

Automated data management system on AWS – DynamoDB and EC2

Michael Ohene Aboagye

Cloud Engineer

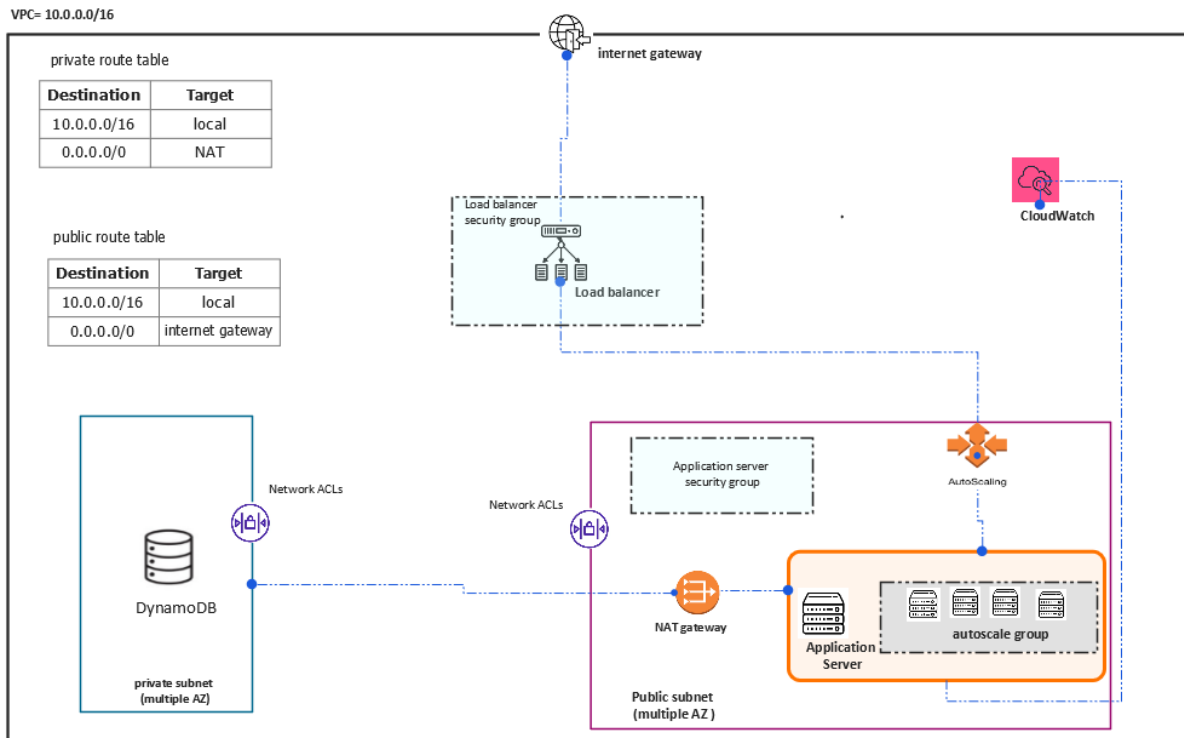
7/7/2025

SUMMARY

This project demonstrates the setup of a cloud solution, specifically a logger database using DynamoDB. Security measures were prioritized by creating a custom Virtual Private Cloud (VPC), setting up subnets for the respective instances, and implementing security features such as Network Access Control Lists (NACLs) and security groups.

To enhance the reliability of the application, load balancing and auto-scaling features were implemented. The load balancer distributes incoming traffic evenly to an auto-scaling group, allowing for the addition of instances to handle increased loads, thus minimizing potential downtime.

Python 3 was used as the scripting tool to automate the logging of data into the DynamoDB database, utilizing Boto3, the Amazon SDK for Python.



fig

fig.1 Automated data management system on AWS Architecture Diagram

1. SETTING UP THE NETWORK ARCHITECTURE

- 1.1. Create a **VPC** which serves as an isolated space on AWS cloud environment to host application,
Set VPC with CIDR block of 10.0.0.0/16

