# AWS ASG Task

**Task Objective:**

The objective of this assignment is to design, deploy, and validate a complete, production-ready AWS infrastructure environment that demonstrates core cloud engineering and DevOps capabilities. This includes creating a secure VPC architecture, deploying public and private subnets, configuring routing, launching EC2 instances, installing web services, setting up load balancing, enabling logging and monitoring, and implementing an Auto Scaling Group that dynamically adjusts capacity based on CPU utilization. The goal is to ensure high availability, scalability, observability, and secure network communication across all deployed resources, while providing proper documentation and evidence of successful implementation.

1. Create one VPC in N. Virginia region.

**Create a VPC in N. Virginia**

**Objective:**

To create a dedicated Virtual Private Cloud (VPC) in the N. Virginia (us-east-1) region that will serve as the core network environment for deploying all required AWS resources in this assignment.

**Prerequisites:**

1. Logged into the AWS Management Console as root user.
2. Region set to N. Virginia (us-east-1).
3. Basic understanding of VPC and CIDR blocks.
4. No existing VPC conflicts with the CIDR range

**Steps:**

1. Go to **VPC Console → Your VPCs → Create VPC**
2. Select **VPC only**
3. Enter:
   * **Name:** ASN01
   * **IPv4 CIDR:** 172.31.0.0/16
4. Click **Create VPC**

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

The VPC was successfully created

1. Create two subnets: one public subnet and one private subnet.

**Task Title**

Create Public and Private Subnets inside the VPC

**Objective**

To segment the VPC into a public-facing subnet and a secure private subnet.

**Prerequisites**

* VPC from Task 1
* CIDR planning
* Selected AZs: us-east-1a & us-east-1b

**Step-by-Step Implementation**

**Create Subnets**

**Public Subnet**

1. Go to **Subnets → Create Subnet**
2. Select VPC: **ASN01**
3. Add:
   * Name: public-subnet
   * AZ: us-east-1c
   * CIDR: 172.31.0.0/28
4. Click **Create Subnet**

**Private Subnet**

1. Name: private-subnet
2. AZ: us-east-1c
3. CIDR: 172.31.0.16/28
4. Create subnet

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**

Both public and private subnets were created successfully and are ready for routing and compute resources.

1. Attach an IGW to the VPC.

**Task Title**

Attach IGW to VPC

**Objective**

To provide internet access to resources deployed in the public subnet.

**Prerequisites**

* VPC created
* No existing IGW attached to the VPC

**Step-by-Step Implementation**

**Attach Internet Gateway (IGW)**

1. Go to **Internet Gateways → Create IGW**
2. Name: IgwASN01
3. Attach to **ASN01**

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

The Internet Gateway was attached successfully, enabling external communication for public subnet resources.

1. Create one public route table (RT) and one private route table.

**Task Title**

Create Route Tables for Public and Private Subnets

**Objective**

To configure routing so the public subnet uses the IGW and private subnet uses NAT gateway (later task).

**Prerequisites**

* VPC created
* IGW created and attached
* Subnets ready

**Step-by-Step Implementation**

**Create Route Tables**

**Public Route Table**

1. Go to **Route Tables → Create Route Table**
2. Name: publicrt01
3. VPC: **ASN01**
4. Edit **Routes → Add Route**
   * Destination: 0.0.0.0/0
   * Target: **IgwASN01**
5. Edit **Subnet associations → select public-subnet**

A screenshot of a computer

AI-generated content may be incorrect.

**Private Route Table**

1. Name: privatert01
2. VPC: **AGN01**  
   (We will add NAT route later)

A screenshot of a computer

AI-generated content may be incorrect.

1. Deploy a NAT gateway in the public subnet and attach the NAT gateway to the private subnet.

**Task Title**

Deploy NAT Gateway for Private Subnet Internet Access

**Objective**

To enable outbound internet access from private EC2 instances using a NAT Gateway placed in the public subnet**.**

**Prerequisites**

* VPC created
* Public subnet created
* Elastic IP available
* Private route table created
* IGW already attached

**Step-by-Step Implementation**

**Deploy NAT Gateway in Public Subnet**

1. Go to **NAT Gateways → Create NAT Gateway**
2. Subnet: **public-subnet**
3. Allocate new Elastic IP
4. Create

**Attach NAT Gateway to Private Route Table**

Edit **private-rt → Routes → Add route**

* Destination: 0.0.0.0/0
* Target: **NAT Gateway**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

The NAT Gateway was successfully deployed and attached to the private route table, enabling secure outbound internet access for private EC2 instances.

1. Create two instances, one in the public subnet and one in the private subnet.

**Task Title**

Launch Public and Private EC2 Instances

**Objective**

To create a public-facing EC2 instance and a private EC2 instance for backend operations.

**Prerequisites**

* Public & private subnets
* Key pair available
* Security groups created
* NAT gateway running (for private EC2)
* AMI: Amazon Linux 2

**Step-by-Step Implementation**

**Create Two EC2 Instances**

**Public EC2 Instance**

1. Launch EC2 → Name: Ec2Public01
2. Subnet: **public-subnet**
3. Auto-assign public IP: **Enable**
4. Security group allow:
   * SSH (22) My IP
   * HTTP (80) Anywhere
5. Create instance

**Private EC2 Instance**

1. Launch EC2 → Name: Ec2Private01
2. Subnet: **private-subnet**
3. Auto-assign public IP: **Disable**
4. SG allow inbound port 80 from LB SG
5. Create instance

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

Public and private EC2 instances were launched successfully with correct network configurations and SSH access.

1. Deploy Apache server on both EC2 instances with a sample index.html file.

**Task Title**

Install Apache Web Server on Both EC2 Instances

**Objective**

To deploy an Apache HTTP server with a simple web page on both EC2 instances.

**Prerequisites**

* EC2 instances running
* SSH access established
* YUM repository access (NAT required for private EC2)
* Security group allowing port 80

sudo yum update -y

sudo yum install httpd -y

sudo systemctl start httpd

sudo systemctl enable httpd

/var/www/html/index.html

A screen shot of a computer

AI-generated content may be incorrect.

* Once installed start and enable service.

A black screen with a black background

AI-generated content may be incorrect.

* Once everything is installed and running Use this command “cd /var/www/index.html/” to change directory and ls. We can see index.html below.
* Vi index.html (I have added shoppiex site index.html code here)

A screen shot of a computer screen

AI-generated content may be incorrect.

* Copy ec2 public01 public Id to browser search for it by adding port number 80 at the end.
* As we can below that website is visible.

A screenshot of a computer

AI-generated content may be incorrect.

* To connect to private ec 2 we use jump server and using private key which is in my local machine.
* Below is my pem key

A screenshot of a computer screen

AI-generated content may be incorrect.

* Now using vi created test.pem and pasted pem key here which we coped in before step.
* Gave permissions to it
* Using ssh -i test.pem ec2-user@ private Ip adress here
* Now we are sucessfully connected to private ec2.

A screenshot of a computer

AI-generated content may be incorrect.

sudo yum update -y

sudo yum install httpd -y

sudo systemctl start httpd

sudo systemctl enable httpd

/var/www/html/index.html

A black screen with white text

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

* After in installing httpd and editing index.html. to check the connectivity, we use curl. We are doing this because it is private ec2
* Use command” curl http://privateIP address: add port number at the end of Ip”
* As we can see below output it is connected.

A screen shot of a computer

AI-generated content may be incorrect.

**Conclusion**

Apache was installed and configured successfully on both EC2 instances with the sample webpage working correctly.

1. Create one application load balancer and attach it to both EC2 instances.

**Task Title**

Deploy Application Load Balancer and Register EC2 Targets

**Objective**

To distribute incoming HTTP traffic evenly between public and private EC2 instances.

**Prerequisites**

* Public subnets in at least two AZs
* EC2 instances running Apache
* Security group for ALB (allow 80)
* Target group ready (HTTP)

**Step-by-Step Implementation**

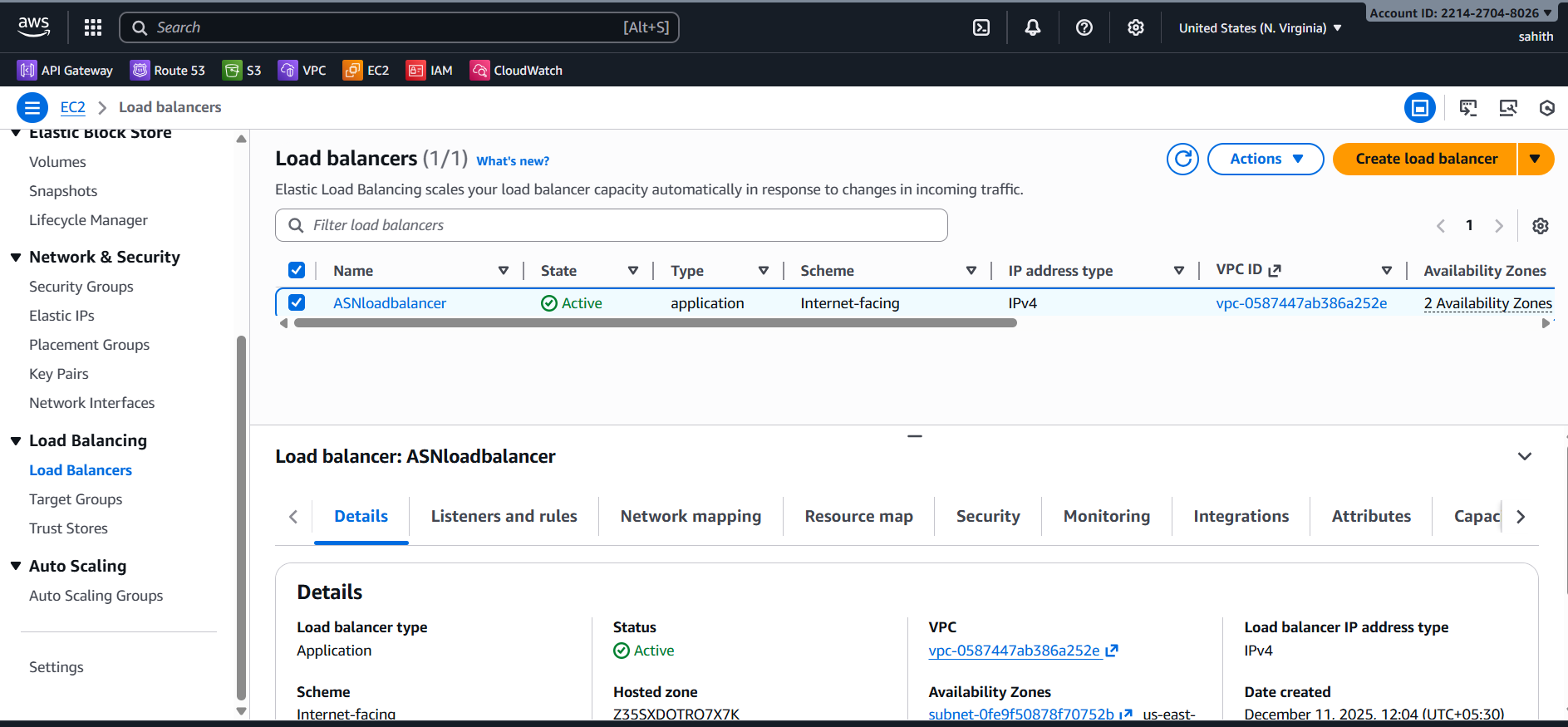
**Create Application Load Balancer (ALB)**

