

Project 1: Multi-Tier Web Application with Auto-Scaling

Project Report

Author: Gurdeep Singh

Date: December 2025

1. Introduction

This project demonstrates the implementation of a **highly available, fault-tolerant, and scalable web application architecture** on AWS using:

- EC2 Auto Scaling Group (ASG)
- Application Load Balancer (ALB)
- Target Groups
- CloudWatch Monitoring
- Scaling Policies
- Apache JMeter Load Testing

The primary objective was to validate the **scale-out** and **scale-in** behaviour of the Auto Scaling Group under simulated heavy traffic.

2. Architecture Overview

A scalable architecture was deployed where the ALB routes traffic across EC2 instances managed by the ASG. CloudWatch monitors performance metrics and triggers automatic scaling events.

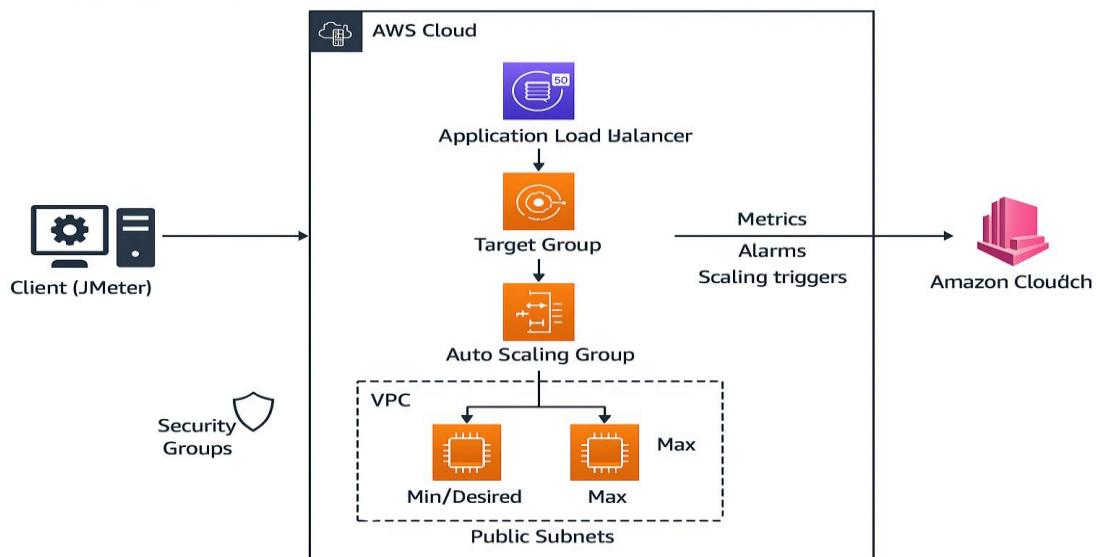


Fig: Architecture Diagram

3. AWS Components Used

3.1 EC2 Launch Template

- Configured with Amazon Linux 2 AMI
- Bootstrapped using user data to install Apache web server
- Created a custom HTML page: "Gurdeep-ASG Web Server"

The screenshot shows the AWS EC2 Launch Template details page for a launch template named 'Gurdeep-WebServer-LT'. The page includes sections for 'Launch template details' and 'Launch template version details'. In the 'Launch template details' section, the launch template ID is 'lt-02de4123c86ca9790', the name is 'Gurdeep-WebServer-LT', the default version is '1', and the owner is 'am:awsiam:339712988423:user/Deep'. The 'Launch template version details' section shows the version '1 (Default)', description 'Fixed auto assigned ip', date created '2025-11-27T15:44:53.000Z', and created by 'am:awsiam:339712988423:user/Deep'. Configuration details include AMI ID 'ami-0fa3fe0fa7920f68e', instance type 't3.micro', availability zone 'us-east-1a', and security group 'sg-0a1a4b43f20839ae'. The left sidebar shows navigation links for EC2, Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, Network & Security, and Load Balancing.

