

Nébuleuse

Offre d'hébergement d'Erminig : une VM dédiée par client, applications libres à la carte, administration par Ansible et bastion Teleport.

- [Découvrir](#)
 - [Présentation](#)
 - [Mise en place du serveur Ansible](#)
 - [Inventaire et variables du client](#)
 - [Articulation des services](#)
- [Socle](#)
 - [Socle système et Docker Engine](#)
 - [Bastion Teleport](#)
- [Services](#)
 - [Traefik](#)
 - [Authentik](#)
 - [Nextcloud](#)
 - [Immich](#)
 - [Vaultwarden](#)
 - [Fail2ban](#)
- [Exploiter](#)
 - [Déploiement d'un client](#)
 - [Pièges et dépannage](#)
- [Clients](#)
 - [Fiche Atelier Penhoat](#)
 - [Fiche Cabinet Kerjean](#)

Découvrir

Présentation

En bref

Niveau : présentation, sans manipulation

Durée : 10 minutes de lecture

Prérequis : aucun

Niveau de test : reconstitué

Nébuleuse, c'est quoi ?

Nébuleuse est l'offre d'hébergement d'Erminig, une ESN bretonne fictive. Elle propose des applications auto-hébergées commandées à la carte, sur le modèle des hébergeurs en ligne. Elle repose sur des technologies libres : peu coûteuses, adaptées à un démarrage, et en phase avec les organisations qui veulent des outils libres ou une souveraineté nationale et européenne. Le client choisit ses applications sur un formulaire web, indique son nom de domaine, et reçoit une machine virtuelle dédiée, configurée et maintenue par Ansible depuis un serveur central.

Dans le scénario, cette documentation est celle de l'équipe d'exploitation de Nébuleuse, dont je suis le rédacteur. Les clients d'exemple sont l'Atelier Penhoat (`penhoat.example`) et le Cabinet Kerjean (`kerjean.example`).

Composant	Rôle
Formulaire de commande	Recueille le client, son domaine et la liste des applications choisies
Serveur Ansible	Nœud de contrôle : héberge le dépôt, le Vault et lance les playbooks
Bastion Teleport	Point d'entrée unique des administrateurs vers les VM, avec authentification forte et journal des sessions
Réseau d'administration	Réseau privé <code>10.99.0.0/24</code> , réservé à l'exploitation
VM client	Nœud géré : Debian 12 livrée avec une clé SSH, reçoit le socle puis les applications commandées

Commande web → Fichier de variables du client → Serveur Ansible → VM dédiée → Docker Compose → Traefik → Application

Concrètement, le formulaire de commande et la création de la VM (provisioning) sont hors périmètre de cette documentation. Le formulaire produit un fichier de variables, et Ansible reçoit une machine Debian 12 vierge, accessible en SSH, dont il s'occupe de tout le reste.

Le catalogue

Le catalogue de Nébuleuse regroupe l'ensemble des applications que l'hébergeur sait déployer. Le client en choisit autant qu'il le souhaite : chaque application est un rôle Ansible indépendant, activé par une liste dans les variables du client.

Application	Usage	Documentée dans ce TP
Nextcloud	Fichiers, agenda, contacts	Oui
Immich	Photos et vidéos	Oui
Vaultwarden	Gestionnaire de mots de passe	Oui
Jellyfin	Médiathèque	Non
PaperMerge	Gestion électronique de documents	Non
Home Assistant	Domotique	Non
Matrix (Element)	Messagerie	Non

Pour garder une documentation lisible, seules les applications de base sont détaillées ici. Les autres suivent exactement le même modèle de rôle : un dossier de templates, un enregistrement dans le SSO, une règle Traefik.

Le socle commun

Quelle que soit la commande, chaque VM reçoit le même socle. Il n'est pas facturé comme une application : il fait partie du service.

Brique	Rôle
Durcissement et Docker	Système à jour, pare-feu, moteur de conteneurs
Traefik	Publie les applications en HTTPS
Authentik	Authentification unique (SSO) des applications, sauf Vaultwarden
Fail2ban	Bloque les adresses qui multiplient les échecs de connexion

Brique	Rôle
Agent Teleport	Rattache la VM au bastion pour l'administration

Une VM par client : pourquoi ?

Deux architectures sont possibles pour héberger plusieurs clients. En multi-locataire, une seule installation sert tous les clients, ce qui est économique mais mélange les données et les pannes. En mono-locataire, chaque client a sa propre machine, ce qui coûte plus cher mais isole les données, les secrets et les incidents. Nébuleuse retient la seconde : une compromission ou une erreur de configuration chez un client ne peut pas toucher les autres, et la restauration d'un client est indépendante.

Le bastion

Les humains et l'automatisation n'utilisent pas le même chemin. L'automatisation (Ansible) se connecte par SSH depuis le serveur Ansible, dans le réseau d'administration. Les administrateurs, eux, passent par un bastion Teleport : une porte unique, protégée par une authentification forte, qui enregistre chaque session. Aucune VM n'accepte donc de connexion SSH depuis Internet, et toute intervention manuelle laisse une trace.

L'installation de Teleport et l'enrôlement d'une ressource sont détaillés dans le livre Teleport de cette documentation (pages « Installation » et « Enroler des ressources »). Cette documentation ne reprend que la partie propre à Nébuleuse : le déploiement de l'agent par Ansible.

Objectifs et contraintes

1. **Reproductibilité** : un nouveau client est déployé à partir d'une VM vierge avec une seule commande, hors saisie des variables.
2. **Commande à la carte** : la liste des applications est une simple variable, sans modifier le code.
3. **Isolation** : chaque client a ses secrets, son domaine et son fichier de variables chiffré.
4. **Aucun secret en clair dans le dépôt** : tout passe par Ansible Vault.
5. **Administration tracée** : SSH n'écoute que sur le réseau d'administration, et les interventions humaines passent par le bastion.
6. **Idempotence** : rejouer le déploiement sur un client à jour ne modifie rien.

Adressage et nommage

Machine	Nom	Adresse d'administration	Adresse publique
Serveur Ansible	ansible-01	10.99.0.2	—
Bastion Teleport	bastion	10.99.0.3	192.0.2.5
Premier client (Atelier Penhoat)	client-alpha	10.99.0.21	192.0.2.21
Second client (Cabinet Kerjean)	client-beta	10.99.0.22	192.0.2.22

Chaque client possède son propre domaine (`penhoat.example`, `kerjean.example`) et des sous-domaines par service : `sso.`, `cloud.`, `photos.`, `vault.`. Le domaine `nebuleuse.example` ne sert qu'au service lui-même (bastion, serveur Ansible). Les conteneurs sont nommés `<client>_<service>`, les jails Fail2ban `nebuleuse-<service>`, et les variables de rôle sont préfixées par le nom du rôle (`traefik_`, `authentik_`).

Structure du dépôt

```
nebuleuse/
├─ ansible.cfg
├─ requirements.txt
├─ requirements.yml
├─ site.yml
├─ inventory/
│   └─ hosts.yml
│   └─ group_vars/all/
│   └─ host_vars/client-alpha/
└─ roles/
    ├─ base-hardening/
    ├─ docker-engine/
    ├─ teleport-agent/
    ├─ traefik/
    ├─ authentik/
    ├─ fail2ban/
    ├─ nextcloud/
    └─ immich/
```

Comment lire cette documentation

Les pages se lisent dans l'ordre pour un premier déploiement. La page **Articulation des services** donne la vue d'ensemble : elle explique pourquoi les rôles sont appliqués dans cet ordre et comment les services se référencent entre eux. Chaque page de service suit ensuite le même plan : principe, prérequis, fichiers commentés, déploiement, vérification, pièges.

Les fichiers sont inspirés d'un déploiement réel sur Debian 12. Ils seront rejoués de bout en bout sur deux VM avant d'être annoncés comme testés.

Page suivante : **Mise en place du serveur Ansible.**

Mise en place du serveur Ansible

En bref

Niveau : intermédiaire

Durée : 30 minutes

Prérequis : une VM Debian 12 dans le réseau d'administration

Niveau de test : reconstitué

Le serveur Ansible, c'est quoi ?

Le serveur Ansible est le nœud de contrôle : la seule machine qui détient le dépôt, la clé SSH d'administration et le mot de passe du Vault. Les VM clientes n'ont rien d'installé en plus que SSH et Python.

Élément	Rôle
Environnement virtuel Python	Isole la version d'Ansible du système
Clé SSH <code>ed25519</code>	Authentifie le serveur Ansible auprès des VM
Fichier <code>.vault_pass</code>	Contient le mot de passe de déchiffrement du Vault

Dépôt Git → ansible-playbook → SSH (réseau d'administration) → VM client

Le serveur Ansible est le chemin des machines : il se connecte aux VM de façon automatique. Les interventions humaines, elles, passent par le bastion Teleport (voir le livre Teleport et la page de présentation du projet).

Concrètement, la version d'Ansible fournie par `apt` sous Debian 12 est ancienne. L'installation se fait donc dans un environnement virtuel avec `pip`, ce qui permet de figer la version et de la mettre à jour sans toucher au système.

Prérequis

1. **Une VM Debian 12** dans le réseau d'administration (`10.99.0.2`), avec 2 Go de mémoire.
2. **Un utilisateur non-root** avec `sudo` : la suite est exécutée avec cet utilisateur.
3. **Un accès SSH** aux VM clientes par la clé publique, avec l'utilisateur `deploy` (créé lors du provisioning).

Étape 1 : mettre le système à jour et installer les outils

```
# Mise à jour de l'index des paquets puis des paquets installés.
sudo apt update && sudo apt full-upgrade -y

# python3-venv : création d'environnements virtuels
# python3-pip   : installateur Python
# git           : gestion du dépôt
# jq, curl      : utiles pour interroger des API (voir la page Authentik)
sudo apt install -y python3-venv python3-pip git jq curl
```

Étape 2 : créer l'environnement virtuel et installer Ansible

```
# Création du dépôt de travail. Tout le projet vivra ici.
mkdir -p ~/nebuleuse && cd ~/nebuleuse
git init

# Création de l'environnement virtuel dans un dossier caché du répertoire personnel.
python3 -m venv ~/.venv/ansible

# Activation : le prompt se préfixe de (ansible), et "pip" / "ansible" pointent
# désormais vers l'environnement virtuel.
source ~/.venv/ansible/bin/activate
```

Le fichier `requirements.txt` fige la version d'Ansible utilisée par le projet.

```
# requirements.txt
# ansible-core : le moteur d'Ansible, sans les collections de modules (installées à part).
# La plage de versions évite une montée de version majeure involontaire.
ansible-core>=2.17,<2.19

# jmespath : requis par le filtre "json_query", utile pour exploiter les réponses d'API.
jmespath
```

```
pip install -r requirements.txt
ansible --version # vérification : affiche la version et le chemin de l'interpréteur Python
```

Les collections de modules sont déclarées dans un second fichier, indépendant de la version du moteur.

```
# requirements.yml
collections:
  # community.docker : modules docker_compose_v2, docker_network...
  - name: community.docker
  # community.general : modules divers (ufw, etc.) utilisés par certains rôles
  - name: community.general
  # ansible.posix : modules système (sysctl, authorized_key...)
  - name: ansible.posix
```

```
ansible-galaxy collection install -r requirements.yml
```

Étape 3 : créer la clé SSH d'administration

```
# -t ed25519 : algorithme moderne, clés courtes et rapides
# -f          : chemin du fichier de clé
# -C          : commentaire pour reconnaître la clé dans les authorized_keys
# La clé n'a volontairement pas de passphrase : elle est utilisée par des playbooks
# non interactifs. En contrepartie, le fichier doit rester en mode 600 et le serveur
# Ansible n'est accessible que depuis le réseau d'administration.
ssh-keygen -t ed25519 -f ~/.ssh/nebuleuse_ed25519 -C "ansible@nebuleuse"
```

La clé publique (`~/.ssh/nebuleuse_ed25519.pub`) est celle que le processus de provisioning dépose dans `~deploy/.ssh/authorized_keys` de chaque VM.

