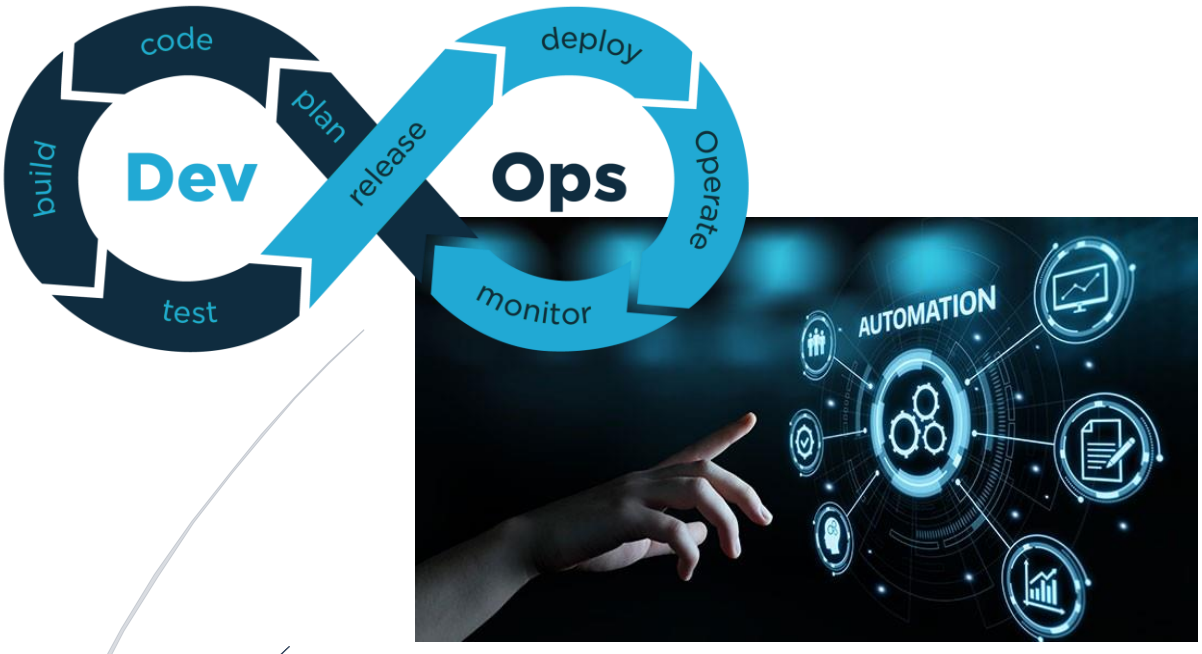
EVALUATION EN COURS DE FORMATION

PARCOURS ADMINISTRATEUR SYSTEME DEVOPS



Automatiser le déploiement d'une infrastructure

Evaluation : ECF2

Version : du 10/11/2025

Auteur : FT

Sommaire

Table des matières

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF2	Titre	**Automatiser le déploiement d'une infrastructure**	Du	10/11/2025

1. PRESENTATION

1.1. DESCRIPTION DE LA COMPETENCE – PROCESSUS DE MISE EN ŒUVRE

- Connaissance d'un outil d'automatisation de déploiement d'infrastructure de type Ansible et Terraform
- Dialoguer avec les principaux fournisseurs de services
- Consulter de la documentation technique rédigée en anglais
    - Ansible Galaxy
    - Terraform by HashiCorp
- Effectuer une veille technologique

1.2. CONTEXTE(S) PROFESSIONNEL(S) DE MISE EN ŒUVRE

Ce travail s'effectue seul, en relation avec les équipes en charge du réseau et de la sécurité.

1.3. CRITERES DE PERFORMANCE

- Définir l'architecture système à déployer
- Décrire la configuration des serveurs à déployer.
- Concevoir les scripts nécessaires
- Ecrire et tester les scripts, lancer le déploiement
- Les serveurs sont fonctionnels
- La configuration est conforme au cahier des charges
- Les documentations sont mises à jour

2. ECF2 - QUESTIONS

2.1. Connaissance d'un outil d'automatisation de déploiement d'infrastructure type Ansible et Terraform

a. Quelle est la principale utilité de manipuler les IaC ?

L'Infrastructure as Code est définie comme la **pratique continue d'utilisation de langages de programmation et d'un code lisible par la machine pour gérer et configurer l'infrastructure informatique**, au lieu d'une configuration manuelle.

L'infrastructure informatique comprend les serveurs, les centres de stockage, les bases de données, les réseaux et les serveurs web.

Le **processus de provisionnement de l'infrastructure est automatisé**, évitant ainsi aux développeurs de procéder à des configurations manuelles à chaque étape de développement et de déploiement de l'application.

Les avantages de l'IaC sont les suivants :

1. **Cohérence et redéploiement accrues :**
   L'utilisation d'une infrastructure automatisée évite les interruptions de service des applications et réduit le risque d'erreurs lors de la configuration manuelle.
   Les environnements configurés peuvent être réutilisés pour différentes applications.
2. **Collaboration et contrôle des versions :**
   Les modifications manuelles apportées aux systèmes existants, comme l'ajout de nouvelles fonctionnalités, sont risquées.
   Iac garantit le stockage des versions précédentes du code source. Ainsi, les utilisateurs peuvent rapidement revenir aux anciennes versions en cas de failles de sécurité ou d'imprévus.
3. **Gestion des coûts :**
   Le provisionnement et le développement traditionnels s'effectuent dans des centres de données physiques. Leur maintenance par les organisations est coûteuse et n'est pas avantageuse pour les petites entreprises. Les fournisseurs de services cloud fournissent des environnements virtuels pour l'automatisation et le déploiement, selon un système de paiement à l'utilisation.
4. **Sécurité et conformité :**
   L'utilisation d'une infrastructure automatisée garantit que les entreprises et les organisations déploient leurs applications conformément aux directives établies, dans des conditions de travail sécurisées. L'infrastructure est construite en tenant compte des protocoles de sécurité.

b. Quelle est la différence entre Terraform et Ansible ?

Terraform et Ansible peuvent tous deux orchestrer et provisionner l'infrastructure.

Terraform est plus complet dans la **gestion de l'infrastructure**, tandis **qu'Ansible** excelle dans la **gestion de la configuration.**

Terraform est particulièrement adapté aux activités du 1er jour, comme la création de l'infrastructure, tandis qu'Ansible est idéal pour les activités des jours suivants.

Dans certains cas, il peut être préférable d'utiliser Terraform et Ansible ensemble plutôt que de choisir l'un plutôt que l'autre. Si Terraform excelle dans l'orchestration d'infrastructures sur plusieurs plateformes cloud, Ansible excelle dans la gestion du provisionnement et de la configuration.

Pour résumer :

- **Ansible** se concentre sur l'installation et la gestion des logiciels sur les serveurs existants.

Il permet d'automatiser les tâches de gestion de configuration, facilitant ainsi la configuration cohérente de tous les serveurs d'un environnement.

- **Terraform** se concentre sur la configuration initiale des serveurs et autres composants de l'infrastructure. Il automatise le processus de provisionnement des nouvelles infrastructures, le rendant ainsi plus rapide et plus fiable.

c. Quel est le principal langage utilisé par Terraform et par Ansible ?

Terraform utilise le langage HCL déclaratif pour définir l'infrastructure en tant que code, permettant une gestion efficace des dépendances, en décrivant l'état souhaité.

À l'inverse, **Ansible utilise des scripts YAML impératifs** qui s'exécutent séquentiellement, permettant une gestion détaillée des tâches. YAML est un format de données populaire et simple, facile à comprendre pour les humains.

	Ansible	Terraform
Type	Outil de Gestion de la Configuration	Outil d'Orchestration
Syntax	YAML	HCL
Language	Impératif (gestion détaillée des tâches)	Déclaratif (décrit l'état souhaité)
Approche	Infrastructure mutable (modifiable après déploiement)	Infrastructure immuable (ne change jamais après déploiement)

2.2. Dialoguer avec les principaux fournisseurs de services

a. Quels sont les principaux fonctionnements de IaC ?

Les principaux fonctionnements de l'Infrastructure as Code reposent sur plusieurs concepts:

- **Codification de l'infrastructure**

  Au lieu de configurer manuellement les serveurs, les réseaux, le stockage, les services, l'IaC permet de définir cette infrastructure à l'aide de fichiers de configuration.

  Ces fichiers sont généralement écrits dans un langage de description spécifique (comme YAML, JSON) ou dans un langage de programmation.

- **Automatisation du déploiement :**

  Une fois l'infrastructure définie avec le code, des outils d'IaC automatisent le processus de provisionnement et de configuration de cette infrastructure.

- **Gestion de version :**

  Les fichiers de configuration de l'IaC peuvent être stockés dans des systèmes de contrôle de version (comme Git) et permettent de suivre les modifications apportées à l'infrastructure au fil du temps, de revenir à des états précédents en cas de problème, et de faciliter la collaboration entre les équipes.

- **Approche déclarative vs. impérative :**
  - ✓ **Déclarative :**

    L'état final souhaité de l'infrastructure est décrit, et l'outil d'IaC se charge de déterminer les étapes nécessaires pour atteindre cet état. Des outils comme Terraform et AWS CloudFormation utilisent cette approche.

  - ✓ **Impérative :**

    On spécifie les commandes exactes à exécuter pour configurer l'infrastructure. Des outils comme Ansible et Chef peuvent être utilisés de manière impérative.

b. Comment communique-t-il avec les principaux fournisseurs de services ?

La communication entre les outils IaC et les principaux fournisseurs de services cloud comme AWS, Azure, Google Cloud Platform s'effectue principalement par le biais de plusieurs mécanismes :

- **API (Application Programming Interface) :**

  C'est le mécanisme principal de communication. Les fournisseurs de services cloud exposent des API RESTful qui permettent aux outils d'IaC d'envoyer des requêtes pour créer, modifier, lire ou supprimer des ressources.

- **SDK (Software Development Kit) :** Les fournisseurs de cloud proposent souvent des SDK dans différents langages de programmation (Python, Java, Go, …).

  Ces SDK encapsulent les appels d'API et offrent une interface plus conviviale pour interagir avec les services cloud.

- **CLI (Command-Line Interface) :**

  Des interfaces en ligne de commande (comme AWS CLI, Azure CLI, gcloud CLI) sont utilisées pour exécuter des commandes spécifiques nécessaires à la gestion de l'infrastructure.

- **Providers spécifiques :**

  Les outils d'IaC comme Terraform, Ansible, et CloudFormation utilisent des modules spécifiques à chaque fournisseur de cloud.

  Par exemple, Terraform a un module AWS, un module Azure, etc.

c. Citer trois exemples de communication (AWS, Kubernetes, docker)

- **Communication entre AWS et Ansible:**

Cette communication repose principalement sur l'utilisation des modules AWS d'Ansible. Ces modules, écrits en Python, utilisent la librairie boto3 (le SDK Python pour AWS) pour interagir avec les API des services AWS.

L'exécution de tâches se fait par la création d'un Playbook qui définit une tâche nommée comme "Lancer une instance EC2", la configuration de variables sont définies pour la région AWS, le type d'instance, l'AMI (Amazon Machine Image), et la clé EC2, des informations d'authentification sont aussi renseignées pour pouvoir interagir avec l'API AWS.

- **Communication entre AWS et Docker:**

La communication entre Docker et AWS est principalement axée sur l'exécution et la gestion d'applications conteneurisées avec Docker sur l'infrastructure Amazon Web Services.

Cette communication peut se faire avec le module ECS qui est un service d'orchestration de conteneurs entièrement géré par AWS, et Docker est la technologie de conteneurisation qu'ECS utilise pour exécuter et gérer les applications

La communication entre ECS et Docker repose sur :

- ✓ L'Agent ECS qui agit comme un intermédiaire entre le service de contrôle ECS et le daemon Docker.
- ✓ L'API Docker utilisée par l'Agent ECS pour donner des instructions au moteur Docker sur la gestion des conteneurs.
- ✓ Les Task Definitions qui fournissant les spécifications nécessaires à la création et à l'exécution des conteneurs Docker.

- **Communication entre Kubernetes et Docker:**

La communication entre Kubernetes et Docker est fondamentale pour le fonctionnement de Kubernetes en tant qu'orchestrateur de conteneurs.

Exemple de communication :

1. On applique un fichier YAML avec la commande kubectl (pour interagir avec un cluster Kubernetes)
2. Le kubelet reçoit l'instruction (agent qui s'exécute sur chaque nœud dans un cluster Kubernetes)
3. Le kubelet demande à Docker de lancer un conteneur avec une image spécifique.
4. Docker télécharge l'image et crée le conteneur.
5. Le kubelet surveille l'état du conteneur via Docker.

2.3. Consulter de la documentation technique rédigée en anglais (Ansible/Terraform)

a. Ansible Galaxy

[Expliquer le fonctionnement des playbooks et des rôles dans Ansible en utilisant la documentation Ansible]

- **Les Playbooks**

Les playbooks sont des modéles d'automatisation, au format YAML, qu'Ansible utilise pour déployer et configurer les nœuds d'un inventaire, en exécutant les fonctions suivantes :

- ✓ **Exécution de tâches** avec des privilèges élevés ou en tant qu'utilisateur différent.
- ✓ **Utilisation de boucles** pour répéter des tâches pour les éléments d'une liste.
- ✓ **Déléguer des playbooks** pour exécuter des tâches sur différentes machines.
- ✓ **Exécution de tâches conditionnelles** et évaluation des conditions avec des tests de playbook.
- ✓ **Utiliser des blocs** pour regrouper des ensembles de tâches.

Il est possible d'utiliser les playbooks Ansible en utilisant des collections, en créant des fichiers et des rôles réutilisables, en incluant et en important des playbooks et en exécutant des parties sélectionnées d'un playbook avec des balises.

L'objectif principal d'un playbook est d'automatiser la configuration, le déploiement et la gestion de systèmes de manière reproductible et prévisible.

