Lab 3- Déploiements dynamiques avec les variables

Objectif

Pour que votre code Terraform devient partageable, mieux contrôlé et sécurisé par git, cet article vous présenter l'utilisation des **variables Input et Ouput** Terraform comme moyen de le faire.

1. Input variables

Création d'une Input variable

Les Input variables ou "variables d'entrée", sont généralement définies en indiquant un nom, un type et une valeur par défaut. Cependant, le type et les valeurs par défaut ne sont pas strictement nécessaires. Terraform est assez intelligent et peut déduire le type de votre variable.

Dans notre première modification, nous allons permettre à notre utilisateur de modifier d'abord la région AWS. Pour ce faire, nous extrairons d'abord notre région dans une variable.

Commencez d'abord par créer à la racine de votre projet un autre fichier avec l'extension **.tf** dans mon cas je vais le nommer **vars.tf** et y rajouter le contenu suivant :

```
variable "AWS_REGION" {
    type = "string"
    default = "us-east-2"
}
```

Comme mentionné précédemment, on n'est pas obligé de spécifier le type, on peut donc écrire notre code autrement :

```
variable "AWS_REGION" {
    default = "us-east-2"
}
```

Ici, nous avons créé la variable `AWS_REGION` avec `us-east-2` comme valeur par défaut qui sera utilisée si elle n'est pas définie ailleurs. D'autre part, si vous ne définissez aucune valeur par défaut dans votre bloc de variable, la valorisation de cette variable sera alors obligatoire.

Vous pouvez également décrire brièvement l'objectif de votre variable à l'aide de l'argument description qui est facultatif :

```
variable "AWS_REGION" {
    type = "string"
    default = "us-east-2"
    description = "Région de notre instance ec2"
}
```

Accéder à une Input variable

Maintenant, dans votre code principal voici comment vous accédez à votre variable AWS_REGION :

```
provider "aws" {
    region = var.AWS_REGION
}
```

Ou :

```
provider "aws" {
    region = "${var.AWS_REGION}"
}
```

Ici, on utilise la variable nommée **AWS_REGION** préfixée par le mot-clé **var**. Si vous souhaitez concaténer votre variable avec une autre chaîne de caractères alors précédez la avec le symbole dollar $ entouré par des accolades {} et des doubles/simples guillemets ".

Définir une Input variable

En ligne de commande

Vous pouvez définir des variables directement depuis la ligne de commande avec l'option -var :

```
terraform apply -var 'AWS_REGION=us-east-2'
```

L'option **-var** peut-être utilisée un certain nombre de fois dans une seule commande, par exemple :

```
terraform apply -var 'AWS_REGION=us-east-2' -var 'AWS_AMI=ami-07c1207a9d40bc3bd'
```

La définition de variables de cette manière est utile pour tester rapidement votre code Terraform avec différentes valorisations, mais ne les enregistrera pas et devront être saisie à plusieurs reprises lors de l'exécution des commandes.

À partir d'un fichier

Pour conserver les valeurs des variables, nous créerons un fichier avec l'extension **.tfvars** et nous affecterons des variables dans ce fichier. Commencez donc par créer un fichier avec cette extension et rajoutez-y le contenu suivant :

```
AWS_REGION = "us-east-2"
```

Pour information, si vous nommez votre fichier à la lettre près **terraform.tfvars** ou toute variation de ***.auto.tfvars**, alors Terraform le chargera **automatiquement**, sinon si vous décidez de nommer votre fichier autrement, vous pouvez utiliser l'option **-var-file** pour spécifier votre fichier de valorisation.

```
terraform apply -var-file="aws-access.tfvars" -var-file="aws-config.tfvars"
```

Mode interactif

Si vous exécutez la commande **terraform apply** avec des variables non spécifiées, alors Terraform vous demandera de saisir les valeurs de manière interactive. Ces valeurs ne sont pas enregistrées, mais cela fournit un flux de travail pratique lors du démarrage de Terraform.

