

High Availability Web Application with AWS Elastic Load Balancer (BookNow Project)

What is Ec2 Web Server?

An EC2 web server is a virtual server running on Amazon's Elastic Compute Cloud (EC2) that is set up with web server software like Apache or Nginx. It allows you to host websites or applications on the cloud without needing physical hardware. EC2 makes it easy to start, scale, and manage servers on demand, making it flexible and cost-effective for web hosting.

What is Elastic Load Balancer?

Elastic Load Balancer (ELB) is an AWS service that automatically distributes incoming application or website traffic across multiple targets such as EC2 instances, containers, and IP addresses. It improves availability, scalability, and fault tolerance by ensuring no single server is overwhelmed when traffic increases.

Key features of Elastic Load Balancer:

High Availability → if one server fails, traffic is automatically routed to healthy servers.

Scalability → supports adding/removing servers based on demand.

Flexibility → supports different types of load balancers (Application, Network, Gateway).

Project Description:

Project: BookNow Website Hosting with AWS Elastic Load Balancer

This project demonstrates deploying and scaling a sample web application called **BookNow** on AWS. The application was developed with basic functionality and UI customization. To showcase **load balancing in action**, two different versions of the website were deployed:

- **Instance 1:** Blue background theme

- **Instance 2:** Black background theme

The application was hosted on **Amazon EC2 instances** and integrated with an **Elastic Load Balancer (ELB)**. The ELB distributes incoming user traffic across the instances, ensuring that no single server is overloaded. When accessing the Load Balancer DNS, users may see either the blue or black themed page depending on which instance handles the request.

This setup demonstrates how AWS services can be used for **high availability, scalability, fault tolerance, and efficient resource utilization** in real-world applications.

Steps:

1) Launch two EC2 instances (Amazon Linux)

1. Go to **EC2 → Instances → Launch instances**
2. **Name and tags**
 - Instance 1: My Elastic Instance 1

Name and tags [Info](#)

Name

My Elastic instance 1

Add additional tags

- Instance 2: My Elastic Instance 2 (set “Number of instances” to **2** so both launch together, or launch twice.)
3. **Application and OS Images (AMI): Amazon Linux (Amazon Linux 2 or 2023)**

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose **Browse more AMIs**.

Search our full catalog including 1000s of application and OS images

Recents Quick Start

| | | | | | | |
|--------------|-------|--------|---------|---------|------------|--------|
| Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Linux | Debian |
| | | | | | | |

Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

| | |
|--|--------------------|
| Amazon Linux 2023 kernel-6.1 AMI ami-0861f4e788f5069dd (64-bit (x86), uefi-preferred) / ami-0fad8318b9405c6fb (64-bit (Arm), uefi) Virtualization: hvm ENA enabled: true Root device type: ebs | Free tier eligible |
|--|--------------------|

4. Instance type: t2.micro (Free Tier)

▼ Instance type [Info](#) | [Get advice](#)

Instance type

| | |
|--|--------------------------|
| t2.micro | Free tier eligible |
| Family: t2 | t2.micro |
| 1 GiB Memory | Current generation: true |
| On-Demand Windows base pricing: 0.017 USD per Hour | |
| On-Demand RHEL base pricing: 0.0268 USD per Hour | |
| On-Demand Linux base pricing: 0.0124 USD per Hour | |
| On-Demand Ubuntu Pro base pricing: 0.0142 USD per Hour | |
| On-Demand SUSE base pricing: 0.0124 USD per Hour | |

[Additional costs apply for AMIs with pre-installed software](#)

All generations Compare instance types

5. Key pair:

- If you want to SSH with a client: **Create a new key pair** (recommended).
- If you don't want to manage keys, you can use **EC2 Instance Connect (browser)** later, but still keep **Port 22** open from your IP in the security group (next step).

