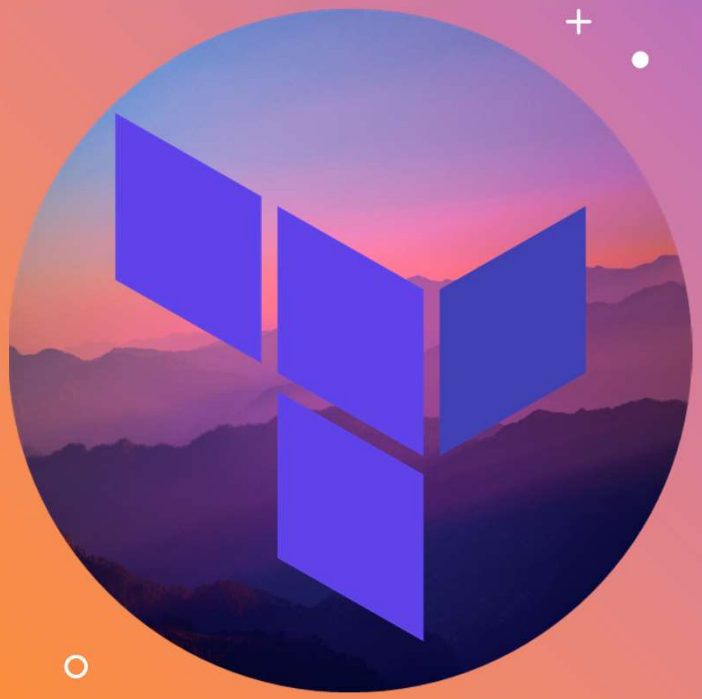
INTRODUCTION À TERRAFORM

Kévin ANSARD

SOMMAIRE



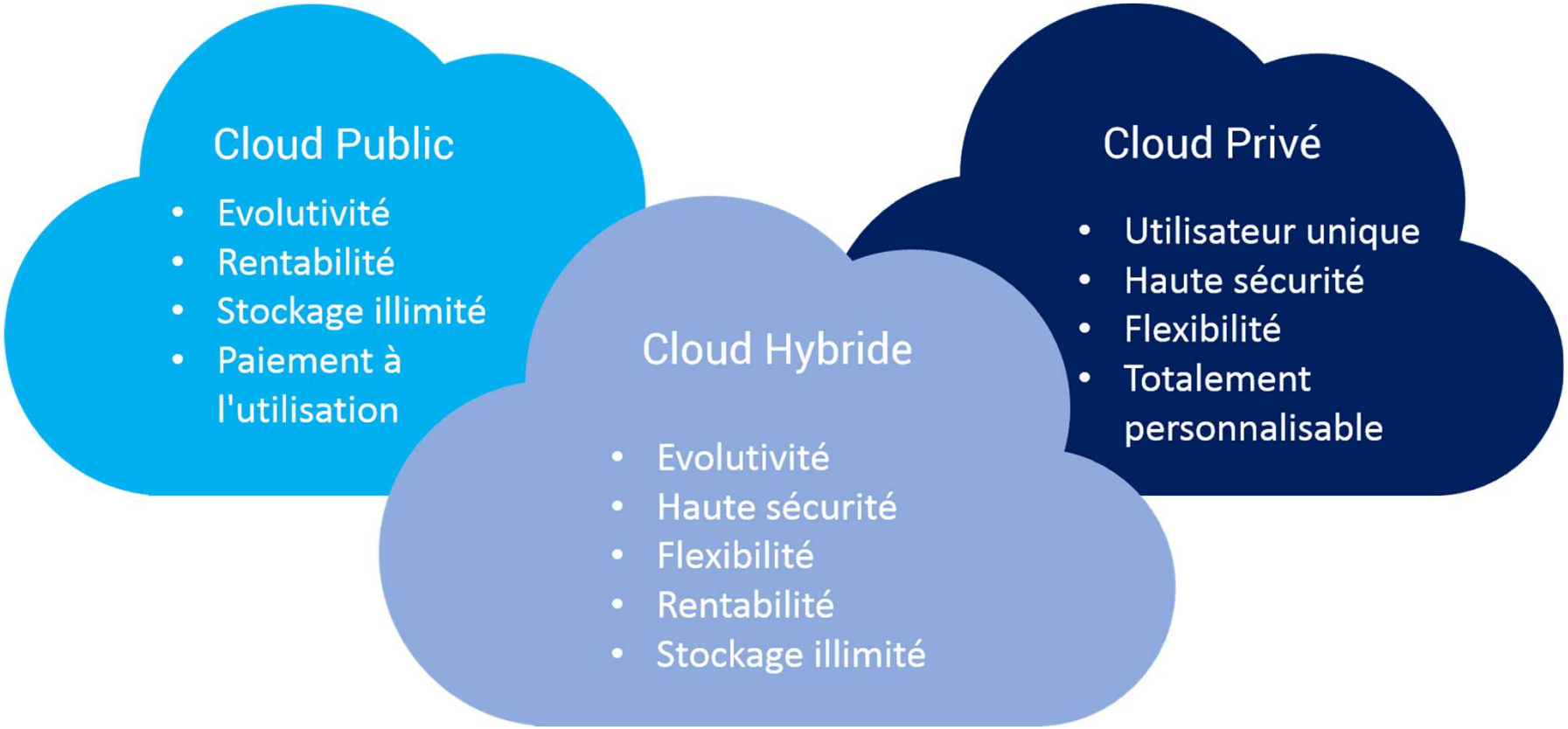
NOTIONS DE BASES

Qu’est ce qu’un cloud public ?

Un cloud public est une infrastructure informatique dans laquelle un fournisseur de services met des ressources à la disposition du public via internet. Les ressources varient selon le fournisseur mais peuvent inclure des capacités