**AUTOSCALLING TASKS**

1. Create one VPC in N.virginia region.

**🔹 Step 1: Select N. Virginia Region**

1. Go to the **AWS Management Console**
2. In the top-right corner, click on the **region selector**
3. Select **N. Virginia (us-east-1)**

**🔹 Step 2: Go to VPC Dashboard**

1. In the Services menu, search for **VPC**
2. Click **“VPC Dashboard”**

**🔹 Step 3: Create a VPC**

1. Click **“Create VPC”**
2. Choose **“VPC only”** option

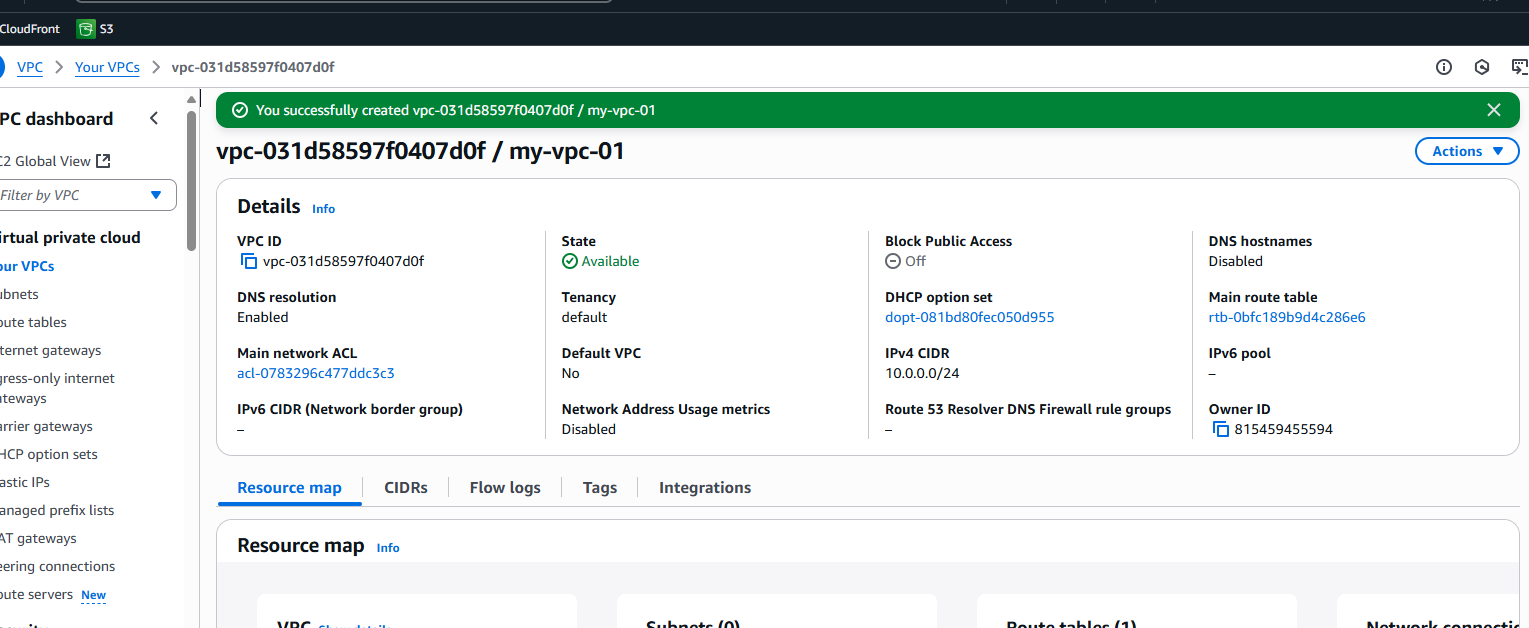
**🔹 Step 4: Configure VPC Settings**

1. **Name tag**: MyVPC (or any name)
2. **IPv4 CIDR block**: 10.0.0.0/16 (or your preferred block)
3. Leave **IPv6 CIDR block** as "No IPv6 CIDR block"
4. **Tenancy**: Default
5. Click **Create VPC**

**🔹 Step 5: Confirm VPC Creation**

1. After successful creation, click **“View VPC”**
2. You will see your VPC listed with the specified CIDR

✅ Done! You've successfully created a VPC in the **N. Virginia** region.



1. Create two subnets. One Public subnet and one private subnet.

**🔹 Step 1: Go to VPC Dashboard**

1. Open **AWS Console**
2. Go to **VPC → Subnets**
3. Click **“Create subnet”**

**🔹 Step 2: Create Public Subnet**

1. **VPC ID**: Select your VPC (e.g., MyVPC)
2. **Subnet name**: PublicSubnet
3. **Availability Zone**: Choose one (e.g., us-east-1a)
4. **IPv4 CIDR block**: e.g., 10.0.1.0/24
5. Click **Create subnet**

**🔹 Step 3: Create Private Subnet**

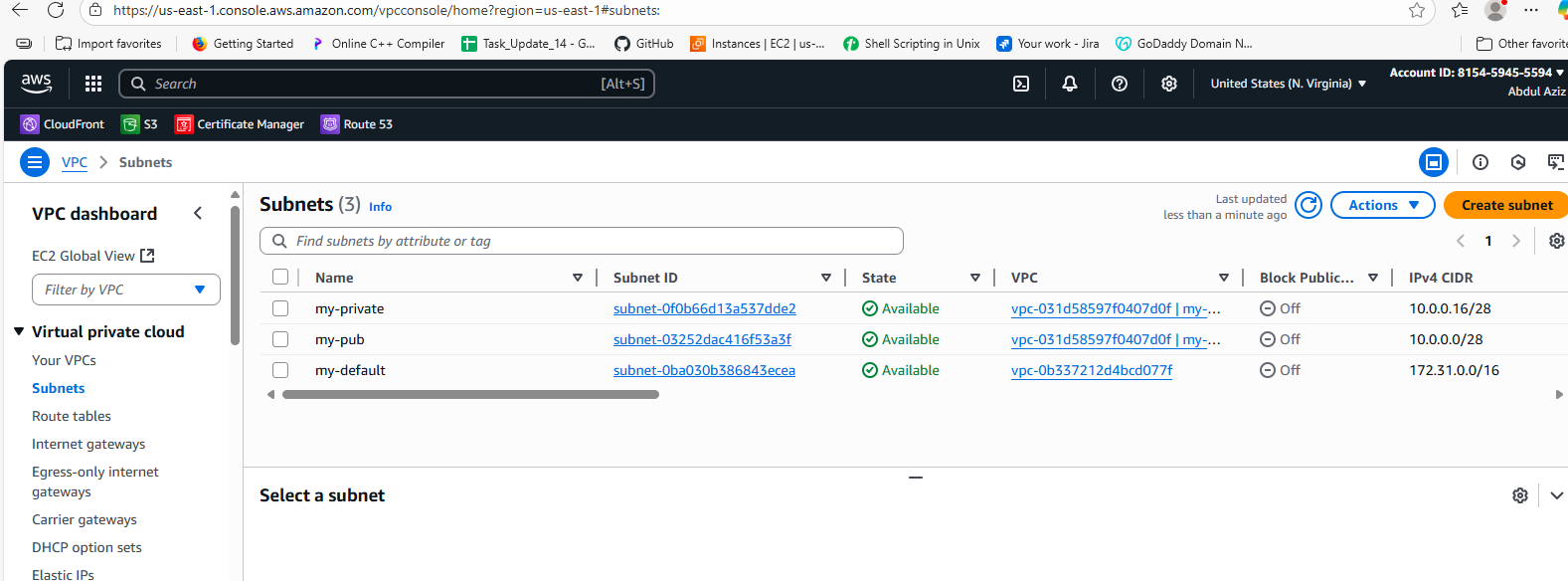
1. Click **“Create subnet”** again
2. **VPC ID**: Same VPC
3. **Subnet name**: PrivateSubnet
4. **Availability Zone**: Choose another or same (e.g., us-east-1b)
5. **IPv4 CIDR block**: e.g., 10.0.2.0/24
6. Click **Create subnet**