The screenshot shows the 'Launch Template settings' page for the same launch template. It lists various configuration options grouped into columns: IAM instance profile, Instance auto-recovery, Shutdown behavior, Purchasing option; Stop-hibernate behavior, Termination protection, Stop protection, Hostname type; Resource-based IPv4 DNS, Resource-based IPv6 DNS, Detailed CloudWatch monitoring, T2/T3 Unlimited; Placement group, Target partition, Capacity reservation preference, Capacity reservation target; EBS optimized instance, Tenancy, Tenancy host resource group, Tenancy host ID; Tenancy affinity, RAM disk ID, Kernel ID, Enclave; License configurations, Core count, Threads per core, Metadata accessible; Token hop limit, Metadata version, Metadata IPv6 endpoint, Allow tags in metadata; Instance Configurable Bandwidth, User data, and a preview of the user data script. The user data script contains the following code:

```
#!/bin/bash
sudo dnf install -y httpd
sudo systemctl start httpd
sudo systemctl enable httpd
echo "<h1>Gurdeep-ASG Web Server</h1>" | sudo tee /var/www/html/index.html
```

A note at the bottom states: 'Base64-encoded user data has been decoded for readability.'

Launch Template settings

3.2 Auto Scaling Group (ASG)

- Minimum capacity: 1

- Desired capacity: **1**
- Maximum capacity: **3**
- Subnets: **us-east-1a, us-east-1b**

3.3 Application Load Balancer (ALB)

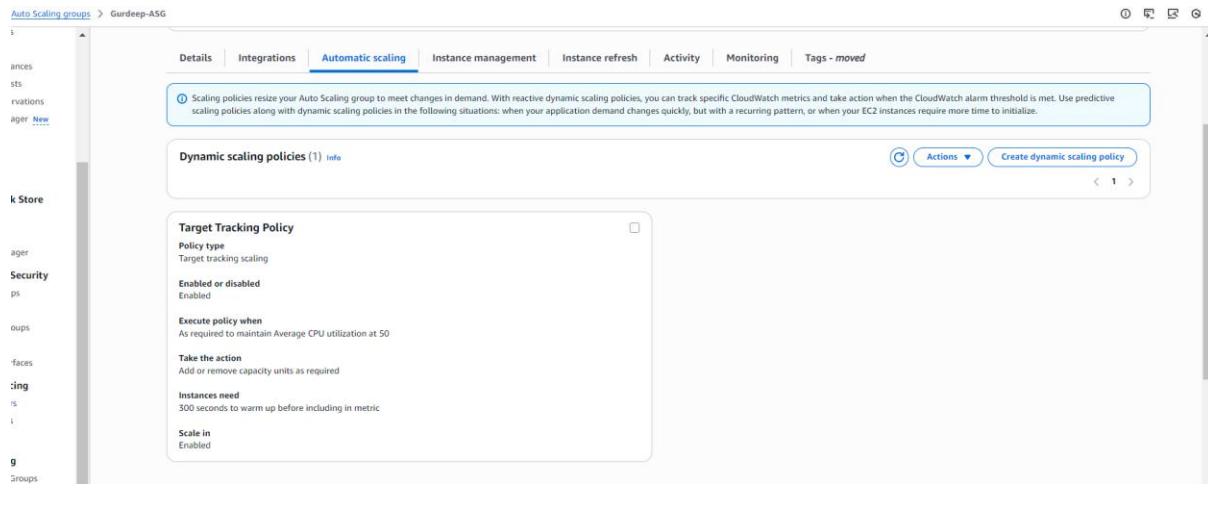
- Internet-facing
- Listener: **HTTP 80**
- Default action: Forward to Target Group

3.4 Target Group

- Protocol: **HTTP**
- Port: **80**
- Health checks: **/**

3.5 Scaling Policies

- **Scale Out:** CPU > 50%
- **Scale In:** CPU < 20%
- Instance warmup: 300 seconds



The screenshot shows the AWS Auto Scaling Groups console for the group 'Gurdeep-ASG'. The 'Automatic scaling' tab is selected. A 'Dynamic scaling policies' section contains one policy named 'Target Tracking Policy'. The policy configuration includes:

- Policy type: Target tracking scaling
- Enabled or disabled: Enabled
- Execute policy when: As required to maintain Average CPU utilization at 50
- Take the action: Add or remove capacity units as required
- Instances need: 300 seconds to warm up before including in metric
- Scale in: Enabled

A 'Create dynamic scaling policy' button is located at the top right of the policy list.

