ESSENTIELS DE L'INFRASTRUCTURE DU SI PARTIE 2

INFRA AS CODE ET APPROCHES

L'Infrastructure as Code (IaC) est la gestion et la provision des infrastructures informatiques par le biais de fichiers de configuration plutôt que par des processus manuels. Bien souvent des fichiers en yaml, oui le langage psychorigide.

Principes Clés :

Automatisation : L'IaC permet d'automatiser la création, la modification et la distribution des infrastructures.

Idempotence : La capacité de lancer le script de configuration plusieurs fois de suite sans changer l'état final de l'infrastructure.

Documentation du Système : Les fichiers de configuration servent de documentation, montrant clairement comment l'infrastructure est configurée.

Avantages :

Consistance et Standardisation : Minimise les erreurs humaines et assure une configuration cohérente à travers l'environnement.

Vitesse et Efficacité : Permet de déployer rapidement de nouvelles infrastructures ou de mettre à jour les existantes.

Traçabilité et Audit : Les changements apportés à l'infrastructure sont suivis et peuvent être audités.

Approche Impérative :

Définition : Décrit les étapes spécifiques nécessaires pour atteindre l'état désiré.

Exemple : Utilisation de scripts qui exécutent une série de commandes dans un ordre spécifique.

Caractéristiques : Plus de contrôle sur le processus, mais peut être plus complexe et difficile à maintenir.

Approche Déclarative :

Définition : Décrit l'état désiré de l'infrastructure sans spécifier explicitement comment y parvenir.

Exemple : Fichiers de configuration Terraform ou Kubernetes qui définissent l'état souhaité.

Caractéristiques : Plus facile à maintenir, peut être plus abstrait et moins contrôlable sur les détails fins.

Comparaison :

Simplicité vs Contrôle : L'approche déclarative est souvent plus simple et plus facile à suivre, tandis que l'impérative offre plus de contrôle sur les détails.

Adaptabilité : Les approches déclaratives s'adaptent mieux aux changements, car elles nécessitent moins de modifications pour ajuster l'état global.

TERRAFORM : SORTIES , GESTION DES ÉTATS ET BONNNES PRATIQUES

Sorties dans Terraform

Les sorties dans Terraform sont utilisées pour extraire des informations sur les ressources créées. Par exemple, après avoir déployé un serveur, vous pouvez utiliser une sortie pour obtenir son adresse IP.

Ces informations peuvent être utilisées pour configurer d'autres outils, ou simplement pour fournir un résumé des ressources déployées à l'utilisateur.

Gestion des États

Terraform stocke l'état de votre infrastructure dans un fichier appelé terraform.tfstate.

Ce fichier d'état est crucial car il contient le mapping entre les objets dans le fichier de configuration Terraform et les ressources réelles dans le cloud.

La gestion sécurisée de ce fichier d'état est essentielle. Il doit être stocké de manière sécurisée et sauvegardé régulièrement.

Bonnes Pratiques

Modularité : Structurez vos configurations Terraform en modules réutilisables pour faciliter la gestion et la maintenance.

Gestion des Versions : Utilisez un système de contrôle de version pour suivre les modifications apportées à vos fichiers Terraform.

Révision de Code : Intégrez des revues de code dans votre processus pour s'assurer que les modifications sont examinées et validées.

Automatisation : Intégrez Terraform dans votre pipeline CI/CD pour automatiser le déploiement de votre infrastructure.

Sécurité : Soyez attentif à la sécurité, notamment en gérant les secrets (comme les clés API) de manière sécurisée et en suivant les principes du moindre privilège.

Depuis 2023, HashiCorp a changé la licence de Terraform (passage à la BSL).

Cela n'affecte pas les usages éducatifs ou personnels, mais interdit l'usage commercial pour des produits concurrents.

Pour les cours et projets non commerciaux, Terraform reste utilisable normalement.

Pour un environnement totalement open-source, vous pouvez utiliser OpenTofu, un fork communautaire libre 100 % compatible avec Terraform 1.5+.

