



Terraform



HashiCorp

Terraform

Table des Matières

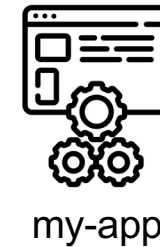
1. Introduction IAC
2. Première Infrastructure
3. Terraform State
4. Les variables
5. Les provisionneurs
6. Les modules
7. Les expressions

1.

Introduction

Avant l'automatisation

- Déployer une application
 - Installer les serveurs
 - Configurer les serveurs
 - Configurer le réseau
 - Installer les softs
 - Configurer les applications
 - Installer les bases de données
- Mise en œuvre manuelle
 - Ressources humaines (admins)
 - Risques d'erreurs importants
- Maintenances
- Multiples environnements



Automatisation avec l'IAC

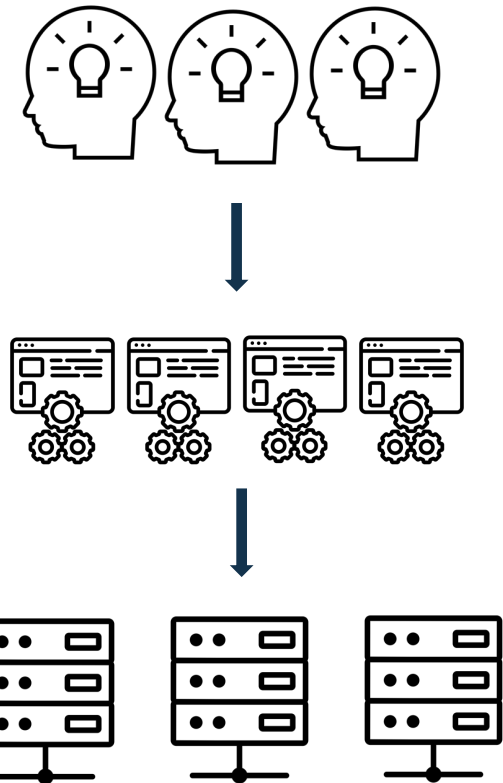
Automatiser toutes ces tâches

Connaissances et Expertises des administrateurs et exploitants converties en programmes

Ces programmes sont exécutés pour créer l'infrastructure

IAC est un concept

Utilise des outils

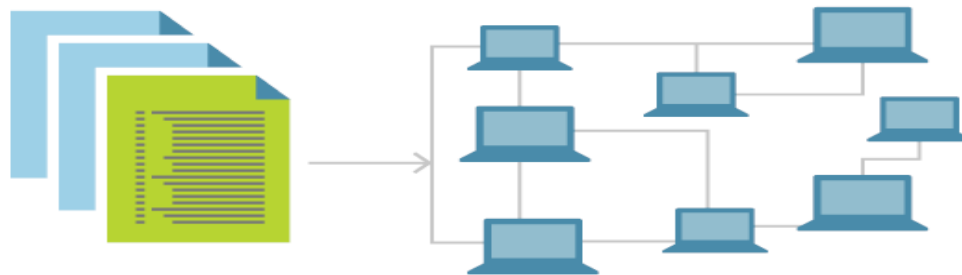


IAC définition

L'infrastructure en tant que code (IaC) est la gestion de l'infrastructure (réseaux, machines virtuelles, équilibreurs de charge et topologie de connexion des services gérés) dans un modèle descriptif, utilisant le contrôle de version pour stocker les fichiers.

Déployer une infrastructure via un logiciel pour obtenir un environnement prédictible et consistant

- Définir via du code
- Gérer avec des versions



Pourquoi tous ces outils?

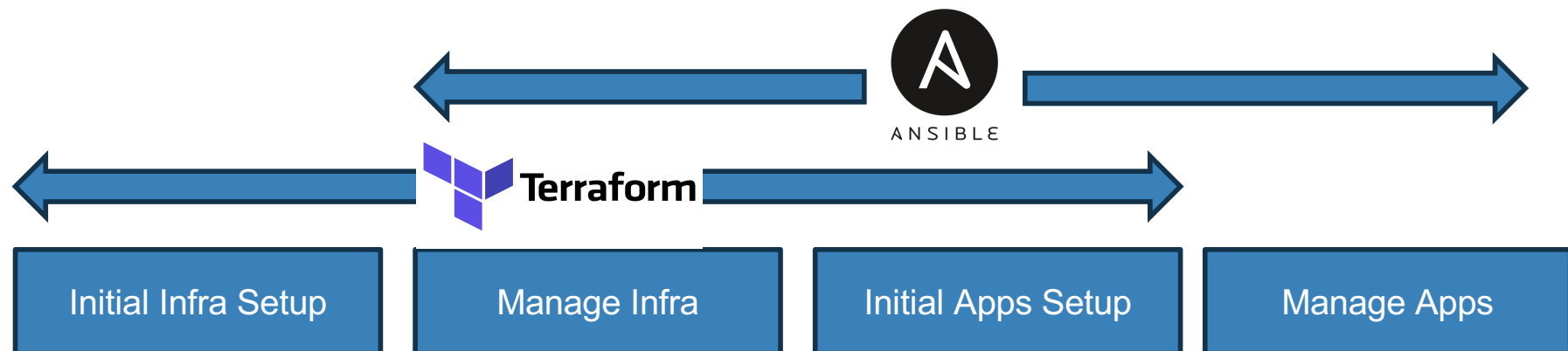
- Aucun outil ne peut tout faire tout du début à la fin
- Chaque outil est spécialisé dans son domaine