1. Go to **EC2 → Load Balancers → Create Load Balancer**
2. Select **Application Load Balancer**
3. Name: **ASN**
4. Scheme: Internet-facing
5. Listener: HTTP (80)
6. Select VPC: **ASN01**
7. Select subnets: **public-subnet + private subnet**
8. Create Target Group:
   * Type: Instance
   * Health check: /
9. Register both **public** and **private** EC2 instances
10. Create ALB

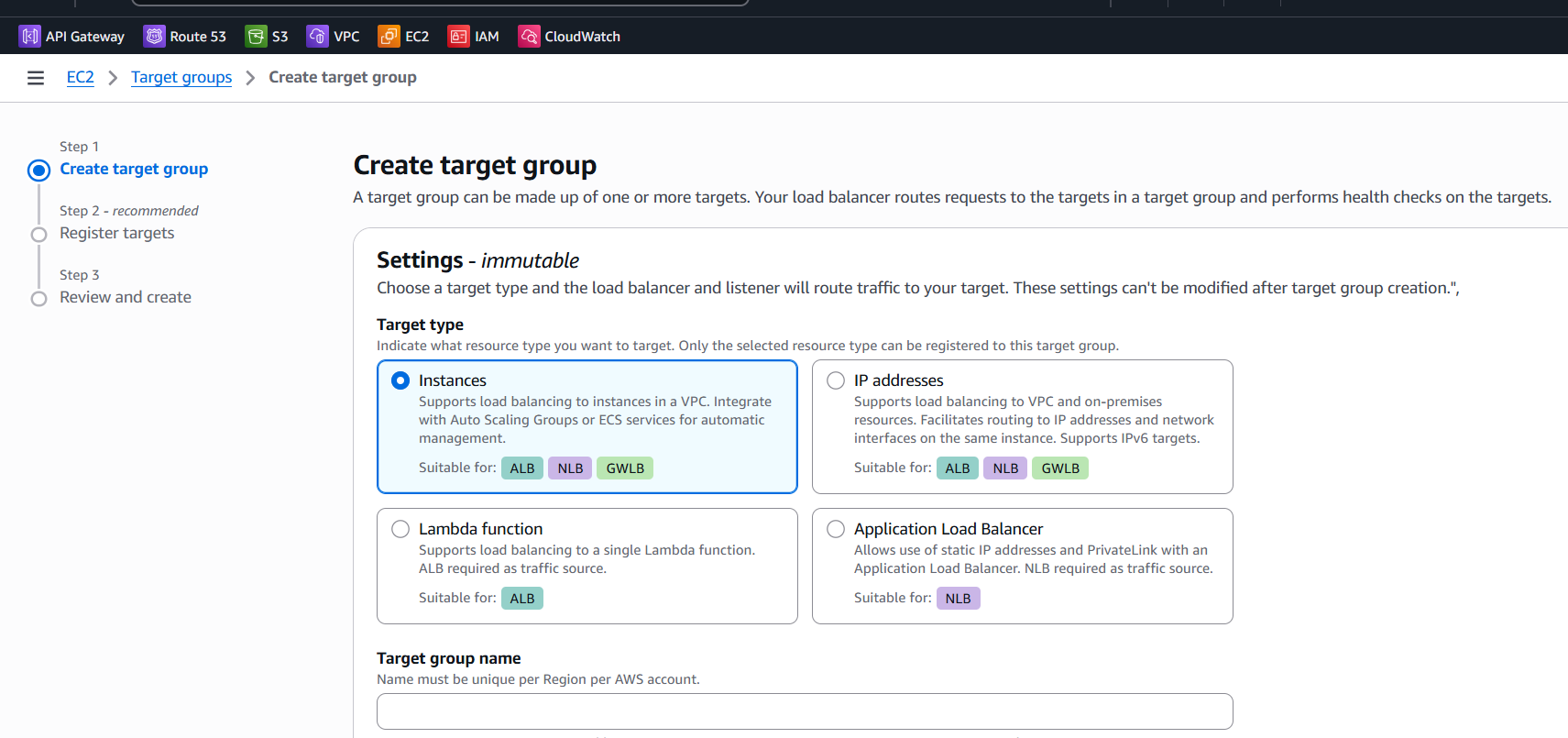
A screenshot of a computer

AI-generated content may be incorrect.

* Application load balancer is created successfully



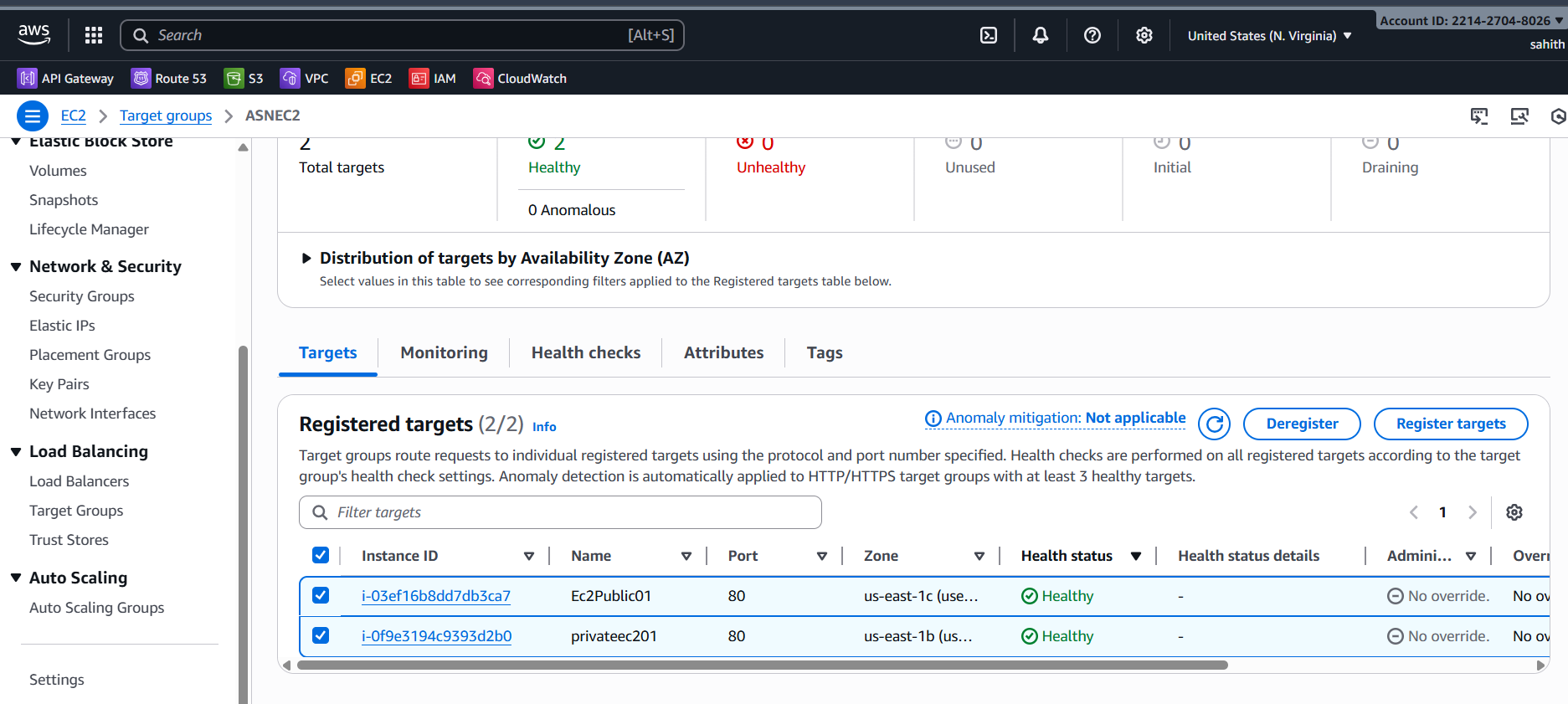
* Creating a target group to which I need to attach load balancer