Exemple de playbook :

```
---
- name: Afficher un message
  hosts: all
  tasks:
    - name: Afficher un message de bienvenue
      debug:
        msg: "Bienvenue dans Ansible !"
```

Explication du playbook :

1. **--- :** Indique le **début** du document YAML.
2. **- name: Afficher un message :** Définit le **nom** de ce "play". Ce play a pour objectif d'afficher un message.
3. **hosts: all :** Spécifie qu'il sera exécuté **sur tous les hôtes** définis dans l'inventaire Ansible.
4. **tasks: :** Contient la **liste des tâches** à exécuter. Ici, il n'y a qu'une seule tâche.
5. **- name: Afficher un message de bienvenue :** Définit le nom de cette tâche spécifique.
6. **debug: :** Utilise le module Ansible debug. Ce module est utile pour afficher des informations pendant l'exécution du playbook, comme des messages ou des valeurs de variables.
7. **msg: "Bienvenue dans Ansible !" :** Paramètre du module debug. msg spécifie le message à afficher sur la console des hôtes gérés lors de l'exécution de cette tâche.

- ## Les Rôles

Les Rôles permettent de charger automatiquement des variables, fichiers, tâches, gestionnaires et autres artefacts Ansible associés, en fonction d'une structure de fichiers connue.

Une fois le contenu regroupé en rôles, on peut facilement le réutiliser et le partager avec d'autres utilisateurs.

Ils permettent de décomposer les playbooks complexes en unités plus petites et gérables.

Un rôle Ansible possède une structure de répertoires définie, composée **de sept répertoires principaux** standard.

```
my_role/
├── defaults/
│   └── main.yml      # Variables par défaut du rôle (priorité la plus basse)
├── files/
│   └── ...           # Fichiers statiques à copier sur les hôtes
├── handlers/
│   └── main.yml      # Handlers (tâches spéciales exécutées sur notification)
├── meta/
│   └── main.yml      # Métadonnées du rôle (auteur, licence, dépendances, etc.)
├── tasks/
│   └── main.yml      # Tâches principales exécutées par le rôle
├── templates/
│   └── ...           # Fichiers de modèles Jinja2 à générer sur les hôtes
└── vars/
    └── main.yml      # Variables spécifiques au rôle (priorité plus élevée que defaults)
```

Pour utiliser un rôle dans un playbook, on utilise la directive **roles**.

```
---
- name: Configurer des serveurs web avec des rôles
- hosts: webservers
  roles:
    - role : common          # Applique le rôle "common"
    - role : webservers      # Applique le rôle "webservers"
        vars:                # Définition de variables spécifiques à cette instance du rôle
          http_port: 8080 # Utilise le port web 8080
    - role: database         # Applique le rôle "database"
```

En résumé, les rôles Ansible sont un mécanisme puissant pour organiser, réutiliser et partager l'automatisation Ansible.
Ils imposent une structure de répertoire standard et permettent de décomposer des configurations complexes en unités logiques et gérables.

b. Terraform by HashiCorp

[Expliquer le fonctionnement des providers et leurs rôles dans l'architecture de Terraform et aussi expliquer la syntaxe du provider Docker]

- ### Les Providers

Chaque fournisseur ajoute un ensemble de types de ressources et/ou de sources de données que Terraform peut gérer. Chaque type de ressource est implémenté par un fournisseur ; sans fournisseurs, Terraform ne peut gérer aucun type d'infrastructure.

Le registre Terraform est le répertoire principal des fournisseurs Terraform accessibles au public et héberge des fournisseurs pour la plupart des principales plates-formes d'infrastructure.

✓ **Roles :**

Le rôle principal des providers Terraform est de servir de ponts entre Terraform et les différentes infrastructures. Ils permettent à Terraform de :

- **Comprendre et interagir avec l'API spécifique** de chaque fournisseur (cloud, SaaS, etc.).
- **Définir les types de ressources** que Terraform peut gérer pour ce fournisseur (ex: instances, bases de données, réseaux).
- **Implémenter les opérations CRUD** (Créer, Lire, Mettre à jour, Supprimer) pour chaque type de ressource via l'API du fournisseur.
- **Traduire la configuration Terraform déclarative** en appels d'API spécifiques au fournisseur.
- **Gérer l'état actuel des ressources** en communiquant avec l'API du fournisseur et en mettant à jour le fichier d'état Terraform.
- **Fournir des sources de données** pour récupérer des informations sur des ressources existantes sans les gérer directement.
- **Gérer l'authentification et l'autorisation** nécessaires pour interagir avec l'API du fournisseur.

✓ **Fonctionnement :**

Le fonctionnement des providers Terraform repose sur les étapes suivantes :

1. **Initialisation (terraform init)** : Terraform télécharge les providers spécifiés dans votre configuration depuis le Terraform Registry ou d'autres sources configurées.
2. **Configuration** : Dans les fichiers .tf, on déclare les ressources à gérer et on spécifie le provider à utiliser
3. **Planification (terraform plan)** : Terraform Core lit la configuration.
4. **Application (terraform apply) :** Pour chaque action (création, mise à jour, suppression), Terraform Core appelle le provider approprié.
5. **Mise à jour de l'État :** Terraform Core met à jour le fichier d'état (terraform.tfstate) avec de nouvelles informations.

- ## Docker provider

✓ Le fournisseur Docker permet d'interagir avec les conteneurs et images Docker.

✓ Grâce à l'utilisation de l'API Docker, le fournisseur Docker est immédiatement compatible avec un serveur Docker unique, mais aussi avec Swarm (orchestration des conteneurs).

✓ La syntaxe du provider Docker pour Terraform se définit dans un bloc provider "docker" {}.

Ce bloc permet de configurer la connexion à une instance Docker et de spécifier d'autres paramètres nécessaires à l'interaction avec le Docker Daemon.

```
terraform {
  required_providers {                    # Spécifie les providers Terraform nécessaires pour exécuter cette configuration
    docker = {                            # Définit le provider Docker requis.
      source  = "kreuzwerker/docker"      # Indique la source du provider Docker
      version = "3.0.2"
    }
  }
}
provider "docker" {                       # Ce bloc configure l'instance du provider Docker
  host = "unix:///var/run/docker.sock"    # Spécifie l'adresse du Docker Daemon auquel Terraform doit se connecter
}
# Pulls the image
resource "docker_image" "ubuntu" {        # Indique que l'on souhaite gérer une image Docker.
  name = "ubuntu:latest"                  # Spécifie le nom de l'image Docker à récupérer (pull)
}
# Create a container
resource "docker_container" "foo" {       # Indique que l'on souhaite gérer un conteneur Docker.
  image = docker_image.ubuntu.image_id    # Spécifie l'image Docker à utiliser pour créer ce conteneur
  name  = "foo"
}
```

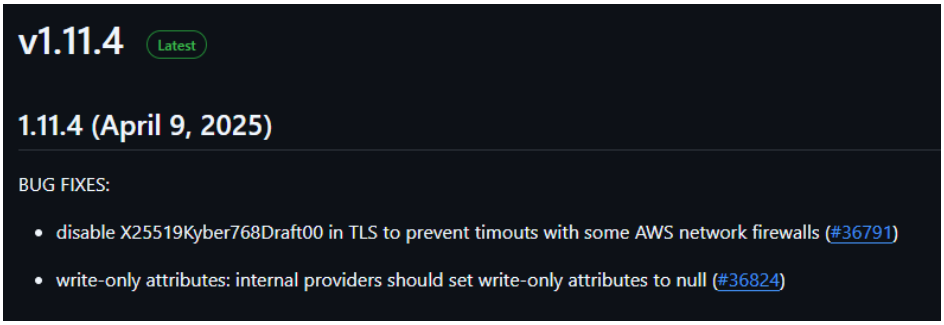
2.4. Effectuer une Veille Technologique

a. Terraform

- **Qu'est-ce que le repos GitHub de Terraform provider docker, AWS, kubernetes ?**
  - ✓ **Terraform Provider Docker :**
    Le repository principal est : **https://github.com/kreuzwerker/terraform-provider-docker**
  - ✓ **Terraform Provider AWS :**
    Le repository principal est : **https://github.com/hashicorp/terraform-provider-aws**
  - ✓ **Terraform Provider Kubernetes :**
    Le repository principal est : **https://github.com/hashicorp/terraform-provider-kubernetes**

Dans ces repositories, on trouve le code source des providers, la documentation (souvent dans le README ou un dossier docs), les exemples d'utilisation, les issues (pour signaler des bugs ou demander des fonctionnalités), et les pull requests.

- **Les dernières versions de la cli Terraform ?**
  - ✓ La liste complète des versions et les notes de publication sur la page officielle des releases de Terraform sur GitHub : https://github.com/hashicorp/terraform/releases
  - ✓ Les dernières versions de la CLI Terraform sont les suivantes (au 9 avril 2025) :
    - Dernière version stable : 1.11.4 (publiée récemment)
    - Dernière version bêta : 1.12.0-beta1 (publiée le 2 avril 2025)
    - Dernière version alpha : 1.12.0-alpha20250319 (publiée le 19 mars 2025)
- **Qu'est-ce qu'ils ont rajouté de nouveau sur la dernière version par rapport celle d'avant ?**



b. Ansible - Quelle est la dernière version?

La dernière version stable d'Ansible est Ansible 11.4.0, sortie le 25 mars 2025 :
https://pypi.org/project/ansible/11.4.0/

Il est important de noter qu'il existe également ansible-core, qui est le moteur sous-jacent d'Ansible.

La dernière version stable d'ansible-core est 2.18.4, également sortie le 25 mars 2025 :
https://github.com/ansible/ansible/releases

3. Partie 1 : Preparation de l'Environnement

3.1. Introduction

Dans le cadre de ce projet, la mise en place d'une machine sous Debian servira à la fois de poste de développement et de contrôleur pour Ansible et Terraform .

Ce poste aura une place centrale, et permettra de gérer l'infrastructure as code (IaC) et de développer dans un environnement uniforme et contrôlé.

L'objectif est de créer un système où la configuration, le déploiement et la gestion de l'infrastructure seront automatisés et réplicables, réduisant ainsi les risques d'erreurs manuelles et augmentant la productivité.

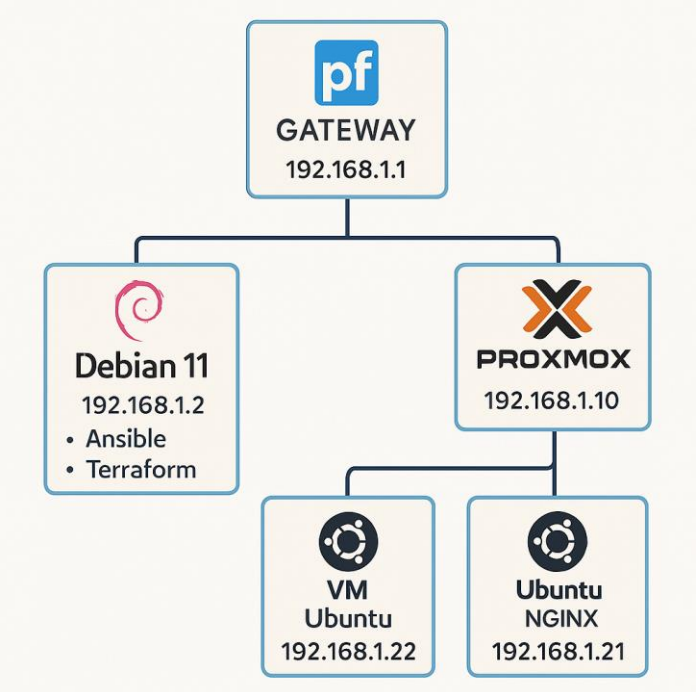
L'environnement de déploiement sera effectué sur **PROXMOX VIRTUAL ENVIRONNEMENT** qui est une solution de virtualisation libre de type "bare metal" basée sur l'hyperviseur Linux KVM.

- Le packaging de Proxmox VE est fourni sur une image ISO.
- L'installateur fourni par Proxmox comprend :

✓ Système d'exploitation complet (Debian 12 Stable 64 bits) ✓ Partitionnement de disque dur avec LVM2 ✓ Support de LXC (containers) et du module KVM (virtualisation complète)	✓ Outils de sauvegarde et de restauration ✓ Interface web d'administration et de supervision ✓ Fonctions de clustering

- Liens de téléchargement :

✓ **Debian 11** sous VMware Workstation 17 pro : debian-bullseye-DI-rc3-amd64-DVD-1.iso
✓ **Pfsense** qui servira de passerelle : pfSense-CE-2.6.0-RELEASE-amd64.iso.gz
✓ Hyperviseur KVM **PROXMOX VE** : proxmox-ve_8.4-1.iso



3.2. Installation (Pfsense/Debian11/Ansible/SSH/Terraform)

a. Installation de Pfsense

Pfsense est un OS transformant n'importe quel ordinateur en routeur/pare-feu et fera office de passerelle.

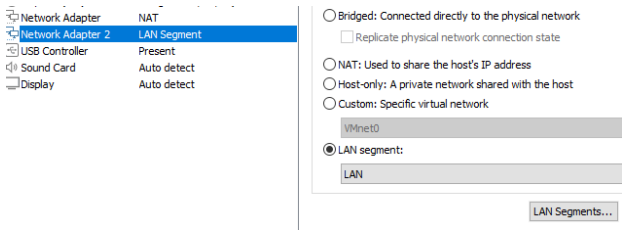
Basé sur FreeBSD, connu pour sa fiabilité et surtout sa sécurité, Pfsense est un produit OpenSource adapté à tout type d'entreprise.

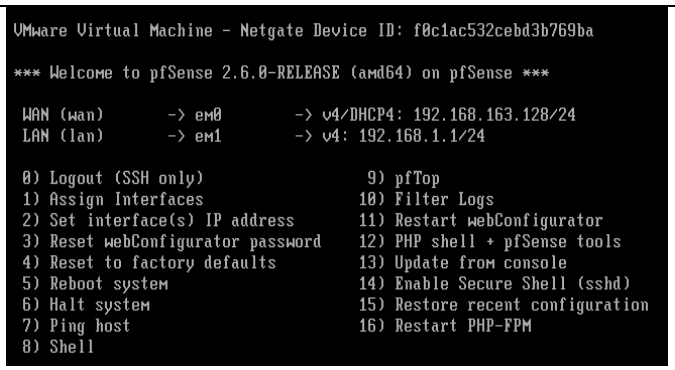
L'installation de Pfsense ne sera pas expliquée ici n'étant pas le sujet de cette évaluation.

De base son installation est très simple en suivant les écrans en faisant "suivant" et en ayant surtout à configurer l'adresse IP du réseau LAN, le WAN utilisera le DHCP VMware Workstation pour l'accès Internet.

