**ASG**

1. **Create one VPC in N. Virginia region.**

**Task Title**

Creation of a Virtual Private Cloud (VPC) in AWS

## ****Objective****

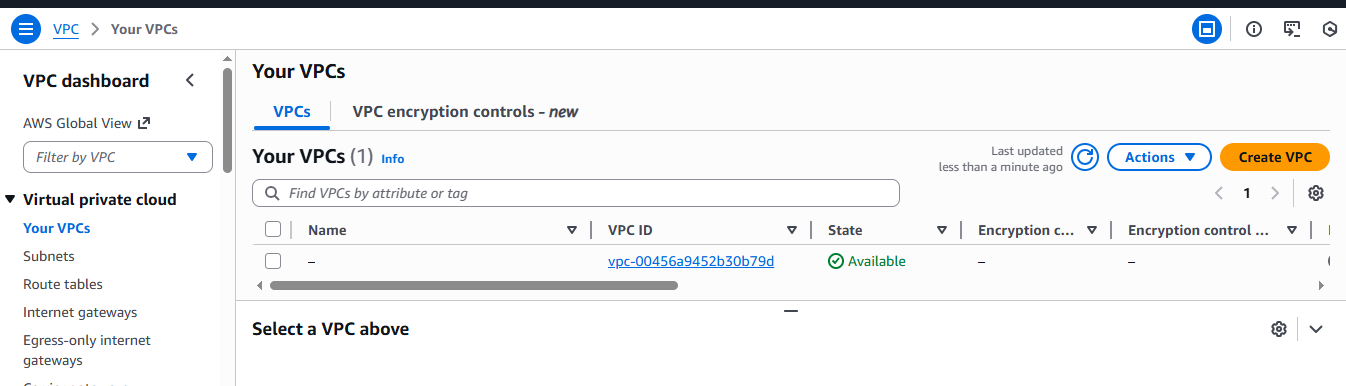
To create a new Virtual Private Cloud (VPC) in AWS with a custom IPv4 CIDR block and verify its successful deployment through the AWS Management Console.

## ****Prerequisites****

* Active AWS account
* IAM user with VPC full access permissions
* Basic understanding of AWS networking concepts
* Access to AWS Management Console

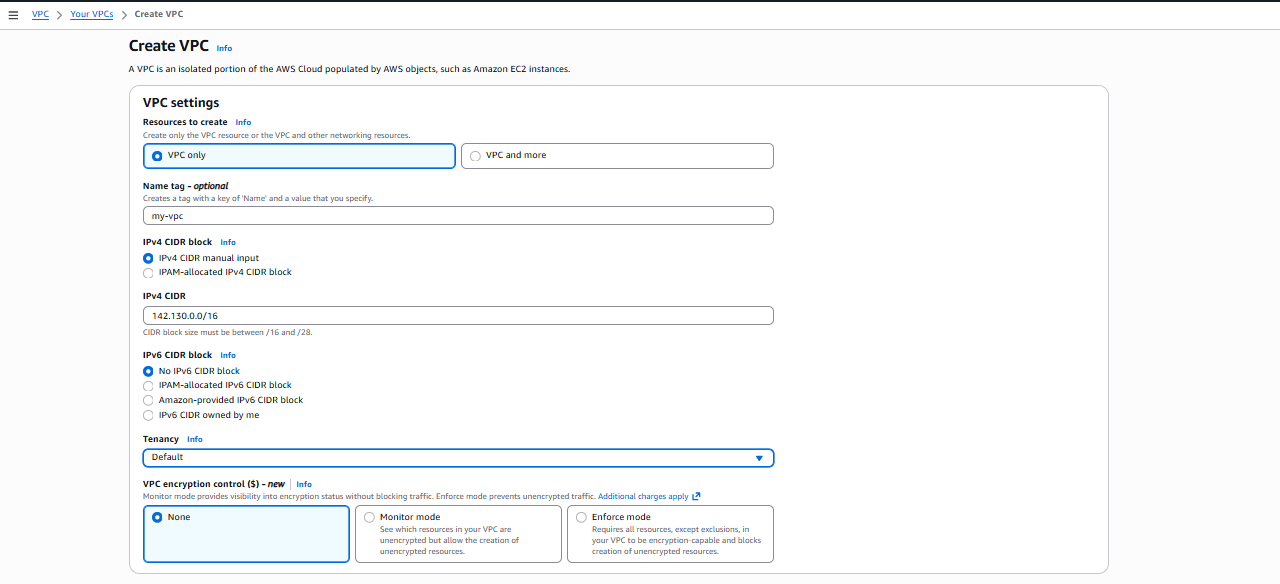
## ****Step-by-Step Implementation****

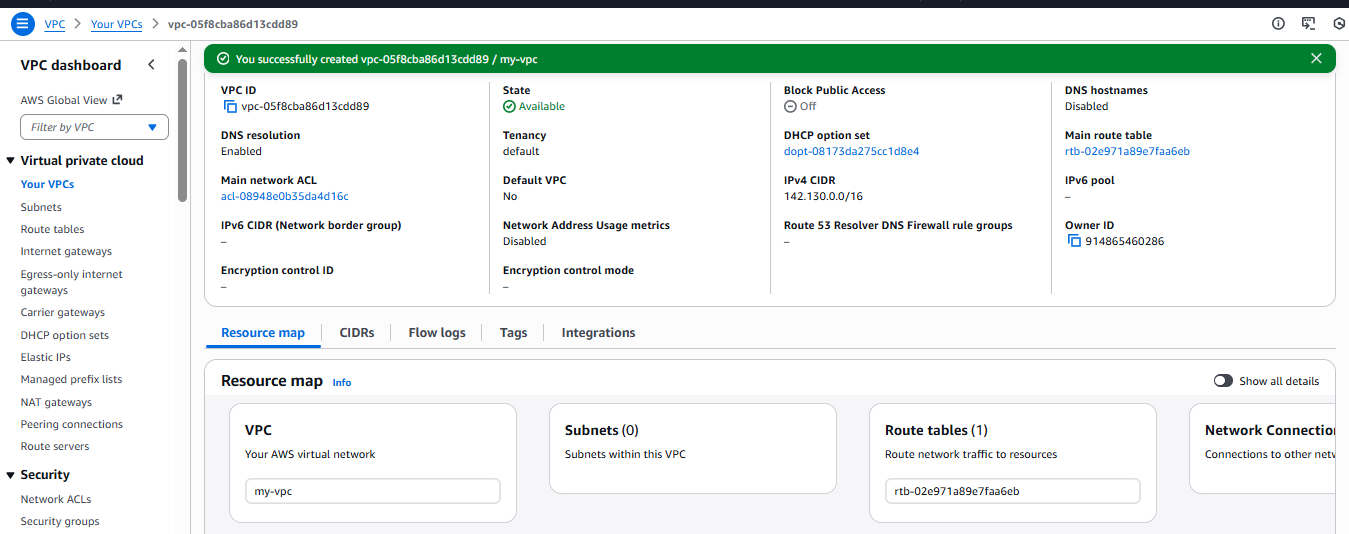
1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard**.
3. Click on **Your VPCs** from the left navigation panel.
4. Click on the **Create VPC** button.
5. Select **VPC only** under resources to create.
6. Enter the **VPC name** as my-vpc.
7. Choose **IPv4 CIDR manual input**.
8. Enter the CIDR block as **142.130.0.0/16**.
9. Leave IPv6 CIDR as **No IPv6 CIDR block**.
10. Keep **Tenancy** as Default.
11. Set **VPC Encryption Control** to None.
12. Click on **Create VPC**.
13. Verify the success message confirming VPC creation.

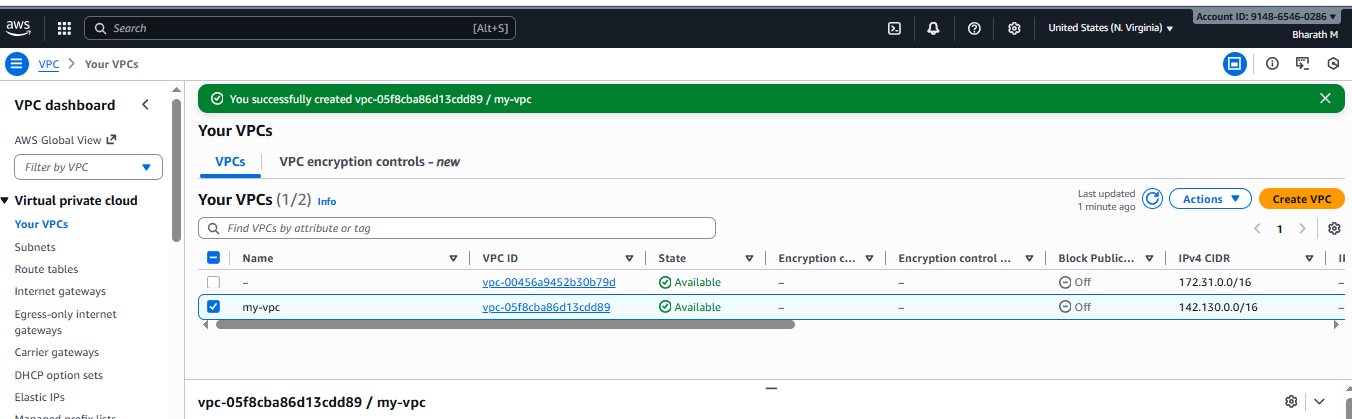
****

## ****Validation Steps****

1. Navigate to **Your VPCs** section in the VPC dashboard.
2. Confirm that the VPC named **my-vpc** appears in the list.
3. Verify the following details:
   * **VPC ID** is generated
   * **State** shows Available
   * **IPv4 CIDR** is 142.130.0.0/16
   * **DNS resolution** is Enabled
4. Open the VPC and check:
   * **Route Table** is automatically created
   * **Subnets count** is zero (as none were created yet)

****

****

****

## ****Issues Faced****

* No major issues were encountered during the VPC creation process.
* Initial confusion regarding IPv4 CIDR block selection was resolved by choosing a valid private range

## ****Issues Faced (Additional / Final Notes)****

* Need to manually create subnets, internet gateway, and route tables to make the VPC functional for internet access.
* Public access is disabled by default and must be configured based on use case.

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

## ****Task Title****

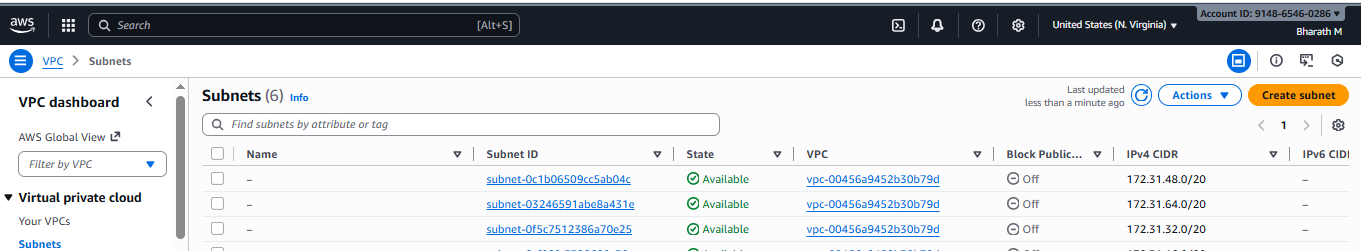
Creation of Public and Private Subnets in AWS VPC

## ****Objective****

To create one public subnet and one private subnet within the previously created VPC (my-vpc) and verify their successful deployment.

## ****Prerequisites****

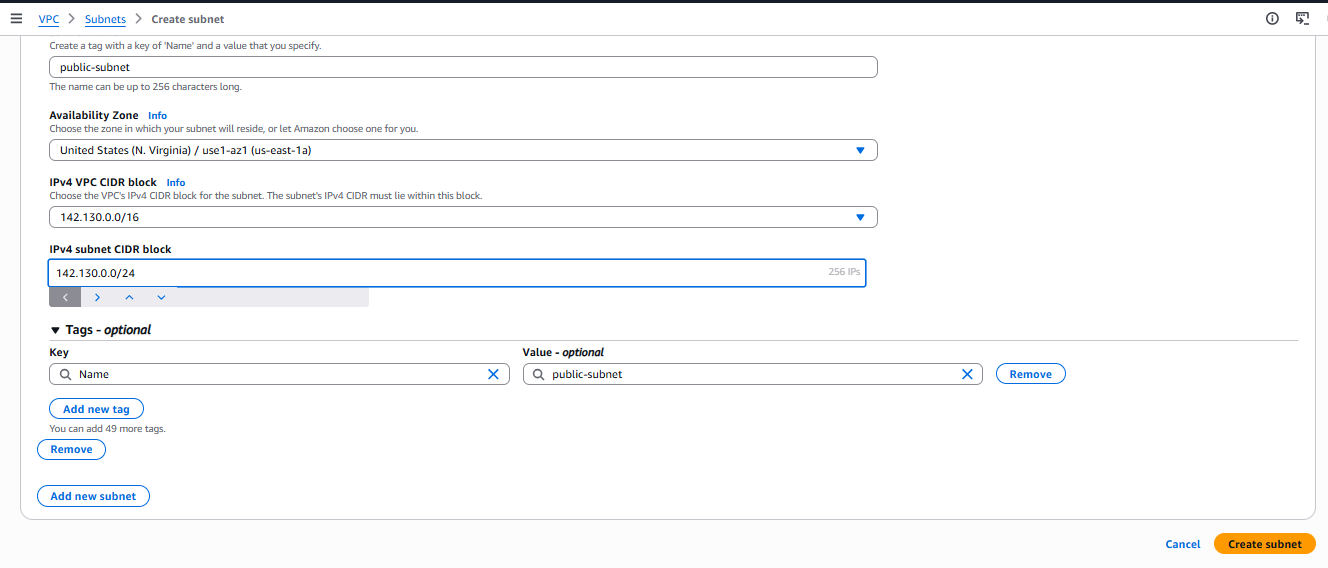
* Active AWS account
* Existing VPC (my-vpc)
* IAM user with VPC and Subnet permissions
* Knowledge of CIDR block concept
* Access to AWS Management Console



## ****Step-by-Step Implementation****

### **A. Creation of Public Subnet**

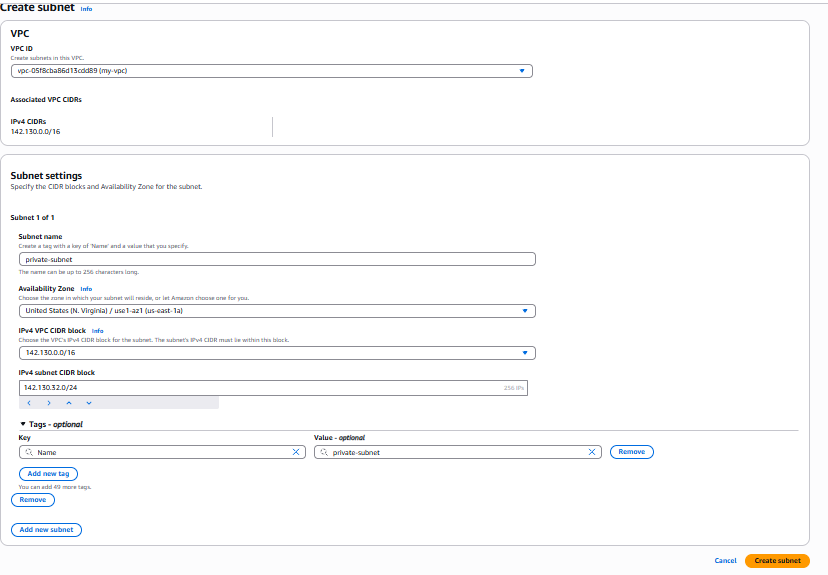
1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard** → **Subnets**.
3. Click on **Create Subnet**.
4. Select the VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
5. Enter **Subnet name** as public-subnet.
6. Select **Availability Zone**: us-east-1a (N. Virginia).
7. Choose **IPv4 VPC CIDR block**: 142.130.0.0/16.
8. Enter **IPv4 Subnet CIDR block**: **142.130.0.0/24**.
9. Add a tag:
   * Key: Name
   * Value: public-subnet
10. Click on **Create Subnet**.
11. Verify the success message confirming subnet creation.

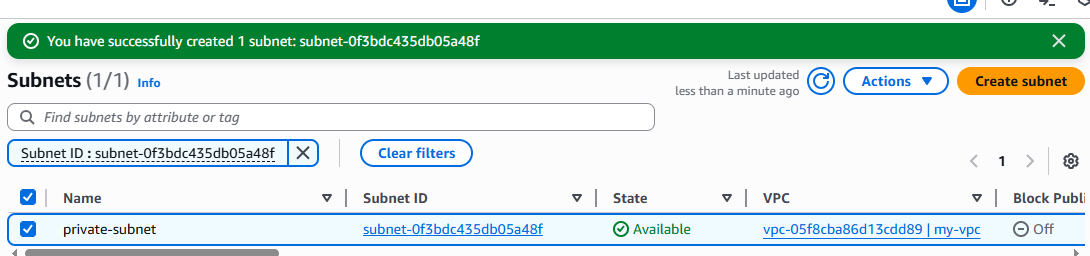


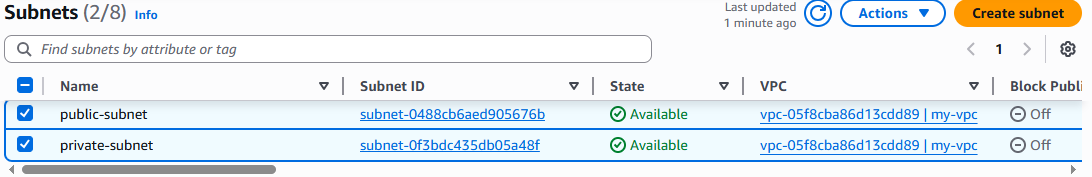


### **B. Creation of Private Subnet**

1. From the **Subnets** section, click on **Create Subnet** again.
2. Select the same VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
3. Enter **Subnet name** as private-subnet.
4. Select **Availability Zone**: us-east-1a (N. Virginia).
5. Choose **IPv4 VPC CIDR block**: 142.130.0.0/16.
6. Enter **IPv4 Subnet CIDR block**: **142.130.32.0/24**.
7. Add a tag:
   * Key: Name
   * Value: private-subnet
8. Click on **Create Subnet**.
9. Verify the success message confirming subnet creation.







## ****Validation Steps****

1. Navigate to **VPC Dashboard → Subnets**.
2. Verify that both subnets appear in the list:
   * **public-subnet** → CIDR: 142.130.0.0/24
   * **private-subnet** → CIDR: 142.130.32.0/24
3. Confirm that:
   * Subnet **State** is Available.
   * Both subnets belong to **my-vpc**.
   * **Block Public Access** shows Off.
4. Open each subnet and verify its configuration details.

## ****Issues Faced****

* No major technical issues were encountered during subnet creation.
* Initial confusion occurred while selecting non-overlapping CIDR blocks, which was resolved by choosing different /24 ranges.

## ****Issues Faced (Additional Notes)****

* Public and private subnets are not fully functional until:
  + An **Internet Gateway** is attached to the VPC
  + **Route tables** are properly configured
  + A **NAT Gateway** is added for private subnet internet access

1. **Attach an IGW to the VPC.**

## ****Task Title****

Creation and Attachment of Internet Gateway to AWS VPC

## ****Objective****

To create an Internet Gateway (IGW) and attach it to the existing VPC (my-vpc) in order to enable internet connectivity for resources in the public subnet.

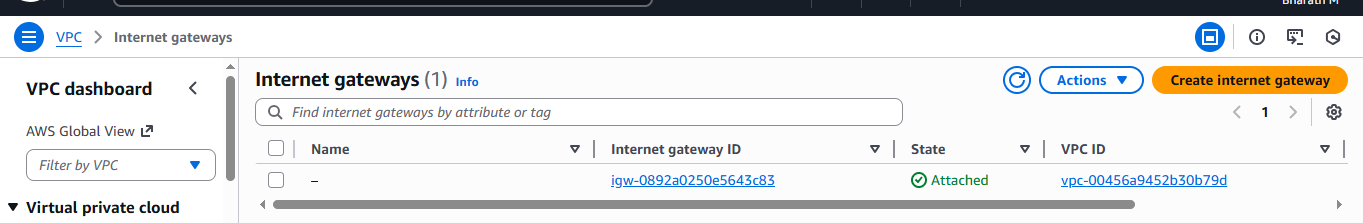
## ****Prerequisites****

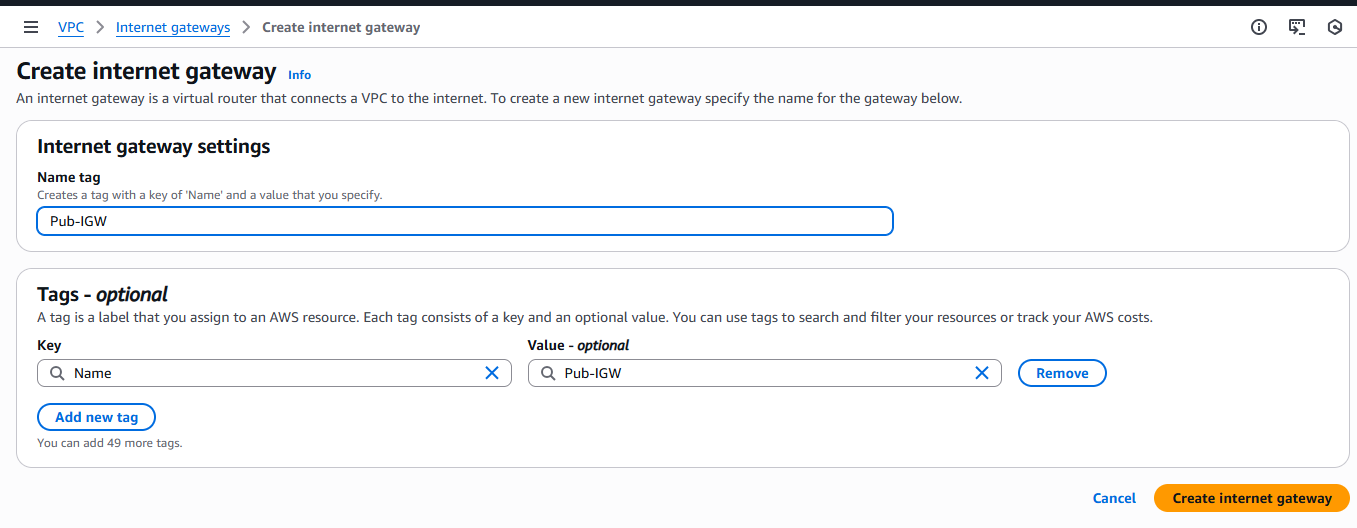
* Active AWS account
* Existing VPC (my-vpc)
* Public subnet already created
* IAM user with VPC and Internet Gateway permissions
* Access to AWS Management Console

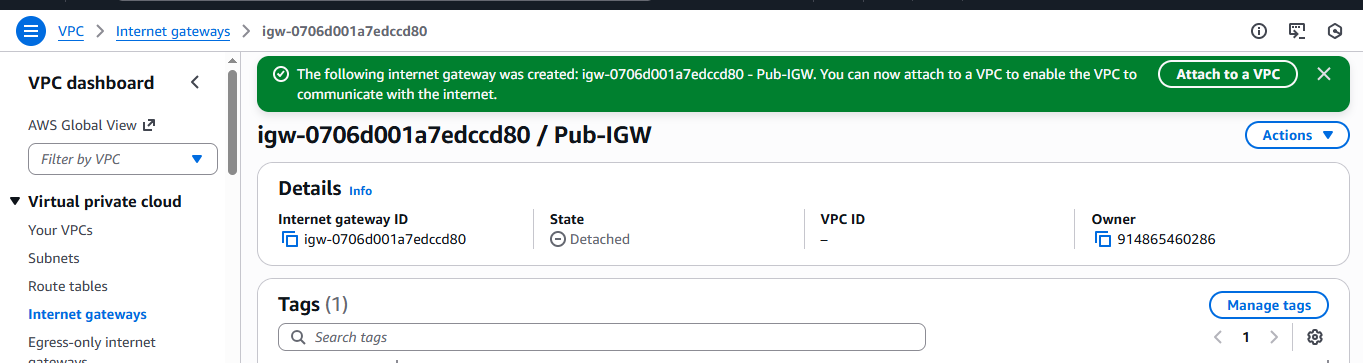
## ****Step-by-Step Implementation****

### **A. Creation of Internet Gateway**

1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard**.
3. Click on **Internet Gateways** from the left panel.
4. Click on **Create Internet Gateway**.
5. Enter the **Name tag** as: Pub-IGW.
6. Add a tag:
   * Key: Name
   * Value: Pub-IGW
7. Click on **Create Internet Gateway**.
8. Verify the success message confirming Internet Gateway creation.

