



Introduction

Les systèmes logiciels deviennent de plus en plus sophistiqués et complexes, ce qui nécessite une approche innovante des technologies. Selon le [rapport sur l'état de l'infrastructure informatique](#) à partir de 2020, les trois quarts des entreprises devront mettre à niveau leur infrastructure informatique pour utiliser les nouvelles technologies. De nombreuses entreprises emploient encore du travail manuel lors de la mise en place, de l'exploitation et de la gestion de leurs infrastructures. Cette tendance entraîne des retards, des erreurs, des incohérences et des divergences. De plus, les organisations gaspillent des talents dans des tâches banales et répétitives alors qu'elles pourraient créer de nouveaux outils et contribuer aux innovations de l'entreprise. De plus, le travail manuel équivaut à des coûts élevés et à des frais généraux de gestion. L'infrastructure en tant que code peut vous aider à résoudre ces problèmes et à assurer l'automatisation et l'optimisation. Dans cet article, nous répondrons à la question de savoir ce qu'est l'infrastructure en tant que code et explorerons ses avantages, ses meilleures pratiques et ses principaux outils.

I. Qu'est-ce qui ne va pas avec l'infrastructure traditionnelle ?

Les processus traditionnels de gestion de l'infrastructure informatique sont manuels, ce qui entraîne inévitablement des erreurs humaines et des coûts plus élevés. Les gestionnaires d'infrastructure configurent manuellement les serveurs pour répondre aux exigences du système d'exploitation et des applications avant de déployer l'application. Le premier inconvénient de l'approche traditionnelle est le coût et les ressources, car une entreprise doit embaucher des professionnels capables d'effectuer ces tâches fastidieuses mais complexes. Ces processus nécessitent pas mal d'experts ainsi que la direction pour contrôler les opérations. Cela crée à son tour des frais généraux et complique la communication au sein de l'entreprise.

De plus, la configuration manuelle est beaucoup plus lente que les processus automatisés, ce qui entraîne des retards et plus tard une indisponibilité de l'application. Les organisations perdent l'évolutivité dont elles ont tant besoin, en particulier pendant les pics, et risquent de maintenir leurs applications indisponibles pendant de plus longues périodes. De plus, chaque fois que vous rencontrez un problème lié à l'infrastructure, il devient extrêmement difficile d'en identifier la cause. Sans une surveillance et des rapports continus fournis par les outils IAC, les professionnels peuvent passer beaucoup de temps à essayer de trouver une cause profonde. Cela conduira éventuellement à un taux de désabonnement et à une mauvaise réputation parmi les utilisateurs.

- Gestion des Infrastructures informatiques manuelle entraînant des erreurs et la perte de temps
- Gestion d'infrastructures informatiques coûteuses et nécessitant beaucoup de ressources.
- Difficile d'identifier la cause en cas de problème lié à l'infarctus
- Impossible d'automatiser une ressource.

- **Definition “Infrastructure as Code”**

Infrastructure as Code (IaC) est une terminologie répandue parmi **DevOps** professionnels. Il s'agit du processus de gestion et de provisionnement de l'infrastructure informatique complète (comprenant des machines physiques et virtuelles) à l'aide de fichiers de définition lisibles par machine. C'est une approche de génie logiciel vers les opérations. Il aide à automatiser le centre de données complet en utilisant des scripts de programmation.

II. Présentation Terraform

1. DEFINITION

Terraform est une infrastructure open-source en tant qu'outil de code développé par HashiCorp. Il est utilisé pour définir et provisionner l'infrastructure complète à l'aide d'un langage déclaratif facile à maîtriser.

Il s'agit d'un outil de provisioning d'infrastructure dans lequel vous pouvez stocker la configuration de votre infrastructure cloud sous forme de codes. C'est très similaire à des outils tels que Formation Nuage, que vous utiliseriez pour automatiser votre infrastructure AWS, mais vous ne pouvez l'utiliser que sur AWS. Avec Terraform, vous pouvez l'utiliser sur d'autres plateformes cloud également.