Étape 4 : le mot de passe du Vault

```
# Génère 32 caractères aléatoires et les écrit dans .vault_pass.  
openssl rand -base64 24 > .vault_pass  
  
# Lecture/écriture pour le propriétaire uniquement.  
chmod 600 .vault_pass
```

Le fichier `.gitignore` empêche que le mot de passe du Vault parte dans le dépôt.

```
# .gitignore  
# Ne jamais versionner le mot de passe du Vault ni la clé privée.  
.vault_pass  
*.pem  
*_ed25519
```

Le mot de passe du Vault doit aussi être conservé hors du serveur Ansible (gestionnaire de mots de passe, coffre physique). Sans lui, les fichiers chiffrés sont définitivement illisibles.

Étape 5 : le fichier de configuration `ansible.cfg`

```
# ansible.cfg  
[defaults]  
# Inventaire utilisé par défaut : évite de passer -i à chaque commande.  
inventory = inventory/hosts.yml  
  
# Dossier des rôles du projet.  
roles_path = roles  
  
# Utilisateur SSH sur les VM clientes.
```

```

remote_user = deploy

# Clé privée utilisée pour la connexion.
private_key_file = ~/.ssh/nebuleuse_ed25519

# Fichier contenant le mot de passe du Vault (créé à l'étape 4).
# Doit exister : Ansible échoue au démarrage si le fichier est introuvable.
vault_password_file = .vault_pass

# Nombre de machines traitées en parallèle.
forks = 10

# Détection automatique de l'interpréteur Python sur les cibles, sans avertissement.
interpreter_python = auto_silent

[privilege_escalation]
# Les tâches s'exécutent en root via sudo : l'utilisateur "deploy" a un sudo sans mot de
passe.
become = True
become_method = sudo

[ssh_connection]
# Pipelining : réduit le nombre d'opérations SSH par tâche, donc accélère les playbooks.
pipelining = True

# ControlMaster : réutilise une connexion SSH ouverte pendant 60 s.
# accept-new : accepte la clé d'hôte d'une VM inconnue à la première connexion,
# puis refuse tout changement ultérieur (protection contre l'usurpation).
ssh_args = -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKeyChecking=accept-new

```

Étape 6 : un premier inventaire

```

# inventory/hosts.yml
all:
  children:
    clients:                                # groupe regroupant toutes les VM clientes
    hosts:

```

```
client-alpha:
  # Adresse du réseau d'administration, pas l'adresse publique.
  ansible_host: 10.99.0.21
client-beta:
  ansible_host: 10.99.0.22
```

Vérification

```
# Affiche l'arbre de l'inventaire : les deux clients doivent apparaître sous "clients".
ansible-inventory --graph

# Test de connectivité SSH et de Python (ce n'est pas un ping réseau).
ansible clients -m ansible.builtin.ping
```

Résultat attendu pour chaque client :

```
client-alpha | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Pièges

1. `vault_password_file` **introuvable** : le message « The vault password file ... was not found » apparaît dès la première commande. Il suffit de créer `.vault_pass` (étape 4) avant tout.
2. **Permission denied (publickey)** : la clé publique n'est pas dans les `authorized_keys` de l'utilisateur `deploy`, ou la clé privée n'est pas en mode 600.
3. **Mauvais** `ansible` : si l'environnement virtuel n'est pas activé, la commande `ansible` appelle la version système. Le préfixe `(ansible)` du prompt sert de contrôle.
4. **Pas de** `changed` **sur un** `ping` : normal, ce module est en lecture seule.

Page suivante : **Inventaire, packs et variables.**

Inventaire et variables du client

En bref

Niveau : intermédiaire

Durée : 20 minutes

Prérequis : serveur Ansible en place

Niveau de test : reconstitué

Inventaire et variables, c'est quoi ?

L'inventaire dit **sur quelles machines** Ansible agit. Les variables disent **comment** configurer chacune. Dans Nébuleuse, la commande d'un client se résume à un dossier de variables : le code (les rôles) ne change jamais d'un client à l'autre.

Emplacement	Contenu	Chiffré
<code>inventory/hosts.yml</code>	Liste des machines et leur adresse d'administration	Non
<code>inventory/group_vars/all/vars.yml</code>	Valeurs communes à tous les clients	Non
<code>inventory/host_vars/<client>/vars.yml</code>	Valeurs propres à un client, dont la liste des applications	Non
<code>inventory/host_vars/<client>/vault.yml</code>	Secrets du client	Oui (Ansible Vault)

Commande web → `vars.yml` **du client** → `nebuleuse_apps` → **rôles activés par** `site.yml`

Concrètement, le formulaire de commande génère deux fichiers dans `host_vars/<client>/` : `vars.yml` (le domaine et les applications choisies) et `vault.yml` (des secrets générés aléatoirement, puis chiffrés). Un nouveau client, c'est donc un dossier de plus, sans modifier de rôle.

Prérequis

1. Le serveur Ansible est en place (page précédente), avec `.vault_pass` créé.
2. L'inventaire contient `client-alpha` dans le groupe `clients`.

Précédence des variables

Une même variable peut être définie à plusieurs niveaux. Ansible retient la valeur du niveau le plus fort. Dans ce projet, seuls ces niveaux sont utilisés, du plus faible au plus fort :

1. Les valeurs par défaut du rôle (`roles/<rôle>/defaults/main.yml`)
2. Les variables communes (`group_vars/all`)
3. Les variables du client (`host_vars/<client>`)
4. Les variables passées en ligne de commande (`-e`)

Les valeurs par défaut portent donc les choix raisonnables, `group_vars` les choix de l'hébergeur, et `host_vars` ce qui est propre au client.

Les variables communes

```
# inventory/group_vars/all/vars.yml
# Valeurs partagées par tous les clients de Nébuleuse. Aucun secret ici.

# Domaine de l'hébergeur : sert au bastion et aux adresses techniques.
nebuleuse_domain: nebuleuse.example

# Adresse e-mail déclarée auprès de l'autorité de certification (Let's Encrypt).
# Elle reçoit les alertes d'expiration de certificat.
nebuleuse_acme_email: certificats@nebuleuse.example

# Réseau Docker partagé entre Traefik et les applications.
docker_proxy_network: proxy

# Réseau d'administration : seul à pouvoir joindre le port SSH.
admin_network_cidr: 10.99.0.0/24

# Adresse du bastion Teleport (proxy) auquel chaque agent se rattache.
teleport_proxy_addr: bastion.nebuleuse.example:443
```

```
# Catalogue : applications que Nébuleuse sait déployer.
# Une application commandée doit figurer ici, sinon le déploiement s'arrête.
nebuleuse_catalogue:
  - nextcloud
  - immich
  - vaultwarden
  - jellyfin
  - papermerge
  - homeassistant
  - matrix
```

Les variables d'un client

```
# inventory/host_vars/client-alpha/vars.yml
# Généré à partir de la commande du client. Aucun secret ici.

# Identifiant court du client : sert de préfixe aux conteneurs (alpha_nextcloud...).
client_name: alpha

# Domaine du client : chaque application aura un sous-domaine.
client_domain: penhoat.example

# Contact administrateur du client (compte initial du SSO, notifications).
client_admin_email: admin@penhoat.example

# Applications commandées. C'est la seule variable qui pilote ce qui est déployé.
nebuleuse_apps:
  - nextcloud
  - immich
  - vaultwarden

# Sous-domaines de chaque service. Le SSO est toujours présent (socle).
client_subdomains:
  sso: sso
  nextcloud: cloud
  immich: photos
  vaultwarden: vault
```

Les secrets d'un client

Les secrets suivent une convention : chaque secret est défini dans `vault.yml` sous un nom préfixé par `vault_`, puis référencé dans `vars.yml` ou dans un rôle sous son nom normal. Ainsi, on repère d'un coup d'œil les variables sensibles, et on voit où une valeur est utilisée sans déchiffrer quoi que ce soit.

```
# inventory/host_vars/client-alpha/vault.yml (AVANT chiffrement)
# Toutes les valeurs ci-dessous sont fictives.

vault_authentik_secret_key: "FICTIF-ab12cd34ef56ab12cd34ef56ab12cd34ef56"
vault_authentik_db_password: "FICTIF-db-motdepasse-1"
vault_nextcloud_db_password: "FICTIF-db-motdepasse-2"
vault_nextcloud_admin_password: "FICTIF-admin-motdepasse-3"
vault_immich_db_password: "FICTIF-db-motdepasse-4"
vault_vaultwarden_admin_token: "FICTIF-token-admin-5"
vault_immich_admin_password: "FICTIF-admin-motdepasse-6"
```

```
# Chiffrement du fichier avec le mot de passe de .vault_pass (voir ansible.cfg).
ansible-vault encrypt inventory/host_vars/client-alpha/vault.yml

# Pour modifier plus tard : ouvre le fichier déchiffré dans l'éditeur, puis le rechiffre à la
sauvegarde.
ansible-vault edit inventory/host_vars/client-alpha/vault.yml
```

Après chiffrement, le fichier commence par `$ANSIBLE_VAULT;1.1;AES256` suivi d'un bloc illisible. Il peut être versionné sans risque tant que le mot de passe du Vault reste secret.

Dans un rôle, le secret est consommé sous son nom propre :

```
# roles/authentik/defaults/main.yml
# Le rôle ne connaît que "authentik_secret_key" : il ignore d'où vient la valeur.
authentik_secret_key: "{{ vault_authentik_secret_key }}"
authentik_db_password: "{{ vault_authentik_db_password }}"
```

Vérifier la commande avant de déployer

Un garde-fou en début de playbook arrête le déploiement si le client commande une application inconnue ou oublie un secret. Mieux vaut échouer tout de suite que découvrir l'erreur au milieu d'un déploiement.

```
# site.yml (extrait : tâches de contrôle)
pre_tasks:
  - name: Vérifier que toutes les applications commandées existent au catalogue
    ansible.builtin.assert:
      that:
        # "difference" retourne les éléments de la commande absents du catalogue.
        - nebuleuse_apps | difference(nebuleuse_catalogue) | length == 0
      fail_msg: >-
        Application(s) inconnue(s) : {{ nebuleuse_apps | difference(nebuleuse_catalogue) }}
      quiet: true

  - name: Vérifier que le client a bien un domaine
    ansible.builtin.assert:
      that:
        - client_domain is defined
        - client_domain | length > 0
      fail_msg: "La variable client_domain doit être définie pour {{ inventory_hostname }}."
      quiet: true
```

Vérification

```
# Affiche toutes les variables fusionnées d'un client (le Vault est déchiffré).
# Attention : la sortie contient les secrets en clair, ne pas la partager.
ansible-inventory --host client-alpha --yaml

# Contrôle la syntaxe du playbook sans rien exécuter.
ansible-playbook site.yml --syntax-check
```

Pour s'assurer qu'aucun fichier de secrets n'est resté en clair avant un commit :

