



Final Project
Cloud Computing Track

Ala'a Abdelqader

Supervised by:
Eng. Yazan AlShannik
Soft Skills Instructor: Dr. Sawsan Sayeh
English Language Instructor: Dr. Basheer Mufleh

AlHussein Technical University (HTU)

June/2025

Table of Contents

a. Objective	6
b. Overview of the Four Scenarios	6
2. Scenario #1 – Hosting My Portfolio on AWS	7
a. Objective	7
b. Architecture Diagram	7
c. Implementation Steps	7
1) Created a Private S3 Bucket	7
2) Configured Bucket Policy for CloudFront	7
3) Set Up CloudFront Distribution.....	7
4) Created a Custom SSL Certificate	8
5) Configured Route 53 for Domain Routing.....	8
6) Tested the Website.....	8
d. Screenshots	8
e. Conclusion	12
3. Scenario #2 – Auto-Scaling Web Application.....	13
a. Objective	13
b. Architecture Diagram	13
c. Implementation Steps	14
1) Created an S3 Bucket for Static Content.....	14
2) Built a Launch Template for EC2 Instances.....	14
3) Created an Auto Scaling Group (ASG)	14
4) Attached Scaling Policies.....	14
5) Set Up an Application Load Balancer (ALB).....	15
6) Enabled CloudWatch Monitoring and Alarms.....	15
7) Performed Load Testing with stress Tool.....	15
8) Verified Notifications and Instance Behavior.....	16
9) Monitored with CloudWatch Dashboards	16
d. Screenshots	16
e. Conclusion	28
4) Scenario #3 – Secure Linux Server Setup on AWS	29
a. Objective	29
b. Implementation Steps	29
1) Launched the EC2 Instance.....	29
2) Connected to EC2 Using SSH.....	29
3) Created Folder Structure for Departments	29
4) Created User Groups and Users.....	29
5) Configured Directory Permissions	30
6) Wrote GitHub Backup Script	30
7) Set Up Cron Job for Daily Automation	30
8) Validated the Automation.....	30

d. Screenshots	31
e. Conclusion	35
5. Scenario #4 – Serverless HealthBooking on AWS	35
a. Objective	35
b. Implementation Steps	35
1) Designed and Created DynamoDB Tables.....	35
2) Developed Lambda Functions and APIs.....	36
3) Configured Notifications Using Amazon SNS	36
4) Set Up Daily Automation with EventBridge	36
5) Built and Deployed the Frontend Using AWS Amplify.....	36
6) Implemented Two Main Web Pages.....	37
7) Set Up Monitoring and Logging Using CloudWatch	37
d. Screenshots	37
Alarm for Invocation Count (Spike Monitoring)	49
Alert if Latency > 2s	49
e. Conclusion	50

Table of Figures

Figure 1-Scenario #1 Diagram	7
Figure 2-Created a Private S3 Bucket.....	8
Figure3 -Configured Bucket Policy for CloudFront	9
Figure4 -Block All Public Access	9
Figure 5 -Set Up CloudFront Distribution	9
Figure 6 - added (custom domain+ SSL certificate)	10
Figure 7-CloudFront cache invalidation.....	10
Figure 8-Created a Custom SSL Certificate	10
Figure 9-Configured Route 53 for Domain Routing.....	11
Figure 10- To validate the domain, I added the provided CNAME record to Route 53	11
Figure 11-Tested the Website	12
Figure 12-Tested the Website	12
Figure 13-Scenario #2 Diagram	13
Figure 14- S3 Bucket for Static Content	16
Figure 15-IAM Role to allowed access To S3 from EC2	17
Figure 16- Enable Static Website	17
Figure 17-Instance Template.....	17
Figure 18- User Data in instance Template	18
Figure 19-Attached a security group named web-security	18
Figure 20-Auto Scaling Group	19
Figure 21-Added Target Tracking Scaling Policies to the ASG	19
Figure 22 -Added a listener on port 80 (HTTP).....	20

Figure 23-Application Load Balancer details.....	20
Figure 24- target group named WebApp-TG.....	21
Figure 25-Configured health checks for the target group to monitor /index.html	21
Figure 26-GroupDesiredCapacity = 4 to detect if max capacity is reached.....	22
Figure 27- Enabled detailed monitoring	22
Figure 28-History CloudWatch	23
Figure 29 - Actions of CloudWatch	23
Figure 30- SNS To Email	23
Figure 31- SNS Details	23
Figure 32-Connected to 2 EC2 instances via SSH and Installed the stress tool.	24
Figure 33-Ran command to simulate high CPU usage for 10 minutes.....	24
Figure 34-before stress	25
Figure 35-After stress on 2 instance	25
Figure 36-reaching maximum configured capacity	25
Figure 37-Auto Scaling Trigger: New EC2 Instance Launched Due to High CPU	25
Figure 38- Auto Scaling Alert email Notification: Maximum Capacity Reached.....	26
Figure 39- Auto Scaling Activity Log: EC2 Launch and Termination Events	26
Figure 40-Analyze Scaling Behavior and Trends from CloudWatch Metrics	27
Figure 41-Analyzing Scaling Behavior Using CloudWatch Metrics	28
Figure 42-EC2 instance using Amazon Linux 2	31
Figure 43-custom security group that allows SSH access (port 22) only from my IP address	31
Figure 44-Connected to EC2 Using SSH	32
Figure 45>Create Groups and Users-assign them to their respective groups	32
Figure 46-Ownership and Permissions	32
Figure 47-SSH Key Generation for GitHub Backup	33
Figure 48-clone to my repo on GitHub	33
Figure 49-Viewing and Configuring SSH Deploy Key	33
Figure 50-Add key on the repo.....	33
Figure 51-Connection to GitHub is Done	33
Figure 54 -Scheduled Cron Job for Daily GitHub Backup at 9:45 PM	34
Figure 52-Backup script.....	34
Figure 53-Test the script manually.	34
Figure 55 -Automated Daily Backup Files Uploaded to Private GitHub Repository	34
Figure 56-Log Output Showing Successful GitHub Backup Commit.....	35
Figure 57-Appointment table	37
Figure 58-slots table.....	38
Figure 59-All Lambda Functions.....	38
Figure 60- Test create Appointment Function	38
Figure 61-All permissions for create Appointment	38
Figure 62-Cloudwatch permission for create Appointment	39
Figure 63-SNS permission for the create Appointment.....	39
Figure 64-DynamoDB permission for create Appointment	39
Figure 65 - Test get Available Slots function	39
Figure 66-Dynamo Permission for get Available Slots	40

Figure 67 -Test get All Appointments function.....	40
Figure 68 - DyanmoDB permission for get All Appointments	40
Figure 69-Test update Status Appointment function.....	41
Figure 70 -DynamoDB permission for update Status Appointment	41
Figure 71-Test for reset daily bookings.....	41
Figure 72-All permission for reset daily bookings	42
Figure 73-DynamoDB permission for reset daily bookings.....	42
Figure 74-All my APIs	43
Figure 75-Book an Appointment API Test Postman	43
Figure 76-Update Appointment Status API Test Postman	44
Figure 77-Test Get Available Time Slots API Postman	44
Figure 78-Get Appointments API test Postman	44
Figure 79 - SNS doctor booking.....	45
Figure 80 -A new appointment Notification (SNS TEST)	45
Figure 81 -Reset Confirmation Notification (SNS Test)	45
Figure 82- Daily Automation with Event Bridge	46
Figure 83 - EventBridge Target	46
Figure 84-Deployed the Frontend Using AWS Amplify.....	46
Figure 85 -Book an Appointment Page	47
Figure 86-Doctor Appointments page.....	47
Figure 87 -Enabled logging for all Lambda functions	47
Figure 88 -create Appointment Logging Page	48
Figure 89- Alarm for Invocation Count (Spike Monitoring)	48
Figure 90 -Alert if Latency > 2s (Cloudwatch)	49

1. Introduction

The **ICT Upskilling Program** is a training initiative organized by the Centre of Professional Development and Community Outreach. It's designed to help participants build real, hands-on experience in cloud computing, manual software testing, soft skills, and business English. Under the guidance of industry instructors like Eng. Yazan Al-Shannik, Dr. Sawsan Alsayeh, and Dr. Basheer Mufleh, this program blends technical learning with communication and presentation skills to prepare us for real-world IT careers.

a. Objective

The main goal of the capstone is to prove that we can design, build, and explain cloud-based solutions using AWS services. We're expected to apply our technical skills (like creating secure infrastructure and automating tasks), demonstrate soft skills (like presenting clearly), and communicate professionally in English.

b. Overview of the Four Scenarios

1. Hosting My Portfolio on AWS

I build and host a personal portfolio website using Amazon S3. It's delivered securely through CloudFront and linked to a custom domain using Route 53.

2. Auto-Scaling Web Application with Monitoring

I create a scalable web app using EC2, Auto Scaling Groups, and a Load Balancer. Static content is served from S3, and I use CloudWatch and SNS to monitor and send alerts based on system activity.

3. Secure Linux Server Setup

I set up a Linux server for three different departments (HR, Dev, and Ops), configure user and group permissions, and automate daily backups to a private GitHub repository using shell scripts and cron jobs.

4. Serverless HealthBooking on AWS

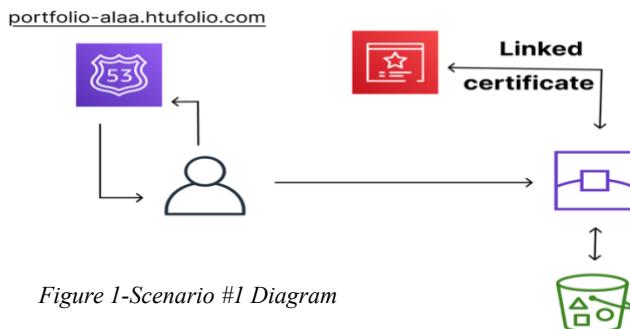
I deploy a fully serverless application that simulates an Health booking system. It includes Lambda functions, API Gateway, DynamoDB, and other AWS services. The goal is to explore how serverless setups compare to traditional server-based systems.

2. Scenario #1 – Hosting My Portfolio on AWS

a. Objective

The goal of this scenario is to host a personal developer portfolio on AWS in a way that's secure, fast, and professional. This includes storing the website files in a private S3 bucket, delivering them using CloudFront, and connecting the site to a custom domain through Route 53. The final result should be a fully functional portfolio that reflects my work .

b. Architecture Diagram



c. Implementation Steps

1) Created a Private S3 Bucket

- I named my bucket portfolio.alaaaa.com .
- To keep it secure, I enabled **Block All Public Access** for the bucket.
- I uploaded my website files, including index.html, images, CSS, and JS folders.

2) Configured Bucket Policy for CloudFront

- Since the bucket is private, I added a policy that allows **only CloudFront** to access its contents.
- This was done using a JSON policy that grants the CloudFront distribution permission to get objects from the bucket.

3) Set Up CloudFront Distribution

- I created a CloudFront distribution and set the S3 bucket as the origin.
- I used **Origin Access Control (OAC)** to securely connect CloudFront with the private S3 bucket.
- I added my custom domain portfolio-alaa.htufolio.com as an alternate domain name.
- Linked the distribution to a valid **SSL certificate** (issued by AWS Certificate Manager).
- I ran a **CloudFront cache invalidation** to make sure new changes appear immediately.

4) Created a Custom SSL Certificate

- I requested a new certificate for portfolio-alaa.htufolio.com using AWS Certificate Manager.
- To validate the domain, I added the provided **CNAME record** to Route 53.
- Once validated, the status showed "**Success**", and I used it in CloudFront for HTTPS support.

5) Configured Route 53 for Domain Routing

- I used Route 53 to set up a **Hosted Zone** for my domain (htufolio.com).
- I created a **CNAME record** pointing portfolio-alaa.htufolio.com to the CloudFront domain (d1pedsexnd0a7l.cloudfront.net).
- This allows users to visit my portfolio using a clean and professional domain.

6) Tested the Website

- After completing all the setup, I tested the domain in the browser.
- The website loaded successfully using HTTPS, and the content was served via CloudFront from the private S3 bucket.

d. Screenshots

- **S3 Bucket**

portfolio.alaaaa.com Info						
Objects		Metadata	Properties	Permissions	Metrics	Management
Actions		Copy S3 URI	Copy URL	Download	Open	Delete
<input type="checkbox"/>	Find objects by prefix	<input checked="" type="radio"/> Show versions				
Name	Type	Last modified	Size	Storage class		
css/	Folder	-	-	-	-	
docs/	Folder	-	-	-	-	
error.html	html	June 1, 2025, 14:10:40 (UTC+03:00)	2.0 KB	Standard		
fonts/	Folder	-	-	-	-	
images/	Folder	-	-	-	-	
index.html	html	June 1, 2025, 14:10:40 (UTC+03:00)	28.2 KB	Standard		
js/	Folder	-	-	-	-	
prepros-6.config	config	June 1, 2025, 14:10:41 (UTC+03:00)	18.2 KB	Standard		
scss/	Folder	-	-	-	-	
single.html	html	June 1, 2025, 14:10:41 (UTC+03:00)	21.8 KB	Standard		

Figure 2-Created a Private S3 Bucket

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

On

► Individual Block Public Access settings for this bucket

Figure 4 -Block All Public Access

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about [using Amazon S3 Block Public Access](#)

```
{
  "Version": "2008-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::portfolio.alaaaa.com/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::211839440217:distribution/E157KGV8CMJIXD"
        }
      }
    }
  ]
}
```

Copy

Figure 3 -Configured Bucket Policy for CloudFront

● Cloudfront

E157KGV8CMJIXD Standard

[View metrics](#)

General [Security](#) [Origins](#) [Behaviors](#) [Error pages](#) [Invalidations](#) [Tags](#) [Logging](#)

Details

Distribution domain name [d1pedsexnd0a7l.cloudfront.net](#)

ARN [arn:aws:cloudfront::211839440217:distribution/E157KGV8CMJIXD](#)

Last modified [Deploying](#)

Settings

Description -

Price class Use all edge locations (best performance)

Supported HTTP versions HTTP/2, HTTP/1.1, HTTP/1.0

Alternate domain names [portfolio-alaa.htufolio.com](#)

Custom SSL certificate [portfolio-alaa.htufolio.com](#)

Security policy TLSv1.2_2021

Standard logging [Off](#)

Cookie logging [Off](#)

Default root object index.html

Continuous deployment [Info](#)

[Create staging distribution](#)

Figure 5 -Set Up CloudFront Distribution

The screenshot shows the 'Settings' tab for a CloudFront distribution. Under 'Anycast static IP list', it says 'There are no Anycast static IP lists available'. Under 'Price class', 'Use all edge locations (best performance)' is selected. In the 'Alternate domain name (CNAME) - optional' section, 'portfolio-alaa.htufolio.com' is listed with a 'Remove' button. A note says '(?) To add a list of items, use the bulk editor.' Under 'Custom SSL certificate - optional', it says 'Associate a certificate from AWS Certificate Manager. The certificate must be in the US East (N. Virginia) Region (us-east-1).'. A dropdown menu shows 'portfolio-alaa.htufolio.com (489fba17-1914-4523-82eb-3e2dcb015d3c)' and a link to 'Request certificate'.