Catégories d'outils

- Le provisionnement de l'infrastructure
 - Créer de nouveaux serveurs
 - Configurer le réseau
- La configuration de l'infrastructure
 - Installer les logiciels
 - Installer tout le nécessaire pour déployer votre propre application (db ...)
- Le déploiement de votre application
 - Sur cette infrastructure provisionnée et configurée,

Phases

- Configuration initiale
 - Provisionner l'infrastructure
 - Configurer l'infrastructure
 - Installation des logiciels
 - Configuration initiale
- Maintenance
 - Mettre à jour de l'infrastructure
 - Ajouter ou supprimer des serveurs
 - Mise à jour des logiciels
 - Reconfiguration



Déclaratif vs Impératif

■ Déclaratif

- Une description des objets que l'on veut créer



■ Immuable

- Créer un nouvel objet avec des caractéristiques différentes et supprimer l'ancien objet

■ AgentLess

■ Impératif

- Une description des actions que le logiciel doit effectuer pour créer les objets



■ Mutable

- Modifier directement les caractéristiques de l'objet

■ Agent

IAC

Mutable Infrastructure

- Modifiable
- C'est le modèle classique
- Le legacy reste sur ce modèle

Immutable Infrastructure

- Non modifiable
- Un changement implique la création d'une nouvelle infrastructure (Création d'une nouvelle VM par exemple)
- Pattern d'architecture "cloud native"
- Les architectures applicatives doivent le prendre en compte tôt dans sa conception

Avantages de l'IAC

Les modifications sont donc poussées dans un outils de gestion de configuration (Git, SVN, etc.).

Réduit le drift et le snowflake

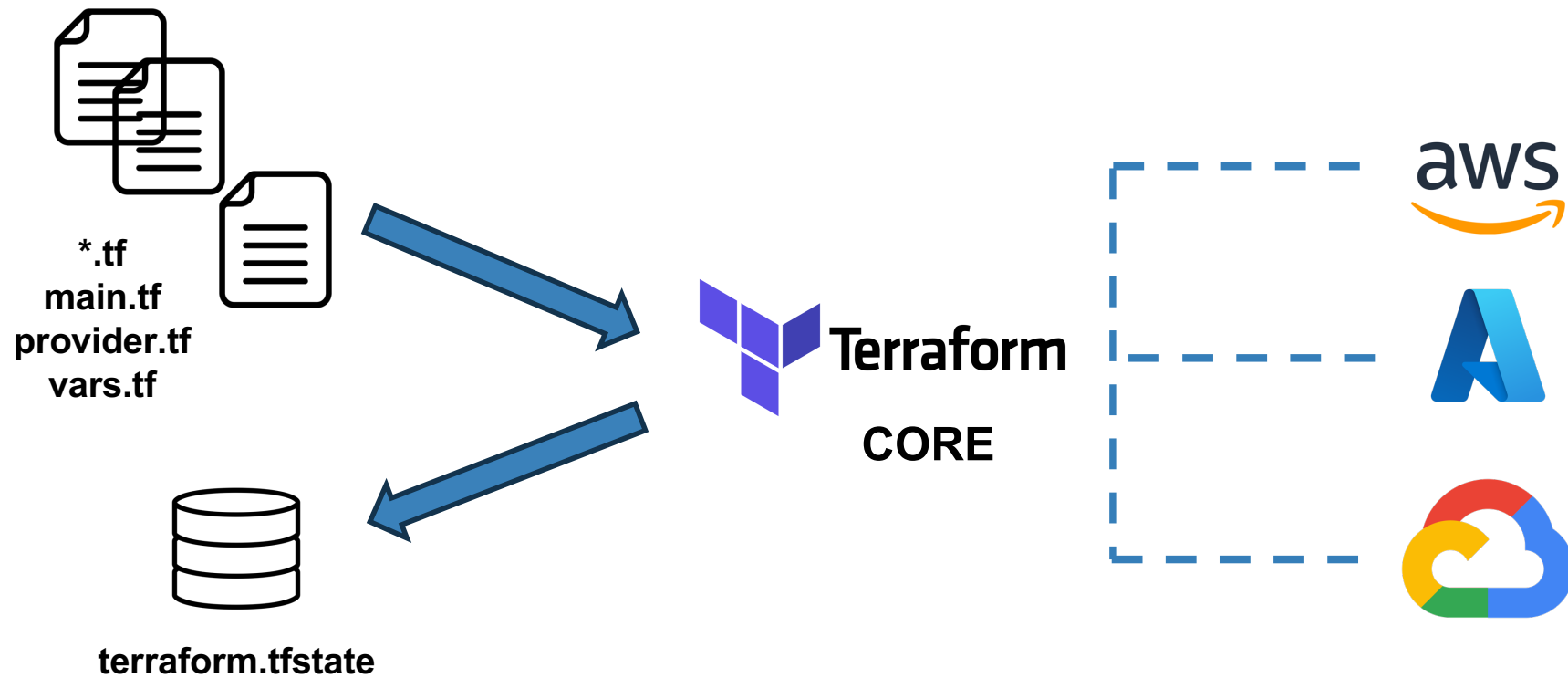
- Drift : Ecart entre la configuration de l'infrastructure que l'on peut provisionner facilement avec résultat de la maintenance normales de l'infrastructure
- Snowflake : Mise à jour des machines de l'infrastructure au "cas pas cas" (mauvaise pratique car souvent mal tracée)