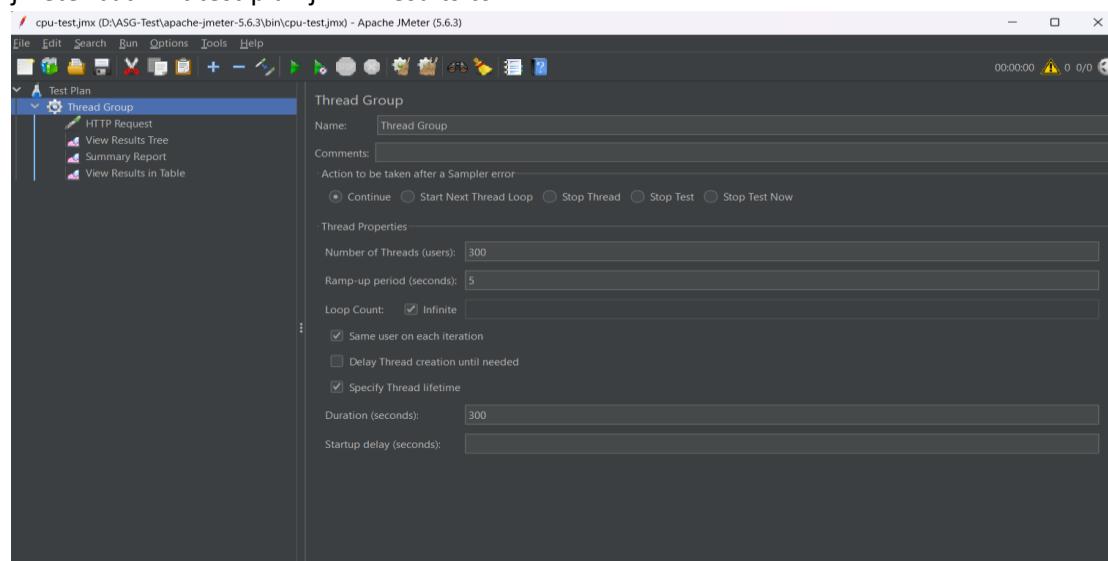
4. Load Testing with Apache JMeter

Apache JMeter was used to simulate real-world high traffic to validate scaling behaviour.

4.1 Test Plan Configuration

- Threads (users): **300**
- Ramp-up: **5 seconds**
- Test Duration: **300 seconds**
- Target: ALB DNS endpoint
- Test executed using **CLI mode**:

```
jmeter.bat -n -t test-plan.jmx -l results.csv
```



The screenshot shows the Apache JMeter 5.6.3 interface. The left pane displays the 'Test Plan' tree with a 'Thread Group' node selected. The right pane shows the 'Thread Group' configuration panel with the following settings:

- Name: Thread Group
- Number of Threads (users): 300
- Ramp-up period (seconds): 5
- Loop Count: Infinite
- Same user on each iteration:
- Delay Thread creation until needed:
- Specify Thread lifetime:
- Duration (seconds): 300
- Startup delay (seconds): (empty)

cpu-test.jmx (D:\ASG-Test\apache-jmeter-5.6.3\bin\cpu-test.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

0:00:00 ⚠ 0 0/0

Test Plan

- Thread Group
 - HTTP Request
 - View Results Tree
 - Summary Report**
 - View Results in Table

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: D:\ASG-Test\apache-jmeter-5.6.3\bin\results.csv

Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
HTTP Request	3212273	27	21	7314	10.65	0.14%	1643.6/sec	487.87	242.36	304.0
TOTAL	3212273	27	21	7314	10.65	0.14%	1643.6/sec	487.87	242.36	304.0

cpu-test.jmx (D:\ASG-Test\apache-jmeter-5.6.3\bin\cpu-test.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

0:00:00 ⚠ 0 0/0

Test Plan

- Thread Group
 - HTTP Request**
 - View Results Tree
 - Summary Report
 - View Results in Table

HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

Protocol [http]: Server Name or IP: Gurdeep-ALB-194321633.us-east-1.elb.amazonaws.com Port Number:

HTTP Request

GET Path: / Content encoding:

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?

