

# Scalable PHP Web Application on AWS using ALB & Auto Scaling

---

## PROJECT GOAL

The goal of this project is to design and implement a **scalable PHP web application on AWS** where:

- Users access the application through a **single public URL**
- Incoming traffic is **automatically load balanced**
- Backend servers **scale up and down automatically** based on CPU load
- High availability is ensured using **multiple Availability Zones**

## Core AWS Services Used

- Amazon EC2
- Application Load Balancer (ALB)
- Auto Scaling Group (ASG)
- Amazon CloudWatch

---

## STEP 1 — Launch EC2 & Host PHP App (FOUNDATION)

### Objective



Create a **single working PHP server** that will later be used as the **reference template** for scaling.

---

## ✓ 1. Launch EC2 Instance

**AWS Console → EC2 → Launch instance**

Configuration:

- AMI: **Amazon Linux 2023**
- Instance type: **t2.micro**
- Key pair: Created and downloaded
- Security Group:
  - Allow **SSH (22)** from my IP
  - Allow **HTTP (80)** from anywhere

Instance launched successfully.

---

## ✓ 2. Connect to EC2

```
ssh -i yourkey.pem ec2-user@PUBLIC_IP
```

---

## ✓ 3. Install Apache & PHP

```
sudo yum update -y  
sudo yum install httpd php -y
```



```
sudo systemctl start httpd
sudo systemctl enable httpd
```

---

#### ✓ 4. Create PHP Test Application

```
sudo nano /var/www/html/index.php
```

```
<?php
echo "<h1>PHP App Working</h1>";
echo "Server IP: " . $_SERVER['SERVER_ADDR'];
?>
```

Restart Apache:

```
sudo systemctl restart httpd
```

---

#### ✓ 5. Verification

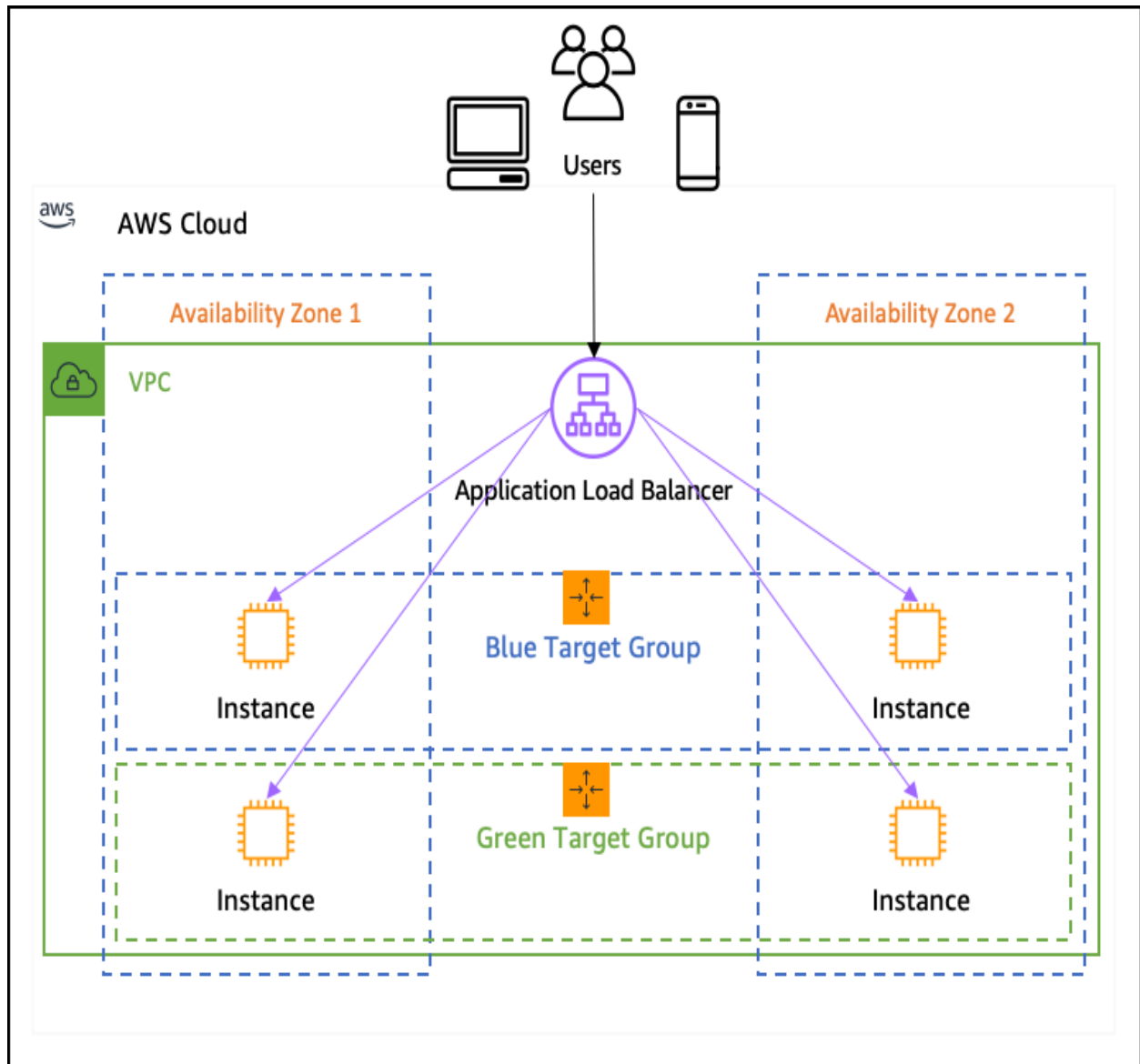
- Open browser: [http://EC2\\_PUBLIC\\_IP](http://EC2_PUBLIC_IP)
- Page loads successfully
- Server IP is displayed

