

Lab 4 – Le provisionneurs

Objectif

Terraform nous aide non seulement à créer et à gérer les infrastructures, mais également à les approvisionner lors de la création ou de la suppression de ressources.

Les provisionneurs sont utilisés pour exécuter des scripts ou des commandes shell sur une machine locale ou distante dans le cadre de la création/suppression de ressources. Ils sont similaires aux user-data dans une instance EC2 qui ne s'exécutent qu'une seule fois lors de la création.

1. Les provisionneurs Terraform

Il existe de nombreux provisionneurs disponibles, nous allons cependant étudier les provisionneurs Terraform les plus importants.

1.1. Le provisionneur local-exec

Le provisionneur **local-exec** appelle un exécutable local après la création d'une ressource.

Dans l'exemple ci-dessous, nous voulons écrire l'adresse IP publique de notre instance EC2 dans un fichier nommé ip_address.txt :

```
provider "aws" {
  profile      = "default"
  region      = "us-west-2"
}

resource "aws_instance" "my_ec2" {
  ami          = "ami-06ffade19910cbfc0"
  instance_type = "t2.micro"
}

provisioner "local-exec" {
  command = "echo ${aws_instance.my_ec2.public_ip} > ip_address.txt"
}
```

Nous spécifions alors un bloc **provisioner** en spécifiant le type de provisionneur à utiliser, dans pour notre exemple, nous utilisons le type local-exec.

D'autres options sont également prises en charge par ce type de provisionneur :

- **working_dir** (Facultatif) : spécifie le répertoire de travail où la commande sera exécutée.
- **interpreter** (Facultatif) : S'il est fourni, il s'agit d'une liste d'arguments d'interpréteurs utilisés pour exécuter votre commande. Le premier argument est l'interpréteur lui-même.

```
provider "aws" {
    region = "us-east-2"
    access_key = "Access Key"
    secret_key = "Secret Key"
}

resource "aws_instance" "my_ec2" {
    ami = "ami-07c1207a9d40bc3bd"
    instance_type = "t2.micro"
    tags = {
        Name = "terraform test"
    }
    provisioner "local-exec" {
        command = "echo ${aws_instance.my_ec2.public_ip} > ip_address.txt"
    }
}
```

```
[root@master-node ex1]# terraform init && terraform apply
```

```
Initializing the backend...
```

```
...
```

```
aws_instance.my_ec2: Provisioning with 'local-exec'...
```

```
aws_instance.my_ec2 (local-exec): Executing: ["/bin/sh" "-c" "echo 18.191.139.160 > ip_address.txt"]
```

```
aws_instance.my_ec2: Creation complete after 42s [id=i-03b638af97aa882cd]
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

1.2. Le provisionneur remote-exec

Paire de clé ssh

Le provisionneur **remote-exec** permet d'appeler un script sur une ressource distante une fois qu'elle est créée. Nous pouvons fournir une liste de chaînes de commandes qui sont exécutées dans l'ordre où elles sont fournies.

Dans cet exemple, nous utiliserons le mode de connexion en SSH téléchargée depuis notre compte AWS.

```
[root@master-node ex2]# ssh-keygen -t rsa
```

À ce moment-là, vous pouvez utiliser le provisionneur remote-exec, comme suit :

```
provider "aws" {
  region    = "us-west-2"
  access_key = "Access key "
  secret_key = "Secret key "
}
resource "aws_key_pair" "my_ec2" {
  key_name     = "devops.pem"
  public_key   = file("./devops.pem.pub")
}
resource "aws_security_group" "instance_sg" {
  name = "terraform-test-sg"
  egress {
    from_port        = 0
    to_port          = 0
    protocol          = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "my_ec2" {
  key_name          = aws_key_pair.my_ec2.key_name
  ami               = "ami-06ffade19910cbfc0"
  instance_type     = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance_sg.id]
  connection {
    type = "ssh"
  }
}
```

```

    user          = "ubuntu"
    private_key   = file("./devops.pem")
    host          = self.public_ip
}

provisioner "remote-exec" {
    inline = [
        "sudo apt-get -y update",
        "sudo apt-get install -y apache2",
        "sudo systemctl start apache2",
        "sudo systemctl enable apache2",
        "sudo sh -c 'echo \"<h1>Hello devopssec</h1>\" >
/var/www/html/index.html'",
    ]
}
}

```

Nous commençons par créer une ressource **aws_key_pair** requise pour les connexions SSH sur notre instance EC2 nous spécifions d'abord le nom de la paire de clés ainsi que notre clé publique créée précédemment afin d'autoriser notre machine locale à se connecter sur notre instance ec2.

Ensuite, nous imbriquons le bloc **connection** en spécifiant qu'on souhaite se connecter avec le protocole ssh sur la machine cible avec l'utilisateur **ubuntu** (utilisateur par défaut sur les AMIs Ubuntu) et notre clé privée créée précédemment.

Nous sommes également obligés de fournir l'adresse de la ressource à laquelle se connecter dans l'argument host, cependant les blocs **connection** ne peuvent pas faire référence à leurs ressources parent par leur nom. Au lieu de cela, nous utilisons l'objet **spécial self** qui représente la ressource parent de la connexion et possède tous les arguments de cette ressource. Dans notre cas nous l'utilisons pour récupérer l'adresse IP publique de notre instance EC2.

Enfin nous indiquons qu'on souhaite utiliser le provisionneur **remote-exec** en spécifiant les commandes à exécuter sur la machine distante.

Nom d'utilisateur et mot de passe

Pour utiliser un nom d'utilisateur et un mot de passe, il suffit tout simplement de les spécifier dans votre bloc de connexion, comme suit :