Diminue donc l'erreur humaine

Qu'est ce que Terraform

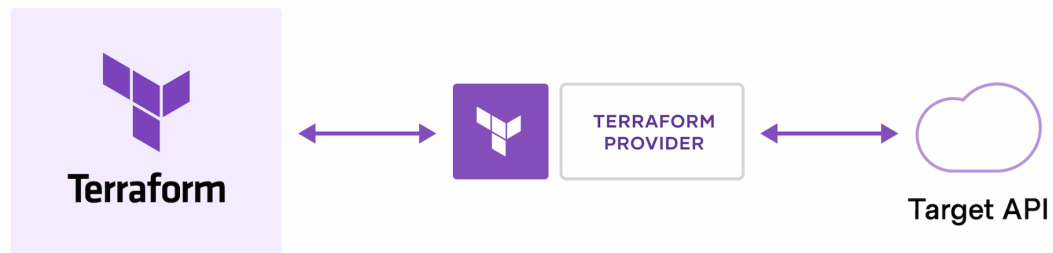
- Un outil déclaratif de provisioning basé sur le principe d' Infrastructure as a Code
- Utilise son propre langage - HCL (Hashicorp Configuration Language)

Architecture Terraform



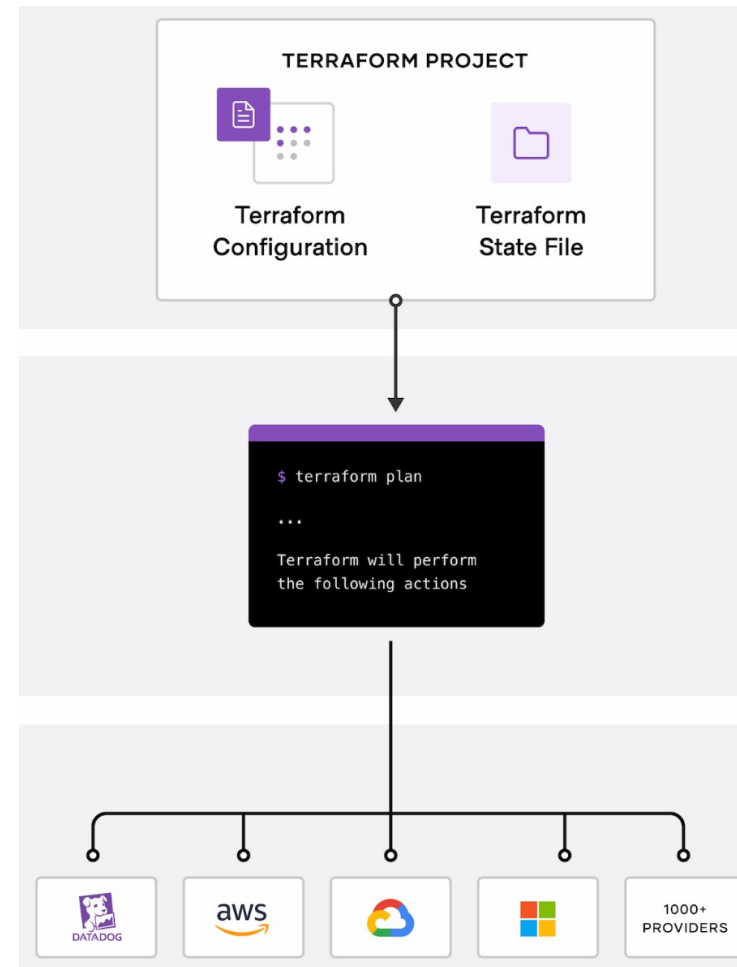
Architecture Terraform

- Terraform crée et gère des ressources sur des plates-formes cloud et d'autres services via leurs interfaces de programmation d'applications (API).
- Les fournisseurs permettent à Terraform de fonctionner avec pratiquement n'importe quelle plate-forme ou service avec une API accessible.



Architecture Terraform

- Write : Décrire les ressources à créer, qui peuvent se trouver sur plusieurs fournisseurs et services cloud.
- Plan: Terraform crée un plan d'exécution décrivant l'infrastructure qu'il va créer, mettre à jour ou détruire en fonction de l'infrastructure existante et de votre configuration
- Apply: après approbation, Terraform exécute les opérations proposées dans le bon ordre, en respectant toutes les dépendances de ressources



2.