Figure 6 - added (custom domain+SSL certificate)

The screenshot shows the 'Invalidation details' page. It has sections for 'Date created' (June 2, 2025 at 5:02:15 PM UTC), 'Status' (Completed), and 'Object paths' (/*).

Figure 7-CloudFront cache invalidation

• Certificate

The screenshot shows the ACM 'Certificates' page. A new certificate '489fba17-1914-4523-82eb-3e2dcb015d3c' is listed. Its 'Certificate status' is 'Issued'. The 'Domains' section shows one domain, 'portfolio-alaa.htufolio.com', with a status of 'Success'. The 'Details' section provides certificate metadata: Serial number 0f:4afe:50:72:16:18:15:8b:f0:7d:2b:ef:4a:dc:d9, Public key info RSA 2048, Signature algorithm SHA-256 with RSA, and various timestamps for requests, issuance, and renewals.

Figure 8-Created a Custom SSL Certificate

- **Route 53**

Record details

Record name: portfolio-alaa.htufolio.com

Record type: CNAME

Value: d1pedsexnd0a7l.cloudfront.net

Alias: No

TTL (seconds): 3600

Routing policy: Simple

Record name	Type	Routing p...	Alias	Value/Route traffic to	TTL (s...)	Health ...
htufolio.com	NS	Simple	No	ns-1296.awsdns-34.org. ns-1546.awsdns-01.co.uk. ns-859.awsdns-43.net. ns-454.awsdns-56.com.	172800	-
htufolio.com	SOA	Simple	No	ns-1296.awsdns-34.org. aw...	900	-
portfolio-alaa.htufolio.com	CNAME	Simple	No	d1pedsexnd0a7l.cloudfront.net	3600	-
_79270f5d7bbfc902c40f52686053003a.portfolio-...	CNAME	Simple	No	_599f6651315cd786bcba5a...	300	-

Figure 9-Configured Route 53 for Domain Routing

Record details

Record name: _79270f5d7bbfc902c40f52686053003a

Record type: CNAME

Value: _599f6651315cd786bcba5aab3f8e3feexlggrmvvl.acm-validation.aws.

Alias: No

TTL (seconds): 300

Routing policy: Simple

Record name	Type	Routing p...	Alias	Value/Route traffic to	TTL (s...)	Health ...
htufolio.com	NS	Simple	No	ns-1296.awsdns-34.org. ns-1546.awsdns-01.co.uk. ns-859.awsdns-43.net. ns-454.awsdns-56.com.	172800	-
htufolio.com	SOA	Simple	No	ns-1296.awsdns-34.org. aw...	900	-
portfolio-alaa.htufolio.com	CNAME	Simple	No	d1pedsexnd0a7l.cloudfront.net	3600	-
_79270f5d7bbfc902c40f52686053003a.portfolio-...	CNAME	Simple	No	_599f6651315cd786bcba5a...	300	-

Figure 10- To validate the domain, I added the provided CNAME record to Route 53

- Tested The Website

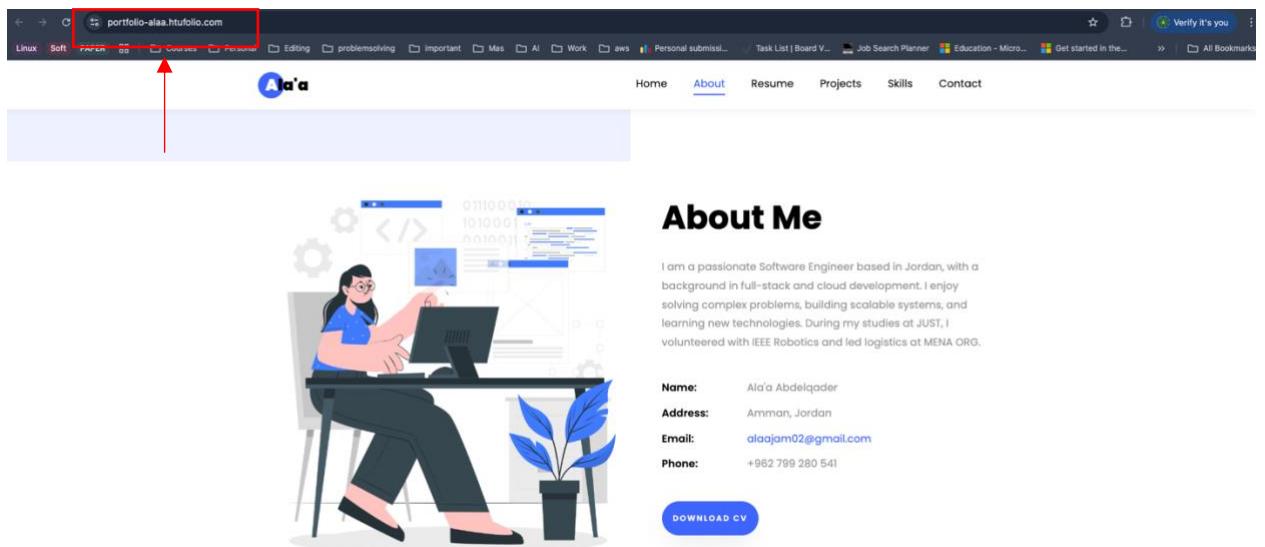


Figure 11-Tested the Website

e. Conclusion

The architecture was designed following AWS best practices to ensure **high availability and fault tolerance** by using **CloudFront** as a global content delivery network, which serves the website content from edge locations closest to the user, reducing latency and ensuring consistent performance. The content is hosted in an **S3 bucket with restricted public access**, and access is securely managed through **Origin Access Control (OAC)**. Additionally, **Route 53** was used for domain routing, providing DNS failover capability and improved availability.

In the future, I could improve the setup by:

- **Enabling versioning** on the S3 bucket to track and recover from accidental changes or deletions.
- **Adding a custom error page setup** in CloudFront for better user experience.
- **Implementing CloudWatch monitoring** to track access logs, performance, and detect issues early.
- **Setting up CI/CD pipelines** using AWS CodePipeline or GitHub Actions to automate future updates.

3. Scenario #2 – Auto-Scaling Web Application

a. Objective

The goal of this scenario is to build a web application that can **automatically scale up or down** based on how much traffic it's getting. This means the app can handle more users when needed and save costs when it's not busy. We used **EC2 instances** to run the web server and **S3** to store static content like the HTML file. To make sure the app stays available and performs well, we connected it to an **Auto Scaling Group** and a **Load Balancer**. We also added monitoring tools like **CloudWatch** to track how the system is doing, and **SNS notifications** to alert us by email if something changes (like launching new servers when traffic increases). The overall idea is to create a **resilient, scalable, and well-monitored** application infrastructure—just like real companies do when they want their services to stay online and responsive under any load.

b. Architecture Diagram

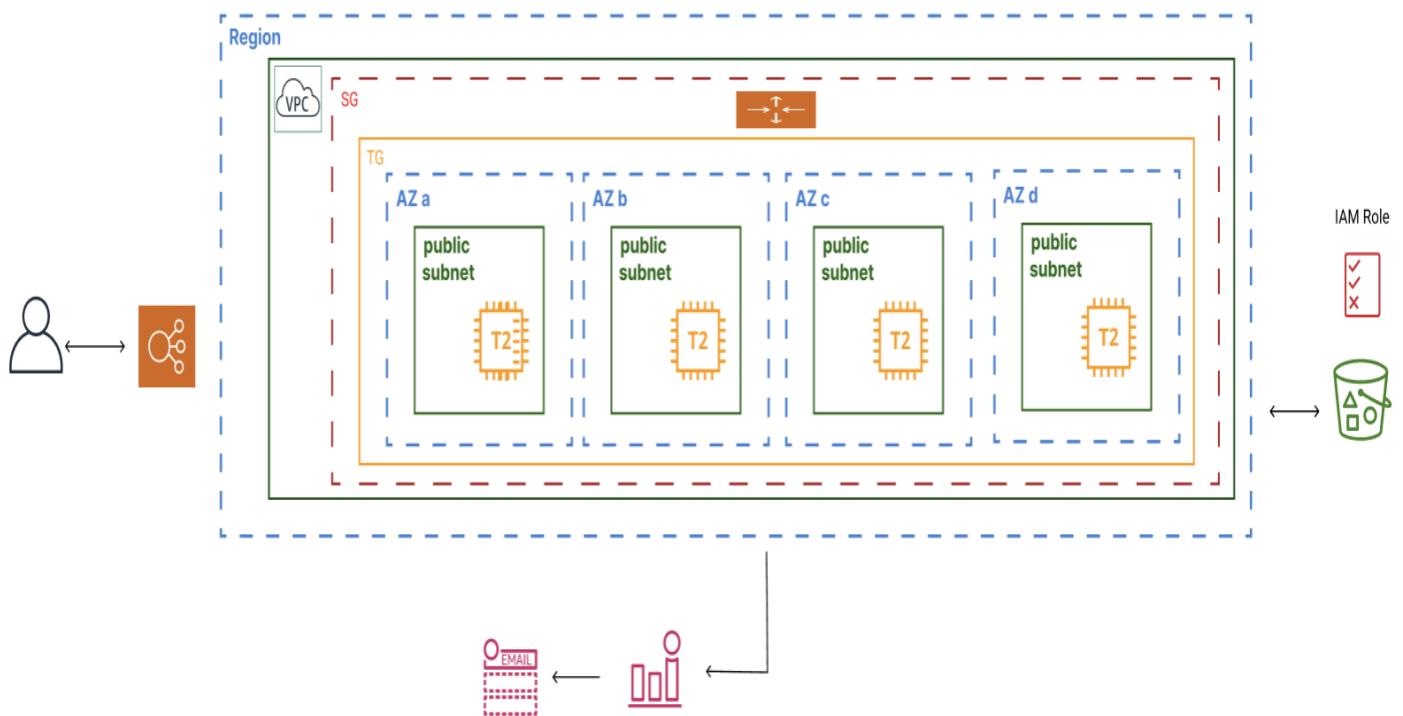


Figure 13-Scenario #2 Diagram

c. Implementation Steps

1) Created an S3 Bucket for Static Content.

- I created an S3 bucket named htu-webpage to host the static HTML content (index.html).
- Enabled S3 static website.
- Kept Block all public access **enabled** to improve security, and instead allowed access through EC2 using an IAM Role.

2) Built a Launch Template for EC2 Instances.

- Created a **Launch Template** named WebAppTemplate using the Amazon Linux 2 AMI.
- Attached a security group named web-security that allows:
 - HTTP traffic on port 80 from anywhere.
 - SSH access only from my IP for management
- In the **User Data** script:
 - Installed Apache
 - Started the HTTP service.
 - Pulled index.html from the S3 bucket and placed it in /var/www/html.

3) Created an Auto Scaling Group (ASG)

- I created an Auto Scaling Group named ASG-web using the above launch template.
- Set the minimum capacity to 1, desired capacity to 1, and maximum capacity to 4.
- Attached the group to four subnets in different Availability Zones for high availability.
- This ensures the application scales in or out based on actual demand.

4) Attached Scaling Policies

- Added **Target Tracking Scaling Policies** to the ASG: Policy to **scale out** when average CPU Utilization > 50%.
- Minimum number of healthy instances required for scaling out: 1.

- CloudWatch automatically triggers these policies when thresholds are breached.

5) Set Up an Application Load Balancer (ALB)

- Created an Application Load Balancer named ASG-web-LB.

Scheme: **Internet-facing**, type: **Application**.

- Added the four public subnets to support high availability.
- Added a listener on port 80 (HTTP).
- Forwarded requests to a target group named WebApp-TG.
- Configured health checks for the target group to monitor /index.html.

6) Enabled CloudWatch Monitoring and Alarms

- Enabled detailed monitoring on the ASG and EC2 instances.
- Created CloudWatch Alarm for GroupDesiredCapacity = 4 to detect if max capacity is reached.
- Verified metrics visually via the **CloudWatch dashboard**.
- Used the SNS service to receive email notifications when alarms go off.

7) Performed Load Testing with stress Tool

- Connected to 2 EC2 instances via SSH.
- Installed the stress tool.
- Ran the following command to simulate high CPU usage for 10 minutes:
- Verified from CloudWatch and EC2 dashboard that:
 - CPU usage spiked above 50%.
 - Auto Scaling Group launched additional instances up to (3).
 - Received SNS email notifications about the scaling events.

