AWS Assignment

Done By:
Lana Alnimreen
Supervisor:
Dr.Motasem Aldiab

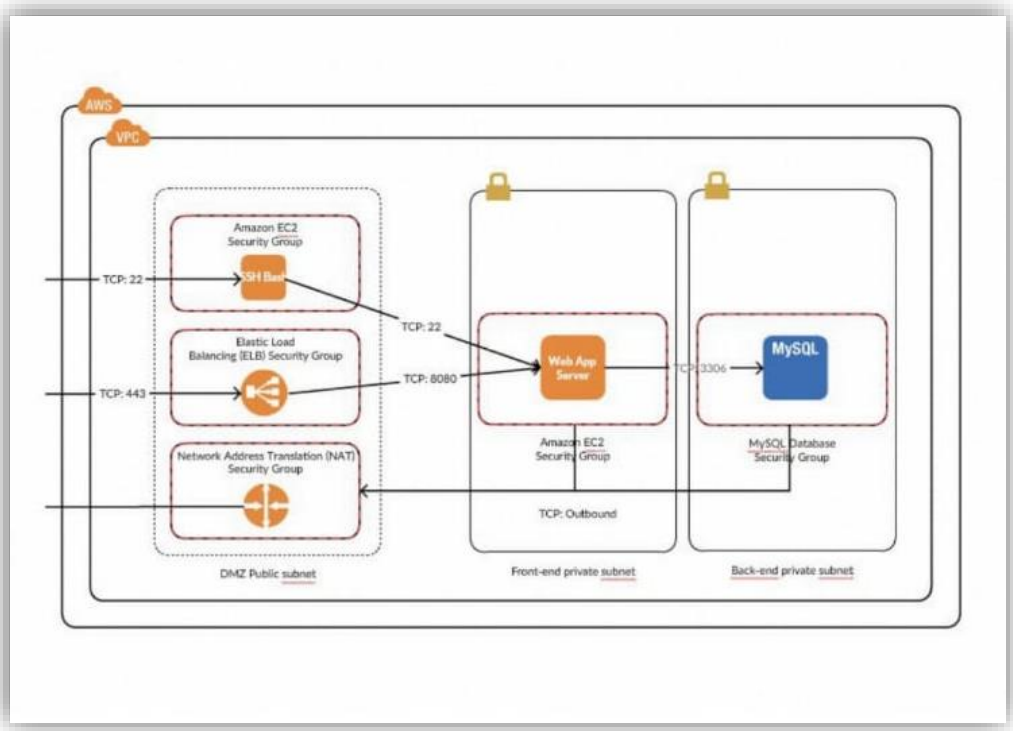
Abstract

The AWS assignment focuses on deploying and maintaining cloud infrastructure using Amazon Web Services (AWS). It includes setting up multiple AWS resources such as EC2 instances, VPCs, subnets, security groups, and load balancers with Infrastructure as Code (IaC) tools like Terraform. The task also includes creating instances with the appropriate software, testing load balancing, and ensuring secure access via key management and private networking. This assignment provides participants with hands-on experience automating cloud deployments and managing scalable, reliable, and secure applications in the AWS environment.

Contents

Introduction

Amazon Web Services (AWS) is an early adopter in cloud computing. Let's start with the example we are going to go through. They offering a wide range of services, including servers (EC2), and databases (RDS), load balancers and a lot more. In this report, I will discuss a real-world example of using AWS, but with Terraform, which is an automation tool that allows you to automate the resource creation process across all cloud service providers. Without Terraform, you would have to go to each resource page on the cloud service provider and create each resource, then remember each one and repeat the procedure to remove them. Here we have the example that we have to build in this assignment.



We have one VPC that contains three subnets, the front-end and the back-end are private. Then we have a single public subnet called DMZ, this subnet adds an extra layer of protection by isolating the database and the web application from public traffic, they can only be reached by going through the DMZ subnet and its security groups. The DMZ contains three components, the EC2 Bastion is an EC2 instance used to SSH into the front-end web application instances. The Bastion can only be accessed using a private key that is distributed to the employees with the right permission.

We have one VPC with three subnets, the front-end and back-end are private. And a single public subnet that consists of three components: the EC2 Bastion, which is an EC2 instance used to SSH into the front-end web application instances. Then we have the application load balancer, it distributes the incoming HTTP traffic on port 80 to the web application instances using the round-robin scheduling algorithm. about it later. Finally, we have the NAT gateway which allows the back-end and front-end subnets to access the internet when they need. The NAT gateway allows traffic in one direction, in this case, from the two subnets towards the internet, but not vice-versa.

