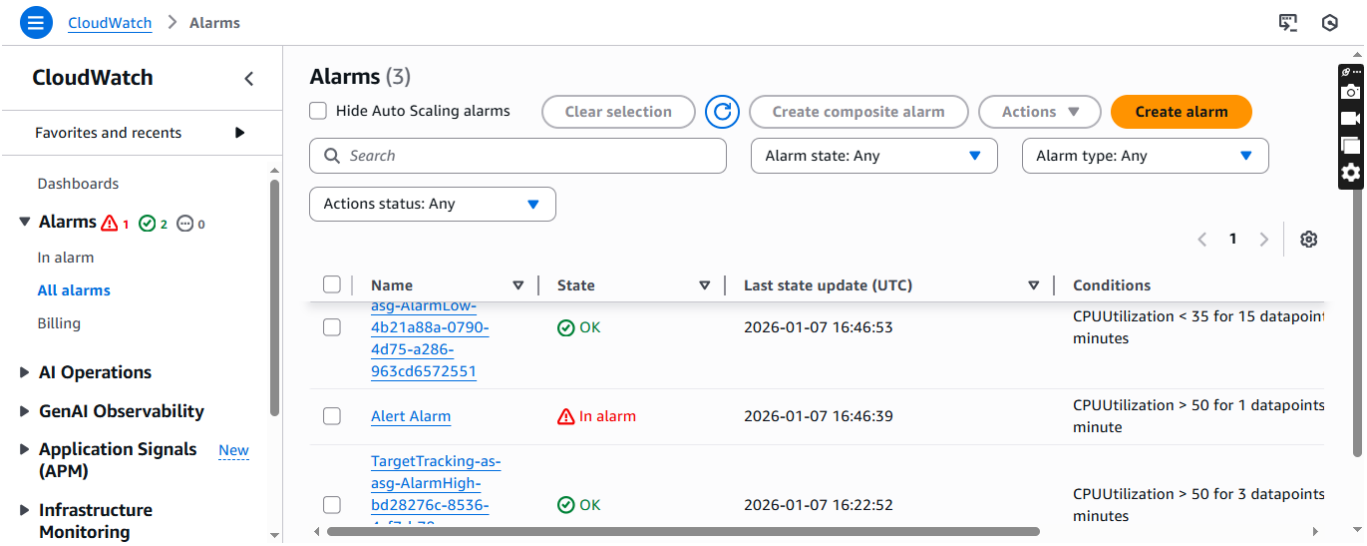
- **New EC2 instance launched**
- **Total instances increase to 2**

