

# FULLY AUTOMATED AWS INFRASTRUCTURE BY TERRAFORM

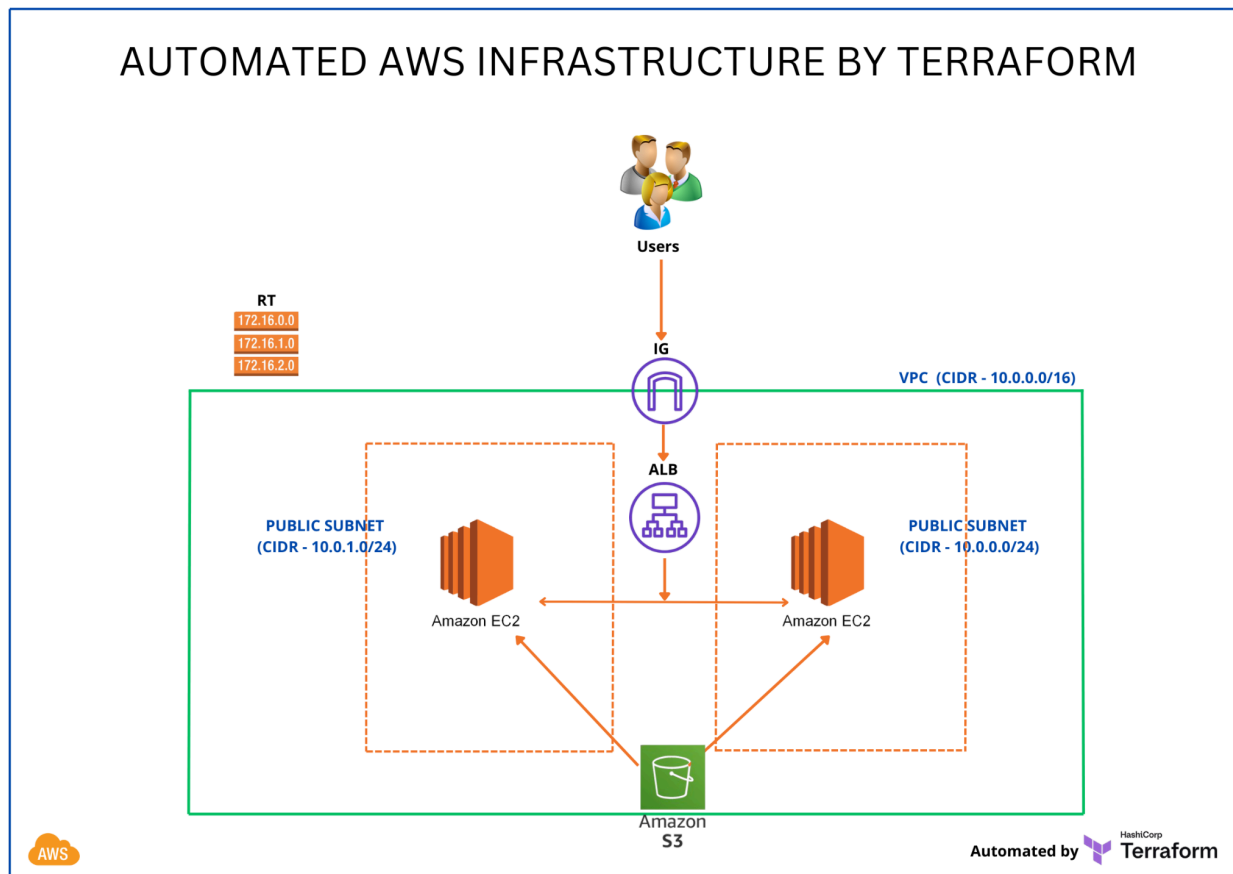


Figure 01

# Step by Step Comprehensive guide to deploy an automated AWS Infrastructure

## Article Description

In this article, we delve into the automation of AWS infrastructure using Terraform. The provided Terraform configuration script demonstrates how to set up a comprehensive AWS environment, including a Virtual Private Cloud (VPC), subnets, an Internet Gateway, route tables, security groups, EC2 instances, an S3 bucket, and an Application Load Balancer (ALB) with target groups and listeners.