**🔹 Step 4: Make the Public Subnet Public**

1. Go to **VPC → Route Tables**
2. Find the main or custom route table associated with PublicSubnet
3. Edit routes:
   * Add route: 0.0.0.0/0 → target Internet Gateway
4. Go to **Subnet Associations**
   * Associate the route table with PublicSubnet

✅ Done! You now have:

* One **Public Subnet** (with internet access)
* One **Private Subnet** (no direct internet access)



1. Provide the IGW to the vpc.

**🔹 Step 1: Go to VPC Dashboard**

1. Open **AWS Console**
2. Navigate to **VPC → Internet Gateways**

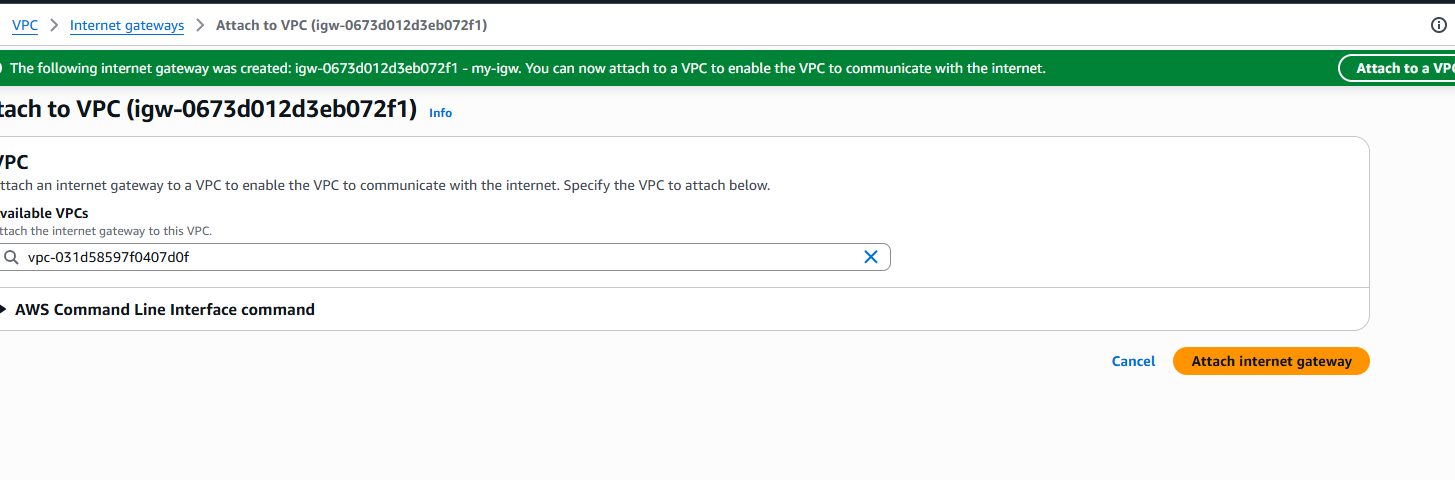
**🔹 Step 2: Create Internet Gateway**

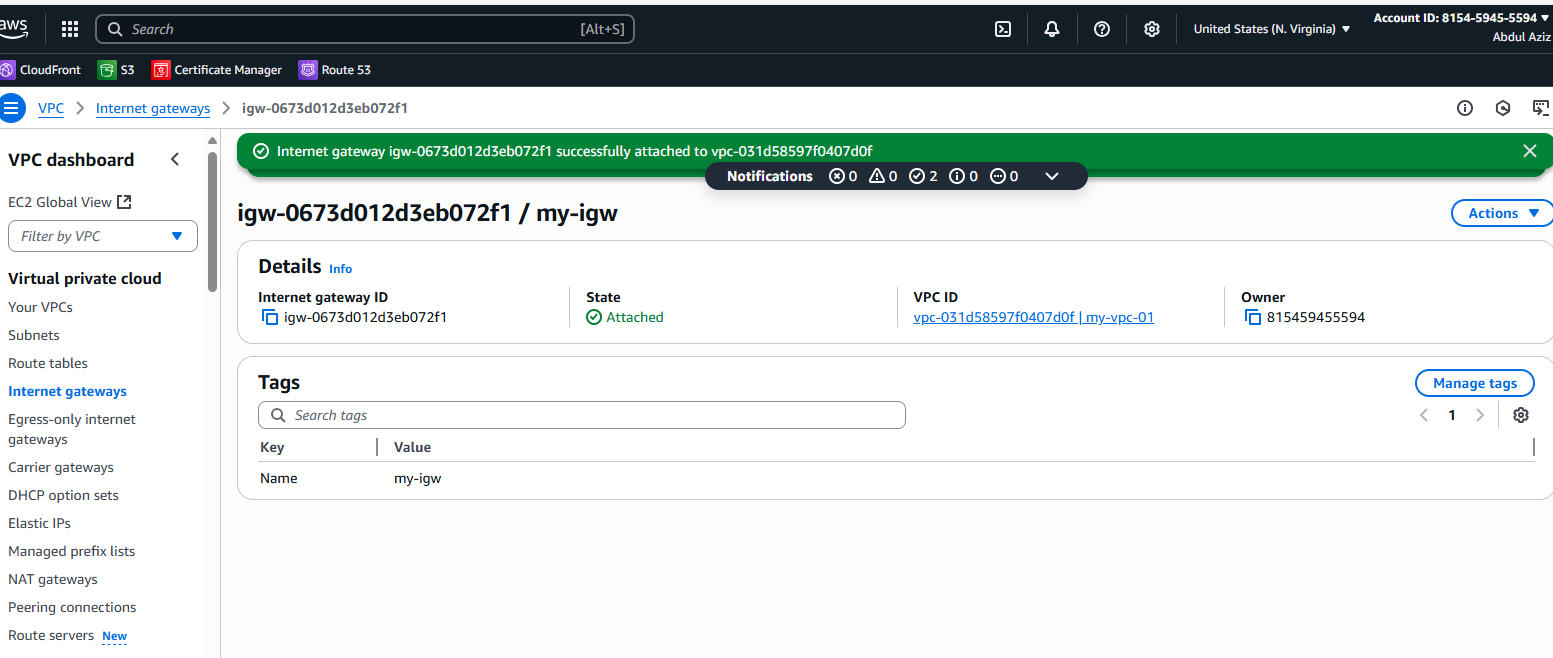
1. Click **“Create internet gateway”**
2. **Name tag**: MyIGW (or any name)
3. Click **Create internet gateway**

**🔹 Step 3: Attach IGW to Your VPC**

1. After creating, click **“Actions → Attach to VPC”**
2. Select your **VPC** (e.g., MyVPC)
3. Click **Attach internet gateway**

✅ Done! The IGW is now attached to your VPC, allowing **public subnets** with the correct route to access the internet.





