Sadek Adel, 28/08/2026
Documentation technique - Infrastructure as Code

# Terraform + Proxmox VE 9 + Ansible

Déploiement automatisé d'un conteneur LXC, inventaire dynamique et passage de relais vers Ansible


Objectif : partir d'un Proxmox VE 9.0, déclarer un conteneur Debian avec Terraform, lui attribuer son réseau et une clé SSH, puis laisser Ansible le découvrir automatiquement via l'API Proxmox et appliquer un rôle Wazuh.

## Table des matières

# 1. Introduction

Dans cette documentation nous allons reprendre de A à Z le laboratoire réalisé autour de Terraform, Proxmox VE 9.0 et Ansible. Le but n'est pas seulement de recopier des commandes : chaque étape est expliquée, avec les notions importantes à retenir et les erreurs réellement rencontrées pendant le lab.

Le résultat final est une chaîne d'automatisation complète : Terraform construit l'infrastructure dans Proxmox, Proxmox expose les informations du conteneur via son API, l'inventaire dynamique Ansible récupère ces informations et Ansible configure ensuite la machine en SSH.

> **Idée centrale -** Terraform décrit et maintient l'infrastructure. Ansible décrit et maintient la configuration du système. Les deux outils se complètent au lieu de faire le même travail.

## 1.1 Ce que nous avons construit

| Élément | Valeur utilisée dans le lab |
|---|---|
| Hyperviseur | Proxmox VE 9.0 |
| Nœud Proxmox | ns576493 |
| Terraform | v1.16.0 dans le lab |
| Provider | bpg/proxmox v0.111.1 |
| Stockage | local (type dir) |
| Template LXC | Debian 13 standard 13.6-1 amd64 |
| VMID | 200 |
| Hostname | debian-tf-01 |
| Bridge interne | vmbr1 - 10.10.10.1/24 |
| IP du LXC | 10.10.10.5/24 |
| Passerelle | 10.10.10.1 |
| DNS | 1.1.1.1 |
| Tags Proxmox | terraform ; wazuh_clients |

## 1.2 Architecture logique


- **Terraform :** crée et modifie le conteneur LXC via l'API Proxmox.
- **Proxmox :** héberge le LXC, son disque, son réseau et ses métadonnées comme les tags.
- **Inventaire dynamique Ansible :** interroge Proxmox au moment de l'exécution et transforme les informations de la VM/LXC en hôtes Ansible.
- **SSH :** sert de canal d'administration entre le contrôleur Ansible et le LXC.
- **Ansible :** installe et configure les logiciels, ici l'agent Wazuh.

### 1.3 Petit point théorique : Infrastructure as Code

Avec une approche classique, on clique dans l'interface Proxmox pour créer chaque machine. Avec l'Infrastructure as Code (IaC), on écrit l'état désiré dans des fichiers. Terraform compare cet état avec ce qui existe et calcule les actions nécessaires.

```
Code Terraform : "je veux un LXC avec 1 CPU, 512 Mo de RAM et cette IP"
                        ↓
Terraform compare avec l'état actuel
                        ↓
Terraform propose : créer / modifier / supprimer
                        ↓
Après validation : API Proxmox → infrastructure réelle
```

**À retenir -** Terraform est déclaratif : on décrit le résultat souhaité. On ne programme pas chaque clic ou chaque étape interne de Proxmox.

## 2. Préparer l'environnement de travail

### 2.1 Pourquoi utiliser VS Code

Le laboratoire a été réalisé depuis VS Code avec un terminal distant. VS Code est particulièrement pratique pour Terraform, YAML et Ansible car les extensions détectent la syntaxe, proposent de l'autocomplétion et signalent rapidement les erreurs d'indentation ou de mots-clés.

| Type de fichier | Extension VS Code conseillée |
|---|---|
| Terraform .tf | HashiCorp Terraform |
| YAML .yml / .yaml | YAML - Red Hat |
| Playbooks / rôles Ansible | Ansible - Red Hat |
| Connexion à un serveur Debian | Remote - SSH |

**Conseil -** Pour des fichiers YAML ou Terraform, VS Code est plus confortable que nano. Nano reste utile pour une correction rapide directement sur le serveur.

### 2.2 Prérequis du lab

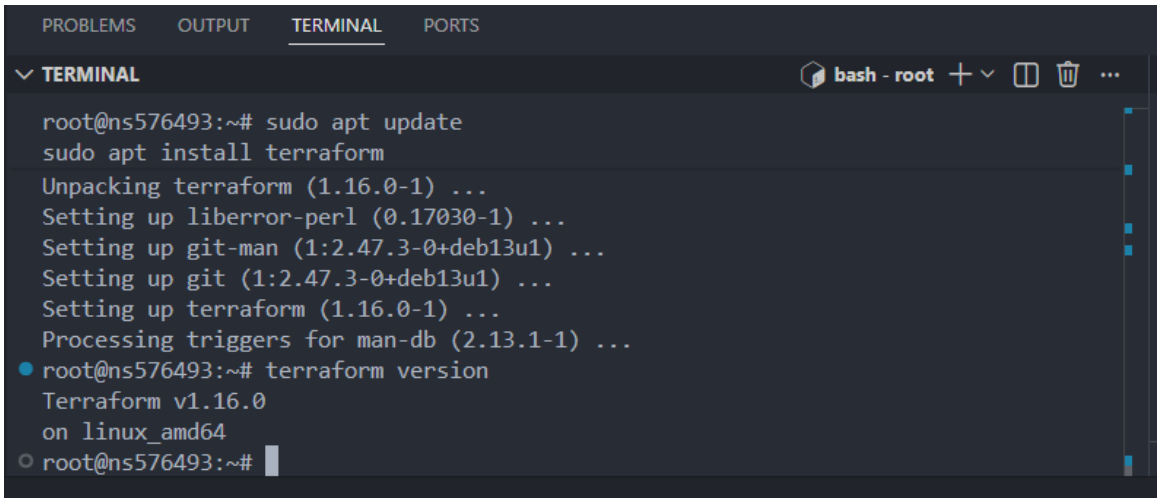
- Un serveur Proxmox VE 9.0 accessible en HTTPS sur le port 8006.
- Un accès root au nœud Proxmox pour créer l'utilisateur API et gérer les templates LXC.
- Un bridge réseau pour le conteneur. Dans le lab : vmbr1, réseau interne 10.10.10.0/24.
- Une machine Debian pouvant exécuter Terraform. Dans notre lab, Terraform a été exécuté directement sur ns576493.
- Ansible installé sur le contrôleur qui devra prendre le relais.

# 3. Installer Terraform sur Debian

Nous avons choisi d'installer Terraform sur Debian plutôt que sur Windows. Le principe reste le même : le CLI Terraform tourne sur une machine de contrôle et communique avec Proxmox via son API HTTPS.