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select

Create new key pair

○

6. Network settings (Security group) → Create a new SG named web-sg:

- **Inbound rules**
 - **HTTP | Port 80 | Source 0.0.0.0/0**
 - **SSH | Port 22 | Source My IP** (recommended)
- **Outbound rules: All traffic** (default)

Allow HTTP traffic from the internet

To set up an endpoint, for example when creating a web server

7. Storage: leave defaults (8–10 GB gp3/gp2 is fine)

▼ Configure storage [Info](#) [Advanced](#)

1x GiB Root volume, 3000 IOPS, Not encrypted

[Add new volume](#)

8. Launch instances → Wait until running and Status checks = 2/2.

[Cancel](#)[Launch instance](#)[Preview code](#)

Check Two instances are Running

| | | | | | |
|--------------------------|--------------------|---------------------|---------|----------|-----|
| <input type="checkbox"/> | My Elastic inst... | i-04d678dacba805385 | Running | t2.micro | ... |
| <input type="checkbox"/> | My Elastic inst... | i-0f2f80a009219bc88 | Running | t2.micro | ... |

◆ Install Apache and Deploy Pages (on instance 1)

For Instance 1 (Blue background):

1. Connect to your EC2 instance using **EC2 Instance Connect** or your SSH client.

2. Switch to the root user.

3. Update the server packages.

4. Install the Apache web server.

```
systemctl start httpd
```

```
systemctl enable httpd
```

5. Enable Apache .

```
systemctl enable httpd
```

6. Start the Apache service.

```
systemctl start httpd
```

7. Verify that Apache is running with the status command.

8. Navigate to /var/www/html and create a new file named index.html.

9. Paste the **blue background HTML code** into the file.

10.Save and close the file.

```
[root@ip-172-31-2-71 ~]# httpd -v
Server version: Apache/2.4.64 (Amazon Linux)
Server built: Jul 15 2025 00:00:00

systemctl start httpd
systemctl enable httpd
systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
systemctl status httpd

# cd /var/www/html

        case 'auto': return 'fas fa-car';
        case 'dental': return 'fas fa-tooth';
        case 'fitness': return 'fas fa-dumbbell';
        case 'consultation': return 'fas fa-graduation-cap';
        default: return 'fas fa-calendar-check';
    }
}

function formatDate(dateString) {
    const date = new Date(dateString);
    return date.toLocaleDateString('en-US', {
        weekday: 'short',
        month: 'short',
        day: 'numeric'
    });
}

function showToast(message, type = 'success') {
    // In a real app, this would show a toast notification
    alert(` ${type.toUpperCase()}: ${message}`);
}
</script>
</body>
</html>
```

11.Set the file permission to be readable by everyone (644).

```
# chmod 644 index.html
```

```
[root@ip-172-31-11-235 ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
   Active: active (running) since Sun 2025-08-31 14:46:29 UTC; 35s ago
     Docs: man:httpd.service(8)
 Main PID: 26908 (httpd)
    Status: "Total requests: 2; Idle/Busy workers 100/0;Requests/sec: 0.069; Bytes served/sec:
 Tasks: 177 (limit: 1111)
 Memory: 13.2M
      CPU: 76ms
 CGroup: /system.slice/httpd.service
         └─26908 /usr/sbin/httpd -DFOREGROUND
              ├─26909 /usr/sbin/httpd -DFOREGROUND
              ├─26910 /usr/sbin/httpd -DFOREGROUND
              ├─26911 /usr/sbin/httpd -DFOREGROUND
              └─26912 /usr/sbin/httpd -DFOREGROUND

Aug 31 14:46:29 ip-172-31-11-235.ap-south-1.compute.internal systemd[1]: Starting httpd.service
Aug 31 14:46:29 ip-172-31-11-235.ap-south-1.compute.internal systemd[1]: Started httpd.service
Aug 31 14:46:29 ip-172-31-11-235.ap-south-1.compute.internal httpd[26908]: Server configured, 1
[root@ip-172-31-11-235 ~]# cd /var/www/html
[root@ip-172-31-11-235 html]# vi index.html
[root@ip-172-31-11-235 html]# chmod 644 index.html
[root@ip-172-31-11-235 html]# systemctl restart httpd
[root@ip-172-31-11-235 html]# █
```

For Instance 2 (Black background):

1. Connect to the second EC2 instance.
2. Switch to the root user.
3. Update the server packages.
4. Install the Apache web server.
5. Enable Apache to start automatically on boot.
6. Start the Apache service.
7. Verify that Apache is running.
8. Navigate to /var/www/html and create a new file named index.html.
9. Paste the **black background HTML code** into the file.
10. Save and close the file.
11. Set the file permission to 644.