Les types de variables

Type string (chaîne de caractères)

Le type string est sûrement le type de variable les plus couramment utilisées. C'est une séquence de caractères Unicode représentant du texte, comme "hello".

```
variable "MY-STRING" {
    type = "string"
    default = "hello"
}
```

Type number (nombre)

Ce type de variable vous permet de spécifier une valeur numérique. Il peut se représenter à la fois des nombres entiers comme "80" et des valeurs décimaux comme "3.14" :

```
variable "INGRESS-PORT" {
    type = "number"
    default = 80
}
```

Type bool (booléen)

Ce type de variable donne la possibilité d'utiliser de simples valeurs vraies ou fausses :

```
variable "DELETE-TERMINATION" {
    type = "bool"
    default = true
}
```

Type list (les Listes)

Ils fonctionnent comme un catalogue de valeurs numéroté. Chaque valeur peut être appelée par son index correspondant dans la liste. Les éléments d'une liste sont identifiés par des nombres entiers consécutifs, en commençant par zéro. Voici un exemple de définition d'une variable de type liste :

```
variable "USERS" {
    type = "list"
    default = ["root", "user1", "user2"]
}
```

Pour accéder à un élement d'une liste vous devrez indiquer l'index de la valeur que vous recherchez :

```
username = "${var.USERS[0]}"
```

Maps

Une map est une structure de données sous forme de clé/valeur qui peut également contenir d'autres clés et valeurs. Ceux-ci peuvent être utiles pour sélectionner des valeurs basées sur des paramètres prédéfinis. Par exemple, dans AWS les AMI sont spécifiques à la région géographique utilisée, on peut dans ce cas s'orienter vers l'utilisation des maps pour guider l'utilisateur dans ses choix :

```
variable "AWS_AMIS" {
  type = "map"
  default = {
    "us-east-1" = "ami-085925f297f89fce1"
    "us-east-2" = "ami-07c1207a9d40bc3bd"
  }
}
```

Voici comment utiliser votre map dans votre code terraform :

```
resource "aws_instance" "my_ec2_instance" {
  ami           = var.AWS_AMIS[var.region]
  instance_type = "t2.micro"
}
```

Si vous souhaitez indiquer la valeur en statique, la région peut être codée en dur comme suit :

```
resource "aws_instance" "my_ec2_instance" {
  ami           = "${var.AWS_AMIS["us-east-1"]}"
  instance_type = "t2.micro"
}
```

2. Les variables Output

Les Output variables ou "variables de sortie", constituent un moyen pratique d'obtenir des informations utiles sur votre infrastructure. Comme vous l'avez peut-être remarqué, la plupart des détails du serveur sont calculés lors du déploiement et ne deviennent disponibles qu'après. À l'aide des variables de sortie, vous pouvez extraire toutes ces informations spécifiques à votre infrastructure.

La configuration des variables de sortie est vraiment assez simple. Il vous suffit de définir un nom pour la sortie et quelle valeur elle doit représenter. Par exemple, vous pouvez demander à Terraform d'afficher l'adresse IP publique de notre serveur après le déploiement, comme suit :

```
output "public_ip" {
    value = aws_instance.my_ec2_instance.public_ip
}
```

Ceci définit une variable de sortie nommée "**public_ip**" de notre ressource "aws_instance" nommée "**my_ec2_instance**". D'ailleurs plusieurs blocs d'output peuvent être définis pour spécifier plusieurs variables de sortie.

3. Mise en pratique

Dans cet exemple, nous allons reprendre l'exemple du Lab 2. La première chose à faire consiste à déclarer nos différentes variables dans un fichier avec l'extension **vars.tf**, rajouter le contenu suivant :

```
variable "AWS_ACCESS_KEY" {}
variable "AWS_SECRET_KEY" {}

variable "AWS_REGION" {
    default = "us-east-1"
}

variable "AWS_AMIS" {
    type = map(string)
    default = {
        "us-east-1" = "ami-085925f297f89fce1"
        "us-east-2" = "ami-07c1207a9d40bc3bd"
    }
}
```