de stockage, des applications ou des machines virtuelles.

Le cloud public permet une évolutivité et un partage des ressources qu'une seule organisation ne pourrait pas réaliser autrement.

Cloud public vs Cloud privée

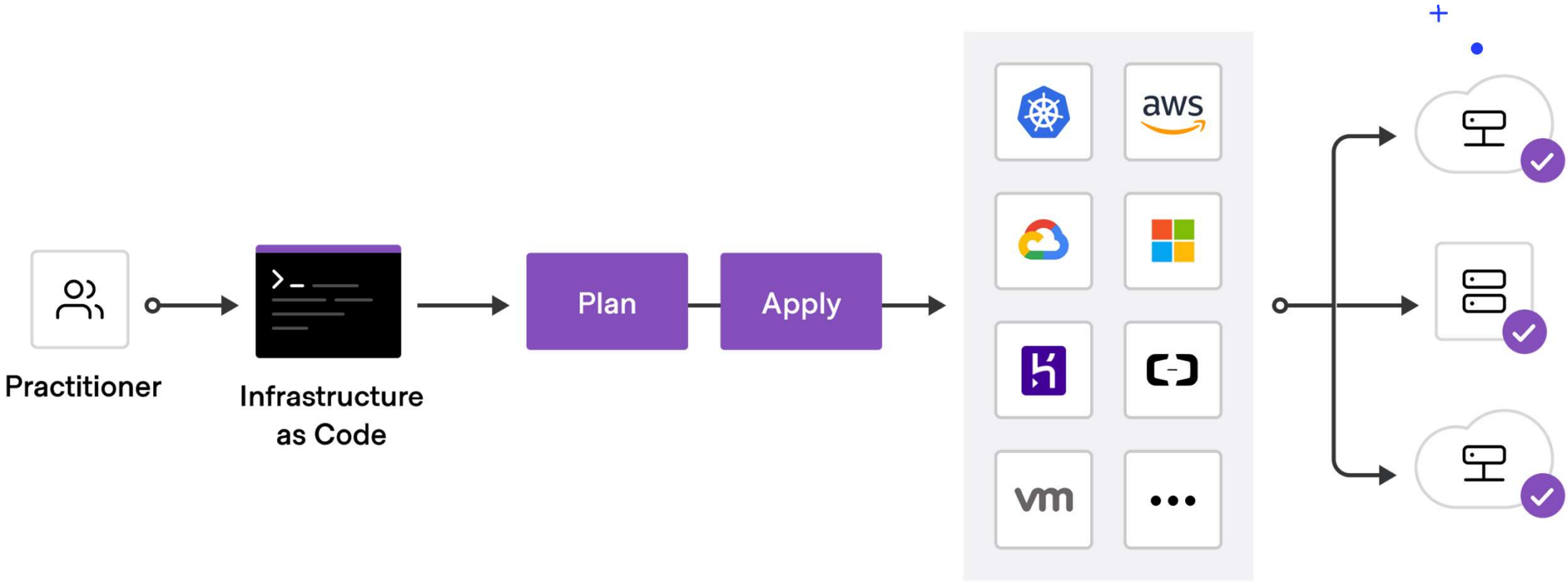


Infrastructure as Code (IaC)

(IaC) est la gestion et l'approvisionnement de l'infrastructure via le code plutôt que via des processus manuels.