Windows PowerShell

```
/apache-jmeter-5.6.3/lib/xstream-1.4.20.jar)
WARNING: Please consider reporting this to the maintainers of class com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
PS D:\ASG-Test\apache-jmeter-5.6.3\bin> .\jmeter.bat -n -t cpu-test.jmx -l results.csv
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider (file:/D:/ASG-Test
/apache-jmeter-5.6.3/lib/xstream-1.4.20.jar)
WARNING: Please consider reporting this to the maintainers of class com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Creating summariser <summary>
Created the tree successfully using cpu-test.jmx
Starting standalone test @ 2025 Dec 3 12:52:47 GMT-05:00 (1764784367608)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 107624 in 00:00:12 = 8711.3/s Avg: 27 Min: 21 Max: 275 Err: 44 (0.04%) Active: 300 Started: 300 Finished: 0
summary + 319394 in 00:00:30 = 10651.1/s Avg: 28 Min: 21 Max: 403 Err: 497 (0.16%) Active: 300 Started: 300 Finished: 0
summary = 426918 in 00:00:42 = 10085.5/s Avg: 27 Min: 21 Max: 403 Err: 541 (0.13%)
summary + 326038 in 00:00:30 = 10863.2/s Avg: 27 Min: 21 Max: 293 Err: 570 (0.17%) Active: 300 Started: 300 Finished: 0
summary = 752956 in 00:01:12 = 10488.1/s Avg: 27 Min: 21 Max: 403 Err: 1111 (0.15%)
summary + 298835 in 00:00:30 = 9938.8/s Avg: 30 Min: 21 Max: 598 Err: 609 (0.20%) Active: 300 Started: 300 Finished: 0
summary = 1058991 in 00:01:42 = 10270.6/s Avg: 28 Min: 21 Max: 598 Err: 1720 (0.16%)
summary + 327624 in 00:00:30 = 10919.0/s Avg: 27 Min: 21 Max: 287 Err: 453 (0.14%) Active: 300 Started: 300 Finished: 0
summary = 1378615 in 00:02:12 = 10147.6/s Avg: 28 Min: 21 Max: 598 Err: 2173 (0.16%)
summary + 296913 in 00:00:30 = 9898.4/s Avg: 30 Min: 21 Max: 287 Err: 570 (0.19%) Active: 300 Started: 300 Finished: 0
summary = 1675526 in 00:02:42 = 10321.7/s Avg: 28 Min: 21 Max: 598 Err: 2742 (0.16%)
summary + 323380 in 00:00:30 = 10776.7/s Avg: 27 Min: 22 Max: 488 Err: 591 (0.18%) Active: 300 Started: 300 Finished: 0
summary = 1998828 in 00:03:12 = 10392.6/s Avg: 28 Min: 21 Max: 598 Err: 3334 (0.17%)
summary + 324895 in 00:00:30 = 10826.5/s Avg: 27 Min: 21 Max: 7314 Err: 575 (0.18%) Active: 300 Started: 300 Finished: 0
summary = 2323623 in 00:03:42 = 10451.2/s Avg: 28 Min: 21 Max: 7314 Err: 3909 (0.17%)
summary + 337000 in 00:00:30 = 11157.4/s Avg: 26 Min: 21 Max: 269 Err: 58 (0.02%) Active: 300 Started: 300 Finished: 0
summary = 2658333 in 00:00:12 = 10535.1/s Avg: 26 Min: 21 Max: 7314 Err: 3967 (0.15%)
summary + 322742 in 00:00:30 = 11091.4/s Avg: 26 Min: 21 Max: 269 Err: 352 (0.11%) Active: 300 Started: 300 Finished: 0
summary = 2991075 in 00:00:12 = 10591.3/s Avg: 26 Min: 21 Max: 7314 Err: 4319 (0.14%)
summary + 201198 in 00:00:18 = 11333.8/s Avg: 26 Min: 21 Max: 266 Err: 107 (0.05%) Active: 0 Started: 300 Finished: 300
summary = 3192273 in 00:05:00 = 10638.0/s Avg: 27 Min: 21 Max: 7314 Err: 4426 (0.14%)
Tidying up ... @ 2025 Dec 3 12:57:47 GMT-05:00 (1764784667753)
... end of run
PS D:\ASG-Test\apache-jmeter-5.6.3\bin> .\jmeter.bat
```

5. Results & Observations

5.1 Scale-Out Behavior

During the load test, the ASG successfully launched new instances as CPU utilization surpassed the scaling threshold.