8) Verified Notifications and Instance Behavior

- SNS notifications were triggered for launching new instances due to CPU spike and ASG reaching maximum configured capacity (4 instance).
- After stress test completed, idle instances were **automatically shut down**.
- Verified that EC2 instance count was scaled down again to match actual load.

9) Monitored with CloudWatch Dashboards

- Used CloudWatch to monitor: GroupDesiredCapacity
- All metrics confirmed correct scaling behavior throughout the test period.

d. Screenshots

- S3 Bucket

The screenshot shows the AWS S3 console interface for the 'htu-webpage' bucket. The 'Objects' tab is active, showing a single object named 'index.html'. The object details are as follows:

Name	Type	Last modified
index.html	html	June 7, 2025, 11:21:24 (UTC+03:00)

At the top of the page, there are tabs for Objects, Metadata, Properties, Permissions, Metrics, Management, and Access Points. Below the tabs, there are buttons for Copy S3 URI, Copy URL, and Download. A search bar labeled 'Find objects by prefix' is also present.

Figure 14- S3 Bucket for Static Content

ec2-s3-read Info

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date
June 07, 2025, 11:54 (UTC+05:00)

Last activity
18 minutes ago

ARN
arn:aws:iam::730335665786:role/ec2-s3-read

Instance profile ARN
arn:aws:iam::730335665786:instance-profile/ec2-s3-read

Maximum session duration
1 hour

Permissions **Trust relationships** **Tags** **Last Accessed** **Revoke sessions**

Permissions policies (1) Info

You can attach up to 10 managed policies.

Filter by Type
All types

Policy name	Type	Attached entities
AmazonS3ReadOnlyAccess	AWS managed	1

Simulate **Remove** **Add permissions**

Figure 15-IAM Role to allowed access To S3 from EC2

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

We recommend using AWS Amplify Hosting for static website hosting

Deploy a fast, secure, and reliable website quickly with AWS Amplify Hosting. Learn more about [Amplify Hosting](#) or [View your existing Amplify apps](#)

S3 static website hosting
Enabled

Hosting type
Bucket hosting

Bucket website endpoint

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)

<http://htu-webpage.s3-website-us-east-1.amazonaws.com>

Figure 16- Enable Static Website

- Instance Template

WebAppTemplate (lt-0b03608422ca54811)

Actions **Delete template**

Launch template details

Launch template ID
lt-0b03608422ca54811

Launch template name
WebAppTemplate

Default version
1

Owner
arn:aws:iam::730335665786:user/Alaa-admin

Details **Versions** **Template tags**

Launch template version details

Version
1 (Default)

Description
WebAppTemplate description

Date created
2025-06-07T09:28:39.000Z

Created by
arn:aws:iam::730335665786:user/Alaa-admin

Instance details **Storage** **Resource tags** **Network interfaces** **Advanced details**

AMI ID
ami-02457590d33d576c3

Instance type
t2.micro

Availability Zone
-

Key pair name
key-pair

Security groups
sg-0bd0270b69ba6e539

Figure 17-Instance Template

```

User data


```

#!/bin/bash

yum update -y

Install Apache and AWS CLI
yum install -y httpd awscli

Start and enable Apache
systemctl start httpd
systemctl enable httpd

Download the HTML file from S3
aws s3 cp s3://htu-webpage/index.html /var/www/html/index.html

```


```

Figure 18- User Data in instance Template

- Web Security Group

sg-0bd0270b69ba6e539 - web-security

Details

Security group name <input type="checkbox"/> web-security	Security group ID <input type="checkbox"/> sg-0bd0270b69ba6e539	Description <input type="checkbox"/> Allows to http and SSH	VPC ID <input type="checkbox"/> vpc-0b63e5b0ab3c2e3da [?]
Owner <input type="checkbox"/> 730335665786	Inbound rules count 2 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules | Outbound rules | Sharing - new | VPC associations - new | Tags

Inbound rules (2)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0f3e719b9d30dff3d	IPv4	SSH	TCP	22	94.249.50.196/32	-
-	sgr-0914bafdbe5327890	IPv4	HTTP	TCP	80	0.0.0.0/0	-

Figure 19-Attached a security group named web-security

- Auto Scaling Group

The screenshot shows the 'ASG-web Capacity overview' page. At the top, there's a header with the ASG name and an 'Edit' button. Below it, there are four main sections: 'Desired capacity' (set to 2), 'Scaling limits (Min - Max)' (set to 1 - 4), 'Desired capacity type' (set to 'Units (number of instances)'), and 'Status' (showing 'Updating capacity'). A note below says 'Date created' and provides the creation time. Below these sections are tabs for 'Details', 'Integrations - new', 'Automatic scaling', 'Instance management', 'Instance refresh', 'Activity', and 'Monitoring'. Under the 'Details' tab, there's a 'Launch template' section with fields for 'Launch template' (set to 'lt-0b03608422ca54811 WebAppTemplate'), 'AMI ID' (set to 'ami-02457590d53d576c3'), 'Instance type' (set to 't2.micro'), 'Owner' (set to 'arn:aws:iam::730335665786:user/Alaa-admin'), 'Version' (set to 'Default'), 'Security groups' (set to '-'), 'Security group IDs' (set to 'sg-0bd0270b69ba6e539'), 'Create time' (set to 'Sat Jun 07 2025 12:28:39 GMT+0300 (GMT+03:00)'), 'Description' (set to 'WebAppTemplate description'), 'Storage (volumes)' (set to '-'), 'Key pair name' (set to 'key-pair'), and 'Request Spot Instances' (set to 'Yes'). There's also a link to 'View details in the launch template console'.

Figure 20-Auto Scaling Group

- Target Tracking

The screenshot shows the 'Target Tracking Policy' configuration page. It includes fields for 'Policy type' (set to 'Target tracking scaling'), 'Enabled or disabled' (set to 'Enabled'), 'Execute policy when' (set to 'As required to maintain Average CPU utilization at 50'), 'Take the action' (set to 'Add or remove capacity units as required'), 'Instances need' (set to '300 seconds to warm up before including in metric'), and 'Scale in' (set to 'Enabled').

Figure 21-Added Target Tracking Scaling Policies to the ASG

- Application Load Balancer

ASG-web-LB

▼ Details

Load balancer type Application	Status Active	VPC vpc-0b65e5b0ab3c2e3da	Load balancer IP address type IPv4
Scheme Internet-facing	Hosted zone Z355XDOTRQ7XK	Availability Zones subnet-0ec96fbe25e3dde64 us-east-1b (use1-az2) subnet-08fe1c2f5f039dc14 us-east-1d (use1-az6) subnet-0350cfc608b70ad us-east-1a (use1-az1) subnet-0604a4e230dfe246d us-east-1c (use1-az4)	Date created June 7, 2025, 12:58 (UTC+03:00)
Load balancer ARN arn:aws:elasticloadbalancing:us-east-1:73035665786:loadbalancer/app/ASG-web-LB/2ee635827073fa5f	DNS name ASG-web-LB-1890784892.us-east-1.elb.amazonaws.com (A Record)		

Figure 23-Application Load Balancer details

▼ Details

A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.

Protocol:Port HTTP:80	Load balancer ASG-web-LB	Default actions Forward to target group <ul style="list-style-type: none"> WebApp-TG 1 (100%) Target group stickiness: Off 		
Listener ARN arn:aws:elasticloadbalancing:us-east-1:73035665786:listener/app/ASG-web-LB/2ee635827073fa5f/773ec577a3ba0a02				
Rules	Attributes	Tags		
Listener rules (2) Info				
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.				
Filter rules	Actions (Then)	ARN	Tags	Actions
<input type="checkbox"/> Name tag	Priority ▲	Conditions (If)		
<input type="checkbox"/> Path Pattern is /error/*	2		Return fixed response	ARN 0 tags Edit Delete
			<ul style="list-style-type: none"> Response code: 404 Response body: <!DOCTYPE html> <html> <head> <title>Page Not Found</title> </head> <body> <h1>404 - Page Not Found</h1> <p>Sorry, the page you requested could not be found.</p> </body> </html> Response content type: text/html 	
<input type="checkbox"/> Default	Last (default)	If no other rule applies	Forward to target group	ARN 0 tags Edit Delete
			<ul style="list-style-type: none"> WebApp-TG 1 (100%) Target group stickiness: Off 	

Figure 22 -Added a listener on port 80 (HTTP).

- Target group

WebApp-TG

Details

arn:aws:elasticloadbalancing:us-east-1:730335665786:targetgroup/WebApp-TG/69a6c5043eb272c1

Target type Instance	Protocol : Port HTTP: 80	Protocol version HTTP1	VPC vpc-0b63e5b0ab3c2e3da	
IP address type IPv4	Load balancer ASG-web-LB			
1 Total targets	1 Healthy	0 Unhealthy	0 Unused	0 Initial
	0 Anomalous			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.

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (1) Info

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.

Filter targets	Instance ID	Name	Port	Zone	Health status	Health status details	Administrative o...	Override details	Launch...	Anomaly d...
<input type="checkbox"/>	i-0f0b27d6afb94ab1c		80	us-east-1d (us...)	Healthy	-	No override	No override is curre...	June 8, 20...	Normal

Figure 24- target group named WebApp-TG.

Targets | **Monitoring** | **Health checks** | Attributes | Tags

Health check settings

Protocol HTTP	Path /index.html	Port Traffic port	Healthy threshold 5 consecutive health check successes
Unhealthy threshold 2 consecutive health check failures	Timeout 5 seconds	Interval 30 seconds	Success codes 200

Edit

Figure 25-Configured health checks for the target group to monitor /index.html

- CloudWatch Monitoring and Alarms

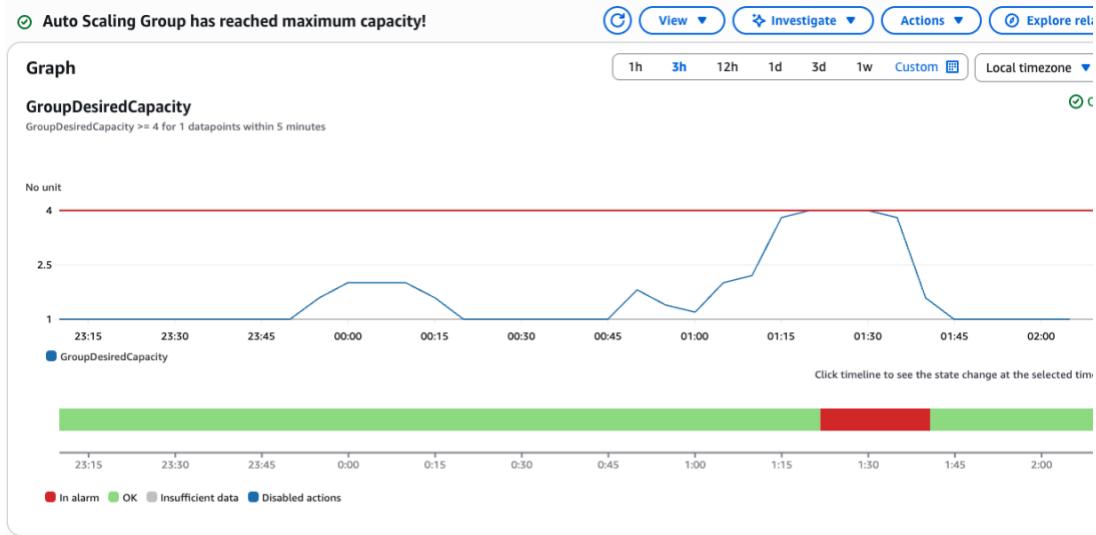


Figure 26-GroupDesiredCapacity = 4 to detect if max capacity is reached.

