BATCH :

LESSON : TERRAFORM

DATE :

SUBJECT : DEMO PROJECT 1- PART 2

techproeducation

techproeducation

techproeducation

techproeducation

techproedu

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

DEMO PROJECT 1- PART 2

Ssh Key Pair'i Variable Haline Getirme

- server-key-pair gibi önemli bir dosyayı güvenlik ve dinamiklik açısından variable olarak kullanmak daha mantıklı.Bu yüzden;
- Öncelikle my_private_key variable'ını main.tf e oluşturalım:

```
variable my_private_key{}
```

- Bu değişken key pair'deki my_private_key'in konumunu verir.

DEMO PROJECT 1- PART 2

Ssh Key Pair'i Variable Haline Getirme

terraform-dev.tfvars'a my_private_key değişkenini tanımlayalım:

```
my_private_key= "server-key-pair-2"
```

DEMO PROJECT 1- PART 2

Ssh Key Pair'i Variable Haline Getirme

- Kodu şu şekilde güncelleyelim:

```
resource "aws_instance" "myapp-server"{
    ami = data.aws_ami.latest-amazon-linux-image.id
    instance_type = var.instance_type

    # Bu kısım opsiyonel, kodu optimize etmek için yapıyoruz.
    subnet_id = aws_subnet.myapp-subnet-1.id
    vpc_security_group_ids = [aws_default_security_group.default-sg.id]
    availability_zone = var.avail_zone

    # Bu kısım gerekli
    associate_public_ip_address = true

    # Aşağıda key-pairi aws_instance resource umuza bağladık.
    key_name = var.my_private_key # DEĞİŞTİ

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

DEMO PROJECT 1- PART 2

Ssh Key Pair'i Variable Haline Getirme

- Sayfadaki kodu main.tf deki resuorce "myapp-server" nin altına yazalım:

```
output "ec2_public_ip" {
  value = aws_instance.myapp-server.public_ip
}
```

DEMO PROJECT 1- PART 2

Ssh Key Pair'i Variable Haline Getirme

- EC2 server'ımız çalışır vaziyette, Networking kısmını ayarladık. Ancak server'ımızın içinde henüz hiçbir şey çalışmıyor. Docker'ı yüklemedik, docker containerlar çalışmıyor. Server'ımız boş şu an. Yapmadığımız şeyleri de automated yolla yapmamız gerekiyor. Instance'larımız hazır olur olmaz bir dizi komutla her şeyi halletmek istiyoruz. Bunu da(resource "aws_instance" "myapp-server"ın içine):

```
resource "aws_instance" "myapp-server"{
    .
    .
    .
    key_name = var.my_private_key

    user_data = <<EOF
                #!/bin/bash
                sudo yum update -y && sudo yum install -y docker
                sudo systemctl start docker
                sudo usermod -aG docker ec2-user
                docker run -p 8080:80 nginx
            EOF
    /*
    Üstteki Kısım Linux Komutu
    ilk satır: bash script
    ikinci satır: bütün paketleri ve repoları günceller && güncellemeden sonra docker ı yükle
    Üçüncü Satır: yüklenen docker ı çalıştır
    Dördüncü Satır: Docker komutlarını sudo komutu olmadan çalıştırmak istiyoruz--
    -- Bunun için de EC2 kullanıcısını docker group a eklemeliyiz
    Beşinci Satır:  Docker ı çalıştırır

    ** Bu kısım yalnızca bir kere çalışır.
    */
    ```
    tags = {
      Name: "${var.env_prefix}-server"
    }
}
```
```

DEMO PROJECT 1- PART 2

Extract to shell script

- Elimizde daha karmaşık bir scriptin olduğu durumlarda EOF kısmını main.tf'dense ayrı bir dosyada saklamalıyız. entry-script.sh i root directory'mize oluşturalım.
- user_data yı şu şekilde güncelleyelim:

```
user_data = file("entry-script.sh")
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Entry-script.sh terraform un görev alanına girmiyor. Bu yüzden burada alacağımız hatayı açıklayamaz ve bu da işlerin bizim için daha zor olmasını sağlar. Terraform dan komutları çalıştırmanın başka yolları da var. Bu yollar da terraform un kapsama alanına girmiyor ama bilmekte fayda var.
- Provisioner bunlardan birisi
- "myapp-server" ın içine tags in tam üstüne bir provisioner tanımlayalım:

```
resource "aws_instance" "myapp-server"{
    .
    .
    .
    provisioner "remote-exec" {
        # remote server a bağlanmamızı sağlayan provisioner
        inline = [ # remote server'da çalıştırılacak komutları içine tanımlarız.
            "export ENV_dev",
            "mkdir newdir"
        ]
    }
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Ne zaman provisioner "remote-exec"i oluşturacak olursak terraforma ayrıca remote server'a nasıl bağlanması gerektiğini de söylemeliyiz.

```
connection { # YENİ
# ne zaman provisioner "remote-exec"i oluşturacak olsa terraforma aynı remote server a nasıl bağlanması gerektiğini de söylemeli
    type = "ssh" # YENİ
    host = self.public_ip # YENİ
    user = "ec2-user" # YENİ
    private_key = var.my_private_key # YENİ
} # YENİ

provisioner "remote-exec" { # YENİ ...
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Uygulamada script İle devam etmek daha mantıklı bu yüzden :

```
provisioner "remote exec" {
    # remote server a bağlanmamızı sağlayan provisioner
#   inline  [ # remote server'da çalıştıracak komutları içine tanımlarız.
#       "export ENV=dev",
#       "mkdir newdir"
#   ]

# Uygulamada inline bloğundansa scriptt ile devam etmek daha mantıklı:
script = file("entry-script.sh")
}
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Bir provisioner daha oluşturacağız:

```
provisioner "file" {
    # dosyaları veya directory leri local dan eni oluşturulan resource a kopyalar.
    source = "entry-script.sh"
    destination = "entry-script.sh" # cmd ye pwd yaptığımız zaman çıkan yer
}

provisioner "local-exec" {
    # bir kaynak oluşturulduktan sonra yerel bir yürütülebilir dosyayı çağırır
    command = "echo ${self.public_ip} >output.txt" # yukarıdaki server'ın public IP adresini cmd'ye yazdırmak
}
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Bir provisioner daha oluşturacağız:

```
provisioner "file" {
    # dosyaları veya directory leri local dan eni oluşturulan resource a kopyalar.
    source = "entry-script.sh"
    destination = "entry-script.sh" # cmd ye pwd yaptığımız zaman çıkan yer
}

provisioner "local-exec" {
    # bir kaynak oluşturulduktan sonra yerel bir yürütülebilir dosyayı çağırır
    command = "echo ${self.public_ip} >output.txt" # yukarıdaki server'ın public IP adresini cmd'ye yazdırmak
}
```

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Provisionları kullanmak terraform tarafından önerilmiyor çünkü: - use user_data if avaliable - breaks idempotency concept - TF doesn’t know what you are execute - Breaks current-desired state comparison - Ayrıca dengesizdirler; bazen çalışırlar bazen çalışmazlar. Kodun bazı kısımlarını okumayabilirler.

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Onun yerine remote exec durumları için: - user_data yı kullan. - Server provision edildikten sonra Configuration Management Toolların kullan. Chef, Puppet, Ansible gibi - “local-exec” provisioner in yerine mesela HasHiCorp un “local” provider kullanabilirsin. (https://registry.terraform.io/providers/hashicorp/local/latest/docs/resources/file) - Execute scipts separate from Terraform - From CI/CD tool

DEMO PROJECT 1- PROVISIONERS

- **PROVISIONERS**
- Ama her halükarda başkasının config dosyasında mesela provisioner kullanılmış, ne yapmanız gerektiğini bilmeniz gerekiyor. O yüzden öğrenmekte fayda var.
- Uygulamada bir şey değişmedi arkadaşlar çünkü entry-script.sh imiz aynı dosya. Provisioner burada bize ne sağladı? entry-script.sh i daha yönetilebilir kıldı. Eğer entry-script.sh de bir hata olsaydı provisioner sayesinde hatayı cmd de görebilecek önlemimizi ona göre alacaktık.

DEMO PROJECT 1- MODULES

- **MODULES**

- Config dosyası oluşturduk. EC2 instance ı da tanımladık. Bunlar görece basit kodlar olmasına rağmen 100 satırdan fazla kod yazdık. Bunu da tek bir main.tf dosyası oluşturup bütün kodu onun içine yazdık. Bu kullanılabilir değil arkadaşlar, programlama mantığına da ters. Bundansa kodumuzu modüllere ayırmalı ve bu modülleri birbirine bağlamalıyız. Modülleri fonksiyonlara benzetebiliriz, ya da class lara. Bu metaforumuzda:
  - Input Variables mesela fonksiyon argümanlarına
  - Output Values da fonksiyonların return değerine benzetilebilir.

Reusable, composable, battle-tested

TERRAFORM MODULES

DEMO PROJECT 1- MODULES

- **MODULES**
- Kendi modülümüzü oluşturabiliriz, ama çoğu durum için Terraform un hali hazırda bir modülü var. Bu yüzden öncelikle bu modülleri bulmalı, incelemeli, amacımıza hizmet etmiyorlarsa kendi modülümüzü oluşturmalıyız. Terraform registy ye gidecek olursak(registry.terraform.io/browse/modules) modülleri daha yakından görebiliriz. Vpc modülüne gidelim mesela.

Reusable, composable, battle-tested

TERRAFORM MODULES

DEMO PROJECT 1- MODULES

- **MODULES**

- Oldukça güzel bir arayüz, toplu bir data. Parameterse'a erişebiliyoruz. Açıklamaları yazıyor. Çoğu durumda bunlardan birkaçını kullanırız. Kullanmadıklarımız default değerleriyle çalışır. Dependency kısmımız da var modülümüzde. Bu section da modülümüz başka providerlardan neleri refer ediyor onu görebiliriz. Vpc modülünün aws dependency si var mesela. Bu demek oluyor ki ne zaman VPC modülünü kullansak sistem arkada AWS I de import edip çalıştırır.

Reusable, composable, battle-tested

TERRAFORM MODULES

DEMO PROJECT 1- MODULES

- **Modularize Our Project**

- Main.tf I modülize etmeye başlayabiliriz.
- Yeni modülümü oluşturmadan önce main.tf I bi temizleyelim. Bu noktada 4 adet terraform dosyası oluşturacağız:
- **main.tf**
- **variables.tf** : (variable)değişkenleri buraya taşıyoruz
- **outputs.tf**: output dosyalarını buraya taşıyoruz.
- **providers.tf**: providerları buraya taşıyoruz.
- Yukarıdaki tf dosyalarını oluştur ve açıklamalardaki gibi taşı(providers hariç çünkü bir tane provider ımız var; sadece bunun için yeni bir dosya oluşturmanın anlamı yok). Terraform un iyi bir yanı bu dosyaları birbirine bağlamak için kod yazmamıza gerek olmaması. Otomatik olarak terraform bağlıyor zaten.

Reusable, composable, battle-tested

TERRAFORM MODULES

DEMO PROJECT 1- MODULES

- **Modularize Our Project**

- modules isimli bir klasörün oluşturalım. Bu klasörün içine de asıl modülleri içeren klasörler oluşturacağız. webserver ve subnet isimli klasörleri içine oluşturduk. Her bir modülün kendine ayrı **main.tf outputs.tf** ve **variables.tf** dosyaları olacak. Bu dosyaları VS Code Terminalinden oluşturacağız çünkü daha kolay. VS Terminal e:

```
cd modules
cd webserver
New-Item main.tf
New-Item variables.tf
New-Item outputs.tf
cd ../subnet
New-Item main.tf
New-Item variables.tf
New-Item outputs.tf
```

DEMO PROJECT 1- MODULES

- **Modularize Our Project**
- Ana variables.tf den subnet_cidr_block{}, avail_zone{} ve env_prefix{} variablelarını kopyala ve modules/subnet/variables.tf e yapıştır. Ayrıca vpc_id ve default_route_table_id variable larını da aynı dosyanın içine oluşturalım:

```
modules/subnet/variables.tf
variable subnet_cidr_block {}
variable avail_zone {}
variable env_prefix {}
variable vpc_id {}
variable default_route_table_id {}
```

DEMO PROJECT 1- MODULES

- **Use the Module**
- Main.tf imiz var, variablelar da variables.tf in içinde tanımlı. Peki subnet/main.tf imizi root main.tf den nasıl refer ederiz? module keyword'üyle tabiki. Alttaki kodu root main.tf in içine myapp-vpc & default-sg'nin arasına yazıyoruz.

```
module "myapp-subnet" {
    source = "./modules/subnet"
    subnet_cidr_block = var.subnet_cidr_block # modulün kaynağı için
    # main.tf den refer edeceğimiz variable'lar
    avail_zone = var.avail_zone
    env_prefix = var.env_prefix

    # tanımladığımız variable lar
    vpc_id = aws_vpc.myapp-vpc.id
    default_route_table_id = aws_vpc.myapp-vpc.default_route_table_id
}
```

DEMO PROJECT 1- MODULES

- **Module Output**
- Child module daki resource lara nasıl erişebiliriz? subnet/outputs.tf i aç.
- Output Values module un return değeri gibidir. Resource değerlerini parent module e taşır.

```
output "subnet" {
    value = aws_subnet.myapp-subnet-1
}
```

DEMO PROJECT 1- MODULES

- **Module Output**
- Child module daki resource lara nasıl erişebiliriz? subnet/outputs.tf i açalım.
- Output Values module un return değeri gibidir. Resource değerlerini parent module e taşır.

```
output "subnet" {
    value = aws_subnet.myapp-subnet-1
}
```