

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:

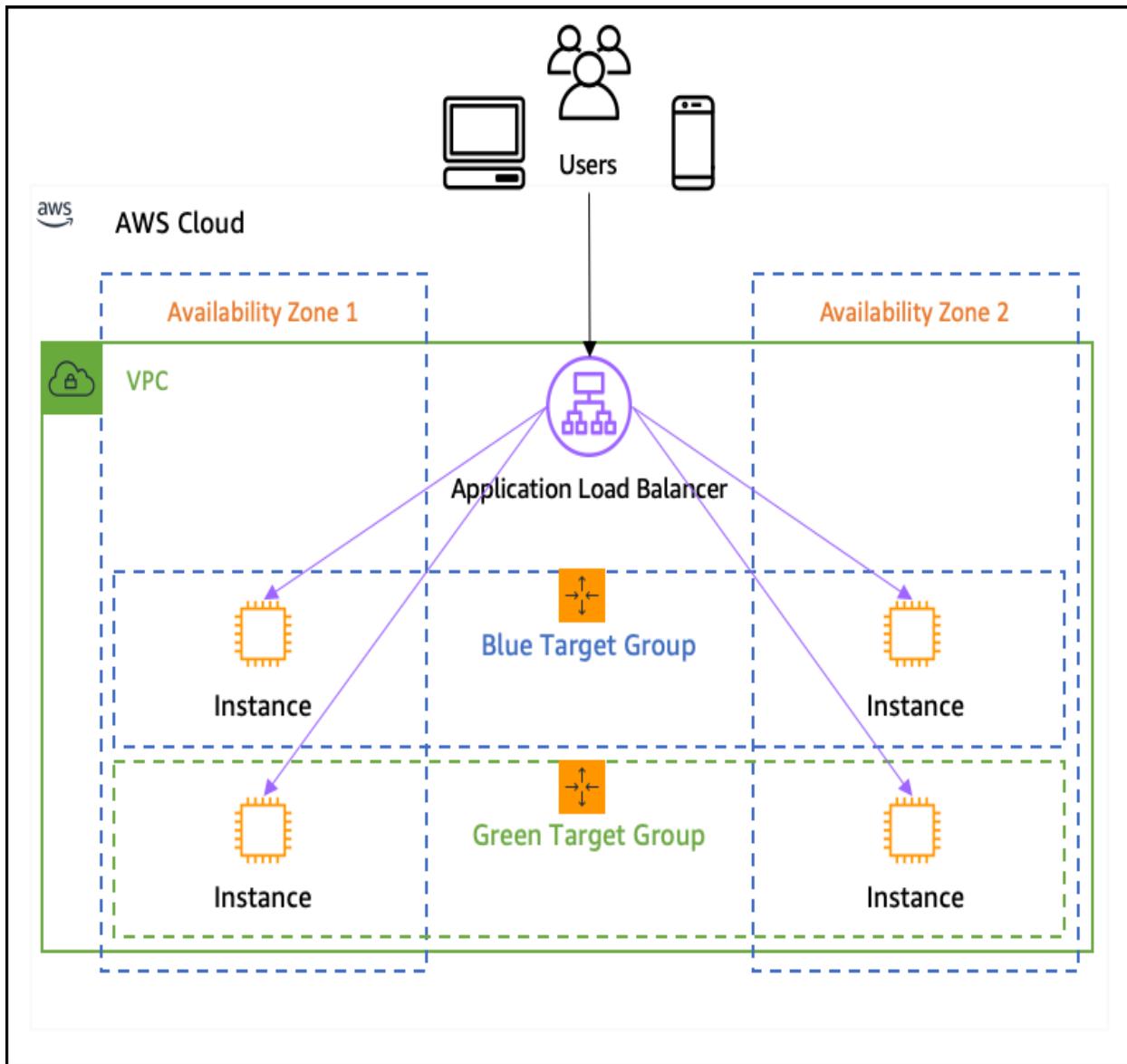
```
sudo systemctl restart httpd
```

✓ 5. Verification

- Open browser: http://EC2_PUBLIC_IP
- 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.

The screenshot shows the AWS EC2 Load Balancers console. On the left, a navigation sidebar includes options like EC2, Images, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and Settings. The main area displays a table titled 'Load balancers (1/1)'. The table has columns for Name, State, Type, Scheme, IP address type, VPC ID, and Availability Zones. One row is selected for 'ALB-PHP', which is active, an application load balancer, internet-facing, using IPv4, associated with VPC [vpc-0e6feb31f67cea745](#), and located in two availability zones. Below the table, a detailed view for 'Load balancer: ALB-PHP' is shown, with tabs for Details, Listeners and rules, Network mapping, Resource map, Security, Monitoring, Integrations, Attributes, and Capacity. The 'Details' tab is selected, displaying information such as Load balancer type (Application), Status (Active), Scheme (Internet-facing), Hosted zone ([ZP97RAFLXTNZK](#)), VPC ([vpc-0e6feb31f67cea745](#)), Availability Zones ([subnet-08eb071b861d76e72](#) in ap-south-1b), Load balancer IP address type (IPv4), and Date created (December 24, 2025, 16:04 UTC+05:30). The bottom of the screen shows the Windows taskbar with various pinned icons.

