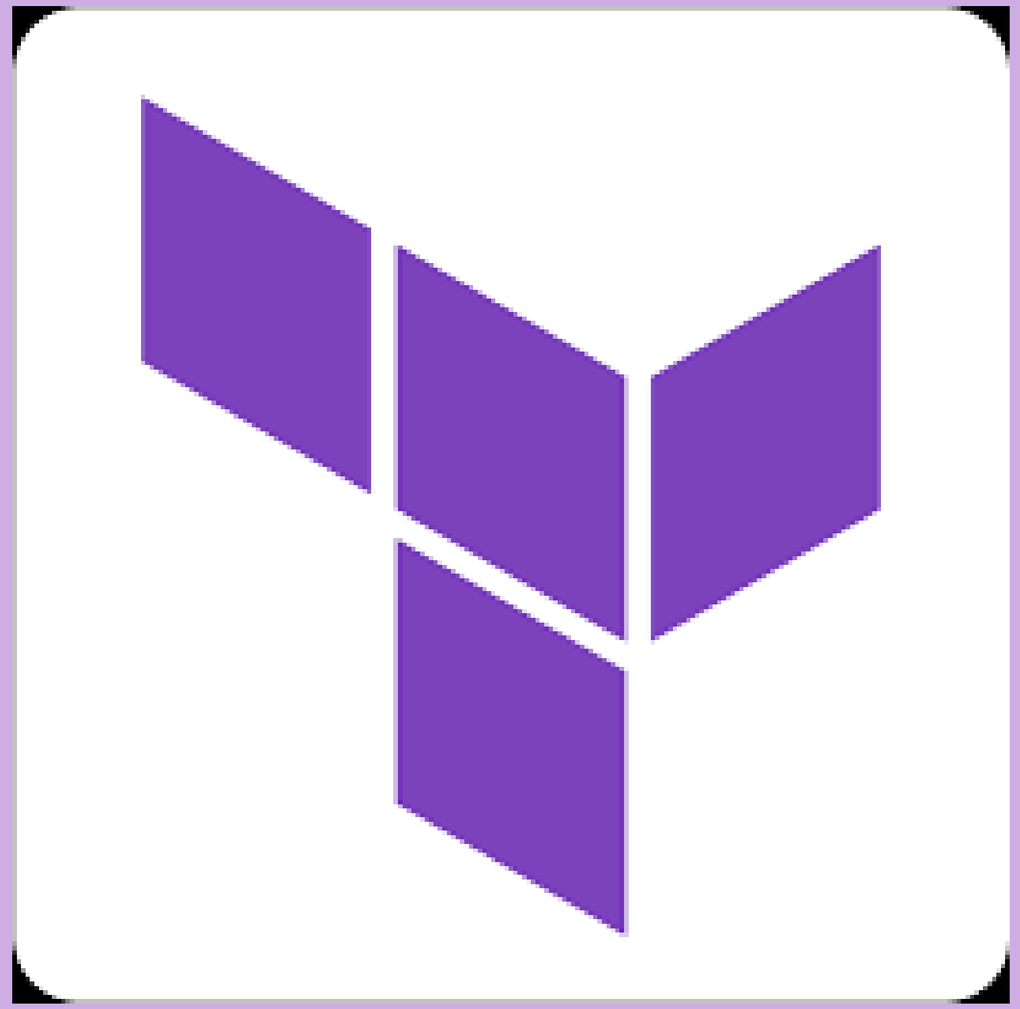
Demo Terraform

Module Terraform AWS EC2 – Infrastructure as Code sécurisée et testable



Par NDEYE FATOU NIASSY

Sommaire

de la présentation

01

Objectifs du projet

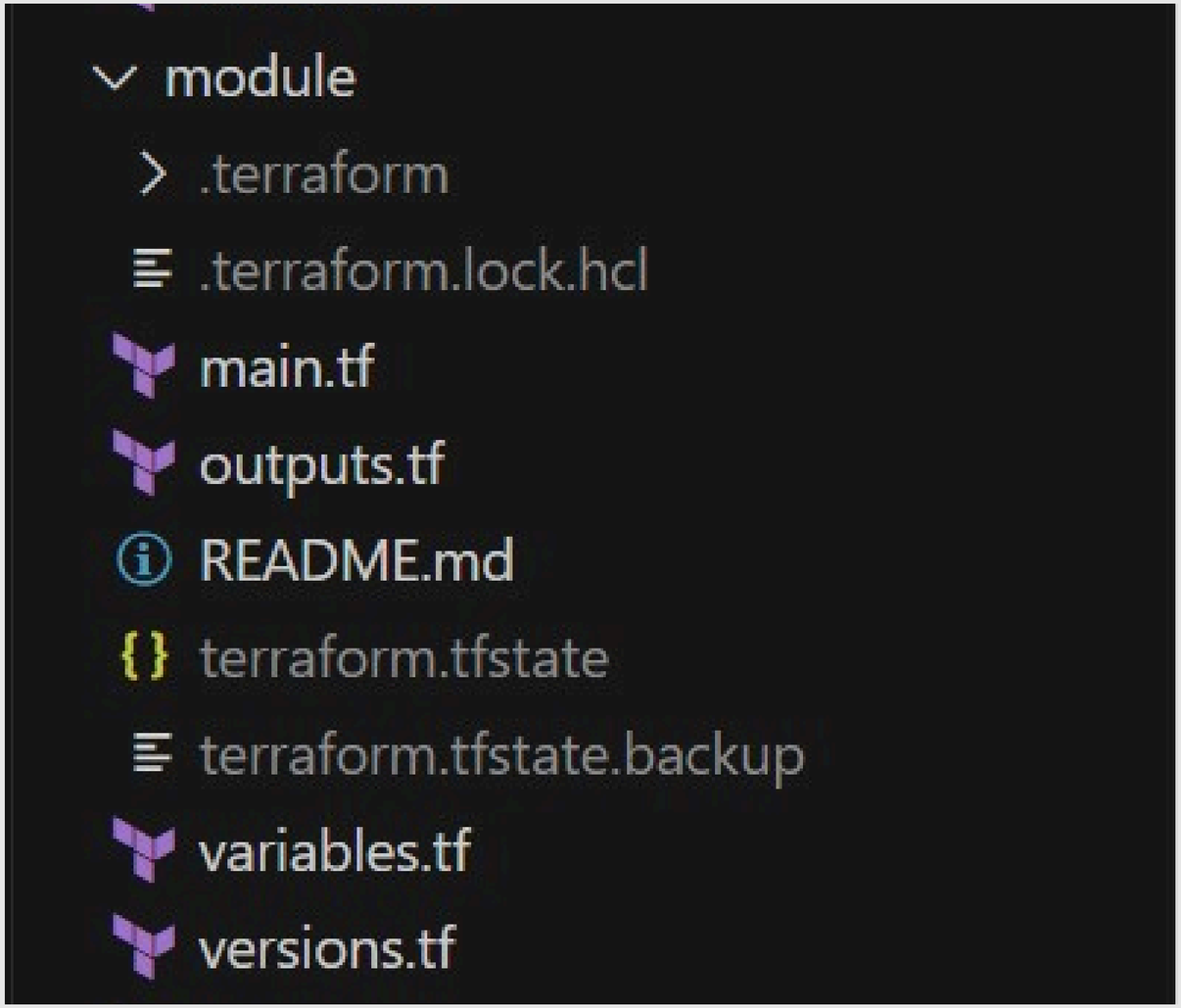
L’objectif ici est donc de créer un module Terraform générique, intégrant des bonnes pratiques de sécurité et des outils DevOps pour le rendre testable et maintenable dans le temps. Le module a été créé en utiliant une instance EC2 de AWS pour créer une VM

Présentation de l’instance EC2 de AWS

Une instance EC2 (Elastic Compute Cloud) est une machine virtuelle proposée par AWS qui permet d’exécuter des applications dans le cloud. Elle offre une capacité de calcul scalable, sécurisée et configurable selon les besoins du projet.

Elle fournit une machine virtuelle à la demande, avec la puissance, la mémoire, le stockage et la configuration réseau souhaités, pour héberger des applications, serveurs web, bases de données, etc.

