DEPLOY THREE-TIER ARCHITECHTURE IN AWS BY USING TERRAFORM

SHAIK.GOUSERABBANI

Method-1

- First open the AWS account by using credentials and go with EC2 instances.
- Now create or launch the server by selecting AMI's and key-pairs and also give the port ranges like SSH(22) and HTTP(80).
- Now connect the launched server through GIT-Bash terminal.

- Now install the terraform by using commands those are available in Google browse it by the name <Terraform download>.

- Now create vpc by using yaml scripting for that create a file as <vi file.tf> and write the yaml script to create vpc and save it by using “:wq”.

- #Create the vpc
- provider "aws" {
- region = "us-west-1"
- access_key = "AKIA352V5INCBUYKAMX2"
- secret_key = "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"
- }
- #Create the vpc
- resource "aws_vpc" "mvpc" {
- cidr_block = "10.0.0.0/16"
- instance_tenancy = "default"
- tags = {
- Name = "myvpc"
- }
- }

- Now change the environment into terraform mode by using command as <terraform init>.

- Now set the yaml script alignment by using command as <terraform fmt>.

- Now validate the script for correcting the spelling mistakes by using command as <terraform validate>.

- Now create the vpc by using command as <terraform apply>.

- Now go to your AWS account and open vpc services and check your vpc created or not.

- Now create the IGW and attach it to created vpc for that create a file as <vi igw.tf>

```
#Creating IGW
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.mvpc.id
  tags = {

    Name = "my-igw"
  }
}
```

- Now create the subnets by using yaml script for that use this command <vi subnets.tf>

```
#Creating pub-subnet1
resource "aws_subnet" "pub-sub-1" {
  vpc_id                  = aws_vpc.mvpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-west-1a"
  map_public_ip_on_launch = "true"
  tags = {

    Name = "pub-subnet-1"
  }

}
```

```
#Creating private subnet1
resource "aws_subnet" "pvt-sub-1" {
  vpc_id                  = aws_vpc.mvpc.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "us-west-1a"
  map_public_ip_on_launch = "false"

  tags = {
    Name = "pvt-subnet-1"
  }
}

#Creating pub-subnet2
resource "aws_subnet" "pub-sub-2" {
  vpc_id                  = aws_vpc.mvpc.id
  cidr_block              = "10.0.3.0/24"
  availability_zone       = "us-west-1b"
  map_public_ip_on_launch = "true"
  tags = {

    Name = "pub-subnet-2"
  }

}
```

```
#Creating pvt-subnet2
resource "aws_subnet" "pvt-sub-2" {
  vpc_id                  = aws_vpc.mvpc.id
  cidr_block              = "10.0.4.0/24"
  availability_zone       = "us-west-1b"
  map_public_ip_on_launch = "false"

  tags = {
    Name = "pvt-subnet-2"
  }
}
```

- Now by using single command I can create the resource that commad is <terraform apply --auto-approve>

- Now create the NAT-gateway for that create a .tf file and enter the yaml script.

```
# Create elastic ip
resource "aws_eip" "elastic" {
  vpc = true
}

# Create Nat-gateway
resource "aws_nat_gateway" "nat" {

  allocation_id     = aws_eip.elastic.id
  subnet_id         = aws_subnet.pub-sub-1.id
  connectivity_type = "public"
  tags = {
    Name = "mynat"
  }
}
```

- Now go to create the route table and associate subnets, Internet gateway and NAT gate way for that create a .tf file by using the vi mode as <vi route.tf>

```
#Create Pub-Route-table
resource "aws_route_table" "pub-route" {
  vpc_id = aws_vpc.mvpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id

  }
  tags = {
    Name = "pub-rot"
  }

}
```

```
#Create Pvt-Route-table
resource "aws_route_table" "pvt-route" {
  vpc_id = aws_vpc.mvpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.nat.id
  }

  tags = {
    Name = "pvt-rot"
  }

}

#subnets Associations
#pub-subnet-association
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.pub-sub-1.id
  route_table_id = aws_route_table.pub-route.id
}
```

```
#pvt subnet-association
resource "aws_route_table_association" "b" {
  subnet_id      = aws_subnet.pvt-sub-1.id
  route_table_id = aws_route_table.pvt-route.id

}
```

```
#subnets Associations
#pub-subnet-association
resource "aws_route_table_association" "c" {
  subnet_id      = aws_subnet.pub-sub-2.id
  route_table_id = aws_route_table.pub-route.id
}

#pvt subnet-association
resource "aws_route_table_association" "d" {
  subnet_id      = aws_subnet.pvt-sub-2.id
  route_table_id = aws_route_table.pvt-route.id

}
```

- Now create the bash script file by the data.sh

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

- Now create the public instance by using yaml scripting.

```
# Create instance
resource "aws_instance" "app" {
  ami                    = "ami-0f5e8a042c8bfcd5e"
  instance_type          = "t2.micro"
  count                  =  1
  key_name               = "terraform"
  subnet_id              = aws_subnet.pub-sub-1.id
  vpc_security_group_ids = ["${aws_security_group.sg.id}"]
  user_data              = "${file("data.sh")}"
  tags = {
    Name = "webapplication"
  }
}
```

```
# Create Security-group
resource "aws_security_group" "sg" {
  vpc_id = aws_vpc.mvpc.id

  #Inbound Rules
  # HTTP acces from anywhere
  ingress {
    from_port   = 80

    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # HTTPS access from any where
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
# SSh access from any where
  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"
  }

}
```

