# 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

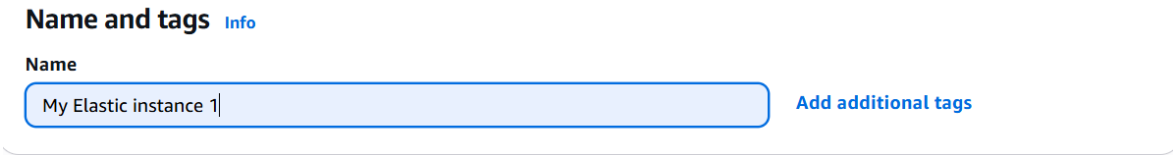
The application was hosted on **Amazon EC2 instances** and integrated with an **Elastic Load Balancer (ELB)**. The ELB distributes incoming user traffic across the instances, ensuring that no single server is overloaded. When accessing the Load Balancer DNS, users may see either the blue or black themed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

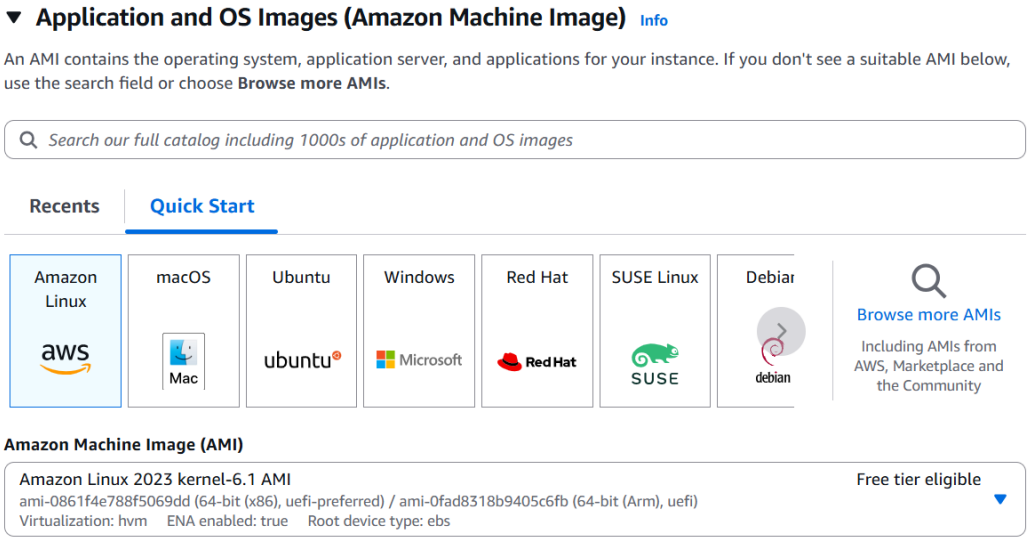
Steps:

## 1) Launch two EC2 instances (Amazon Linux)

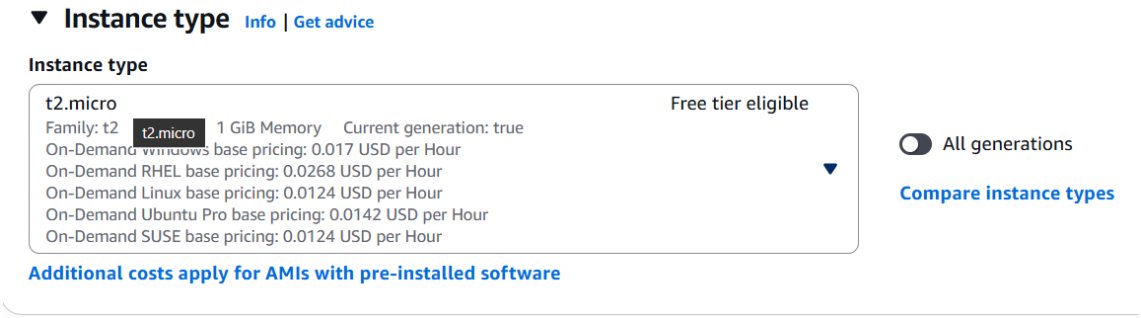
1. Go to **EC2 → Instances → Launch instances**
2. **Name and tags**
    - Instance 1: My Elastic Instance 1