I will guide you through each component of the configuration, explaining its purpose and how it contributes to the overall architecture. By the end of this article, you will have a clear understanding of how to automate the deployment of a scalable and secure AWS infrastructure using Terraform, enabling you to manage your cloud resources efficiently and effectively."

## Understanding

I have deployed 2 Ec2 instances in 2 different Availability zones (`us-east-1a`, `us-east-1b`) with 2 different CIDR IP blocks. User traffic will flow through the Load balancer while ensuring high availability and fault tolerance. S3 bucket supplies the static web contents to the web application hosted in EC2 instances.

## Steps to Implement the AWS Automation Project with Terraform

1. Install Terraform:
  - Download and install Terraform from the official website.
  - Verify the installation by running `terraform -v` in your terminal.
2. Set Up AWS CLI:
  - Install the AWS CLI from the official website.
  - Configure the AWS CLI with your credentials by running `aws configure`.
3. Create a New Directory for Your Project:
  - Open a terminal and create a new directory for your Terraform project.

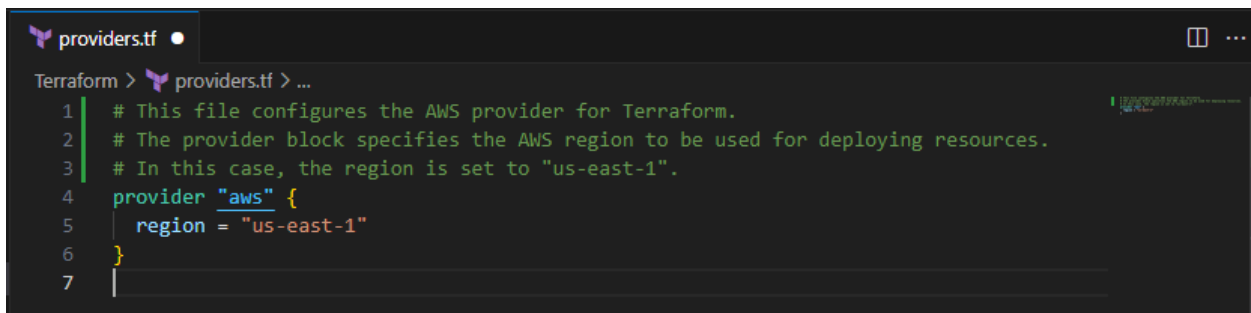
```
mkdir terraform-aws-project  
cd terraform-aws-project
```

Figure 02

4. Create the Main Terraform Configuration File:
  - Create a file named main.tf and add the provided Terraform configuration code.

## Configurations with Descriptive codes

1. Create Provider.tf file to assign the platform/platforms are going to use. In our case I am going to use the AWS Cloud platform. If we need a multi cloud platform to use we have to define the provider's names in this file.



```
providers.tf  
Terraform > providers.tf > ...  
1 | # This file configures the AWS provider for Terraform.  
2 | # The provider block specifies the AWS region to be used for deploying resources.  
3 | # In this case, the region is set to "us-east-1".  
4 | provider "aws" {  
5 |   region = "us-east-1"  
6 | }  
7 |
```

Figure 03

2. We can use separate files to keep the separation along with the project.  
Eg: variables.tf, terraform.tfvars,



```
variables.tf  
Terraform > variables.tf > variable "cidr"  
1 | # This variable defines the CIDR block for the network.  
2 | # The default value is set to "10.0.0.0/16".  
3 | variable "cidr" {  
4 |   default = "10.0.0.0/16"  
5 | }
```

