AWS Auto Scaling Nginx Web Server Architecture

A scalable, fault-tolerant, and cost-efficient web server deployment on AWS using EC2 instances in private subnets behind an Application Load Balancer (ALB) in public subnets. Integrated an Auto Scaling Group (ASG) with CloudWatch alarms to dynamically scale based on CPU utilization. Configured Nginx on Ubuntu and automated provisioning using EC2 user data.

📌 Architecture Overview

VPC with public/private subnets across 2 AZs

Nginx EC2 instances in private subnets

ALB in public subnets routing traffic to EC2s

Auto Scaling Group with CloudWatch scaling policies

Bastion host for secure SSH access

NAT Gateway for outbound internet access

🛠️ Technologies Used

AWS EC2

Auto Scaling Group

Elastic Load Balancer (ALB)

CloudWatch

SSM / Bastion Host

Nginx on Ubuntu

User Data scripting

🧱 Step-by-Step Setup

1️⃣ VPC & Subnets

VPC CIDR: 10.0.0.0/16

Subnets:

PublicSubnet1 (10.0.1.0/24) - AZ us-east-1a

PublicSubnet2 (10.0.3.0/24) - AZ us-east-1b

PrivateSubnet1 (10.0.2.0/24) - AZ us-east-1a

PrivateSubnet2 (10.0.4.0/24) - AZ us-east-1b

Attach an Internet Gateway to your VPC

Update Route Table to allow 0.0.0.0/0 → Internet Gateway

Associate public subnets with this route table

2️⃣ Security Groups

ALB-SG

Inbound: HTTP (80) from 0.0.0.0/0

Outbound: Default (All traffic)

EC2-SG

Inbound: HTTP (80) from ALB-SG

Outbound: Default

3️⃣ Launch EC2 with Nginx (Private Subnet)

AMI: Ubuntu

Subnet: PrivateSubnet1

Security Group: EC2-SG

User Data Script:

bash

Copy

Edit

#!/bin/bash

apt-get update -y

apt-get install nginx -y

systemctl start nginx

systemctl enable nginx

4️⃣ Set Up Application Load Balancer

Type: Internet-facing

Listener: HTTP (port 80)

Subnets: PublicSubnet1 & PublicSubnet2

Security Group: ALB-SG

Target Group:

Type: Instance

Protocol: HTTP

Port: 80

Register EC2 instances

5️⃣ Create Auto Scaling Group (ASG)

Create Launch Template

AMI: Ubuntu

Instance Type: t2.micro

Security Group: EC2-SG

User Data: (Nginx install script above)

Create Auto Scaling Group

VPC: Select your custom VPC

Subnets: PrivateSubnet1 & PrivateSubnet2

Attach to Target Group

Enable ELB Health Checks

6️⃣ Set Up Dynamic Scaling Policies

Scale Out (CPU > 70%)

Policy Type: Step scaling

CloudWatch Alarm:

Metric: ASG CPUUtilization

Threshold: > 70%

Action: Increase desired capacity by 1

Scale In (CPU < 40%)

Policy Type: Step scaling

CloudWatch Alarm:

Metric: ASG CPUUtilization

Threshold: ≤ 40%

Action: Decrease desired capacity by 1

7️⃣ Test the Architecture

Open ALB DNS URL → Should load default Nginx welcome page

SSH into EC2 via Bastion Host

Simulate CPU load to trigger ASG scaling:

bash

Copy

Edit

sudo apt install stress -y

stress --cpu 2 --timeout 300

Observe CloudWatch triggering scale-out policy

🧪 Troubleshooting Example: ALB Showing 502 Bad Gateway

Problem:

ALB returned 502 Bad Gateway and EC2 targets appeared unhealthy

Diagnosis:

SSH into EC2 instance → nginx was not installed

Launch Template was missing user data script

Health checks failed due to no HTTP service running

Fix:

Installed nginx manually for confirmation:

bash

Copy

Edit

sudo apt update

sudo apt install -y nginx

sudo systemctl start nginx

sudo systemctl enable nginx

Updated Launch Template with correct user data

Re-deployed EC2 via ASG with fixed script

ALB health checks passed, and Nginx welcome page loaded successfully

✅ Outcome

Achieved automated horizontal scaling using CPU metrics

Ensured high availability across two AZs

Fully isolated private infrastructure with public-facing ALB

Hands-on experience with VPC networking, autoscaling, load balancing, CloudWatch, and NAT Gateway