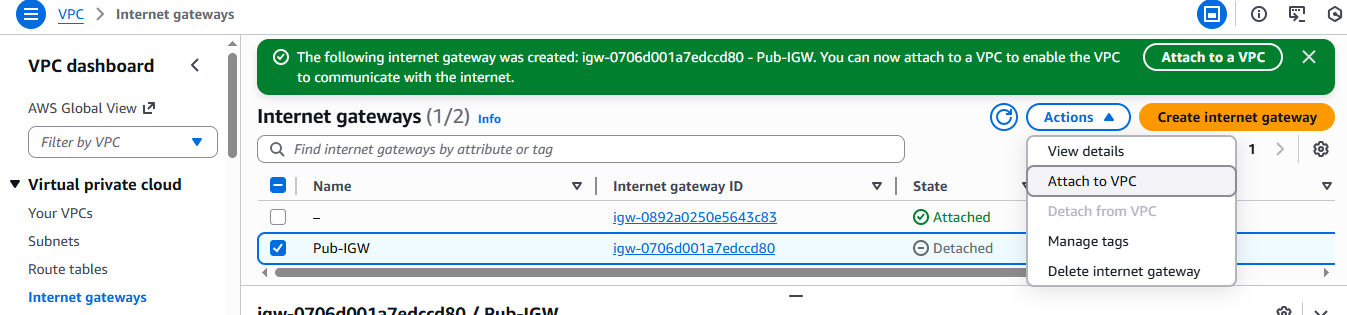
****

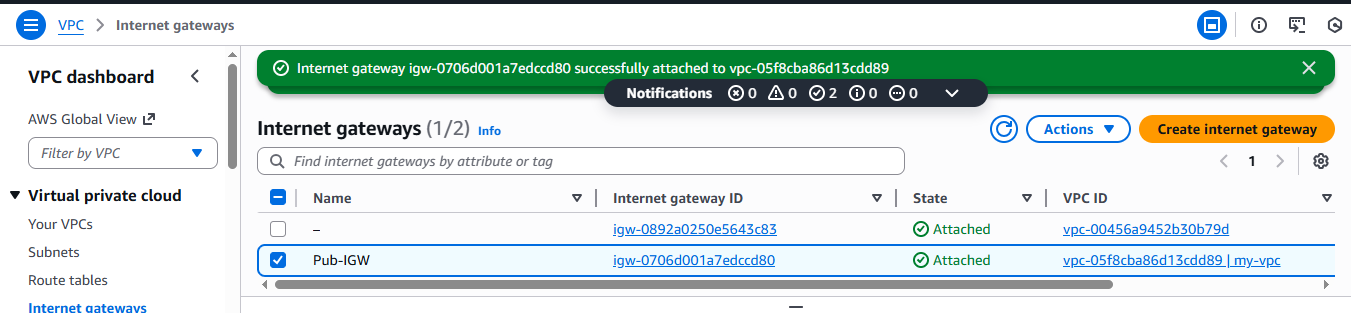
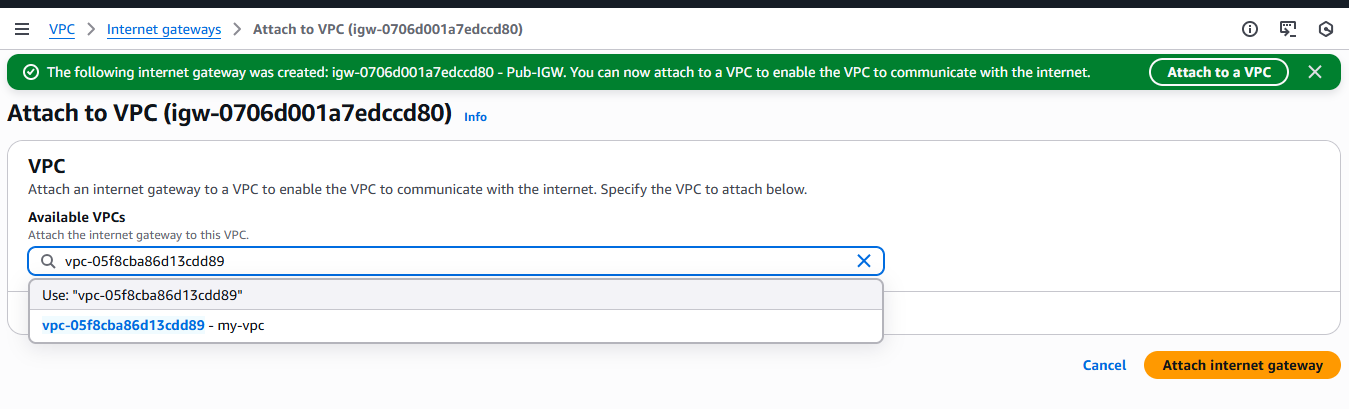
****

****

### **B. Attaching Internet Gateway to VPC**

1. Select the newly created **Internet Gateway (Pub-IGW)**.
2. Click on **Actions → Attach to VPC**.
3. Select the VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
4. Click on **Attach Internet Gateway**.
5. Verify the success message confirming that the Internet Gateway is successfully attached to the VPC.

****

****

## ****Validation Steps****

1. Navigate to **VPC Dashboard → Internet Gateways**.
2. Verify that **Pub-IGW** appears in the list.
3. Confirm the following details:
   * **State**: Attached
   * **VPC ID**: vpc-05fc8ba8613cdd89 (my-vpc)
4. Open the Internet Gateway and verify its attachment details.

## ****Issues Faced****

* No major issues were encountered during the creation and attachment of the Internet Gateway.
* Initial confusion occurred between multiple Internet Gateways, which was resolved by confirming the correct VPC ID before attachment.

## ****Issues Faced (Additional Notes)****

* Simply attaching an Internet Gateway does not provide internet access until:
  + The **Route Table** is updated with a route to the Internet Gateway
  + The route is associated with the **public subnet**
* Private subnets still require a **NAT Gateway** for outbound internet access.

✅ **Status:** Internet Gateway successfully created and attached to my-vpc.

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

## ****Task Title****

Creation and Configuration of Public and Private Route Tables in AWS VPC

## ****Objective****

To create separate route tables for public and private subnets, associate them with the appropriate subnets, and configure internet routing using the Internet Gateway for public subnet communication.

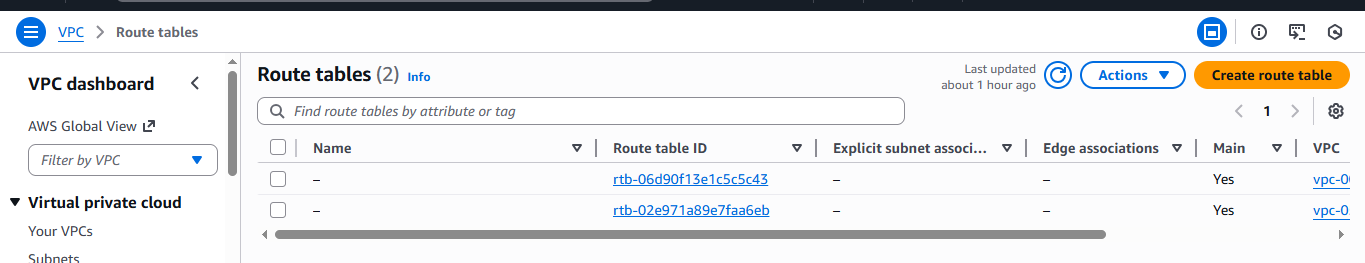
## ****Prerequisites****

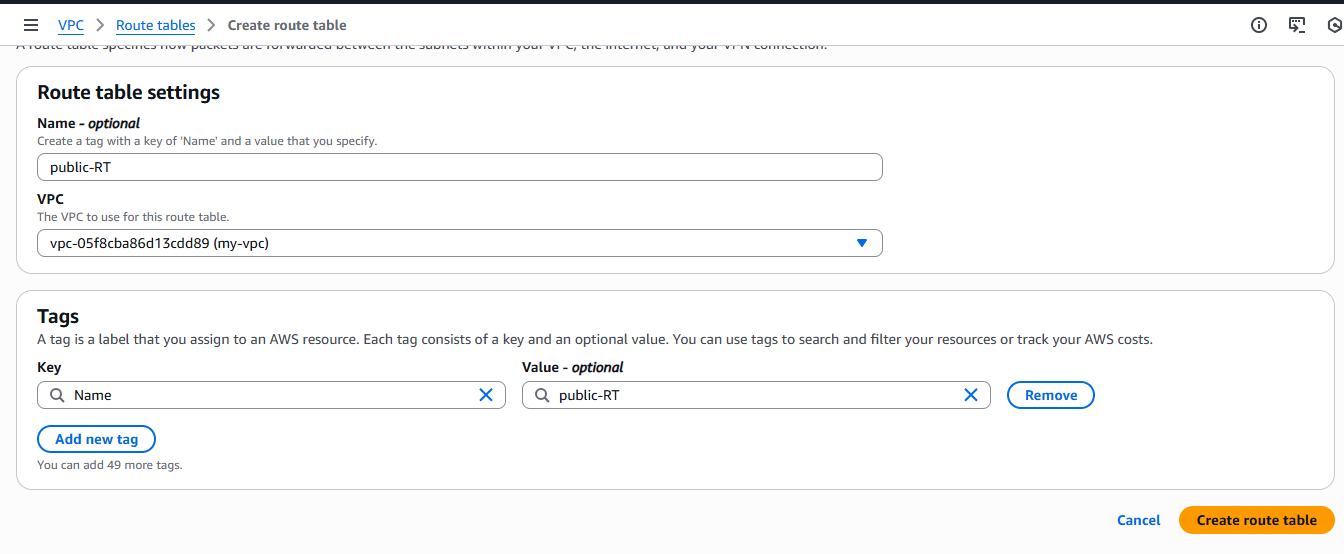
* Active AWS account
* Existing VPC (my-vpc)
* Public and Private subnets created
* Internet Gateway (Pub-IGW) attached to the VPC
* IAM user with Route Table permissions
* Access to AWS Management Console

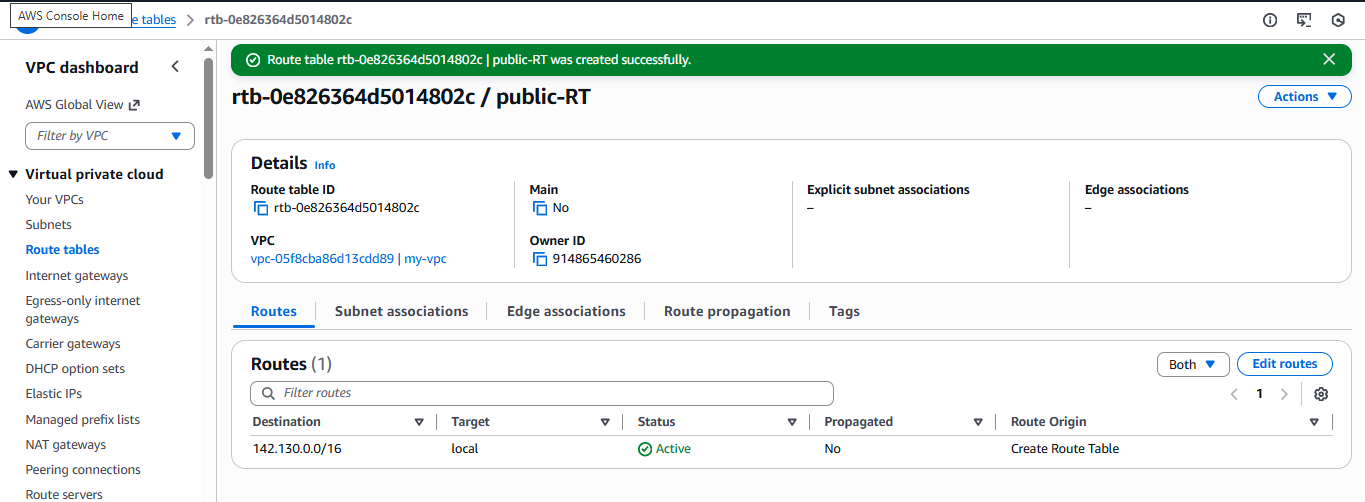
## ****Step-by-Step Implementation****

### **Creation of Public Route Table**

1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard → Route Tables**.
3. Click on **Create Route Table**.
4. Enter **Name** as public-RT.
5. Select the VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
6. Add a tag:
   * Key: Name
   * Value: public-RT
7. Click on **Create Route Table**.
8. Verify the success message confirming creation of **public-RT**.

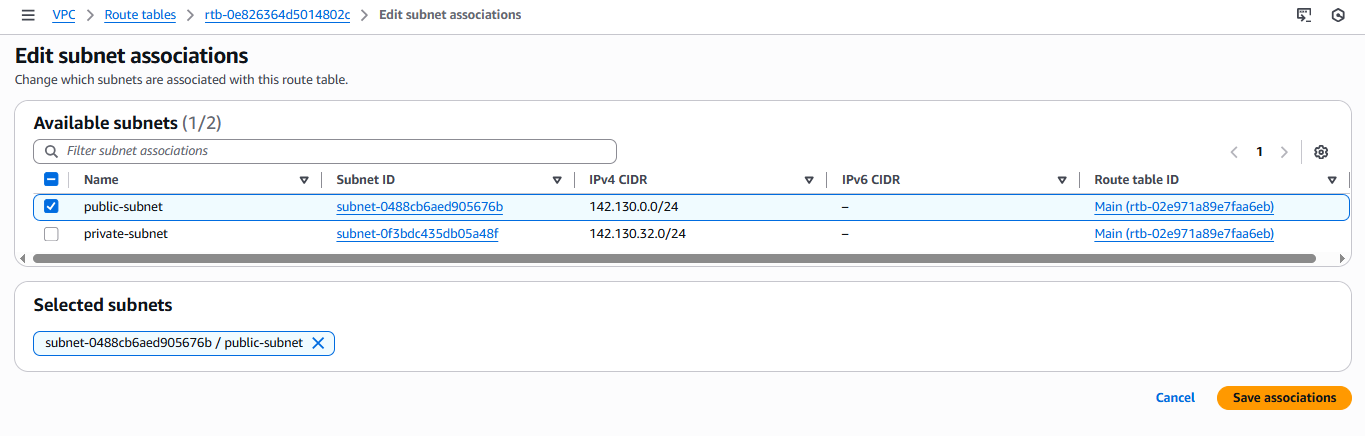
****

****

****

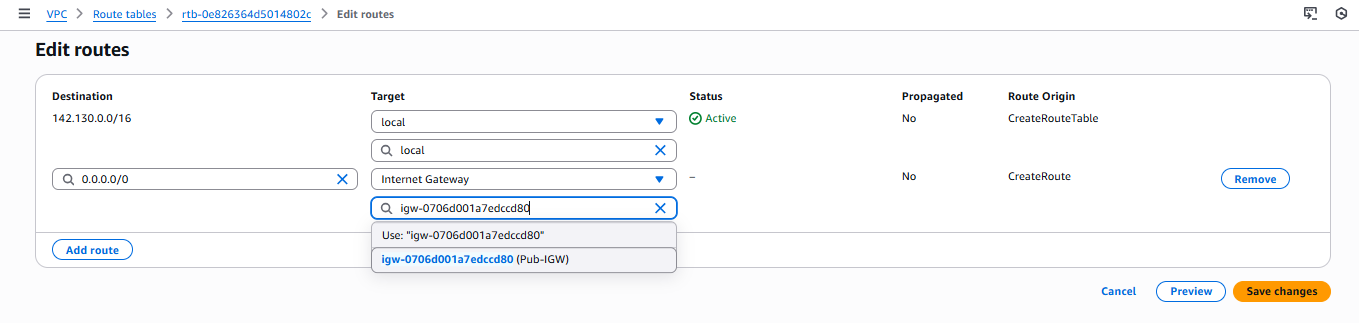
### **Associating Public Subnet with Public Route Table**

1. Select **public-RT**.
2. Open the **Subnet Associations** tab.
3. Click **Edit subnet associations**.
4. Select **public-subnet**.
5. Click **Save associations**.
6. Verify that **public-subnet** is now associated with **public-RT**.

****

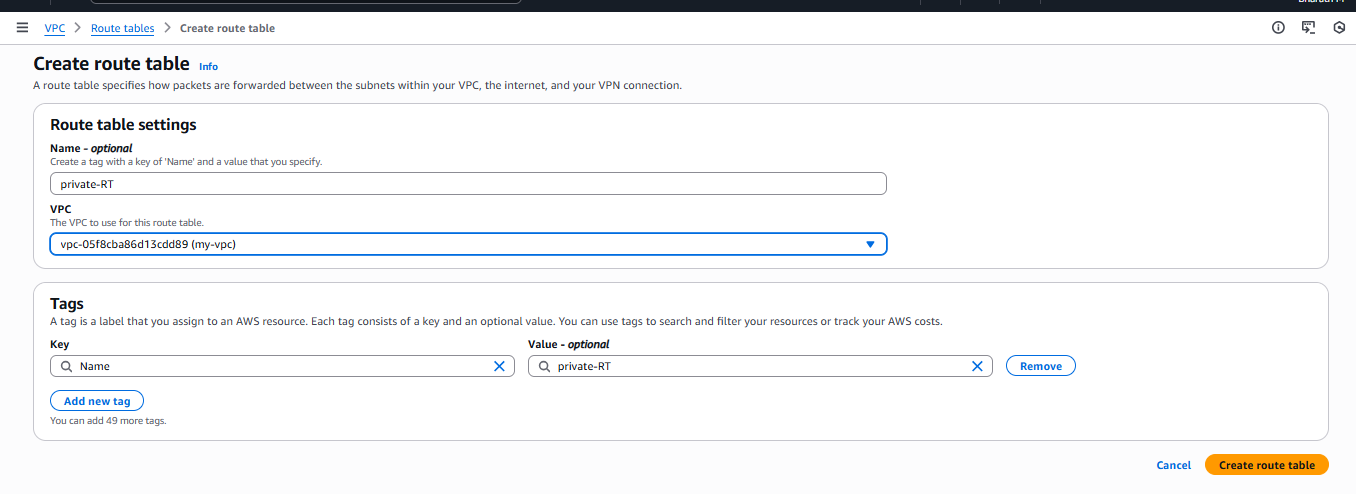
### **Adding Internet Route to Public Route Table**

1. Select the **public-RT** route table.
2. Go to the **Routes** tab and click **Edit routes**.
3. Click **Add route**.
4. Enter:
   * **Destination:** 0.0.0.0/0
   * **Target:** Internet Gateway (Pub-IGW)
5. Click **Save changes**.
6. Verify that the new route shows:
   * Destination: 0.0.0.0/0
   * Target: igw-xxxxxxxx (Pub-IGW)
   * Status: Active

****

### **Creation of Private Route Table**

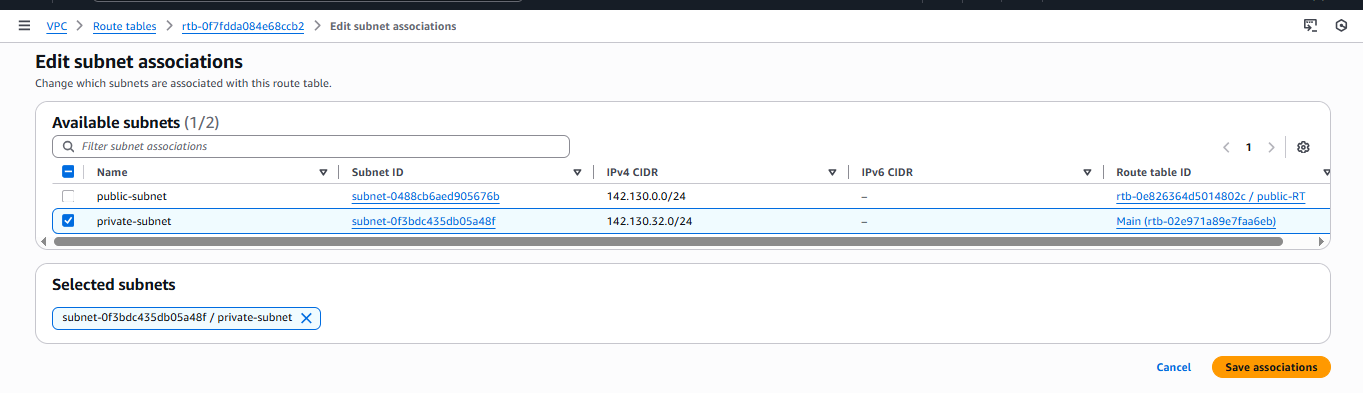
1. Navigate back to **Route Tables**.
2. Click **Create Route Table**.
3. Enter **Name** as private-RT.
4. Select the same VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
5. Add a tag:
   * Key: Name
   * Value: private-RT
6. Click **Create Route Table**.
7. Verify the success message confirming creation of **private-RT**.