Avec IaC, des fichiers de configuration contenant les spécifications de votre infrastructure sont créés, ce qui facilite la modification et la distribution des configurations

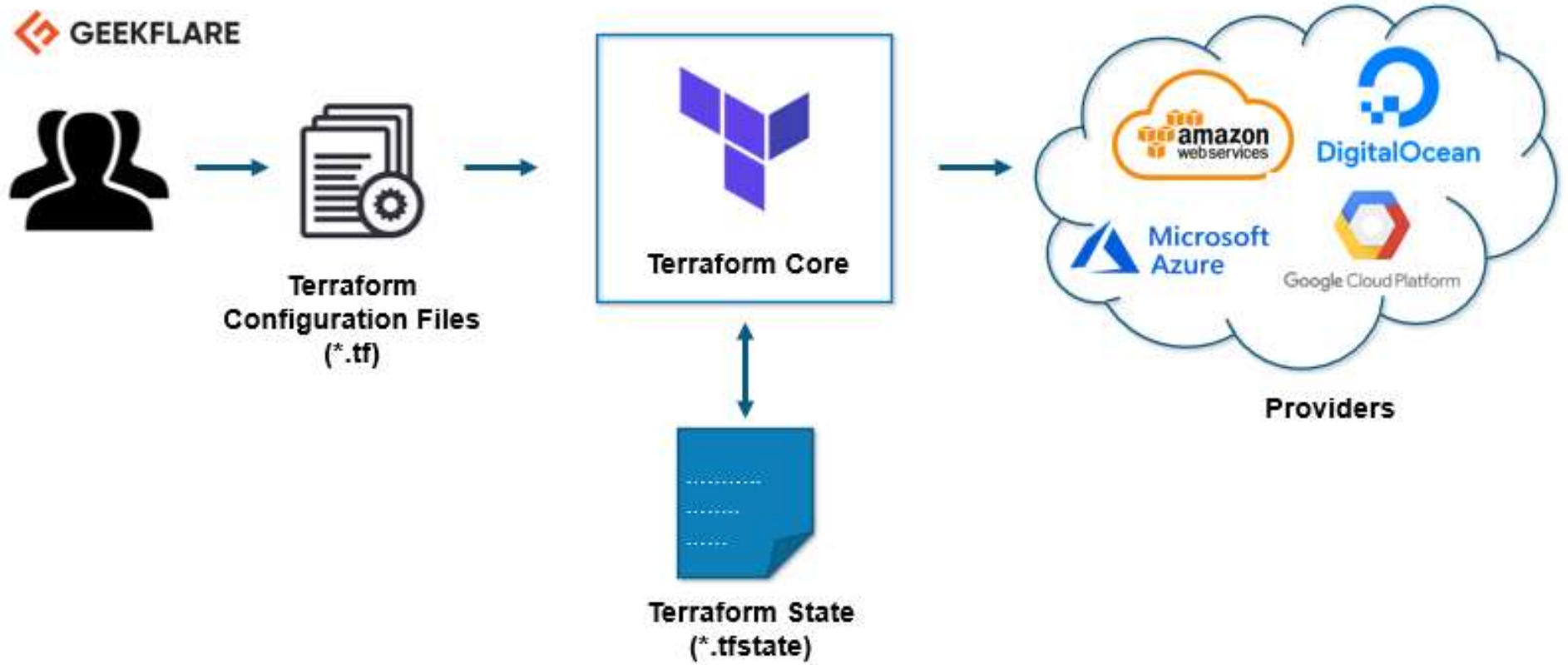
Infrastructure as Code



Qu’est ce que Terraform ?

Terraform est un outil logiciel IaC open source développé en GO créé par HashiCorp. Les utilisateurs définissent et fournissent l'infrastructure du centre de données à l'aide d'une configuration déclarative. Les langages utilisés sont HCL ou Json.

Comment fonctionne Terraform ?

