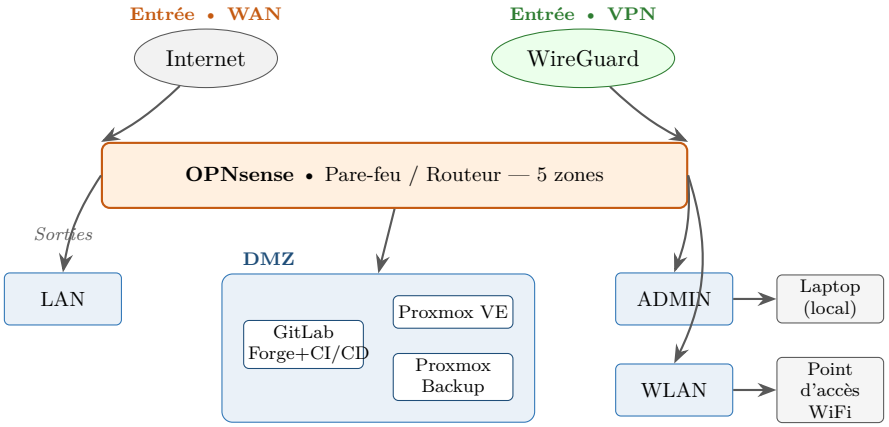
Infrastructure LinuXmaine v2.0

Provisionnement *Infrastructure as Code*

Proxmox • OpenTofu • Ansible • Cloud-init



Document technique

Auteur	Thierry Gayet
Courriel	Thierry.Gayet@gmail.com
Organisation	LinuXmaine.org
Dépôt IaC	framagit.org/linuxmaine/infra-linuxmaine-v2.0
Date	31 mai 2026
Version	1.0

Table des matières

Avant-propos

Ce document décrit l'infrastructure de l'association **LinuXmaine** et les mécanismes de provisionnement *Infrastructure as Code* (IaC) employés pour la déployer et la maintenir. Le dépôt Git de référence est hébergé sur Framagit :

`https://framagit.org/linuxmaine/infra-linuxmaine-v2.0/-/tree/main/iac`

Le répertoire `iac/` y héberge l'ensemble des fichiers de provisionnement **OpenTofu** (langage HCL2) et **Ansible** (playbooks YAML). Le document est organisé en trois parties :

1. **Introduction** aux briques technologiques (Proxmox, l'API Proxmox et le provider OpenTofu associé, OpenTofu, Ansible, Cloud-init) ;
2. **Provisionnement Ansible seul**, pour une machine déjà créée ;
3. **Provisionnement OpenTofu** couplé à un provisionnement *shell* ou *Ansible*, avec personnalisation Cloud-init.

   Un **lexique** détaillé et une **bibliographie** complètent l'ouvrage.

Architecture générale

L'infrastructure repose sur quatre briques matérielles / logicielles principales placées derrière un pare-feu, et pilotées par une chaîne IaC.

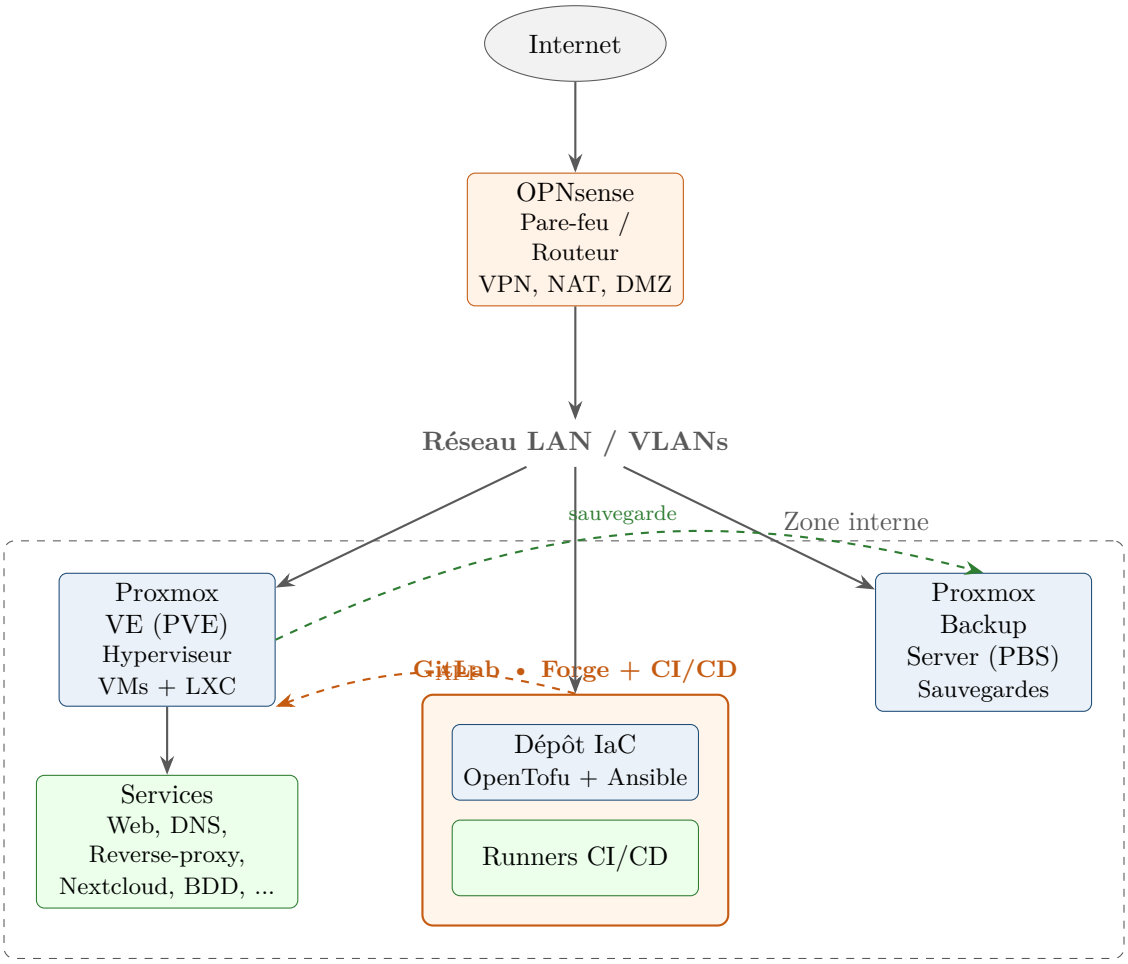


Figure 1: Vue d'ensemble de l'infrastructure LinuXmaine

- **OPNsense** : pare-feu et routeur en frontal d'Internet (filtrage, NAT, VPN, segmentation en VLAN/DMZ).
- **Proxmox VE (PVE)** : hyperviseur exécutant les machines virtuelles (KVM/QEMU) et les conteneurs (LXC) qui hébergent les services de l'association.
- **Proxmox Backup Server (PBS)** : serveur de sauvegarde déduppliquée et chiffrée des VMs et conteneurs de PVE.
- **GitLab** : forge logicielle hébergeant le code et exécutant les pipelines CI/CD qui appliquent OpenTofu et Ansible sur PVE via son API.

Segmentation réseau (vue logique)

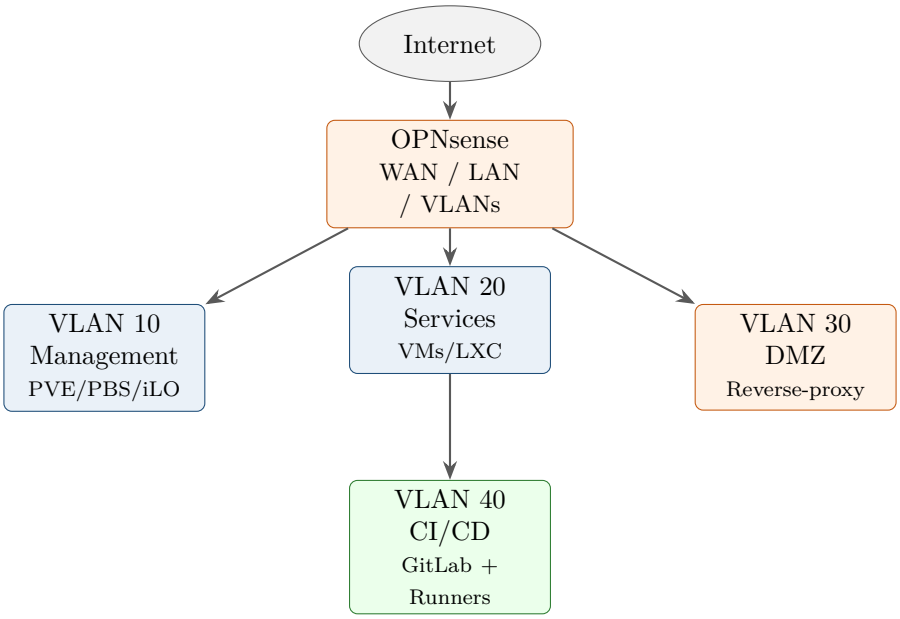


Figure 2: Segmentation logique en VLANs derrière le pare-feu OPNsense

Partie 1

Introduction aux briques technologiques

Cette première partie présente les fondations technologiques de l'infrastructure : l'hyperviseur **Proxmox** et son **API**, le **provider Proxmox d'OpenTofu**, l'outil de provisionnement **OpenTofu**, l'outil de configuration **Ansible**, et le mécanisme de personnalisation au premier démarrage **Cloud-init**.

1.1 Proxmox

1.1.1 Présentation

Proxmox Virtual Environment (Proxmox VE, abrégé **PVE**) est une plateforme de virtualisation libre (licence AGPLv3) basée sur Debian GNU/Linux. Elle combine deux technologies de virtualisation complémentaires :

- **KVM/QEMU** pour les *machines virtuelles* (VM) : chaque VM dispose d'un noyau complet et émule un matériel ; idéal pour exécuter n'importe quel système d'exploitation (Linux, Windows, BSD).
- **LXC** (*Linux Containers*) pour les *conteneurs système* : virtualisation légère partageant le noyau de l'hôte, très économe en ressources, réservée aux distributions Linux.

PVE fournit une interface web, une API REST, une CLI (`pct`, `qm`, `pvesh`) et la mise en cluster de plusieurs nœuds avec migration à chaud.

VM ou LXC ?

On privilégie le **LXC** pour les services Linux légers (faible empreinte, démarrage instantané) et la **VM** lorsqu'on a besoin d'un noyau dédié, d'un OS non Linux, ou d'une isolation matérielle forte.

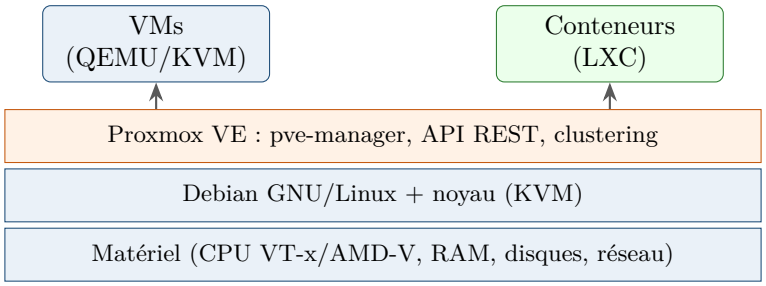


Figure 1.1: Pile logicielle de Proxmox VE

1.1.2 Proxmox Backup Server (PBS)

PBS est un produit distinct, dédié à la sauvegarde des VMs et conteneurs. Il apporte la *déduplication* au niveau bloc, le chiffrement côté client, les sauvegardes incrémentielles et la vérification d'intégrité. PVE s'y connecte comme à un *datastore* de sauvegarde.

1.2 L'API Proxmox

L'ensemble des fonctionnalités de Proxmox est exposé par une **API REST** sur le port `8006` (HTTPS). C'est cette API qu'utilisent l'interface web, la CLI `pvesh`, et surtout le **provider OpenTofu**.

1.2.1 Principes et arborescence

L'API est arborescente, calquée sur la hiérarchie de Proxmox. Quelques points d'entrée typiques :

Chemin	Rôle
`/access/ticket`	Authentification (obtention d'un *ticket*)
`/nodes`	Liste des nœuds du cluster
`/nodes/{node}/qemu`	Machines virtuelles KVM d'un nœud
`/nodes/{node}/lxc`	Conteneurs LXC d'un nœud
`/nodes/{node}/storage`	Stockages disponibles
`/cluster/resources`	Vue agrégée de toutes les ressources
`/access/users`, `/access/roles`	Gestion des droits (RBAC)

1.2.2 Authentification

Deux mécanismes coexistent :

1. **Ticket + CSRF** : on échange identifiant/mot de passe contre un *ticket* (cookie) valable 2 heures et un jeton anti-CSRF. C'est le mode historique, utilisé par l'interface web.
2. **Jeton d'API** (*API Token*, recommandé pour l'automatisation) : un couple `USER@REALM!TOKENID=SECRET` indépendant du mot de passe, révocable, et auquel on attache des permissions précises. C'est le mode à privilégier pour OpenTofu et la CI/CD.

```
# Sur le noeud PVE, créer un utilisateur de service et un rôle dédié
pveum user add tofu@pve
pveum role add TerraformProv -privs "VM.Allocate VM.Clone VM.Config.CDROM \
  VM.Config.Cloudinit VM.Config.CPU VM.Config.Disk VM.Config.HWType \
  VM.Config.Memory VM.Config.Network VM.Config.Options VM.Monitor \
  VM.Audit VM.PowerMgmt Datastore.AllocateSpace Datastore.Audit \
  SDN.Use Sys.Audit Sys.Console Sys.Modify Pool.Allocate"
pveum acl modify / -user tofu@pve -role TerraformProv

# Générer le jeton d'API (noter le secret affiché : non récupérable ensuite)
pveum user token add tofu@pve provisioning --privsep 0
```

Listing 1.1: Création d'un jeton d'API et d'un rôle dédié sur Proxmox

```
# Authentification par mot de passe (récupération du ticket)
curl -sk -d "username=root@pam&password=MONMDP" \
  https://pve.example.org:8006/api2/json/access/ticket

# Lister les VMs d'un noeud avec un jeton d'API
curl -sk -H "Authorization: PVEAPIToken=tofu@pve!provisioning=SECRET" \
  https://pve.example.org:8006/api2/json/nodes/pve01/qemu
```

Listing 1.2: Exemples d'appels directs à l'API REST

Sécurité du jeton

Le secret du jeton d'API n'est affiché qu'**une seule fois**. Il doit être stocké dans un coffre (Ansible Vault, variables CI/CD masquées) et jamais commité en clair. On limite ses droits au strict nécessaire (principe de moindre privilège).

1.2.3 Récupérer pas à pas un jeton d'API pour OpenTofu

Le jeton d'API est l'identifiant que consommera OpenTofu pour piloter Proxmox. Voici la procédure complète, par l'interface web puis par la ligne de commande.

Méthode A — Interface web Proxmox

1. Se connecter à l'interface web `https://pve.example.org:8006`.
2. Aller dans **Datacenter** → **Permissions** → **Users** et créer l'utilisateur de service `tofu@pve`.
3. Dans **Permissions** → **Roles**, créer le rôle `TerraformProv` avec les privilèges listés plus bas.
4. Dans **Permissions**, cliquer **Add** → **User Permission** : chemin `/`, utilisateur `tofu@pve`, rôle `TerraformProv`.
5. Aller dans **Datacenter** → **Permissions** → **API Tokens** → **Add**. Choisir `tofu@pve`, un *Token ID* (ex. `provisioning`), et **décocher** « Privilege Separation » (sinon le jeton n'hérite pas des droits de l'utilisateur).
6. Proxmox affiche le couple **Token ID** et **Secret** : **copier immédiatement le secret** (non récupérable ensuite).