****

### **Associating Private Subnet with Private Route Table**

1. Select **private-RT**.
2. Open the **Subnet Associations** tab.
3. Click **Edit subnet associations**.
4. Select **private-subnet**.
5. Click **Save associations**.
6. Verify that **private-subnet** is now associated with **private-RT**.

✅ Note: No internet route (0.0.0.0/0) is added to the private route table, keeping the private subnet isolated from direct internet access.

****

## ****Validation Steps****

1. Navigate to **VPC Dashboard → Route Tables**.
2. Confirm the following route tables exist:
   * **public-RT**
   * **private-RT**
3. Open **public-RT** and verify:
   * There is a route for 0.0.0.0/0 pointing to **Internet Gateway (Pub-IGW)**
   * **public-subnet** is associated
4. Open **private-RT** and verify:
   * Only local route (142.130.0.0/16 → local) exists
   * **private-subnet** is associated
5. Ensure both route tables belong to **my-vpc**.

## ****Issues Faced****

* No major technical issues occurred during route table creation.
* Initially, the Internet Gateway was not visible in route options, which was resolved by confirming IGW was properly attached to the VPC.

## ****Issues Faced (Additional Notes / Final Observations)****

* Public subnet gained internet access only after:
  + Internet Gateway attachment
  + Correct route (0.0.0.0/0) added
  + Proper subnet association
* Private subnet remains isolated and will require a **NAT Gateway** for outbound internet access if needed.

✅ **Status:** Public and Private Route Tables successfully created, configured, and associated with their respective subnets.

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

## ****Task Title****

Creation and Configuration of NAT Gateway for Private Subnet Internet Access

## ****Objective****

To create a NAT Gateway in the public subnet and configure the private route table to allow private subnet instances to access the internet securely without being directly exposed.

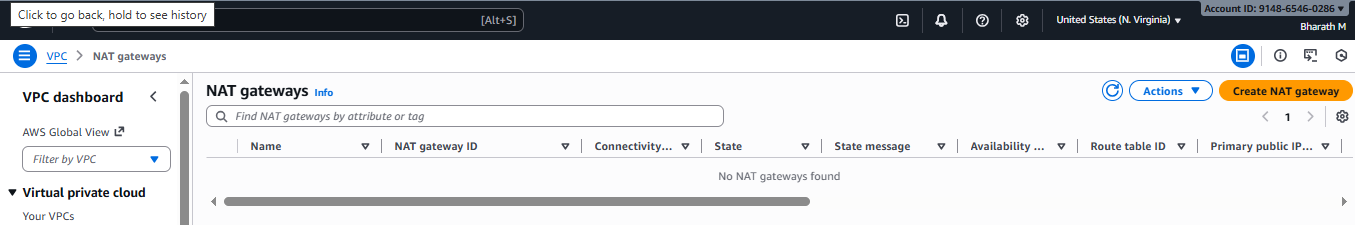
## ****Prerequisites****

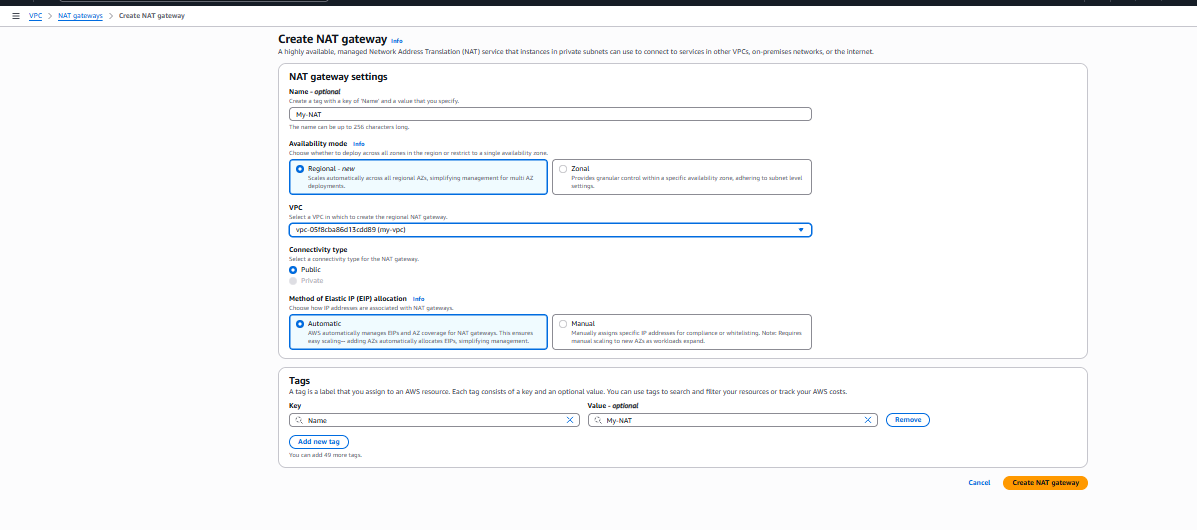
* Active AWS account
* Existing VPC: my-vpc
* Public and Private subnets created
* Internet Gateway attached to the VPC
* Public Route Table and Private Route Table created
* Elastic IP allocation permission
* IAM user with NAT Gateway permissions
* Access to AWS Management Console

## ****Step-by-Step Implementation****

### **A. Creation of NAT Gateway**

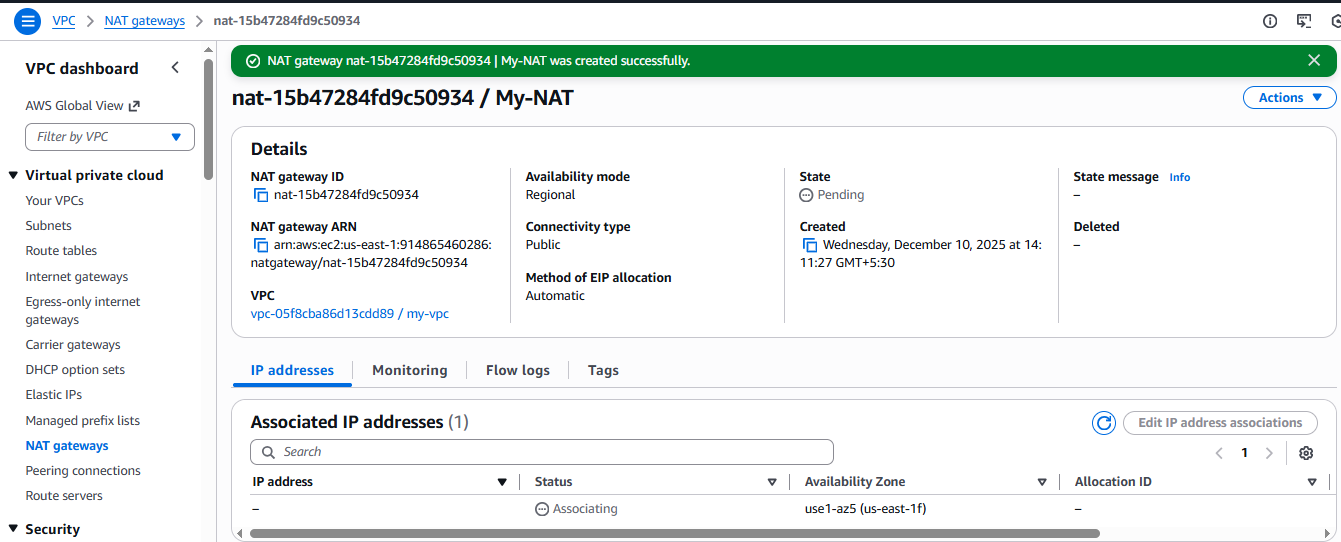
1. Log in to the **AWS Management Console**.
2. Navigate to **VPC Dashboard → NAT Gateways**.
3. Click on **Create NAT Gateway**.
4. Enter the **Name** as My-NAT.
5. Select **Availability Mode** as Regional.
6. Select the VPC: **vpc-05fc8ba8613cdd89 (my-vpc)**.
7. Choose **Connectivity Type** as Public.
8. For **Elastic IP allocation**, select Automatic.
9. Add a tag:
   * Key: Name
   * Value: My-NAT
10. Click on **Create NAT Gateway**.
11. Verify the success message confirming NAT Gateway creation.
12. Wait until the **State** changes from Pending to Available.

****

****

### **B. Verification of NAT Gateway**

1. Open the newly created NAT Gateway (My-NAT).
2. Verify the following:
   * **NAT Gateway ID** is generated
   * **State** is Available
   * **Connectivity Type** is Public
   * **Elastic IP** is successfully associated
   * **VPC** is my-vpc

****

### **C. Updating Private Route Table for NAT Gateway Routing**

1. Navigate to **VPC Dashboard → Route Tables**.
2. Select the **private-RT** route table.
3. Go to the **Routes** tab and click **Edit routes**.
4. Click **Add route**.
5. Enter:
   * **Destination:** 0.0.0.0/0
   * **Target:** NAT Gateway (My-NAT)
6. Click on **Save changes**.
7. Verify that the new route is added and shows **Status: Active**.

✅ This ensures that instances in the private subnet can access the internet through the NAT Gateway

****

## ****Validation Steps****

1. Navigate to **VPC Dashboard → NAT Gateways**.
2. Confirm that **My-NAT** shows:
   * **State:** Available
   * **Elastic IP:** Associated
3. Navigate to **Route Tables → private-RT**.
4. Verify the routes:
   * 142.130.0.0/16 → local
   * 0.0.0.0/0 → nat-xxxxxxxx (My-NAT)
5. Confirm that:
   * **private-subnet** is associated with **private-RT**
   * **public-subnet** remains associated with **public-RT**

## ****Issues Faced****

* NAT Gateway initially remained in Pending state for a few minutes, which was resolved automatically after Elastic IP association completed.
* The NAT Gateway option was not immediately visible in route table targets until the gateway became Available.

## ****Issues Faced (Additional Notes / Final Observations)****

* NAT Gateway allows **only outbound internet access** for private instances.
* Private instances still cannot receive inbound traffic directly from the internet.
* NAT Gateway incurs **additional AWS charges** based on:
  + Hourly usage
  + Data processed
* Proper routing via **private-RT** is mandatory for NAT to function.

✅ **Status:** NAT Gateway successfully created and configured. Private subnet now has secure outbound internet access.

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

## ****Task Title****

Launching EC2 Instances in Public and Private Subnets

## ****Objective****

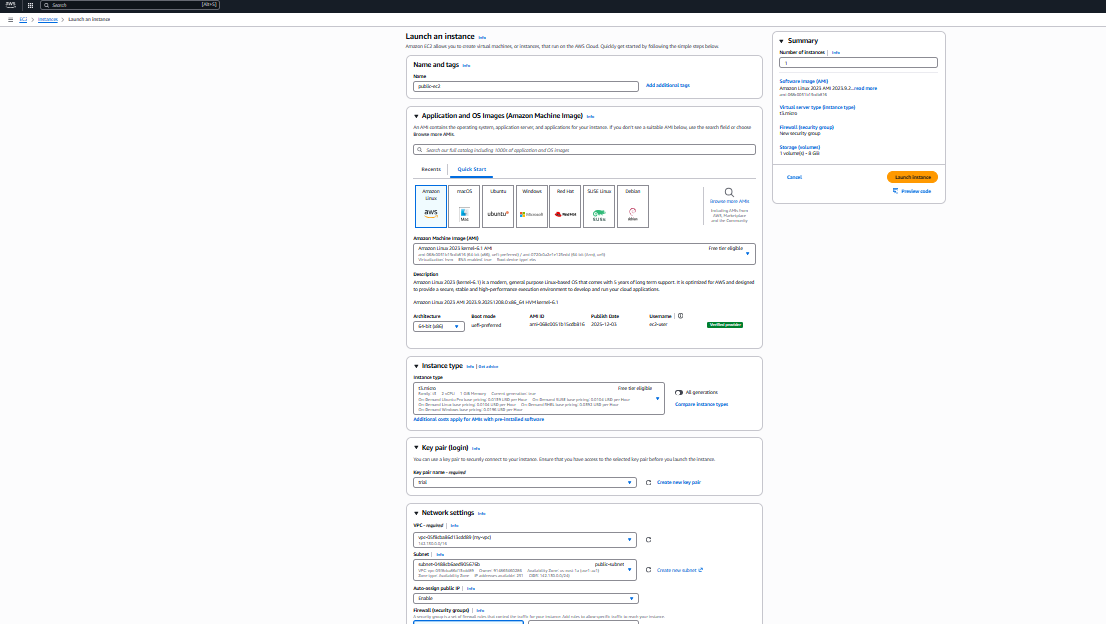
To launch one EC2 instance in the public subnet and one EC2 instance in the private subnet of the VPC (my-vpc) and verify their successful deployment and running state.

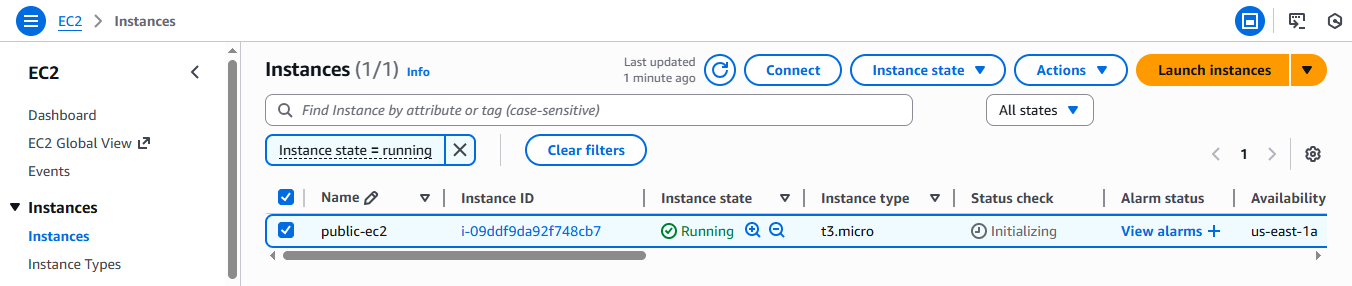
## ****Prerequisites****

* Active AWS account
* Existing VPC (my-vpc)
* Public and Private subnets configured
* Internet Gateway attached to VPC
* NAT Gateway configured for private subnet
* Proper Route Tables associated
* Key pair created for EC2 login
* IAM user with EC2 full access permissions

### **A. Launching EC2 Instance in Public Subnet**

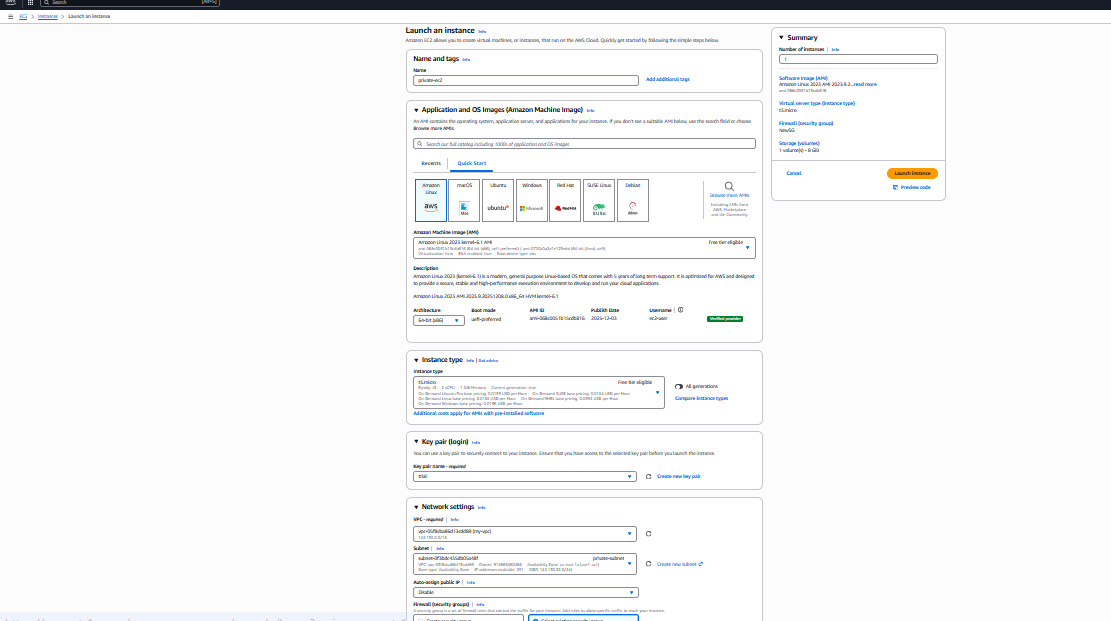
1. Log in to the **AWS Management Console**.
2. Navigate to **EC2 Dashboard → Instances**.
3. Click on **Launch Instance**.
4. Enter the **Instance Name** as public-ec2.
5. Select **Amazon Machine Image (AMI)**: Amazon Linux 2.
6. Choose **Instance Type**: t3.micro.
7. Select an existing **Key Pair** for SSH access.
8. In **Network Settings**:
   * Select VPC: **my-vpc**
   * Select Subnet: **public-subnet**
   * Enable **Auto-assign Public IP**
   * Configure Security Group to allow:
     + SSH (Port 22)
     + HTTP (Port 80), if required
9. Keep storage and other settings as default.
10. Click on **Launch Instance**.
11. Verify that the instance state becomes **Running**.

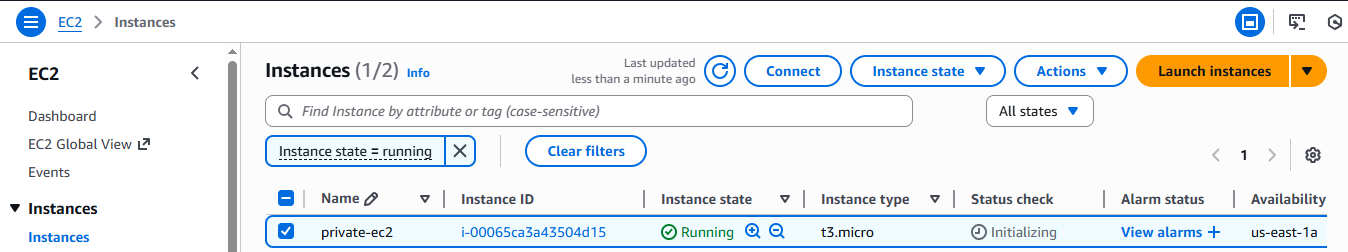
****

****

### **B. Launching EC2 Instance in Private Subnet**

1. Navigate again to **EC2 Dashboard → Instances**.
2. Click on **Launch Instance**.
3. Enter the **Instance Name** as private-ec2.
4. Select **Amazon Machine Image (AMI)**: Amazon Linux 2.
5. Choose **Instance Type**: t3.micro.
6. Select the same **Key Pair**.
7. In **Network Settings**:
   * Select VPC: **my-vpc**
   * Select Subnet: **private-subnet**
   * Disable **Auto-assign Public IP**
   * Use a Security Group allowing:
     + SSH (Port 22) only from the public subnet or bastion host
8. Keep remaining settings as default.
9. Click on **Launch Instance**.
10. Verify that the instance state becomes **Running**.

****

****

## ****Validation Steps****

1. Navigate to **EC2 Dashboard → Instances**.
2. Verify that both instances are listed:
   * **public-ec2**
   * **private-ec2**
3. Confirm the following for both instances:
   * **Instance State:** Running
   * **Instance Type:** t3.micro
   * **Availability Zone:** us-east-1a
4. Validate connectivity:
   * Public EC2 is accessible via **Public IP using SSH**.
   * Private EC2 has **no public IP** and can access the internet only via **NAT Gateway**.
5. Verify outbound internet access from private EC2 using:
   * ping google.com or yum update

## ****Issues Faced****

* Initially, the public EC2 instance did not allow SSH access due to a missing inbound rule in the Security Group, which was resolved by allowing **Port 22 from Anywhere**.
* The private EC2 instance had no internet access initially due to missing **NAT Gateway route**, which was resolved after updating the **private route table**.

## ****Issues Faced (Additional Notes / Final Observations)****

* Public EC2 instances receive a **public IP address** and are directly reachable from the internet.
* Private EC2 instances remain **securely isolated** and can access the internet only through the **NAT Gateway**.
* Security Groups play a critical role in controlling inbound and outbound traffic.
* Proper subnet and route table association is mandatory for successful connectivity.

✅ **Status:** EC2 instances in both public and private subnets were successfully launched and are running.

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

## ****Task Title****

Creation of NAT Gateway, EC2 Instances (Public & Private), and Web Server Deployment

## ****Objective****

To enable internet access for private subnet instances using a NAT Gateway, launch EC2 instances in both public and private subnets, and deploy an Apache web server on the public EC2 instance for verification of internet connectivity.

## ****Prerequisites****

* Active AWS account
* Existing VPC with public and private subnets
* Internet Gateway attached to the VPC
* Proper route tables created
* Key pair for EC2 access
* Security groups allowing:
  + SSH (Port 22)
  + HTTP (Port 80)
* Elastic IP allocation available for NAT Gateway

### **1. Route Table Creation**

* Created **Public Route Table** (public-RT)
* Created **Private Route Table** (private-RT)
* Associated:
  + Public subnet → Public Route Table
  + Private subnet → Private Route Table

### **2. Internet Gateway Configuration**

* Internet Gateway already attached to VPC
* Added route in **Public Route Table**:
  + 0.0.0.0/0 → Internet Gatewa

### **3. NAT Gateway Creation**

* Created a **Public NAT Gateway**
* Elastic IP assigned automatically
* NAT Gateway placed in **Public Subnet**

### **4. Private Route Table Update**

* Added route:
  + 0.0.0.0/0 → NAT Gateway
* This allows private subnet instances to access the internet securely.