Figure 04

3. Now create the main.tf file to resource allocations.
  - ❖ Create VPC - CIDR - 10.0.0.0/16

```
main.tf M ●
Terraform > main.tf > ...
1 | # Description: This Terraform configuration defines an AWS VPC resource with a CIDR block specified by a variable.
2 | # The VPC is tagged with the name "myvpc".
3 | resource "aws_vpc" "myvpc" {
4 |     cidr_block = var.cidr
5 |
6 |     tags = {
7 |         Name = "myvpc"
8 |     }
9 | }
10
```

Figure 05

4. Create Subnet 1 - CIDR - 10.0.0.0/24

```
10
11 | # This resource block creates an AWS subnet within a specified VPC.
12 | # The subnet is configured with a CIDR block of 10.0.0.0/24 and is located in the us-east-1a availability zone.
13 | # The subnet is set to automatically assign public IP addresses to instances launched within it.
14 | # A tag is added to the subnet with the name "sub1".
15 | resource "aws_subnet" "sub1" {
16 |     vpc_id = aws_vpc.myvpc.id
17 |     cidr_block = "10.0.0.0/24"
18 |     availability_zone = "us-east-1a"
19 |     map_public_ip_on_launch = true
20 |
21 |     tags = {
22 |         Name = "sub1"
23 |     }
24 | }
25
```

Figure 06

5. Create Subnet 2 - CIDR - 10.0.1.0/24

```
26 | # This resource block defines an AWS subnet named "sub2" within a specified VPC.
27 | # The subnet is assigned a CIDR block of "10.0.1.0/24" and is located in the "us-east-1b" availability zone.
28 | # The "map_public_ip_on_launch" attribute is set to true, which means instances launched in this subnet will
29 | # automatically receive a public IP address.
30 | # The subnet is tagged with the name "sub2".
31 | resource "aws_subnet" "sub2" {
32 |     vpc_id = aws_vpc.myvpc.id
33 |     cidr_block = "10.0.1.0/24"
34 |     availability_zone = "us-east-1b"
35 |     map_public_ip_on_launch = true
36 |
37 |     tags = {
38 |         Name = "sub2"
39 |     }
40 | }
```

Figure 07

## 6. Create Internet Gateway to the VPC

```
42 # This resource block creates an AWS Internet Gateway and attaches it to the specified VPC.
43 # The 'vpc_id' attribute references the ID of the VPC to which the Internet Gateway will be attached.
44 # The 'tags' attribute assigns a name tag to the Internet Gateway for easier identification.
45 resource "aws_internet_gateway" "myigw" {
46     vpc_id = aws_vpc.myvpc.id
47
48     tags = {
49         Name = "myigw"
50     }
51 }
52 }
```

Figure 08

## 7. Create route table and associate with internet gateway

```
54 # Description: This Terraform configuration defines an AWS route table resource named "myRT" associated with a specified
55 #VPC.
56 # It includes a route that directs all traffic (0.0.0.0/0) to an internet gateway.
57 # Additionally, it tags the route table with the name "myRT".
58 resource "aws_route_table" "myRT" {
59     vpc_id = aws_vpc.myvpc.id
60
61     route {
62         cidr_block = "0.0.0.0/0"
63         gateway_id = aws_internet_gateway.myigw.id
64     }
65
66     tags = {
67         Name = "myRT"
68     }
69 }
```

Figure 09

## 8. Create route table association with Subnet 1

```
72 # Associates a subnet with a route table to define the routing rules for the subnet.
73 # This resource links the specified subnet (subnet_id) with the specified route table (route_table_id).
74 resource "aws_route_table_association" "myRTA1" {
75     subnet_id = aws_subnet.sub1.id
76     route_table_id = aws_route_table.myRT.id
77 }
78
79
```

