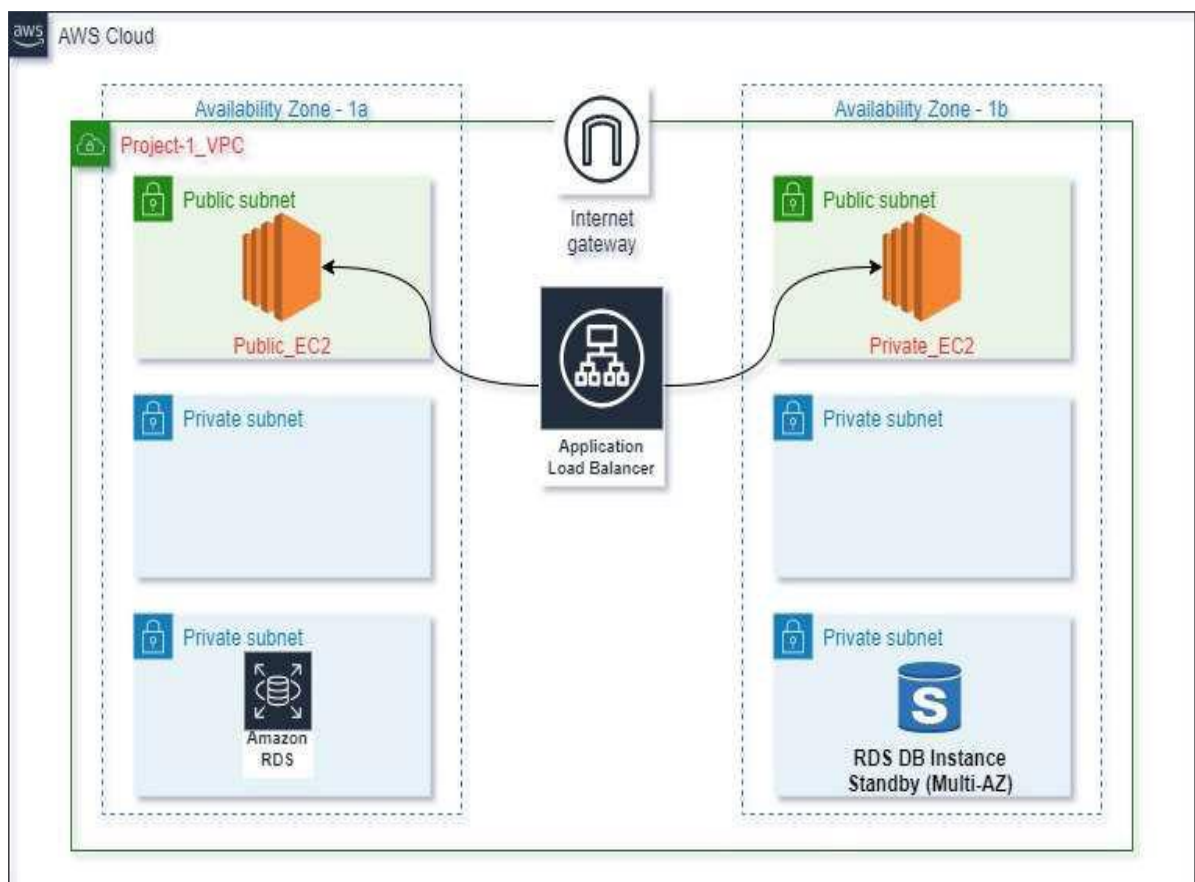


Deploy Three-Tier Architecture in AWS using Terraform

❖ What is Terraform?

Terraform is an open-source infrastructure as a code (IAC) tool that allows to create, manage & deploy the production-ready environment. Terraform codifies cloud APIs into declarative configuration files. Terraform can manage both existing service providers and custom in-house solutions.

❖ In this project, we will deploy a three-tier application in AWS using Terraform.

