GROUP 13

Capstone Project 1 – Part 1: VPC Creation Report

Introduction

This report outlines the steps in building a secure Virtual Private cloud (VPC) and subnets configuration in the AWS infrastructure. the report provides a detailed guide to setup a customized Virtual private cloud (VPC) and a subnet architecture. A VPC is created with a defined CIDR block, public and private subnets are provisioned across multiple availability zones and route tables are configured to direct traffic appropriately between public and private resources. This configuration of our customized VPC supports the foundation for deploying scalable, secure and highly available applications in the cloud environment.

Step 1: VPC and Subnet Configuration

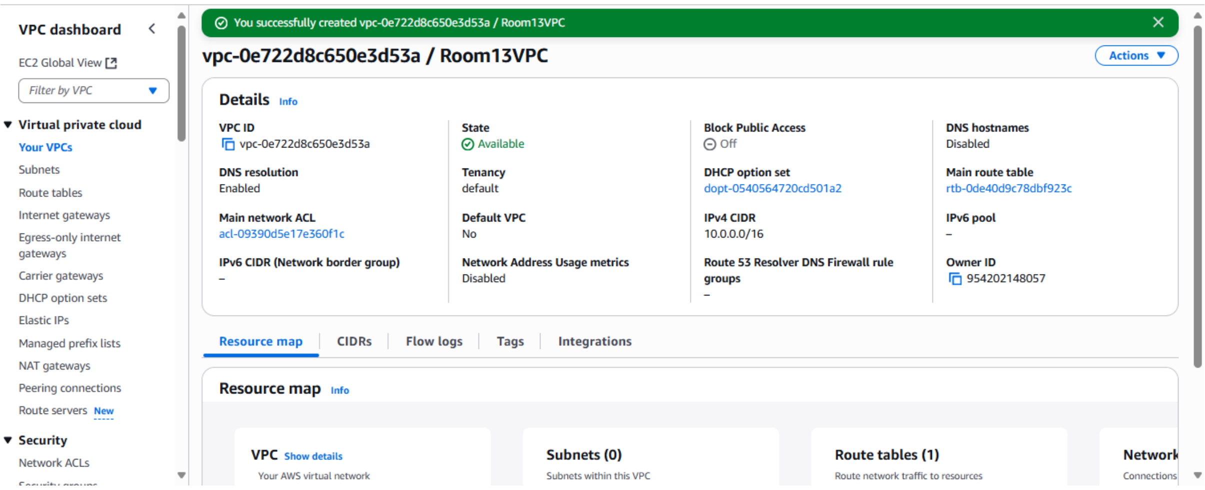
1. Creation of a New VPC

The process began by accessing the AWS Console and navigating to the **VPC service** section. The **Create VPC** option was selected to initiate the VPC creation.

The following details were entered:

- **Name**: Room13VPC
- **IPv4 CIDR Block**: 10.0.0.0/16
- **IPv6 CIDR Block**: Left as *No IPv6 CIDR Block*
- **Tenancy**: Default

Upon completion of the configuration, the **Create VPC** button was clicked to successfully create the new VPC. A screenshot was taken as evidence of successful creation.



You successfully created vpc-0e722d8c650e3d53a / Room13VPC

vpc-0e722d8c650e3d53a / Room13VPC

Details

VPC ID vpc-0e722d8c650e3d53a	State Available	Block Public Access Off	DNS hostnames Disabled
DNS resolution Enabled	Tenancy default	DHCP option set dopt-0540564720cd501a2	Main route table rtb-0de40d9c78dbf923c
Main network ACL acl-09390d5e17e360f1c	Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool –
IPv6 CIDR (Network border group) –	Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups –	Owner ID 954202148057

Resource map

VPC Show details
Your AWS virtual network

Subnets (0)
Subnets within this VPC

Route tables (1)
Route network traffic to resources

Network
Connections

2. Creation of Subnets

To ensure high availability, a total of four subnets were created: two public subnets and two private subnets, distributed across two Availability Zones.

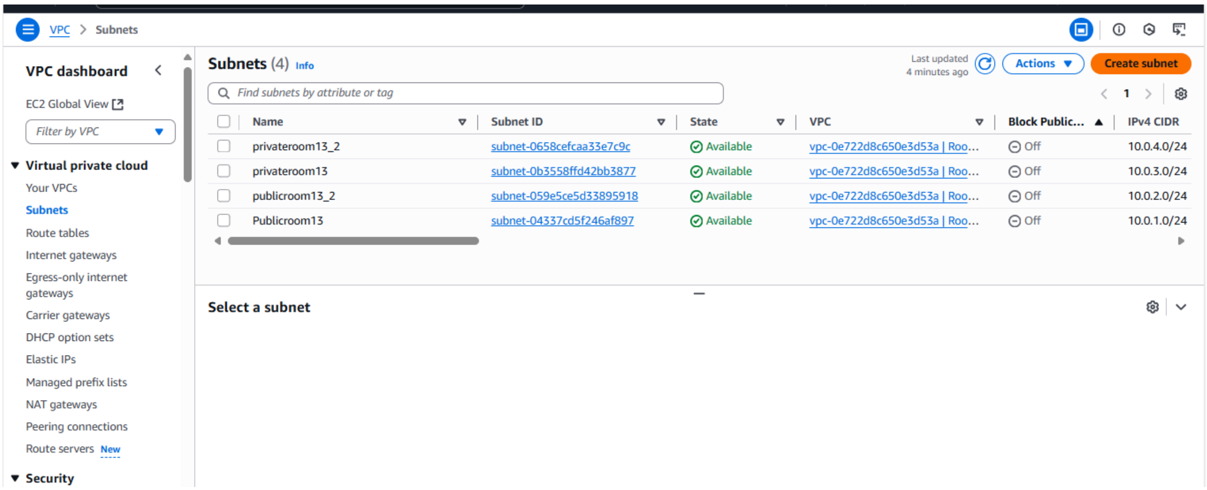
The subnet configuration was as follows:

Availability Zone	Public Subnet	Private Subnet	CIDR Block
us-east-1a	publicroom13	privateroom13	10.0.1.0/24 (public), 10.0.3.0/24 (private)
us-east-1b	publicroom13_2	privateroom13_2	10.0.2.0/24 (public), 10.0.4.0/24 (private)

The **Subnets** section was accessed, and the **Create Subnet** option was selected. For each subnet:

- The **VPC** selected was Room13VPC.
- The **Name** was assigned accordingly (e.g., publicroom13, publicroom13_2, privateroom13, privateroom13_2).
- Appropriate **Availability Zones** (us-east-1a, us-east-1b) were selected.
- The corresponding **CIDR blocks** were entered.

All four subsets were created at once, and a screenshot was taken after a successful creation.



□	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
□	privateroom13_2	subnet-0658cefcaa33e7c9c	✓ Available	vpc-0e722d8c650e3d53a | Roo...	⊖ Off	10.0.4.0/24
□	privateroom13	subnet-0b3558ffd42bb3877	✓ Available	vpc-0e722d8c650e3d53a | Roo...	⊖ Off	10.0.3.0/24
□	publicroom13_2	subnet-059e5ce5d33895918	✓ Available	vpc-0e722d8c650e3d53a | Roo...	⊖ Off	10.0.2.0/24
□	Publicroom13	subnet-04337cd5f246af897	✓ Available	vpc-0e722d8c650e3d53a | Roo...	⊖ Off	10.0.1.0/24

3. Route Table Configuration

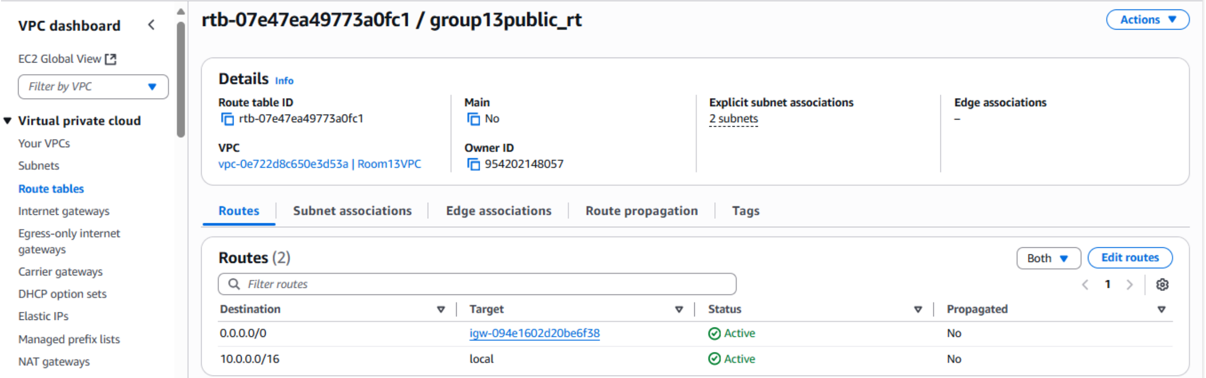
Public Route Table

The **Route Tables** section was accessed, and a new route table named **group13public_rt** was created and associated with Room13VPC.

A route was then added with the following configuration:

- **Destination**: `0.0.0.0/0`
- **Target**: Internet Gateway (to be created in the next step)

The **group13public_rt** route table was associated with both public subnets (publicroom13 and publicroom13_2). A screenshot was captured for documentation.



rtb-07e47ea49773a0fc1 / group13public_rt

Details

Route table ID: rtb-07e47ea49773a0fc1
Main: No
Explicit subnet associations: 2 subnets
Edge associations: –
VPC: vpc-0e722d8c650e3d53a | Room13VPC
Owner ID: 954202148057

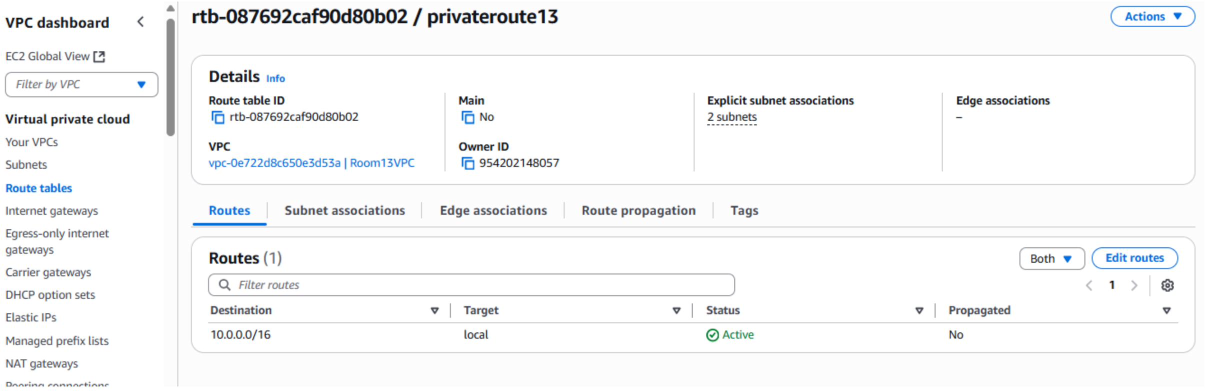
Routes (2)

Destination	Target	Status	Propagated
0.0.0.0/0	igw-094e1602d20be6f38	Active	No
10.0.0.0/16	local	Active	No

Private Route Table

Similarly, another route table named **privateroute13** was created for private subnets, associated with Room13VPC. Initially, no default route (`0.0.0.0/0`) was added, as this would be configured after the NAT Gateway creation.

The **privateroute13** route table was associated with both private subnets (privateroom13 and privateroom13_2). A screenshot was taken upon completion.



rtb-087692caf90d80b02 / privateroute13

Details

Route table ID: rtb-087692caf90d80b02
Main: No
Explicit subnet associations: 2 subnets
Edge associations: –
VPC: vpc-0e722d8c650e3d53a | Room13VPC
Owner ID: 954202148057

Routes (1)

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

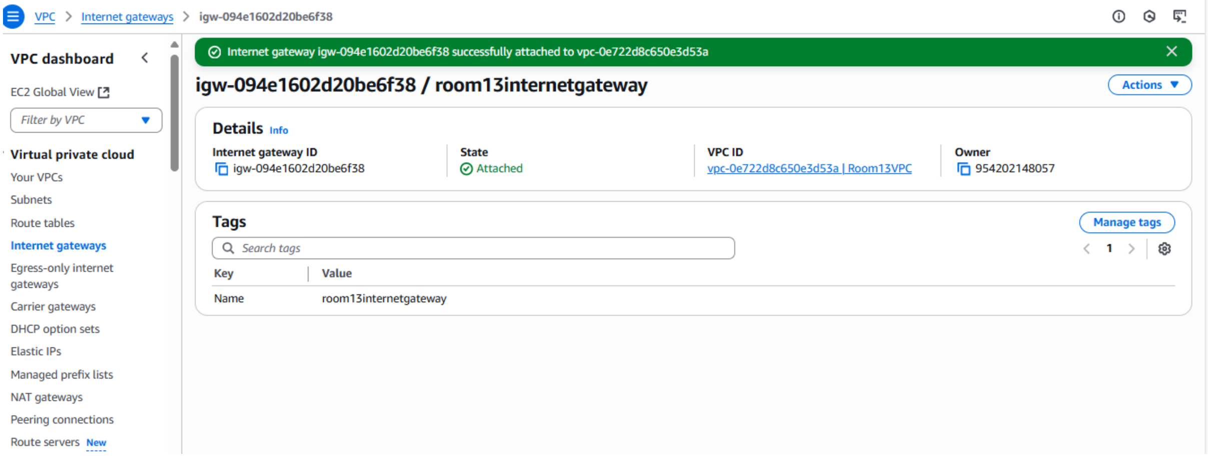
Step 2: Security Configurations

1. Creation and Attachment of Internet Gateway

An Internet Gateway was created by navigating to the **Internet Gateway** section and selecting **Create internet gateway**.

- **Name**: room13internetgateway

The Internet Gateway was successfully created and attached to the Room13VPC. A screenshot was taken to confirm attachment.



Internet gateway igw-094e1602d20be6f38 successfully attached to vpc-0e722d8c650e3d53a

igw-094e1602d20be6f38 / room13internetgateway

Details Info

Internet gateway ID	State	VPC ID	Owner
igw-094e1602d20be6f38	Attached	vpc-0e722d8c650e3d53a | Room13VPC	954202148057

Tags

Key	Value
Name	room13internetgateway

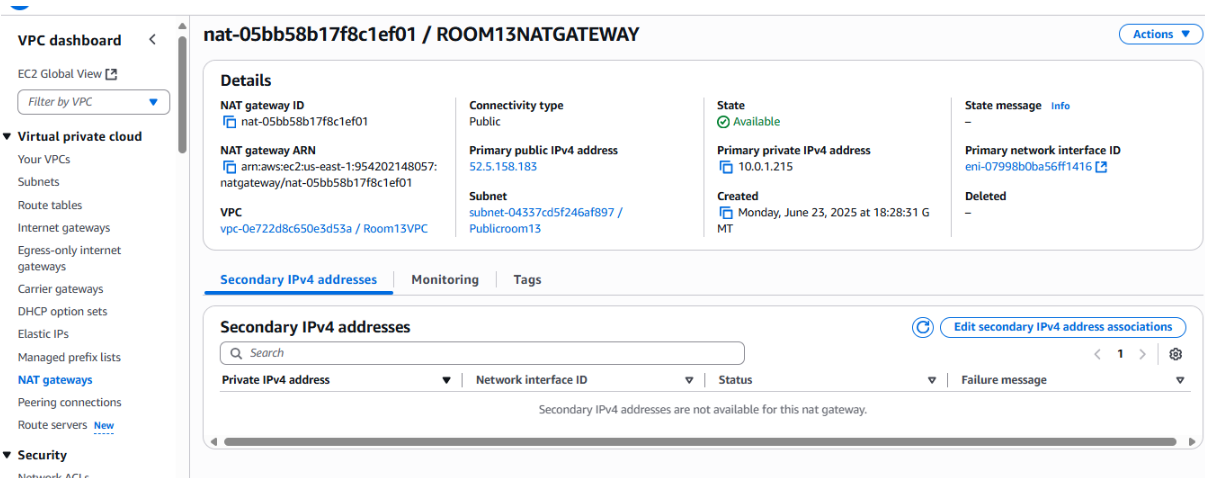
2. NAT Gateway Setup

An Elastic IP address was first allocated under the **Elastic IPs** section to be used for the NAT Gateway.

