Terraform

GROUPE 4 AWS/restart Cohorte 2

Mame Sandeck Niang
Hapsatou Abou Sow
Aliou Cisse
Khady Diagne
Oumar Bounekhatap Faye

I. Problématique

Les méthodes traditionnelles de déploiement et les migrations mettaient trop de temps et pouvaient avoir des erreurs dans le déploiement et dans l'administration. Et c'est l'idée de l'Infrastructure as Code fait naitre qui est un est une solution pour automatiser le déploiement de serveurs et d'infrastructures en général.

Les développeurs doivent faire face à plusieurs problèmes à savoir :

- Difficulté de la mise en place des infrastructures manuellement étant donnée qu'il faut renseigner plusieurs paramètres (RAM, CPU, types d'instances)
- Problèmes relatifs aux mises à jour : nous avons du mal à savoir quoi mettre à jour à un moment donné et surtout savoir si le changement que nous souhaitons apporter ne serait pas susceptible d'empêcher le bon fonctionnement de notre environnement.
- Si vous avez une infrastructure de grande taille, il est très facile de mal configurer une ressource ou de mettre à disposition des services dans le mauvais ordre.
- Problème de compatibilité entre l'environnement mis à disposition et l'environnement réel.
- La plupart des autres outils IAC (infrastructure as code) sont conçus pour fonctionner avec un seul fournisseur cloud.
- La plupart des outils IaC créent une infrastructure non immuable ce qui signifie que l'infrastructure peut changer pour prendre en charge des modifications telles qu'une mise à niveau du middleware ou un nouveau serveur de stockage

Cette liste est non exhaustive et terraform a été mis en œuvre pour pallier ces problèmes

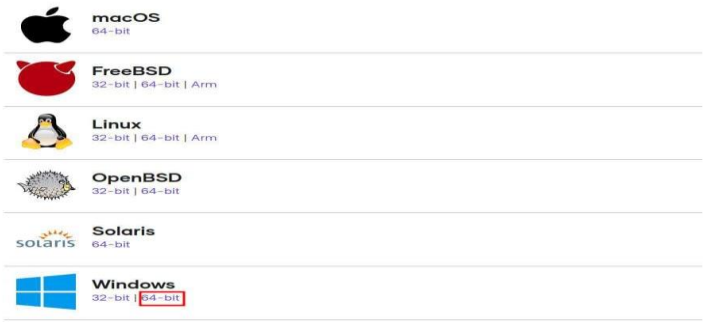
II. Présentation Terraform

Terraform est l'un des outils d'infrastructure en tant que code (IaC) les plus populaires, utilisé par les équipes DevOps pour automatiser les tâches d'infrastructure. Il est utilisé pour automatiser le provisionnement de vos ressources cloud. Terraform est un outil d'approvisionnement open source et indépendant du cloud développé par HashiCorp et écrit en langage GO. Il peut gérer des composants de bas niveau comme les ressources de calcul, de stockage et de mise en réseau, ainsi que des composants de haut niveau comme les entrées DNS et les fonctionnalités SaaS.