04 *Architecture et conception*



Présentation du module

main.tf *: Contient la déclaration principale des ressources*

variables.tf *: Déclare les variables d'entrée du module*

outputs.tf *: Définit les valeurs de sortie à exposer*

versions.tf *: Spécifie les versions minimales de Terraform et de AWS*

03 Architecture et conception

```hcl
# EC2 instance resource
resource "aws_instance" "this" {
  ami                    = var.ami_id
  instance_type          = var.instance_type
  key_name               = var.key_name
  vpc_security_group_ids = var.security_group_ids
  subnet_id              = var.subnet_id

  associate_public_ip_address = var.associate_public_ip

  root_block_device {
    volume_size           = var.root_volume_size
    volume_type           = var.root_volume_type
    delete_on_termination = true
    encrypted             = var.encrypt_root_volume
  }

  tags = merge(
    {
      Name = var.name
    },
    var.tags
  )
```

Main.tf : Explication des resources

aws_instance *: Déploie une machine virtuelle EC2*

- *ami : ID de l'image à utiliser (OS préconfiguré)*
- *instance_type : Type de machine (ex: t2.micro)*
- *vpc_security_group_ids et subnet_id : Connecte l'instance au réseau*
- *key_name : Clé SSH pour accès sécurisé*
- *associate_public_ip_address : Attribue une IP publique si activé*
- *root_block_device :*
  - *volume_size : Taille du disque*
  - *encrypted : Chiffrement du volume pour sécurité*
- *metadata_options :*
  - *Active IMDSv2, une bonne pratique de sécurité AWS*
- *monitoring : Active les métriques détaillées CloudWatch*
- *lifecycle.create_before_destroy : Crée une nouvelle instance avant de supprimer l'ancienne*

04 Architecture et conception

```hcl
resource "aws_security_group" "launch-wizard-1" {
  count       = var.create_ssh_security_group ? 1 : 0
  name        = "${var.name}-ssh-sg"
  description = "Security group for SSH access to ${var.name}"
  vpc_id      = var.vpc_id

  ingress {
    description = "SSH access"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = var.ssh_cidr_blocks
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = merge(
    {
      Name = "${var.name}-ssh-sg"
    },
    var.tags
  )
}

```

Main.tf : Explication des resources

aws_security_group : Groupe de sécurité pour accès SSH (port 22)

- *ingress : Autorise les connexions entrantes SSH depuis les IPs spécifiées (cidr_blocks)*
- *egress : Autorise toutes les connexions sortantes*
- *Utilisé de manière conditionnelle avec count = var.create_ssh_security_group ? 1 : 0*

```hcl
resource "aws_eip" "this" {
  count    = var.create_elastic_ip ? 1 : 0
  instance = aws_instance.this.id
  domain   = "vpc"

  tags = merge(
    {
      Name = "${var.name}-eip"
    },
    var.tags
  )
}
```

Main.tf : Explication des resources

aws_eip : Crée une Elastic IP publique, associée à l'instance

Bonnes pratiques DevOps implémentées

Documentation automatisée avec terraform-docs

1. Installation de Terraform_docs sous windows

```
# Avec Chocolatey
choco install terraform-docs
```

2. Configuration via un fichier YAML

Création d'un fichier .terraform-docs.yml à la racine du projet :

```yaml
formatter: markdown table

sections:
  show:
    - requirements
    - providers
    - inputs
    - outputs
    - resources
    - modules

output:
  file: README.md
  mode: inject
  template: |-
    <!-- BEGINNING OF PRE-COMMIT-TERRAFORM DOCS HOOK -->
    {{ .Content }}
    <!-- END OF PRE-COMMIT-TERRAFORM DOCS HOOK -->

sort:
  enabled: true
  by: name

```

3. Puis exécutez :

terraform-docs .

Analyse de sécurité avec Trivy

trivy config --severity HIGH,CRITICAL .

Report Summary

Target	Type	Misconfigurations
exemple/simple	terraform	0
module	terraform	0

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

Trivy permet d'analyser les fichiers Terraform pour détecter les mauvaises configurations pouvant exposer l'infrastructure à des risques de sécurité. Il identifie par exemple les ports ouverts à tous, les ressources non chiffrées, ou encore l'absence de bonnes pratiques. Cela aide à corriger les erreurs avant le déploiement effectif sur AWS . En résumé, Trivy agit comme un scanner de sécurité pour le code Terraform.