Note: As same as for the instance 2 also like paste your black colr background code here

3) Create a Target Group

1. Go to **EC2 → Target groups → Create target group**

2. Target type: Instances

Basic configuration

Settings in this section can't be changed after the target group is created.

Choose a target type

Instances

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.

3. Name: myloadtarget (or your myloadtargetgroup)

Target group name

MyLoadTargetgroup

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

4. Protocol: HTTP, Port: 80

Protocol version

HTTP1

Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.

5. VPC: your default or the one where instances run

6. Health checks

- Protocol: **HTTP**
- Path: /
- (Leave thresholds/intervals default)

[Cancel](#)

[Next](#)

7. Next → Register targets

- Select **both instances** → **Include as pending below** → **Create target group**
- Wait 30–60s then check **Targets tab** → should become **healthy**.

Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

| Available instances (2) | | | | |
|--------------------------|---------------------|-----------------------|---------|------------------|
| Filter instances | | Instance ID | Name | State |
| <input type="checkbox"/> | i-04d678dacba805385 | My Elastic instance 1 | Running | launch-wizard-16 |
| <input type="checkbox"/> | i-0f2f80a009219bc88 | My Elastic instance 2 | Running | launch-wizard-16 |

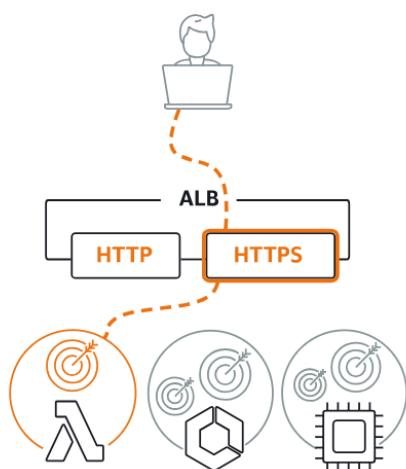
Select include as pending as below and create

4) Create an Application Load Balancer (ALB)

1. EC2 → Load balancers → Create load balancer → Application Load Balancer



Application Load Balancer Info



2. Name: my-elastic-load-balancer

Basic configuration

Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.

MyBooknow Load Balancer

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

3. Scheme: Internet-facing | IPv4

Scheme | [Info](#)

Scheme can't be changed after the load balancer is created.

Internet-facing

- Serves internet-facing traffic.
- Has public IP addresses.
- DNS name resolves to public IPs.
- Requires a public subnet.

4. **Network mapping:** Select your VPC and **at least two subnets in different AZs** (e.g., ap-south-1a & ap-south-1b).

ap-south-1a (aps1-az1)

ap-south-1b (aps1-az3)

ap-south-1c (aps1-az2)

5. **Security groups:** Create/select an SG like alb-sg that allows:

- **Inbound:** HTTP 80 from **0.0.0.0/0**
- **Outbound: All traffic**

Basic details

Security group name [Info](#)

MyLoadSecuritygroup

Name cannot be edited after creation.

Description [Info](#)

Allow HTTP traffic

Inbound rules [Info](#)

Type [Info](#) Protocol [Info](#) Port range [Info](#) Source [Info](#)

HTTP TCP 80 Any... 0.0.0.0/0 [X](#)

Add rule

Outbound rules [Info](#)

Type [Info](#) Protocol [Info](#) Port range [Info](#) Destination [Info](#)

All traffic All All Cust... 0.0.0.0/0 [X](#)

Add rule

Create security group

6. Listeners and routing:

- Listener: **HTTP : 80**
- Default action: **Forward to** your target group **myloadtarget**

Default action [Info](#)

Forward to *Select a target group* [▼](#)

[Create target group](#)

Ports for the selected instances
Ports for routing traffic to the selected instances.

80
1-65535 (separate multiple ports with commas)

[Include as pending below](#)