III. Installation et configuration de Terraform

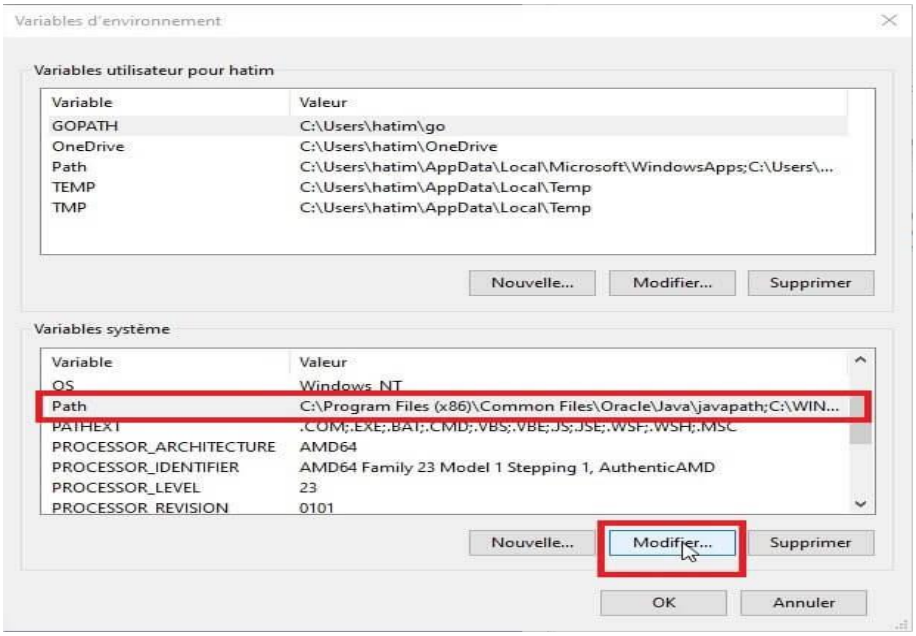
Pour télécharger, on va aller sur le site de Terraform, **terraform.io.** Ensuite on va aller sur la partie Download, et on va aller chercher la distribution pour un environnement de type Windows.

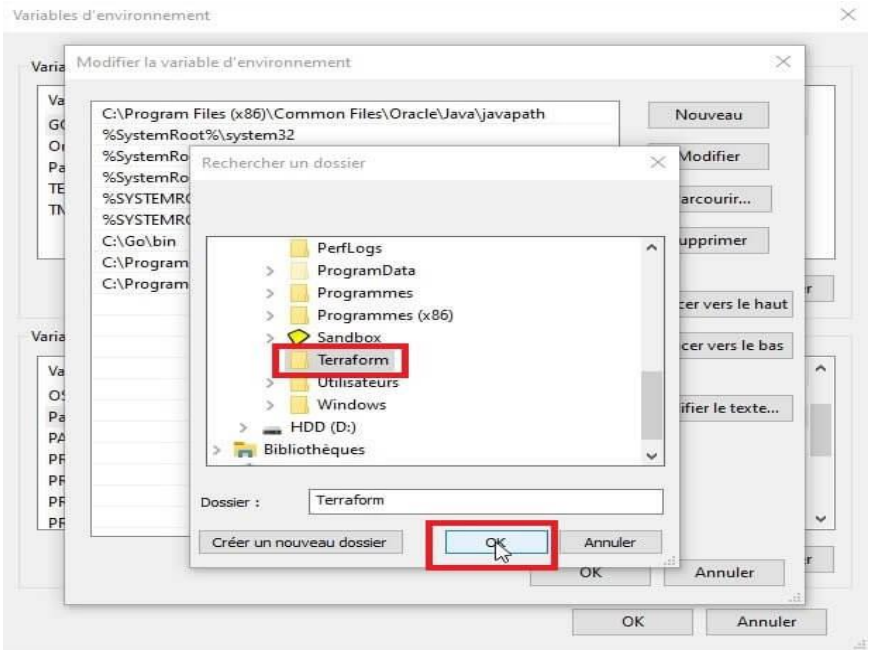
Choisissez le système d'exploitation avec lequel vous travailler :



Une fois le téléchargement terminé, créez un dossier sur votre lecteur *C:* où vous pourrez placer l'exécutable Terraform. Allez ensuite trouver le binaire Terraform dans l'explorateur de fichiers et extrayez ce fichier zip dans le dossier que vous avez créé précédemment

Maintenant il suffit de rajouter votre exécutable Terraform a votre variable d'environnement nommé PATH comme ceci :





Il ne reste plus qu'à vérifier si Terraform est installé avec succès sur votre machine Windows**,** ouvrez votre PowerShell et lancez la commande suivante