Figure 10

## 9. Create route table association with Subnet 2

```
80 # Associates a subnet with a route table
81 # This resource links the specified subnet (subnet_id) to the specified route table (route_table_id).
82 # The association ensures that the routes defined in the route table are applied to the subnet.
83 resource "aws_route_table_association" "myRTA2" {
84     subnet_id = aws_subnet.sub2.id
85     route_table_id = aws_route_table.myRT.id
86 }
```

Figure 11

## 10. Create a Security group

```
Terraform > main.tf > ...
88 # This Terraform configuration defines an AWS Security Group resource named "mySG".
89 # The security group is associated with a VPC identified by aws_vpc.myvpc.id.
90 #
91 # Ingress rules:
92 # - Allows inbound SSH traffic (port 22) from any IP address (0.0.0.0/0).
93 # - Allows inbound HTTP traffic (port 80) from any IP address (0.0.0.0/0).
94 #
95 # Egress rules:
96 # - Allows all outbound traffic to any IP address (0.0.0.0/0).
97 #
98 # The security group is tagged with the name "my-SG".
99 resource "aws_security_group" "mySG" {
100     vpc_id = aws_vpc.myvpc.id
101     name = "mySG"
102
103     ingress {
104         from_port = 22
105         to_port = 22
106         protocol = "tcp"
107         cidr_blocks = ["0.0.0.0/0"]
108     }
109
110     ingress {
111         from_port = 80
112         to_port = 80
113         protocol = "tcp"
114         cidr_blocks = ["0.0.0.0/0"]
115     }
116
117     egress {
118         from_port = 0
119         to_port = 0
120         protocol = "-1"
121         cidr_blocks = ["0.0.0.0/0"]
122     }
123
124     tags = {
125         Name = "my-SG"
126     }
127 }
```

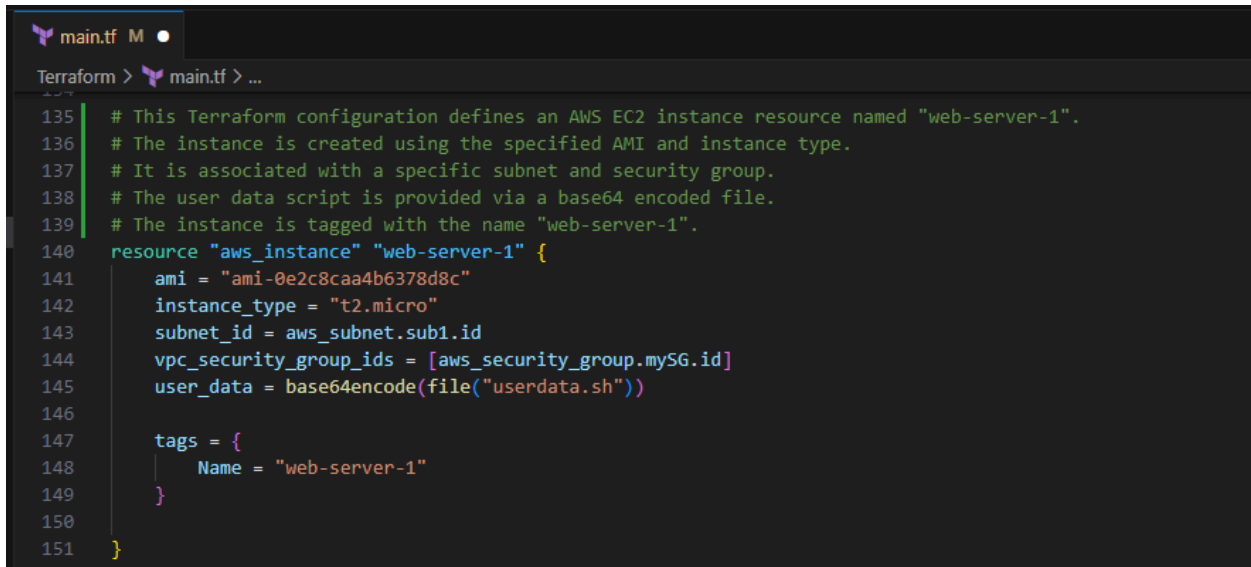
Figure 12

## 11. Create a S3 bucket

```
129 # This Terraform configuration defines an AWS S3 bucket resource.
130 # The bucket is named "udantha-test-bucket-123".
131 resource "aws_s3_bucket" "mybucket" {
132     bucket = "udantha-test-bucket-123"
133 }
134
```

