BATCH :

LESSON : TERRAFORM

DATE :

SUBJECT : DEMO PROJECT 1

techproeducation

techproeducation

techproeducation

techproeducation

techproedu

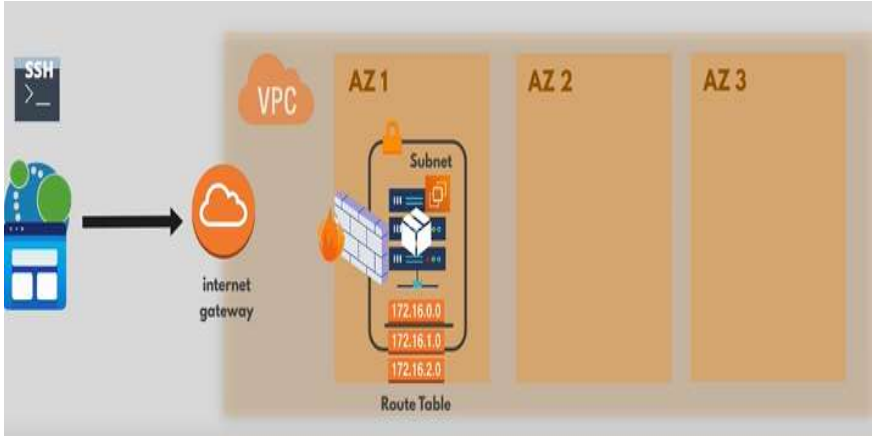
techproeducation.com info@techproeducation.com +1 (917) 768-7466

DEMO PROJECT 1

EC2 Server'ı Oluşturma

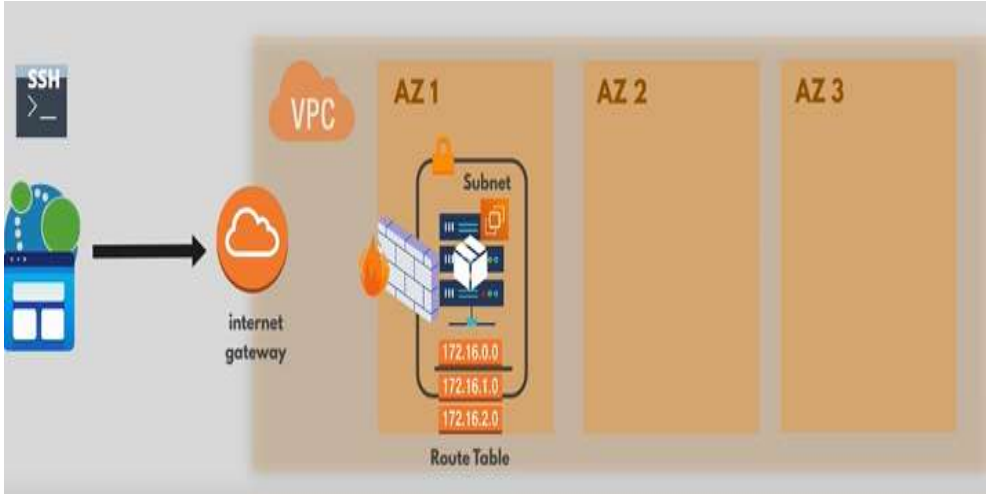
PREVIEW

- AWS e EC2 instance I deploy edeceğiz. İçinde de ngix Docker container çalıştıracağız. Ama öncelikle AWS ten infrastructure inşaa edeceğiz. Buna da:git
- - Custom bir VPC oluşturarak -1-
- - VPC nin içine custom bir subnet(birden fazla da inşaa edebiliriz) inşa ederek -2-
- - VPC leri de route table oluşturarak ve internet gateway kullanarak birbirlerine bağlayacağız -3-



PREVIEW

- VPC lerin içine de EC2 Instance ları inşaa edeceğiz.-5-
- - EC2 Instance ları da nginx Docker container larını çalıştıracak.
- - İnşa ettiğimiz şeyleri browser üzerinden erişeceğimizden dolayı bir de Firewall oluşturacağız. Server üzerinden ayrıca SSH portu da açacağız. Bu portlar için de bir Security Group u oluşturacağız. -4-
-