Subsequently, the **NAT Gateway** section was accessed, and a new NAT Gateway was created with the following configuration:

- **Name**: ROOM13NATGATEWAY
- **Subnet**: Selected one of the public subnets (publicroom13)
- **Elastic IP**: Selected the allocated Elastic IP

The NAT Gateway was successfully created, and a screenshot was taken as proof of completion.



nat-05bb58b17f8c1ef01 / ROOM13NATGATEWAY

Details

NAT gateway ID: nat-05bb58b17f8c1ef01

NAT gateway ARN: arn:aws:ec2:us-east-1:954202148057:natgateway/nat-05bb58b17f8c1ef01

VPC: vpc-0e722d8c650e3d53a / Room13VPC

Connectivity type: Public

Primary public IPv4 address: 52.5.158.183

Subnet: subnet-04337cd5f246af897 / Publicroom13

State: Available

Primary private IPv4 address: 10.0.1.215

Created: Monday, June 23, 2025 at 18:28:31 GMT

State message Info: –

Primary network interface ID: eni-07998b0ba56ff1416

Deleted: –

Secondary IPv4 addresses

Private IPv4 address	Network interface ID	Status	Failure message

Secondary IPv4 addresses are not available for this nat gateway.

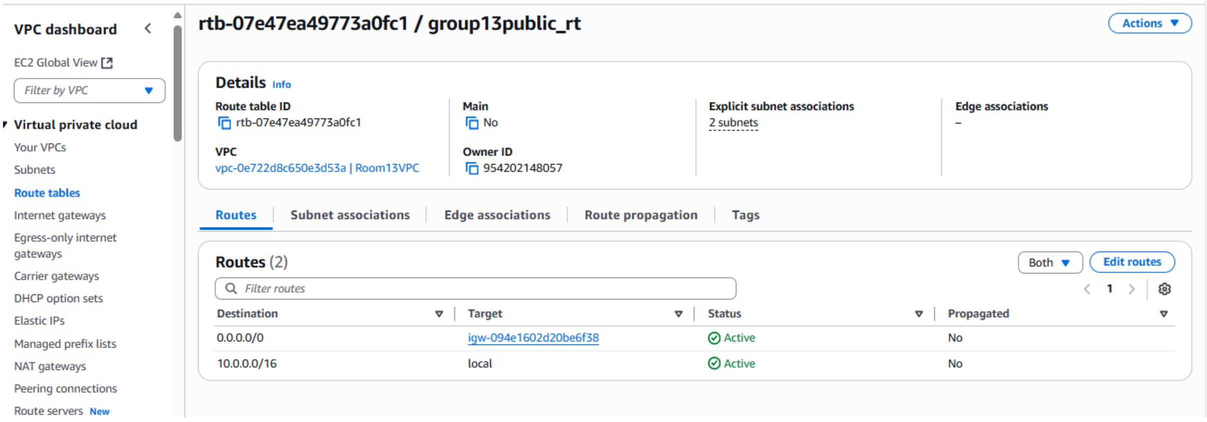
3. Update of Route Tables

Public Route Table

Verification was performed to ensure that the **group13public_rt** already contained the route:

- **Destination**: `0.0.0.0/0`
- **Target**: Internet Gateway (room13internetgateway)

A screenshot was taken after updating the public route table



rtb-07e47ea49773a0fc1 / group13public_rt

Details Info

Route table ID: rtb-07e47ea49773a0fc1
Main: No
Explicit subnet associations: 2 subnets
Edge associations: –
VPC: vpc-0e722d8c650e3d53a | Room13VPC
Owner ID: 954202148057

Routes (2)

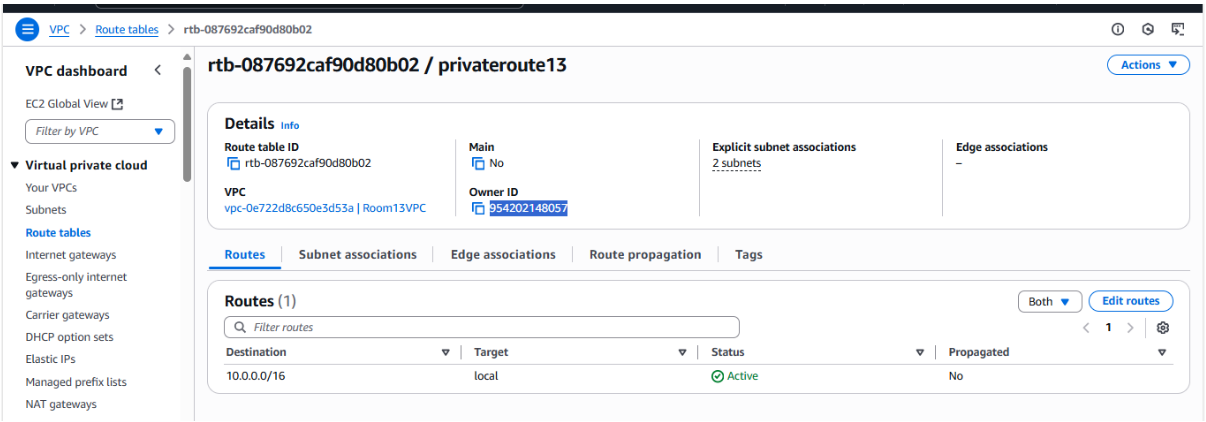
Destination	Target	Status	Propagated
0.0.0.0/0	igw-094e1602d20be6f38	Active	No
10.0.0.0/16	local	Active	No

Private Route Table

For the **privateroute13**, a new route was added:

- **Destination**: `0.0.0.0/0`
- **Target**: NAT Gateway (ROOM13NATGATEWAY)

A screenshot was taken after updating the private route table



rtb-087692caf90d80b02 / privateroute13

Details Info

Route table ID: rtb-087692caf90d80b02
Main: No
Explicit subnet associations: 2 subnets
Edge associations: –
VPC: vpc-0e722d8c650e3d53a | Room13VPC
Owner ID: 954202148057

Routes (1)

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No

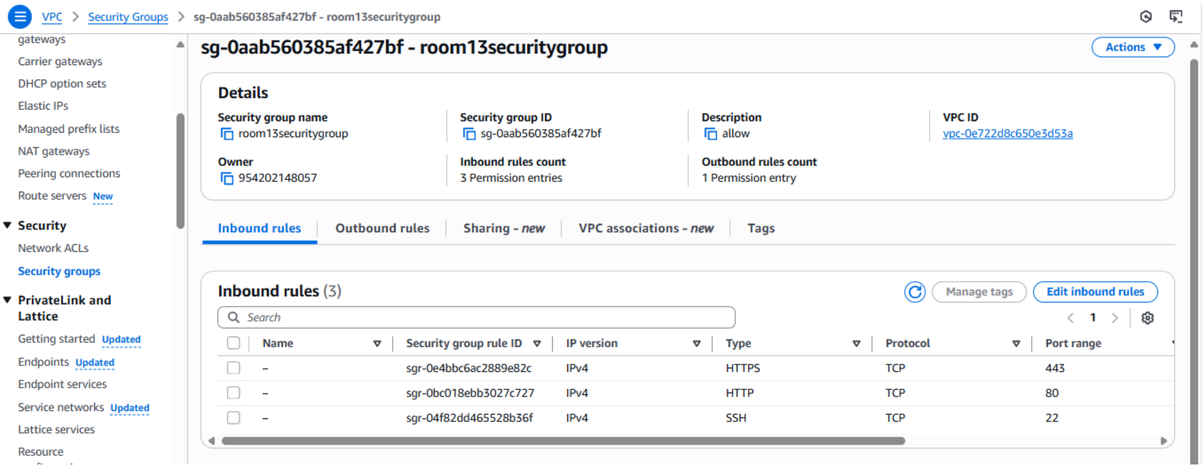
4. Security Group Configuration

The **Security Groups** section was accessed, and a security group was created:

For Public Instances (e.g., Load Balancer)

- **Name**: room13securitygroup
- **Inbound Rules**:
  - HTTP (port 80) – Allow from Anywhere (`0.0.0.0/0`)
  - HTTPS (port 443) – Allow from Anywhere (`0.0.0.0/0`)
  - SSH (port 22) – Allow from specific IP (administrator's IP)
- **Outbound Rules**: Allow all traffic (default configuration)

A screenshot was captured for both security group.



sg-0aab560385af427bf - room13securitygroup

Details

Security group name: room13securitygroup

Security group ID: sg-0aab560385af427bf

Description: allow

VPC ID: vpc-0e722d8c650e3d53a

Owner: 954202148057

Inbound rules count: 3 Permission entries

Outbound rules count: 1 Permission entry

Inbound rules (3)

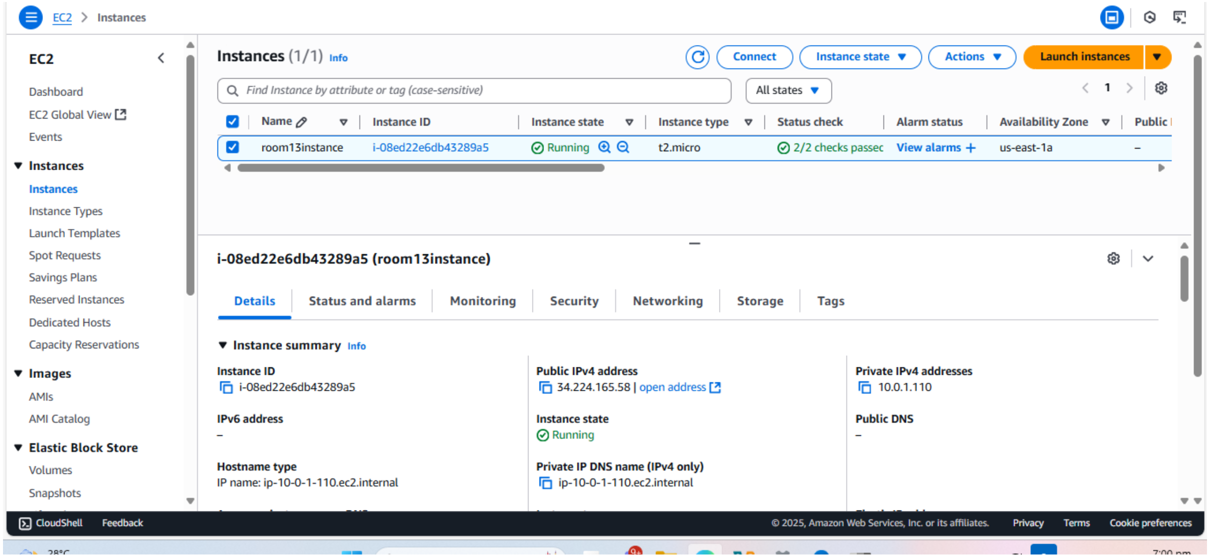
Name	Security group rule ID	IP version	Type	Protocol	Port range
–	sgr-0e4bbc6ac2889e82c	IPv4	HTTPS	TCP	443
–	sgr-0bc018ebb3027c727	IPv4	HTTP	TCP	80
–	sgr-04f82dd465528b36f	IPv4	SSH	TCP	22

5. VPC Functionality Test

A test EC2 instance was launched within one of the public subnets:

- The room13securitygroup security group was assigned.
- An Elastic IP was associated with the instance.

A screenshot was created after creating the EC2 instance.



☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public I
☑	room13instance	i-08ed22e6db43289a5	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	–

i-08ed22e6db43289a5 (room13instance)

Instance ID: i-08ed22e6db43289a5

Public IPv4 address: 34.224.165.58 | open address

Private IPv4 addresses: 10.0.1.110

IPv6 address: –

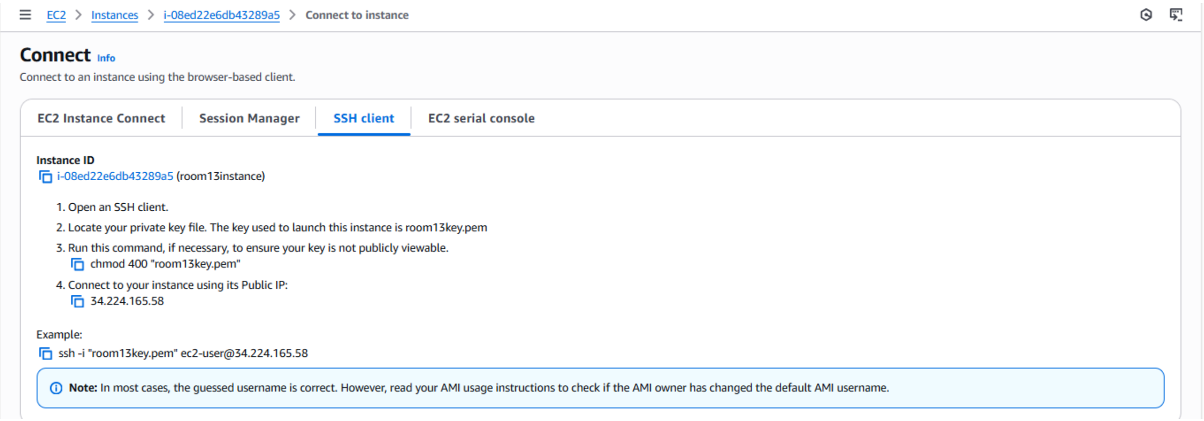
Instance state: Running

Public DNS: –

Hostname type: IP name: ip-10-0-1-110.ec2.internal

Private IP DNS name (IPv4 only): ip-10-0-1-110.ec2.internal

An SSH connection was initiated to verify internet connectivity. Upon successful connection, a screenshot was taken to confirm proper functionality of the VPC setup.



Connect to an instance using the browser-based client.