LAN : 192.168.1.1/24 (LAN Segment créé dans VMware)

WAN : DHCP en VMnet8 (NAT)





b. Installation de Debian 11

Cette installation sur VMware Workstation a déjà été expliquée dans l'évaluation ECF1, la version utilisée sera Debian Bullseye 11.

Configuration réseau :

- ✓ IP 192.168.1.2/24
- ✓ Passerelle 192.168.1.1

Pour une gestion sécurisée, un utilisateur administrateur spécifique (fabrice) sera créé.

Cet utilisateur dispose des droits nécessaires pour effectuer des tâches d'administration système sans recourir à l'utilisation directe de l'utilisateur root, renforçant ainsi la sécurité de la machine.

Enfin, l'installation d'OpenSSH a été choisie dans les tâches de sélection pour permettre l'accès à distance.

Après avoir installé le système d'exploitation, vient l'installation d'Ansible et de Terraform qui jouent un rôle essentiel dans l'automatisation et la gestion de configurations au sein de ce projet permettant de déployer et de gérer l'infrastructure de manière efficace et reproductible.

- ✓ Pour **Ansible**, la version installée sera la version 2.15.13 d'ansible-core.
- ✓ Pour **Terraform**, la version installée sera terraform 1.11.4

c. Installation d'Ansible

```
sudo apt update
sudo apt -y upgrade
sudo apt install build-essential libssl-dev libffi-dev python3-dev python3-pip
pip3 install ansible --user
pip3 install argcomplete # l'objectif principal de cette commande est d'activer la complétion automatique pour les commandes ansible
sudo activate-global-python-argcomplete
```

```
root@debian11:/home/fabrice# ansible --version
ansible [core 2.15.13]
  config file = None
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/mod
ules']
  ansible python module location = /root/.local/lib/python3.9/site-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /home/fabrice/.local/bin/ansible
  python version = 3.9.2 (default, Mar 20 2025, 02:07:39) [GCC 10.2.1 20210110] (/usr/bin/python3)
  jinja version = 3.1.6
  libyaml = True
```

Test après installation avec la commande :

```
ansible all -i "localhost," -c local -m shell -a 'echo Salut fabrice'
```

```
root@debian11:/home/fabrice# ansible all -i "localhost," -c local -m shell -a 'echo Salut fabrice'
localhost | CHANGED | rc=0 >>
Salut fabrice
```

d. Installation de SSH

- **Installation**

SSH, ou Secure Shell, est un protocole permettant d'établir une communication chiffrée, donc sécurisée qui permet aux administrateurs d'accéder à distance à des serveurs

Ansible fonctionne sans agent et n'a pas besoin d'installer un logiciel sur les machines distantes pour les gérer, à la place il utilise SSH pour se connecter aux serveurs et exécuter des tâches.

```
sudo apt update

sudo apt install openssh-server

#démarrage du service

sudo systemctl start sshd

#activation du service
sudo systemctl enable sshd
```

```
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2025-04-10 19:33:11 CEST; 39s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 3086 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 3087 (sshd)
      Tasks: 1 (limit: 1072)
     Memory: 1.0M
        CPU: 47ms
     CGroup: /system.slice/ssh.service
             └─3087 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

avril 10 19:33:11 debian11 systemd[1]: Starting OpenBSD Secure Shell server...
avril 10 19:33:11 debian11 sshd[3087]: Server listening on 0.0.0.0 port 22.
avril 10 19:33:11 debian11 sshd[3087]: Server listening on :: port 22.
avril 10 19:33:11 debian11 systemd[1]: Started OpenBSD Secure Shell server.
```

- **Génération des clés SSH**

ssh-keygen -b -t rsa

#clé privé
[fabrice@debian11~]$ **~/.ssh/id_rsa**

#clé publique
[fabrice@debian11~]$ **~/.ssh/id_rsa.pub**

```
fabrice@debian11:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/fabrice/.ssh/id_rsa):
Created directory '/home/fabrice/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/fabrice/.ssh/id_rsa
Your public key has been saved in /home/fabrice/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:i97du3dvAf51NEvbBPxX7HgChNSjl79DNqxpEH28LwQ fabrice@debian11
The key's randomart image is:
+---[RSA 3072]----+
|         ..+.. . |
|          . + o o|
|           o = =.|
|          o E.=oB|
|        S  o.=o**|
|       . ..  .O++|
|      . .  . *.++|
|     . . . .+ =.+|
|      . . ..++ =o|
+----[SHA256]-----+
```

Les clés installées sont dans le dossier **/home/fabrice/.ssh/**

```
fabrice@debian11:~/.ssh$ ls
id_rsa  id_rsa.pub
```

- **Envoie de la paire de clé sur le serveur**

Pour pouvoir se connecter sur un serveur sans identifiant, il faut lui transmettre une clé, pour faire ça, on le fait avec la commande suivante :

```
ssh-copy-id utilisateur@ip-serveur
```

Il est désormais possible de se connecter sur la machine sans mot de passe avec la commande :

```
ssh utilisateur@ip-serveur
```

Cette opération est réalisable depuis plusieurs clients, sur le serveur elle va ajouter plusieurs lignes dans le fichier ~/.ssh/**authorized_keys** sur le serveur.

e. Installation de Terraform

1. Installation des dépendances

```
apt-get install wget curl unzip software-properties-common gnupg2 –y
```

2. Ajouter la clé GPG HashiCorp
   (GNU Privacy Guard) système de chiffrement basé sur le standard OpenPGP, pour signer cryptographiquement les produits distribués par HashiCorp.

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gp
```

3. Ajouter le dépôt Debian HashiCorp (dans le fichier **/etc/apt/sources.list**)

```
apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb_release -cs) main"

# apt-add-repository : modification fichier /etc/apt/sources.list

# arch=$(dpkg --print-architecture)] : Spécifie l'architecture du système

# $(lsb_release -cs) : insère dynamiquement le nom de code de la version Debian
```

4. Mise à jour de la liste des paquets : sudo apt update –y

```
Atteint :5 http://debian.univ-tlse2.fr/debian bullseye-updates InRelease
Réception de :6 https://apt.releases.hashicorp.com bullseye InRelease [12,9 kB]
Réception de :7 https://apt.releases.hashicorp.com bullseye/main amd64 Packages [176 kB]
Lecture des listes de paquets... Fait
```

5. Installation Terraform : sudo apt install terraform –y

```
Les NOUVEAUX paquets suivants seront installés :
  git git-man liberror-perl terraform
0 mis à jour, 4 nouvellement installés, 0 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 35,0 Mo dans les archives.
Après cette opération, 129 Mo d'espace disque supplémentaires seront utilisés.
Réception de :1 http://security.debian.org/debian-security bullseye-security/main amd64 git-man all
1:2.30.2-1+deb11u4 [1 831 kB]
Réception de :2 http://debian.univ-tlse2.fr/debian bullseye/main amd64 liberror-perl all 0.17029-1 [
31,0 kB]
Réception de :3 https://apt.releases.hashicorp.com bullseye/main amd64 terraform amd64 1.11.4-1 [27,
6 MB]
```

6. Vérification installation : terraform –v

```
root@debian11:/home/fabrice/.ssh# terraform -v
Terraform v1.11.4
on linux_amd64
```

3.3. Installation de Proxmox VE

- Accédez au site de téléchargement officiel en cliquant le lien suivant : https://www.proxmox.com/en/downloads/proxmox-virtual-environment/iso
- Dans la catégorie « **Proxmox Virtual Environment** », cliquez sur « **Download** »

- Installation en ANNEXE

a. Création de Templates Proxmox

- **Préparation du Template**

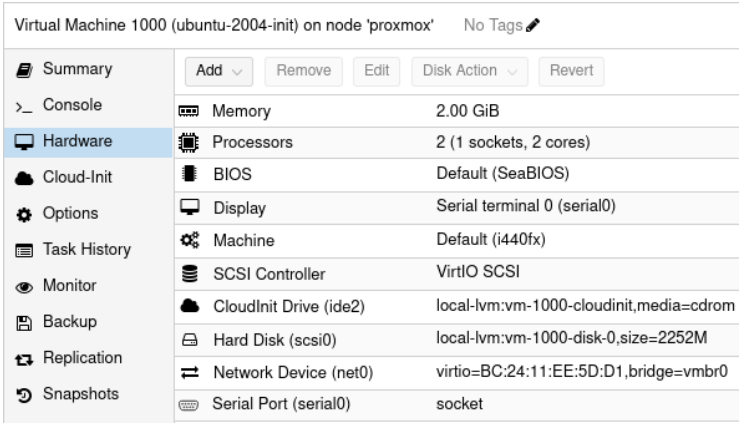
Une image cloud-init Ubuntu est une image de système d'exploitation Ubuntu préconfigurée avec le paquet cloud-init.

cloud-init est un utilitaire multi-distribution qui gère l'initialisation précoce des instances de machines virtuelles (VM) dans le cloud. Son rôle principal est d'automatiser la configuration initiale d'une instance cloud lors de son premier démarrage, sans nécessiter d'intervention manuelle.

Etapes de création en ANNEXE :

1. Télécharger une image cloud Ubuntu de base
2. Créer une VM Proxmox à l'aide de l'image
3. Ajout de l'image cloud-init à la VM
4. Attacher le disque à la VM
5. Ajouter un lecteur CD-ROM virtuel à la VM
6. Définir le démarrage de la VM sur le disque dur
7. Définir une console série pour interagir avec la VM

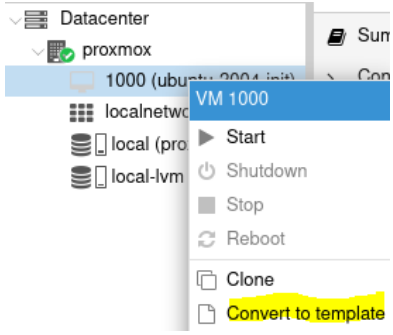
Vérification dans l'interface Proxmox que la VM est correctement configurée :

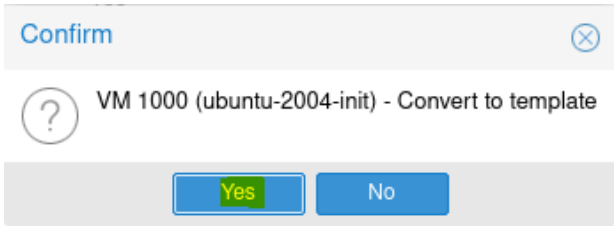


- **Configuration de Cloud-Init**

Pour configurer les futures VM avec Cloud-Init,

il faut d'abord transformer la VM en template





b. Création utilisateur pour accès à Proxmox VE

- **Utilisateur Terraform**

Pour que Terraform puisse interagir avec l'environnement Proxmox, il faut créer un utilisateur dédié et générer une Token API pour Terraform.

Etape de création en ANNEXE

1. Créer un utilisateur Proxmox dédié à Terraform	**3.** Attribuer le rôle à l'utilisateur Terraform
2. Créer un rôle Terraform	**4.** Créer une clé d'API pour l'utilisateur Terraform

- **Configurer le provider Proxmox dans Terraform**

Dans le fichier de configuration Terraform **main.tf**, on configurera le provider Proxmox en utilisant l'ID du token et le Secret :

```
terraform {
  required_providers {
    proxmox = {
      source = "Telmate/proxmox" # Provider Proxmox
      version = "3.0.1-rc8"                    # Version du provider
    }
  }
}
provider "proxmox" {
  pm_api_url       = https://192.168.1.10:8006/api2/json          # Adresse IP ou nom serveur Proxmox VE
  pm_api_token_id    = "terraform@pam!terraform-token"
  pm_api_token_secret = "<api_token_secret>"        # Secret de la clé d'API
  pm_tls_insecure    = true # À utiliser uniquement si on n'a pas de certificat TLS valide
}
```

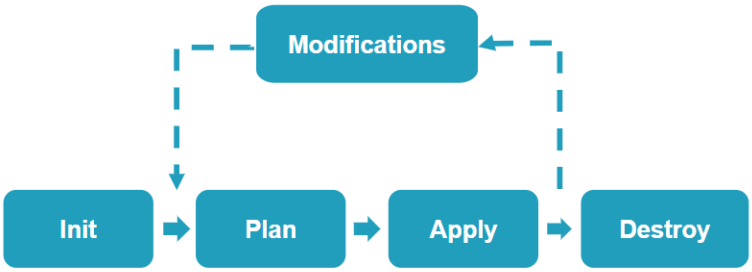
4. PARTIE 2 : DEPLOIEMENT VMS UBUNTU AVEC TERRAFORM

4.1. Etapes du plan Terraform

1. Installer Terraform (en PARTIE 1)
2. Déterminer la méthode d'authentification pour que Terraform interagisse avec Proxmox (clés API – en PARTIE 1 – 3.4.b)
3. Initialisation de base de Terraform et installation du fournisseur *
4. **Création** du plan Terraform (en ANNEXE 7.4a)
5. **Test** Plan Terraform (en ANNEXE 7.4b)
6. **Application** du plan Terraform (en ANNEXE 7.4c)

3. * Initialisation de base de Terraform et installation du fournisseur

Terraform comporte trois étapes principales : **init, planification et application**.



Les plans sont des fichiers stockés dans des répertoires.

Créez un nouveau répertoire "terraform" et créer deux fichiers : **main.tf** et **vars.tf**.