The whole VPC is connected to an internet gateway, so it is able to communicate with the outer internet.

The back-end subnet has MYSQL database running. I used RDS database which is a service provided by AWS for creating and hosting databases to create database called `lana_db`. And the front-end subnet has EC2 instance running our web application then I add another EC2 instance so the load balancer distributes the traffic between them. Now let's move to the implementation.

Resources

In this section, I will go through the process of creation for each component.

- Terraform setup.
- VPC, internet gateway, and route tables.
- Subnets and route table associations.
- Security Groups
- RDS Database.
- NAT gateway.
- Load balancer.
- Bastion setup.
- Outputs.

Implementation

Terraform Setup

I am telling Terraform that we are going to use AWS as a cloud service. Also, I am specifying the AWS region used.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}
```

VPC, Internet Gateway and Route Tables

In the resource below, I am specifying the name of the VPC and the CIDR block used for it, since 16 is the mask for the VPC, then the subnets inside the VPC must have a larger mask like 24.

```
resource "aws_vpc" "main_vpc" {
  cidr_block = "10.0.0.0/16"
}
```

Then the internet gateway is created to give `main_vpc` the ability to communicate with the internet.

```
resource "aws_internet_gateway" "internet_gw" {
  vpc_id = aws_vpc.main_vpc.id
  tags = {
    Name = "Internet-Gateway"
  }
}
```

The VPC has two route tables, one is associated with the public subnet, and another is associated with the private subnets.

```
resource "aws_route_table" "public_route_table" {
  vpc_id = aws_vpc.main_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.internet_gw.id
  }

  tags = {
    Name = "Public-Route-Table"
  }
}

#private_route_table
resource "aws_route_table" "private_route_table" {
  vpc_id = aws_vpc.main_vpc.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat_gw.id
  }

  tags = {
    Name = "Private-Route-Table"
  }
}
```

Subnets and Route Table Associations

```
resource "aws_subnet" "public_subnet_a" {
  vpc_id     = aws_vpc.main_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
  tags = {
    Name = "Public-Subnet_a"
  }
}

resource "aws_subnet" "public_subnet_b" {
  vpc_id                  = aws_vpc.main_vpc.id
  cidr_block              = "10.0.4.0/24"
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true

  tags = {
    Name = "public_subnet_b"
  }
```

```
}


resource "aws_subnet" "front_end_private_subnet" {
  vpc_id     = aws_vpc.main_vpc.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-1a"
  tags = {
    Name = "Frontend-Private-Subnet"
  }
}

resource "aws_subnet" "back_end_private_subnet_a" {
  vpc_id     = aws_vpc.main_vpc.id
  availability_zone = "us-east-1a"
  cidr_block = "10.0.6.0/24"
  tags = {
    Name = "Backend-Private-Subnet_a"
  }
}


resource "aws_subnet" "back_end_private_subnet_b" {
  vpc_id     = aws_vpc.main_vpc.id
  availability_zone = "us-east-1b"
  cidr_block = "10.0.5.0/24"
  tags = {
    Name = "Backend-Private-Subnet_b"
  }
}
```

We have public_subnet_b that located in another availability zone. This subnet is created because the load balancer requires two to have two subnets in different availability zones.

The front-end subnet is associated with the private route table that is connected to the NAT gateway.

Then we have the backend subnet, exactly like the load balancer situation, RDS requires to have two subnets each one in a different availability zone. And also, they have associated with the private route table that is connected to the NAT gateway.

```
resource "aws_route_table_association" "public_route_table_assoc_a" {
  subnet_id      = aws_subnet.public_subnet_a.id
  route_table_id = aws_route_table.public_route_table.id
}

resource "aws_route_table_association" "public_route_table_assoc_b" {
  subnet_id      = aws_subnet.public_subnet_b.id
  route_table_id = aws_route_table.public_route_table.id
}

resource "aws_route_table_association" "frontend_private_route_table_assoc" {
  subnet_id      = aws_subnet.front_end_private_subnet.id
  route_table_id = aws_route_table.private_route_table.id
}

resource "aws_route_table_association" "backend_private_route_table_assoc_a" {
  subnet_id      = aws_subnet.back_end_private_subnet_a.id
  route_table_id = aws_route_table.private_route_table.id
}
```

```
resource "aws_route_table_association" "backend_private_route_table_assoc_b" {
  subnet_id      = aws_subnet.back_end_private_subnet_b.id
  route_table_id = aws_route_table.private_route_table.id
}
```

Security Groups

The first group allows the SSH traffic from anywhere. This group is used for the EC2 instance that resides inside the public subnet, also it is used for the EC2 instances inside the front-end subnet. The second group allows traffic from anywhere through HTTP port 80. The group is used for the load balancer to accept any HTTP traffic.

The next group allows SQL traffic (port 3306) from anywhere. This group is used for the RDS MYSQL instance inside the back-end subnet. Then we have the group that allows the traffic through port 8080. The group is used for the EC2 instances that reside inside the front-end subnet. Finally, the group that allows all egress traffic through all ports. This group is used to allow private instances to access anything from the internet.

RDS Database

I created a subnet group to run the database inside. The group must contain two subnets at least, each one must reside in a different availability zone. The database storage is 20 GB, it runs MYSQL.