2. Avantages

1) Gérer rapidement son infrastructure

Le principal bénéfice de Terraform est sans doute le **gain de temps**. En effet, grâce aux fichiers de configuration, vous avez le contrôle pour approvisionner, définir ou paramétrer vos **ressources** de manière répétable et prévisible. Vous réduisez ainsi considérablement les erreurs de déploiement et d'administration.

De même, il vous est possible de **déployer** le même modèle plusieurs fois dans le même environnement de développement, de test et de production. Ces environnements peuvent être créés à la demande.

Les méthodes traditionnelles de **déploiement** peuvent prendre des jours, voire des semaines. Alors que le déploiement complet de votre **infrastructure** avec Terraform prend généralement quelques minutes. C'est également le cas pour les migrations, elles sont prises en charge rapidement.

De plus, Terraform applique l'approche « **DRY** » (Dont Repeat Yourself). Grâce à sa structure modulaire. L'outil utilise des **fonctions répétables** vous permettant d'automatiser vos tâches manuelles. La réutilisabilité du code vous apportera un gain de temps significatif.

Une nouvelle caractéristique qui renforcera votre productivité concerne la composition de Terraform. Terraform est **agentless**, cela signifie que vous n'avez pas besoin d'installer un agent pour utiliser cette technologie. Terraform peut facilement être installé et utilisé.

2) Un outil multiplateforme

L'agnosticisme est un autre avantage de **Terraform**. En informatique, un système **agnostique** est un système universel pouvant s'exécuter sur n'importe quelle plateforme.

Terraform est alors multiplateforme, vous pouvez ainsi utiliser le même **fichier de configuration** pour gérer simultanément les ressources d'une [grande variété de fournisseurs cloud](#) dont :

- AWS (Amazon Web Services)
- GCP (Google Cloud Platform)
- Oracle Cloud Infrastructure
- Alibaba Cloud
- Azure

Les équipes **DevOps** peuvent ainsi gérer toutes leurs infrastructures cloud grâce à une solution unifiée. Ceci entraînant un immense gain de temps.

3) Bonne capacité de suivi de vos performances

Grâce à la fonctionnalité de **Monitoring as Code**, Terraform simplifie vos actions de monitoring. Cette technologie stocke l'état de votre infrastructure afin de suivre ses modifications et partager ses configurations.

Le **Monitoring as Code** révolutionne le monitoring traditionnel. En effet, le contrôle à grande échelle est une tâche fastidieuse. Les flux de travail manuel posent également des problèmes de **collaboration** en raison du manque de transparence entre les différentes équipes.

Le Monitoring as Code permet une implémentation de la méthodologie agile grâce à l'utilisation de workflows de **CI/CD** (intégration et déploiement en continu).

Pour résumer, Terraform démontre une bonne capacité de suivi grâce à :

- Un **provisioning** plus efficace qui assure une meilleure scalabilité
- Plus de transparence et des rollbacks plus simples
- Une unification des process dans un **workflow** en CI/CD

4) Une collaboration renforcée

La **collaboration** est renforcée avec Terraform que ce soit pour les petites équipes ou les grandes entreprises. Cela est possible grâce à différentes fonctionnalités clés :

- La gestion des **états** : Elle permet une gestion complète à travers les versions des fichiers d'état de Terraform
- Des plans et des applications centralisés : L'exécution des plans et des applications ainsi que leur examen se font au même endroit
- Le registre de **modules** : Il permet de partager des modules réutilisables au sein d'une même équipe

Au-delà de ses caractéristiques, Terraform ne limite pas son nombre d'**utilisateurs** et de **workspaces**. Le cryptage est assuré par Hashicorp Vault, un outil reconnu en cybersécurité.

Avec cet outil, il n'y a pas de **documentations** séparées. Le code écrit pour l'infrastructure devient la documentation. Un atout important pour assurer la transparence au sein de vos équipes.

5) Une grande communauté

Terraform est détenu par **Hashicorp**, une société américaine, leader mondial dans l'automatisation des infrastructures pour les environnements multiclouds. La technologie est alors régulièrement mise à jour et dispose d'une grande **communauté** (34 100 étoiles et 1 656 contributeurs sur Github en septembre 2022).