The screenshot shows the AWS CloudWatch Metrics Insights interface. A query is being run against CloudWatch Metrics Insights metrics. The results table displays three rows of data, each representing a metric event. The columns include Metric Name, Metric Value, and a timestamp. The first two rows show values of 1.0, and the third row shows a value of 2.0. The interface includes various filters and sorting options for the results.

Metric Name	Metric Value	Timestamp
ASG-Web-Server	1.0	2023-09-12T10:00:00Z
Gurdeep-Web-...	1.0	2023-09-12T10:00:00Z
ASG-Web-Server	2.0	2023-09-12T10:00:00Z

5.2 Scale-In Behavior

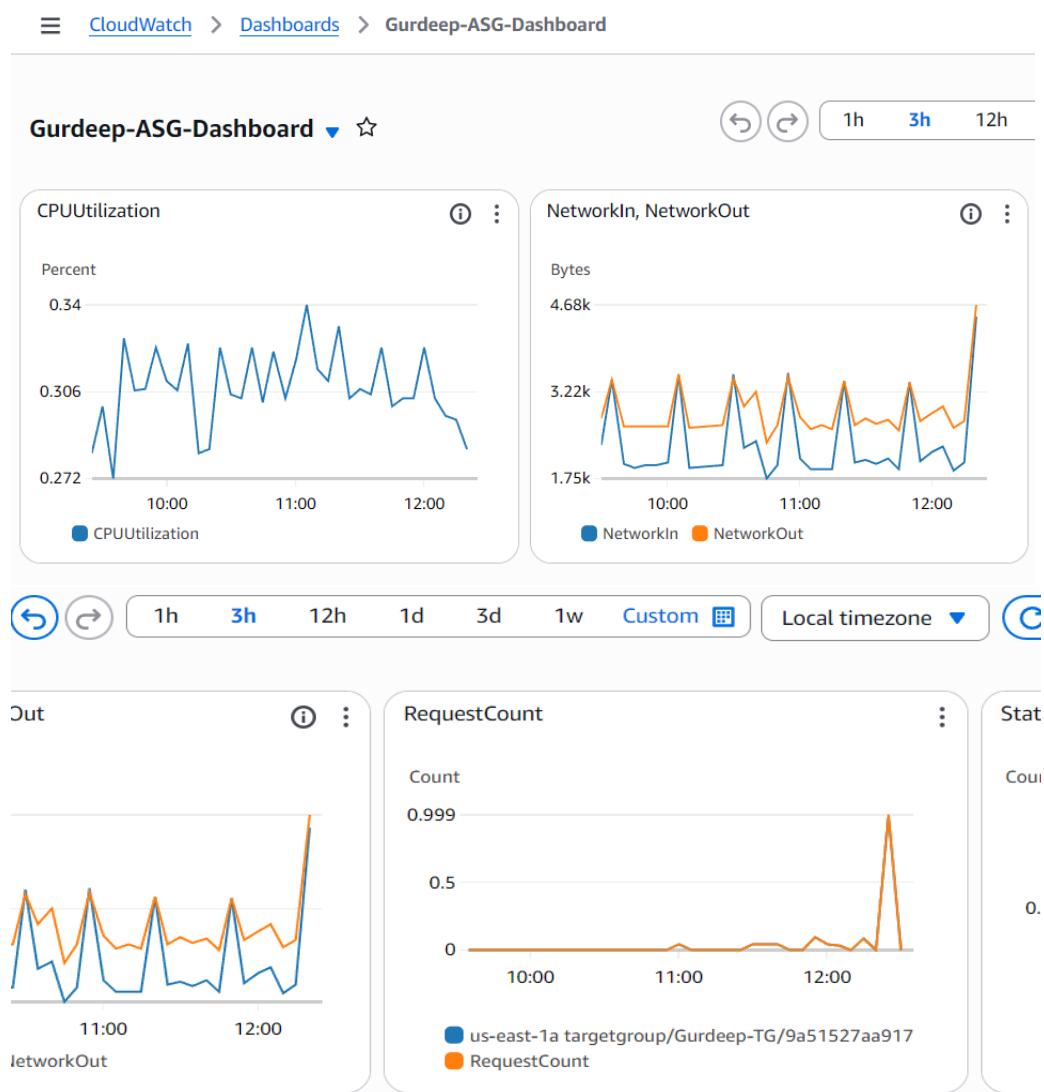
After test completion, CPU utilization dropped and the ASG terminated extra instances, returning to the desired count.