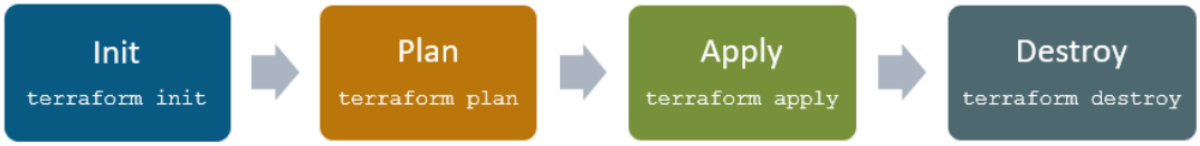
```
C:\Users\Easy Services Pro>terraform -v
Terraform v0.11.13

Your version of Terraform is out of date! The latest version
is 1.3.8. You can update by downloading from www.terraform.io/downloads.html
```

IV. Les commandes de bases de Terraform

Nous avons quatre commandes de terraform qui sont plus utilisés à savoir

- Terraform init : Pour initialiser le projet
- Terraform plan : Plan d'exécution
- Terraform apply : Valider le plan d'exécution
- Terraform destroy : Détruire l'architecture



V. Les Concepts

Vous trouverez ci-dessous les principaux concepts utilisés dans Terraform :

- **Etat** : Terraform enregistre des informations sur l'infrastructure créée dans un fichier d'état Terraform. Avec le fichier d'état, Terraform est capable de retrouver les ressources qu'il a créées précédemment, censées les gérer et les mettre à jour en conséquence.

- **Variables** : Terraform a des variables d'entrée et de sortie, c'est une paire clé-valeur. Les variables d'entrée sont utilisées comme paramètres pour saisir des valeurs au moment de l'exécution afin de personnaliser nos déploiements. Les variables de sortie sont des valeurs de retour d'un module terraform qui peuvent être utilisées par d'autres configurations
- **Provider** : C'est un plugin pour interagir avec les API de service et accéder à ses ressources associées.
- **Module** : C'est un dossier avec des modèles Terraform où toutes les configurations sont définies

- **Région** : Il se compose d'informations mises en cache sur l'infrastructure gérée par Terraform et les configurations associées.
- **Ressources** : Il s'agit d'un bloc d'un ou plusieurs objets d'infrastructure (instances de calcul, réseaux virtuels, etc.), qui sont utilisés dans la configuration et la gestion de l'infrastructure.
- **La source de données** : Il est implémenté par les fournisseurs pour renvoyer des informations sur les objets externes à terraform.
- **Valeurs locales** : Les valeurs locales peuvent être utiles pour éviter de répéter plusieurs fois les mêmes valeurs ou expressions dans une configuration, mais si elles sont trop utilisées, elles peuvent également rendre la lecture d'une configuration difficile

A. Fonctionnement de 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.

HashiCorp et la communauté Terraform ont déjà écrit des milliers de fournisseurs pour gérer de nombreux types de ressources et de services. Vous pouvez trouver tous les fournisseurs accessibles au public sur le registre Terraform, y compris Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog et bien d'autres.

Le flux de travail principal de Terraform se compose de trois étapes :

- **Écrire :** vous définissez des ressources, qui peuvent se trouver sur plusieurs fournisseurs et services cloud. Par exemple, vous pouvez créer une configuration pour déployer une application sur des machines virtuelles dans un réseau Virtual Private Cloud (VPC) avec des groupes de sécurité et un équilibreur de charge.

- **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.

- **Appliquer :** après approbation, Terraform exécute les opérations proposées dans le bon ordre, en respectant toutes les dépendances de ressources. Par exemple, si vous mettez à jour les

propriétés d'un VPC et modifiez le nombre de machines virtuelles dans ce VPC, Terraform recréera le VPC avant de redimensionner les machines virtuelles.