```
fabrice@debian11:~/terraform$ cd ..
fabrice@debian11:~$ rm -rf terraform/
fabrice@debian11:~$ cd ~
fabrice@debian11:~$ mkdir terraform && cd terraform
fabrice@debian11:~/terraform$ touch main.tf vars.tf
fabrice@debian11:~/terraform$ ls
main.tf  vars.tf
```

Dans **main.tf** on indique à Terraform d'utiliser le provider Proxmox de Telmate (connecteur de l'entité avec laquelle Terraform interagira) : il suffit de spécifier le nom et la version, et Terraform les récupère sur GitHub et les installe.

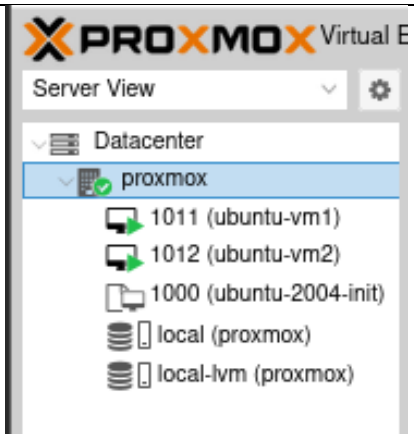
```
terraform {
  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "3.0.1-rc8"
    }
  }
}
```

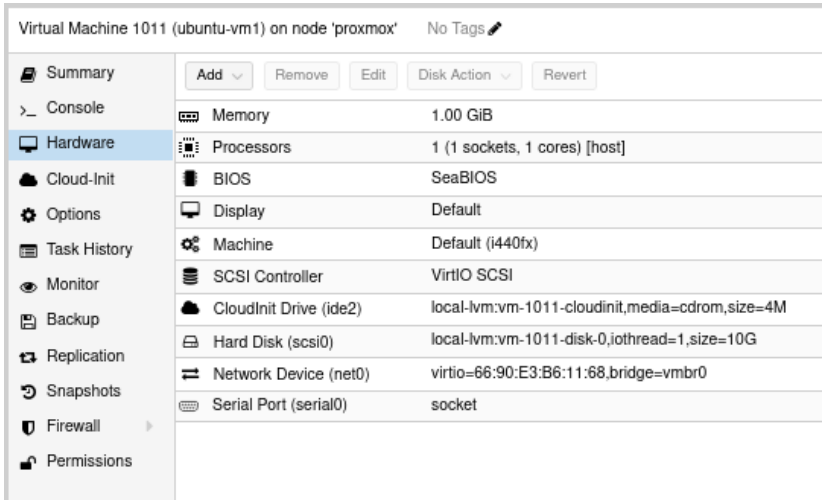
Terraform est initialisé avec le plan de base (**terraform init**), ce qui le forcera à récupérer le fournisseur.

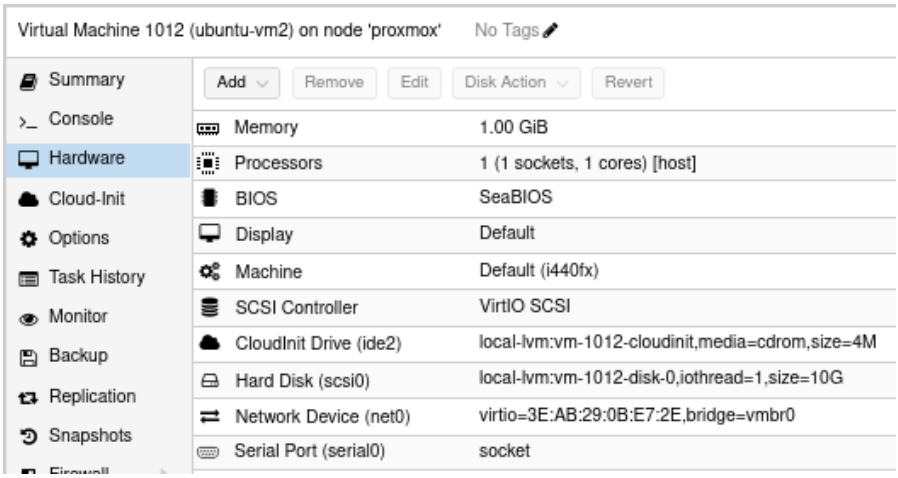
```
fabrice@debian11:~/terraform$ terraform init -upgrade
Initializing the backend...
Initializing provider plugins...
- Finding telmate/proxmox versions matching "3.0.1-rc8"...
- Installing telmate/proxmox v3.0.1-rc8...
- Installed telmate/proxmox v3.0.1-rc8 (self-signed, key ID A9EBBE091B35AFCE)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://developer.hashicorp.com/terraform/cli/plugins/signing
Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!
```

4.2. Vérification de la Création des VMs

```
proxmox_vm_qemu.ubuntu-vm[0]: Creation complete after 58s [id=proxmox/qemu/1011]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m0s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m10s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m20s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m30s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m40s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m50s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Creation complete after 1m57s [id=proxmox/qemu/1012]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Vérification création des 2 VMs ubuntu dans la console Proxmox	





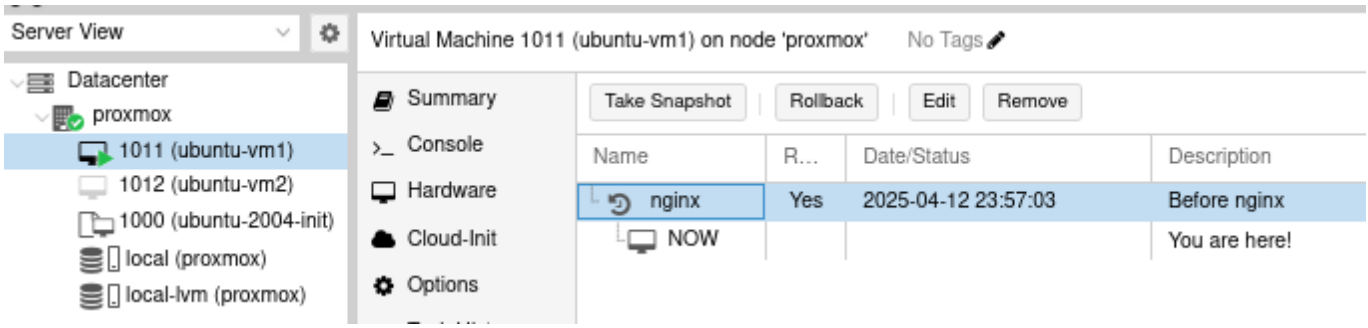
Test de connexion entre les 2 VMs Ubuntu : OK

```
root@ubuntu-vm1:/etc/netplan# ping 192.168.1.22
PING 192.168.1.22 (192.168.1.22) 56(84) bytes of data.
64 bytes from 192.168.1.22: icmp_seq=1 ttl=64 time=4.21 ms
64 bytes from 192.168.1.22: icmp_seq=2 ttl=64 time=0.832 ms
64 bytes from 192.168.1.22: icmp_seq=3 ttl=64 time=1.79 ms
64 bytes from 192.168.1.22: icmp_seq=4 ttl=64 time=1.03 ms
64 bytes from 192.168.1.22: icmp_seq=5 ttl=64 time=1.28 ms
64 bytes from 192.168.1.22: icmp_seq=6 ttl=64 time=1.60 ms
64 bytes from 192.168.1.22: icmp_seq=7 ttl=64 time=2.11 ms
64 bytes from 192.168.1.22: icmp_seq=8 ttl=64 time=0.845 ms
^C
--- 192.168.1.22 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7015ms
rtt min/avg/max/mdev = 0.832/1.711/4.211/1.037 ms
```

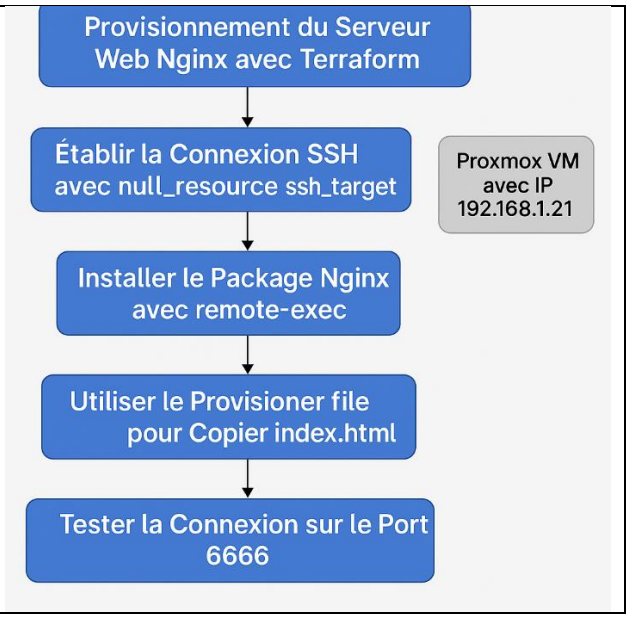
5. PARTIE 3 : DEPLOIEMENT NGINX AVEC TERRAFORM

5.1. Etapes de déploiement

1. **Snapshot** du serveur cible



2. **Mise en place de la connexion** ssh avec resource "null_resource" "ssh_target" avec les variables
3. **Installation du package Nginx** avec provisioner "remote-exec"
4. Utiliser le provisioner "file" pour **copier le fichier index.html**
5. Ensuite avec provisioner "remote-exec » copier l'**index.html** dans le bon répertoire /var/www/html et aussi la conf Nginx pour **changer le port d'écoute en 6666**
6. Enfin un **test de connexion sur le port 6666** avec un curl sur l'adresse IP 192.168.1.21 du serveur pour obtenir un résultat.

Les fichiers suivants seront utilisés : terraform/ ├── **main.tf** ├── **variables.tf** ├── **index.html** └── **nginx_custom.conf**	ANNEXE – Plan Terraform 1. Création du plan Terraform 2. Test du plan Terraform 3. Application du plan Terraform	

5.2. Vérification accès au serveur Nginx (page d'acceuil)

- FIN du **Terraform Plan** -> SUCCES

```
null_resource.ssh_target: Still creating... [1m50s elapsed]
null_resource.ssh_target: Provisioning with 'file'...
null_resource.ssh_target: Still creating... [2m0s elapsed]
null_resource.ssh_target: Provisioning with 'file'...
null_resource.ssh_target: Provisioning with 'remote-exec'...
null_resource.ssh_target (remote-exec): Connecting to remote host via SSH...
null_resource.ssh_target (remote-exec):   Host: 192.168.1.21
null_resource.ssh_target (remote-exec):   User: ubuntu
null_resource.ssh_target (remote-exec):   Password: false
null_resource.ssh_target (remote-exec):   Private key: true
null_resource.ssh_target (remote-exec):   Certificate: false
null_resource.ssh_target (remote-exec):   SSH Agent: true
null_resource.ssh_target (remote-exec):   Checking Host Key: false
null_resource.ssh_target (remote-exec):   Target Platform: unix
null_resource.ssh_target (remote-exec): Connected!
null_resource.ssh_target: Provisioning with 'remote-exec'...
null_resource.ssh_target (remote-exec): Connecting to remote host via SSH...
null_resource.ssh_target (remote-exec):   Host: 192.168.1.21
null_resource.ssh_target (remote-exec):   User: ubuntu
null_resource.ssh_target (remote-exec):   Password: false
null_resource.ssh_target (remote-exec):   Private key: true
null_resource.ssh_target (remote-exec):   Certificate: false
null_resource.ssh_target (remote-exec):   SSH Agent: true
null_resource.ssh_target (remote-exec):   Checking Host Key: false
null_resource.ssh_target (remote-exec):   Target Platform: unix
null_resource.ssh_target (remote-exec): Connected!
null_resource.ssh_target (remote-exec):     <h1>Bienvenue à la formation Terraform</h1>
null_resource.ssh_target: Creation complete after 2m8s [id=7599069795779604095]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

- Vérification **status** service **nginx** sur VM ubuntu-vm1 -> RUNNING

```
root@proxmox:~# qm terminal 1011
starting serial terminal on interface serial0 (press Ctrl+O to exit)

ubuntu@ubuntu-vm1:~$ systemctl nginx status
Unknown operation nginx.
ubuntu@ubuntu-vm1:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset:
     Active: active (running) since Sun 2025-04-13 10:57:28 UTC; 2h 25min ago
       Docs: man:nginx(8)
    Process: 5965 ExecReload=/usr/sbin/nginx -g daemon on; master_process on; -
   Main PID: 5624 (nginx)
      Tasks: 2 (limit: 1117)
     Memory: 6.8M
     CGroup: /system.slice/nginx.service
             ├─5624 nginx: master process /usr/sbin/nginx -g daemon on; master_
             └─5966 nginx: worker process
```

- Accès à la page **index.html** sur serveur nginx 192.168.1.21 (port 6666) -> OK

6. Partie 4 : ANSIBLE – Creation Utilisateur et test SSH

6.1. Création des playbook Ansible

1. **Création d'un utilisateur admin** sur chaque serveur Ubuntu hébergé sur Proxmox.
2. **Générer une paire de clés SSH** sur chaque serveur
3. **Échanger la clé publique** avec l'autre serveur pour permettre une connexion SSH
4. Effectuer un **test de connexion SSH** entre chaque serveur

a. Étape 1 : Configuration de l'inventaire Ansible (fichier hosts)

```
[ubuntu_servers]        # Déclaration des serveurs
server1 ansible_host=192.168.1.21
server2 ansible_host=192.168.1.22

[ubuntu_servers:vars]
ansible_user=ubuntu    # Utilisateur cible pour la connexion SSH
ansible_ssh_private_key_file=~/.ssh/id_rsa      # Clé privée utilisée pour la connexion (utilisateur courant)
```

b. Étape 2 : Création des Playbook Ansible (fichiersYML)

Dans un premier temps le module "**check**" sera utilisé pour simuler l'exécution du playbook sans apporter de réelles modifications et après vérification et correction si besoin, sera relancé sans cet argument.

1. **Création** "**admin**" sur les 2 serveurs (ANNEXE 7.6a) : **create_admin_user.yml**

   ansible-playbook -i hosts create_admin_user.yml --check

2. **Créer et Échanger les clés SSH** entre les 2 serveurs (ANNEXE 7.6b) : **exchange_ssh_keys.yml**

   ansible-playbook -i hosts exchange_ssh_keys.yml --check

3. **Test de connexion SSH** entre les 2 serveurs (ANNEXE 7.6c) : **test_ssh.yml**

   ansible-playbook -i hosts test_ssh.yml –check

   Pour ce playbook la variable ansible_user sera modifiée de ubuntu -> admin

```
[ubuntu_servers:vars]
ansible_user=admin          # Utilisateur cible pour la connexion SSH
ansible_ssh_private_key_file=~/.ssh/id_rsa      # Clé privée utilisée pour la connexion (utilisateur courant)
```

Pour effectuer une gestion des erreurs la commande suivante pourra être exécutée si nécessaire:

ansible-playbook playbook.yaml –v

6.2. Exécution des Playbook Ansible

1. Créer l'utilisateur admin sur les 2 serveurs

```
fabrice@debian11:~/ansible/ansible_ssh$ ansible-playbook -i hosts create_admin_user.yml

PLAY [Création de l'utilisateur admin sur les serveurs] *********************************

TASK [Gathering Facts] ******************************************************************
ok: [server2]
ok: [server1]

TASK [Création de l'utilisateur admin] **************************************************
changed: [server1]
changed: [server2]