```
# Liste les vault.yml qui NE commencent PAS par l'en-tête de chiffrement.  
# Aucune ligne ne doit s'afficher.  
grep -L '^$ANSIBLE_VAULT' inventory/host_vars/*/vault.yml
```

Pièges

1. **Variable `vault_...` non définie** : un secret référencé dans un rôle mais absent de `vault.yml` provoque « is undefined » au milieu du déploiement. Les tâches de contrôle ci-dessus évitent ce cas pour les variables essentielles.
2. **Un fichier `host_vars/client-alpha.yml` et un dossier `host_vars/client-alpha/`** : les deux sont lus, ce qui rend l'origine d'une valeur difficile à retrouver. Il faut choisir un seul format, ici le dossier.
3. **Secret affiché dans un journal** : une tâche qui manipule un secret doit porter `no_log: true`, sinon la valeur apparaît dans la sortie d'Ansible.
4. **Fichier versionné avant chiffrement** : si `vault.yml` est commité en clair, le secret reste dans l'historique Git même après chiffrement. Il faut alors le renouveler, pas seulement chiffrer le fichier.
5. **Mot de passe du Vault perdu** : les fichiers sont illisibles et les secrets doivent être régénérés. D'où la sauvegarde hors serveur recommandée à la page précédente.

Page suivante : **Articulation des services.**

Articulation des services

En bref

Niveau : présentation, sans manipulation

Durée : 15 minutes de lecture

Prérequis : pages précédentes lues

Niveau de test : reconstitué

Pourquoi cette page ?

Les services d'une VM Nébuleuse ne sont pas indépendants. Une application attend que le SSO existe, le SSO attend que le reverse proxy le publie, et le pare-feu applicatif lit les journaux des deux. Cette page donne la vue d'ensemble : qui dépend de qui, ce que chaque service doit savoir des autres, et dans quel ordre Ansible doit les installer. Les pages suivantes détaillent chaque service ; celle-ci explique pourquoi ils s'emboîtent.

Service	Rôle dans l'ensemble
Traefik	Reçoit tout le trafic web et le dirige selon le nom de domaine
Authentik	Identifie les utilisateurs pour toutes les applications
Applications	S'appuient sur Traefik pour être publiées, sur Authentik pour connecter les utilisateurs
Fail2ban	Lit les journaux et bloque les adresses fautives
Agent Teleport	Rattache la VM au bastion, indépendamment du reste

Utilisateur → Traefik → Application → redirection vers Authentik → retour à l'application connecté

Vue d'ensemble

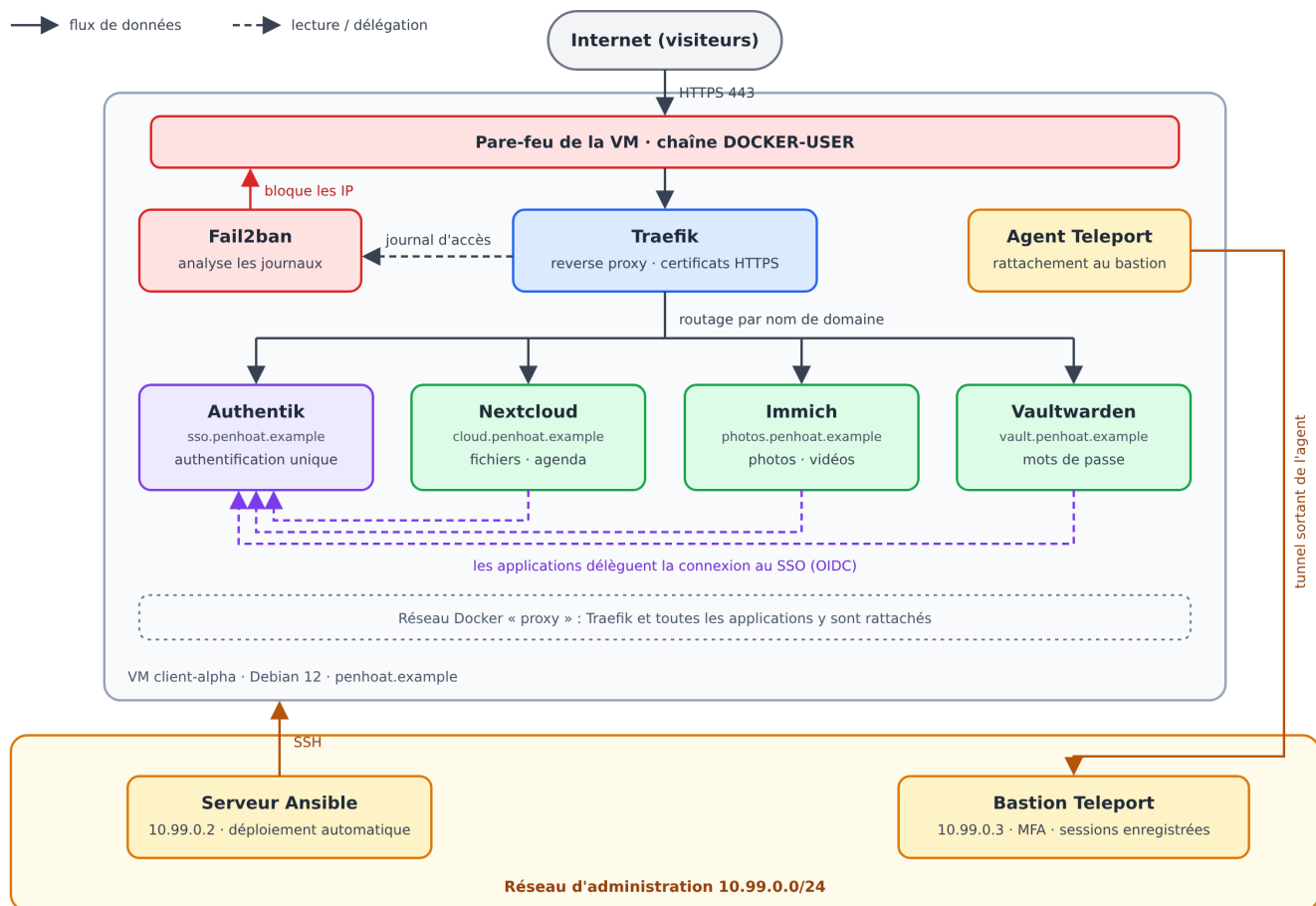


Figure 1 : architecture d'une VM client Nébuleuse et de son réseau d'administration.

Le schéma se lit de haut en bas. Le trafic des visiteurs traverse le pare-feu de la VM, atteint Traefik, qui l'aiguille vers l'application selon le nom de domaine. Les applications n'identifient pas elles-mêmes les utilisateurs : elles renvoient vers Authentik (flèches violettes). Fail2ban lit le journal d'accès de Traefik et bloque les adresses fautives au niveau du pare-feu. En bas, le réseau d'administration porte deux accès distincts : SSH depuis le serveur Ansible pour l'automatisation, et un tunnel sortant de l'agent Teleport vers le bastion pour les interventions humaines.

Les relations, service par service

Chaque ligne ci-dessous est une information qu'un service doit connaître d'un autre. Si l'une d'elles est fausse, la fonctionnalité correspondante ne marche pas, et le symptôme est rarement évident. C'est pourquoi ces valeurs sont des variables partagées, définies une seule fois.

Service	A besoin de	Information échangée
Traefik	Le réseau Docker <code>proxy</code>	Nom du réseau, pour trouver les conteneurs
Traefik	Un e-mail ACME	Pour obtenir les certificats HTTPS

Service	A besoin de	Information échangée
Chaque application	Le réseau <code>proxy</code>	Pour être joignable par Traefik
Chaque application	Des labels Traefik	Nom de domaine (<code>Host(...)</code>) et port interne
Chaque application	Authentik	Adresse de l'émetteur (issuer), identifiant client, secret client
Authentik	Chaque application	Adresse de retour (redirect URI) exacte
Fail2ban	Le journal d'accès de Traefik	Chemin du fichier, format des lignes
Fail2ban	La chaîne <code>DOCKER-USER</code>	Le point d'insertion des blocages
Agent Teleport	Le bastion	Adresse du proxy et jeton d' enrôlement

Vaultwarden fait exception : son coffre est chiffré avec le mot de passe principal de chaque utilisateur, qui ne peut pas être délégué au SSO. Il n'a donc pas de connexion dans Authentik (voir sa page).

Deux relations demandent une attention particulière :

1. **L'adresse de retour (redirect URI)** doit être identique, caractère pour caractère, dans Authentik et dans l'application. Un `/` final en trop suffit à provoquer une erreur de connexion.
2. **Les adresses réelles des visiteurs** : les applications ne voient que l'adresse de Traefik, sauf si elles sont configurées pour lire l'en-tête `X-Forwarded-For` envoyé par un proxy de confiance. Sans cela, les journaux ne contiennent que l'adresse interne, et Fail2ban bloquerait la mauvaise adresse, ou aucune.

Ordre de déploiement

L'ordre des rôles dans `site.yml` n'est pas arbitraire. Chaque étape suppose que la précédente est terminée.

1. **Socle et Docker** : le système est à jour, Docker est installé, le réseau `proxy` existe et la chaîne `DOCKER-USER` est disponible.
2. **Agent Teleport** : l'accès administrateur est établi avant les services applicatifs. Si un déploiement ultérieur échoue, la machine reste accessible par le bastion.
3. **Traefik** : il écoute sur les ports 80 et 443 et obtient les certificats. Il doit exister avant les applications, qui déclarent leurs routes par labels.
4. **Authentik** : une fois démarré derrière Traefik, son API est joignable. Les applications en ont besoin pour créer leurs connexions SSO.

5. **Applications commandées** : chacune crée sa connexion dans Authentik (API), récupère l'identifiant et le secret, puis génère son fichier Compose.
6. **Fail2ban** : il vient en dernier, car ses règles pointent vers des journaux qui n'existent qu'une fois Traefik et les applications démarrés.

```
# site.yml (extrait : ordre d'application des rôles)
roles:
  # 1. Socle : système, pare-feu, Docker et réseau "proxy".
  - { role: base-hardening, tags: [base] }
  - { role: docker-engine, tags: [docker] }

  # 2. Accès administrateur par le bastion.
  - { role: teleport-agent, tags: [teleport] }

  # 3. Publication web et certificats.
  - { role: traefik, tags: [traefik] }

  # 4. SSO : toujours présent, il fait partie du socle.
  - { role: authentik, tags: [sso] }

  # 5. Applications : déployées uniquement si elles figurent dans la commande.
  - { role: nextcloud, tags: [nextcloud], when: "'nextcloud' in nebuleuse_apps" }
  - { role: immich, tags: [immich], when: "'immich' in nebuleuse_apps" }
  - { role: vaultwarden, tags: [vaultwarden], when: "'vaultwarden' in nebuleuse_apps" }

  # 6. Protection : en dernier, elle lit les journaux des rôles précédents.
  - { role: fail2ban, tags: [fail2ban] }
```

La condition `when` d'un rôle s'applique à chacune de ses tâches : si l'application n'est pas commandée, aucune tâche du rôle ne s'exécute.

Comment une valeur voyage d'un service à l'autre

Un identifiant client créé par Authentik doit se retrouver dans le fichier Compose de l'application. Il ne vient pas d'une variable écrite à l'avance : il est créé pendant le déploiement. Ansible le récupère donc dans la réponse de l'API, le mémorise, puis le fournit au modèle de fichier.

Appel API Authentik → réponse (identifiant, secret) → variable de la tâche suivante → modèle `docker-compose.yml.j2` → fichier sur la VM

C'est ce qui oblige à placer Authentik avant les applications, et à rendre ces appels rejouables : si la connexion existe déjà, la tâche récupère ses valeurs au lieu de la recréer. Le détail est donné dans la page Authentik.

Attendre qu'un service soit prêt

Démarrer un conteneur ne signifie pas que le service répond. Authentik met plusieurs dizaines de secondes à devenir joignable, et une tâche qui l'appelle trop tôt échoue sans raison valable. Le rôle attend donc que l'adresse de santé réponde, en réessayant à intervalles réguliers.

