TD2 : Créer un module Terraform réutilisable

Objectifs

- Comprendre l'intérêt des variables et des outputs
- Structurer un projet Terraform
- Créer et utiliser un module
- Manipuler Terraform de manière autonome

Contexte

Vous travaillez dans une équipe DevOps.

Votre équipe doit déployer plusieurs environnements de test pour différentes équipes :

- frontend
- backend
- data

Chaque équipe a besoin :

- d'un conteneur web
- accessible via un port différent

Problème actuel :

- duplication du code Terraform
- erreurs fréquentes (ports, noms…)
- difficile à maintenir

Objectif :

Créer une solution réutilisable permettant de déployer plusieurs conteneurs rapidement et sans erreur.

1. Structure du projet

```
mkdir -p terraform-td2/modules/nginx
cd terraform-td2
```

```
terraform-td2/
  main.tf
  modules/
    nginx/
      main.tf
      variables.tf
      outputs.tf
```

2. Création du module

Objectif : créer un composant réutilisable

2.1 Variables du module

Fichier : `modules/nginx/variables.tf`

```
variable "container_name" {
  description = "Nom du conteneur"
}

variable "external_port" {
  description = "Port exposé"
}
```

Question :

- Pourquoi ne pas coder ces valeurs en dur ?

2.2 Ressources

Fichier : `modules/nginx/main.tf`

```
resource "docker_image" "nginx" {
  name = "nginx:latest"
}

resource "docker_container" "nginx" {
  name  = var.container_name
  image = docker_image.nginx.name

  ports {
    internal = 80
    external = var.external_port
  }
}
```

2.3 Outputs du module

Fichier : `modules/nginx/outputs.tf`

```
output "url" {
  value = "http://localhost:${var.external_port}"
```

```
}
```

Question :

- À quoi peut servir cet output dans un projet réel ?

3. Utilisation du module

Fichier : `main.tf`

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
    }
  }
}

provider "docker" {}

module "nginx1" {
  source = "./modules/nginx"

  container_name = "frontend"
  external_port  = 8080
}

module "nginx2" {
  source = "./modules/nginx"

  container_name = "backend"
  external_port  = 8081
}
```

4. Exécution

Dans le dossier `terraform-td2` :

```
terraform init
terraform plan
terraform apply
```

Vérifier :

- http://localhost:8080

- http://localhost:8081

5. Problème volontaire

Fichier : `main.tf`

Modifier la configuration pour utiliser le même port :

```
external_port = 8080
```

Relancer :

```
terraform apply
```

Questions :

- Que se passe-t-il ?
- L'erreur vient-elle de Terraform ou de Docker ?
- Pourquoi ?

Corriger le problème.

6. Exploitation des outputs

```
terraform output
```

Question :

- En quoi cet output est utile dans une chaîne CI/CD ?

7. Compréhension

- Pourquoi utiliser un module plutôt que copier-coller ?
- Quel est le rôle des variables ?
- Que contient le fichier terraform.tfstate ?
- Comment Terraform détecte les changements ?

8. Extension

Fichier : `modules/nginx/variables.tf`

Rendre le module plus flexible :

```
variable "image_name" {
  description = "Image Docker"
  default     = "nginx:latest"
}
```

Modifier :

```
resource "docker_image" "nginx" {
  name = var.image_name
}
```

Test :

- utiliser nginx:alpine

9. Challenge

Créer un troisième environnement :

- nom : data
- port : 8082

10. Bonus

Fichier : `main.tf`

Afficher toutes les URLs :

```
output "urls" {
  value = [
    module.nginx1.url,
    module.nginx2.url
  ]
}
```

Question :

- Comment automatiser encore davantage ce code si on a 10 conteneurs ?