TASK [Création du dossier .ssh] *********************************************************
changed: [server1]
changed: [server2]

PLAY RECAP ******************************************************************************
server1                    : ok=3    changed=2    unreachable=0    failed=0    skipped=0
server2                    : ok=3    changed=2    unreachable=0    failed=0    skipped=0
```

2. Créer et Échanger les clés SSH

```
fabrice@debian11:~/ansible/ansible_ssh$ ansible-playbook -i hosts exchange_ssh_keys.yml

PLAY [Échange des clés SSH entre les 2 serveurs ubuntu] *********************************

TASK [Générer la paire de clés SSH pour l'utilisateur admin] ****************************
changed: [server1]
changed: [server2]

TASK [Ajouter la clé publique de server1 sur server2] ***********************************
skipping: [server1]
changed: [server2]

TASK [Ajouter la clé publique de server2 sur server1] ***********************************
skipping: [server2]
changed: [server1]

PLAY RECAP ******************************************************************************
server1                    : ok=2    changed=2    unreachable=0    failed=0    skipped=1
server2                    : ok=2    changed=2    unreachable=0    failed=0    skipped=1
```

3. Test de connexion SSH

```
fabrice@debian11:~/ansible/ansible_ssh$ ansible-playbook -i hosts test_ssh.yml

PLAY [Test de connexion SSH entre les serveurs] *****************************

TASK [Gathering Facts] ******************************************************
ok: [server2]
ok: [server1]

TASK [Test de connexion SSH entre chaque serveur] ***************************
changed: [server2]
changed: [server1]

TASK [Affiche le résultat] **************************************************
ok: [server1] => {
    "msg": "  ✅ Connexion SSH réussie entre server1 et 192.168.1.22 !\n"
}
ok: [server2] => {
    "msg": "  ✅ Connexion SSH réussie entre server2 et 192.168.1.21 !\n"
}

PLAY RECAP ******************************************************************
server1                    : ok=3    changed=1    unreachable=0    failed=0    skipped=0
server2                    : ok=3    changed=1    unreachable=0    failed=0    skipped=0
```

6.3. Vérification sur les VMs ubuntu-vm1 et 2

1. **Créer l'utilisateur admin** sur les 2 serveurs (vérification présence utilisateur **admin**)

- **ubuntu-vm1 (vérification présence utilisateur admin)**

```
root@ubuntu-vm1:/home# id admin
uid=1001(admin) gid=100(users) groups=100(users),117(admin)
```

- **ubuntu-vm2**

```
root@ubuntu-vm2:/# id admin
uid=1001(admin) gid=100(users) groups=100(users),117(admin)
```

2. **Créer et Échanger les clés SSH** (vérification **clés privé/publique** + **autorisées**)

- **ubuntu-vm1**

```
root@ubuntu-vm1:/home/admin/.ssh# ls -l
total 12
-rw------- 1 admin users  726 Apr 13 18:29 authorized_keys
-rw------- 1 admin admin 3357 Apr 13 18:28 id_rsa
-rw------- 1 admin admin  726 Apr 13 18:28 id_rsa.pub
```

- **ubuntu-vm2**

```
root@ubuntu-vm2:/home/admin/.ssh# ls -al
total 20
drwx------ 2 admin users 4096 Apr 13 18:23 .
drwxr-xr-x 3 admin users 4096 Apr 13 18:03 ..
-rw------- 1 admin users  726 Apr 13 18:21 authorized_keys
-rw------- 1 admin admin 3357 Apr 13 18:21 id_rsa
-rw------- 1 admin admin  726 Apr 13 18:21 id_rsa.pub
```

3. **Test de connexion SSH**

- **ubuntu-vm1 -> ubuntu-vm2** (192.168.1.22)

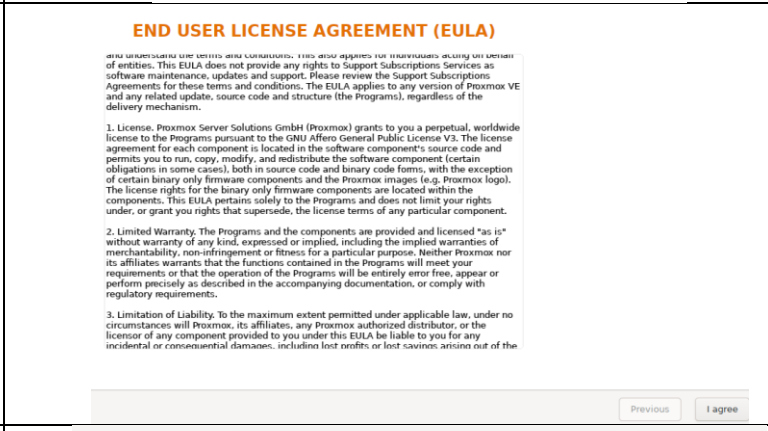
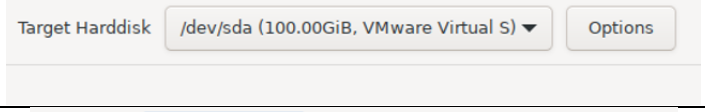
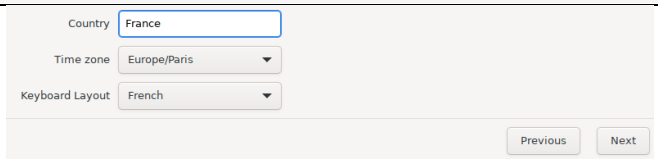
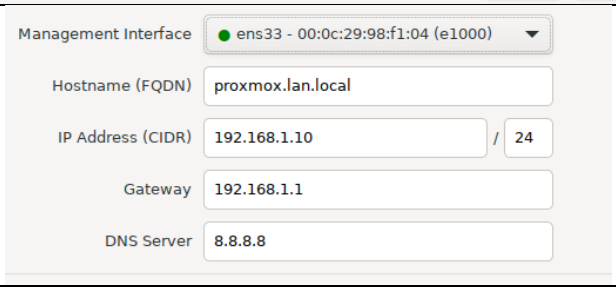
```
admin@ubuntu-vm1:~$ ssh admin@192.168.1.22
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-212-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon Apr 14 09:50:45 UTC 2025

  System load: 0.0               Memory usage: 19%   Processes:       103
  Usage of /:  17.4% of 9.51GB   Swap usage:   0%    Users logged in: 0

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Mon Apr 14 09:49:37 2025 from 192.168.1.2
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin@ubuntu-vm2:~$ hostname
ubuntu-vm2
```

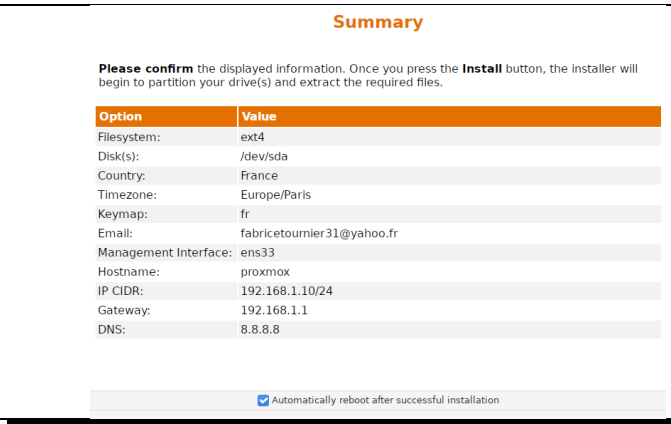
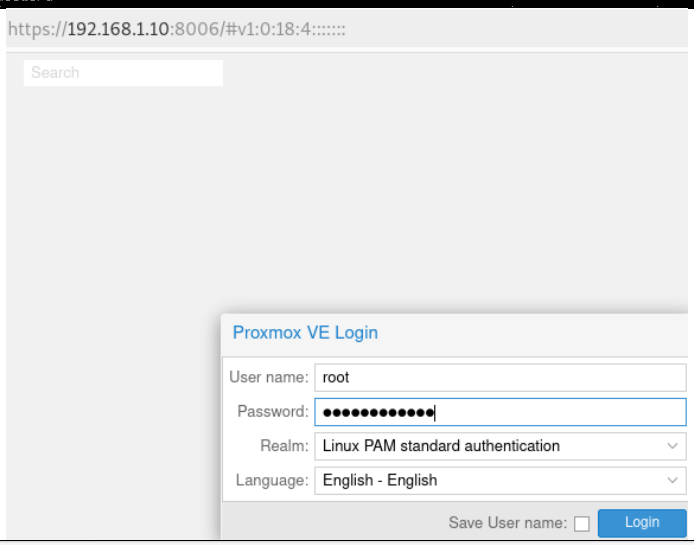
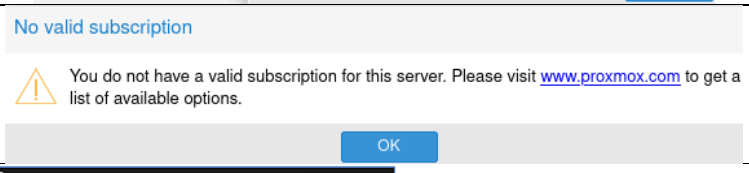
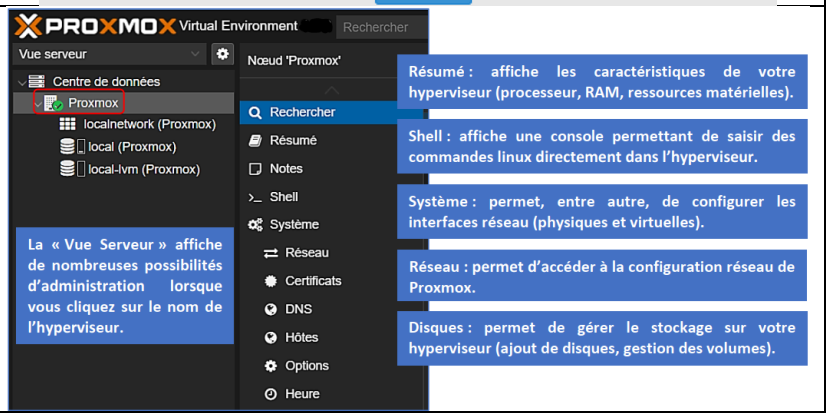
- **ubuntu-vm2 -> ubuntu-vm1** (192.168.1.21)

```
admin@ubuntu-vm2:~$ ssh admin@192.168.1.21
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-212-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon Apr 14 09:49:41 UTC 2025

  System load: 0.08              Memory usage: 20%   Processes:       109
  Usage of /:  17.4% of 9.51GB   Swap usage:   0%    Users logged in: 0