ATELIER 1 : OPENTOFU : MISE EN PRATIQUE DE L'APPROCHE DECLARATIVE

Objectif :

Mettre en œuvre l'Infrastructure as Code à l'aide d'OpenTofu, un outil libre et compatible avec Terraform.

L'objectif est de déployer une petite infrastructure cloud (ex : une instance AWS) de manière déclarative, reproductible et automatisée.

Aucune ressource payante, aucune dépendance à un cloud public.

Étapes :

Étape 1 : Prérequis

Assurez-vous d'avoir :

Docker installé et fonctionnel (docker version)

OpenTofu installé (tofu version)

Installation rapide (Linux) :

```
# Docker
sudo apt update && sudo apt install -y docker.io
sudo systemctl enable --now docker

# OpenTofu
curl -fsSL https://get.opentofu.org/install.sh | sudo bash
tofu version
```

Étape 2 : Créer le projet

Crée un dossier pour ton projet IaC :

```
mkdir opentofu-docker-demo
cd opentofu-docker-demo
```

Étape 3 : Créer le fichier main.tf

Crée un fichier nommé main.tf avec ce contenu :

```hcl
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}

provider "docker" {}

# Ressource 1 : image Docker Nginx
resource "docker_image" "nginx" {
  name = "nginx:latest"
  keep_locally = false
}

# Ressource 2 : conteneur basé sur cette image
resource "docker_container" "nginx_container" {
  image = docker_image.nginx.name
  name  = "demo-nginx"
  ports {
    internal = 80
    external = 8080
  }
}

# Sortie : URL du conteneur
output "nginx_url" {
  value = "http://localhost:8080"
}
```

Étape 4 : Initialiser OpenTofu

Télécharge le provider Docker :

```
tofu init
```

Étape 5 : Planifier et appliquer

Simule la création puis lance le déploiement :

```
tofu plan
tofu apply -auto-approve
```

Vérifier le résultat avec docker ps, tu dois visualiser la page d'accueil nginx.

Étape 7 : Détruire l'infrastructure

Quand tu as fini, détruis tout :

```
tofu destroy -auto-approve
```

Étape 8 : Gestion locale de l'état

OpenTofu crée un fichier terraform.tfstate dans ton dossier.

Il garde la trace des conteneurs créés.

Bonnes pratiques :

Ne jamais le versionner (ajoute-le à .gitignore)

Si tu veux recommencer à zéro :

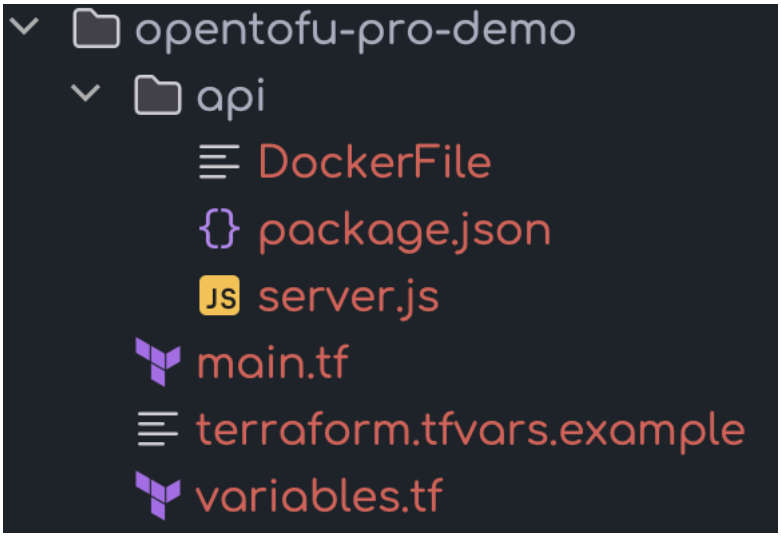
```
rm -f terraform.tfstate*
```

ATELIER 2 : STACK LOCALE "CATALOGUE API" (OPENTOFU + DOCKER)