On ne dénombre pas moins de 120 000 **utilisateurs** de Terraform Cloud et 1 200 entreprises utilisant Terraform Entreprise. Terraform est présent dans 45 pays à travers le monde.

Une grande communauté vous permet d'obtenir de l'aide sur le fonctionnement de **Terraform** et ainsi de corriger vos éventuelles erreurs de configuration. Il existe deux options pour consulter la [communauté Terraform](#) :

- Le [forum](#) : pour poser des questions ou pour accéder à la réponse d'une question déjà posée
- Le [bug tracker](#) sur GitHub : cet endroit est à utiliser pour **référencer** ses bugs, il ne peut pas être utilisé pour demander de l'aide (pour cela, on utilise le forum de Terraform)

6) Un fonctionnement sans maître

Terraform est **sans maître**, par défaut, il exploite directement l'API des fournisseurs cloud. Il n'a donc pas besoin d'un nœud maître pour garder la trace de toutes les configurations et distribuer les mises à jour. La suppression du nœud maître permet une **économie importante** des coûts d'infrastructure et de maintenance.

3. Ansible VS Terraform

Terraform et Ansible sont tous deux des outils d'infrastructure en tant que code, mais il existe quelques différences significatives entre les deux :

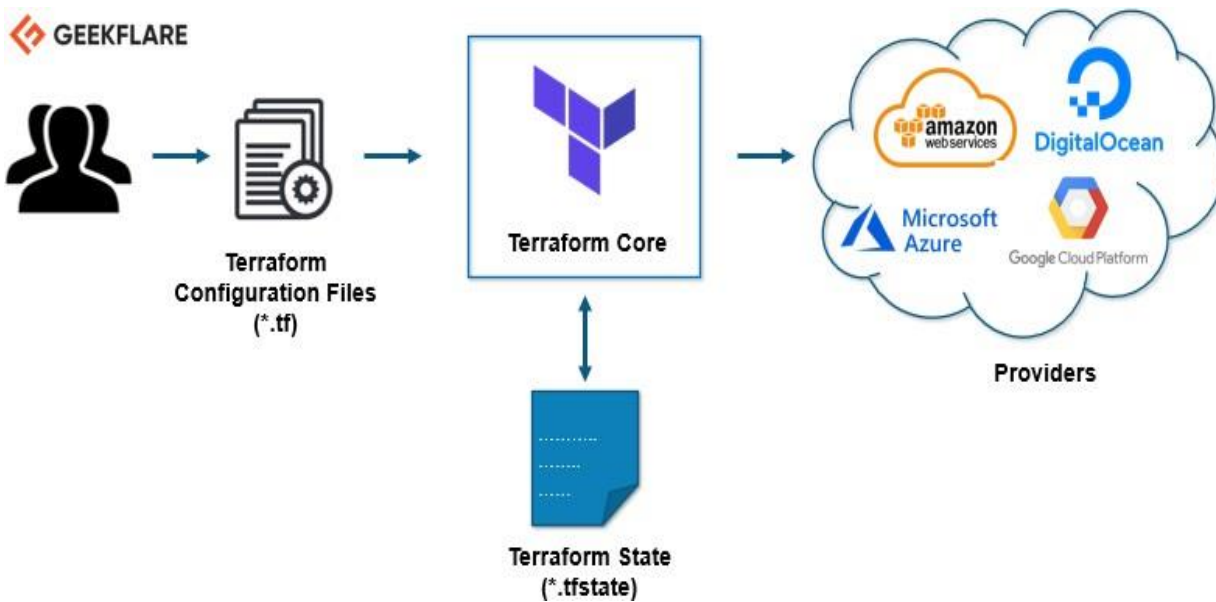
- Alors que Terraform est purement un outil déclaratif, Ansible combine à la fois une configuration déclarative et *procédurale*. Dans la configuration procédurale, vous spécifiez les étapes, ou la manière précise, dont vous souhaitez provisionner l'infrastructure à l'état souhaité. La configuration procédurale demande plus de travail, mais elle offre plus de contrôle.
- Terraform est open source ; Ansible est développé et vendu par Red Hat.