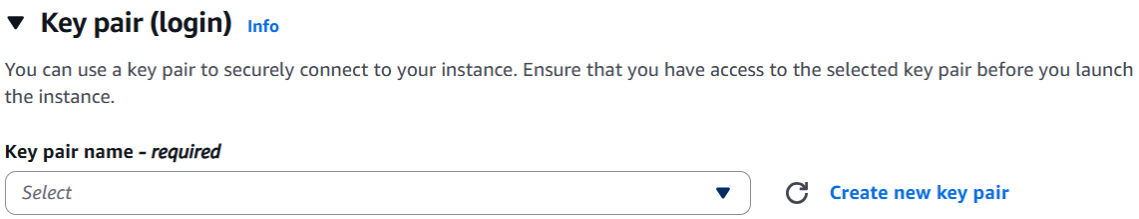

    - Instance 2: My Elastic Instance 2 (set “Number of instances” to **2** so both launch together, or launch twice.)
3. **Application and OS Images (AMI)**: **Amazon Linux** (Amazon Linux 2 or 2023)



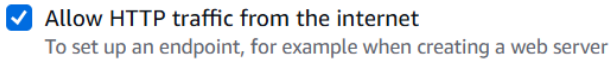
4. **Instance type**: **t2.micro** (Free Tier)



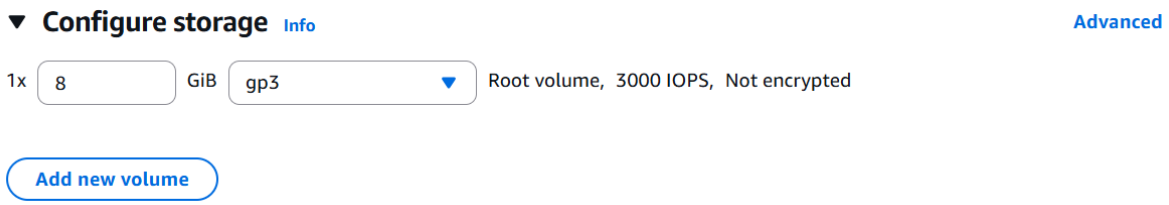

5. **Key pair**:
   - If you want to SSH with a client: **Create a new key pair** (recommended).
   - If you don't want to manage keys, you can use **EC2 Instance Connect (browser)** later, but still keep **Port 22** open from your IP in the security group (next step).



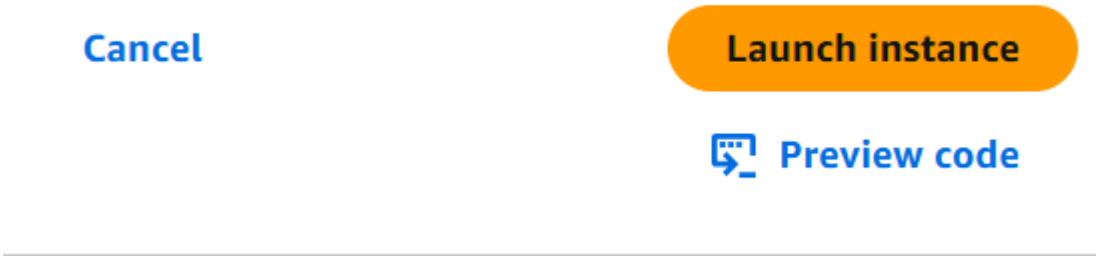

6. **Network settings (Security group)** → Create a new SG named web-sg:
   - **Inbound rules**
     - **HTTP** | Port **80** | Source **0.0.0.0/0**
     - **SSH** | Port **22** | Source **My IP** (recommended)
   - **Outbound rules**: **All traffic** (default)




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




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




Check Two intances are Running


- **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
temd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
systemctl status httpd
```