1. Create One public RT and one private RT.

**🔹 Step 1: Go to VPC Dashboard**

1. Open **AWS Console**
2. Navigate to **VPC → Route Tables**
3. Click **“Create route table”**

**🔹 Step 2: Create Public Route Table**

1. **Name tag**: PublicRT
2. **VPC**: Select your VPC
3. Click **Create route table**

**🔹 Step 3: Edit Public Route Table for Internet Access**

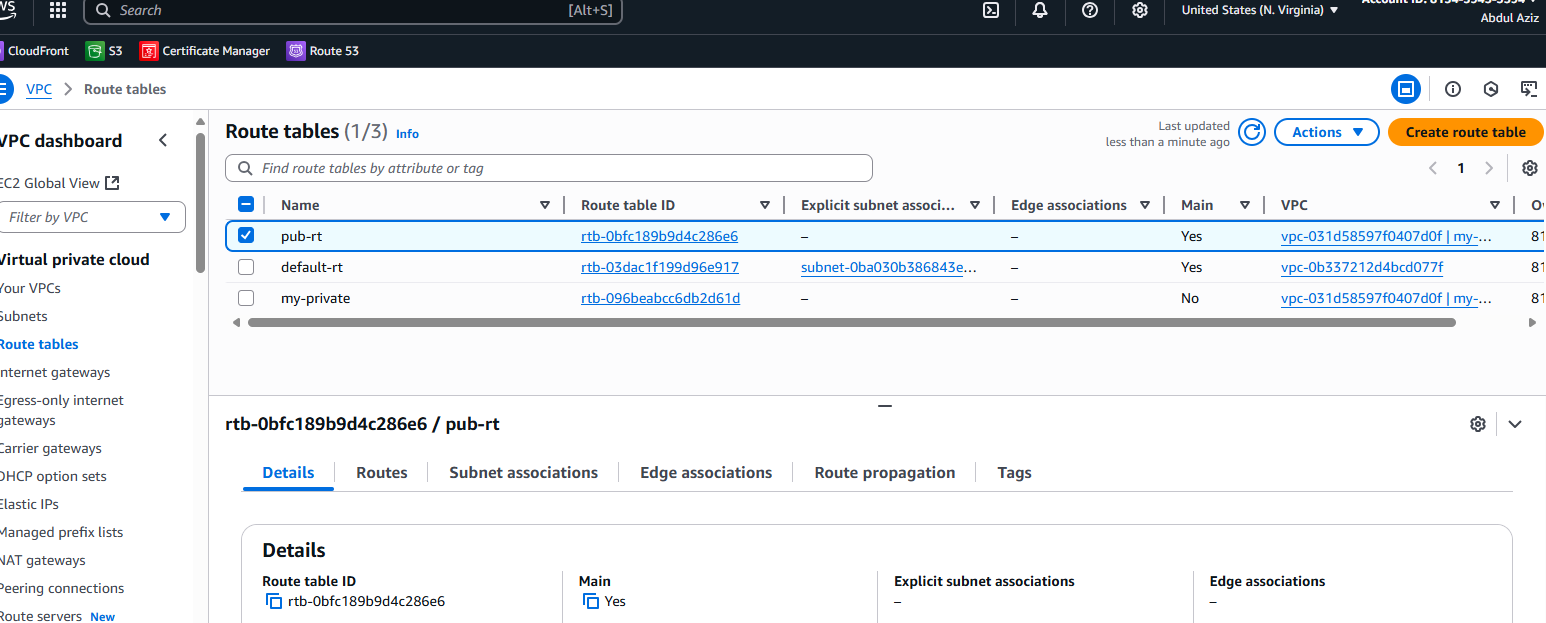
1. Select PublicRT → Click **“Routes” → Edit routes**
2. Add route:
   * **Destination**: 0.0.0.0/0
   * **Target**: Select your **Internet Gateway (IGW)**
3. Click **Save changes**

**🔹 Step 4: Create Private Route Table**

1. Go back to **Route Tables**
2. Click **“Create route table”**
3. **Name tag**: PrivateRT
4. **VPC**: Select the same VPC
5. Click **Create route table**

✅ Done! You now have:

* PublicRT with a route to the internet
* PrivateRT for private subnet traffic without internet access by default



1. Deploy NAT gateway on public subnet and attach the NAT gatewat to private subnet.

**🔹 Step 1: Allocate an Elastic IP**

1. Go to **EC2 → Elastic IPs**
2. Click **“Allocate Elastic IP address”**
3. Click **Allocate**

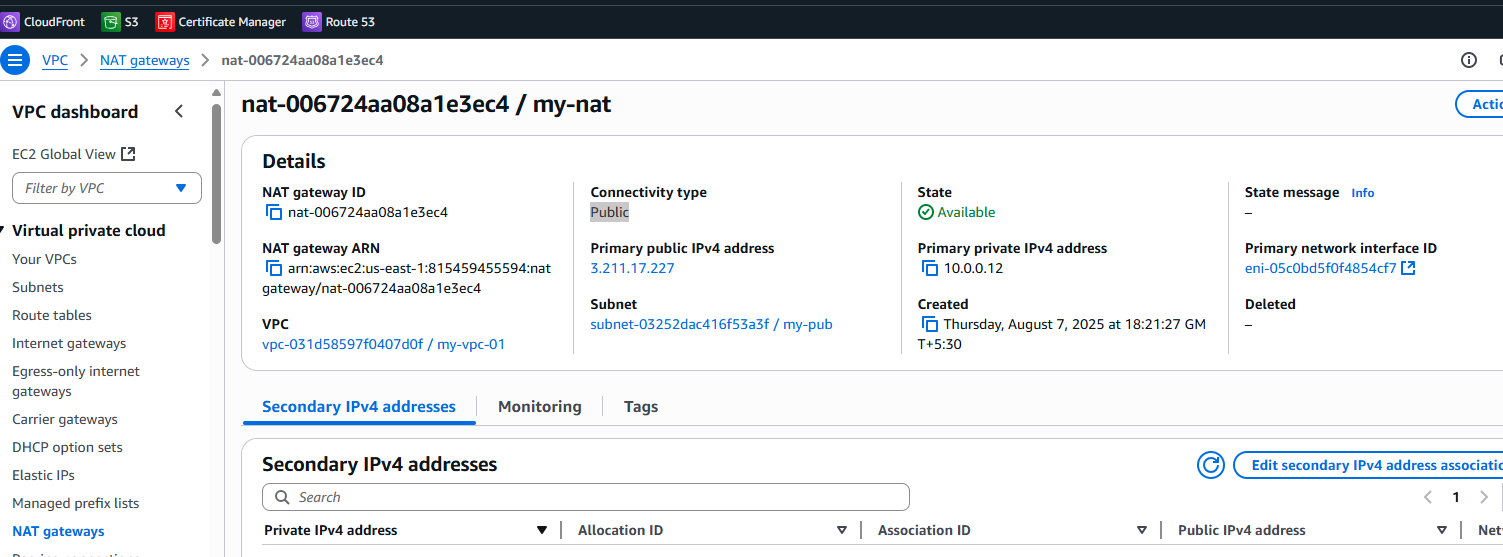
**🔹 Step 2: Create NAT Gateway in Public Subnet**