### **Launching Public EC2 Instance**

1. Navigate to **EC2 → Launch Instance**.
2. Enter:
   * **Name:** public-ec2
   * **AMI:** Amazon Linux 2 / 2023
   * **Instance Type:** t3.micro
   * **Key Pair:** Selected existing key
   * **VPC:** my-vpc
   * **Subnet:** Public Subnet
   * **Auto-assign Public IP:** Enabled
3. Configure Security Group:
   * Allow **SSH (22)**
   * Allow **HTTP (80)**
4. Click **Launch Instance**.
5. Verify instance state is **Running**.

### **Connecting to Public EC2 via SSH**

ssh -i trial.pem ec2-user@3.239.224.72

* SSH fingerprint verification accepted.
* Logged in successfully as ec2-user

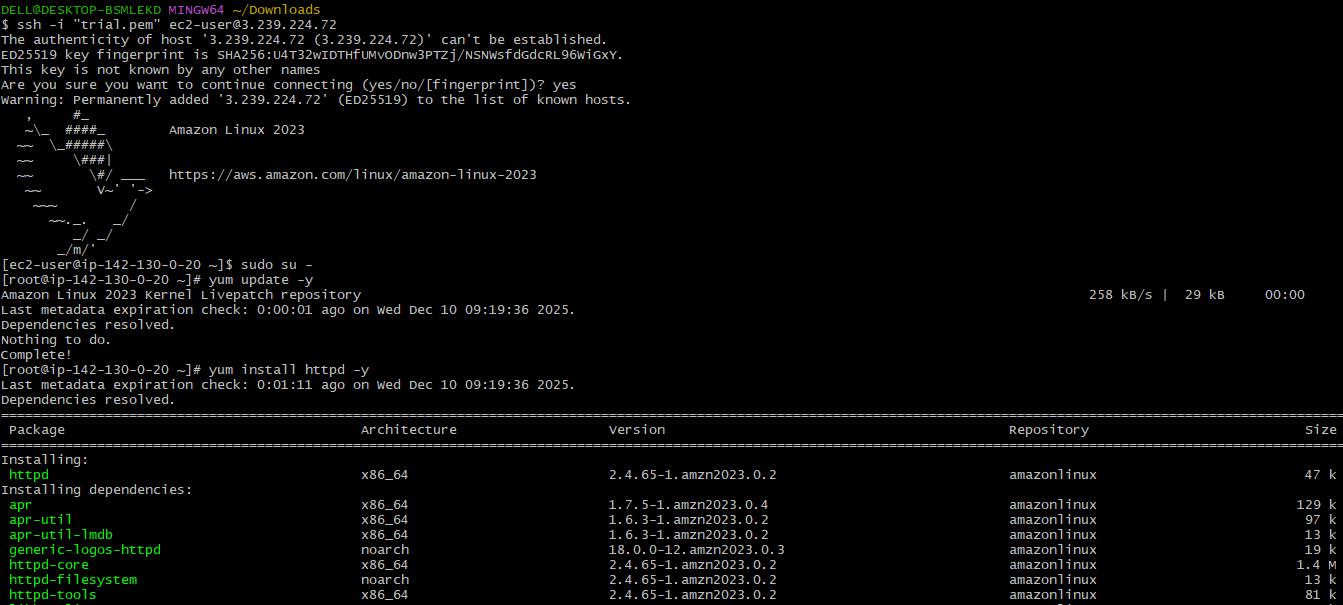
### **Installing Apache Web Server**

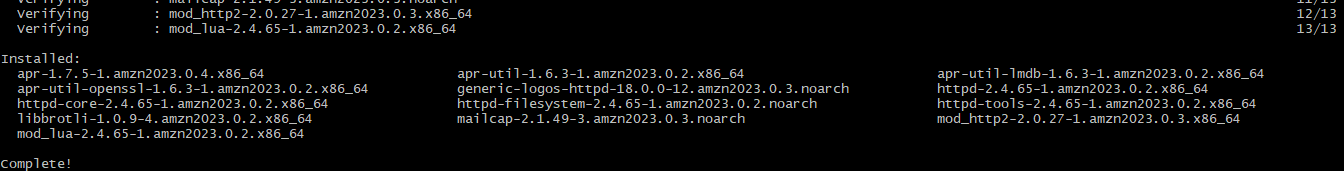
sudo su -

yum update -y

yum install httpd -y

* All required dependencies installed successfully.

****

****

### **Creating Web Page**

vi /var/www/html/index.html

****

Entered content:

WELCOME TO INDIA

Verify file:

cat /var/www/html/index.html

****

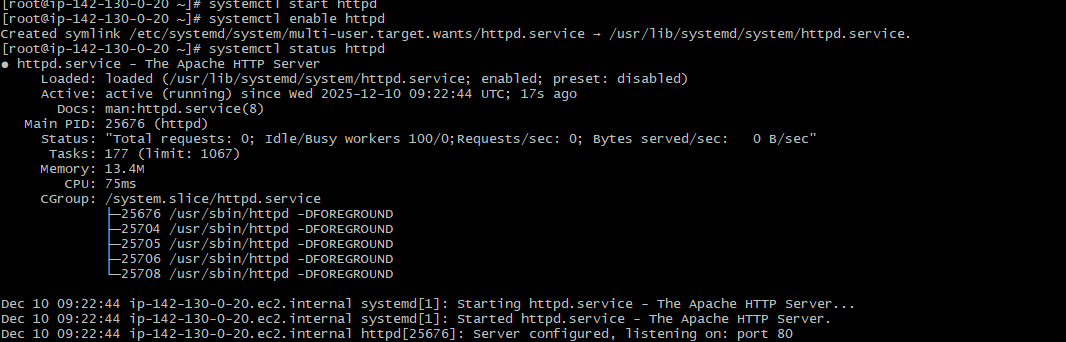
### **Starting and Enabling Apache Service**

systemctl start httpd

systemctl enable httpd

systemctl status httpd

* Service status: **Active (Running)**
* Server listening on **Port 80**.

****

### **Web Application Validation**

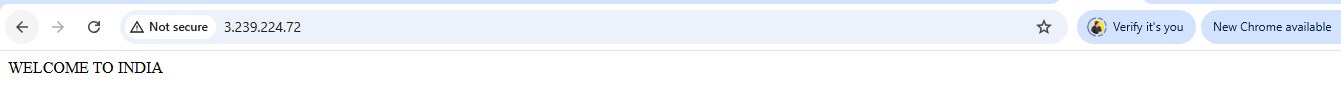
1. Open browser.
2. Enter public IP:

http://3.239.224.72

1. Webpage displays:

WELCOME TO INDIA

✅ Web server deployment successfully verified

****

## ****Validation Steps****

* NAT Gateway state: **Available**
* Private Route Table: Route 0.0.0.0/0 → My-NAT is **Active**
* Public EC2 instance: **Running with public IP**
* Private EC2 instance: **Running without public IP**
* Apache service: **Active**
* Web application accessible through browser

## ****Issues Faced****

* Initially NAT Gateway was in **Pending** state; resolved after waiting a few minutes.
* SSH connection failed initially due to missing **port 22 rule** in security group.
* HTTP page was not accessible until **port 80** was allowed in the security group.

## ****Issues Faced (Additional Observations)****

* NAT Gateway must be placed in a **public subnet** to provide internet access to private instances.
* Without updating the **private route table**, the private EC2 had no outbound internet.
* Apache service must be **enabled** for auto-start after reboot.

## ****Prerequisites****

* AWS Account with VPC & EC2 access
* Key pair (.pem file) for SSH access
* Basic knowledge of:
  + VPC, Subnets, Route Tables
  + Internet Gateway & NAT Gateway
  + Linux commands
  + Apache web server
* SSH client (Git Bash / PowerShell / MobaXterm)
* Security Groups allowing:
  + SSH (Port 22)
  + HTTP (Port 80)

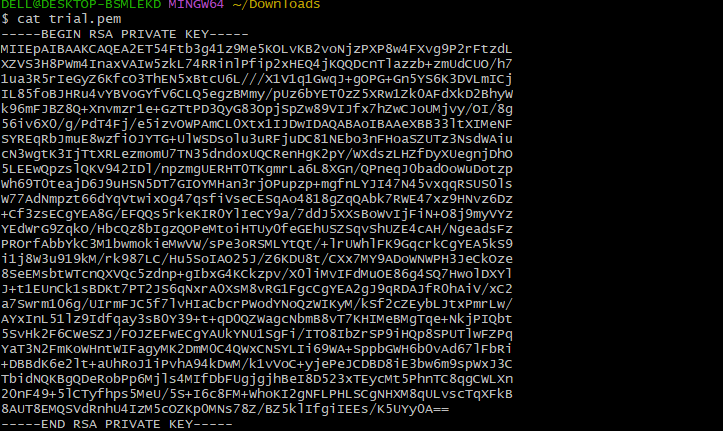
### **Private Server Access via SSH**

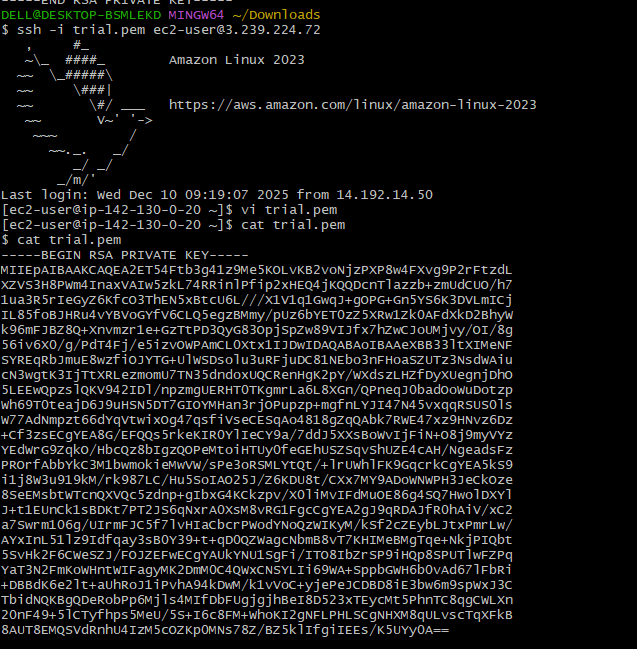
* Copied key to public server
* Changed permission:

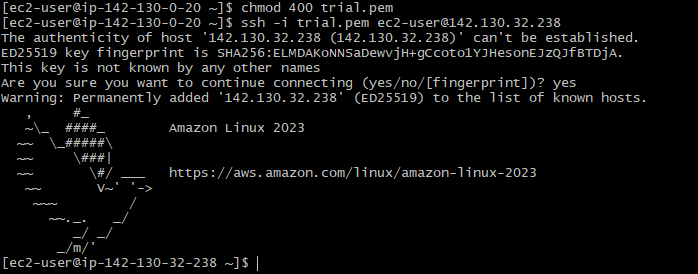
chmod 400 trial.pem

* Connected to private EC2 using:

ssh -i trial.pem ec2-user@<private-ip>

****

****

****

### **Apache Setup on Private Instance**

yum install httpd -y

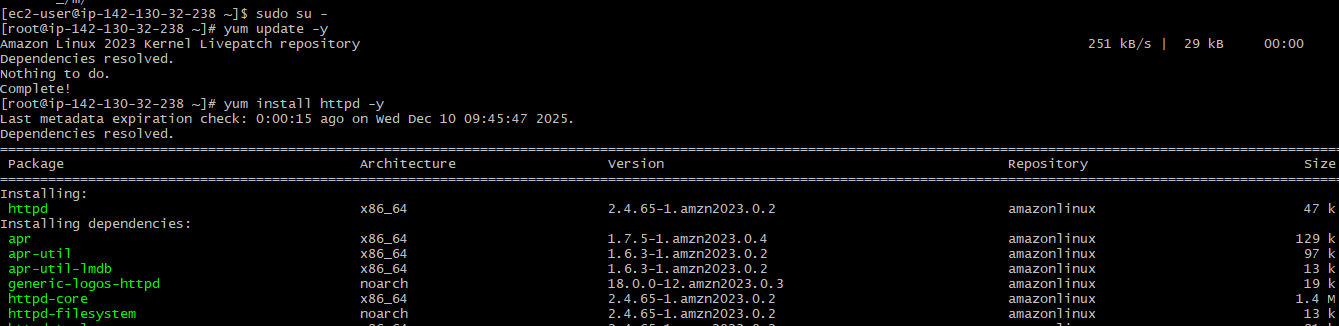
vi /var/www/html/index.html

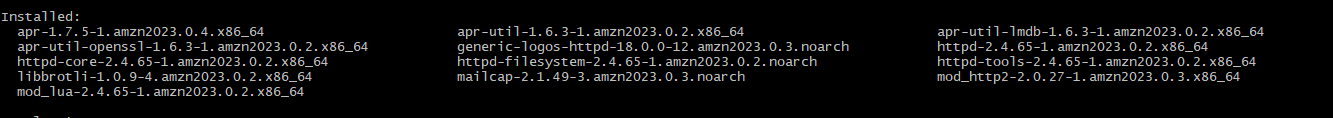
systemctl start httpd

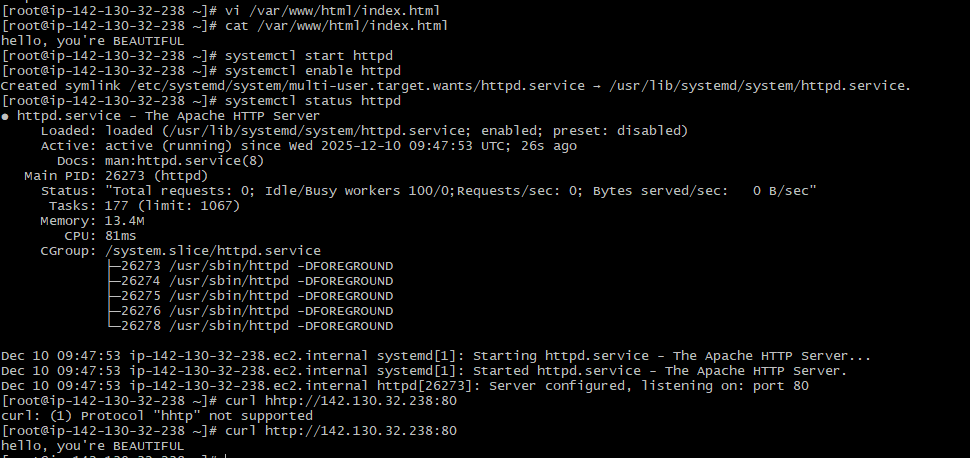
systemctl enable httpd

curl http://<private-ip>

* Output verified using curl command internally.

****

****

****

**Validation Steps**

| **Validation Item** | **Status** |
| --- | --- |
| Public EC2 reachable via SSH | ✅ Success |
| Web page accessible via browser | ✅ Success |
| Private EC2 reachable via SSH from Public EC2 | ✅ Success |
| Apache running on private instance | ✅ Success |
| NAT gateway allowing outbound access | ✅ Verified |
| Route tables properly associated | ✅ Verified |

## ****Issues Faced****

1. **Permission denied for PEM file**
   * Resolved using:
   * chmod 400 trial.pem
2. **Private instance not accessible directly**
   * Reason: No public IP
   * Solution: Accessed via Public EC2 (Jump Server)
3. **HTTP not accessible initially**
   * Security Group did not allow port 80
   * Fixed by updating inbound rules.

## ****Conclusion****

* Successfully built a **secure AWS network architecture** with public & private subnets.
* Deployed **web servers** on both instances.
* Configured **NAT Gateway for private subnet internet access**.
* Verified internal and external connectivity successfully.

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

# **Task Title**

**Setup Application Load Balancer (ALB) With EC2 Instances Hosting Apache Web Server**

# **Objective**

To deploy two EC2 instances running Apache HTTP server, register them in a Target Group, and configure an Application Load Balancer (ALB) to distribute traffic across them. Validate the setup by accessing the ALB DNS name and confirming that the webpage loads correctly.

# **Prerequisites**

* AWS Account with necessary IAM permissions
* Basic understanding of EC2, Target Groups, and Load Balancers
* VPC with at least **2 public subnets**
* SSH key pair for connecting to EC2
* Security Groups allowing:
  + SSH (22) from your IP
  + HTTP (80) from ALB + public access

### **Launch Two EC2 Instances**

* Amazon Linux 2 AMI
* t2.micro
* Select your key pair
* Place them in public and private subnets
* Assign a security group allowing:
  + SSH (22)
  + HTTP (80)

### **Install and Configure Apache on Each Instance**

SSH into each EC2 instance:

sudo su

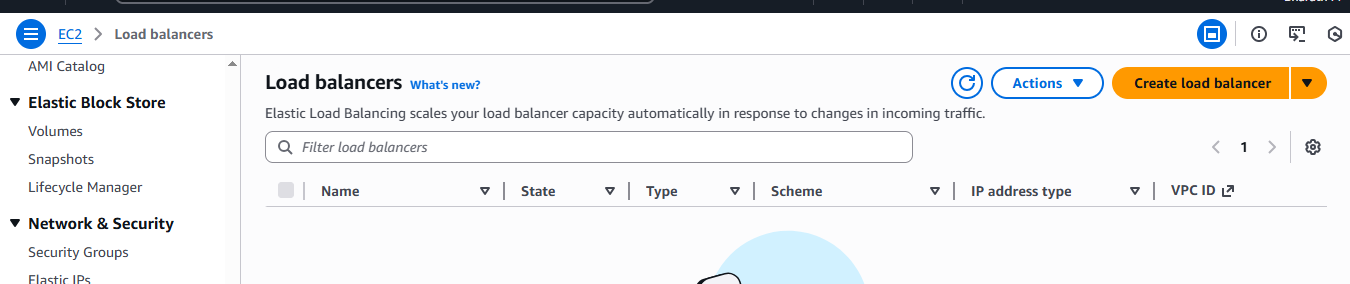
yum install httpd -y

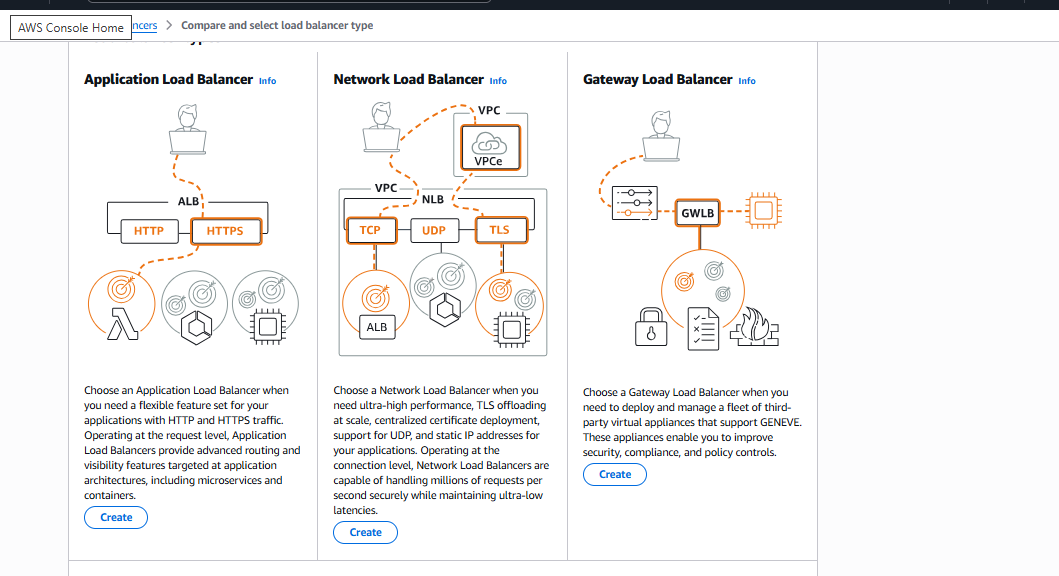
systemctl start httpd

systemctl enable httpd

Create a custom index.html:

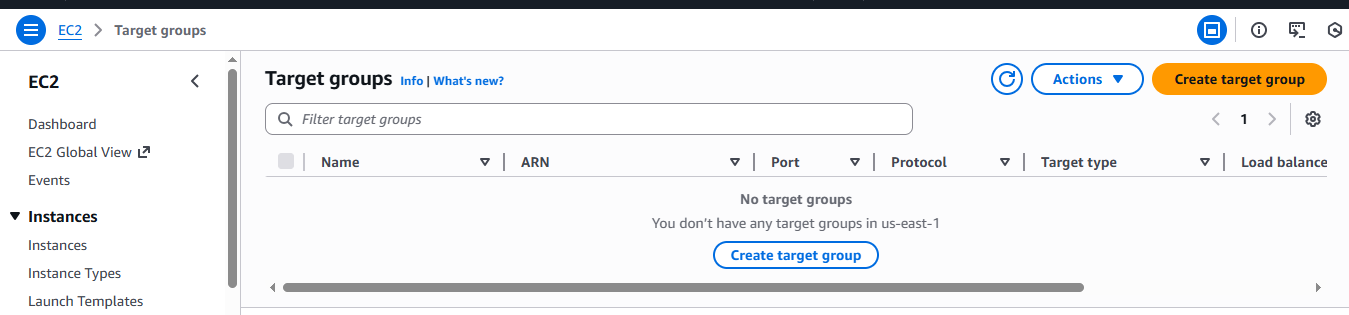
echo "WELCOME TO INDIA" > /var/www/html/index.html

