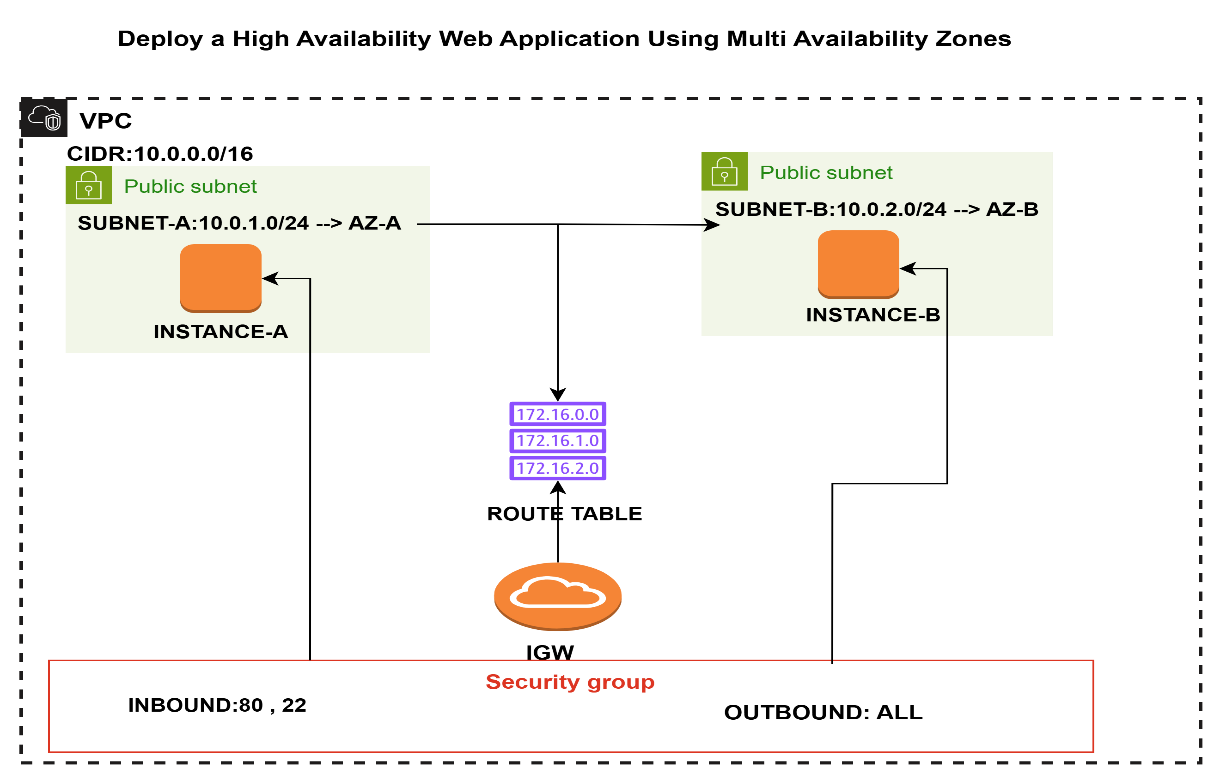
**Deploy a Highly Available Web Application Using Multi-AZ Setup**

**Multi AZ:** Multi-AZ architecture involves deploying your resources across multiple Availability Zones within an AWS Region. multiple Availability Zones (AZs) in AWS gives you some serious reliability and resilience muscle.

**Architecture Diagram:**



**Configuration Steps:**

Step 1: Create a VPC with Two Public Subnets (AZ-a, AZ-b)

1. Create a VPC (CIDR: 10.0.0.0/16)
2. Create two public subnets:
   * Subnet-A (10.0.1.0/24) → AZ-a
   * Subnet-B (10.0.2.0/24) → AZ-b
3. Attach an Internet Gateway (IGW)
4. Associate both subnets with a route table that routes 0.0.0.0/0 → IGW

Step 2: Create Security Group

Allow:

* **Inbound**: HTTP (80), SSH (22) from your IP
* **Outbound**: All

Step 3: Launch EC2 Instances in Each AZ

1. Launch **EC2-1** in Subnet-A
2. Launch **EC2-2** in Subnet-B
3. Use **Amazon Linux 2 AMI**
4. Assign public Ips

Install Apache on both:

sudo yum install httpd -y

sudo systemctl start httpd

sudo systemctl enable httpd

echo "Welcome from EC2 in AZ-a" | sudo tee /var/www/html/index.html # EC2-1

echo "Welcome from EC2 in AZ-b" | sudo tee /var/www/html/index.html #EC2-2

Note: I Have written sample code for small website using html and css

**create one index file in both servers for writing sample html code:**

. sudo nano /var/www/html/index.html

HTML CODE:

<!DOCTYPE html>

<html>

<head>

<title>CloudTech Net - Student Details</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #eef2f3;

padding: 20px;

}

.container {

background-color: white;

padding: 25px;

border-radius: 8px;

max-width: 600px;

margin: auto;

box-shadow: 0 2px 8px rgba(0,0,0,0.1);

}

h2 {

text-align: center;

color: #2e86c1;

}

.details {

margin-top: 20px;

}

.details p {

line-height: 1.6;

}

</style>

</head>

<body>

<div class="container">

<h2>CloudTech Net - Student Profile</h2>

<div class="details">

<p><strong>Name:</strong> Lakshmana Rao</p>

<p><strong>Phone Number:</strong> +91-9494000000</p>

<p><strong>IT Experience:</strong> 3 years</p>

<p><strong>Domain Knowledge:</strong> AWS, Terraform, Python Automation</p>

<p><strong>Trainer Name:</strong> Rajesh Naidu</p>

</div>

</div>

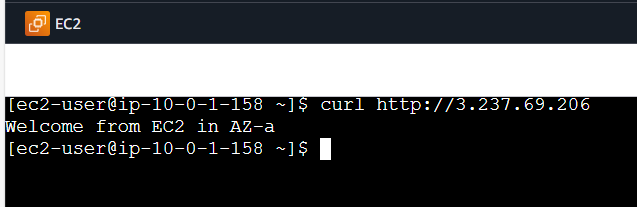
</body>

</html>

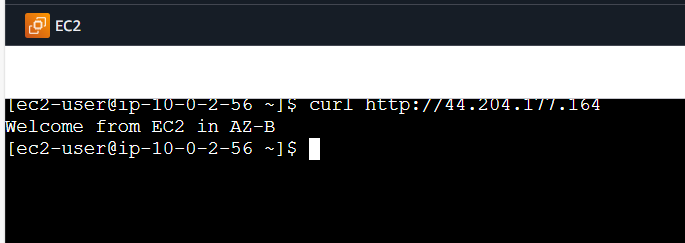
Test after the code after installation via instance public ip