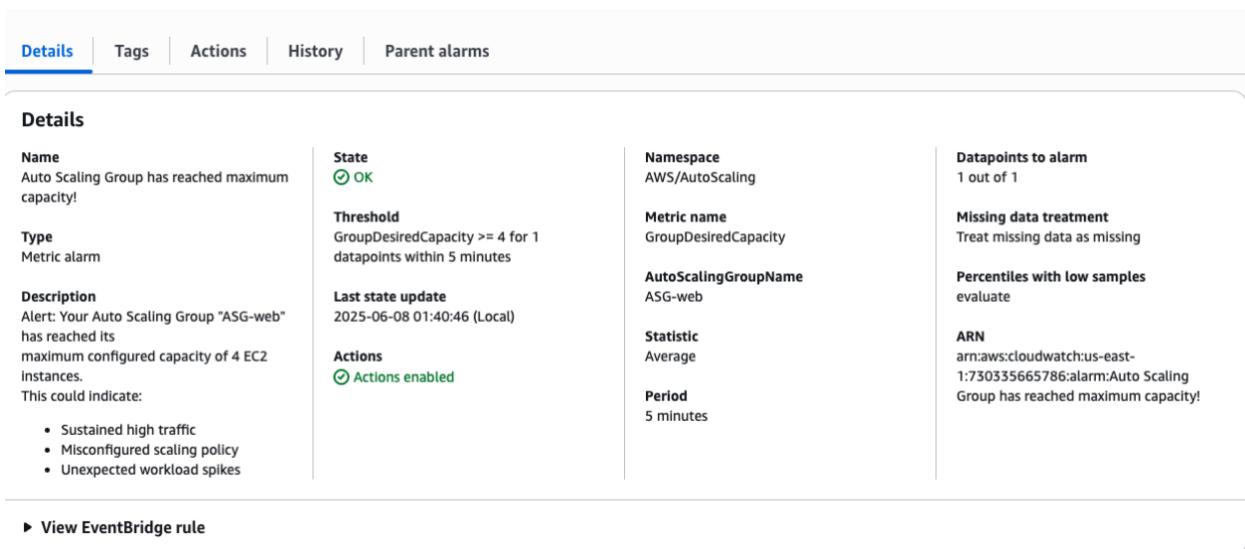


Figure 27- Enabled detailed monitoring

History (5)		
<input type="text"/> Search		
Date (Local)	Type	Description
2025-06-08 01:40:46	State update	Alarm updated from In alarm to OK .
2025-06-08 01:21:46	Action	Successfully executed action arn:aws sns:us-east-1:730335665786:asg-events-topic
2025-06-08 01:21:46	State update	Alarm updated from OK to In alarm .
2025-06-07 13:21:46	State update	Alarm updated from Insufficient data to OK .
2025-06-07 13:20:57	Configuration update	Alarm "Auto Scaling Group has reached maximum capacity!" created

Figure 28-History CloudWatch

Type	Description	Config
Notification	When in alarm, send message to topic "asg-events-topic"	-

Figure 29 - Actions of CloudWatch

- SNS

Send to
alaajam02@gmail.com

Figure 30- SNS To Email

ID	Endpoint	Status	Protocol
d28f43f9-8cb0-4358-8d6e-e60704038867	alaajam02@gmail.com	Confirmed	EMAIL

Figure 31- SNS Details

- Testing with stress Tool

Figure 32-Connected to 2 EC2 instances via SSH and Installed the stress tool.

```
alaaabdalgader -> ec2-user@ip-172-31-6-14:~$ ssh -i ~/Downloads/key-pair.pem...  
stress x86_64 1.0.7-2.amzn2023.0.1  
amazonlinux 34 k  
  
Transaction Summary  
=====  
Install 1 Package  
  
Total download size: 34 k  
Installed size: 68 k  
Downloading Packages:  
stress-1.0.7-2.amzn2023.0.1.x86_64.rpm  
    789 kB/s | 34 kB 00:00  
  
Total  
    442 kB/s | 34 kB 00:00  
Running transaction check  
Transaction check succeeded.  
Running transaction test  
Transaction test succeeded.  
Running transaction  
Preparing :  
    1/1  
Installing : stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
Running scriptlet: stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
Verifying : stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
  
Installed:  
stress-1.0.7-2.amzn2023.0.1.x86_64  
  
Complete!  
[ec2-user@ip-172-31-6-14 ~]$ stress --cpu 2 --timeout 600  
stress: info: [29473] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd  
  
alaaabdalgader -> ec2-user@ip-172-31-87-195:~$ ssh -i ~/Downloads/key-pair.pem...  
Total download size: 34 k  
Installed size: 68 k  
Downloading Packages:  
stress-1.0.7-2.amzn2023.0.1.x86_64.rpm 807 kB/s | 34 kB 00:00  
  
Total 482 kB/s | 34 kB 00:00  
Running transaction check  
Transaction check succeeded.  
Running transaction test  
Transaction test succeeded.  
Running transaction  
Preparing :  
    1/1  
Installing : stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
Running scriptlet: stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
Verifying : stress-1.0.7-2.amzn2023.0.1.x86_64  
    1/1  
  
Installed:  
stress-1.0.7-2.amzn2023.0.1.x86_64  
  
Complete!  
[ec2-user@ip-172-31-87-195 ~]$ sudo dnf install -y epel-release  
Last metadata expiration check: 0:08:39 ago on Sat Jun 7 22:04:49 2025.  
No match for argument: epel-release  
Error: Unable to find a match: epel-release  
[ec2-user@ip-172-31-87-195 ~]$ sudo dnf install -y stress  
Last metadata expiration check: 0:08:43 ago on Sat Jun 7 22:04:49 2025.  
Package stress-1.0.7-2.amzn2023.0.1.x86_64 is already installed.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[ec2-user@ip-172-31-87-195 ~]$ sudo dnf install -y stress  
Last metadata expiration check: 0:08:55 ago on Sat Jun 7 22:04:49 2025.  
Package stress-1.0.7-2.amzn2023.0.1.x86_64 is already installed.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[ec2-user@ip-172-31-87-195 ~]$ stress --cpu 2 --timeout 600  
stress: info: [26650] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

Figure 33-Ran command to simulate high CPU usage for 10 minutes.

Recent alarms info

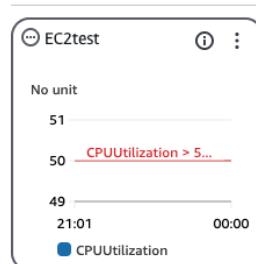
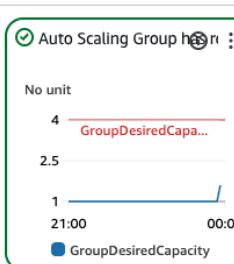


Figure 34-before stress

View recent alarms dashboard



Recent alarms info

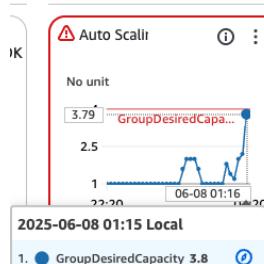
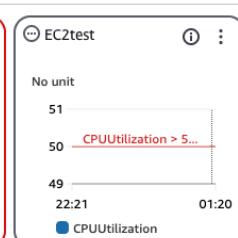


Figure 35-After stress on 2 instance

View recent alarms dashboard



Instances (6) Info



Connect

Instance state ▾

Actions ▾

Launch instances



Find Instance by attribute or tag (case-sensitive)

All states ▾



1



<input type="checkbox"/> Name	Instance ID	Instance state	Instance type	Status check	Alarm
<input type="checkbox"/>	i-08c863c5011c64fec	Running	t2.micro	2/2 checks passed	
<input type="checkbox"/>	i-0fb27d6afb94ab1c	Running	t2.micro	2/2 checks passed	
<input type="checkbox"/>	i-0b3cf52b44b82249b	Running	t2.micro	2/2 checks passed	
<input type="checkbox"/>	i-0d7f8dbd92003fc5d	Terminated	t2.micro	-	
<input type="checkbox"/>	i-0d4e4cde509d214f7	Terminated	t2.micro	-	
<input type="checkbox"/>	i-0ab4a382d2bce0191	Running	t2.micro	2/2 checks passed	

Figure 36-reaching maximum configured capacity

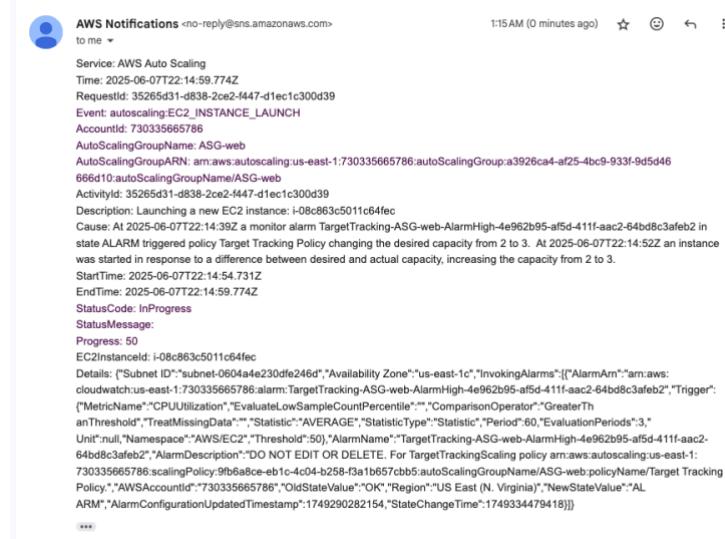


Figure 37-Auto Scaling Trigger: New EC2 Instance Launched Due to High CPU

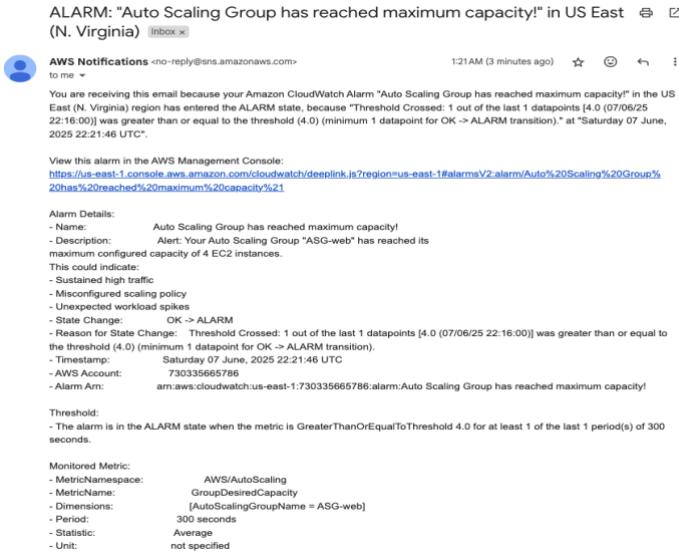


Figure 38- Auto Scaling Alert email Notification: Maximum Capacity Reached

Status	Description	Cause	Start time	End time
⌚ Successful	Launching a new EC2 instance: i-0f0b27d6afb94ab1c	At 2025-06-07T22:16:39Z a monitor alarm TargetTracking-ASG-web-AlarmHigh-4e962b95-af5d-411f-aac2-64bd8c3afeb2 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 4. At 2025-06-07T22:16:53Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 3 to 4.	2025 June 08, 01:16:55 AM +03:00	2025 June 08, 01:22:27 AM +03:00
⌚ Successful	Launching a new EC2 instance: i-08c863c5011c64fec	At 2025-06-07T22:14:39Z a monitor alarm TargetTracking-ASG-web-AlarmHigh-4e962b95-af5d-411f-aac2-64bd8c3afeb2 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 3. At 2025-06-07T22:14:52Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2025 June 08, 01:14:54 AM +03:00	2025 June 08, 01:19:59 AM +03:00
⌚ Successful	Launching a new EC2 instance: i-0ab4a382d2bce0191	At 2025-06-07T22:04:11Z a user request update of AutoScalingGroup constraints to min: 1, max: 4, desired: 2 changing the desired capacity from 1 to 2. At 2025-06-07T22:04:19Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2025 June 08, 01:04:20 AM +03:00	2025 June 08, 01:04:26 AM +03:00
⌚ Successful	Terminating EC2 instance: i-0d4e4cde509d214f7	At 2025-06-07T21:57:07Z a monitor alarm TargetTracking-ASG-web-AlarmLow-51415ad4-af53-40c1-b8a6-a880d4cd45e1 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 1. At 2025-06-07T21:57:12Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2025-06-07T21:57:12Z instance i-0d4e4cde509d214f7 was selected for termination.	2025 June 08, 12:57:12 AM +03:00	2025 June 08, 01:02:55 AM +03:00
⌚ Successful	Launching a new EC2 instance: i-0d4e4cde509d214f7	At 2025-06-07T21:51:12Z a user request update of AutoScalingGroup constraints to min: 1, max: 4, desired: 2 changing the desired capacity from 1 to 2. At 2025-06-07T21:51:22Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2025 June 08, 12:51:24 AM +03:00	2025 June 08, 12:51:41 AM +03:00

Figure 39- Auto Scaling Activity Log: EC2 Launch and Termination Events

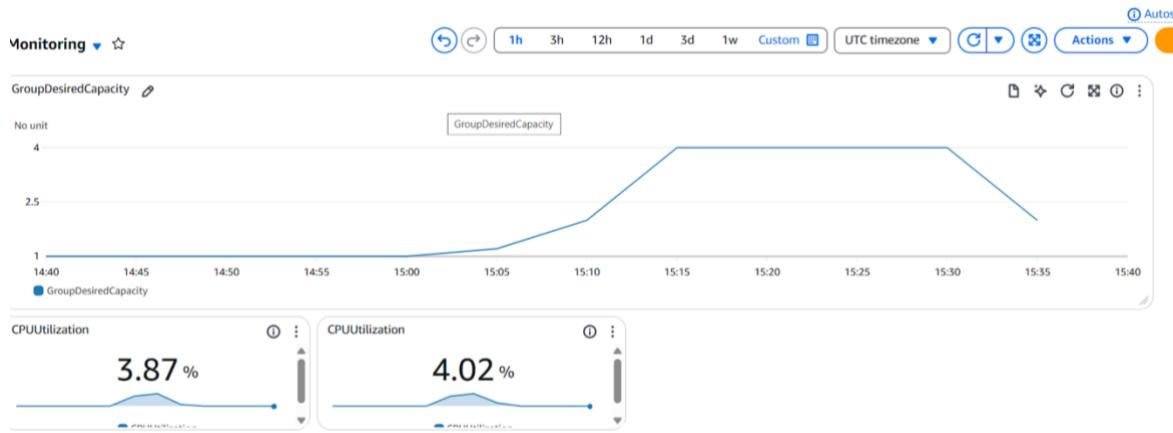


Figure 40-Analyze Scaling Behavior and Trends from CloudWatch Metrics

From the CloudWatch metrics, we can see how the Auto Scaling Group reacted to the increase in workload. At first, the desired capacity was set to 1 instance because everything was running smoothly. But once the CPU usage started to go up (around 4%), the Auto Scaling policy kicked in and started adding more instances — eventually reaching 4, which was the maximum allowed. After the stress test ended, the system automatically reduced the number of instances again. This shows that Auto Scaling is working well, but it also means the CPU threshold might be a bit too low. To make it work more efficiently, we could adjust the CPU threshold or add a cooldown period so the system doesn't scale up too quickly.

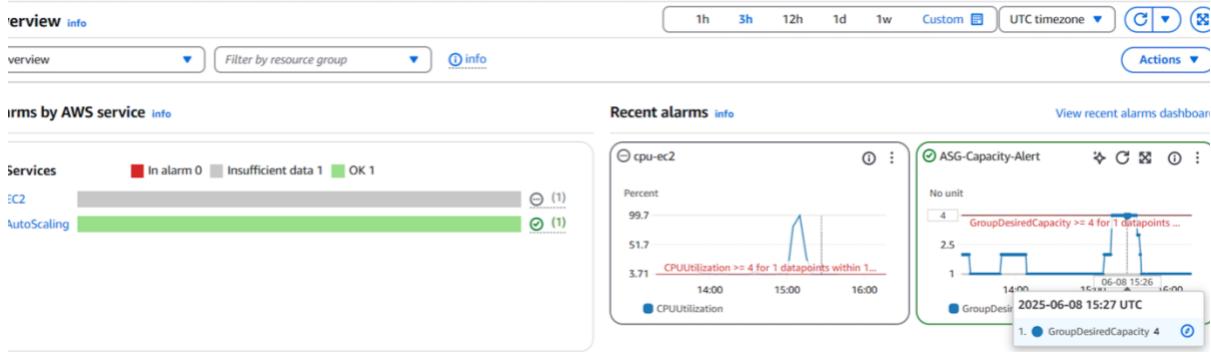


Figure 41-Analyzing Scaling Behavior Using CloudWatch Metrics

we can clearly see how the Auto Scaling Group responded to the CPU utilization. When the CPU usage spiked (as shown in the left graph), the system reacted by increasing the desired capacity (middle graph) to handle the higher load. It added instances until it reached the maximum configured limit of 4.

This trend confirms that the scaling policy worked correctly based on the defined CPU threshold (in this case, around 4%). Once the workload dropped again, the desired capacity began to decrease, showing that the system is also able to scale down automatically. This kind of visibility helps ensure that our infrastructure can react efficiently to real-time demand.

e. Conclusion

In this scenario, I successfully deployed a scalable and monitored web application using EC2, an Application Load Balancer (ALB), Auto Scaling Group (ASG), CloudWatch, and S3 integration. The setup dynamically adjusted the number of EC2 instances based on CPU utilization, ensuring high availability under stress conditions.

During testing, the Auto Scaling Group scaled out and in correctly, reacting to increased load and later terminating excess instances when demand dropped. Notifications via SNS confirmed each scaling event, proving the infrastructure was responsive and functioning as expected.

To further improve efficiency, I suggest fine-tuning the scaling thresholds. For example:

- **Adding additional CloudWatch metrics**, such as memory or network utilization, could improve decision-making beyond just CPU.

- **Monitoring the average response time** per instance in CloudWatch to complement CPU metrics and avoid premature scaling.

4) Scenario #3 – Secure Linux Server Setup on AWS

a. Objective

The goal of this scenario is to set up a secure Linux-based EC2 server that hosts isolated environments for three departments: Human Resources (HR), Development (Dev), and Operations (Ops). Each department should only access its own files. Additionally, the solution must include an automated backup process that archives each department's data and pushes it daily to a private GitHub repository using Bash scripts and cron jobs.

b. Implementation Steps

1) Launched the EC2 Instance

- I launched a new EC2 instance using **Amazon Linux 2** for its compatibility and long-term support.
- Chose instance type **t2.micro** to stay within the AWS free tier.
- I **generated a new key pair** to securely connect to the instance via SSH.
- I created a **custom security group** named SG-web that allows SSH access (port 22) **only from my IP address**.
- This security group was attached during instance launch to ensure limited and safe access.

2) Connected to EC2 Using SSH

- After launching the instance, I connected using the terminal command:

```
ssh -i ~/Downloads/secure-key.pem ec2-user@<EC2-Public-IP>
```

3) Created Folder Structure for Departments

- Once connected, I created isolated directories for each department under a main /company directory:

```
sudo mkdir -p /company/hr /company/dev /company/ops
```

4) Created User Groups and Users

- I created user groups for each department:

```
sudo groupadd HR
```

- Then I added two users per department and added them to their groups:

```
sudo useradd hr1 -G HR sudo useradd hr2 -G HR
```

5) Configured Directory Permissions

- I made each department folder owned by the root user but grouped to the corresponding department:

```
sudo chown root:HR /company/hr
```

- Set strict permissions so only the corresponding group has access:

```
sudo chmod 770 /company/hr
```

6) Wrote GitHub Backup Script

- I created a Bash script at /usr/local/bin/github_backup.sh to archive and push daily backups.

7) Set Up Cron Job for Daily Automation

- I used the crontab to schedule daily backups at 2 AM.

8) Validated the Automation

To verify that the backups were running and being pushed correctly:

- I reviewed the /var/log/github_backup.log file.
- I checked the private GitHub repository commit history to confirm daily uploads.

d. Screenshots

The screenshot shows the 'Instance summary' page for an EC2 instance named 'i-0eaefacde8a9c6b9d'. The instance is in the 'Secure Department' group and has been updated less than a minute ago. Key details include:

- Public IPv4 address:** 18.207.248.233 (with a link to open address)
- Private IPv4 address:** 172.31.90.205
- Public DNS:** ec2-18-207-248-233.compute-1.amazonaws.com (with a link to open address)
- Instance state:** Running
- Instance type:** t2.micro
- VPC ID:** vpc-0b1aeb7d9e4f042b5 (with a link to view)
- Subnet ID:** subnet-0a6266711d78642a7 (with a link to view)
- Instance ARN:** arn:aws:ec2:us-east-1:211839440217:instance/i-0eaefacde8a9c6b9d
- Elastic IP addresses:** None
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendations.
- Auto Scaling Group name:** None
- Managed:** false

Figure 42-EC2 instance using Amazon Linux 2

The screenshot shows the 'Security' tab of a custom security group. The group has the following details:

- Owner ID:** 211839440217
- Launch time:** Tue Jun 10 2025 23:07:00 GMT+0300 (GMT+03:00)
- Security groups:** web-SG (with a link to view)

Inbound rules:

Name	Security group rule ID	Port range	Protocol	Source	Security groups
-	sgr-0ddde00b9064efb89	22	TCP	94.249.50.196/32	web-SG (with a link to view)

Outbound rules:

Name	Security group rule ID	Port range	Protocol	Destination	Security groups
-	sgr-0835aaa6ad6acc352	All	All	0.0.0.0/0	web-SG (with a link to view)

Figure 43-custom security group that allows SSH access (port 22) only from my IP address

```

alaa@alaa: $ ssh -i /home/alaa/Desktop/KP-secure-dep.pem ec2-user@18.207.248.233
,
~\_ #####      Amazon Linux 2
~~ \#####\
~~ \###|      AL2 End of Life is 2026-06-30.
~~ \#/ __>
~~ V~, '-->
~~ / A newer version of Amazon Linux is available!
~~ ._. _/ Amazon Linux 2023, GA and supported until 2028-03-15.
~/m/ https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-90-205 ~]$ sudo yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core                                         | 3.6 kB     00:00

Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.47.1-1.amzn2.0.2 will be installed
--> Processing Dependency: git-core = 2.47.1-1.amzn2.0.2 for package: git-2.47.1-1.amzn2.0.2.x86_64
--> Processing Dependency: git-core-doc = 2.47.1-1.amzn2.0.2 for package: git-2.47.1-1.amzn2.0.2.x86_64
--> Processing Dependency: perl-Git = 2.47.1-1.amzn2.0.2 for package: git-2.47.1-1.amzn2.0.2.x86_64
--> Processing Dependency: perl(Git) for package: git-2.47.1-1.amzn2.0.2.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.47.1-1.amzn2.0.2.x86_64
--> Running transaction check
--> Package git-core.x86_64 0:2.47.1-1.amzn2.0.2 will be installed
--> Package git-core-doc.noarch 0:2.47.1-1.amzn2.0.2 will be installed
--> Package perl-Git.noarch 0:2.47.1-1.amzn2.0.2 will be installed

```

Figure 44-Connected to EC2 Using SSH

```

Complete!
[ec2-user@ip-172-31-90-205 ~]$ sudo mkdir -p /company/hr /company/dev /company/ops
[ec2-user@ip-172-31-90-205 ~]$ sudo groupadd HR
[ec2-user@ip-172-31-90-205 ~]$ sudo groupadd Dev
[ec2-user@ip-172-31-90-205 ~]$ sudo groupadd Ops
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd hr1 -G HR
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd hr2 -G HR
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd dev1 -G Dev
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd dev2 -G Dev
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd ops1 -G Ops
[ec2-user@ip-172-31-90-205 ~]$ sudo useradd ops2 -G Ops

```

Figure 45>Create Groups and Users-assign them to their respective groups

```

[ec2-user@ip-172-31-90-205 ~]$ sudo chown :HR /company/hr
[ec2-user@ip-172-31-90-205 ~]$ sudo chown :Dev /company/dev
[ec2-user@ip-172-31-90-205 ~]$ sudo chown :Ops /company/ops
[ec2-user@ip-172-31-90-205 ~]$ sudo chmod 770 /company/hr
[ec2-user@ip-172-31-90-205 ~]$ sudo chmod 770 /company/dev
[ec2-user@ip-172-31-90-205 ~]$ sudo chmod 770 /company/ops

```

Figure 46-Ownership and Permissions

```
[ec2-user@ip-172-31-90-205 ~]$ git clone git@github.com:eylem2002/secure-department.git
Cloning into 'secure-department'...
Warning: Permanently added the ECDSA host key for IP address '140.82.112.3' to the list of known hosts.
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 47 (delta 21), reused 23 (delta 10), pack-reused 0 (from 0)
Receiving objects: 100% (47/47), 11.01 KiB | 239.00 KiB/s, done.
Resolving deltas: 100% (21/21), done.
```

Figure 48-clone to my repo on GitHub

```
[ec2-user@ip-172-31-90-205 ~]$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/github_backup -C "backup_key" -N ""
Generating public/private rsa key pair.
/home/ec2-user/.ssh/github_backup already exists.
Overwrite (y/n)? y
Your identification has been saved in /home/ec2-user/.ssh/github_backup.
Your public key has been saved in /home/ec2-user/.ssh/github_backup.pub.
The key fingerprint is:
SHA256:cXxkXRuZr02np53ALZHE8zfCJdfd0Mfmn4Cd+7XJV0Q backup_key
The key's randomart image is:
+---[RSA 4096]---+
|          o.o=|
|         . o. o=E|
|        . o.+B+*o|
|       o o**=.o|
|      S    =+++=|
|     ...+oB|
|    +oo*|
|   +*+|
|  .+o|
+---[SHA256]---+
```

Figure 47-SSH Key Generation for GitHub Backup

```
[ec2-user@ip-172-31-90-205 ~]$ cat ~/.ssh/github_backup.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQADQDZIu3qaO3LSzUds4L2hzY+fV/CdKRnJh72j2u2DNA6IuWo1cH/L20JQJji7uSPdWJI2iy7MrbohQcm9QI/Yea0Y/vS0V
Qjj7CRF0+Qz+XlnOMzN3K0RBGAGLlisLmaKgzaJk/Vy+rb51w1F84wYhMs1xYMFssqTELckrjvj608+C01XCU0XNC3g/he2g5jea1bEJJRwh03mjj47GwbJ5nszoC/AM
Pircu3ZtheL53yBzQzPiil/81wEaEyo2zQzMzllT0laeJsxBdkmKkp6g6bj17mmNmhor0VW59hyQQITSMTXrqeCJMii1Q6eQxrWEZtaeubs2u612oihw45odLxtpzcleSk
9J9CKkt3/MccQD52bDdkQwy3youRwNcCzz8iNzGJB/xFSQ/Q823+hCeMgAzGh+amwHukCQgI2en03jv20sJu2Q7y93R8PeCXADjDoMpAhVq/FuXCEvC/kdyc7bz0cS65M
aAocfEecH+5Yz/0pyreYlcEyW0FgG/az9xnaTkd52oLPnrQ/yikXuxu7QMSApvQ/k8xeAC55i2SaeLbFDYOb/AzUqambHG86XiYSsFTugPeF0qMo2tosHSOyFThjc6a0nV+
```

Figure 49-Viewing and Configuring SSH Deploy Key

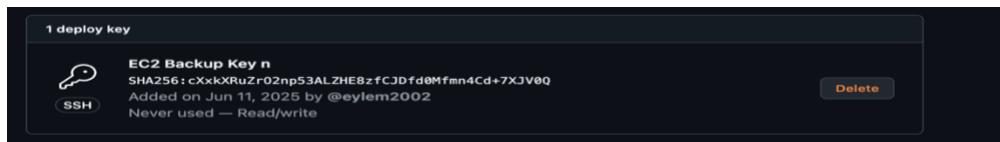
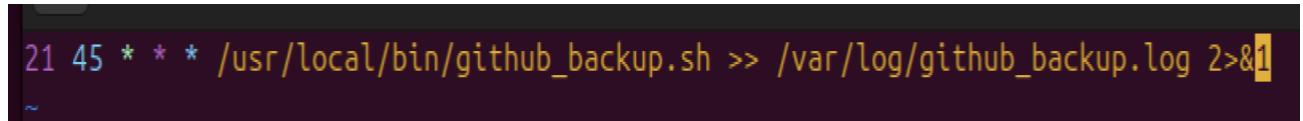


Figure 50-Add key on the repo

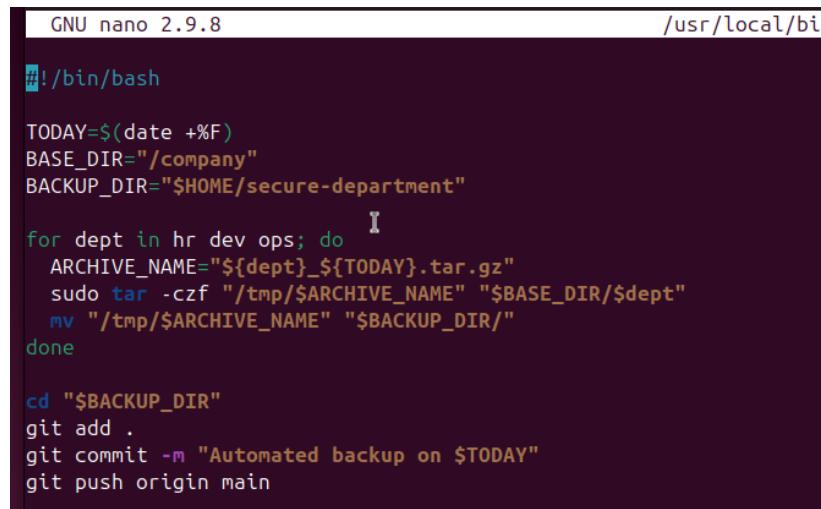
```
[ec2-user@ip-172-31-90-205 ~]$ chmod 600 ~/.ssh/github_backup
[ec2-user@ip-172-31-90-205 ~]$ ssh -T git@github.com
Hi eylem2002/secure-department! You've successfully authenticated, but GitHub does not provide shell access.
```

Figure 51-Connection to GitHub is Done