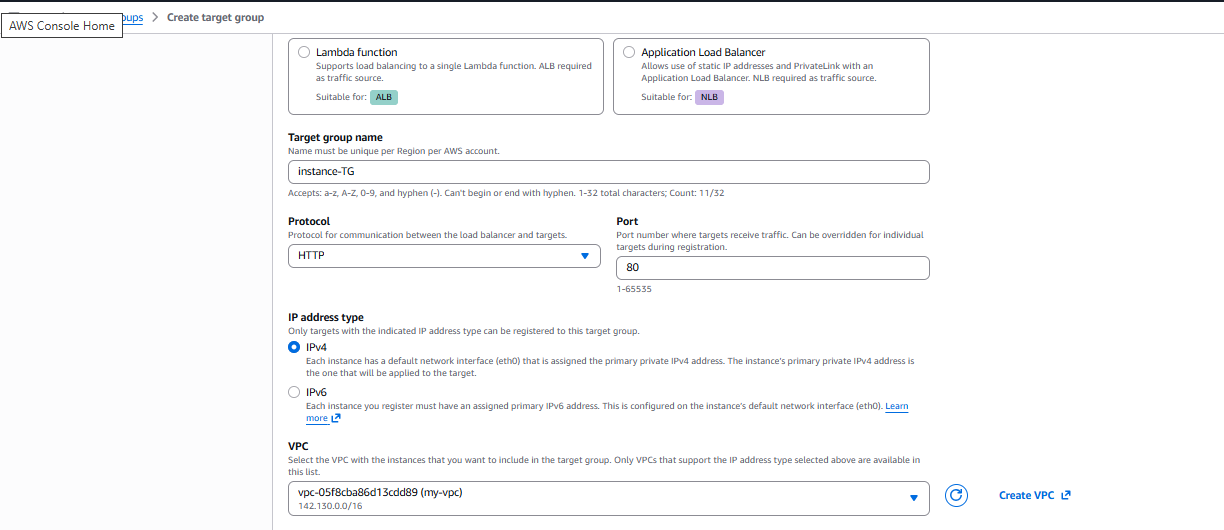
****

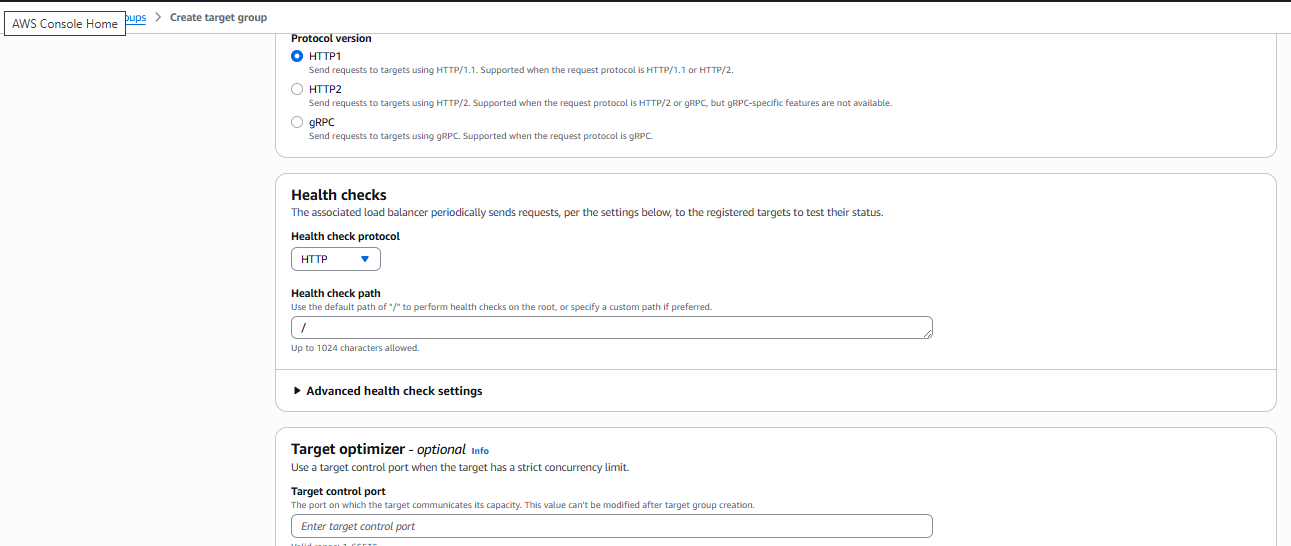
****

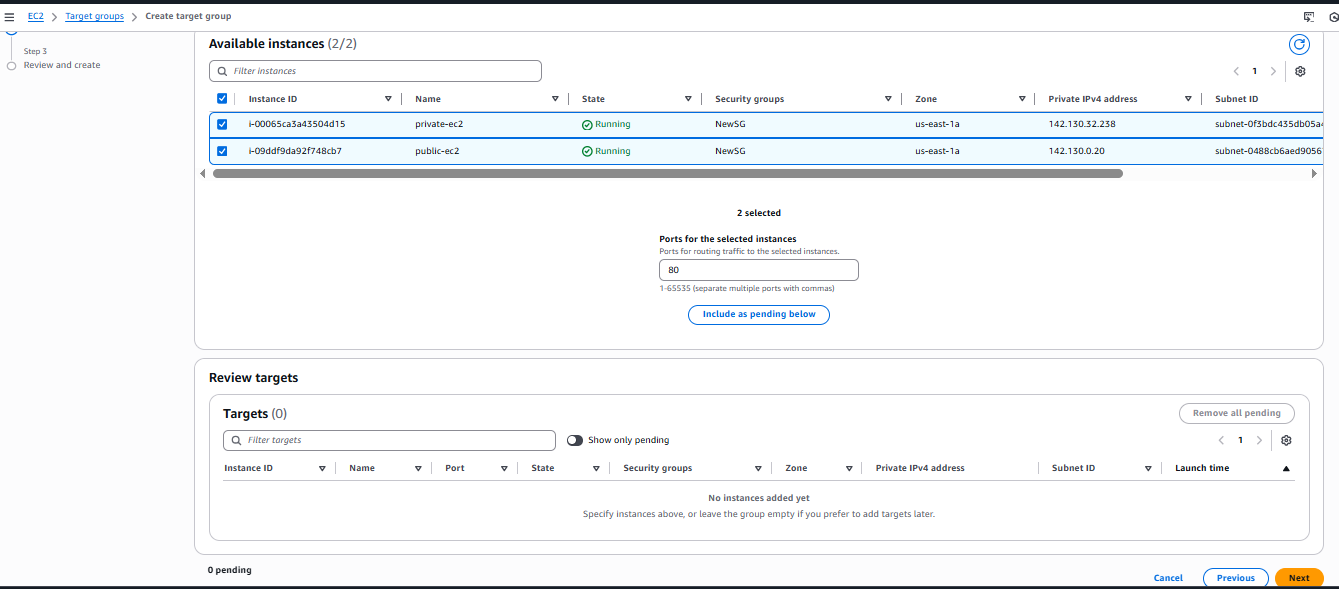
### **Create a Target Group**

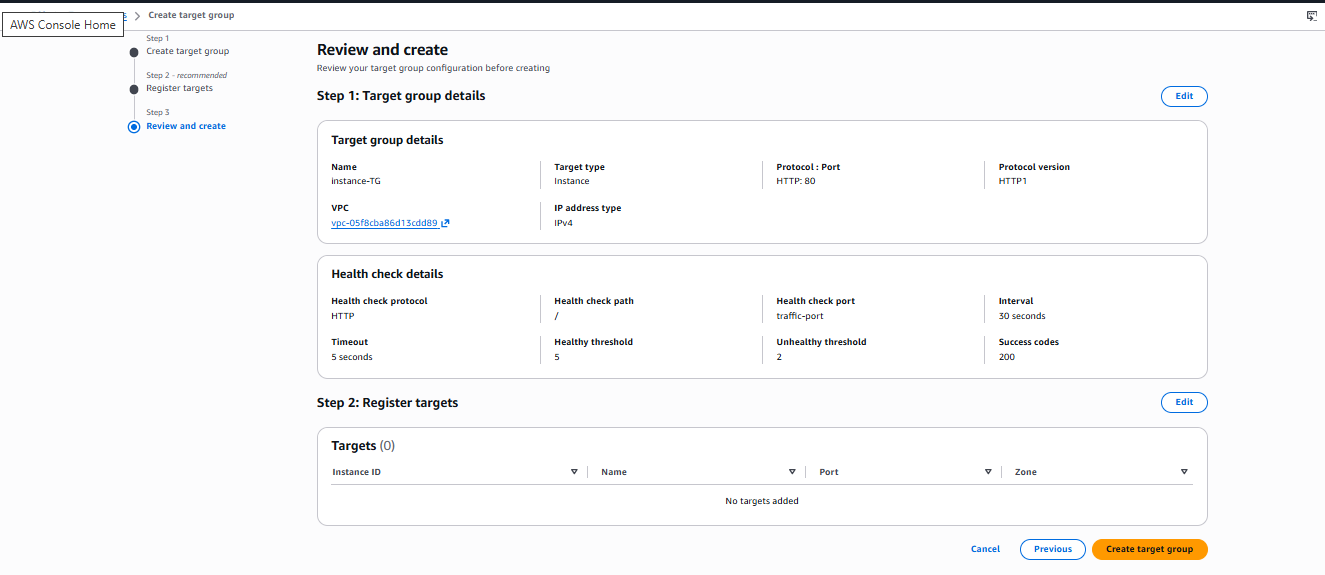
* Go to EC2 → Target Groups
* Create **HTTP Target Group** (instance type)
* Register both EC2 instances
* Ensure health check path is /
* Verify that targets become **healthy**

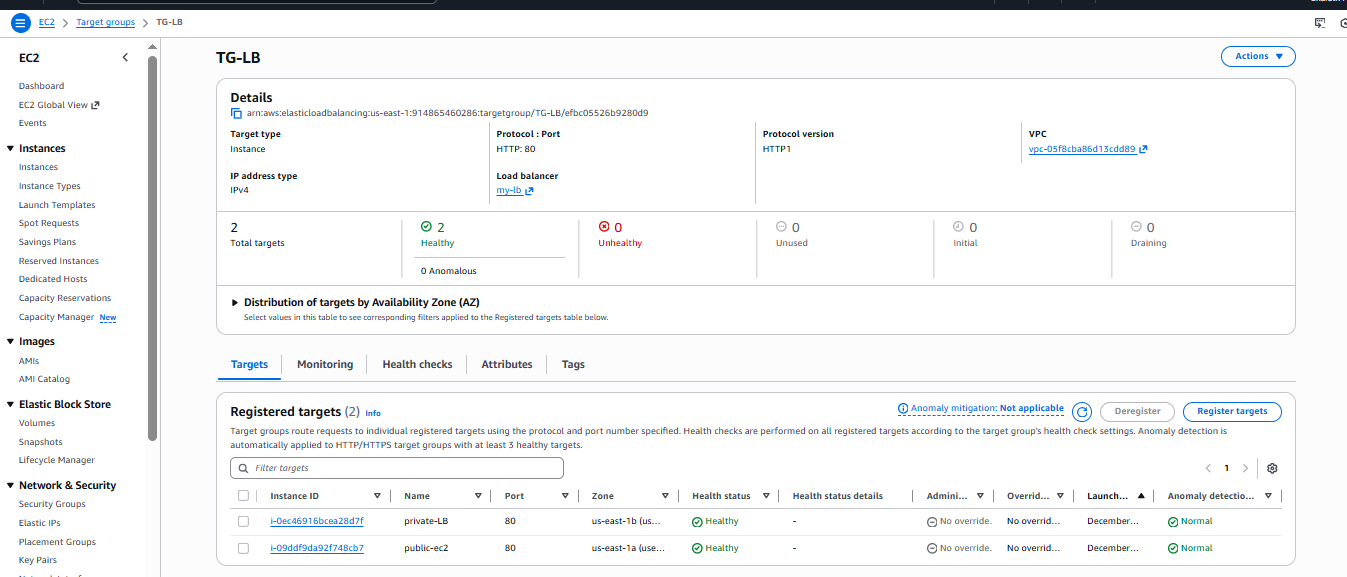
****

****

****

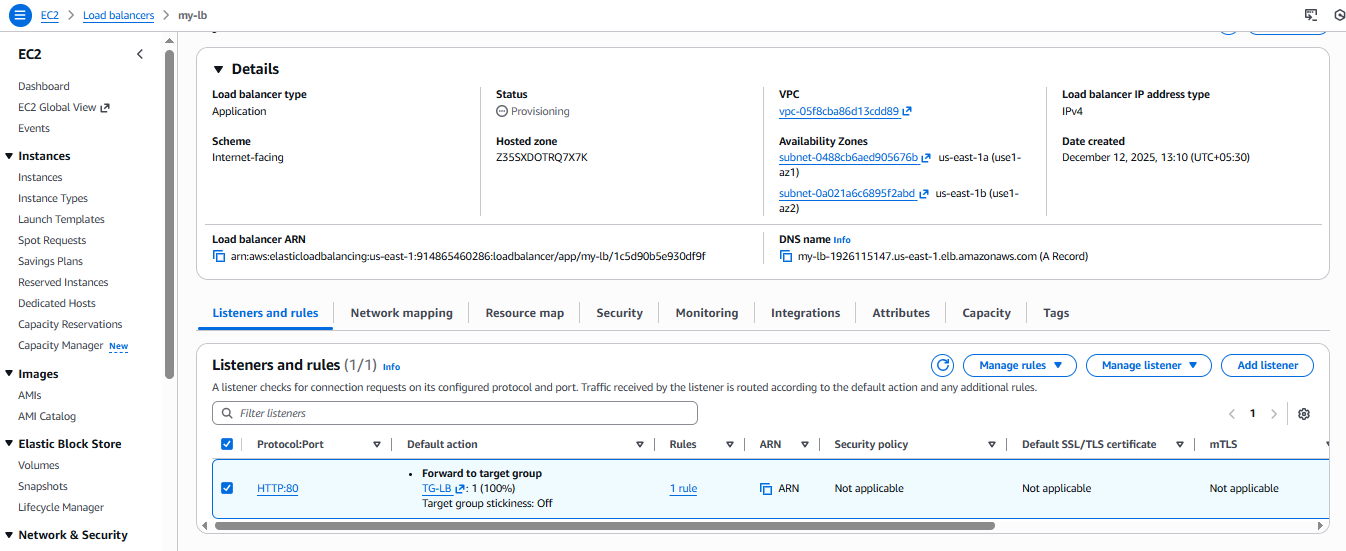
****

****

****

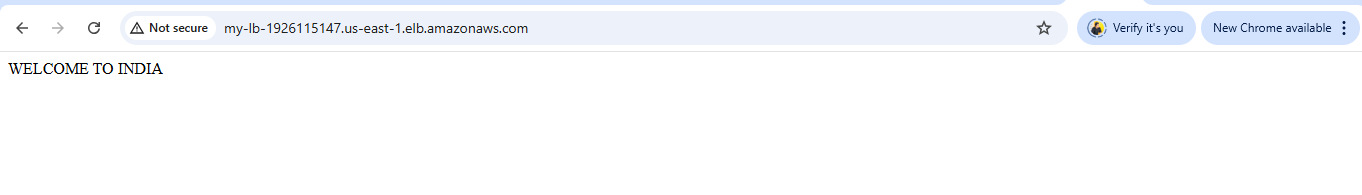
### **Create an Application Load Balancer (ALB)**

* Choose **Application Load Balancer**
* Internet-facing
* Select at least 2 public subnets
* Assign SG allowing HTTP (80)
* Add listener: HTTP (80) → Forward to your target group

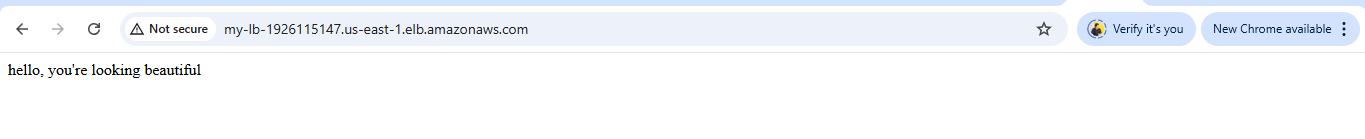
****

### **Validate ALB**

* Copy Load Balancer DNS URL
* Paste into browser
* You should see:  
  **WELCOME TO INDIA**
* Refresh multiple times to verify load balancing between instances

****

The EC2 instance returned the page containing the message:  
**"hello, you're looking beautiful"**

****

# **Validation Steps**

* ALB status = Active
* Target Group health status = healthy for both instances
* Apache service running on both instances
* Accessing ALB DNS shows the expected webpage
* Security Groups correctly configured to allow traffic

# **Issues Faced** (Based on screenshots)

Here are possible issues you may have encountered:

1. **Apache not running on one instance**
   * Solved by starting and enabling the service.
2. **Target not showing “Healthy”**
   * Usually caused by incorrect security group or Apache not running.
   * Fixed after ensuring both instances allow HTTP and Apache is running.
3. **ALB unable to serve traffic**
   * Occurs if subnets were not public or security groups blocked HTTP.
   * Resolved by adding proper inbound rules.
4. **Permission denied during SSH**
   * Usually wrong key permissions.
   * Solved by running chmod 400 key.pem.
5. **Store application load balancer logs in S3.**

# **Task Title**

**Enable Application Load Balancer (ALB) Access Logs to S3.**

# **Objective**

To deploy an Application Load Balancer (ALB), configure public/private subnets with a NAT Gateway for outbound internet access, and enable ALB access logging by delivering logs to an S3 bucket.

# **Prerequisites**

* An AWS account with necessary IAM permissions (EC2, VPC, S3, IAM policies, ELB).
* A VPC with:
  + **Public subnets** (for ALB)
  + **Private subnets** (for EC2 instances, if used)
* An **Internet Gateway** attached to the VPC.
* Basic knowledge of EC2, VPC routing, and S3.

## ****Create an S3 Bucket for ALB Access Logs****

1. Go to **S3 → Create bucket**.
2. Name: my-unique-application-logs-us-east-1
3. Disable public access block **only if you know what you’re doing** (recommended to leave ON).
4. Create the bucket.

## ****Add Required Bucket Policy****

AWS requires a specific policy so ALB can deliver logs.

Add this policy under **Permissions → Bucket Policy**:

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": {

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

},

"Action": "s3:PutObject",

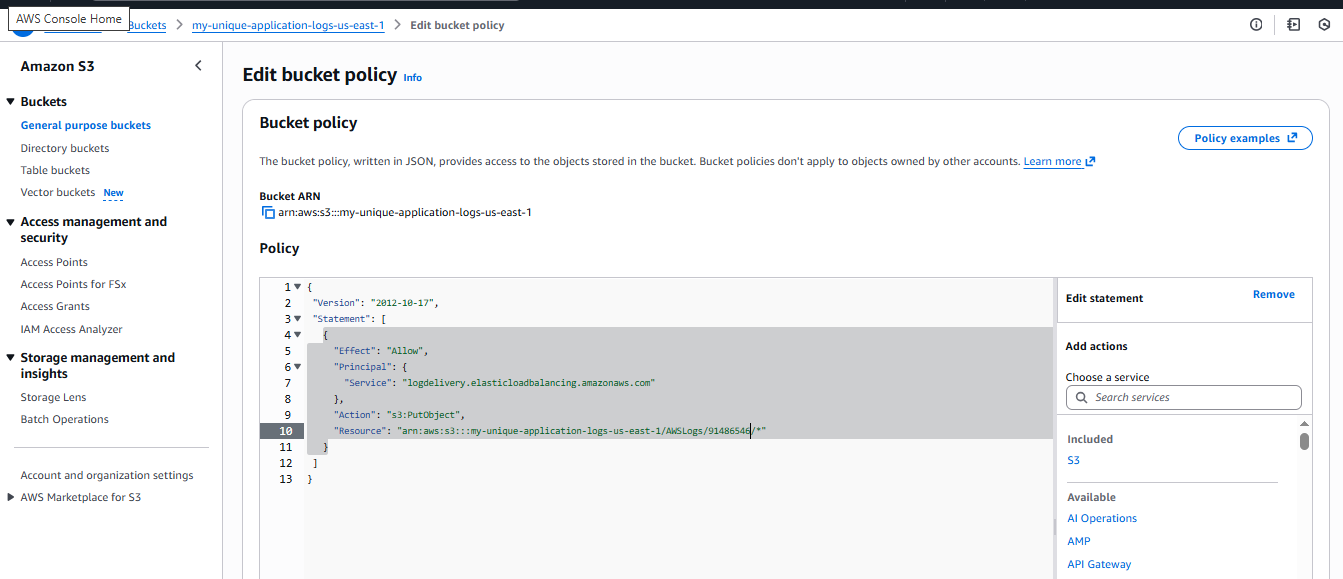
"Resource": "arn:aws:s3:::my-unique-application-logs-us-east-1/AWSLogs/914865460286/\*"

}

]