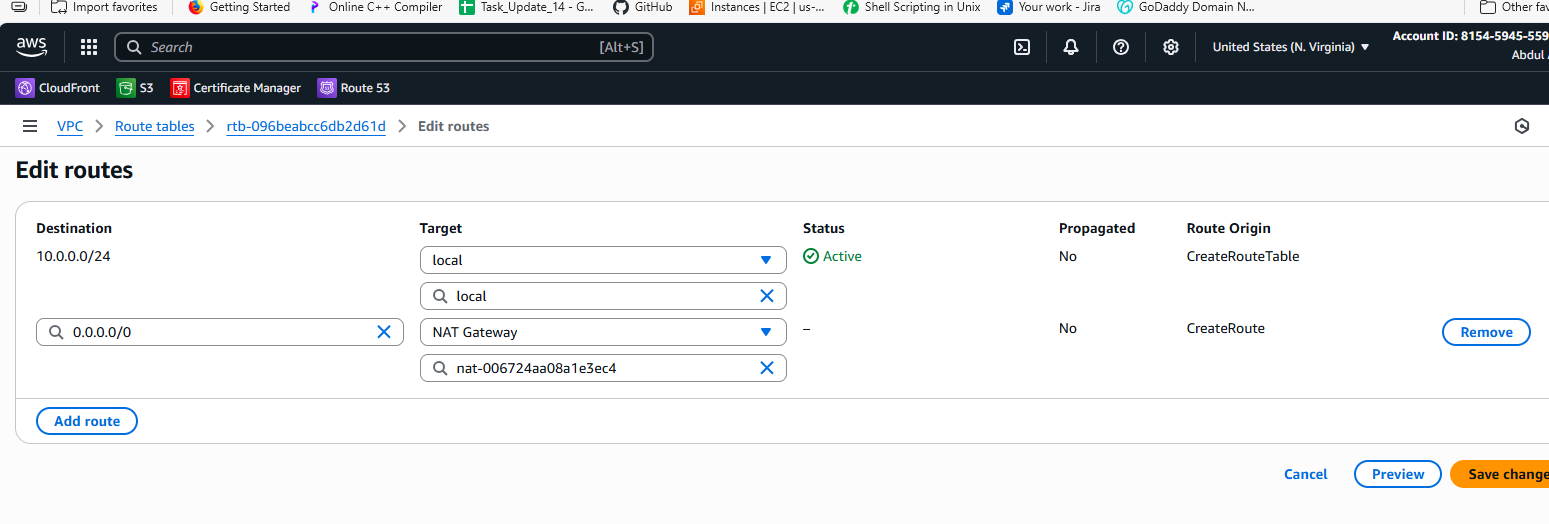
1. Go to **VPC → NAT Gateways**
2. Click **“Create NAT Gateway”**
3. **Name**: MyNATGateway
4. **Subnet**: Select your **Public Subnet**
5. **Elastic IP**: Select the EIP you just allocated
6. Click **Create NAT Gateway**

**🔹 Step 3: Update Private Route Table**

1. Go to **VPC → Route Tables**
2. Select your **Private Route Table**
3. Click **“Routes” → Edit routes**
4. Add a route:
   * **Destination**: 0.0.0.0/0
   * **Target**: Select the **NAT Gateway**
5. Click **Save changes**

✅ Done! Now, instances in your **private subnet** can access the internet **via the NAT Gateway**, while still remaining **inaccessible from outside**.





1. Create Two instances,one in public subnet and one in private subnet.

**🔹 Step 1: Go to EC2 Console**

1. Open **AWS Console**
2. Navigate to **EC2 → Instances**
3. Click **“Launch instance”**

**🔹 Step 2: Create Instance in Public Subnet**

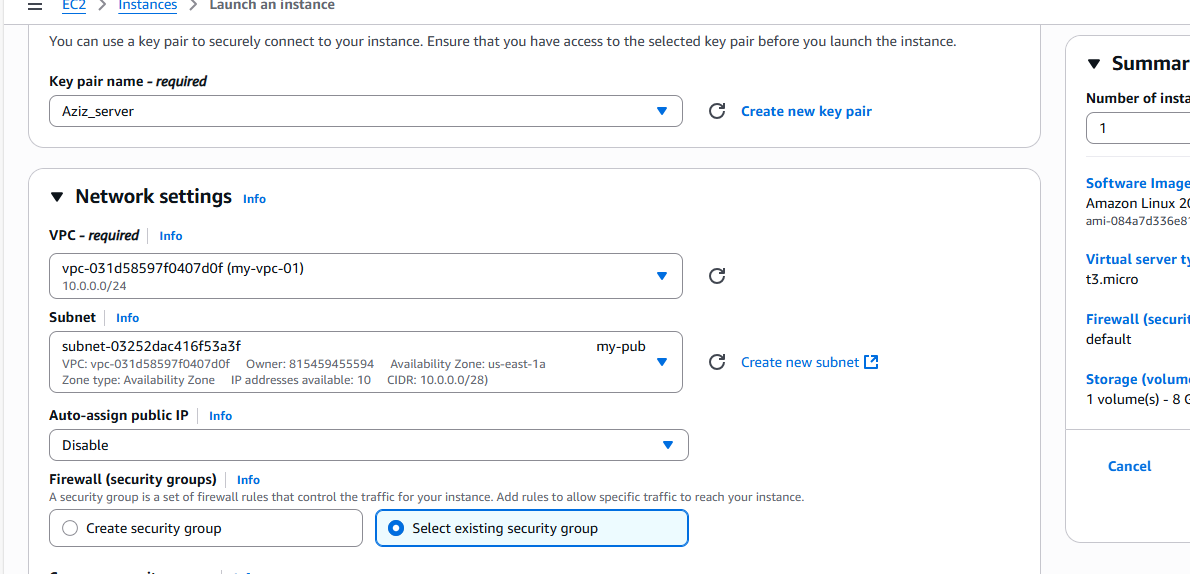
1. **Name**: PublicInstance
2. **AMI**: Choose Amazon Linux 2 or preferred AMI
3. **Instance type**: t2.micro (free tier eligible)
4. **Key pair**: Select or create a key pair
5. **Network settings**:
   * **VPC**: Select your VPC
   * **Subnet**: Choose **Public Subnet**
   * **Auto-assign public IP**: **Enable**
   * **Security group**: Allow SSH (port 22)
6. Click **Launch Instance**