Ce que tu vas mettre en place :

- API Node.js (port 3000) — simple endpoint /health + ping DB/cache
- PostgreSQL 16 (stockage persistant)
- Redis 7 (cache)
- Réseau Docker dédié + Volumes persistants
- Orchestration OpenTofu (provider kreuzwerker/docker)

Structure:



Guide de réalisation :

1. MISE EN PLACE DU PROJET

- Crée un dossier pour le projet : par exemple opentofu-pro-demo.
- À l'intérieur, tu organiseras une arborescence claire :
  - des fichiers de configuration OpenTofu (main.tf, variables.tf, etc.)
  - un sous-dossier api/ contenant le code de ton microservice Node.js.

L'objectif est de séparer la **définition d'infrastructure** (fichiers .tf) du **code applicatif**.

2. DEFINIR L'INFRASTRUCTURE

Dans cette partie, tu décris **ce que tu veux obtenir** :

- un réseau Docker dédié à la stack ;
- des volumes persistants pour PostgreSQL et Redis ;
- trois conteneurs : **PostgreSQL**, **Redis**, **API** ;
- des règles de dépendances (l'API démarre après la DB et Redis).

L'approche déclarative : tu dis *ce que tu veux*, OpenTofu s'occupe du *comment*.

3. PARAMETRER LES VARIABLES

- Prévois des variables (dans un fichier .tfvars) pour stocker :
  - le port de ton API,
  - le nom de la base,
  - les identifiants PostgreSQL (utilisateur/mot de passe).
- L'idée est de rendre ton infrastructure **réutilisable et personnalisable** sans modifier le code principal.

4. INITIALISER LE PROJET

- Dans le terminal, exécute la commande d'initialisation OpenTofu.
- Cela télécharge les composants nécessaires et prépare le projet à être appliqué.

Si tout est correct, le terminal affiche la détection du provider Docker et la configuration initiale.

5. VERIFIER LE PLAN

- Lance la commande de planification : elle montre **ce qui va être créé** (conteneurs, volumes, réseau).
- Aucune modification n'est encore faite — c'est une **simulation**.
- Ce mécanisme est crucial en entreprise : on *valide* avant d'exécuter.

6. APPLIQUER LES CHANGEMENTS

- Une fois le plan validé, applique-le.
- OpenTofu va alors :
  - créer le réseau Docker,
  - créer les volumes,
  - lancer PostgreSQL et Redis,
  - construire l'image Docker de ton API,

- déployer ton microservice Node.js.

Le processus prend quelques minutes la première fois (téléchargement des images Docker).

7. VERIFIER LE DEPLOIEMENT

- Affiche la liste des conteneurs Docker actifs : tu dois voir trois services.
- Teste ensuite le bon fonctionnement :
  - un endpoint de santé (/health) doit confirmer que PostgreSQL et Redis sont accessibles ;
  - un endpoint /products doit retourner des données de démo.

Ces vérifications simulent les tests de "health check" souvent utilisés dans les pipelines CI/CD.

8. EXPLORER LE RESULTAT

- Ouvre ton navigateur sur le port configuré (par défaut : http://localhost:3000).
- Observe le comportement :
  - les données produits viennent de PostgreSQL,
  - le message de cache vient de Redis,
  - le tout s'exécute dans des conteneurs isolés.

Ce TP illustre la **cohérence entre application et infrastructure** : la stack entière peut être détruite et recréée en un seul fichier.

---

9. NETTOYER L'ENVIRONNEMENT

- Une simple commande OpenTofu suffit pour **tout supprimer** :
  - conteneurs,
  - réseau,
  - configuration.
- Les volumes de données (PostgreSQL, Redis) peuvent être conservés ou supprimés selon ton choix.

