What is a Load Balancer?

A **Load Balancer** is a service that distributes incoming network traffic across multiple servers to ensure no single server is overwhelmed, improving application availability, fault tolerance, and scalability. It helps handle high traffic loads efficiently and ensures better performance by routing requests to the best available instance.

How Load Balancer Works:

1. A client sends a request to your application.
2. The Load Balancer receives the request and determines which server (EC2 instance) should handle it.
3. The request is forwarded to the selected server based on factors like health, availability, and predefined rules.
4. The server processes the request and returns the response.
5. The Load Balancer ensures that if one server fails, traffic is rerouted to healthy instances.

Step-by-Step Guide to Setting Up Load Balancing & Auto Scaling in AWS

Step 1: Launch Multiple EC2 Instances

1. Go to **AWS Console → EC2 Dashboard**.
2. Click **Launch Instance** and create 2-3 instances with the same AMI and configurations.
3. Install and start a web server (Apache or Nginx) on each instance.

   sudo yum update -y

   sudo yum install -y httpd

   sudo systemctl start httpd

   sudo systemctl enable httpd

   echo "Hello from Server 1" | sudo tee /var/www/html/index.html
4. Repeat the process for other instances, changing "Server 1" to "Server 2", etc.

Step 2: Create a Load Balancer

1. Go to **AWS Console → EC2 Dashboard → Load Balancers**.
2. Click **Create Load Balancer** and choose **Application Load Balancer**.
3. Configure:
   - Name: MyApp-ALB

- Scheme: Internet-facing
- Listener: HTTP:80
- VPC: Select the VPC of your EC2 instances.
- Subnets: Choose at least 2 subnets in different Availability Zones.

4. Click **Next: Configure Security Settings** and skip (for HTTP).
5. Click **Next: Configure Security Groups**:
    - Create or use an existing security group that allows HTTP (port 80).
6. Click **Next: Configure Routing**:
    - Create a new **Target Group** (MyApp-TG).
    - Target Type: Instance
    - Health Check Path: /
    - Protocol: HTTP
7. Click **Next: Register Targets**, select the EC2 instances, and click **Add to registered**.
8. Click **Next: Review and Create → Create Load Balancer**.

Step 3: Configure Auto Scaling

1. Go to **AWS Console → EC2 Dashboard → Auto Scaling Groups**.
2. Click **Create Auto Scaling Group**.
3. Configure:
    - Name: MyApp-ASG
    - Launch Template: Create or choose an existing one.
    - Instance Type: Same as your EC2 instances.
    - Desired Capacity: 2 (Initial number of instances).
    - Min Size: 2
    - Max Size: 5
    - Attach to an existing **Target Group** (MyApp-TG).
4. Configure Scaling Policies:
    - Choose **Target Tracking Scaling**.
    - Metric: **CPU Utilization** (Target: 50%).
    - AWS will automatically scale instances based on CPU usage.

5. Click **Next: Review and Create → Create Auto Scaling Group**.

Step 4: Test Load Balancer & Auto Scaling

1. Get the **DNS Name** of your Load Balancer from **EC2 → Load Balancers**.
2. Open a browser and visit http://<Load-Balancer-DNS>.
3. Refresh multiple times to see responses from different servers.
4. Simulate high traffic using stress testing tools (ab, siege) to trigger Auto Scaling.

Benefits of Load Balancer & Auto Scaling

✅ **High Availability:** Traffic is distributed across multiple instances.
✅ **Fault Tolerance:** If an instance fails, traffic is redirected to healthy ones.
✅ **Scalability:** Automatically adds/removes instances based on demand.
✅ **Cost Optimization:** Ensures resources are used efficiently.

Application Load Balancer (ALB) vs. Network Load Balancer (NLB)

AWS provides two main types of load balancers:

1. Application Load Balancer (ALB)

💡 **Best for:** Web applications needing intelligent routing (Layer 7 - Application Layer).

Key Features:

- Routes traffic based on **URL, hostname, headers, and query parameters**.
- Supports **path-based and host-based routing**.
- Works well with **microservices (ECS, EKS)** and **WebSockets**.
- Integrates with **AWS WAF** for security.

Use Cases:

✅ Websites or APIs with multiple services (/user, /admin).
✅ Microservices architectures.
✅ Applications requiring authentication (AWS Cognito).

Example:

A web app routes /user to **User Service**, /admin to **Admin Service**.

2. Network Load Balancer (NLB)

💡 **Best for:** High-performance, low-latency traffic routing (Layer 4 - Transport Layer).

Key Features:

- Handles **TCP, UDP, and TLS** traffic.
- Supports **millions of requests per second** with **low latency**.
- Preserves **static IP addresses** and **original client IP**.

Use Cases:

✅ Real-time applications (VoIP, gaming, financial services).
✅ Load balancing **database connections**.

Example:

A gaming app using **low-latency TCP connections** for real-time multiplayer gaming.