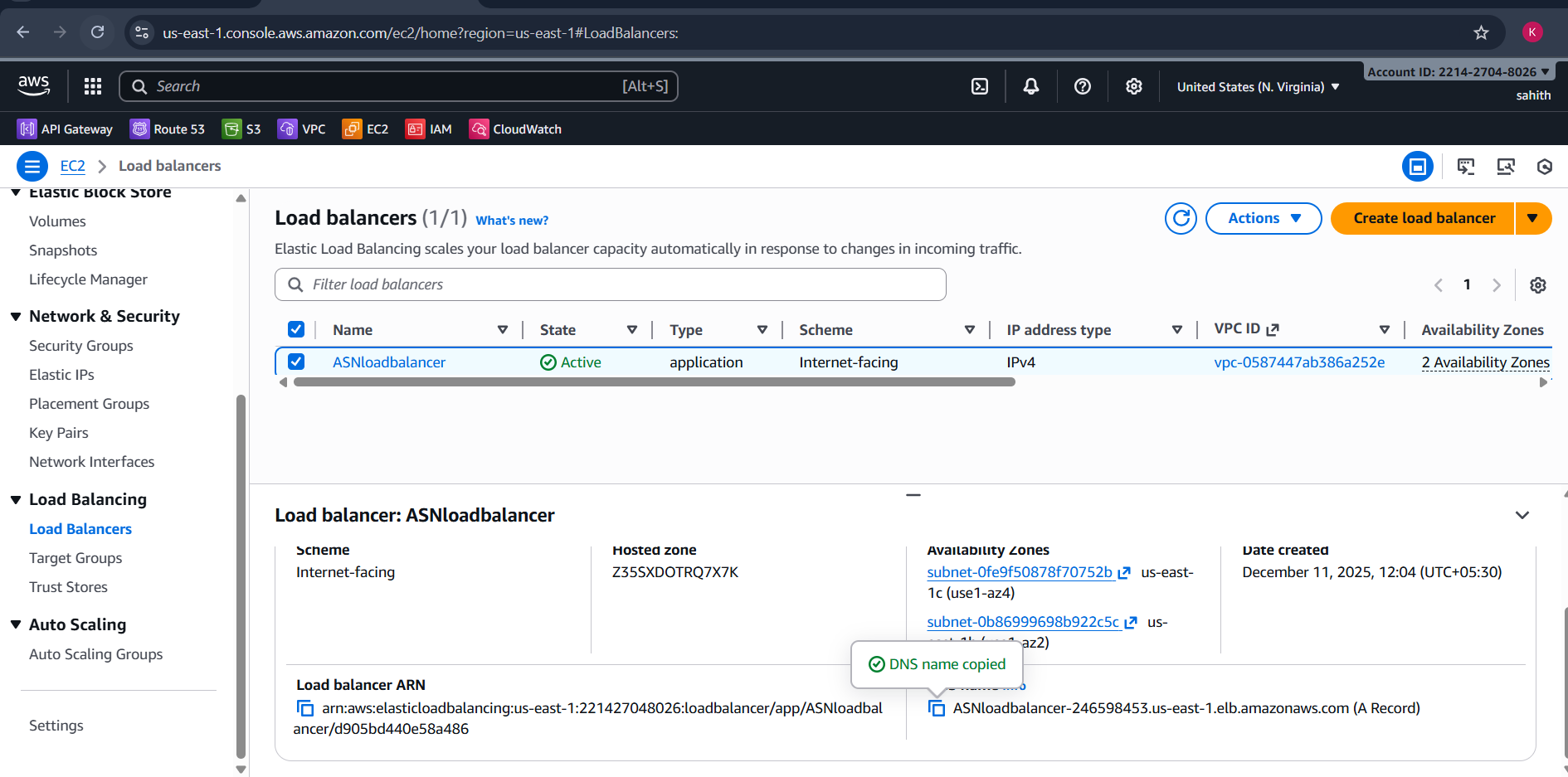
A screenshot of a computer

AI-generated content may be incorrect.

* As we can see I have attached two instances to target groups and both are running successfully
* We will know it in health status if shows healthy it is running or it will show error

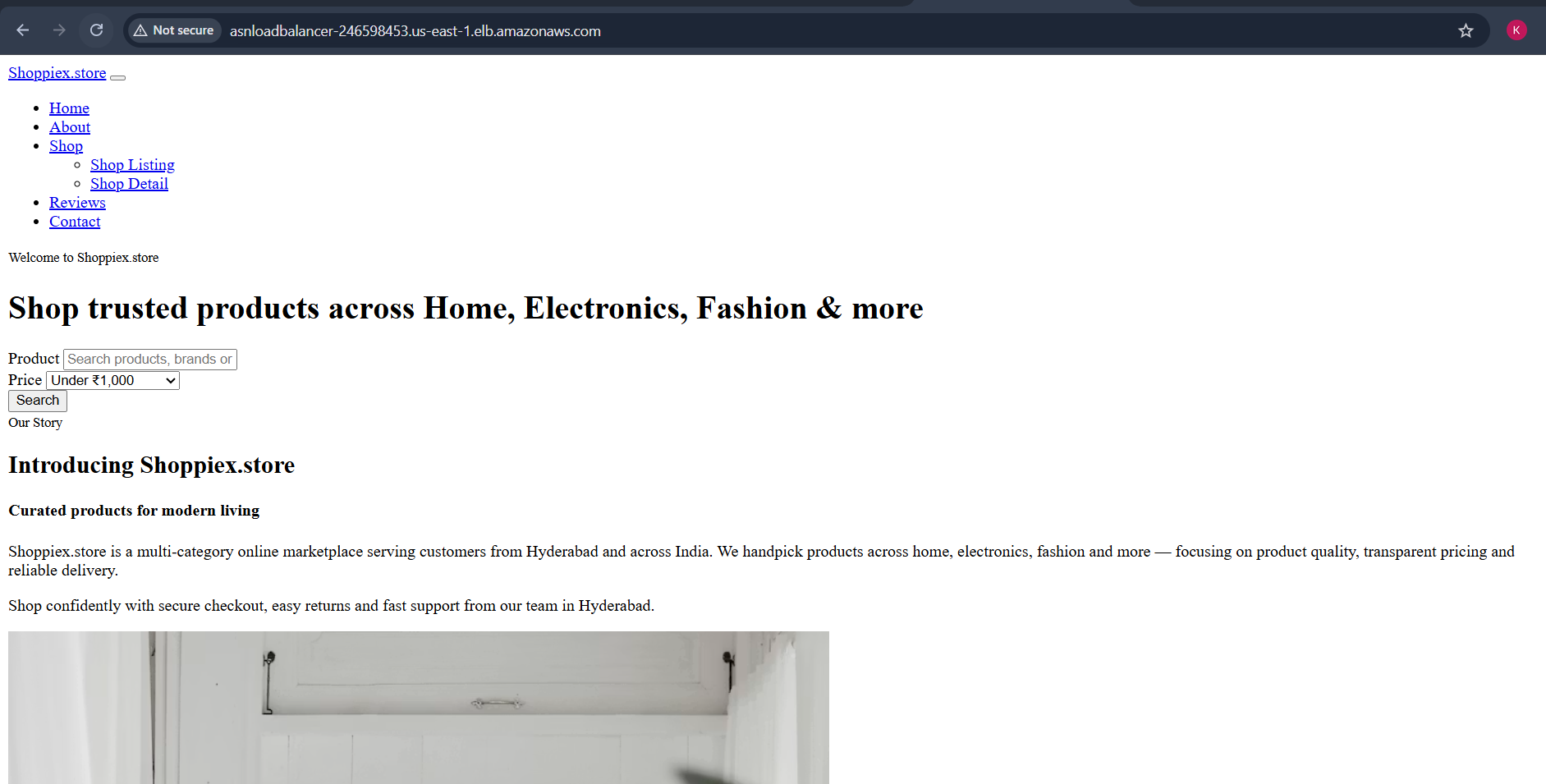


* Now redirect to load balancer > details > copy ASN load balancer from down below
* Using that URL go to browser and run it.



* As we can see it showing private ec2 index.html if we refresh page after few seconds, it will show publicec2 index.html file.





**Issues Faced**

• If target health remains **unhealthy**:

* Confirm security group allows ALB health-check source (the ALB uses its own IPs — best to allow 0.0.0.0/0 for HTTP in test environments or the ALB SG’s ID in stricter setups).
* Ensure the web server responds on the health-check path (default /) and port (80).
* Check instance OS firewall (iptables/ufw) blocking port 80. • If ALB returns 503 or site unreachable:
* Verify target group has at least one healthy target in each AZ required.
* Confirm routing and subnets are correct (ALB subnets must be public subnets with route to IGW).
* While creating load balancer Aws account issues

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

ALB was created and configured with a target group pointing to two EC2 instances (HTTP:80). Targets reported Healthy and application pages are reachable via the ALB DNS name. Task completed successfully as validated by ALB target health and browser verification.

1. Store application load balancer logs in S3.

**Task Title**

Enable ALB Access Logging to S3 Bucket

**Objective**

To enable access logging on the Application Load Balancer and store log files in an S3 bucket for monitoring and debugging purposes.