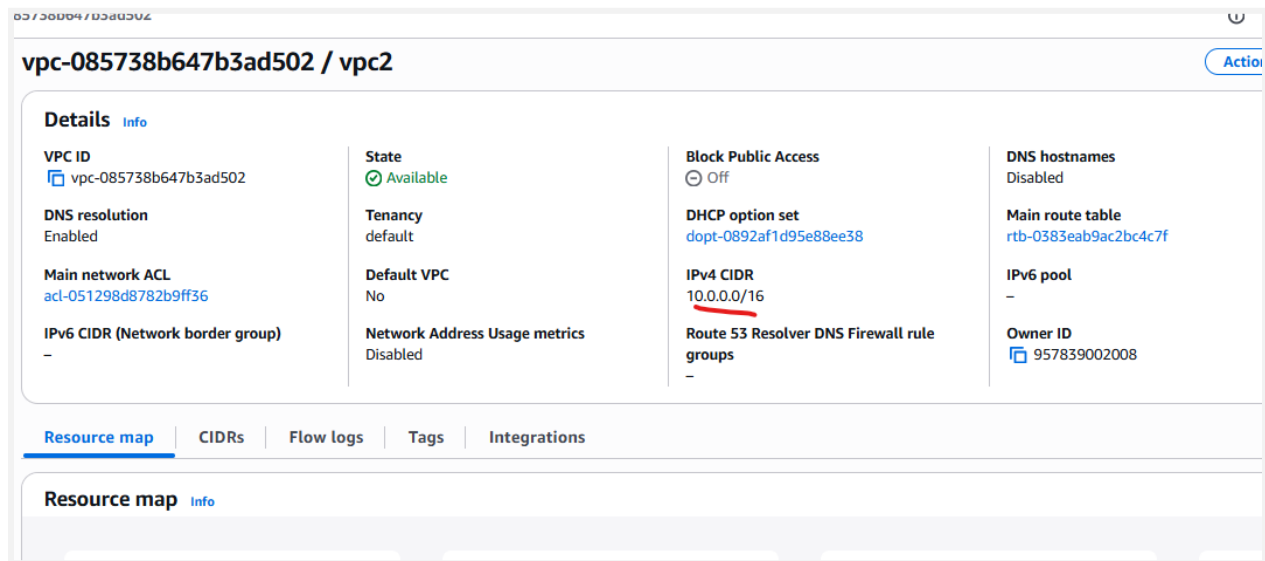


Fig 2. VPC configuration

- 1.2. Now setup subnets to host the instances and resource.

Public Subnets:

Created in two Availability Zones.

Assigned for Load Balancer and NAT Gateway.

CIDRs: 10.0.1.0/24, 10.0.3.0/24

Private Subnets:

Created in different AZs for EC2 and backend apps.

CIDRs: 10.0.2.0/24, 10.0.4.0/24

<input type="checkbox"/>	Name	Subnet ID	State	VPC
<input type="checkbox"/>	Public_Subnet_main	subnet-038fdd00f68de3274	Available	vpc-097f09d2a5b67d867
<input type="checkbox"/>	Private_Subnet_main	subnet-03eca2d41878bc323	Available	vpc-097f09d2a5b67d867
<input type="checkbox"/>	private-sub1	subnet-05473d211d8564481	Available	vpc-085738b647b3ad502 vpc2
<input type="checkbox"/>	private-sub2	subnet-06a7ebb427b1cf163	Available	vpc-085738b647b3ad502 vpc2
<input type="checkbox"/>	public-sub2	subnet-00010452ec72faacf	Available	vpc-085738b647b3ad502 vpc2
<input type="checkbox"/>	public-sub1	subnet-0a692b3d52a055b0b	Available	vpc-085738b647b3ad502 vpc2

Fig 3. Public and private subnet for the DynamoDB and the application server.

Private and public subnets was created, the application server was placed in the private subnet to allow access to the internet.

The DynamoDB instance was placed in the private subnet to allow internal traffic only.

1.3. set route tables, NAT gateways

In other to control traffic, both internal and external, two route tables was created. Public route to control public traffics and private route table to control the internal traffics.

to control traffic from the public subnet, **public route table** was attached to the public subnets control inbound and outbound traffic. The traffic was restricted to HTTP, HTTPS, TCP and SSH.

Private routable was attached to the **private subnet** to route traffic internally and also to access the internet via the NAT gateway specifically for downloads and updates.

Public Route Table:

Route: 0.0.0.0/0 → Internet Gateway

Associated with Public Subnets

Private Route Table:


Route: 0.0.0.0/0 → NAT Gateway

Associated with Private Subnets


rtb-0a0cf0968ece9c442


rtb-0a0cf0968ece9c442 / private-route

Details [Info](#)

Route table ID
 rtb-0a0cf0968ece9c442

VPC
vpc-085738b647b3ad502 | vpc2

Main
 No

Owner ID
 957839002008

Explicit subnet associations
2 subnets

[Routes](#) | [Subnet associations](#) | [Edge associations](#) | [Route propagation](#) | [Tags](#)

Routes (2)


Destination	Target	Status
0.0.0.0/0	nat-0144ff8a4bd8bbc9a	✓ Active
10.0.0.0/16	local	✓ Active

restricted to internal traffic only


Fig 4. Setting up private route tables and attaching to the private subnets.