```

resource "aws_instance" "my_ec2" {
    ...

```

```
connection {  
    type      = "ssh"  
    user      = "ubuntu"  
    password  = "mot de passe"  
    host      = self.public_ip  
}  
  
provisioner "remote-exec" {  
    ...  
}  
}
```

2. Exercice d'application 3

A partir du code de l'exercice d'application 2, vous allez le modifier pour installer nginx sur votre VM

- Vous allez récupérer l'ip , id et la zone de disponibilité de la VM et vous les mettrez dans un fichier nommé infos_ec2.txt
- Supprimez vos ressources avec terraform destroy
- Créez un dossier « Exercice d'application 2 » pour conserver votre code

Correction de l'exercice d'application 3

Pour commencer dupliquer le répertoire travail de l'exercice d'application 2 pour y modifier en ajoutant les étapes nécessaires de l'approvisionnement.

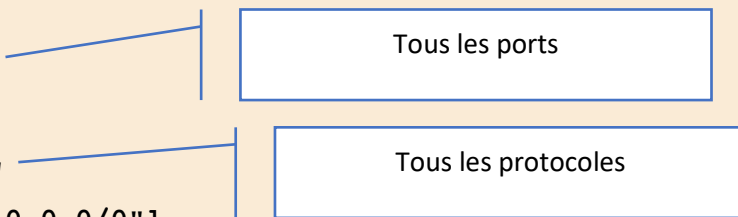
```
cp -r terraform-exapp-2/ terraform-exapp-3
```

- Ajoutez le Traffic en entrée en SSH. C'est le mécanisme utiliser pour la connexion depuis la machine local vers votre instance AWS.

```
ingress {  
  description = "Autoriser le trafic SSH"  
  from_port   = 22  
  to_port     = 22  
  protocol    = "tcp"  
  cidr_blocks = ["0.0.0.0/0"]  
}
```

- Ajoutez une règle pour autoriser l'accès sortant de votre instance, pour pouvoir télécharger et installer les paquets nécessaires du serveur nginx. La règle qui permet l'accès sortant c'est **egress**

```
egress {  
  from_port = 0  
  to_port   = 0  
  protocol  = "-1"  
  cidr_blocks = ["0.0.0.0/0"]  
}
```



- Définir le provisionneur local pour récupérer les informations exigées

```
resource "aws_instance" "myec2" {  
  ...  
  provisioner "local-exec" {  
    command = "echo PUBLIC IP: ${aws_instance.myec2.public_ip} ; ID:  
${aws_instance.myec2.id} > ip_address.txt"  
  }  
}
```

- Ajoutez à la définition de votre ressource, le provisionneur distant pour installer nginx

```
provisioner "remote-exec" {
  inline = [
    "sudo amazon-linux-extras install -y nginx1.19",
    "sudo systemctl start nginx"
  ]

  connection {
    type = "ssh"
    user = "ec2-user"
    private_key = file("./devops-msm.pem")
    host = self.public_ip
  }
}
```

Voici le déploiement complet

```
provider "aws" {
  region      = "us-east-1"
  access_key  = "AKIAR4KJP3BGWYHHVGG5"
  secret_key  = "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"
}

data "aws_ami" "app_ami" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm*"]
  }
}

resource "aws_instance" "myec2" {
  ami              = data.aws_ami.app_ami.id
  instance_type    = var.instance_type
  key_name         = "devops-msm"
  tags             = var.aws_common_tag
  security_groups  = [aws_security_group.allow_http_https.name]

  provisioner "local-exec" {
    command = "echo PUBLIC IP: ${aws_instance.myec2.public_ip} ; ID:
${aws_instance.myec2.id} ; AZ: ${aws_instance.myec2.availability_zone}; >>
infos_ec2.txt"
  }
  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras install -y nginx1.12",
      "sudo systemctl start nginx"
    ]
  }

  connection {
    type     = "ssh"
    user     = "ec2-user"
    private_key = file("./devops-msm.pem")
    host     = self.public_ip
  }
}

resource "aws_security_group" "allow_http_https" {
  name        = "exapp-3-sg"
  description = "Allow http and https inbound traffic"

  ingress {
    description = "TLS from VPC"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```



```

ingress {
  description = "http from VPC"
  from_port   = 80
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

ingress {
  description = "ssh from VPC"
  from_port   = 22
  to_port     = 22
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

egress {
  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
}

}

resource "aws_eip" "lb" {
  instance = aws_instance.myec2.id
  vpc      = true
}

output "public_ip" {
  value = aws_instance.myec2.public_ip
}

```

L'IP retournée par l'instance myec2, n'est pas celle qui est joignable par Internet vu que nous avons créé une elastic IP. Pour afficher la bonne adresse, déplacez le provisionneur local dans le bloc de l'elastic ip

```

resource "aws_eip" "lb" {
  instance = aws_instance.myec2.id
  vpc      = true
  provisioner "local-exec" {
    command = "echo PUBLIC IP: ${aws_eip.lb.public_ip} ; ID: ${aws_instance.myec2.id} ;
AZ: ${aws_instance.myec2.availability_zone}; >> infos_ec2.txt"
  }
}

```