```
# cd /var/www/html
```

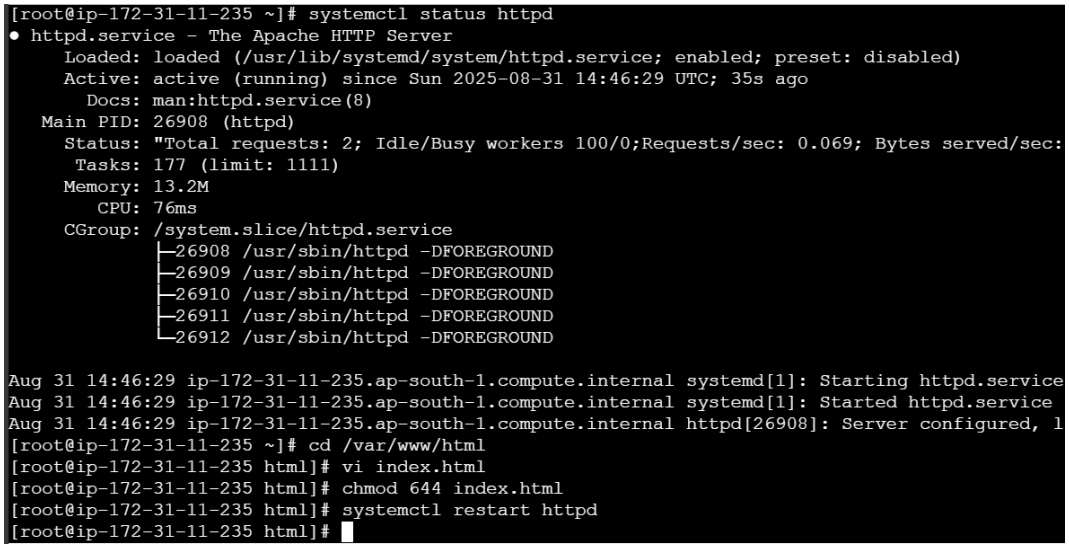
```
                case 'auto': return 'fas fa-car';
                case 'dental': return 'fas fa-tooth';
                case 'fitness': return 'fas fa-dumbbell';
                case 'consultation': return 'fas fa-graduation-cap';
                default: return 'fas fa-calendar-check';
        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