rtb-0309acbeec06f6f87 / public-route

Details [Info](#)

Route table ID
 rtb-0309acbeec06f6f87

VPC
vpc-085738b647b3ad502 | vpc2

Main
 No

Owner ID
 957839002008

Explicit subnet association
2 subnets

[Routes](#) | [Subnet associations](#) | [Edge associations](#) | [Route propagation](#) | [Tags](#)

Routes (2)

Destination	Target	Status
0.0.0.0/0	igw-05a54a3c94df4e96a	✓ Active
10.0.0.0/16	local	✓ Active

Fig 5. Public route controls traffic outside the VPC. It routes traffic to the internet via the internet gateway.

1.4. setup security groups for the load balancer as “**load-balancer-sg**” and the EC instances/application server(s) as “**app-server-sg**”.

First, we create security group for the load balancer to filter out inbound traffic to HTTP, HTTPS, and SSH.

then, we set up security group for the EC2 instance to allow traffic from the load balancer only.

sg-072e7ddf75d4ef2a2 - load-balancer-sg

Details

Security group name load-balancer-sg	Security group ID sg-072e7ddf75d4ef2a2	Description allow traffic for load balancer	VPC ID vpc-085738b647b3ad502
Owner 957839002008	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-074a9603ac3bda564	IPv4	HTTP	TCP	80
-	sgr-078468aea569374b6	IPv4	HTTPS	TCP	443
-	sgr-08bb5f61bd0c32e6f	IPv4	SSH	TCP	22

Fig 6. Load balancer security group configuration.

sg-0803623744a6cae22 - app-server-sg

Details

Security group name app-server-sg	Security group ID sg-0803623744a6cae22	Description allow traffic from the load balancer	VPC ID vpc-085738b647b3ad502
Owner 957839002008	Inbound rules count 4 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules (4)

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0ac5760f3e3a09c7c	-	SSH	TCP	22
-	sgr-0f5b7711f96adb070	IPv4	SSH	TCP	22
-	sgr-0998433c3ffc97319	-	HTTPS	TCP	443
-	sgr-064ba139301b34c5c	-	HTTP	TCP	80

Fig 7. EC2 instance security group configuration.

2. PREPARE THE EC2 INSTANCE.

2.1. Create an EC2 instance with the required configurations and test for connectivity.

The screenshot displays the AWS Management Console interface for EC2 instances. At the top, there's a header with 'Instances (1/2)' and 'Info'. Below this is a search bar and a table of instances. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Actions. Two instances are listed: 'app-server-ec2' (Terminated) and 'app-server-1' (Running). The 'app-server-1' instance is selected, and its details are shown below the table. The details are organized into sections: Instance summary, Public IPv4 address, Private IPv4 addresses, Instance state, Private IP DNS name (IPv4 only), Instance type, VPC ID, Subnet ID, Instance ARN, Elastic IP addresses, AWS Compute Optimizer finding, Auto Scaling Group name, and Managed. The 'app-server-1' instance is a t2.micro instance, running in the us-east-1 region, with a public IP address of 35.173.127.107 and a private IP address of 10.0.1.172. It is associated with the vpc-085738b647b3ad502 and subnet-0a692b3d52a055b0b.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Actions
app-server-ec2	i-05b90b8f77791e3aa	Terminated	t2.micro	-	View alarms +	u
app-server-1	i-095cae4d9d6d5ab35	Running	t2.micro	2/2 checks passed	View alarms +	u

i-095cae4d9d6d5ab35 (app-server-1)

- Instance summary**
 - Instance ID: i-095cae4d9d6d5ab35
 - IPv6 address: -
 - Hostname type: P name: ip-10-0-1-172.ec2.internal
 - Answer private resource DNS name: -
 - Auto-assigned IP address: 35.173.127.107 [Public IP]
 - IAM Role: -
 - IMDSv2: Required
- Public IPv4 address**: 35.173.127.107 | open address
- Instance state**: Running
- Private IP DNS name (IPv4 only)**: ip-10-0-1-172.ec2.internal
- Instance type**: t2.micro
- VPC ID**: vpc-085738b647b3ad502 (vpc2)
- Subnet ID**: subnet-0a692b3d52a055b0b (public-sub1)
- Instance ARN**: arn:aws:ec2:us-east-1:957839002008:instance/i-095cae4d9d6d5ab35

- Private IPv4 addresses**: 10.0.1.172
- Public DNS**: -
- Elastic IP addresses**: -
- AWS Compute Optimizer finding**: Opt-in to AWS Compute Optimizer for recommendations. Learn more
- Auto Scaling Group name**: -
- Managed**: false

Fig 8. EC2 instance detail configuration.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[ec2-user@ip-10-0-1-172 ~]$ sudo yum install nginx -y
Last metadata expiration check: 0:06:41 ago on Wed Jul 2 22:31:41 2025.
Dependencies resolved.

