


Terraform (logiciel)

🌐 10 langues

[Article](#) [Discussion](#)[Lire](#) [Modifier](#) [Modifier le code](#) [Voir l'historique](#) Pour les articles homonymes, voir [Terraform](#).

Terraform est un environnement logiciel d'« infrastructure as code » publié en open-source par la société **HashiCorp**. Cet outil permet d'automatiser la construction des ressources d'une infrastructure de centre de données comme un réseau, des machines virtuelles, un groupe de sécurité ou une base de données.

L'infrastructure est décrite sous forme du langage de configuration **Hashicorp Configuration Language** (HCL). Il est aussi possible d'utiliser le langage **JSON**.

Terraform permet notamment de définir des topologies cloud pour les principaux fournisseurs d'infrastructure cloud, tels qu'Amazon Web Services, IBM Cloud (anciennement Bluemix), Google Cloud Platform, Linode^{3,4}, Microsoft Azure, Oracle Cloud Infrastructure, OVHcloud^{5,6} ou VMware, vSphere ainsi que OpenStack^{7,8,9,10,11,12}.

Les ressources décrites dans le code HCL Terraform sont dépendantes du fournisseur (« provider ») de l'infrastructure cloud. Par exemple, une ressource Terraform définie pour une topologie Amazon ne peut pas être réutilisée pour une topologie OpenStack ou Microsoft Azure puisqu'elle n'ont pas les mêmes propriétés.

Terraform

 **Terraform**

Informations

Développé par	HashiCorp (en) 
Dernière version	1.3.7 (4 janvier 2023) ¹ 
Dépôt	github.com/hashicorp/terraform 
Écrit en	Go 
Système d'exploitation	Linux, macOS et Microsoft Windows 
Licence	MPL-2.0 
Site web	www.terraform.io 

IAC

- Mise en service des ressources à l'aide du code
- Créer un script Shell/Python pour la création d'une machine virtuelle
- Mais écrire / maintenir un tel code est une tâche fastidieuse

Créer N/W

Attendez que l'étape ci-dessus termine Provisionner le sous-réseau

Créer une règle de pare-feu

Attendez la fin de l'étape ci-dessus

Instance de moteur de calcul avec tous les paramètres

```
resource "google_compute_instance"
"first-instance"{
  name = "hello-1"
  zone = "us-central1-a"
  machine_type = "n1-standard-1"
  boot_disk {
    initialize_params {
      image = "debian-
cloud/debian-9"
    }
  }
  network_interface {
    network = "default"
  }
}
```

Terraform

- Terraform est l'un des outils les plus populaires pour le provisionnement d'infrastructures
- Gratuit – Open source
- Développé par HashiCorp
- Prise en main rapide et facile avec un seul fichier binaire
- Master HCL – terraform en peu de temps
- Terraform a plusieurs fournisseurs sont disponibles



- Terraform a plusieurs fournisseurs sont disponibles.
- Outre le cloud public, de nombreux autres fournisseurs sont disponibles pour réseau, DNS, pare-feu, base de données
- Écrire la configuration dans HCL/JSON.
- HCL est préférable.
- Terraform est un outil sans agent
- Ce n'est pas un outil de configuration. Travaillez bien avec Ansible.



outil natif

- Outil Cloud Native disponible pour le provisionnement de l'infrastructure
- Azure – Template
- Google – Deployment manager
- AWS - Cloud Formation
- JSON/Yaml
- Terraform est cloud agnostic.
- Avec plusieurs fournisseurs, les ressources peuvent être provisionnées pour plusieurs clouds.