Première Infrastructure

Providers

- Provider
 - Exposer des ressources pour une plate-forme d'infrastructure spécifique
 - Responsable de la compréhension de l'API de cette plate-forme.
 - Liste des providers <https://registry.terraform.io/browse/providers>



kubernetes

Providers : déclaration

```
$ cat provider.tf
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.8.0"
    }
  }
}
provider "aws" {
  region = "eu-west-1"
}
```

Providers : installation

```
$ terraform init
```

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

```
Initializing provider plugins...
```

- Finding hashicorp/aws versions matching "5.8.0"...
- Installing hashicorp/aws v5.8.0...
- Installed hashicorp/aws v5.8.0 (signed by HashiCorp)

```
Terraform has created a lock file .terraform.lock.hcl to  
record the provider  
selections it made above. you run "terraform init" in the  
future.
```

```
Terraform has been successfully initialized!
```

Providers : multi-cloud

```
$ cat providers.tf
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.8.0"
    }
    oci = {
      source = "oracle/oci"
      version = "5.4.0"
    }
  }
}
```

Providers : multi-cloud

```
$ cat main.tf
provider "aws" {
    region = "eu-central-1"
    access_key = "AKIA27KE26ZMHER3AZ6S"
    secret_key = "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"
}

provider "oci" {
    # Configuration options
}
```

Providers : installation

```
$ terraform init
```

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

```
Initializing provider plugins...
```

```
- Reusing previous version of hashicorp/aws from the  
dependency lock file
```

```
- Finding oracle/oci versions matching "5.4.0"...
```

```
- Using previously-installed hashicorp/aws v5.8.0
```

```
- Installing oracle/oci v5.4.0...
```

```
- Installed oracle/oci v5.4.0 (signed by a HashiCorp  
partner, key ID 1533A49284137CEB)
```

```
Terraform has made some changes to the provider dependency  
selections recorded
```

```
in the .terraform.lock.hcl file. Review those changes and  
commit them to your
```

```
Terraform has been successfully initialized!
```

Ressources exposées par l'API

- Services
 - Liste des services accessibles via l'API
- Ressources
 - Liste des ressources associées à un service

aws



EC2



aws_instance

Types de blocs

- Resource
 - Ressource que vous souhaitez créer
 - Les blocs sont composés de couples clé / valeur pour décrire les données d'initialisation d'une ressource.
- Variable
 - Définition des variables d'entrée d'un module
- Output
 - Données en sortie qui seront utilisées en entrée d'un autre module.
- Data
 - Collecte des données de ressources existantes en vue d'être utilisées ensuite

Déclarer une Ressource

- **Type de ressource** fournie par le provider
- **Nom de la ressource** équivalent à une variable

```
Resource "Fournisseur_Type" "nom_ressource" {  
    [argument = valeur]  
}
```

```
resource "aws_instance" "my_ec2" {  
    ami = "ami-0c960b947cbb2dd16"  
    instance_type = "t2.micro"  
    tags = {  
        Name = "First"  
    }  
}
```

Référencer une Ressource

`Fournisseur_Type.nom_ressource.attribut`

```
resource "aws_vpc" "vpc-dev" {
  cidr_block = "10.0.0.0/16"
}
resource "aws_subnet" "subnet-dev-1" {
  vpc_id      = aws_vpc.vpc-dev.id
  cidr_block = "10.0.1.0/24"
}
resource "aws_subnet" "subnet-dev-2" {
  vpc_id      = "${aws_vpc.vpc-dev.id}"
  cidr_block = "10.0.2.0/24"
}
```

Data source

- Une source de données représente une information
 - en lecture seule
 - extraite d'un fournisseur
 - données disponibles pour le reste de votre code Terraform.
 - elles sont très utiles pour obtenir des informations dynamiques à partir de l'API du provider.

```
data "Fournisseur_Type" "nom_ressource" {  
  [argument = valeur]  
}
```

```
data "aws_vpc" "vpc-default" {  
  default = true  
}
```

Référencer une Data

`data.Fournisseur_Type.nom_data.attribut`

```
data "aws_vpc" "default-pvc" {
  default = true
}

resource "aws_subnet" "subnet-dev-3" {
  vpc_id      = data.aws_vpc.default-pvc.id
  cidr_block = "10.0.3.0/24"
}
```

Data Filter

- Les sources de données qui renvoient des listes de ressources prennent en charge le filtrage.

```
data "Fournisseur_Type" "nom_ressource" {  
  filter {  
    name = ""  
    values = [""]  
  }  
}
```

```
data "aws_ami" "ubuntu-ami" {  
  most_recent = true  
  filter {  
    name      = "name"  
    values    = ["ubuntu/images/hvm-ssd/ubuntu-18.04-amd64*"]  
  }  
  owners = ["099720109477"] # Canonical  
}
```

3.

State

State

Fichier d'état est indispensable pour créer des plans et apporter ainsi des modifications à votre infrastructure.

En effet, lorsque vous exécutez la commande

- `terraform apply` pour la première fois, Terraform créera le nouveau fichier en local.
- modifications dans votre code
- `terraform apply` apportera des modifications sur ce même fichier pour ainsi tracer les changements à apporter sur votre infrastructure

State


terraform.tfstate

état courant

terraform.tfstate.backup

état précédent

```
$ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.5.3",
  "serial": 16,
  "lineage": "9d8b12b2-e311-0fad-abfa-6df34b0fc95f",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```



Pas de ressources

State Commands

Usage: terraform [global options] state <subcommand> [options] [args]

Subcommands:

list	List resources in the state
show	Show a resource in the stateResource

```
$ terraform state list  
data.aws_vpc.default-pvc  
aws_subnet.subnet-dev-1  
aws_subnet.subnet-dev-2  
aws_subnet.subnet-dev-3  
aws_vpc.vpc-dev
```

State Commands