=====
Package                                Architecture  Version                                Repository
=====
Installing:
nginx                                  x86_64        1:1.28.0-1.amzn2023.0.1              amazonlinux
Installing dependencies:
generic-logos-httpd                   noarch        18.0.0-12.amzn2023.0.3              amazonlinux
gperftools-libs                       x86_64        2.9.1-1.amzn2023.0.3                amazonlinux
libunwind                             x86_64        1.4.0-5.amzn2023.0.2                amazonlinux
nginx-core                            x86_64        1:1.28.0-1.amzn2023.0.1              amazonlinux
nginx-filesystem                      noarch        1:1.28.0-1.amzn2023.0.1              amazonlinux
nginx-mimetypes                      noarch        2.1.49-3.amzn2023.0.3              amazonlinux
=====

Transaction Summary
=====
Install 7 Packages

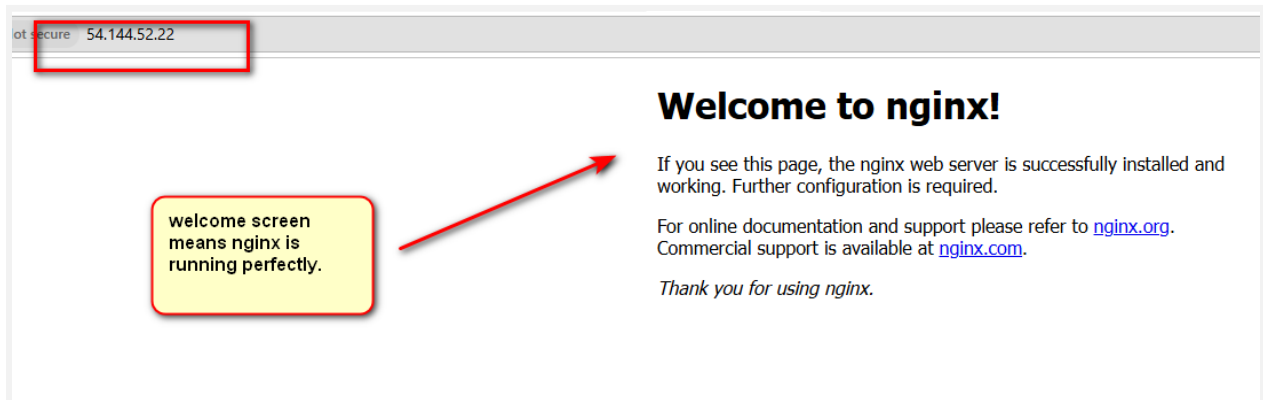
Total download size: 1.1 M
Installed size: 3.7 M
Downloading Packages:
(1/7): libunwind-1.4.0-5.amzn2023.0.2.x86_64.rpm 1.8 MB/s | 66 kB 00:01
(2/7): generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch.rpm 485 kB/s | 19 kB 00:01
=====
Ln 3, Col 1 (55 selected) Spaces: 4 UTF-8 CRLF ( ) Plain Text
```

Fig 9. Nginx was installed and run to use as test for internet connectivity.

Run the code below on the terminal to start `nginx` application.