Concepts

1. Architecture

Terraform à deux composants principaux qui composent son architecture:

- Noyau Terraform
- Fournisseurs (Provider)



- **Le Noyau Terraform**

Terraform Core utilise deux sources d'entrée pour faire son travail.

La première source d'entrée est une configuration Terraform que vous configurez en tant qu'utilisateur. Ici, vous définissez ce qui doit être créé ou provisionné. Et la **seconde** source d'entrée est un état dans lequel terraform maintient l'état à jour de la configuration actuelle de l'infrastructure.

Donc, ce que fait le noyau de terraform, c'est qu'il prend les entrées, et il élabore le plan de ce qui doit être fait. Il compare l'état, quel est l'état actuel et quelle est la configuration que vous désirez dans le résultat final. Il détermine ce qui doit être fait pour atteindre cet état souhaité dans le fichier de configuration. Il indique ce qui doit être créé, ce qui doit être mis à jour, ce qui doit être supprimé pour créer et provisionner l'infrastructure.

- **Les Providers**

Le deuxième composant de l'architecture sont les fournisseurs de technologies spécifiques. Il peut s'agir de fournisseurs de cloud comme AWS, Azure, GCP ou d'autres infrastructures en tant que plate-forme de services. C'est également un fournisseur de composants de plus haut niveau comme **Kubernetes** ou d'autres outils de plate-forme en tant que service, même certains logiciels en tant qu'outil en libre-service.

Il vous donne la possibilité de créer une infrastructure à différents niveaux.

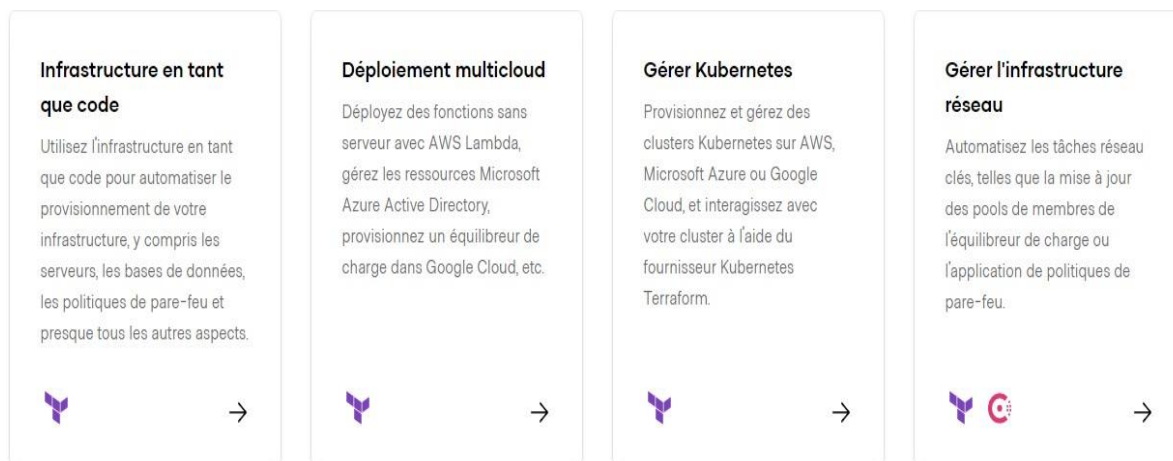
Par exemple, créez une infrastructure AWS, puis déplacez Kubernetes dessus, puis créez des services / composants à l'intérieur de ce cluster Kubernetes.

