# LES DATAS SOURCE SUR TERRAFORM

## Introduction

Dans cet article, nous discuterons des Datas Source Terraform qui nous offrent un autre moyen pour **mieux gérer vos Inputs Terraform**.

### C'est quoi ?

Une Data source ou "source de données" en français, représente une **information en lecture seule qui est extraite d'un fournisseur** (dans notre cas, AWS) à chaque fois que vous exécutez Terraform. L'ajout d'une Data Source à vos configurations Terraform ne crée pas une ressource supplémentaire, ça reste juste un moyen d'interroger les API du fournisseur pour récupérer des données et de **rendre ces données disponibles pour le reste de votre code Terraform**. Elles sont très utiles pour **fournir des informations dynamiques** à partir de l'API du provider.

## Data Source

### Création d'une Data Source

Voici à quoi ressemble la **syntaxe d'utilisation de la création d'une Data source** qui reste très similaire à la syntaxe d'une ressource :

```
data "<DATA_SOURCE_NAME>" "&lg;NAME>" {
    [CONFIG ...]
}
```

- `DATA_SOURCE_NAME` : correspond à la ressource sur laquelle vous souhaitez récupérer des informations, la liste de toutes les Data Sources est disponible ici.
- `NAME` : identifiant que vous pouvez utiliser dans le code Terraform pour faire référence à cette source de données.
- `CONFIG` : un ou plusieurs arguments qui sont spécifiques à cette Data Source.

Dans nos exemples précédents, nous avions spécifié manuellement notre AMI Ubuntu dans notre code Terraform, cependant rappelez-vous que cette information est dynamique et peut donc être modifiée à tout moment. Pour résoudre ce problème nous utiliserons une Data source afin de **récupérer cette information automatiquement dans notre code Terraform**. Pour ce faire, nous utiliserons la Data Source "aws_ami", et voici les arguments que nous utiliserons pour le moment pour ce type de Data source :

- `owners` (Obligatoire) : cet argument représente la liste des propriétaires de l'AMI. Au moins 1 valeur doit être spécifiée. Seules les valeurs suivantes sont valides : l'id du compte AWS propriétaire de l'AMI, `self` (le propriétaire du compte courant), ou les fournisseurs suivants : amazon, aws-marketplace et microsoft.
- `most_recent` (Facultatif) : dans cet argument, si plusieurs résultats sont renvoyés, Terraform utilisera l'AMI la plus récente.

Dans ce cas nous souhaitons récupérer l'AMI Ubuntu, nous devons donc récupérer l'id du compte AWS propriétaire officiel de cette image (Canonical) de cette AMI. Pour cela, nous utiliserons la CLI AWS avec la commande `aws ec2 describe-images` afin de **récupérer le propriétaire de notre AMI** Ubuntu de la région us-east-1 :

```
aws ec2 describe-images --image-ids ami-085925f297f89fce1 --region us-east-1
```

**Résultat :**

```
{
    "Images": [
        {
            "VirtualizationType": "hvm",
            "Description": "Canonical, Ubuntu, 18.04 LTS, amd64 bionic image build o
            ...
            "ImageId": "ami-085925f297f89fce1",
            ...
            "RootDeviceType": "ebs",
            "OwnerId": "099720109477",
            "Name": "ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200408
        }
    ]
}
```

Dans le résultat, nous récupérons l'id du propriétaire depuis la clé `OwnerId` qui a comme valeur `099720109477`. Nous avons donc obtenus toutes les informations pour les intégrer dans notre Data Source qui ressemblera à ceci dans notre code Terraform :

```
data "aws_ami" "ubuntu-ami" {
    most_recent = true
    owners = ["099720109477"] # Canonical
}
```

Maintenant, si vous lancez votre code avec cette Data Source tel quel, vous n'êtes pas sûr de récupérer la version d'ubuntu souhaitée. Car en effet, rappelez-vous que si vous spécifiez l'argument `most_recent` à `True` Terraform récupérera automatiquement pour vous la version la plus récente de cette AMI, puisque l'id de l'owner "099720109477" correspondant à la société Canonical maintient plusieurs AMI d'Ubuntu. Vous retrouverez la liste complète de ses images avec la commande suivante :

```
aws ec2 describe-images --owners 099720109477
```

Dans notre cas, on souhaite récupérer la version 18.04 LTS bionic (amd64) de la région us-east-1, il nous faut donc une **méthode pour filtrer notre résultat**.

## Filter

Les sources de données qui renvoient des listes de ressources prennent en charge le filtrage. Pour utiliser un filtre, incluez ce bloc de code dans votre définition de Data Source:

```
filter {
    name = ""
    values = [""]
}
```

- `name` : correspond au nom de la propriété à filtrer.
- `values` : correspond à une liste de valeurs de la propriété à filter ,elle peut contenir une ou plusieurs valeurs avec lesquelles filtrer.

Pour connaître la propriété à filter il suffit de se rendre dans la documentation officielle de la Data Source AMI. Pour notre besoin nous filtrerons le résultat depuis la propriété `name`, nous aurons ainsi le code suivant :

```
data "aws_ami" "ubuntu-ami" {
    most_recent = true

    filter {
        name   = "name"
        values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200408"]
    }

    owners = ["099720109477"] # Canonical
}
```

**Information**

Depuis la cli AWS ce code nous donnera la commande suivante : `aws ec2 describe-images --owners 099720109477 --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200408'`

## Extraire les données d'une Data Source

Pour **récupérer les données d'une Data Source**, vous utilisez la syntaxe de référence d'attribut suivante :

```
data.<DATA_SOURCE_NAME.&lg;NAME>.<ATTRIBUTE>
```