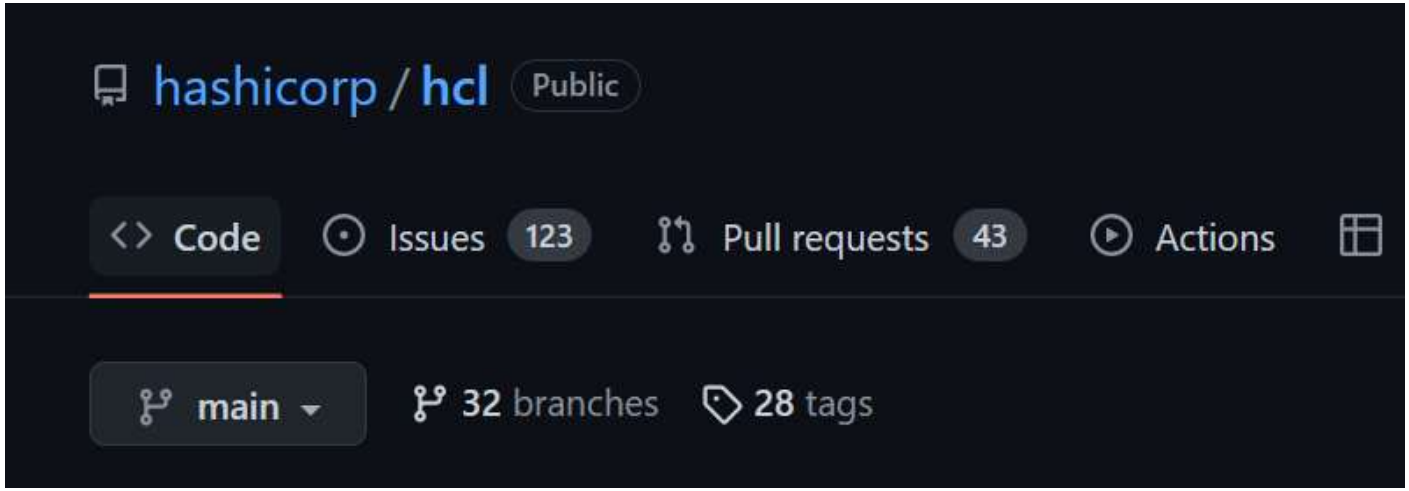


CONCEPTES TERRAFORM

Terraform un langage maison

(La topologie de terraform est basée sur un langage crée par Hashicop, le HCL (Hashicorp Configuration Language).

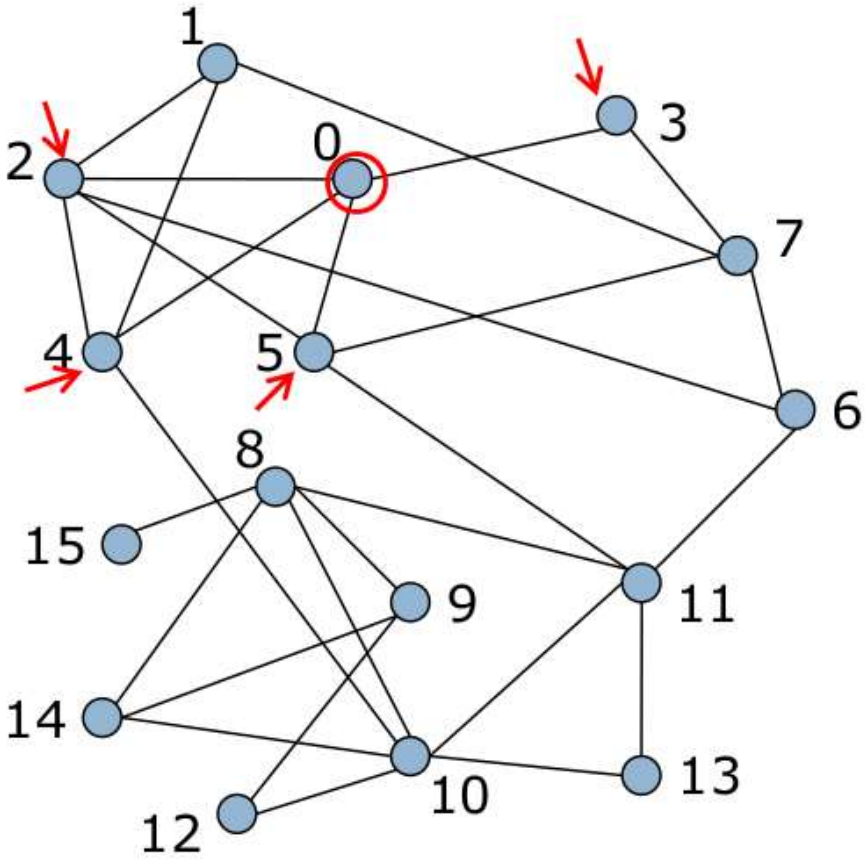
Le but de HCL est d'être compréhensible et éditable par des humains et par des machines (ANSIBLE, etc.).



Terraform et l'idempotency

$$f(f(x)) = f(x)$$

Terraform graph



Terraform State ou .TfState

Les providers

Terraform va communiquer avec différent providers via leurs api respective .

