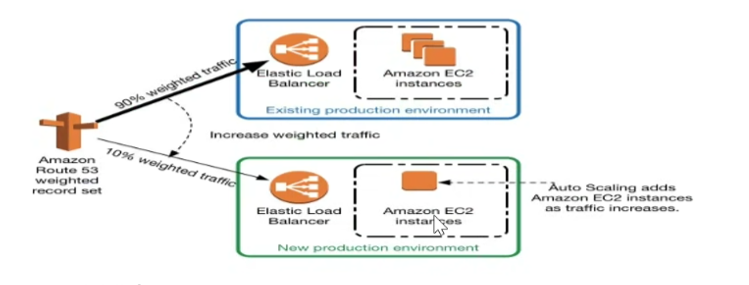
**Problem Statement:** You have been asked to set up a Blue/Green (B/G) solution for a mission-critical web application running on AWS. It should consist of **two separate environments**. The blue environment contains Amazon EC2 instances in an Auto Scaling group that runs the current production version of the application. The green environment contains EC2 instances in another Auto Scaling group that runs the new version of the application. Each Auto Scaling group is behind its own Application Load Balancer (ALB), so you can configure **two alias records as endpoints** in Amazon Route 53 and use a **routing policy** to gradually shift traffic from the ALB for the blue environment to the ALB for the green environment. Route 53 endpoint must be accessible from the internet through a domain name and **HTTPS protocol only**. Setup this combination to achieve B/G deployment and explain how different **backend database endpoints** will be handled in this situation. Use proper tool to build an infrastructure as code (IaC) solution



**by setting up the VPC and Networking Layer. This will include:**

VPC: A Virtual Private Cloud where all the resources will reside.

Subnets: Public subnets for the ALBs and private subnets for the EC2 instances.

Internet Gateway: To allow access from the internet.

Route Tables: For traffic routing between subnets and the internet.

Here’s Terraform setup for this:

**Step 1: VPC and Networking Layer**

 Created a VPC (aws\_vpc.blue\_green\_vpc) to host the blue/green environments.

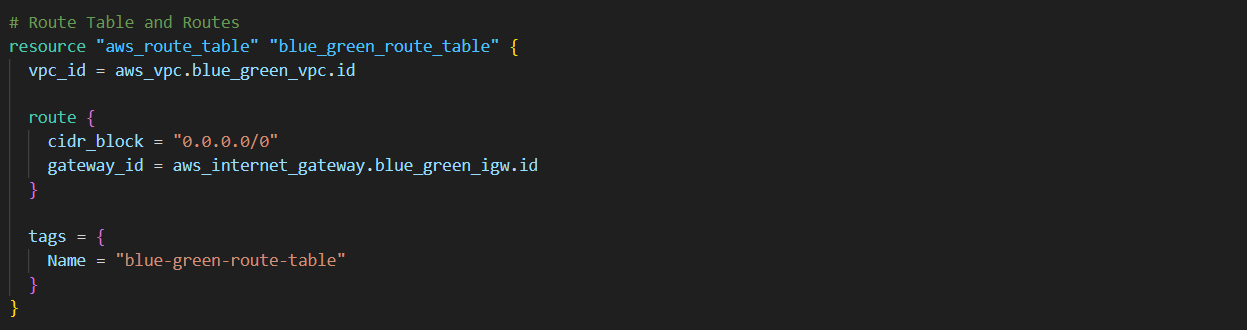
 Defined two public subnets in different Availability Zones (us-east-2a and us-east-2b) to ensure high availability.