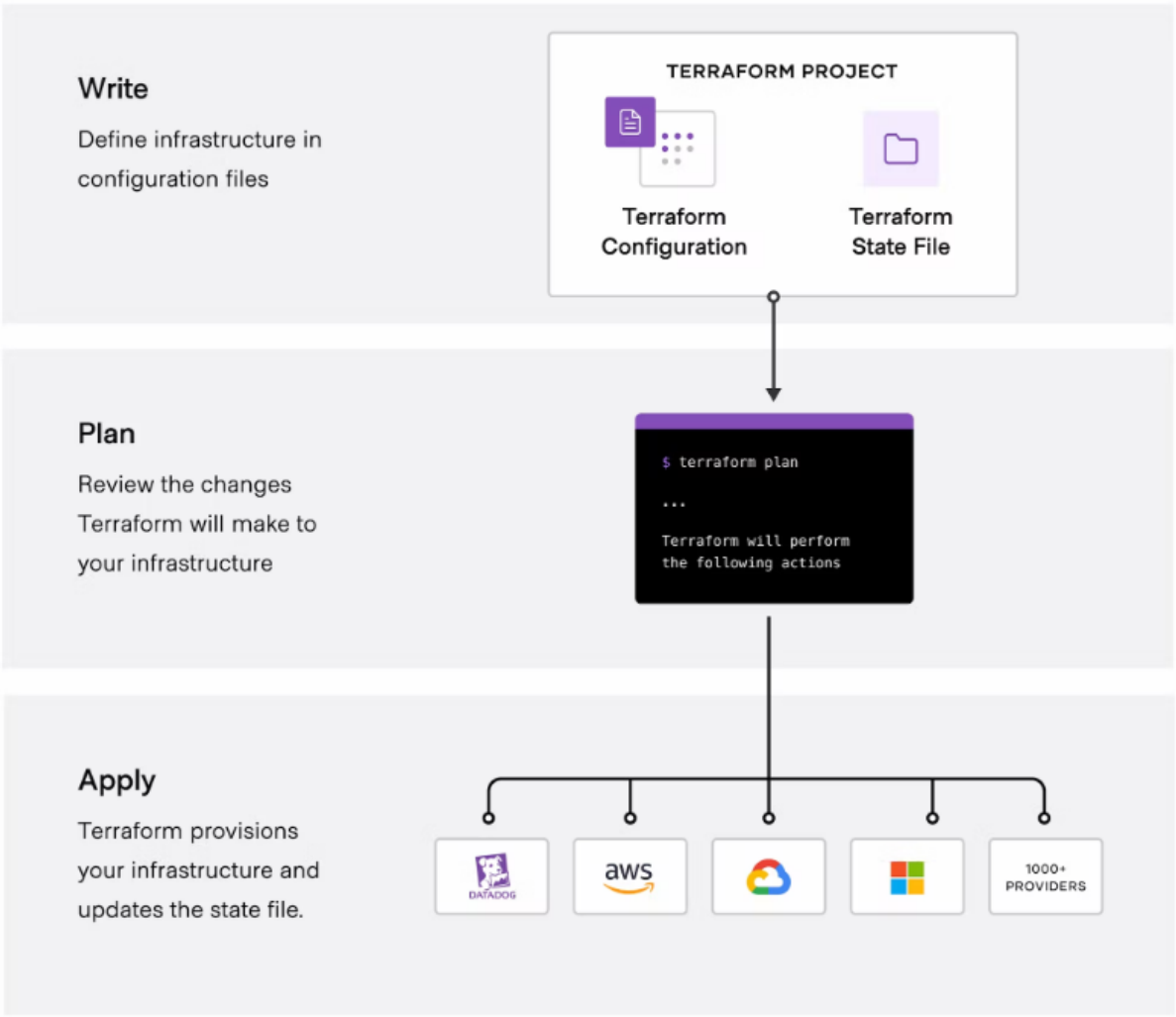


Figure : fonctionnement de Terraform

B. Fonctionnalités Terraform

- Terraform est capable de gérer de nombreux environnements.
- Il est assez simple de gérer les prestataires de services externes.
- Pour augmenter la tolérance aux pannes, il peut gérer de nombreux nuages.
- Pour un développement plus rapide, il utilise une technique déclarative.
- Il aide à présenter le modèle généré dans un format graphique et compréhensible.
- Son code modulaire contribue de manière significative à la cohérence, à la réutilisation et à la collaboration.
- Il prend en charge la mise en réseau définie par logiciel.