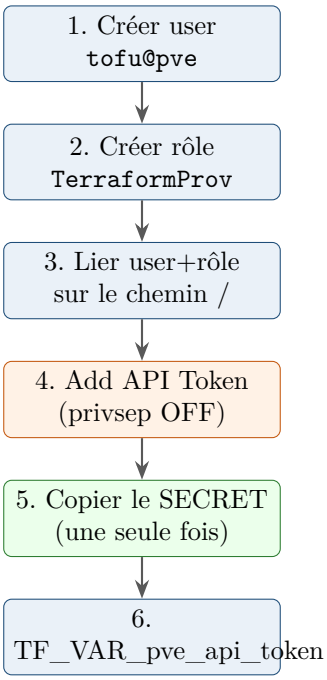


Figure 1.2: Étapes de récupération d'un jeton d'API Proxmox pour OpenTofu

Méthode B — Ligne de commande (`pveum`)

```
pveum user add tofu@pve
pveum role add TerraformProv -privs "VM.Allocate VM.Clone VM.Config.CDROM \
  VM.Config.Cloudinit VM.Config.CPU VM.Config.Disk VM.Config.Memory \
  VM.Config.Network VM.Config.Options VM.Monitor VM.Audit VM.PowerMgmt \
  Datastore.AllocateSpace Datastore.Audit SDN.Use Sys.Audit Sys.Console \
  Sys.Modify Pool.Allocate"
pveum acl modify / -user tofu@pve -role TerraformProv
pveum user token add tofu@pve provisioning --privsep 0
# Sortie :
#  full-tokenid : tofu@pve!provisioning
#  value        : 12345678-9abc-def0-1234-56789abcdef0   <-- LE SECRET
```

Listing 1.3: Récupération du jeton en CLI ; la dernière commande affiche le secret

Assemblage de la valeur attendue par OpenTofu

OpenTofu (provider `bpg/proxmox`) attend le jeton concaténé sous la forme `USER@REALM!TOKENID=SECRET` :

```
export TF_VAR_pve_endpoint="https://pve.example.org:8006/"
export TF_VAR_pve_api_token="tofu@pve!provisioning=12345678-9abc-def0
    -1234-56789abcdef0"
tofu plan
```

Listing 1.4: Injection du jeton dans OpenTofu

Privilèges minimaux du rôle `TerraformProv`

`VM.Allocate`, `VM.Clone`, `VM.Config.*`, `VM.Audit`, `VM.PowerMgmt`, `VM.Monitor`, `Datastore.AllocateSpace`, `Datastore.Audit`, `SDN.Use`, `Sys.Audit`, `Sys.Console`, `Sys.Modify`, `Pool.Allocate`.

1.3 Le provider Proxmox pour OpenTofu

Pour piloter Proxmox depuis OpenTofu, on utilise un **provider** qui traduit les ressources HCL2 en appels à l'API REST. Deux providers font référence :

Provider	Caractéristiques	Source
`bpg/proxmox`	Moderne, très actif, support complet VM/LXC, Cloud-init, fichiers, téléchargement d'images, SDN. *Recommandé.*	`bpg/proxmox`
`Telmate/proxmox`	Historique, largement répandu, parfois en retard sur les nouveautés de PVE.	`Telmate/proxmox`

1.3.1 Déclaration et authentification du provider

```
terraform {
  required_version = ">= 1.6"
  required_providers {
    proxmox = {
      source  = "bpg/proxmox"
      version = "~> 0.66"
    }
  }
}

provider "proxmox" {
  endpoint  = var.pve_endpoint          # https://pve.example.org:8006/
  api_token = var.pve_api_token         # tofu@pve!provisioning=SECRET
  insecure  = false                      # true si certificat auto-signé

  # Le provider exécute aussi certaines actions via SSH (upload d'images,
  # snippets Cloud-init, création de LXC personnalisés).
  ssh {
    agent    = true
    username = "root"
  }
}
```

Listing 1.5: Configuration du provider bpg/proxmox (versions.tf + provider.tf)

1.3.2 Principales ressources

Ressource	Rôle
`proxmox_virtual_environment_vm`	Création / clonage d'une VM KVM
`proxmox_virtual_environment_container`	Création d'un conteneur LXC
`proxmox_virtual_environment_file`	Upload d'images, ISO, snippets Cloud-init
`proxmox_virtual_environment_download_file`	Téléchargement d'une image cloud
`proxmox_virtual_environment_pool`	Pool de ressources
`proxmox_virtual_environment_user` / `..._role`	RBAC (utilisateurs, rôles)

Bonne pratique

Le provider `bpg/proxmox` gère nativement Cloud-init via le bloc `initialization {...}` d'une VM (utilisateur, clés SSH, IP, DNS) et permet de pousser des fichiers *snippets* de configuration avancée. C'est lui qui sert de pivot dans la partie 3.

1.4 OpenTofu

1.4.1 Présentation

OpenTofu est un outil de *provisionnement d'infrastructure* déclaratif, fork libre (sous la *Linux Foundation*) de Terraform, né en 2023 après le changement de licence de ce dernier. Il reste compatible avec le langage **HCL2** et l'écosystème de providers Terraform.

Principe : on **décrit l'état souhaité** de l'infrastructure (VMs, réseaux, volumes) dans des fichiers `.tf`, et OpenTofu calcule puis applique les changements nécessaires pour atteindre cet état.

Référence du langage HCL2

HCL2 (*HashiCorp Configuration Language*, v2) est le langage de configuration déclaratif des fichiers `.tf`. Spécification et syntaxe :

- Dépôt et spécification HCL : https://github.com/hashicorp/hcl
- Syntaxe du langage (OpenTofu) : https://opentofu.org/docs/language/syntax/
- Syntaxe du langage (Terraform) : https://developer.hashicorp.com/terraform/language/syntax/configuration

1.4.2 Concepts clés

- **Provider** : greffon parlant à une API cible (Proxmox, AWS, etc.).
- **Resource** : un objet géré (`proxmox_virtual_environment_vm`).
- **Variable / output** : entrées paramétrables et sorties.
- **State** (`terraform.tfstate`) : le fichier d'état qui mémorise la correspondance entre le code et les objets réels. Il peut être *distant* (backend S3, GitLab, etc.) et doit être protégé.
- **Plan / Apply** : `tofu plan` prévisualise, `tofu apply` applique.

1.4.3 Cycle de vie

```
tofu init          # télécharge les providers, initialise le backend d'état
tofu fmt           # reformate le code HCL
tofu validate      # vérifie la syntaxe et la cohérence
tofu plan          # calcule le différentiel (dry-run)
```

```
tofu apply          # applique les changements
tofu destroy        # détruit l'infrastructure gérée
```

Listing 1.6: Commandes OpenTofu de base

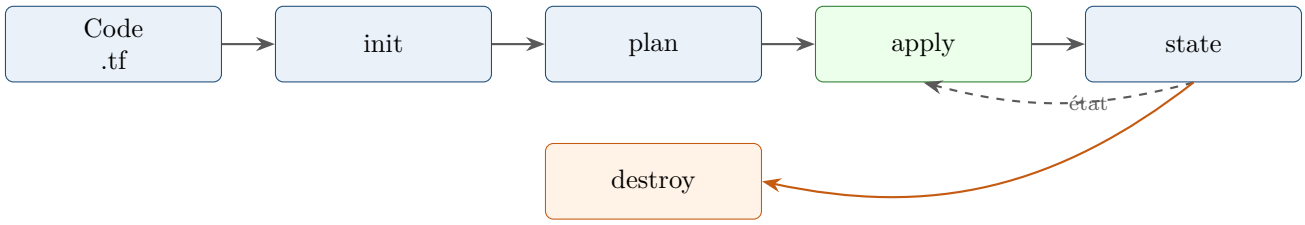


Figure 1.3: Cycle de vie OpenTofu : code → init → plan → apply → state

Détail technique du cycle, pour un utilisateur en ligne de commande :

- **Code .tf** — les fichiers HCL2 du répertoire courant (`*.tf`) décrivent providers, ressources et variables. OpenTofu les fusionne (pas d'ordre imposé entre fichiers).
- `tofu init` — télécharge les *plugins* providers dans `.terraform/`, écrit le verrou de versions `.terraform.lock.hcl`, et initialise le *backend* d'état (local `terraform.tfstate` ou distant).
- `tofu plan` — construit le graphe de dépendances, rafraîchit l'état réel via l'API, puis calcule le différentiel (`+` création, `~` modification, `-` destruction). Aucun changement n'est appliqué.
- `tofu apply` — exécute le plan ; les ressources sont créées dans l'ordre du graphe (parallélisme contrôlé par `-parallelism`).
- **state** — l'état est mis à jour après chaque opération. La flèche en pointillés rappelle que `plan`/`apply` relisent l'état pour détecter la *dérive* (*drift*). `destroy` s'appuie sur l'état pour supprimer ce qui a été créé.

L'état est sensible

Le fichier `terraform.tfstate` contient les identifiants et parfois des secrets en clair. Il doit être stocké dans un *backend* distant verrouillé et à accès restreint (jamais commité en clair dans Git).

1.4.4 Diagramme d'états / transitions d'une ressource OpenTofu

Du point de vue de l'*état* (`tfstate`), chaque ressource gérée par OpenTofu suit une machine à états bien définie :

Lecture détaillée des états et des transitions :

- **Absent** — la ressource n'existe ni dans l'état ni sur l'infrastructure. Transition `plan` vers **Planifié**.
- **Planifié** — le différentiel a été calculé et approuvé (un plan décrit l'action `create`). Transition `apply` vers **Présent**.
- **Présent (appliqué)** — la ressource existe et l'état la reflète. Trois boucles possibles :
  - `apply` de nouveau : **no-op** si rien n'a changé (idempotence) ;
  - **modif code** : une modification du `.tf` renvoie au `plan` (calcul d'un `update` ou `replace`) ;
  - **drift / refresh** : si la ressource a été modifiée hors OpenTofu (à la main sur Proxmox), le rafraîchissement détecte la **Dérive**.
- **Dérive détectée** — l'état réel diverge du code. `apply` *réconcilie* en réimposant l'état déclaré (retour à **Présent**).

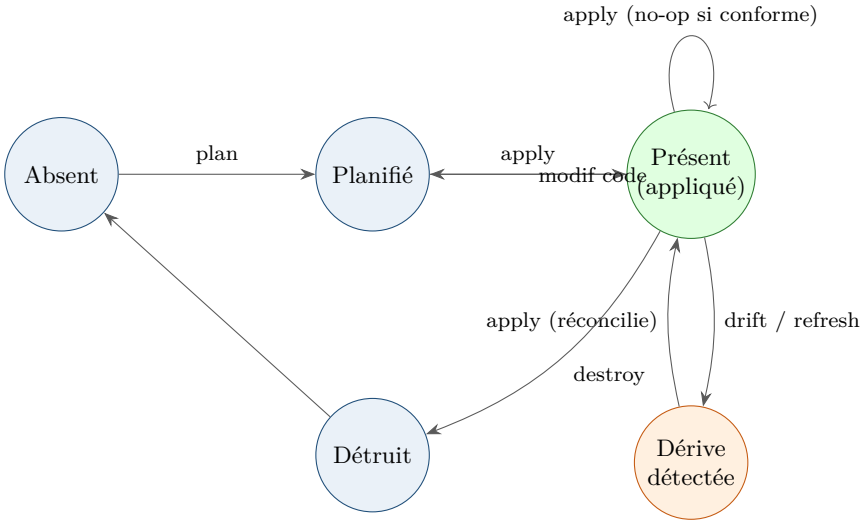


Figure 1.4: États / transitions d'une ressource OpenTofu

- **Détruit** — `destroy` supprime la ressource ; elle redevient **Absent**.

Différence clé avec un script impératif

Contrairement à un script shell qui exécute des commandes, OpenTofu **converge** vers un état cible : quel que soit l'état de départ, il calcule les seules actions nécessaires. C'est le modèle *déclaratif*.

OpenTofu vs Terraform

La syntaxe et les commandes sont identiques (`tofu` remplace `terraform`). OpenTofu est garanti libre (licence MPL 2.0) et gouverné par une communauté ouverte, ce qui motive son adoption ici.

1.5 Ansible

1.5.1 Présentation

Ansible est un outil de *gestion de configuration* et d'*orchestration* **sans agent** (*agentless*) : il se connecte aux machines cibles en **SSH** (ou WinRM) et y exécute des modules idempotents. La configuration est décrite en **YAML** sous forme de *playbooks*.

1.5.2 Concepts clés

- **Inventaire** (*inventory*) : liste des hôtes, regroupés et paramétrés (statique en INI/YAML, ou dynamique).
- **Playbook** : suite de *plays* associant des hôtes à des *tâches*.
- **Tâche** (*task*) : appel à un **module** (`apt`, `copy`, `user`, `template`, ...).
- **Rôle** : unité réutilisable structurée (`tasks/`, `handlers/`, `templates/`, `defaults/`, `vars/`).
- **Idempotence** : ré-exécuter un playbook ne change rien si l'état est déjà conforme.
- **Facts** : informations collectées automatiquement sur les cibles.

1.5.3 Diagramme d'états / transitions d'une tâche Ansible

Lors de l'exécution, chaque tâche Ansible passe par une machine à états dont le résultat conditionne la suite (notification de *handlers*, arrêt sur erreur) :

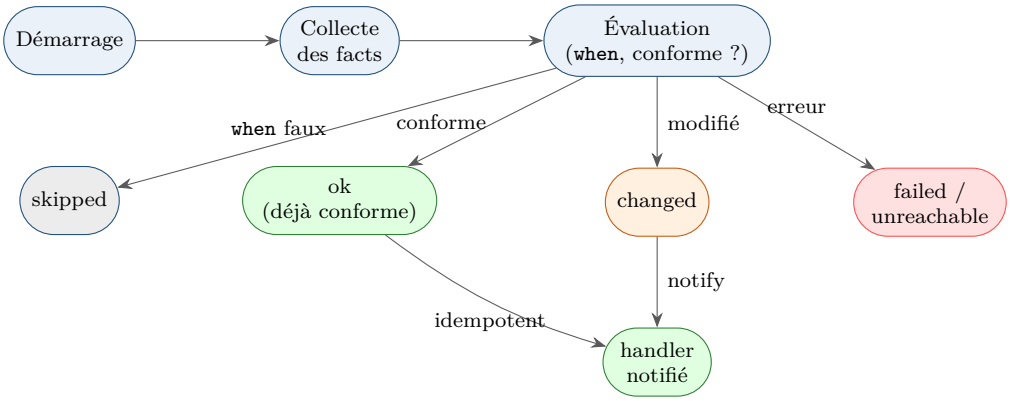


Figure 1.5: États / transitions d'une tâche Ansible (ok / changed / skipped / failed)

Détail technique des états (ce sont exactement les statuts affichés dans la sortie d'`ansible-playbook` et le *recap* final) :

- **Démarrage** — le *play* sélectionne les hôtes et établit la connexion.
- **Collecte des facts** — exécutée si `gather_facts: true` (module `setup`) ; renseigne les variables `ansible_*`. On peut la désactiver pour accélérer.
- **Évaluation** — pour chaque tâche, Ansible teste d'abord la condition `when`, puis interroge l'état courant pour décider de l'action.
- `skipped` (gris) — la condition `when` est fausse : la tâche est ignorée.
- `ok` (vert) — la cible est **déjà conforme** : aucune modification (idempotence). C'est le cas dominant lors des réexécutions.
- `changed` (orange) — un changement a été appliqué ; ce statut peut déclencher un *handler* via `notify` (ex. redémarrer un service).
- `failed` / `unreachable` (rouge) — `failed` : le module a renvoyé une erreur (arrête l'hôte sauf `ignore_errors`) ; `unreachable` : la connexion SSH a échoué (hôte injoignable).
- **handler notifié** — les handlers notifiés s'exécutent *une seule fois*, à la fin du *play* (sauf `meta: flush_handlers`).

Lire le *recap*

À la fin, Ansible affiche par hôte : `ok=N changed=N unreachable=N failed=N skipped=N`. Sur une infrastructure stable, on vise `changed=0` (aucune dérive à corriger).

1.5.4 Différence avec OpenTofu

OpenTofu **crée et détruit** l'infrastructure (le « contenant ») ; Ansible **configure** le système d'exploitation et les applications à l'intérieur (le « contenu »). Les deux sont complémentaires, comme illustré en partie 3.

1.6 Cloud-init

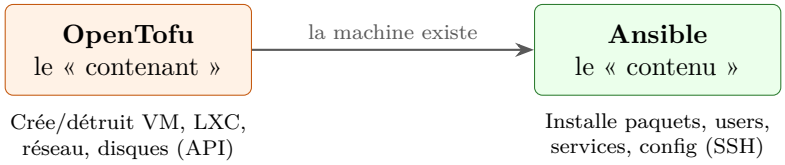


Figure 1.6: Complémentarité : OpenTofu provisionne, Ansible configure

1.6.1 Présentation

Cloud-init est le standard de *personnalisation au premier démarrage* d'une instance Linux. Au tout premier boot, il lit une source de configuration (*datasource*) et applique : nom d'hôte, utilisateurs, clés SSH, configuration réseau, paquets, disposition clavier, scripts.