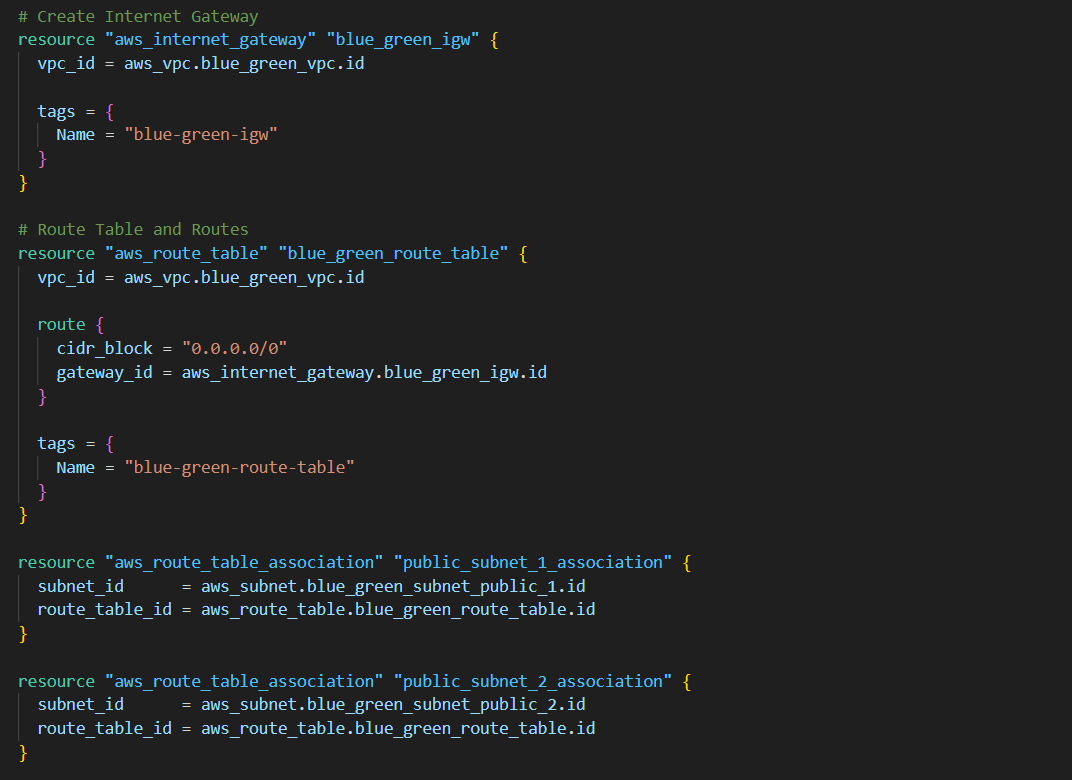
# **Define Internet Gateway**

Set up an **Internet Gateway** and attached it to the VPC.



**# Route Tables**

**# Associate public route table with public subnets**



Step-2

Step 2: Auto Scaling Groups and Load Balancers

 Blue and Green ASGs were defined, each running a different version of the application.

* Used Launch Configurations (aws\_launch\_configuration.blue\_launch\_config and aws\_launch\_configuration.green\_launch\_config) for the two environments, deploying EC2 instances running the application.
* Configured desired capacities for both ASGs to scale based on demand.

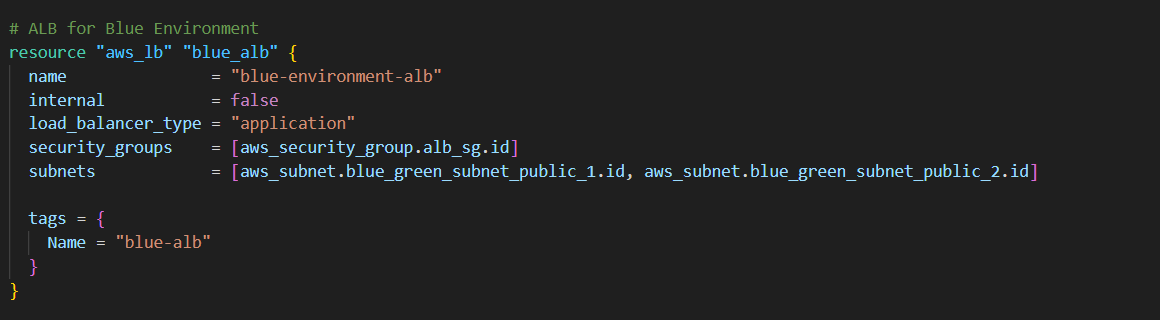
 You added security groups to control instance traffic, allowing HTTP/HTTPS access.# Security Group for ALB (allow HTTP/HTTPS traffic)