04

Bonnes pratiques DevOps implémentées

Tests d'infrastructure avec Terratest

```go
package test

import (
    "fmt"
    "testing"

    "github.com/gruntwork-io/terratest/modules/random"
    "github.com/gruntwork-io/terratest/modules/terraform"
    "github.com/stretchr/testify/assert"
)

func TestTerraformAwsEc2Instance(t *testing.T) {
    t.Parallel()

    uniqueID := random.UniqueId()
    instanceName := fmt.Sprintf("terratest-ec2-%s", uniqueID)
    awsRegion := "us-east-1"

    terraformOptions := &terraform.Options{
        TerraformDir: "../exemple/simple", // Chemin relatif vers votre code Terraform

        Vars: map[string]interface{}{
            "region":        awsRegion,
            "ami_id":        "ami-084568db4383264d4", // Remplacez par un AMI valide dans votre région
            "instance_type": "t2.micro",
            "name":          instanceName,
            "vpc_id":        "vpc-0734c73b4f9dbc03c",    // Remplacez par un VPC valide
            "subnet_id":     "subnet-0e7a0d1908ebd0550", // Remplacez par un Subnet valide
            "key_name":      "my-ssh-key",               // Remplacez par une clé SSH valide
        },

        EnvVars: map[string]string{
            "AWS_DEFAULT_REGION": awsRegion,
        },
    }

    defer terraform.Destroy(t, terraformOptions)
    terraform.InitAndApply(t, terraformOptions)

    instanceID := terraform.Output(t, terraformOptions, "instance_id")
    publicIP := terraform.Output(t, terraformOptions, "instance_public_ip")

    assert.NotEmpty(t, instanceID, "L'ID de l'instance ne doit pas être vide")
    assert.NotEmpty(t, publicIP, "L'adresse IP publique ne doit pas être vide")
}

```

Vérifie que le module Terraform déploie correctement une instance EC2 fonctionnelle sur AWS.

Ce que le test fait :

1. Il génère un nom unique pour l'instance EC2 à tester.
2. Il exécute terraform init puis apply sur le code situé dans ../exemple/simple, avec des variables d'entrée spécifiques (comme l'AMI, le type d'instance, la VPC, le subnet, etc.).
3. Il récupère les outputs Terraform (instance_id et instance_public_ip) et vérifie qu'ils ne sont pas vides, donc que l'instance est bien créée.
4. Enfin, il détruit automatiquement l'infrastructure après le test pour éviter de laisser des ressources inutiles.

Ce test valide donc automatiquement que :

- Le module Terraform fonctionne sans erreur.
- L'instance EC2 est bien lancée avec une IP publique.
- L'infrastructure est éphémère et propre après test.

cd test

go test -v -timeout 30m

```
TestTerraformAwsEc2Instance 2025-05-14T11:46:21Z logger.go:67:
--- PASS: TestTerraformAwsEc2Instance (159.51s)
PASS
ok      github.com/ndeyefatouniassy/terraform_project/test      161.595s
```

Intégration des hooks pre-commit

I. Installation

pre-commit install

terraform_fmt

Ce hook formate automatiquement tous les fichiers .tf selon la syntaxe standard de Terraform.
Cela garantit une cohérence du style dans tout le code.

terraform_validate

Ce hook vérifie que le code Terraform est valide syntaxiquement.
Il empêche les commits si la configuration contient des erreurs ou est malformée.

terraform_trivy

Ce hook utilise Trivy pour détecter des erreurs de configuration ou vulnérabilités potentielles dans le code Terraform.
Il ajoute une couche de sécurité dès la phase de développement.

terraform_docs

Ce hook met automatiquement à jour la documentation (README.md) à partir des variables et outputs du code Terraform.
Cela permet de générer une documentation claire et à jour à chaque modification.

```yaml
repos:
  - repo: https://github.com/antonbabenko/pre-commit-terraform
    rev: v1.99.0
    hooks:
      - id: terraform_fmt
        #language: system  # ← Nécessaire pour Windows
        types: [terraform]
        files: \.tf$

      - id: terraform_validate
       # language: system  # ← Nécessaire pour Windows

      - id: terraform_trivy
       # language: system  # ← Nécessaire pour Windows

      - id: terraform_docs
        name: terraform-docs
        entry: terraform-docs markdown table --output-file README.md --output-
        language: system
        pass_filenames: false

```

```
HP 1030 G2@DESKTOP-NIASSY MINGW64 ~/Bureau/Terraform_Project (master)
$ pre-commit run --all-files
Terraform fmt.....................................................Passed
Terraform validate................................................Passed
Terraform validate with trivy.....................................Passed
terraform-docs....................................................Passed
```

Merci pour votre attention !

Avez-vous des questions ?