1.6.2 Comment l'utiliser

1. Partir d'une image « cloud »

On utilise une image officielle *cloud* (par ex. *Debian genericcloud*, *Ubuntu cloud image*) qui contient déjà le paquet `cloud-init`.

2. Fournir la configuration via une datasource

Sur Proxmox, on attache un **lecteur Cloud-init** (`ide2: cloudinit`) à la VM. Proxmox génère alors une datasource de type *NoCloud*. On renseigne :

- **user-data** : utilisateurs, mot de passe, clés SSH, paquets, commandes (`runcmd`), clavier.
- **meta-data** : identité de l'instance (*instance-id*, hostname).
- **network-config** : adressage IP, DNS, passerelle.

```
#cloud-config
hostname: web01
manage_etc_hosts: true
# Disposition clavier française
keyboard:
  layout: fr
  variant: ""
locale: fr_FR.UTF-8
timezone: Europe/Paris
users:
  - name: linuxmaine
    groups: [sudo]
    shell: /bin/bash
    sudo: ALL=(ALL) NOPASSWD:ALL
    lock_passwd: true
    ssh_authorized_keys:
      - ssh-ed25519 AAAAC3NzaC1lZDI1... admin@linuxmaine
package_update: true
packages: [qemu-guest-agent, python3]
runcmd:
  - systemctl enable --now qemu-guest-agent
```

Listing 1.7: Exemple de user-data Cloud-init (clavier FR + clé SSH)

3. Pilotage par Proxmox / OpenTofu

Proxmox expose des champs dédiés (`ciuser`, `cipassword`, `sshkeys`, `ipconfig0`, `nameserver`) que le provider OpenTofu remplit via le bloc `initialization`. Pour la configuration avancée (clavier, `runcmd`), on pousse un *snippet* YAML.

1.6.3 Tout ce qui est personnalisable avec Cloud-init

Cloud-init couvre l'essentiel de la configuration d'un système au premier démarrage. Le tableau ci-dessous recense les domaines personnalisables et le *module* (mot-clé) `cloud-config` correspondant.

Domaine	Mot-clé cloud-config	Description
Nom d'hôte	`hostname`, `fqdn`, `prefer_fqdn_over_hostname`	Nom de la machine et FQDN
Fichier hosts	`manage_etc_hosts`	Gestion de `/etc/hosts`
Utilisateurs	`users`, `groups`	Comptes, groupes, sudo, shell
Mot de passe	`password`, `chpasswd`, `ssh_pwauth`	Mots de passe, expiration, auth. SSH par mot de passe
Clés SSH (autorisées)	`ssh_authorized_keys`, `users[].ssh_authorized_keys`	Clés publiques déposées
Clés d'hôte SSH	`ssh_keys`, `ssh_deletekeys`, `ssh_genkeytypes`	Régénération des clés d'hôte
Clavier	`keyboard.layout`, `.variant`, `.model`	Disposition (ex. `fr`)
Locale / langue	`locale`, `locale_configfile`	Ex. `fr_FR.UTF-8`
Fuseau horaire	`timezone`	Ex. `Europe/Paris`
Réseau	`network` (v1/v2), `ipconfig` (PVE)	IP, passerelle, DNS, VLAN, bonding
Résolution DNS	`resolv_conf`, `manage_resolv_conf`	Serveurs DNS, domaines de recherche
Dépôts de paquets	`apt`, `yum_repos`, `zypper`	Miroirs, sources, clés GPG
Mise à jour	`package_update`, `package_upgrade`	`apt update` / `upgrade`
Installation paquets	`packages`	Liste de paquets à installer
Écriture de fichiers	`write_files`	Création de fichiers (contenu, droits, base64)
Commandes au boot	`runcmd`, `bootcmd`	Scripts exécutés au premier / chaque boot
Disques	`disk_setup`, `fs_setup`, `growpart`, `resizefs`	Partitionnement, formatage, agrandissement
Montages	`mounts`, `swap`	`/etc/fstab`, fichier d'échange
NTP	`ntp`	Serveurs de temps
Chiffrement / CA	`ca_certs`	Certificats de confiance
Journalisation	`output`, `rsyslog`	Redirection des logs cloud-init
Utilisateurs avancés	`snap`, `ansible`, `puppet`, `chef`, `salt_minion`	Bootstrap d'agents de configuration
Phusion / final	`final_message`, `power_state`	Message final, reboot/arrêt programmé

Ce que LinuXmaine personnalise systématiquement

Pour chaque machine : `hostname`, `keyboard: fr`, `locale: fr_FR.UTF-8`, `timezone: Europe/Paris`, un utilisateur `linuxmaine` (sudo), la **clé SSH publique d'administration**, la **configuration réseau** (IP fixe + DNS), l'installation de `qemu-guest-agent`, et un `runcmd` de finalisation.

1.6.4 Distributions Linux supportées

Cloud-init est disponible (image cloud officielle) pour la grande majorité des distributions :

Famille	Distributions avec image cloud / support cloud-init
Debian / dérivées	Debian, Ubuntu, Linux Mint (serveur), Kali
Red Hat / dérivées	RHEL, Rocky Linux, AlmaLinux, CentOS Stream, Fedora
SUSE	openSUSE Leap / Tumbleweed, SLES
Autres	Arch Linux, Gentoo, Alpine (via `cloud-init` paquet), Amazon Linux, Oracle Linux, FreeBSD (support partiel)

Conteneurs LXC et Cloud-init

Les conteneurs **LXC** de Proxmox n'utilisent pas le même mécanisme que les VMs : Proxmox applique directement les options (`hostname`, `ssh-public-keys`, `net0`, `password`) au moment de la création via l'API, sans datasource NoCloud complète. Pour une personnalisation avancée d'un LXC, on enchaîne plutôt avec un provisionnement *shell* ou *Ansible*.

1.7 Les grandes règles d'un bon provisionnement IaC

Un provisionnement *Infrastructure as Code* de qualité ne se limite pas à « créer des machines » : il doit être reproductible, sûr, observable et récupérable. Cette section récapitule les grands principes et, pour chacun, une liste de technologies couramment employées dans l'écosystème libre / GNU-Linux.

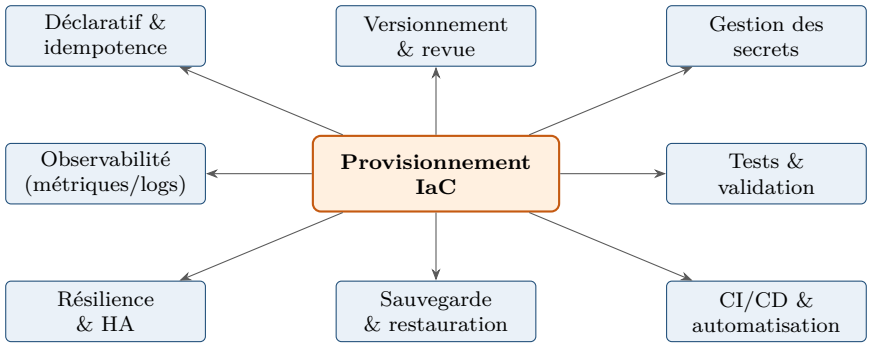


Figure 1.7: Les piliers d'un provisionnement IaC de qualité

1.7.1 Principes et technologies associées

Principe	Objectif	Technologies (exemples)
Déclaratif & idempotence	Décrire l'état cible, convergence sans effet de bord	OpenTofu/Terraform, Ansible, Cloud-init, Puppet, Salt
Versionnement & revue de code	Tout est dans Git, revu avant fusion	Git, GitLab (Merge Requests), pre-commit, conventional commits
Gestion des secrets	Ne jamais exposer mots de passe, clés, jetons	Ansible Vault, SOPS + age, HashiCorp Vault, OpenBao, variables CI/CD masquées, git-crypt
Observabilité — métriques	Mesurer l'état et les performances	Prometheus, Grafana, node_exporter, Telegraf + InfluxDB (serveur de métriques Proxmox), Netdata, Zabbix

Principe	Objectif	Technologies (exemples)
Observabilité — logs	Centraliser et corréler les journaux	Grafana Loki + Promtail, ELK (Elasticsearch/Logstash/Kibana), Graylog, Fluent Bit, rsyslog, journald
Alerting	Être prévenu des incidents	Alertmanager, Grafana Alerting, Karma, Apprise
Tests & validation	Vérifier avant d'appliquer	tofu validate, tflint, tfsec, Checkov, ansible-lint, yamllint, Molecule, Terratest, InSpec, Goss
CI/CD & automatisation	Appliquer de façon répétable et tracée	GitLab CI/CD + runners, Atlantis, pipelines, webhooks
Résilience & haute dispo.	Survivre à la panne d'un composant	Proxmox VE cluster + HA (Corosync), keepalived, HAProxy, réplication ZFS, Ceph
Sauvegarde & restauration	Pouvoir tout reconstruire (règle 3-2-1)	Proxmox Backup Server, restic, BorgBackup, rsync, snapshots ZFS/LVM
Sécurité & durcissement	Réduire la surface d'attaque (DevSecOps)	gitleaks, Trivy, OpenSCAP, CIS Benchmarks, Lynis, fail2ban, Wazuh, CrowdSec
Gestion de l'état (*state*)	État distant, verrouillé, chiffré	Backend GitLab *managed state*, S3/MinIO, verrou (lock), chiffrement
Immutabilité & repro.	Images « golden », versions épinglées	Packer, templates Proxmox, `.terraform.lock.hcl`, conteneurs OCI
Documentation & traçabilité	Comprendre et auditer	terraform-docs, MkDocs Material, README, journaux d'audit GitLab/Proxmox
Réseau & segmentation	Cloisonner (*firewall as code*)	OPNsense, VLANs, WireGuard, nftables, règles versionnées

La règle de sauvegarde 3-2-1

3 copies des données, sur **2** supports différents, dont **1** hors site. Pour LinuXmaine : les VMs/LXC sur PVE (1), sauvegardées sur PBS (2), avec une réplication / un export distant (3).

Observabilité native de Proxmox

Proxmox VE peut pousser ses métriques vers un **serveur externe** (`InfluxDB` ou `Graphite`) depuis *Datacenter* → *Metric Server*, métriques ensuite visualisées dans **Grafana**. Les journaux des hôtes (Debian) sont gérés par `journald`/`rsyslog` et peuvent être centralisés vers Loki ou Graylog.

1.8 Refcards : aide-mémoire OpenTofu et Ansible

Deux aide-mémoire (*refcards*) récapitulent les commandes et options les plus utiles au quotidien.

1.8.1 Refcard OpenTofu

Aide-mémoire OpenTofu (`tofu`)

Cycle de base	`tofu init`	Initialise (providers, backend d'état)
	`tofu init -upgrade`	Met à jour les versions de providers
	`tofu fmt -recursive`	Reformate le code HCL
	`tofu validate`	Vérifie syntaxe et cohérence
	`tofu plan`	Prévisualise les changements (dry-run)
	`tofu plan -out=f.tfplan`	Enregistre le plan dans un fichier
	`tofu apply`	Applique les changements
	`tofu apply f.tfplan`	Applique un plan enregistré
	`tofu apply -auto-approve`	Applique sans confirmation (CI)
	`tofu destroy`	Détruit l'infrastructure gérée
Variables et ciblage	`-var "k=v"`	Définit une variable en ligne
	`-var-file=prod.tfvars`	Charge un fichier de variables
	`TF_VAR_k=v (env)`	Variable via l'environnement
	`-target=res.addr`	Cible une seule ressource
	`-replace=res.addr`	Force le remplacement d'une ressource
État et inspection	`tofu state list`	Liste les ressources de l'état
	`tofu state show addr`	Détaille une ressource
	`tofu state rm addr`	Retire une ressource de l'état
	`tofu state mv a b`	Renomme/déplace dans l'état
	`tofu import addr id`	Importe une ressource existante
	`tofu output`	Affiche les sorties
	`tofu show`	Affiche l'état / le plan courant
	`tofu providers`	Liste les providers requis
	`tofu console`	Console interactive d'expressions
	`tofu workspace list`	Liste les espaces de travail

1.8.2 Refcard Ansible

Aide-mémoire Ansible

Exécution de playbooks

`ansible-playbook -i inv site.yml`	Exécute un playbook
`-limit web01`	Restreint à un hôte/groupe
`-tags base`	N'exécute que les tâches taguées
`-skip-tags slow`	Saute des tâches taguées
`-check`	Mode simulation (dry-run)
`-diff`	Affiche les différences appliquées
`-step`	Confirme chaque tâche
`-b / -become`	Élévation de privilèges (sudo)
`-K`	Demande le mot de passe sudo
`-e "k=v"`	Variable supplémentaire

Commandes ad hoc et inventaire

`ansible all -m ping`	Teste la connectivité
`ansible web -m apt -a "name=vim"`	Module ad hoc
`ansible all -m setup`	Affiche les *facts*
`ansible-inventory -list`	Affiche l'inventaire résolu
`ansible-inventory -graph`	Vue arborescente des groupes

Ansible Vault

`ansible-vault create f.yml`	Crée un fichier chiffré
`ansible-vault edit f.yml`	Édite un fichier chiffré
`ansible-vault encrypt f.yml`	Chiffre un fichier existant
`ansible-vault decrypt f.yml`	Déchiffre un fichier
`ansible-vault rekey f.yml`	Change le mot de passe
`-ask-vault-pass`	Demande le mot de passe
`-vault-password-file f`	Fichier de mot de passe (CI)

Rôles, collections et qualité

`ansible-galaxy install r`	Installe un rôle
`ansible-galaxy collection install c`	Installe une collection
`ansible-galaxy init role`	Squelette de rôle
`ansible-lint`	Analyse statique des playbooks
`ansible-playbook -syntax-check`	Vérifie la syntaxe
`ANSIBLE_VERBOSITY / -vvv`	Mode verbeux (débogage)

Partie 2

Provisionnement Ansible seul (machine déjà créée)

2.1 Introduction

Dans ce scénario, la machine (VM ou serveur physique) **existe déjà** et dispose d'un accès SSH. On ne crée donc *aucune* infrastructure : OpenTofu n'intervient pas. On se contente de **configurer** le système avec Ansible depuis un poste de contrôle (ou un *runner* GitLab).

Objectifs typiques de ce premier provisionnement :

- mise à jour et installation de paquets de base ;
- création d'un utilisateur d'administration ;
- dépôt de la clé SSH publique pour l'accès ultérieur ;
- durcissement minimal (SSH, pare-feu).

Cas d'usage : les machines « hors Proxmox »

Cette partie est tout particulièrement utile pour les équipements et serveurs qui **ne sont pas provisionnés par OpenTofu** parce qu'ils existent déjà ou qu'ils ne sont pas des VMs/LXC pilotables par l'API Proxmox :

- **OPNsense** — le pare-feu (sous FreeBSD). Ansible s'y connecte en SSH ; on cible un interpréteur Python adéquat (`ansible_python_interpreter`) ou on utilise les modules génériques / l'API d'OPNsense.
- **Proxmox VE (PVE)** et **Proxmox Backup Server (PBS)** — les hôtes hyperviseurs eux-mêmes (Debian), que l'on configure et durcit avec Ansible (utilisateurs, SSH, dépôts, sauvegardes).
- **GitLab** — le serveur de forge / CI-CD, configuré et maintenu via Ansible (paquets, runners, sauvegardes, certificats).

Pour toutes ces machines « socle », on applique la démarche décrite ici : inventaire, connexion SSH, dépôt de la clé d'administration, installation de base et durcissement — *sans* OpenTofu.