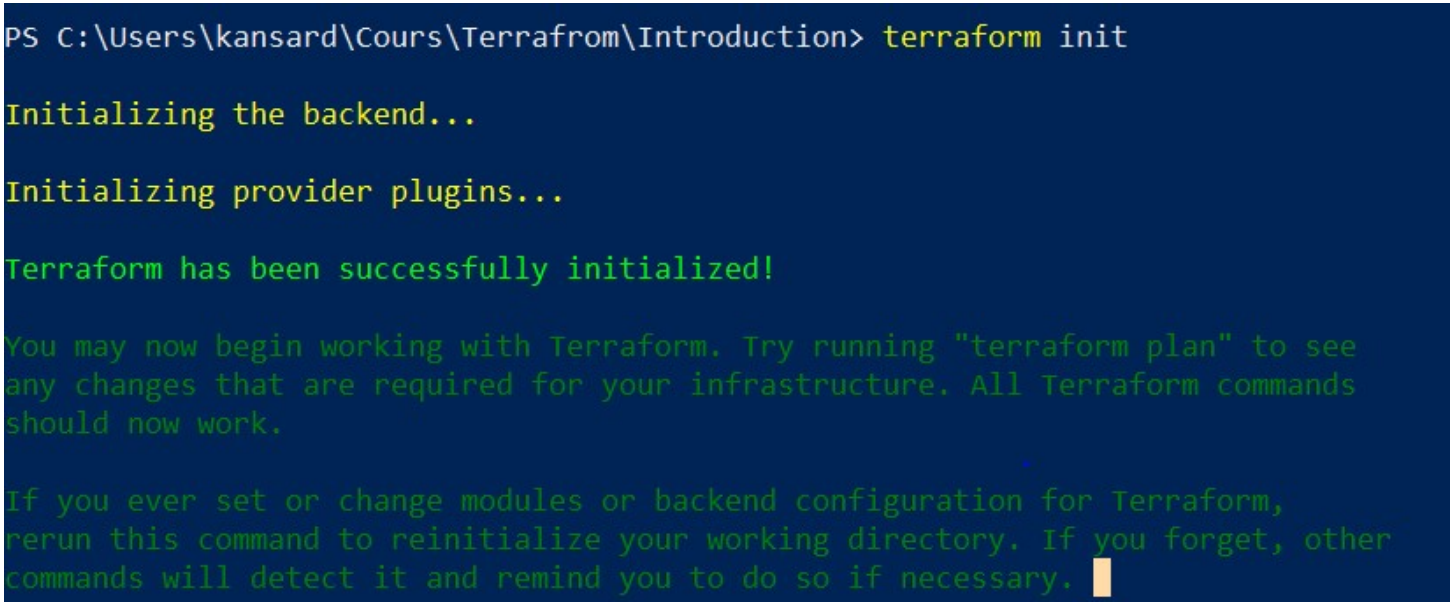
Pour Azure le service ce nomme Azure Ressources manager. On communique avec celui-ci via des templates json.

LES COMMANDES TERRAFORM

Terraform commandes de bases

```
Main commands:
  init            Prepare your working directory for other commands
  validate        Check whether the configuration is valid
  plan            Show changes required by the current configuration
  apply           Create or update infrastructure
  destroy         Destroy previously-created infrastructure
```

Terraform init



Terraform plan

```
PS C:\Users\kansard\Cours\Terrafrom\Introduction> terraform plan

Changes to Outputs:
  + monoutput = (sensitive value)

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
PS C:\Users\kansard\Cours\Terrafrom\Introduction>
```