DEMO PROJECT 1

- Best Practice:
  - Infrastructure u temelden oluştur
  - AWS tarafından oluşturulmuş default değerleri olduğu gibi bırak

DEMO PROJECT 1

- **Custom VPC yi oluşturma ve 2. VPC nin içine custom bir subnet inşa etme**

```
variable vpc_cidr_block{}
variable subnet_cidr_block{}
variable avail_zone {}
variable env_prefix {}

resource "aws_vpc" "myapp-vpc" {
    cidr_block = var.vpc_cidr_block
    tags = {

 # Her bir component için deploy edildikleri enviromentları ön ad olarak verelim.

        Name: "${var.env_prefix}-vpc" # string interpolatation
    }
}

resource "aws_subnet" "myapp-subnet-1" {
    vpc_id = aws_vpc.myapp-vpc.id
    cidr_block = var.subnet_cidr_block
    availability_zone = var.avail_zone
    tags = {
        Name: "${var.env_prefix}-subnet-1" # subnet-1 post-suffix arkadaşlar
    }
}
```

DEMO PROJECT 1

- **Custom VPC yi oluşturma ve 2. VPC nin içine custom bir subnet(birden fazla da inşaa edebiliriz) inşa etme**

```
variable vpc_cidr_block{}
variable subnet_cidr_block{}
variable avail_zone {}
variable env_prefix {}

resource "aws_vpc" "myapp-vpc" {
    cidr_block = var.vpc_cidr_block
    tags = {

 # Her bir component için deploy edildikleri enviromentları ön ad olarak verelim.

        Name: "${var.env_prefix}-vpc" # string interpolatation
    }
}

resource "aws_subnet" "myapp-subnet-1" {
    vpc_id = aws_vpc.myapp-vpc.id
    cidr_block = var.subnet_cidr_block
    availability_zone = var.avail_zone
    tags = {
        Name: "${var.env_prefix}-subnet-1" # subnet-1 post-suffix arkadaşlar
    }
}
```

DEMO PROJECT 1

- terraform.tfvars ı da güncelledik:

```
vpc_cidr_block = "10.0.0.0/16"
subnet_cidr_block = "10.0.10.0/24"
avail_zone = "us-east-1b"
env_prefix = "dev"
```

DEMO PROJECT 1

- **VPC leri de route table oluşturarak ve internet gateway kullanarak birbirlerine baglama**
- Bu aşamada route table ı terraform da oluşturacağız. Best Practice: Default componentları—bu componentlardan biri de rout table dır--kullanmaktansa yenilerini oluşturmak.

```
# Route Table Oluşturma
resource "aws_route_table" "myapp-route-table" {
    vpc_id = aws_vpc.myapp-vpc.id # id mizi tanımladık
```

DEMO PROJECT 1

- VPC leri de route table oluşturarak ve internet gateway kullanarak birbirlerine baglama

```
# Route Table Oluşturma
resource "aws_route_table" "myapp-route-table" {
    vpc_id = aws_vpc.myapp-vpc.id # id mizi tanımladık

    route {
        # Bunun içine de route table daki entry leri tanımlayacağız
        # 10.0.0.0/16 için route oluşturmamıza gerek yok çünkü zaten default tanımlanmış
        cidr_block = "0.0.0.0/0" # birinci attribute umuz

        # route table ımız için internet gateway id
        # default değerimizde yok bu yüzden oluşturmamız gerekecek
        # alttaki resource ile oluşturalım
        gateway_id = aws_internet_gateway.myapp-igw.id
    }
    tags = {

        # Bütün component ları tag edelim ki bizim tarafımızdan oluşturulduklarını daha rahat anlayalım
        Name: "${var.env_prefix}-rtb"
    }
}

# Internet gateway Oluşturma
resource "aws_internet_gateway" "myapp-igw" {
    # custom için internet gateway oluşturuyoruz şuan
    # Bunu sanal bir modem olarak düşünebiliriz; bizi internete bağlıyor
    vpc_id = aws_vpc.myapp-vpc.id
    tags = {
        Name: "${var.env_prefix}-igw"
    }
}
```