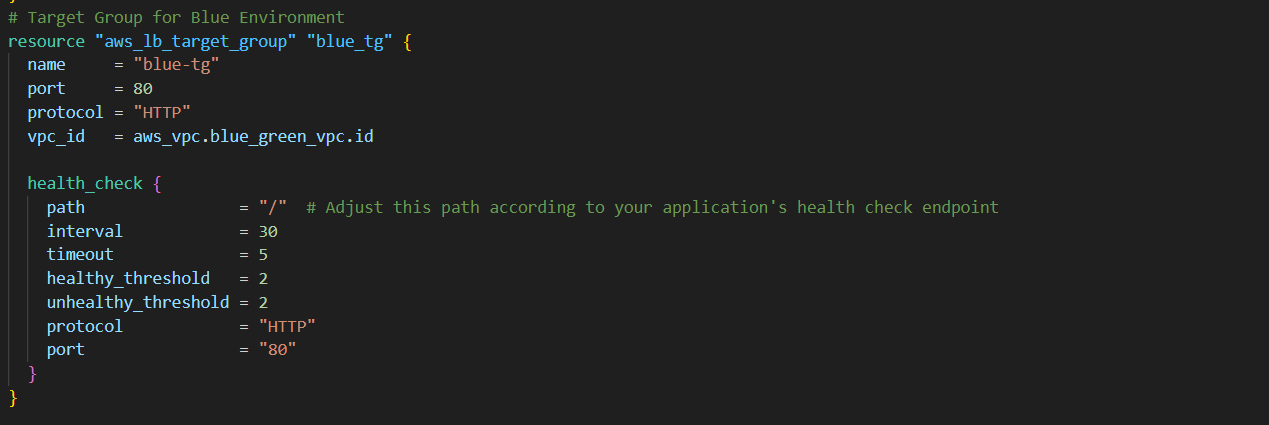
 Created separate **ALBs** for the blue and green environments (aws\_lb.blue\_alb and aws\_lb.green\_alb).

 Configured **target groups** and **listeners** to manage HTTP traffic between the ALBs and ASGs.

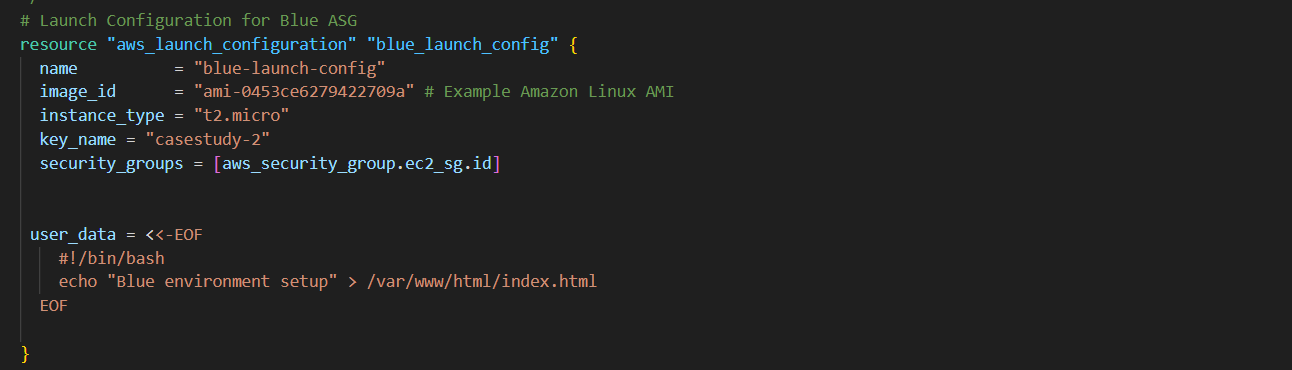
 Set up health checks on the ALBs to monitor instance health and ensure only healthy instances receive traffic.