✓ PHP application is working correctly

---

### STEP 3 — Architecture Diagram



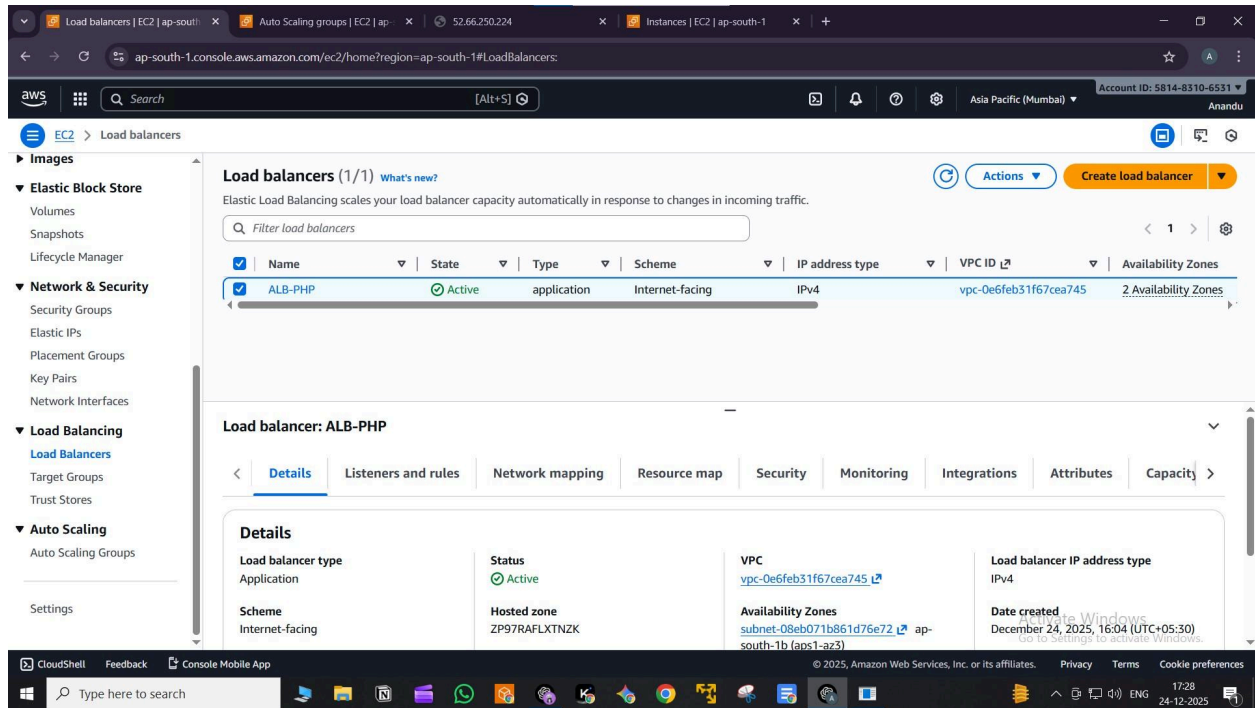


### STEP 3 — Create Application Load Balancer (ALB)

#### Purpose

- Provide **one public entry point**
- Route traffic to EC2 instances
- Prepare the system for Auto Scaling.





## ✅ STEP 2.1 — Create Target Group (Mandatory)

**AWS Console → EC2 → Target Groups → Create target group**

Configuration:

- Target type: **Instance**
- Protocol: **HTTP**
- Port: **80**
- VPC: Same as EC2
- Health check path: /



Register target:

- Select EC2 instance
- Include as pending
- Create target group

✓ Health status becomes **Healthy**

Successfully deregistered 1 target.

Target groups (1/1) Info | What's new?

Filter target groups

<input checked="" type="checkbox"/>	Name	Port	Protocol	Target type	Load balancer	VPC ID
<input checked="" type="checkbox"/>	lasticloadbalancin...	80	HTTP	Instance	ALB-PHP	vpc-0e6feb31f67cea745

Target group: ALB

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Filter targets

<input type="checkbox"/>	Instance ID	Name	Port	Zone	Health status	Health status details	Admini...	Overri...
<input type="checkbox"/>	i-01e47e06c1aa29fe5	php-asg	80	ap-south-1a (a...	Healthy	-	No override.	No over
<input type="checkbox"/>	i-0c0905959f128f1b8	server1	80	ap-south-1b (a...	Draining	Target deregistration i...	No override.	No over

