Automating Scalable AWS Infrastructure with Terraform: A Complete IaC Deployment

1. Project Overview

Modern cloud environments require scalable, automated, and repeatable infrastructure deployment. Terraform, a leading Infrastructure as Code (IaC) tool, enables developers to define, provision, and manage cloud infrastructure declaratively.

This project automates the deployment of a highly available AWS architecture for hosting web applications, ensuring scalability, fault tolerance, and security. Terraform provisions key AWS resources, including:

- **Networking:** VPC, public subnets, Internet Gateway, and Route Table.
- **Compute:** Two EC2 instances with security groups for controlled access.
- **Storage & State Management:** S3 bucket for object storage and Terraform state tracking.
- **Load Balancing:** Application Load Balancer (ALB) for efficient traffic distribution.
- **Access Control:** IAM role and policies granting EC2 instances necessary permissions.

By leveraging Terraform, this project enables infrastructure automation, version control, and repeatability, streamlining cloud management while following DevOps best practices.

Table of Contents

2. About Terraform

Terraform is an open-source Infrastructure-as-Code (IaC) tool that allows you to define, provision, and manage cloud infrastructure using configuration files. With Terraform, you can create, update, and version your infrastructure in a consistent and repeatable way.

Terraform is used to automate the management of cloud resources and supports multiple providers such as AWS, Azure, Google Cloud, and others. Terraform enables developers and infrastructure teams to:

- Define infrastructure using a high-level configuration language (HCL - HashiCorp Configuration Language).
- Provision and manage resources in a declarative way.
- Version control infrastructure alongside application code.

3. Terraform Commands

Before diving into the code breakdown, let's first understand some core Terraform commands that you will be using to interact with the infrastructure.

3.1. terraform init

The terraform init command is used to initialize a Terraform configuration. It prepares the working directory, downloads the necessary provider plugins (e.g., AWS), and sets up the backend where Terraform stores the state of the infrastructure. This command is typically run once when you first set up your project.

```
terraform init
```

What it does:

- Downloads the AWS provider and any other required plugins.
- Initializes the backend configuration (if using remote state).
- Creates necessary files like .terraform that Terraform needs for further operations.

3.2. terraform plan

The terraform plan command is used to generate an execution plan, showing the actions Terraform will take to achieve the desired state described in the configuration files. This step is important as it allows you to preview the changes that will be applied to the infrastructure.

```
terraform plan
```

What it does:

- Compares the current state of the infrastructure with the desired state defined in the configuration files.

- Shows a detailed plan of what will be created, modified, or destroyed.
- Helps you review the changes before actually applying them.

3.3. terraform apply

The terraform apply command is used to apply the changes specified in the Terraform configuration files. It will create, modify, or delete resources in the cloud provider to match the desired state.

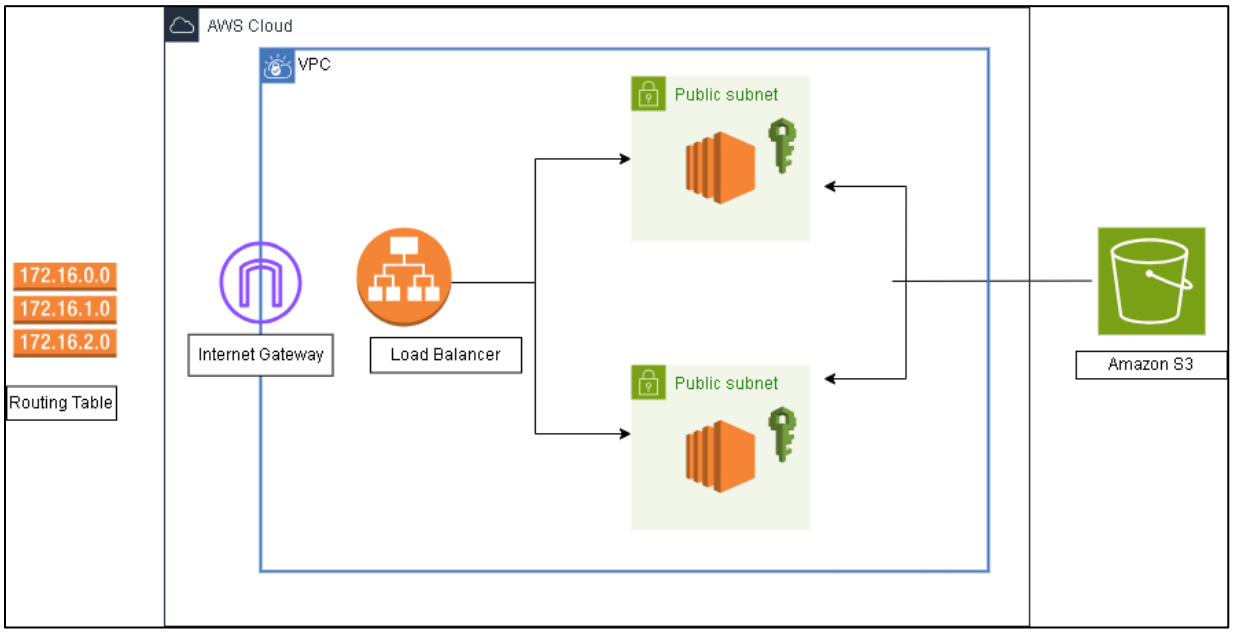
```
terraform apply
```

What it does:

- Applies the changes planned and provisions AWS infrastructure as described in the Terraform configuration.
- Provisions the infrastructure resources such as EC2 instances, load balancers, and more.
- Updates the state file to reflect the current state of the infrastructure.
- This command may require user confirmation before execution

4. Architectural Diagram

Below is the diagram that illustrates the relationship between the various AWS components in the infrastructure. This visual will help you understand the flow of traffic and the structure of the setup.