```
21 45 * * * /usr/local/bin/github_backup.sh >> /var/log/github_backup.log 2>&1
```

Figure 52 -Scheduled Cron Job for Daily GitHub Backup at 9:45 PM



```
GNU nano 2.9.8 /usr/local/bi

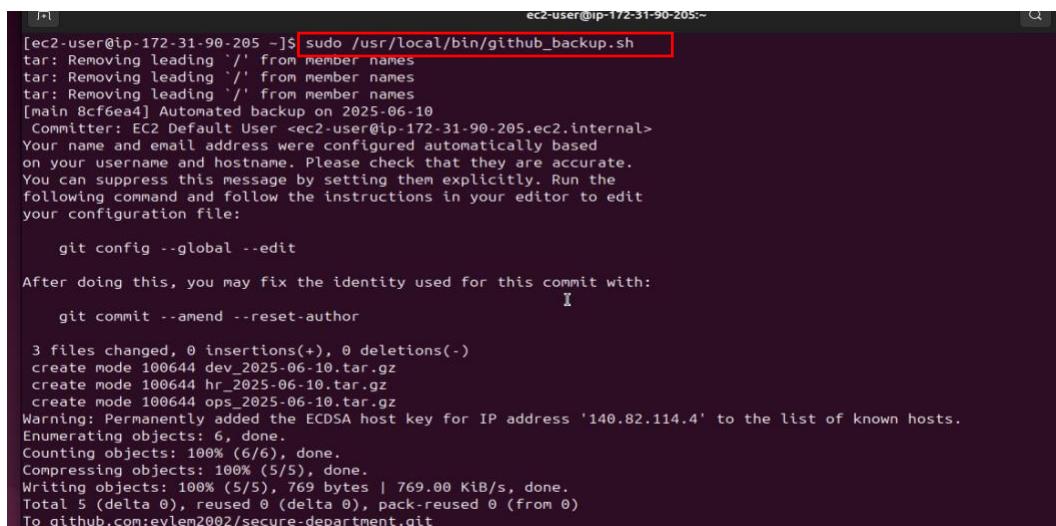
#!/bin/bash

TODAY=$(date +%F)
BASE_DIR="/company"
BACKUP_DIR="$HOME/secure-department"

for dept in hr dev ops; do
    ARCHIVE_NAME="${dept}_${TODAY}.tar.gz"
    sudo tar -czf "/tmp/$ARCHIVE_NAME" "$BASE_DIR/$dept"
    mv "/tmp/$ARCHIVE_NAME" "$BACKUP_DIR/"
done

cd "$BACKUP_DIR"
git add .
git commit -m "Automated backup on $TODAY"
git push origin main
```

Figure 53-Backup script



```
[ec2-user@ip-172-31-90-205 ~]$ sudo /usr/local/bin/github_backup.sh
tar: Removing leading '/' from member names
tar: Removing leading '/' from member names
tar: Removing leading '/' from member names
[main 8cf6ea4] Automated backup on 2025-06-10
Committer: EC2 Default User <ec2-user@ip-172-31-90-205.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:
git commit --amend --reset-author

3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 dev_2025-06-10.tar.gz
create mode 100644 hr_2025-06-10.tar.gz
create mode 100644 ops_2025-06-10.tar.gz
Warning: Permanently added the ECDSA host key for IP address '140.82.114.4' to the list of known hosts.
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 769 bytes | 769.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:ylem2002/secure-department.git
```

Figure 54-Test the script manually.

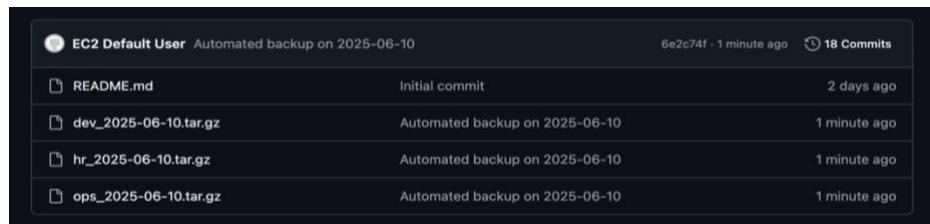


Figure 55 -Automated Daily Backup Files Uploaded to Private GitHub Repository

```

CronTab: no changes made to CronTab
[ec2-user@ip-172-31-90-205 ~]$ cat /var/log/github_backup.log
tar: Removing leading '/' from member names
tar: Removing leading '/' from member names
tar: Removing leading '/' from member names
[main 6e2c74f] Automated backup on 2025-06-10
  Committer: EC2 Default User <ec2-user@ip-172-31-90-205.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

  git config --global --edit

After doing this, you may fix the identity used for this commit with:

  git commit --amend --reset-author

  3 files changed, 0 insertions(+), 0 deletions(-)
Warning: Permanently added the EDSSA host key for IP address '140.82.113.3' to the list of known hosts.
To github.com:eylem2002/secure-department.git
  8cf6ea4..6e2c74f main -> main
  You have new mail in /var/spool/mail/ec2-user

```

Figure 56-Log Output Showing Successful GitHub Backup Commit

e. Conclusion

This scenario successfully demonstrates the ability to configure a secure and isolated Linux environment for multiple departments using EC2. I also built a working automation flow that archives and backs up data daily to GitHub with zero manual intervention.

Suggestions for improvement:

- Add automated email alerts if the backup script fails.
- Encrypt backup files before pushing to GitHub for additional security.
- Use AWS Systems Manager instead of manual SSH to improve access control and logging.
- Expand the solution to sync with an S3 bucket or centralized logging system in future enterprise environments.

5. Scenario #4 – Serverless HealthBooking on AWS

a. Objective

The goal of this project was to design and deploy a serverless web-based medical appointment booking system using AWS-native tools. The application enables patients to book appointments and allows doctors to manage them. All backend components were built using AWS serverless services, and the frontend was deployed using AWS Amplify.

b. Implementation Steps

1) Designed and Created DynamoDB Tables

- I began by designing the data model for the system. The solution relies on two DynamoDB tables , Both tables were created through the DynamoDB Console using on-demand capacity mode to ensure cost-efficiency and scalability.

- The **Appointments** table stores appointment details including patient name, symptoms, status, and time slot.
- The **Slots** table maintains the status of available time slots to prevent overbooking.

2) Developed Lambda Functions and APIs

- I developed five Lambda functions to manage the backend logic of the application:
 - **GET Time Slots:** Retrieves available (unbooked) time slots.
 - **POST Appointments:** Creates a new appointment, books the time slot, and triggers an SNS email notification.
 - **GET Appointments:** Retrieves all appointment records for the doctor/admin view.
 - **PATCH Appointment Status:** Allows status updates (e.g., Pending, In Progress, Completed) for any appointment.
 - **Reset Daily Bookings:** Scheduled using EventBridge to run daily at 5 PM UTC. It clears all current day bookings and resets slot availability.
 - Each Lambda function was integrated with API Gateway and tested thoroughly before frontend integration.

3) Configured Notifications Using Amazon SNS

- I created an SNS topic to notify the doctor whenever a new appointment is made or the daily reset occurs.
- My email was subscribed to the topic to receive real-time notifications, ensuring visibility into system activity without manual checking.

4) Set Up Daily Automation with EventBridge

- A scheduled rule was set up using Amazon EventBridge to invoke the **Reset Daily Bookings** Lambda function daily at 5 PM UTC.
- The function deletes all appointments for the current day and updates slot statuses to available.
- A confirmation email is sent via SNS after every reset operation to ensure that the system's availability is maintained for the next day.

5) Built and Deployed the Frontend Using AWS Amplify

- The frontend application was built using Vue.js with Vite as the build tool and PrimeVue as the UI framework.

- I forked the provided GitHub repository and updated all API URLs to match my deployed API Gateway endpoints.
- The repository was then connected to AWS Amplify, which was configured for automatic CI/CD deployment triggered on every push to the main branch.

6) Implemented Two Main Web Pages

- **Book Appointment Page**
 - Allows users to enter their name, describe symptoms, and choose a time slot.
 - Submits the booking information to the **POST Appointments** API for processing.
- **Appointments Dashboard**
 - Enables doctors or admins to view all appointments.
 - Includes options to update the status of any appointment using the PATCH API.

7) Set Up Monitoring and Logging Using CloudWatch

- I enabled logging for all Lambda functions through AWS CloudWatch to ensure proper visibility into errors and execution logs.
- Additionally, I configured CloudWatch metrics to monitor API response times and function invocations, helping me debug and improve performance throughout the project lifecycle.

d. Screenshots

Table: Appointments - Items returned (4)

Scan started on June 20, 2025, 12:31:35

<input type="checkbox"/> appointmentId (String)	createdAt	patientName	slot	status	symptoms
861ba991-9bf5-47ba-bf37-46172b977567	2025-06-20...	Alaa	9 - 10	Completed	Headachfatigue
63308838-27fb-4034-902d-7b9b29780aa5	2025-06-20...	Lina	11 - 12	In Progress	Cough and fever
57ea89e-962f-4fab-8327-a140da8a658f	2025-06-20...	Omar	10 - 11	In Progress	Stomach pain
6c7de744-3b5b-451b-91c8-3e17a99daf1f	2025-06-20...	Yousef	8 - 9	Pending	Cold and sore throat

Figure 57-Appointment table

Table: Slots - Items returned (5)

Scan started on June 20, 2025, 12:26:27

	slot (String)	isBooked
<input type="checkbox"/>	11 - 12	false
<input type="checkbox"/>	10 - 11	false
<input type="checkbox"/>	12 - 1	false
<input type="checkbox"/>	8 - 9	false
<input type="checkbox"/>	9 - 10	true

Figure 58-slots table

Function name

[updateStatusAppointment](#)
[getAvailableSlots](#)
[getAllAppointments](#)
[createAppointment](#)
[resetDailyBookings](#)

Figure 59-All Lambda Functions

🕒 Executing function: succeeded ([Logs](#))
[Details](#)

Test event [Info](#)

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but can't save it.

Test event action

Create new event Edit saved event

Event name

Event JSON

```

1 [{"body": "{\"patientName\": \"lilo\", \"symptoms\": \"Headache and dizziness\", \"slot\": \"9 - 10\"}"}]
2
3
4

```

Figure 60- Test create Appointment Function

Execution role

Role name [readUpdateSlots-writeAppointment-Role](#)

[Edit](#) [View role document](#)

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs 3 actions, 2 resources	▲
Amazon CloudWatch Logs 5 actions, 2 resources	✓
Amazon DynamoDB 7 actions, 2 resources	
Amazon SNS 1 action, 1 resource	

Figure 61-All permissions for create Appointment

Amazon CloudWatch Logs	
3 actions, 2 resources	
Resource	Actions
arn:aws:logs:us-east-1:730335665786:*	Allow: logs>CreateLogGroup
arn:aws:logs:us-east-1:730335665786:log-group:createAppointmentLog:*	Allow: logs>CreateLogStream Allow: logs:PutLogEvents

Figure 62-Cloudwatch permission for create Appointment

Resource summary	
To view the resources and actions that your function has permission to access, choose a service.	
Amazon SNS	
1 action, 1 resource	
By action	By resource
Resource	Actions
arn:aws:sns:us-east-1:730335665786:doctor-booking	Allow: sns>Publish

Figure 63-SNS permission for the create Appointment

Resource summary	
To view the resources and actions that your function has permission to access, choose a service.	
Amazon DynamoDB	
7 actions, 2 resources	
By action	By resource
Resource	Actions
arn:aws:dynamodb:us-east-1:730335665786:table/Appointments	Allow: dynamodb:PutItem Allow: dynamodb>DeleteItem Allow: dynamodb:UpdateItem
arn:aws:dynamodb:us-east-1:730335665786:table/Slots	Allow: dynamodb:GetItem Allow: dynamodb:Scan Allow: dynamodb:Query Allow: dynamodb:UpdateItem Allow: dynamodb:GetRecords

Figure 64-DynamoDB permission for create Appointment

Executing function: succeeded (logs)
▼ Details
<pre>{ "statusCode": 200, "headers": { "Content-Type": "application/json" }, "body": "[{"isBooked": false, "slot": "12 - 1"}]" }</pre>

Figure 65 - Test get Available Slots function

Execution role

Role name
GetAVSlotsRole [Edit](#)

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon DynamoDB
4 actions, 1 resource

By action [By resource](#)

Resource	Actions
arn:aws:dynamodb:us-east-1:730335665786:table/Slots	Allow: dynamodb:GetItem Allow: dynamodb:Scan Allow: dynamodb:Query Allow: dynamodb:GetRecords

ⓘ Lambda obtained this information from the following policy statements:

- Managed policy ReadAccessFromSlotsTable, statement VisualEditor0

Figure 66-Dynamo Permission for get Available Slots

Executing function: succeeded ([Logs](#))

▼ Details