Instance ID

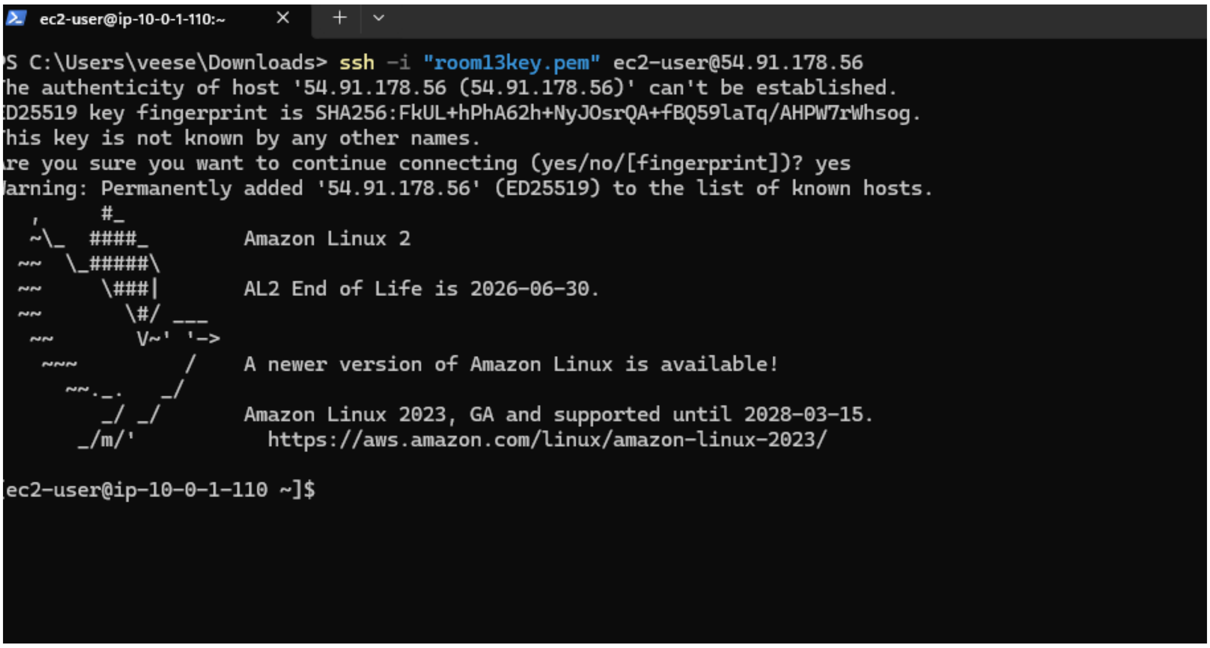
i-08ed22e6db43289a5 (room13instance)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is room13key.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   ```
   chmod 400 "room13key.pem"
   ```
4. Connect to your instance using its Public IP:
   34.224.165.58

Example:

```
ssh -i "room13key.pem" ec2-user@34.224.165.58
```

Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.



```
S C:\Users\veese\Downloads> ssh -i "room13key.pem" ec2-user@54.91.178.56
he authenticity of host '54.91.178.56 (54.91.178.56)' can't be established.
D25519 key fingerprint is SHA256:FkUL+hPhA62h+NyJOsrQA+fBQ59laTq/AHPW7rWhsog.
his key is not known by any other names.
re you sure you want to continue connecting (yes/no/[fingerprint])? yes
arning: Permanently added '54.91.178.56' (ED25519) to the list of known hosts.
  ,     #_
  ~\_  ####_        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-10-0-1-110 ~]$
```

Challenges Encountered

During the course of this project, a key challenge was identified:

- Initially, after creating the route tables (**group13public_rt** and **privateroute13**), the team neglected to associate these route tables with the corresponding subnets. As a result, network traffic routing failed, and the test EC2 instance was unable to access the internet.
- Upon diagnosing the issue, it was determined that proper association of the route tables to their respective public and private subnets was missing. After correcting these associations, full connectivity was achieved.
- This experience emphasized the importance of thorough verification of route table associations during VPC configuration.

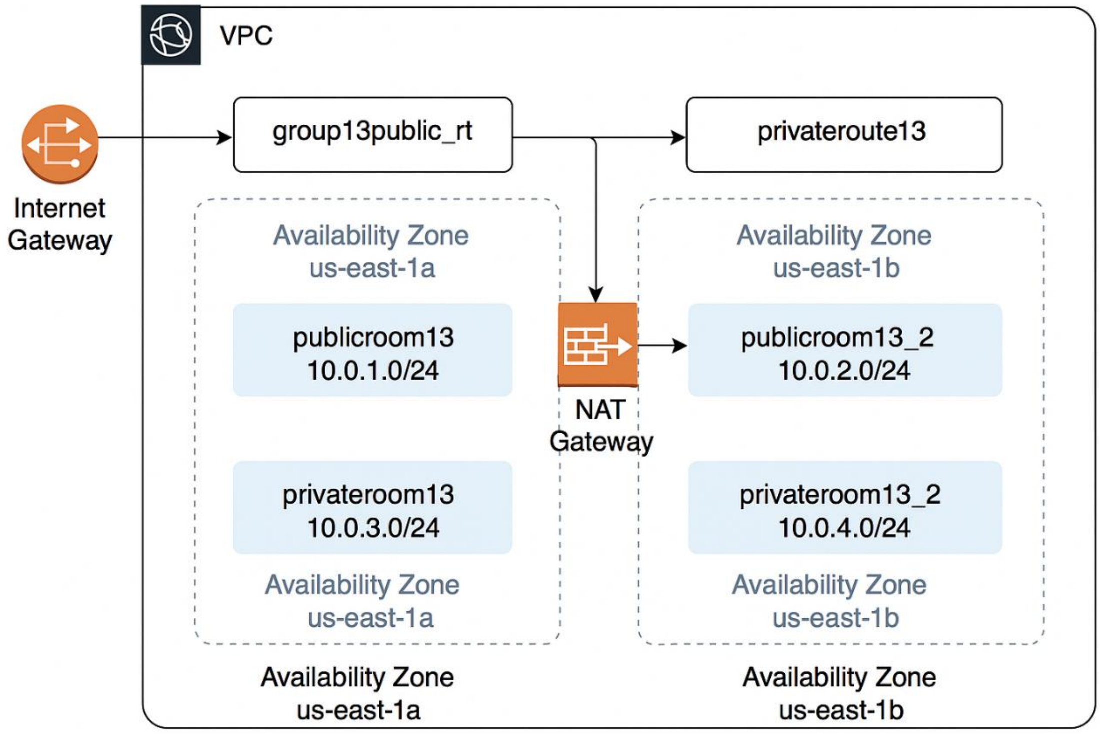
Summary and Conclusion

In this project, a secure and highly available Virtual Private Cloud (VPC) was successfully designed and deployed using AWS services. The group created one VPC with the CIDR block `10.0.0.0/16`, two public subnets and two private subnets across multiple availability zones, Internet Gateway for public internet access, NAT Gateway for controlled outbound access from private subnets and a security group for public instances.

The project not only reinforced our understanding of the core AWS networking services but also provided hands-on experience in troubleshooting common configuration oversights. The challenge encountered with route table associations reinforced the importance of attention to detail when designing cloud infrastructure.

Overall, the objectives of the Capstone Project Part 1 were fully met, providing a solid foundation for subsequent phases of cloud infrastructure development.

Architectural Diagram


VPC
Internet Gateway
group13public_rt
privateroute13
Availability Zone us-east-1a
publicroom13 10.0.1.0/24
privateroom13 10.0.3.0/24
Availability Zone us-east-1a
NAT Gateway
Availability Zone us-east-1b
publicroom13_2 10.0.2.0/24
privateroom13_2 10.0.4.0/24
Availability Zone us-east-1b
Availability Zone us-east-1a
Availability Zone us-east-1b

The diagram above provides a clear and concise overview of the VPC architecture, including key components and the direction of traffic flow.

Capstone Project 1 – Part 2:

Implementation Report: Scalable AWS Architecture with Load Balancing and Auto Scaling

Introduction

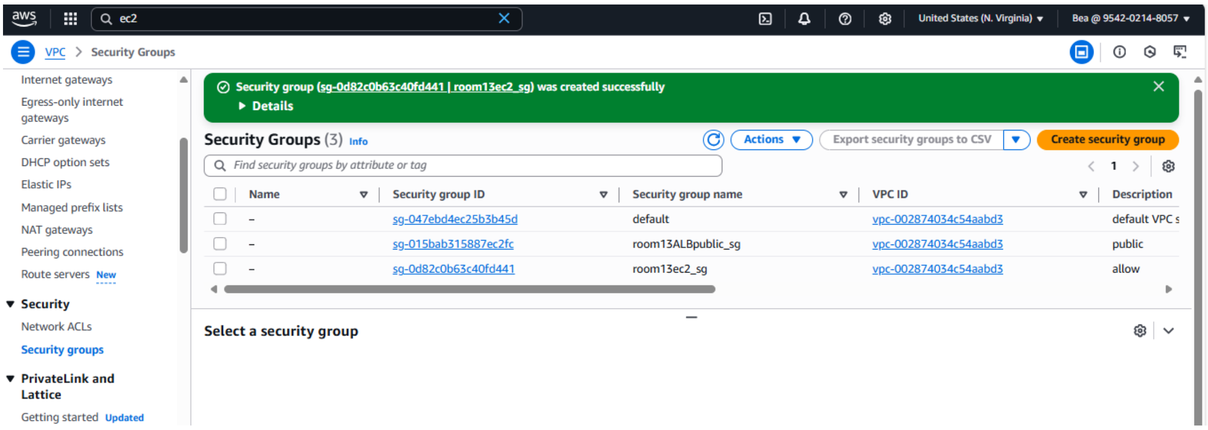
In this project, the team implemented a highly available, secure, and scalable web application environment on AWS. The process began with the configuration of security groups to enforce controlled access—allowing public traffic to reach the load balancer while restricting backend EC2 instances to accept traffic only from the load balancer.

Subsequently, EC2 instances were launched and configured in public subnets. A web application was installed, and an (AMI) Amazon Machine was created to support auto scaling. An **Application Load Balancer (ALB)** was then set up to intelligently route incoming traffic and perform health checks, ensuring that only healthy instances receive requests.

Finally, an **Auto Scaling Group (ASG)** was configured to span multiple Availability Zones. Dynamic scaling policies were defined based on CPU utilization to maintain responsiveness during peak traffic. Collectively, these steps resulted in a robust cloud infrastructure capable of ensuring high reliability.

Step 1: Security Group Configuration

Two distinct security groups were created to control network access:



Security group (sg-0d82c0b63c40fd441 | room13ec2_sg) was created successfully

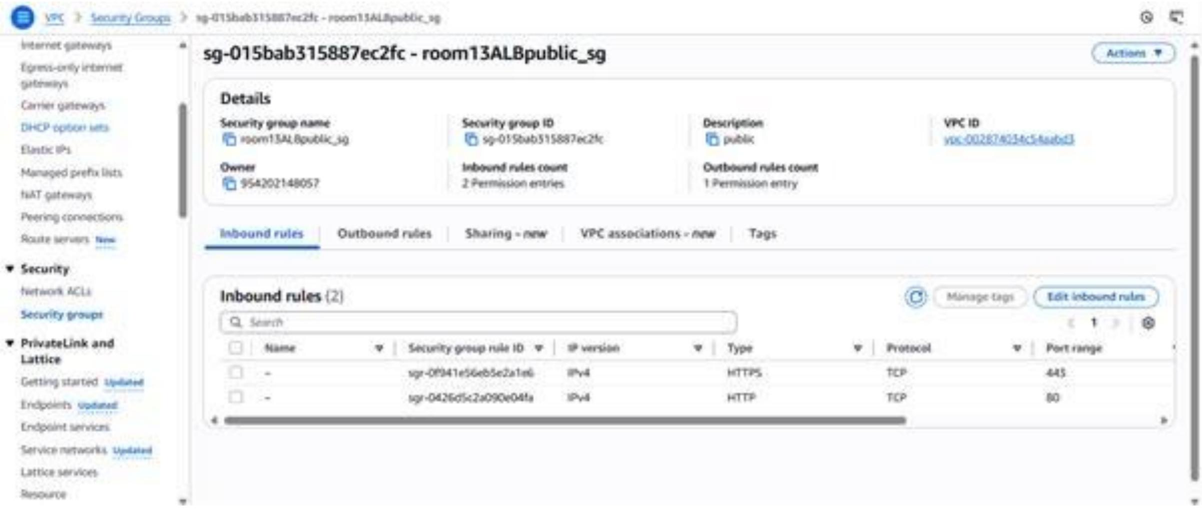
▸ Details

Name	Security group ID	Security group name	VPC ID	Description
–	sg-047ebd4ec25b3b45d	default	vpc-002874034c54aabd3	default VPC s
–	sg-015bab315887ec2fc	room13ALBpublic_sg	vpc-002874034c54aabd3	public
–	sg-0d82c0b63c40fd441	room13ec2_sg	vpc-002874034c54aabd3	allow

1.1 Load Balancer Security Group (ALB-SG)

A security group was configured specifically for the Application Load Balancer. It was set to allow inbound traffic on the following ports:

- **HTTP (port 80)**
- **HTTPS (port 443)**
- **Source:** Anywhere (`0.0.0.0/0`)



sg-015bab315887ec2fc - room13ALBpublic_sg

Details

Security group name	Security group ID	Description	VPC ID
room13ALBpublic_sg	sg-015bab315887ec2fc	public	vpc-002874034c54aabd3
Owner	**Inbound rules count**	**Outbound rules count**	
954202148057	2 Permission entries	1 Permission entry	

Inbound rules (2)

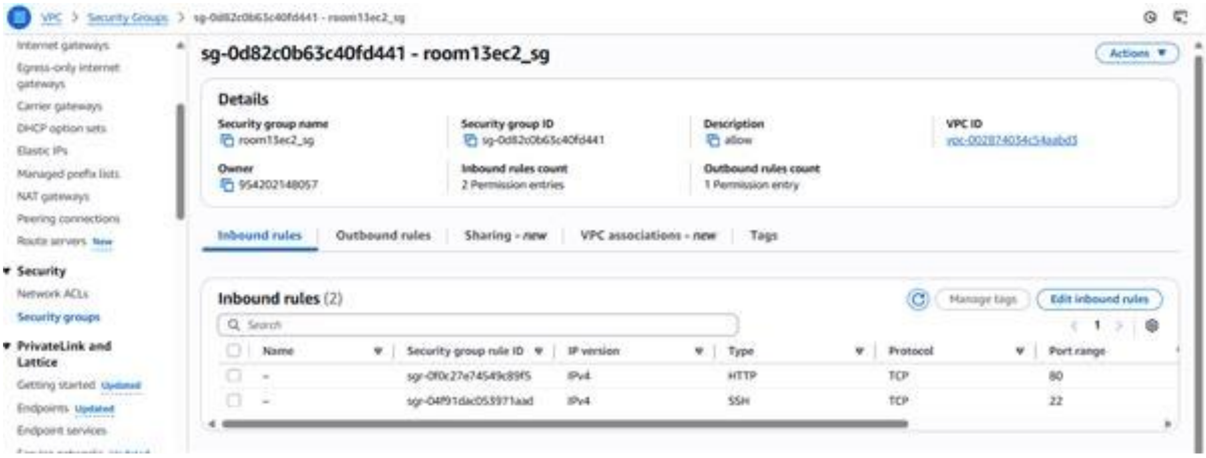
Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0f941e56eb5e2a1e6	IPv4	HTTPS	TCP	443
-	sgr-0426d5c2a090e04fa	IPv4	HTTP	TCP	80

1.2 EC2 Instance Security Group (EC2-SG)

A separate security group was assigned to EC2 instances. This group allowed inbound traffic only on:

- **HTTP (port 80)**
- **Source:** The ALB security group

This ensured that only traffic routed through the load balancer could reach the EC2 instances.



sg-0d82c0b63c40fd441 - room13ec2_sg

Details

Security group name	Security group ID	Description	VPC ID
room13ec2_sg	sg-0d82c0b63c40fd441	allow	vpc-002874034c54aabd3
Owner	**Inbound rules count**	**Outbound rules count**	
954202148057	2 Permission entries	1 Permission entry	

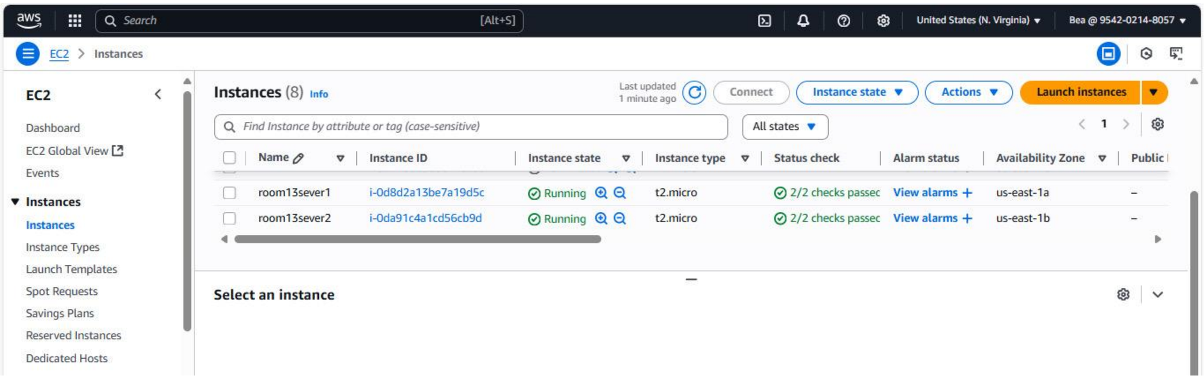
Inbound rules (2)

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0f0c27e74549c89f5	IPv4	HTTP	TCP	80
-	sgr-04f91dac053971aad	IPv4	SSH	TCP	22

Step 2: EC2 Instance Deployment and AMI Creation

2.1 Instance Launch and Configuration

Two EC2 instances (`t2.micro`) were launched in a public subnet. (room13sever1 and room13sever2)



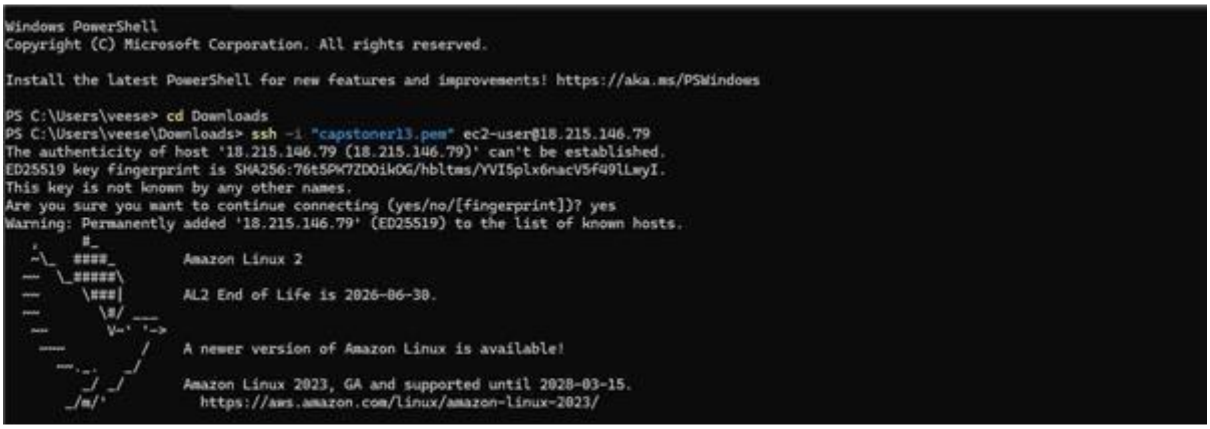
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public
□	room13sever1	i-0d8d2a13be7a19d5c	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	–
□	room13sever2	i-0da91c4a1cd56cb9d	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1b	–

A basic web server was installed using the following commands:

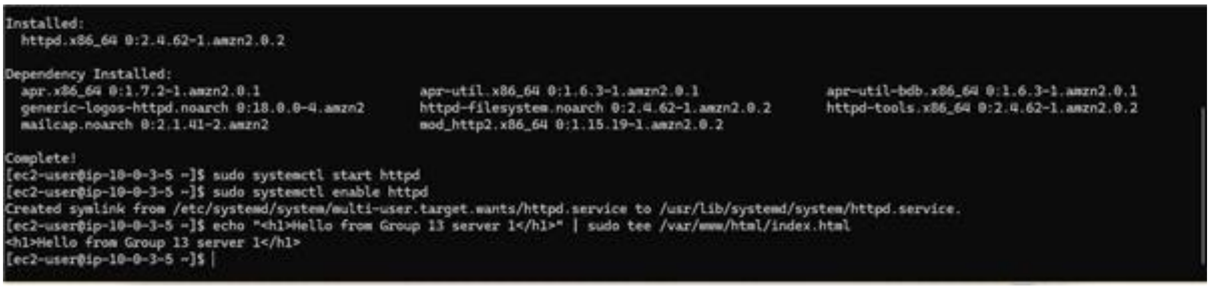
```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
echo "Hello from $(hostname)" > /var/www/html/index.html
```

Functionality was validated by accessing the instance’s public IP to confirm web server availability.

Server 1:

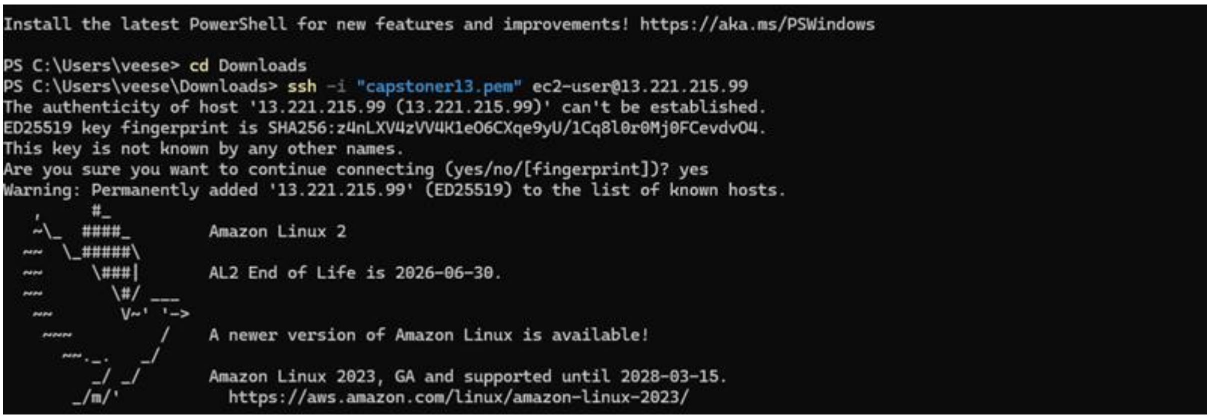


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\veese> cd Downloads
PS C:\Users\veese\Downloads> ssh -i "capstoner13.pem" ec2-user@18.215.146.79
The authenticity of host '18.215.146.79 (18.215.146.79)' can't be established.
ED25519 key fingerprint is SHA256:76t5PK7ZDOikOG/hbltms/YVI5plx6nacV5f49lLmyI.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '18.215.146.79' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        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/
```