```
# roles/authentik/tasks/main.yml (extrait)
- name: Attendre que l'API d'Authentik réponde
  # curl -f : retourne une erreur si le serveur répond autre chose qu'un succès.
  # -s -S : silencieux, mais affiche les erreurs.
  ansible.builtin.command: >-
    curl -fsS https://sso.{{ client_domain }}/-/health/live/
  register: authentik_health
  # Réessaie jusqu'à ce que la commande réussisse : 30 essais, 10 secondes d'écart.
  until: authentik_health.rc == 0
  retries: 30
  delay: 10
  # Ne modifie rien : l'indicateur "changed" n'a pas de sens ici.
  changed_when: false
```

Vérification

```
# Depuis le serveur Ansible : liste l'ordre exact des tâches pour un client,
# sans rien exécuter, pour contrôler que les rôles sont enchaînés comme prévu.
ansible-playbook site.yml --limit client-alpha --list-tasks

# Sur la VM : tous les conteneurs doivent être "Up" et sur le réseau "proxy".
docker ps --format '{{.Names}}\t{{.Status}}'
docker network inspect proxy --format '{{range .Containers}}{{.Name}} {{end}}'
```

Pièges

1. **Dépendance circulaire apparente** : Authentik est publié par Traefik, et les applications ont besoin d'Authentik. Il n'y a pas de cycle tant que l'ordre Traefik, puis Authentik, puis applications est respecté.
2. **Rejouer une partie seulement** : relancer uniquement le tag d'une application ne recrée pas les éléments de l'étape précédente. Si une valeur de Traefik ou d'Authentik change, il faut relancer aussi les rôles qui en dépendent.
3. **Fail2ban lancé trop tôt** : une jail dont le fichier journal n'existe pas ne démarre pas. D'où sa position en dernier.
4. **Résolution du nom du SSO depuis un conteneur** : une application doit contacter `sso.<domaine>` depuis l'intérieur de la VM. Selon la manière dont l'hébergeur attache l'adresse publique, ce chemin peut ne pas aboutir. Le cas est traité dans la page Traefik.

Page suivante : **Socle système et Docker Engine.**

Socle

Socle système et Docker Engine

En bref**Niveau** : intermédiaire**Durée** : 30 minutes**Prérequis** : inventaire et VM cliente Debian 12 accessible en SSH**Niveau de test** : reconstitué

Le socle, c'est quoi ?

Le socle est ce que reçoit toute VM Nébuleuse avant la moindre application : un système à jour, un accès SSH restreint, un pare-feu, puis Docker avec son réseau partagé. Il est porté par deux rôles, `base-hardening` et `docker-engine`, appliqués dans cet ordre.

Rôle	Rôle dans le socle
<code>base-hardening</code>	Mises à jour, SSH restreint au réseau d'administration, pare-feu <code>ufw</code>
<code>docker-engine</code>	Docker depuis le dépôt officiel, journaux limités, réseau <code>proxy</code>

VM Debian 12 vierge → `base-hardening` → `docker-engine` → **machine prête à recevoir Traefik**

Concrètement, la VM livrée par le provisioning possède déjà un utilisateur `deploy` autorisé par clé SSH et membre de `sudo` sans mot de passe. Le socle ne crée pas cet utilisateur : il en dépend, et il le verrouille comme seul compte autorisé à se connecter.

Prérequis

1. Les pages précédentes sont en place : inventaire, variables communes (`admin_network_cidr`) et `ansible ... -m ping` fonctionnel.
2. L'utilisateur `deploy` se connecte par clé et dispose de `sudo` sans mot de passe.

3. L'adresse `ansible_host` de chaque client est son adresse du réseau d'administration, car c'est celle sur laquelle SSH écouterait.

Arborescence des deux rôles

```
roles/
├─ base-hardening/
│   ├─ defaults/main.yml
│   ├─ handlers/main.yml
│   ├─ tasks/main.yml
│   └─ templates/sshd-nebuleuse.conf.j2
└─ docker-engine/
    ├─ defaults/main.yml
    ├─ handlers/main.yml
    ├─ tasks/main.yml
    └─ templates/daemon.json.j2
```

Le rôle `base-hardening`

Variables

```
# roles/base-hardening/defaults/main.yml

# Seul utilisateur autorisé à ouvrir une session SSH.
base_ssh_user: deploy

# Adresse sur laquelle SSH écoute : celle du réseau d'administration.
# "ansible_host" vaut cette adresse dans l'inventaire (voir page Inventaire).
base_ssh_listen_address: "{{ ansible_host }}"

# Réseau autorisé à joindre SSH (défini une fois pour tous dans group_vars/all).
base_admin_network_cidr: "{{ admin_network_cidr }}"

# Ports ouverts à tout Internet : uniquement le web.
base_public_tcp_ports:
```

- 80
- 443

Configuration SSH

```
# roles/base-hardening/templates/sshd-nebuleuse.conf.j2
# Fichier g  r   par Ansible : toute modification manuelle sera   cras  e.

# Pas de connexion directe en root.
PermitRootLogin no

# Authentification par cl   uniquement : aucun mot de passe n'est accept  .
PasswordAuthentication no
KbdInteractiveAuthentication no
PubkeyAuthentication yes

# Un seul compte peut se connecter.
AllowUsers {{ base_ssh_user }}

# SSH n'  coute que sur l'adresse d'administration, jamais sur l'adresse publique.
ListenAddress {{ base_ssh_listen_address }}
```

Sous Debian 12, le fichier principal `/etc/ssh/sshd_config` commence par une directive `Include /etc/ssh/sshd_config.d/*.conf`. Pour chaque param  tre, SSH retient la premi  re valeur lue : le fichier d  pos   dans ce dossier l'emporte donc sur le fichier principal, sans avoir    le modifier.

T  ches

```
# roles/base-hardening/tasks/main.yml

- name: Mettre    jour l'index des paquets et les paquets install  s
  ansible.builtin.apt:
    update_cache: true
    # "dist"   quivaut    "apt dist-upgrade" : g  re aussi les changements de d  pendances.
    upgrade: dist
    # Ne relit l'index qu'au bout d'une heure :   vite un t  l  chargement inutile    chaque
    passage.
```

```
cache_valid_time: 3600
```

- name: Installer les paquets du socle

```
ansible.builtin.apt:
```

```
name:
```

- ufw # pare-feu de l'hôte
- unattended-upgrades # installation automatique des mises à jour de sécurité
- ca-certificates # certificats racine pour les téléchargements HTTPS
- curl # utilisé par d'autres rôles

```
state: present
```

- name: Activer les mises à jour de sécurité automatiques

```
ansible.builtin.copy:
```

```
dest: /etc/apt/apt.conf.d/20auto-upgrades
```

```
content: |
```

```
APT::Periodic::Update-Package-Lists "1";
```

```
APT::Periodic::Unattended-Upgrade "1";
```

```
owner: root
```

```
group: root
```

```
mode: "0644"
```

- name: Déposer la configuration SSH durcie

```
ansible.builtin.template:
```

```
src: sshd-nebuleuse.conf.j2
```

```
dest: /etc/ssh/sshd_config.d/10-nebuleuse.conf
```

```
owner: root
```

```
group: root
```

```
mode: "0644"
```

```
# Vérifie la syntaxe avant d'écraser le fichier : une erreur ici couperait l'accès.
```

```
validate: /usr/sbin/sshd -t -f %s
```

```
notify: Redémarrer ssh
```

```
# Le pare-feu : l'ordre compte. La règle SSH est créée AVANT l'activation
```

```
# du pare-feu, sinon la connexion en cours serait coupée.
```

- name: Refuser par défaut tout trafic entrant

```
community.general.ufw:
```

```
default: deny
```

```
direction: incoming
```

- name: Autoriser SSH depuis le réseau d'administration uniquement

```
community.general.ufw:
  rule: allow
  port: "22"
  proto: tcp
  src: "{{ base_admin_network_cidr }}"

- name: Autoriser le web (HTTP et HTTPS) depuis Internet
  community.general.ufw:
    rule: allow
    port: "{{ item }}"
    proto: tcp
  loop: "{{ base_public_tcp_ports }}"

- name: Activer ufw
  community.general.ufw:
    state: enabled
```

```
# roles/base-hardening/handlers/main.yml
# Un handler ne s'exécute qu'à la fin des tâches, et seulement si une tâche l'a notifié.
- name: Redémarrer ssh
  ansible.builtin.service:
    name: ssh          # le service s'appelle "ssh" sous Debian, pas "sshd"
    state: restarted
```

Le rôle `docker-engine`

Variables

```
# roles/docker-engine/defaults/main.yml

# Nom du réseau partagé entre Traefik et les applications (voir group_vars/all).
docker_engine_proxy_network: "{{ docker_proxy_network }}"

# Taille maximale d'un fichier de journal de conteneur, et nombre de fichiers conservés.
# Sans limite, les journaux d'un conteneur bavard finissent par remplir le disque.
docker_engine_log_max_size: "10m"
```

```
docker_engine_log_max_files: "3"
```

Configuration du démon

```
# roles/docker-engine/templates/daemon.json.j2
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "{{ docker_engine_log_max_size }}",
    "max-file": "{{ docker_engine_log_max_files }}"
  },
  "live-restore": true
}
```

Le paramètre `live-restore` garde les conteneurs en marche pendant un redémarrage ou une mise à jour du démon Docker : une mise à jour du moteur ne coupe donc pas les services des clients.

Tâches

```
# roles/docker-engine/tasks/main.yml

- name: Créer le dossier des clés de dépôt
  ansible.builtin.file:
    path: /etc/apt/keyrings
    state: directory
    mode: "0755"

- name: Récupérer la clé de signature du dépôt Docker
  ansible.builtin.get_url:
    url: https://download.docker.com/linux/debian/gpg
    dest: /etc/apt/keyrings/docker.asc
    mode: "0644"

- name: Déclarer le dépôt officiel Docker
  ansible.builtin.apt_repository:
    # "signed-by" lie ce dépôt à sa clé : les paquets sont vérifiés avec elle.
    # ansible_distribution_release vaut "bookworm" sur Debian 12.
    repo: >-
```

```
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian {{ ansible_distribution_release }} stable
filename: docker
state: present
```

- name: Installer Docker Engine et le plugin Compose

```
ansible.builtin.apt:
```

```
name:
```

- docker-ce
- docker-ce-cli
- containerd.io
- docker-buildx-plugin
- docker-compose-plugin # fournit la commande "docker compose"
- python3-docker # bibliothèque Python requise par les modules

```
community.docker
```

```
state: present
```

```
update_cache: true
```

- name: Déposer la configuration du démon Docker

```
ansible.builtin.template:
```

```
src: daemon.json.j2
```

```
dest: /etc/docker/daemon.json
```

```
owner: root
```

```
group: root
```

```
mode: "0644"
```

```
notify: Redémarrer docker
```

- name: Activer et démarrer Docker

```
ansible.builtin.service:
```

```
name: docker
```

```
state: started
```

```
enabled: true
```

```
# Les handlers sont lancés ici plutôt qu'en fin de rôle : le réseau ci-dessous
```

```
# doit être créé sur un démon déjà configuré.
```

- name: Appliquer les changements de configuration du démon

```
ansible.builtin.meta: flush_handlers
```

- name: Créer le réseau partagé entre Traefik et les applications

```
community.docker.docker_network:
```

```
name: "{{ docker_engine_proxy_network }}"
driver: bridge
```

```
# roles/docker-engine/handlers/main.yml
- name: Redémarrer docker
  ansible.builtin.service:
    name: docker
    state: restarted
```

L'utilisateur `deploy` n'est volontairement pas ajouté au groupe `docker`. Appartenir à ce groupe revient à avoir les droits root sur la machine. Ansible utilise déjà `sudo`, ce qui laisse une trace de chaque commande.

Docker et le pare-feu : ce qu'il faut savoir

Docker gère lui-même ses règles de pare-feu. Quand un conteneur publie un port (comme Traefik sur 80 et 443), Docker l'ouvre directement, sans passer par `ufw`. Ainsi, une règle `ufw` ne bloque pas un port publié par un conteneur.

Ce comportement est connu, et il explique deux choix du projet : 1. Seul Traefik publie des ports vers l'extérieur. Les autres conteneurs restent sur le réseau `proxy`, sans port publié. 2. Les blocages dynamiques (Fail2ban) ne passent pas par `ufw` mais par la chaîne `DOCKER-USER`. Docker crée cette chaîne et ne la modifie jamais ; les règles qu'on y place sont donc appliquées avant celles de Docker.

Le playbook du socle

Pour tester dès maintenant, un `site.yml` minimal suffit. Il sera complété à la page de déploiement.