Terraform apply

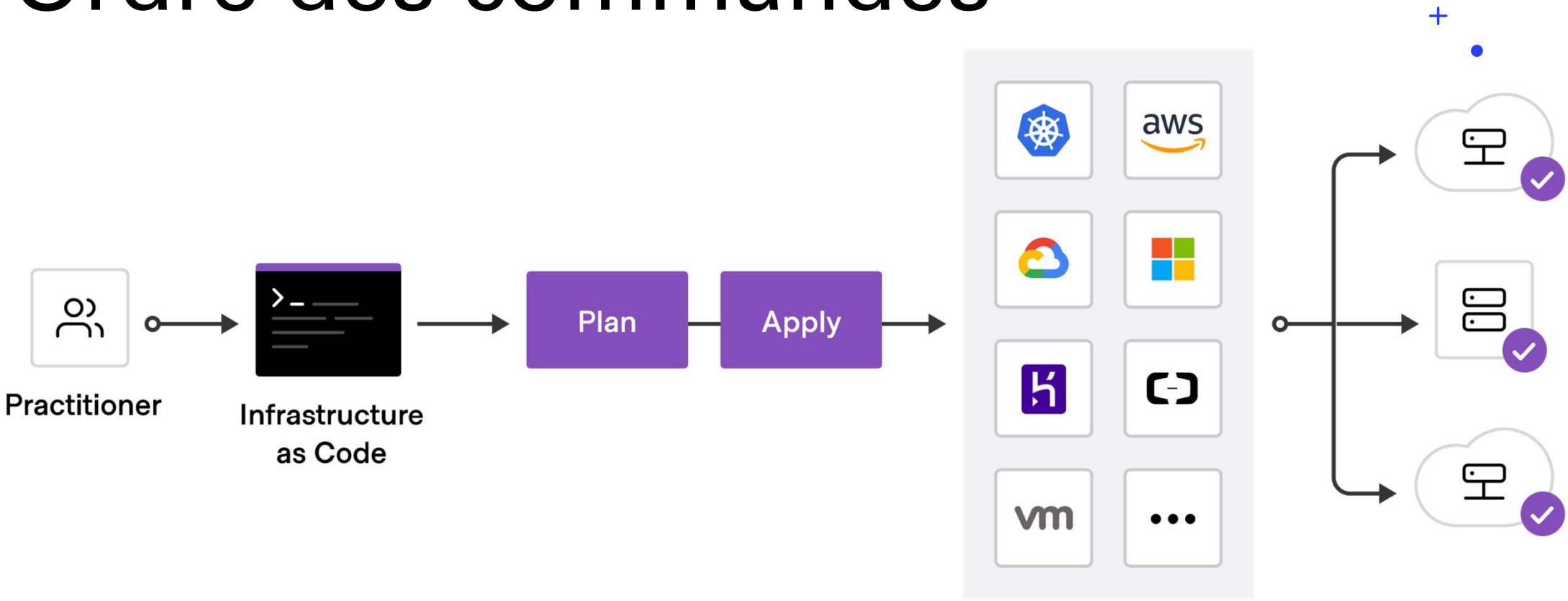
```
PS C:\Users\kansard\Cours\Terrafrom\Introduction> terraform

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your

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

Outputs:

monoutput = "Hello world!!!"
```

Terraform apply et tfState

Terraform validate

```
PS C:\Users\kansard\Cours\Terrafrom\Introduction> terraform validate
Success! The configuration is valid.
```

Ordre des commandes



LEXIQUE TERRAFORM
CONCEPTES INTERNES

Les Ressources

Les ressources sont les éléments les plus importants du langage Terraform. Chaque bloc de ressources décrit un ou plusieurs objets d'infrastructure.

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

Les Datasources

Une datasource est une source de données extérieure à Terraform ; on les déclare grâce au mot clé ***data***.

Terraform va donc pouvoir interroger les différents providers à sa disposition pour récupérer des informations sur différentes ressources

```
data "aws_ami" "example" {
  most_recent = true

  owners = ["self"]
  tags = {
    Name   = "app-server"
    Tested = "true"
  }
}
```

Les Variables

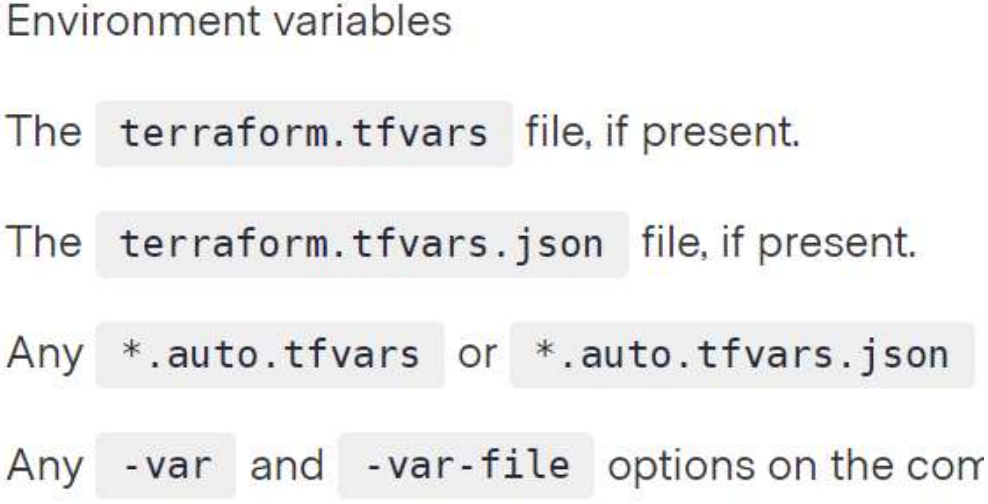
Les variables sont semblables aux variables de tout autre langage :

```
variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}
```

Les Variables et la precedence

Si plusieurs valeurs sont attribuées à la même variable, Terraform utilise la dernière valeur trouvée, en remplaçant toutes les valeurs précédentes.