DEMO PROJECT 1

Subnet Association with Route Table

```
# Subnet Associations Oluşturalım
resource "aws_route_table_association" "a-rtb-subnet" {
    subnet_id = aws_subnet.myapp-subnet-1.id
    route_table_id = aws_route_table.myapp-route-table.id
}
```

DEMO PROJECT 1

- **DEFAUT ROUTE TABLE'I MODIFY ETME**
- Peki kendi route table ımızı kullanmaktansa default olanla nasıl devam edebiliriz?-- modify ederek-- Şöyle:

```
resource "aws_default_route_table" "main-rtb" {
    # cmd ye terraform state show aws_vpc.myapp-vpc yazmamızdan sonraki kısım
    default_route_table_id = aws_vpc.myapp-vpc.default_route_table_id

    # route ve tags kısmını myapp-route table dan yapıştırdık.
    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.myapp-igw.id
    }
    tags = {
        Name: "${var.env_prefix}-main-rtb"
    }
}
```

DEMO PROJECT 1

- **İnşa ettiğimiz şeyleri browser üzerinden erişeceğimizden dolayı bir de Firewall(8080) oluşturacağız. Server üzerinden ayrıca SSH portu(22) da açacağız. Bu portlar için de bir Security Group u oluşturacağız.**

- my_ip isimli bir variable tanımlayalım:

```
variable my_ip{}
```

- terraform.tfvars ın içine my_ip nin değerini set edelim: Comment deki faydalarının yanında ip yi terraform.tfvars ın içine tanımlamanın bir diğer yararı GitHub daki repomuzda ip adresimizi görünür kılmamamız.

```
my_ip = "141.255.5.197/32" # ip adresini whatsmip.org dan aldık.
# Birden fazla veya dynamic ip adresin varsa değişken tanımlayabilirsin üste.
```

DEMO PROJECT 1

- **İnşa ettiğimiz şeyleri browser üzerinden erişeceğimizden dolayı bir de Firewall(8080) oluşturacağız. Server üzerinden ayrıca SSH portu(22) da açacağız. Bu portlar için de bir Security Group u oluşturacağız.**

- my_ip isimli bir variable tanımlayalım:

```
variable my_ip{}
```

- terraform.tfvars ın içine my_ip nin değerini set edelim: Comment deki faydalarının yanında ip yi terraform.tfvars ın içine tanımlamanın bir diğer yararı GitHub daki repomuzda ip adresimizi görünür kılmamamız.

```
my_ip = "141.255.5.197/32" # ip adresini whatsmip.org dan aldık.
# Birden fazla veya dynamic ip adresin varsa değişken tanımlayabilirsin üste.
```

DEMO PROJECT 1

- **Firewall oluşturalım. Browser üzerinden erişim için port 8080, SSH için de port 22 yi oluşturalım.**

```
# SECURITY
resource "aws_security_group" "myapp-sg" {
    name = "myapp-sg"
    vpc_id = aws_vpc.myapp-vpc.id

    # Firewall Kuralları
    # a) Incoming Traffic Rules: ssh into EC2 & access from browser
    # 1. SSH
    ingress{ # ingress for incoming
        from_port = 22 # ikisi de 22 çünkü aralıktansa tek bir port istiyoruz
        to_port = 22 # yani burada diyoruz ki port 22 den port 22 ye
        protocol = "tcp"
        #hangi ip adreslerini 22 numaralı porta erişmeye yetkili onu tanımlayalım
        cidr_blocks = [var.my_ip]
    }

    # 2. Firewall
    ingress { # ingress for incoming
        from_port = 8080
        to_port = 8080
        protocol = "tcp"
        # hangi ip adresleri 8080 numaralı porta erişmeye yetkili onu tanımla
        cidr_blocks = ["0.0.0.0/0"] # Bunu yazarak tarayıcıdan giren bütün
        #               ip adreslerini port 8080 için erişilebilir kıldık.

    }

    # b) Outgoing Traffic Rules: installations & fetch Docker Image
    egress { # egress for exiting
        # Hiçbir şeyi kısıtlamıyoruz çünkü içerideki bütün trafiğin dışarı çıkmasını istiyoruz.
        from_port = 0
        to_port = 0
        protocol = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
    tags = {
        Name: "${var.env_prefix}-sg" # sg= security group
    }
}
```