## 3.1 Installation simple utilisée dans le lab

```
sudo apt update
sudo apt install terraform
terraform version
```




*Vérification dans le lab : Terraform v1.16.0 sur linux_amd64.*

> **Comprendre la commande -** apt installe le binaire Terraform. La commande terraform version confirme que le programme est bien dans le PATH et exécutable.

## 3.2 Si le paquet Terraform n'est pas trouvé

Sur une Debian vierge, le dépôt HashiCorp peut ne pas être configuré. Dans ce cas on ajoute le dépôt officiel, puis on installe Terraform :

```
sudo apt-get update
sudo apt-get install -y gnupg software-properties-common wget

wget -O- https://apt.releases.hashicorp.com/gpg |   gpg --dearmor |   sudo tee
/usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(grep -oP '(?<=UBUNTU_CODENAME=).*' /etc/os-release || lsb_release -
cs) main" |   sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update
sudo apt install terraform
terraform version
```

# 4. Créer le projet Terraform et apprendre à lire la documentation du provider

## 4.1 Créer le dossier de travail

```
mkdir -p /root/terraform-proxmox
cd /root/terraform-proxmox
pwd
```

Le dossier du projet contient les fichiers .tf et, après initialisation, le state et le fichier de verrouillage du provider.

## 4.2 Lire correctement Terraform Registry

Le provider utilisé est bpg/proxmox. La documentation du provider est la référence principale pour savoir quels blocs et arguments sont disponibles. Il faut éviter de recopier un exemple ancien trouvé au hasard, car la syntaxe peut changer entre les versions.

1. Ouvrir Terraform Registry et rechercher bpg/proxmox.
2. Vérifier la version affichée. Dans le lab : 0.111.1.
3. Dans Documentation, distinguer Provider, Resources et Data Sources.
4. Pour créer un objet, ouvrir la page Resource correspondante. Exemple : proxmox_virtual_environment_container.
5. Lire d'abord Example Usage pour comprendre la structure générale.
6. Descendre ensuite dans Schema / Argument Reference : Required = obligatoire, Optional = facultatif, Read-Only / Computed = valeur retournée par le provider.
7. Pour un bloc imbriqué comme initialization, ip_config ou user_account, lire les sous-arguments exactement comme une arborescence.

**Documentation du provider :** bpg/proxmox sur Terraform Registry

**Ressource LXC :** proxmox_virtual_environment_container

> **Méthode de lecture -** Quand Terraform affiche "Unsupported block type" ou "Missing name for resource", revenir d'abord à la documentation du resource et comparer la structure caractère par caractère.

## 4.3 Premier main.tf : déclarer le provider

```
terraform {
  required_providers {
    proxmox = {
      source  = "bpg/proxmox"
      version = "0.111.1"
    }
  }
}

provider "proxmox" {
  endpoint = "https://51.222.153.159:8006/"
  insecure = true
}
```

- **terraform.required_providers :** indique quels plugins Terraform doit télécharger.
- **source = "bpg/proxmox" :** identifie le provider dans Terraform Registry.
- **version = "0.111.1" :** fige la version utilisée pendant le lab pour rendre le comportement reproductible.
- **endpoint :** URL de l'API Proxmox. Le port 8006 correspond au proxy/API HTTPS de Proxmox.
- **insecure = true :** accepte le certificat autosigné du lab. À éviter en production si une PKI/certificat valide est disponible.

## 4.4 Initialiser le projet

```
terraform init
```

terraform init télécharge le provider et crée notamment .terraform.lock.hcl. Ce fichier verrouille les versions sélectionnées afin qu'un autre poste utilise les mêmes dépendances par défaut.

| Commande | Rôle |
|---|---|
| terraform init | Initialiser le dossier et télécharger les providers/modules |
| terraform fmt | Formater automatiquement les fichiers .tf |
| terraform validate | Vérifier la syntaxe et la cohérence du code |
| terraform plan | Afficher les changements prévus sans les appliquer |
| terraform apply | Appliquer le plan |
| terraform destroy | Détruire les ressources gérées par le projet |
| terraform state list | Afficher les objets connus dans le state |

## 4.5 Erreur rencontrée : mauvais nom du provider

```
Error: Failed to query available provider packages
Could not retrieve the list of available versions for provider bgp/proxmox:
provider registry registry.terraform.io does not have a provider named
registry.terraform.io/bgp/proxmox
```

Cause : nous avions écrit bgp/proxmox au lieu de bpg/proxmox. Une simple inversion de lettres suffit à faire chercher à Terraform un provider qui n'existe pas.

```
# Incorrect
source = "bgp/proxmox"

# Correct
source = "bpg/proxmox"
```

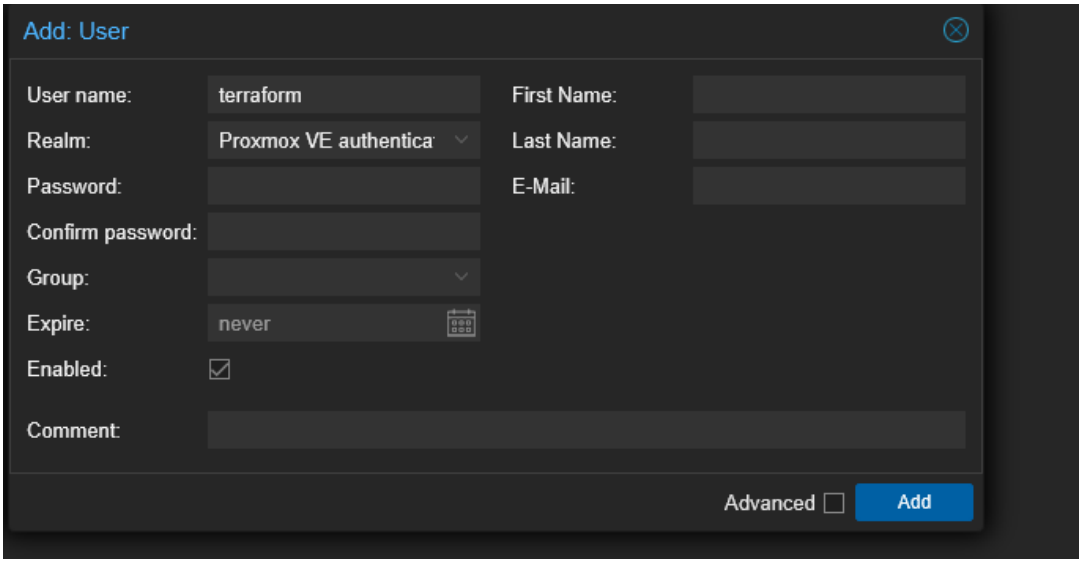
**Rectification -** Après correction du source, terraform init a installé bpg/proxmox v0.111.1 et a terminé par "Terraform has been successfully initialized!".