Notez qu'une même variable ne peut pas se voir attribuer plusieurs valeurs dans une même source.



Les Outputs

Les blocs output permettent d'écrire dans la console lors de l'exécution

```
output "api_base_url" {
  value = "https://${aws_instance.example.private_dns}:8433/"

  # The EC2 instance must have an encrypted root volume.
  precondition {
    condition     = data.aws_ebs_volume.example.encrypted
    error_message = "The server's root volume is not encrypted."
  }
}
```

Les Modules

Terraform permet de créer des modules pour pouvoir réutiliser du code. Les modules sont le système d'abstraction de Terraform

```
module "consul" {
  source  = "hashicorp/consul/aws"
  version = "0.0.5"

  servers = 3
}
```

EXERCICES

Exercice 1

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui écrit **le nom de votre film préféré** dans le fichier **film.txt**

Exercice 2

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui écrit **le nom de votre film préféré** dans le fichier **film.txt** le nom du film doit être contenu dans une variable nommé `movie_name`.

De plus vous devez ajouté un output « Fichier film.txt créer ».

Utilisation d'une variable de type String.

Exercice 3

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui écrit **le nom de vos 3 films préférés** dans le fichier **Movies.txt** le nom des film doit être contenu dans une variable nommé `movies_name`. Chaque film doit avoir son commentaire associé. (Utilisation d'un variable de type MAP).

Exercice 4

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui écrit **le nom de vos 3 films préférés** dans le fichier **Movies.txt** le nom des film doit être contenu dans une variable nommé `movies_name`. Chaque film doit avoir son commentaire associé. (Utilisation d'un variable de type Array).

Exercice 5

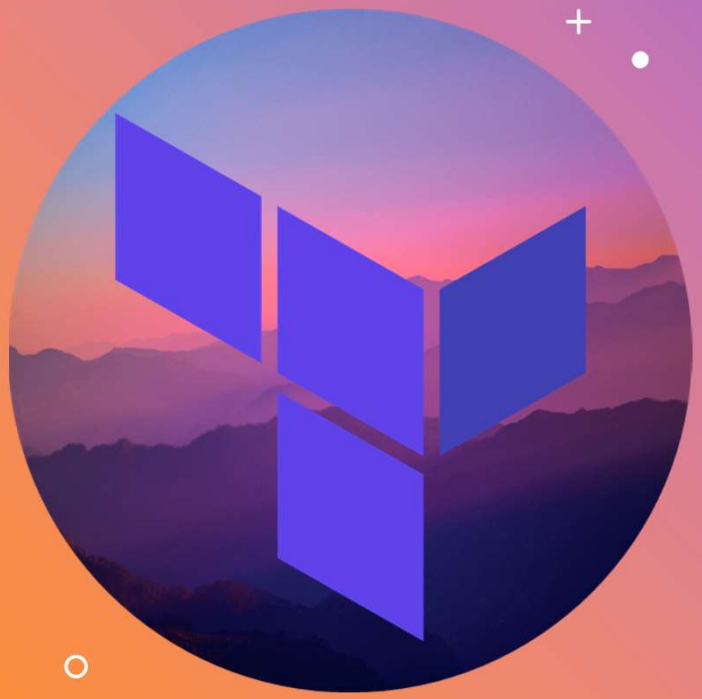
Le but de cet exercice est de créer une configuration Terraform (main.tf) qui écrit **le nom de vos 3 films préférés** dans le fichier **Movies.txt** le nom des film doit être contenu dans une variable nommé `movies_name`. Chaque film doit avoir son commentaire associé. (Utilisation d'un variable de type Array). Vous devez utiliser le principe de la précédence de données.

THANK YOU

Kévin ANSARD

ansardkev@hotmail.fr

SOMMAIRE PARTIE 2



- Conceptes interne avancé
- Connection ssh
- Le provider remote_exec ?
- Comment nous connecter?
- Comment éxécuter nos commande ?
- Mise en pratique du remote_exec !
- Le provider file
- Comment copier un fichier en remote ?
- Mise en pratique de file !
- Les modules introduction avancé

CONCEPTES INTERNES AVANCÉ

Le provider local_exec

Terraform pocède un provider nommé « local_exec » celui-ci permet d'exécuter des commandes sur le cmd de la machine de lancement

```
resource null_resource node1 {
  provisioner "local-exec" {
    command = "echo '${var.script_name}' > BlackAdam.txt"
  }
}
```

Les triggers

Terraform permet de verifié les changements d'état d'une variable ou d'une data en générale

```
triggers = {
  foo = element(var.movies_name, count.index)
}
```

EXERCICES INTERMEDIAIRE

Exercice 6

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui install l'outil figlet avec la commande suivante :

```
sudo apt install -y figlet
```

Attention vous devez pensez à récupérer le mot de passe de votre utilisateur.