DEMO PROJECT 1

- Default security grouplardan birini kullanmak istersek napmalıyız?

```
resource "aws_default_security_group" "default-sg" {
    vpc_id = aws_vpc.myapp-vpc.id
}
```

ve burayı

```
tags = {
    Name: "${var.env_prefix}-default-sg"
}
```

güncellemen yeterli.

DEMO PROJECT 1

- **Automate AWS Infrastructure**
- EC2 instance oluşturmamız için daha neye ihtiyacımız var? Hangi configuration ları yapmalıyız? Veya başka neler yapmalıyız? AWS EC2 instance için bir configuration oluşturalım..
- Şu web adresine gir: https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:
- AWS Instance için bir attribute tanımlamalıyız
  - buna ilgili web sitesinden seçeceğimiz image in id sini ekleyelim
- AMI değerini hardcode yapmamalıyız çünkü her yeni pakette bu değer değişiyor.
- Bunun yerine AMI değerini dinamik olarak fetch edebiliriz.

```
# AMI değerine hardcode etmeden query leyerek ulaşma
data "aws_ami" "latest-amazon-linux-image" {
    most_recent = true # most recent image version
    owners = ["amazon"] # image ın sahibi amazon olsun
    filter {# query in için kriterlerin neler burada belirleyebilirsin
        name = "name"
        values = ["amzn-ami-hvm-*-x86_64-gp2"] # başlangıcı -*- öncesi
                                               # bitişi -*- sonrası olan AMI leri query le.
    }
    filter {
        name = "virtualization-type"
        values = ["hvm"]
    }
}

#Filterlarımızın output u neymiş bakalım
output "aws_ami_id" {
    value = data.aws_ami.latest-amazon-linux-image.id
}
```

DEMO PROJECT 1

- **VPC lerin içine de EC2 Instance ları inşaa et**

- instance_type isimli bir variable oluştur:

```
variable instance_type{}
```

- terraform-dev.tfvars a da değerini gir:

```
instance_type = "t2.micro"
```

DEMO PROJECT 1

- **VPC lerin içine de EC2 Instance ları inşaa et**

```
resource "aws_instance" "myapp-server"{ # resource u aws_instance olan
                        #myapp-server isimli bir configuration oluşturduk.
# Bu noktada 2 adet instance oluşturmamız gerekiyor:
    ami = data.aws_ami.latest-amazon-linux-image.id # EC2 server'ının dayandığı image
    instance_type = var.instance_type


    # Bu kısım opsiyonel, kodu optimize etmek için yapıyoruz.
    subnet_id = aws_subnet.myapp-subnet-1.id
    availability_zone = var.avail_zone

    # Yukarıdaki 2 satırda şunu dedik:
    # server'ı subnet_id den al,
    #belirttiğimiz zone a ata


    # Bu kısım gerekli
    associate_public_ip_address = true # ip adreslerine browserdan eriş
```

DEMO PROJECT 1

- **VPC lerin içine de EC2 Instance ları inşaa et**

- AWS den EC2 ya gir. Key pairs e tıkla. Create key pairse tıkla.
- Name: server-key-pair, ppk seçili kalsın, Create key pair e tıkla.
- İndirilen dosyayı .ssh'e taşı--.ssh iniz yoksa ya da hata alıyorsanız direk directory nize de taşıyabilirsiniz--: Elle yap bunu
- Yukarıdaki kod devan ediyor..

```
    key_name = "server-key-pair"

    tags = {
      Name: "${var.env_prefix}-server"
    }
}
```

- Cmd ye terraform plan ve terraform apply –auto-approve yaz.