Figure 13

## 12. Create EC2 instance 1

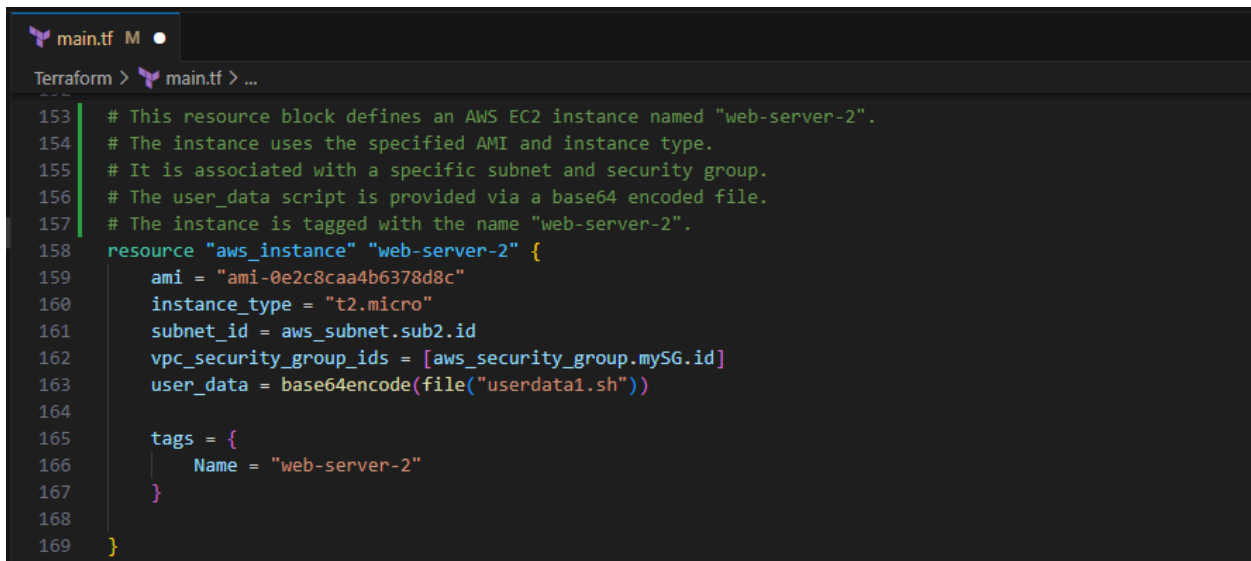


```
main.tf M
Terraform > main.tf > ...

135 # This Terraform configuration defines an AWS EC2 instance resource named "web-server-1".
136 # The instance is created using the specified AMI and instance type.
137 # It is associated with a specific subnet and security group.
138 # The user data script is provided via a base64 encoded file.
139 # The instance is tagged with the name "web-server-1".
140 resource "aws_instance" "web-server-1" {
141     ami = "ami-0e2c8caa4b6378d8c"
142     instance_type = "t2.micro"
143     subnet_id = aws_subnet.sub1.id
144     vpc_security_group_ids = [aws_security_group.mySG.id]
145     user_data = base64encode(file("userdata.sh"))
146
147     tags = {
148         Name = "web-server-1"
149     }
150
151 }
```