**Prerequisites**

* Application Load Balancer created (Task 8)
* S3 bucket created for log storage
* ALB must be in active state
* Bucket policy allowing ALB to write logs
* Proper IAM permissions (root user OK)

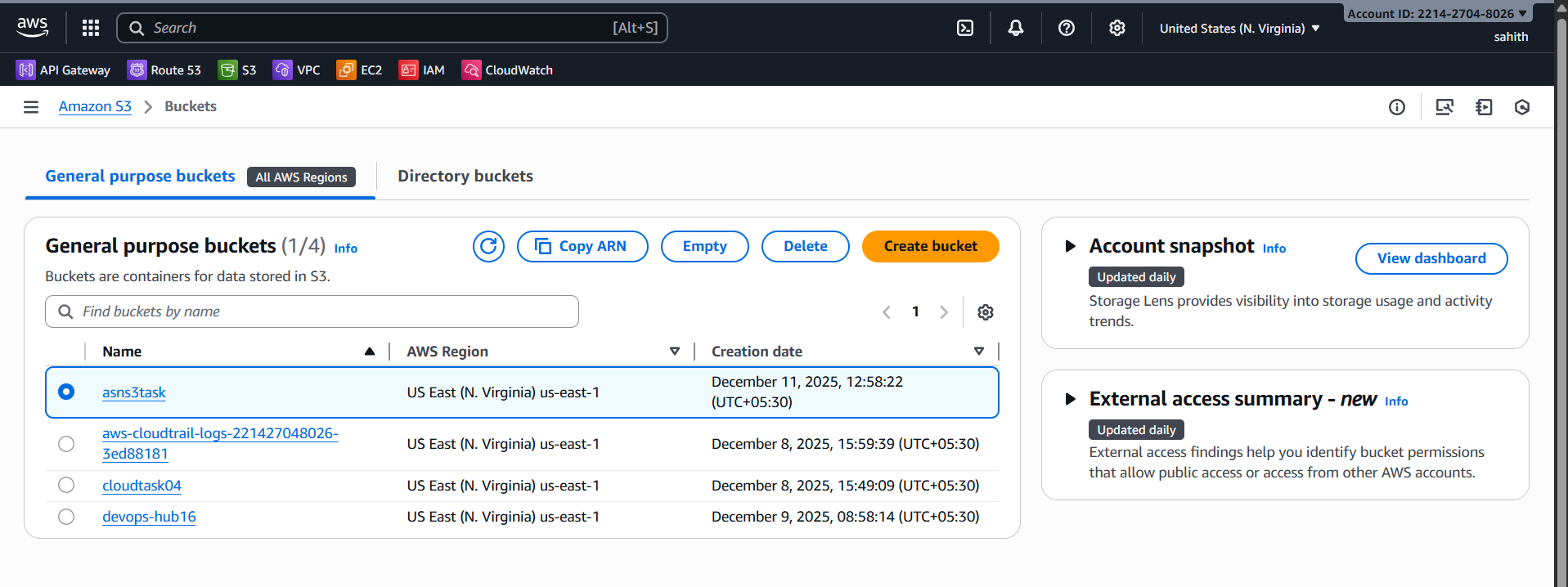
**Step-by-Step Implementation**

**Store ALB Logs in S3 Bucket**

**Create S3 Bucket**

1. Name: asns3task
2. Block public access: enabled
3. Go to **ALB → Attributes → Access Logs → Enable**
4. Select S3 bucket
5. Save

* Step1: created a s3 bucket



* Select the created s3 bucket > permissions > edit bucket policy > edit Json format script here by giving our aws account Id and s3 bucket name and click on save it.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowALBLogs",

"Effect": "Allow",

"Principal": {

"Service": "logdelivery.elasticloadbalancing.amazonaws.com"

},

"Action": "s3:PutObject",

"Resource": "arn:aws:s3:::asns3task/AWSLogs/221427048026/\*",

"Condition": {

"StringEquals": {

"s3:x-amz-acl": "bucket-owner-full-control"

}

}

},

{

"Sid": "AllowALBLogBucketAccess",

"Effect": "Allow",

"Principal": {

"Service": "logdelivery.elasticloadbalancing.amazonaws.com"

},

"Action": "s3:GetBucketAcl",

"Resource": "arn:aws:s3:::asns3task"

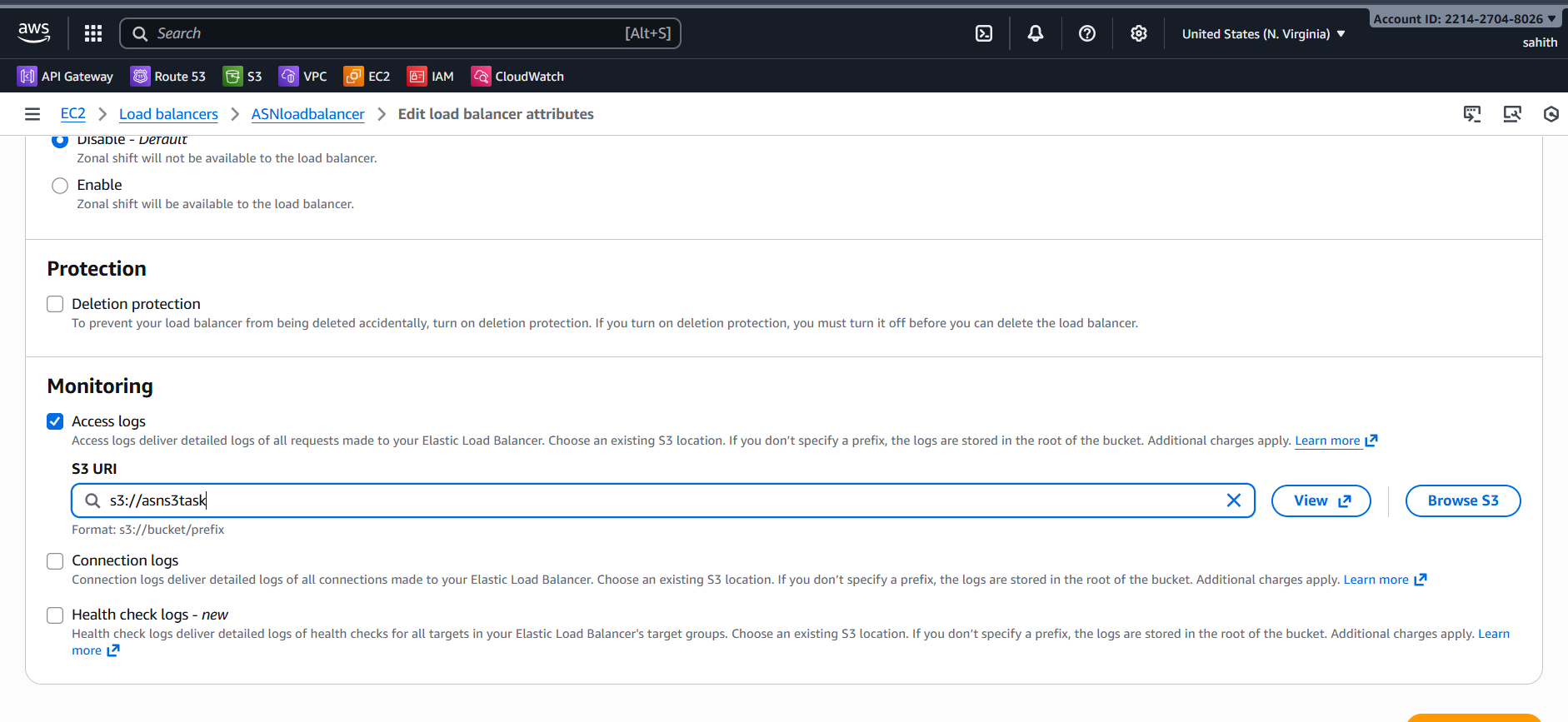
}

]