# 5. Créer un compte API dédié à Terraform dans Proxmox

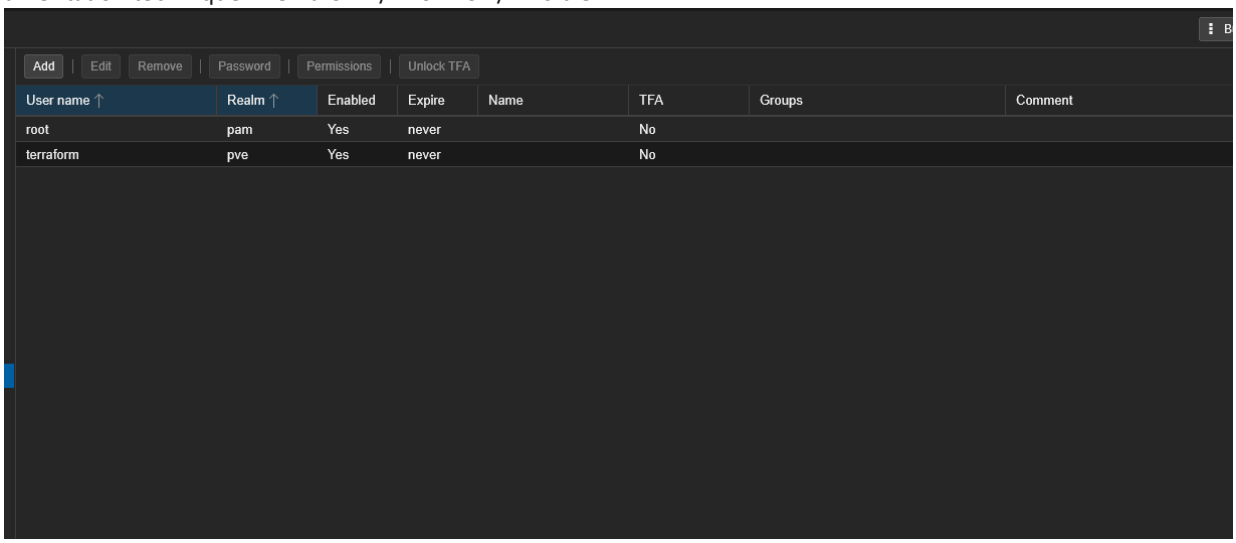
Terraform ne doit pas dépendre du mot de passe root de l'interface Proxmox. Nous avons créé un compte dédié terraform@pve puis un token API. Le provider peut alors s'authentifier sans stocker le secret dans main.tf.

## 5.1 Création de l'utilisateur dans l'interface

Dans Datacenter > Permissions > Users > Add, nous avons créé l'utilisateur terraform dans le realm Proxmox VE authentication server (pve).




*Création de l'utilisateur terraform dans le realm pve.*




*L'utilisateur terraform apparaît ensuite comme activé dans la liste Proxmox.*

## 5.2 Donner les droits pour le laboratoire

L'interface peut être utilisée, mais dans le lab nous sommes passés en CLI. Sur le nœud Proxmox, en root :

```
pveum aclmod / -user terraform@pve -role Administrator
```

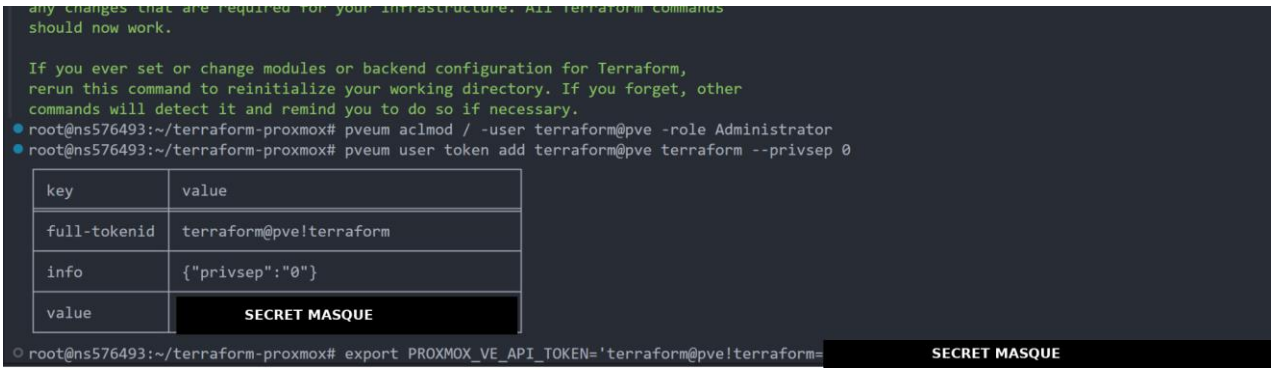
Cette commande donne à terraform@pve le rôle Administrator sur le chemin /, donc à l'échelle du datacenter. C'est volontairement large pour le lab afin de ne pas bloquer l'apprentissage sur les ACL.

**Sécurité -** Administrator est trop permissif pour un environnement de production. La documentation du provider et celle de Proxmox recommandent de créer un rôle adapté aux privilèges réellement nécessaires.

## 5.3 Créer le token API

```
pveum user token add terraform@pve terraform --privsep 0
```

- **terraform@pve :** compte propriétaire du token.
- **terraform :** Token ID choisi.
- **--privsep 0 :** le token n'a pas de jeu de droits séparé et hérite des droits de son utilisateur.




*Création du token API et export de la variable d'environnement. Le secret a été totalement masqué.*

**Important -** La valeur secrète du token n'est affichée qu'au moment de sa création. Il faut la conserver dans un gestionnaire de secrets et ne jamais la versionner dans Git.

### 5.4 Fournir le token à Terraform sans l'écrire dans main.tf

```
export PROXMOX_VE_API_TOKEN='terraform@pve!terraform=TON_SECRET'
```

Le provider bpg/proxmox sait lire automatiquement PROXMOX_VE_API_TOKEN. La syntaxe complète est user@realm!tokenid=secret. Le ! doit être protégé dans le shell, d'où les apostrophes autour de la valeur.

**Durée de vie -** Un export fait dans Bash ne vaut que pour la session courante. En fermant la session, il faudra réexporter la variable ou utiliser un mécanisme de secret persistant sécurisé.