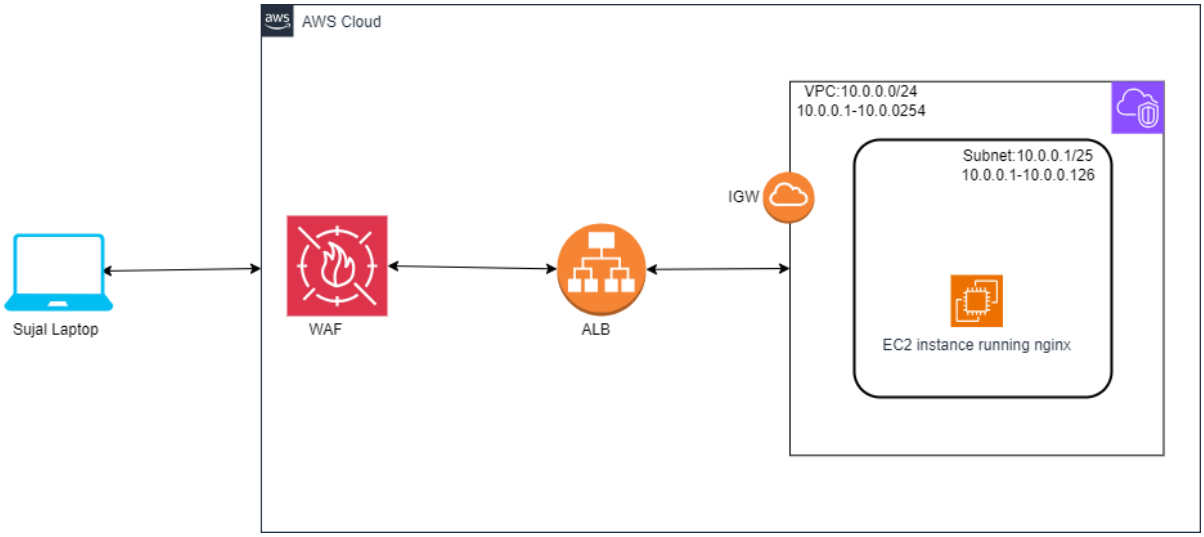
ALB vs. NLB: Key Differences

Feature	ALB (Layer 7)	NLB (Layer 4)
Traffic Type	HTTP/HTTPS	TCP, UDP, TLS
Routing	URL, headers	IP-based
Best for	Websites, APIs	Real-time apps, Databases
Performance	Moderate	High (millions of requests/sec)

Choose ALB for intelligent HTTP routing, **NLB** for ultra-fast TCP/UDP handling. 🚀

WAF: Web Application Firewall

In today's digital landscape, web applications are vulnerable to various security threats such as SQL injection, cross-site scripting (XSS), and malicious bots. As a DevOps enthusiast, I recently explored **AWS WAF (Web Application Firewall)** to protect a sample web application hosted on AWS. In this blog, I'll share my hands-on experience setting up **AWS WAF** and how you can implement it for your own applications.



What is AWS WAF?

AWS WAF is a cloud-based firewall that helps protect web applications by filtering and monitoring HTTP/HTTPS requests based on security rules. It safeguards against common threats and allows you to control access based on IP addresses, request patterns, and rate limiting.

Key Features:

- ✅ Protection against common web attacks (SQL injection, XSS, etc.)
- ✅ Customizable rule sets
- ✅ Real-time monitoring and logging
- ✅ Integration with AWS services (ALB, API Gateway, CloudFront)

Project: Implementing AWS WAF for Web Application Protection

Scenario:

We need to secure a web application hosted on an EC2 instance using AWS WAF. The goal is to filter malicious requests and block unauthorized access.

Step 1: Deploy a Sample Web Application on EC2

1. **Launch an EC2 Instance:**
    - Select **Amazon Linux 2** or **Ubuntu** AMI.
    - Choose instance type (e.g., t2.micro).
    - Configure security groups to allow HTTP (port 80) and SSH (port 22).
    - Launch and connect via SSH.
2. **Install a Web Server:**

sudo apt update -y # For Ubuntu

sudo apt install -y apache2 # Install Apache on Ubuntu

sudo systemctl start httpd # Start Apache

sudo systemctl enable httpd # Enable Apache on startup

3. **Create a Sample Web Page:**

echo "<h1>Welcome to My Secure Web App</h1>" | sudo tee /var/www/html/index.html

sudo systemctl restart httpd

✅ Test by accessing http://<EC2-Public-IP> in your browser.

Step 2: Configure AWS WAF

1. **Navigate to AWS WAF Console:**
   - Go to **AWS WAF → Web ACLs → Create Web ACL**.
2. **Set Up Web ACL:**
   - Name: MyWebApp-WAF
   - Region: Choose the region where your EC2 instance or ALB is hosted.
   - AWS Resource: Select **Application Load Balancer (ALB)** or CloudFront.
3. **Add Protection Rules:**
   - **AWS Managed Rules:**
     - AWSManagedRulesCommonRuleSet (Protects against SQL Injection & XSS)
     - AWSManagedRulesAmazonIpReputationList (Blocks known bad IPs)
   - **Custom Rules (Example):**
     - Block a specific IP address:

       Condition: IF Source IP = <Blocked-IP>

       Action: BLOCK

     - Block requests with certain patterns:

       Condition: IF URI Path contains "/admin"

       Action: BLOCK
4. **Attach Web ACL to Your Application Load Balancer (ALB):**
   - Go to **EC2 → Load Balancers**.
   - Select your ALB → **Edit Web ACL** → Attach MyWebApp-WAF.

- Save changes.

Step 3: Testing AWS WAF

1. **Access Your Web Application Normally:**
   - Open http://<EC2-Public-IP> → Page should load successfully.
2. **Test SQL Injection Protection:**
   - Try accessing http://<EC2-Public-IP>?id=1' OR '1'='1' --.
   - AWS WAF should **block** this request.
3. **Test IP Blocking:**
   - If your IP is blocked, you should see a **403 Forbidden** response.
4. **Monitor Logs:**
   - Open **AWS WAF Console → Web ACLs →** MyWebApp-WAF **→ View Requests**.
   - Analyze logs for allowed/blocked requests.