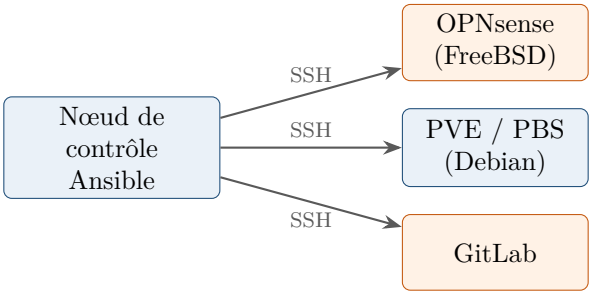


Figure 2.1: Ansible configure les machines « socle » hors périmètre OpenTofu

2.2 Diagrammes

2.2.1 Diagramme de flux

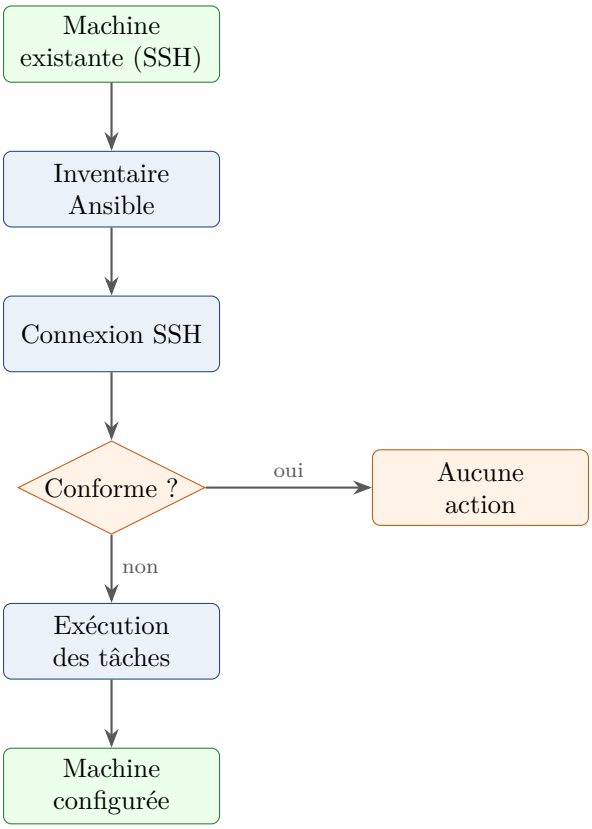


Figure 2.2: Flux du provisionnement Ansible (idempotent)

Ce diagramme décrit, du point de vue d'un administrateur GNU/Linux, le cheminement d'une exécution Ansible sur une machine déjà installée. Détail technique de chaque étape :

1. **Machine existante (SSH)** — prérequis : un service `sshd` actif sur la cible, un compte accessible (par clé publique de préférence), et un interpréteur `python3` présent (`/usr/bin/python3`), qu'Ansible utilise pour exécuter ses modules. Aucun agent permanent n'est requis (*agentless*).
2. **Inventaire Ansible** — Ansible lit l'inventaire (`hosts.yml`/`ini`) pour résoudre l'hôte, ses variables (`ansible_host`, `ansible_user`, `ansible_ssh_private_key_file`) et les groupes auxquels il appartient (`group_vars/`).
3. **Connexion SSH** — ouverture d'une session SSH (transport `ssh` par défaut, multiplexée via `ControlPersist` pour accélérer les tâches successives). Ansible pousse ensuite ses modules dans un répertoire temporaire (`~/.ansible/tmp`) et les exécute à distance.
4. **Décision « Conforme ?»** — pour chaque tâche, le module compare l'état courant de la cible à l'état déclaré. C'est le cœur de l'**idempotence** : si l'état est déjà conforme, la branche `oui` mène à « Aucune action » (statut `ok`) ; sinon la branche `non` déclenche la modification.
5. **Exécution des tâches** — application effective (installation de paquets, écriture de fichiers, création d'utilisateurs...). Chaque changement est rapporté comme `changed` et peut notifier un *handler*.
6. **Machine configurée** — état final conforme. Une réexécution ultérieure ne produira que des `ok` (aucune dérive).

2.2.2 Diagramme de séquence

Le diagramme se lit de gauche à droite, dans l'ordre chronologique. Trois acteurs interviennent :

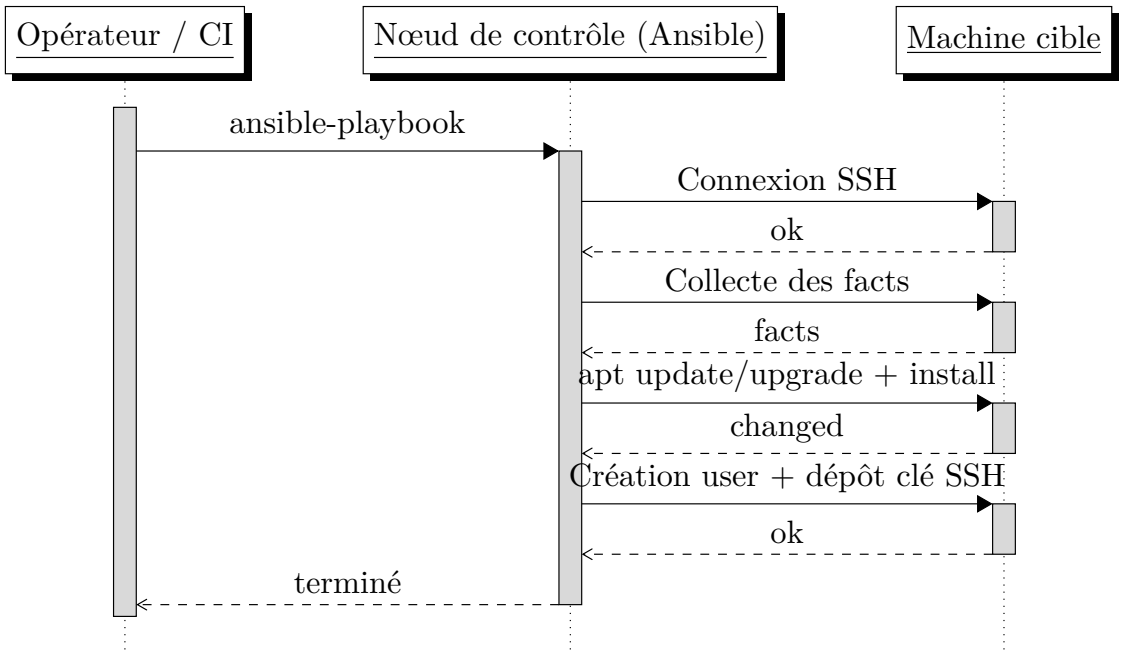


Figure 2.3: Séquence d'un playbook Ansible sur une machine existante

- **Opérateur / CI** — la personne (ou le *runner* GitLab) qui lance la commande `ansible-playbook`.
- **Nœud de contrôle** — la machine qui exécute Ansible (Python + Ansible installés) ; elle ne nécessite rien sur la cible hormis SSH.
- **Machine cible** — le serveur déjà créé que l'on configure.

Déroulé des échanges :

1. **Lancement** — l'opérateur invoque `ansible-playbook` ; le nœud de contrôle lit l'inventaire et le playbook.
2. **Connexion SSH** — le nœud de contrôle ouvre une session SSH vers la cible (authentification par clé) ; c'est le seul prérequis côté machine (*agentless*).
3. **Collecte des *facts*** — Ansible interroge la cible pour récupérer ses caractéristiques (OS, réseau, matériel), qui pourront conditionner les tâches.
4. **Installation de base** — exécution des modules (`apt update/upgrade`, installation des paquets). La cible renvoie l'état : `changed` si une modification a eu lieu, `ok` sinon (idempotence).
5. **Compte et clé SSH** — création de l'utilisateur d'administration et dépôt de la clé publique pour les connexions futures.
6. **Fin** — le playbook se termine et renvoie un récapitulatif (*recap*) : nombre de tâches `ok` / `changed` / `failed` par hôte.

Point clé

À chaque étape, la cible renvoie un statut au nœud de contrôle. Grâce à l'**idempotence**, une seconde exécution ne produit que des `ok` : aucune action superflue n'est rejouée.

2.3 Mise en place du provisionnement Ansible

2.3.1 Inventaire et structure

```yaml
all:
  children:
    webservers:
      hosts:
        web01:
          ansible_host: 192.168.10.21
          ansible_user: linuxmaine
          ansible_ssh_private_key_file: ~/.ssh/id_ed25519
```

Listing 2.1: inventory/hosts.yml

2.3.2 Installation de base : différentes techniques

Plusieurs approches sont possibles pour installer les paquets de base. On les présente de la plus simple à la plus structurée.

Technique 1 — Commande ad hoc

Pour une action ponctuelle, sans playbook :

```
ansible webservers -i inventory/hosts.yml -b -m apt \
  -a "update_cache=yes upgrade=dist"
```

Listing 2.2: Module ad hoc

Technique 2 — Playbook avec le module `apt`

Approche déclarative recommandée, idempotente :

```yaml
---
- name: Installation de base
  hosts: webservers
  become: true
  tasks:
    - name: Mettre à jour le cache et le système
      ansible.builtin.apt:
        update_cache: true
        upgrade: dist
        cache_valid_time: 3600

    - name: Installer les paquets de base
      ansible.builtin.apt:
        name:
          - sudo
          - vim
          - htop
          - curl
          - fail2ban
          - qemu-guest-agent
        state: present
```

Listing 2.3: playbooks/base.yml

Technique 3 — Boucle paramétrée par variable

On externalise la liste des paquets dans une variable (réutilisable, surchargée par hôte/groupe) :

```yaml
# group_vars/webservers.yml
base_packages: [sudo, vim, htop, curl, fail2ban]

# tâche
- name: Installer les paquets de base (boucle)
  ansible.builtin.apt:
    name: "{{ base_packages }}"
    state: present
```

Listing 2.4: Liste de paquets via variable + boucle

Technique 4 — Rôle réutilisable

La forme la plus maintenable : un rôle `common` encapsule l'installation de base. C'est l'approche utilisée dans le dépôt `iac/`.

```
roles/common/
|-- defaults/main.yml       # variables par défaut (base_packages)
|-- tasks/main.yml          # tâches d'installation
|-- handlers/main.yml       # redémarrages de services
`-- templates/              # fichiers de configuration
```

Listing 2.5: Arborescence d'un rôle Ansible

Bonne pratique

Le module `apt` est spécifique à Debian/Ubuntu. Pour un playbook multi-distributions, utiliser le module générique `ansible.builtin.package` ou conditionner les tâches sur `ansible_os_family`.

2.3.3 Upload de la clé SSH : différentes techniques

Déposer une clé publique pour permettre l'accès sans mot de passe est une étape clé. Plusieurs techniques existent.

Technique 1 — Module `ansible.posix.authorized_key`

Méthode recommandée, idempotente et dédiée :

```yaml
- name: Déposer la clé SSH d'administration
  ansible.posix.authorized_key:
    user: linuxmaine
    state: present
    key: "{{ lookup('file', 'files/id_ed25519.pub') }}"
    exclusive: false
```

Listing 2.6: Dépôt de clé via authorized_key

Technique 2 — Création de l'utilisateur avec sa clé

Le module `user` peut créer le compte ; on dépose ensuite la clé :

```yaml
- name: Créer l'utilisateur d'administration
  ansible.builtin.user:
    name: linuxmaine
    groups: sudo
    shell: /bin/bash
    create_home: true
```

```
7
8  - name: Autoriser la clé publique
9    ansible.posix.authorized_key:
10     user: linuxmaine
11     key: "{{ vault_admin_ssh_pubkey }}"
```

Listing 2.7: Création d'utilisateur + clé

Technique 3 — Copie directe du fichier `authorized_keys`

Moins souple (écrase tout), à réserver à un contrôle total :

```
1  - name: Installer authorized_keys
2    ansible.builtin.copy:
3      src: files/authorized_keys
4      dest: /home/linuxmaine/.ssh/authorized_keys
5      owner: linuxmaine
6      group: linuxmaine
7      mode: "0600"
```

Listing 2.8: Copie via le module copy

Technique 4 — Génération de la paire de clés sur la cible

Quand la machine doit elle-même disposer d'une clé (ex. accès à un dépôt Git) :

```
1  - name: Générer une paire de clés ed25519
2    community.crypto.openssh_keypair:
3      path: /home/linuxmaine/.ssh/id_ed25519
4      type: ed25519
5      owner: linuxmaine
```

Listing 2.9: Génération de clé sur l'hôte

2.3.4 Sécurité : Ansible Vault

Les données sensibles (mots de passe, clés privées, secrets API) ne doivent jamais figurer en clair. **Ansible Vault** chiffre ces valeurs (AES-256).

```
1  # Créer / éditer un fichier chiffré
2  ansible-vault create  group_vars/all/vault.yml
3  ansible-vault edit    group_vars/all/vault.yml
4
5  # Chiffrer une seule valeur (inline)
6  ansible-vault encrypt_string 'S3cr3t!' --name 'vault_db_password'
7
8  # Exécuter en fournissant le mot de passe du coffre
9  ansible-playbook -i inventory/hosts.yml playbooks/base.yml --ask-vault-pass
10 # ... ou via un fichier de mot de passe (CI/CD)
11 ansible-playbook ... --vault-password-file .vault_pass
```

Listing 2.10: Commandes Ansible Vault

```
1  # Contenu déchiffré (à l'édition) :
2  vault_admin_ssh_pubkey: "ssh-ed25519 AAAAC3NzaC1lZDI1...
       admin@linuxmaine"
3  vault_become_password: "MotDePasseSudoTresLong"
4
5  # Usage dans un playbook (vars non sensibles pointant vers le coffre)
6  # group_vars/all/main.yml :
```

```
7 admin_ssh_pubkey: "{{ vault_admin_ssh_pubkey }}"
8 ansible_become_password: "{{ vault_become_password }}"
```

Listing 2.11: group_vars/all/vault.yml (chiffré) et son usage

Bonne pratique

Convention courante : une variable « publique » (`admin_ssh_pubkey`) référence une variable « privée » chiffrée (`vault_admin_ssh_pubkey`). Cela permet de voir quelles variables sont sensibles tout en gardant le code lisible.

Partie 3

Provisionnement OpenTofu et variantes

3.1 Introduction

Cette partie couvre le scénario complet : **OpenTofu crée la machine** (LXC ou VM) sur Proxmox, applique une **personnalisation Cloud-init** (clavier français, clé SSH, réseau), puis enchaîne un **provisionnement** soit *shell*, soit *Ansible*. On obtient ainsi un déploiement intégralement reproductible « du néant à la machine prête ».