```
“sudo systemctl start nginx”
```

`nginx` welcome screen will be displayed on the web browser by running the public IP address of the EC2 instance.



- 2.2. Create launch template from the current instance, this will be used by the auto-scale to create preconfigured instances as needed.

Launch template contents
Specify the details of your launch template below. Leaving a field blank will result in the field not being included in the launch template.

▼ Application and OS Images (Amazon Machine Image) - required [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to start an instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents

My AMIs

Quick Start

☒ Owned by me

☐ Shared with me

Amazon Machine Image (AMI)

app-server-img1
ami-0ad4e8c1324a0e3c5
2025-07-03T07:35:23.000Z

Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred

Description
ec2 image

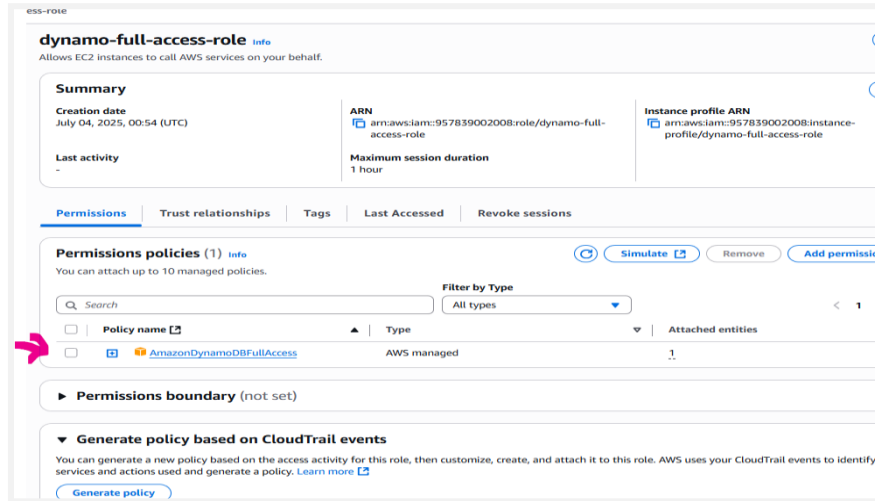
Architecture	AMI ID
x86_64	ami-0ad4e8c1324a0e3c5

Fig 10. Creating launch template from the preconfigured AMI

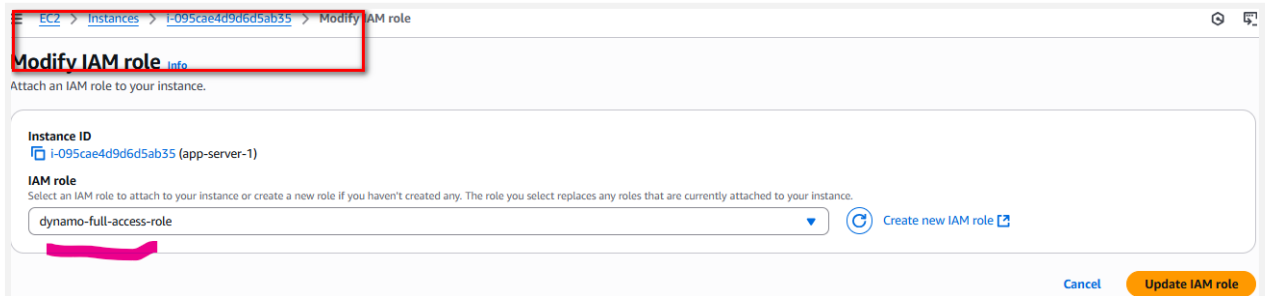
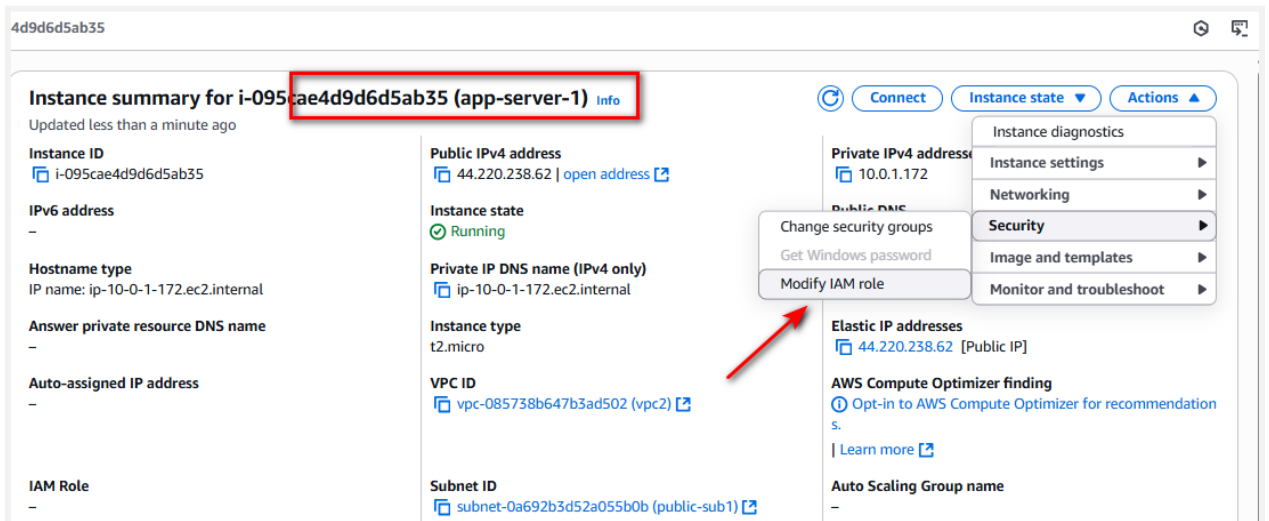
2.3. set up IAM role for the ec2 instance to access DynamoDB resource.

Here, the ec2 instance was allowed administrative access to DynamoDB resource.

i. Create role (AmazonDynamoDBFullAccess)



ii. Attach role to the instance



3. SETUP LOAD BALANCER AND AUTOSCALING GROUP.

3.1.set up the load balancer, set the listeners to the required target group, then load balancer security group was attached (filters traffic to only HTTP, HTTPS, SSH).

The screenshot shows the 'Create Application Load Balancer' page in the AWS Management Console, specifically the 'Review' step. The page is titled 'Load balancers > Create Application Load Balancer'. Below the title, there is a 'Benefits and considerations' section. The main content area is divided into several sections: 'Summary', 'Basic configuration', 'Network mapping', 'Security groups', and 'Listeners and routing'. The 'Summary' section provides a high-level overview of the configuration. The 'Basic configuration' section shows the name 'app-load-balancer', the scheme 'Internet-facing', and the IP address type 'IPv4'. The 'Network mapping' section shows the VPC 'vpc-085738b647b3ad502', the public IPv4 IPAM pool, and the availability zones and subnets. The 'Security groups' section shows the security group 'sg-072e7ddf75d4ef2a2'. The 'Listeners and routing' section shows the listener 'HTTP:80' and the target group 'ec2-target-group'. Red boxes highlight the 'Network mapping' and 'Listeners and routing' sections, with arrows pointing to them from external text boxes. One text box says 'Configured the appropriate VPC and subnets.' and another says 'Listeners and routing: HTTP:80 | Target group: ec2-target-group'.

Fig 11. Summary of the load balancer configuration.

3.2.The auto-scale group was configured.