✓ **STEP 2.2 — Create Application Load Balancer**

**EC2 → Load Balancers → Create load balancer**

Configuration:

- Type: **Application Load Balancer**



- Scheme: **Internet-facing**
  - Listener: **HTTP : 80**
  - VPC: Same as EC2
  - Subnets: Minimum **2 AZs**
  - Forward traffic to target group: **ALB**
  - Security Group: Allow HTTP 80 from **0.0.0.0/0**
- 

### **Expected Result**

- Target Group shows:
  - Healthy: **1**
  - Unused: **0**
- Load balancer name appears instead of *None associated*

✓ ALB → Target Group → EC2 → PHP App path confirmed

---

## **STEP 3 — Create Launch Template**

### **Objective**

Define a blueprint so AWS can **automatically create identical EC2 instances**.



---

## ✓ STEP 3.1 — Create Launch Template

EC2 → Launch Templates → Create launch template

Configuration:

- Name: **php-launch-template**
- AMI: Amazon Linux 2023
- Instance type: t2.micro
- Key pair: Same as before
- Security Group: Same as EC2

---

## ⚠ USER DATA (CRITICAL)

```
#!/bin/bash
yum install httpd php -y
systemctl start httpd
systemctl enable httpd

cat <<EOF > /var/www/html/index.php
<?php
echo "<h1>Auto Scaled PHP Server</h1>";
echo "<br>Server IP: " . \$_SERVER['SERVER_ADDR'];
?>
EOF
```



- ✓ Ensures every new instance is ready automatically
- ✓ No manual SSH required

Launch template created successfully.