5. Terraform Code Breakdown

Here's an in-depth look at the code that provisions the infrastructure:

5.1. Provider Configuration

The provider.tf file sets up the AWS provider, specifying the region where the infrastructure will be deployed.

provider.tf

```hcl
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-2"
}

```

5.2. VPC and Subnets

We define the VPC and two public subnets in different availability zones for high availability.

main.tf

```hcl
// Creating VPC
resource "aws_vpc" "mainvpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "vpc_1"
  }
}
```

- Purpose: Creates a VPC with the CIDR block 10.0.0.0/16.
- Key Attributes:

- CIDR Block: Defines the IP range for the VPC.
- Instance Tenancy: Set to 'default' to allow instances to run on shared hardware.

5.3 Subnet Creation

Subnet 1

```
//Creating Subnet1
resource "aws_subnet" "sub1" {
  vpc_id     = aws_vpc.mainvpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-2a"
  map_public_ip_on_launch = true

  tags = {
    Name = "Subnet1"
  }
}
```

- **Purpose**: Creates a public subnet in us-east-2a with CIDR block 10.0.1.0/24.
- **Key Attributes**:
  - **Availability Zone**: Ensures high availability across multiple zones.
  - **Public IP**: Automatically assigns public IPs to instances upon launch.

Subnet 2

```
//Creating Subnet 2
resource "aws_subnet" "sub2" {
  vpc_id     = aws_vpc.mainvpc.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-2b"
  map_public_ip_on_launch = true

  tags = {
    Name = "subnet2"
  }
}
```

Purpose: Creates another public subnet in us-east-2b with CIDR block 10.0.2.0/24.

5.4. Internet Gateway

The internet gateway is required for instances in public subnets to access the internet.

```
34
35  //Creating Internet Gateway
36  resource "aws_internet_gateway" "igw" {
37    vpc_id = aws_vpc.mainvpc.id
38
39    tags = {
40      Name = "IGW"
41    }
42  }
43
```

Purpose: Creates an internet gateway and associates it with the VPC to allow instances to communicate with the internet.

5.5 Route Table and Associations

```
43
44  //Creating Route table
45  resource "aws_route_table" "RTable1" {
46    vpc_id = aws_vpc.mainvpc.id
47
48    route {
49      cidr_block = "0.0.0.0/0"
50      gateway_id = aws_internet_gateway.igw.id
51    }
52    tags = {
53      Name = "Route_Table_Tf"
54    }
55  }
56
```

- **Purpose:** Defines a route table that enables outbound internet traffic from public subnets by forwarding requests (0.0.0.0/0) to the Internet Gateway.
- **Associations:** Subnet1 and Subnet2 are each associated with this route table, allowing internet access from both subnets.

5.6 EC2 Instances and Security Groups

We create two EC2 instances and configure security groups to allow HTTP and SSH traffic.

```
//Creating Security Group
resource "aws_security_group" "secgroup" {
  name        = "SecurityGroup1"
  description = "For creating EC2 instances"
  vpc_id      = aws_vpc.mainvpc.id

  ingress {
    description  = "This is for HTTP"
    //self      = true
    from_port = 80
    to_port   = 80
    protocol="tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
 ingress {
    description  = "This is for SSH"
    //self      = true
    from_port = 22
    to_port   = 22
    protocol="tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

}
```

- **Purpose:** Creates a security group allowing inbound HTTP (port 80) and SSH (port 22) traffic from any IP address.
- **Key Attributes:**

  Ingress Rules: For HTTP and SSH access.

  Egress Rules: Allows all outbound traffic.

5.7 EC2 Instances

Instance 1

- **Purpose:** Launches an EC2 instance in subnet1 and in subnet2 with the specified AMI and instance type.
- **Key Attributes:**

  Security Group: Attached to the previously defined security group.

  IAM Instance Profile: Allows the EC2 instance to assume the IAM role and interact with other AWS services (e.g., S3).

```
105
106  //Creating EC2 Instance - Instance1
107  resource "aws_instance" "Instance1" {
108    ami                    = "ami-088b41ffb0933423f"
109    instance_type          = "t2.micro"
110    vpc_security_group_ids = [aws_security_group.secgroup.id]
111    subnet_id = aws_subnet.sub1.id
112    user_data = base64encode(file("userdata1.sh"))
113    iam_instance_profile    = aws_iam_instance_profile.ec2_profile.name
114  }
115
116  //Creating EC2 Instance - Instance2
117  resource "aws_instance" "Instance2"{
118    ami = "ami-088b41ffb0933423f"
119    instance_type = "t2.micro"
120    vpc_security_group_ids = [aws_security_group.secgroup.id]
121    subnet_id = aws_subnet.sub2.id
122    user_data = base64encode(file("userdata2.sh"))
123    iam_instance_profile    = aws_iam_instance_profile.ec2_profile.name
124  }
125
```

5.8 IAM Role and Policy

i. IAM Role

```
126  //Creating an IAM Role for EC2
127
128  resource "aws_iam_role" "Iam_Ec2_Role" {
129    name = "EC2_S3_Access_Role"
130
131    # Terraform's "jsonencode" function converts a
132    # Terraform expression result to valid JSON syntax.
133    assume_role_policy = jsonencode({
134      Version = "2012-10-17"
135      Statement = [
136        {
137          Action = "sts:AssumeRole"
138          Effect = "Allow"
139          Sid    = ""
140          Principal = {
141            Service = "ec2.amazonaws.com"
142          }
143        },
144      ]
145    })
146
147  }
```