Cela montre la puissance d'un outil IaC : *reproductible, réversible, propre.*

Résultat, exécution attendue :

```
# 1) se placer dans le dossier
cd opentofu-pro-demo

# 2) préparer les variables
cp terraform.tfvars.example terraform.tfvars
# (modifie pg_password si tu veux)

# 3) initialiser
tofu init

# 4) revoir le plan
tofu plan

# 5) déployer
tofu apply -auto-approve
```

PUPPET : AUTOMATISATION DE LA CONFIGURATION

Introduction à Puppet

Puppet est un outil de gestion de configuration open-source. Il permet d'automatiser l'administration des infrastructures, notamment la gestion des configurations des serveurs.

Architecture de Puppet : Comprend un serveur Puppet Master et des clients Puppet Agents. Le Master compile et distribue les configurations aux Agents.

Concepts Clés de Puppet

Manifests : Des fichiers Puppet écrits en langage Puppet, qui définissent les ressources et leur état désiré.

Modules : Regroupements de Manifests, templates, et fichiers supplémentaires pour configurer un service spécifique ou une application.

Resources : Éléments de base de la configuration Puppet, comme les utilisateurs, les paquets, et les services.

Automatisation avec Puppet

Définir des Configurations Basiques : Créer un manifest Puppet simple pour gérer des ressources comme les utilisateurs, les paquets, et les services.

Appliquer des Manifests : Utiliser le Puppet Agent pour appliquer les configurations définies sur les serveurs clients.

PUPPET : ORGANISATION DES MANIFESTS DANS DES MODULES

Qu'est-ce qu'un Module dans Puppet ?

Un module est une collection de manifests, de fichiers, de templates, et d'autres ressources qui sont utilisées pour configurer un aspect spécifique d'une machine, comme un service ou une application. Les modules permettent de regrouper des configurations liées et de les réutiliser à travers différents nodes ou projets.

Création d'un Module

1/Structure de Base d'un Module :

Un module Puppet typique contient les répertoires suivants :

- manifests/ : Contient les fichiers manifest (.pp).
- files/ : Contient des fichiers statiques que le module peut gérer.
- templates/ : Contient des templates ERB (Embedded Ruby) pour générer des fichiers de configuration dynamiques.
- examples/ : Contient des exemples d'utilisation du module.

2/Définir un Manifest de Module

Dans le répertoire manifests/, créez un fichier init.pp. C'est le point d'entrée principal du module.

Ce fichier définit une ou plusieurs classes qui encapsulent la configuration que le module gère.

Exemple de Création de Module

Supposons que vous créez un module pour gérer Nginx :

1/Créez la Structure du Module

- Créez un nouveau répertoire pour votre module, par exemple /etc/puppet/modules/nginx.
- À l'intérieur, créez les répertoires manifests/, files/ et templates/.

2/Écrivez la Classe Nginx

Dans /etc/puppet/modules/nginx/manifests, créez un fichier init.pp.

À l'intérieur, définissez une classe nginx qui installe et configure Nginx :

```
class nginx {
  package { 'nginx':
    ensure => installed,
  }
  service { 'nginx':
    ensure => running,
    require => Package['nginx'],
  }
}
```

3/Utilisation du Module dans votre Manifest Principal

Dans votre manifest principal (site.pp), incluez la classe nginx pour appliquer la configuration à un node :

```
node 'server.example.com' {
  include nginx
}
```

Avantages de l'Utilisation des Modules

Réutilisabilité : Les modules peuvent être utilisés dans plusieurs projets ou sur plusieurs nodes.

Maintenabilité : Facilite la gestion et les mises à jour des configurations.

Partage et Collaboration : Les modules peuvent être partagés avec la communauté ou entre membres de l'équipe.

Partie 1 : Création d'une VM Linux

1. Installation de VirtualBox

- Téléchargez et installez VirtualBox depuis ici.

- Ouvrez VirtualBox après l'installation.

2. Création de la VM

- Cliquez sur "Nouvelle" pour créer une nouvelle VM.

- Nommez votre VM (par exemple, "PuppetLab") et sélectionnez "Linux" et "Ubuntu (64-bit)".