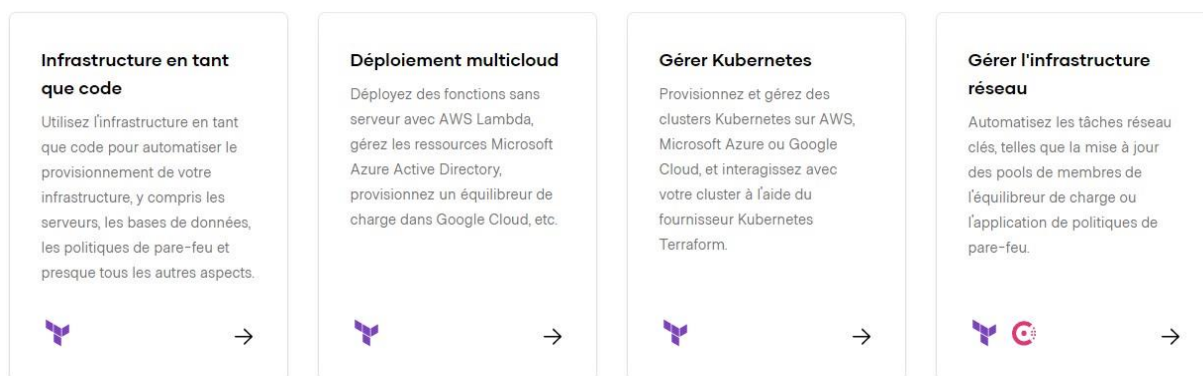
Terraform compte plus d'une centaine de fournisseurs pour différentes technologies, et chaque fournisseur donne ensuite aux utilisateurs de terraform l'accès à ses ressources.

Ainsi, via le fournisseur AWS, par exemple, vous avez accès à des centaines de ressources AWS telles que les instances EC2, les utilisateurs AWS, etc.

C'est ainsi que Terraform fonctionne, et de cette façon, il essaie de vous aider à provisionner et à couvrir la configuration complète de l'application depuis l'infrastructure jusqu'à l'application.

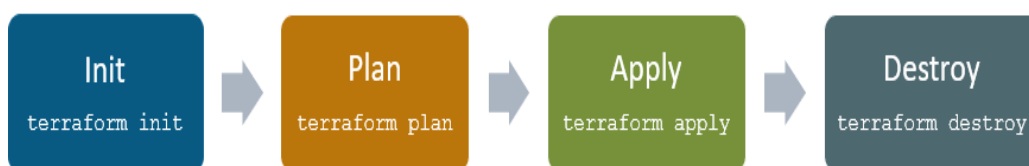
2. Use Case





3. LifeCycle

Le cycle de vie de Terraform se compose de - init, plan, vous inscrire et détruire.



- Terraform init initialise le répertoire de travail qui comprend tous les fichiers de configuration
- Le plan Terraform est utilisé pour créer un plan d'exécution pour atteindre l'état souhaité de l'infrastructure. Les changements dans les fichiers de configuration sont effectués afin d'atteindre L'état souhaité.
- Terraform apply effectue ensuite les modifications de l'infrastructure telles que définies dans le plan, et l'infrastructure arrive à l'état souhaité.
- Terraform destroy est utilisé pour supprimer toutes les anciennes ressources d'infrastructure, qui sont marquées comme corrompues après la phase d'application.

Démo

Pour réaliser ce tutoriel, vous devez créer un dossier de projet dans lequel vous ajoutez les fichiers suivants :

- Main.tf
- Vars.tf
- Outputs.tf
- terraform.tfvars
- index.html dans lequel vous afficher **HELLO WORD**

Après avoir créé le projet, placez-vous sur le chemin du dossier et taper la commande si dessous pour initialiser votre projet.



On commence par créer nos variables de connexion dans le fichier vars.tf.




```
variable "AWS_ACCESS_KEY" {}  
  
variable "AWS_SECRET_KEY" {}  
variable "AWS_REGION" {  
  default = "us-east-1"  
}
```

Les clés sont des variables trop sensibles pour être exposées, raison pour laquelle nous allons leur attribuer leurs valeurs dans le dossier terraform.tfvars.



```
AWS_ACCESS_KEY="VALEUR_DE_ACCESS_KEY"  
AWS_SECRET_KEY="VALEUR_DE_SECRET_KEY"
```

A présent nous pouvons créer notre provider dans le fichier main.tf.



```
provider "aws" {  
  region = var.AWS_REGION  
  access_key = var.AWS_ACCESS_KEY  
  secret_key = var.AWS_SECRET_KEY  
}
```