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Mon Apr 14 09:45:59 2025 from 192.168.1.2
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin@ubuntu-vm1:~$ hostname
ubuntu-vm1
```

7. Annexe

7.1. Installation Proxmox VE

Par défaut le mode « **Graphical** » est sélectionné.	
Contrat de **licence**	
Sélectionnez le **disque** sur lequel l'hyperviseur Proxmox installera son système	
Dans la rubrique « **Country** », saisir « **France** » et cliquer sur « **Next** »	
Configuration réseau	

Résumé avant Installation	
L'écran de la machine serveur affiche	
Lancer un navigateur et saisir l'adresse IP du serveur avec le port « 8006 » : https://192.168.1.10:8006	
Etant donné qu'on est sans support payant, Proxmox affiche une alerte. Cliquer sur « **OK** » pour passer outre	
L'interface d'accueil de Proxmox s'affiche : Vue **« SERVEUR »**	

✓ « **ens33** » correspond à **l'interface réseau physique**.
Il s'agit, ici, de la **carte réseau physique** sur laquelle on est connecté pour accéder au serveur Proxmox.
✓ « **vmbr0** » est une interface qui a été automatiquement configurée lors de l'installation de Proxmox et correspond à une **interface réseau virtuelle sur laquelle on connectera les futures machines virtuelles**.
- ✓ On constate que l'interface réseau virtuelle « **vmbr0** » est bien connectée à la carte physique du serveur « **ens33** » puisque le « Ports/Esclaves » est bien « **ens33** » ici.
- ✓ Cette interface réseau virtuelle « vmbr0 » est connectée en « mode pont » sur l'interface physique du serveur ce qui fait que les machines virtuelles auront un accès à internet via la passerelle (**Pfsense**).
- ✓ Il est possible de créer de nouvelles interfaces réseau virtuelles en cliquant sur le bouton « Créer »

Name	Type	Active	Autostart	VLAN a…	Ports/Slaves	Bo…	CIDR	Gateway
ens33	Network Device	Yes	No	No				
vmbr0	Linux Bridge	Yes	Yes	No	ens33		192.168.1.10/24	192.168.1.1

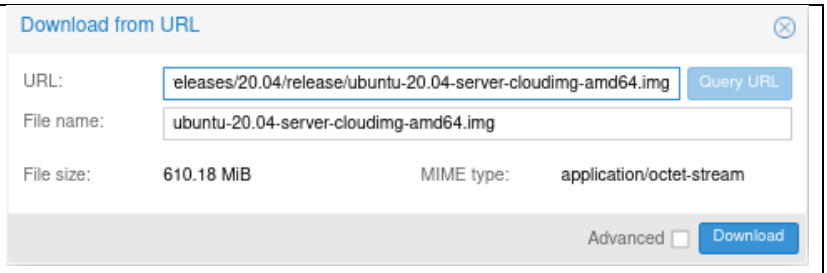
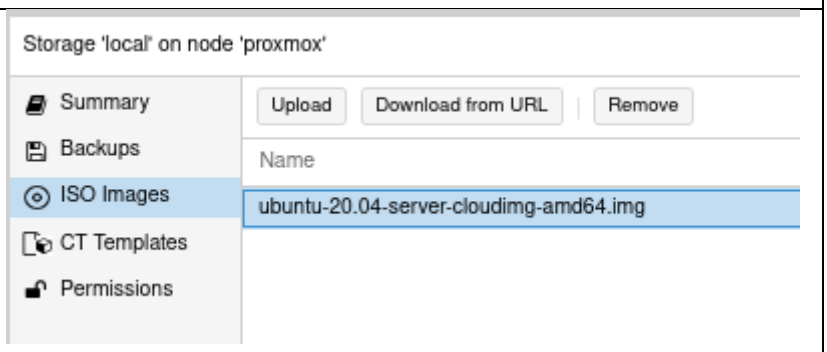
7.2. Création Templates Proxmox VE

1. Télécharger une image cloud Ubuntu de base

Ubuntu fournit des images de base régulièrement mises à jour : https://cloud-images.ubuntu.com/ .

- La version téléchargée sera Ubuntu 20.04 :

https://cloud-images.ubuntu.com/releases/20.04/release/ubuntu-20.04-server-cloudimg-amd64.img

Download de l'image dans Proxmox	
Image stockée dans le stockage « local » /var/lib/vz/template/iso	

- Modification de l'image

  Installation des outils : sudo apt update -y && sudo apt install libguestfs-tools -y

  La commande "**virt-customize**" est utilisée pour une personnalisation de l'image.

sudo virt-customize -a ubuntu-20.04-server-cloudimg-amd64.img --install qemu-guest-agent

sudo virt-customize -a ubuntu-20.04-server-cloudimg-amd64.img --root-password password:fabrice

qemu-guest-agent est un petit programme qui s'exécute à l'intérieur d'une machine virtuelle et permet à l'hôte Proxmox de communiquer avec le système d'exploitation invité.

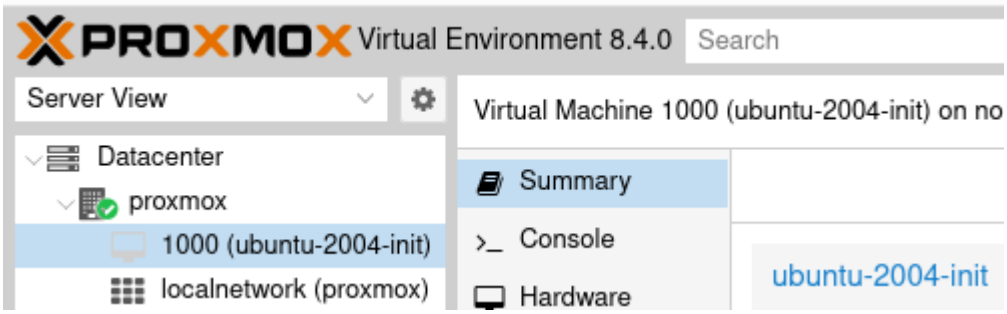
Bien que cette commande soit pratique pour automatiser la configuration initiale, il est **fortement déconseillé** d'utiliser des mots de passe en clair directement dans les scripts ou les commandes, surtout pour des environnements de production, ce cas se fera uniquement dans le cadre de cette évaluation.

2. **Création d'une VM**

   sudo qm create 1000 --name ubuntu-2004-init --memory 1024 --core 1 --net0 virtio,bridge=vmbr0

   Cette commande créé une **VM avec l'ID 1000**, 1 Go de RAM, 1 cœurs, le nom "**ubuntu-2004-init**", et une interface réseau. (Ces valeurs seront modifiées lors de la création de VMs via Terraform)

   Après avoir exécuté cette commande, une nouvelle VM apparait dans l'interface Proxmox.

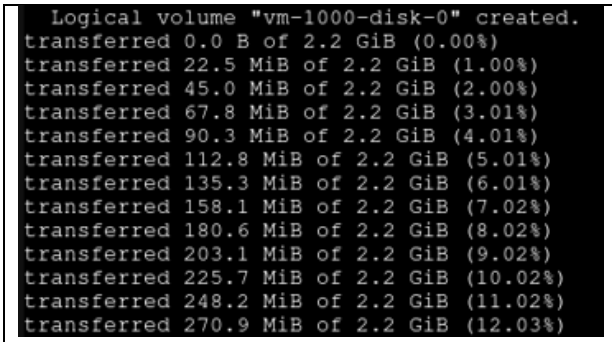


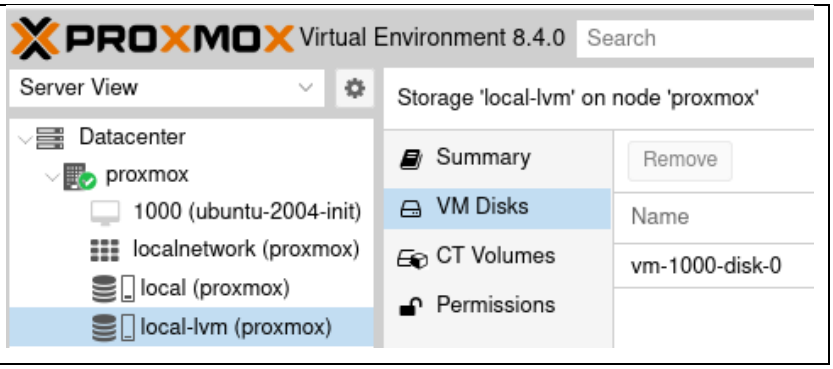
3. **Ajout de l'image cloud-init à la VM**

   sudo qm importdisk 1000 ubuntu-20.04-server-cloudimg-amd64.img local-lvm

   Cette commande importe le disque dur dans la VM 1000, en utilisant le stockage "**local-lvm**".

   Après l'exécution, on obtient le retour suivant :





4. **Attacher le disque à la VM**

   sudo qm set 1000 --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-1000-disk-0

```
root@proxmox:/var/lib/vz/template/iso# sudo qm set 1000 --scsihw virtio-sc
sk-0
update VM 1000: -scsi0 local-lvm:vm-1000-disk-0 -scsihw virtio-scsi-pci
```

5. **Ajout lecteur CD-ROM virtuel à la VM**

   Ajouter un lecteur CD-ROM virtuel à la VM pour que Proxmox puisse injecter la **configuration Cloud-Init**

   qm set 1000 --ide2 local-lvm:cloudinit

```
root@proxmox:/var/lib/vz/template/iso# sudo qm set 1000 --ide2 local-lvm:cloudinit
update VM 1000: -ide2 local-lvm:cloudinit
  Logical volume "vm-1000-cloudinit" created.
ide2: successfully created disk 'local-lvm:vm-1000-cloudinit,media=cdrom'
generating cloud-init ISO
```

6. **Définir le démarrage de la VM sur le disque dur**

   qm set 1000 --boot c --bootdisk scsi0

```
root@proxmox:/var/lib/vz/template/iso# sudo qm set 1000 --boot c --bootdisk scsi0
update VM 1000: -boot c -bootdisk scsi0
```

7. **Définir une console série pour interagir avec la VM**

   qm set 1000 --serial0 socket --vga serial0

```
root@proxmox:/var/lib/vz/template/iso# sudo qm set 1000 --serial0 socket --vga serial0
update VM 1000: -serial0 socket -vga serial0
```

8. **Activation de l'agent invité QEMU pour la machine virtuelle :** sudo qm set 1000 --agent enabled=1
9. **Création du modèle qui sera utilisé par Terraform :** sudo qm template 1000

***** Rappel des commandes de génération du Modèle de VM sur Proxmox*****

```
sudo virt-customize -a ubuntu-20.04-server-cloudimg-amd64.img --install qemu-guest-agent
sudo virt-customize -a ubuntu-20.04-server-cloudimg-amd64.img --root-password password:fabrice
sudo qm create 1000 --name "ubuntu-2004-init" --memory 1024 --cores 1 --net0 virtio,bridge=vmbr0
sudo qm importdisk 1000 ubuntu-20.04-server-cloudimg-amd64.img local-lvm
sudo qm set 1000 --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-1000-disk-0
sudo qm set 1000 --boot c --bootdisk scsi0
sudo qm set 1000 --ide2 local-lvm:cloudinit
sudo qm set 1000 --serial0 socket --vga serial0
sudo qm set 1000 --agent enabled=1
sudo qm template 1000
```

7.3. Création utilisateur pour accès à Proxmox VE

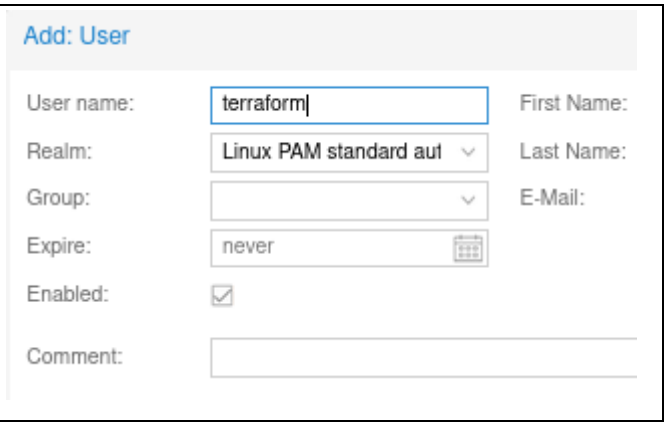
1. Créer un utilisateur Proxmox dédié à Terraform

Naviguez vers "**Datacenter**"

- ✓ Cliquer sur l'onglet "**Permissions**".
- ✓ Cliquer sur le bouton "**Users**"
- ✓ Cliquer sur le bouton "**Add**"

Remplir les informations requises :

- ✓ **User name :** terraform
- ✓ **Realm :** Linux PAM



2. Créer un Rôle Terraform

Sélectionnez les privilèges nécessaires à Terraform pour gérer les ressources (VMs, LXCs, stockage, etc…)

TerraformRole	Datastore.Allocate Datastore.AllocateSpace Pool.Allocate Pool.Audit Sys.Audit VM.Allocate VM.Clone VM.Config.CDROM VM.Config.CPU VM.Config.Cloudinit VM.Config.Disk VM.Config.HWType VM.Config.Memory VM.Config.Network VM.Config.Options VM.Console VM.Monitor VM.PowerMgmt

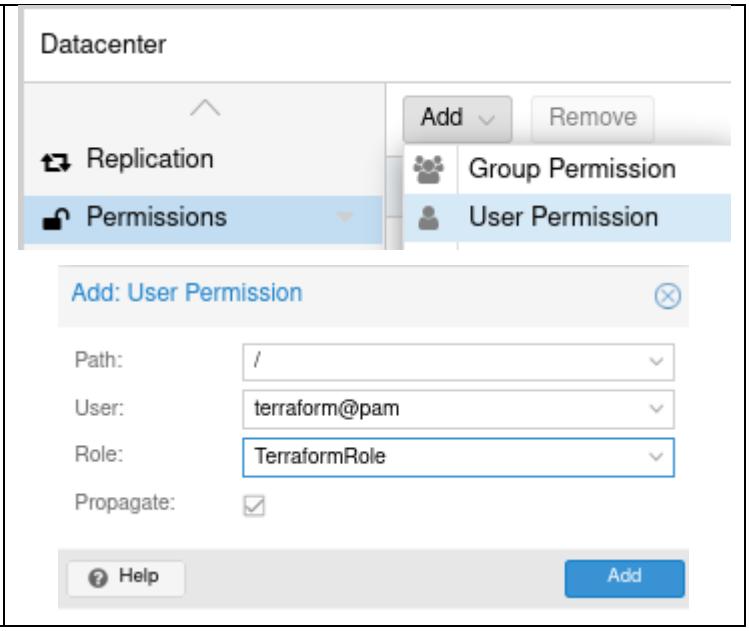
3. Attribuer le rôle à l'utilisateur Terraform

Naviguez vers "**Datacenter**"

- ✓ Cliquer sur "**Permissions**".

Dans la fenêtre "**User Permission**" :

- ✓ **Path**: Laissez **/** pour appliquer les permissions au niveau du Datacenter
- ✓ **User/Group**: Sélectionnez l'utilisateur Terraform
- ✓ **Role**: Sélectionnez le rôle **TerraformRole**
- ✓ Cliquez sur "**Add**".



4. Créer une clé d'API pour l'utilisateur Terraform

L'utilisation d'une clé API est la méthode d'authentification recommandée pour Terraform plutôt que d'utiliser un simple login-mot de passe.

Naviguez vers "**Datacenter**"

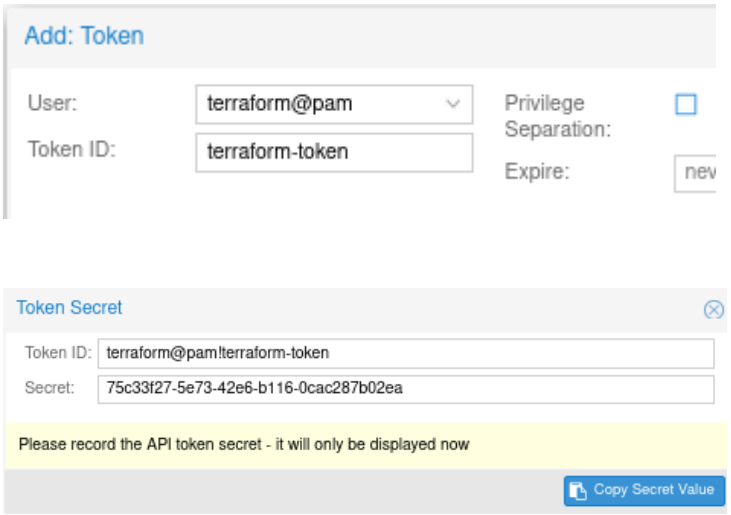
- ✓ Cliquer sur "**Permissions**".
- ✓ Cliquez sur l'onglet "**API Tokens**".
- ✓ Cliquez sur le bouton "**Add**".

Sélectionnez l'utilisateur Terraform

- ✓ **Token ID**: Donnez un nom au token, par exemple terraform-token
- ✓ **Privilege separation**: Laisser cette case décochée si un rôle a déjà été attribué à l'utilisateur.
- ✓ Cliquez sur "**Add**".

Une fenêtre affichera l'ID et le Secret du nouveau token.

Conserver précieusement le Secret, car il ne sera affiché qu'une seule fois.



7.4. Plan Terraform "Création VMs Ubuntu"

a. Création du plan Terraform (maint.tf + vars.tf)

```
terraform {
  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "3.0.1-rc8"
    }
  }
}


provider "proxmox" {

  pm_api_url = "https://192.168.1.10:8006/api2/json"

  pm_api_token_id = "terraform@pam!terraform-token"
  pm_api_token_secret = "66091622-5711-4c42-819b-3c5ab85bd133"
  pm_tls_insecure = true
}

resource "proxmox_vm_qemu" "ubuntu-vm" {
  count = 2
  name  = "ubuntu-vm${count.index + 1}"
  target_node = var.proxmox_host
  vmid = "101${count.index + 1}"
  clone = var.template_name
  agent = 1
  os_type = "cloud-init"
  cores = 1
  sockets = 1
  memory  = 1024
  scsihw  = "virtio-scsi-pci"
  bootdisk = "scsi0"

  disk {
    slot = 0
    size = "10G"
    type = "scsi"
    storage = "local-lvm"
    iothread = 1
  }

    storage = "local-lvm"
    iothread = 1
  }

  network {
    model = "virtio"
    bridge = "vmbr0"
    firewall = false
    link_down = false
  }

  ipconfig0 = "ip=192.168.1.2${count.index + 1}/24,gw=192.168.1.1"
  sshkeys = <<EOF
  ${var.ssh_key}
  EOF
}
```