        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
```




**For Instance 2 (Black background):**

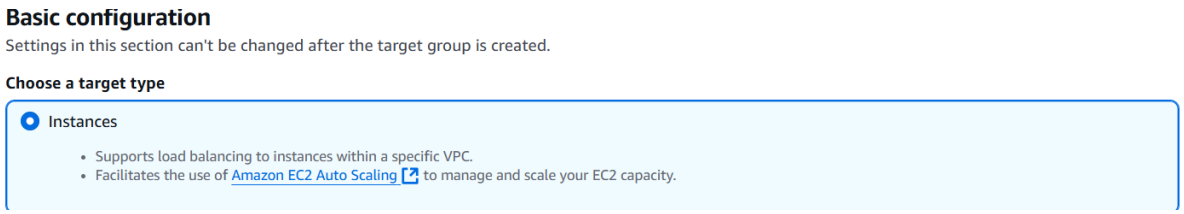
1. Connect to the second EC2 instance.
2. Switch to the root user.
3. Update the server packages.
4. Install the Apache web server.
5. Enable Apache to start automatically on boot.
6. Start the Apache service.
7. Verify that Apache is running.
8. Navigate to /var/www/html and create a new file named 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

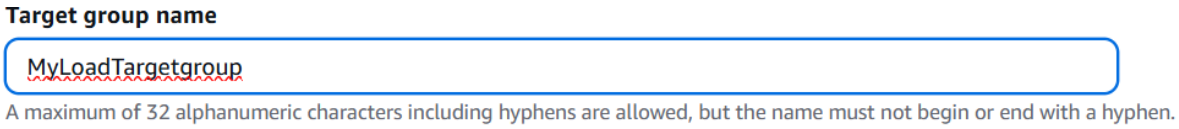