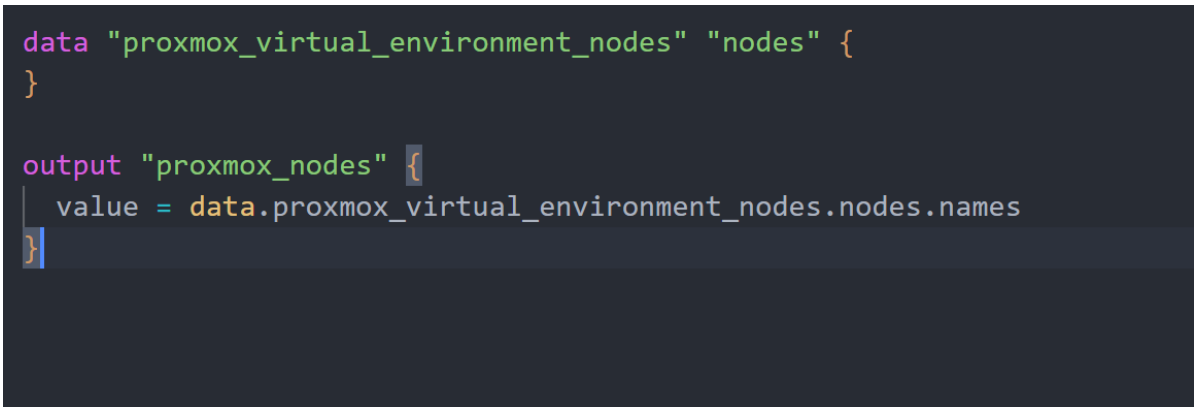
## 6. Tester la communication Terraform → Proxmox avant de créer quoi que ce soit

Avant de lancer la création d'un LXC, nous avons volontairement commencé par une lecture seule de Proxmox. Cela permet de vérifier l'URL, le certificat, le token et les permissions sans modifier l'infrastructure.

### 6.1 Data source et output

```
data "proxmox_virtual_environment_nodes" "nodes" {
}

output "proxmox_nodes" {
  value = data.proxmox_virtual_environment_nodes.nodes.names
}
```

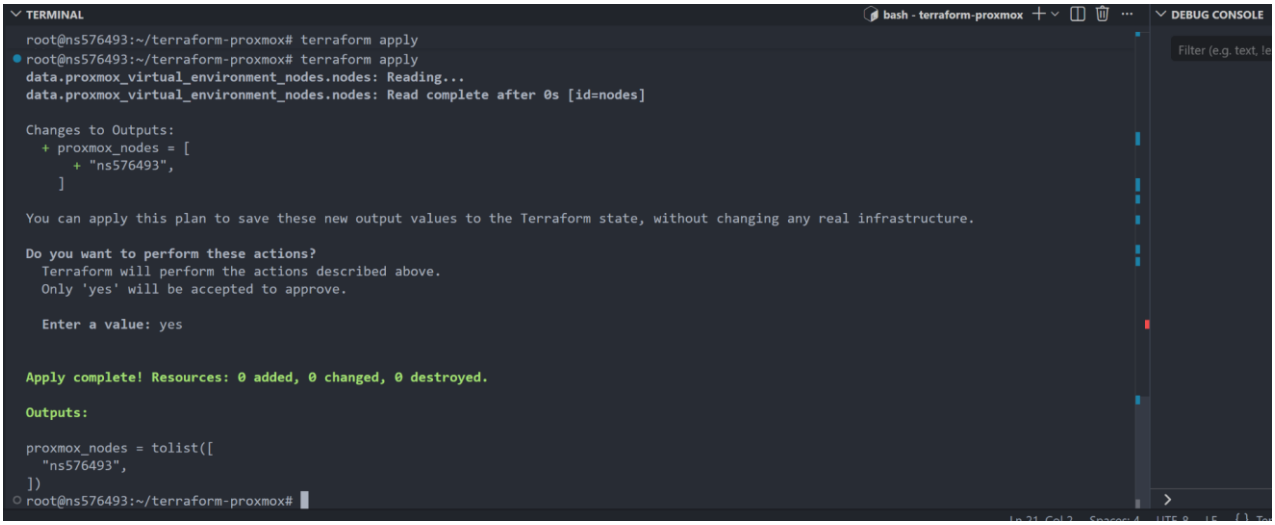



*Bloc data utilisé pour demander la liste des nœuds Proxmox.*

- **data :** lit une information déjà existante. Ici, aucun nœud n'est créé.
- **output :** affiche une valeur calculée ou récupérée par Terraform.
- **nodes.names :** attribut Read-Only fourni par le data source.

## 6.2 Validation et première exécution

```
terraform fmt
terraform validate
terraform plan
terraform apply
```




*Premier apply réussi : Terraform lit le nœud ns576493 et ne modifie aucune ressource.*

Le résultat Resources: 0 added, 0 changed, 0 destroyed est normal : le data source ne crée rien. Il prouve simplement que Terraform a réussi à interroger Proxmox et que le token d'API a été détecté automatiquement.

> **À retenir -** Toujours tester d'abord l'authentification avec une lecture simple avant de créer des ressources. Le diagnostic est beaucoup plus facile.

# 7. Préparer le template LXC Debian

## 7.1 Afficher les templates disponibles

```
pveam update
pveam available
```

pveam est le Proxmox VE Appliance Manager. La liste contient des images système minimalistes et des images TurnKey Linux déjà préparées pour des usages spécifiques (WordPress, Nextcloud, PostgreSQL, etc.).

```
system          debian-12-standard_12.12-1_amd64.tar.zst
system          debian-13-standard_13.6-1_amd64.tar.zst
system          ubuntu-24.04-standard_24.04-2_amd64.tar.zst
...
turnkeylinux    debian-12-turnkey-wordpress_18.2-1_amd64.tar.gz
```

Pour le laboratoire nous avons choisi Debian 13 standard, car nous voulions un système propre et léger à configurer ensuite avec Ansible.

## 7.2 Vérifier le stockage Proxmox

```
pvesm status
```

```
Name   Type  Status  Total (KiB)  Used (KiB)  Available (KiB)
local  dir   active  877919616    775552      877144064
```

Le stockage local est de type dir. Il peut héberger le cache des templates LXC. Le type de contenu Proxmox pour les templates de conteneurs est vztmpl.

## 7.3 Télécharger Debian 13

```
pveam download local debian-13-standard_13.6-1_amd64.tar.zst
```

Proxmox télécharge l'archive, calcule son checksum, vérifie l'intégrité puis la place dans le cache du stockage local.

