TD3 : Stack applicative 3 tiers avec Terraform et Ansible

Objectifs

- Provisionner une stack front / back / db avec Terraform
- Configurer chaque service avec Ansible
- Utiliser les groupes, variables et handlers Ansible
- Identifier les limites d'idempotence d'un playbook

Contexte

Vous travaillez dans une equipe DevOps pour une startup.

L'equipe doit deployer un environnement de developpement compose de :

- un frontend NGINX (page statique)
- un backend Python Flask
- une base de donnees PostgreSQL

Probleme actuel :

- les developpeurs configurent les services a la main
- les environnements different d'une machine a l'autre
- le backend demarre avant la base de donnees et plante

Objectif :

Automatiser le deploiement et la configuration de la stack complete.

1. Preparation du projet

```
mkdir td3-stack
cd td3-stack
mkdir terraform ansible ansible/group_vars
```

Structure attendue :

```
td3-stack/
  terraform/
    main.tf
    variables.tf
    outputs.tf
  ansible/
    inventory.ini
```

```
    playbook.yml
    group_vars/
      frontend.yml
      backend.yml
      database.yml
```

2. Provisionning Terraform

2.1 Fichier de base

Fichier : `terraform/main.tf`

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

provider "docker" {}

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

resource "docker_image" "flask" {
  name = "python:3.11-slim"
}

resource "docker_network" "stack" {
  name = "td3-network"
}

resource "docker_container" "frontend" {
  name  = "frontend"
  image = docker_image.nginx.name

  networks_advanced {
    name = docker_network.stack.name
  }

  ports {
    internal = 80
    external = 8080
  }
}

resource "docker_container" "backend" {
  name  = "backend"
  image = docker_image.flask.name
```

```
  networks_advanced {
    name = docker_network.stack.name
  }

  ports {
    internal = 5000
    external = 5000
  }

  command = ["sleep", "infinity"]
}

# A COMPLETER : image et conteneur PostgreSQL
# port interne : 5432, port externe : 5432
# variable d'environnement POSTGRES_PASSWORD = "secret"
# connecte a td3-network
```

2.2 Outputs

Fichier : `terraform/outputs.tf`

```
output "frontend_url" {
  value = "http://localhost:8080"
}

output "backend_url" {
  value = "http://localhost:5000"
}

output "database_host" {
  value = "database"
}
```

2.3 Deploiement

```
cd terraform
terraform init
terraform apply
```

Verifier que les trois conteneurs sont en cours d'execution :

```
docker ps
docker network inspect td3-network
```

3. Inventaire Ansible

Fichier : `ansible/inventory.ini`

```
[frontend]
localhost ansible_connection=local

[backend]
localhost ansible_connection=local

[database]
localhost ansible_connection=local
```

4. Variables par groupe

Fichier : `ansible/group_vars/frontend.yml`

```
container_name: frontend
html_content: "<h1>Frontend - TD3</h1>"
```

Fichier : `ansible/group_vars/backend.yml`

```
container_name: backend
flask_app: |
  from flask import Flask
  app = Flask(__name__)

  @app.route('/')
  def index():
      return 'Backend OK'

  if __name__ == '__main__':
      app.run(host='0.0.0.0', port=5000)
packages:
  - flask
```

Fichier : `ansible/group_vars/database.yml`

```
container_name: database
db_user: postgres
db_name: appdb
db_password: secret
```

5. Playbook

Fichier : `ansible/playbook.yml`

```
---
- name: Configuration frontend
  hosts: frontend
  gather_facts: false

  tasks:
    - name: Copier la page HTML dans le conteneur
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: >
          bash -c "echo '{{ html_content }}' > /usr/share/nginx/html/index.html"

    - name: Recharger nginx
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: nginx -s reload

- name: Configuration backend
  hosts: backend
  gather_facts: false

  tasks:
    - name: Installer les dependances Python
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: pip install {{ packages | join(' ') }}

    - name: Copier le code Flask
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: >
          bash -c "cat > /app.py << 'EOF'
          {{ flask_app }}
          EOF"

    - name: Demarrer Flask
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: bash -c "nohup python /app.py &"

- name: Configuration base de donnees
  hosts: database
  gather_facts: false

  tasks:
    - name: Attendre que PostgreSQL soit pret
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: pg_isready -U {{ db_user }}
```

```
      register: pg_ready
      retries: 10
      delay: 3
      until: pg_ready.rc == 0

    - name: Creer la base de donnees applicative
      community.docker.docker_container_exec:
        container: "{{ container_name }}"
        command: psql -U {{ db_user }} -c "CREATE DATABASE {{ db_name }};"
```

5.1 Premier lancement

```
cd ansible
ansible-playbook -i inventory.ini playbook.yml
```

Verifier que les services repondent :

```
curl http://localhost:8080
curl http://localhost:5000
```

5.2 Deuxieme lancement

Relancer le playbook sans modifier quoi que ce soit :

```
ansible-playbook -i inventory.ini playbook.yml
```

Observer attentivement les erreurs.

Questions :

- Quelles taches echouent ? Pour quelle raison precise ?
- Parmi les taches qui reussissent, lesquelles produisent quand meme un effet indesirable ?
- Qu'est-ce que cela implique si ce playbook est execute automatiquement dans une pipeline CI/CD ?
- Comment corrigeriez-vous chaque tache concernee ?

6. Challenge

Ajouter un reverse proxy devant la stack.

Specifications :

- image : nginx:latest
- nom du conteneur : proxy
- port externe : 80
- connecte a td3-network
- rediriger / vers le frontend et /api vers le backend

Contrainte : le playbook Ansible de configuration du proxy doit etre idempotent.

7. Bonus

Generer l'inventaire Ansible automatiquement depuis les outputs Terraform.

```
cd terraform
terraform output -json > ../ansible/tf_outputs.json
```

Ecrire un script qui lit `tf_outputs.json` et produit un fichier `inventory.ini` valide.

Le resultat doit etre identique a l'inventaire ecrit manuellement en section 3.

Question :

- Dans quel cas cet inventaire genere devient-il indispensable par rapport a un inventaire ecrit a la main ?