**# Application Load Balancer for Blue environment**

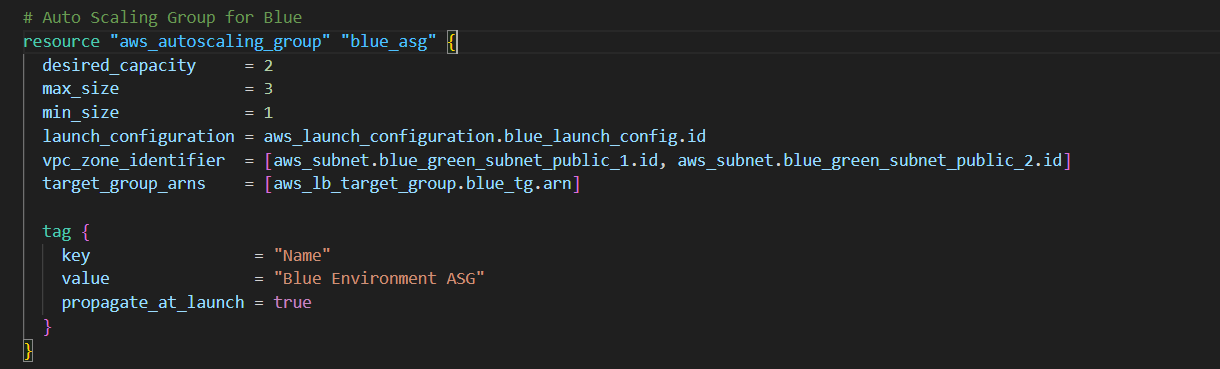


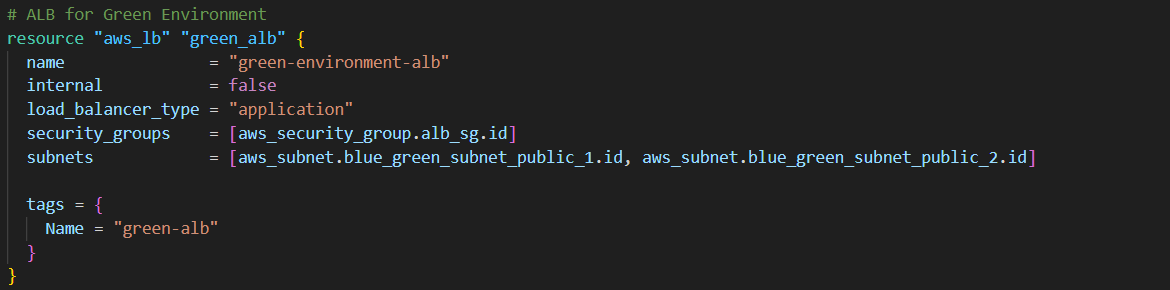


**# Auto Scaling Launch Template for Blue environment**

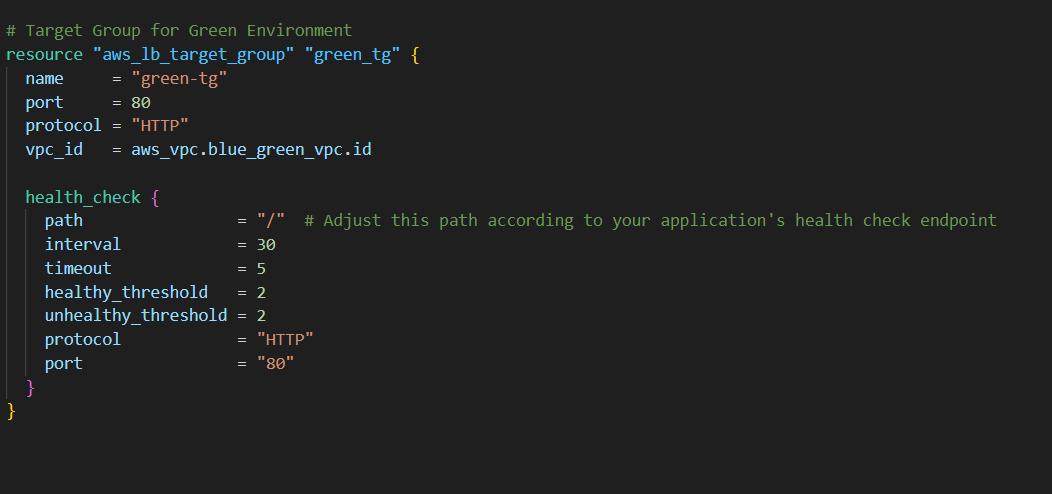
****

**# Auto Scaling Group for Blue environment**





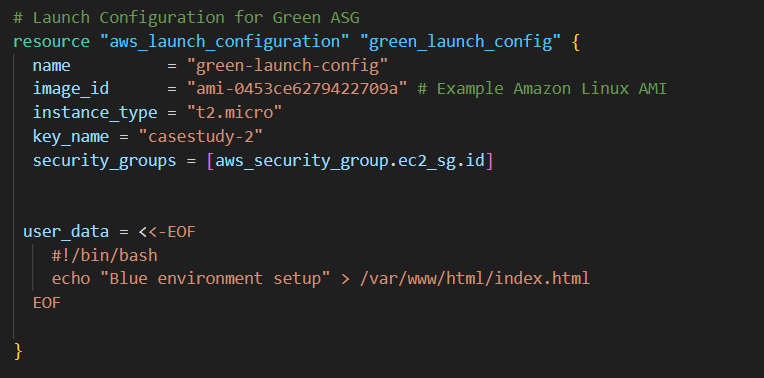
**# ALB Target Group for Green environment**