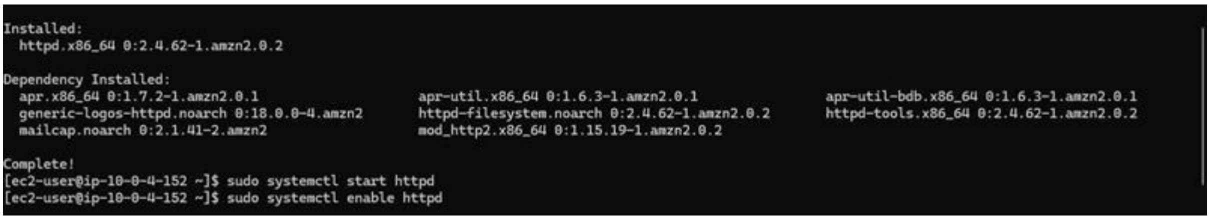


```
Installed:
  httpd.x86_64 0:2.4.62-1.amzn2.0.2

Dependency Installed:
  apr.x86_64 0:1.7.2-1.amzn2.0.1                  apr-util.x86_64 0:1.6.3-1.amzn2.0.1                 apr-util-bdb.x86_64 0:1.6.3-1.amzn2.0.1
  generic-logos-httpd.noarch 0:18.0.0-4.amzn2     httpd-filesystem.noarch 0:2.4.62-1.amzn2.0.2        httpd-tools.x86_64 0:2.4.62-1.amzn2.0.2
  mailcap.noarch 0:2.1.41-2.amzn2                 mod_http2.x86_64 0:1.15.19-1.amzn2.0.2

Complete!
[ec2-user@ip-10-0-3-5 ~]$ sudo systemctl start httpd
[ec2-user@ip-10-0-3-5 ~]$ sudo systemctl enable httpd
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-10-0-3-5 ~]$ echo "<h1>Hello from Group 13 server 1</h1>" | sudo tee /var/www/html/index.html
<h1>Hello from Group 13 server 1</h1>
[ec2-user@ip-10-0-3-5 ~]$ |
```

Server 2:



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\veese> cd Downloads
PS C:\Users\veese\Downloads> ssh -i "capstoner13.pem" ec2-user@13.221.215.99
The authenticity of host '13.221.215.99 (13.221.215.99)' can't be established.
ED25519 key fingerprint is SHA256:z4nLXV4zVV4K1eO6CXqe9yU/1Cq8l0r0Mj0FCevdvO4.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.221.215.99' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        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/
```



```
Installed:
  httpd.x86_64 0:2.4.62-1.amzn2.0.2

Dependency Installed:
  apr.x86_64 0:1.7.2-1.amzn2.0.1                  apr-util.x86_64 0:1.6.3-1.amzn2.0.1            apr-util-bdb.x86_64 0:1.6.3-1.amzn2.0.1
  generic-logos-httpd.noarch 0:18.0.0-4.amzn2     httpd-filesystem.noarch 0:2.4.62-1.amzn2.0.2   httpd-tools.x86_64 0:2.4.62-1.amzn2.0.2
  mailcap.noarch 0:2.1.41-2.amzn2                 mod_http2.x86_64 0:1.15.19-1.amzn2.0.2