Purpose: Creates an IAM role that EC2 instances can assume to gain permissions to access other AWS services.

ii. IAM Policy for S3 Access

```
148
149   //Attach an IAM Policy for S3 Access
150
151     resource "aws_iam_policy" "s3_full_access" {
152     name        = "S3FullAccess"
153     description = "Allows EC2 instances full access to S3"
154
155     policy = jsonencode({
156       Version = "2012-10-17"
157       Statement = [{
158           Sid     = "AllowFullS3Access"
159         Effect    = "Allow"
160         Action    = "s3:*"
161         Resource = "arn:aws:s3:::*"
162       }]
163     })
164   }
```

Purpose: Grants the EC2 instances full access to S3.

iii. Attach Policy to Role

```
166   // Attaching IAM Policy to the Role
167
168   resource "aws_iam_role_policy_attachment" "attach_s3_full_access"{
169     role       = aws_iam_role.Iam_Ec2_Role.name
170     policy_arn = aws_iam_policy.s3_full_access.arn
171   }
172
173   // Creating IAM Instance Profile to Attach Role to EC2
174   resource "aws_iam_instance_profile" "ec2_profile" {
175     name = "ec2_instance_profile"
176     role = aws_iam_role.Iam_Ec2_Role.name
177   }
```

Purpose: Associates the IAM role with the defined S3 access policy, enabling EC2 instances to interact with S3 securely.

5.9 Load Balancer and Target Groups

i. Load Balancer

```
178
179  //Creating Load balancer
180  resource "aws_lb" "myLoadBalancer" {
181    name               = "myLoadBalancer"
182    internal           = false
183    load_balancer_type = "application"
184    security_groups    = [aws_security_group.secgroup.id]
185    subnets            = [aws_subnet.sub1.id, aws_subnet.sub2.id]
186
187    }
```

Purpose: Creates an Application Load Balancer (ALB) that distributes HTTP traffic to the EC2 instances in subnet1 and subnet2.

ii. Target Groups

```
189  //Creating Target group
190
191  resource "aws_lb_target_group" "alb-target-group" {
192    name        = "alb-target-group"
193    port        = 80
194    protocol    = "HTTP"
195    vpc_id      = aws_vpc.mainvpc.id
196
197    health_check {
198      path = "/"
199      port = 80
200      protocol = "HTTP"
201
202    }
203  }
```

Purpose: Defines a target group for the load balancer to send traffic to, with health checks on the / path.

iii. Target Group Attachments

Purpose: Attaches EC2 instance Instance1 and Instance2 to the target group for routing.

```
204
205  //Attaching Target groups to Instance1
206
207  resource "aws_lb_target_group_attachment" "tga1" {
208    target_group_arn = aws_lb_target_group.alb-target-group.arn
209    target_id        = aws_instance.Instance1.id
210    port             = 80
211  }
212
213  //Attaching Target groups to Instance2
214
215  resource "aws_lb_target_group_attachment" "tga2" {
216    target_group_arn = aws_lb_target_group.alb-target-group.arn
217    target_id        = aws_instance.Instance2.id
218    port             = 80
219  }
220
```

5.10 HTTP Listener

```
221
222  resource "aws_lb_listener" "http_listener" {
223    load_balancer_arn = aws_lb.myLoadBalancer.arn
224    port              = 80
225    protocol          = "HTTP"
226
227    default_action {
228      type             = "forward"
229      target_group_arn = aws_lb_target_group.alb-target-group.arn
230    }
231  }
```

- **Purpose:** Configures an HTTP listener for the Application Load Balancer (ALB) to listen for incoming traffic on port 80 (HTTP).
- **Key Attributes:**
    - **Load Balancer ARN:** The listener is associated with the ALB created earlier (aws_lb.myLoadBalancer).
    - **Port**: The listener listens on port 80 for HTTP traffic.
    - **Protocol:** The listener uses the HTTP protocol for communication.
    - **Default Action:** The listener's default action is to forward traffic to the target group (aws_lb_target_group.alb-target-group) created earlier.

  The listener processes incoming HTTP requests and forwards them to the target group, distributing traffic efficiently among EC2 instances.

5.11 S3 Bucket

```
68
69  //Creating S3 bucket
70  resource "aws_s3_bucket" "s3bucket" {
71    bucket = "reesasabu-terraform-2025"
72  }
```

- **Purpose:** Creates an S3 bucket with the name reesasabu-terraform-2025.
- **Key Attributes:**
    - **Bucket Name:** This should be globally unique, and the name specified here is reesasabu-terraform-2025.

  This bucket can be used to store and manage your data, and the IAM roles and policies defined earlier give EC2 instances access to interact with this bucket.

5.12 Terraform Backend Configuration

This setup ensures that Terraform state is stored remotely in an S3 bucket instead of locally, providing better state management, collaboration, and security.

```
232
233 terraform {
234   backend "s3" {
235     bucket         = "reesasabu-terraform-state"
236     key            = "terraform.tfstate"
237     region         = "us-east-2"
238     encrypt        = true
239   }
240 }
241
```