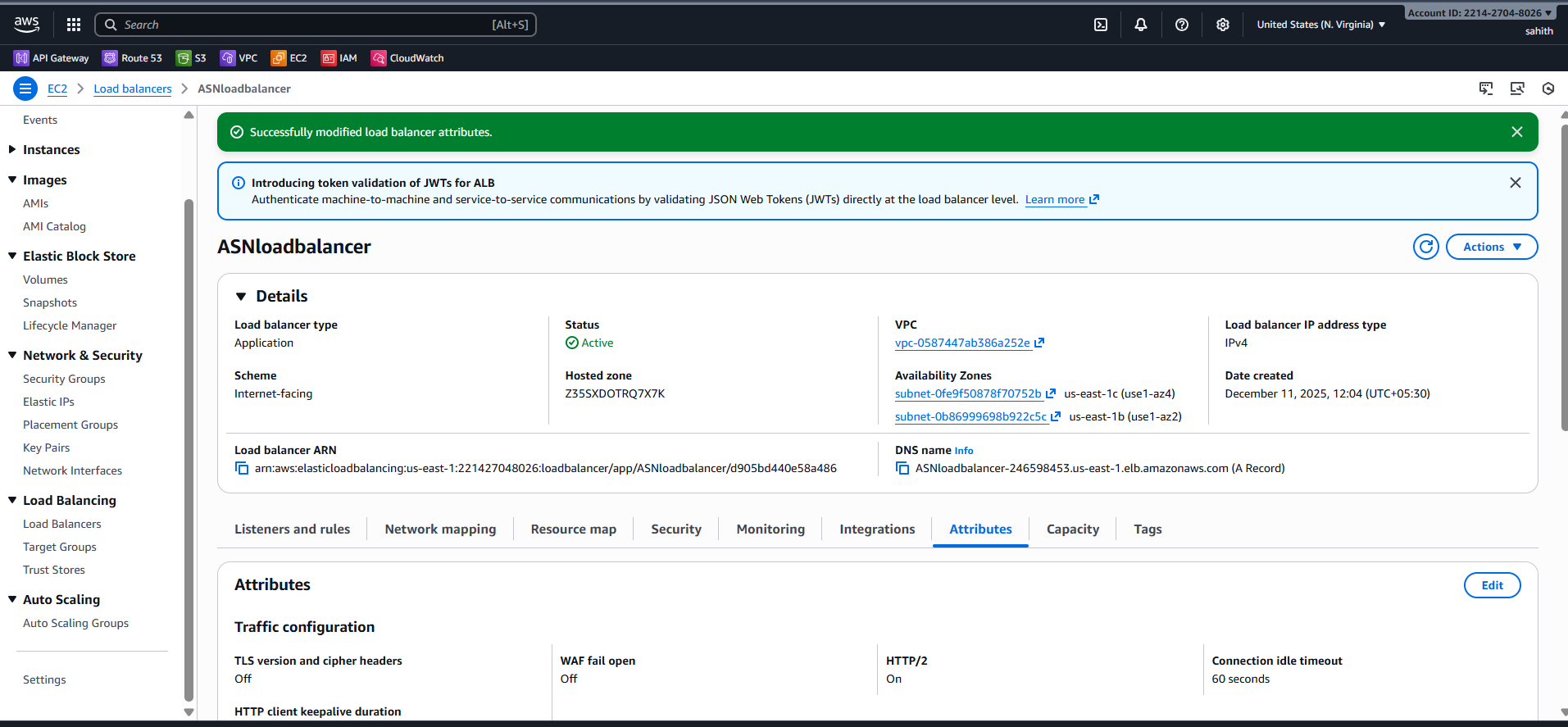
}



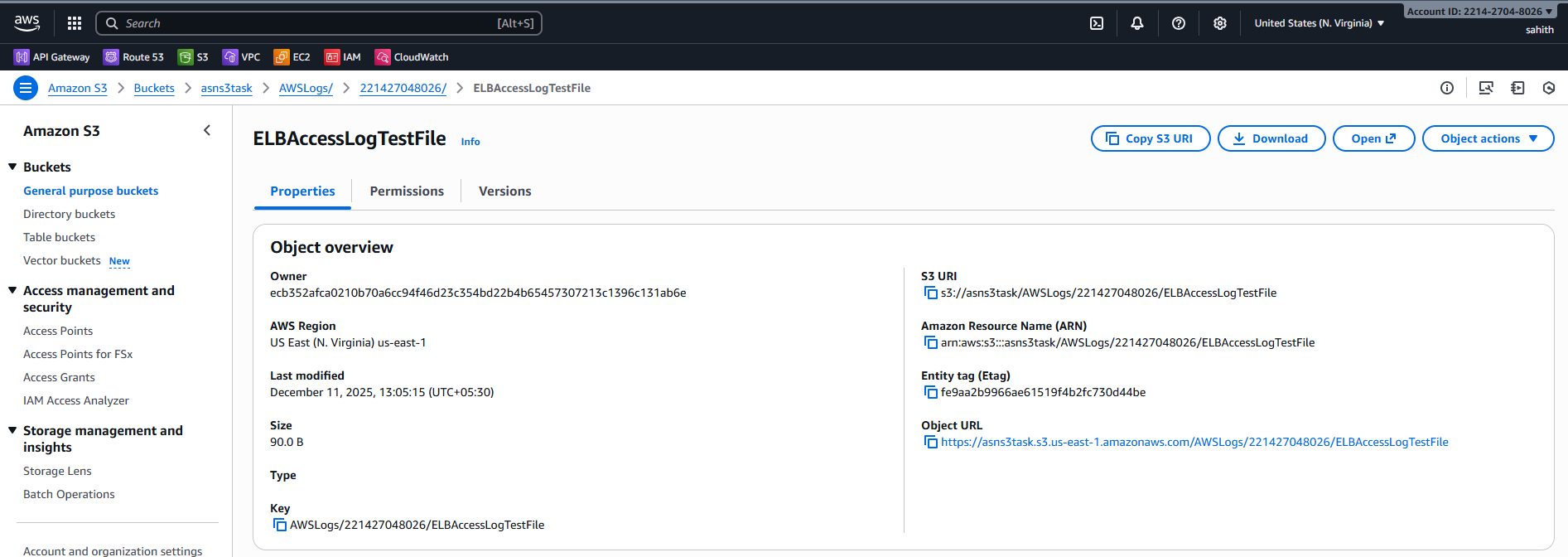
* Now redirect to load balancer > select created load balancer > click on edit attributes
* In monitoring > click on access logs > using browser s3 select s3 bucket that we created for logs in before step > click on create.
* Successfully attached s3 bucket to load balancer to store logs



* Successfully attached s3 bucket to load balancer to store logs



* As we see in below that logs in s3 bucket



**Conclusion:**

ALB logging is functioning, ensuring traceability and audit capability.

1. Store the VPC flow logs in a CloudWatch log group.

**Task Title**

Enable VPC Flow Logs and Store in CloudWatch

**Objective**

To capture VPC-level network traffic logs and store them in a CloudWatch Logs group for monitoring, auditing, and troubleshooting.

**Prerequisites**

* VPC created (Task 1)
* IAM role for VPC flow logs (auto-created)
* CloudWatch permissions available
* CloudWatch Log Group created or permission to auto-create
* No existing flow log for the same VPC