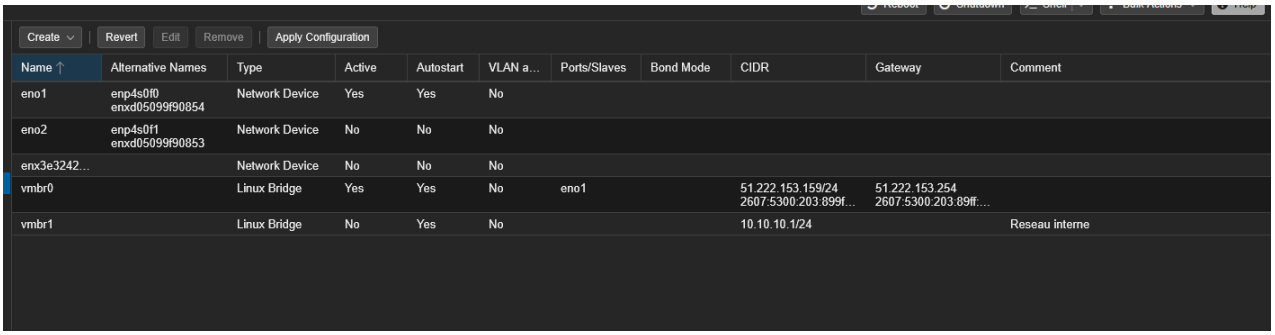
```
pveam list local
```

```
NAME                                                    SIZE
local:vztmpl/debian-13-standard_13.6-1_amd64.tar.zst    123.93MB
```

**Référence importante -** Terraform n'utilise pas simplement le nom du fichier : il référence le volume Proxmox complet local:vztmpl/debian-13-standard_13.6-1_amd64.tar.zst.

# 8. Préparer le réseau interne vmbr1

Le conteneur doit être placé sur un réseau interne. Nous avions déjà créé vmbr1 avec l'adresse 10.10.10.1/24. Ce bridge n'est pas directement relié à une carte physique : il sert de segment virtuel interne. Le routage/NAT vers Internet était déjà configuré en dehors de ce tutoriel.




*vmbr1 est configuré en 10.10.10.1/24, mais la capture montre qu'il était encore "Active: No" avant application.*

## 8.1 Vérifier qu'un bridge existe réellement dans le noyau

```
ip -br link | grep vmbr
```

Au moment de l'erreur, seule vmbr0 apparaissait. Cela signifie que vmbr1 était enregistré dans la configuration de Proxmox mais pas encore appliqué dans le système réseau actif.

Dans l'interface Proxmox, le bouton Apply Configuration permet d'appliquer la configuration. On peut ensuite vérifier :

```
ip -br link | grep vmbr
ip addr show vmbr1
```

# 9. Déclarer le premier conteneur LXC dans Terraform

## 9.1 Comprendre la syntaxe HCL

```
resource "TYPE" "NOM_TERRAFORM" {
  parametre = "valeur"
}
```

Un resource a obligatoirement deux labels : le type du resource puis le nom local utilisé par Terraform. Le hostname du serveur est un autre paramètre et ne remplace pas ce nom Terraform.

> **Exemple -** resource "proxmox_virtual_environment_container" "debian_test" crée une adresse Terraform appelée proxmox_virtual_environment_container.debian_test.

## 9.2 Variables sensibles et clé SSH

Nous avons voulu choisir le mot de passe root à l'avance sans l'écrire en clair dans main.tf. Nous avons également voulu préparer le passage de relais à Ansible en injectant sa clé publique SSH dès la création du LXC.

```
variable "lxc_root_password" {
  type      = string
  sensitive = true
}

variable "ansible_ssh_public_key" {
  type = string
}
```

```
export TF_VAR_lxc_root_password='MOT_DE_PASSE_CHOISI'
export TF_VAR_ansible_ssh_public_key="$(cat /root/.ssh/id_ed25519_ansible.pub)"
```

Terraform associe automatiquement une variable d'environnement nommée TF_VAR_xxx à la variable Terraform xxx.

> **Attention au state -** sensitive = true masque la valeur dans une partie des sorties CLI, mais ne chiffre pas automatiquement terraform.tfstate. Un secret utilisé par une ressource peut être présent dans le state : il faut donc protéger ce fichier.

## 9.3 Générer une clé dédiée à Ansible

Sur le contrôleur Ansible, si aucune clé dédiée n'existe :

```
ssh-keygen -t ed25519 -f /root/.ssh/id_ed25519_ansible -N ""
cat /root/.ssh/id_ed25519_ansible.pub
```

- **id_ed25519_ansible :** clé privée. Elle reste uniquement sur le contrôleur Ansible.
- **id_ed25519_ansible.pub :** clé publique. Elle peut être transmise à Terraform et injectée dans root@debian-tf-01.

**Règle -** On copie la clé publique vers les machines gérées. On ne copie jamais la clé privée sur Proxmox ou dans le LXC.

## 9.4 Ressource LXC finale

```
resource "proxmox_virtual_environment_container" "debian_test" {
  node_name = "ns576493"
  vm_id     = 200

  unprivileged = true

  features {
    nesting = true
  }

  tags = [
    "terraform",
    "wazuh_clients"
  ]

  initialization {
    hostname = "debian-tf-01"

    dns {
      servers = ["1.1.1.1"]
    }

    ip_config {
      ipv4 {
        address = "10.10.10.5/24"
        gateway = "10.10.10.1"
      }
    }

    user_account {
      password = var.lxc_root_password
      keys     = [var.ansible_ssh_public_key]
    }
  }

  operating_system {
    template_file_id = "local:vztmpl/debian-13-standard_13.6-1_amd64.tar.zst"
    type             = "debian"
  }

  cpu {
    cores = 1
  }

  memory {
    dedicated = 512
  }

  disk {
    datastore_id = "local"
    size         = 4
  }

  network_interface {
    name   = "eth0"
    bridge = "vmbr1"
  }
}
```

### 9.5 Lire le plan avant l'apply

```
terraform fmt
terraform validate
terraform plan
```

Dans un plan Terraform, les symboles principaux sont :

| Symbole | Signification |
|---|---|
| + | Création |
| ~ | Modification sur place |
| - | Suppression |
| -/+ | Remplacement : suppression puis recréation |

Pour une première création du LXC, le résultat attendu est Plan: 1 to add, 0 to change, 0 to destroy.

```
terraform apply
```

## 10. Erreurs rencontrées et rectifications

Cette partie est importante : elle montre comment lire les messages Terraform/Proxmox au lieu de recommencer au hasard.

### 10.1 Missing name for resource

```
Error: Missing name for resource
resource "proxmox_virtual_environment_container"{
All resource blocks must have 2 labels (type, name).
```

Cause : le bloc resource ne possédait que le type. Terraform exige un nom local en deuxième label.

```
# Incorrect
resource "proxmox_virtual_environment_container" {

# Correct
resource "proxmox_virtual_environment_container" "debian_test" {
```

### 10.2 Invalid block definition : oubli du signe =

```
# Incorrect
node_name "ns576493"
hostname "debian-tf-01"

# Correct
node_name = "ns576493"
hostname  = "debian-tf-01"
```

En HCL, une affectation simple s'écrit nom = valeur. Sans =, Terraform peut croire que nous essayons de déclarer un nouveau bloc.