```
$ terraform state show aws_subnet.subnet-dev-1
# aws_subnet.subnet-dev-1:
resource "aws_subnet" "subnet-dev-1" {
  arn                                = "arn:aws:ec2:eu-central-
1:754448004632:subnet/subnet-02a87dc6fe94c0c47"
  assign_ipv6_address_on_creation    = false
  availability_zone                  = "eu-central-1b"
  availability_zone_id               = "euc1-az3"
  cidr_block                         = "10.1.1.0/24"
  enable_dns64                       = false
  enable_lni_at_device_index         = 0
  enable_resource_name_dns_a_record_on_launch = false
  enable_resource_name_dns_aaaa_record_on_launch = false
  id                                 = "subnet-02a87dc6fe94c0c47"
  ipv6_native                        = false
  map_customer_owned_ip_on_launch    = false
  map_public_ip_on_launch            = false
  owner_id                           = "754448004632"
  private_dns_hostname_type_on_launch = "ip-name"
  tags                               = {}
  tags_all                           = {}
  vpc_id                             = "vpc-01635a048efdbbba4"
}
```

4.

Variables

Variables input

- Les Input variables sont généralement définies en indiquant
 - un nom
 - un type
 - une valeur par défaut
- Le type et les valeurs par défaut ne sont pas strictement nécessaires.
 - Terraform peut déduire le type de votre variable
 - Si la valeur n'est pas définie, la commande `terraform apply` va initier une saisie via le terminal

```
variable "aws_region" {  
    type = string  
    default = "eu-central-1"  
    description = "Région par défaut"  
}
```

Utilisation dans le code

- Utilisation simple

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

- Utilisation complexe

```
resource "aws_vpc" "vpc-dev" {  
  cidr_block = "10.1.0.0/16"  
  tags = {  
    Name = "vpc-${var.aws_region}"  
  }  
}
```

Ligne de commande

Vous pouvez définir des variables directement depuis la ligne de commande avec

- l'option -var

L'option -var est prioritaire

```
$ terraform apply -var 'aws_region=eu-west-3'
```

Le fichier de variables

- Fichier de variables reconnu par Terraform pour l'initialisation des variables

```
$ cat terraform.tfvars  
aws_region = "eu-west-3"
```


Le fichier de variables par environnement

Fichier de variables par environnement

```
$ cat terraform-dev.tfvars  
aws_region = "eu-west-3"  
environment = "development"
```

```
$ terraform apply -var-file terraform-dev.tfvars
```

Types de variables

Type string

```
variable "hello" {  
    type = string  
    default = "bonjour"  
}
```

Types de variables

- Type number

```
variable "instance_count" {  
    description = "Number of instances to provision."  
    type        = number  
    default     = 2  
}
```

- Référence

```
resource "aws_instance" "my_ec2_instance" {  
  
    instance_count = var.instance_count  
}
```

Types de variables

Type booleen

```
variable "aws_create" {  
  type = "bool"  
  default = true  
}
```

Types de variables

- Type liste

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

- Référence

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

Types de variables

- Type map

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

- Référence

```
resource "aws_instance" "my_ec2_instance" {  
  ami          = "${var.aws_amis["us-east-1"]}"  
}
```

Variables output

Les Output variables constituent un moyen pratique d'obtenir des informations utiles sur votre infrastructure.

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.

Considérées comme des valeurs de retour de fonctions

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

Variables d'environnement

- Terraform reconnaît les variables d'environnement avec le préfixe TF_VAR_

```
export TF_VAR_image_id='ami-abc123'
```

- Définition de la variable dans les fichiers *.tf

```
variable image_id {}
```

- Utilisation de la variable

```
resource "aws_instance" "my_ec2_instance" {  
  ami          = var.image_id  
}
```


5.

Provisionneurs

Provisioner

- Terraform permet de créer et gérer les infrastructures
- Permet d'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

user_data

- `user_data` : attribut de `aws_instance`
- Reconnu par tous les providers cloud
- Exécute des commandes après création et démarrage de l'instance
- C'est le provider (aws) qui exécute cette tâche

```
resource "aws_instance" "my_ec2" {  
  instance_type = "t2.micro"  
  user_data = <<-EOF  
    #!/bin/bash  
    sudo apt-get update  
    sudo apt-get install -y apache2  
  EOF  
}
```

user_data

`user_data` : attribut de `aws_instance`

Autre définition de l'attribut

```
$ cat install_apache.sh  
#!/bin/bash  
sudo apt-get update  
sudo apt-get install -y apache2
```

```
resource "aws_instance" "my_ec2" {  
  instance_type = "t2.micro"  
  user_data = "${file("install_apache.sh")}"  
}
```

Provisioner local-exec

- Le provisionneur local-exec appelle un exécutable sur le poste local après la création d'une ressource.
 - `working_dir` : spécifie le répertoire de travail où la commande sera exécutée.
 - `interpreter` : liste d'arguments d'interpréteurs utilisés pour exécuter votre commande. (Ex : `interpreter = ["perl", "-e"]`)

```
resource "aws_instance" "my_ec2" {  
  ami          = "ami-06ffade19910cbfc0"  
  instance_type = "t2.micro"  
}  
provisioner "local-exec" {  
  command = "echo ${aws_instance.my_ec2.public_ip} > ip_addr.txt"  
}
```

Provisioner remote-exec

- Le provisionneur `remote-exec` permet d'exécuter un script sur une ressource distante une fois qu'elle est créée.
- Il faut définir la méthode de connexion dans un bloc `connection`
- A l'inverse de `user_data`, c'est Terraform qui exécute cette tâche