Vous retrouverez la liste des attributs récupérables également directement dans la documentation officielle de la Data Source aws_ami. Pour notre besoin on souhaite récupérer l'id de notre AMI, nous intercepterons ainsi l'attribut `id` depuis notre Data Source. Ce qui nous donnera le code final suivant :

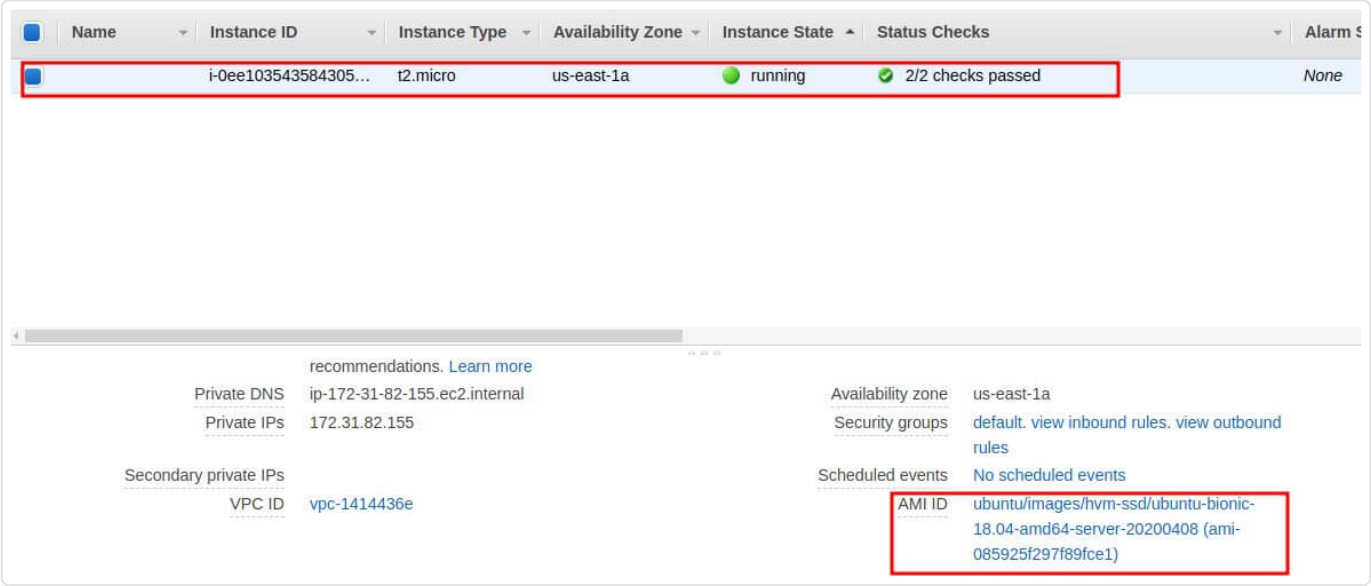
```
provider "aws" {
    region = "us-east-1"
}

data "aws_ami" "ubuntu-ami" {
    most_recent = true

    filter {
        name   = "name"
        values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200408"]
    }

    owners = ["099720109477"] # Canonical
}

resource "aws_instance" "my_ec2_instance" {
    ami = data.aws_ami.ubuntu-ami.id
```

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

Exécutez votre code Terraform avec la commande suivante :

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

Retournez ensuite sur votre console EC2 afin de vérifier la création de votre instance avec l'AMI adéquate :




# Data Source Externe

Une Data Source externe **permet à un programme externe d'agir comme une source de données**, en exposant des données qu'on peut réutiliser ailleurs dans notre configuration Terraform. Cependant, je vous mets en garde, une Data Source externe est susceptible de nuire à la portabilité de votre configuration Terraform en créant des dépendances sur des programmes et bibliothèques externes qui peuvent ne pas être disponibles sur différents systèmes d'exploitation. Donc, à utiliser seulement comme dernier recours, quand le provider ne possède pas l'information que vous souhaitez récupérer.

Nous allons d'abord commencer par créer un script qui récupère ou crée une information, vous pouvez choisir votre langage de programmation préféré, dans mon cas, je vais utiliser le langage python sous sa version 3. Chose importante, vous devez **afficher vos résultats au format JSON**! Mon script python ci-dessous renvoie simplement une chaîne de caractères aléatoire dans une clé nommée `random_name` au format json :

```
import random, string, json

def getRandomName(size:int = 6):
    return ''.join(random.SystemRandom().choice(string.ascii_uppercase + string.digit

data = {
  "random_name": getRandomName(10)
}

print(json.dumps(data))
```

Une fois le script créé, vous pouvez l'utiliser désormais en tant que Data Source externe dans votre code. La syntaxe reste la même qu'une Data Source normal seul le type change. Voici à quoi ressemblera notre code :

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

data "external" "random" {
     program= ["python3", "scripts/random-name.py"]
}

resource "aws_instance" "my_ec2_instance" {
    ami = "ami-085925f297f89fce1"
    instance_type = "t2.micro"

     tags = {
        Name = "${data.external.random.result.random_name}-ec2"
    }
}
```

Dans le bloc de code `data`, nous utilisons l'interpréteur sous sa version 3 et nous spécifions ainsi le chemin relatif de notre script python que j'ai déposé dans le dossier `scripts/` (vous pouvez également définir un chemin absolu). Enfin, les Datas Source externes exportent l'attribut `result` qui correspond aux valeurs de chaîne renvoyées par le programme externe.

## Conclusion

Une Data Source reste un bon moyen pour personnaliser vos entrées Terraform, grâce à eux vous pouvez récupérer n'importe quel type d'information afin de mieux paramétrer votre Code Terraform. J'espère que vous avez apprécié cet article. Dans le prochain article, nous discuterons des modules terraform.