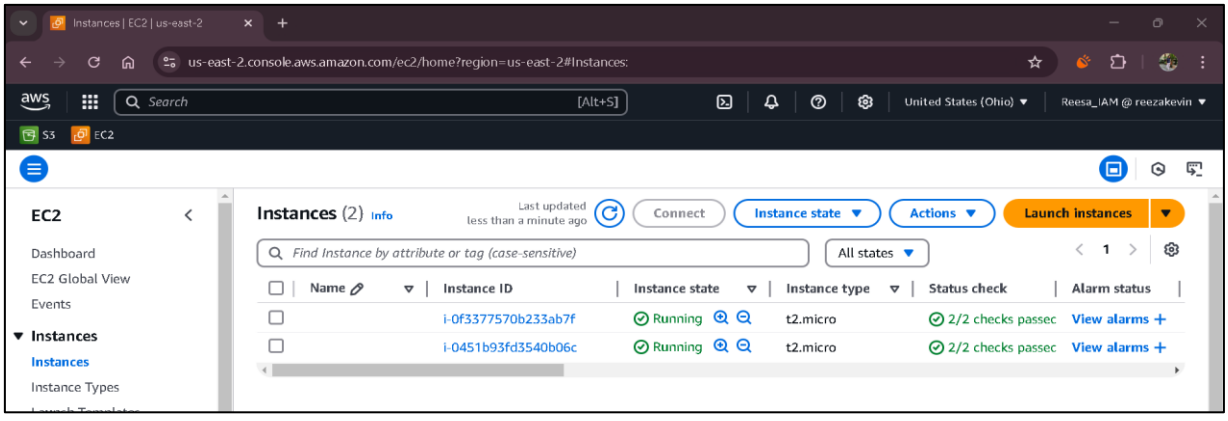
- **bucket:** The name of the S3 bucket where the state file will be stored. In this case, the bucket name is reesasabu-terraform-state.
- **key:** The path within the S3 bucket where the state file will be stored. In this example, it's stored as terraform.tfstate.
- **region:** The AWS region where the S3 bucket is located. For example, us-east-2**.**
- **encrypt:** Enabling encryption (true) ensures that the state file is stored securely in the S3 bucket.

6. Deployment and Outputs

The following screenshots show the AWS resources that were successfully created using Terraform.

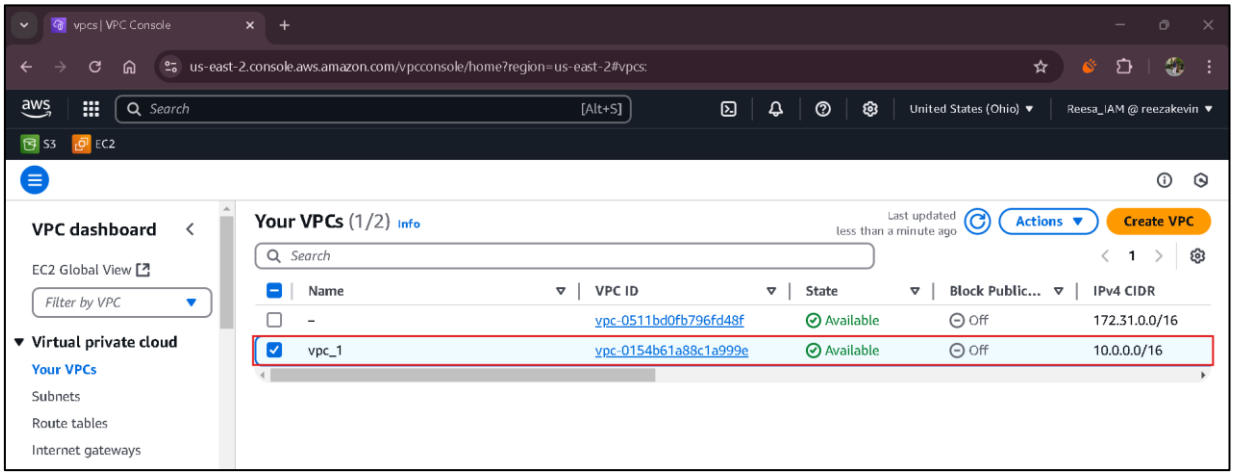
6.1 EC2 Instances

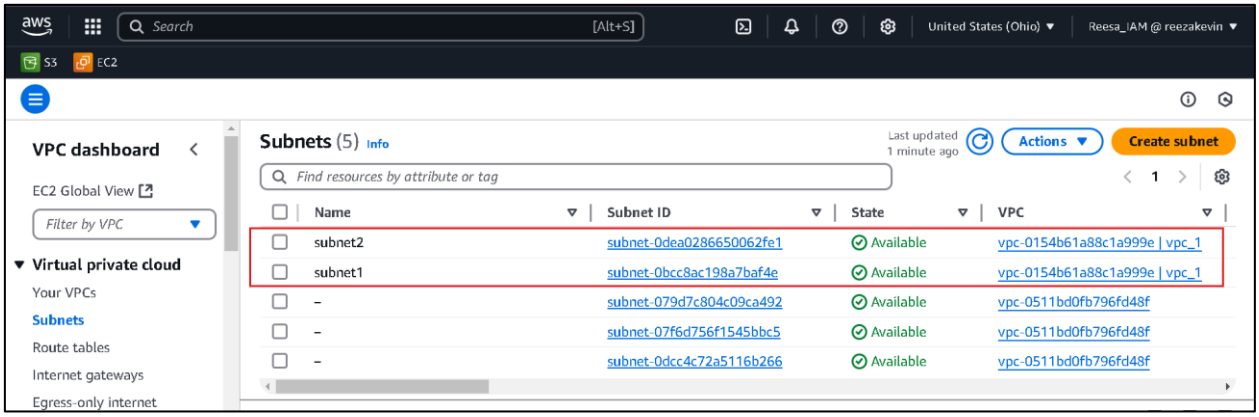
The screenshot below shows the two EC2 instances that were provisioned using Terraform. These instances are deployed in separate public subnets and are associated with the specified security group.