Terraform Installation

- Disponible pour tous les principaux systèmes d'exploitation :
- Visité : <https://www.terraform.io/downloads.html>
- Télécharger Binary
- Décompressez-le.
- Export Path variable
 - Windows – verba
 - export PATH=\$PATH:path to terraform binary
- Vérifier avec la version de Terraform
- Éditeur - Libre d'utiliser l'un de vos éditeurs préférés



Terraform workflow



Portée et auteur

- Identifier les ressources à provisionner
- Créer un fichier local – exemple.txt avec du contenu
- Écrire le fichier de configuration pour cela en langage HCL

The screenshot shows a code editor with a file named `main.tf`. The code defines a resource block:

```
1 resource local_file sample_res {
2   filename = "sample.txt"
3   content  = "I love Terraform"
4 }
```

Annotations with callouts point to specific parts of the code:

- Block**: Points to the `resource` keyword.
- Resource Type - Local**: Points to `local_file`.
- Resource Name**: Points to `sample_res`.
- Arguments**: Points to the `filename` and `content` attributes.
- .tf Extension**: Points to the `.tf` extension of the file name.

A sidebar on the right shows a file explorer with `local provider`, `Resources` (containing `local_file`), and `Data Sources`.

Créer le premier fichier Terraform

init, Plan & apply

- init
 - Première commande après l'écriture des fichiers de configuration
 - Initialiser un répertoire de travail
 - Download plugin
 - `local_file`
 - `random`
- plan
 - Crée un plan d'exécution
 - Ne modifie aucune infrastructure
- apply
 - Exécuter toutes les modifications et provisionner les ressources spécifiées dans les fichiers de configuration

Variables

Main.tf

```
resource local_file sample_res {  
  filename = "sample.txt"  
  content = "I love Terraform"  
}
```



```
resource local_file sample_res {  
  filename = var.filename  
  content = var.content  
}
```

variables.tf

```
variable filename {  
  type      = string  
  default   = "sample.txt"  
}  
  
variable content {  
  type      = string  
  default   = "I Love Terraform"  
}
```

Fichier de définition de variable

- terraform.tfvars
- terraform.tfvars.json
- *.auto.tfvars
- *.auto.tfvars.json

Dépendance explicite

```
resource "local_file" "rand_res" {  
  filename = "explicit.txt"  
  content = "I love terraform "  
}
```

```
resource "random_string" "rand_name" {  
  length = 20  
}
```

```
resource "local_file" "rand_res" {  
  filename = "implicit.txt"  
  content = "I love random text ${random_string.rand_name.id}"  
  depend_on = [random_string.rand_name]  
}
```

Output


```
resource "random_string" "rand_name" {  
  length = 20  
}
```

```
output name {  
  value = random_string.rand_name.id  
}
```

output.tf

Règles de cycle de vie

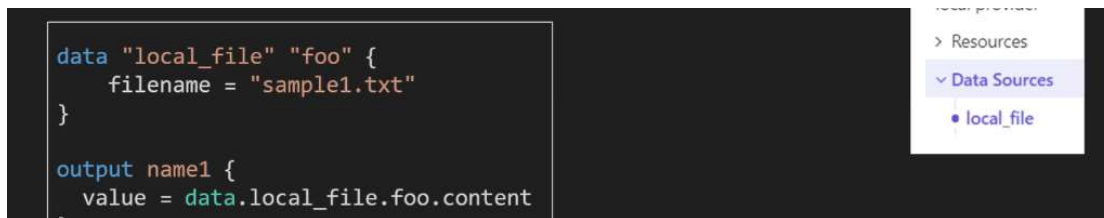
- lifecycle – attributs de ressource
- create before destroy - Créez d'abord la ressource, puis détruisez l'ancienne
- prevent destroy - Empêche la destruction d'une ressource
- ignore changes - Ignorer les modifications apportées à la ressource – Balise spécifique ou toutes

Provider version

```
terraform {  
  required_providers {  
    local = {  
      source = "hashicorp/local"  
      version = "2.1.0"  
    }  
  }  
}  
  
provider "local" {  
  # Configuration options  
}
```

Source des données

- local_file reads a file from the local filesystem.