```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "body": "[{"createdAt": "\2025-06-20T09:26:06.786521\\"", "appointmentId": "\861ba991-9bf5-47ba-bf37-46172b977567\\"", "slot": "\9 - 10\", \"patientName\": \"Alaa\", \"symptoms\": \"Headachfatigue\", \"status\": \"Completed\", \"createdAt\": \\"2025-06-20T09:27:02.525793\\"", \"appointmentId\": \\"63308838-27fb-4034-902d-7b9b29780a05\\"", \"slot\": \\"11 - 12\\\", \"patientName\": \"Lina\", \"symptoms\": \"Cough and fever\", \"status\": \"In Progress\", \"createdAt\": \\"2025-06-20T09:27:16.325332\\"", \"slot\": \\"8 - 9\\\", \"appointmentId\": \\"6c7de744-305b-451b-91c8-3e17a99da1f1\\"", \"patientName\": \"Yousef\", \"symptoms\": \"Cold and sore throat\", \"status\": \"Pending\", \"createdAt\": \\"2025-06-20T09:27:09.545316\\"", \"appointmentId\": \\"57ea0a9e-962f-8327-a140dd0a658f\\"", \"slot\": \\"10 - 11\\\", \"patientName\": \"Omar\", \"symptoms\": \"Stomach pain\", \"status\": \"In Progress\"}]"
```

Summary

Figure 67 -Test get All Appointments function

Role name
[readOnlyFromAppointment-Role1](#)

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon DynamoDB
4 actions, 1 resource

By action [By resource](#)

Resource	Actions
arn:aws:dynamodb:us-east-1:730335665786:table/Appointments	Allow: dynamodb:GetItem Allow: dynamodb:Scan Allow: dynamodb:Query Allow: dynamodb:GetRecords

Figure 68 - DyanmoDB permission for get All Appointments

Executing function: succeeded (logs [?](#))
 Details

Test event [Info](#)
 To Invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event

Test event action
 Create new event

Event name
 update_appointment_status

Event JSON

```

1 * {
2   "httpMethod": "PATCH",
3   "pathParameters": {
4     "appointmentId": "861ba991-9bf5-47ba-bf37-46172b977567"
5   },
6   "body": "{\"status\": \"Completed\"}"
7 }
8
  
```

Figure 69-Test update Status Appointment function

Execution role [Edit](#) | [Copy](#)

Role name
 UpdateSlotsAppointment-Role [?](#)

Resource summary
 To view the resources and actions that your function has permission to access, choose a service.

Resource	Actions
arn:aws:dynamodb:us-east-1:730335665786:table/Appointments	Allow: dynamodb:UpdateItem
arn:aws:dynamodb:us-east-1:730335665786:table/Slots	Allow: dynamodb:UpdateItem

By action [By resource](#)

Figure 70 -DynamoDB permission for update Status Appointment

Executing function: succeeded (logs [?](#))
 Details

```
{
  "statusCode": 200,
  "body": "Slots and appointments reset successfully, notification sent."
}
```

Figure 71-Test for reset daily bookings

Execution role

Role name
[deleteAppointmentUpdateSlotsRole](#)

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon DynamoDB
4 actions, 2 resources

Q |

Amazon CloudWatch Logs
3 actions, 2 resources

Amazon DynamoDB
4 actions, 2 resources

Amazon SNS
1 action, 1 resource

[View document](#)

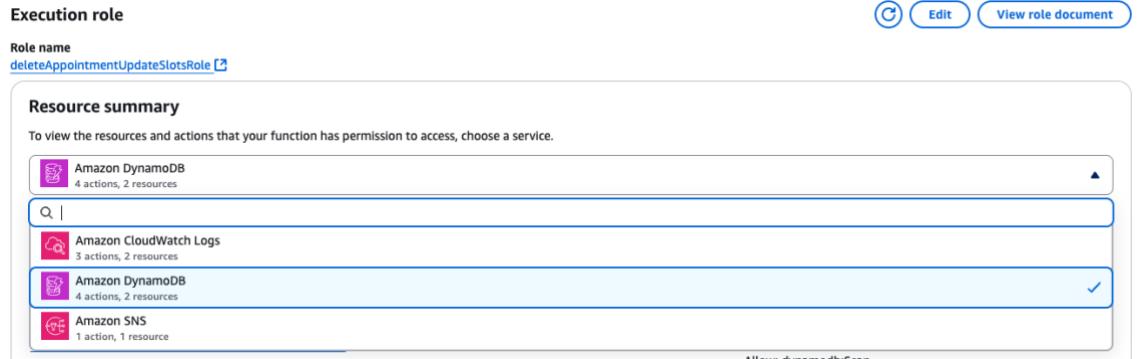


Figure 72-All permission for reset daily bookings

Execution role

Role name
[deleteAppointmentUpdateSlotsRole](#)

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon DynamoDB
4 actions, 2 resources

[By action](#) [By resource](#)

Resource	Actions
arn:aws:dynamodb:us-east-1:730335665786:table/Slots	Allow: dynamodb:BatchWriteItem Allow: dynamodb:PutItem Allow: dynamodb:Scan
arn:aws:dynamodb:us-east-1:730335665786:table/Appointments	Allow: dynamodb:BatchWriteItem Allow: dynamodb:PutItem Allow: dynamodb:DeleteItem Allow: dynamodb:Scan

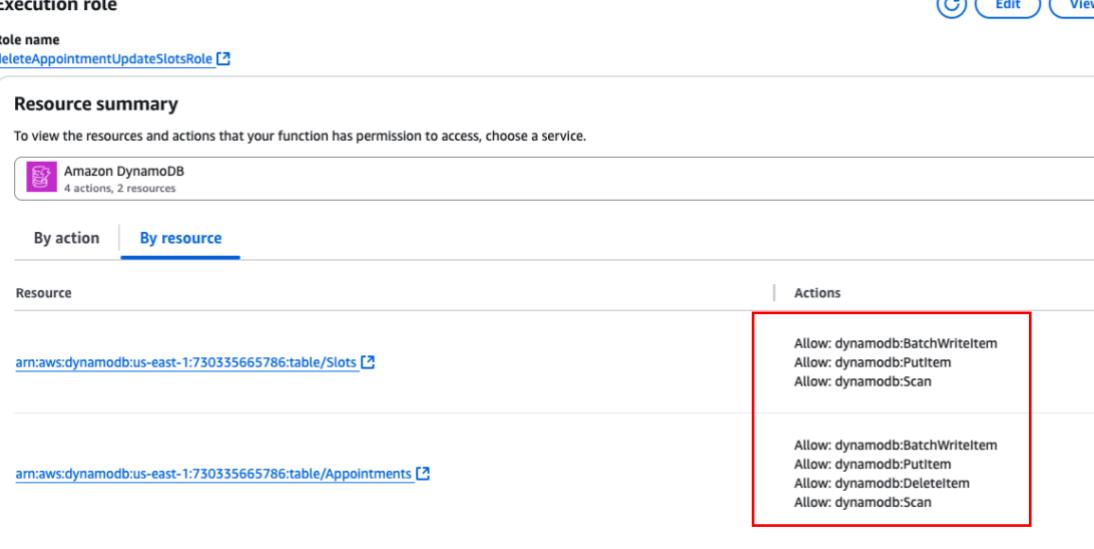


Figure 73-DynamoDB permission for reset daily bookings

Resources

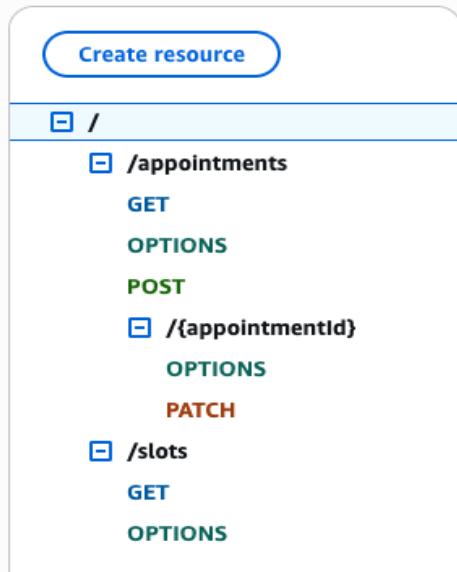


Figure 74-All my APIs

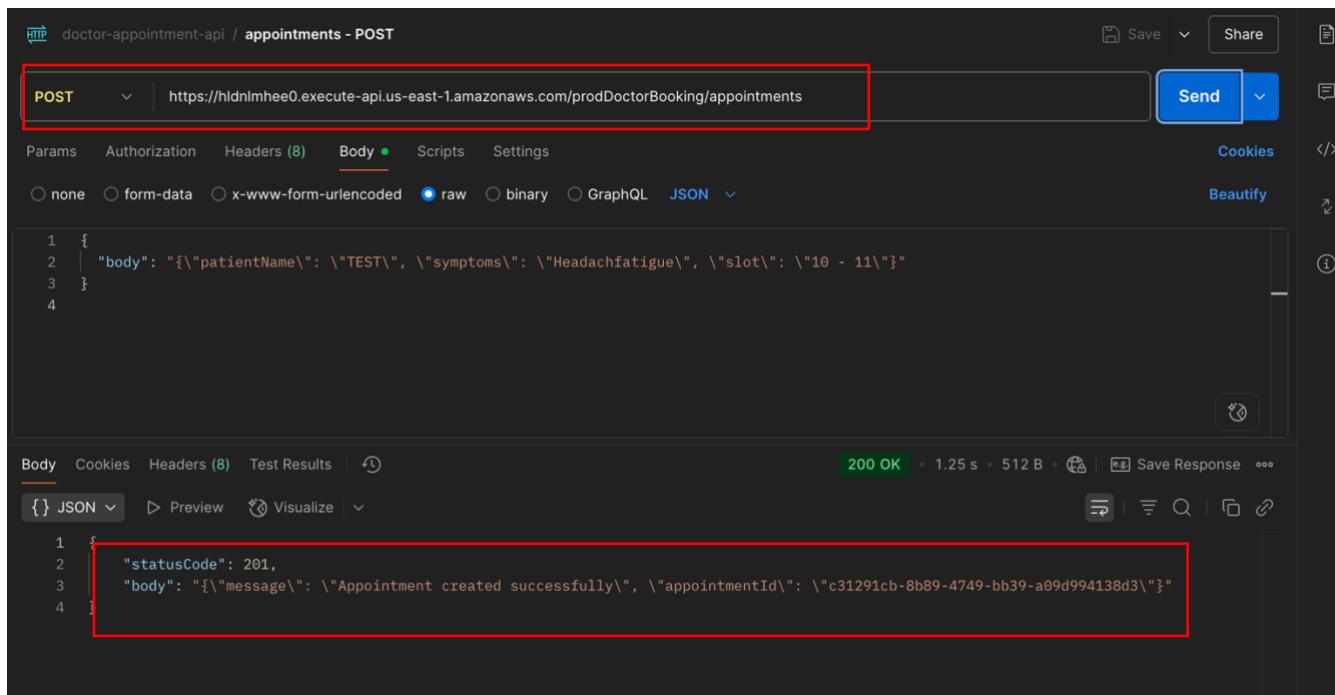


Figure 75-Book an Appointment API Test Postman

The screenshot shows a POST request to <https://hdnlmhee0.execute-api.us-east-1.amazonaws.com/prodDoctorBooking/appointments/c31291cb-8b89-4749-bb39-a09d994138d3>. The Body tab contains the following JSON:

```

1 | {
2 |     "status": "In Progress"
3 | }
4 |

```

The response status is 200 OK, with a message: "Status updated successfully."

Figure 76-Update Appointment Status API Test Postman

The screenshot shows a GET request to <https://hdnlmhee0.execute-api.us-east-1.amazonaws.com/prodDoctorBooking/slots>. The response status is 200 OK, with the following JSON:

```

1 | {
2 |     "statusCode": 200,
3 |     "headers": {
4 |         "Content-Type": "application/json"
5 |     },
6 |     "body": "[{"isBooked": false, "slot": "\u0010 - \u0011"}]"
7 |

```

Figure 77-Test Get Available Time Slots API Postman

The screenshot shows a GET request to <https://hdnlmhee0.execute-api.us-east-1.amazonaws.com/prodDoctorBooking/appointments>. The response status is 200 OK, with the following JSON:

```

1 | {
2 |     "statusCode": 200,
3 |     "headers": {
4 |         "Content-Type": "application/json"
5 |     },
6 |     "body": "[{"createdAt": "\u00272025-06-20T09:51:22.912752\u0027", "slot": "\u0019 - \u0010", "appointmentId": "\u0027a09450d1-9d5e-40aa-a358-5900dfc31f46\u0027", "patientName": "\u0027AlaaAbdelgader\u0027", "symptoms": "\u0027Headach,fatigue\u0027", "status": "\u0027Pending\u0027"}, {"createdAt": "\u00272025-06-20T09:51:37.392141\u0027", "slot": "\u0011 - \u0012", "appointmentId": "\u002790d0202c-3329-4577-9242-c00357d0c7ee\u0027", "patientName": "\u0027Sara\u0027", "symptoms": "\u0027Back pain and fatigue\u0027", "status": "\u0027Pending\u0027"}, {"createdAt": "\u00272025-06-20T09:51:53.752208\u0027", "slot": "\u0012 - \u0011", "appointmentId": "\u0027825db72c-3b29-4577-9242-c00357d0c7ee\u0027", "patientName": "\u0027Yousef\u0027", "symptoms": "\u0027Cold and sore throat\u0027", "status": "\u0027Pending\u0027"}, {"createdAt": "\u00272025-06-20T09:51:45.432095\u0027", "slot": "\u0018 - \u0019", "appointmentId": "\u00274d03f207-6550-4799-b52c-1c8f2f9ad3ec\u0027", "patientName": "\u0027Sara\u0027", "symptoms": "\u0027Headache\u0027", "status": "\u0027Pending\u0027"}]"
7 |

```

Figure 78-Get Appointments API test Postman

The screenshot shows the AWS SNS console for the 'doctor-booking' topic. At the top, there are buttons for 'Edit', 'Delete', and 'Publish message'. Below the title 'doctor-booking', there's a 'Details' section with fields for Name (doctor-booking), ARN (arn:aws:sns:us-east-1:730335665786:doctor-booking), and Type (Standard). To the right, there's a 'Display name' field set to 'doctor-booking' and a 'Topic owner' field with the ID 730335665786. Below this, tabs for 'Subscriptions', 'Access policy', 'Data protection policy', 'Delivery policy (HTTP/S)', 'Delivery status logging', 'Encryption', 'Tags', and 'Integrations' are visible. The 'Subscriptions' tab is selected, showing two entries:

ID	Endpoint	Status	Protocol
2fe0af01-8d2f-4364-8c3b-2d8ffc3e7ec2	alaajam02@gmail.com	Confirmed	EMAIL
6ce8f019-4749-40ab-be3a-99bb92e34f77	arn:aws:lambda:us-east-1:730335665786:function...	Confirmed	LAMBDA

Figure 79 - SNS doctor booking

New Appointment Booked

 **doctor-booking** <no-reply@sns.amazonaws.com>
 to me ▾
 A new appointment has been booked:
 Patient: Alaa
 Slot: 9 - 10
 Symptoms: Headachfatigue
 Time: 2025-06-20T09:26:06.786521
 --
 If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:730335665786:doctor-booking-2fe0af01-8d2f-4364-8c3b-2d8ffc3e7ec2>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact the sender.

Figure 80 -A new appointment Notification (SNS TEST)

Reset Confirmation

 **doctor-booking** <no-reply@sns.amazonaws.com>
 to me ▾
 Daily reset of appointments and slots completed successfully.
 --
 If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:730335665786:doctor-booking-6ce8f019-4749-40ab-be3a-99bb92e34f77>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact the sender.

Figure 81 -Reset Confirmation Notification (SNS Test)

Rule details

Rule name daily-resttttt	Status Enabled	Event bus name default	Type Scheduled Standard
Description	Rule ARN arn:aws:events:us-east-1:730335665786:rule/daily-resttttt	Event bus ARN arn:aws:events:us-east-1:730335665786:event-bus/default	

Event schedule

Cron expression
0 17 * * ? *

Next 10 trigger date(s)

UTC

Mon, 23 Jun 2025 17:00:00 UTC
Tue, 24 Jun 2025 17:00:00 UTC
Wed, 25 Jun 2025 17:00:00 UTC
Thu, 26 Jun 2025 17:00:00 UTC
Fri, 27 Jun 2025 17:00:00 UTC
Sat, 28 Jun 2025 17:00:00 UTC
Sun, 29 Jun 2025 17:00:00 UTC
Mon, 30 Jun 2025 17:00:00 UTC
Tue, 01 Jul 2025 17:00:00 UTC
Wed, 02 Jul 2025 17:00:00 UTC

Figure 82- Daily Automation with Event Bridge

Targets

Details	Target Name	Type	ARN	Input	Role
▼	resetDailyBookings	Lambda function	arn:aws:lambda:us-east-1:730335665786:function:resetDailyBookings	Matched event	Amazon_EventBridge_Invoke_Lambda_1164312354

Input to target: Matched event
Additional parameters: --
Dead-letter queue (DLQ): -

Figure 83 - EventBridge Target

Serverless-HealthBooking

App ID: d1256viqd8s1ho

▶ Get to production

Branches 1 Search...

main
Deployed

Domain
<https://main.d1256viqd8s1ho.amplifyapp.com>

Last deployment
13 hours ago

Last commit
Add SNS to the rest function /Serverless-HealthBooking:main

Figure 84-Deployed the Frontend Using AWS Amplify

Book an Appointment

Sara

Flu

8 - 9

Book

Figure 85 -Book an Appointment Page

Doctor Appointments

Appointments

Name	Symptoms	Time	Status	Update
Lina	Cough and fever	11 - 12	Completed	Completed
Sara	Flu	8 - 9	Pending	Pending

Figure 86-Doctor Appointments page

Log groups (5)

By default, we only load up to 10000 log groups.

Filter log groups or try pattern search Exact match

Log group	Log class	Anomaly d...	Data prote
createAppointmentLog	Standard	Configure	-
getAllAppointmentsLog	Standard	Configure	-
getAvailableSlotsLog	Standard	Configure	-
resetDailyBookingsLog	Standard	Configure	-
updateStatusAppointmentLog	Standard	Configure	-

Figure 87 -Enabled logging for all Lambda functions

createAppointmentLog

[Actions ▾](#) [View in Logs Insight](#)

▼ Log group details

Log class Info	Metric filters 0	Data protection -
Standard	Subscription filters 0	Sensitive data count -
ARN arn:aws:logs:us-east-1:730335665786:log-group:createAppointmentLog:*	Contributor Insights rules -	Field indexes Configure
Creation time 14 hours ago	KMS key ID -	Transformer Configure
Retention Never expire	Anomaly detection Configure	
Stored bytes -		

[Log streams](#) [Tags](#) [Anomaly detection](#) [Metric filters](#) [Subscription filters](#) [Contributor Insights](#) [Data protection](#) [Field indexes - new](#)

Log streams (8)

[Filter log streams or try prefix search](#) Exact match Show expired [Info](#)

<input type="checkbox"/> Log stream	Last event time
2025/06/20/createAppointment[\$LATEST]b8948a9ce674481daddef92839f90429	2025-06-20 13:02:53 (UTC+03:00)
2025/06/20/createAppointment[\$LATEST]9661b1cb44e04b68877bd7a720a9da20	2025-06-20 13:02:52 (UTC+03:00)
2025/06/20/createAppointment[\$LATEST]0c72bdfc881649449a33092d2ff2f697	2025-06-20 13:02:52 (UTC+03:00)

[Delete](#) [Create](#)

Figure 88 -create Appointment Logging Page

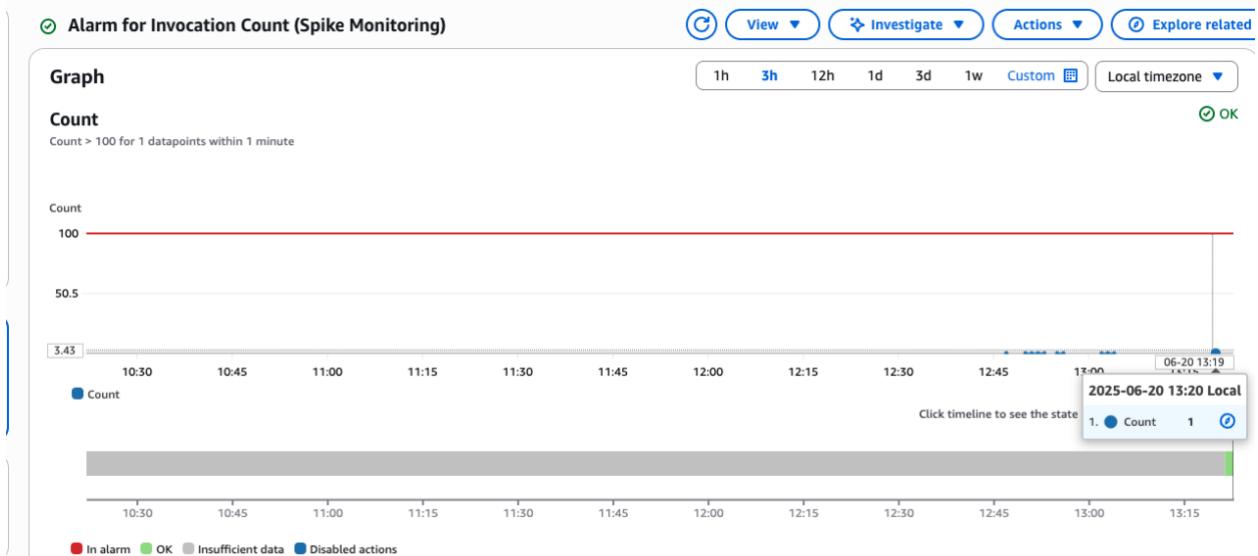


Figure 89- Alarm for Invocation Count (Spike Monitoring)

Alarm for Invocation Count (Spike Monitoring)

This graph monitors the number of API invocations to detect sudden traffic spikes. The red line represents the threshold, set at 100 requests per minute. The blue dots show the actual number of invocations. At 13:20, for example, the count was only 1 request, which is far below the threshold. This confirms that the traffic to the API is currently normal. If the number of invocations suddenly spikes above 100, the alarm will be triggered, alerting me to potential issues such as bot attacks or system errors.

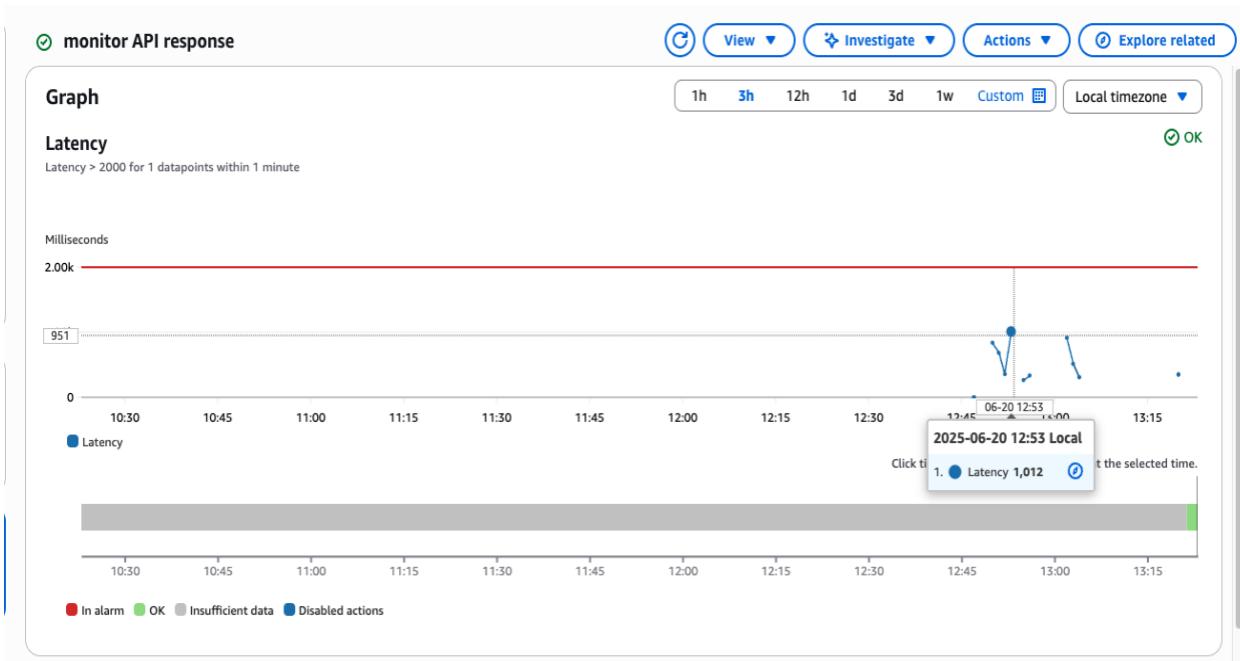


Figure 90 -Alert if Latency > 2s (Cloudwatch)

Alert if Latency > 2s

This graph shows the latency monitoring setup for my API. The red line indicates the threshold set at 2000 milliseconds (2 seconds). The blue dots represent the actual latency recorded during each minute. As seen in the data point at 12:53, the latency was 1012 milliseconds, which is well below the threshold. This indicates that the API is currently performing well and responding within an acceptable time. If latency ever exceeds 2 seconds, CloudWatch will trigger an alarm to help me investigate and take action quickly.

e. Conclusion

This scenario successfully demonstrates the ability to build and deploy a fully serverless medical appointment booking system using AWS-native services. By integrating Lambda, DynamoDB, API Gateway, SNS, EventBridge, CloudWatch, and AWS Amplify, I created a scalable and maintainable architecture with no traditional server management required. The system is secure, cost-effective, and offers a seamless experience for both patients and doctors.

Suggestions for improvement:

- Add user authentication using Amazon Cognito to restrict admin access.
- Implement input validation and security layers such as WAF for enhanced protection.
- Enable detailed metrics and alerts in CloudWatch for real-time monitoring of system health.
- Introduce integration with S3 or RDS to support uploading medical files or patient history in future expansions.

• References

1. Amazon Web Services. (2024). *Amazon S3 Documentation*. Retrieved from <https://docs.aws.amazon.com/s3/>
2. Amazon Web Services. (2024). *CloudFront Developer Guide*. Retrieved from <https://docs.aws.amazon.com/AmazonCloudFront/>
3. Amazon Web Services. (2024). *AWS Lambda Developer Guide*. Retrieved from <https://docs.aws.amazon.com/lambda/>
4. Amazon Web Services. (2024). *Auto Scaling User Guide*. Retrieved from <https://docs.aws.amazon.com/autoscaling/>
5. Amazon Web Services. (2024). *Amazon Route 53 Documentation*. Retrieved from <https://docs.aws.amazon.com/route53/>
6. Bash, L. (2022). *Serverless Architectures on AWS*. O'Reilly Media.