TD1 : Introduction à l'Infrastructure as Code (IaC)

Objectifs

- Comprendre les limites des approches manuelles
- Découvrir les principes de l'Infrastructure as Code
- Manipuler Terraform et Ansible sur un exemple simple

Contexte

Vous intégrez une équipe DevOps.

Actuellement :

- les développeurs lancent des conteneurs manuellement
- les configurations sont faites "à la main"
- les environnements sont incohérents

Problèmes :

- erreurs humaines fréquentes
- impossible de reproduire un environnement
- perte de temps en debug

Objectif :

Automatiser :

- la création d'un service web (Terraform)
- sa configuration (Ansible)

1. Préparation de l'environnement

1.1 Prérequis

- Linux / macOS / WSL recommandé sous Windows
- Docker installé
- Accès terminal

1.2 Vérification outils



Vérifier :

- terraform version
- ansible –version
- docker –version



Versionner ses outils :

- garantit la reproductibilité des environnements,
- évite les incompatibilités entre équipes,
- permet de diagnostiquer plus facilement les problèmes liés aux différences de versions.

2. Arborescence du projet

Fichier : terminal

```
mkdir iac-demo
cd iac-demo
mkdir terraform ansible
```

Structure :

```
iac-demo/
├── terraform/
│   └── main.tf
└── ansible/
    ├── inventory.ini
    └── playbook.yml
```

3. Création de l’infrastructure avec Terraform

3.1 Objectif

Créer un conteneur NGINX

3.2 Configuration

Fichier : `terraform/main.tf`

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

provider "docker" {}
```

```
resource "docker_container" "nginx" {
  image = "nginx:latest"
  name  = "mon_nginx"

  ports {
    internal = 80
    external = 8090
  }
}
```

3.3 Exécution

Fichier : terminal

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



Validation :

- Ouvrir http://localhost:8090

3.4 Observation

Fichier : terminal

```
terraform state list
```



Le state Terraform contient l'état réel des ressources gérées (IDs, propriétés, dépendances).

Terraform s'en sert pour savoir ce qui existe déjà, calculer les changements à appliquer et éviter de recréer inutilement des ressources.

4. Configuration avec Ansible

4.1 Inventory

Fichier : `ansible/inventory.ini`

```
localhost ansible_connection=local
```

4.2 Playbook

Fichier : `ansible/playbook.yml`

```
- hosts: localhost
  connection: local

  tasks:
    - name: créer une page web temporaire
      copy:
        dest: /tmp/index.html
        content: |
          <h1>IaC avec Ansible</h1>

    - name: copier la page dans le conteneur nginx
      command: docker cp /tmp/index.html mon_nginx:/usr/share/nginx/html/index.html
```

4.3 Exécution

Fichier : terminal

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



Validation :

- Rafraîchir http://localhost:8090

5. Problème volontaire

Modifier le nom du conteneur dans Terraform :

Fichier : `terraform/main.tf`

```
name = "mon_nginx_v2"
```

Relancer :

```
terraform apply
```

Puis rejouer Ansible :

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



- Terraform recrée le conteneur avec un nouveau nom (mon_nginx_v2).
- Ansible échoue car il tente toujours de cibler l'ancien conteneur (mon_nginx), qui n'existe plus.
- Le problème vient d'un manque de synchronisation entre Terraform (provisioning) et Ansible (configuration).

6. Correction

Corriger le playbook pour utiliser le bon nom de conteneur.



Pour éviter ce problème, il faut :

- centraliser les variables (ex : nom du conteneur),
- utiliser des outputs Terraform ou une source dynamique pour Ansible,
- éviter les valeurs codées en dur.

7. Compréhension globale



- Le déclaratif décrit l'état final souhaité (Terraform), tandis que le procédural décrit les étapes à exécuter (Ansible).
- Terraform et Ansible sont séparés pour distinguer le provisioning (infrastructure) de la configuration (logicielle).
- Relancer Terraform est idempotent : si rien n'a changé, aucune modification n'est appliquée.

8. Extensions (progression libre)

Objectif :

- consolider la compréhension
- préparer le TD suivant

Extension 1 — Modification simple



Modifier votre configuration :



- changer le port exposé (ex : 8085)
- modifier le contenu de la page HTML

Appliquer les changements.



Terraform modifie la ressource si le changement est compatible avec une mise à jour. Sinon, la ressource est recréée.

Ce comportement dépend du type de ressource et des contraintes du provider Docker.

Extension 2 — Ajouter un second conteneur



Créer un second conteneur :

- nom : nginx_test
- port : 8091

1. Adapter votre configuration Terraform.
2. Adapter votre playbook Ansible pour configurer les deux conteneurs.



Questions :

- Quelles parties du code avez-vous dupliqué ?
- Est-ce problématique ?

Extension 3 — Multiplier les environnements



Créer un troisième conteneur :

- nom : nginx_data
- port : 8092

Observer votre code :



L'augmentation du nombre de conteneurs rend le code difficile à lire et à maintenir :

- Le nombre de lignes dupliquées augmente fortement.
- Avec un grand nombre de conteneurs, le code devient lourd, répétitif et source d'erreurs.

Extension 4 — Réflexion

Questions :

- Quel est le principal problème de votre configuration actuelle ?
- Comment pourriez-vous éviter de copier-coller ce code ?

Bonus

Question :

Existe-t-il un moyen en Terraform de réutiliser du code ?

9. Points clés

- Terraform → provisioning
- Ansible → configuration
- séparation des responsabilités
- reproductibilité

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/eadl/bloc4/fm2/td1?rev=1777816192

Last update: **2026/05/03 15:49**