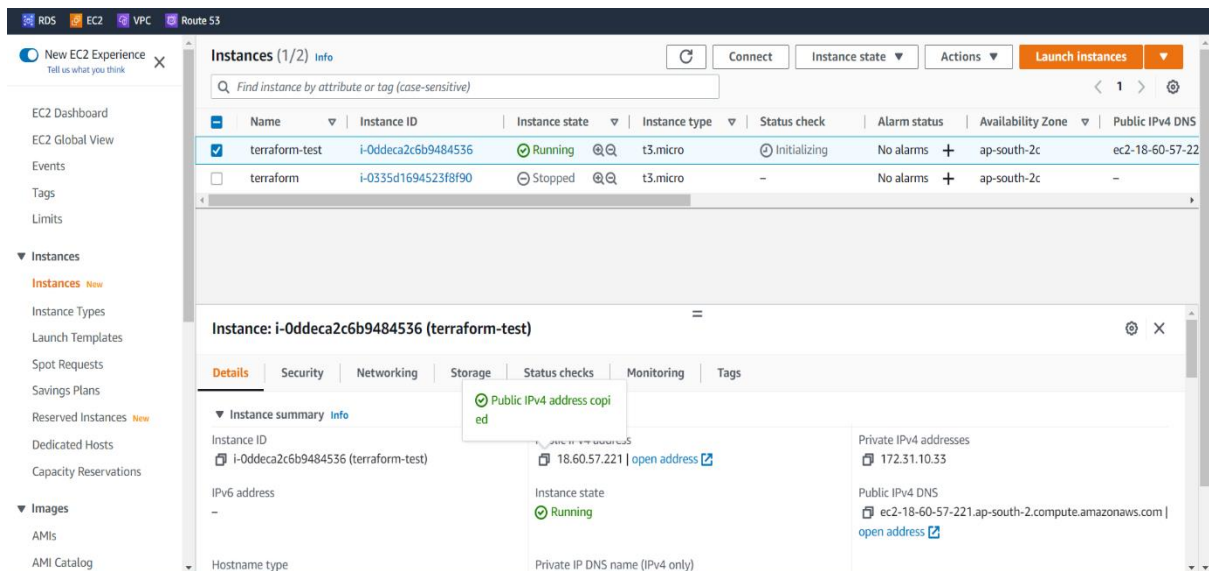


❖ Prerequisites:

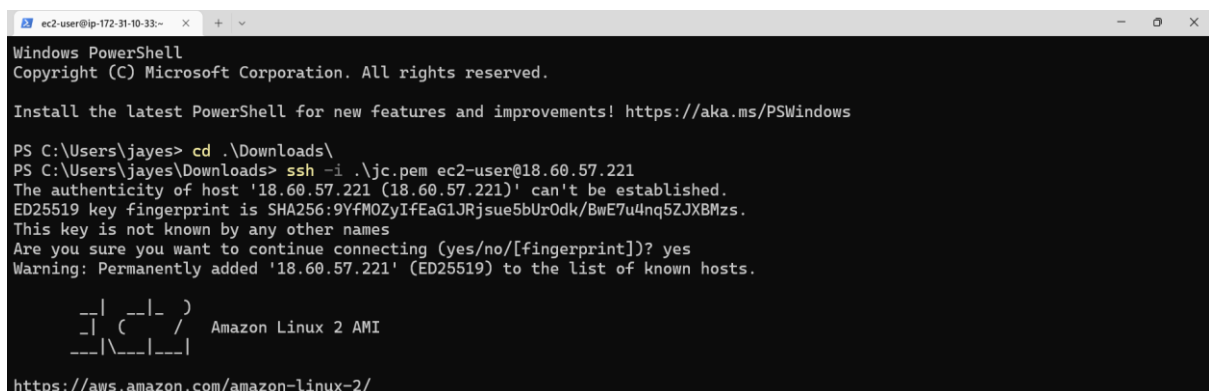
- ✓ Basic knowledge of AWS & Terraform
- ✓ AWS Account
- ✓ IAM User
- ✓ GitHub Account
- ✓ AWS Access & Secret Key

❖ Procedure:

- Launch a EC2 instance using amazon Linux 2 image and give SSH access to it.



- Connect the instance through SSH using terminal.



➤ Install Terraform in the Instance using the below commands:

- `sudo yum -y install yum-utils`

```
[ec2-user@ip-172-31-10-33 ~]$ sudo yum install -y yum-utils
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package yum-utils-1.1.31-46.amzn2.0.1.noarch already installed and latest version
Nothing to do
```

- `sudo yum-config-manager --add-repo`
<https://rpm.releases.hashicorp.com/amazonLinux/hashicorp.repo>

```
[ec2-user@ip-172-31-10-33 ~]$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
adding repo from: https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
grabbing file https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo to /etc/yum.repos.d/hashicorp.repo
repo saved to /etc/yum.repos.d/hashicorp.repo
[ec2-user@ip-172-31-10-33 ~]$
```

- `sudo yum -y install terraform`

```
[ec2-user@ip-172-31-10-33 ~]$ sudo yum -y install terraform
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core | 3.7 kB
00:00:00
hashicorp | 1.4 kB
00:00:00
hashicorp/x86_64/primary | 122 kB
00:00:00
hashicorp
873/873
Resolving Dependencies
--> Running transaction check
--> Package terraform.x86_64 0:1.3.6-1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
===== Package                Arch                Version             Reposi
tory                Size
=====
=====Installing:
terraform            x86_64              1.3.6-1             hashicorp
```

- Create terraform files of requirements that we want to create.

Step 1: Create a file for the VPC

- Create vpc.tf file and add the below code to it

```
# Creating VPC
resource "aws_vpc" "demovpc" {
  cidr_block = "${var.vpc_cidr}"
  instance_tenancy = "default"
  tags = {
    Name = "Demo VPC"
  }
}
```

A screenshot of a terminal window with a dark background. The terminal shows the same Terraform code as the previous block, starting with a comment '# Creating VPC' followed by the resource definition for 'aws_vpc' named 'demovpc'. The code includes attributes for 'cidr_block', 'instance_tenancy', and 'tags' with a name of 'Demo VPC'. The terminal window has a title bar at the top showing 'ec2-user@ip-172-31-10-32:~/I' and standard window controls.