## 10.3 Template LXC introuvable : faute dans le nom

```
TASK ERROR: unable to create CT 200 - volume
'local:vztmpl/debian-13-standad_13.6-1_amd64.tar.zst' does not exist
```

Cause : le fichier réellement téléchargé s'appelait debian-13-standard_13.6-1_amd64.tar.zst. Nous avions écrit standad au lieu de standard.

```
pveam list local
# puis recopier exactement la valeur affichée :
local:vztmpl/debian-13-standard_13.6-1_amd64.tar.zst
```

La création avait déjà commencé à formater le disque avant l'échec. En cas de tentative partielle, vérifier pct list et le répertoire /var/lib/vz/images/200 avant de relancer, afin de détecter un conteneur ou un disque orphelin.

```
pct list
ls -lah /var/lib/vz/images/200/
```

## 10.4 bridge vmbr1 does not exist

```
Error: Container start
TASK ERROR: bridge 'vmbr1' does not exist
```

Cause : vmbr1 existait dans la configuration Proxmox, mais il n'était pas actif dans le réseau Linux. La commande ip -br link | grep vmbr ne montrait que vmbr0.

1. Ouvrir Node > System > Network dans Proxmox.
2. Vérifier que vmbr1 existe, Autostart = Yes, CIDR = 10.10.10.1/24.
3. Cliquer sur Apply Configuration.
4. Vérifier à nouveau avec ip -br link | grep vmbr.
5. Si le CT 200 existe déjà mais est arrêté, le démarrer avec pct start 200 puis relancer terraform apply pour resynchroniser le state.

## 10.5 Avertissement Systemd 257 : activer nesting

```
Warning: Container reboot
WARN: Systemd 257 detected. You may need to enable nesting.
```

Debian 13 utilise un systemd récent. Le provider bpg/proxmox documente l'activation de nesting pour les distributions Linux récentes dans les conteneurs non privilégiés.

```
unprivileged = true

features {
  nesting = true
}
```

# 11. Vérifier le conteneur créé

## 11.1 Vérifier son état

```
pct list
```

```
VMID  Status   Name
200   running  debian-tf-01
```

## 11.2 Vérifier la configuration

```
pct config 200
```

```
arch: amd64
cores: 1
hostname: debian-tf-01
memory: 512
nameserver: 1.1.1.1
net0: name=eth0,bridge=vmbr1,gw=10.10.10.1,ip=10.10.10.5/24,type=veth
rootfs: local:200/vm-200-disk-0.raw,size=4G
ostype: debian
onboot: 1
```

Nous retrouvons donc bien l'état déclaré dans Terraform : CPU, mémoire, disque, bridge, IP, passerelle et DNS.

## 11.3 Comprendre le state Terraform

```
terraform state list
```

Le state fait le lien entre le bloc HCL et l'objet réel Proxmox. Sans ce lien, Terraform ne saurait pas que proxmox_virtual_environment_container.debian_test correspond au CT 200.

> **Ne pas confondre -** main.tf décrit l'état désiré. terraform.tfstate mémorise l'état géré et les identifiants. Le state n'est pas une simple sauvegarde facultative : c'est un composant central de Terraform.

# 12. Modifier une ressource existante avec Terraform

Une fois le LXC créé, nous pouvons modifier un paramètre dans le code. Par exemple passer la RAM de 512 à 1024 Mo :

```
memory {
  dedicated = 1024
}
```

```
terraform plan
```

Terraform doit montrer une modification (~) et non une nouvelle création. Après terraform apply, Proxmox est mis en conformité avec le code. C'est l'intérêt principal de l'approche déclarative : les changements passent par le même fichier et restent traçables.

# 13. Préparer le passage de relais SSH vers Ansible

L'objectif n'est pas que Terraform configure tout Debian. Terraform doit s'arrêter après le provisioning de l'infrastructure et l'accès initial. La configuration détaillée (packages, services, fichiers, Wazuh...) est ensuite confiée à Ansible.

## 13.1 Pourquoi injecter une clé SSH plutôt que modifier PermitRootLogin pour les mots de passe

Pour un contrôleur Ansible, l'authentification par clé est plus adaptée. Le bloc initialization.user_account.keys du provider place la clé publique dans le compte root du LXC. Il n'est donc pas nécessaire d'autoriser volontairement root par mot de passe si la configuration SSH autorise déjà root par clé.

> **Séparation des responsabilités -** Terraform fournit l'accès initial (clé publique + IP). Ansible utilise ensuite cet accès pour gérer la configuration du système.

## 13.2 Tester SSH avant Ansible

```
ssh -i /root/.ssh/id_ed25519_ansible root@10.10.10.5
```

Si le template ne possède pas de serveur SSH, Ansible ne pourra pas prendre la main. Dans ce cas, on peut effectuer un bootstrap ponctuel depuis l'hôte Proxmox :

```
pct exec 200 -- apt update
pct exec 200 -- apt install -y openssh-server
pct exec 200 -- systemctl enable --now ssh
```

Dans le lab final, la connexion Ansible a fonctionné et le module ping a retourné pong.

# 14. Mettre en place l'inventaire dynamique Ansible Proxmox

Nous avions déjà un inventaire statique /etc/ansible/hosts avec un groupe [wazuh_clients]. Le problème d'un fichier statique est qu'il faut ajouter chaque nouvelle IP à la main. Proxmox possède déjà les noms, IP, états et tags de ses LXC : Ansible peut donc les récupérer directement via l'API.

## 14.1 Installer les collections nécessaires

```
ansible-galaxy collection install community.proxmox
ansible-galaxy collection install ansible.utils
apt install -y python3-netaddr
```

Dans le lab, community.proxmox était déjà installé : Ansible a répondu "Nothing to do. All requested collections are already installed." Ce message n'est pas une erreur.

## 14.2 Créer le fichier d'inventaire dynamique

```
mkdir -p /etc/ansible/inventory
nano /etc/ansible/inventory/proxmox.proxmox.yml
```

**Nom du fichier -** Le plugin attend un fichier qui se termine par .proxmox.yml ou .proxmox.yaml.

```
plugin: community.proxmox.proxmox

url: "https://51.222.153.159:8006"
user: "terraform@pve"
token_id: "terraform"

validate_certs: false
exclude_nodes: true
want_facts: true
want_proxmox_nodes_ansible_host: false
strict: true

filters:
  - "'terraform' in (proxmox_tags_parsed | default([]))"

groups:
  wazuh_clients: "'wazuh_clients' in (proxmox_tags_parsed | default([]))"

compose:
  ansible_host: proxmox_ipconfig0.ip | default(proxmox_net0.ip) | ansible.utils.ipaddr('address')
  ansible_user: "'root'"
  ansible_python_interpreter: "'/usr/bin/python3'"
```

## 14.3 Donner le secret du token à Ansible

```
export PROXMOX_TOKEN_SECRET='TON_SECRET'
```

Le plugin community.proxmox.proxmox sait lire PROXMOX_TOKEN_SECRET. Comme pour Terraform, cela évite de mettre le secret directement dans le fichier YAML.