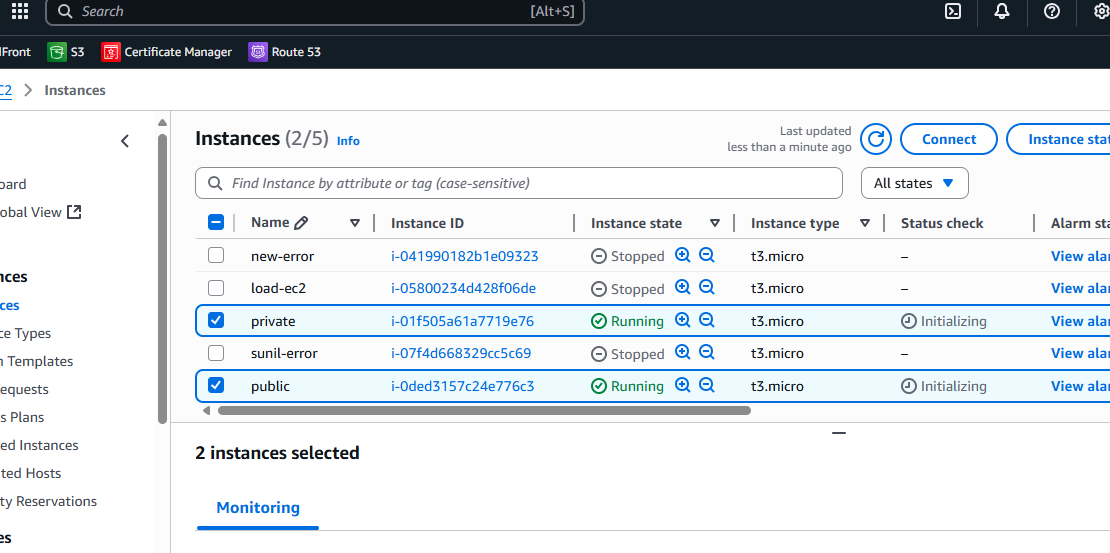
**🔹 Step 3: Create Instance in Private Subnet**

1. Go back to **Launch Instance**
2. **Name**: PrivateInstance
3. **AMI**: Same as above
4. **Instance type**: t2.micro
5. **Key pair**: Same or new key pair
6. **Network settings**:
   * **VPC**: Same VPC
   * **Subnet**: Choose **Private Subnet**
   * **Auto-assign public IP**: **Disable**
   * **Security group**: Allow SSH from the **PublicInstance**'s private IP
7. Click **Launch Instance**

✅ Done!

* PublicInstance can be accessed via the internet
* PrivateInstance can be accessed **via SSH from the PublicInstance** (using private IP) if required.





1. Deploy Apache server on both the ec2 instances with sample index.html file.

**🔹 Step 1: Connect to Each EC2 Instance via SSH**

1. Use your key pair to SSH into each instance:

bash

CopyEdit

ssh -i key.pem ec2-user@<public-ip>

**🔹 Step 2: Install Apache Web Server**

Run the following commands **on both instances**:

bash

CopyEdit

sudo yum update -y # For Amazon Linux

sudo yum install httpd -y

**🔹 Step 3: Start and Enable Apache**

bash

CopyEdit

sudo systemctl start httpd

sudo systemctl enable httpd

**🔹 Step 4: Add Sample index.html Page**

bash

CopyEdit

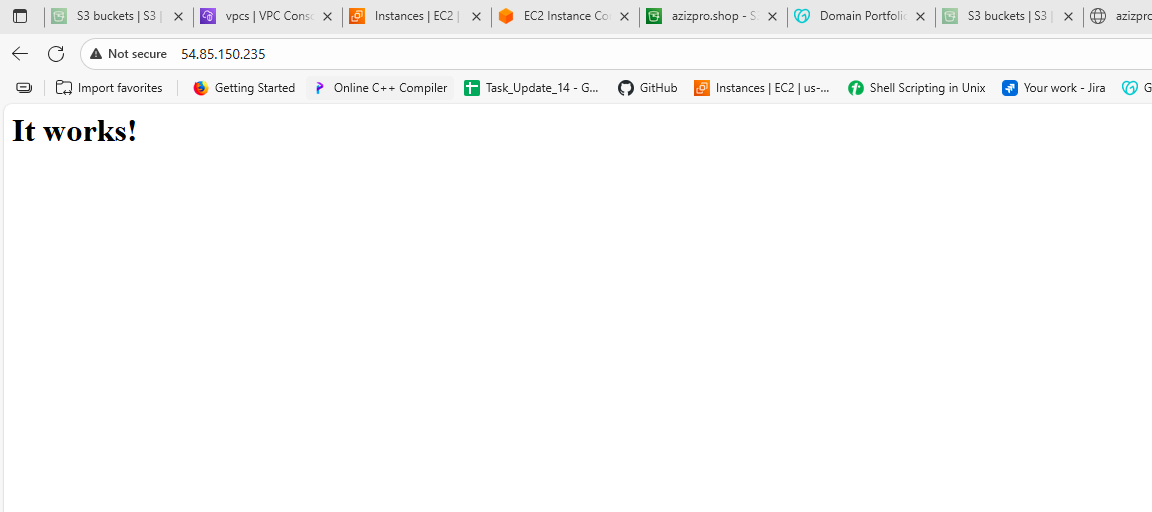
echo "<h1>Welcome to Apache on $(hostname)</h1>" | sudo tee /var/www/html/index.html

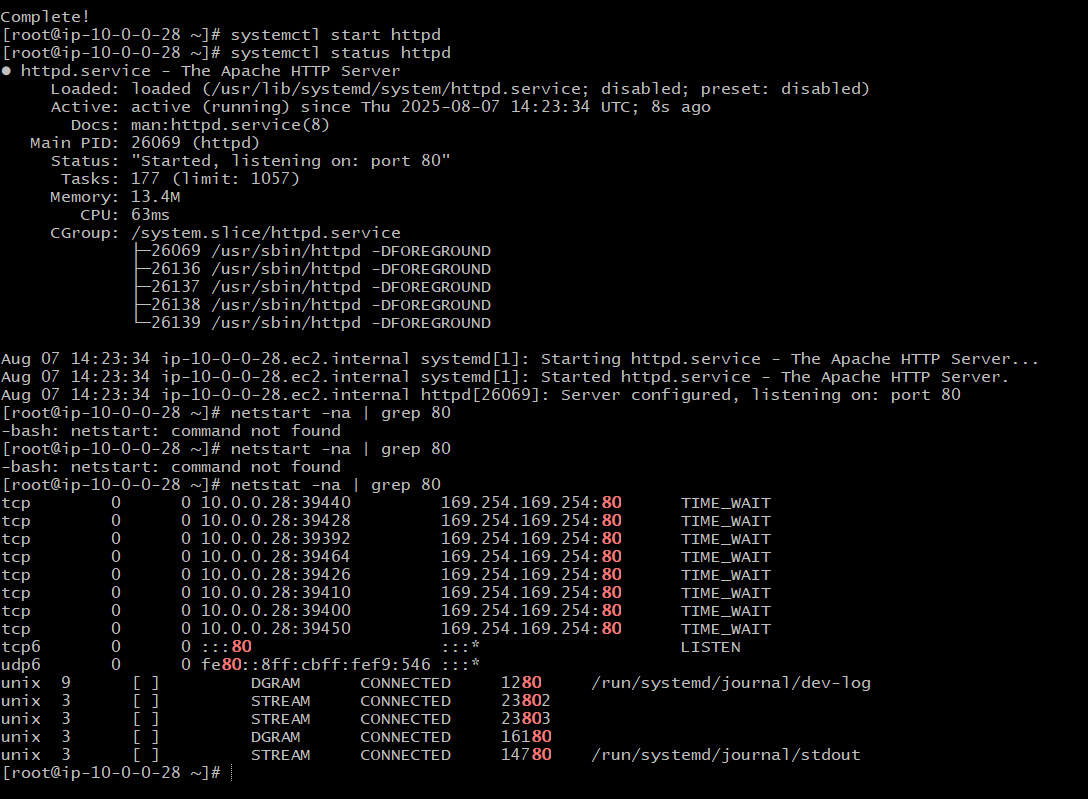
**🔹 Step 5: Adjust Security Group (for Public Instance)**

1. Go to **EC2 → Security Groups**
2. Edit **Inbound Rules** of the Public Instance
   * Add rule: **HTTP** | Port: 80 | Source: 0.0.0.0/0

✅ Done!