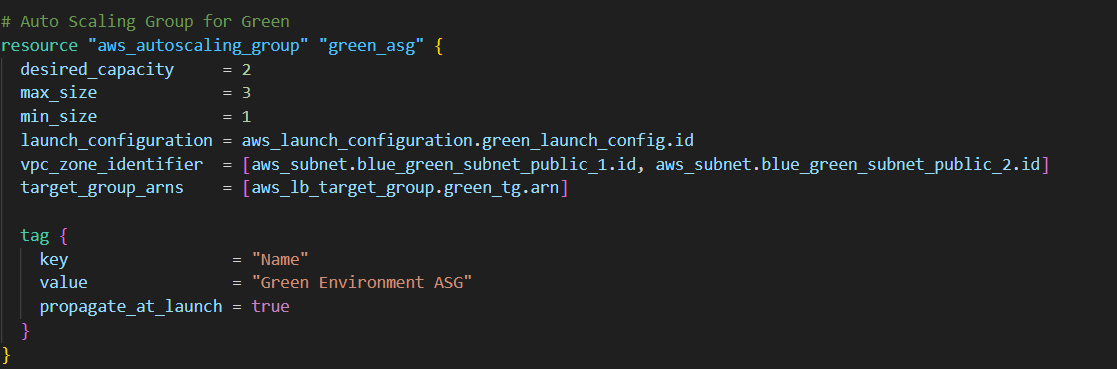
**# ALB Listener for Green environment**



**# Auto Scaling Launch Template for Green environment**



**# Auto Scaling Group for Green environment**



**Step 3: Setting Up Route 53 for Traffic Routing.**

**Step 3: Route 53 Configuration**

In this step, we'll set up Amazon Route 53 to manage traffic between the Blue and Green environments using a routing policy. We’ll use a weighted routing policy to gradually shift traffic from the Blue environment to the Green environment.

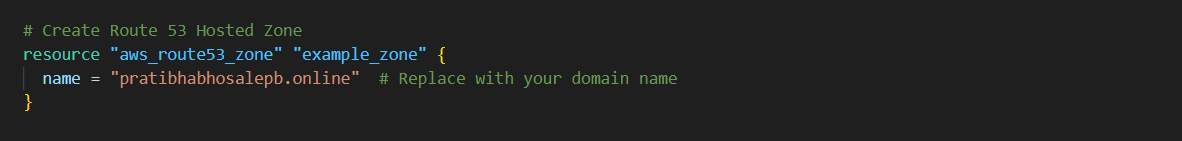
**1. Create a Hosted Zone**

**# Hosted Zone**

resource "aws\_route53\_zone" "main" {

name = "pratibhabhosalepb.online"