Figure 14

## 13. Create EC2 instance 2



```
main.tf M
Terraform > main.tf > ...

153 # This resource block defines an AWS EC2 instance named "web-server-2".
154 # The instance uses the specified AMI and instance type.
155 # It is associated with a specific subnet and security group.
156 # The user_data script is provided via a base64 encoded file.
157 # The instance is tagged with the name "web-server-2".
158 resource "aws_instance" "web-server-2" {
159     ami = "ami-0e2c8caa4b6378d8c"
160     instance_type = "t2.micro"
161     subnet_id = aws_subnet.sub2.id
162     vpc_security_group_ids = [aws_security_group.mySG.id]
163     user_data = base64encode(file("userdata1.sh"))
164
165     tags = {
166         Name = "web-server-2"
167     }
168
169 }
```

Figure 15

## 14. Create the Load Balancer

```
171 # This resource creates an AWS Application Load Balancer (ALB) named "mylb".
172 # The ALB is internet-facing (internal = false) and uses the specified security group and subnets.
173 # Tags are added to help identify the resource.
174 resource "aws_lb" "mylb" {
175     name = "mylb"
176     internal = false
177     load_balancer_type = "application"
178     security_groups = [aws_security_group.mySG.id]
179     subnets = [aws_subnet.sub1.id, aws_subnet.sub2.id]
180
181     tags = {
182         Name = "mylb"
183     }
184 }
185
186 }
```

Figure 16

## 15. Create Load balancer Target group

```
188 # DESCRIPTION: This Terraform configuration defines an AWS Load Balancer Target Group resource.
189 # The target group is named "myTG" and listens on port 80 using the HTTP protocol.
190 # It targets instances within a specified VPC and includes a health check configuration.
191 # The health check monitors the root path ("/") over HTTP, with a check interval of 30 seconds,
192 # a timeout of 5 seconds, and thresholds for healthy and unhealthy states set to 2.
193 # Additionally, the target group is tagged with the name "myTG".
194 resource "aws_lb_target_group" "myTG" {
195     name = "myTG"
196     port = 80
197     protocol = "HTTP"
198     target_type = "instance"
199     vpc_id = aws_vpc.myvpc.id
200
201     health_check {
202         path = "/"
203         protocol = "HTTP"
204         port = "traffic-port"
205         interval = 30
206         timeout = 5
207         healthy_threshold = 2
208         unhealthy_threshold = 2
209     }
210
211     tags = {
212         Name = "myTG"
213     }
214 }
```

Figure 17



## 16. Create Load balancer target group attachment 1

```
Terraform > main.tf > ...
217 # Attaches an EC2 instance to an ALB target group
218 #
219 # This resource attaches the specified EC2 instance to the specified
220 # Application Load Balancer (ALB) target group. The instance will
221 # receive traffic on the specified port.
222 #
223 # Arguments:
224 # - target_group_arn: The ARN of the target group to which the instance will be attached.
225 # - target_id: The ID of the EC2 instance to attach to the target group.
226 # - port: The port on which the instance will receive traffic from the load balancer.
227 resource "aws_alb_target_group_attachment" "attach1" {
228     target_group_arn = aws_lb_target_group.myTG.arn
229     target_id = aws_instance.web-server-1.id
230     port = 80
231 }
232 }
```

Figure 18

## 17. Create Load balancer target group attachment 2

```
Terraform > main.tf > ...
234 # This resource attaches an EC2 instance to an ALB target group.
235 #
236 # Arguments:
237 # - target_group_arn: The ARN of the target group to which the instance will be attached.
238 # - target_id: The ID of the EC2 instance to attach to the target group.
239 # - port: The port on which the target is listening.
240 resource "aws_alb_target_group_attachment" "attach2" {
241     target_group_arn = aws_lb_target_group.myTG.arn
242     target_id = aws_instance.web-server-2.id
243     port = 80
244 }
245 }
```