* You can access the public instance in a browser using:  
  http://<public-ip>





1. Create one application load balancer and attach the load balancer to both the ec2 instances.

**🔹 Step 1: Go to EC2 Console → Load Balancers**

1. Open **AWS Console**
2. Navigate to **EC2 → Load Balancers**
3. Click **“Create Load Balancer”**
4. Choose **Application Load Balancer** → Click **Create**

**🔹 Step 2: Configure Load Balancer**

1. **Name**: MyALB
2. **Scheme**: internet-facing
3. **IP address type**: IPv4
4. **Listeners**: HTTP (port 80) is default
5. **Availability Zones**:
   * Select your **VPC**
   * Check **both subnets** (public ones) for different AZs

**🔹 Step 3: Configure Security Group**

1. Select or create a **Security Group** that allows **inbound HTTP (port 80)**

**🔹 Step 4: Configure Target Group**

1. **Target group name**: MyTargetGroup
2. **Target type**: Instance
3. **Protocol**: HTTP | Port: 80
4. **VPC**: Choose your VPC
5. Click **Next**

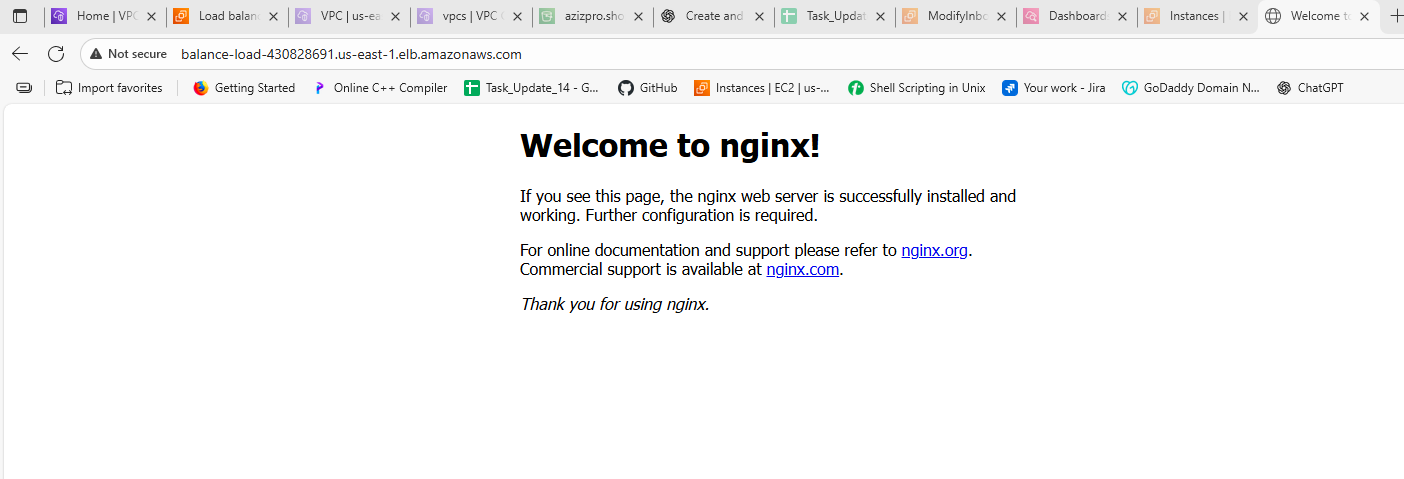
**🔹 Step 5: Register Targets (EC2 Instances)**

1. Select **both EC2 instances** (public and private)
2. Click **Add to registered**
3. Click **Create target group**

**🔹 Step 6: Review and Create ALB**

1. Go back to ALB config
2. Select the **target group** you just created
3. Review and click **Create load balancer**

✅ Done!  
Your **Application Load Balancer** now distributes traffic to both EC2 instances. You can access it via the ALB's **DNS name** shown in the EC2 → Load Balancer dashboard.



1. Store Application load balancer logs to s3.

**🔹 Step 1: Create an S3 Bucket for Logs**

1. Go to **S3 → Buckets → Create bucket**
2. **Name**: e.g., alb-logs-bucket-yourname
3. Choose Region (same as ALB)
4. Keep other settings default (or configure as needed)
5. Click **Create bucket**

**🔹 Step 2: Enable Access Logging on the ALB**

1. Go to **EC2 → Load Balancers**
2. Select your **Application Load Balancer**
3. Click on the **“Description”** tab
4. Click **“Edit attributes”**

**🔹 Step 3: Configure Logging**

1. Check ✅ **Enable access logs**
2. **S3 bucket**: Enter the bucket name (alb-logs-bucket-yourname)
3. (Optional) Set a prefix (e.g., logs/)
4. Click **Save changes**

**🔹 Step 4: Set Bucket Permissions (if needed)**

1. Go to the **S3 bucket**
2. Click **Permissions → Bucket policy**
3. Add policy to allow ALB to write logs:

json

CopyEdit

{

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

"Statement": [

{

"Sid": "AWSALBLoggingPolicy",

"Effect": "Allow",

"Principal": {

"Service": "elasticloadbalancing.amazonaws.com"

},

"Action": "s3:PutObject",

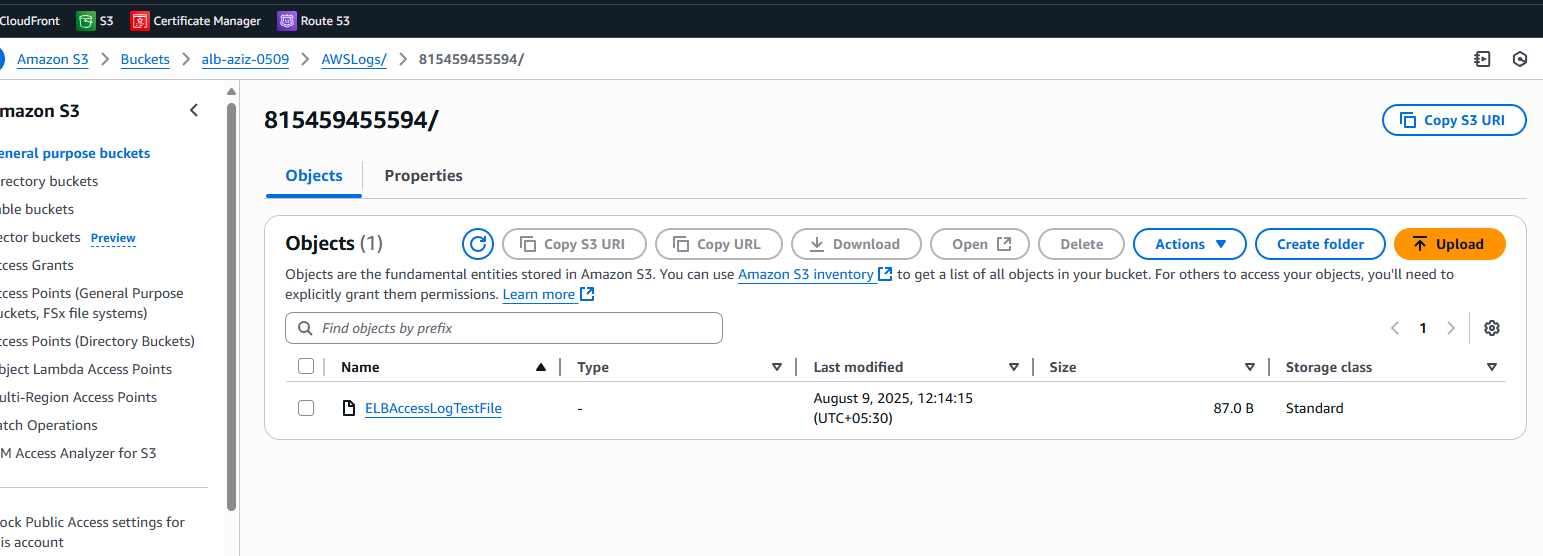
"Resource": "arn:aws:s3:::alb-logs-bucket-yourname/\*"