The screenshot shows the 'Scaling' page in the AWS Management Console for an Auto Scaling group. The page is titled 'Scaling Info'. Below the title, there is a section 'You can resize your Auto Scaling group manually or automatically to meet changes in demand.' followed by 'Scaling limits'. The 'Scaling limits' section allows setting limits on how much the desired capacity can be increased or decreased. The 'Min desired capacity' is set to 1 and the 'Max desired capacity' is set to 4. Below these, there is a section 'Automatic scaling - optional' with the instruction 'Choose whether to use a target tracking policy'. The 'Target tracking scaling policy' is selected. A red box highlights the 'Target tracking scaling policy' section, with an arrow pointing to it from an external text box. The text box contains the following information: 'autoscaling configuration page. desired capacity was set to 2. min capacity = 1 max capacity = 4 target tracking policy was adopted. with CPU threshold of 60%'. Below the 'Target tracking scaling policy' section, there is a 'Scaling policy name' field with the value 'Target Tracking Policy'. The 'Metric type' section shows 'Average CPU utilization' selected. The 'Target value' is set to 60. The 'Instance warmup' section shows '300 seconds'.

Fig 12. Auto-scale configuration.

We set the scaling limits with desired capacity of 2, minimum capacity of 1 and maximum capacity of 4

This means on normal workload, two instances will be actively running. When there this minimum CPU for a period of time, auto-scale terminates the instances and scale-in to only 1 instance. In a situation where there is high amount of workload, auto-scale scales out the number of instance to 4

3.3. the load balancer and autoscaling configurationn was tested for connectivity and response to incoming workload.

EC2 Instance connectivity is tested by running the DNS name of the load balancer, while making sure the nginx is still running on the instance.

Load balancers (1/1)
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter load balancers

<input checked="" type="checkbox"/>	Name	DNS name	Status	VPC ID	Availability Zones	Type	Date created
<input checked="" type="checkbox"/>	ALB	ALB-1693749645.us-east-1...	Active	vpc-085738b647b3ad502	2 Availability Zones	application	July 4, 2025, 00:17 (UTC+00:00)

Load balancer: ALB

Details | Listeners and rules | Network mapping | Resource map | Security | Monitoring | Integrations | Attributes | Capacity | Tags

Details

Load balancer type
Application

Scheme
Internet-facing

Status
Active

Hosted zone
Z355XDOTRQ7X7K

VPC
vpc-085738b647b3ad502

Availability Zones
subnet-0f03d849915ae246b us-east-1b (use1-az4)
subnet-00010452ec72faacf us-east-1a (use1-az2)

Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:957839002008:loadbalancer/app/ALB/0415a700b4022700

DNS name info
ALB-1693749645.us-east-1.elb.amazonaws.com (A Record)

Fig 13. Load balancer with active DNS name.

secure app-load-balancer-369131124.us-east-1.elb.amazonaws.com

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For more information on the nginx web server please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

load balancer DNS name

Fig 14. Load balancer DNS name tested for connectivity.

3.4. we test for the Auto-Scale response of the load balancer on traffic.

i. We install and apply stress test using the code below.