**Bonne pratique -** Dans un vrai environnement, on peut utiliser Ansible Vault ou un gestionnaire de secrets. Le même token a été réutilisé pour le lab, mais on peut créer un token Ansible distinct avec des droits de lecture adaptés.

## 14.4 Pourquoi les tags font le lien avec les groupes Ansible

Terraform a placé deux tags sur le LXC : terraform et wazuh_clients. Avec want_facts: true, le plugin récupère proxmox_tags_parsed. L'expression groups ajoute ensuite automatiquement le LXC au groupe wazuh_clients si le tag du même nom existe.

```
tags = [
  "terraform",
  "wazuh_clients"
]
```

Ainsi, on n'ajoute plus manuellement la ligne debian-tf-01 ansible_host=10.10.10.5 ansible_user=root dans /etc/ansible/hosts.

## 14.5 Tester la découverte

```
ansible-inventory -i /etc/ansible/inventory/proxmox.proxmox.yml --graph
```

```
@all:
  |--@proxmox_all_lxc:
  |  |--debian-tf-01
  |--@proxmox_all_running:
  |  |--debian-tf-01
  |--@wazuh_clients:
  |  |--debian-tf-01
```

```
ansible-inventory   -i /etc/ansible/inventory/proxmox.proxmox.yml   --host debian-tf-01
```

Le plugin a récupéré directement depuis Proxmox : l'architecture amd64, le hostname, les interfaces, l'IP 10.10.10.5/24, le DNS 1.1.1.1, le bridge vmbr1, le VMID 200, l'état running et les tags.

## 14.6 Erreur rencontrée : ansible_host absent

```
ansible-inventory ... --host debian-tf-01 | grep ansible
"ansible_user": "root"
```

Au premier test, ansible_user était bien créé mais ansible_host manquait. Le filtre ipaddr utilisé par compose n'était pas disponible correctement. Nous avons installé ansible.utils et python3-netaddr, puis utilisé le filtre qualifié ansible.utils.ipaddr.

```
ansible-galaxy collection install ansible.utils
apt install -y python3-netaddr
```

```
compose:
  ansible_host: proxmox_ipconfig0.ip | default(proxmox_net0.ip) |
ansible.utils.ipaddr('address')
  ansible_user: "'root'"
```

Après correction :

```
"ansible_host": "10.10.10.5",
"ansible_user": "root",
```

**Astuce diagnostic -** strict: true transforme les erreurs de composition ignorées en erreurs explicites. C'est très utile pendant la mise au point de l'inventaire.

## 14.7 Conserver l'inventaire statique existant

Les anciennes machines peuvent rester dans /etc/ansible/hosts. Les nouvelles machines Terraform viennent d'une deuxième source. Ansible peut charger les deux en même temps et fusionner les groupes portant le même nom.

```
ansible-inventory   -i /etc/ansible/hosts   -i /etc/ansible/inventory/proxmox.proxmox.yml   --
graph
```

Le groupe wazuh_clients peut ainsi contenir à la fois les hôtes historiques statiques et les LXC créés dynamiquement par Terraform.

# 15. Valider le passage de relais Terraform → Ansible

## 15.1 Test Ansible ping

```
ansible   -i /etc/ansible/inventory/proxmox.proxmox.yml   wazuh_clients   -m ping
```

```
debian-tf-01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.13"
    },
    "changed": false,
    "ping": "pong"
}
```

Le pong est le point de validation majeur : Ansible a découvert le LXC via l'API Proxmox, a calculé son ansible_host, puis s'est connecté en SSH et a exécuté un module Python dans Debian.

## 15.2 Avertissement Python 3.13

Ansible a indiqué que /usr/bin/python3.13 avait été découvert automatiquement et qu'une installation future d'un autre interpréteur pourrait changer ce choix. Ce n'était pas une erreur. Pour stabiliser le comportement, nous pouvons composer :

```
ansible_python_interpreter: "'/usr/bin/python3'"
```

## 15.3 Lancer le playbook Wazuh sur le seul LXC de test

```
ansible-playbook   -i /etc/ansible/inventory/proxmox.proxmox.yml
/etc/ansible/playbooks/wazuh-agent.yml   --limit debian-tf-01
```

Le --limit est important pendant les tests : il évite d'appliquer le playbook à tous les wazuh_clients existants.

```
PLAY RECAP

debian-tf-01 : ok=8  changed=6  unreachable=0  failed=0
```

- Les prérequis ont été installés.
- La clé GPG Wazuh a été importée.
- Le dépôt Wazuh a été ajouté.
- Le cache APT a été mis à jour.
- Wazuh Agent a été installé.
- Le service a été activé au démarrage puis démarré.

**Résultat -** La chaîne complète a fonctionné sans hôte unreachable et sans tâche failed.

### 15.4 Tester l'idempotence

Une bonne automatisation Ansible doit être idempotente : relancer le même playbook sur une machine déjà conforme ne doit pas répéter inutilement les changements. Il est donc recommandé de relancer le playbook et de vérifier que la majorité des tâches passent de changed à ok.

## 16. Comment fonctionne exactement le passage de main entre Terraform et Ansible

Il n'y a pas un mécanisme magique où Terraform "donne" directement un objet à Ansible. Le passage de main est construit grâce à quatre informations communes : l'infrastructure Proxmox, les tags, l'adresse IP et la clé SSH.

| Étape | Ce qui se passe |
|---|---|
| 1. Terraform | Crée le LXC et configure son hostname, IP, réseau, disque, DNS, tags et clé SSH publique. |
| 2. Proxmox | Enregistre la configuration du CT 200 et l'expose via son API. |
| 3. Inventaire dynamique | Ansible interroge l'API, récupère proxmox_net0.ip et proxmox_tags_parsed, construit ansible_host et les groupes. |
| 4. SSH | Ansible utilise ansible_host et la clé privée correspondante pour ouvrir une session root. |
| 5. Playbook | Ansible applique les rôles et maintient la configuration du système. |

**Point important -** Terraform n'a pas besoin d'écrire directement dans /etc/ansible/hosts. L'inventaire dynamique supprime cette dépendance et évite de synchroniser deux fichiers manuellement.