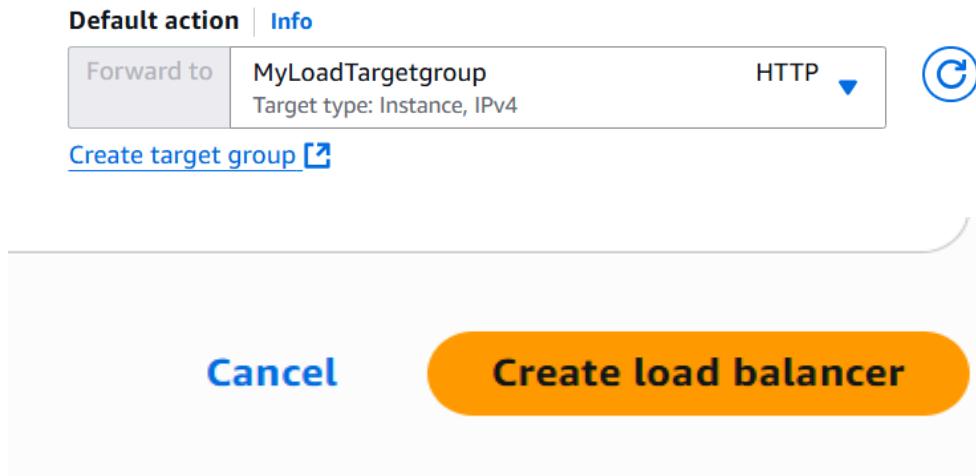
2 selections are now pending below. Include more or register targets when ready.

Review targets

Targets (2)

Filter targets [▼](#) Show only pending

| Instance ID | Name | Port | State | Security groups | Zone |
|---------------------|-----------------------|------|----------------------|------------------|-------------|
| i-04d678dacba805385 | My Elastic instance 1 | 80 | Running | launch-wizard-16 | ap-south-1b |
| i-0f2f80a009219bc88 | My Elastic instance 2 | 80 | Running | launch-wizard-16 | ap-south-1b |



7. Click **Create load balancer** and wait until **State = Active**.

5) Test the Load Balancer

1. Open the ALB's DNS name from **Load balancer → Description**

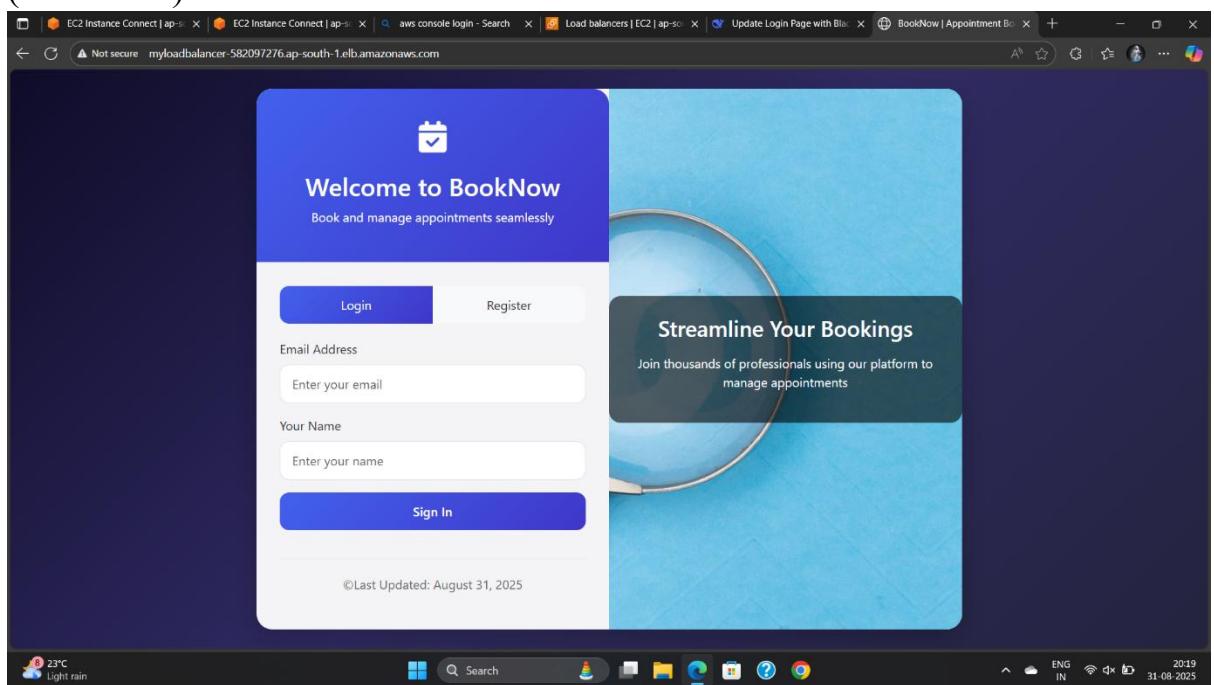
- Looks like: my-elastic-load-balancer-123456.ap-south-1.elb.amazonaws.com