. curl http://<ec2-public-ip>

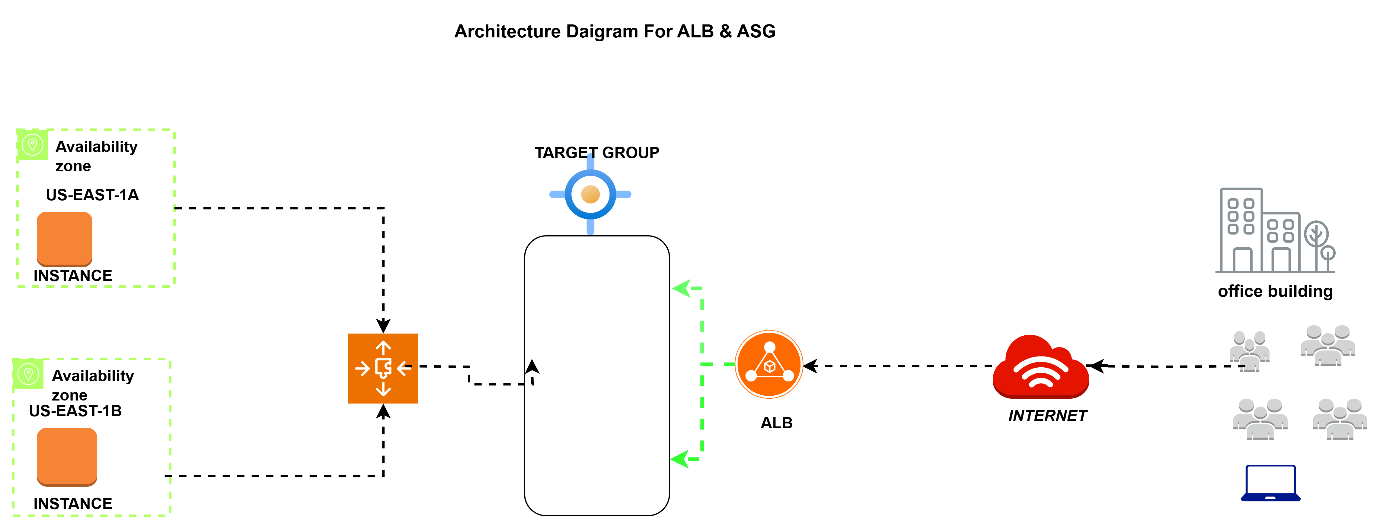
**EC2 1 OUTPUT:**

****

**EC2-2 OUTPUT:**

****

**Architecture Diagram for ALB & ASG:**



Step 4: Create an Application Load Balancer (ALB)

1. Go to EC2 → Load Balancers → Create ALB
2. Choose Application Load Balancer
3. Scheme: Internet-facing
4. Select both Subnet-A and Subnet-B
5. Security Group: same as EC2
6. Listener: HTTP on port 80

Create Target Group:

* Type: Instance
* Protocol: HTTP
* Register EC2-1 and EC2-2

Step 5: Create Auto Scaling Group (Optional but Recommended)

1. Launch Template:
   * AMI: Amazon Linux 2
2. Auto Scaling Group:
   * VPC: Select your VPC
   * Subnets: Select Subnet-A and Subnet-B (Multi-AZ)
   * Attach to the same Target Group (ALB)

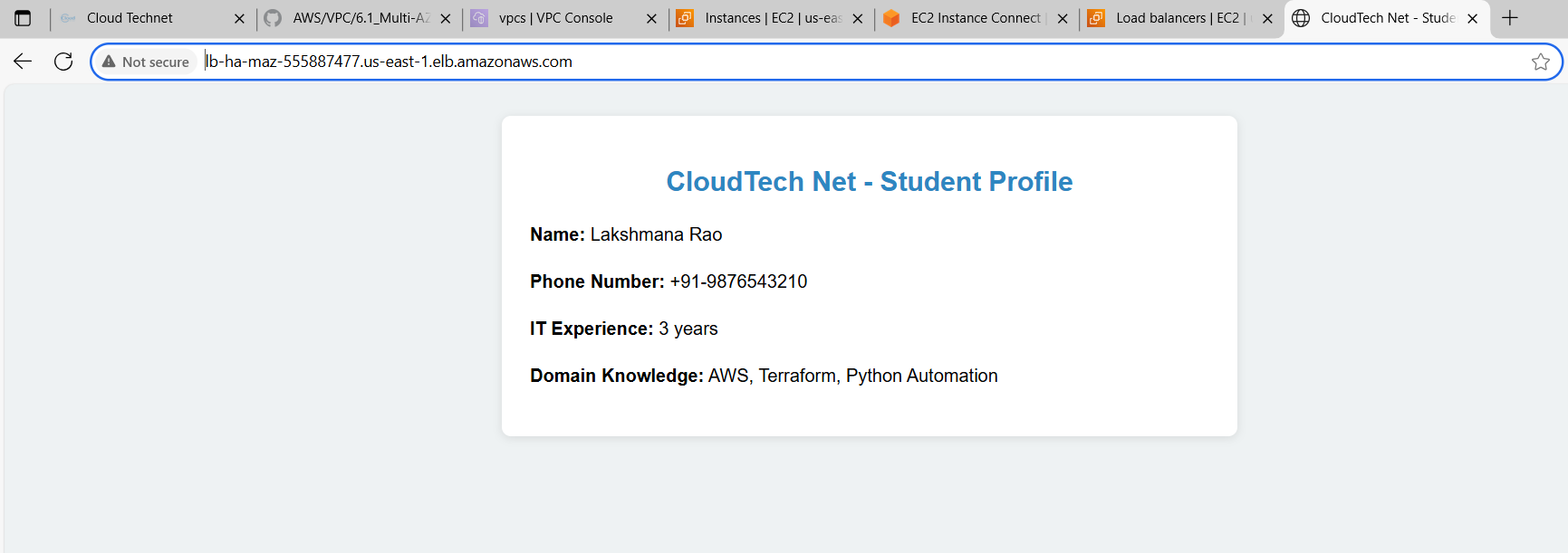
Set desired capacity: 2 Min: 4 Max: 3 Desired