```
# site.yml
- name: Socle Nébuleuse
  hosts: clients
  become: true          # les tâches s'exécutent en root via sudo
  roles:
    - { role: base-hardening, tags: [base] }
```

```
- { role: docker-engine, tags: [docker] }
```

```
# Simulation : montre ce qui changerait, sans rien modifier.  
ansible-playbook site.yml --limit client-alpha --check --diff  
  
# Déploiement réel.  
ansible-playbook site.yml --limit client-alpha
```

Vérification

```
# Depuis le serveur Ansible : un second passage ne doit rien modifier.  
# La ligne finale doit afficher "changed=0".  
ansible-playbook site.yml --limit client-alpha  
  
# Sur la VM : valeurs SSH effectives.  
sudo sshd -T | grep -E 'listenaddress|passwordauthentication|permitrootlogin|allowusers'  
  
# Sur la VM : règles du pare-feu.  
sudo ufw status verbose  
  
# Sur la VM : version de Docker, pilote de journaux et live-restore.  
docker version --format '{{.Server.Version}}'  
docker info --format 'journaux={{.LoggingDriver}} live-restore={{.LiveRestoreEnabled}}'  
  
# Sur la VM : le réseau partagé existe et la chaîne DOCKER-USER est disponible.  
docker network ls --filter name=proxy  
sudo iptables -L DOCKER-USER -n
```

Résultat attendu pour `sshd -T` : `listenaddress 10.99.0.21:22`, `passwordauthentication no`, `permitrootlogin no`, `allowusers deploy`.

Pièges

1. **Se couper l'accès en modifiant SSH** : une erreur de `ListenAddress` ou une règle de pare-feu oubliée coupe la connexion. La validation `sshd -t` et l'ordre « règle avant activation » limitent le risque. Avant de fermer une session de test, ouvrir une seconde connexion pour vérifier que l'accès fonctionne encore.

2. `ansible_host` **qui n'est pas l'adresse d'administration** : SSH n'écouterait pas là où Ansible se connecte. L'inventaire de la première page respecte la règle ; une machine ajoutée avec son adresse publique fausserait le rôle.
3. `apt` **bloqué par un autre processus** : juste après le démarrage d'une VM neuve, `unattended-upgrades` peut tenir le verrou `apt`. Ansible patiente un moment, puis échoue avec « Could not get lock ». Relancer le playbook suffit.
4. **Pare-feu** `ufw` **qui « ne bloque rien » pour un conteneur** : voir la section précédente. Ce n'est pas un défaut de configuration.
5. `docker_network` **en erreur sur la bibliothèque Python** : le paquet `python3-docker` doit être installé avant toute tâche `community.docker`. Il fait partie de la liste des paquets du rôle.
6. **Journaux de conteneurs qui remplissent le disque** : sans `max-size`, aucune rotation n'est faite. La configuration `daemon.json` ne s'applique qu'aux conteneurs créés après elle ; les conteneurs existants doivent être recréés.

Page suivante : **Bastion Teleport : agent et enrôlement.**

Bastion Teleport

En bref**Niveau** : avancé**Durée** : 45 minutes**Prérequis** : socle en place, bastion Teleport installé**Niveau de test** : reconstitué

Le bastion, c'est quoi ?

Un bastion est une porte d'entrée unique pour l'administration des machines. Au lieu d'ouvrir SSH sur chaque VM, les administrateurs se connectent au bastion, qui les authentifie, vérifie leurs droits, puis les met en relation avec la machine demandée en enregistrant la session. Nébuleuse utilise Teleport pour cela.

Composant	Rôle
Proxy Teleport (sur le bastion)	Reçoit les connexions des administrateurs et des agents, sert le portail web
Auth Teleport (sur le bastion)	Authentifie, émet les certificats, applique les droits, conserve les enregistrements
Agent Teleport (sur chaque VM)	Rattache la VM au bastion et ouvre les sessions demandées

Administrateur → Bastion (authentification forte) → Tunnel déjà ouvert par l'agent → VM client

Concrètement, c'est l'agent qui établit la connexion vers le bastion, pas l'inverse. La VM n'a donc aucun port d'administration à exposer : le tunnel est sortant, sur le port 443.

L'installation du bastion lui-même (serveur Teleport, nom de domaine, création d'utilisateurs) est détaillée dans le livre Teleport de cette documentation, pages « Installation » et « Enroler des ressources ». Cette page ne traite que ce qui est propre à Nébuleuse : déployer l'agent par Ansible, sans passer par l'interface web.

Pourquoi un déploiement par Ansible ?

L'interface web de Teleport génère une commande à copier sur la machine, avec un jeton temporaire. C'est pratique pour une machine, mais cela ne passe pas à l'échelle : chaque nouveau client demanderait une intervention manuelle. Le rôle `teleport-agent` fait la même chose automatiquement : il demande un jeton au bastion, installe l'agent, écrit sa configuration et le démarre.

Prérequis

1. Un bastion Teleport en service, joignable à `bastion.nebuleuse.example:443` depuis les VM clientes.
2. Le bastion est présent dans l'inventaire (voir ci-dessous), car Ansible doit y exécuter une commande.
3. La version de l'agent est inférieure ou égale à celle du bastion : un agent plus récent que le serveur n'est pas pris en charge.
4. Le socle système est appliqué sur la VM (page précédente).

Ajouter le bastion à l'inventaire

Le bastion n'est pas un client : il forme son propre groupe, ce qui permet à Ansible de lui déléguer la création des jetons.

```
# inventory/hosts.yml (version complétée)
all:
  children:
    bastion:                                # groupe du bastion : une seule machine
      hosts:
        bastion:
          ansible_host: 10.99.0.3
    clients:                                # groupe des VM clientes
      hosts:
        client-alpha:
          ansible_host: 10.99.0.21
```

```
client-beta:
  ansible_host: 10.99.0.22
```

Arborescence du rôle

```
roles/teleport-agent/
├─ defaults/main.yml
├─ handlers/main.yml
├─ tasks/main.yml
└─ templates/teleport.yaml.j2
```

Variables

```
# roles/teleport-agent/defaults/main.yml

# Version et édition de Teleport (édition communautaire, gratuite).
# Doit être inférieure ou égale à la version du bastion.
teleport_agent_version: "17.4.3"
teleport_agent_edition: "oss"

# Adresse du proxy du bastion (définie une fois pour tous dans group_vars/all).
teleport_agent_proxy: "{{ teleport_proxy_addr }}"

# Durée de validité du jeton d'enrôlement. Il ne sert qu'à la première connexion :
# une durée courte limite l'exposition si le jeton fuit.
teleport_agent_token_ttl: "30m"

# Étiquettes (labels) attachées à la machine dans Teleport.
# Les droits d'accès des administrateurs s'appuient dessus (voir plus bas).
teleport_agent_labels:
  client: "{{ client_name }}"
  env: production
```

Configuration de l'agent

```

# roles/teleport-agent/templates/teleport.yaml.j2
# Fichier g  r   par Ansible : toute modification manuelle sera   cras  e.
version: v3

teleport:
  # Nom affich   dans Teleport : le nom d'h  te de l'inventaire (unique par client).
  nodename: {{ inventory_hostname }}
  data_dir: /var/lib/teleport
  # Adresse du bastion : l'agent s'y connecte en sortant.
  proxy_server: {{ teleport_agent_proxy }}
{% if teleport_join_token.stdout | default('') | trim | length > 0 %}
  # Pr  sent uniquement lors du premier enr  lement. Une fois la machine enr  l  e,
  # elle conserve son identit   dans data_dir et n'a plus besoin du jeton.
  join_params:
    method: token
    token_name: {{ teleport_join_token.stdout | trim }}
{% endif %}
  log:
    output: stderr
    severity: INFO

# Cette machine n'est ni un serveur d'authentification ni un proxy.
auth_service:
  enabled: "no"
proxy_service:
  enabled: "no"

# Le service SSH de Teleport : celui qui ouvre les sessions pour les administrateurs.
ssh_service:
  enabled: "yes"
  labels:
{% for nom, valeur in teleport_agent_labels.items() %}
    {{ nom }}: {{ valeur }}
{% endfor %}

```

T  ches

```

# roles/teleport-agent/tasks/main.yml

# 1. Installer Teleport seulement s'il est absent.
- name: Chercher un Teleport déjà installé
  ansible.builtin.command: which teleport
  register: teleport_agent_installed
  changed_when: false
  failed_when: false          # un code de retour non nul signifie simplement "absent"

- name: Installer Teleport (script officiel, version figée)
  # Même méthode que la page « Installation » du livre Teleport.
  # "set -o pipefail" : un échec de curl fait échouer toute la commande.
  ansible.builtin.shell: |
    set -o pipefail
    curl -fsSL https://cdn.teleport.dev/install-v{{ teleport_agent_version }}.sh \
      | bash -s {{ teleport_agent_version }} {{ teleport_agent_edition }}
  args:
    executable: /bin/bash
  when: teleport_agent_installed.rc != 0

# 2. Savoir si la machine est déjà enrôlée.
# Le fichier host_uuid est créé au premier démarrage réussi de l'agent.
- name: Chercher une identité Teleport existante
  ansible.builtin.stat:
    path: /var/lib/teleport/host_uuid
  register: teleport_agent_identity

# 3. Demander un jeton au bastion, seulement pour une première installation.
- name: Générer un jeton d'enrôlement sur le bastion
  ansible.builtin.command: >-
    tctl tokens add --type=node --ttl={{ teleport_agent_token_ttl }} --format=text
  delegate_to: bastion          # la commande s'exécute sur le bastion, pas sur la VM
  register: teleport_join_token
  changed_when: true           # un nouveau jeton est créé à chaque exécution
  no_log: true                 # le jeton est un secret : il ne doit pas apparaître dans les
                                journaux
  when: not teleport_agent_identity.stat.exists

# 4. Écrire la configuration (accès restreint : elle contient le jeton au premier passage).
- name: Déposer la configuration de l'agent

```

```
ansible.builtin.template:
  src: teleport.yaml.j2
  dest: /etc/teleport.yaml
  owner: root
  group: root
  mode: "0600"
notify: Redémarrer teleport
no_log: true

# 5. Démarrer l'agent.
- name: Activer et démarrer l'agent Teleport
  ansible.builtin.service:
    name: teleport
    state: started
    enabled: true
```

```
# roles/teleport-agent/handlers/main.yml
- name: Redémarrer teleport
  ansible.builtin.service:
    name: teleport
    state: restarted
```

Les droits des administrateurs

L'agent rattache la machine, mais ne dit pas qui a le droit de s'y connecter. Cela se définit dans des rôles Teleport, qui s'appuient sur les étiquettes de la machine. Dans Nébuleuse, une personne du support ne voit que les machines des clients qu'elle suit.

