TP Guidé : Créer et Gérer des Buckets S3 avec Terraform

Contexte : Les participants vont créer des buckets S3 sur AWS, en utilisant des variables, des outputs, des boucles/conditions, un module, et explorer le state ainsi qu'une gestion basique des environnements.

Prérequis :

- AWS CLI configuré avec des credentials.
- Terraform installé.
- Les participants savent déjà utiliser terraform init, plan, apply, et destroy.

Étape 1 : Configuration Initiale

Objectif : Créer une ressource S3 simple.

1. Créez un dossier terraform-tp et un fichier main.tf

```
provider "aws" {
  region = "eu-west-1"
}

resource "aws_s3_bucket" "my_bucket" {
  bucket = "mon-bucket-tp-2025"
  tags = {
    Name = "mon-bucket-tp"
  }
}
```

Explication ligne par ligne

1. **resource** :
   - C'est le mot-clé qui indique à Terraform que tu définis une ressource.
   - Il dit : "Je vais créer ou gérer quelque chose dans l'infrastructure."
2. **"aws_s3_bucket"** :
   - C'est le **type de ressource**, qui dépend du provider utilisé (ici, AWS).
   - La syntaxe est toujours "nom_du_provider_nom_de_la_ressource".

Exemples :

- aws_s3_bucket pour un bucket S3.
- aws_instance pour une machine virtuelle EC2.
- google_storage_bucket pour un bucket sur Google Cloud.

Ce type est défini par le provider et documenté dans le **Terraform Registry** (par exemple, sur registry.terraform.io).

Unicité : Deux ressources du même type (ex. : aws_s3_bucket) ne peuvent pas avoir le même nom logique (ex. : mon_bucket) dans le même projet.

3. Exécutez :
    - terraform init
    - terraform plan
    - terraform apply --auto-approve
4. Vérifiez dans la console AWS que le bucket est créé.

Étape 2 : Ajout de Variables et Outputs (20 min)

Objectif : Paramétrer avec des variables et afficher des résultats.

1. Créez un fichier variables.tf

```
variable "region" {
  description = "Région AWS"
  type        = string
  default     = "eu-west-1"
}

variable "bucket_name" {
  description = "Nom du bucket"
  type        = string
  default     = "mon-bucket-tp-2025"
}
```

2. Modifiez main.tf

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

resource "aws_s3_bucket" "my_bucket" {
  bucket = var.bucket_name
  tags = {
    Name = var.bucket_name
  }
}
```

3. Exécutez terraform apply et observez l'ARN dans la sortie.

Qu’est-ce que terraform.tfvars ?

- Le fichier **terraform.tfvars** est un fichier spécial dans Terraform utilisé pour **définir des valeurs pour les variables** que tu as déclarées dans ton code (généralement dans un fichier variables.tf).
- Il permet de personnaliser ton infrastructure sans modifier directement les fichiers de configuration principaux (comme main.tf ou variables.tf).
- C’est une manière simple et pratique de passer des valeurs aux variables, surtout quand tu veux gérer différents environnements (dev, prod, etc.) ou éviter de coder en dur des valeurs sensibles.

Créez un fichier terraform.tfvars :

```
region      = "us-east-1"
bucket_name = "mon-bucket-tp-us-2025"
```

1. Relancez terraform apply et notez le changement.

Pourquoi utiliser terraform.tfvars ?

1. **Séparation des préoccupations** : Tu gardes la logique (dans main.tf) séparée des valeurs spécifiques (dans terraform.tfvars).
2. **Réutilisabilité** : Tu peux avoir plusieurs fichiers .tfvars pour différents contextes (ex. : dev.tfvars, prod.tfvars).
3. **Simplicité** : Pas besoin de passer des variables en ligne de commande ou de modifier le code pour chaque déploiement.

Comment ça fonctionne ?

1. Tu déclares des variables dans un fichier comme variables.tf.
2. Tu fournis leurs valeurs dans terraform.tfvars.
3. Terraform utilise ces valeurs lors de l’exécution des commandes (plan, apply, etc.).

Exemple de base

1. Déclarer des variables dans variables.tf

```
variable "region" {
  description = "Région AWS"
  type        = string
  default     = "eu-west-1"
}

variable "bucket_name" {
  description = "Nom du bucket S3"
  type        = string
}
```

- ci, region a une valeur par défaut (eu-west-1), mais bucket_name n'en a pas et doit être défini.

2. Créer terraform.tfvars