6.2 VPC and Subnets

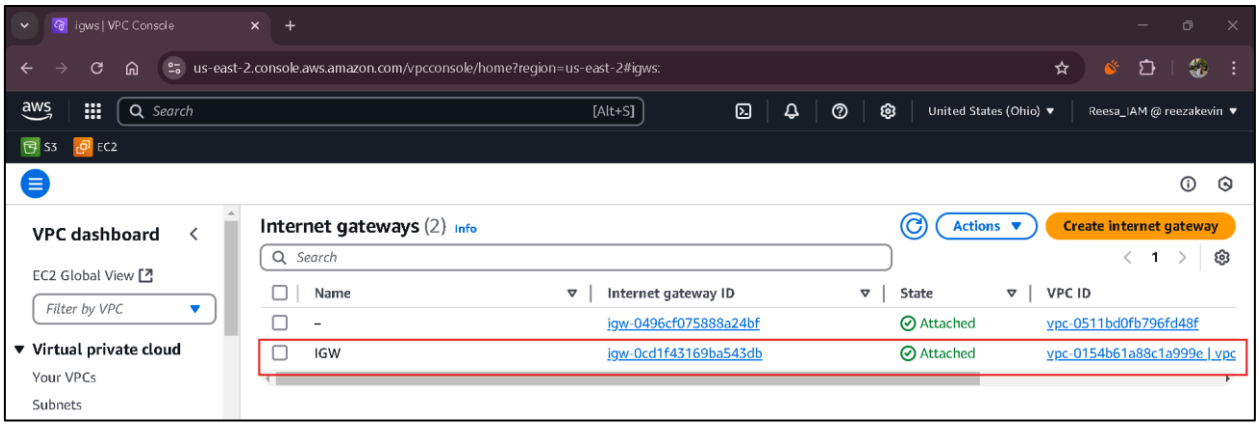
The screenshot below displays the VPC and its associated subnets. The VPC was created with a CIDR block of 10.0.0.0/16, and two public subnets were assigned in different availability zones.





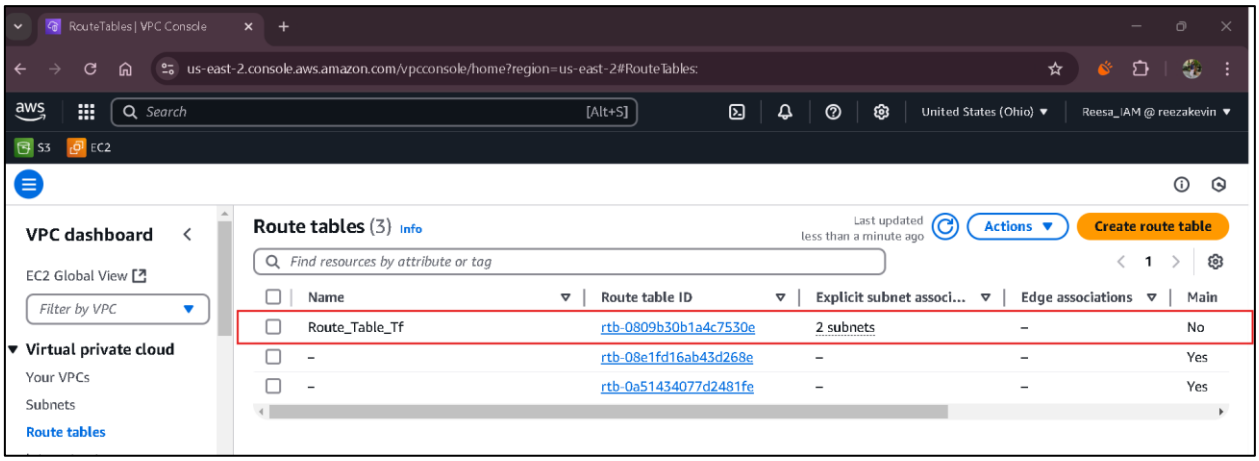
6.3 Internet Gateway

An Internet Gateway was created and attached to the VPC, allowing public access to instances. The screenshot below confirms its creation.



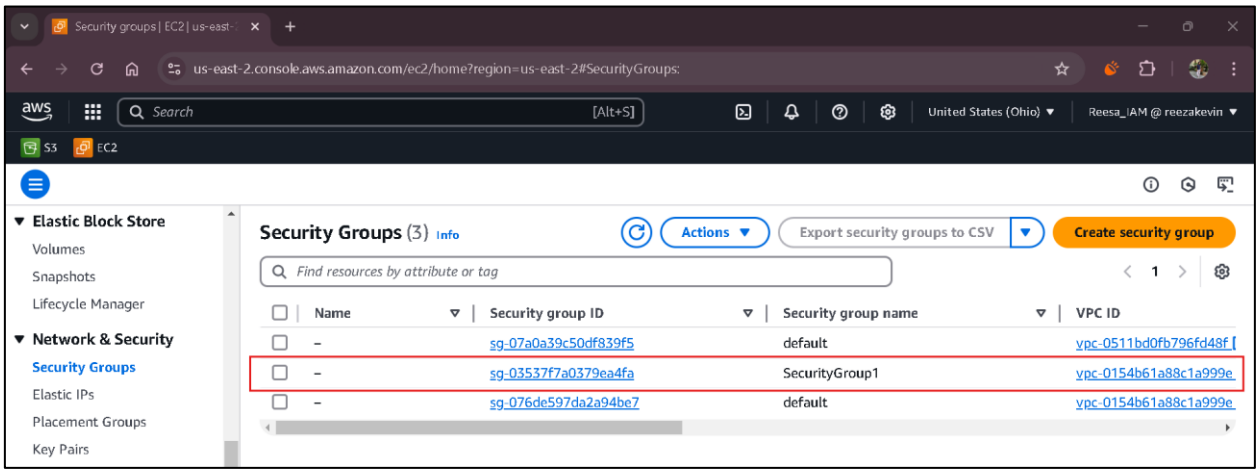
6.4 Route Table and Associations

The route table below is configured to direct traffic from the subnets to the Internet Gateway.