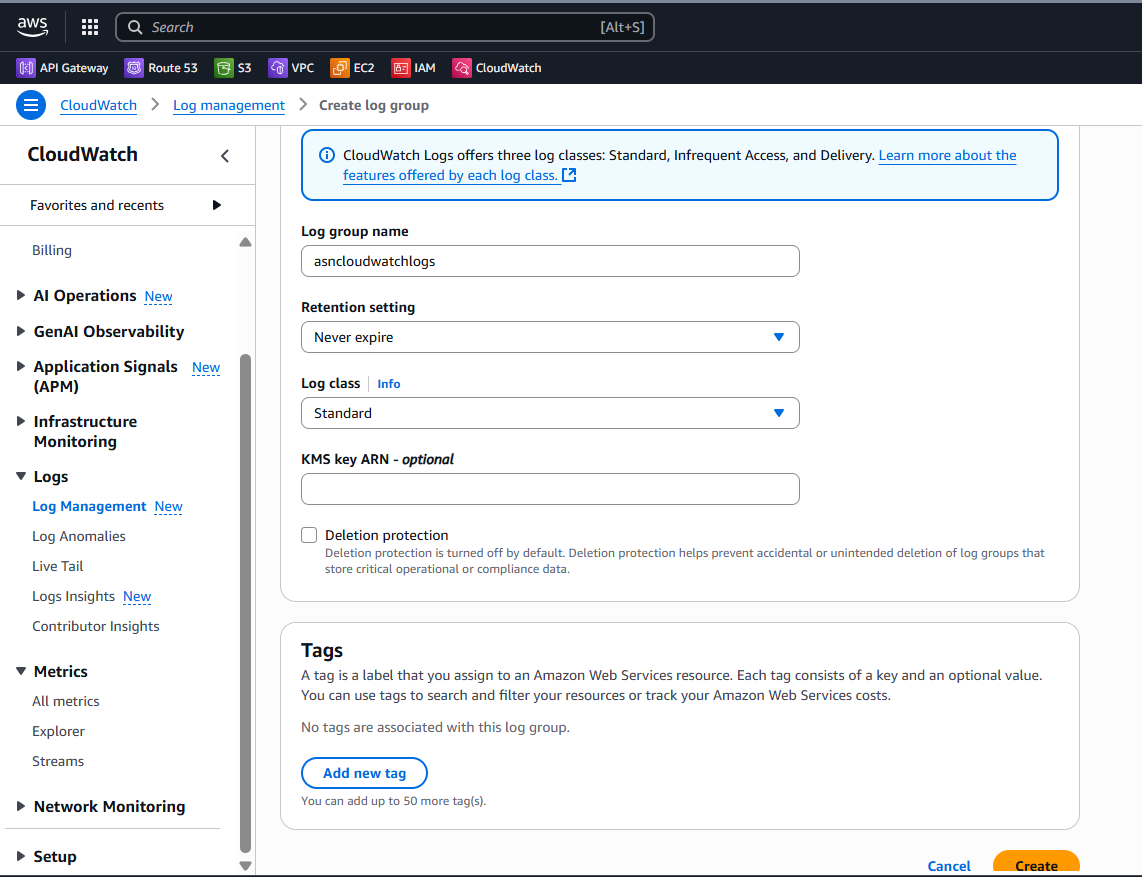
**Step-by-Step Implementation**

**Enable VPC Flow Logs to CloudWatch**

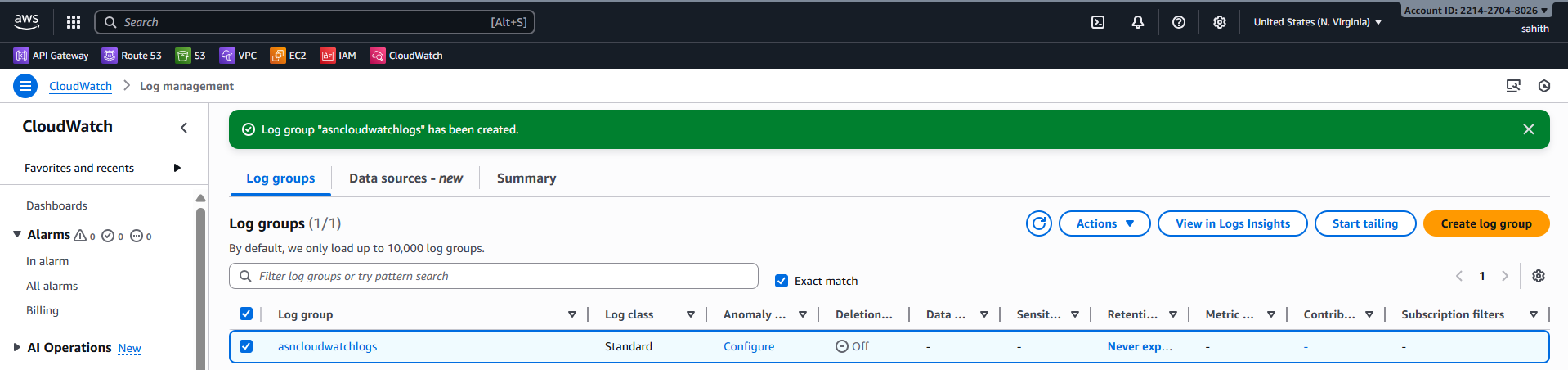
**🔹 Steps Performed**

1. VPC → Flow Logs → Create
2. Filter: ALL
3. Destination: CloudWatch Logs
4. Log group: /vpc/flow-logs

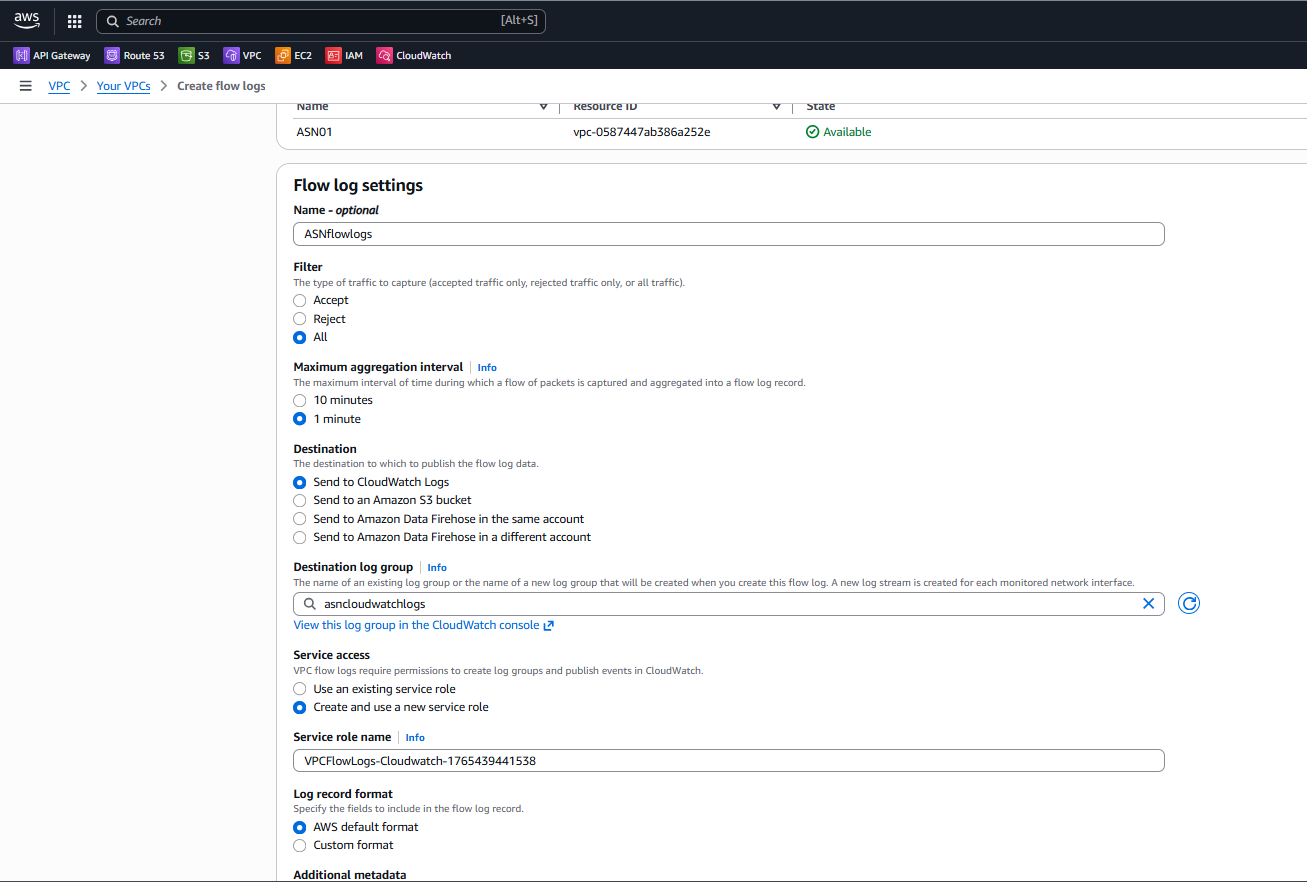
* First create CloudWatch > log management > create



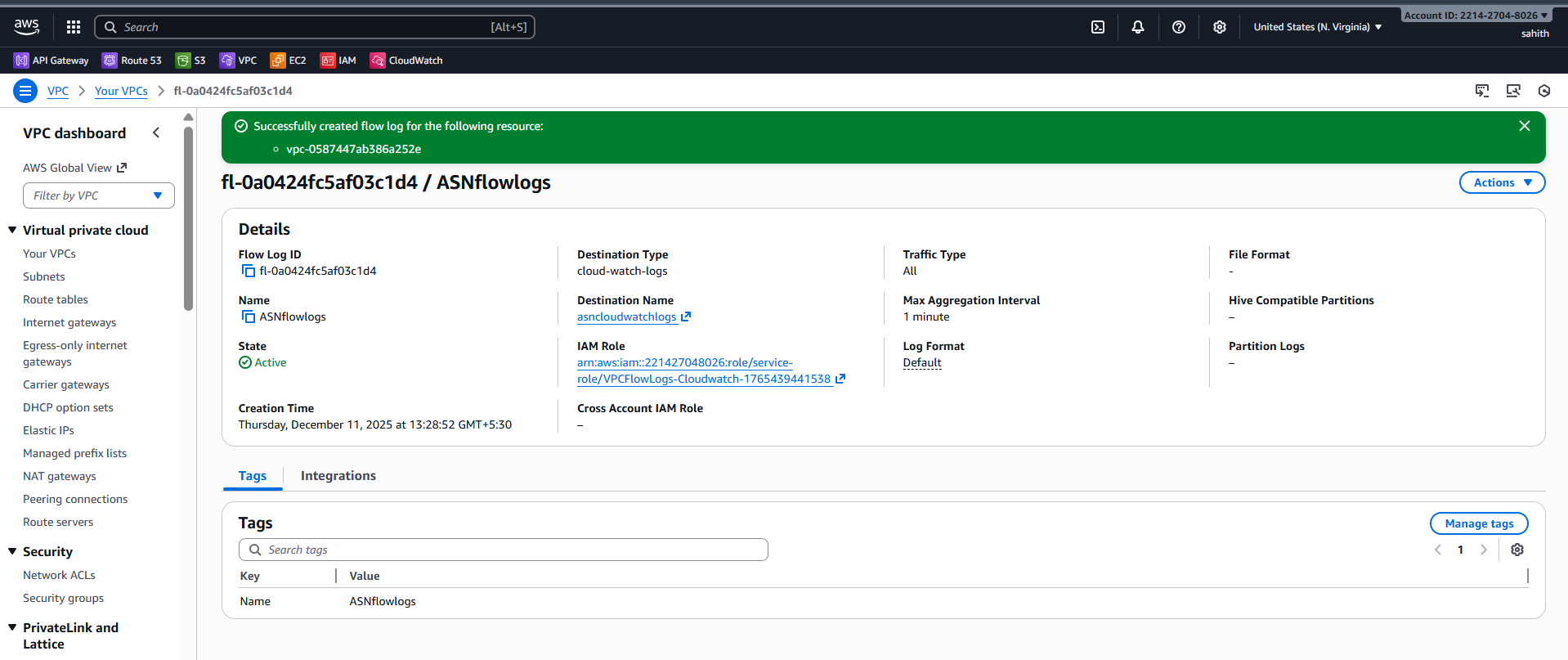
* As we can see log group has been created successfully



* Create vpc flow logs > filter: All > maximum: 1mintue > send to cloud watch > destination: cloud watch > create

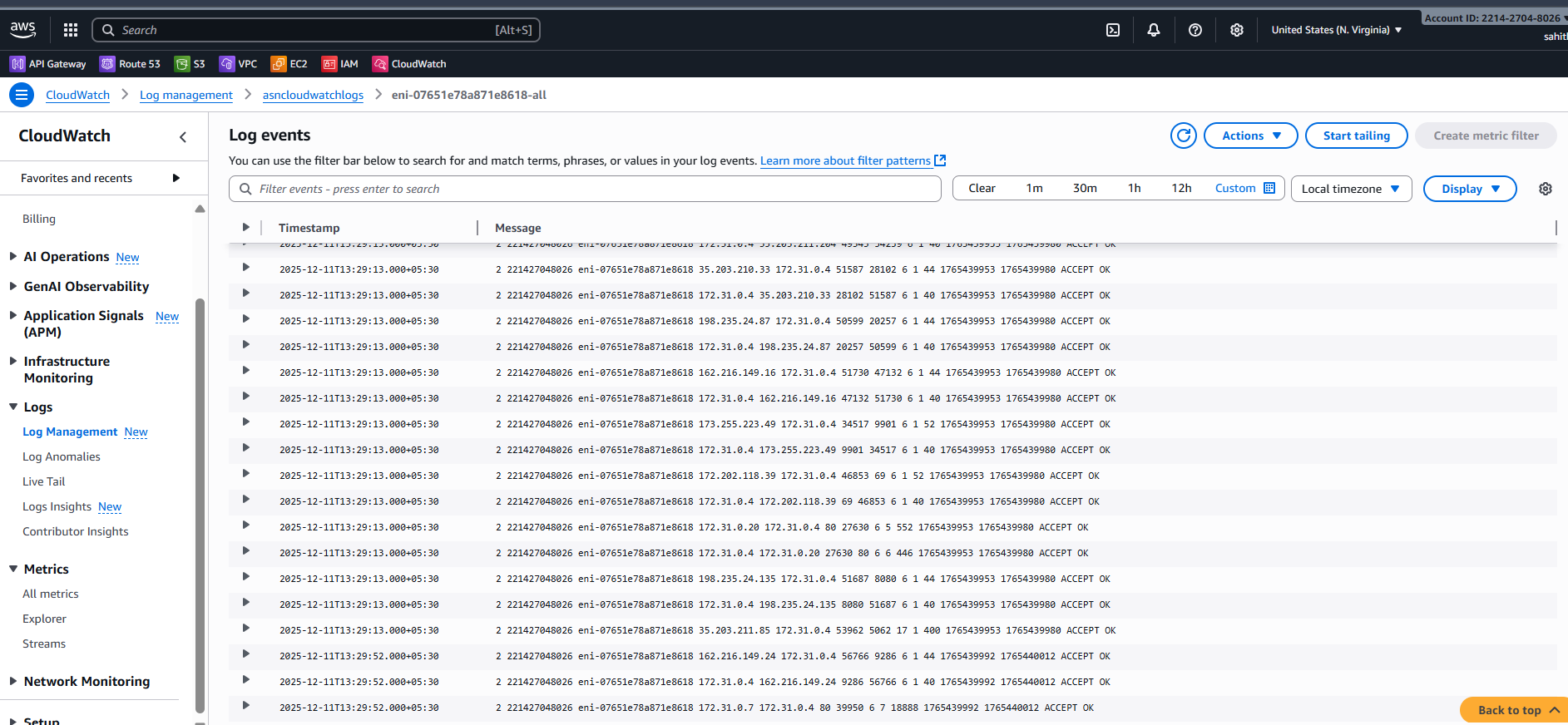


* Vpc flow logs has been created successfully



**Validation:**

* Go to cloud watch > logs management > click logs
* We can see logs in below screenshot



**Objective:**

To enable VPC Flow Logs for capturing detailed network traffic information at the VPC level and store these logs in a CloudWatch Log Group. This helps in monitoring, troubleshooting connectivity issues, analysing traffic patterns, and enhancing network security visibility.