```
# support-alpha.yaml : rôle Teleport (à créer une fois sur le bastion)
kind: role
version: v7
metadata:
  name: support-alpha
spec:
  allow:
    # Comptes Unix sur la VM avec lesquels la session peut s'ouvrir.
    logins: [deploy]
    # Seules les machines portant l'étiquette client=alpha sont visibles.
```

```
node_labels:
  client: alpha
```

```
# Sur le bastion : crée ou met à jour le rôle.
sudo tctl create -f support-alpha.yaml

# Crée un utilisateur avec ce rôle (même commande que dans le livre Teleport).
sudo tctl users add camille --roles=support-alpha
```

Dans cet exemple, la session s'ouvre avec le compte `deploy`, qui a `sudo` sans mot de passe : donner ce login revient à donner les droits administrateur sur la VM. Pour un vrai support, un compte dédié avec des droits limités est préférable. Il s'ajouterait au socle système.

L'authentification forte

La page de présentation parle d'authentification forte sur le bastion. Elle se configure côté serveur Teleport, dans la section `auth_service`. Voici le principe :

```
# /etc/teleport.yaml sur le bastion (extrait)
auth_service:
  authentication:
    type: local
    # La deuxième authentification est exigée pour chaque utilisateur.
    # "webauthn" accepte les clés de sécurité matérielles et les passkeys.
    second_factor: webauthn
    webauthn:
      rp_id: bastion.nebuleuse.example
```

Ce paramètre appartient à l'installation du bastion, pas à l'agent. Il est à valider sur ta version de Teleport avant de le recopier dans la page d'installation.

Déploiement

Le rôle s'insère dans `site.yaml` juste après le socle :

```
# site.yml (extrait)
roles:
  - { role: base-hardening, tags: [base] }
  - { role: docker-engine, tags: [docker] }
  - { role: teleport-agent, tags: [teleport] }
```

```
# Un seul rôle, sur un seul client.
ansible-playbook site.yml --limit client-alpha --tags teleport
```

Vérification

```
# Sur la VM : le service tourne et la connexion au bastion est établie.
systemctl status teleport --no-pager
journalctl -u teleport -n 20 --no-pager

# Sur le bastion : la machine est enrôlée, avec ses étiquettes.
sudo tctl nodes ls
```

Sur le portail web, la machine apparaît dans la liste des ressources sous le nom `client-alpha`, avec l'étiquette `client=alpha`. Depuis le poste d'un administrateur, la connexion se fait ensuite avec `tsh ssh deploy@client-alpha` après `tsh login`.

Pièges

1. **Jeton expiré** : si le bastion est injoignable ou si le déploiement traîne, le jeton (30 minutes) expire avant le démarrage de l'agent. Le premier démarrage échoue, et les journaux de l'agent indiquent un jeton refusé. Il suffit de relancer le rôle : le fichier d'identité n'existant pas encore, un nouveau jeton est créé.
2. **Deuxième passage qui « change » quelque chose** : au premier passage, la configuration contient le jeton. Au deuxième, la machine est enrôlée, aucun jeton n'est généré, et le fichier est réécrit sans la section `join_params`. Ce changement unique est normal ; à partir du troisième passage, plus rien ne change.
3. **Agent plus récent que le bastion** : si la version de `teleport_agent_version` dépasse celle du bastion, l'agent est refusé. Mettre à jour le bastion en premier.
4. **Machine absente du portail** : vérifier que la VM atteint `bastion.nebuleuse.example:443` en sortie (pare-feu, DNS). L'agent n'a besoin d'aucun port entrant.
5. **Jeton visible dans un journal** : les tâches qui le manipulent portent `no_log: true`. Ne pas retirer cette ligne pour déboguer, ou le faire de façon temporaire et sur une VM de

test.

6. **Chemin du binaire** : le test d'installation utilise la commande `which`, ce qui évite de dépendre de l'emplacement exact du binaire selon la version du paquet.

Page suivante : **Traefik**.

Services

Traefik

En bref**Niveau** : intermédiaire**Durée** : 30 minutes**Prérequis** : socle système et Docker en place**Niveau de test** : reconstitué

Traefik, c'est quoi ?

Traefik est un reverse proxy conçu pour Docker. Il reçoit tout le trafic web d'une VM, lit les étiquettes (labels) posées sur les conteneurs, et en déduit seul vers quelle application envoyer chaque requête. Il obtient aussi les certificats HTTPS et les renouvelle automatiquement.

Élément	Rôle
Point d'entrée <code>web</code> (port 80)	Reçoit le HTTP et redirige vers HTTPS ; sert au défi de validation Let's Encrypt
Point d'entrée <code>websecure</code> (port 443)	Reçoit le HTTPS et applique le certificat correspondant au nom demandé
Fournisseur Docker	Surveille les conteneurs et crée les routes à partir de leurs labels
Résolveur de certificats	Demande et renouvelle les certificats Let's Encrypt

Visiteur → `cloud.penhoot.example:443` → **Traefik** → **conteneur Nextcloud**

Concrètement, ajouter une application ne demande jamais de modifier Traefik : il suffit que son conteneur soit sur le réseau `proxy` et porte les bons labels. Traefik le détecte à chaud, sans redémarrage.

Prérequis

1. Le socle est en place : Docker installé, réseau `proxy` créé, ports 80 et 443 ouverts dans le pare-feu.

2. Un enregistrement DNS pointe chaque nom du client vers l'adresse publique de sa VM. Une entrée générique `*.penhoat.example` suffit.
3. Le port 80 est joignable depuis Internet : Let's Encrypt s'en sert pour vérifier que le domaine appartient bien au client.
4. Le domaine est réel. `penhoat.example` étant fictif, aucun certificat réel ne peut lui être délivré. Pour reproduire ce TP, remplacer le domaine par un domaine dont on dispose.

Pendant les essais, activer le mode « staging » de Let's Encrypt (variable `traefik_acme_staging`). Les certificats délivrés ne sont pas reconnus par les navigateurs, mais le serveur de test n'impose pas les mêmes limites de demandes que le serveur réel, qui bloque un domaine après quelques échecs répétés.

Arborescence du rôle

```
roles/traefik/  
├─ defaults/main.yml  
├─ handlers/main.yml  
├─ tasks/main.yml  
└─ templates/  
    ├─ docker-compose.yml.j2  
    ├─ logrotate-traefik.j2  
    └─ traefik.yml.j2
```

Sur la VM, le rôle crée :

```
/opt/nebuleuse/traefik/  
├─ docker-compose.yml  
├─ traefik.yml          # configuration statique  
└─ acme/acme.json      # certificats (créé par Traefik, accès réservé à root)  
  
/var/log/traefik/access.log  # journal d'accès, lu par Fail2ban
```

Variables

```
# roles/traefik/defaults/main.yml
```

```
# Version de Traefik. Choisir une version récente : avec les versions récentes de Docker,
```

```
# une version trop ancienne de Traefik ne parvient plus à lire l'API Docker.
traefik_version: "v3.6"

# Dossier de travail sur la VM.
traefik_dir: /opt/nebuleuse/traefik

# Dossier du journal d'accès (monté dans le conteneur, lu par Fail2ban).
traefik_log_dir: /var/log/traefik

# Adresse e-mail déclarée auprès de Let's Encrypt (définie dans group_vars/all).
traefik_acme_email: "{{ nebuleuse_acme_email }}"

# true : utilise le serveur de test de Let's Encrypt (certificats non reconnus, sans limites
strictes).
traefik_acme_staging: false

# Réseau Docker partagé avec les applications (défini dans group_vars/all).
traefik_network: "{{ docker_proxy_network }}"

# Noms de domaine que les applications doivent joindre à l'intérieur de la VM.
# Voir la section « Joindre le SSO depuis un conteneur ».
traefik_network_aliases:
  - "{{ client_subdomains.sso }}.{{ client_domain }}"
```

La configuration statique

La configuration statique décrit Traefik lui-même : ses ports, ses sources de configuration, ses certificats. Elle n'est lue qu'au démarrage.

```
# roles/traefik/templates/traefik.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.

global:
  # Pas de vérification de mise à jour ni de télémétrie : la VM n'a pas à contacter ces
services.
  checkNewVersion: false
  sendAnonymousUsage: false
```

```
log:
  level: INFO

# Journal d'accès : une ligne JSON par requête. Fail2ban le lit pour repérer les attaques.
accessLog:
  filePath: /var/log/traefik/access.log
  format: json

providers:
  docker:
    # Un conteneur n'est publié que s'il porte le label "traefik.enable=true".
    exposedByDefault: false
    # Réseau utilisé pour joindre les conteneurs.
    network: {{ traefik_network }}

entryPoints:
  web:
    address: ":80"
    http:
      # Toute requête HTTP est redirigée vers HTTPS.
      redirections:
        entryPoint:
          to: websecure
          scheme: https
  websecure:
    address: ":443"
    http:
      tls:
        # Tout routeur sur ce point d'entrée reçoit automatiquement un certificat
        # pour le nom de sa règle Host(...).
        certResolver: letsencrypt

certificatesResolvers:
  letsencrypt:
    acme:
      email: {{ traefik_acme_email }}
      # Emplacement des certificats : volume persistant, pour ne pas les redemander à chaque
      redémarrage.
      storage: /acme/acme.json
{% if traefik_acme_staging %}
```

```
# Serveur de test de Let's Encrypt.
caServer: https://acme-staging-v02.api.letsencrypt.org/directory
{% endif %}

# Validation par HTTP : Let's Encrypt vérifie le domaine sur le port 80.
httpChallenge:
  entryPoint: web
```

Cette configuration obtient un certificat par nom de domaine. Pour un certificat unique couvrant tous les sous-domaines (certificat « wildcard »), il faut passer par la validation DNS, qui demande les identifiants de l'API du fournisseur de noms de domaine. C'est plus pratique, mais cela ajoute un secret par client.

Le fichier Docker Compose

```
# roles/traefik/templates/docker-compose.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.
services:
  traefik:
    image: traefik:{{ traefik_version }}
    container_name: {{ client_name }}_traefik
    restart: unless-stopped
    # Interdit au processus de gagner de nouveaux privilèges.
    security_opt:
      - no-new-privileges:true
    # Seul conteneur de la VM à publier des ports vers Internet.
    ports:
      - "80:80"
      - "443:443"
    volumes:
      # Lecture de la liste des conteneurs. L'accès est en lecture seule, mais il donne
      # tout de même une vue étendue sur Docker : voir « Pour aller plus loin ».
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - {{ traefik_dir }}/traefik.yml:/etc/traefik/traefik.yml:ro
      - {{ traefik_dir }}/acme:/acme
      - {{ traefik_log_dir }}:/var/log/traefik
    networks:
      {{ traefik_network }}:
```

```

# Noms que les autres conteneurs peuvent joindre directement, sans sortir de la VM.
aliases:

{% for alias in traefik_network_aliases %}
    - {{ alias }}
{% endfor %}

networks:
  {{ traefik_network }}:
    # Réseau créé par le rôle docker-engine : Compose ne doit pas le recréer.
    external: true

```

Rotation du journal d'accès

Un journal qui grossit sans limite finit par remplir le disque. Docker fait tourner les journaux de ses conteneurs, mais pas les fichiers que Traefik écrit lui-même. Une règle `logrotate` s'en charge.

```

# roles/traefik/templates/logrotate-traefik.j2
{{ traefik_log_dir }}/access.log {
    # Un fichier par jour, sept jours conservés.
    daily
    rotate 7
    compress
    delaycompress
    missingok
    notifempty
    # Copie puis vide le fichier : Traefik continue d'écrire dans le même fichier ouvert.
    copytruncate
}

```

Tâches

```

# roles/traefik/tasks/main.yml

- name: Créer les dossiers de Traefik
  ansible.builtin.file:
    path: "{{ item.path }}"

```

```

    state: directory
    mode: "{{ item.mode }}"
loop:
  - { path: "{{ traefik_dir }}",      mode: "0755" }
  # Le dossier des certificats est réservé à root : il contient les clés privées.
  - { path: "{{ traefik_dir }}/acme", mode: "0700" }
  - { path: "{{ traefik_log_dir }}",  mode: "0755" }

- name: Déposer la configuration statique de Traefik
  ansible.builtin.template:
    src: traefik.yml.j2
    dest: "{{ traefik_dir }}/traefik.yml"
    mode: "0644"
  # Un changement de configuration statique demande un redémarrage.
  notify: Redémarrer traefik

- name: Déposer le fichier Docker Compose de Traefik
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ traefik_dir }}/docker-compose.yml"
    mode: "0644"

- name: Déposer la règle de rotation du journal d'accès
  ansible.builtin.template:
    src: logrotate-traefik.j2
    dest: /etc/logrotate.d/traefik
    mode: "0644"

- name: Démarrer Traefik
  community.docker.docker_compose_v2:
    project_src: "{{ traefik_dir }}"
    # "present" crée ou met à jour les conteneurs, sans toucher à ceux déjà conformes.
    state: present

- name: Attendre que Traefik écoute sur le port 443
  # Le module s'exécute sur la VM : "127.0.0.1" désigne donc la VM elle-même.
  ansible.builtin.wait_for:
    host: 127.0.0.1
    port: 443
    timeout: 60

```

```
# roles/traefik/handlers/main.yml
- name: Redémarrer traefik
  community.docker.docker_compose_v2:
    project_src: "{{ traefik_dir }}"
    state: restarted
```

Comment une application se déclare

Traefik n'a aucune liste d'applications : chaque conteneur se décrit lui-même avec des labels. Toutes les pages d'application suivent ce modèle, ici pour Nextcloud.