- Now create the public instance by using yaml scripting.

```
# Create instance
resource "aws_instance" "app2" {
  ami                    = "ami-0f5e8a042c8bfcd5e"
  instance_type          = "t2.micro"
  count                  =  1
  key_name               = "terraform"
  subnet_id              = aws_subnet.pub-sub-2.id
  vpc_security_group_ids = ["${aws_security_group.sg.id}"]
  user_data              = "${file("data.sh")}"
  tags = {
    Name = "webapplication"
  }
}

# Create Security-group
resource "aws_security_group" "sg2" {
  vpc_id = aws_vpc.mvpc.id
```

```
#Inbound Rules
 # HTTP acces from anywhere
 ingress {
   from_port   = 80
   to_port     = 80
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
 # HTTPS access from any where
 ingress {
   from_port   = 443
   to_port     = 443
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
```

```
# SSh access from any where
  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-sg2"
  }

}
```

- Now create the pvt instance by using the yaml script of pvt instance .

```
# Create instance
resource "aws_instance" "web-app" {
  ami                    = "ami-0f5e8a042c8bfcd5e"
  instance_type          = "t2.micro"
  count                  =  1
  key_name               = "terraform"
  subnet_id              = aws_subnet.pvt-sub-1.id
  vpc_security_group_ids = ["${aws_security_group.demo.id}"]
  tags = {
    Name = "web-pvt"
  }
}

# Create Security-group
resource "aws_security_group" "demo" {
  vpc_id = aws_vpc.mvpc.id
```

```
#Inbound Rules
 # HTTP acces from anywhere
 ingress {
   from_port   = 80
   to_port     = 80
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
 # HTTPS access from any where
 ingress {
   from_port   = 443
   to_port     = 443
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
```

```
  # SSh access from any where
  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"
  }

}
```

- Now create the RDS(relational database) service by using yaml scripting.

```
# Create subnet groups
resource "aws_db_subnet_group" "db-sub" {
subnet_ids = [aws_subnet.pvt-sub-1.id, aws_subnet.pvt-sub-2.id]
tags = {
        Name = "dbsubnet"
        }
}
# Create the data-base
resource "aws_db_instance" "data" {
 allocated_storage = "10"
        db_subnet_group_name = aws_db_subnet_group.db-sub.id
        db_name = "mydata"
        engine = "mysql"
        engine_version = "8.0.28"
    instance_class = "db.t2.micro"
    multi_az = "true"
    username = "admin"
    password = "admin1230"
    vpc_security_group_ids = ["${aws_security_group.db-sg.id}"]
    skip_final_snapshot = "true"

}
```

```
# Create the security groups
resource "aws_security_group" "db-sg" {
        vpc_id = aws_vpc.mvpc.id

  ingress {
  from_port = 3306
  to_port = 3306
  protocol = "tcp"
  security_groups = [aws_security_group.demo.id]
  }

    egress {
    from_port = 32768
    to_port = 65535
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]

  }
  tags = {
  Name = "database-sg"
  }
}
```

- Now create the load-balancer by using yaml script.

```
# Create load_balancer
resource "aws_lb" "application" {
  internal           = false
  name               = "APPLI-LB"
  load_balancer_type = "application"
  security_groups    = [aws_security_group.sg.id]
  subnets            = [aws_subnet.pub-sub-1.id, aws_subnet.pub-sub-2.id]
}

resource "aws_lb_target_group" "tar" {
  name     = "target"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.mvpc.id
  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    interval            = 30
    protocol            = "HTTP"
    port                = 80
    path                = "/ping"
    matcher             = 200
  }
}
```

```
resource "aws_lb_target_group_attachment" "att" {
  target_group_arn = aws_lb_target_group.tar.arn
  target_id        = aws_instance.app.id
  port             = 80
  depends_on       = [aws_instance.app,]
}
resource "aws_lb_target_group_attachment" "att2" {
  target_group_arn = aws_lb_target_group.tar.arn
  target_id        = aws_instance.app2.id
  port             = 80
  depends_on       = [aws_instance.app2,]
}
```

```
resource "aws_lb_listener" "lb-lis" {
  load_balancer_arn = aws_lb.application.arn
  port              = 80
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.tar.arn
  }
}
```

```
#getting the DNS of load-balancer
output "lb_dns_name" {
  description = "the name of the loadbalancer"
  value = "${aws_lb.application.dns_name}"
}
```

- Now go to EC2 instances and select the public ip(ipv4) and browse it on Google.

METHOD-2

- First open the AWS account by using credentials and go with EC2 instances.

- Now create or launch the server by selecting AMI's and key-pairs and also give the port ranges like SSH(22) and HTTP(80).

- Now connect the launched server through GIT-Bash terminal.

- Now install the terraform by using commands those are available in Google browse it by the name <Terraform download>.

- Now create vpc by using yaml scripting for that create a file as <vi file.tf> and write the yaml script to create vpc and save it by using ":wq".

- Now change the environment into terraform mode by using command as <terraform init>.
- Before creating complete resource you have to change the data.sh file and write a bash script to host Wordpress web application along with the instance launch.

```
#!/bin/bash
sudo yum -y update
sudo yum -y install git docker
sudo systemctl start docker
sudo systemctl enable docker
sudo chmod 666 /var/run/docker.sock
sudo curl -L https://github.com/docker/compose/releases/download/1.25.4/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
git clone https://github.com/GOUSERABBANI44/wordpress.git
cd wordpress
docker-compose up -d
```

- Now go to EC2 instances and select the public ip(ipv4) and browse it on Google.

- Now copy the DNS of load-balancer browse in Google and see the result .

THANK YOU