1. Create monitoring dashboards to monitor CPU utilization and to monitor the Apache service.

**Task Title**

Create CloudWatch Dashboards for EC2 CPU & Apache Monitoring

**Objective**

To create a CloudWatch dashboard displaying EC2 CPU utilization and track the Apache web server’s health and service activity.

**Prerequisites**

* EC2 instances running
* Apache installed (Task 7)
* CloudWatch Agent installed
* CloudWatch Logs enabled

**Step-by-Step Implementation**

**Steps Performed**

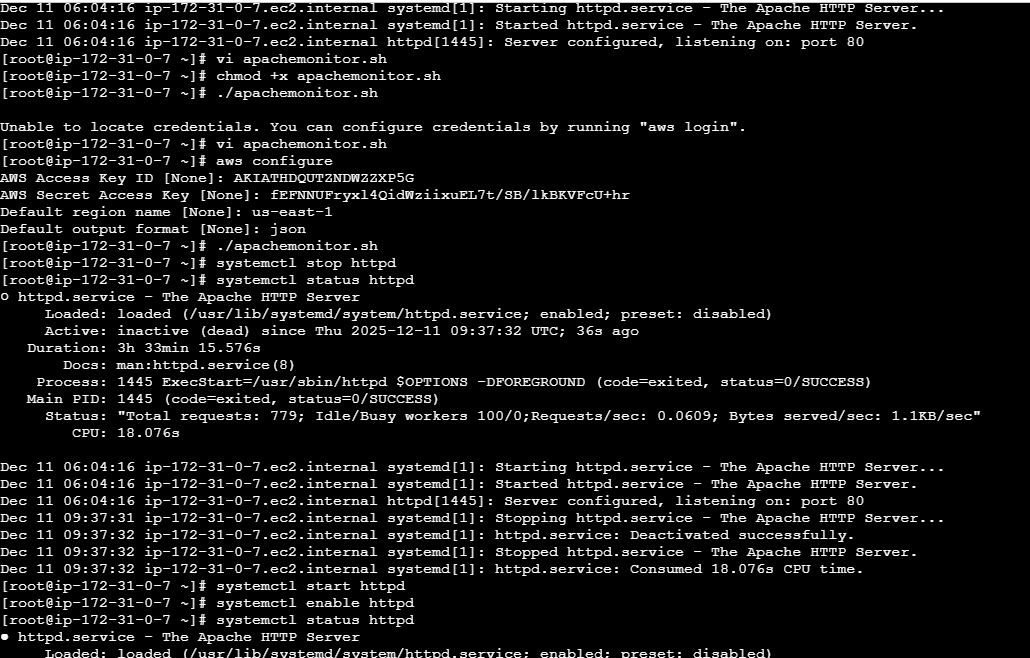
1. Go to **Dashboards → Create Dashboard**
2. Add CPU widget
3. Install CloudWatch agent
4. Add Apache log/service monitoring

* **Script:**

#!/bin/bash

REGION="us-east-1"  
NAMESPACE="ApacheService"  
METRIC\_NAME="httpd\_status"  
SERVICE\_NAME="httpd"  
AWS\_CLI="/usr/bin/aws"# Fetch Instance ID (IMDSv2)  
TOKEN=$(curl -s -X PUT "<http://169.254.169.254/latest/api/token>" \  
  -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")INSTANCE\_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \  
  <http://169.254.169.254/latest/meta-data/instance-id>)# Check Apache Status  
if systemctl is-active --quiet "$SERVICE\_NAME"; then  
  STATUS=1  
else  
  STATUS=0  
  systemctl start httpd     
fi# Push Metric to CloudWatch  
$AWS\_CLI cloudwatch put-metric-data \  
  --region "$REGION" \  
  --namespace "$NAMESPACE" \  
  --metric-name "$METRIC\_NAME" \  
  --value "$STATUS" \  
  --dimensions Name=InstanceId,Value="$INSTANCE\_ID"

* As we already installed and running the httpd
* Now, using vi apachemonitor.sh enter above script in it and save it
* Gave permissions to file and executed it. But, it through an error saying aws login.
* Using access key and secret key logon to aws and now I executed here



* Installed cronjobs here to trigger alert
* Start, enable and status cronjob
* Using command: crontab -e: insert this in vi “\* \* \* \* \* apachemonitor.sh >> /tmp/httpd.log 2>&1”

A black screen with white text

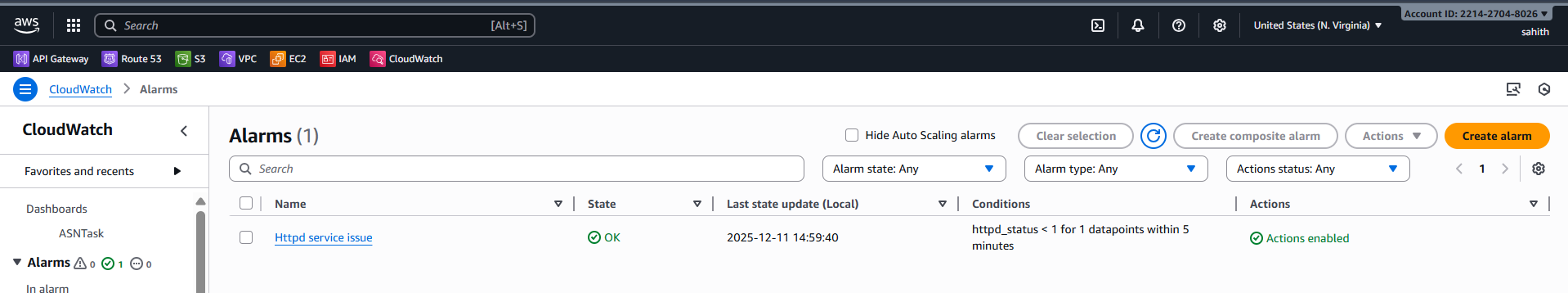
AI-generated content may be incorrect.

* Created a dashboard for both httpd status and ec2 CPU utilization

A screenshot of a computer

AI-generated content may be incorrect.

* Created an alarm for httpd if fails to run and one alarm for CPU utilization if it exceeds threshold.



**Conclusion:**

The CloudWatch dashboard was successfully created and now provides real-time visibility into EC2 CPU utilization and the Apache service status. With these widgets, system performance and web server health can be monitored continuously, enabling faster detection of issues and improved operational efficiency. The dashboard is fully functional and displays accurate metrics from both EC2 instances.

1. If CPU utilization is more than 70%, then it should trigger auto scaling and launch new instance.

**Task Title**

Create Auto Scaling Policy Based on CPU Utilization Threshold

**Objective**

To automatically scale the EC2 environment by launching additional instances when CPU exceeds 70%.

**Prerequisites**

* Launch Template created
* Apache user-data configured
* Target group attached to ALB
* CloudWatch metrics available
* IAM service role for Auto Scaling

**Step-by-Step Implementation**

**Auto Scaling When CPU > 70%**

**Create Launch Template**

**Create Auto Scaling Group (ASG)**

**Steps**

1. Create ASG
2. Attach Launch Template
3. Attach ALB Target Group
4. Scaling Policy:
   * CPU > 70% → Add capacity

A screenshot of a computer

AI-generated content may be incorrect.

* Created new launch template

A screenshot of a computer

AI-generated content may be incorrect.

* Now, creating Auto scaling group

A screenshot of a computer

AI-generated content may be incorrect.

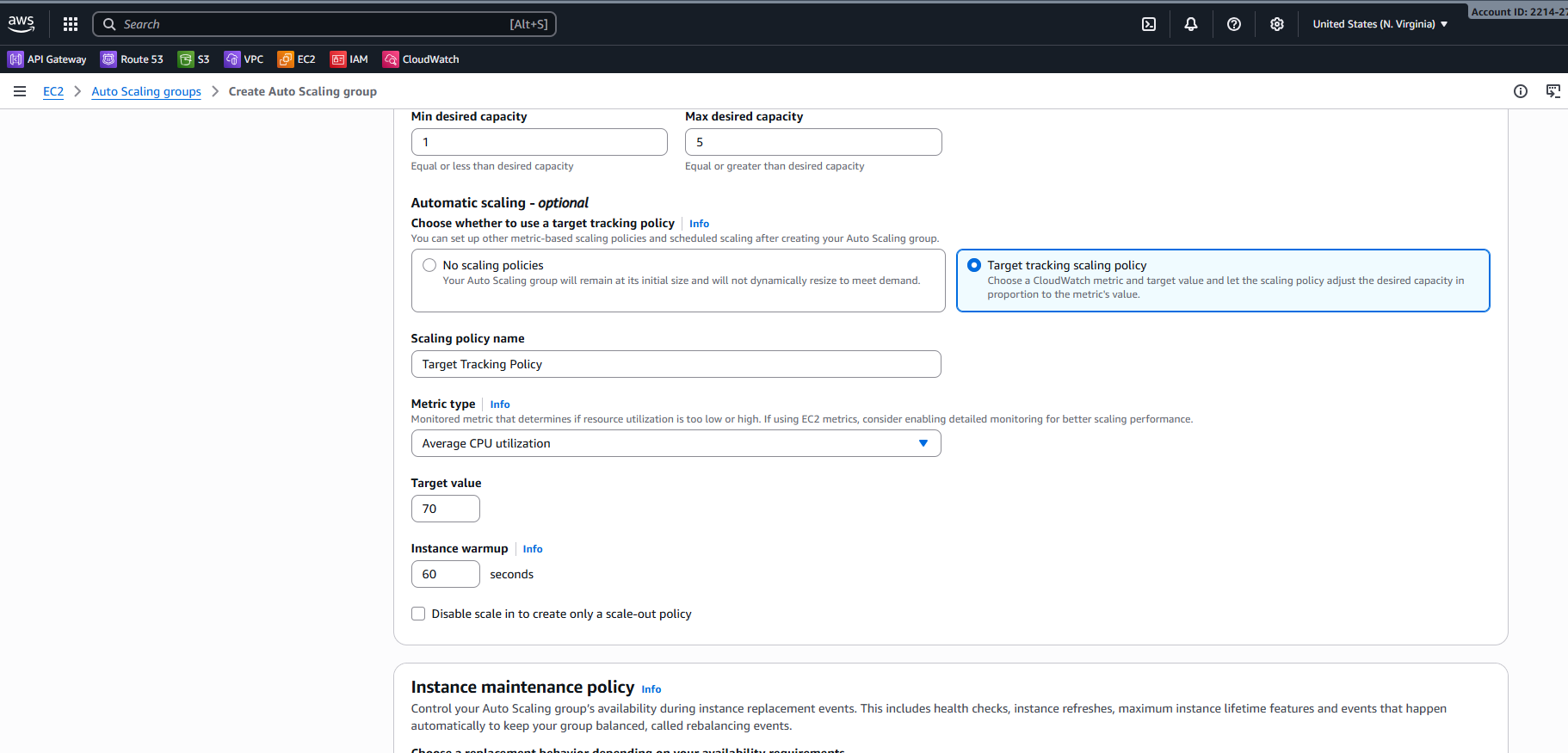
* Select vpc and availability zones and subnets > availability zone distribution: balanced best effort