Step 2: Create a file for the Subnet

- For this project, we will create total 6 subnets for the front-end tier and back-end tier with a mixture of public & private subnet.
- Create subnet.tf file and add the below code to it

```
# Creating 1st web subnet
resource "aws_subnet" "public-subnet-1" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet_cidr}"
  map_public_ip_on_launch = true
  availability_zone = "us-east-1a"
  tags = {
    Name = "Web Subnet 1"
  }
}

# Creating 2nd web subnet
resource "aws_subnet" "public-subnet-2" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet1_cidr}"
  map_public_ip_on_launch = true
```

```

availability_zone = "us-east-1b"
tags = {
  Name = "Web Subnet 2"
}
}
# Creating 1st application subnet
resource "aws_subnet" "application-subnet-1" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet2_cidr}"
  map_public_ip_on_launch = false
  availability_zone = "us-east-1a"
  tags = {
    Name = "Application Subnet 1"
  }
}
# Creating 2nd application subnet
resource "aws_subnet" "application-subnet-2" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet3_cidr}"
  map_public_ip_on_launch = false
  availability_zone = "us-east-1b"
  tags = {
    Name = "Application Subnet 2"
  }
}
# Create Database Private Subnet
resource "aws_subnet" "database-subnet-1" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet4_cidr}"
  availability_zone = "us-east-1a"
  tags = {
    Name = "Database Subnet 1"
  }
}
# Create Database Private Subnet
resource "aws_subnet" "database-subnet-2" {
  vpc_id = "${aws_vpc.demovpc.id}"
  cidr_block = "${var.subnet5_cidr}"
  availability_zone = "us-east-1a"
  tags = {
    Name = "Database Subnet 1"
  }
}

```

```
Windows PowerShell x Windows PowerShell x + v
# Creating 1st web subneclient_loop: send disconnect: Connection reset
PS C:\Users\javes\Downloads> |-subnet-1" {
vpc_id = "${aws_vpc.demovpc.id}"
cidr_block = "${var.subnet_cidr}"
map_public_ip_on_launch = true
availability_zone = "ap-south-2a"
tags = {
Name = "Web Subnet 1"
}
}
# Creating 2nd web subnet
resource "aws_subnet" "public-subnet-2" {
vpc_id = "${aws_vpc.demovpc.id}"
cidr_block = "${var.subnet1_cidr}"
map_public_ip_on_launch = true
availability_zone = "ap-south-2b"
tags = {
Name = "Web Subnet 2"
}
}
# Creating 1st application subnet
resource "aws_subnet" "application-subnet-1" {
vpc_id = "${aws_vpc.demovpc.id}"
cidr_block = "${var.subnet2_cidr}"
map_public_ip_on_launch = false
availability_zone = "ap-south-2a"
tags = {
Name = "Application Subnet 1"
}
}
# Creating 2nd application subnet
resource "aws_subnet" "application-subnet-2" {
1,25 Top
```

Step 3: Create a file for the Internet Gateway

- Create igw.tf file and add the below code to it

```
# Creating Internet Gateway
resource "aws_internet_gateway" "demogateway" {
vpc_id = "${aws_vpc.demovpc.id}"
}
```

```
ec2-user@ip-172-31-10-33:~/1 x Windows PowerShell x + v
# Creating Internet Gateway
resource "aws_internet_gateway" "demogateway" {
vpc_id = "${aws_vpc.demovpc.id}"
}
~
~
~
~
```

Step 4: Create a file for the Route table

- Create route_table_public.tf file and add the below code to it

```
# Creating Route Table
resource "aws_route_table" "route" {
vpc_id = "${aws_vpc.demovpc.id}"
route {
```

```

cidr_block = "0.0.0.0/0"
gateway_id = "${aws_internet_gateway.demogateway.id}"
}
tags = {
Name = "Route to internet"
}
}
# Associating Route Table
resource "aws_route_table_association" "rt1" {
subnet_id = "${aws_subnet.demosubnet.id}"
route_table_id = "${aws_route_table.route.id}"
}
# Associating Route Table
resource "aws_route_table_association" "rt2" {
subnet_id = "${aws_subnet.demosubnet1.id}"
route_table_id = "${aws_route_table.route.id}"
}
}

```

The screenshot shows a Windows PowerShell terminal window with the following Terraform code:

```

# Creating Route Table
resource "aws_route_table" "route" {
vpc_id = "${aws_vpc.demovpc.id}"
route {
cidr_block = "0.0.0.0/0"
gateway_id = "${aws_internet_gateway.demogateway.id}"
}
tags = {
Name = "Route to internet"
}
}
# Associating Route Table
resource "aws_route_table_association" "rt1" {
subnet_id = "${aws_subnet.public-subnet-1.id}"
route_table_id = "${aws_route_table.route.id}"
}
# Associating Route Table
resource "aws_route_table_association" "rt2" {
subnet_id = "${aws_subnet.public-subnet-2.id}"
route_table_id = "${aws_route_table.route.id}"
}
}
~
~
~
~
~
~

```

The terminal window title bar shows "ec2-user@ip-172-31-10-33:~/1" and "Windows PowerShell". The bottom right corner of the terminal displays "19, 41" and "All".