```
resource "aws_instance" "my_ec2" {  
  connection {  
    type = "ssh"  
  }  
  provisioner "remote-exec" {  
    inline = [  
      "sudo apt-get -y update",  
      "sudo apt-get install -y apache2"  
    ]  
  }  
}
```

Provisioner file

- Le provisioner `file` copie les fichiers ou les répertoires de la machine exécutant Terraform vers la ressource nouvellement créée. Connexion par `ssh` ou `winrm`.
- Les arguments :
 - `source`: fichier local à copier
 - `content`: données à copier
 - `destination`: dossier ou fichier de l'instance créée

```
provisioner "file" {  
  source      = "init-script.sh.m"      # terraform machine  
  destination = "/tmp/init-script.sh"   # remote machine  
}  
provisioner "remote-exec" {  
  script = file("/tmp/init-script.sh")  
}
```

Définition de la connexion

- Globale: donc connue de tous les provisioners
- Locale: connue uniquement par le provisioner

```
resource "aws_instance" "my_ec2" {  
  connection {  
    type = "ssh"  
  }  
  provisioner "remote-exec" {  
    inline = ["sudo apt-get -y update"]  
  }  
  provisioner "file" {  
    connection {  
      type = "winrm"  
      host = autre_serveur.public_ip  
    }  
  }  
}
```


Provisioner de destruction

- On peut modifier le comportement d'un provisioner
- Les arguments :
 - `when=destroy` : exécution du provisioner à la destruction de l'instance
 - `on_failure=continue` : poursuit la commande `apply`

```
provisioner "local-exec" {  
  when          = destroy  
  on_failure    = continue  
  command = "echo destruction > output.txt"  
}
```

Taint

- Si un provisionneur au moment de la création d'une ressource échoue, la ressource est marquée comme "Taint »
- Une ressource contaminée sera planifiée pour être détruite et recréée lors de la prochaine exécution de la commande `terraform apply`.

```
$ terraform state list
```

```
aws_instance.my_ec2
```

```
$ terraform taint aws_instance.my_ec2
```

```
Resource instance aws_instance.my_ec2 has been marked as  
tainted.
```

```
$ terraform plan
```

```
Terraform will perform the following actions:
```

```
  # aws_instance.my_ec2 is tainted, so must be replaced  
-/+ resource "aws_instance" "my_ec2" {
```

6.

Modules

Module définition

- Les modules sont des conteneurs pour plusieurs ressources qui sont utilisées ensemble.
- Un module consiste en une collection de fichiers .tf conservés ensemble dans un répertoire.
- Les modules sont le principal moyen de regrouper et de réutiliser les configurations de ressources avec Terraform.
- Avantages:
 - Organiser et regrouper les configurations
 - Code Terraform réutilisable, facilement maintenable.

Module exemple

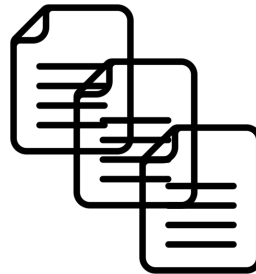
Un module est un dossier contenant les fichiers .tf nécessaires pour créer une instance d'un certain type

Exemple : un serveur web

Constitué :

- une instance ec2
- des paires de clés ssh
- un groupe de sécurité

***.tf**
main.tf
provider.tf
vars.tf



Module input/output

- Comme une fonction, un module peut recevoir des paramètres en entrée et fournir des résultats en sorties



Modules

- Modules créés par vous
- Modules créées par Terraform ou d'autres éditeurs

<https://registry.terraform.io/browse/modules>



terraform-aws-modules / iam

Terraform module which creates IAM resources on AWS



terraform-aws-modules / vpc

Terraform module which creates VPC resources on AWS

Provision Instructions

Copy and paste into your Terraform configuration, insert the variables, and run `terraform init`:

```
module "vpc" {  
  source = "terraform-aws-modules/v  
  version = "5.1.0"  
}
```

Structure d'un module

- `main.tf` : uniquement les déclarations des ressources
- `variables.tf` : définition des variables
- `outputs.tf` : les valeurs de sorties pouvant être utilisées par le module appelant

```
.  
├── main.tf  
├── outputs.tf  
└── variables.tf
```


Le module racine

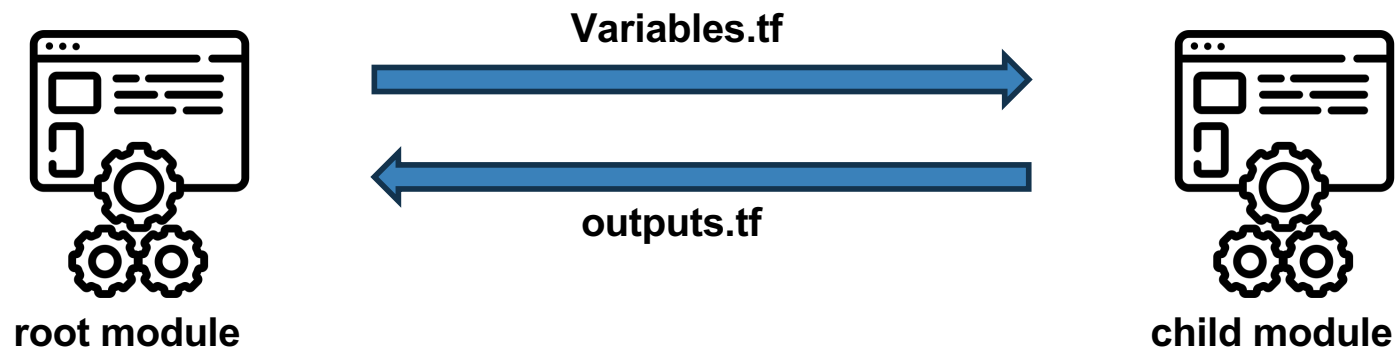
- Le premier module, du dossier de votre projet
- `terraform.tfvars` : valeurs des variables associées à votre projet
- `modules`: dossier des modules