Figure 19

## 18. Create a Listener

```
Terraform > main.tf > ...
247 # This resource defines an AWS Load Balancer Listener.
248 # The listener is associated with a specific load balancer and listens on port 80 using the HTTP protocol.
249 # It forwards incoming traffic to a target group.
250 #
251 # Arguments:
252 # - load_balancer_arn: The ARN of the load balancer to associate with the listener.
253 # - port: The port on which the listener will accept incoming traffic (80 in this case).
254 # - protocol: The protocol for connections from clients to the load balancer (HTTP in this case).
255 # - default_action: The action to take when a request does not match any of the listener's rules.
256 # - type: The type of action (forward in this case).
257 # - target_group_arn: The ARN of the target group to forward traffic to.
258 resource "aws_lb_listener" "listener" {
259     load_balancer_arn = aws_lb.mylb.arn
260     port = 80
261     protocol = "HTTP"
262
263     default_action {
264         type = "forward"
265         target_group_arn = aws_lb_target_group.myTG.arn
266     }
267
268 }
269 }
```

Figure 20

## 19. Finally Create an output to get the Load balancer dns name

```
271 # This output block exports the DNS name of the AWS load balancer (mylb).
272 # The value is retrieved from the 'dns_name' attribute of the 'aws_lb' resource named 'mylb'.
273 output "loadbalancerdns" {
274     value = aws_lb.mylb.dns_name
275 }
276 }
```

Figure 21

20. Now we have to create a 2 scripts to install apache server while provisioning the 2 of Ec2 instances to start setting up 2 different web contents.

```
$ userdata.sh
Terraform > $ userdata.sh
1  #!/bin/bash
2  apt update
3  apt install -y apache2
4
5  # Get the instance ID using the instance metadata
6  INSTANCE_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)
7
8  # Install the AWS CLI
9  apt install -y awscli
10
11 # Download the images from S3 bucket
12 aws s3 cp s3://myterraformprojectbucket2023/project.webp /var/www/html/project.png --acl public-read
13
14 # Create a simple HTML file with the portfolio content and display the images
15 cat <<EOF > /var/www/html/index.html
16 <!DOCTYPE html>
17 <html>
18 <head>
19 <title>Web_Server_01</title>
20 <style>
21 /* Add animation and styling for the text */
22 @keyframes colorChange {
23 0% { color: red; }
24 50% { color: green; }
25 100% { color: blue; }
26 }
27 h1 {
28 animation: colorChange 2s infinite;
29 }
30 </style>
31 </head>
32 <body>
33 <h1>Terraform Project Server 1</h1>
34 <h2>Instance ID: <span style="color:green">${INSTANCE_ID}</span></h2>
35 <p>Welcome to Udantha's Web Server 01</p>
36 </body>
37 </html>
38 EOF
39
40 # Start Apache and enable it on boot
41 systemctl start apache2
42 systemctl enable apache2
```

Figure 22

### 21. Initialize the Terraform Project:

Run `terraform init` in your project directory to initialize the project and download the necessary providers.

### 22. Plan the Infrastructure:

Run a **terraform plan** to see a preview of the changes that will be made by your configuration.

### 23. Apply the Configuration:

Run **terraform apply** to create the resources defined in your configuration. Confirm the action when prompted.

### 24. Verify the Deployment:

Log in to your AWS Management Console and verify that the VPC, subnets, and other resources have been created as specified.

### 25. Clean Up Resources:

When you are done with the project, you can clean up the resources by running **terraform destroy**.

## Conclusion

In this project, I have demonstrated how to automate the deployment of a comprehensive AWS infrastructure using Terraform. By following the steps outlined, I have successfully created and configured various AWS resources, including a VPC, subnets, an Internet Gateway, route tables, security groups, EC2 instances, an S3 bucket, and an Application Load Balancer with target groups and listeners.

This automation not only simplifies the process of setting up and managing cloud resources but also ensures consistency and repeatability in your infrastructure deployments. By leveraging Terraform's declarative approach, you can easily version control your infrastructure code, collaborate with team members, and scale your environment as needed.

# THANK YOU