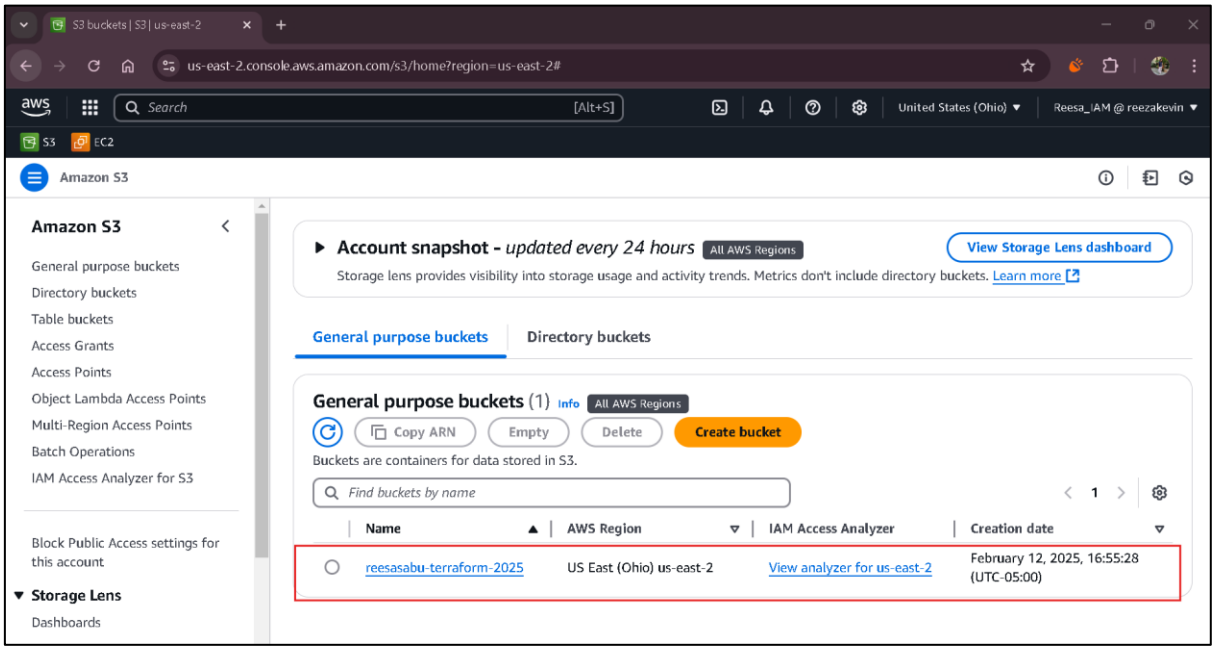
6.5 Security Group

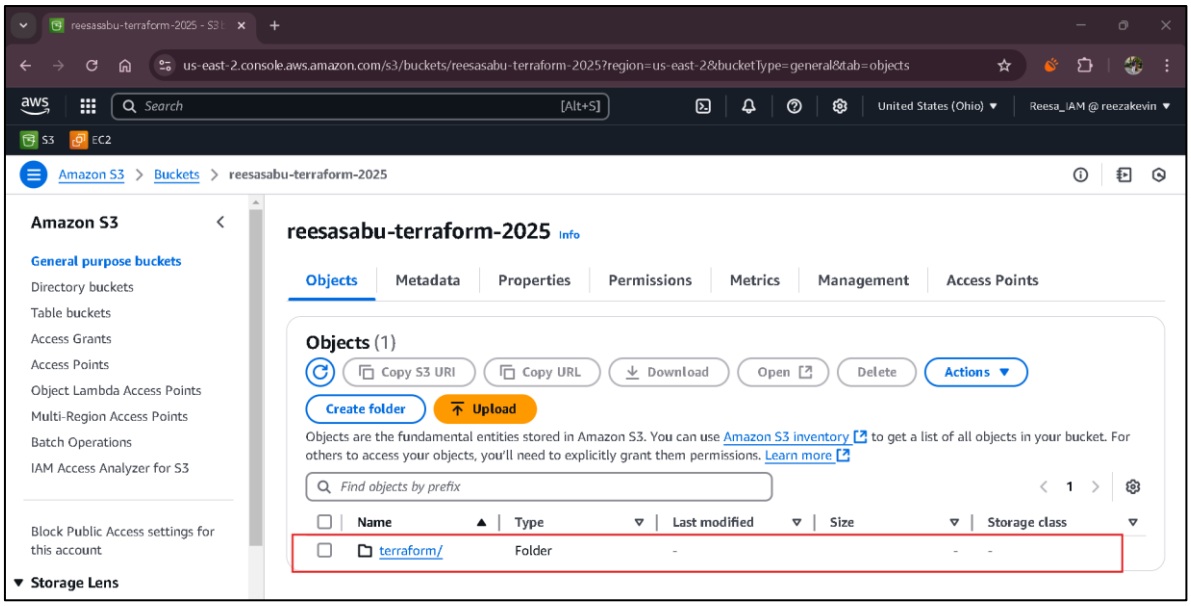
The security group was created with inbound rules allowing SSH (port 22) and HTTP (port 80) access. The screenshot below confirms these settings.