---

## **STEP 4 — Create Auto Scaling Group (ASG)**

### **Objective**

- Automatically manage EC2 instances
  - Attach instances to ALB
  - Scale based on CPU load
- 

### **STEP 4.1 — Create Auto Scaling Group**

**EC2 → Auto Scaling Groups → Create Auto Scaling group**

- Name: **php-asg**
  - Launch template: **php-launch-template**
  - Version: Default
- 

### **STEP 4.2 — Network & AZs**

- VPC: Same as ALB
- Subnets: At least **2 AZs**



- ✓ Provides high availability
- 

#### ✓ STEP 4.3 — Attach Load Balancer

- Attach to existing load balancer
- Target group: **ALB**
- Health check type: **ELB**
- Grace period: **300 seconds**

- ✓ New instances auto-register
  - ✓ Unhealthy instances auto-replaced
- 

#### ✓ STEP 4.4 — Capacity Settings

- Minimum: **1**
  - Desired: **1**
  - Maximum: **3**
- 

#### ✓ STEP 4.5 — Scaling Policy (Core Logic)

- Policy type: **Target tracking**
- Metric: **Average CPU utilization**

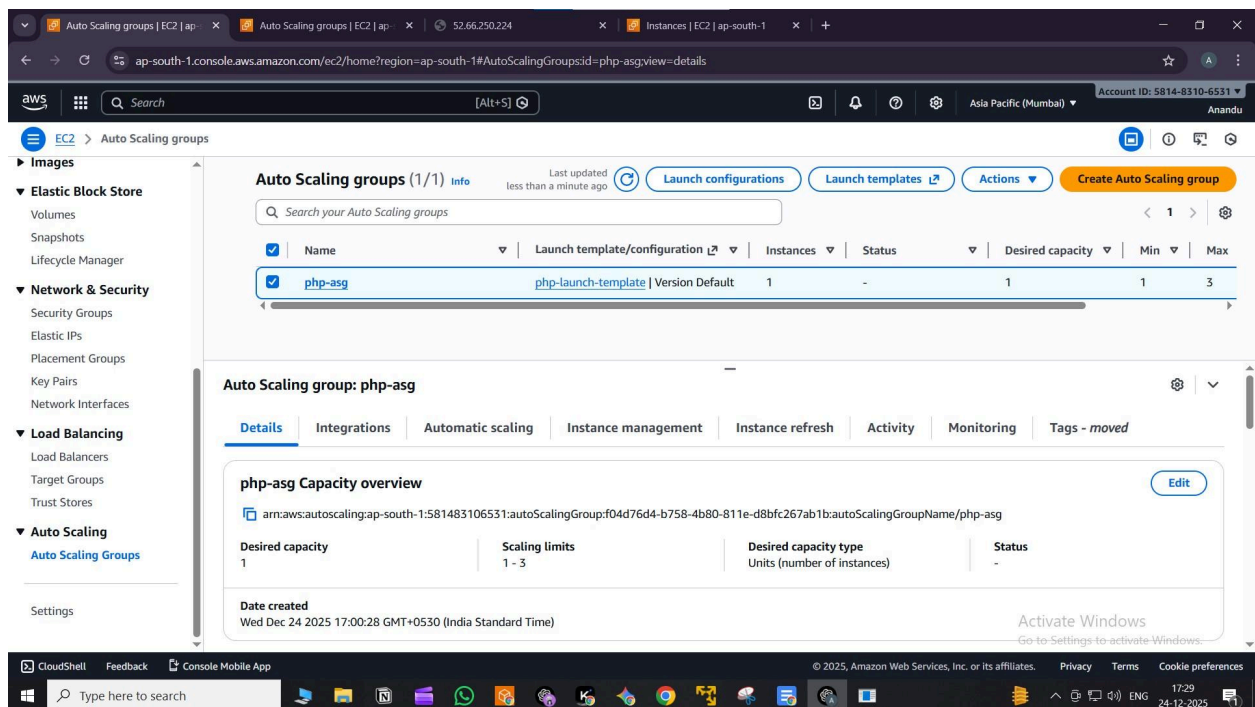


- Target value: **50%**

AWS behavior:

- CPU > 50% → Scale out
- CPU < 50% → Scale in

ASG created successfully.



## ✓ Verification

- ASG Activity shows: *Launching new EC2 instance*
- Target Group shows **2 healthy instances**

🎉 Auto Scaling is live.