Installation CLI AWS :

https://docs.aws.amazon.com/fr_fr/cli/latest/userguide/getting-started-install.html

```
C:\Users\Administrateur>aws --version
aws-cli/2.10.1 Python/3.9.11 Windows/10 exe/AMD64 prompt/off
```

Après l'installation, il faut configurer notre AWS CLI :

https://docs.aws.amazon.com/fr_fr/cli/latest/userguide/cli-configure-quickstart.html

Il faut dans un 1er temps créer une clé d'accès :

Interface de ligne de commande (CLI)
 Vous prévoyez d'utiliser cette clé d'accès pour permettre à AWS CLI d'accéder à votre compte AWS.

```
C:\Users\Administrateur>aws configure
AWS Access Key ID [None]: AKIAZKAF3QXWEQS5FK6D
AWS Secret Access Key [*****qTQa]: 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
Default region name [eu-west-1]:
Default output format [json]:
```

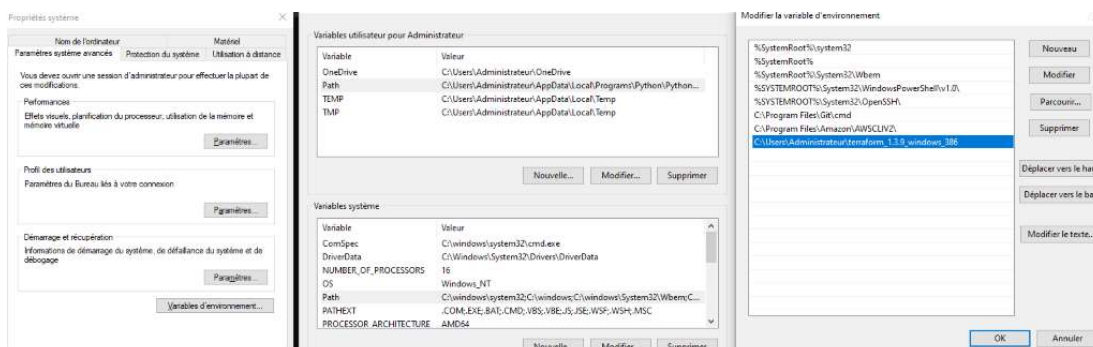
Pour voir la liste de nos users sur AWS depuis le CLI : `aws iam list-users`

```
C:\Users\Administrateur>aws iam list-users
{
  "Users": [
    {
      "Path": "/",
      "UserName": "adnane",
      "UserId": "AIDAZKAF3QXWKYDDX02EY",
      "Arn": "arn:aws:iam::639962416620:user/adnane",
      "CreateDate": "2023-02-15T09:16:40+00:00",
      "PasswordLastUsed": "2023-02-22T08:40:45+00:00"
    }
  ]
}
```

Installer Terraform sous windows :

<https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>

<https://phoenixnap.com/kb/how-to-install-terraform>



```
C:\Users\Administrateur>terraform -version
Terraform v1.3.9
on windows_386
```

Une fois **AWS CLI** et **Terraform** installé : nous pouvons déployer notre infrastructure avec **Terraform** sur **AWS** via notre configuration **AWS CLI**

Création d'un fichier de test :

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

Pour que Terraform sache où aller : ces deux blocs sont obligatoire :

```
main.tf > provider "aws"
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 4.0"
6     }
7   }
8 }
9
10 # Configure the AWS Provider
11 provider "aws" {
12   profile = "default"
13   region = "eu-west-1"
14 }
```

Créer une instance :

```
resource "aws_instance" "web" {
  ami           = "ami-0dfcb1ef8550277af"
  instance_type = "t2.micro"

  tags = {
    Name = "HelloWorld"
  }
}
```

Pour initialiser notre ec2 : nous devons taper 3 commandes :

Commande : Terraform init

```
C:\Users\Administrateur\Documents\Infrastructure as Code\Terraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.0"...
- Installing hashicorp/aws v4.55.0...
- Installed hashicorp/aws v4.55.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
```

any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

Commande : Terraform plan

```
C:\Users\Administrateur\Documents\Infrastructure as Code\Terraform>terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
+ create

Terraform will perform the following actions:
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
```