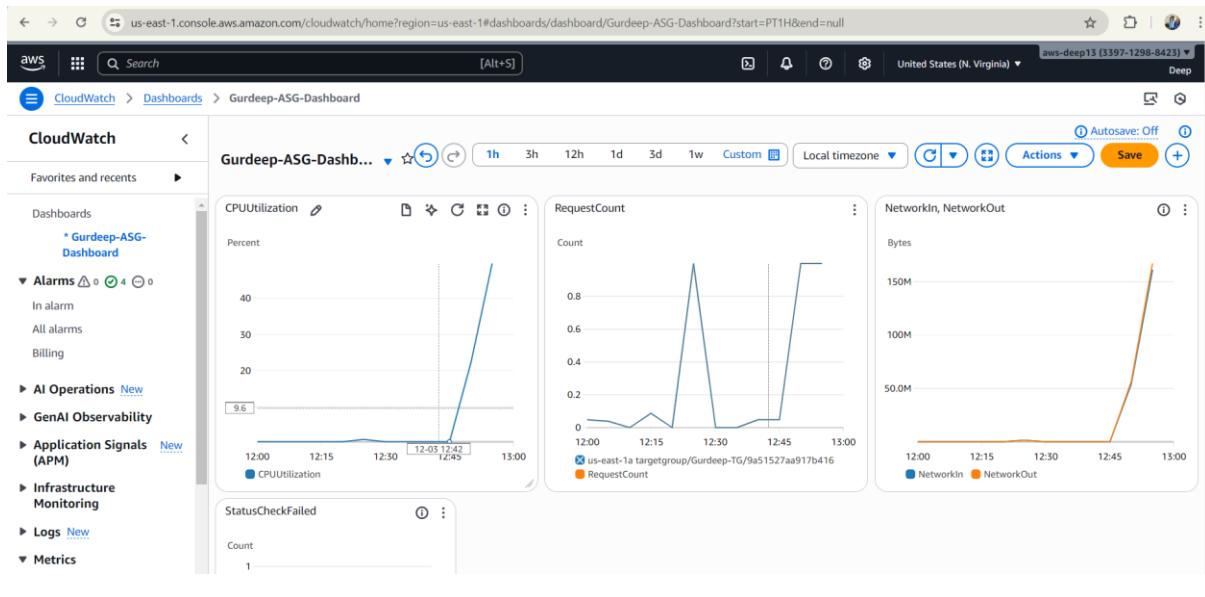
The screenshot shows the AWS CloudWatch Metrics Insights interface. A query is being run against CloudWatch Metrics Insights metrics. The results table displays three rows of data, each representing a metric event. The first two rows show values of 1.0, and the third row shows a value of 2.0. The interface includes various filters and sorting options for the results.

Metric Name	Metric Value	Timestamp
ASG-Web-Server	1.0	2023-09-12T10:00:00Z
Gurdeep-Web-...	1.0	2023-09-12T10:00:00Z
ASG-Web-Server	2.0	2023-09-12T10:00:00Z

5.3 CloudWatch Metrics Observed

- **CPU Utilization** spiked during load test
- **RequestCount** increased significantly
- **NetworkIn/NetworkOut** displayed heavy throughput
- **Auto Scaling Events** were logged





6. Conclusion

This project successfully demonstrates:

- Deployment of a **production-ready scalable architecture** using EC2 Auto Scaling
- Efficient traffic distribution through an **Application Load Balancer**
- Elastic and automated scaling through **CloudWatch alarms**
- Load testing validation using **Apache JMeter**
- Proper scale-out and scale-in behaviour during demand fluctuation

The system is **highly available, fault tolerant, and cost optimized.**

7. Deliverables

Included in project ZIP file:

File	Description
Documentation.pdf	Final project report
Architecture-Diagram.png	System architecture
results.csv	JMeter load testing results
test-plan.jmx	JMeter test plan
screenshots/	All project screenshots

8. Cleanup Steps (to avoid AWS charges)

1. Delete **ALB** (costs money continuously)
2. Delete **Target Group**
3. Delete **Auto Scaling Group**
4. Delete **Launch Template**
5. Terminate **EC2 instances**
6. Delete **Security Groups** (if unused)
7. Remove **CloudWatch Dashboards**
8. Delete **CloudWatch Alarms**