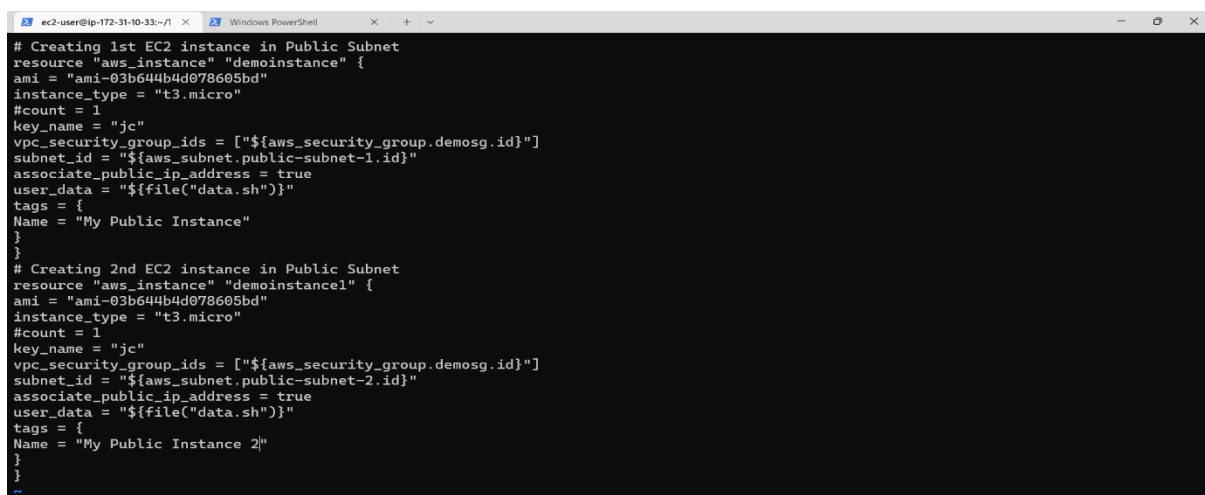
- In the above code, we are creating a new route table and forwarding all the requests to the 0.0.0.0/0 CIDR block.
- we also attaching this route table to the subnet created earlier. So, it will work as the Public Subnet

Step 5: Create a file for EC2 instances

- Create ec2.tf file and add the below code to it

```
# Creating 1st EC2 instance in Public Subnet
resource "aws_instance" "demoinstance" {
  ami = "ami-087c17d1fe0178315"
  instance_type = "t2.micro"
  count = 1
  key_name = "tests"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id = "${aws_subnet.demoinstance.id}"
  associate_public_ip_address = true
  user_data = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance"
  }
}

# Creating 2nd EC2 instance in Public Subnet
resource "aws_instance" "demoinstance1" {
  ami = "ami-087c17d1fe0178315"
  instance_type = "t2.micro"
  count = 1
  key_name = "tests"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id = "${aws_subnet.demoinstance.id}"
  associate_public_ip_address = true
  user_data = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance 2"
  }
}
```



```
ec2-user@ip-172-31-10-33:~$ cat ec2.tf
# Creating 1st EC2 instance in Public Subnet
resource "aws_instance" "demoinstance" {
  ami = "ami-03b644b4d078605bd"
  instance_type = "t3.micro"
  #count = 1
  key_name = "jc"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id = "${aws_subnet.public-subnet-1.id}"
  associate_public_ip_address = true
  user_data = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance"
  }
}

# Creating 2nd EC2 instance in Public Subnet
resource "aws_instance" "demoinstance1" {
  ami = "ami-03b644b4d078605bd"
  instance_type = "t3.micro"
  #count = 1
  key_name = "jc"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id = "${aws_subnet.public-subnet-2.id}"
  associate_public_ip_address = true
  user_data = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance 2"
  }
}
```