```
variable "ssh_key" {
  default = "ssh-rsa AAAAB3NzaC1y
ZTwqJ01cPILI7ghGeimJ0Tpj0mKWA8mZk
+OnmnHuexrniwwwS2vX/FDQMRp6bl9MOD
pWz2HtP50V+hoOCuDZYIakTx1SCMBIIAD
teO3b0EPQDb+Ll7p+7AB/-
S+xeK4hxGYHZxrWymQhAzcrV245iyD3Wm
2heQdWkPuHzAEwCqRAnN6NCcFaEZXRCYc
}

variable "proxmox_host" {
    default = "proxmox"
}
variable "template_name" {
    default = "ubuntu-2004-init"
}
```

b. Test du plan Terraform (terraform plan)

```
proxmox_vm_qemu.ubuntu-vm[0]: Refreshing state... [id=proxmox/qemu/1001]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  +create

Terraform will perform the following actions:

  # proxmox_vm_qemu.ubuntu-vm will be created
  # moved from proxmox_vm_qemu.ubuntu-vm)
  + resource "proxmox_vm_qemu" "ubuntu-vm" {
      + additional_wait            = 5
      + agent                      = 1
      + automatic_reboot           = true
      + balloon                    = 0
      + bios                       = "seabios"
      + boot                       = (known after apply)
      + bootdisk                   = "scsi0"
      + clone                      = "ubuntu-2004-init"
      + clone_wait                 = 10
      + cores                      = 1
      + cpu                        = "host"
      + default_ipv4_address       = (known after apply)
      + define_connection_info     = true
      + force_create               = false
      + full_clone                 = true
      + guest_agent_ready_timeout = 100
      + hotplug                    = "network,disk,usb"
      + id                         = (known after apply)
      + ipconfig0                  = "ip=192.168.1.21/24,gw=192.168.1.1"
      + kvm                        = true
      + linked_vmid                = (known after apply)
      + memory                     = 1024
      + name                       = "ubuntu-vm-1"
      + nameserver                 = (known after apply)
      + onboot                     = false
      + oncreate                   = false
      + os_type                    = "cloud-init"
      + preprovision               = true
      + reboot_required            = (known after apply)
      + scsihw                     = "virtio-scsi-pci"
      + searchdomain               = (known after apply)
      + sockets                    = 1
```

```
+ sockets                    = 1
+ ssh_host                   = (known after apply)
+ ssh_port                   = (known after apply)
+ sshkeys                    = <<-EOT
      ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC8y8saIIHts4/ER4BZ
  EOT
+ tablet                     = true
+ tags                       = (known after apply)
+ target_node                = "proxmox"
+ unused_disk                = (known after apply)
+ vcpus                      = 0
+ vlan                       = -1
+ vm_state                   = "running"
+ vmid                       = (known after apply)

+ disks {
    + virtio {
        + virtio0 {
            + disk {
                + backup               = true
                + cache                = "writeback"
                + format               = "raw"
                + id                   = (known after apply)
                + iops_r_burst         = 0
                + iops_r_burst_length  = 0
                + iops_r_concurrent    = 0
                + iops_wr_burst        = 0
                + iops_wr_burst_length = 0
                + iops_wr_concurrent   = 0
                + linked_disk_id       = (known after apply)
                + mbps_r_burst         = 0
                + mbps_r_concurrent    = 0
                + mbps_wr_burst        = 0
                + mbps_wr_concurrent   = 0
                + size                 = 10
                + storage              = "local-lvm"
              }
          }
      }
  }
```

```
      + network {
          + bridge    = "vmbr0"
          + firewall  = false
          + link_down = false
          + macaddr   = (known after apply)
          + model     = "virtio"
          + queues    = (known after apply)
          + rate      = (known after apply)
          + tag       = -1
        }

      + smbios (known after apply)
    }

  # proxmox_vm_qemu.ubuntu-vm will be created
+ resource "proxmox_vm_qemu" "ubuntu-vm" {
      + additional_wait           = 5
      + agent                     = 1
      + automatic_reboot          = true
      + balloon                   = 0
      + bios                      = "seabios"
      + boot                      = (known after apply)
      + bootdisk                  = "scsi0"
      + clone                     = "ubuntu-2004-init"
      + clone_wait                = 10
      + cores                     = 1
      + cpu                       = "host"
      + default_ipv4_address      = (known after apply)
      + define_connection_info    = true
      + force_create              = false
      + full_clone                = true
      + guest_agent_ready_timeout = 100
      + hotplug                   = "network,disk,usb"
      + id                        = (known after apply)
      + ipconfig0                 = "ip=192.168.1.22/24,gw=192.168.1.1"
      + kvm                       = true
      + linked_vmid               = (known after apply)
      + memory                    = 1024
      + name                      = "ubuntu-vm-2"
      + nameserver                = (known after apply)
      + onboot                    = false
```

```
+ oncreate                    = false
+ os_type                     = "cloud-init"
+ preprovision                = true
+ reboot_required             = (known after apply)
+ scsihw                      = "virtio-scsi-pci"
+ searchdomain                = (known after apply)
+ sockets                     = 1
+ ssh_host                    = (known after apply)
+ ssh_port                    = (known after apply)
+ sshkeys                     = <<-EOT
      ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC8y8saIIHts4/ER4BZXBY
  EOT
+ tablet                      = true
+ tags                        = (known after apply)
+ target_node                 = "proxmox"
+ unused_disk                 = (known after apply)
+ vcpus                       = 0
+ vlan                        = -1
+ vm_state                    = "running"
+ vmid                        = (known after apply)

+ disks {
    + virtio {
        + virtio0 {
            + disk {
                + backup               = true
                + cache                = "writeback"
                + format               = "raw"
                + id                   = (known after apply)
                + iops_r_burst         = 0
                + iops_r_burst_length  = 0
                + iops_r_concurrent    = 0
                + iops_wr_burst        = 0
                + iops_wr_burst_length = 0
                + iops_wr_concurrent   = 0
                + linked_disk_id       = (known after apply)
                + mbps_r_burst         = 0
                + mbps_r_concurrent    = 0
                + mbps_wr_burst        = 0
                + mbps_wr_concurrent   = 0
                + size                 = 10
                + storage              = "local-lvm"
```

```
                    }
                }
            }
        }

      + network {
          + bridge    = "vmbr0"
          + firewall  = false
          + link_down = false
          + macaddr   = (known after apply)
          + model     = "virtio"
          + queues    = (known after apply)
          + rate      = (known after apply)
          + tag       = -1
        }

      + smbios (known after apply)
    }

Plan: 2 to add, 0 to change, 0 to destroy.
```

```
Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
```

c. Application du plan Terraform (terraform apply)

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

proxmox_vm_qemu.ubuntu-vm[0]: Creating...
proxmox_vm_qemu.ubuntu-vm[1]: Creating...
proxmox_vm_qemu.ubuntu-vm[0]: Still creating... [10s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [10s elapsed]
proxmox_vm_qemu.ubuntu-vm[0]: Still creating... [20s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [20s elapsed]
proxmox_vm_qemu.ubuntu-vm[0]: Still creating... [30s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [30s elapsed]
proxmox_vm_qemu.ubuntu-vm[0]: Still creating... [40s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [40s elapsed]
proxmox_vm_qemu.ubuntu-vm[0]: Still creating... [50s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [50s elapsed]
proxmox_vm_qemu.ubuntu-vm[0]: Creation complete after 58s [id=proxmox/qemu/100]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m0s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m10s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m20s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m30s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m40s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Still creating... [1m50s elapsed]
proxmox_vm_qemu.ubuntu-vm[1]: Creation complete after 1m57s [id=proxmox/qemu/101]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
fabrice@debian11:~/terraform$
```

7.5. Plan Terraform "Création Serveur Web Nginx"

a. Création du plan Terraform pour Nginx

Pour rappel fichiers utilisés dans la création du plan Terraform :

terraform/

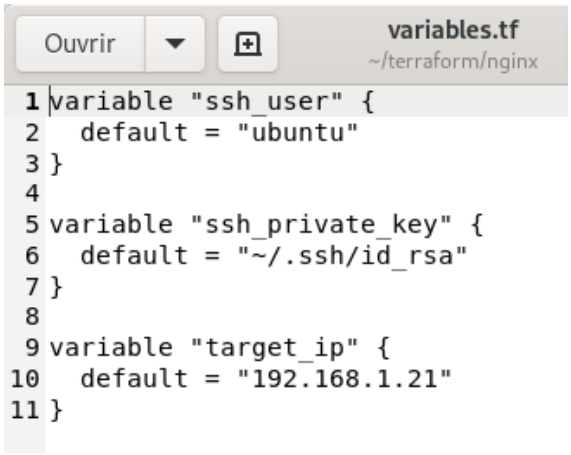
├── **main.tf**

├── **variables.tf**

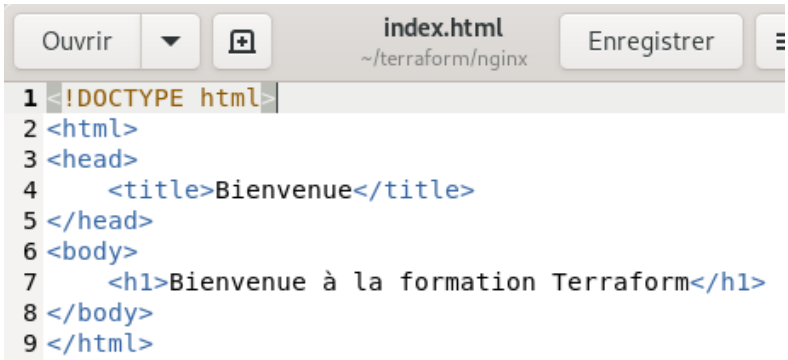
├── **index.html**

└── **nginx_custom.conf**

- **variables.tf**



- **index.html**



- **nginx_custom.conf**

Ouvrir — nginx_custom.conf — ~/terraform/nginx — Enr

```
server {

# Indique que le serveur écoute les connexions entrantes sur le port TCP 6666
    listen 6666 default_server;
    listen [::]:6666 default_server; # pour les connexions IPv6

# chemin absolu vers le répertoire racine
    root /var/www/html;

#fichier qui sera servit dans ce répertoire
    index index.html;

# Définit les noms de domaine pour lesquels ce serveur répondra
    server_name _;

# configuration pour les requêtes dont l'URI (Uniform Resource Identifier, la partie de l'URL après le nom de domaine)
    location / {

# Si aucun fichier n'est trouvé, le serveur renverra une erreur HTTP 404 (Not Found)
        try_files $uri $uri/ =404;
    }
}
```

- **main.tf**

```
terraform {
  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "3.0.1-rc8"
    }
  }
}


provider "proxmox" {

  pm_api_url = "https://192.168.1.10:8006/api2/json"

  pm_api_token_id = "terraform@pam!terraform-token"
  pm_api_token_secret = "66091622-5711-4c42-819b-3c5ab85bd133"
  pm_tls_insecure = true
}



  # 1. Connexion ssh sur le serveur cible
resource "null_resource" "ssh_target" {
  connection {
    type        = "ssh"
    user        = var.ssh_user
    host        = var.target_ip
    private_key = file(var.ssh_private_key)
  }
```

```
  # 2. Installation NGINX
  provisioner "remote-exec" {
    inline = [
      "sudo apt update -y",
      "sudo apt install nginx -y"
    ]
  }

  # 3. Copie du fichier HTML file
  provisioner "file" {
    source      = "index.html"
    destination = "/tmp/index.html"
  }

  # 4. Copie du fichier de configuration nginx
  provisioner "file" {
    source      = "nginx_custom.conf"
    destination = "/tmp/nginx_custom.conf"
  }

  # 5. Déplacement des fichiers et chargement de la conf nginx
  provisioner "remote-exec" {
    inline = [
      "sudo mv /tmp/index.html /var/www/html/index.html",
      "sudo mv /tmp/nginx_custom.conf /etc/nginx/sites-available/default",
      "sudo systemctl reload nginx"
    ]
  }

  # 6. Test du serveur nginx sur le port 6666
  provisioner "remote-exec" {
    inline = [
      "sleep 2",
      "curl -s http://localhost:6666 | grep 'Bienvenue à la formation Terraform'"
    ]
  }
}
```

b. Test du plan Terraform (terraform plan) pour Nginx

```
fabrice@debian11:~/terraform/nginx$ terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # null_resource.ssh_target will be created
  + resource "null_resource" "ssh_target" {
      + id = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

c. Application du plan Terraform (terraform apply) pour Nginx