}

]

}

✅ Done!  
Your **ALB access logs** will now be delivered to the specified **S3 bucket**.



1. Store the VPC flow logs to CloudWatch group.

**🔹 Step 1: Go to VPC Dashboard**

1. Open **AWS Console**
2. Navigate to **VPC → Your VPCs**
3. Select your **VPC**

**🔹 Step 2: Create Flow Log**

1. Click on **“Actions → Create flow log”**
2. **Filter**: Select All, Accept, or Reject (e.g., All)
3. **Destination**: Select Send to CloudWatch Logs

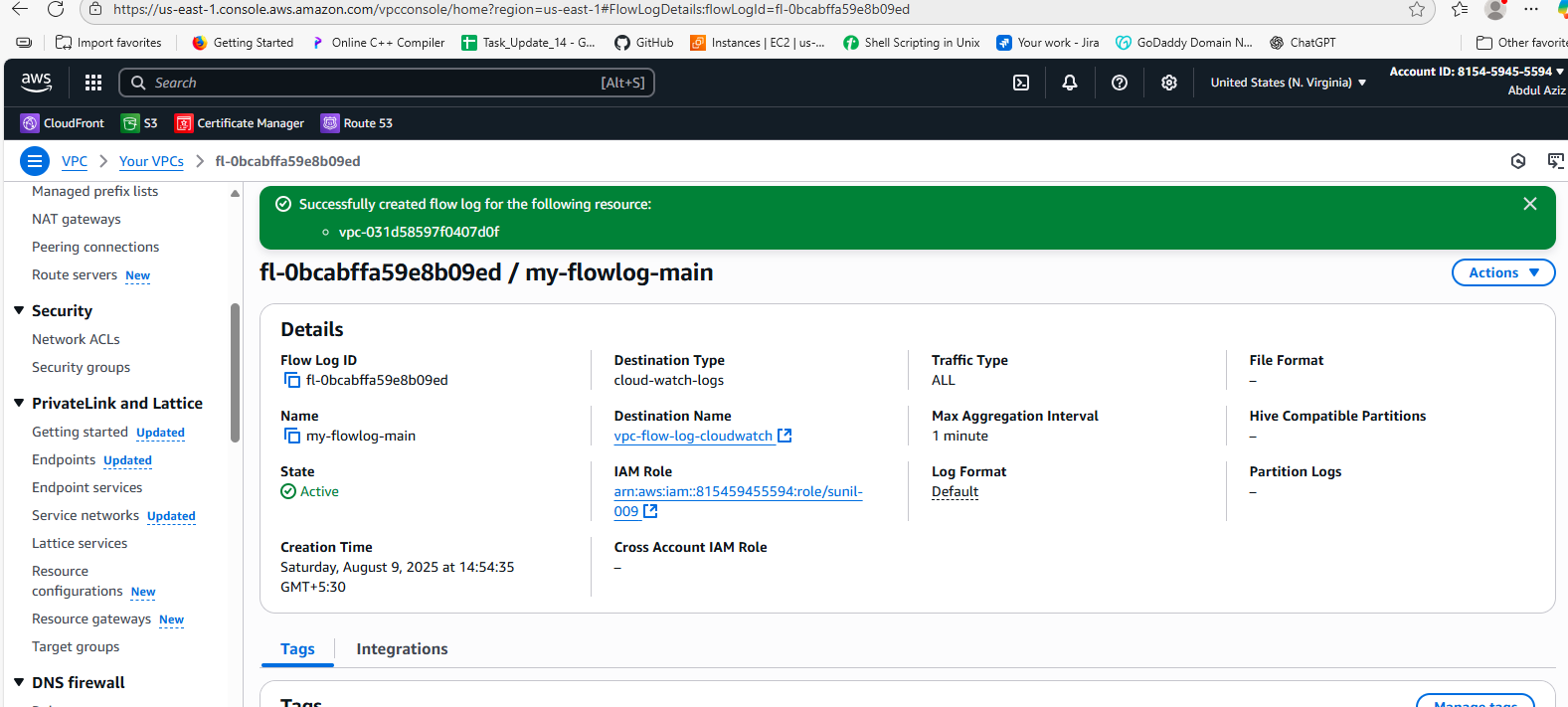
**🔹 Step 3: Set Log Group and IAM Role**

1. **Log group**:
   * Select existing or create new (e.g., /vpc/flowlogs)
2. **IAM Role**:
   * Choose existing role or click **Set up permissions**
   * AWS will auto-generate a role like FlowLogsRole

**🔹 Step 4: Finalize and Create**

1. Click **Create flow log**
2. It starts logging VPC traffic to the selected **CloudWatch log group**

✅ Done!  
Your **VPC Flow Logs** are now stored in **CloudWatch Logs**, where you can monitor and analyze them.



1. Create Monitoring Dashboards to monitor CPU utilization and to monitor Apache service.

**🔹 Step 1: Open CloudWatch Console**

1. Open **AWS Console**
2. Navigate to **CloudWatch → Dashboards**
3. Click **Create dashboard**
4. Enter a **Dashboard name** (e.g., MyMonitoringDashboard)
5. Click **Create dashboard**

**🔹 Step 2: Add CPU Utilization Widget**

1. Choose **“Line” widget**
2. Click **Configure**
3. Under **Metrics**, select:
   * **EC2 → Per-Instance Metrics → CPUUtilization**
   * Select your EC2 instance(s)
4. Click **Create widget**

**🔹 Step 3: Create Apache Service Monitoring Metric (Custom Metric)**

1. On each EC2 instance, run a script or CloudWatch Agent to push Apache status (e.g., up/down) as a **custom metric** to CloudWatch (see note below).
2. Alternatively, create a **CloudWatch Logs Metric Filter** if Apache logs are pushed to CloudWatch Logs.

