Tout sur Terraform : Créez votre premier projet | Backend distant | Modules | Questions d'entretien

Plan du cours

---

1. Introduction à Terraform

Terraform est un outil open-source d'Infrastructure as Code (IaC) développé par HashiCorp. Il permet de définir, provisionner et gérer des infrastructures de manière déclarative. Terraform est compatible avec plusieurs fournisseurs cloud tels qu'AWS, Azure, Google Cloud, et peut également gérer des infrastructures sur site.

Pourquoi utiliser Terraform ?

- Automatisation complète du provisionnement et de la gestion de l'infrastructure.
- Consistance et reproductibilité grâce à la gestion sous forme de code.
- Collaboration améliorée entre équipes grâce aux fichiers de configuration partagés.
- Gestion efficace des dépendances entre ressources.

Terraform utilise un langage de configuration appelé HCL (HashiCorp Configuration Language), qui est lisible et simple à comprendre.

---

2. Cas d'utilisation de Terraform

Terraform est utilisé pour :

- **Gérer toute infrastructure** : Cloud (AWS, Azure, GCP), on-premise, hybride.
- **Suivre l'évolution de l'infrastructure** : Permet de versionner et suivre les changements d'infrastructure.
- **Automatiser les modifications** : Terraform applique les changements sans intervention manuelle.

- **Standardiser les configurations** : Assure que les mêmes ressources sont configurées de manière identique.
- **Collaborer efficacement** : Partage et réutilisation des configurations entre équipes.

Terraform repose sur un modèle simple :

```
Script Terraform -> Fournisseur Terraform -> API cible
```

---

3. Cycle de vie de Terraform

Terraform suit un cycle de vie bien défini pour gérer les ressources :

1. **Write (Écrire)** : Définir l'infrastructure sous forme de code dans des fichiers `.tf`.
2. **Plan (Planifier)** : Analyser les modifications qui seront appliquées sans les exécuter.
3. **Apply (Appliquer)** : Appliquer les modifications planifiées pour créer ou modifier l'infrastructure.
4. **Delete (Supprimer)** : Détruire les ressources lorsque cela est nécessaire.

Diagramme du cycle de vie Terraform :

```
Écriture -> Planification -> Application -> Suppression
```

---

4. Installation et configuration de Terraform

Installation sur macOS :

```
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
brew upgrade hashicorp/tap/terraform
```

Installation sur Linux :

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
```

Vérification de l'installation :

```
terraform -version
```

---

5. Écriture de votre premier projet Terraform

Un projet Terraform est généralement composé de plusieurs fichiers :

- `main.tf` : Contient la configuration principale des ressources.
- `variables.tf` : Définit les variables utilisées dans la configuration.
- `outputs.tf` : Définit les sorties de Terraform.

Exemple de configuration Terraform pour AWS EC2

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
```

```
}
```

---

6. Comprendre le fichier d'état de Terraform

Terraform utilise un fichier d'état (`terraform.tfstate`) qui stocke l'état actuel des ressources.

Bonnes pratiques :

- **Ne jamais stocker le fichier d'état localement ou sur GitHub.**
- **Utiliser un backend distant (ex : AWS S3 + DynamoDB pour le verrouillage).**
- **Ne jamais modifier manuellement ce fichier.**

Exemple d'utilisation d'un backend distant avec S3 :

```
terraform {
  backend "s3" {
    bucket         = "mon-bucket-terraform"
    key            = "chemin/vers/statefile.tfstate"
    region         = "us-east-1"
    dynamodb_table = "terraform-lock"
  }
}
```

---

7. Bonnes pratiques avec Terraform

- Utiliser des **modules** pour organiser le code.
- Séparer les fichiers `main.tf`, `variables.tf`, et `outputs.tf`.
- Versionner les fichiers Terraform avec Git.
- Utiliser **terraform plan** avant d'exécuter **terraform apply**.

---

8. Utilisation des modules Terraform

Les modules permettent de rendre le code Terraform réutilisable et modulaire.

Types de modules :

- **Modules existants** : Disponibles sur le Terraform Registry.
- **Modules personnalisés** : Créés pour des besoins spécifiques.

Exemple de module Terraform pour créer une instance EC2 :

```
module "ec2_instance" {
  source        = "terraform-aws-modules/ec2-instance/aws"
  instance_type = "t2.micro"
  ami           = "ami-12345678"
}
```

---

9. Problèmes courants avec Terraform

1. **Le fichier d'état est la source unique de vérité.**
2. **Les modifications manuelles des ressources cloud ne sont pas détectées automatiquement.**
3. **Ne s'intègre pas facilement avec les outils GitOps (FluxCD, ArgoCD).**
4. **Complexité croissante avec des infrastructures volumineuses.**
5. **Difficile à utiliser comme outil de gestion de configuration.**

---

10. Questions d'entretien Terraform

Questions courantes :

1. **Qu'est-ce qu'un module dans Terraform ?**
2. **Comment configurer Terraform ?**
3. **Quel est le rôle de S3 et DynamoDB dans Terraform ?**
4. **Citez quelques commandes essentielles de Terraform.**

Ce cours détaillé couvre tout ce dont vous avez besoin pour bien maîtriser Terraform et vous préparer à l'utiliser en entreprise ou en entretien technique.