Complete!
[ec2-user@ip-10-0-4-152 ~]$ sudo systemctl start httpd
[ec2-user@ip-10-0-4-152 ~]$ sudo systemctl enable httpd
```

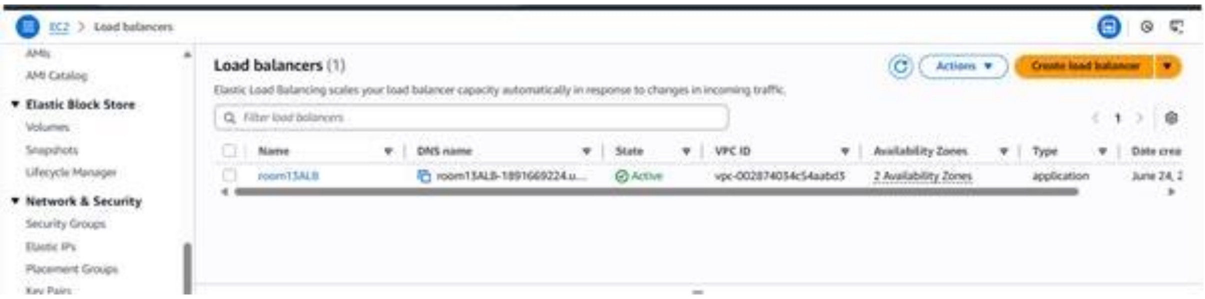
2.2 AMI Creation

After configuration, an Amazon Machine Image (AMI) was created from the running EC2 instance. This AMI served as the template for instances launched within the Auto Scaling Group.

Step 3: Application Load Balancer (ALB) Configuration

An **Application Load Balancer** was created from the EC2 Dashboard with the following configuration:

- **Scheme:** Internet-facing
- **Listener:** HTTP on port 80
- **Availability Zones:** Two public subnets were selected across different AZs



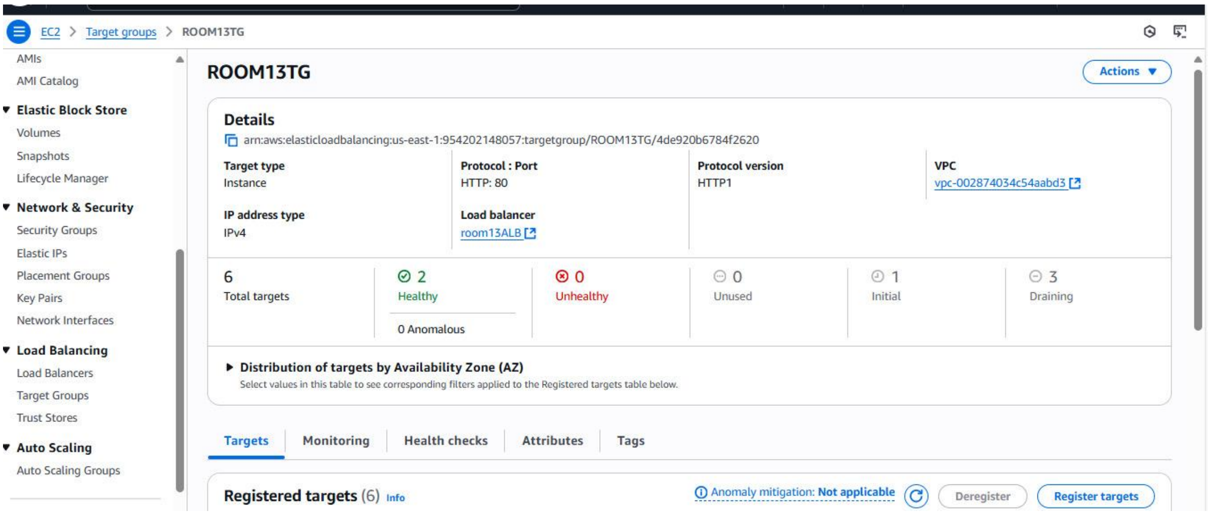
Load balancers (1)

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

Name	DNS name	State	VPC ID	Availability Zones	Type	Date crea
room13ALB	room13ALB-1891669224.u...	Active	vpc-002874034c54aabd3	2 Availability Zones	application	June 24, 2

A target group was created and registered with the EC2 instance. Health checks were defined with the following parameters:

- **Path:** /
- **Protocol:** HTTP



ROOM13TG

Details

arn:aws:elasticloadbalancing:us-east-1:954202148057:targetgroup/ROOM13TG/4de920b6784f2620

Target type: Instance
Protocol : Port: HTTP: 80
Protocol version: HTTP1
VPC: vpc-002874034c54aabd3
IP address type: IPv4
Load balancer: room13ALB

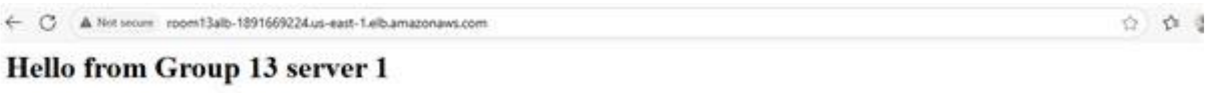
6 Total targets | 2 Healthy (0 Anomalous) | 0 Unhealthy | 0 Unused | 1 Initial | 3 Draining

Distribution of targets by Availability Zone (AZ)

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

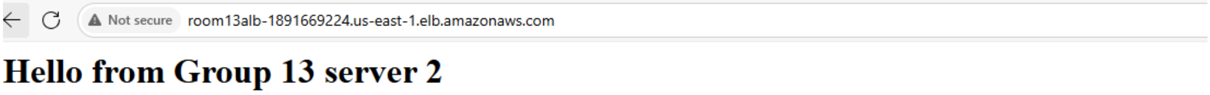
Registered targets (6) Info

Anomaly mitigation: Not applicable


Not secure room13alb-1891669224.us-east-1.elb.amazonaws.com

Hello from Group 13 server 1

The load balancer was tested by lunching the DNS name in a new tab and this displayed “Hello from Group 13 server 1 and as the page was refreshed it displayed sever 2 as indicated below.



Not secure room13alb-1891669224.us-east-1.elb.amazonaws.com

Hello from Group 13 server 2

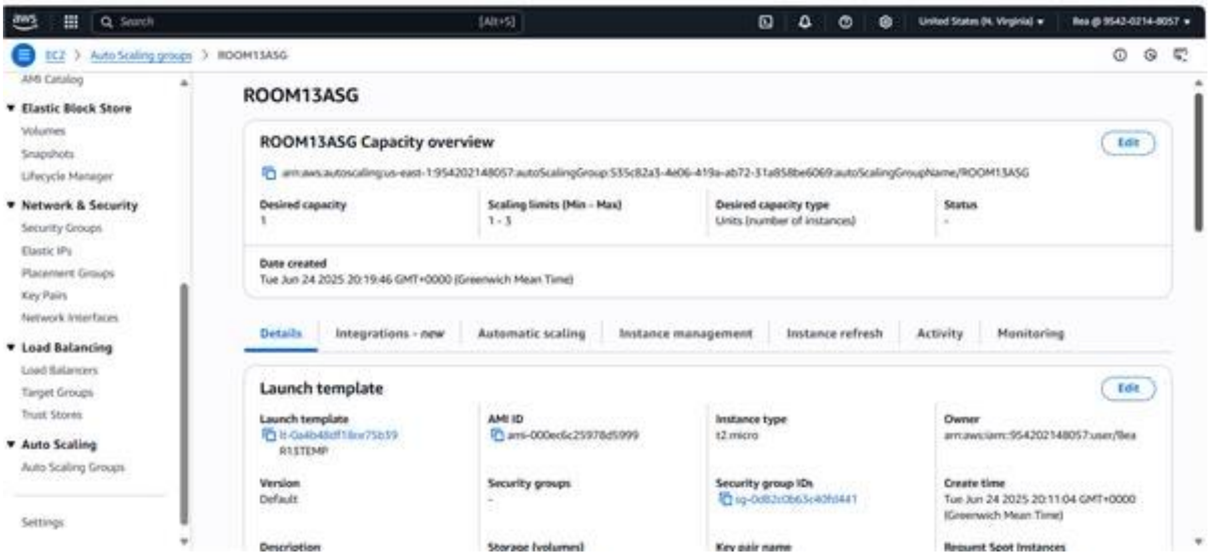
The ALB was associated with the previously created `ALB-SG`.

Step 4: Auto Scaling Group (ASG) Setup

An **Auto Scaling Group** was configured using the AMI created in Step 2. Key configurations included:

- **Availability Zones:** Spanned multiple zones for redundancy
- **Target Group:** Attached to the ALB's target group
- **Instance Counts:**

- Minimum: 1
- Desired: 2
- Maximum: 4



ROOM13ASG

ROOM13ASG Capacity overview

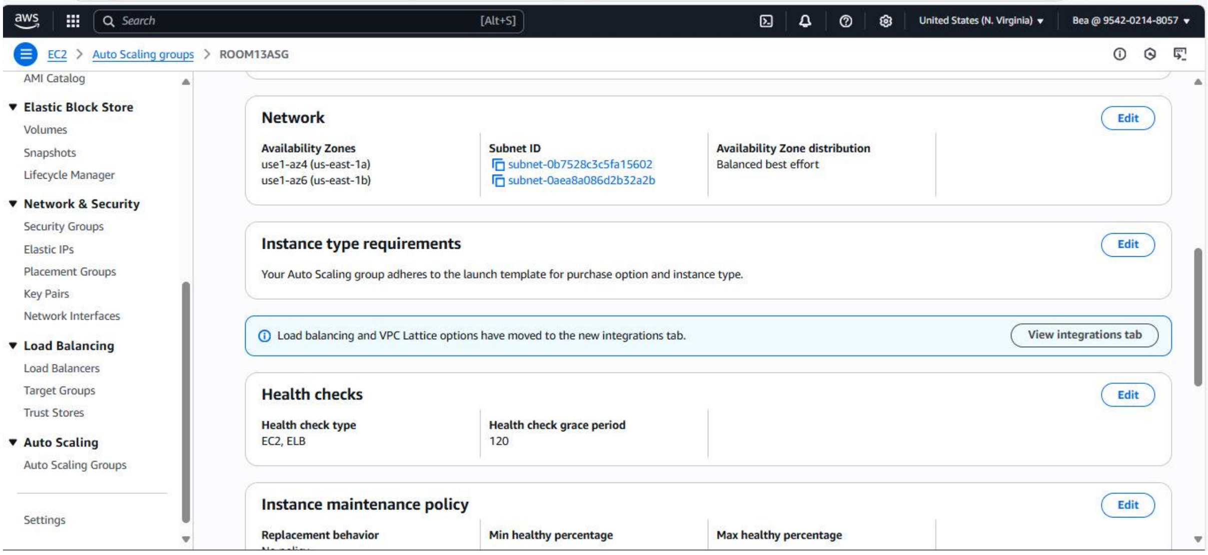
arn:aws:autoscaling:us-east-1:954202148057:autoScalingGroup:535c82a3-4e06-419a-ab72-31a858be6069:autoScalingGroupName/ROOM13ASG

Desired capacity	Scaling limits (Min - Max)	Desired capacity type	Status
1	1 - 3	Units (number of instances)	-

Date created
Tue Jun 24 2025 20:19:46 GMT+0000 (Greenwich Mean Time)

Launch template

Launch template	AMI ID	Instance type	Owner
lt-0a4b48df18ce75b59 R13TEMP	ami-000ec6c25978d5999	t2.micro	arn:aws:iam::954202148057:user/Bea
Version	**Security groups**	**Security group IDs**	**Create time**
Default	-	sg-0d82c0b63c40fd441	Tue Jun 24 2025 20:11:04 GMT+0000 (Greenwich Mean Time)
Description	**Storage (volumes)**	**Key pair name**	**Request Spot Instances**



Network

Availability Zones	Subnet ID	Availability Zone distribution
use1-az4 (us-east-1a) use1-az6 (us-east-1b)	subnet-0b7528c3c5fa15602 subnet-0aea8a086d2b32a2b	Balanced best effort

Instance type requirements

Your Auto Scaling group adheres to the launch template for purchase option and instance type.

Load balancing and VPC Lattice options have moved to the new integrations tab.

Health checks

Health check type	Health check grace period
EC2, ELB	120

Instance maintenance policy

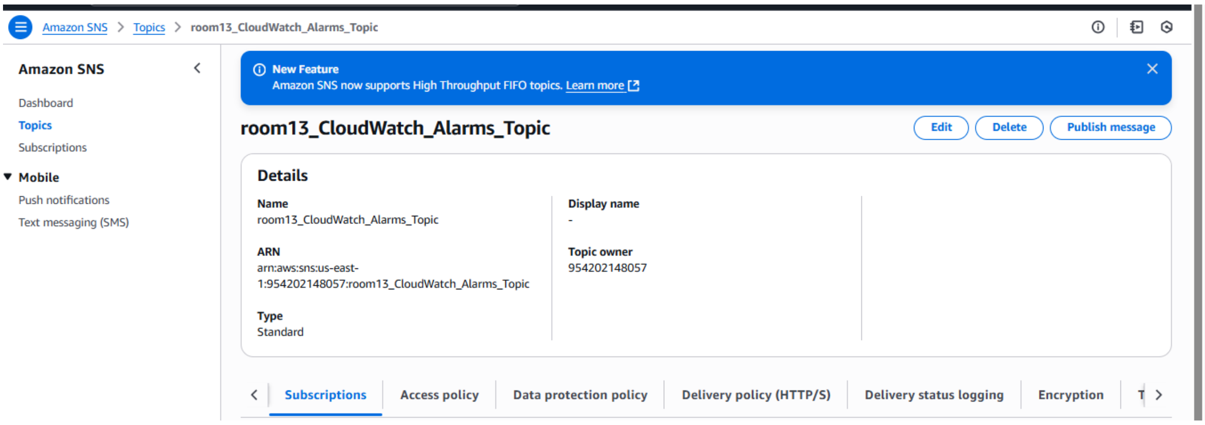
Replacement behavior	Min healthy percentage	Max healthy percentage

Scaling Policies

Scaling policies were defined using **CloudWatch alarms**. The configuration included:

- **Scale Out:** When CPU utilization exceeded 60%
- **Scale In:** When CPU utilization dropped below 30%

This allowed the infrastructure to dynamically adapt to changes in demand.



New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more

room13_CloudWatch_Alarms_Topic

Details

Name
room13_CloudWatch_Alarms_Topic

ARN
arn:aws:sns:us-east-1:954202148057:room13_CloudWatch_Alarms_Topic

Type
Standard

Display name
-

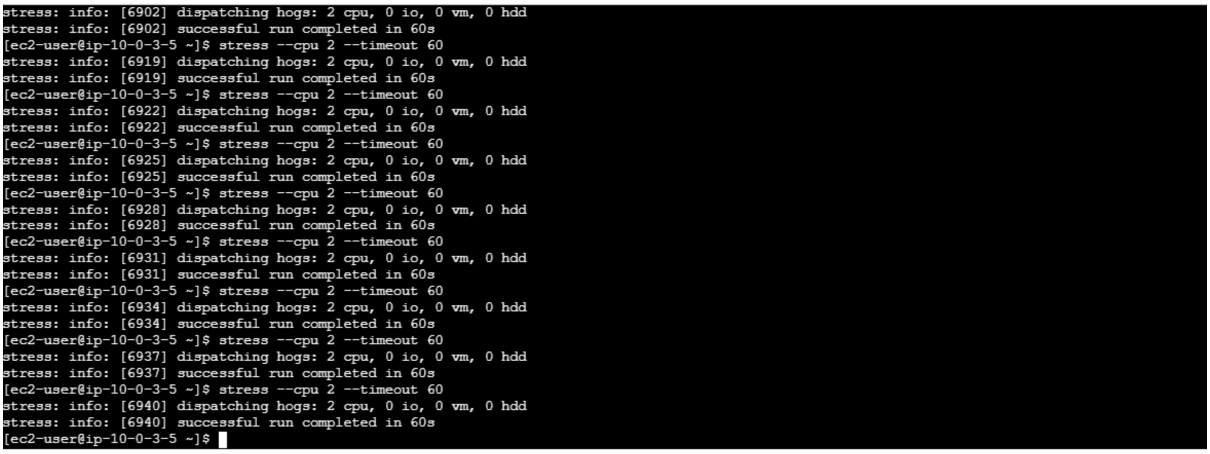
Topic owner
954202148057

Step 5: Testing and Optimization

The system was subjected to multiple tests to validate performance, availability, and fault tolerance:

5.1 Connectivity Testing

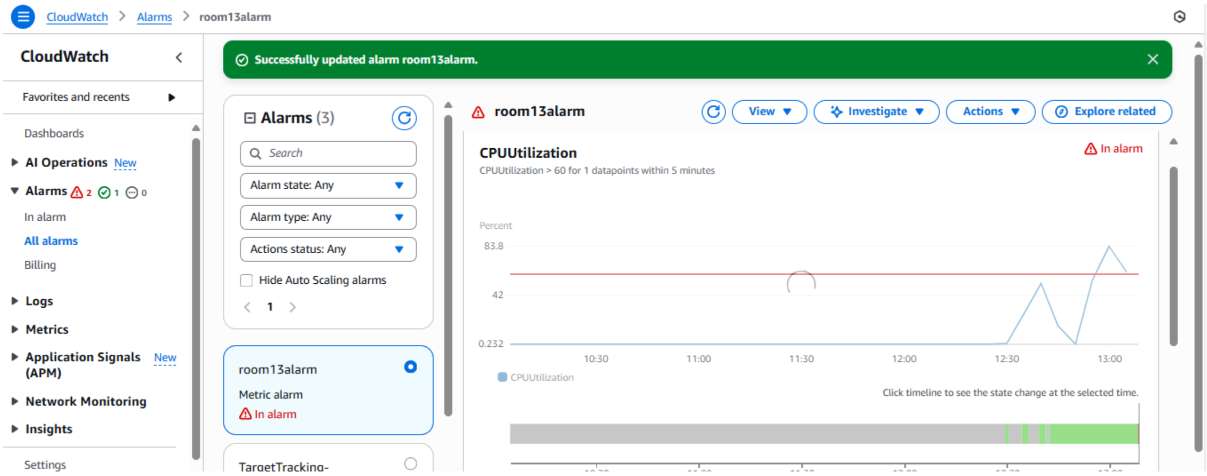
The ALB DNS endpoint was used to verify that traffic was correctly routed to healthy EC2 instances.



