

# Introduction à l'Infrastructure as Code (IaC)

## Objectifs

- Comprendre les limites des approches traditionnelles
- Identifier les problématiques résolues par l'IaC
- Découvrir les principes fondamentaux de l'IaC
- Distinguer les rôles de Terraform et Ansible

## 1. Problèmes sans Infrastructure as Code

### 1.1 Gestion manuelle des infrastructures

Dans une approche classique :

- Création manuelle des machines (console cloud)
- Configuration à la main (SSH, scripts)
- Déploiements non standardisés

Problèmes :

- Erreurs humaines fréquentes
- Temps de déploiement long
- Difficulté à reproduire un environnement

### 1.2 Dérive de configuration (Configuration Drift)

Définition :

Un système évolue au fil du temps et ne correspond plus à sa configuration initiale.

Exemple :

- Un serveur en production a été modifié manuellement
- L'environnement de test n'est plus identique
- Bugs impossibles à reproduire

### 1.3 Non reproductibilité

Sans automatisation :

- Impossible de recréer rapidement un environnement
- Dépendance à des connaissances implicites
- Documentation souvent obsolète

## 1.4 Passage à l'échelle difficile

Exemple :

- Créer 1 serveur → faisable à la main
- Créer 50 serveurs → ingérable

Problèmes :

- Temps
- cohérence
- erreurs cumulées

## 1.5 Manque de traçabilité

Questions difficiles :

- Qui a modifié quoi ?
- Quand ?
- Pourquoi ?

## 1.6 Transition

L'infrastructure devient :

- trop complexe
- trop dynamique
- trop critique

⇒ Besoin d'automatisation et de standardisation

# 2. Concepts de l'Infrastructure as Code

## 2.1 Définition

Infrastructure as Code :

Décrire l'infrastructure sous forme de code afin de pouvoir :

- versionner
- automatiser
- reproduire

## 2.2 Principes fondamentaux

### Déclaratif vs procédural

- Déclaratif : décrire l'état cible

- Procédural : décrire les étapes

## Idempotence

Appliquer plusieurs fois une configuration produit le même résultat

## Versioning

- Code stocké dans Git
- Historique des modifications
- rollback possible

## Automatisation

- Déploiement rapide
- Réduction des erreurs humaines

## Reproductibilité

- Même code = même infrastructure

## 2.3 Cycle de vie

- Écriture du code
- Planification (diff)
- Application
- Mise à jour
- Destruction

## 3. Deux approches complémentaires

### 3.1 Vision globale

Deux grandes catégories d'outils :

- Provisioning d'infrastructure
- Configuration des systèmes

### 3.2 Terraform

Rôle :

- Provisionner l'infrastructure

Exemples :

- VM (EC2)

- Réseau (VPC, subnets)
- Load balancer
- Base de données

Caractéristiques :

- Approche déclarative
- Gestion d'un état (state)
- Interaction via API cloud

## 3.3 Ansible

Rôle :

- Configurer les systèmes
- Déployer des applications

Exemples :

- Installer NGINX
- Configurer un service
- Déployer une application

Caractéristiques :

- Approche orientée tâches
- Exécution séquentielle
- Idempotence

## 3.4 Comparaison

Terraform :

- Déclaratif
- Orienté infrastructure
- Gère un état global

Ansible :

- Orienté tâches
- Configuration logicielle
- Pas de state central

## 3.5 Complémentarité

Dans un workflow réel :

- Terraform crée l'infrastructure
- Ansible configure les serveurs

## 4. Conclusion

L'Infrastructure as Code permet :

- d'industrialiser les déploiements
- de fiabiliser les environnements
- de gagner en rapidité et en reproductibilité

Terraform et Ansible ne sont pas concurrents mais complémentaires.

## 5. Question de réflexion

Pourquoi est-il risqué de modifier une infrastructure manuellement en production ?

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/eadi/bloc4/fm2/intro?rev=1777075475>

Last update: **2026/04/25 02:04**