```
region      = "us-east-1"
bucket_name = "mon-bucket-2025"
```

Format : nom_de_la_variable = valeur.
Les valeurs peuvent être des chaînes (entre guillemets), des nombres, des listes, ou des maps (objets clé-valeur).

3. Utiliser les variables dans main.tf

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

resource "aws_s3_bucket" "mon_bucket" {
  bucket = var.bucket_name
  tags = {
    Name = var.bucket_name
  }
}
```

4. Exécuter Terraform

- Lance terraform init puis terraform apply.
- Terraform lira automatiquement terraform.tfvars et utilisera us-east-1 comme région et mon-bucket-2025 comme nom du bucket.

Syntaxe et types de valeurs dans terraform.tfvars

1. **Chaîne (string)** :

```
region = "us-west-2"
```

2. Nombre (number) :

```
instance_count = 3
```

3. Liste (list) :

```
availability_zones = ["us-east-1a", "us-east-1b"]
```

4. Map (objet clé-valeur) :

```
tags = {
  Name        = "Mon Bucket"
  Environment = "Dev"
}
```

5. Booléen (boolean) :

```
enable_versioning = true
```

Comment Terraform charge terraform.tfvars ?

- **Automatique** : Si un fichier nommé terraform.tfvars existe dans le répertoire courant, Terraform le charge sans que tu aies à le spécifier.
- **Fichiers personnalisés** : Si tu utilises un autre nom (ex. : dev.tfvars), tu dois le passer explicitement avec l'option -var-file :

```
terraform apply -var-file="dev.tfvars"
```

Exemple avec plusieurs environnements

variables.tf

```hcl
variable "region" {
  type = string
}

variable "bucket_name" {
  type = string
}

variable "bucket_count" {
  type = number
}
```

dev.tfvars

```hcl
region       = "us-east-1"
bucket_name  = "dev-bucket-2025"
bucket_count = 1
```

prod.tfvars

```hcl
region       = "eu-west-1"
bucket_name  = "prod-bucket-2025"
bucket_count = 3
```

main.tf

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

resource "aws_s3_bucket" "my_buckets" {
  count  = var.bucket_count
  bucket = "${var.bucket_name}-${count.index}"
  tags = {
    Name = "${var.bucket_name}-${count.index}"
  }
}
```

Bonnes pratiques avec terraform.tfvars

1. **Ne pas versionner les secrets** : Évite de mettre des clés d'accès ou mots de passe dans terraform.tfvars. Utilise plutôt des variables d'environnement ou un gestionnaire de secrets (ex. : AWS Secrets Manager).
    - Exemple avec variable d'environnement : (on va expliquer ça just après)

```
export TF_VAR_region="us-east-1"
```

2. Utiliser des noms explicites : Si tu as plusieurs fichiers .tfvars, nomme-les selon leur usage (ex. : staging.tfvars, prod.tfvars).

3. Valeurs par défaut : Si une variable a une valeur par défaut dans variables.tf, elle sera écrasée par celle de terraform.tfvars si définie.

4. Validation : Assure-toi que les types des valeurs dans .tfvars correspondent aux types définis dans variables.tf (string, number, etc.).

Points clés à retenir

- terraform.tfvars est un moyen de **fournir des valeurs aux variables** sans toucher au code principal.
- Il est **facultatif** : si toutes tes variables ont des valeurs par défaut, tu n'as pas besoin de ce fichier.
- Il est **flexible** : tu peux avoir plusieurs fichiers .tfvars pour différents cas d'utilisation.

export TF_VAR_region="us-east-1". C'est une alternative à l'utilisation de terraform.tfvars pour passer des valeurs aux variables dans Terraform, et ça repose sur les **variables d'environnement**

- **export** : Une commande qui crée ou met à jour une variable d'environnement dans ton terminal.
- **TF_VAR_region** : Le nom de la variable d'environnement, où TF_VAR_ est un préfixe spécial que Terraform reconnaît, suivi du nom de la variable Terraform (ici, region).
- **"us-east-1"** : La valeur assignée à la variable region.

Comment Terraform reconnaît cette variable ?

Terraform recherche automatiquement les variables d'environnement qui commencent par **TF_VAR_**, suivi du nom d'une variable définie dans ton code (ex. : dans variables.tf). Si une telle variable d'environnement existe, Terraform utilise sa valeur.

Exemple

1. Déclare une variable dans variables.tf