Exercice 7

Le but de cet exercice est de désinstaller le package « figlet »:

```
sudo apt-get remove figlet
```

<u>Attention vous devez pensez à récupérer le mot de passe de votre utilisateur.</u>

Exercice 8

Le but de cet exercice est de créer une configuration Terraform (main.tf) qui install l'outil AsciiArt avec la commande suivante mais cette fois-ci vous devez stocker le package à installer dans une variable:

```
sudo apt install -y figlet
```

Attention vous devez pensez à récupérer le mot de passe de votre utilisateur selon votre OS.

Exercice 9

Le but de cet exercice est de refaire l'exercice 8 mais cette fois ci en ajoutant un un trigger :

```
sudo apt install -y figlet
```

Attention vous devez pensez à récupérer le mot de passe de votre utilisateur selon votre OS.

Exercice 10

Le but de cet exercice est de refaire l'exercice 7 mais cette fois ci en ajoutant un une variable et un trigger :

```
sudo apt-get remove -y figlet
```

Attention vous devez pensez à récupérer le mot de passe de votre utilisateur selon votre OS.

RAPPEL TECHNIQUE

Qu’est ce q’une connexion ssh?

Le **SSH**, pour Secure Shell, désigne à la fois un protocole de communication et un programme informatique. Il permet la connexion d'une machine distante (serveur) via une liaison sécurisée dans le but de transférer des fichiers ou des commandes en toute sécurité.

DÉMO MISE EN PLACE D'UN SSH AVEC UNE PRIVATE KEY

Le provider remote exec

Le remote exec appelle un script sur une ressource distante après sa création.

Cela peut être utilisé pour exécuter un outil de gestion de configuration, démarrer dans un cluster, etc.

Le remote exec nécessite une connexion et prend en charge à la fois ssh et winrm.

Le provider connection

Le provider « connection » permet à l'utilisateur de spécifié comment le remote-exec ce connecte à la machine distante

```
connection {
  type     = "ssh"
  user     = "root"
  password = var.root_password
  host     = self.public_ip
}
```

Le provider remote exec

Le provider remote-exec permet via « inline » de passer une suite de commande à exécuter sur la machine distante

```
provisioner "remote-exec" {
  inline = [
    "puppet apply",
    "consul join ${aws_instance.web.private_ip}",
  ]
}
```

Le provider file

Le provider file permet de copier un fichier locale sur la machine distante.

Attention vous ne pourrez copier vos fichier uniquement dans le répertoire /tmp/

```
provisioner "file" {
  source      = "script.sh"
  destination = "/tmp/script.sh"
}
```

EXERCICES

Exercice 11

Le but de cet exercice est de ce connecter à une machine distante et d'installer figlet sur celle-ci via la commande suivante:

```
sudo apt-get install figlet
```

Exercice 12

Le but de cet exercice est de refaire l'exercie 11 mais cette fois-ci en utilisant un fichier terraform.tfvars

```
sudo apt-get install figlet
```

Exercice 13

Le but de cet exercice est de copier un fichier locale sur la machine distante puis de le déplacer dans un autre répertoire.

MINI PROJET

Mini projet =)

Le but de ce projet est d'installer l'outil de votre choix (ex : tree / emacs / etc) à la fois sur votre machine en locale et sur votre machine distante :

Mini projet 2 =')

Le but de ce projet est d'installer l'outil Docker sur votre machine distante :

LES MODULES

Les Modules

Terraform permet de créer des modules pour pouvoir réutiliser du code. Les modules sont le système d'abstraction de Terraform

```
module "consul" {
  source  = "hashicorp/consul/aws"
  version = "0.0.5"

  servers = 3
}
```

AZURE ET TERRAFORM

AWS ET TERRAFORM

Les Modules

Terraform permet de créer des modules pour pouvoir réutiliser du code. Les modules sont le système d'abstraction de Terraform

```
module "consul" {
  source  = "hashicorp/consul/aws"
  version = "0.0.5"

  servers = 3
}
```

THANK YOU

Kévin ANSARD

ansardkev@hotmail.fr