```
project/
├── main.tf
├── modules
│   └── module
│       ├── main.tf
│       ├── outputs.tf
│       └── variables.tf
├── outputs.tf
├── providers.tf
├── README.md
├── terraform.tfvars
└── variables.tf
```

Appel du Module

- A partir du module racine (root module)

```
module "ec2_instance" {  
  source = "../modules/ec2_instance"  
}
```



Variables de Module

- Un module est appelé comme une fonction :
 - Input variables: paramètres d'appel de la fonction x et y
 - Output variables: valeurs de retour de la fonction z

```
z=func (x, y)
```

Variables de Module

- Appel avec Input variables

```
module "ec2_instance" {  
  source = "../modules/ec2_instance"  
  instance_type = "t2.micro" #paramètre initialisé directement  
  instance_type = var.instance_type #initialisé par variables  
}
```

- Input variables du module Root :
 - Définies dans variables.tf
 - Initialisées dans terraform.tfvars
 - Valeurs passées au module appelé comme des arguments

Variables de Module

Utilisation Output variables du Child Module dans le Root Module

```
module "app_subnet" {  
  source = "../modules/app_subnet"  
}  
  
resource "aws_instance" "my_app" {  
  subnet_id = module.app_subnet.subnet.id  
}
```

`module.app_subnet.subnet.id`

`app_subnet:` Nom du Child module

`subnet:` Nom de la variable output du Child module

Variables de Module

- Ouput variables du Root Module

```
$ cat output.ft  
output "public_ip" {  
    value      = module.ec2_instance.public_ip  
}
```

- Ouput variables du Child Module

```
$ cat modules/ec2_instance/outputs.tf  
output "public_ip" {  
    value = aws_instance.web_server[*].public_ip  
}
```

Modules registry

- Les modules sont paratagés par HashiCorp ou d'autres éditeurs

<https://registry.terraform.io>

- Associés aux providers

The screenshot displays the Terraform Registry interface. The top navigation bar includes the Terraform logo, 'Registry', a search bar, and links for 'Browse', 'Publish', and 'Sign-in'. Below the navigation bar, there are tabs for 'Providers', 'Modules', 'Policy Libraries', and 'Run Tasks'. The 'Modules' tab is selected.

Filters [Clear Filters](#)

Tier

- ☐ Partner
Display only modules that are maintained by a Hashicorp partner.

Provider

- ☐ Alibaba
- ☒ aws Aws
- ☐ Azure
- ☐ Boundary
- ☐ Consul
- ☐ Google
- ☐ Helm
- ☐ Nomad

Modules

Modules are self-contained packages of Terraform configurations that are ma

- terraform-aws-modules / vpc**
Terraform module which creates VPC resources on AWS 🇺🇸
4 days ago 47.0M
- terraform-aws-modules / iam**
Terraform module which creates IAM resources on AWS 🇺🇸
15 days ago 46.3M

Modules registry

Les modules sont paratagés
par HashiCorp ou d'autres
éditeurs

<https://registry.terraform.io>

Associés aux providers

The screenshot displays the Terraform Registry interface. The top navigation bar includes the Terraform logo, a search bar, and links for Browse, Publish, and Sign-in. Below the navigation bar, there are tabs for Providers, Modules, Policy Libraries, and Run Tasks. The left sidebar contains a 'Filters' section with a 'Clear Filters' link. Under the 'Tier' filter, the 'Partner' checkbox is unchecked, with a description: 'Display only modules that are maintained by a Hashicorp partner.' Under the 'Provider' filter, the 'aws' checkbox is checked, while others like Alibaba, Azure, Boundary, Consul, Google, Helm, and Nomad are unchecked. The main content area is titled 'Modules' and contains a description: 'Modules are self-contained packages of Terraform configurations that are ma'. Below this, there are two module listings. The first is 'terraform-aws-modules / vpc', described as 'Terraform module which creates VPC resources on AWS', published 4 days ago with 47.0M downloads. The second is 'terraform-aws-modules / iam', described as 'Terraform module which creates IAM resources on AWS', published 15 days ago with 46.3M downloads.

Filters [Clear Filters](#)

Tier

☐ Partner
Display only modules that are maintained by a Hashicorp partner.

Provider

☐ Alibaba

☒ aws Aws

☐ Azure

☐ Boundary

☐ Consul

☐ Google

☐ Helm

☐ Nomad

Modules

Modules are self-contained packages of Terraform configurations that are ma

terraform-aws-modules / vpc
Terraform module which creates VPC resources on AWS

4 days ago 47.0M

terraform-aws-modules / iam
Terraform module which creates IAM resources on AWS

15 days ago 46.3M

7.

Les expressions

Expressions

- Terraform reste avant tout un langage déclaratif
- Il fournit des Expressions pour :
 - faire référence
 - calculer des valeurs
 - des conditions
 - des boucles
 - des opérateurs arithmétique
 - des fonctions intégrées.

Expression conditionnelle

`<CONDITION> ? <TRUE_RESULT> : <FALSE_RESULT>`

Les deux valeurs de résultat peuvent être de n'importe quel type, mais elles doivent toutes deux être du même type afin que Terraform puisse déterminer le type que l'expression conditionnelle entière renverra sans connaître la valeur de la condition.