```
stress: info: [6902] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6902] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6919] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6919] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6922] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6922] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6925] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6925] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6928] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6928] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6931] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6931] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6934] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6934] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6937] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6937] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ stress --cpu 2 --timeout 60
stress: info: [6940] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [6940] successful run completed in 60s
[ec2-user@ip-10-0-3-5 ~]$ 
```

5.2 Load Simulation

A stress tool was used to simulate traffic surges. Auto Scaling behavior was observed, confirming that additional instances were launched and terminated as expected and The instance alarm was successfully triggered once CPU utilization exceeded the set threshold.



Successfully updated alarm room13alarm.

room13alarm

CPUUtilization

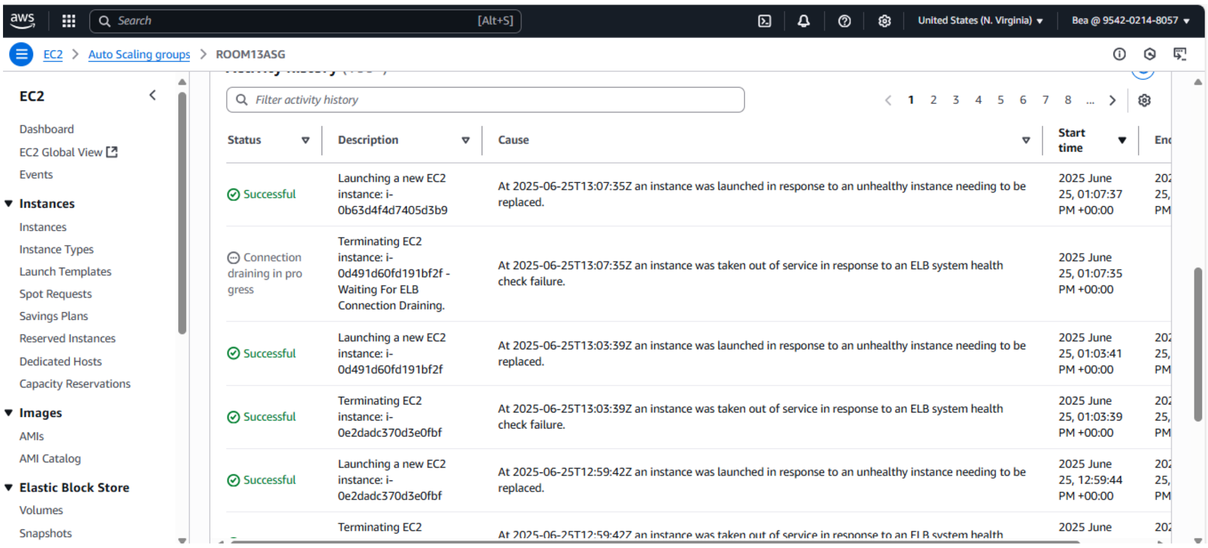
CPUUtilization > 60 for 1 datapoints within 5 minutes

Click timeline to see the state change at the selected time.

5.3 Monitoring Review

CloudWatch dashboards were reviewed for:

- CPU utilization trends
- Target health checks
- Scaling activity history



Status	Description	Cause	Start time	End
Successful	Launching a new EC2 instance: i-0b63d4f4d7405d3b9	At 2025-06-25T13:07:35Z an instance was launched in response to an unhealthy instance needing to be replaced.	2025 June 25, 01:07:37 PM +00:00	202 25, PM
Connection draining in pro gress	Terminating EC2 instance: i-0d491d60fd191bf2f - Waiting For ELB Connection Draining.	At 2025-06-25T13:07:35Z an instance was taken out of service in response to an ELB system health check failure.	2025 June 25, 01:07:35 PM +00:00	
Successful	Launching a new EC2 instance: i-0d491d60fd191bf2f	At 2025-06-25T13:03:39Z an instance was launched in response to an unhealthy instance needing to be replaced.	2025 June 25, 01:03:41 PM +00:00	202 25, PM
Successful	Terminating EC2 instance: i-0e2dadc370d3e0fbf	At 2025-06-25T13:03:39Z an instance was taken out of service in response to an ELB system health check failure.	2025 June 25, 01:03:39 PM +00:00	202 25, PM
Successful	Launching a new EC2 instance: i-0e2dadc370d3e0fbf	At 2025-06-25T12:59:42Z an instance was launched in response to an unhealthy instance needing to be replaced.	2025 June 25, 12:59:44 PM +00:00	202 25, PM
	Terminating EC2	At 2025-06-25T12:59:42Z an instance was taken out of service in response to an ELB system health	2025 June	202

5.4 Optimization Actions

Based on test results, the following optimizations were applied:

- Adjusted scaling thresholds for responsiveness
- Tuned instance types and cooldown periods
- Enhanced logging and alerting rules

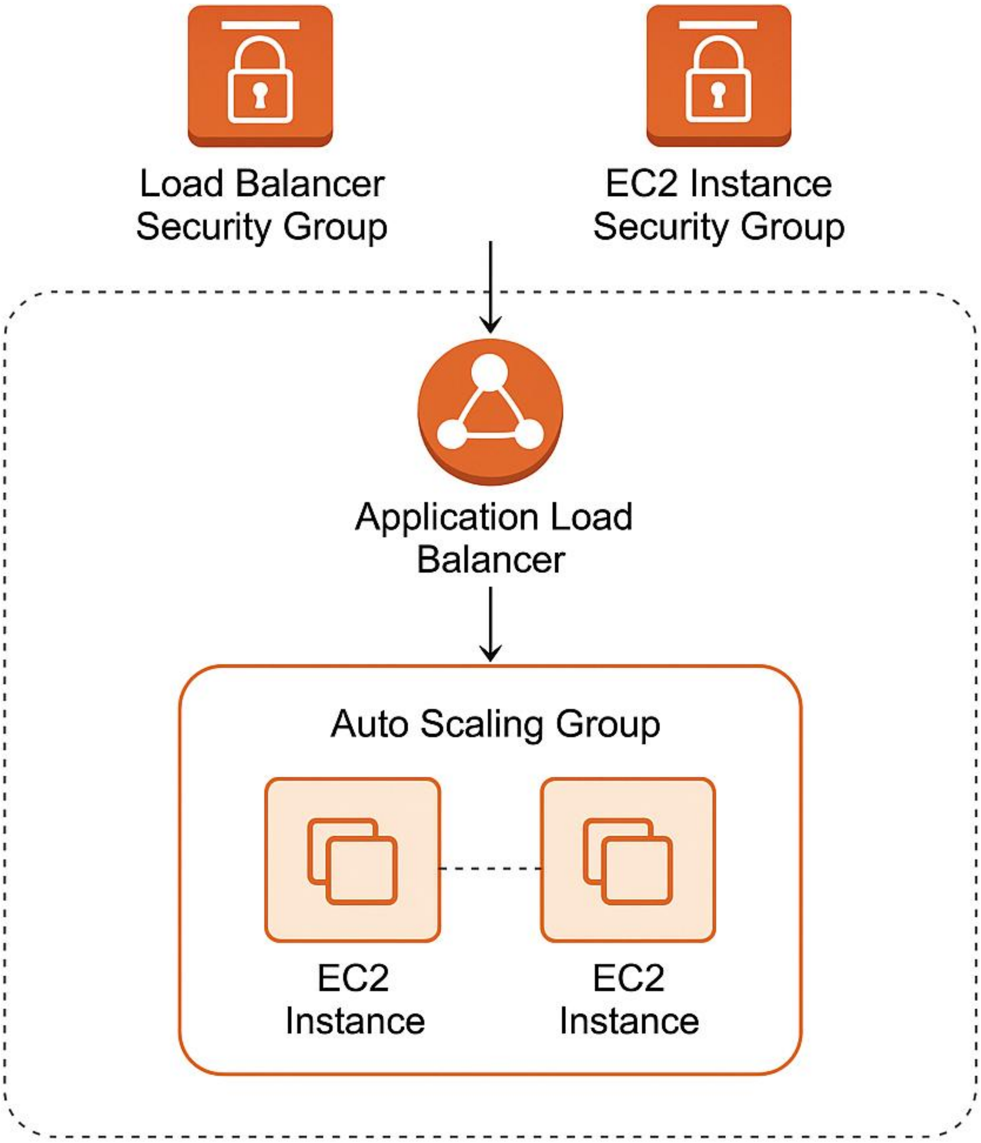
Final Outcome

The implementation successfully delivered a secure, auto-scaled, and load-balanced infrastructure using AWS-native services. The key components included:

- **Elastic Load Balancer (ALB)**
- **Auto Scaling Group (ASG)**
- **EC2 Instances launched from a custom AMI**
- **Amazon CloudWatch for monitoring and scaling triggers**
- **Tightly scoped Security Groups for controlled access**

This architecture ensures high availability, performance under load, and operational efficiency in dynamic environments.

ARCHITECTURAL DIAGRAM FOR LOAD BALANCER AND AUTO SCALING


Load Balancer Security Group
EC2 Instance Security Group
Application Load Balancer
Auto Scaling Group
EC2 Instance
EC2 Instance

Summary and conclusion

The report presents the design and deployment of a secure and scalable AWS infrastructure for a web application. The team configured security groups and launched EC2 instances with a basic web server. An Application Load Balancer was set up to route traffic across healthy instances, while an Auto Scaling Group ensured responsiveness using dynamic CPU-based policies. The Load was tested to confirm performance under varying demands and optimizations. The instance alarm was successfully triggered once CPU utilization exceeded the set threshold. The project successfully depicted a cloud-native solution with high availability and adaptability for modern web workloads.

Capstone Project 1 – Part 3:

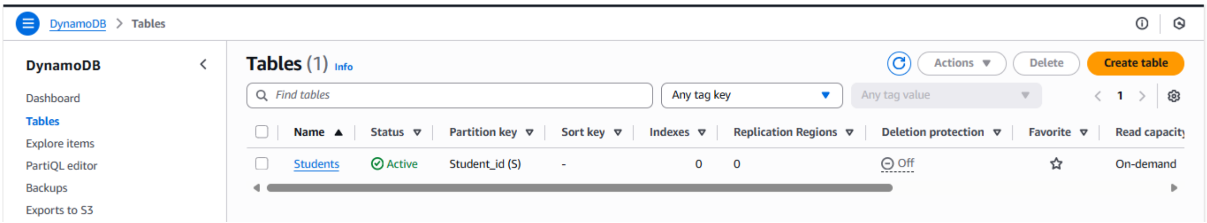
Implementation Report: Integrating EC2 with DynamoDB Using Boto3

Step 1: DynamoDB Table Setup

The project began by creating a DynamoDB table named `Students` to store structured student data. The following actions were completed:

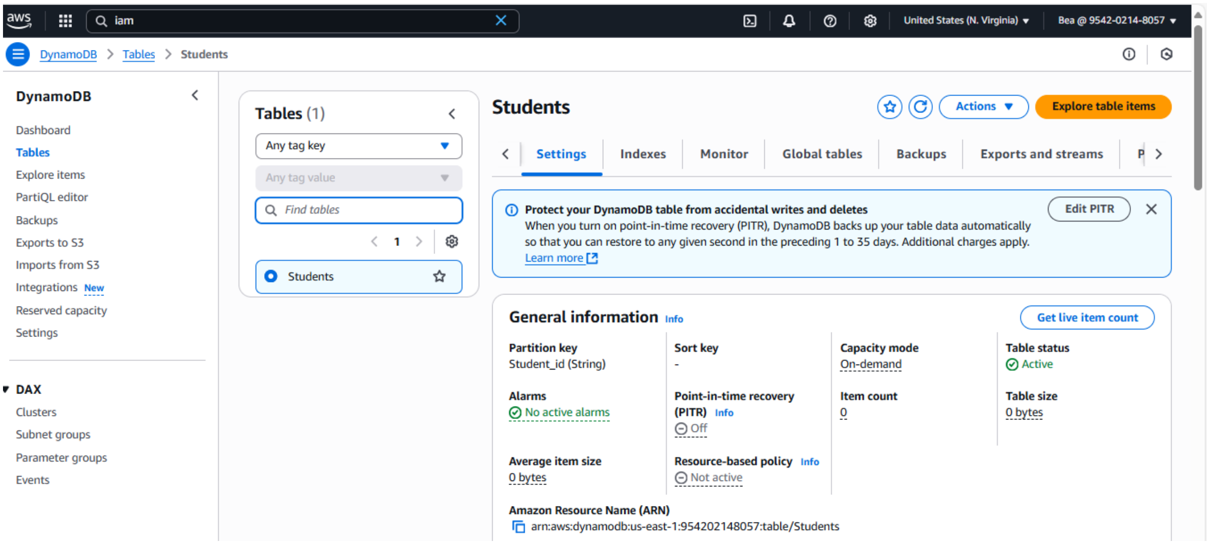
1. Logged into the **AWS Management Console**.
2. Navigated to the **DynamoDB** service.
3. Clicked **“Create table.”**
4. Provided the table name: `Students`.
5. Set the **Partition key** to `Student_id` of type **String**.
6. Retained default settings for the rest of the configuration.
7. Clicked **“Create”** to provision the table.

Upon successful creation, the `Students` table was ready for read and write operations.



Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
Students	✓ Active	Student_id (S)	-	0	0	⊖ Off	☆	On-demand



Students

Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply.
Learn more

General information Info

Partition key: Student_id (String)
Sort key: -
Capacity mode: On-demand
Table status: ✓ Active
Alarms: ✓ No active alarms
Point-in-time recovery (PITR) Info: ⊖ Off
Item count: 0
Table size: 0 bytes
Average item size: 0 bytes
Resource-based policy Info: ⊖ Not active
Amazon Resource Name (ARN): arn:aws:dynamodb:us-east-1:954202148057:table/Students

Step 2: EC2 Instance Preparation

An EC2 instance was configured to serve as the interface for inserting records into DynamoDB via Python scripts. The following setup was completed:

2.1 Launch EC2 Instance

- An EC2 instance was launched with **Amazon Linux 2**.
- The instance had **Python 3** installed as part of the setup.

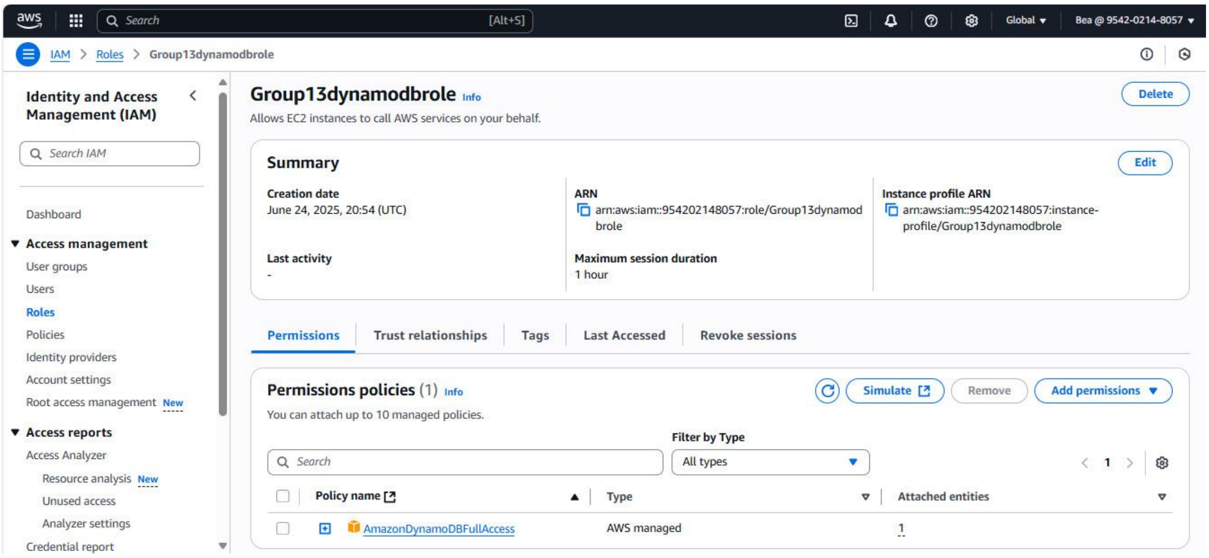
2.2 Create IAM Role with DynamoDB Access

To enable the EC2 instance to communicate securely with DynamoDB:

1. Navigated to **IAM → Roles** and selected **Create role**.
2. Selected **AWS service** as the trusted entity and chose **EC2** as the use case.
3. Attached the **`AmazonDynamoDBFullAccess`** policy.
4. Provided a role name and created the role.

2.3 Attach IAM Role to EC2 Instance

1. In the **EC2 Console**, the target instance was selected.
2. Under **Actions → Security → Modify IAM Role**, the created role was attached.
3. Changes were saved to finalize role assignment.



Group13dynamodbrole Info

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date: June 24, 2025, 20:54 (UTC)

ARN: arn:aws:iam::954202148057:role/Group13dynamodbrole

Instance profile ARN: arn:aws:iam::954202148057:instance-profile/Group13dynamodbrole

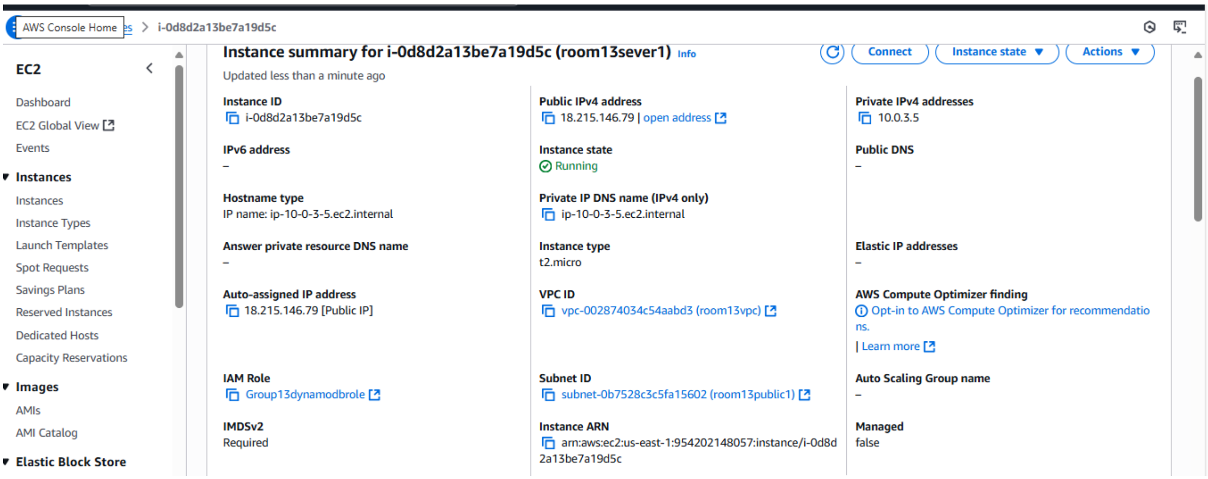
Last activity: -

Maximum session duration: 1 hour

Permissions policies (1) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	1



Instance summary for i-0d8d2a13be7a19d5c (room13sever1) Info

Updated less than a minute ago

Instance ID
i-0d8d2a13be7a19d5c

Public IPv4 address
18.215.146.79 | open address

Private IPv4 addresses
10.0.3.5

IPv6 address
–

Instance state
Running

Public DNS
–

Hostname type
IP name: ip-10-0-3-5.ec2.internal

Private IP DNS name (IPv4 only)
ip-10-0-3-5.ec2.internal

Answer private resource DNS name
–

Instance type
t2.micro

Elastic IP addresses
–

Auto-assigned IP address
18.215.146.79 [Public IP]

VPC ID
vpc-002874034c54aabd3 (room13vpc)

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations.
| Learn more

IAM Role
Group13dynamodbrole

Subnet ID
subnet-0b7528c3c5fa15602 (room13public1)

Auto Scaling Group name
–

IMDSv2
Required

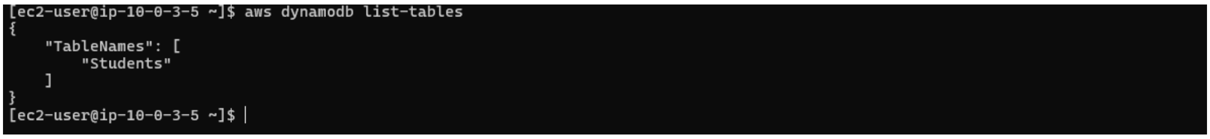
Instance ARN
arn:aws:ec2:us-east-1:954202148057:instance/i-0d8d2a13be7a19d5c

Managed
false

2.4 Verify Permissions

To confirm access to DynamoDB, the command below was used