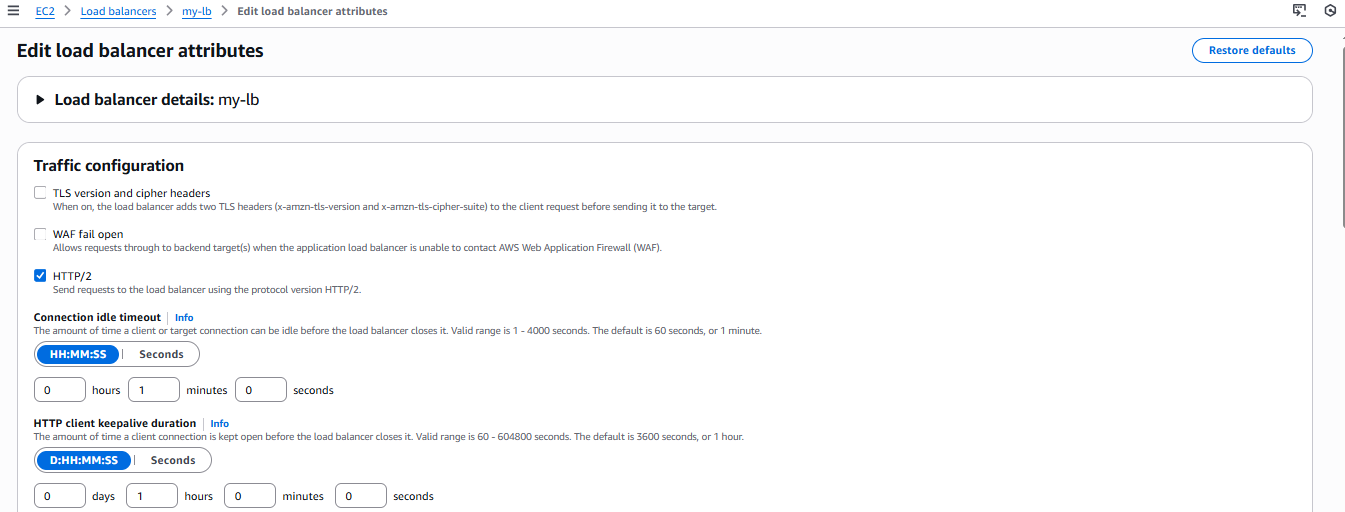
}

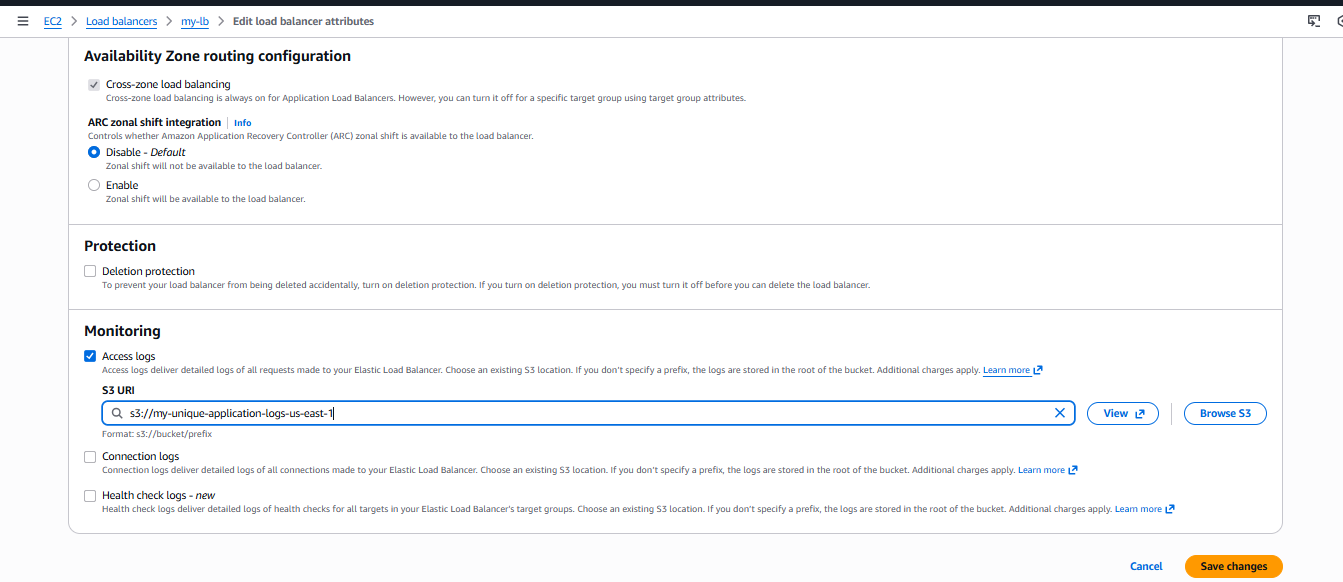
Confirm correct:  
✔ Bucket ARN used  
✔ Your AWS Account ID is correct  
✔ Prefix AWSLogs/<AccountID>/ is present

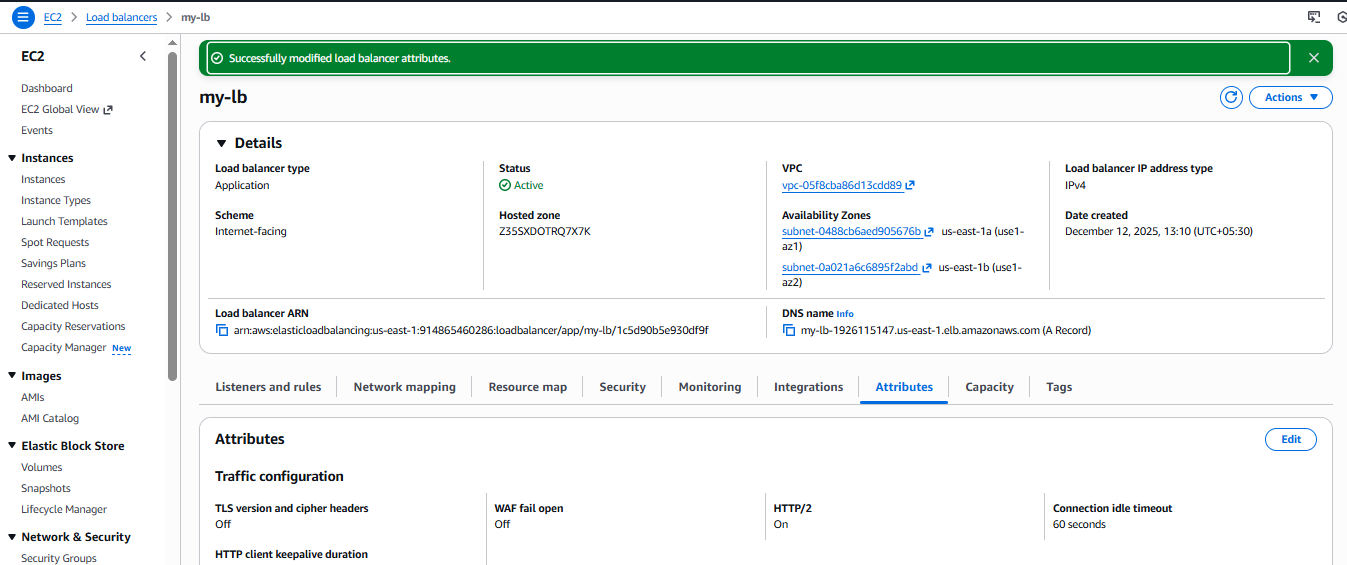
****

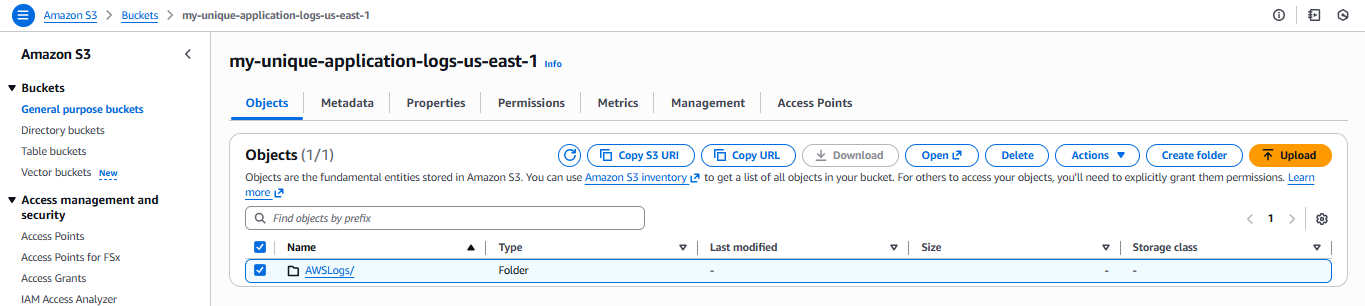
## ****Enable ALB Access Logging****

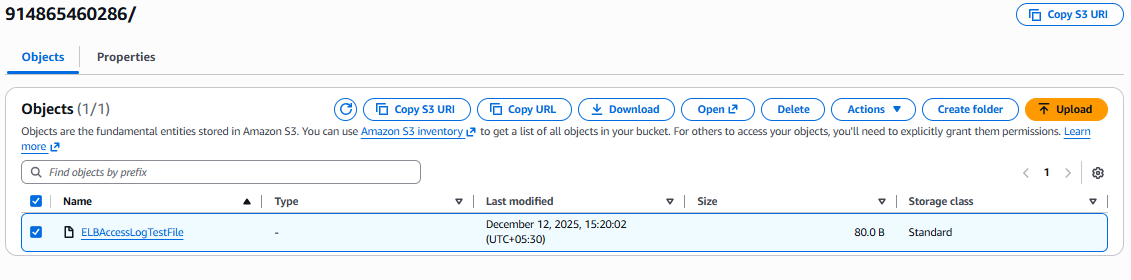
1. Open **EC2 → Load Balancers → my-lb**
2. Go to **Attributes** → Edit
3. Under **Monitoring**:
   * Enable **Access logs**
   * Provide S3 URI:
   * s3://my-unique-application-logs-us-east-1
4. Save changes.
5. You will see: **Successfully modified load balancer attributes.**

****

****

****

****

****

# **Validation Steps**

## ****1. Confirm ALB Access Logs are Delivered****

1. Go to **S3 → my-unique-application-logs-us-east-1**
2. Navigate to:
3. AWSLogs/914865460286/elasticloadbalancing/us-east-1/
4. Confirm folder structure exists.
5. You should see a test log file like:  
   ELBAccessLogTestFile

This indicates **logs are correctly configured**.

## ****2. Validate Routing****

From any EC2 inside private subnet:

* Run curl https://amazon.com
* If successful → NAT Gateway routing is correct.

From public browser:

* Access ALB DNS → response is correct and healthy.

# **Issues Faced**

During the setup, the following issues can typically occur (add/remove based on your environment):

### **1. NAT Gateway stuck in “Pending”**

* Newly created NAT Gateways take time to receive an Elastic IP.

### **2. “Access Denied” delivering ALB logs**

Resolved by applying correct bucket policy.

### **3. Logs not appearing in S3**

* Prefix incorrect
* Wrong account ID
* Incorrect bucket policy
* ALB attributes not saved
* Region mismatch

### **4. ALB returns 503 or 504**

* Target group health checks misconfigured
* Instance security group not allowing ALB traffic

**10.Store the VPC flow logs in a CloudWatch log group.**

# **Task Title**

**Configure and Validate VPC Flow Logs for ALB VPC**

# **Objective**

To enable **VPC Flow Logs** for my-vpc and deliver logs to **CloudWatch Logs** to monitor, troubleshoot, and analyze network traffic associated with ALB and other VPC components.

# **Prerequisites**

* Existing VPC (my-vpc, ID: vpc-05f8cba86d13cdb89)
* Access to AWS Console
* IAM permissions to:
  + Create VPC Flow Logs
  + Create IAM roles for Flow Logs delivery
  + Create/modify CloudWatch Log Groups
* CloudWatch Logs enabled in the region

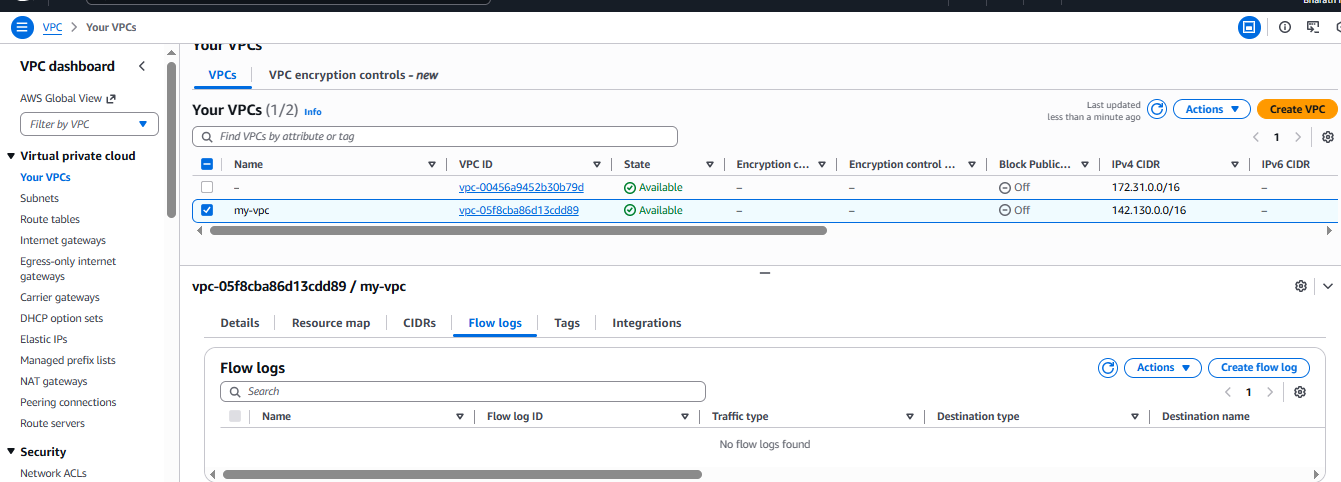
# **Step-by-Step Implementation**

### **1. Navigate to VPC Console**

* Go to **VPC → Your VPCs**
* Select the VPC my-vpc.

### **2. Create a New Flow Log**

* Open **Flow logs** tab
* Click **Create flow log**

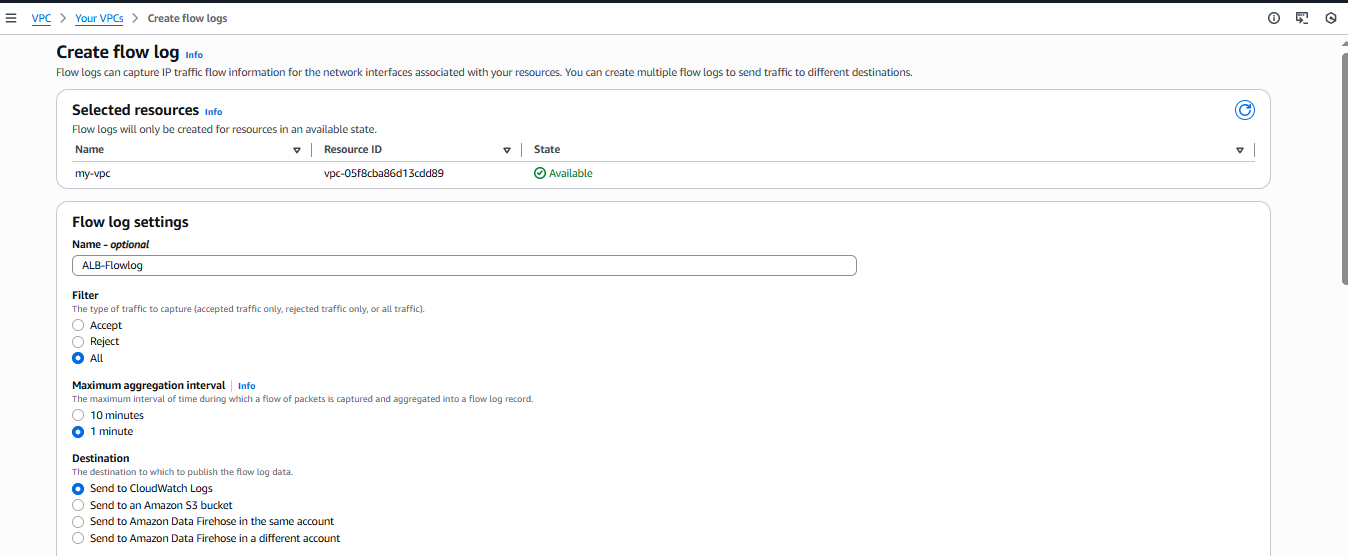
****

### **3. Configure Flow Log Settings**

* **Name:** ALB-Flowlog
* **Filter:** All (captures both accepted and rejected traffic)
* **Maximum Aggregation Interval:** 1 minute

### **4. Configure Destination**

* **Destination type:** CloudWatch Logs
* **Log group:** ALB-logs  
  (Created automatically)
* **Service Access:** Create a new IAM role
  + IAM Role Created:  
    VPCFlowLogs-Cloudwatch-1765534149736

****

### **5. Log Format**

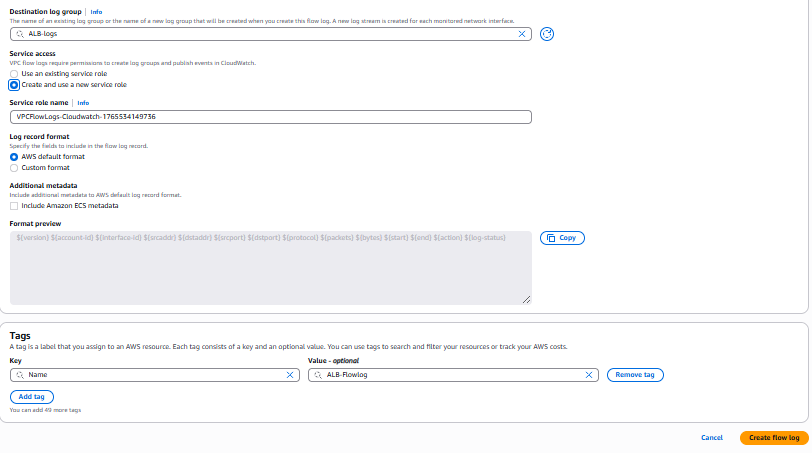
* Choose **AWS default format**

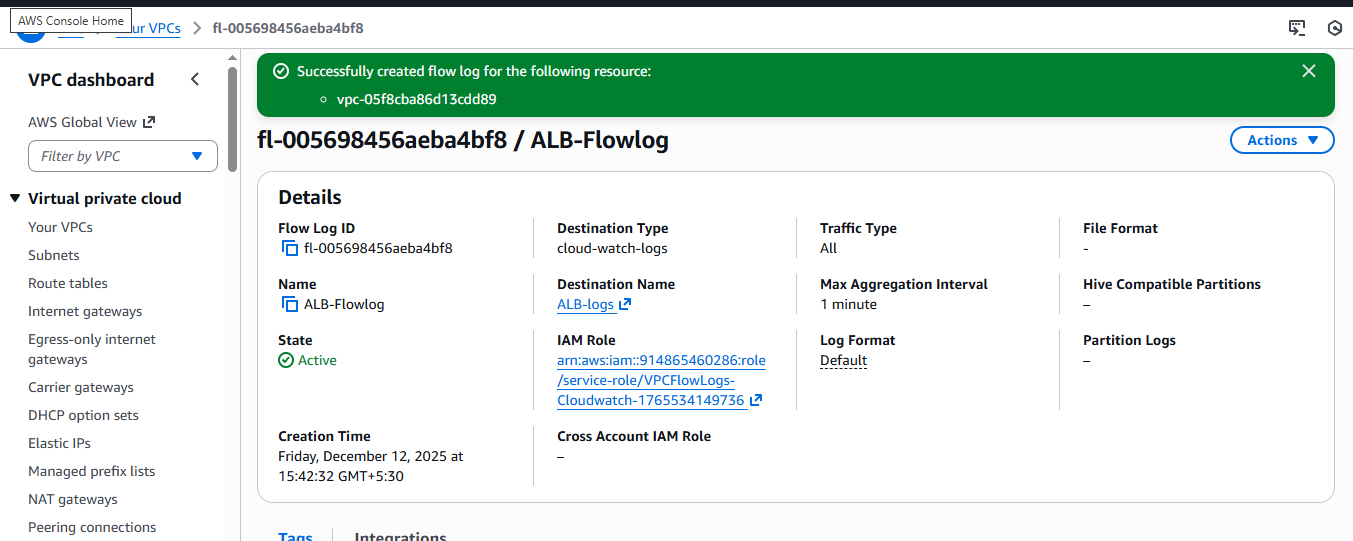
### **6. Add Tags**

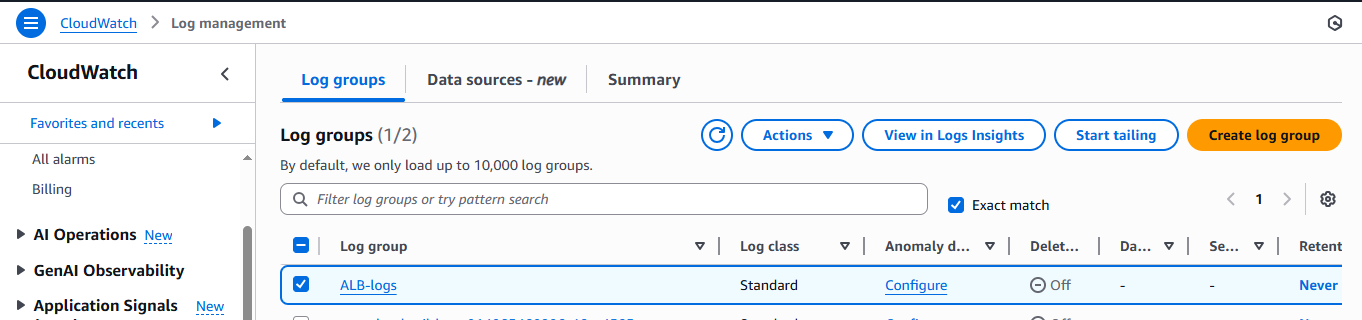
* Name: ALB-Flowlog

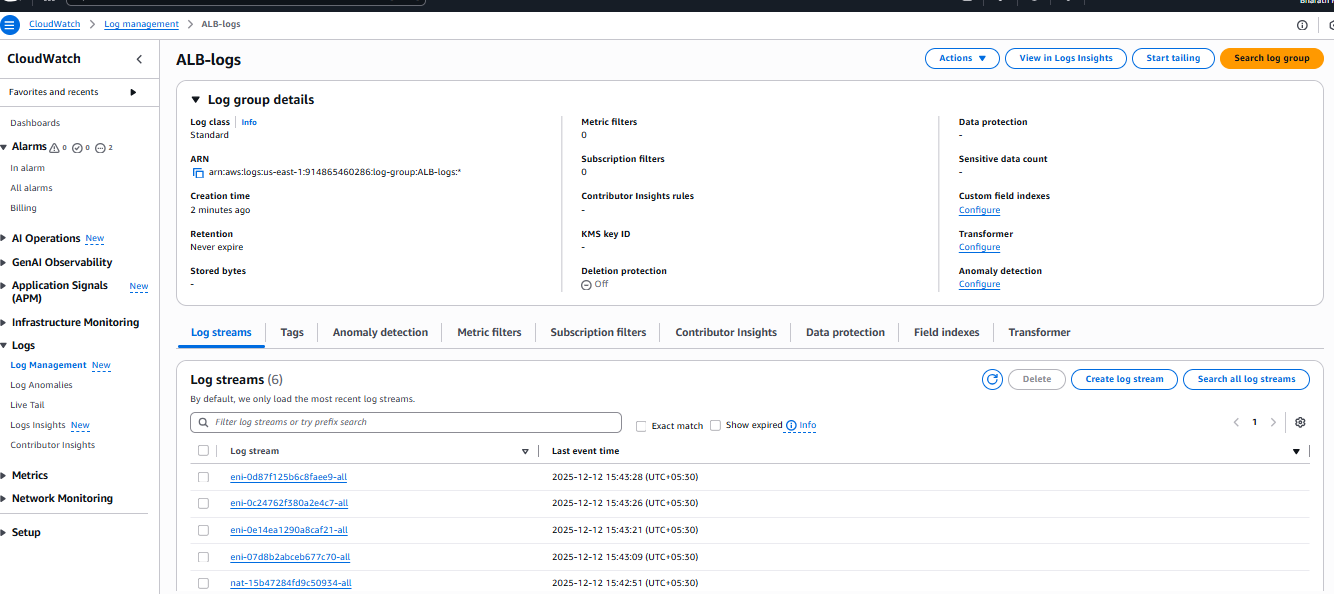
### **7. Create Flow Log**

* Result: Flow log created successfully  
  Flow Log ID: fl-005698456aeba4bf8
* Status shows **Active**

****

****

****

****

# **Validation Steps**

### **1. Validate in VPC Console**

* Flow Log state = **Active**

### **2. Validate CloudWatch Log Group**

* Go to CloudWatch → Log groups
* Confirm ALB-logs exists

### **3. Verify Log Streams**

* Inside ALB-logs
* Multiple log streams generated
* Timestamps show ongoing ingestion  
  → Confirms flow logs are successfully delivered

# **Issues Faced**

No issues observed.  
Flow log creation succeeded, IAM role auto-created, and logs are actively visible in CloudWatch.