On distingue quatre combinaisons :

1. LXC + provisionnement shell
2. VM + provisionnement shell
3. LXC + provisionnement Ansible
4. VM + provisionnement Ansible

3.2 Diagrammes

3.2.1 Diagramme de flux

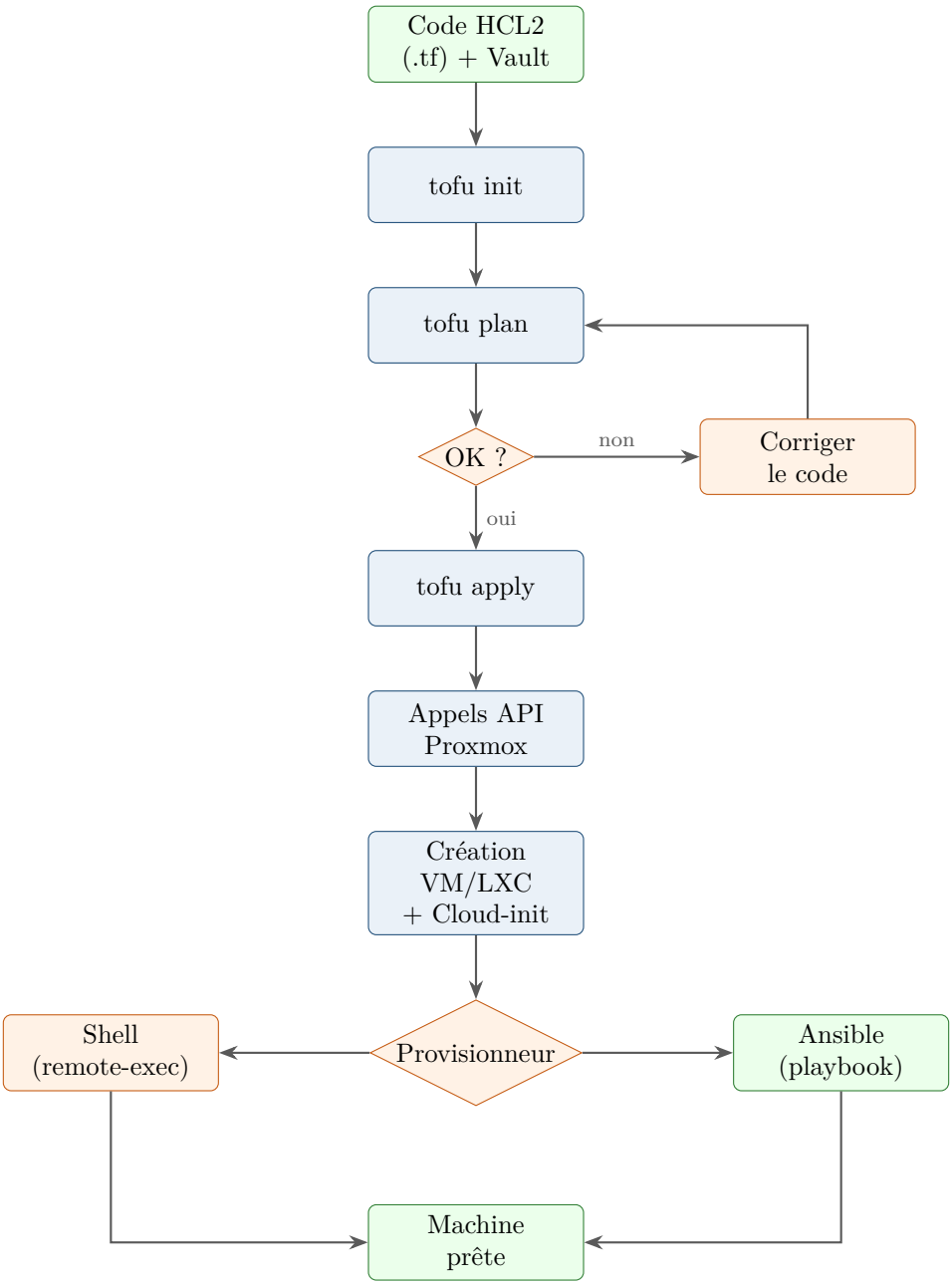


Figure 3.1: Flux OpenTofu : du code à la machine provisionnée

Ce diagramme décrit le cheminement complet, du code jusqu'à la machine prête à l'emploi. Il se lit de haut en bas et comporte deux points de décision (losanges) :

1. **Code HCL2 + Vault** — point de départ : les fichiers `.tf` décrivent les ressources et les secrets sont fournis (TF_VAR / SOPS, cf. 3.3.1).
2. `tofu init` — téléchargement des providers et initialisation du *backend* d'état.
3. `tofu plan` — calcul du différentiel entre l'état souhaité et l'existant.
4. **Décision « OK ? »** — si le plan est incorrect, on **corrige le code** et on reboucle sur `plan` ; sinon on poursuit.
5. `tofu apply` — application du plan : OpenTofu émet les **appels à l'API Proxmox** (authentifiés par le jeton).

6. **Création VM/LXC + Cloud-init** — Proxmox instancie la machine et applique la personnalisation Cloud-init (clavier FR, clé SSH, réseau).
7. **Décision « Provisionneur »** — selon le cas, OpenTofu enchaîne sur un provisionnement **shell** (`remote-exec`, cf. 3.3) ou **Ansible** (`local-exec`, cf. 3.4).
8. **Machine prête** — la cible est créée *et* configurée.

Le rebouclage `plan` → correction → `plan` matérialise la boucle de validation : on n'applique jamais un plan que l'on n'a pas relu.

3.2.2 Diagramme de séquence

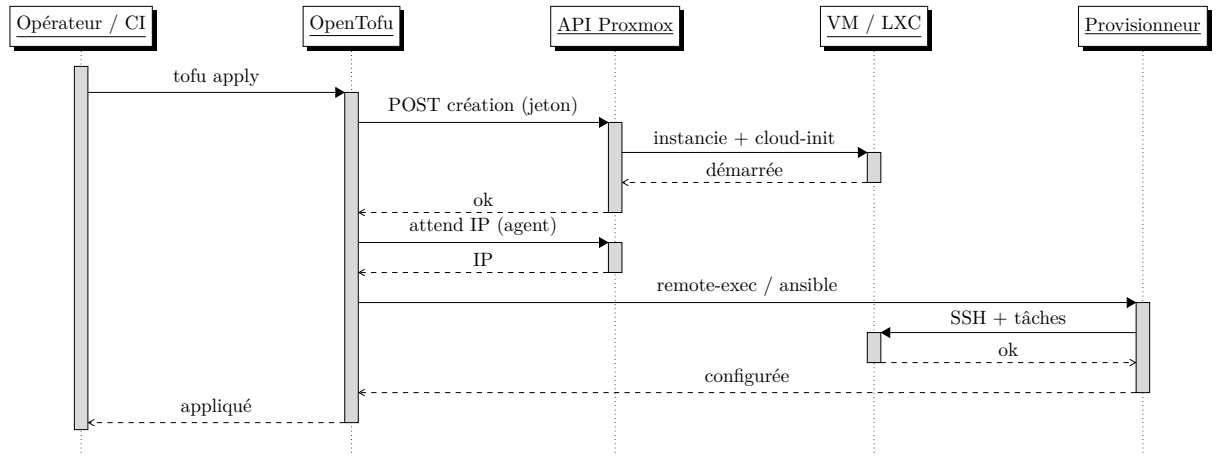


Figure 3.2: Séquence : OpenTofu, API Proxmox, Cloud-init puis provisionneur

Le diagramme de séquence détaille les échanges entre les cinq acteurs, dans l'ordre temporel :

- **Opérateur / CI** — déclenche `tofu apply`.
- **OpenTofu** — orchestre l'ensemble.
- **API Proxmox** — reçoit les ordres de création (port 8006, jeton).
- **VM / LXC** — la machine instanciée.
- **Provisionneur** — `remote-exec` (shell) ou `local-exec` (Ansible).

Déroulé :

1. **`tofu apply`** — l'opérateur lance l'application ; OpenTofu prend la main.
2. **Création de la ressource** — OpenTofu appelle l'API Proxmox en s'authentifiant avec le **jeton** (cf. partie 1) ; l'API accuse réception.
3. **Instanciation + Cloud-init** — Proxmox crée la VM/LXC et applique la personnalisation Cloud-init au premier démarrage ; la machine démarre.
4. **Attente de l'IP** — OpenTofu attend que l'*agent QEMU* remonte l'adresse IP (étape indispensable avant de pouvoir s'y connecter).
5. **Provisionnement** — OpenTofu déclenche le provisionneur : `remote-exec` exécute un script via SSH, ou `local-exec` lance `ansible-playbook`. Le provisionneur se connecte en SSH et applique les tâches.
6. **Retour** — une fois la cible configurée, le contrôle revient à OpenTofu, qui clôt l'`apply` et met à jour son état (`tfstate`).

Ordre critique

L'étape d'**attente de l'IP** (agent QEMU) est essentielle : sans elle, le provisionneur tenterait de se connecter à une machine pas encore jointe sur le réseau, et échouerait. C'est pourquoi l'agent (`agent { enabled = true }`) est activé sur les VMs.

3.3 OpenTofu + provisionnement shell

3.3.1 Variables et secrets (Vault)

OpenTofu n'a pas de coffre intégré comme Ansible Vault. Trois approches :

- variables d'environnement `TF_VAR_*` (alimentées par la CI/CD) ;
- fichier `*.tfvars` chiffré (SOPS) ou variables CI/CD masquées ;
- lecture d'un secret depuis un gestionnaire externe (Vault HashiCorp).

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

variable "pve_api_token" {
  type      = string
  sensitive = true        # masque la valeur dans les logs/plan
}

variable "ssh_public_key" { type = string }
variable "ci_password" {
  type      = string
  sensitive = true
}
```

Listing 3.1: variables.tf — déclaration des entrées sensibles

```
# Variables CI/CD GitLab "masquées" -> exportées en TF_VAR_*
export TF_VAR_pve_api_token="tofu@pve!provisioning=SECRET"
export TF_VAR_ci_password="$(sops -d secrets/ci.enc.txt)"
tofu apply -auto-approve
```

Listing 3.2: Alimentation des secrets via l'environnement (CI/CD GitLab)

3.3.2 LXC + provisionnement shell

Création d'un conteneur LXC, puis configuration par script shell distant. Comme indiqué en partie 1, le LXC reçoit ses options de personnalisation (hostname, clé SSH, réseau, clavier via `runcmd`) directement à la création.

```
resource "proxmox_virtual_environment_container" "web" {
  node_name = "pve01"
  vm_id     = 201

  initialization {
    hostname = "web-lxc01"

    # Clavier/locale ne sont pas natifs au LXC : appliqués via le script
    ip_config {
      ipv4 {
        address = "192.168.10.31/24"
        gateway = "192.168.10.1"
      }
    }
    dns { servers = ["192.168.10.1"] }

```

```
    user_account {
      keys     = [var.ssh_public_key]   # clé SSH injectée
      password = var.ci_password
    }
  }

  operating_system {
    template_file_id = "local:vztmpl/debian-12-standard_12.7-1_amd64.tar.zst"
    type             = "debian"
  }

  cpu    { cores = 2 }
  memory { dedicated = 1024 }

  disk {
    datastore_id = "local-lvm"
    size         = 8
  }

  network_interface {
    name   = "eth0"
    bridge = "vmbr0"
  }

  # ----- Provisionnement shell (clavier FR + paquets) -----
  connection {
    type        = "ssh"
    host        = "192.168.10.31"
    user        = "root"
    private_key = file("~/.ssh/id_ed25519")
  }

  provisioner "remote-exec" {
    inline = [
      "localectl set-keymap fr || true",
      "echo 'KEYMAP=fr' > /etc/vconsole.conf",
      "apt-get update",
      "apt-get install -y sudo vim curl qemu-guest-agent",
    ]
  }
}
```

Listing 3.3: lxc-shell.tf — LXC personnalisé + provisionnement shell

3.3.3 VM + provisionnement shell

Création d'une VM KVM clonée depuis un *template* d'image cloud, avec Cloud-init complet (clavier FR via snippet) puis script shell.

```
# 1) Snippet Cloud-init avancé (clavier FR, locale) poussé sur Proxmox
resource "proxmox_virtual_environment_file" "ci_user" {
  content_type = "snippets"
  datastore_id = "local"
  node_name    = "pve01"
  source_raw {
    file_name = "web01-user.yaml"
    data      = <<-EOF
      #cloud-config
      keyboard:
        layout: fr
      locale: fr_FR.UTF-8
      timezone: Europe/Paris
      packages: [qemu-guest-agent, sudo, vim, curl]
      runcmd:
```

```
        - systemctl enable --now qemu-guest-agent
    EOF
  }
}

# 2) VM clonée + Cloud-init
resource "proxmox_virtual_environment_vm" "web" {
  name      = "web-vm01"
  node_name = "pve01"
  vm_id     = 301

  clone {
    vm_id = 9000           # template Debian 12 cloud
  }

  agent { enabled = true }
  cpu    { cores = 2 }
  memory { dedicated = 2048 }

  initialization {
    user_data_file_id = proxmox_virtual_environment_file.ci_user.id
    ip_config {
      ipv4 {
        address = "192.168.10.41/24"
        gateway = "192.168.10.1"
      }
    }
    dns { servers = ["192.168.10.1"] }
    user_account {
      username = "linuxmaine"
      keys     = [var.ssh_public_key]   # clé SSH injectée
      password = var.ci_password
    }
  }

  # ----- Provisionnement shell -----
  connection {
    type        = "ssh"
    host        = "192.168.10.41"
    user        = "linuxmaine"
    private_key = file("~/.ssh/id_ed25519")
  }
  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y htop fail2ban",
    ]
  }
}
```

Listing 3.4: vm-shell.tf — VM Cloud-init + provisionnement shell

Customisation Cloud-init des deux variantes

Dans les deux cas ci-dessus, la **clé SSH est injectée** (`keys`) et le **clavier français** est configuré (champ `keyboard` pour la VM, commande `localectl` / `runcmd` pour le LXC). On ajoute aussi la locale `fr_FR.UTF-8` et le fuseau `Europe/Paris`.

3.4 OpenTofu + provisionnement Ansible

Ici, OpenTofu crée la machine puis **déclenche un playbook Ansible** (provisionneur `local-exec` appelant `ansible-playbook`). Les playbooks proviennent du dépôt `iac/` de LinuXmaine :

https://framagit.org/linuxmaine/infra-linuxmaine-v2.0/-/tree/main/iac

C'est le motif le plus puissant : OpenTofu pour le « contenant », Ansible pour le « contenu », chacun avec son coffre (Vault).

3.4.1 LXC + provisionnement Ansible

```
resource "proxmox_virtual_environment_container" "app" {
  node_name = "pve01"
  vm_id     = 202

  initialization {
    hostname = "app-lxc01"
    ip_config {
      ipv4 {
        address = "192.168.10.32/24"
        gateway = "192.168.10.1"
      }
    }
    user_account {
      keys     = [var.ssh_public_key]
      password = var.ci_password
    }
  }
  operating_system {
    template_file_id = "local:vztmpl/debian-12-standard_12.7-1_amd64.tar.zst"
    type             = "debian"
  }
  cpu    { cores = 2 }
  memory { dedicated = 1024 }
  disk   { datastore_id = "local-lvm"; size = 8 }
  network_interface { name = "eth0"; bridge = "vmbr0" }

  # ----- Bascule vers Ansible -----
  provisioner "local-exec" {
    command = <<-EOT
      ansible-playbook \
        -i '192.168.10.32,' \
        --private-key ~/.ssh/id_ed25519 \
        -u root \
        --vault-password-file .vault_pass \
        ../ansible/playbooks/app.yml
    EOT
  }
}
```

Listing 3.5: lxc-ansible.tf — LXC + bascule vers Ansible

3.4.2 VM + provisionnement Ansible

```
resource "proxmox_virtual_environment_vm" "app" {
  name      = "app-vm01"
  node_name = "pve01"
  vm_id     = 302

  clone { vm_id = 9000 }
  agent { enabled = true }
  cpu    { cores = 2 }
  memory { dedicated = 2048 }

  initialization {
```

```
    user_data_file_id = proxmox_virtual_environment_file.ci_user.id  # clavier FR
    ip_config {
      ipv4 {
        address = "192.168.10.42/24"
        gateway = "192.168.10.1"
      }
    }
    user_account {
      username = "linuxmaine"
      keys     = [var.ssh_public_key]
      password = var.ci_password
    }
  }

  # Attendre que l'agent QEMU remonte l'IP avant Ansible
  provisioner "local-exec" {
    command = <<-EOT
      ansible-playbook \
        -i '192.168.10.42,' \
        --private-key ~/.ssh/id_ed25519 \
        -u linuxmaine --become \
        --vault-password-file .vault_pass \
        ../ansible/playbooks/app.yml
    EOT
  }
}
```

Listing 3.6: vm-ansible.tf — VM Cloud-init + bascule vers Ansible

3.4.3 Playbook Ansible appelé (avec Vault)