```
resource "aws_db_subnet_group" "db_subnet_group" {
  name       = "db_subnet_group"
  subnet_ids = [
    aws_subnet.back_end_private_subnet_a.id,
    aws_subnet.back_end_private_subnet_b.id,
  ]

  tags = {
    Name = "DB-Subnet-Group"
  }
}

resource "aws_db_instance" "mysql_db" {
  allocated_storage    = 20
  storage_type            = "gp2"
  engine               = "mysql"
  engine_version       = "8.0.33"
  instance_class       = "db.t3.micro"
  username             = "admin"
  password             = "password"
  parameter_group_name = "default.mysql8.0"
  skip_final_snapshot  = true
  vpc_security_group_ids = [aws_security_group.allow-sql.id, aws_security_group.allow-
all-outbound.id,aws_security_group.allow-sql-from-ec2.id]
  db_subnet_group_name = aws_db_subnet_group.db_subnet_group.id
  db_name = "lana_db"

  tags = {
    Name = "MySQL-DB"
  }
}
```

NAT Gateway

It is needed components in private subnets can reach out for the internet, without letting the internet reach the components, it works only in one direction. I put it inside the public subnet because this subnet is linked to the internet gateway.

```
resource "aws_eip" "nat_eip" {
  vpc = true
}

resource "aws_nat_gateway" "nat_gw" {
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = aws_subnet.public_subnet_a.id

  tags = {
    Name = "NAT-Gateway"
  }
}
```

Load Balancer

First, I create A target group that tells the load balancer where to send the requests, then we have the load balancer resource itself. It depends on the EC2 instances, so the instance must run first to get the load balancer working correctly. Finally, we have a listener which is required for the load balancer to receive traffic on HTTP port 80 before forwarding it to the instances.

```
resource "aws_lb_target_group" "tg1" {
  name     = "tg"
  port     = 80
  protocol = "HTTP"
  target_type = "instance"
  vpc_id   = aws_vpc.main_vpc.id

  health_check {
    enabled             = true
    interval            = 10
    path                = "/health"
    port                = "traffic-port"
    protocol            = "HTTP"
    timeout             = 6
    healthy_threshold   = 2
    unhealthy_threshold = 2
  }

  tags = {
    Name = "App-Target-Group"
  }
}
resource "aws_lb" "app_lb" {
  name               = "app-lb"
  depends_on         = [aws_instance.web-server]
```

```
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.allow-http.id, aws_security_group.allow-
all-outbound.id]
  subnets            = [
    aws_subnet.public_subnet_a.id,
    aws_subnet.public_subnet_b.id
  ]

  tags = {
    Name = "App-Load-Balancer"
  }
}

resource "aws_lb_listener" "listener" {
  load_balancer_arn = aws_lb.app_lb.arn
  port              = 80
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.tg1.arn
  }
}
```

Bastion Setup

Instances that stand as a firewall preventing any traffic from reaching our resources. We can SSH throw the bastion, then through it we can SSH into the front-end instances.

```
resource "aws_instance" "bastion-server" {
  depends_on                  = [aws_key_pair.ec2-key-pair]
  subnet_id                   = aws_subnet.public_subnet_a.id
  ami                         = "ami-014d544cfef21b42d"
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = aws_key_pair.ec2-key-pair.key_name
  vpc_security_group_ids      = [aws_security_group.allow-ssh.id,
aws_security_group.allow-all-outbound.id]

  tags = {
    Name    = "Bastion Server"
  }
}
```

Testing

First, we run “terraform init” in the same directory where terraform file locate. Then we run “terraform plane” and “terraform apply” and enter yes to create all resources.

And here, we can see how the load balancer distributes the requests on all instances.

Conclusion

I can tell how Terraform simplifies work by automating the process of generating cloud computing resources. A few years ago, I used AWS to do some homework for a university course. After finishing, I forgot to destroy a resource and I was charged a large amount for it. But using Terraform you may delete all resources with a single command from the terminal.