```
sudo yum install stress -y #For Amazon Linux

stress --cpu 4 --timeout 60 #configures and run stress test.
```

ii. After applying the stress test, the auto-scale group adds new instance when the CPU exceed the **60%** threshold and terminated instance when in goes below the 60% threshold.

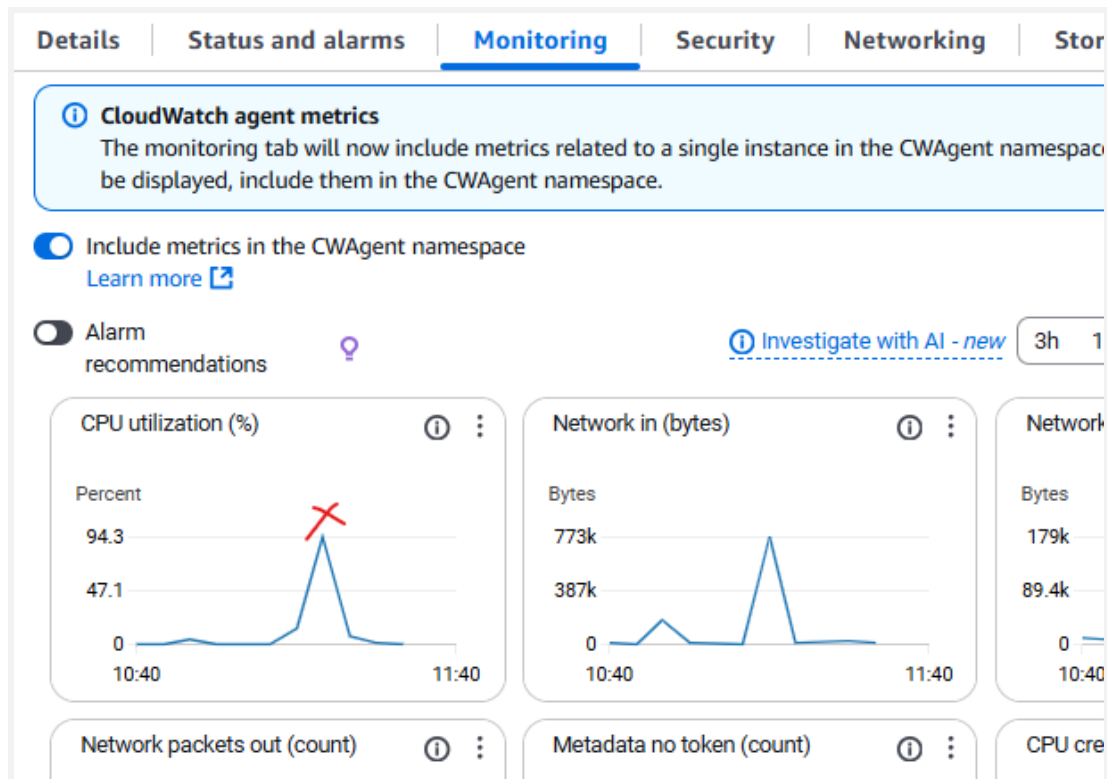


Fig 15. CPU utilization graph exceeding 60% threshold value.

The image shows the AWS EC2 'Instances' console. It displays a list of three EC2 instances. The first instance, 'app-server-1', is highlighted with a red arrow pointing to its Instance ID 'i-00b322171663b27a8'. The second instance, 'app-server-1', is also highlighted with a red arrow pointing to its Instance ID 'i-095cae4d9d6d5ab35'. The third instance, 'app-server-1', is highlighted with a red arrow pointing to its Instance ID 'i-0ad5e0c0105bb05d6'. All instances are in the 'Running' state and are t2.micro instances. The Status check for all instances is '2/2 checks passed'. The Alarm status for all instances is 'View alarms +'. The Availability Zone for all instances is 'us-east-1a'. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
	i-00b322171663b27a8	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b
app-server-1	i-095cae4d9d6d5ab35	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a
	i-0ad5e0c0105bb05d6	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a

Fig.16 new instance being deployed due to auto-scale configuration.

4. SETUP DYNAMODB TABLE, CONFIGURE PYTHON TO CONNECT TO DYNAMODB

4.1. Python3 and boto3, which is an Amazon SDK for python. The code below was used to configure python and boto3 (assuming python3 and pip3 is already installed).

`pip install boto3`

```
[ec2-user@ip-10-0-1-172 ~]$ pip3 install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
    |████████████████████████████████████████| 139 kB 15.8 MB/s
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 MB)
    |████████████████████████████████████████| 13.8 MB 41.7 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (0.10.0)
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    |████████████████████████████████████████| 85 kB 7.9 MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (2.8.1)
from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
```

Fig 17. Installing boto3 through the EC2 CLI.

4.2. Staff_logger.py file was created on the EC2 instance using nano text editor, it contained the code to automatically logs staff information to the table.