```
variable "region" {
  description = "Région AWS"
  type        = string
}
```

2. Utilise cette variable dans main.tf :

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

resource "aws_s3_bucket" "mon_bucket" {
  bucket = "mon-bucket-2025"
}
```

3. Définis la variable d'environnement dans ton terminal :

```
export TF_VAR_region="us-east-1"
```

4. Lance Terraform :

```
terraform apply
```

→ Terraform utilisera us-east-1 comme valeur pour var.region, sans besoin de terraform.tfvars.

NB 🎉

echo $TF_VAR_region

unset TF_VAR_region

Tableau comparatif

Rôle	Déclare les variables	Fournit des valeurs	Fournit des valeurs
Exemple	`variable "region" { default = "eu-west-1" }`	`region = "us-east-1"`	`export TF_VAR_region="us-east-1"`
Obligatoire ?	Oui (pour utiliser `var.` dans le code)	Non (si valeurs par défaut suffisent)	Non (alternative à `.tfvars`)
Persistance	Permanent (fichier)	Permanent (fichier)	Éphémère (terminal)
Sécurité	N/A (pas de valeurs)	Faible (valeurs en clair)	Élevée (pas dans un fichier)
Complexité supportée	N/A (déclaration seulement)	Élevée (listes, maps)	Moyenne (simple à écrire)
Priorité	Plus basse (valeur par défaut)	Moyenne	Haute (écrase `.tfvars`)
Utilisation typique	Définir la structure des variables	Config statique ou multi-environnements	Secrets ou déploiements dynamiques

Qu’est-ce qu’un output dans Terraform ?

Un **output** est une manière de **récupérer et d’afficher des informations** sur les ressources que tu as créées ou gérées avec Terraform. Après avoir exécuté terraform apply, les outputs te permettent de sortir des valeurs spécifiques (comme un ID, une URL, ou un ARN) pour les utiliser ailleurs, les partager, ou simplement les consulter.

- **Définition** : Un output est un bloc dans ton code Terraform qui spécifie une valeur à exporter.
- **Utilité** : Ils rendent visibles des attributs des ressources qui ne sont pas forcément connus avant le déploiement (ex. : un ID généré automatiquement par AWS).

Syntaxe d’un output

Un output est déclaré avec le mot-clé **output** dans un fichier Terraform (souvent main.tf ou un fichier dédié comme outputs.tf). Voici la structure de base :

```
output "nom_de_l_output" {
  value       = expression
  description = "Description optionnelle"
}
```

- **output** : Mot-clé qui indique que tu définis une sortie.
- **nom_de_l_output** : Un nom que tu choisis pour identifier cette sortie (doit être unique dans ton projet).
- **value** : La valeur que tu veux exporter (souvent un attribut d'une ressource).
- **description** (optionnel) : Une explication de ce que représente cet output.

Exemple simple

Imaginons que tu crées un bucket S3 et que tu veux récupérer son ARN (Amazon Resource Name) après le déploiement.

Code dans main.tf

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

resource "aws_s3_bucket" "mon_bucket" {
  bucket = "mon-bucket-2025"
  tags = {
    Name = "Mon Bucket"
  }
}

output "bucket_arn" {
  value       = aws_s3_bucket.mon_bucket.arn
  description = "L'ARN du bucket S3 créé"
}
```

Résultat

Après avoir exécuté terraform apply, tu verras quelque chose comme ça dans le terminal :

```
Outputs:

bucket_arn = "arn:aws:s3:::mon-bucket-2025"
```

Pourquoi utiliser des outputs ?

1. **Information utile** : Récupérer des données générées par le provider (ex. : ID d'une ressource, adresse IP).
2. **Intégration** : Passer des valeurs à d'autres outils ou projets Terraform (ex. : via un module).
3. **Débogage** : Vérifier que les ressources ont été créées comme prévu.
4. **Documentation** : Fournir un résumé clair des résultats d'un déploiement.

aws_s3_bucket.mon_bucket.arn : Référence l'attribut arn de la ressource mon_bucket.
Tu peux aussi consulter les outputs à tout moment avec terraform output.