```
var.a != "" ? var.a : "default-a"
```

Boucles avec count

- Le paramètre `count` permet de mettre à l'échelle les ressources
- `count.index` : le numéro d'index du tableau (liste) des ressources

```
resource "aws_instance" "my_ec2_instance" {  
  count = 2  
  tags = {  
    Name = "server-${count.index}"  
  }  
}
```

```
+ create  
# aws_instance.my_ec2_instance[0] will be created  
# aws_instance.my_ec2_instance[1] will be created
```

Variables et fonctions

La fonction `length` retourne le nombre d'éléments d'une liste

```
variable "dev-users" {  
    type = list  
    default = ["Anne", "Paul", "Marie"]  
}  
  
resource "aws_iam_user" "dev_user" {  
    count = length(var.dev-users)  
    name  = var.dev-users[count.index]  
}
```

Boucles avec `for_each`

- Permet de créer plusieurs instances de la même ressource
- Prend en entrée une variable de type `map`
- Utilise la clé de la `map` comme index des instances de la ressource
- S'utilise avec des variables de type `map`
- Il faut convertir les variables de type `liste` en type `map` à l'aide de la fonction `toiset()`

Boucles avec for_each

```
variable "dev-users" {  
    type = list  
    default = ["Anne", "Paul", "Marie"]  
}  
resource "aws_iam_user" "dev_users" {  
    for_each = toset(var.dev-users)  
    name     = each.value  
    tags = {  
        key = each.key  
        value = each.value  
    }  
}
```

Boucles avec for

```
[for <KEY>, <VALUE> in <MAP> : <OUTPUT>]
```

- Permet de boucler sur une map
 - MAP est la variable à parcourir
 - KEY et VALUE sont respectivement la clé et la valeur de chaque élément de la map
 - OUTPUT est le résultat final retourné par chaque itération de la boucle

Boucles avec for

```
variable "role_users" {  
  type = map(string)  
  default = {  
    "Anne" = "dev"  
    "Paul" = "admin"  
    "Marie" = "manager"  
  }  
}  
output "users" {  
  value = [for name, role in var.role_users : "${name} est ${role}"]  
}
```

8.

Backend & Workspace

Backend

- Un backend définit où Terraform stocke ses fichiers de données d'état.
- Terraform utilise des données d'état persistantes pour suivre les ressources qu'il gère
- Ce fichier d'état est donc indispensable pour créer des plans et apporter ainsi des modifications à votre infrastructure
- Types de Backend :
 - `local`: par défaut
 - `consul`: service discovery
 - `http`: stocke l'état à l'aide d'un client REST.
 - `s3`: stocke l'état dans un bucket Amazon S3

Configuration

- Configuration par le bloc `terraform`

```
terraform {  
  backend "<type>" {  
    [config]  
  }  
}
```

Backend s3

- Configuration par le bloc terraform

```
terraform {  
  backend "s3" {  
    bucket = "mybucket"  
    key    = "path/to/my/key"  
    region = "us-east-1"  
  }  
}
```

Workspace

- Les workspaces de Terraform sont des espaces de travail où sont stockées les states
- Par défaut une seul workspace existe (default)

```
$ terraform workspace list  
* default
```

Chaque workspace possède son propre state

```
resource "aws_instance" "my_ec2" {  
  tags = {  
    Name = "my_ec2-${terraform.workspace}"  
  }  
}
```

Workspace use case

- Inconvénients de l'approche Dossier par Environnement
 - Terraform installe un cache séparé de plugins et de modules pour chaque répertoire de travail, donc le maintien, upgrade des versions de modules pour chaque répertoires
- Les espaces de travail sont pratiques car ils vous permettent de créer différents ensembles d'infrastructures avec la même copie de travail de votre configuration et les mêmes caches de plugins et de modules.
- Une utilisation courante des espaces de travail multiples consiste à créer une copie parallèle et distincte d'un ensemble d'infrastructures pour tester un ensemble de modifications avant de modifier l'infrastructure de production.

Workspace

Créer un workspace

```
$ terraform workspace new prod
```

```
Created and switched to workspace "prod"!
```

```
You're now on a new, empty workspace. Workspaces isolate their  
state,  
so if you run "terraform plan" Terraform will not see any  
existing state  
for this configuration.
```


Workspace

- Lister le workspace courant

```
$ terraform workspace show  
prod
```

Lister tous les workspaces

```
$ terraform workspace list  
default  
* prod
```

```
$ terraform workspace select default  
Switched to workspace "default".
```