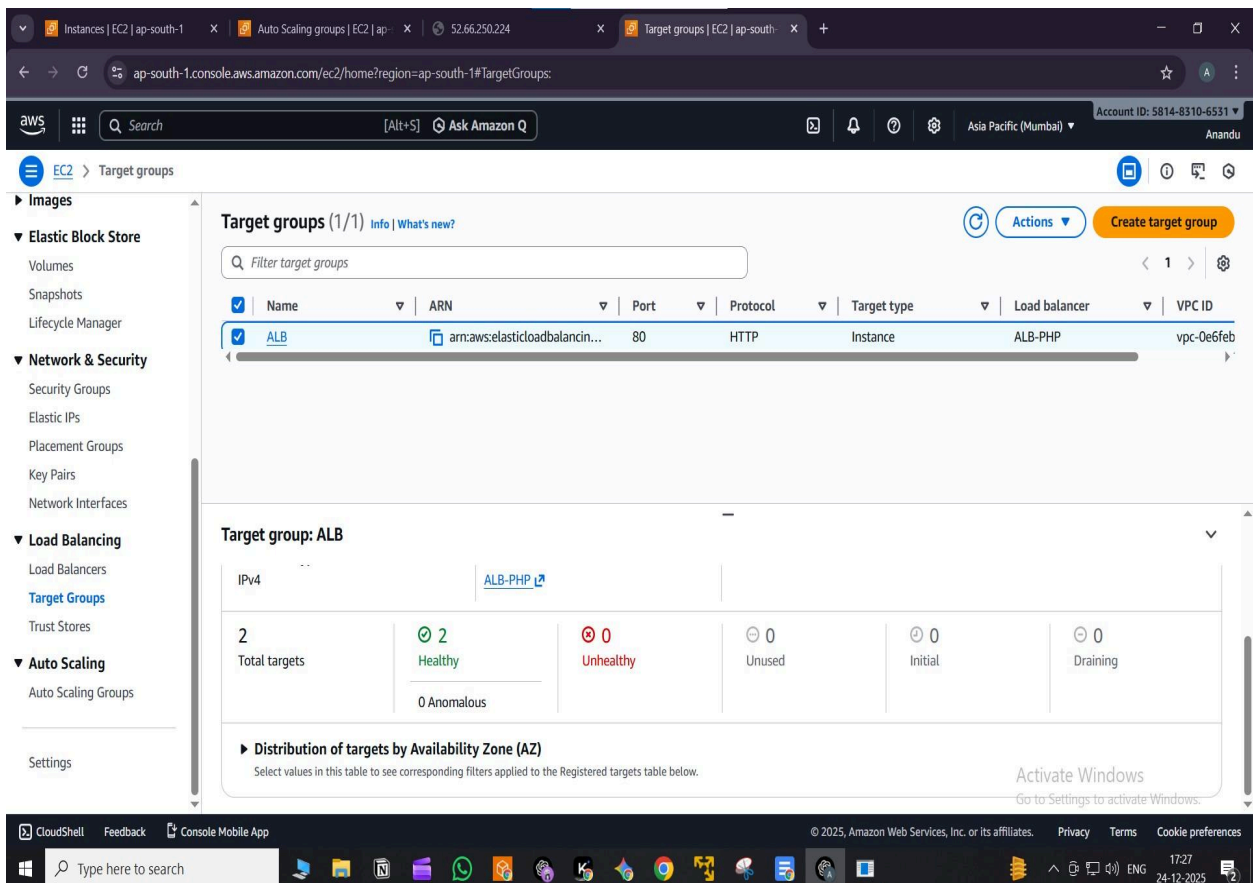


---

## 🧠 Why Two Healthy Targets Appeared

Two EC2 instances were registered:

1. **server1** – manually created
2. **php-arg instance** – auto-scaled



The screenshot displays the AWS Management Console interface for the 'Target groups' page. The left-hand navigation pane lists various AWS services, including 'Load Balancing' and 'Auto Scaling'. The main content area shows a table of target groups. A single target group named 'ALB' is listed with an ARN of 'arn:aws:elasticloadbalancing...'. Below the table, the 'Target group: ALB' section provides a detailed view of the targets. It shows 2 total targets, all of which are 'Healthy'. There are 0 'Unhealthy', 'Unused', 'Initial', and 'Draining' targets. A 'Distribution of targets by Availability Zone (AZ)' section is also visible, indicating that the targets are distributed across different availability zones. The bottom of the screenshot shows the Windows taskbar with various application icons and the system clock.