```
# Extrait d'un docker-compose.yml.j2 d'application (exemple : Nextcloud)
labels:
  # Publier ce conteneur (obligatoire, car exposedByDefault vaut false).
  - "traefik.enable=true"
  # Réseau à utiliser si le conteneur est sur plusieurs réseaux.
  - "traefik.docker.network={{ docker_proxy_network }}"
  # Règle de routage : les requêtes pour ce nom vont vers ce conteneur.
  # Le nom du routeur (nextcloud-alpha) doit être unique sur la VM.
  - "traefik.http.routers.nextcloud-{{ client_name }}.rule=Host(`{{
client_subdomains.nextcloud }}.{{ client_domain }}`)\"
  # Point d'entrée HTTPS : le certificat est demandé automatiquement.
  - "traefik.http.routers.nextcloud-{{ client_name }}.entrypoints=websecure"
  # Port d'écoute de l'application À L'INTÉRIEUR du conteneur.
  - "traefik.http.services.nextcloud-{{ client_name }}.loadbalancer.server.port=80"
```

Joindre le SSO depuis un conteneur

Une application doit parfois contacter le SSO de la VM, par exemple pour échanger un code de connexion contre un jeton. Elle le fait avec son adresse publique, `sso.penhoat.example`. Or ce nom pointe vers l'adresse publique de la VM, et un conteneur qui cherche à la joindre fait un détour par l'extérieur : le chemin fonctionne ou non selon la façon dont l'hébergeur attache l'adresse publique à la machine (adresse portée directement par la VM ou traduite par une passerelle).

Le plus fiable est d'éviter ce détour. Le rôle donne à Traefik un **alias** de réseau : sur le réseau `proxy`, le nom `sso.penhoat.example` est résolu vers le conteneur Traefik lui-même. L'application joint donc Traefik directement, sur le port 443, avec le même nom et le même certificat. Le résultat est le même pour l'utilisateur, sans dépendre de l'adresse publique.

```
# Extrait de docker-compose.yml.j2 (généré par le rôle)

networks:
  proxy:
    aliases:
      - sso.penhoat.example
```

Seul le nom du SSO a besoin de cet alias, car c'est le seul que les applications appellent entre elles. La variable `traefik_network_aliases` permet d'en ajouter d'autres si une application doit en joindre une autre.

Vérification

```
# Sur la VM : Traefik est démarré et publie les ports 80 et 443.
docker ps --filter name=traefik --format '{{.Names}}\t{{.Status}}\t{{.Ports}}'

# La redirection HTTP vers HTTPS répond "301".
curl -sI http://127.0.0.1 | head -n 1

# Un nom sans application répond "404" : Traefik écoute, mais n'a aucune route pour lui.
curl -skI https://127.0.0.1 | head -n 1

# Journal de démarrage : aucune erreur de certificat attendue.
docker logs --tail 30 alpha_traefik

# Journal d'accès : le champ ClientHost doit contenir l'adresse du visiteur, pas celle de Docker.
sudo tail -n 3 /var/log/traefik/access.log | jq '{ClientHost, RequestHost, DownstreamStatus}'
```

Une fois une application déployée, le certificat apparaît avec :

```
# Affiche l'émetteur et les dates de validité du certificat de l'application.
echo | openssl s_client -connect cloud.penhoat.example:443 -servername cloud.penhoat.example
2>/dev/null | openssl x509 -noout -issuer -dates
```

Pour aller plus loin

- **Protéger l'accès au socket Docker.** Même en lecture seule, le socket expose beaucoup d'informations sur la machine. Un intermédiaire comme `docker-socket-proxy` limite Traefik aux seules requêtes nécessaires.
- **Certificats wildcard** par validation DNS, pour ne pas dépendre de l'accès au port 80.
- **En-têtes de sécurité** (HSTS, `X-Content-Type-Options`) ajoutés par un middleware commun à toutes les applications.

Pièges

1. **Limites de Let's Encrypt** : des demandes répétées en échec bloquent temporairement un domaine. Tester avec `traefik_acme_staging: true`, puis repasser à `false` en supprimant le contenu du dossier `acme` pour obtenir les vrais certificats.
2. **DNS pas encore propagé** : le défi échoue et le journal indique que le domaine ne pointe pas vers la VM. Vérifier l'enregistrement avant de redémarrer Traefik.
3. **Application inaccessible, erreur 404** : le label `traefik.enable=true` ou la règle `Host(...)` est absente ou erronée. Le routeur n'existe pas, donc Traefik ne sait pas où envoyer la requête.
4. **Application inaccessible, erreur 502 ou 504** : Traefik trouve la route mais n'atteint pas le conteneur. Causes habituelles : mauvais port dans le label `loadbalancer.server.port`, conteneur absent du réseau `proxy`, ou conteneur sur plusieurs réseaux sans label `traefik.docker.network`.
5. **Traefik ne détecte plus aucun conteneur** : avec les versions récentes de Docker, une version trop ancienne de Traefik n'arrive plus à lire l'API Docker. Le journal mentionne une version d'API trop ancienne. Mettre à jour `traefik_version`.
6. **Adresse du visiteur remplacée par celle de la passerelle Docker** : le journal d'accès montre l'adresse interne de Docker au lieu de celle du visiteur, ce qui rend Fail2ban inefficace. Le cas se rencontre surtout avec les connexions IPv6 ou lorsqu'un intermédiaire est placé devant la VM. Le contrôle se fait avec la dernière commande de vérification, depuis une adresse extérieure.
7. **Noms de routeur en double** : deux conteneurs qui déclarent le même nom de routeur se contredisent, et seul l'un des deux fonctionne. Le nom du client dans le nom du routeur évite les collisions entre projets.
8. **Modification de `traefik.yml` sans effet** : la configuration statique n'est lue qu'au démarrage. Le handler redémarre Traefik quand le fichier change ; les labels, eux, sont pris en compte à chaud.

Page suivante : **Authentik (SSO).**

Authentik

En bref

Niveau : avancé

Durée : 45 minutes

Prérequis : Traefik en place

Niveau de test : reconstitué

Authentik, c'est quoi ?

Authentik est un fournisseur d'identité : il détient les comptes des utilisateurs et dit aux applications qui est connecté. Une application qui lui délègue la connexion n'a plus de mots de passe à gérer : l'utilisateur se connecte une fois sur Authentik, puis accède à toutes les applications de la VM. Le protocole utilisé est OpenID Connect (OIDC).

Élément	Rôle
Serveur Authentik	Interface web, API et pages de connexion
Worker Authentik	Tâches de fond (envoi d'e-mails, nettoyage, synchronisations)
PostgreSQL	Base de données d'Authentik (comptes, configuration)
Provider OIDC	Définit comment une application se connecte (identifiant, secret, adresses de retour)
Application	Objet Authentik qui rattache un provider à une application et à ses règles d'accès

Utilisateur → Application → redirection vers Authentik → connexion → retour à l'application avec un jeton

Concrètement, chaque application commandée par un client a besoin d'un provider et d'une application dans Authentik, créés avec les bons réglages. Faire ces réglages à la main dans l'interface est long et source d'erreurs. Le rôle les fait donc par l'API d'Authentik, avec un jeu de tâches réutilisable que toutes les pages d'application appellent.

Prérequis

1. Traefik est en place et publie `sso.penhoat.example` (page précédente).
2. Quatre secrets existent dans le Vault du client : `vault_authentik_secret_key`, `vault_authentik_db_password`, `vault_authentik_bootstrap_password` et `vault_authentik_bootstrap_token`. Les deux derniers complètent la liste de la page « Inventaire et variables ».
3. `curl` est installé sur la VM (fait par le socle).

Le mot de passe et le jeton d'amorçage ne sont lus qu'au tout premier démarrage d'Authentik : ils créent le compte administrateur `akadmin` et son jeton d'API. Les modifier ensuite dans le Vault ne change rien dans Authentik.

Arborescence du rôle

```
roles/authentik/  
├─ defaults/main.yml  
├─ tasks/  
│   ├─ main.yml          # déploiement, attente, découverte des objets communs  
│   ├─ oidc_app.yml      # création d'un provider et d'une application OIDC (réutilisable)  
│   └─ first_user.yml    # création du premier compte du client  
└─ templates/  
    └─ docker-compose.yml.j2
```

Variables

```
# roles/authentik/defaults/main.yml  
  
# Version d'Authentik : remplacer par la dernière version stable (voir les notes de version).  
# Depuis la version 2025.10, Redis n'est plus nécessaire. Sur une version plus ancienne,  
# un conteneur Redis doit être ajouté au fichier Docker Compose.  
authentik_version: "2025.10"  
  
# Dossier de travail sur la VM.  
authentik_dir: /opt/nebuleuse/authentik  
  
# Nom de domaine d'Authentik et adresse de son API.  
authentik_host: "{{ client_subdomains.sso }}.{{ client_domain }}"
```

```

authentik_api: "https://{{ authentik_host }}/api/v3"

# Secrets : définis dans le Vault, jamais en clair ici.
authentik_secret_key: "{{ vault_authentik_secret_key }}"
authentik_db_password: "{{ vault_authentik_db_password }}"
authentik_bootstrap_password: "{{ vault_authentik_bootstrap_password }}"
authentik_bootstrap_token: "{{ vault_authentik_bootstrap_token }}"

# Réseau Docker partagé avec Traefik.
authentik_network: "{{ docker_proxy_network }}"

# Flux d'autorisation utilisé par les applications.
# "explicit-consent" : l'utilisateur valide un écran de consentement à la première connexion.
# Il existe aussi une variante "implicit-consent", sans écran de validation.
authentik_authorization_flow_slug: default-provider-authorization-explicit-consent
authentik_invalidation_flow_slug: default-provider-invalidation-flow

# Fichier d'en-tête contenant le jeton d'API (lu par curl, jamais passé en argument).
authentik_api_header_file: /root/.authentik-api-header

# Base de toutes les commandes curl qui appellent l'API.
#   -sS          : silencieux, mais affiche les erreurs
#   -k          : n'exige pas de certificat reconnu (nécessaire en mode "staging")
#   --fail-with-body : échoue sur une erreur HTTP, en affichant la réponse du serveur
#   --resolve     : résout le nom d'Authentik vers la VM elle-même, sans passer par le
DNS
#   -H @fichier   : lit l'en-tête d'authentification dans un fichier
authentik_curl_base:
  - curl
  - -sS
  - -k
  - --fail-with-body
  - --resolve
  - "{{ authentik_host }}:443:127.0.0.1"
  - -H
  - "@{{ authentik_api_header_file }}"

```

Le fichier Docker Compose

```

# roles/authentik/templates/docker-compose.yml.j2
# Fichier g  r   par Ansible : toute modification manuelle sera   cras  e.
# Il contient des secrets : acc  s r  serv      root.