A screenshot of a computer

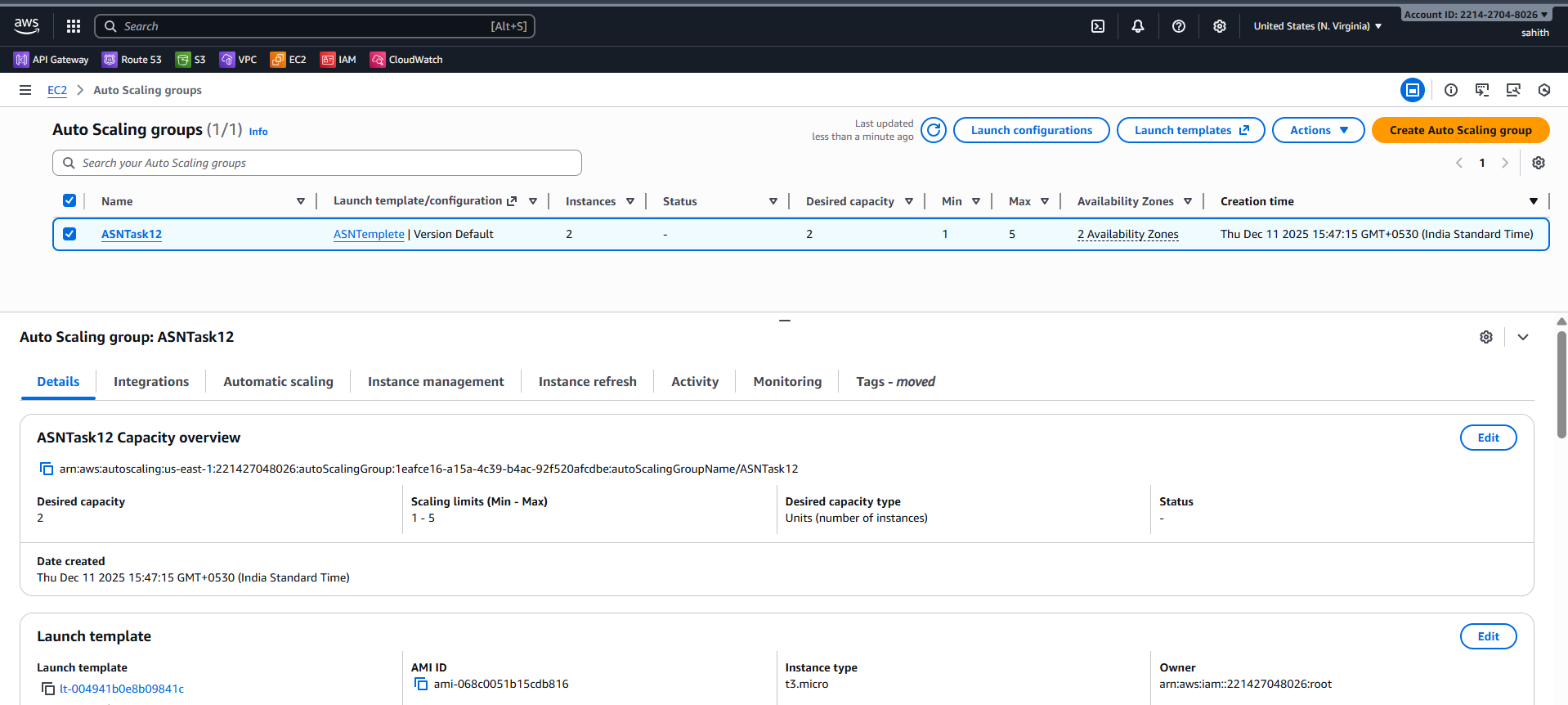
AI-generated content may be incorrect.

select

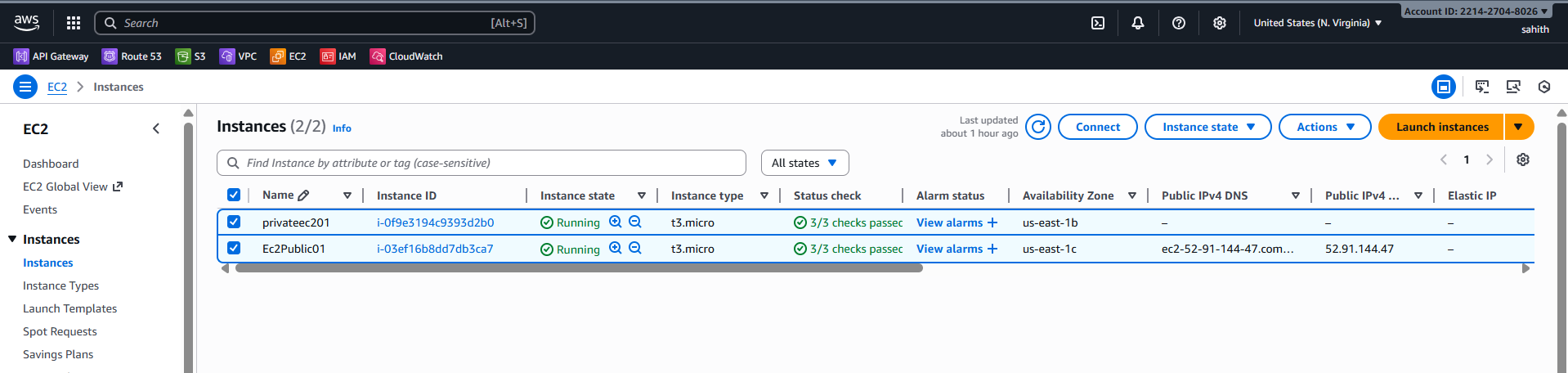
* minimum: 1
* Desired: 2
* Maximum: 5
* Automatic scaling: target tracking scaling policy > Target value: 70 > warmup: 60 seconds



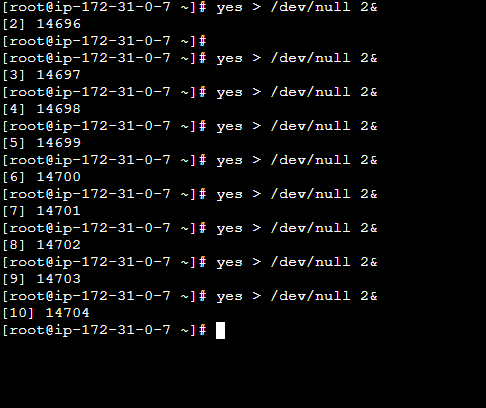
* Auto scaling has been created successfully and attached



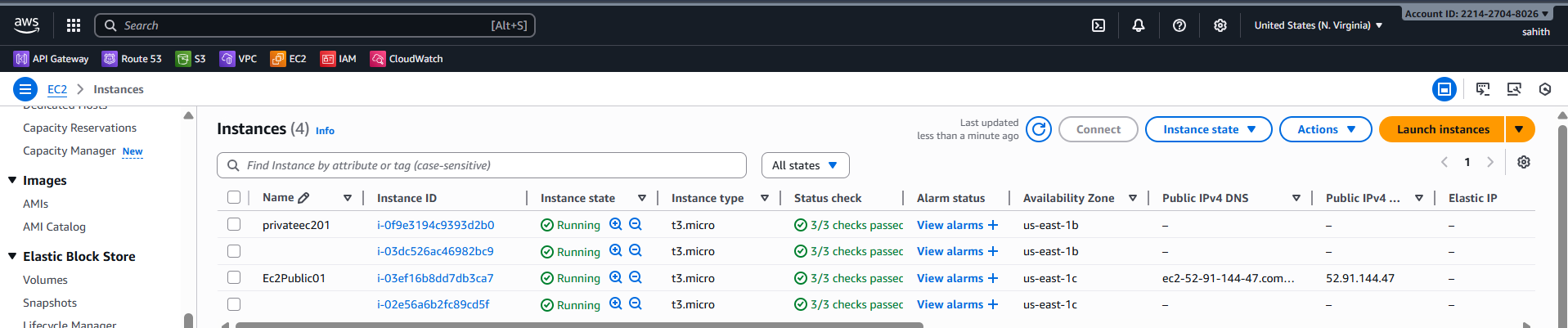
* **Validation:**
* As we can first, I had two ec2 instances as I increased load on CPU it created few new instances as we can below.



* Created fake load on CPU



* As we can see below ASG is creating new instances



**Conclusion:**

The Auto Scaling Group (ASG) was successfully configured with a CPU-based scaling policy that triggers when utilization exceeds 70%. When the threshold is reached, new EC2 instances are automatically launched using the defined launch template, ensuring increased application capacity and maintaining performance during high workloads. This setup enables seamless scalability, high availability, and improved fault tolerance for the application environment.