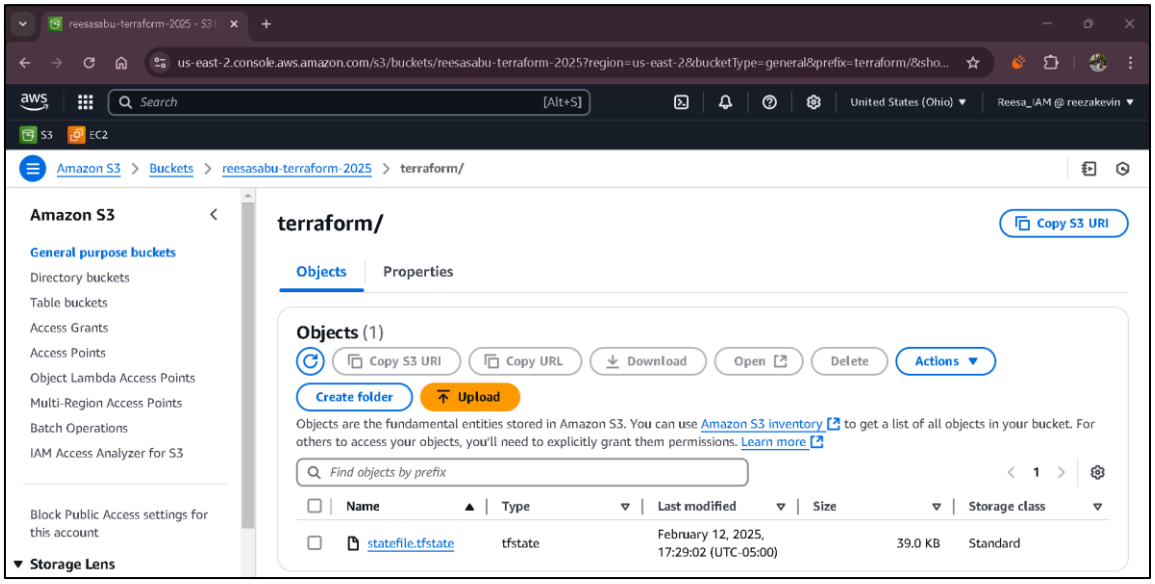


6.6 S3 Bucket

The S3 bucket reesasabu-terraform-2025 was successfully created to store files In addition to storing general files, this S3 bucket is configured as the backend for Terraform state management. The terraform.tfstate file stored here helps track the infrastructure state, ensuring consistency and enabling remote collaboration

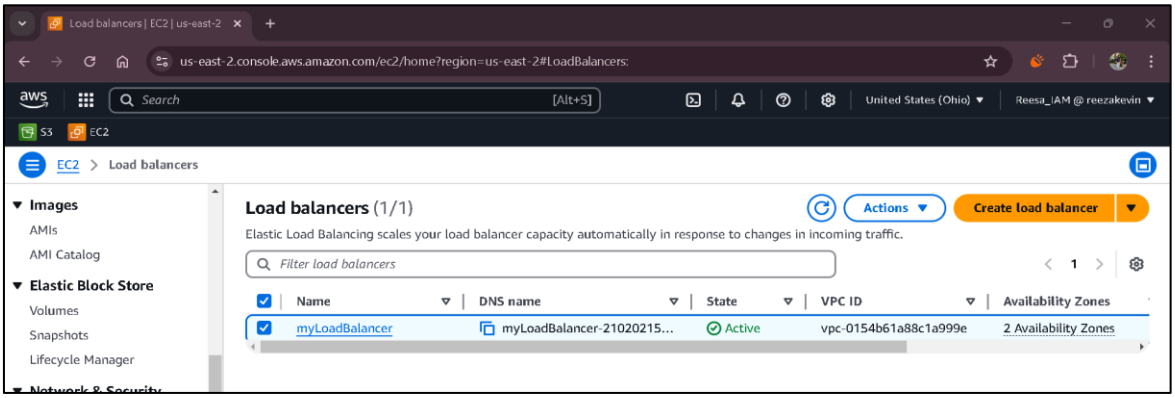


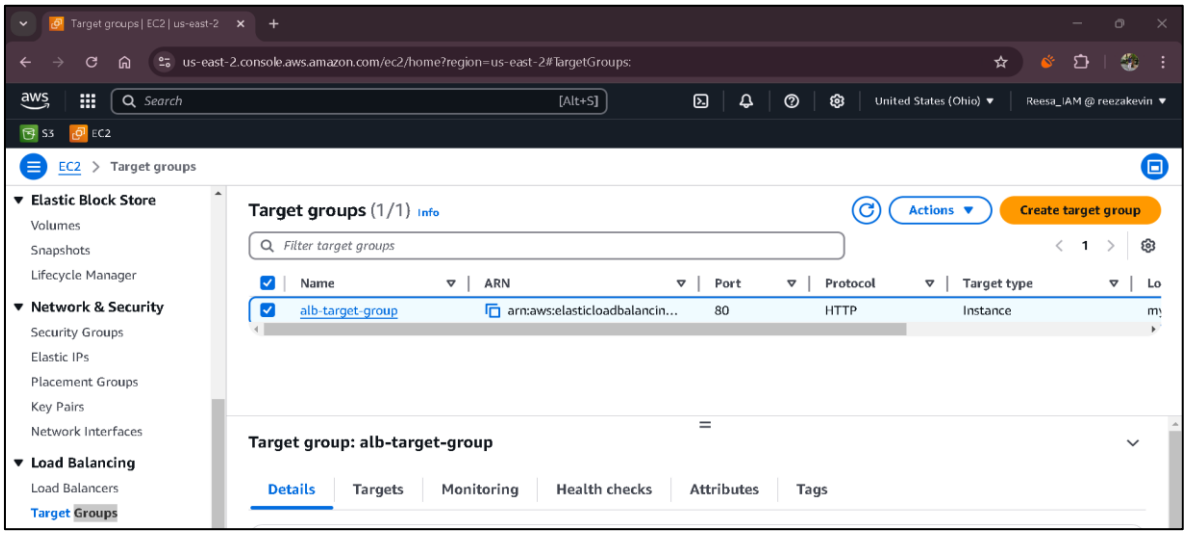




6.7 Application Load Balancer

The ALB was successfully created and is routing traffic to the EC2 instances. The screenshot below shows the load balancer details.