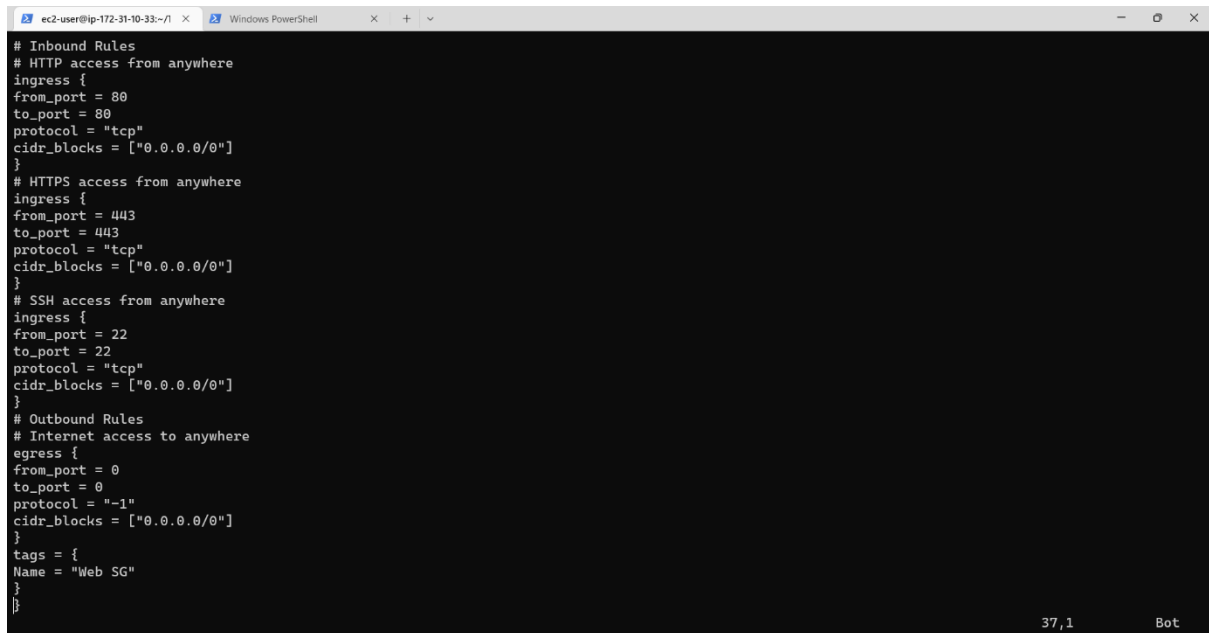

- We will use the userdata to configure the EC2 instance, we will discuss data.sh file later

Step 6: Create a file for Security Group for the Frontend tier

- Create web_sg.tf file and add the below code to it

```
# Creating Security Group
resource "aws_security_group" "demosg" {
  vpc_id = "${aws_vpc.demovpc.id}"
  # Inbound Rules
  # HTTP access from anywhere
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # HTTPS access from anywhere
  ingress {
    from_port = 443
    to_port = 443
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # SSH access from anywhere
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # Outbound Rules
  # Internet access to anywhere
  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "Web SG"
  }
}
```

```
}  
}
```



```
ec2-user@ip-172-31-10-33:~/l x Windows PowerShell x + v  
# Inbound Rules  
# HTTP access from anywhere  
ingress {  
  from_port = 80  
  to_port = 80  
  protocol = "tcp"  
  cidr_blocks = ["0.0.0.0/0"]  
}  
# HTTPS access from anywhere  
ingress {  
  from_port = 443  
  to_port = 443  
  protocol = "tcp"  
  cidr_blocks = ["0.0.0.0/0"]  
}  
# SSH access from anywhere  
ingress {  
  from_port = 22  
  to_port = 22  
  protocol = "tcp"  
  cidr_blocks = ["0.0.0.0/0"]  
}  
# Outbound Rules  
# Internet access to anywhere  
egress {  
  from_port = 0  
  to_port = 0  
  protocol = "-1"  
  cidr_blocks = ["0.0.0.0/0"]  
}  
tags = {  
  Name = "Web SG"  
}  
}
```

37,1 Bot

- Here, we opened 80,443 & 22 ports for the inbound connection and we are opened all the ports for the outbound connection

Step 7: Create a file for Security Group for the Database tier

- Create database_sg.tf file and add the below code to it

```
# Create Database Security Group  
resource "aws_security_group" "database-sg" {  
  name = "Database SG"  
  description = "Allow inbound traffic from application layer"  
  vpc_id = aws_vpc.demovpc.id  
  ingress {  
    description = "Allow traffic from application layer"  
    from_port = 3306  
    to_port = 3306  
    protocol = "tcp"  
    security_groups = [aws_security_group.demosg.id]  
  }  
  egress {  
    from_port = 32768  
    to_port = 65535  
    protocol = "tcp"
```

```

cidr_blocks = ["0.0.0.0/0"]
}
tags = {
  Name = "Database SG"
}
}

```

```

# Create Database Security Group
resource "aws_security_group" "database-sg" {
  name = "Database SG"
  description = "Allow inbound traffic from application layer"
  vpc_id = aws_vpc.demovpc.id
  ingress {
    description = "Allow traffic from application layer"
    from_port = 3306
    to_port = 3306
    protocol = "tcp"
    security_groups = [aws_security_group.demosg.id]
  }
  egress {
    from_port = 32768
    to_port = 65535
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "Database SG"
  }
}

```

18,1 All

- We opened 3306 ports for the inbound connection and We are opened all the ports for the outbound connection.