```
---
- name: Configuration applicative (appelée par OpenTofu)
  hosts: all
  become: true
  vars_files:
    - ../group_vars/all/vault.yml      # secrets chiffrés (Vault)
  roles:
    - common                            # paquets de base + clé SSH
    - app
  tasks:
    - name: Déposer la clé SSH d'admin (depuis le Vault)
      ansible.posix.authorized_key:
        user: linuxmaine
        key: "{{ vault_admin_ssh_pubkey }}"
```

Listing 3.7: ansible/playbooks/app.yml

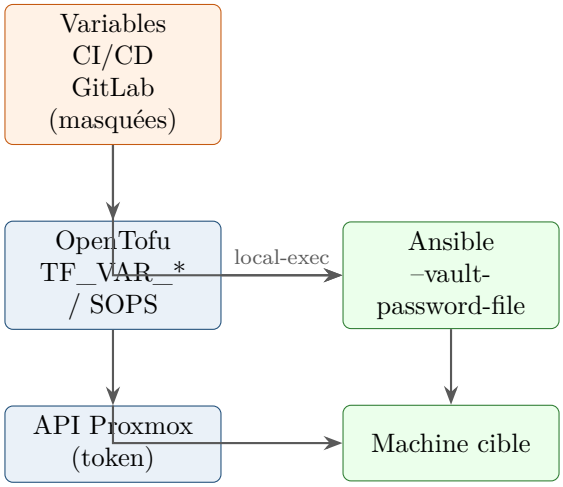


Figure 3.3: Deux coffres complémentaires : secrets OpenTofu et Ansible Vault

3.4.4 Articulation des deux coffres

Synthèse des secrets

- **OpenTofu** : token API Proxmox et mot de passe Cloud-init via `TF_VAR_*` (variables CI/CD masquées) ou `SOPS`, marqués `sensitive`.
- **Ansible** : clés SSH, mots de passe `become`, secrets applicatifs dans `group_vars/all/vault.yml` chiffré (AES-256).
- Le mot de passe du Vault est fourni en CI/CD via un fichier protégé (`.vault_pass`) issu d'une variable masquée.

Partie 4

Chaîne CI/CD DevSecOps avec GitLab

4.1 Introduction

Le dépôt `iac/` est validé en continu par un pipeline GitLab CI/CD qui applique la démarche **DevSecOps** : à chaque *push*, le code est **vérifié** (lint), **validé** (syntaxe/cohérence), **audité** (sécurité, secrets), puis un `tofu plan` est calculé. Le **déploiement** (`tofu apply`) n'a lieu que lorsqu'un **tag** de version est posé, garantissant que seules les versions explicitement publiées modifient l'infrastructure.

★ Documentation GitLab à mettre en évidence

Écriture des fichiers `.gitlab-ci.yml` :

- Référence complète du format : https://docs.gitlab.com/ee/ci/yaml/
- Prise en main de la CI/CD : https://docs.gitlab.com/ee/ci/

API GitLab officielle :

- Documentation REST : https://docs.gitlab.com/ee/api/
- API des pipelines : https://docs.gitlab.com/ee/api/pipelines.html

Module / client Python pour l'API GitLab (`python-gitlab`) :

- Documentation : https://python-gitlab.readthedocs.io/
- Dépôt : https://github.com/python-gitlab/python-gitlab

Les contrôles couvrent tous les langages du dépôt :

Langage / cible	Outil	Rôle
YAML	`yamllint`	Style et validité des fichiers YAML
Ansible	`ansible-lint`	Bonnes pratiques et erreurs de playbooks
HCL2 / OpenTofu	`tofu fmt -check`, `tofu validate`	Formatage et cohérence
HCL2	`tflint`	Règles et erreurs spécifiques Terraform/OpenTofu
Bash	`shellcheck`	Erreurs et pièges des scripts shell
Dockerfile	`hadolint`	Bonnes pratiques d'images
Sécurité IaC	`tfsec` / `checkov`	Vulnérabilités de configuration
Secrets	`gitleaks`	Détection de secrets commités

4.2 Diagramme du pipeline

Figure 4.1: Étapes du pipeline DevSecOps : lint → validate → security → plan → deploy (sur tag)

Détail technique des cinq *stages* (étapes), exécutés en séquence par les *runners* GitLab ; un *stage* ne démarre que si le précédent réussit :

1. **lint** — analyse statique du style et de la forme : `yamllint` (YAML), `ansible-lint` (playbooks), `shellcheck` (scripts `.sh`). Échec = arrêt immédiat.
2. **validate** — `tofu fmt -check` (formatage), `tofu validate` (cohérence HCL) et `tflint` (règles spécifiques). Valide que le code *compile* sans rien déployer.
3. **security** — analyse de sécurité *shift-left* : `tfsec`/`checkov` (mauvaises configurations IaC) et `gitleaks` (détection de secrets commités).
4. **plan** — `tofu plan -out=plan.tfplan` ; le plan est conservé comme *artefact* pour être relu et réutilisé tel quel par le déploiement.
5. **deploy** — `tofu apply` du plan, **uniquement** si un **tag** de version sémantique est présent (`rules: if $CI_COMMIT_TAG`) et après validation manuelle (`when: manual`).

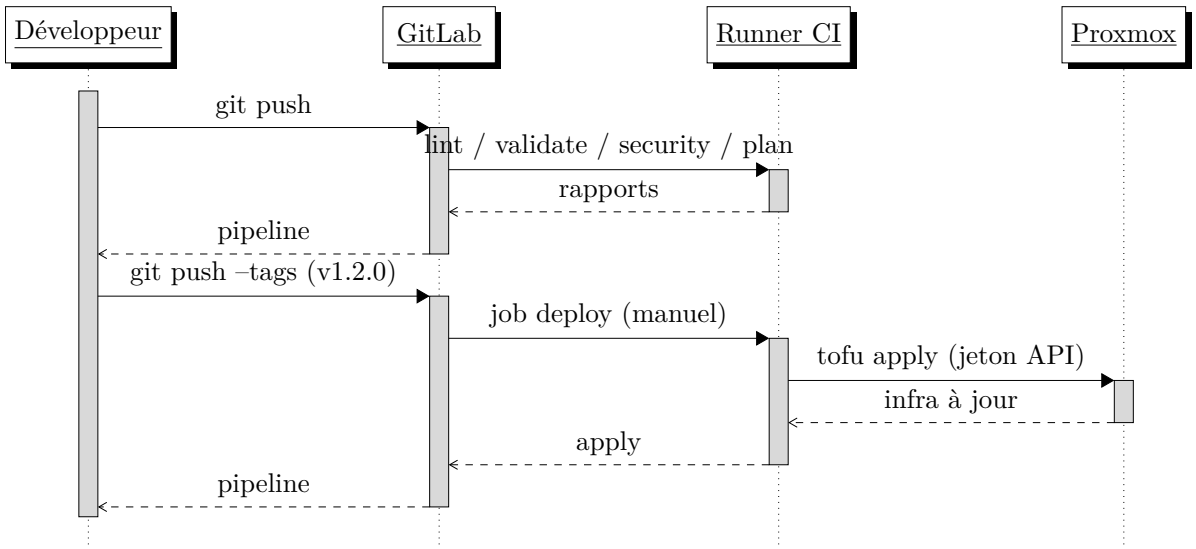


Figure 4.2: Séquence : vérifications sur push, déploiement sur tag

Lecture détaillée de la séquence, du poste du développeur jusqu'à Proxmox :

1. **git push** (sur une branche) — déclenche un pipeline. GitLab confie aux *runners* les stages `lint`, `validate`, `security` et `plan`, qui renvoient leurs *rapports*. **Aucune** modification d'infrastructure à ce stade.
2. **git push --tags** (`v1.2.0`) — la pose d'un tag de version déclenche un pipeline incluant le stage `deploy`, soumis à validation manuelle.
3. **job deploy** — le runner exécute `tofu apply` en s'authentifiant auprès de l'**API Proxmox** avec le jeton (variable CI/CD masquée, cf. 4.4).
4. **infra à jour** — Proxmox applique les changements ; le résultat remonte au runner puis à GitLab (statut du job).

4.3 Fichier `.gitlab-ci.yml` commenté

```
---
stages: [lint, validate, security, plan, deploy]

variables:
  TF_ROOT: "${CI_PROJECT_DIR}/iac/opentofu"
```

```
  ANSIBLE_DIR: "${CI_PROJECT_DIR}/iac/ansible"

# ----------------------- STAGE 1 : LINT -----------------------
yamllint:
  stage: lint
  image: cytopia/yamllint:latest
  script:
    - yamllint -s iac/

ansible-lint:
  stage: lint
  image: registry.gitlab.com/pipeline-components/ansible-lint:latest
  script:
    - ansible-lint ${ANSIBLE_DIR}/

shellcheck:
  stage: lint
  image: koalaman/shellcheck-alpine:latest
  script:
    - find iac/ -name "*.sh" -print0 | xargs -0 -r shellcheck

# --------------------- STAGE 2 : VALIDATE ---------------------
tofu-validate:
  stage: validate
  image: ghcr.io/opentofu/opentofu:latest
  script:
    - cd ${TF_ROOT}
    - tofu fmt -check -recursive       # échoue si non formaté
    - tofu init -backend=false
    - tofu validate

tflint:
  stage: validate
  image: ghcr.io/terraform-linters/tflint:latest
  script:
    - cd ${TF_ROOT}
    - tflint --init
    - tflint --recursive

# ---------------------- STAGE 3 : SECURITY ----------------------
tfsec:
  stage: security
  image: aquasec/tfsec:latest
  script:
    - tfsec ${TF_ROOT} --soft-fail   # rapport sans bloquer (ajustable)

checkov:
  stage: security
  image: bridgecrew/checkov:latest
  script:
    - checkov -d ${TF_ROOT} --quiet

gitleaks:
  stage: security
  image: zricethezav/gitleaks:latest
  script:
    - gitleaks detect --source . --no-banner --redact

# ----------------------- STAGE 4 : PLAN -----------------------
```

```yaml
tofu-plan:
  stage: plan
  image: ghcr.io/opentofu/opentofu:latest
  script:
    - cd ${TF_ROOT}
    - tofu init
    - tofu plan -out=plan.tfplan
  artifacts:
    paths: ["${TF_ROOT}/plan.tfplan"]
    expire_in: 1 week
  # Variables CI/CD masquées injectées par GitLab :
  #   TF_VAR_pve_api_token, TF_VAR_ci_password, ANSIBLE_VAULT_PASSWORD

# ---------------------- STAGE 5 : DEPLOY ----------------------
tofu-apply:
  stage: deploy
  image: ghcr.io/opentofu/opentofu:latest
  script:
    - cd ${TF_ROOT}
    - tofu init
    # Mot de passe du Vault Ansible -> fichier protégé pour local-exec
    - echo "$ANSIBLE_VAULT_PASSWORD" > .vault_pass
    - tofu apply -auto-approve plan.tfplan
  rules:
    # Ne s'exécute QUE si un tag de version est posé (ex. v1.2.0)
    - if: '$CI_COMMIT_TAG =~ /^v[0-9]+\.[0-9]+\.[0-9]+$/'
      when: manual           # validation humaine avant apply
  environment:
    name: production
  dependencies: [tofu-plan]
```

Listing 4.1: .gitlab-ci.yml — vérifications DevSecOps + déploiement sur tag

Gestion des secrets en CI/CD

Les secrets (`TF_VAR_pve_api_token`, `TF_VAR_ci_password`, `ANSIBLE_VAULT_PASSWORD`) sont définis comme variables CI/CD « **Masked** » et « **Protected** » dans GitLab. Ils ne s'affichent jamais dans les logs et ne sont disponibles que sur les branches/tags protégés. Le job `deploy` reconstruit `.vault_pass` à la volée pour les appels `local-exec` vers Ansible (cf. partie 3).

Déclenchement par tag

La clause `rules: if $CI_COMMIT_TAG` cantonne le déploiement aux tags de version sémantique. Combinée à `when: manual`, elle exige une validation humaine : aucune modification d'infrastructure n'est appliquée par un simple *push* sur une branche.

4.4 Gestion des secrets dans les variables de projet GitLab

Le **token de l'API Proxmox**, le mot de passe Cloud-init et le mot de passe du Vault Ansible ne doivent **jamais** apparaître dans le dépôt ni dans les logs. On les déclare comme **variables CI/CD** au niveau du projet (ou du groupe) : **Settings** → **CI/CD** → **Variables**.

4.4.1 Configuration recommandée des variables

Variable	Masked	Protected	Contenu
`TF_VAR_pve_api_token`	oui	oui	Jeton API Proxmox `user!id=secret`
`TF_VAR_ci_password`	oui	oui	Mot de passe Cloud-init
`ANSIBLE_VAULT_PASSWORD`	oui	oui	Mot de passe du Vault Ansible
`TF_VAR_ssh_public_key`	non	oui	Clé SSH publique d'admin (non secrète)

- **Masked** (masquée) : la valeur est remplacée par `[MASKED]` dans tous les logs de jobs. Contrainte : valeur d'au moins 8 caractères, base64-compatible.
- **Protected** (protégée) : la variable n'est exposée qu'aux jobs s'exécutant sur des **branches/-tags protégés** (typiquement `main` et les tags `v*`). Un *push* sur une branche de fonctionnalité n'y a donc pas accès.
- **Expanded** : désactiver l'expansion des variables (`$`) dans les secrets pour éviter les fuites par substitution.
- **Environnement** : restreindre la variable à l'environnement `production` pour qu'elle ne soit lisible que par le job de déploiement.

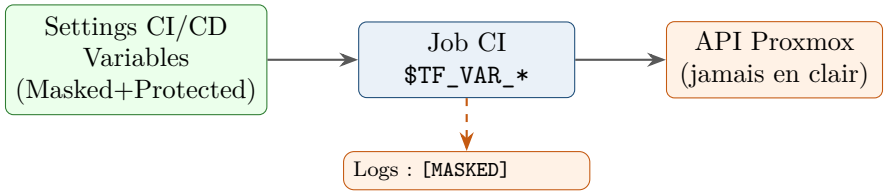


Figure 4.3: Le token de l'API Proxmox transite masqué, jamais affiché dans les logs

Ne jamais dévoiler le token

- Ne pas `echo` une variable secrète dans un script de job.
- Ne pas la passer en argument de ligne de commande (visible dans `ps`/les logs) ; privilégier l'environnement.
- Activer **Masked** *et* **Protected** ; restreindre par environnement.
- Préférer, si disponible, l'intégration **OIDC / HashiCorp Vault** (secrets éphémères) plutôt que des secrets statiques.
- **Faire tourner** (*rotation*) régulièrement le token et le révoquer immédiatement en cas de doute.

4.4.2 Autres recommandations DevSecOps importantes

1. **Moindre privilège** : le token Proxmox porte un rôle dédié et minimal (cf. partie 1) ; un token par usage (CI vs poste local).
2. **Branches protégées + revue de code** : exiger une *merge request* approuvée avant fusion sur `main` ; interdire le *push* direct.
3. **Déploiement gated** : `apply` uniquement sur tag, `when: manual`, avec environnement `production` et éventuellement une fenêtre d'approbation.
4. **Scan de secrets en pré-commit** : `gitleaks` en *pre-commit hook et* en CI, pour bloquer en amont.
5. **Verrouillage des versions** : épingler les versions des providers (`.terraform.lock.hcl` commité) et des images de runners.

6. **SAST / analyse de dépendances** : activer les templates GitLab `SAST`, `Dependency Scanning`, `Container Scanning`.
7. **État OpenTofu distant et chiffré** : backend distant (GitLab managed state) avec verrou, accès restreint ; l'état peut contenir des secrets.
8. **Runners dédiés et isolés** : runners taggés, éphémères, sans accès réseau superflu vers d'autres environnements.
9. **Traçabilité** : journaux d'audit GitLab *et* Proxmox ; chaque `apply` rattaché à un tag et un auteur.
10. **Tests avant apply** : `tofu plan` systématique en artefact, relu avant l'`apply` manuel.
11. **Politique de secrets** : rotation planifiée, expiration des tokens, interdiction des secrets en clair dans les `tfvars` commités (utiliser `SOPS`/Vault).
12. **Conformité « shift-left »** : `tfsec`/`checkov` dès le stage *security*, idéalement bloquant sur les sévérités hautes.

Partie 5

Exemples d'utilisation de l'API Proxmox

5.1 Introduction

Cette partie illustre l'appel de l'**API REST Proxmox** dans les principaux langages modernes. Tous les exemples suivent le même scénario : authentification par **jeton d'API** (en-tête `Authorization: PVEAPIToken=...`) et **liste des VMs** d'un nœud (`GET /api2/json/nodes/{node}/qemu`).

Certificat auto-signé

Proxmox présente souvent un certificat auto-signé. Les exemples désactivent la vérification TLS pour la lisibilité (`verify=False`, `rejectUnauthorized: false`...). **En production**, installez un certificat valide (Let's Encrypt) et **activez** la vérification.

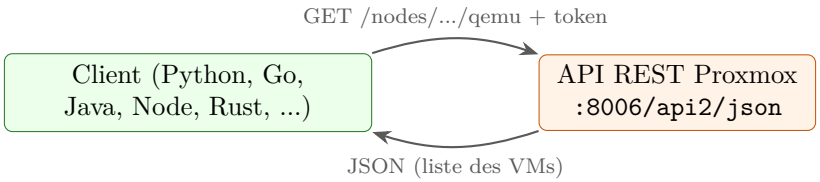


Figure 5.1: Schéma commun à tous les exemples : un client interroge l'API REST

5.2 Shell (curl)

```
TOKEN="tofu@pve!provisioning=12345678-9abc-def0-1234-56789abcdef0"
curl -sk -H "Authorization: PVEAPIToken=${TOKEN}" \
  https://pve.example.org:8006/api2/json/nodes/pve01/qemu | jq .
```

5.3 Python 3

Deux variantes : `requests` (bas niveau) et la bibliothèque dédiée `proxmoxer` (haut niveau).