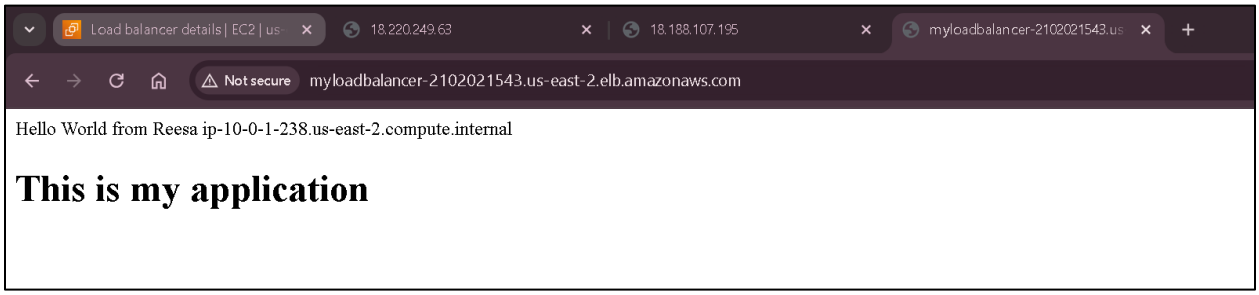


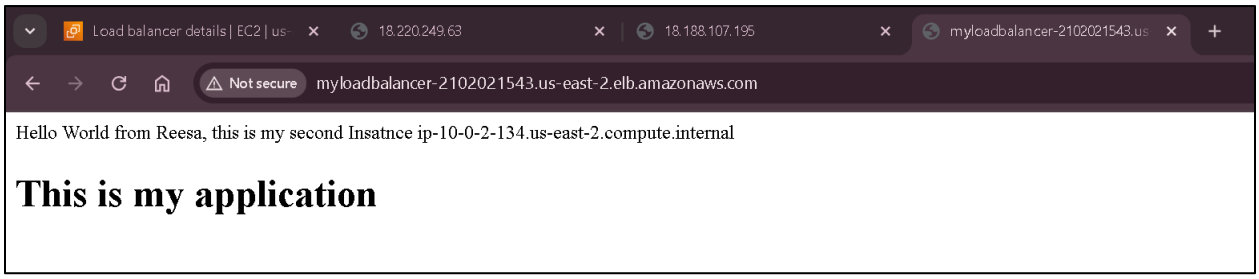
7. Testing the Deployment

After deploying the infrastructure, the DNS name of the Application Load Balancer was obtained. The DNS was accessed through a web browser to verify that the instances were responding correctly.

7.1 Load Balancer DNS Output

The screenshot below demonstrates the successful resolution of the ALB DNS name in a web browser, verifying that the deployed infrastructure is functioning correctly





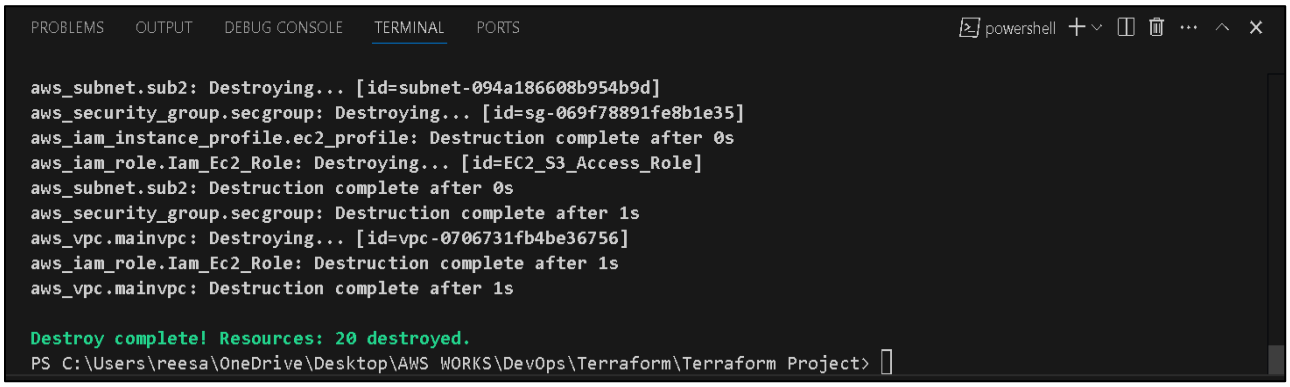
7.2 Cleaning Up (Optional)

If you want to delete all resources:

```
terraform destroy -auto-approve
```

Precautions (e.g., ensure no critical data is lost).

Expected output: Resources being destroyed one by one**.**



8. Conclusion

This Terraform project sets up a fully functional and scalable web infrastructure on AWS. By using Terraform, you have automated the creation and management of key AWS services, including VPC, EC2, ALB, and S3. The solution ensures high availability and scalability, making it suitable for hosting web applications in a production environment.