### 3) Create a Target Group

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

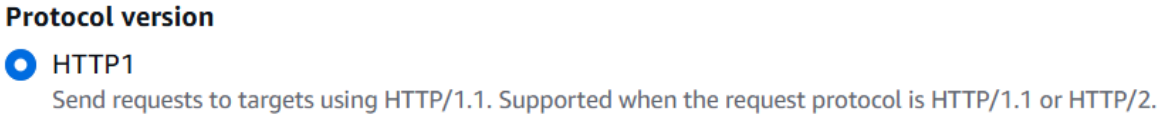
2. **Target type**: **Instances**




3. **Name**: myloadtarget (or your myloadtargetgroup)

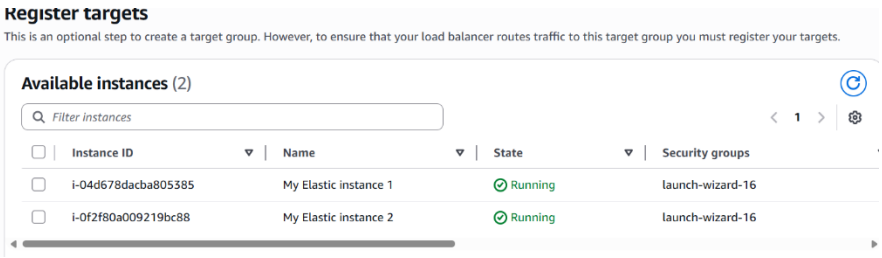



4. **Protocol**: HTTP, **Port**: 80




5. **VPC**: your default or the one where instances run
6. **Health checks**
   - Protocol: **HTTP**
   - Path: /
   - (Leave thresholds/intervals default)

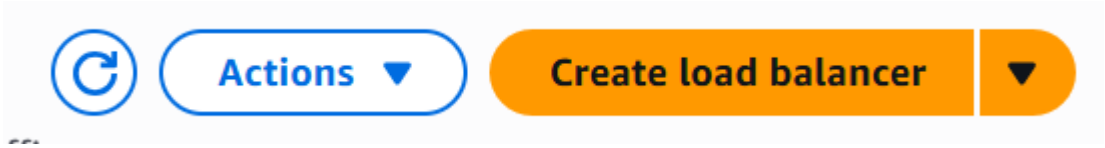

7. **Next → Register targets**