# **Conclusion**

The VPC Flow Logs for **my-vpc** were successfully configured and validated. Logs are being delivered to the **CloudWatch Logs** group ALB-logs, and multiple log streams confirm active ingestion of network traffic data. This setup provides enhanced visibility into traffic flowing through the VPC, enabling improved monitoring, troubleshooting, and security analysis for the Application Load Balancer and associated resources. The implementation met all objectives without encountering issues.

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

## ****Task Title****

**Implementing VPC Flow Logs, ALB Logs, and Custom CloudWatch Metrics with Monitoring Dashboard**

## ****Objective****

The objective of this task is to enable complete observability of an application hosted on EC2 and fronted by an Application Load Balancer (ALB).  
This includes configuring:

* **VPC Flow Logs** to monitor network traffic
* **ALB access logs** for load balancer request visibility
* **Custom CloudWatch Metrics** (httpd\_status) for application-level health monitoring
* **CloudWatch Dashboards** to visualize logs and metrics in a single pane

This setup provides end-to-end monitoring, better troubleshooting capability, and deep visibility into system performance and health.

## ****Prerequisites****

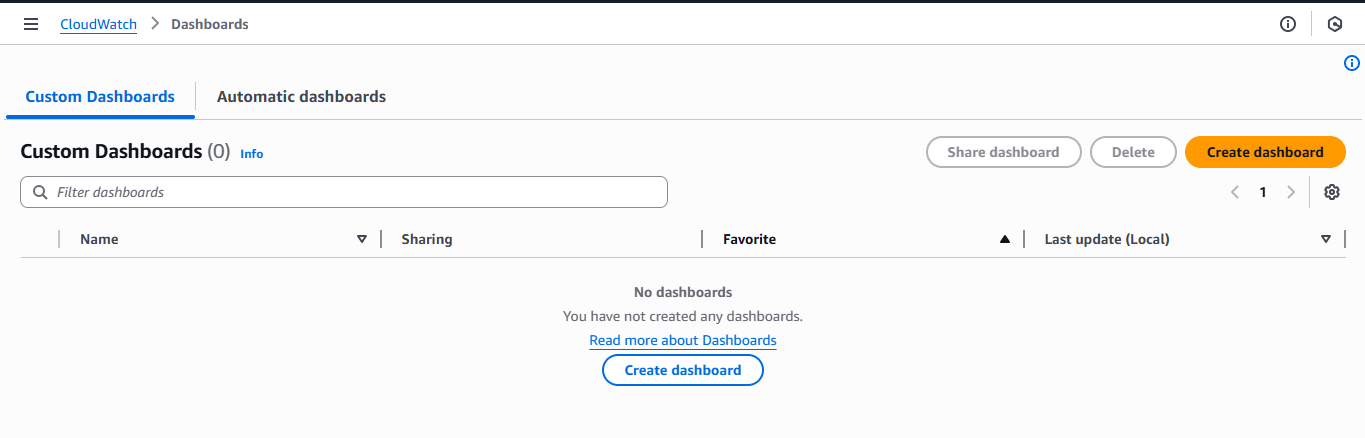
* An AWS account with permissions for:
  + VPC management
  + CloudWatch
  + IAM
  + EC2
* Existing infrastructure:
  + One public ALB
  + EC2 instance(s) running httpd or another web service
* AWS CLI installed on EC2
* Basic understanding of bash scripting and cron jobs

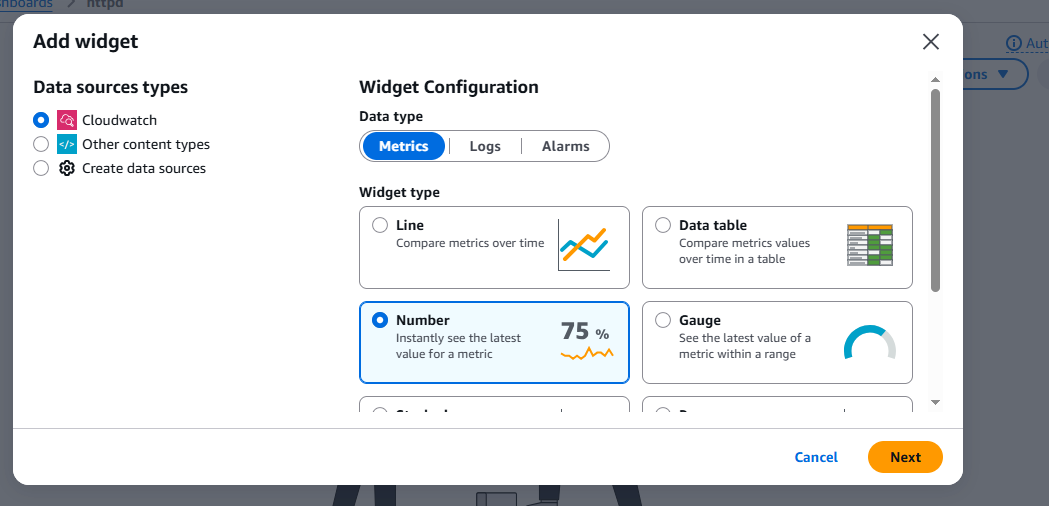
### **Verify ALB Access Logs**

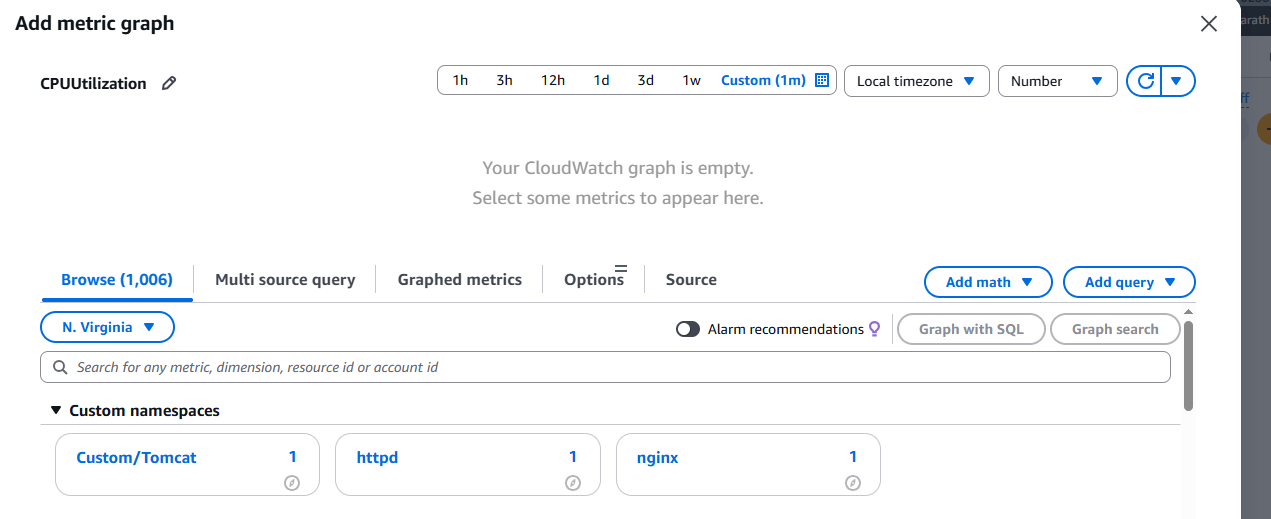
* Check CloudWatch logs for streams automatically generated by ALB
* Confirm new log streams appear when traffic hits the ALB
* Access ALB DNS to verify:
* http://<alb-dns-name>
* Logs appear under ALB-logs.

### **Create a CloudWatch Dashboard**

* Go to **CloudWatch → Dashboards → Create Dashboard**
* Add widgets:
  + **Number Widget** → httpd\_status
  + **Line Widget** → CPUUtilization for EC2 instances
* Add metrics from:
* Custom/httpd
* Save dashboard.

****

****

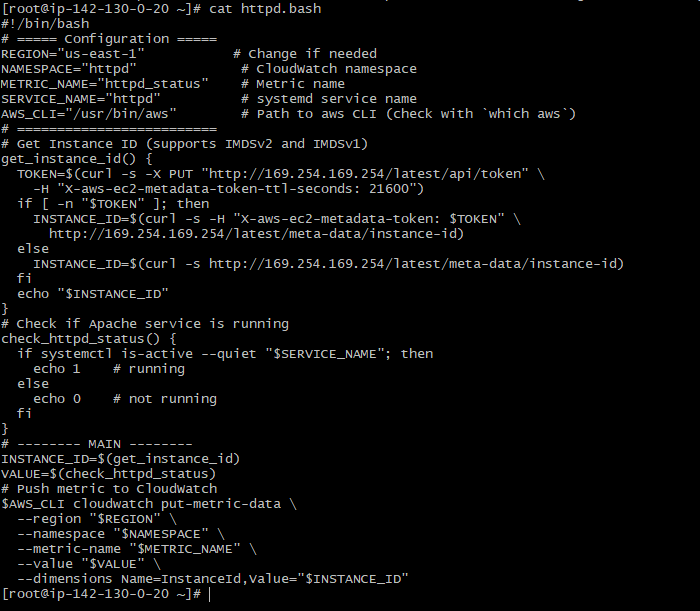
****

### **Create a Custom CloudWatch Metric (httpd\_status)**

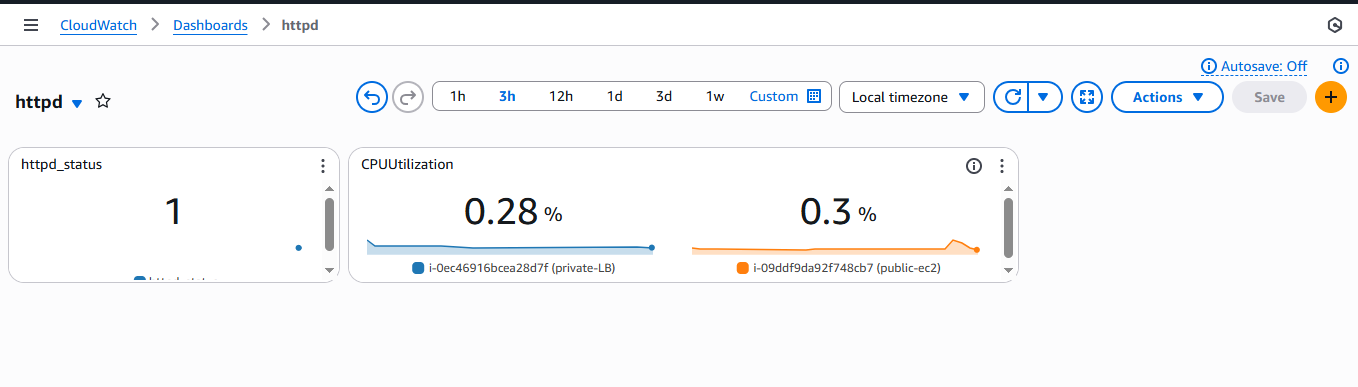
* Create a script (httpd.bash) to check Apache service status:
  + Returns **1** if running
  + Returns **0** if stopped
* Script pushes data to CloudWatch using:
* aws cloudwatch put-metric-data
* Metric namespace: httpd
* Dimension: InstanceId
* Deploy script on EC2 under /home/ec2-user/httpd.bash
* Make executable:
* chmod +x httpd.bash

### **Automate Metric Push with Cron**

* Edit cron:
* crontab -e
* Add entry:
* \* \* \* \* \* /home/ec2-user/httpd.bash >> /tmp/httpd.log 2>&1
* Cron runs every minute and pushes the metric to CloudWatch.

****

****

****

## ****Validation Steps****

* Send traffic through ALB and verify ALB log streams update.
* Check VPC Flow Logs for accepted/rejected entries.
* Verify custom metric:
  + CloudWatch → Metrics → Custom namespace → httpd
  + Value should be 1 for running, 0 for stopped
* Stop Apache to test:
* sudo systemctl stop httpd

CloudWatch metric should change to 0 within 1 minute.

* View CloudWatch Dashboard for real-time monitoring.

## ****Issues Faced****

### **1. Flow Logs Not Appearing**

* Solution: Correct IAM role permissions and ensure log group exists.

### **2. Metric Not Updating**

* Cause: Cron not running or script not executable
* Fix:
* chmod +x httpd.bash
* grep CRON /var/log/cron

### **3. ALB Logs Delayed**

* ALB logs can take 2–5 minutes to deliver
* Verified by generating manual traffic

### **4. Metadata Token Issues**

* IMDSv2 token required for instance ID
* Fixed by using correct curl command with token header.

## ****Conclusion****

This project successfully implemented a complete monitoring and observability system using AWS CloudWatch, VPC Flow Logs, and ALB logs. Additionally, a custom metric was created to monitor Apache service health, automated using a cron job, and visualized on a CloudWatch Dashboard.

The solution provides real-time visibility into network activity, load balancer traffic, EC2 performance, and application-level health. This improves troubleshooting efficiency, operational awareness, and overall reliability of the environment.

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

# **Task Title**

**Create Launch Template and Auto Scaling Group (ASG) with Load Balancer Integration**

# **Objective**

To create an EC2 Launch Template and configure an Auto Scaling Group (ASG) that automatically launches instances across multiple subnets and registers them with an existing Application Load Balancer (ALB) target group.

# **Prerequisites**

* AWS Account
* VPC with public subnets
* Security Group allowing HTTP (port 80)
* Target Group created for ALB
* ALB already configured and active
* Key Pair created
* IAM permissions for EC2, ASG, ALB

# **Step-by-Step Implementation**

### **1. Create Launch Template**

* Open **EC2 Console → Launch Templates → Create launch template**
* Template name: ASG
* Description: test
* AMI: Amazon Linux 2023
* Instance type: t3.micro
* Key pair: trial
* Security group: NewSG
* Storage: 8 GiB EBS
* Create the template

### **2. Create Auto Scaling Group**

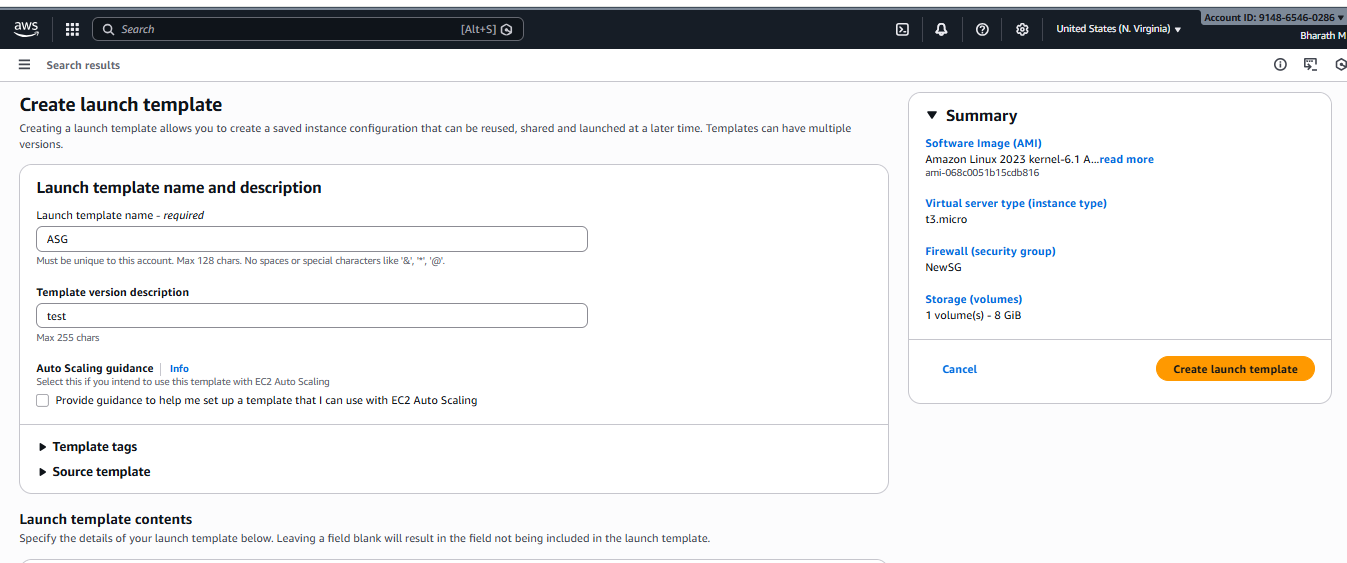
Navigate to: **EC2 → Auto Scaling Groups → Create ASG**

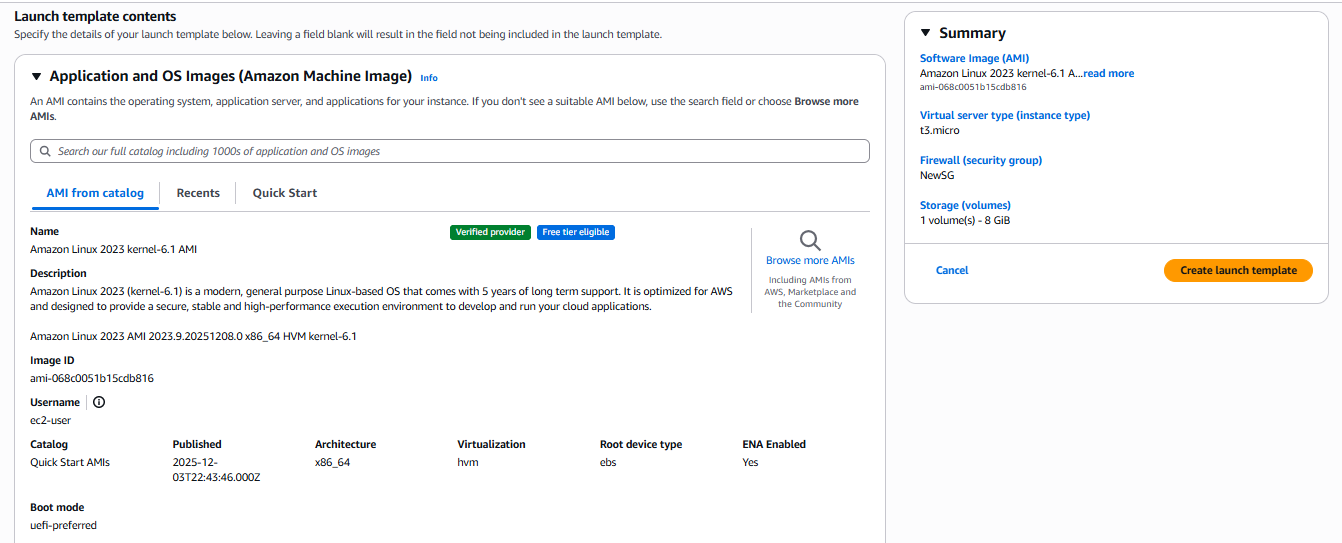
#### **Step 1: Select Launch Template**

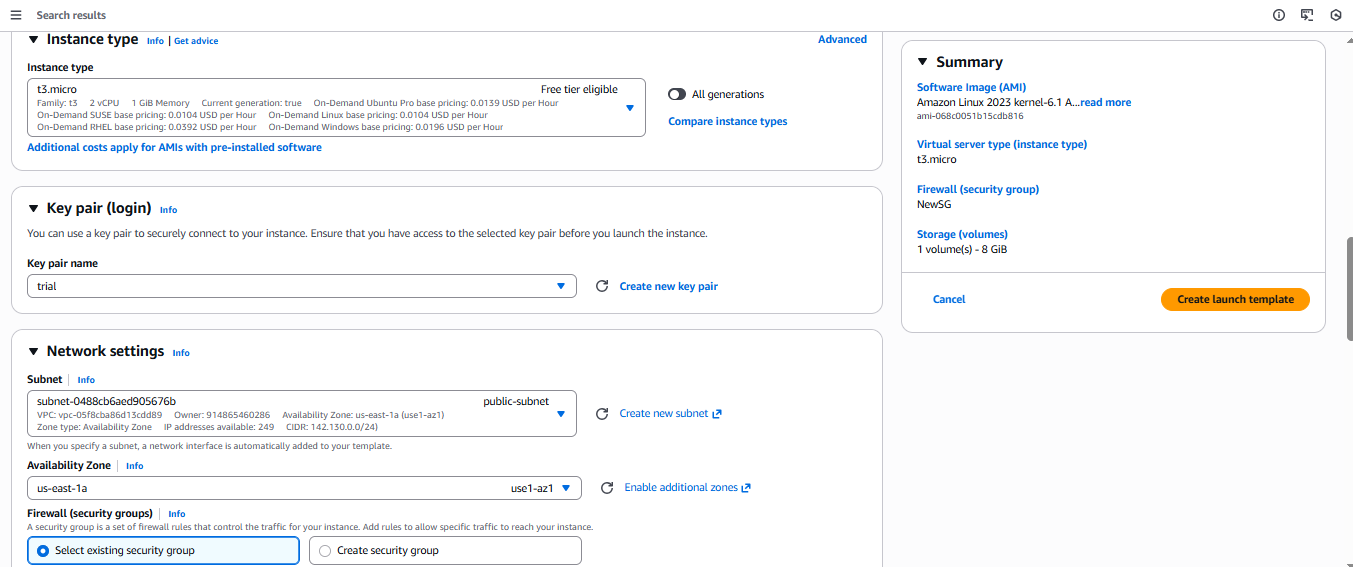
* Choose template: **ASG**
* Version: **Default (1)**