## 17. Fichier main.tf complet utilisé comme base reproductible

Voici une version consolidée du fichier pédagogique. Le secret Proxmox et le mot de passe root ne sont volontairement pas écrits dans ce fichier.

```
terraform {
  required_providers {
    proxmox = {
      source  = "bpg/proxmox"
      version = "0.111.1"
    }
  }
}

provider "proxmox" {
  endpoint = "https://51.222.153.159:8006/"
  insecure = true
}

variable "lxc_root_password" {
  type      = string
  sensitive = true
}

variable "ansible_ssh_public_key" {
  type = string
}

data "proxmox_virtual_environment_nodes" "nodes" {}

output "proxmox_nodes" {
  value = data.proxmox_virtual_environment_nodes.nodes.names
}

resource "proxmox_virtual_environment_container" "debian_test" {
  node_name = "ns576493"
  vm_id     = 200

  unprivileged = true

  features {
    nesting = true
  }

  tags = ["terraform", "wazuh_clients"]

  initialization {
    hostname = "debian-tf-01"

    dns {
      servers = ["1.1.1.1"]
    }

    ip_config {
      ipv4 {
        address = "10.10.10.5/24"
        gateway = "10.10.10.1"
      }
    }

    user_account {
      password = var.lxc_root_password
      keys     = [var.ansible_ssh_public_key]
    }
  }

  operating_system {
    template_file_id = "local:vztmpl/debian-13-standard_13.6-1_amd64.tar.zst"
    type             = "debian"
  }

  cpu {
    cores = 1
  }

  memory {
    dedicated = 512
  }

  disk {
    datastore_id = "local"
    size         = 4
  }

  network_interface {
    name   = "eth0"
    bridge = "vmbr1"
  }
}
```

### 17.1 Séquence de commandes complète

```
cd /root/terraform-proxmox

export PROXMOX_VE_API_TOKEN='terraform@pve!terraform=SECRET'
export TF_VAR_lxc_root_password='MOT_DE_PASSE_CHOISI'
export TF_VAR_ansible_ssh_public_key="$(cat /root/.ssh/id_ed25519_ansible.pub)"

terraform init
terraform fmt
terraform validate
terraform plan
terraform apply

pct list
pct config 200
```

# 18. Annexe - Versionner les rôles Ansible dans un dépôt GitHub privé

Pendant le lab nous avons également versionné /etc/ansible dans Git. Cette étape n'est pas obligatoire pour Terraform, mais elle est fortement conseillée afin de conserver l'historique des playbooks et rôles.

```
cd /etc/ansible
git init
git add .
git commit -m "Premier push de tout mes job ansible"
git branch -M main

gh auth status
gh repo create ansible-infrastructure --private --source=. --remote=origin --push
```

Sur une autre machine, après connexion à GitHub CLI :

```
gh auth login
cd /etc
gh repo clone ansible-infrastructure ansible
cd /etc/ansible
git status
```

**Sécurité Git -** Avant git add ., vérifier qu'aucun token API, mot de passe, clé privée SSH ou fichier Terraform state contenant des secrets n'est présent dans le dépôt.

# 19. Améliorations recommandées après ce premier laboratoire

- Créer un rôle Proxmox dédié à Terraform au lieu d'utiliser Administrator.
- Créer un token distinct pour l'inventaire Ansible en lecture seule si possible.
- Stocker les secrets dans Ansible Vault, un gestionnaire de secrets ou un backend Terraform protégé.
- Placer le state Terraform dans un backend distant avec verrouillage pour un travail en équipe.

- Séparer main.tf en providers.tf, variables.tf, outputs.tf et lxc.tf lorsque le projet grandit.
- Utiliser for_each pour créer plusieurs LXC à partir d'une map de noms, IP, VMID et tags.
- Créer des modules Terraform réutilisables pour standardiser CPU, RAM, réseau et sécurité.
- Conserver les tags Proxmox comme contrat entre Terraform et les groupes dynamiques Ansible.
- Relancer les playbooks pour contrôler l'idempotence.

## 19.1 Exemple de prochaine évolution avec for_each

L'étape suivante naturelle est de remplacer un seul LXC codé en dur par une map de conteneurs. Terraform pourra alors créer plusieurs machines en une seule exécution et Ansible les découvrira automatiquement grâce aux tags.

```
locals {
  lxc = {
    web01 = { vmid = 201, ip = "10.10.10.11/24", tags = ["terraform", "web"] }
    web02 = { vmid = 202, ip = "10.10.10.12/24", tags = ["terraform", "web"] }
    wazuh = { vmid = 203, ip = "10.10.10.13/24", tags = ["terraform", "wazuh_clients"] }
  }
}

# Puis resource ... for_each = local.lxc
```

**Principe -** Le travail manuel ne doit pas augmenter proportionnellement au nombre de machines. C'est précisément ce que Terraform + inventaire dynamique + Ansible permet d'éviter.

# 20. Mémo des commandes à retenir

| Commande | À quoi elle sert |
|---|---|
| terraform init | Initialiser un projet et télécharger le provider |
| terraform fmt | Reformater le HCL proprement |
| terraform validate | Détecter les erreurs de syntaxe/configuration |
| terraform plan | Prévisualiser les changements |
| terraform apply | Appliquer les changements |
| terraform state list | Lister les objets gérés |
| pveam available | Lister les templates LXC disponibles |
| pveam download local <template> | Télécharger un template sur le stockage local |
| pveam list local | Lister les templates déjà téléchargés |
| pvesm status | Afficher les stockages Proxmox |
| pct list | Lister les conteneurs LXC |
| pct config 200 | Afficher la configuration du CT 200 |
| pct enter 200 | Entrer dans le LXC depuis l'hôte Proxmox |
| pveum aclmod ... | Modifier les permissions Proxmox |
| pveum user token add ... | Créer un token API |
| ansible-inventory ... --graph | Visualiser les hôtes/groupes de l'inventaire dynamique |
| ansible ... -m ping | Tester la connexion et l'exécution Ansible |
| ansible-playbook ... --limit hôte | Appliquer un playbook à un hôte précis |

# 21. Conclusion

Nous avons construit une base complète et reproductible : Terraform communique avec Proxmox VE 9 via un token API, télécharge et utilise un template Debian 13, crée un LXC avec réseau statique, DNS, tags et accès SSH, puis Ansible découvre ce LXC automatiquement et applique un playbook Wazuh.

La partie la plus importante n'est pas la création du CT 200 en elle-même, mais la séparation propre des responsabilités. Terraform garde la maîtrise de l'infrastructure. Proxmox expose l'état réel. L'inventaire dynamique transforme cet état en inventaire Ansible. Ansible configure ensuite le système. Nous pouvons désormais ajouter plusieurs LXC sans recopier manuellement leurs IP dans le fichier hosts.

**Validation finale du lab -** Terraform : OK - LXC 200 running - inventaire dynamique : OK - ansible_host 10.10.10.5 : OK - SSH/Ansible ping : pong - playbook Wazuh : failed=0.

# 22. Références officielles utiles

- Installation Terraform - HashiCorp
- Provider bpg/proxmox - Terraform Registry
- Ressource LXC bpg/proxmox
- Data source Proxmox nodes
- Variables sensibles Terraform - HashiCorp
- Inventaire dynamique community.proxmox.proxmox - Ansible
- Guide d'administration Proxmox VE

*Document réalisé par Sadek Adel - Lab Terraform / Proxmox / Ansible - 28/08/2026*