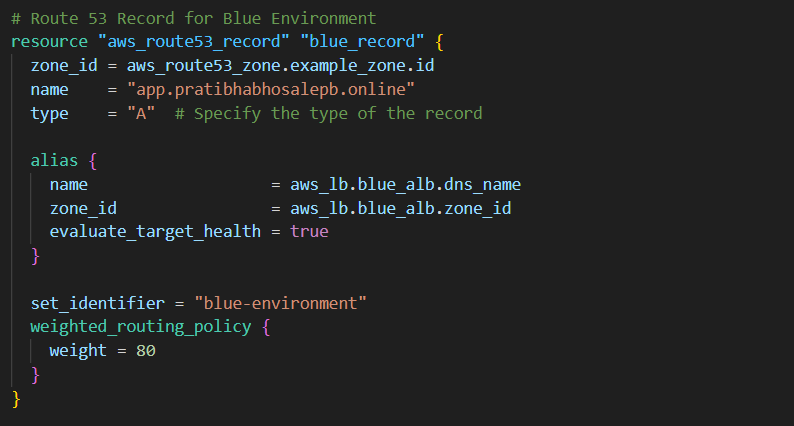
}



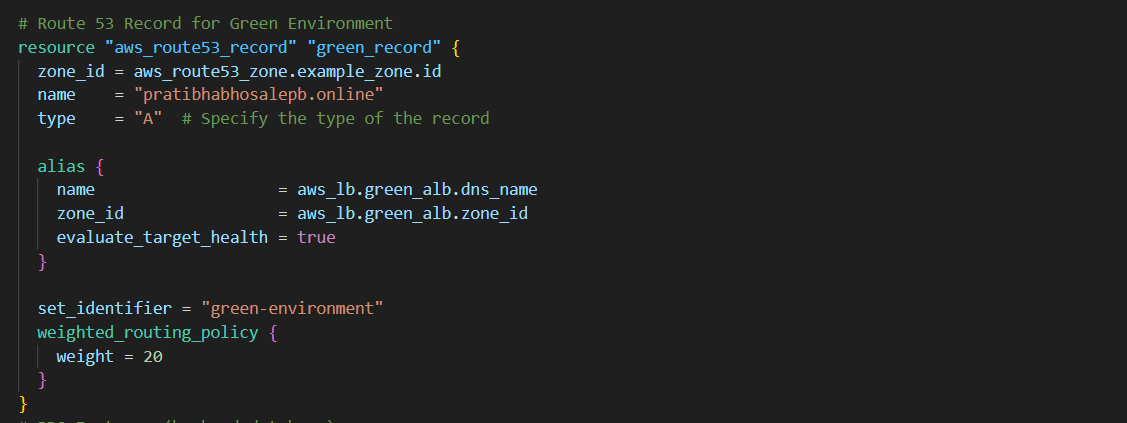
**2. Create Route 53 Alias Records**

We will create two alias records pointing to the respective ALBs for Blue and Green environments.

**# Alias record for Blue environment**



**# Alias record for Green environment**



**3. Weighted Routing Policy**

To gradually shift traffic between the Blue and Green environments, you’ll use a weighted routing policy.

# Weighted routing policy record for Blue environment

resource "aws\_route53\_record" "blue\_weighted\_record" {

zone\_id = aws\_route53\_zone.main.id

name = "app.example.com"

type = "A"

ttl = 60

weighted\_routing\_policy {

weight = 100

}

alias {

name = aws\_lb.blue\_alb.dns\_name

zone\_id = aws\_lb.blue\_alb.zone\_id

evaluate\_target\_health = true

}

}

**# Weighted routing policy record for Green environment**

resource "aws\_route53\_record" "green\_weighted\_record" {

zone\_id = aws\_route53\_zone.main.id

name = "app.example.com"

type = "A"

ttl = 60

weighted\_routing\_policy {

weight = 0

}

alias {

name = aws\_lb.green\_alb.dns\_name

zone\_id = aws\_lb.green\_alb.zone\_id

evaluate\_target\_health = true

}

}

**4. Update the Weight to Shift Traffic**

we can update the weights to control the percentage of traffic that goes to each environment. For example, to shift 10% of the traffic to the Green environment, you would update the weights as follows:

**# Update weights to shift 10% of traffic to Green environment**

resource "aws\_route53\_record" "blue\_weighted\_record" {

zone\_id = aws\_route53\_zone.main.id

name = "app. pratibhabhosalepb.online "

type = "A"

ttl = 60

weighted\_routing\_policy {

weight = 80

}

alias {

name = aws\_lb.blue\_alb.dns\_name

zone\_id = aws\_lb.blue\_alb.zone\_id

evaluate\_target\_health = true

}

}

resource "aws\_route53\_record" "green\_weighted\_record" {

zone\_id = aws\_route53\_zone.main.id

name = "app. pratibhabhosalepb.online "

type = "A"