**🔹 Step 4: Add Apache Service Widget**

1. Back on the dashboard, click **Add widget**
2. Choose **“Number” or “Line” widget**
3. Select your **custom Apache metric namespace**
4. Select metric representing Apache service status
5. Click **Create widget**

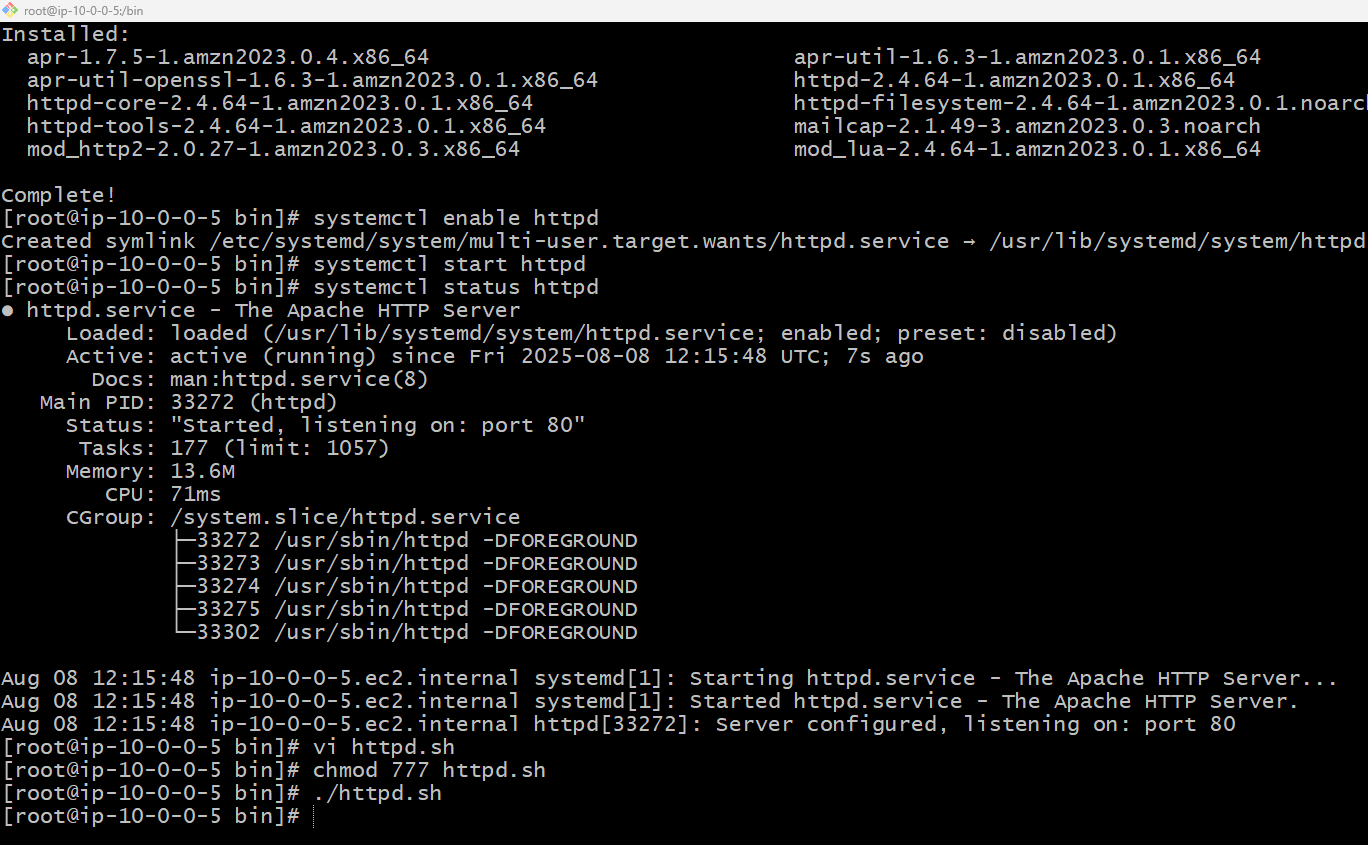
**🔹 Step 5: Save and Review Dashboard**

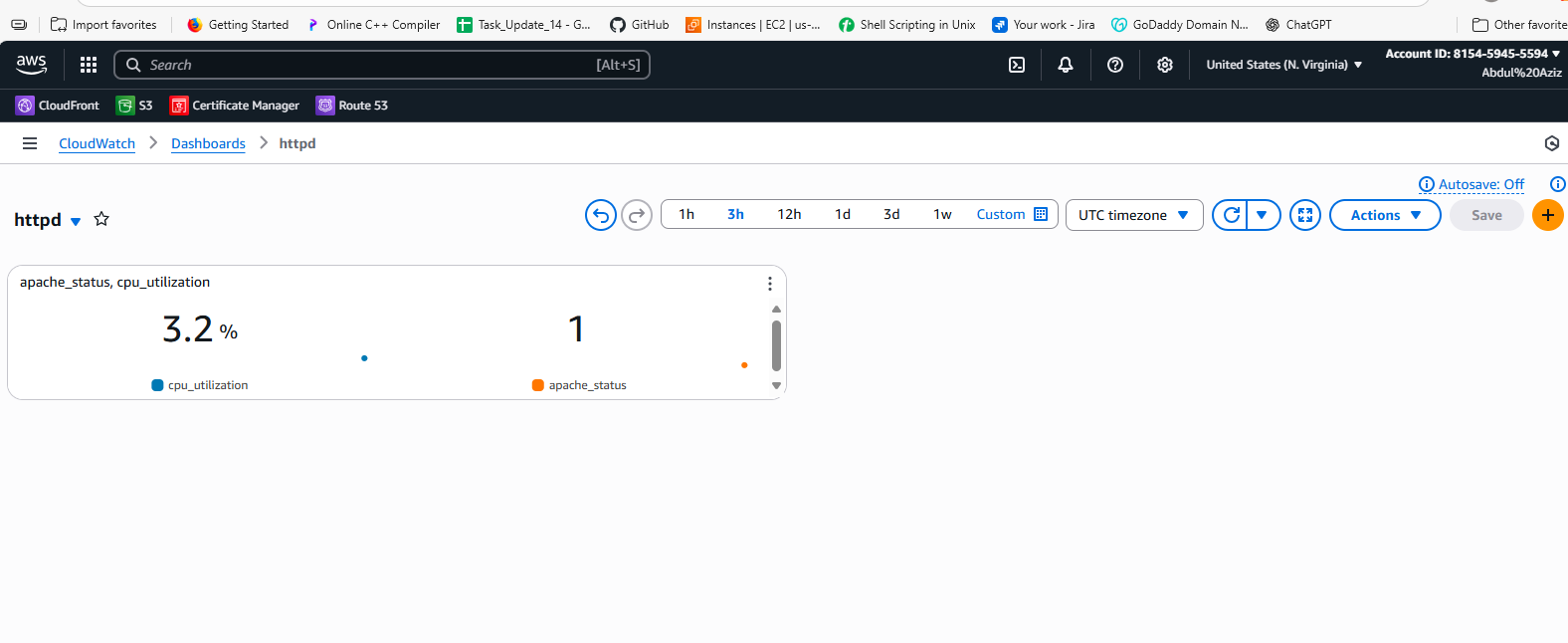
* The dashboard now shows CPU usage and Apache service status.

**Note: Apache monitoring script example**

* Run on EC2 instance and publish custom metric to CloudWatch (e.g., status 1 if running, 0 if stopped).

✅ Done!  
You now have a CloudWatch dashboard monitoring CPU utilization and Apache service health.





1. CPU utilizations more than 70% then it should triggered Autoscaling and launch new instance.