```
# pip install requests
import requests
TOKEN = "tofu@pve!provisioning=12345678-9abc-def0-1234-56789abcdef0"
r = requests.get(
    "https://pve.example.org:8006/api2/json/nodes/pve01/qemu",
    headers={"Authorization": f"PVEAPIToken={TOKEN}"},
    verify=False,
)
for vm in r.json()["data"]:
    print(vm["vmid"], vm["name"], vm["status"])
```

```
# pip install proxmoxer requests  (variante haut niveau)
from proxmoxer import ProxmoxAPI
prox = ProxmoxAPI("pve.example.org", user="tofu@pve",
```

```
                    token_name="provisioning",
                    token_value="12345678-9abc-...", verify_ssl=False)
for vm in prox.nodes("pve01").qemu.get():
    print(vm["vmid"], vm["name"])
```

5.4 C (libcurl)

```
/* gcc api.c -o api -lcurl */
#include <curl/curl.h>
int main(void) {
    CURL *c = curl_easy_init();
    struct curl_slist *h = NULL;
    h = curl_slist_append(h, "Authorization: PVEAPIToken="
        "tofu@pve!provisioning=12345678-9abc-def0-1234-56789abcdef0");
    curl_easy_setopt(c, CURLOPT_URL,
        "https://pve.example.org:8006/api2/json/nodes/pve01/qemu");
    curl_easy_setopt(c, CURLOPT_HTTPHEADER, h);
    curl_easy_setopt(c, CURLOPT_SSL_VERIFYPEER, 0L); /* prod: 1L */
    curl_easy_perform(c);
    curl_easy_cleanup(c);
    return 0;
}
```

5.5 C++ (libcurl)

```
// g++ api.cpp -o api -lcurl -std=c++17
#include <curl/curl.h>
#include <string>
int main() {
    CURL* c = curl_easy_init();
    curl_slist* h = nullptr;
    std::string tok = "tofu@pve!provisioning=12345678-9abc-...";
    h = curl_slist_append(h, ("Authorization: PVEAPIToken=" + tok).c_str());
    curl_easy_setopt(c, CURLOPT_URL,
        "https://pve.example.org:8006/api2/json/nodes/pve01/qemu");
    curl_easy_setopt(c, CURLOPT_HTTPHEADER, h);
    curl_easy_setopt(c, CURLOPT_SSL_VERIFYPEER, 0L);
    curl_easy_perform(c);
    curl_easy_cleanup(c);
}
```

5.6 Java

```
// Java 11+ : java.net.http.HttpClient
import java.net.URI;
import java.net.http.*;
public class Api {
  public static void main(String[] a) throws Exception {
    String token = "tofu@pve!provisioning=12345678-9abc-...";
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest req = HttpRequest.newBuilder()
        .uri(URI.create("https://pve.example.org:8006/api2/json/nodes/pve01/
            qemu"))
        .header("Authorization", "PVEAPIToken=" + token)
        .GET().build();
    HttpResponse<String> resp = client.send(req,
        HttpResponse.BodyHandlers.ofString());
```

```
    System.out.println(resp.body());
  }
}
```

5.7 Node.js (JavaScript)

```
// Node 18+ : fetch natif
const token = "tofu@pve!provisioning=12345678-9abc-...";
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0"; // cert auto-signé (dev)
const res = await fetch(
  "https://pve.example.org:8006/api2/json/nodes/pve01/qemu",
  { headers: { Authorization: `PVEAPIToken=${token}` } });
const { data } = await res.json();
data.forEach(vm => console.log(vm.vmid, vm.name, vm.status));
```

5.8 TypeScript

```
// TypeScript (Node 18+) : npm i -D typescript @types/node
interface VM { vmid: number; name: string; status: string; }
const token: string = "tofu@pve!provisioning=12345678-9abc-...";
const res: Response = await fetch(
  "https://pve.example.org:8006/api2/json/nodes/pve01/qemu",
  { headers: { Authorization: `PVEAPIToken=${token}` } });
const body = (await res.json()) as { data: VM[] };
body.data.forEach((vm: VM) => console.log(vm.vmid, vm.name));
```

5.9 Go

```
// go run api.go
package main

import (
    "crypto/tls"
    "fmt"
    "io"
    "net/http"
)

func main() {
    token := "tofu@pve!provisioning=12345678-9abc-..."
    tr := &http.Transport{TLSClientConfig: &tls.Config{InsecureSkipVerify: true
        }}
    client := &http.Client{Transport: tr}
    req, _ := http.NewRequest("GET",
        "https://pve.example.org:8006/api2/json/nodes/pve01/qemu", nil)
    req.Header.Set("Authorization", "PVEAPIToken="+token)
    resp, _ := client.Do(req)
    defer resp.Body.Close()
    body, _ := io.ReadAll(resp.Body)
    fmt.Println(string(body))
}
```

5.10 Rust

```rust
// Cargo.toml: reqwest = { version="0.12", features=["blocking","json"] }
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let token = "tofu@pve!provisioning=12345678-9abc-...";
    let client = reqwest::blocking::Client::builder()
        .danger_accept_invalid_certs(true) // prod: false
        .build()?;
    let resp = client
        .get("https://pve.example.org:8006/api2/json/nodes/pve01/qemu")
        .header("Authorization", format!("PVEAPIToken={token}"))
        .send()?
        .text()?;
    println!("{resp}");
    Ok(())
}
```

5.11 Ruby

```ruby
require "net/http"
require "uri"
require "json"
token = "tofu@pve!provisioning=12345678-9abc-..."
uri = URI("https://pve.example.org:8006/api2/json/nodes/pve01/qemu")
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE # prod: VERIFY_PEER
req = Net::HTTP::Get.new(uri)
req["Authorization"] = "PVEAPIToken=#{token}"
JSON.parse(http.request(req).body)["data"].each { |vm| puts "#{vm['vmid']} #{vm
    ['name']}" }
```

5.12 PHP

```php
<?php
$token = "tofu@pve!provisioning=12345678-9abc-...";
$ch = curl_init("https://pve.example.org:8006/api2/json/nodes/pve01/qemu");
curl_setopt_array($ch, [
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER     => ["Authorization: PVEAPIToken=$token"],
    CURLOPT_SSL_VERIFYPEER => false, // prod: true
]);
$data = json_decode(curl_exec($ch), true);
foreach ($data["data"] as $vm) { echo "{$vm['vmid']} {$vm['name']}\n"; }
```

5.13 C# / .NET

```csharp
// dotnet : HttpClient
using System.Net.Http;
var token = "tofu@pve!provisioning=12345678-9abc-...";
var handler = new HttpClientHandler {
    ServerCertificateCustomValidationCallback = (_, _, _, _) => true };
using var client = new HttpClient(handler);
client.DefaultRequestHeaders.Add("Authorization", $"PVEAPIToken={token}");
var json = await client.GetStringAsync(
    "https://pve.example.org:8006/api2/json/nodes/pve01/qemu");
Console.WriteLine(json);
```

5.14 PowerShell

```
$token = "tofu@pve!provisioning=12345678-9abc-..."
Invoke-RestMethod -Method Get `
  -Uri "https://pve.example.org:8006/api2/json/nodes/pve01/qemu" `
  -Headers @{ Authorization = "PVEAPIToken=$token" } `
  -SkipCertificateCheck | Select-Object -ExpandProperty data
```

Bibliothèques clientes dédiées

Plutôt que d'appeler l'API « à la main », il existe des clients de haut niveau : `proxmoxer` (Python), `proxmox-api` (Node.js), `go-proxmox` (Go), `Corsinvest.ProxmoxVE.Api` (.NET), `Fabric/Telmate` et le provider `bpg/proxmox` (OpenTofu, cf. parties 1 et 3).

Partie 6

Exemples d'utilisation de l'API GitLab

6.1 Introduction

Cette partie illustre l'appel de l'**API REST GitLab** (version `v4`) dans les principaux langages modernes, sur le même modèle que la partie 5. Tous les exemples suivent le même scénario : authentification par **jeton** (en-tête `PRIVATE-TOKEN`) et **liste des projets accessibles** (`GET /api/v4/projects`).

Authentification GitLab

L'API GitLab accepte plusieurs types de jetons : *Personal Access Token* (PAT), *Project / Group Access Token*, *CI Job Token* (`$CI_JOB_TOKEN`), ou OAuth2. On les transmet soit via l'en-tête `PRIVATE-TOKEN: <jeton>`, soit via `Authorization: Bearer <jeton>`. Le jeton doit porter les *scopes* adéquats (`api`, `read_api`...).

Sécurité du jeton GitLab

Comme pour Proxmox, le jeton ne doit jamais être commité en clair. En CI/CD on utilise `$CI_JOB_TOKEN` ou une variable masquée (cf. partie 4). Les exemples ci-dessous lisent le jeton depuis une variable et ciblent une instance auto-hébergée ; adapter l'URL et activer la vérification TLS en production.

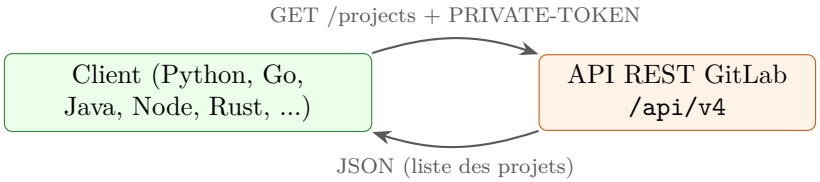


Figure 6.1: Schéma commun à tous les exemples : un client interroge l'API GitLab v4

6.2 Shell (curl)

```
TOKEN="glpat-xxxxxxxxxxxxxxxxxxxx"
curl -s --header "PRIVATE-TOKEN: ${TOKEN}" \
  "https://gitlab.example.org/api/v4/projects?membership=true&per_page=20" | jq
      .

# Déclencher un pipeline sur la branche main :
curl -s --request POST --header "PRIVATE-TOKEN: ${TOKEN}" \
  "https://gitlab.example.org/api/v4/projects/42/pipeline?ref=main"
```

6.3 Python 3

Deux variantes : `requests` (bas niveau) et la bibliothèque dédiée `python-gitlab` (haut niveau, cf. partie 4).

```
# pip install requests
import requests
TOKEN = "glpat-xxxxxxxxxxxxxxxxxxx"
r = requests.get(
    "https://gitlab.example.org/api/v4/projects",
    headers={"PRIVATE-TOKEN": TOKEN},
    params={"membership": True, "per_page": 20},
)
for p in r.json():
    print(p["id"], p["path_with_namespace"])
```

```
# pip install python-gitlab  (variante haut niveau)
import gitlab
gl = gitlab.Gitlab("https://gitlab.example.org", private_token="glpat-...")
gl.auth()
for p in gl.projects.list(membership=True, iterator=True):
    print(p.id, p.path_with_namespace)
```

6.4 C (libcurl)

```
/* gcc gl.c -o gl -lcurl */
#include <curl/curl.h>
int main(void) {
    CURL *c = curl_easy_init();
    struct curl_slist *h = NULL;
    h = curl_slist_append(h, "PRIVATE-TOKEN: glpat-xxxxxxxxxxxxxxxxxxx");
    curl_easy_setopt(c, CURLOPT_URL,
        "https://gitlab.example.org/api/v4/projects?membership=true");
    curl_easy_setopt(c, CURLOPT_HTTPHEADER, h);
    curl_easy_perform(c);
    curl_easy_cleanup(c);
    return 0;
}
```

6.5 C++ (libcurl)

```
// g++ gl.cpp -o gl -lcurl -std=c++17
#include <curl/curl.h>
#include <string>
int main() {
    CURL* c = curl_easy_init();
    curl_slist* h = nullptr;
    std::string tok = "glpat-xxxxxxxxxxxxxxxxxxx";
    h = curl_slist_append(h, ("PRIVATE-TOKEN: " + tok).c_str());
    curl_easy_setopt(c, CURLOPT_URL,
        "https://gitlab.example.org/api/v4/projects?membership=true");
    curl_easy_setopt(c, CURLOPT_HTTPHEADER, h);
    curl_easy_perform(c);
    curl_easy_cleanup(c);
}
```

6.6 Java

```java
// Java 11+ : java.net.http.HttpClient
import java.net.URI;
import java.net.http.*;
public class GlApi {
  public static void main(String[] a) throws Exception {
    String token = "glpat-xxxxxxxxxxxxxxxxxxxx";
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest req = HttpRequest.newBuilder()
        .uri(URI.create("https://gitlab.example.org/api/v4/projects?membership=
            true"))
        .header("PRIVATE-TOKEN", token)
        .GET().build();
    HttpResponse<String> resp = client.send(req,
        HttpResponse.BodyHandlers.ofString());
    System.out.println(resp.body());
  }
}
```

6.7 Node.js (JavaScript)

```javascript
// Node 18+ : fetch natif
const token = "glpat-xxxxxxxxxxxxxxxxxxxx";
const res = await fetch(
  "https://gitlab.example.org/api/v4/projects?membership=true&per_page=20",
  { headers: { "PRIVATE-TOKEN": token } });
const projects = await res.json();
projects.forEach(p => console.log(p.id, p.path_with_namespace));
```

6.8 TypeScript

```typescript
// TypeScript (Node 18+) : npm i -D typescript @types/node
interface Project { id: number; path_with_namespace: string; }
const token: string = "glpat-xxxxxxxxxxxxxxxxxxxx";
const res: Response = await fetch(
  "https://gitlab.example.org/api/v4/projects?membership=true",
  { headers: { "PRIVATE-TOKEN": token } });
const projects = (await res.json()) as Project[];
projects.forEach((p: Project) => console.log(p.id, p.path_with_namespace));
```

6.9 Go

```go
// go run gl.go
package main

import (
    "fmt"
    "io"
    "net/http"
)

func main() {
    token := "glpat-xxxxxxxxxxxxxxxxxxxx"
    req, _ := http.NewRequest("GET",
        "https://gitlab.example.org/api/v4/projects?membership=true", nil)
```

```
    req.Header.Set("PRIVATE-TOKEN", token)
    resp, _ := http.DefaultClient.Do(req)
    defer resp.Body.Close()
    body, _ := io.ReadAll(resp.Body)
    fmt.Println(string(body))
}
```

6.10 Rust

```
// Cargo.toml: reqwest = { version="0.12", features=["blocking","json"] }
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let token = "glpat-xxxxxxxxxxxxxxxxxxxx";
    let client = reqwest::blocking::Client::new();
    let resp = client
        .get("https://gitlab.example.org/api/v4/projects?membership=true")
        .header("PRIVATE-TOKEN", token)
        .send()?
        .text()?;
    println!("{resp}");
    Ok(())
}
```

6.11 Ruby

```
require "net/http"
require "uri"
require "json"
token = "glpat-xxxxxxxxxxxxxxxxxxxx"
uri = URI("https://gitlab.example.org/api/v4/projects?membership=true")
req = Net::HTTP::Get.new(uri)
req["PRIVATE-TOKEN"] = token
res = Net::HTTP.start(uri.host, uri.port, use_ssl: true) { |h| h.request(req) }
JSON.parse(res.body).each { |p| puts "#{p['id']} #{p['path_with_namespace']}" }
```

6.12 PHP

```
<?php
$token = "glpat-xxxxxxxxxxxxxxxxxxxx";
$ch = curl_init("https://gitlab.example.org/api/v4/projects?membership=true");
curl_setopt_array($ch, [
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER     => ["PRIVATE-TOKEN: $token"],
]);
$projects = json_decode(curl_exec($ch), true);
foreach ($projects as $p) { echo "{$p['id']} {$p['path_with_namespace']}\n"; }
```

6.13 C# / .NET

```
// dotnet : HttpClient
using System.Net.Http;
var token = "glpat-xxxxxxxxxxxxxxxxxxxx";
using var client = new HttpClient();
client.DefaultRequestHeaders.Add("PRIVATE-TOKEN", token);
var json = await client.GetStringAsync(
```

```
    "https://gitlab.example.org/api/v4/projects?membership=true");
Console.WriteLine(json);
```

6.14 PowerShell

