LES VARIABLES INPUT ET OUTPUT

Introduction

Dans le chapitre précédent, nous avions vu comment déployer une instance web ec2 automatiquement sur AWS à l'aide de notre code Terraform, cependant notre code n'est pas assez sécurisé et réutilisable. Pourquoi ? Déjà, nous avions codé en dur les clés d'accès de notre utilisateur AMI et en règle générale, **vous ne mettez jamais vos informations d'identification dans votre repo Git** ! Par ailleurs les utilisateurs de votre équipe veulent peut être utiliser une autre ami, autre que Ubuntu ? Sans oublier que les AMI sont différentes selon les régions. Dans ce cas, c'est vraiment bien de les avoir dans des variables afin de faciliter pour les autres et vous-même la **réutilisation de vos fichiers Terraform**. Quand ils sont plus génériques, il est plus facile de les réutiliser pour nos projets.

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

Input variables

Création d'une Input variable

Les Input variables ou "variables d'entrée" en français, 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 et doit être définie à l'aide d'une des méthodes présentées plus loin dans ce chapitre.

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"
}
```

Accèder 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 `"`. Vous remarquerez, qu'il n'est pas nécessaire de spécifier le nom de votre fichier de variable car lors de la construction de votre infrastructure, Terraform analysera pour vous automatiquement tous vos autres fichiers (vachement sympa ce Terraform).

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, comme par exemple :

```
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.

Remarque

Dans les versions 0.11 et antérieures de Terraform, le mode interactif n'est pas pris en charge que pour les variables de type `string`. Les types `list` et `map` doivent être remplies via l'un des autres mécanismes vus précédemment. Terraform 0.12 introduit la possibilité de remplir des types de variables complexes à partir de l'invite de l'interface utilisateur.

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"
}
```

Output Les variables

Je vous ai présenté les variables d'entrée comme moyen de paramétrer les configurations Terraform. Cette fois-ci, je vous introduis aux variables de sortie comme un moyen d'organiser les données à interroger facilement et à les afficher à l'utilisateur.

Les Output variables ou "variables de sortie" en français, 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.

Amélioration de notre projet

Dans cet exemple, nous allons prendre l'exemple du chapitre précédent que vous pouvez télécharger en cliquant ici.

La première chose à faire consiste à déclarer nos différentes variables dans un fichier avec l'extension **.tf**, ici je vais le nommer **vars.tf** et je vais y rajouter le contenu suivant :

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

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

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

Vous remarquerez, que je ne mets pas de valeur par défaut sur les variables `AWS_ACCESS_KEY` et `AWS_SECRET_KEY` car rappelez-vous ne stockez jamais vos informations d'identification dans votre repo Git ! On s'occupera de les valoriser plus

tard.

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_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"]
    }
}

resource "aws_instance" "my_ec2_instance" {
    ami = var.AWS_AMIS[var.AWS_REGION]
    instance_type = "t2.micro"
    vpc_security_group_ids = [aws_security_group.instance_sg.id]

 user_data =  /var/www/html/index.html
 EOF

    tags = {
        Name = "terraform test"
    }
}

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

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, moi je vais utiliser les variables par défaut.

Histoire de sécuriser un peu le tout je vais rajouter un fichier **.gitignore** afin de ne pas versionner mon fichier **terraform.tfvars** :

```
/terraform.tfvars
/.terraform
```

Vous pouvez télécharger le projet complet en cliquant ici. Exécutez ensuite votre projet Terraform à l'aide de la commande suivante :

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

Résultat :

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

public_ip = 54.81.193.196
```

Les variables de sortie définies seront affichées à la fin du déploiement comme dans l'exemple ci-dessus. Cependant, vous pouvez également les récupérer à tout moment à l'aide de la commande `output`. Par exemple, pour obtenir l'adresse IP publique, vous pouvez utiliser l'exemple de commande ci-dessous :

```
terraform output
```

Résultat :

```
54.81.193.196
```

De la même manière, vous pouvez demander à Terraform toute autre variable de sortie.

Conclusion

Les variables Terraform offrent de nombreuses utilisations pratiques pour la gestion de votre infrastructure. La division de votre plan de déploiement et des paramètres de configuration dans leurs propres fichiers permet de tout garder en ordre ! Et vous pouvez en consommer sans modération .