```
staff_logger
1
2 import boto3# Create a DynamoDB resource using the us-east-1 region
3
4 dynamodb= boto3.resource('dynamodb', region_name='us-east-1') # Reference your DynamoDB table
5
6 table= dynamodb.Table('staff') # List of staff to add
7
8 staff = [ {'StaffID': '001', 'name': 'Alice Appiah', 'department': 'IT'},
9           {'StaffID': '002', 'name': 'Rita Okai', 'Department': 'Finance'},
10          {'StaffID': '001', 'name': 'Ansong Ampaw', 'Department': 'IT'},
11          {'StaffID': '002', 'name': 'Rita Okloo', 'Department': 'Security'},
12          {'StaffID': '001', 'name': 'Alice Mensah', 'Department': 'IT'},
13          {'StaffID': '002', 'name': 'Rita Okloo', 'Department': 'Finance'},
14          {'StaffID': '001', 'name': 'Philip Mensah', 'Department': 'IT'},
15          {'StaffID': '002', 'name': 'Rita Yao', 'Department': 'Product'},
16          {'StaffID': '001', 'name': 'Alice Appiah-Sarpong', 'Department': 'IT'},
17          {'StaffID': '002', 'name': 'Akoto Richard', 'Department': 'Finance'}]
18
19 # Loop through and insert each staff
20
21 for emp in staff:
22     response = table.put_item(Item=emp)
23     print(f"Inserted: {emp['name']} | Status: {response['ResponseMetadata']['HTTPStatusCode']}")
```

Fig 18. Python script to logg staff data to the dynamoDB.

4.3. After running the script “staff-logger.py” on the EC2 terminal, outputs of data insertion was displayed.

```
[ec2-user@ip-10-0-1-172 ~]$ nano staff-logger.py
[ec2-user@ip-10-0-1-172 ~]$ python3 staff-logger.py
Inserted: Alice Appiah | Status: 200
Inserted: Rita Okai | Status: 200
Inserted: Ansong Ampaw | Status: 200
Inserted: Rita Okloo | Status: 200
Inserted: Alice Mensah | Status: 200
Inserted: Rita Okloo | Status: 200
Inserted: Philip Mensah | Status: 200
Inserted: Rita Yao | Status: 200
Inserted: Alice Appiah-Sarpong | Status: 200
Inserted: Akoto Richard | Status: 200
[ec2-user@ip-10-0-1-172 ~]$
```

Fig 19. Running staff_logger.py on the EC2 instance/ application server.

4.4. By checking the DynamoDB table on the console, the data was automatically populated to the table after running the python script.

The screenshot shows the AWS DynamoDB console interface. At the top, a red box highlights the 'Table - staff' dropdown menu, with a yellow callout box labeled 'DynamoDB table name' pointing to it. Below this, a green status bar indicates 'Completed · Items returned: 10 · Items scanned: 10 · Efficiency: 100% · RCUs consumed: 10'. A yellow callout box labeled 'Items inserted after running the staff-logger.py script.' points to this status bar. The main section displays 'Table: staff - Items returned (10)' with a scan start time of 'July 04, 2025, 01:43:12'. A red box highlights the table data, which is as follows:

StaffID (String)	Department	name
001	IT	Alice Appiah
002	Finance	Rita Okai
003	IT	Ansong Ampaw
004	Security	Rita Okloo
005	IT	Alice Mensah
006	Finance	Rita Okloo
007	IT	Philip Mensah
008	Product	Rita Yao
009	IT	Alice Appiah-Sarpong
010	Finance	Akoto Richard

Fig 20. DynamoDB table

CONCLUSION

This project effectively showcased the capabilities of AWS cloud technologies in building a scalable, secure, and highly available architecture. By leveraging Auto Scaling, Elastic Load Balancing (ELB), and DynamoDB, we efficiently managed dynamic workloads and ensured low-latency data access. Security was reinforced through IAM, VPC, and CloudWatch, while adherence to AWS best practices enabled cost-efficiency, resilience, and automation.