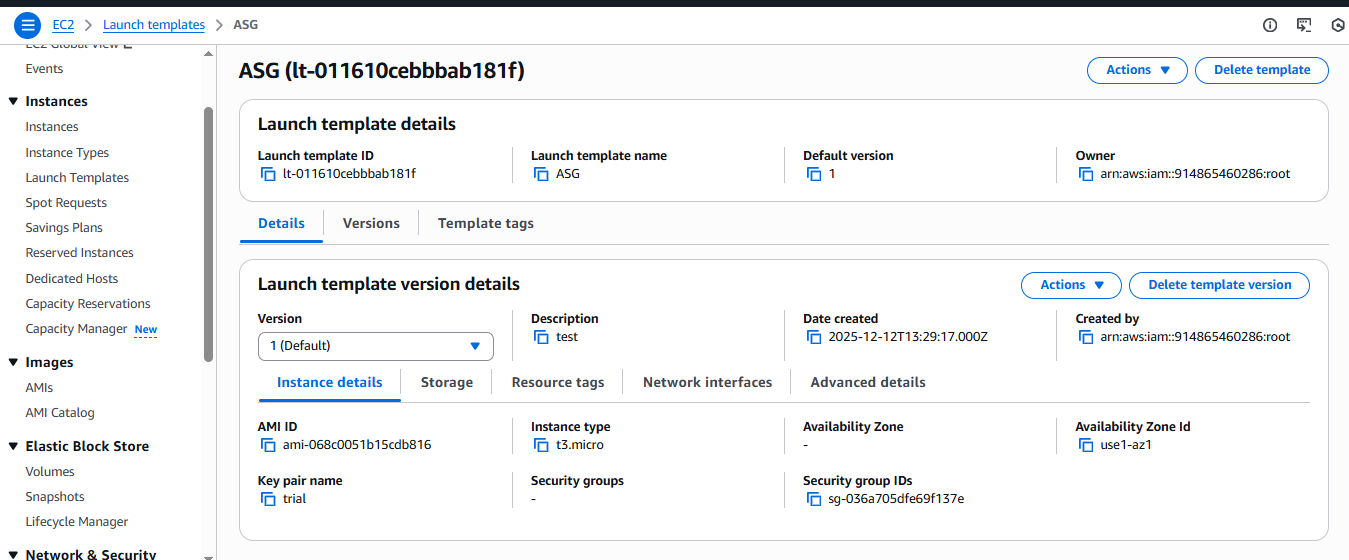
#### **Step 2: Choose Instance Launch Options**

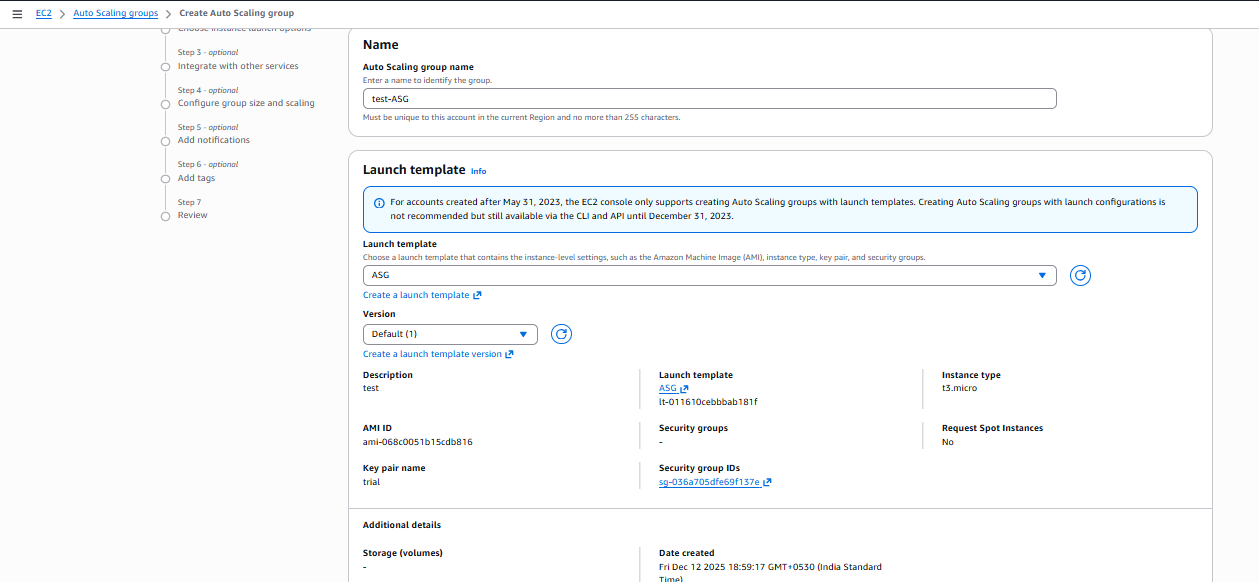
* VPC: Default
* Select two subnets:
  + us-east-1a
  + us-east-1b

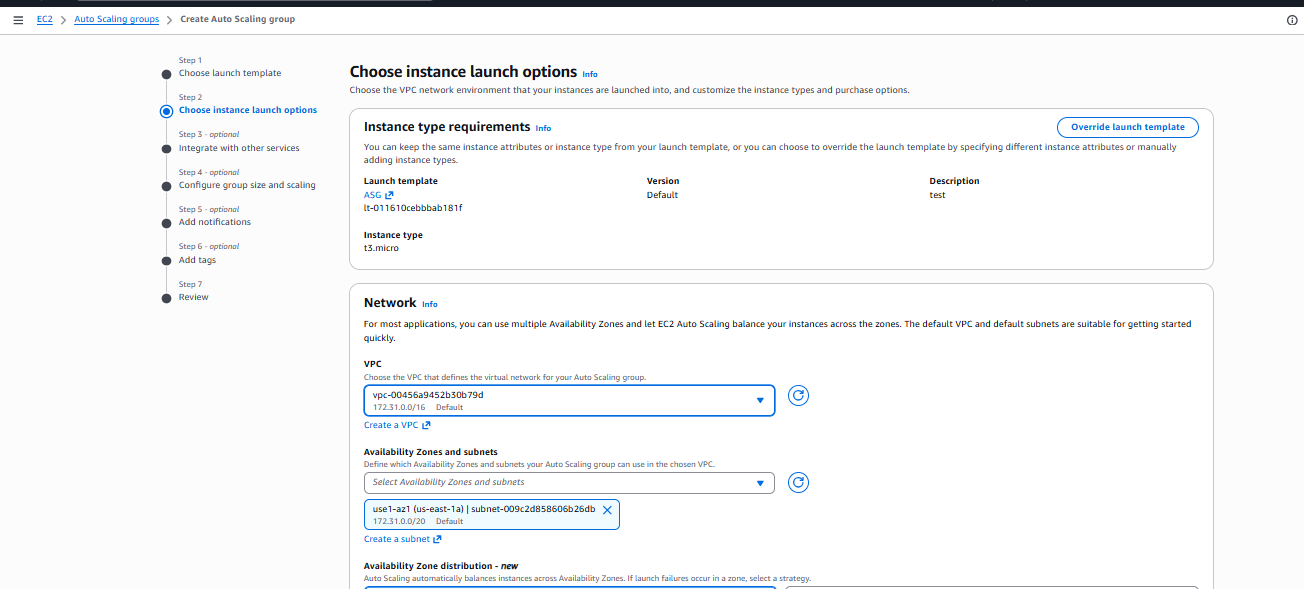
****

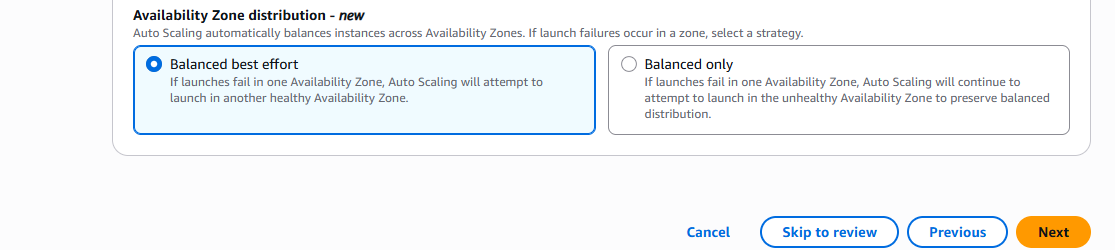
****

****

****

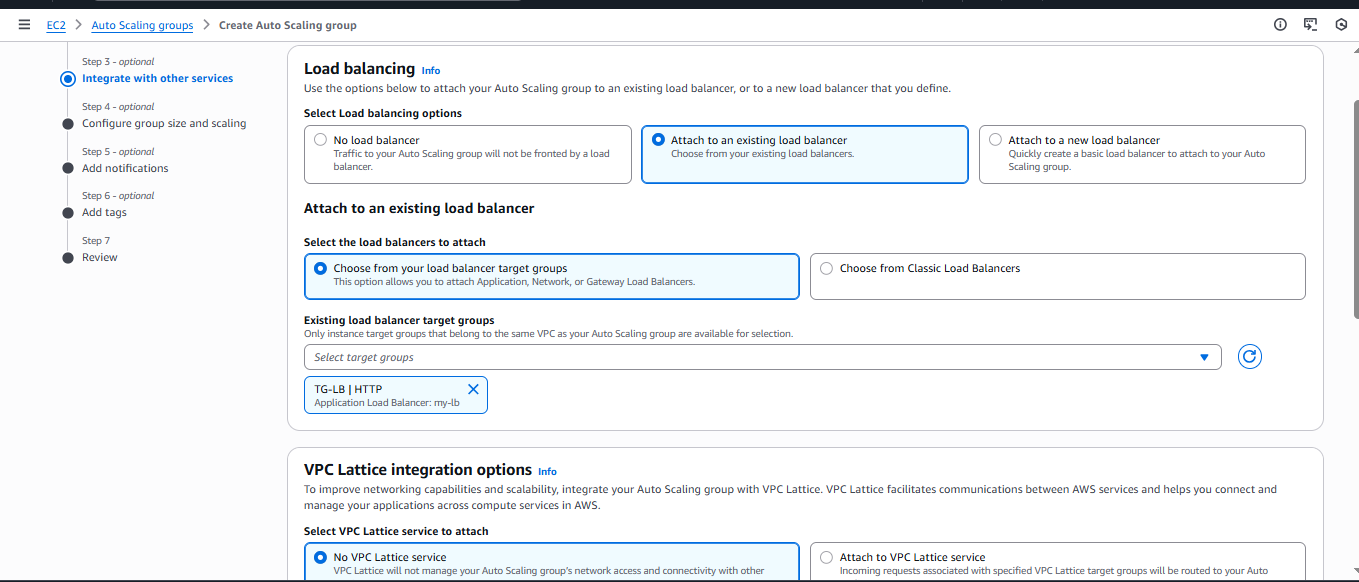
****

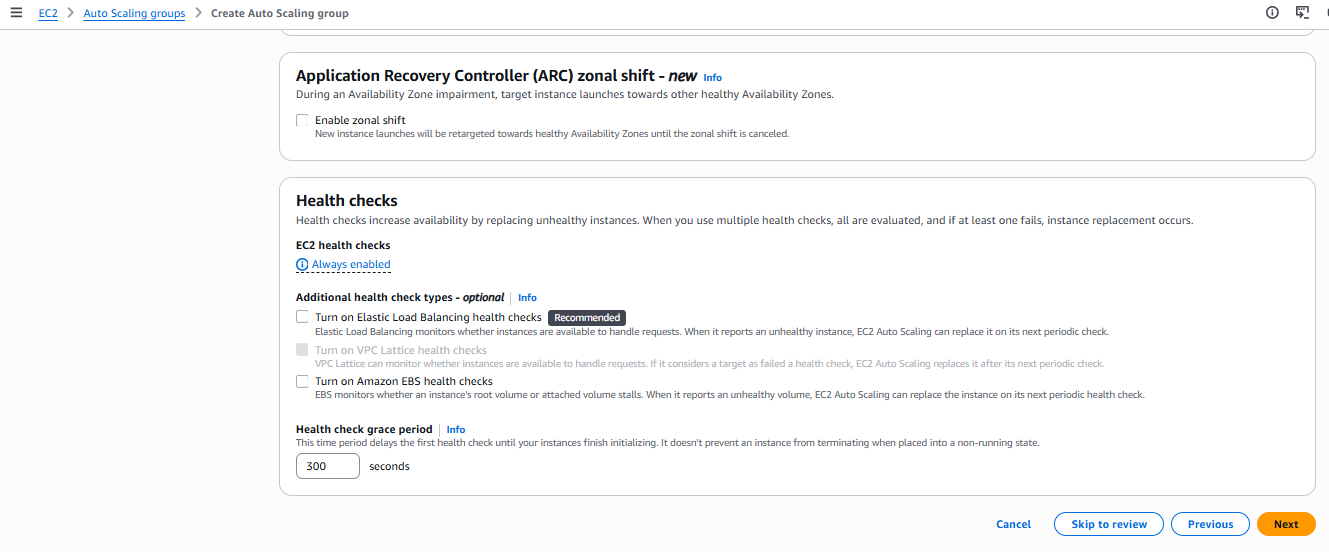
****

****

#### **Integrate with Other Services**

* Select Load Balancing: **Attach to existing load balancer**
* Choose target group: **TG-LB-HTTP**
* Health checks:
  + EC2 health checks (default)
  + Enable **ELB health checks** (recommended)

****

****

#### **Configure Group Size and Scaling**

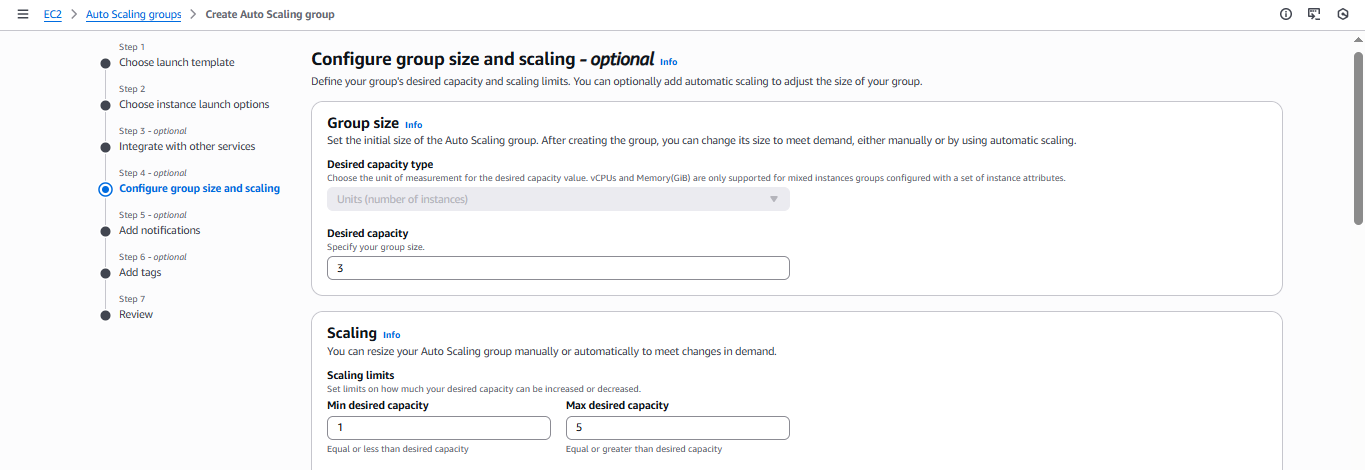
* Desired capacity: **3**
* Min size: **1**
* Max size: **5**

#### **Step 5: Add Notifications** (optional)

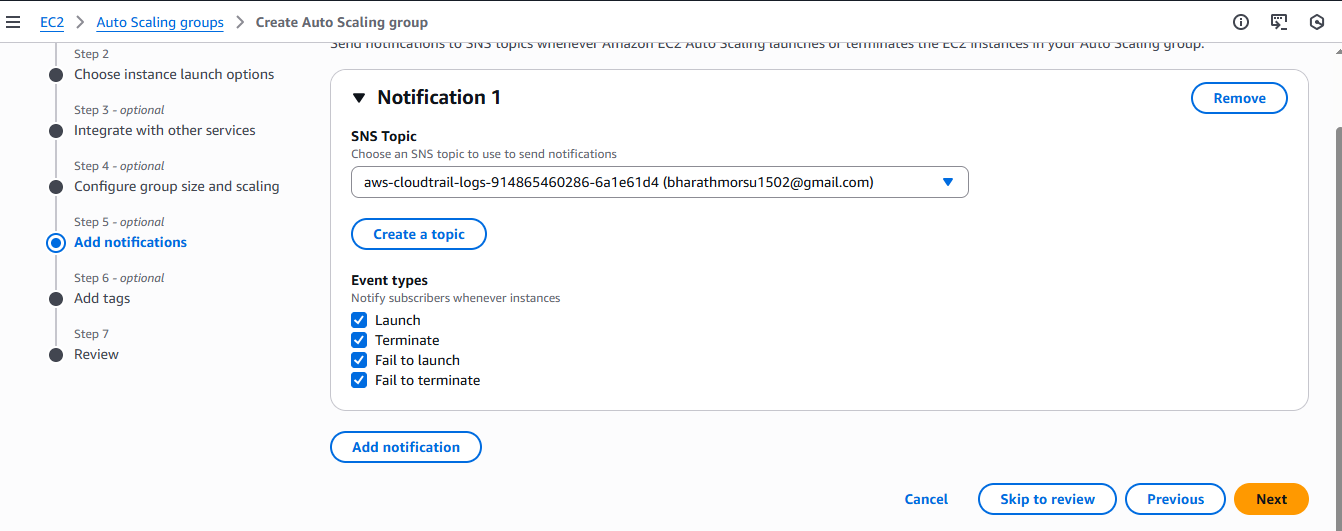
#### **Step 6: Add Tags** (optional)

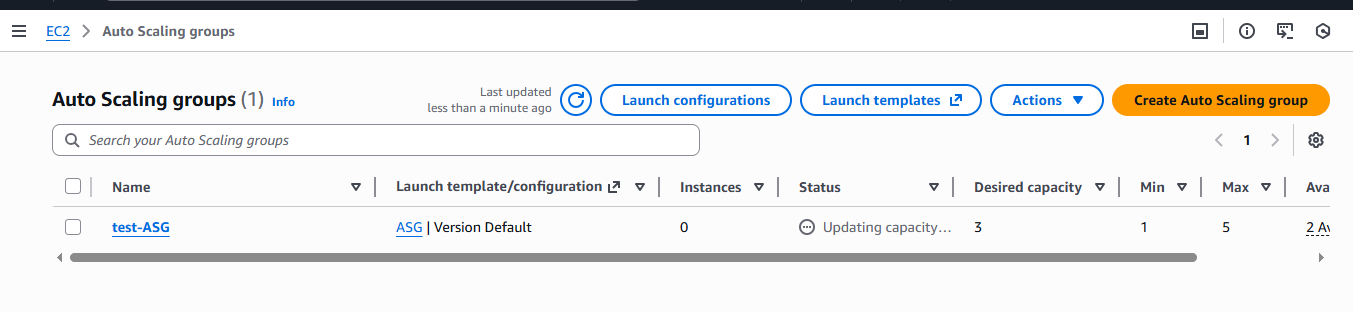
#### **Step 7: Review and Create**

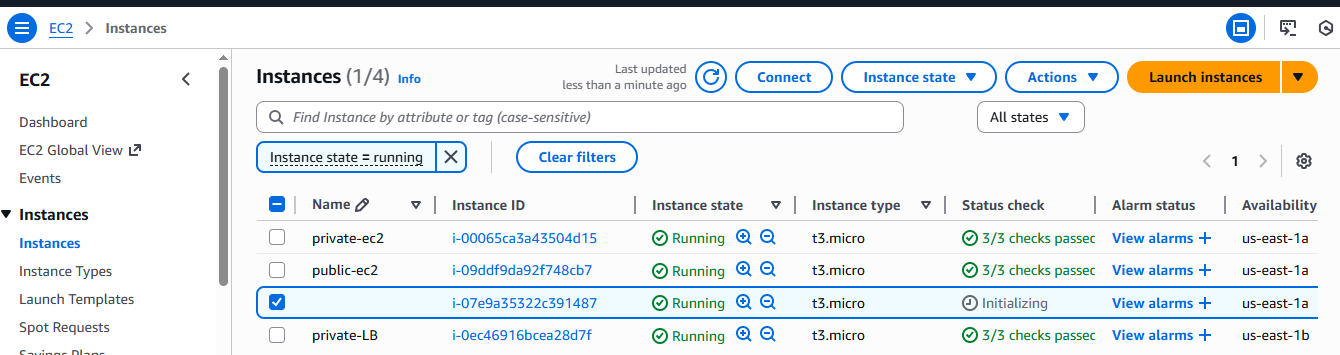
* Create Auto Scaling Group

****

****

****

****

****

# **Validation Steps**

### **1. Check ASG Activity**

* Go to **ASG → Activity**
* Ensure instances are being launched

### **2. Verify EC2 Instances**

* EC2 dashboard should show ~3 running instances
* Each instance should show checks:
  + **2/2 checks passed**
  + **ELB health check passed**

### **3. Confirm ALB Registration**

* Go to **Load Balancer → Target Groups → TG-LB-HTTP**
* Targets should show:
  + **healthy**

### **4. Test the Load Balancer**

* Copy ALB DNS name
* Paste in browser
* Should load the page from one of the EC2 instances

# **Issues Faced**

* Initial instance in ASG may show **Initializing** (normal)
* If security group does not allow port 80 → Target becomes **unhealthy**
* If instance does not install web server → ALB health check fails
* Wrong subnet selection → instance may not launch

# **Conclusion**

You successfully created a Launch Template and configured an Auto Scaling Group with multiple subnets and integrated it with an existing Application Load Balancer. The ASG now maintains instance availability and automatically registers instances with the ALB, providing high availability and scalability.