Ensuite la prochaine étape comprend l'utilisation de nos différentes variables dans notre code principal, soit :

```
provider "aws" {
    region = var.AWS_REGION
    access_key = var.AWS_ACCESS_KEY
    secret_key = var.AWS_SECRET_KEY
}

resource "aws_instance" "my_ec2_instance" {
    ami = var.AWS_AMIS[var.AWS_REGION]
    instance_type = "t2.micro"
    tags = {
        Name = "terraform test"
    }
}
```

Enfin, maintenant vous pouvez passer à l'étape de valorisation, me concernant je vais créer un fichier **terraform.tfvars** afin qu'il soit pris en compte automatiquement par le moteur Terraform :

```
AWS_ACCESS_KEY="votre_cle_dacces"
AWS_SECRET_KEY="votre_cle_secrete"
```

Vous pouvez bien sûr choisir de surcharger les autres variables, ou laisser les variables par défaut.

Exécutez ensuite votre projet Terraform à l'aide de la commande suivante :

```
terraform init && terraform apply
```

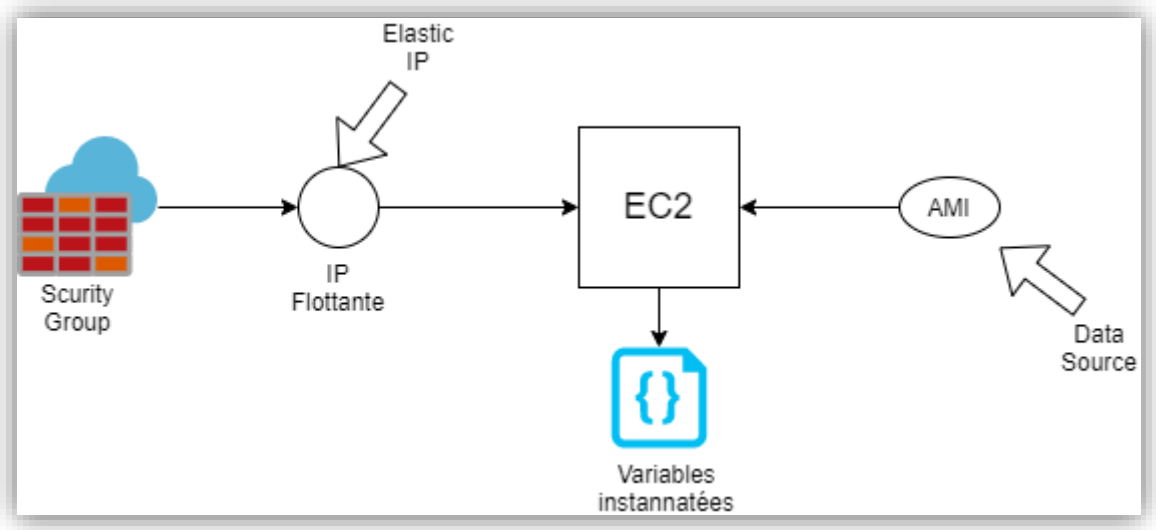
4. Exercice d'application 2

L'objectif est de déployer une instance ec2 avec une ip publique et un security group

- ***IP publique*** : vous allez créer une ip publique pour votre EC2
- ***Security Group*** : créez une security group pour ouvrir le port **80** et **443**, attachez cette security group à votre IP publique
- ***EC2***
  - Votre ec2 doit avoir une taille variabilisée, la valeur par défaut devrait être t2.nano et la valeur à surcharger sera t2.micro
  - L'image AMI à utiliser sera l'image la plus à jour de AMAZON LINUX
  - Spécifiez la key pair à utiliser (devops --<votre-Prnom>)
  - Attachez l'ip publique à votre instance
  - Variabilisez le Tag afin qu'il contienne au moins le tag: « Name: ec2 --<votre prenom » le N est bien en majuscule
- Supprimez vos ressources avec terraform destroy
- Créez un dossier « terraform-exapp-2» comme vous l'avez fait au « terraform-exapp-1» pour conserver votre code

Correction de l'exercice d'application 2

Le schéma ci-dessous résume l'architecture de nore déploiement dans cet exercice



- Créez un nouveau dossier pour le nouveau déploiement

```
mkdir terraform-exapp-2
cd terraform-exapp-2
```