   - Select **both instances → Include as pending below → Create target group**
   - Wait 30–60s then check **Targets tab** → should become **healthy**.

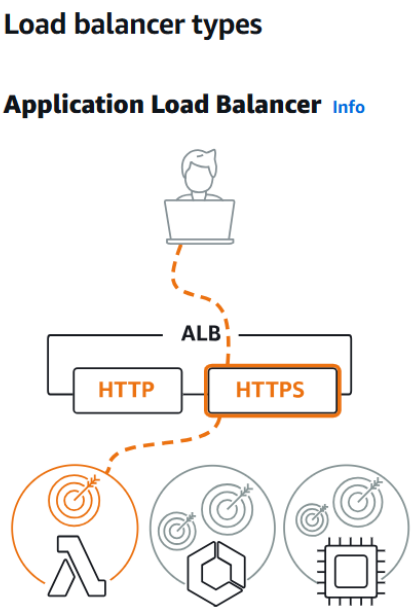



Select include as pending as below and create

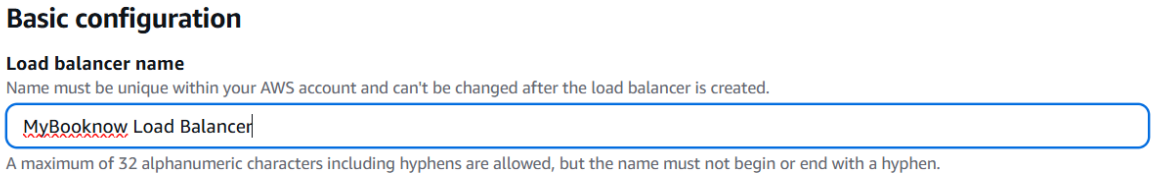
## 4) Create an Application Load Balancer (ALB)

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

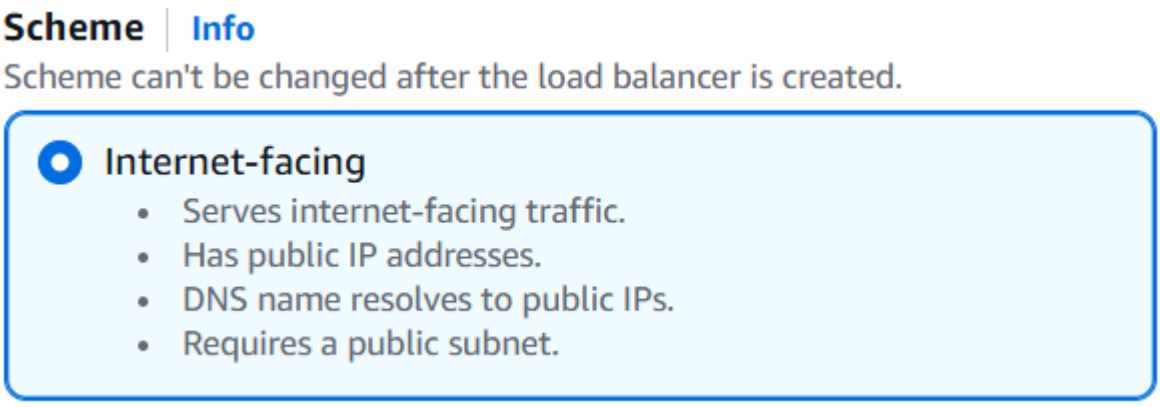


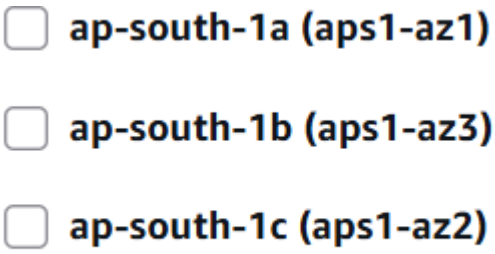

2. **Name**: my-elastic-load-balancer