```
null_resource.ssh_target: Creating...
null_resource.ssh_target: Provisioning with 'remote-exec'...
null_resource.ssh_target (remote-exec): Connecting to remote host via SSH...
null_resource.ssh_target (remote-exec):   Host: 192.168.1.21
null_resource.ssh_target (remote-exec):   User: ubuntu
null_resource.ssh_target (remote-exec):   Password: false
null_resource.ssh_target (remote-exec):   Private key: true
null_resource.ssh_target (remote-exec):   Certificate: false
null_resource.ssh_target (remote-exec):   SSH Agent: true
null_resource.ssh_target (remote-exec):   Checking Host Key: false
null_resource.ssh_target (remote-exec):   Target Platform: unix
null_resource.ssh_target (remote-exec): Connected!
null_resource.ssh_target: Still creating... [10s elapsed]
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Working]
null_resource.ssh_target (remote-exec): Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Connecting to archive.ubuntu.com]
null_resource.ssh_target (remote-exec): Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Waiting for headers]
null_resource.ssh_target (remote-exec): Hit:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Waiting for headers]
null_resource.ssh_target (remote-exec): Hit:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Working]
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Working]
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Working]
null_resource.ssh_target (remote-exec):

null_resource.ssh_target (remote-exec): Reading state information... 0%
null_resource.ssh_target (remote-exec): Reading state information... Done
null_resource.ssh_target (remote-exec): The following additional packages will be installed:
null_resource.ssh_target (remote-exec):   fontconfig-config fonts-dejavu-core
null_resource.ssh_target (remote-exec):   libfontconfig1 libgd3 libjbig0
null_resource.ssh_target (remote-exec):   libjpeg-turbo8 libjpeg8
null_resource.ssh_target (remote-exec):   libnginx-mod-http-image-filter
null_resource.ssh_target (remote-exec):   libnginx-mod-http-xslt-filter
null_resource.ssh_target (remote-exec):   libnginx-mod-mail
null_resource.ssh_target (remote-exec):   libnginx-mod-stream libtiff5
null_resource.ssh_target (remote-exec):   libwebp6 libxpm4 nginx-common
null_resource.ssh_target (remote-exec):   nginx-core
null_resource.ssh_target (remote-exec): Suggested packages:
null_resource.ssh_target (remote-exec):   libgd-tools fcgiwrap nginx-doc
null_resource.ssh_target (remote-exec):   ssl-cert
null_resource.ssh_target (remote-exec): The following NEW packages will be installed:
null_resource.ssh_target (remote-exec):   fontconfig-config fonts-dejavu-core
null_resource.ssh_target (remote-exec):   libfontconfig1 libgd3 libjbig0
null_resource.ssh_target (remote-exec):   libjpeg-turbo8 libjpeg8
null_resource.ssh_target (remote-exec):   libnginx-mod-http-image-filter
null_resource.ssh_target (remote-exec):   libnginx-mod-http-xslt-filter
null_resource.ssh_target (remote-exec):   libnginx-mod-mail
null_resource.ssh_target (remote-exec):   libnginx-mod-stream libtiff5
null_resource.ssh_target (remote-exec):   libwebp6 libxpm4 nginx nginx-common
null_resource.ssh_target (remote-exec):   nginx-core
null_resource.ssh_target (remote-exec): 0 upgraded, 17 newly installed, 0 to remove and 1 not upgraded.
null_resource.ssh_target (remote-exec): Need to get 2438 kB of archives.
null_resource.ssh_target (remote-exec): After this operation, 7926 kB of additional disk space will be used.
null_resource.ssh_target (remote-exec):
null_resource.ssh_target (remote-exec): 0% [Working]
null_resource.ssh_target (remote-exec): Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 fonts-dejavu
```

```
null_resource.ssh_target (remote-exec): Preparing to unpack .../00-fonts-dejavu-core_2.37-1_all.deb ...
Progress: [  1%] [..................] : Unpacking fonts-dejavu-core (2.37-1) ...
Progress: [  3%] [..................] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package fontconfig-config.
null_resource.ssh_target (remote-exec): Preparing to unpack .../01-fontconfig-config_2.13.1-2ubuntu3_all.deb
.
Progress: [  4%] [..................] : Unpacking fontconfig-config (2.13.1-2ubuntu3) ...
Progress: [  6%] [#.................] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package libfontconfig1:amd64.
null_resource.ssh_target (remote-exec): Preparing to unpack .../02-libfontconfig1_2.13.1-2ubuntu3_amd64.deb .
Progress: [  7%] [#.................] : Unpacking libfontconfig1:amd64 (2.13.1-2ubuntu3) ...
Progress: [  9%] [#.................] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package libjpeg-turbo8:amd64.
null_resource.ssh_target (remote-exec): Preparing to unpack .../03-libjpeg-turbo8_2.0.3-0ubuntu1.20.04.3_amd64
deb ...
Progress: [ 10%] [#.................] : Unpacking libjpeg-turbo8:amd64 (2.0.3-0ubuntu1.20.04.3) ...
Progress: [ 12%] [##................] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package libjpeg8:amd64.
null_resource.ssh_target (remote-exec): Preparing to unpack .../04-libjpeg8_8c-2ubuntu8_amd64.deb ...
Progress: [ 13%] [##................] : Unpacking libjpeg8:amd64 (8c-2ubuntu8) ...
Progress: [ 14%] [##................] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package libjbig0:amd64.
null_resource.ssh_target (remote-exec): Preparing to unpack .../05-libjbig0_2.1-3.1ubuntu0.20.04.1_amd64.deb
.
Progress: [ 16%] [##................] : Unpacking libjbig0:amd64 (2.1-3.1ubuntu0.20.04.1) ...
Progress: [ 17%] [###...............] :
null_resource.ssh_target (remote-exec): Selecting previously unselected package libwebp6:amd64.
null_resource.ssh_target (remote-exec): Preparing to unpack .../06-libwebp6_0.6.1-2ubuntu0.20.04.3_amd64.deb
.
Progress: [ 19%] [###...............] : Unpacking libwebp6:amd64 (0.6.1-2ubuntu0.20.04.3) ...
Progress: [ 20%] [###...............] :

null_resource.ssh_target (remote-exec): Selecting previously unselected package nginx.
null_resource.ssh_target (remote-exec): Preparing to unpack .../16-nginx_1.18.0-0ubuntu1.7_all.deb ...
Progress: [ 48%] [########..........] : Unpacking nginx (1.18.0-0ubuntu1.7) ...
Progress: [ 49%] [########..........] :
null_resource.ssh_target (remote-exec): Setting up libxpm4:amd64 (1:3.5.12-1ubuntu0.20.04.2) ...
Progress: [ 52%] [#########.........] : Setting up nginx-common (1.18.0-0ubuntu1.7) ...
Progress: [ 54%] [#########.........] :
null_resource.ssh_target (remote-exec): Created symlink /etc/systemd/system/multi-user.target.wants/nginx.ser
ce → /lib/systemd/system/nginx.service.
null_resource.ssh_target: Still creating... [1m20s elapsed]
Progress: [ 55%] [#########.........] : Setting up libjbig0:amd64 (2.1-3.1ubuntu0.20.04.1) ...
Progress: [ 58%] [##########........] : Setting up libnginx-mod-http-xslt-filter (1.18.0-0ubuntu1.7) ...
Progress: [ 61%] [##########........] :
null_resource.ssh_target (remote-exec): Setting up libwebp6:amd64 (0.6.1-2ubuntu0.20.04.3) ...
Progress: [ 64%] [###########.......] : Setting up fonts-dejavu-core (2.37-1) ...
Progress: [ 67%] [###########.......] : Setting up libjpeg-turbo8:amd64 (2.0.3-0ubuntu1.20.04.3) ...
Progress: [ 70%] [############......] : Setting up libjpeg8:amd64 (8c-2ubuntu8) ...
Progress: [ 72%] [############......] : Setting up libnginx-mod-mail (1.18.0-0ubuntu1.7) ...
Progress: [ 75%] [#############.....] : Setting up fontconfig-config (2.13.1-2ubuntu3) ...
Progress: [ 78%] [#############.....] : Setting up libnginx-mod-stream (1.18.0-0ubuntu1.7) ...
Progress: [ 81%] [##############....] : Setting up libtiff5:amd64 (4.1.0+git191117-2ubuntu0.20.04.14) ...
null_resource.ssh_target: Still creating... [1m30s elapsed]
Progress: [ 84%] [##############....] : Setting up libfontconfig1:amd64 (2.13.1-2ubuntu3) ...
Progress: [ 87%] [###############...] :
null_resource.ssh_target (remote-exec): Setting up libgd3:amd64 (2.2.5-5.2ubuntu2.4) ...
Progress: [ 90%] [################..] : Setting up libnginx-mod-http-image-filter (1.18.0-0ubuntu1.7) ...
Progress: [ 91%] [################..] :
Progress: [ 93%] [################..] : Setting up nginx-core (1.18.0-0ubuntu1.7) ...
Progress: [ 94%] [################..] :
Progress: [ 96%] [#################.] : Setting up nginx (1.18.0-0ubuntu1.7) ...
Progress: [ 99%] [#################.] : Processing triggers for ufw (0.36-6ubuntu1.1) ...
null_resource.ssh_target (remote-exec): Processing triggers for systemd (245.4-4ubuntu3.24) ...
null_resource.ssh_target (remote-exec): Processing triggers for man-db (2.9.1-1) ...
null_resource.ssh_target (remote-exec): Processing triggers for libc-bin (2.31-0ubuntu9.17) ...
```

```
null_resource.ssh_target: Still creating... [1m50s elapsed]
null_resource.ssh_target: Provisioning with 'file'...
null_resource.ssh_target: Still creating... [2m0s elapsed]
null_resource.ssh_target: Provisioning with 'file'...
null_resource.ssh_target: Provisioning with 'remote-exec'...
null_resource.ssh_target (remote-exec): Connecting to remote host via SSH...
null_resource.ssh_target (remote-exec):   Host: 192.168.1.21
null_resource.ssh_target (remote-exec):   User: ubuntu
null_resource.ssh_target (remote-exec):   Password: false
null_resource.ssh_target (remote-exec):   Private key: true
null_resource.ssh_target (remote-exec):   Certificate: false
null_resource.ssh_target (remote-exec):   SSH Agent: true
null_resource.ssh_target (remote-exec):   Checking Host Key: false
null_resource.ssh_target (remote-exec):   Target Platform: unix
null_resource.ssh_target (remote-exec): Connected!
null_resource.ssh_target: Provisioning with 'remote-exec'...
null_resource.ssh_target (remote-exec): Connecting to remote host via SSH...
null_resource.ssh_target (remote-exec):   Host: 192.168.1.21
null_resource.ssh_target (remote-exec):   User: ubuntu
null_resource.ssh_target (remote-exec):   Password: false
null_resource.ssh_target (remote-exec):   Private key: true
null_resource.ssh_target (remote-exec):   Certificate: false
null_resource.ssh_target (remote-exec):   SSH Agent: true
null_resource.ssh_target (remote-exec):   Checking Host Key: false
null_resource.ssh_target (remote-exec):   Target Platform: unix
null_resource.ssh_target (remote-exec): Connected!
null_resource.ssh_target (remote-exec):     <h1>Bienvenue à la formation Terraform</h1>
null_resource.ssh_target: Creation complete after 2m8s [id=7599069795779604095]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

7.6. Création Playbook Ansible

a. Créer l'utilisateur admin sur les 2 serveurs

create_admin_user.yml

```yaml
---
- name: Création de l'utilisateur admin sur les serveurs
  hosts: ubuntu_servers
  become: yes
  tasks:
    - name: Création de l'utilisateur admin
      user:
        name: admin
        groups: admin
        append: yes
        shell: /bin/bash
        state: present
        create_home: yes

    - name: Création du dossier .ssh
      file:
        path: /home/admin/.ssh
        state: directory
        owner: admin
        group: admin
        mode: 0700
```

En résumé :

1. Création d'un nouvel **utilisateur** "**admin**"
2. Ajoute l'utilisateur "admin" au **groupe "admin**"
3. Configure le shell par défaut pour "admin" (**/bin/bash**)
4. S'assure que le compte de l'utilisateur "admin" est présent (state: present)
5. Création du répertoire personnel de l'utilisateur (**/home/admin**)
6. Crée un répertoire .ssh pour"admin" (**/home/admin/.ssh**)
7. Définit le propriétaire et le groupe du répertoire .ssh à "admin".
8. Attribue les **permissions 0700** au répertoire .ssh, donc seul "admin" a un accès complet à ce répertoire.

b. Créer et Echanger les clés SSH entre les 2 serveurs

exchange_ssh_keys.yml

```yaml
---
- name: Échange des clés SSH entre les 2 serveurs ubuntu
  hosts: ubuntu_servers
  gather_facts: false
  become: yes
  tasks :

    - name: Générer la paire de clés SSH pour l'utilisateur admin
      openssh_keypair:
        path: /home/admin/.ssh/id_rsa
        owner: admin
        group: admin
        mode: '0600'
      register: admin_ssh_key


    - name: Ajouter la clé publique de server1 sur server2
      when: inventory_hostname == 'server2'
      authorized_key:
        user: admin
        state: present
        key: "{{ hostvars['server1']['admin_ssh_key']['public_key'] }}"

    - name: Ajouter la clé publique de server2 sur server1
      when: inventory_hostname == 'server1'
      authorized_key:
        user: admin
        state: present
        key: "{{ hostvars['server2']['admin_ssh_key']['public_key'] }}"
```

En résumé :

1. Génère une paire de clés SSH (clé privée **/home/admin/.ssh/id_rsa** et clé publique **/home/admin/.ssh/id_rsa.pub**) pour admin, avec les permissions appropriées qui est enregistrée dans la variable **admin_ssh_key** pour chaque hôte.
2. Sur server2 :
   Ajoute la clé publique d'admin de server1 au fichier **~/.ssh/authorized_keys** sur server2.
3. Sur server1 :
   Ajoute la clé publique d'admin de server2 au fichier **~/.ssh/authorized_keys** sur server1.

c. Test de connexion SSH entre les 2 serveurs

test_ssh.yml

```yaml
---
- name: Test de connexion SSH entre les serveurs
  hosts: ubuntu_servers
  become: false

  vars:

    target_host: "{{ '192.168.1.22' if inventory_hostname == 'server1' else '192.168.1.21' }}"

  tasks:
    - name: Test de connexion SSH entre chaque serveur
      shell: ssh -o StrictHostKeyChecking=no -o ConnectTimeout=5 admin@{{ target_host }} "echo OK"
      register: ssh_result
      ignore_errors: yes

    - name: Affiche le résultat
      debug:
        msg: >
          {% if ssh_result.rc == 0 %}
            ✅ Connexion SSH réussie entre {{ inventory_hostname }} et {{ target_host }} !
          {% else %}
            ❌ Échec de la connexion SSH depuis {{ inventory_hostname }} vers {{ target_host }}.
            Erreur : {{ ssh_result.stderr }}
          {% endif %}
```

En résumé :

1. **Définit dynamiquement l'adresse IP du serveur** cible pour chaque serveur du groupe ubuntu_servers. Si l'hôte courant est server1, la cible est 192.168.1.22, et inversement
2. **Etablissement d'une connexion SSH** depuis chaque serveur du groupe ubuntu_servers vers le serveur cible en utilisant l'utilisateur admin.
3. Enregistre le résultat de chaque tentative de connexion SSH dans la variable **ssh_result**.
4. Affiche un message de débogage pour chaque serveur, indiquant si la **connexion SSH a réussi ou échoué**. En cas d'échec, le message inclut la sortie d'erreur de la commande SSH.