Introduction à l'Infrastructure as Code (IaC)

Objectifs

À la fin de cette séance, vous serez capable de :

- Comprendre les limites de la gestion manuelle d’infrastructure
- Expliquer le concept d’Infrastructure as Code
- Distinguer approche déclarative et impérative
- Comprendre le rôle de Terraform et Ansible dans un workflow réel

1. Mise en situation

Vous arrivez dans une entreprise.

Un développeur vous dit :

```
"Pour lancer l’environnement :
 - démarre un conteneur
 - ouvre le port 8080
 - copie les fichiers à la main"
```

Un autre développeur fait différemment.

Résultat :

- environnements différents
- bugs difficiles à reproduire
- perte de temps

Question :

- Quel est le principal problème ici ?

2. Problème : gestion manuelle

En pratique, sans automatisation :

- actions manuelles (console, SSH)
- dépend de la personne
- pas versionné

Exemples concrets :

- un port ouvert “temporairement” jamais refermé
- configuration différente entre dev et prod
- perte d’informations lors d’un redéploiement

Question :

- Pourquoi ces problèmes sont difficiles à corriger ?

3. Micro-exercice : reproduction d’un environnement

Objectif :

Comprendre la difficulté de reproduire un environnement sans automatisation

Étape 1

Fichier : terminal

```
docker run -d -p 8081:80 --name test_nginx nginx
```

Validation :

- Ouvrir http://localhost:8081

Étape 2

Fichier : terminal

```
echo "<h1>Version 1</h1>" > index.html
docker cp index.html test_nginx:/usr/share/nginx/html/index.html
```

Validation :

- Rafraîchir la page

Étape 3

Fichier : terminal

```
docker rm -f test_nginx
docker run -d -p 8081:80 --name test_nginx nginx
```

Questions :

- Que s'est-il passé ?
- Pourquoi la modification a disparu ?
- Seriez-vous capable de refaire exactement la même configuration ?

4. Transition

Sans automatisation :

- les modifications sont perdues
- rien n'est tracé
- difficile à reproduire

Question :

- Comment éviter ce problème dans une équipe ?

5. Définition de l'Infrastructure as Code

Principe :

- décrire l'infrastructure avec du code
- versionner ce code (Git)
- exécuter automatiquement

Analogie :

- avant : configuration "à la main"
- après : recette écrite et reproductible

6. Exemple simple

Objectif :

- obtenir un serveur web fonctionnel

Sans IaC :

- commandes manuelles
- étapes non tracées

Avec IaC :

- fichier Terraform → crée le serveur
- playbook Ansible → configure le serveur

Résultat :

- reproductible
- automatisé
- partagé avec l’équipe

7. Déclaratif vs Impératif

Approche impérative

On décrit les étapes :

- installer nginx
- démarrer le service

Limite :

- dépend de l’état initial

Approche déclarative

On décrit le résultat :

- “je veux un serveur web”

L’outil gère les actions nécessaires

Question :

- Pourquoi cette approche est-elle adaptée au cloud ?

8. Terraform vs Ansible

Terraform :

- crée l’infrastructure
- maintient un état (state)

Ansible :

- configure les systèmes
- exécute des tâches

Exemple :

- Terraform → crée un conteneur
- Ansible → modifie son contenu

9. Workflow DevOps

- Terraform → infrastructure
- Ansible → configuration
- pipeline → déploiement applicatif

Chaîne :

```
Code → Infrastructure → Configuration → Application
```

10. Transition vers le TD

Dans le TD suivant :

- vous allez automatiser ce que vous venez de faire manuellement
- créer un service avec Terraform
- le modifier avec Ansible

Objectif :

- comprendre la complémentarité des outils

11. À retenir

- IaC = infrastructure gérée comme du code
- objectif : reproductibilité et fiabilité
- Terraform → provisioning
- Ansible → configuration