Step 8: Create a file Application Load Balancer

- Create alb.tf file and add the below code to it

```

# Creating External LoadBalancer
resource "aws_lb" "external-alb" {
  name = "External LB"
  internal = false
  load_balancer_type = "application"
  security_groups = [aws_security_group.demosg.id]
  subnets = [aws_subnet.public-subnet-1.id, aws_subnet.public-subnet-1.id]
}

resource "aws_lb_target_group" "target-elb" {
  name = "ALB TG"
  port = 80
  protocol = "HTTP"
}

```

```

vpc_id = aws_vpc.demovpc.id
}
resource "aws_lb_target_group_attachment" "attachment" {
  target_group_arn = aws_lb_target_group.external-alb.arn
  target_id = aws_instance.demoinstance.id
  port = 80
  depends_on = [
    aws_instance.demoinstance,
  ]
}
resource "aws_lb_target_group_attachment" "attachment" {
  target_group_arn = aws_lb_target_group.external-alb.arn
  target_id = aws_instance.demoinstance1.id
  port = 80
  depends_on = [
    aws_instance.demoinstance1,
  ]
}
resource "aws_lb_listener" "external-elb" {
  load_balancer_arn = aws_lb.external-alb.arn
  port = "80"
  protocol = "HTTP"
  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.external-alb.arn
  }
}
}

```

```

# Creating External LoadBalancer
resource "aws_lb" "external-alb" {
  name = "External-LB"
  internal = false
  load_balancer_type = "application"
  security_groups = [aws_security_group.demosg.id]
  subnets = [aws_subnet.public-subnet-1.id, aws_subnet.public-subnet-2.id]
}
resource "aws_lb_target_group" "target-elb" {
  name = "ALB-TG"
  port = 80
  protocol = "HTTP"
  vpc_id = aws_vpc.demovpc.id
}
resource "aws_lb_target_group_attachment" "attachment" {
  target_group_arn = aws_lb_target_group.target-elb.arn
  target_id = aws_instance.demoinstance.id
  port = 80
  depends_on = [
    aws_instance.demoinstance,
  ]
}
resource "aws_lb_target_group_attachment" "attachment1" {
  target_group_arn = aws_lb_target_group.target-elb.arn
  target_id = aws_instance.demoinstance1.id
  port = 80
  depends_on = [
    aws_instance.demoinstance1,
  ]
}
resource "aws_lb_listener" "external-elb" {
  load_balancer_arn = aws_lb.external-alb.arn
  port = "80"
  protocol = "HTTP"
  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.target-elb.arn
  }
}
}

```

23, 54 ALL

- The above load balancer is of type external
- Load balancer type is set to application
- The `aws_lb_target_group_attachment` resource will attach our instances to the Target Group.
- The load balancer will listen requests on port 80

Step 9: Create a file for the RDS instance

- Create a `rds.tf` file and add the below code to it

```
# Creating RDS Instance
resource "aws_db_subnet_group" "default" {
  name = "main"
  subnet_ids = [aws_subnet.database-subnet-1.id, aws_subnet.database-
subnet-1.id]
  tags = {
    Name = "My DB subnet group"
  }
}

resource "aws_db_instance" "default" {
  allocated_storage = 10
  db_subnet_group_name = aws_db_subnet_group.default.id
  engine = "mysql"
  engine_version = "8.0.20"
  instance_class = "db.t2.micro"
  multi_az = true
  name = "mydb"
  username = "username"
  password = "password"
  skip_final_snapshot = true
  vpc_security_group_ids = [aws_security_group.database-sg.id]
}
```

```
ec2-user@ip-172-31-10-33:~/1 x Windows PowerShell
resource "aws_db_subnet_group" "default" {
  name = "main"
  subnet_ids = [aws_subnet.database-subnet-1.id, aws_subnet.database-subnet-2.id]
  tags = {
    Name = "My DB subnet group"
  }
}

resource "aws_db_instance" "jayesh" {
  allocated_storage = 10
  db_subnet_group_name = aws_db_subnet_group.default.id
  engine = "mysql"
  engine_version = "8.0"
  instance_class = "db.t3.micro"
  multi_az = true
  db_name = "mydb"
  username = "username"
  password = "password"
  skip_final_snapshot = true
  vpc_security_group_ids = [aws_security_group.database-sg.id]
}

~
12,21 All
```

- In the above code, we need to change the value of username & password
- multi-az is set to true for the high availability

Step 10: Create a file for outputs

- Create outputs.tf file and add the below code to it

```
# Getting the DNS of load balancer
output "lb_dns_name" {
  description = "The DNS name of the load balancer"
  value = "${aws_lb.external-alb.dns_name}"
}
```

```
ec2-user@ip-172-31-10-33:~/1 x Windows PowerShell
# Getting the DNS of load balancer
output "lb_dns_name" {
  description = "The DNS name of the load balancer"
  value = "${aws_lb.external-alb.dns_name}"
}

~
~
```

- From the above code, we will get the DNS of the application load balancer.