DNS name [Info](#)

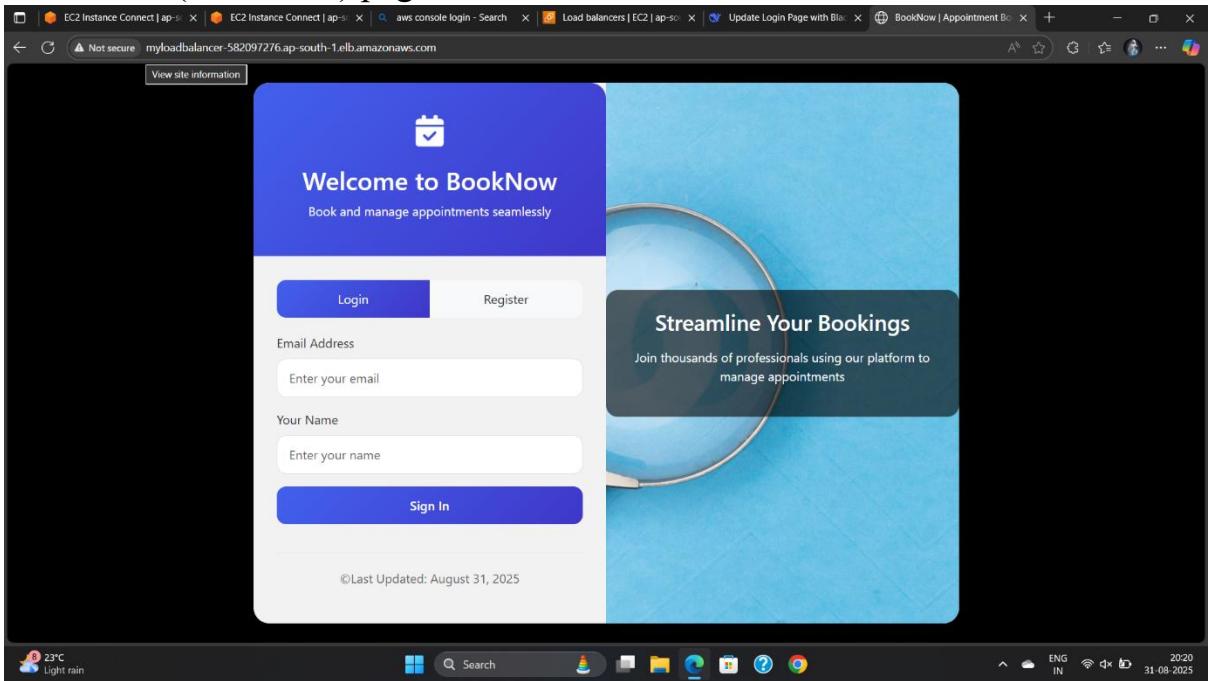
MyLoadBalancer-582097276.ap-south-1.elb.amazonaws.com (A Record)

○

2. Refresh the page a few times: you should alternate between the **blue** (Instance 1)



3. And **black** (Instance 2) pages.



📌 Project Outcome

The **BookNow Website Hosting with AWS Elastic Load Balancer** project successfully demonstrated deploying and scaling a web application on cloud infrastructure. By hosting the application on two separate EC2 instances with different themes (blue and black backgrounds), the project visually showcased how the **Elastic Load Balancer distributes traffic across multiple servers**.

The outcome proved:

- **High availability** – Application remained accessible even if one instance was down.
- **Load distribution** – Traffic was evenly balanced across instances.
- **Scalability** – New instances can be added easily to handle increased traffic.
- **Fault tolerance** – Healthy targets automatically served requests.

This project highlighted the practical use of **AWS services** for real-world cloud application deployment, making it a valuable academic and portfolio project.

More Projects on GitHub

Explore more of my cloud, AWS, Linux, and DevOps projects on my GitHub profile:

 [Sanjaykumar-P \(Sanjaykumar-P\)](#)

Portfolio

Visit my personal portfolio website to see detailed case studies, projects, and achievements:

 [Sanjaykumar P - Linux & Cloud Specialist](#)