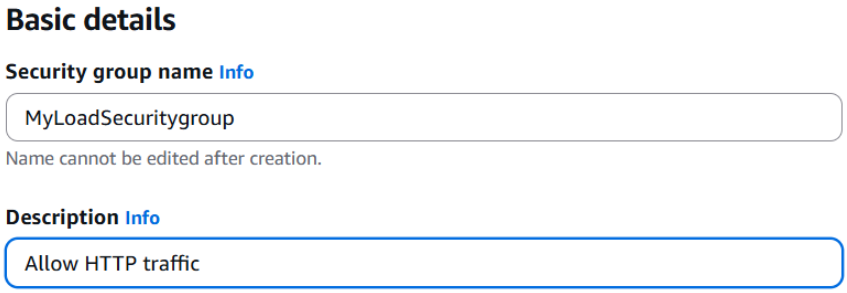

3. **Scheme**: **Internet-facing | IPv4**

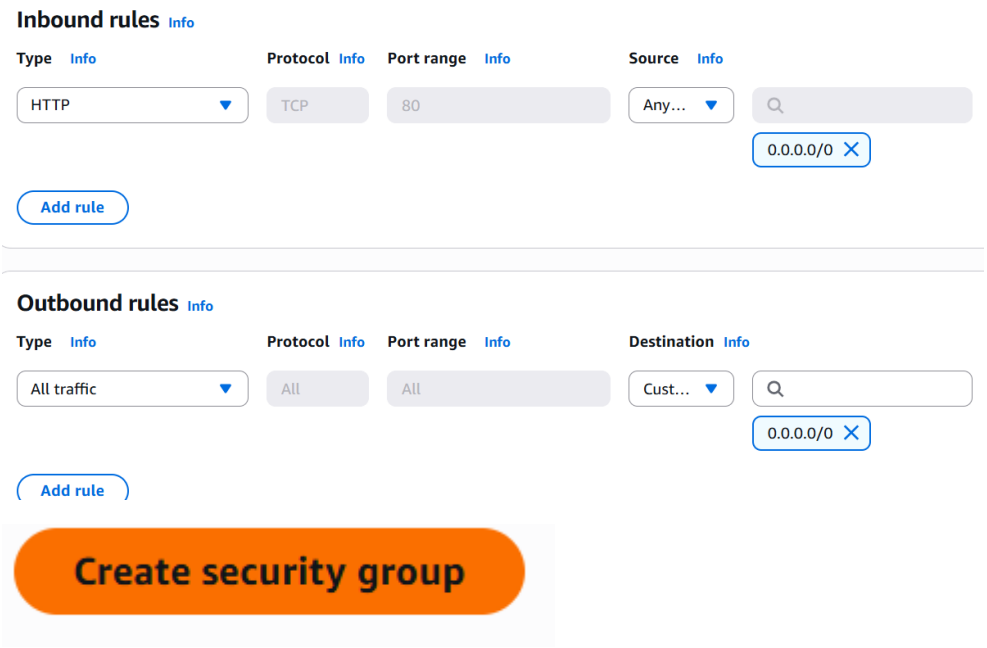



4. **Network mapping**: Select your **VPC** and **at least two subnets in different AZs** (e.g., ap-south-1a & ap-south-1b).

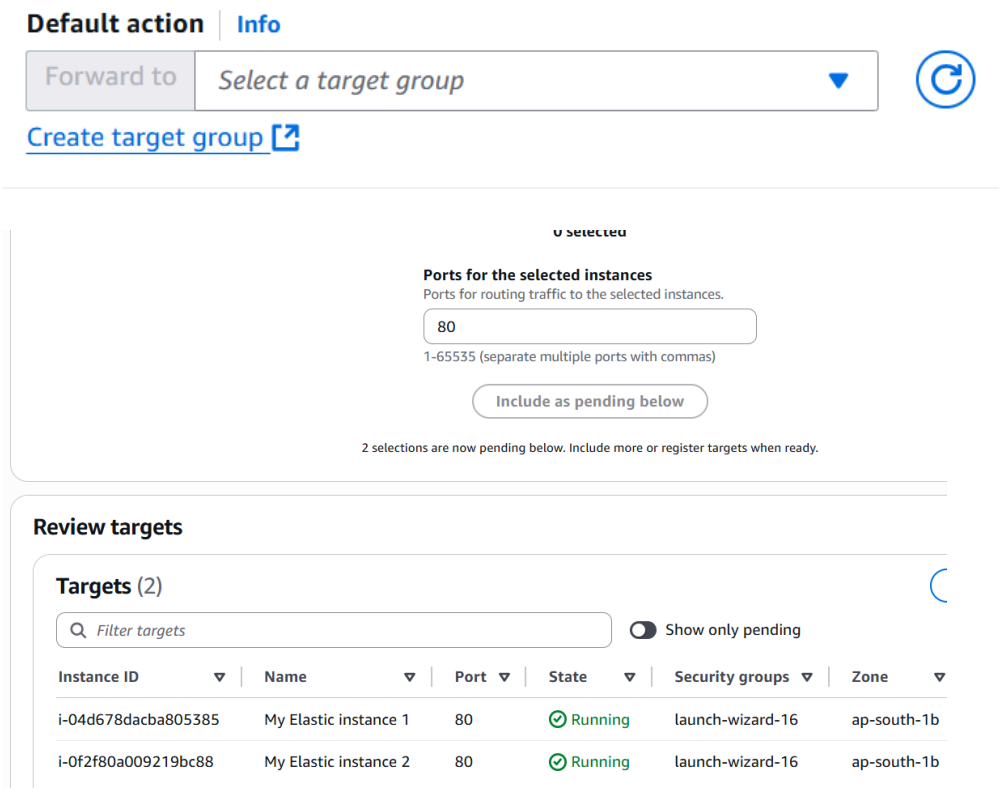


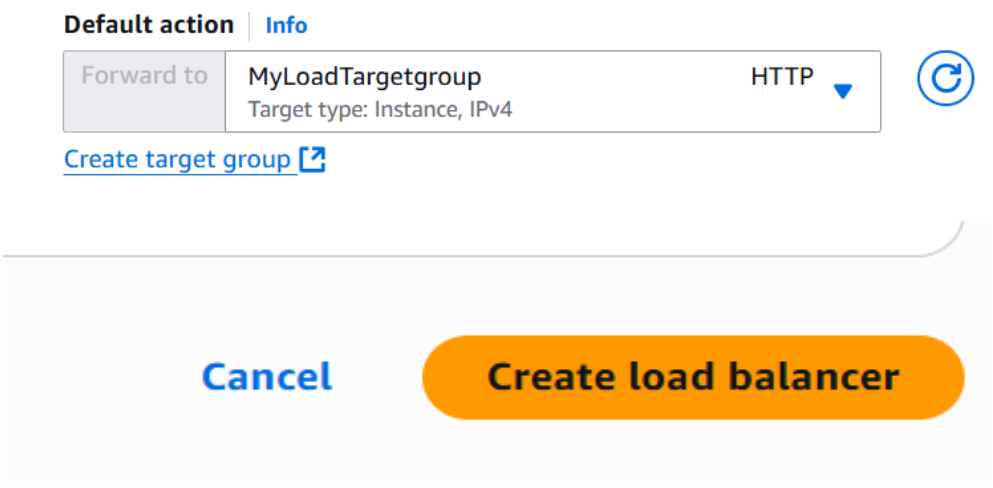

5. **Security groups**: Create/select an SG like alb-sg that allows:
   - **Inbound**: HTTP **80** from **0.0.0.0/0**
   - **Outbound**: **All traffic**






6. **Listeners and routing**:
   - Listener: **HTTP : 80**
   - Default action: **Forward to** your target group **myloadtarget**

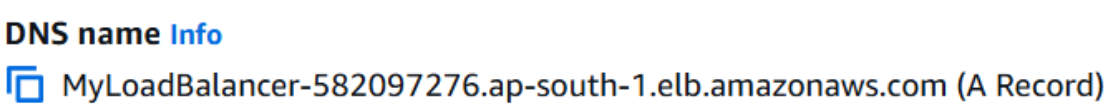





7. Click **Create load balancer** and wait until **State = Active**.

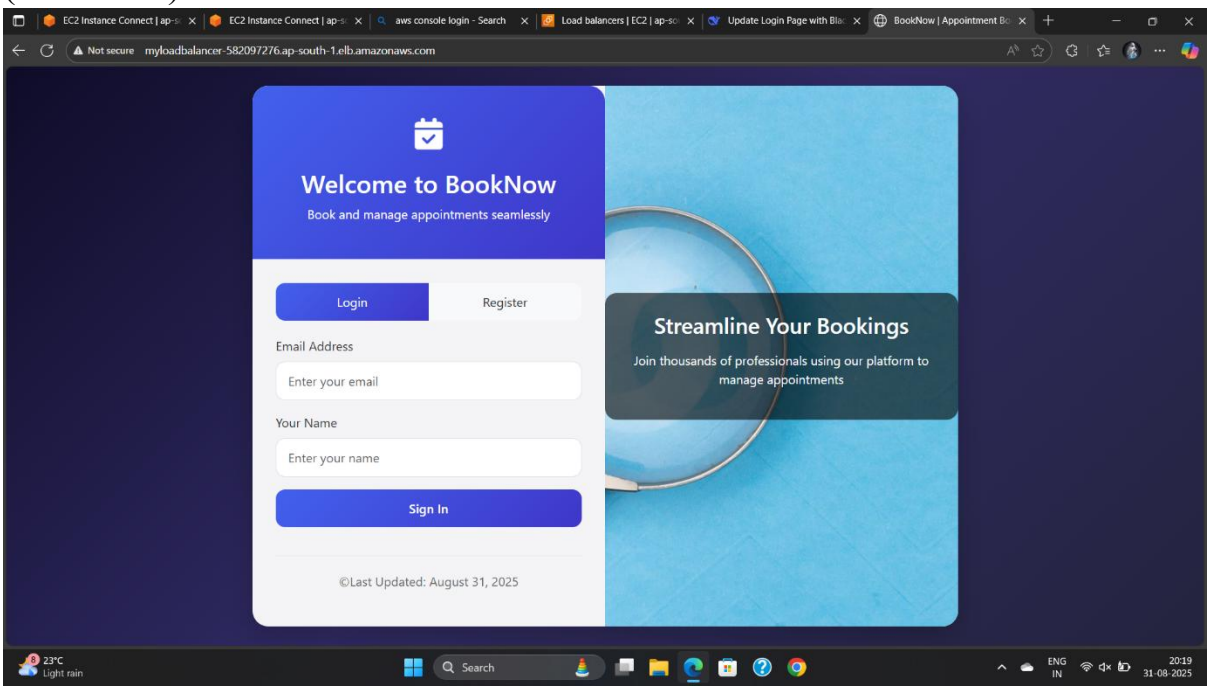
## 5) Test the Load Balancer

1. Open the ALB's **DNS name** from **Load balancer → Description**
   - Looks like: my-elastic-load-balancer-123456.ap-south-1.elb.amazonaws.com

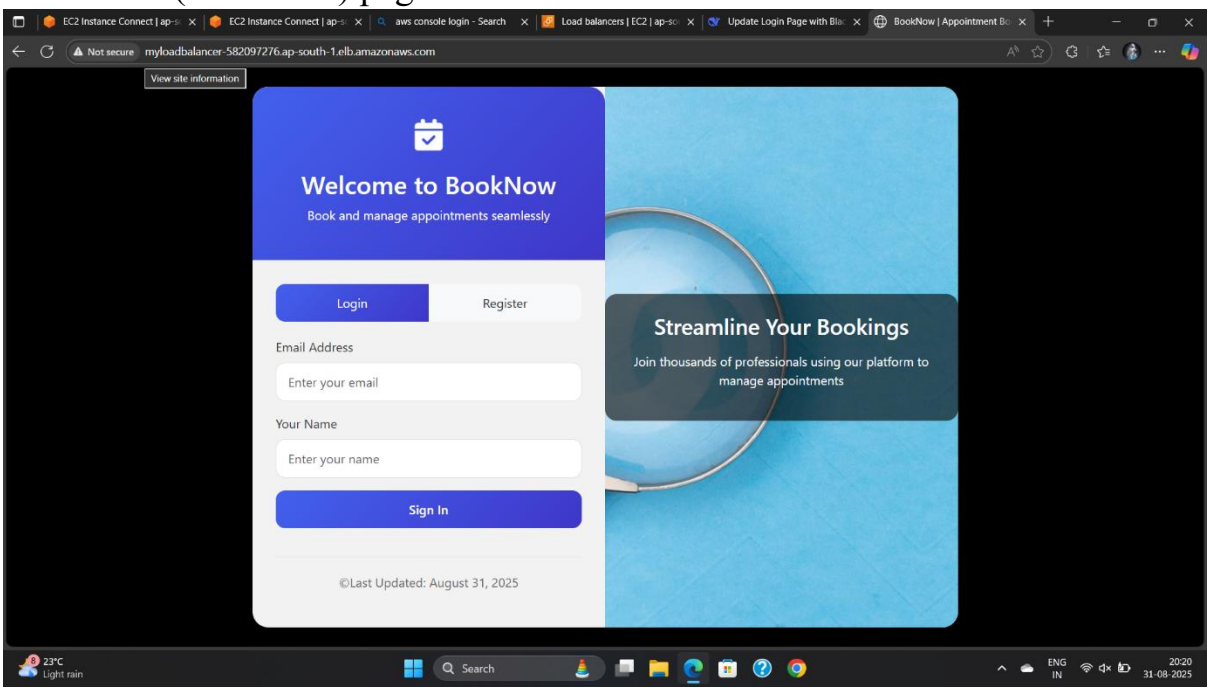



   -

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