Step 11: Create a file for variable

- Create vars.tf file and add the below code to it

```
# Defining CIDR Block for VPC
variable "vpc_cidr" {
  default = "10.0.0.0/16"
}
# Defining CIDR Block for 1st Subnet
variable "subnet_cidr" {
  default = "10.0.1.0/24"
}
# Defining CIDR Block for 2nd Subnet
variable "subnet1_cidr" {
  default = "10.0.2.0/24"
}
# Defining CIDR Block for 3rd Subnet
variable "subnet2_cidr" {
  default = "10.0.3.0/24"
}
# Defining CIDR Block for 3rd Subnet
variable "subnet2_cidr" {
  default = "10.0.4.0/24"
}
# Defining CIDR Block for 3rd Subnet
variable "subnet2_cidr" {
  default = "10.0.5.0/24"
}
# Defining CIDR Block for 3rd Subnet
variable "subnet2_cidr" {
  default = "10.0.6.0/24"
}
```

```
ec2-user@ip-172-31-10-33:~/ / Windows PowerShell
# Defining CIDR Block for VPC
variable "vpc_cidr" {
default = "10.0.0.0/16"
}
# Defining CIDR Block for 1st Subnet
variable "subnet_cidr" {
default = "10.0.1.0/24"
}
# Defining CIDR Block for 2nd Subnet
variable "subnet1_cidr" {
default = "10.0.2.0/24"
}
# Defining CIDR Block for 3rd Subnet
variable "subnet2_cidr" {
default = "10.0.3.0/24"
}
# Defining CIDR Block for 4th Subnet
variable "subnet3_cidr" {
default = "10.0.4.0/24"
}
# Defining CIDR Block for 5th Subnet
variable "subnet4_cidr" {
default = "10.0.5.0/24"
}
# Defining CIDR Block for 6th Subnet
variable "subnet5_cidr" {
default = "10.0.6.0/24"
}
}
```

17,29 All

Step 12: Create a file for user data

- Create data.sh file and add the below code to it

```
#!/bin/bash
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
echo "Hello World from $(hostname -f)" > /var/www/html/index.html
```

```
ec2-user@ip-172-31-10-33:~/ / Windows PowerShell
#!/bin/bash
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
echo "Hello World from $(hostname -f)" > /var/www/html/index.html
~
~
~
~
~
~
~
```

- The above code will install an Apache webserver in the EC2 instances

- After creating all the required files, you have to initialize them using below command
 - terraform init

```
ec2-user@ip-172-31-10-33 ~]$ ls
Terraform
[ec2-user@ip-172-31-10-33 ~]$ cd Terraform/
[ec2-user@ip-172-31-10-33 Terraform]$ ls
alb.tf          data.sh  igw.tf  outputs.tf  route_table_public.tf  terraform.tfstate  vars.tf
web_sg.tf
database_sg.tf  ec2.tf  main.tf  rds.tf      subnet.tf              terraform.tfstate.backup  vpc.tf
[ec2-user@ip-172-31-10-33 Terraform]$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.48.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
[ec2-user@ip-172-31-10-33 Terraform]$
```

- ❖ terraform init is to initialize the working directory and downloading plugins of the provider

- Then use next command to create an execution plan for our code
 - terraform plan

```
[ec2-user@ip-172-31-10-33 Terraform]$ terraform plan
aws_vpc.demovpc: Refreshing state... [id=vpc-0bbb5edc8c98fc022]
aws_subnet.application-subnet-2: Refreshing state... [id=subnet-063a09132d25bc5bf]
aws_subnet.database-subnet-1: Refreshing state... [id=subnet-03a55e99a2971aa4f]
aws_subnet.application-subnet-1: Refreshing state... [id=subnet-0cd7adea8fee0ae5f]
aws_subnet.public-subnet-1: Refreshing state... [id=subnet-056e8972b4e5fd4e3]
aws_lb_target_group.target-elb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05]
aws_subnet.database-subnet-2: Refreshing state... [id=subnet-0bf8964b38b73098a]
aws_internet_gateway.demogateway: Refreshing state... [id=igw-02df5e475c899cd71]
aws_subnet.public-subnet-2: Refreshing state... [id=subnet-077c3681554ab8050]
aws_security_group.demosg: Refreshing state... [id=sg-041169e5214a5f913]
aws_route_table.route: Refreshing state... [id=rtb-011740f0a6aff8dca]
aws_db_subnet_group.default: Refreshing state... [id=main]
aws_lb.external-alb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:loadbalancer/app/External-LB/946ea92e098c63b5]
aws_instance.demoinstance1: Refreshing state... [id=i-0e60e5c6ccc7bfff32]
aws_security_group.database-sg: Refreshing state... [id=sg-052883fba8140cc15]
aws_instance.demoinstance: Refreshing state... [id=i-0351f8487aab189f4]
aws_route_table_association.rtl: Refreshing state... [id=rtbassoc-0903c47f1b725a433]
aws_route_table_association.rt2: Refreshing state... [id=rtbassoc-0d0a38a9ece459c21]
aws_db_instance.jayesh: Refreshing state... [id=terraform-20221230084347024500000002]
aws_lb_listener.external-elb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:listener/app/External-LB/946ea92e098c63b5/70a4aa2354848439]
aws_lb_target_group_attachment.attachment: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05-20221230084406492600000005]
aws_lb_target_group_attachment.attachment1: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05-20221230084406542700000006]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so
no changes are needed.
[ec2-user@ip-172-31-10-33 Terraform]$
```