ttl = 60

weighted\_routing\_policy {

weight = 20

}

alias {

name = aws\_lb.green\_alb.dns\_name

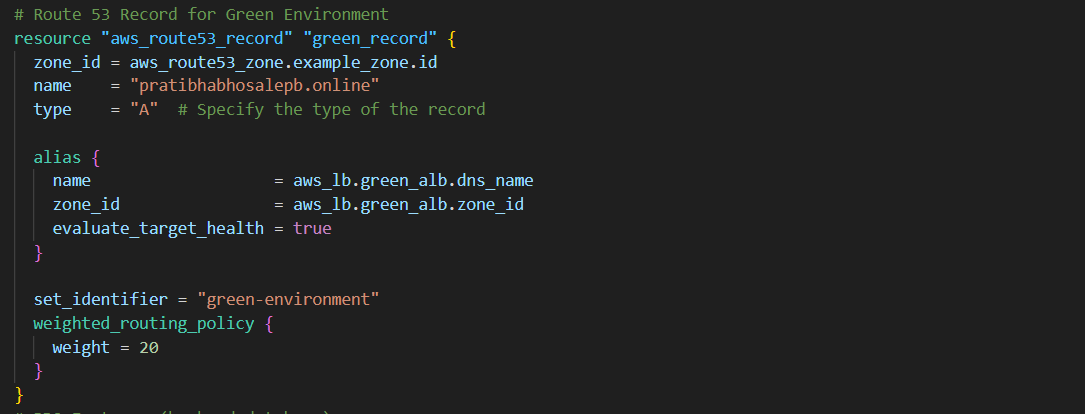
zone\_id = aws\_lb.green\_alb.zone\_id

evaluate\_target\_health = true

}

}

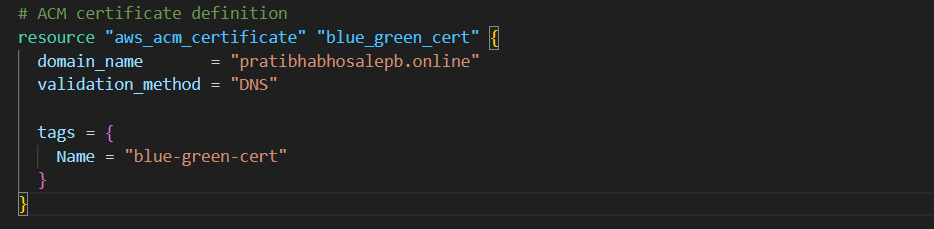




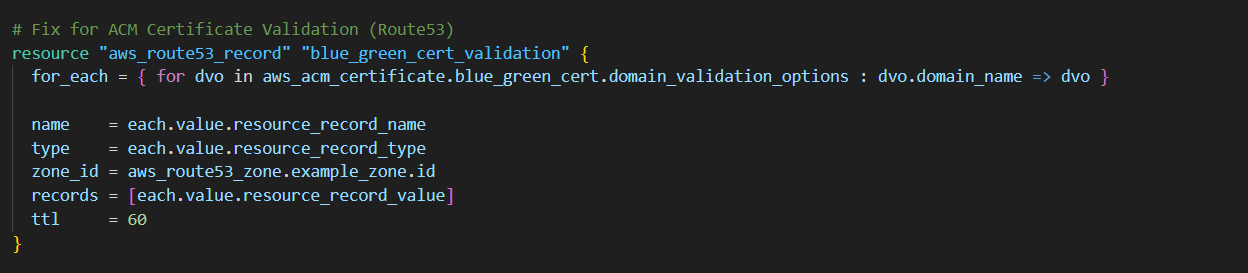
**5. SSL Certificate**

For HTTPS access, you'll need an SSL certificate from AWS Certificate Manager (ACM).

**# SSL Certificate for the domain**



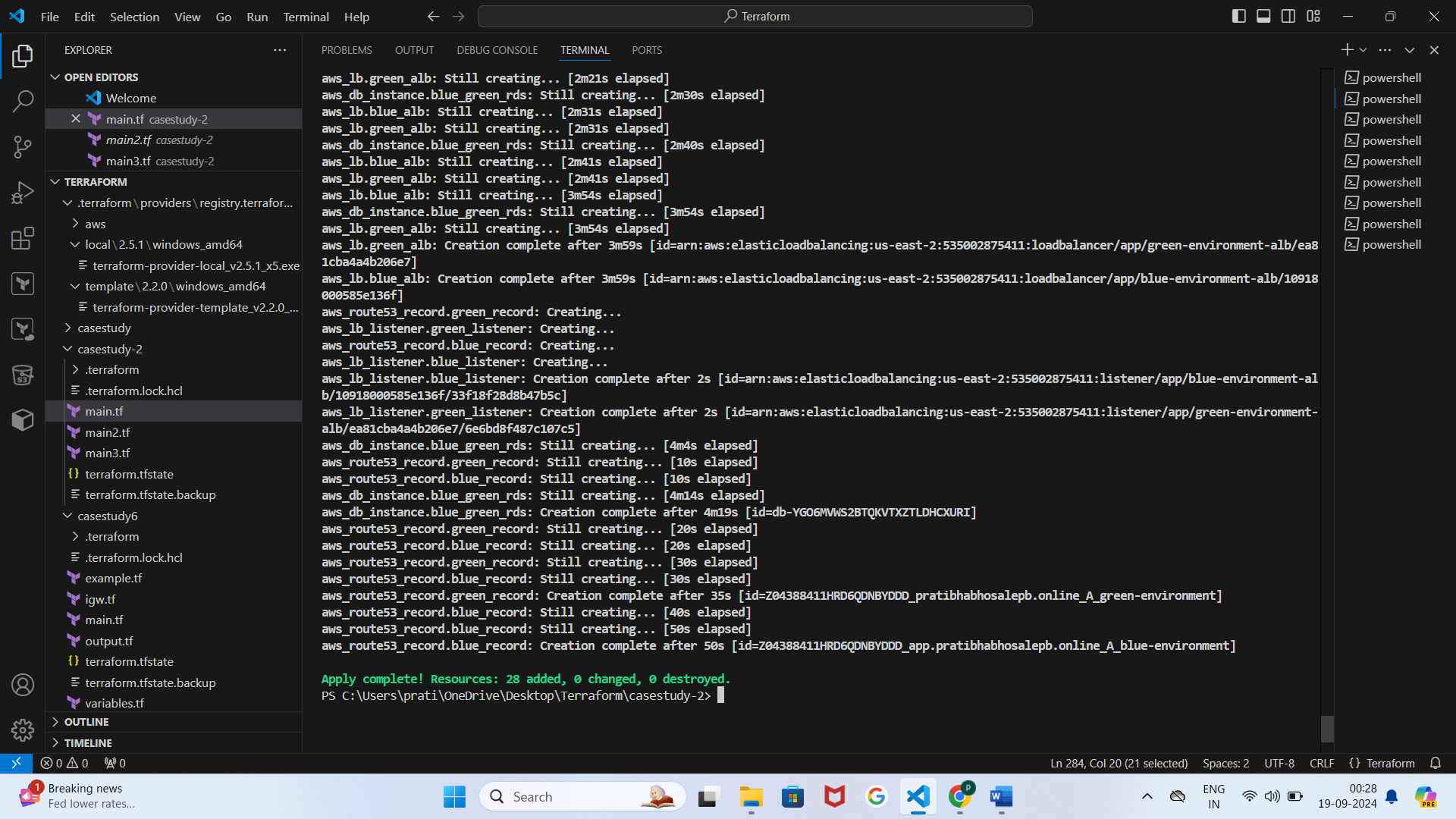
**# DNS validation records**]