```
aws dynamodb list-tables
```



```
[ec2-user@ip-10-0-3-5 ~]$ aws dynamodb list-tables
{
    "TableNames": [
        "Students"
    ]
}
[ec2-user@ip-10-0-3-5 ~]$ |
```

The table `Students` appeared in the output, the permissions were verified to be working correctly.

Step 3: Install Boto3 on EC2

To interact with AWS services from Python, the **Boto3 library** was installed as follows:

1. Verified Python 3 installation:

   ```
   python3 --version
   ```

2. Python 3 and pip were installed:

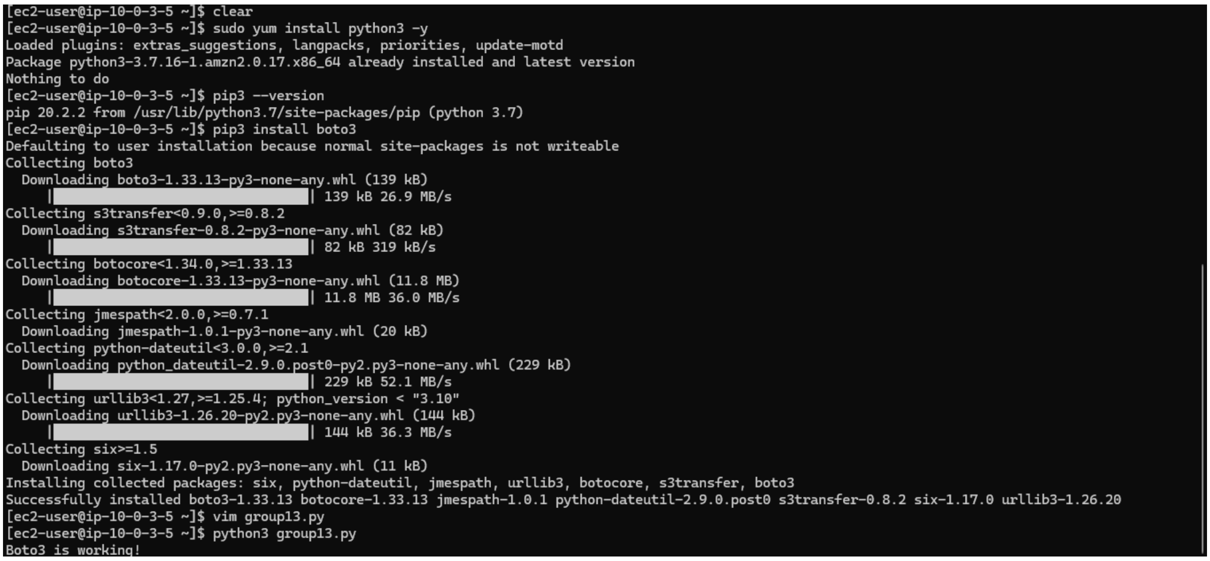
   ```
   sudo yum install python3 -y
   sudo yum install python3-pip -y
   ```

3. Installed Boto3:

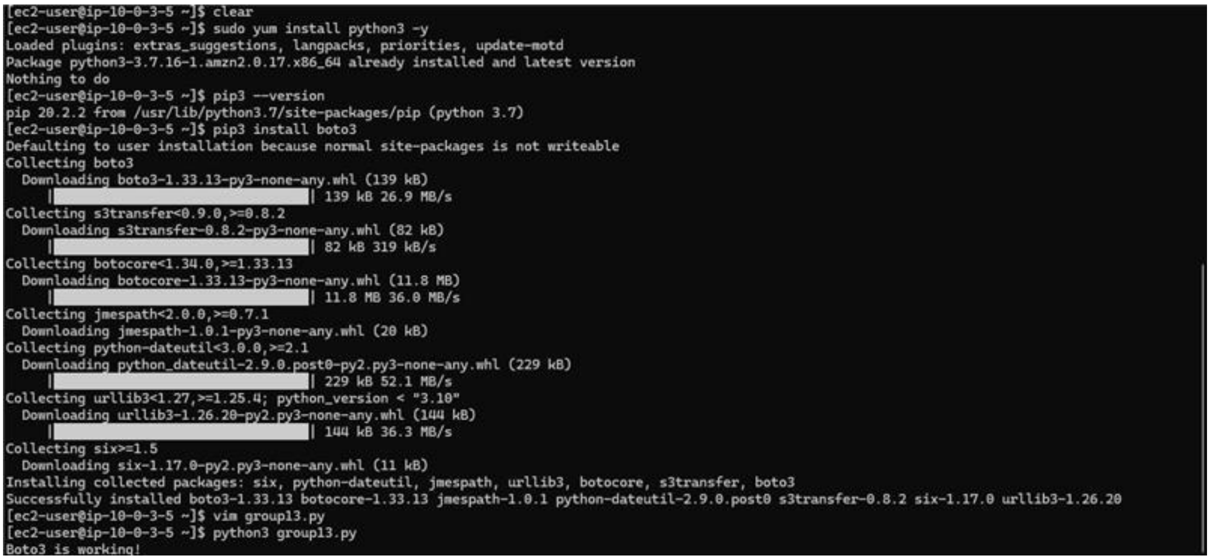
   ```
   pip3 install boto3 --user
   ```

4. Verified Boto3 installation:

   ```
   python3
   >>> import boto3
   >>> print("Boto3 is working!")
   >>> exit()
   ```



```
[ec2-user@ip-10-0-3-5 ~]$ clear
[ec2-user@ip-10-0-3-5 ~]$ sudo yum install python3 -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package python3-3.7.16-1.amzn2.0.17.x86_64 already installed and latest version
Nothing to do
[ec2-user@ip-10-0-3-5 ~]$ pip3 --version
pip 20.2.2 from /usr/lib/python3.7/site-packages/pip (python 3.7)
[ec2-user@ip-10-0-3-5 ~]$ pip3 install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 26.9 MB/s
Collecting s3transfer<0.9.0,>=0.8.2
  Downloading s3transfer-0.8.2-py3-none-any.whl (82 kB)
     |████████████████████████████████| 82 kB 319 kB/s
Collecting botocore<1.34.0,>=1.33.13
  Downloading botocore-1.33.13-py3-none-any.whl (11.8 MB)
     |████████████████████████████████| 11.8 MB 36.0 MB/s
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 52.1 MB/s
Collecting urllib3<1.27,>=1.25.4; python_version < "3.10"
  Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
     |████████████████████████████████| 144 kB 36.3 MB/s
Collecting six>=1.5
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, python-dateutil, jmespath, urllib3, botocore, s3transfer, boto3
Successfully installed boto3-1.33.13 botocore-1.33.13 jmespath-1.0.1 python-dateutil-2.9.0.post0 s3transfer-0.8.2 six-1.17.0 urllib3-1.26.20
[ec2-user@ip-10-0-3-5 ~]$ vim group13.py
[ec2-user@ip-10-0-3-5 ~]$ python3 group13.py
Boto3 is working!
```



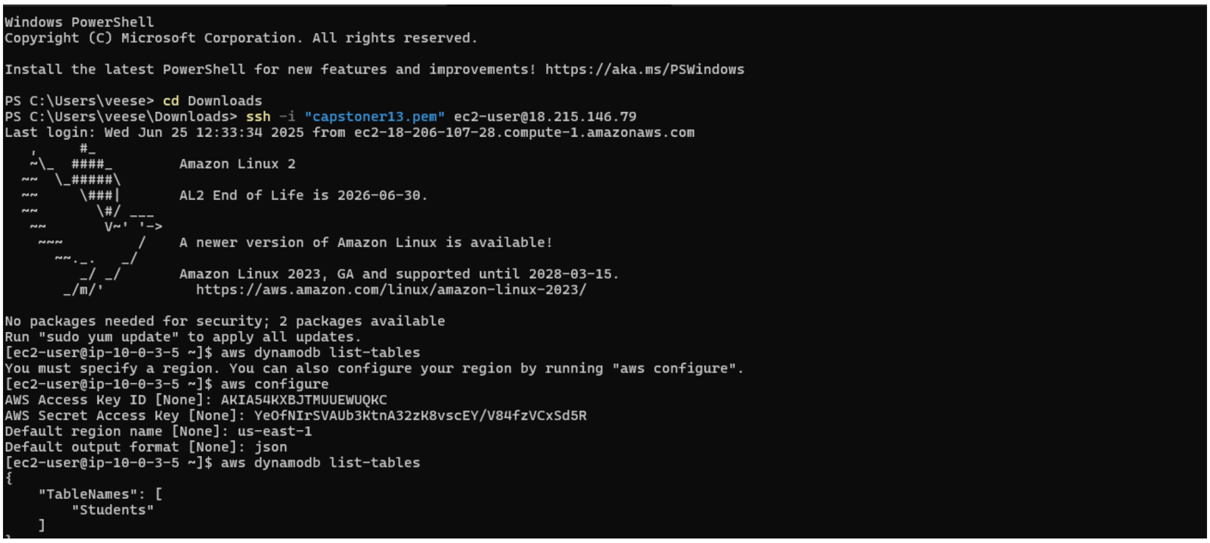
```
[ec2-user@ip-10-0-3-5 ~]$ clear
[ec2-user@ip-10-0-3-5 ~]$ sudo yum install python3 -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package python3-3.7.16-1.amzn2.0.17.x86_64 already installed and latest version
Nothing to do
[ec2-user@ip-10-0-3-5 ~]$ pip3 --version
pip 20.2.2 from /usr/lib/python3.7/site-packages/pip (python 3.7)
[ec2-user@ip-10-0-3-5 ~]$ pip3 install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 26.9 MB/s
Collecting s3transfer<0.9.0,>=0.8.2
  Downloading s3transfer-0.8.2-py3-none-any.whl (82 kB)
     |████████████████████████████████| 82 kB 319 kB/s
Collecting botocore<1.34.0,>=1.33.13
  Downloading botocore-1.33.13-py3-none-any.whl (11.8 MB)
     |████████████████████████████████| 11.8 MB 36.0 MB/s
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 52.1 MB/s
Collecting urllib3<1.27,>=1.25.4; python_version < "3.10"
  Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
     |████████████████████████████████| 144 kB 36.3 MB/s
Collecting six>=1.5
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, python-dateutil, jmespath, urllib3, botocore, s3transfer, boto3
Successfully installed boto3-1.33.13 botocore-1.33.13 jmespath-1.0.1 python-dateutil-2.9.0.post0 s3transfer-0.8.2 six-1.17.0 urllib3-1.26.20
[ec2-user@ip-10-0-3-5 ~]$ vim group13.py
[ec2-user@ip-10-0-3-5 ~]$ python3 group13.py
Boto3 is working!
```

Step 4: Python Script to Interact with DynamoDB

A Python script was developed to log student data into the `Students` table. The process included:

1. Created a new Python file:

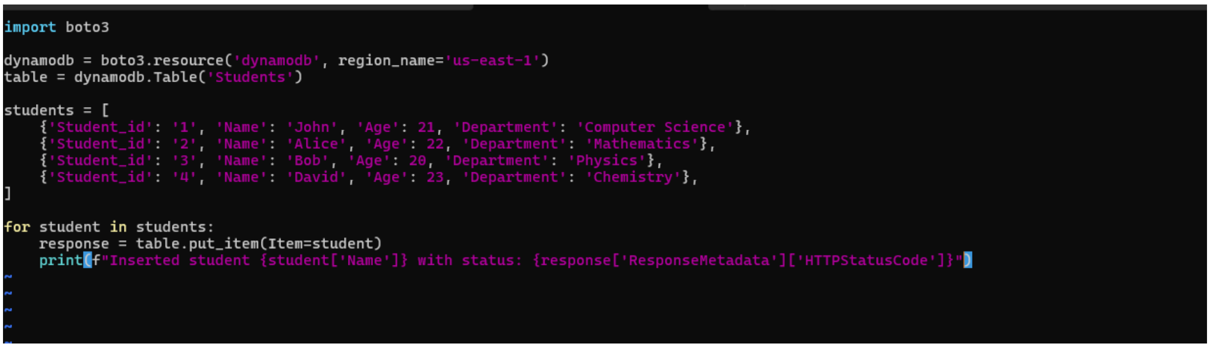
   ```
   nano student_logger.py
   ```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\veese> cd Downloads
PS C:\Users\veese\Downloads> ssh -i "capstoner13.pem" ec2-user@18.215.146.79
Last login: Wed Jun 25 12:33:34 2025 from ec2-18-206-107-28.compute-1.amazonaws.com
   ,     #_
   ~\_  ####_        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/

No packages needed for security; 2 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-3-5 ~]$ aws dynamodb list-tables
You must specify a region. You can also configure your region by running "aws configure".
[ec2-user@ip-10-0-3-5 ~]$ aws configure
AWS Access Key ID [None]: AKIA54KXBJTMUUEWUQKC
AWS Secret Access Key [None]: YeOfNIrSVAUb3KtnA32zK8vscEY/V84fzVCxSd5R
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-10-0-3-5 ~]$ aws dynamodb list-tables
{
    "TableNames": [
        "Students"
    ]
```