❖ terraform plan is to create the execution plan for our code

- Then for creating our infrastructure we have to use below command
- Terraform apply

```
ec2-user@ip-172-31-10-33:~$ terraform apply
Terraform has compared your real infrastructure against your configuration and found no differences, so
no changes are needed.
[ec2-user@ip-172-31-10-33 Terraform]$ terraform apply
aws_vpc.demovpc: Refreshing state... [id=vpc-0bbb5edc8c98fc022]
aws_subnet.public-subnet-1: Refreshing state... [id=subnet-056e8972b4e5fd4e3]
aws_subnet.public-subnet-2: Refreshing state... [id=subnet-077c3681554ab8050]
aws_subnet.database-subnet-1: Refreshing state... [id=subnet-03a55e99a2971aa4f]
aws_subnet.database-subnet-2: Refreshing state... [id=subnet-0bf8964b38b73098a]
aws_subnet.application-subnet-2: Refreshing state... [id=subnet-063a09132d25bc5bf]
aws_security_group.demosg: Refreshing state... [id=sg-041169e5214a5f913]
aws_lb_target_group.target-elb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05]
aws_internet_gateway.demogateway: Refreshing state... [id=igw-02df5e475c899cd71]
aws_subnet.application-subnet-1: Refreshing state... [id=subnet-0cd7adea8fee0ae5f]
aws_route_table.route: Refreshing state... [id=rtb-011740f9a6aff8dca]
aws_db_subnet_group.default: Refreshing state... [id=main]
aws_security_group.database-sg: Refreshing state... [id=sg-052883fba8140cc15]
aws_lb.external-alb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:loadbalancer/app/External-LB/946ea92e098c63b5]
aws_instance.demoinstance: Refreshing state... [id=i-0a60e5c6ccc7b4f32]
aws_instance.demoinstance: Refreshing state... [id=i-0351f8487aab1894]
aws_route_table_association.rt2: Refreshing state... [id=rtbassoc-0d0a38a9ece459c21]
aws_route_table_association.rt1: Refreshing state... [id=rtbassoc-0903c47f1b725a433]
aws_db_instance.jayesh: Refreshing state... [id=terraform-20221230084347024500000000]
aws_lb_listener.external-elb: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:listener/app/External-LB/946ea92e098c63b5/70a4aa2354048439]
aws_lb_target_group_attachment.attachment: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05-20221230084406492600000005]
aws_lb_target_group_attachment.attachment1: Refreshing state... [id=arn:aws:elasticloadbalancing:ap-south-2:420321112085:targetgroup/ALB-TG/db644e1121c01e05-20221230084406492600000006]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:
lb_dns_name = "External-LB-173152265.ap-south-2.elb.amazonaws.com"
[ec2-user@ip-172-31-10-33 Terraform]$
```

❖ terraform apply is to create the actual infrastructure. It will ask you to provide the Access Key and Secret Key in order to create the infrastructure. So, instead of hardcoding the Access Key and Secret Key, it is better to apply at the run time.

Step 13: Verify the resources

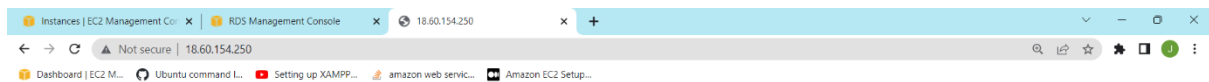
- Terraform will create below resources
 - ✓ VPC
 - ✓ Public & Private Subnets
 - ✓ Route Tables
 - ✓ Internet Gateway
 - ✓ EC2 instances
 - ✓ RDS instance
 - ✓ Application Load Balancer
 - ✓ Security Groups for Web & RDS instances

Once the resource creation finishes you can get the DNS of a load balancer and paste it into the browser and you can see load balancer will send the request to two instances.

That's it now, we have done the project how to create various resources in AWS using Terraform.

❖ Output:

- Then copy the public domain of our instance and paste it in a browser and we will get our output.



Hello World from ip-10-0-1-38.ap-south-2.compute.internal



- To delete all the created infrastructure, enter the below command:
 - terraform destroy