```
$token = "glpat-xxxxxxxxxxxxxxxxxxx"
Invoke-RestMethod -Method Get `
  -Uri "https://gitlab.example.org/api/v4/projects?membership=true" `
  -Headers @{ "PRIVATE-TOKEN" = $token } |
  Select-Object id, path_with_namespace
```

Bibliothèques clientes dédiées GitLab

Des clients de haut niveau évitent d'écrire les requêtes à la main : `python-gitlab` (Python), `@gitbeaker/rest` (Node.js / TypeScript), `go-gitlab` (Go), `gitlab4j-api` (Java), `NGitLab` (.NET), `gitlab` gem (Ruby). Voir la webographie pour les liens.

Lexique français détaillé

Agentless (sans agent)
Mode de fonctionnement (Ansible) où aucun logiciel permanent n'est installé sur les machines cibles ; la communication se fait par SSH.

Ansible
Outil de gestion de configuration et d'orchestration sans agent, décrivant l'état des systèmes en YAML (playbooks).

Ansible Vault
Mécanisme de chiffrement (AES-256) intégré à Ansible pour protéger les données sensibles (mots de passe, clés, secrets) au sein du dépôt.

API (Interface de programmation)
Ensemble de points d'entrée permettant à des programmes de dialoguer. L'API Proxmox est de type REST sur HTTPS (port 8006).

API Token (jeton d'API)
Identifiant d'authentification révocable et à droits limités, indépendant du mot de passe, recommandé pour l'automatisation.

Backend (d'état)
Emplacement de stockage du fichier d'état OpenTofu ; peut être local ou distant (S3, GitLab, etc.) pour le travail en équipe.

Cloud-init
Standard de personnalisation d'une instance Linux au premier démarrage (utilisateurs, clés SSH, réseau, paquets, clavier).

Cluster
Regroupement de plusieurs nœuds Proxmox gérés comme un ensemble, avec migration des VMs et haute disponibilité.

Conteneur (LXC)
Virtualisation légère partageant le noyau de l'hôte ; réservée aux distributions Linux, très économe en ressources.

CSRF (jeton anti-CSRF)
Jeton de sécurité fourni avec le ticket Proxmox pour prévenir les attaques par falsification de requête.

Datasource (Cloud-init)
Source de configuration lue par Cloud-init au boot ; Proxmox utilise le type *NoCloud*.

Datastore
Espace de stockage Proxmox (disques de VMs, ISO, modèles, sauvegardes, snippets).

Déduplication
Technique (PBS) ne stockant qu'une seule fois les blocs de données identiques, réduisant fortement la taille des sauvegardes.

DevOps
Ensemble de pratiques rapprochant développement et exploitation pour automatiser et fiabiliser les livraisons.

DevSecOps
Extension du DevOps intégrant la sécurité à toutes les étapes de la chaîne (« sécurité par conception »).

DMZ (zone démilitarisée)
Segment réseau intermédiaire exposant des services à Internet tout en isolant le réseau interne.

Facts (Ansible)
Informations collectées automatiquement sur une machine cible (OS, réseau, matériel) et utilisables dans les playbooks.

Forge logicielle
Plateforme d'hébergement de code et d'outils de développement (ici GitLab).

GitLab
Forge logicielle auto-hébergée incluant dépôts Git, suivi de tickets et intégration/déploiement continus (CI/CD).

Handler (Ansible)
Tâche déclenchée uniquement sur notification (ex. redémarrer un service après modification de sa configuration).

HCL2 (HashiCorp Configuration Language)
Langage déclaratif utilisé par OpenTofu/Terraform pour décrire l'infrastructure.

Hyperviseur
Logiciel exécutant et gérant des machines virtuelles (Proxmox VE s'appuie sur KVM/QEMU).

IaC (Infrastructure as Code)
Pratique consistant à décrire et gérer l'infrastructure par du code versionné plutôt que manuellement.

Idempotence
Propriété garantissant qu'exécuter plusieurs fois une opération produit le même résultat final, sans effet de bord superflu.

Inventaire (Ansible)
Liste structurée des hôtes gérés et de leurs variables, statique ou dynamique.

KVM/QEMU
Technologie de virtualisation matérielle du noyau Linux (KVM) couplée à l'émulateur QEMU, base des VMs Proxmox.

LXC (Linux Containers)
Technologie de conteneurs système Linux utilisée par Proxmox.

Module (Ansible)
Unité d'action idempotente invoquée par une tâche (`apt`, `copy`, `user`, `template`...).

NAT (traduction d'adresses)
Mécanisme du pare-feu réécrivant les adresses IP entre réseaux (typiquement interne vers Internet).

NoCloud
Type de datasource Cloud-init fournissant la configuration via un support local (utilisé par Proxmox).

Nœud (node)
Serveur physique membre d'un cluster Proxmox.

OPNsense
Distribution libre de pare-feu / routeur (basée sur FreeBSD), assurant filtrage, NAT, VPN et segmentation réseau.

OpenTofu
Outil libre de provisionnement d'infrastructure déclaratif, fork de Terraform sous gouvernance de la Linux Foundation.

Orchestration
Coordination ordonnée de tâches sur plusieurs machines (ex. déploiement multi-tiers).

Pare-feu (firewall)
Dispositif filtrant le trafic réseau selon des règles de sécurité.

pfSense
Distribution libre de pare-feu / routeur basée sur FreeBSD, proche d'OPNsense (dont elle est l'ancêtre commun).

Plan / Apply (OpenTofu)
`plan` prévisualise les changements ; `apply` les exécute.

Playbook (Ansible)
Fichier YAML décrivant des *plays* qui associent des hôtes à des tâches.

Pool (Proxmox)
Regroupement logique de ressources (VMs, stockages) pour en simplifier la gestion et les droits.

Provider (OpenTofu)
Greffon traduisant les ressources HCL2 en appels vers une API cible (ici `bpg/proxmox`).

Provisionnement
Action de mettre à disposition et configurer une ressource informatique (machine, réseau, service).

Provisionneur (OpenTofu)
Mécanisme exécutant des actions après création d'une ressource : `remote-exec` (shell distant), `local-exec` (commande locale, ex. lancer Ansible).

Proxmox Backup Server (PBS)
Serveur de sauvegarde déduppliquée et chiffrée des VMs et conteneurs Proxmox.

Proxmox VE (PVE)
Plateforme libre de virtualisation (VMs KVM et conteneurs LXC) basée sur Debian.

RBAC (contrôle d'accès basé sur les rôles)
Modèle d'autorisation attribuant des permissions via des rôles (utilisé par l'API Proxmox).

Reverse-proxy
Serveur intermédiaire recevant les requêtes Internet et les redirigeant vers les services internes (ex. Nginx, Traefik).

Rôle (Ansible)
Unité réutilisable et structurée regroupant tâches, variables, modèles et handlers.

runcmd
Section Cloud-init listant des commandes exécutées au premier boot.

Snippet (Proxmox)
Fichier de configuration (ex. user-data Cloud-init) stocké sur un datastore et référencé par une VM.

SOPS
Outil de chiffrement de fichiers de secrets (YAML/JSON/.env) souvent utilisé avec OpenTofu.

SSH (Secure Shell)
Protocole d'accès distant chiffré ; clé publique déposée sur la cible, clé privée gardée par le client.

State (état OpenTofu)
Fichier mémorisant la correspondance entre le code et les ressources réelles ; à protéger (peut contenir des secrets).

Template (modèle)
Image de base (VM ou conteneur) servant de patron pour cloner de nouvelles instances.

Terraform
Outil d'IaC d'HashiCorp dont OpenTofu est issu (compatibilité HCL2 et providers).

Ticket (Proxmox)
Jeton d'authentification temporaire (2h) obtenu par identifiant/mot de passe.

VLAN
Réseau local virtuel segmentant logiquement un réseau physique.

VPN (réseau privé virtuel)
Tunnel chiffré reliant des sites ou des utilisateurs distants au réseau interne.

VM (machine virtuelle)
Système complet virtualisé avec son propre noyau, exécuté par l'hyperviseur.

YAML
Format de sérialisation lisible utilisé par Ansible et Cloud-init.

Bibliographie

Sélection d'ouvrages et de ressources de référence, classés par thème. Les éditions évoluent ; privilégier la plus récente.

Proxmox VE et virtualisation

- *Mastering Proxmox* — Wasim Ahmed, Packt Publishing.
- *Proxmox VE Administration Guide* — Documentation officielle, Proxmox Server Solutions GmbH (https://pve.proxmox.com/pve-docs/).
- *Proxmox High Availability* — Simon M.C. Cheng, Packt Publishing.
- *KVM Virtualization Cookbook* — Konstantin Ivanov, Packt Publishing.

OpenTofu et Terraform

- *Terraform: Up & Running* — Yevgeniy Brikman, O'Reilly.
- *Terraform in Action* — Scott Winkler, Manning.
- *The Terraform Book* — James Turnbull (auto-édité).
- Documentation OpenTofu officielle (https://opentofu.org/docs/).
- Provider *bpg/proxmox* — registre OpenTofu / GitHub.

Ansible

- *Ansible for DevOps* — Jeff Geerling, LeanPub.
- *Ansible: Up & Running* — Lorin Hochstein & René Moser, O'Reilly.
- *Mastering Ansible* — James Freeman, Packt Publishing.
- Documentation Ansible officielle (https://docs.ansible.com/).

Pare-feu : OPNsense et pfSense

- *OPNsense Beginner to Professional* — Julio Cesar Bueno de Camargo, Packt.
- *The Book of PF* — Peter N.M. Hansteen, No Starch Press.
- *pfSense: The Definitive Guide* — Christopher M. Buechler & Jim Pingle.
- Documentation OPNsense officielle (https://docs.opnsense.org/).

DevOps et DevSecOps

- *The Phoenix Project* — Gene Kim, Kevin Behr, George Spafford, IT Revolution.
- *The DevOps Handbook* — Gene Kim, Jez Humble, Patrick Debois, John Willis.
- *Site Reliability Engineering* — Betsy Beyer et al., Google / O'Reilly.
- *Accelerate* — Nicole Forsgren, Jez Humble, Gene Kim, IT Revolution.
- *Securing DevOps* — Julien Vehent, Manning.
- *Infrastructure as Code* — Kief Morris, O'Reilly.

Sécurité, SSH et secrets

- *SSH Mastery* — Michael W. Lucas, Tilted Windmill Press.
- *Linux Hardening in Hostile Networks* — Kyle Rankin, Addison-Wesley.
- Documentation *SOPS* et *HashiCorp Vault*.

Webographie et liens de référence

Liens de référence sur l'ensemble des sujets traités dans ce document.

★ Documentation à mettre en évidence

API officielle de Proxmox (détail des points d'entrée REST) :

- Viewer interactif de l'API : https://pve.proxmox.com/pve-docs/api-viewer/
- Guide « Proxmox VE API » : https://pve.proxmox.com/wiki/Proxmox_VE_API
- Documentation administrateur : https://pve.proxmox.com/pve-docs/

Provider OpenTofu / Terraform pour Proxmox :

- Provider `bpg/proxmox` (recommandé) — code et doc : https://github.com/bpg/terraform-provider-proxmox
- Registre OpenTofu : https://search.opentofu.org/provider/bpg/proxmox/latest
- Registre Terraform : https://registry.terraform.io/providers/bpg/proxmox/latest/docs
- Provider historique `Telmate/proxmox` : https://registry.terraform.io/providers/Telmate/proxmox/latest/docs

Proxmox VE et PBS

- Site officiel : https://www.proxmox.com/
- Documentation PVE : https://pve.proxmox.com/pve-docs/
- Documentation PBS : https://pbs.proxmox.com/docs/
- Wiki Proxmox : https://pve.proxmox.com/wiki/Main_Page
- Forum communautaire : https://forum.proxmox.com/

OpenTofu et Terraform

- OpenTofu — site et docs : https://opentofu.org/docs/
- Registre de providers OpenTofu : https://search.opentofu.org/
- Langage HCL : https://developer.hashicorp.com/terraform/language
- TFLint : https://github.com/terraform-linters/tflint

Ansible

- Documentation : https://docs.ansible.com/
- Galaxy (rôles/collections) : https://galaxy.ansible.com/
- Ansible Vault : https://docs.ansible.com/ansible/latest/vault_guide/index.html
- ansible-lint : https://ansible.readthedocs.io/projects/lint/
- Collection `community.general.proxmox` : https://docs.ansible.com/ansible/latest/collections/community/general/

Cloud-init

- Documentation officielle : https://cloudinit.readthedocs.io/
- Référence des modules `cloud-config` : https://cloudinit.readthedocs.io/en/latest/reference/modules.html
- Cloud-init sur Proxmox : https://pve.proxmox.com/wiki/Cloud-Init_Support

OPNsense et pfSense

- OPNsense — documentation : https://docs.opnsense.org/
- OPNsense — API : https://docs.opnsense.org/development/api.html
- pfSense — documentation : https://docs.netgate.com/pfsense/en/latest/

GitLab CI/CD et DevSecOps

- Référence `.gitlab-ci.yml` : https://docs.gitlab.com/ee/ci/yaml/
- Variables CI/CD (masquées/protégées) : https://docs.gitlab.com/ee/ci/variables/
- Sécurité applicative (SAST, etc.) : https://docs.gitlab.com/ee/user/application_security/
- OIDC GitLab → HashiCorp Vault : https://docs.gitlab.com/ee/ci/secrets/

Outils de qualité et de sécurité

- yamllint : https://yamllint.readthedocs.io/
- ShellCheck : https://www.shellcheck.net/
- hadolint : https://github.com/hadolint/hadolint
- tfsec : https://github.com/aquasecurity/tfsec
- Checkov : https://www.checkov.io/
- gitleaks : https://github.com/gitleaks/gitleaks
- SOPS : https://github.com/getsops/sops

Bibliothèques clientes de l'API Proxmox

- proxmoxer (Python) : https://github.com/proxmoxer/proxmoxer
- proxmox-api (Node.js) : https://www.npmjs.com/package/proxmox-api
- go-proxmox (Go) : https://github.com/luthermonson/go-proxmox
- Corsinvest API (.NET) : https://github.com/Corsinvest/cv4pve-api-dotnet

Bibliothèques clientes de l'API GitLab

- python-gitlab (Python) : https://github.com/python-gitlab/python-gitlab
- gitbeaker (Node.js/TypeScript) : https://github.com/jdalrymple/gitbeaker
- go-gitlab (Go) : https://gitlab.com/gitlab-org/api/client-go
- gitlab4j-api (Java) : https://github.com/gitlab4j/gitlab4j-api

- NGitLab (.NET) : https://github.com/ubisoft/NGitLab
- gitlab gem (Ruby) : https://github.com/NARKOZ/gitlab

Blogs, vidéos et influenceurs Dev(Sec)Ops

Ressources pédagogiques de qualité (en français et en anglais) couvrant Ansible, Terraform/OpenTofu, Proxmox, GitLab CI et le DevSecOps :

- **Stéphane Robert** — blog technique francophone très complet (Ansible, Terraform/OpenTofu, Proxmox, CI/CD, sécurité) :
  https://blog.stephane-robert.info/
- **Xavki** — tutoriels vidéo francophones (Ansible, Terraform, GitLab CI, Docker, Kubernetes) :
  chaîne YouTube : https://www.youtube.com/@xavki
  blog : https://blog.xavki.fr/
- **Jeff Geerling** — auteur d'*Ansible for DevOps*, vidéos Ansible / homelab / Proxmox : https://www.youtube.com/@JeffGeerling
- **TechWorld with Nana** — DevOps, CI/CD, conteneurs (anglais) :
  https://www.youtube.com/@TechWorldwithNana
- **Grafikart** — tutoriels développement et DevOps (français) :
  https://grafikart.fr/
- **Anton Putra** — Terraform/OpenTofu, cloud, observabilité :
  https://www.youtube.com/@AntonPutra
- **Techno Tim** — homelab, Proxmox, IaC (anglais) :
  https://www.youtube.com/@TechnoTim
- **Cookie connecté** — vulgarisation cybersécurité (français) :
  https://www.youtube.com/@cookieconnecte

Dépôt du projet

- Dépôt IaC LinuXmaine :
  https://framagit.org/linuxmaine/infra-linuxmaine-v2.0/-/tree/main/iac
- Association LinuXmaine : https://linuxmaine.org/

Document rédigé pour LinuXmaine.org — Thierry Gayet — May 31, 2026