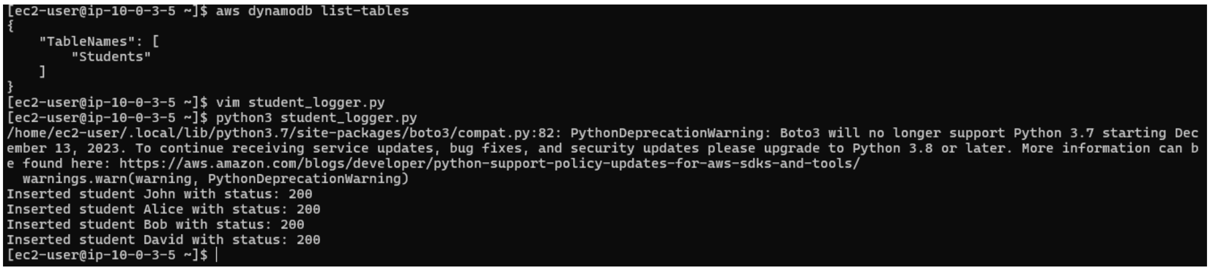
2. Added the following code to insert multiple records:

   ```
   python
   CopyEdit
   import boto3

   # Create a DynamoDB resource in us-east-1 region
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

   # Reference the table
   table = dynamodb.Table('Students')

   # Sample student data
   students = [
       {'student_id': '001', 'name': 'Alice Appiah', 'course': 'AWS Cloud Fundamentals'},
       {'student_id': '002', 'name': 'Rita Okloo', 'course': 'Linux Basics'}
   ]

   # Insert each student into the table
   for student in students:
       response = table.put_item(Item=student)
       print(f"Inserted: {student['name']} | Status: {response['ResponseMetadata']['HTTPStatusCode']}")
   ```



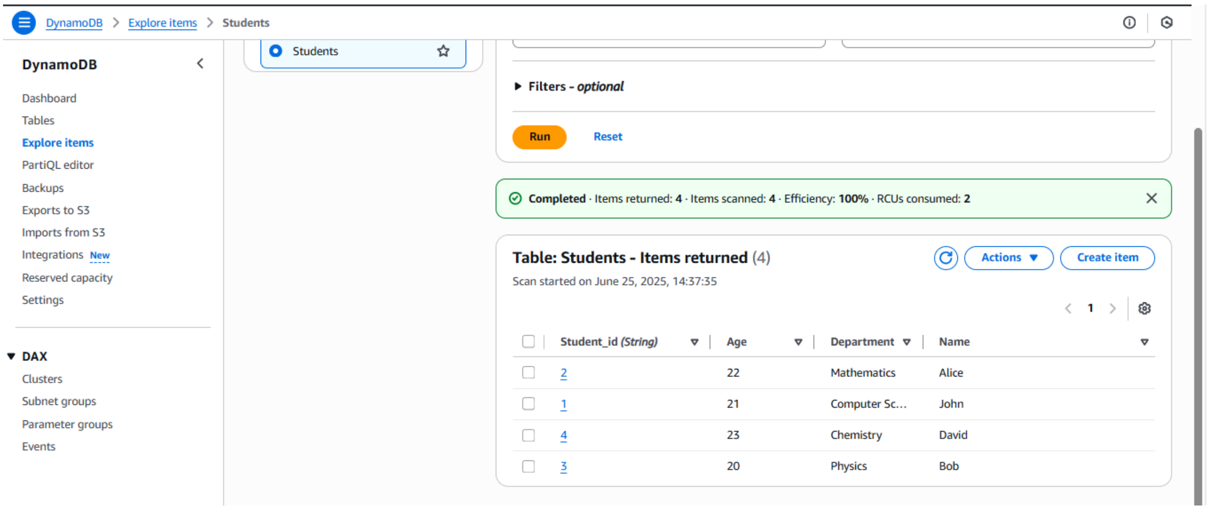
```
import boto3

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table = dynamodb.Table('Students')

students = [
    {'Student_id': '1', 'Name': 'John', 'Age': 21, 'Department': 'Computer Science'},
    {'Student_id': '2', 'Name': 'Alice', 'Age': 22, 'Department': 'Mathematics'},
    {'Student_id': '3', 'Name': 'Bob', 'Age': 20, 'Department': 'Physics'},
    {'Student_id': '4', 'Name': 'David', 'Age': 23, 'Department': 'Chemistry'},
]

for student in students:
    response = table.put_item(Item=student)
    print(f"Inserted student {student['Name']} with status: {response['ResponseMetadata']['HTTPStatusCode']}")
~
~
~
~
```

3. Saved and exited the file using:
   - `Ctrl+O` to save
   - `Enter` to confirm
   - `Ctrl+X` to exit
4. Ran the script using:
   `python3 student_logger.py`



```
[ec2-user@ip-10-0-3-5 ~]$ aws dynamodb list-tables
{
    "TableNames": [
        "Students"
    ]
}
[ec2-user@ip-10-0-3-5 ~]$ vim student_logger.py
[ec2-user@ip-10-0-3-5 ~]$ python3 student_logger.py
/home/ec2-user/.local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting Dec
ember 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can b
e found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
Inserted student John with status: 200
Inserted student Alice with status: 200
Inserted student Bob with status: 200
Inserted student David with status: 200
[ec2-user@ip-10-0-3-5 ~]$ |
```

Upon successful execution, the student records were inserted into the DynamoDB table as shown below.



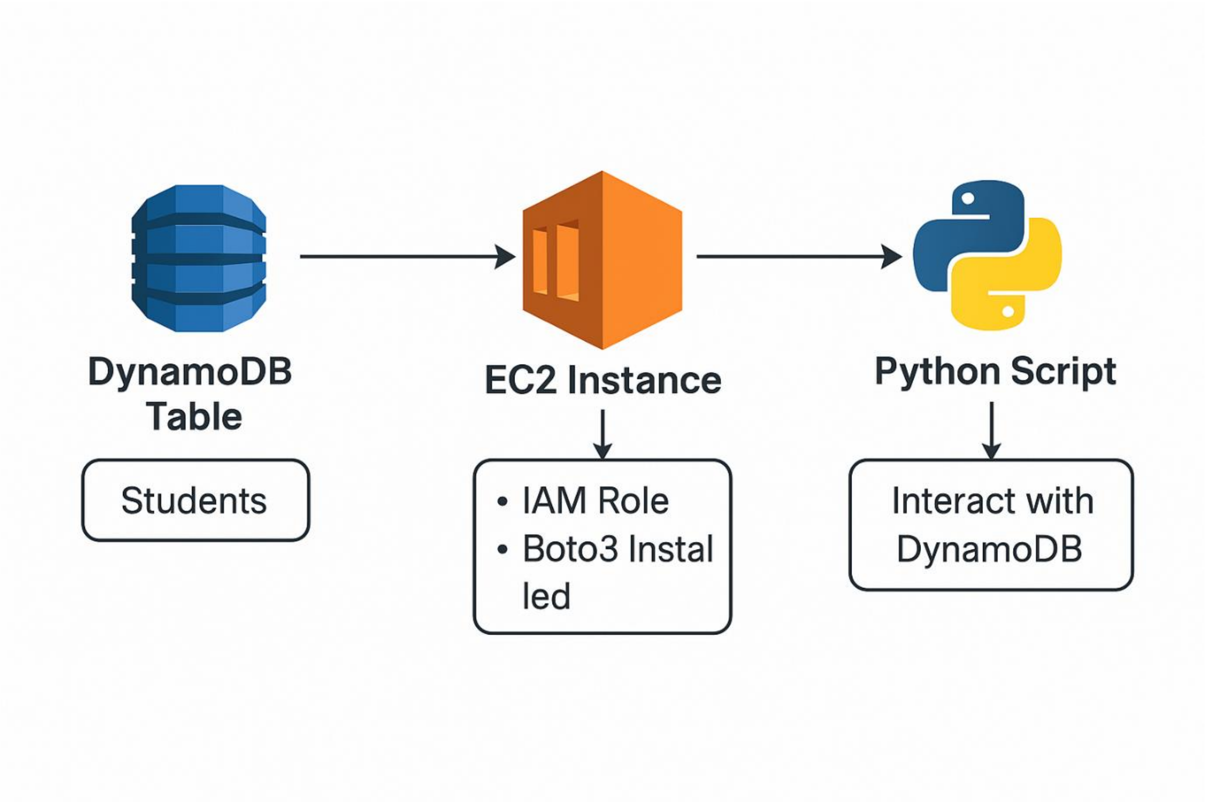
Completed · Items returned: 4 · Items scanned: 4 · Efficiency: 100% · RCUs consumed: 2

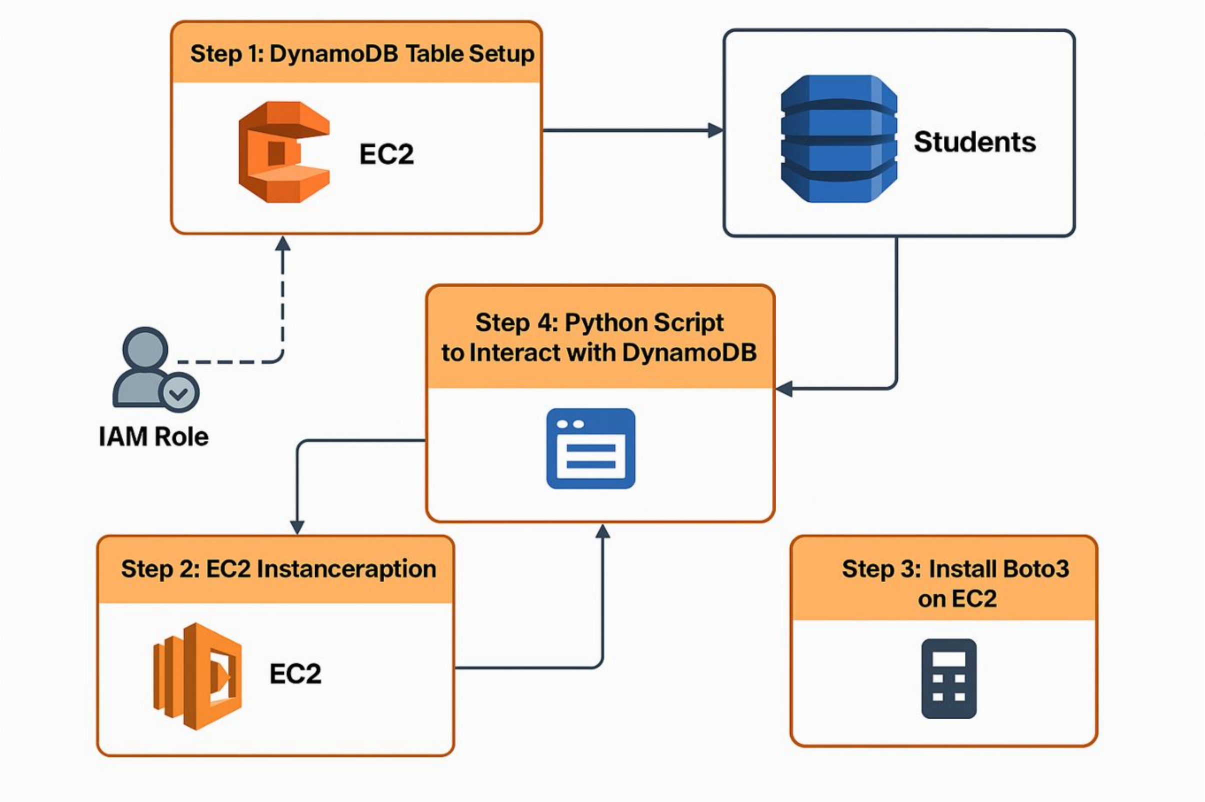
Table: Students - Items returned (4)

Scan started on June 25, 2025, 14:37:35

Student_id (String)	Age	Department	Name
2	22	Mathematics	Alice
1	21	Computer Sc...	John
4	23	Chemistry	David
3	20	Physics	Bob

ARCHITECTURAL DIAGRAMS FOR DYNAMO DB


DynamoDB Table
Students
EC2 Instance
• IAM Role
• Boto3 Installed
Python Script
Interact with DynamoDB


Step 1: DynamoDB Table Setup
EC2
Students
IAM Role
Step 4: Python Script to Interact with DynamoDB
Step 2: EC2 Instanceraption
EC2
Step 3: Install Boto3 on EC2

Summary and Conclusion

This project aimed to integrate **Amazon EC2** with **DynamoDB** using the Python **Boto3 SDK** to automate data logging. The team set up **DynamoDB to** provision a `Students` table with `Student_id` as the partition key. Launched an EC2 instance with Python installed, attached an IAM role granting DynamoDB permissions, and confirmed access using AWS CLI. The team also Installed and validated the Boto3 SDK to interact with AWS services by writing and executing a script to insert student data into the DynamoDB table. This created a reliable and scripted pipeline to move data from EC2 into DynamoDB, reflecting a fundamental backend integration pattern on AWS.

The successful setup confirms that **EC2 instances can securely communicate with DynamoDB** using properly scoped IAM roles and the Boto3 library.

THE END.