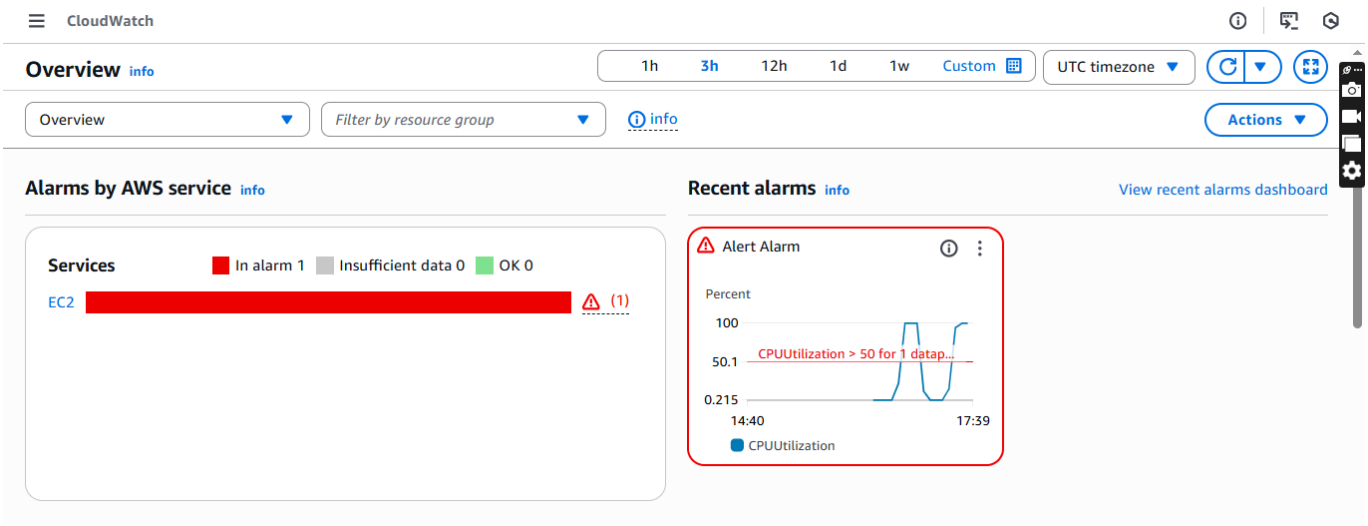


Alert alarm sends notification to email



CloudWatch Alarm States

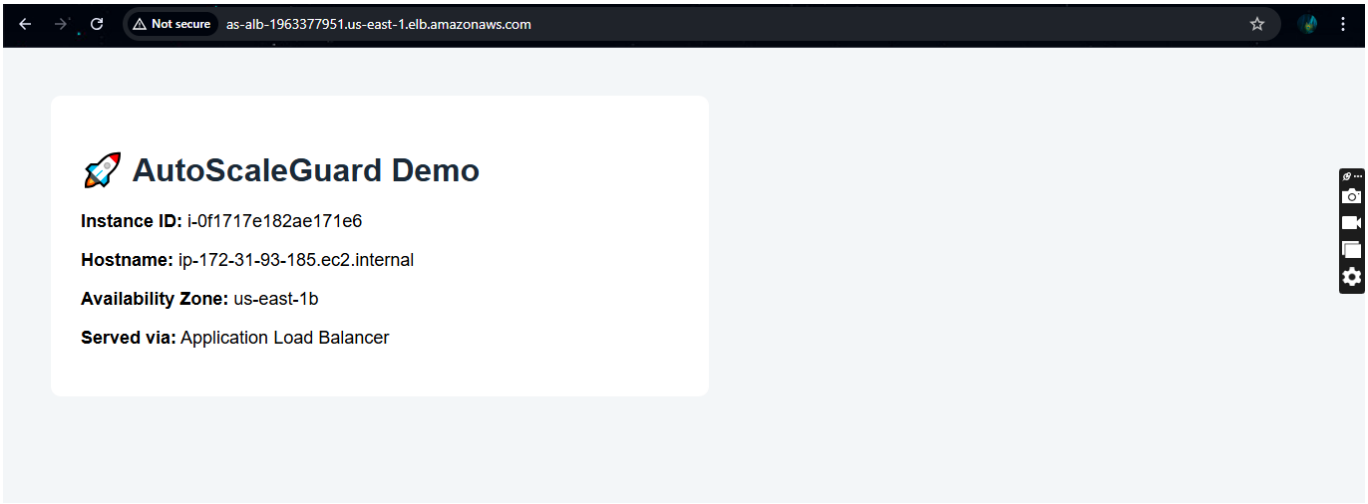




To verify the deployment:

1. Copy the DNS name of the Application Load Balancer

2. Open it in a web browser



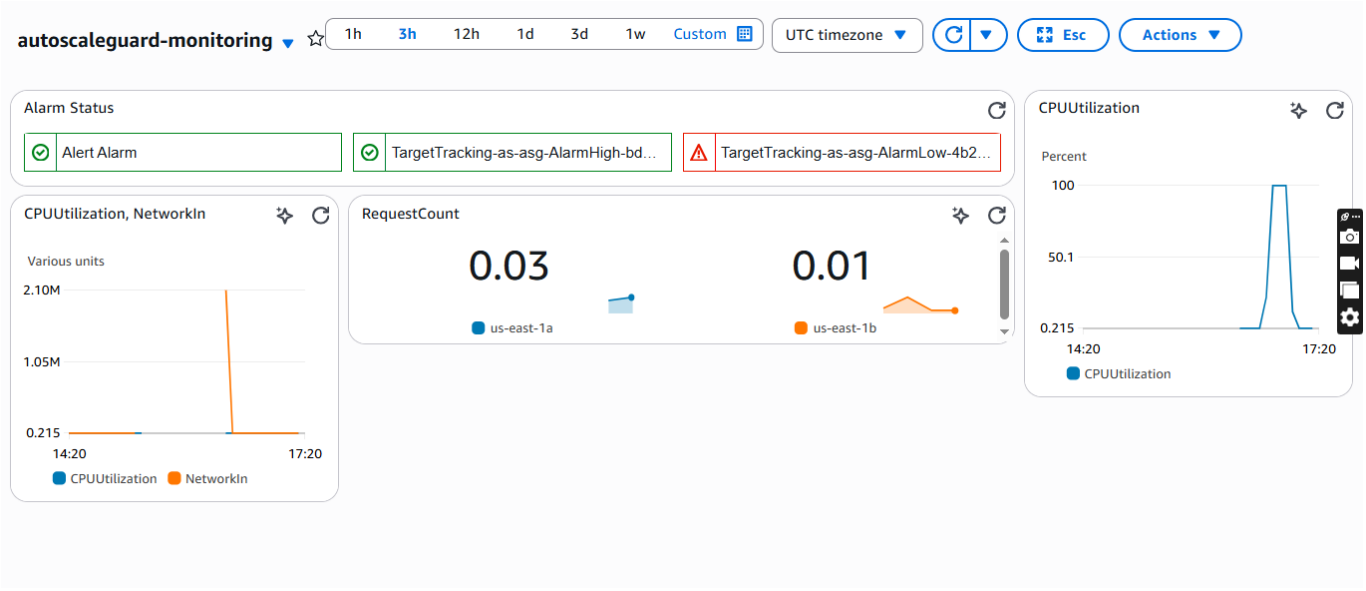
Step 11: Create CloudWatch Dashboard

Navigation:
CloudWatch → Dashboards → Create Dashboard

Dashboard Name: autoscaleguard-monitoring

Widgets

1. Alarm Status (3 alarms)
2. CPU Utilization (per instance)
3. Auto Scaling Metrics:
    - NetworkIn
4. Application Load Balancer Request count



Conclusion

The AutoScaleGuard project demonstrates the design and implementation of a highly available and scalable web application on AWS using native cloud services. By combining an Application Load Balancer, Auto Scaling Group, and EC2 instances, the application automatically adapts to changing traffic conditions.

Amazon CloudWatch provides real-time monitoring and drives scaling decisions based on CPU utilization, while Amazon SNS ensures that DevOps teams are notified promptly when critical thresholds are breached. The use of launch templates and automated instance configuration enables consistency, reliability, and reduced operational overhead.

Overall, this project reflects real-world production patterns and showcases practical experience in building, monitoring, and operating scalable cloud infrastructure.

Key Learnings

- Designed a highly available architecture using ALB and Auto Scaling
- Implemented automatic scale-out and scale-in based on CPU utilization
- Gained hands-on experience with CloudWatch metrics, alarms, and dashboards
- Implemented alerting mechanisms using Amazon SNS
- Learned how to validate scaling behavior using stress testing
- Understood the importance of separating scaling alarms from human alerts
- Followed production best practices for security and networking

Challenges Faced & Troubleshooting

1. CloudWatch Alarms Showing “Insufficient Data”

Issue:
Alarms initially showed insufficient data.

Resolution:
Waited for EC2 metrics to populate and generated CPU load using the stress tool to trigger metric collection.

2. Instance ID Not Displaying on Web Page

Issue:
Instance metadata was not accessible in the initial user data script.

Resolution:
Updated the script to use IMDSv2 tokens to securely retrieve instance metadata.

4. No Scale-Out During Load Test

Issue:
Scaling did not occur as expected during initial tests.

Resolution:
Increased CPU load duration and confirmed scaling policy thresholds and evaluation periods.

Future Enhancements

- Enable **HTTPS** using AWS Certificate Manager (ACM)
- Implement **Infrastructure as Code** using Terraform or CloudFormation
- Add **custom application metrics** to CloudWatch
- Integrate alerts with **Slack or PagerDuty**
- Introduce **blue-green or rolling deployments**
- Enhance security using **IAM roles and least-privilege policies**