- Créez un fichier Terraform **vars.tf** pour les variables à déclarer

```
variable instancetype {
  type        = string
  description = "Type de l'instance aws"
  default     = "t2.nano"
}

variable aws_common_tag {
  type        = map
  description = "Le tags aws"
  default = {
    Name = "ec2-training"
  }
}
```

- Listez les valeurs à surcharger dans le fichier Terraform **terraform.tfvars**

```
instancetype = "t2.micro"
aws_common_tag = {
  Name = "ec2-msm"
}
```

- Recopiez le contenu de la déclaration de votre instance ec2 déclarée dans l'exercice d'application 1 et rendez le dynamique à travers les variables.

```
provider "aws" {
  region     = "us-east-1"
  access_key = "votre acsess key"
  secret_key = "votre secret key"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0083662ba17882949"
  instance_type = "t2.micro"
  key_name      = "devops-msm"
  tags = {
    Name = "ec2-msm"
  }
}
```

Modification de l'image amazon pour télécharger toujours la dernière version AMAZON LINUX. Pour rendre la gestion de l'image dynamique, nous devrons utiliser la ressource data de AWS en filtrant l'image désirée.

Pour plus de détails consultez la documentation suivante https://www.terraform.io/docs/language/data-sources/index.html

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

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

- Type de ressource : aws_ami
- Nom de reource : app_ami
- Image de Amazon

Filtre sur le nom de l'image, pour déterminer la dernière verion

- Filtre : amzn2-ami-hvm*

Puis sil faute mettre à jour la description de la ressource AWS ;

- ***Ami*** : récupérer la valeur depuis la data source créé (aws_ami.app_amià, l'identification c'et à travers de l'ID de la ressource.
- ***instance_type*** : récupérer le type depuis la variable instance_type déclarée dans le fichier vars.tf
- ***tags :*** récupérer les tags depuis la map déclarée dans le fichier vars.tf

```
resource "aws_instance" "myec2" {
  ami             = data.aws_ami.app_ami.id
  instance_type   = var.instancetype
  key_name        = "devops-msm"
  tags            = var.aws_common_tag
}
```

Créer une nouvelle ressource de Security_group ; Pour plus de détails consultez la documentation suivante https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group

Dans notre exemple la ressource security_group concerne l'autorisation des ports 80 et 443 (http et https), pour la créer il faut ajouter la ressource "**aws_security_group**" avec les paramètres suivants :

- ***Name*** : pour identifier la ressource dans la définition de l'instance.
- ***Ingress*** : pour autoriser le service (le port) ainsi que la configuration nécessaire. Nous allons autoriser deux ports le 80 et le 443, donc il faut ajouter deux blocs ingress pour chaque port.

```
resource "aws_security_group" "allow_http_https" {
  name        = "exapp-2-sg"
  description = "Autoriser http et https pour le traffic en entrée"

  ingress {
    description = "HTTPS Traffic"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = " HTTP Traffic"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Autoriser le traffic https (port 443) pour tous les accès ["0.0.0.0/0"]

Autoriser le traffic http (port 80) pour tous les accès ["0.0.0.0/0"]

Puis il faut mettre à jour la description de ressource **"aws_instance"** pour la liaison avec la ressource **"aws_security_group"**

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

Ajout de l'IP Flottante, c'est une ressource **elastic ip aws_eip**

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

Une fois que vous avez fini avec l'édition de votre déploiement, formate votre code à travers la commande terraform fmt pour plus de lisibilité

```
terraform fmt
```

Votre déploiement dynamique de final l'instance aws devra être comme suit :

```
provider "aws" {
  region     = "us-east-1"
  access_key = "votre acsess key"
  secret_key = "votre secret key"
}
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.instancetype
  key_name        = "devops-msm"
  tags            = var.aws_common_tag
  security_groups = [aws_security_group.allow_http_https.name]

}

resource "aws_security_group" "allow_http_https" {
  name        = "exapp-2-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"]
  }
}

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