✓ 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

The screenshot shows the AWS EC2 Target Groups page. A green success message at the top left says "Successfully deregistered 1 target." Below it, the "Target groups (1/1)" section displays a single entry for "ALB". The table shows the following details:

Name	Port	Protocol	Target type	Load balancer	VPC ID
ALB	80	HTTP	Instance	ALB-PHP	vpc-0e6feb31f67cea745

Below the table, the "Target group: ALB" section provides information about health checks and shows two targets:

Instance ID	Name	Port	Zone	Health status	Health status details	Admin...	Override...
i-01e47e06c1aa29fe5	php-asg	80	ap-south-1a (a...)	Healthy	-	No override.	No over...
i-0c0905959f128f1b8	server1	80	ap-south-1b (a...)	Draining	Target deregistration i...	No override.	No over...

At the bottom of the page, there are links for CloudShell, Feedback, and Console Mobile App. The status bar at the bottom right shows "© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 18:03 24-12-2025".

✓ 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.

The screenshot shows the AWS Auto Scaling Groups page. On the left, there's a sidebar with navigation links for Images, Elastic Block Store, Network & Security, Load Balancing, and Auto Scaling (with Auto Scaling Groups selected). The main content area displays a table titled "Auto Scaling groups (1/1)". The table has columns for Name, Launch template/configuration, Instances, Status, Desired capacity, Min, and Max. A single row is shown for "php-asg" with a status of "1", "1", and "1-3". Below the table, a section titled "Auto Scaling group: php-asg" contains tabs for Details, Integrations, Automatic scaling, Instance management, Instance refresh, Activity, Monitoring, and Tags - moved. Under the "Details" tab, there's a "php-asg Capacity overview" section with fields for Desired capacity (1), Scaling limits (1-3), Desired capacity type (Units (number of instances)), and Status (-). It also shows the Date created as "Wed Dec 24 2025 17:00:28 GMT+0530 (India Standard Time)". At the bottom of the page, there's a navigation bar with CloudShell, Feedback, Console Mobile App, a search bar, and links for Activate Windows, Privacy, Terms, and Cookie preferences.

✓ 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-asg instance** – auto-scaled

The screenshot shows the AWS EC2 Target groups console. The left sidebar navigation includes 'Instances', 'Auto Scaling groups', 'Target groups' (which is selected and highlighted in blue), 'Elastic Block Store', 'Network & Security', 'Load Balancing', 'Auto Scaling', and 'Settings'. The main content area displays 'Target groups (1/1)'. A table lists one target group: 'ALB'. The table columns are Name, ARN, Port, Protocol, Target type, Load balancer, and VPC ID. The 'ALB' row has a blue underline under 'Name'. Below the table, the 'Target group: ALB' section shows metrics for IPv4 targets: 2 Total targets (2 Healthy, 0 Unhealthy, 0 Anomalous), 0 Unused, 0 Initial, and 0 Draining. At the bottom, there's a section titled 'Distribution of targets by Availability Zone (AZ)' with a note: 'Select values in this table to see corresponding filters applied to the Registered targets table below.'

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

Instances (1/2) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
server1	i-0c0905959f128f1b8	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-52-66-2
php-asg	i-01e47e06c1aa29fe5	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1a	ec2-13-235

i-01e47e06c1aa29fe5 (php-asg)

CPU utilization (%)
Percent
83.2
41.6
0 11:35 12:30

Network in (bytes)
Bytes
264k
132k
0 11:35 12:30

Network out (bytes)
Bytes
34.0k
17.0k
0 11:35 12:30

Network packets in (count)
Count
198
98
0 11:35 12:30

Network packets out (count)
Count
197
197
0 11:35 12:30

Metadata no token (count)
Count
1
1
0 11:35 12:30

CPU credit usage (count)
Count
2.89
2.89
0 11:35 12:30

CPU credit balance (count)
Count
32.2
32.2
0 11:35 12:30

✓ Auto Scaling confirmed

Instances (4) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
server1	i-0c0905959f128f1b8	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-52-66-2
	i-0dbea9b0aca6b02f	Running	t2.micro	Initializing	View alarms +	ap-south-1b	ec2-13-234
	i-01e47e06c1aa29fe5	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1a	ec2-13-235
	i-0ef292b1aa14433d6	Running	t2.micro	Initializing	View alarms +	ap-south-1a	ec2-3-109-1

Select an instance

```
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 300s
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 300s
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 300s
stress: info: [27651] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [27651] successful run completed in 300s
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 300s
stress: info: [29111] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
^C
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 900
stress: info: [30054] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [30054] successful run completed in 900s
[ec2-user@ip-172-31-45-127 ~]$ ^C
[ec2-user@ip-172-31-45-127 ~]$ ps aux | grep stress
ec2-user 31243 0.0 0.2 22328 2068 pts/0+ 13z1 0:00 grep --color=auto
[ec2-user@ip-172-31-45-127 ~]$ ps aux | grep stress
ec2-user 31243 0.0 0.2 22328 2068 pts/0+ 13z1 0:00 grep --color=auto
[ec2-user@ip-172-31-45-127 ~]$ ^C
[ec2-user@ip-172-31-45-127 ~]$ stress --cpu 2 --timeout 900
stress: info: [31392] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

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