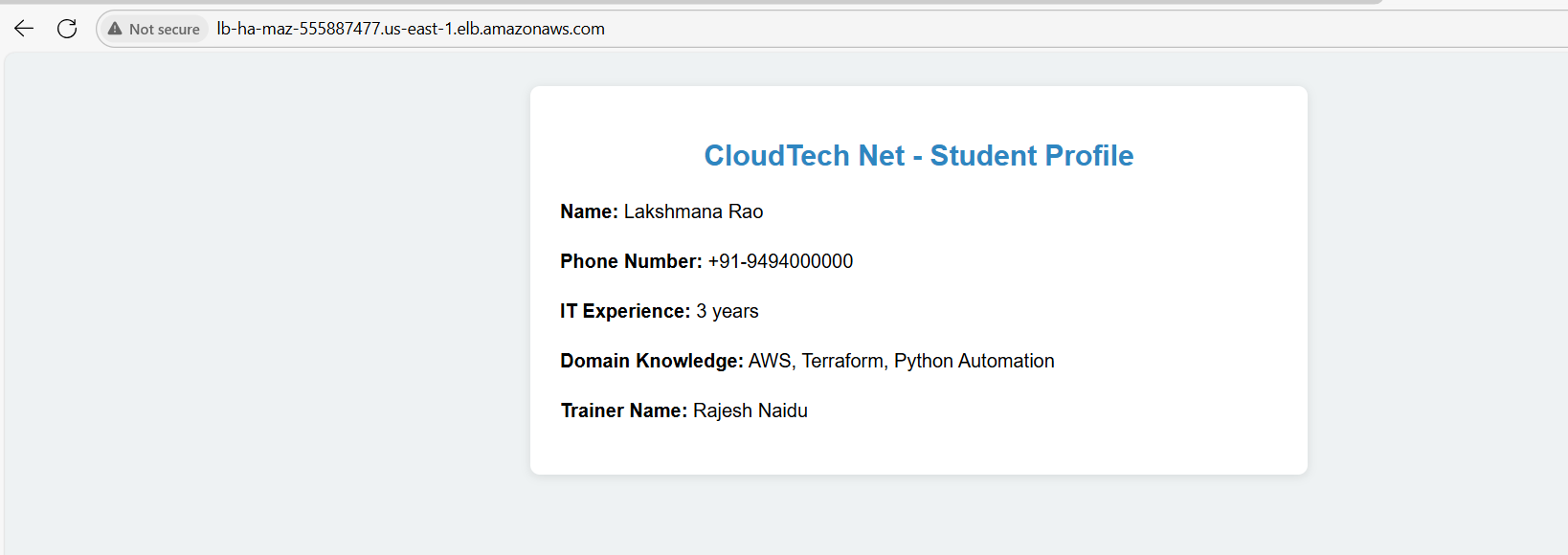
**6: Validate Setup**

1. Get the **ALB DNS name** (e.g., LB-HA-MAZ-555887477.us-east-1.elb.amazonaws.com)
2. Open in browser or run:

curl http://LB-HA-MAZ-555887477.us-east-1.elb.amazonaws.com

Refresh multiple times we will get below outputs for understanding purpose (which server connecting) I have change small change in code level (with Trainer Name & phone number in one server and without Trainer Name & phone number in another server).

****

****

Finally we achieved

**High Availability (HA)**

* If one AZ goes down due to power failure, natural disaster, or hardware issues, your application continues running from the others.
* This ensures *minimal downtime* and a smoother user experience**.**

**Fault Tolerance**

* The system can withstand failures of entire data centers (AZs) without interrupting service.
* Think of it as an insurance policy for uptime.

**Improved Load Distribution**

* Load balancers like **AWS Elastic Load Balancer (ELB)** can smartly route traffic to healthy instances across AZs.
* This prevents overloading any one zone and improves response times.

**Zero-Downtime Deployments**

* You can update instances in one AZ while traffic continues flowing through others.
* That way, updates don’t interrupt service—big win for DevOps practices.

**Cost Optimization (long-term)**

* Though it might seem more expensive upfront, the reduction in outages and downtime can save money and reputation in the long run.

**Better Disaster Recovery**

* In the event of a major disaster affecting a region, having data and services spread across AZs helps with *quicker recovery* and *data durability*.