Terraform init

Terraform plan

Terraform apply



**Step-4 : Handling Backend Database Endpoints**

In a Blue/Green deployment, managing database endpoints is crucial to ensure that the application can access the database seamlessly while switching environments. Here's how we can handle backend database endpoints:

1. Database Endpoint Configuration

Scenario: we have a single database that the application needs to connect to. To avoid downtime during deployment switches, you should ensure that the application’s database configuration is abstracted in a way that allows you to change it easily.

Option 1: Use Route 53 for Database Endpoints

If you're using Amazon RDS or any managed database service, you can manage database endpoints with Route 53. Create a Route 53 record to point to the current database instance.

# Route 53 Record for Database Endpoint

resource "aws\_route53\_record" "db\_endpoint" {

zone\_id = aws\_route53\_zone.main.id

name = "db.example.com"

type = "CNAME"

ttl = 60

records = ["my-current-db-instance.abcdefg12345.us-east-1.rds.amazonaws.com"]

}

When you switch to the new version, you can update this record to point to the new database instance, minimizing downtime.

Option 2: Environment-Specific Database Instances

If you use different database instances for Blue and Green environments (e.g., Blue DB and Green DB), configure your application to switch between these endpoints based on the environment.

2. Update Application Configuration

Ensure your application can dynamically read database configuration from environment variables or a configuration management system.

For instance, if you're using AWS Systems Manager Parameter Store or AWS Secrets Manager:

# SSM Parameter for Database Endpoint

resource "aws\_ssm\_parameter" "db\_endpoint" {

name = "/app/db/endpoint"

type = "String"

value = "my-current-db-instance.abcdefg12345.us-east-1.rds.amazonaws.com"

}

Your application should fetch this parameter during runtime and use it to connect to the database.

3. Deployment Automation

Automate the process of updating database endpoints during deployment. This can be done by using deployment scripts or CI/CD pipelines that:

Update Route 53 records if using a single database instance.

Change environment-specific database configurations if using separate databases.

Here’s a simple example of a deployment script using AWS CLI:

#!/bin/bash

# Update database endpoint for Green environment

aws ssm put-parameter --name "/app/db/endpoint" --value "my-green-db-instance.abcdefg12345.us-east-1.rds.amazonaws.com" --type "String" --overwrite

4. Database Migration and Synchronization

If your Blue and Green environments use separate databases, ensure data consistency between them. Use database migration tools or strategies to synchronize data.

AWS Database Migration Service (DMS): Can be used to replicate data between databases.

Manual Synchronization: Periodically export and import data as needed.

Github-repo- <https://github.com/Pratibha251093/case-study-2.git>