- Attribuez au moins 2 Go de mémoire RAM et cliquez sur "Suivant".

3. Configuration du Disque Dur

- Sélectionnez "Créer un disque dur virtuel maintenant".

- Choisissez VDI (VirtualBox Disk Image) et "Dynamiquement alloué".

- Définissez la taille du disque à 20 Go et cliquez sur "Créer".

4. Installation de Linux

- Téléchargez une image ISO d'Ubuntu Server ici.

- Dans VirtualBox, sélectionnez la VM créée, cliquez sur "Démarrer" et choisissez l'image ISO téléchargée.

- Suivez les instructions pour installer Ubuntu sur la VM.

Partie 2 : Installation et Configuration de Puppet

1. Connexion à la VM et Installation de Puppet

- Connectez-vous à votre VM Ubuntu.

- Ouvrez un terminal et exécutez les commandes suivantes pour installer Puppet :

```
wget https://apt.puppet.com/puppet7-release-focal.deb
sudo dpkg -i puppet7-release-focal.deb
sudo apt-get update
sudo apt-get install puppet-agent
```

2. Configuration de Puppet Master et Agent

- Éditez le fichier /etc/puppet/puppet.conf et sous [main], ajoutez :

```
[main]
certname = puppetmaster.example.com
server = puppetmaster.example.com
environment = production
runinterval = 1h
```

Redémarrez le service Puppet avec sudo systemctl restart puppetmaster.

Partie 3 : Création et Application de Manifests Puppet

1. Création d'un Manifest Puppet

- Créez un nouveau fichier dans /etc/puppet/manifests/site.pp et ajoutez :

```
node default {
  package { 'nginx':
    ensure => installed,
  }
  service { 'nginx':
    ensure     => running,
    enable     => true,
    require    => Package['nginx'],
  }
}
```

2. Application du Manifest

- Exécutez sudo puppet agent --test pour appliquer le manifest.

- Vérifiez que Nginx est installé et en cours d'exécution avec systemctl status nginx.

Partie 4 : Automatisation et Gestion des Ressources

1. Création d'une Classe Simple

- Créez un module simple dans /etc/puppet/modules pour gérer un service comme Nginx ou Apache.

ANSIBLE : GESTION ET DEPLOIEMENT DE CONFIGURATIONS

Introduction à Ansible

Ansible est un outil d'automatisation puissant pour la configuration, le déploiement d'applications, et la gestion des tâches. Il utilise une architecture sans agent, en s'appuyant sur SSH.

Playbooks Ansible : Fichiers YAML qui décrivent les tâches à exécuter.

Concepts Clés d'Ansible

Modules : Des unités de code qui Ansible exécute. Chaque module peut contrôler des aspects spécifiques du système, comme les services ou les paquets.

Variables : Utilisées pour gérer les valeurs dynamiques qui peuvent changer selon l'environnement ou le contexte.

Inventaire : Un fichier qui définit les hôtes et les groupes d'hôtes sur lesquels Ansible exécutera les tâches.

Gestion et Déploiement avec Ansible

Écriture de Playbooks Basiques : Créer un playbook Ansible pour automatiser des tâches courantes comme l'installation de logiciels et la configuration de services.

Exécution de Playbooks : Utiliser Ansible pour déployer ces configurations sur les serveurs ciblés.

Meilleures Pratiques : Structurer les playbooks pour la réutilisabilité, gérer efficacement les inventaires, et utiliser les rôles pour organiser les tâches.

MEILLEURES PRATIQUES POUR L'ÉCRITURE DE PLAYBOOKS ANSIBLE

1. Structuration et Organisation

Modularité : Divisez les tâches complexes en rôles et modules réutilisables. Cela aide à organiser le playbook et facilite la maintenance.

Utilisation de l'Inventaire : Définissez des variables et des groupes dans l'inventaire Ansible pour gérer différents environnements (dev, test, prod) de manière efficace.