- Il prend en charge plusieurs plates-formes cloud
- Il verrouille les modules avant d'appliquer les changements d'état pour s'assurer qu'une seule personne peut apporter des modifications à la fois

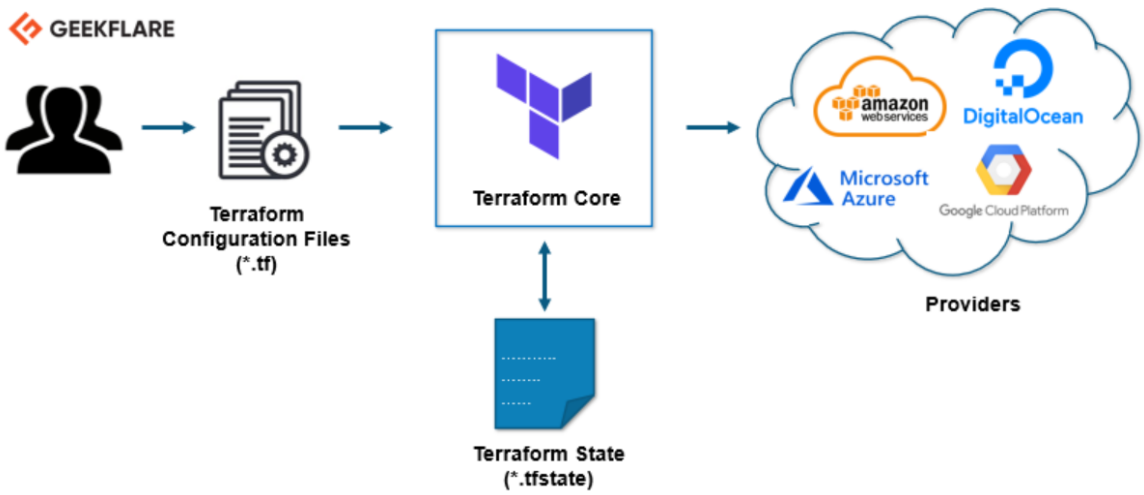
C. Architecture Terraform

Terraform a deux composants principaux qui composent son architecture :

- **Noyau Terraform** : Nous avons deux sources d'entrées

  - **Premier** la 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.

- **Fournisseurs :** 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



VI. Etudes comparative entre Terraform et Ansible

La bataille Ansible contre Terraform continue de s'intensifier chaque jour qui passe à mesure que le mouvement DevOps prend de l'ampleur. Ces deux noms occupent désormais une place importante dans le paysage DevOps, et vous pouvez les entendre fréquemment de temps en temps. Chaque outil est connu pour ses avantages distincts dans la création d'infrastructure en tant que code (IaC).

Ils peuvent tous deux fournir un IaC qui peut aider à déployer des environnements reproductibles traitant de diverses exigences de complexité. Cependant, de nombreuses personnes ne sont pas conscientes de la différence entre Ansible et Terraform. Il peut être facile de choisir parmi une alternative populaire en matière de DevOps

La figure ci-dessous nous fait une petite comparaison entre ces deux outils

	Terraform	Ansible
Type	Orchestration tool	Configuration management tool
Syntex	HCL	YAML
Language	Declarative	Procedural
Default approach	Mutable infrastructure	Immutable infrastructure
Lifecycle management	Does support	Doesn't support
Capabilities	Provisioning and configuring	Provisioning and configuring
Agentless	✓	✓
Masterless	✓	✓

spacelift.io

VII. DEMONSTRATION

Bibliographie

https://blog.gruntwork.io/how-to-manage-terraform-state-28f5697e68fa

https://blog.ippon.fr/2022/04/04/terraform-dans-tous-ses-etats-1-2/

https://datascientest.com/terraform

https://geekflare.com/fr/terraform-for-beginners/