Commande : Terraform apply

```
C:\Users\Administrateur\Documents\Infrastructure as Code\Terraform>terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
+ create

Terraform will perform the following actions:
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
```

```
Enter a value: yes|
```

```
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 31s [id=i-0158328edf32a1d34]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

<input type="checkbox"/> HelloWorld	i-0158328edf32a1d34	En cours d'exé	t2.micro
-------------------------------------	---------------------	----------------	----------

On remarque que Terraform nous a généré des fichiers :

```
✓ TERRAFORM
> .terraform
≡ .terraform.lock.hcl
main.tf
{} terraform.tfstate
≡ terraform.tfstate.backup
```

Rangement des blocs de connexions dans le fichier **provider.tf** pour que ces éléments soient dans un fichier séparé des autres


```

main.tf  variables.tf  provider.tf X
provider.tf > ...
1  terraform {
2    required_providers {
3      aws = {
4        source = "hashicorp/aws"
5        version = "~> 4.0"
6      }
7    }
8  }
9
10 # Configure the AWS Provider
11 provider "aws" {
12   profile = "default"
13   region = "eu-west-1"
14 }
15

```

Création fichier **variables.tf** et création d'une variable **instance_type** :

```

main.tf  variables.tf X  provider.tf
variables.tf > variable "instance_type" > type
1  variable "instance_type" {
2    default = "t2.micro"
3    description = "type pour machine web"
4    type = string
5  }
6

```

Modification du bloc ressource pour utiliser notre variable nouvellement créer :

```

main.tf X  variables.tf  provider.tf
main.tf > resource "aws_instance" "web" > instance_type
1  resource "aws_instance" "web" {
2    ami = "ami-06e0ce9d3339cb039"
3    instance_type = var.instance_type
4
5    tags = {
6      Name = "HelloWorld"
7    }
8  }

```

Rajout d'un **output** dans le fichier main pour récupérer l'adresse ip publique de notre machine **ec2**

```

main.tf > output "ec2_instance_public_ip" > value
1  resource "aws_instance" "web" {
2    ami = "ami-06e0ce9d3339cb039"
3    instance_type = var.instance_type
4
5    tags = {
6      Name = "paris-magique"
7    }
8  }
9
10 output "ec2_instance_public_ip" {
11   description = "afficher adresse IP"

```

```

12     description = "url: https://aws.amazon.com/ec2/instance-types/"
13     value = aws_instance.web.public_ip
14 }

```

On détruit notre instance avec **terraform destroy**

```

C:\Users\Administrateur\Documents\Infrastructure as Code\Terraform>terraform destroy
aws_instance.web: Refreshing state... [id=i-0158328edf32a1d34]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy

Terraform will perform the following actions:

# aws_instance.web will be destroyed

```

```

Enter a value: yes

aws_instance.web: Destroying... [id=i-0158328edf32a1d34]
aws_instance.web: Still destroying... [id=i-0158328edf32a1d34, 10s elapsed]
aws_instance.web: Still destroying... [id=i-0158328edf32a1d34, 20s elapsed]
aws_instance.web: Destruction complete after 30s

Destroy complete! Resources: 1 destroyed.

```

Et je refais mes étapes pour déployer mon instance : **init | deploy | apply**

Avec ma commande "output", au moment du **deploy**, il me génère les changements avec l'output

```

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ ec2_instance_public_ip = (known after apply)

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.

```

Après le apply : il m'affiche son adresse publique grâce à la balise **output**

```

Enter a value: yes

aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Still creating... [40s elapsed]
aws_instance.web: Still creating... [50s elapsed]
aws_instance.web: Still creating... [1m0s elapsed]
aws_instance.web: Still creating... [1m10s elapsed]
aws_instance.web: Creation complete after 1m12s [id=i-02b1546b58f5e0625]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:
ec2_instance_public_ip = "3.253.13.221"

```