ALB marks **any healthy instance as valid**, regardless of how it was created.

---

## 🧠 WHY SCALING DID NOT TRIGGER INITIALLY



Scaling metric used:

### **Average CPU utilization of Auto Scaling Group**

Observed:

- php-asg CPU  $\approx 83\%$
- server1 CPU  $\approx 0\%$

Average CPU:

$$(83 + 0) / 2 \approx 41.5\%$$

 Scaling condition (50%) not met

 System behaved **correctly**

---

### **Deregister Manual EC2 (server1)**

**EC2 → Target Groups → ALB → Targets**

Steps:

- Select **server1**
- Click **Deregister**
- Wait ~60 seconds

Result:

- Total targets: **1**



- Healthy: 1
  - Only ASG-managed instance remains
- 

### **Create Missing Scaling Policy (Fix)**

**ASG → Automatic scaling → Create dynamic scaling policy**

- Type: Target tracking
- Metric: Average CPU utilization
- Target value: **30%** (demo purpose)

- ✓ Policy now visible
  - ✓ CloudWatch alarm auto-created
- 

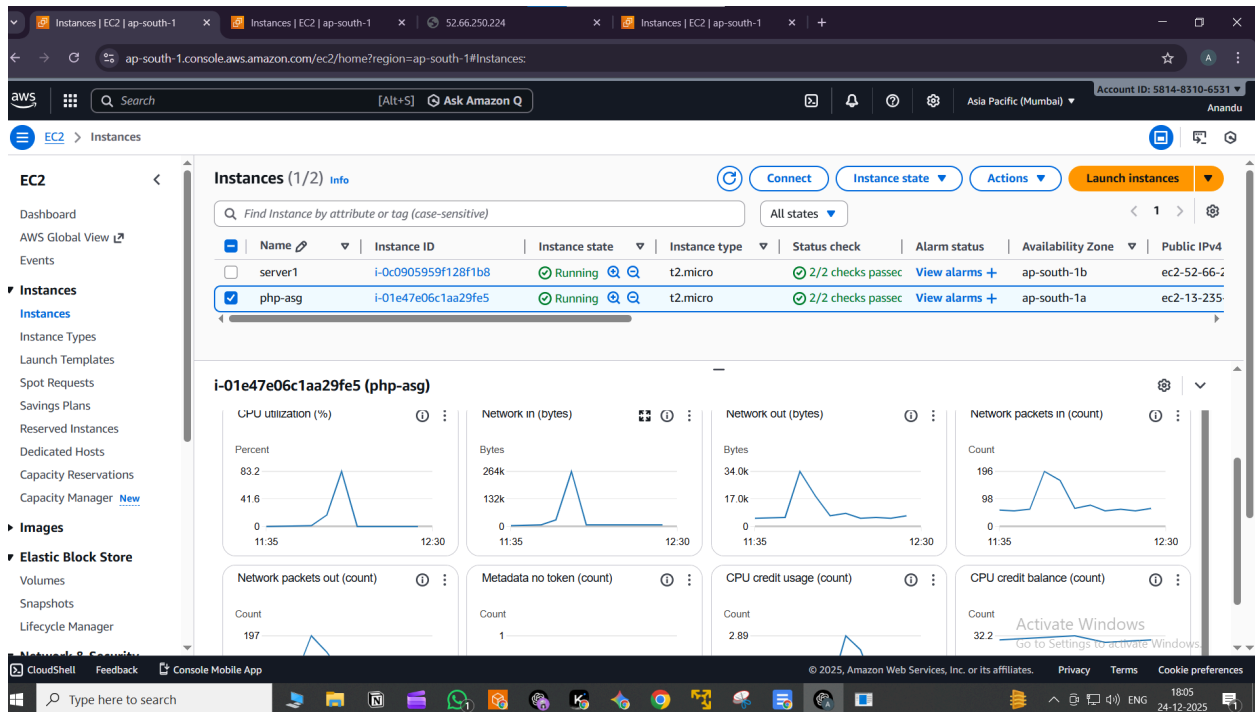
### **Load Test**

```
stress --cpu 2 --timeout 900
```