Maintenant nous pouvons créer notre machine virtuelle mais avant cela, il nous faut un groupe de sécurité pour la machine. Allons donc créer le groupe

de sécurité dans le main.tf.


```
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"]
  }
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_key_pair" "my_ec2" {
  key_name = "terraform-key"
  public_key = file(var.ssh_key)
}

resource "aws_instance" "my_ec2_instance" {
  key_name = aws_key_pair.my_ec2.key_name
  ami = var.AWS_AMIS
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance_sg.id]
  connection {
    type = "ssh"
    user = "ubuntu"
    private_key = file(var.private_key)
    host = self.public_ip
  }
  provisioner "file" {
    source = "./index.html"
    destination = "/tmp/index.html"
  }
  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update --fix-missing",
      "sudo apt-get upgrade -y",
      "sudo apt-get install -y apache2",
      "sudo systemctl start apache2",
      "sudo systemctl enable apache2",
      "sudo cp /tmp/index.html /var/www/html/"
    ]
  }
}
```

Maintenant que nous avons notre groupe de sécurité, nous pouvons créer


notre machine. Mais puisque nous aurons par la suite besoin d'approvisionner notre machine, on va générer. Sur la ligne de commande taper la commande ci-dessous



```
ssh-keygen -t rsa
```

Renseigner le nom du fichiers et laisser vide passphrase

Maintenant vous devez ajouter la clé publique(.pub) à votre compte aws. Pour cela placer vous au chemin de la clé et taper la commande :



```
cat nom_cle.pub
```

Copier la clé et aller sur votre compte aws à la section IAM>utilisateur> sélectionner l'utilisateur concerné, cliquer sur charger une clé SSH, coller la clé et générer

Revenons à notre main.tf, il nous faut une paire de clé. Ce code est déjà dans les fichiers

précédants

```
resource "aws_key_pair" "my_ec2"
{
  key_name = "terraform-key" public_key = file(var.ssh_key)
}
```

Maintenant nous allons créer une variable ami (image de la machine) dans le fichier vars.tf. Les amis sont disponibles sur aws et varie en fonction de la région

```
variable "AWS_AMIS" {
  default = "ami-0434abb28cf6cc2b1"
}
```

Enfin nous allons pouvoir créer notre machine virtuelle, y installer apache et déployer un site web.

```

resource "aws_instance" "my_ec2_instance" {
  key_name = aws_key_pair.my_ec2.key_name ami = var.AWS_AMIS
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance_sg.id] connection {
    type = "ssh"
    user = "ubuntu"
    private_key = file(var.private_key) host = self.public_ip
  }
  provisioner "file" {
    source = "./index.html" destination = "/tmp/index.html"
  }

  provisioner "remote-exec" { inline = [
    "sudo apt-get -f -y update",
    "sudo apt-get install -f -y apache2",
    "sudo systemctl start apache2",
    "sudo systemctl enable apache2",
    "sudo cp /tmp/index.html /var/www/html",
  ]
}
}

```

Vous pouvez aller à la page 16 la plupart du code Main.tf est là-bas

Explication de code :

Nous avons créé une ressource `aws_key_pair` requise pour les connexions SSH sur notre instance EC2 nous spécifions d'abord le nom de la paire de clés ainsi que notre clé publique créée précédemment afin d'autoriser notre machine locale à se connecter sur notre instance ec2.

Ensuite, nous imbriquons le bloc `connection` en spécifiant qu'on souhaite se connecter avec le protocole `ssh` sur la machine cible avec l'utilisateur `ubuntu` (utilisateur par défaut sur les AMIs Ubuntu) et notre clé privée créée précédemment. Nous sommes également obligés de fournir

L'adresse de la ressource à laquelle se connecter dans l'argument `host`, cependant les Blocs connexion ne peuvent pas faire référence à leurs ressources parent par leur nom. Au lieu de cela, nous utilisons l'objet spécial `self` qui représente la ressource parent de la connexion et possède tous les arguments de cette ressource. Dans notre cas nous l'utilisons pour récupérer l'adresse IP publique de notre instance EC2.

Maintenant que nous avons déployé notre site, nous aurons certainement besoin d'y accéder, pour cela nous allons créer un output dans le fichier `output.tf` qui va récupérer notre adresse publique et l'afficher à notre terminale.

C'est fait vous pouvez maintenant lancer votre projet en mettant les deux commandes suivantes.



Pour détruire les ressources, mettez la commande suivante :



```
terraform destroy
```