2. Nommage et Documentation

Noms Clairs et Descriptifs : Utilisez des noms explicites pour les tâches, variables, fichiers et dossiers. Cela rend le playbook plus lisible et compréhensible.

Commentaires et Documentation : Documentez le playbook en ajoutant des commentaires pertinents pour expliquer la logique et les choix effectués.

3. Utilisation de Variables

Centralisation des Variables : Stockez les variables dans des fichiers séparés ou dans l'inventaire pour faciliter leur gestion.

Évitement des Valeurs En Dur : Utilisez des variables au lieu de coder en dur les valeurs, ce qui rend les playbooks plus flexibles et adaptables.

4. Gestion des Erreurs et Idempotence

Gestion des Erreurs : Utilisez les modules failed_when et ignore_errors pour gérer les erreurs de manière appropriée.

Idempotence : Assurez-vous que l'exécution répétée de votre playbook ne modifie pas le système si l'état désiré est déjà atteint.

5. Sécurité et Gestion des Secrets

Ansible Vault : Utilisez Ansible Vault pour chiffrer les mots de passe et autres données sensibles.

Privilèges Minimum Nécessaires : Utilisez become pour les tâches nécessitant des privilèges élevés et évitez de l'exécuter avec des privilèges root inutilement.

6. Tests et Validation

Tests des Playbooks : Testez vos playbooks dans un environnement de staging avant le déploiement en production.

Utilisation de --check et --diff : Utilisez les options --check pour un dry-run et --diff pour voir les changements qui seraient appliqués.

7. Performance et Optimisation

Éviter les Boucles Inutiles : Utilisez les fonctionnalités de boucle d'Ansible uniquement lorsque c'est nécessaire.

Optimisation des Tâches : Fusionnez des tâches similaires et utilisez des modules appropriés pour améliorer la performance.

ATELIER 3 : ANSIBLE : GESTION ET DEPLOIEMENT DE CONFIGURATIONS

Partie 1 : Configuration de la Machine Virtuelle

1/Installation de VirtualBox :

- Avoir VirtualBox installé.

2/Création et Configuration d'une VM Linux :

- Ouvrez VirtualBox, créez une nouvelle VM nommée "AnsibleTarget".
- Sélectionnez "Linux" et "Ubuntu (64-bit)".
- Attribuez 2 Go de RAM et créez un disque dur virtuel de 20 Go.
- Démarrez la VM et installez Ubuntu Server à l'aide de l'image ISO téléchargée.
- Configurez un utilisateur et notez son nom et son mot de passe pour une utilisation ultérieure.

3/Configuration Réseau de la VM :

- Configurer le réseau en mode "Pont" ou "NAT" pour permettre à la machine de contrôle Ansible de communiquer avec la VM.

Partie 2 : Installation et Configuration d'Ansible

1/Installation d'Ansible sur la Machine de Contrôle :

- Ouvrez un terminal et installez Ansible via le gestionnaire de paquets approprié. Par exemple, sur Ubuntu :

```
sudo apt update
sudo apt install -y ansible
ansible --version
```

2/Configuration d'Ansible et de l'Inventaire :

- Créez un fichier d'inventaire dans /etc/ansible/hosts et ajoutez l'adresse IP de la VM sous un groupe [targets].
- Testez la connexion à la VM avec Ansible :

```
ansible all -m ping -u [username] -k
```

Partie 3 : Écriture et Exécution de Playbooks Ansible

1/Création d'un Playbook Simple :

- Créez un playbook Ansible pour installer et démarrer un serveur web (Nginx ou Apache) sur la VM.
- Utilisez YAML pour définir les tâches dans le playbook.

2/Exécution du Playbook :

- Exécutez le playbook pour appliquer la configuration sur la VM :

```
ansible-playbook [nom_du_playbook].yaml -k
```

Remplacez [nom_du_playbook] par le nom de votre fichier playbook.

3/Validation et Tests :

- Vérifiez que le serveur web est installé et en cours d'exécution sur la VM.

```
ansible all -m ping -i /etc/ansible/hosts
```