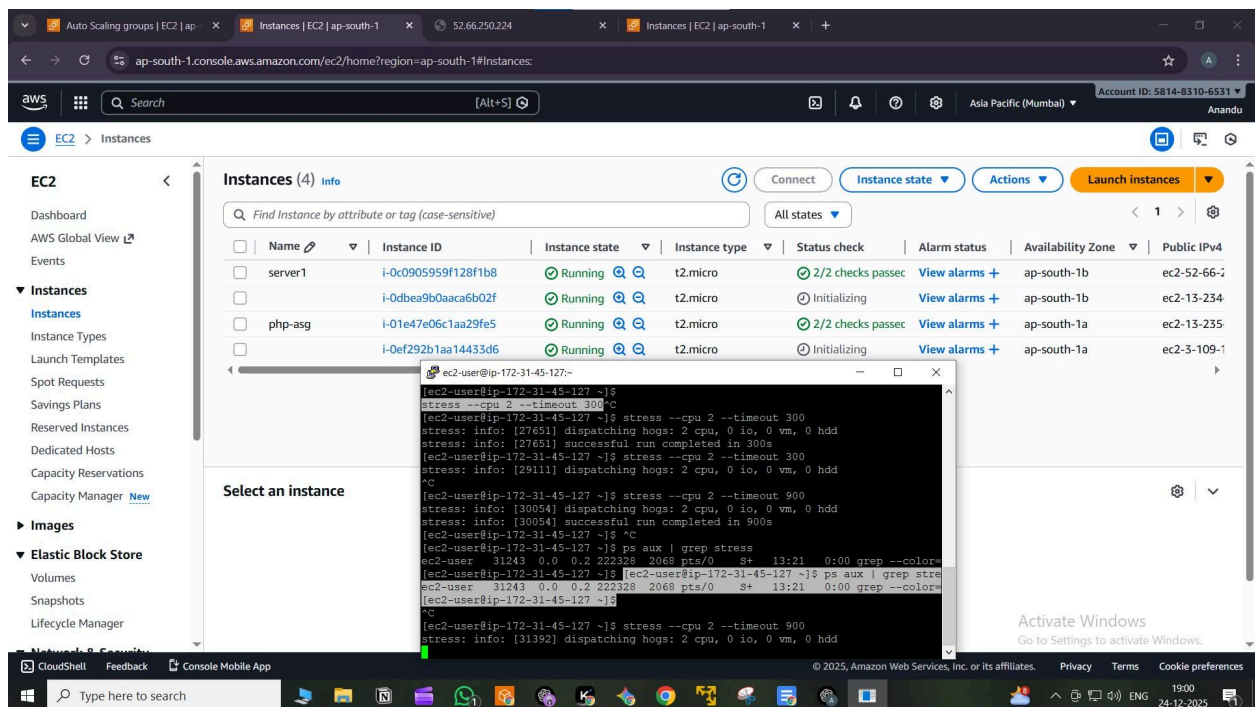
Result:

- CPU spikes above target
- ASG launches new EC2
- Target group shows additional healthy instance





✓ Auto Scaling confirmed





## **CLEANUP (COST CONTROL)**

### **1 Auto Scaling Group**

- Set Min/Desired/Max = 0
- Delete ASG

### **2 Load Balancer**

- Delete ALB

### **3 Target Group**

- Delete target group

### **4 EC2 Instances**

- Terminate all instances

### **5 Launch Template**

- Delete **all versions**

### **6 CloudWatch**

- Verify alarms removed

✓ Zero running resources

✓ Zero ongoing cost

---



## **CONCLUSION**

This project demonstrates:

- Load-balanced PHP application deployment
- Auto Scaling using CPU-based target tracking
- Real-world troubleshooting of scaling behavior
- Proper AWS resource cleanup and cost control