# Variables communes au serveur et au worker (ancree YAML r  utilis  e plus bas).
x-authentik-env: &authentik_env
  AUTHENTIK_SECRET_KEY: "{{ authentik_secret_key }}"
  AUTHENTIK_POSTGRESQL__HOST: db
  AUTHENTIK_POSTGRESQL__USER: authentik
  AUTHENTIK_POSTGRESQL__NAME: authentik
  AUTHENTIK_POSTGRESQL__PASSWORD: "{{ authentik_db_password }}"
  # Lus uniquement au premier d  marrage : cr  ent le compte "akadmin" et son jeton d'API.
  AUTHENTIK_BOOTSTRAP_PASSWORD: "{{ authentik_bootstrap_password }}"
  AUTHENTIK_BOOTSTRAP_TOKEN: "{{ authentik_bootstrap_token }}"
  AUTHENTIK_BOOTSTRAP_EMAIL: "{{ client_admin_email }}"

services:
  server:
    image: ghcr.io/goauthentik/server:{{ authentik_version }}
    container_name: {{ client_name }}_authentik_server
    restart: unless-stopped
    command: server
    environment: *authentik_env
    volumes:
      - {{ authentik_dir }}/media:/media
      - {{ authentik_dir }}/custom-templates:/templates
    depends_on:
      db:
        condition: service_healthy
    networks:
      - {{ authentik_network }}
      - internal
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network={{ authentik_network }}"
      - "traefik.http.routers.authentik-{{ client_name }}.rule=Host(`{{ authentik_host }}`)"
      - "traefik.http.routers.authentik-{{ client_name }}.entrypoints=websecure"
  # Port HTTP interne d'Authentik.
  - "traefik.http.services.authentik-{{ client_name }}.loadbalancer.server.port=9000"

```

```
worker:
  image: ghcr.io/goauthentik/server:{{ authentik_version }}
  container_name: {{ client_name }}_authentik_worker
  restart: unless-stopped
  command: worker
  environment: *authentik_env
  volumes:
    - {{ authentik_dir }}/media:/media
    - {{ authentik_dir }}/custom-templates:/templates
  depends_on:
    db:
      condition: service_healthy
# Le worker n'a pas besoin d'être joignable : réseau interne uniquement.
networks:
  - internal
```

```
db:
  image: postgres:16-alpine
  container_name: {{ client_name }}_authentik_db
  restart: unless-stopped
  environment:
    POSTGRES_USER: authentik
    POSTGRES_DB: authentik
    POSTGRES_PASSWORD: "{{ authentik_db_password }}"
  volumes:
    - {{ authentik_dir }}/db-data:/var/lib/postgresql/data
# Les autres services attendent que la base soit prête avant de démarrer.
healthcheck:
  test: ["CMD-SHELL", "pg_isready -d authentik -U authentik"]
  interval: 5s
  timeout: 5s
  retries: 10
# La base n'est jamais exposée sur le réseau partagé avec Traefik.
networks:
  - internal
```

```
networks:
  {{ authentik_network }}:
    external: true
  internal: {}
```

Seul le serveur est sur le réseau partagé `proxy`. La base de données et le worker restent sur un réseau privé au projet : aucune autre application de la VM ne peut atteindre la base d'Authentik.

Déploiement et découverte des objets communs

La première partie du rôle déploie les conteneurs, attend que l'API réponde, puis retrouve les objets d'Authentik dont tous les providers ont besoin : les deux flux, les mappings de scope, la clé de signature et la règle d'accès.

```
# roles/authentik/tasks/main.yml

- name: Créer les dossiers d'Authentik
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory
    mode: "0755"
  loop:
    - "{{ authentik_dir }}"
    - "{{ authentik_dir }}/media"
    - "{{ authentik_dir }}/custom-templates"
    - "{{ authentik_dir }}/db-data"

- name: Déposer le fichier Docker Compose d'Authentik
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ authentik_dir }}/docker-compose.yml"
    # Le fichier contient des secrets : root uniquement.
    mode: "0600"
  no_log: true

- name: Démarrer Authentik
  community.docker.docker_compose_v2:
    project_src: "{{ authentik_dir }}"
    state: present
```

Le premier démarrage applique les migrations de la base : cela peut durer plusieurs minutes.

- name: Attendre que l'API d'Authentik soit prête

ansible.builtin.command:

argv:

- curl
- -fsS
- -k
- --resolve
- "{{ authentik_host }}:443:127.0.0.1"
- "https://{{ authentik_host }}/-/health/live/"

register: authentik_ready

until: authentik_ready.rc == 0

retries: 60

delay: 5

changed_when: false

Le jeton est écrit dans un fichier d'en-tête plutôt que passé en argument à curl :

il n'apparaît ni dans la liste des processus, ni dans les journaux d'Ansible.

- name: Déposer le fichier d'en-tête d'authentification pour l'API

ansible.builtin.copy:

dest: "{{ authentik_api_header_file }}"

content: "Authorization: Bearer {{ authentik_bootstrap_token }}\n"

owner: root

group: root

mode: "0600"

no_log: true

--- Découverte des objets communs à tous les providers ---

- name: Trouver le flux d'autorisation

ansible.builtin.command:

argv: "{{ authentik_curl_base + [authentik_api ~ '/flows/instances/?slug=' ~
authentik_authorization_flow_slug] }}"

register: authentik_authz_flow_raw

changed_when: false

- name: Trouver le flux d'invalidation

ansible.builtin.command:

argv: "{{ authentik_curl_base + [authentik_api ~ '/flows/instances/?slug=' ~
authentik_invalidation_flow_slug] }}"

```

register: authentik_invalidation_flow_raw
changed_when: false

- name: Lister les mappings de scope
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~
'/propertymappings/provider/scope/?page_size=200'] }}"
  register: authentik_scope_mappings_raw
  changed_when: false

- name: Trouver la clé de signature par défaut
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~
'/crypto/certificatekeypairs/?name=authentik%20Self-signed%20Certificate'] }}"
  register: authentik_signing_key_raw
  changed_when: false
  no_log: true          # la réponse peut contenir des éléments de la clé

- name: Retenir les identifiants des objets trouvés
  ansible.builtin.set_fact:
    authentik_authz_flow_pk: "{{ (authentik_authz_flow_raw.stdout | from_json).results[0].pk
}}"
    authentik_invalidation_flow_pk: "{{ (authentik_invalidation_flow_raw.stdout |
from_json).results[0].pk }}"
    # Clé qui signe les jetons : sans elle, Authentik signe avec le secret client (HS256).
    authentik_signing_key_pk: "{{ (authentik_signing_key_raw.stdout | from_json).results[0].pk
}}"
    # Les trois mappings standard : sans eux, l'application reçoit un jeton sans identité.
    authentik_scope_mapping_pks: >-
      {{ (authentik_scope_mappings_raw.stdout | from_json).results
        | selectattr('scope_name', 'in', ['openid', 'profile', 'email'])
        | map(attribute='pk') | list }}

# --- Règle d'accès : "toute personne connectée peut utiliser l'application" ---

- name: Chercher la règle d'accès Nébuleuse
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~ '/policies/expression/?name=nebuleuse-
allow-all'] }}"
  register: authentik_policy_raw

```

```

changed_when: false

- name: Créer la règle d'accès si elle est absente
  ansible.builtin.command:
    argv: >-
      {{ authentik_curl_base + ['-X', 'POST', '-H', 'Content-Type: application/json',
        '--data-binary', ({'name': 'nebuleuse-allow-all', 'expression': 'return True'}) |
to_json),
        authentik_api ~ '/policies/expression/' ] }}
  register: authentik_policy_created
  when: (authentik_policy_raw.stdout | from_json).results | length == 0

- name: Retenir l'identifiant de la règle d'accès
  ansible.builtin.set_fact:
    authentik_allow_policy_pk: >-
      {{ (authentik_policy_created.stdout | from_json).pk
        if authentik_policy_created is not skipped
        else (authentik_policy_raw.stdout | from_json).results[0].pk }}

```

Les commandes `curl` sont lancées avec `argv` (une liste d'arguments) plutôt qu'avec une chaîne : il n'y a ainsi aucun guillemet à échapper dans le JSON. Dans ce déploiement, le module `ansible.builtin.uri` n'a pas été retenu : `curl` s'est montré plus prévisible, notamment avec `--resolve` et les certificats de test.

Créer une connexion OIDC pour une application

Ce fichier de tâches est le cœur de l'intégration. Il est appelé une fois par application, avec trois paramètres : le nom, l'identifiant court (le « slug ») et les adresses de retour autorisées. Il crée le provider, l'application, lie la règle d'accès, puis retourne l'identifiant client, le secret et l'adresse de l'émetteur.

```

# roles/authentik/tasks/oidc_app.yml
#
# Paramètres attendus :
#   oidc_app_name      : nom affiché dans Authentik (ex. "Nextcloud")
#   oidc_app_slug      : identifiant court, utilisé dans les adresses (ex. "nextcloud")

```

```

#   oidc_redirect_uris : liste d'adresses de retour autorisées
#
#   [{ matching_mode: strict, url: "https://..." }]
#   oidc_sub_mode      : (facultatif) identifiant d'utilisateur transmis à l'application
#
# Résultats :
#   oidc_client_id, oidc_client_secret, oidc_issuer

# 1. Le provider : décrit comment l'application se connecte.
- name: "{{ oidc_app_name }}" : chercher le provider"
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~ '/providers/oauth2/?name=' ~
oidc_app_slug] }}"
  register: oidc_provider_lookup
  changed_when: false
  no_log: true          # la réponse contient le secret client

- name: "{{ oidc_app_name }}" : créer le provider s'il est absent"
  ansible.builtin.command:
    argv: >-
    {{ authentik_curl_base + ['-X', 'POST', '-H', 'Content-Type: application/json',
      '--data-binary', (oidc_provider_payload | to_json),
      authentik_api ~ '/providers/oauth2/'] }}
  vars:
    oidc_provider_payload:
      name: "{{ oidc_app_slug }}"
      # Flux d'AUTORISATION : pas le flux d'authentification, sinon l'utilisateur boucle sur
la connexion.
      authorization_flow: "{{ authentik_authz_flow_pk }}"
      invalidation_flow: "{{ authentik_invalidation_flow_pk }}"
      # "confidential" : l'application garde un secret côté serveur.
      client_type: confidential
      # Signature des jetons en RS256, attendue par la plupart des applications (Immich, par
exemple).
      signing_key: "{{ authentik_signing_key_pk }}"
      # Identifiant transmis à l'application pour désigner l'utilisateur (voir la section
Pièges).
      sub_mode: "{{ oidc_sub_mode | default('hashed_user_id') }}"
      # Adresses de retour : "strict" exige une correspondance exacte, caractère pour
caractère.
      redirect_uris: "{{ oidc_redirect_uris }}"

```

```

    # Les mappings openid, profile et email : sans eux, "scopes demandés non configurés".
    property_mappings: "{{ authentik_scope_mapping_pks }}"
register: oidc_provider_created
when: (oidc_provider_lookup.stdout | from_json).results | length == 0
no_log: true

- name: "{{ oidc_app_name }}" : retenir le provider"
  ansible.builtin.set_fact:
    oidc_provider: >-
    {{ (oidc_provider_created.stdout | from_json)
      if oidc_provider_created is not skipped
      else (oidc_provider_lookup.stdout | from_json).results[0] }}
  no_log: true

# 2. L'application : rattache le provider à un nom et à des règles d'accès.
- name: "{{ oidc_app_name }}" : chercher l'application"
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~ '/core/applications/?slug=' ~
oidc_app_slug] }}"
  register: oidc_app_lookup
  changed_when: false

- name: "{{ oidc_app_name }}" : créer l'application si elle est absente"
  ansible.builtin.command:
    argv: >-
    {{ authentik_curl_base + ['-X', 'POST', '-H', 'Content-Type: application/json',
      '--data-binary', (oidc_app_payload | to_json),
      authentik_api ~ '/core/applications/'] }}
  vars:
    oidc_app_payload:
      name: "{{ oidc_app_name }}"
      slug: "{{ oidc_app_slug }}"
      provider: "{{ oidc_provider.pk }}"
      policy_engine_mode: any
  register: oidc_app_created
  when: (oidc_app_lookup.stdout | from_json).results | length == 0

- name: "{{ oidc_app_name }}" : retenir l'application"
  ansible.builtin.set_fact:
    oidc_app_pk: >-

```

```

    {{ (oidc_app_created.stdout | from_json).pk
      if oidc_app_created is not skipped
      else (oidc_app_lookup.stdout | from_json).results[0].pk }}

# 3. La règle d'accès : sans elle, la connexion se termine par "Accès refusé".
# Elle n'est liée qu'à la création de l'application, pour ne pas multiplier les liens.
- name: "{{ oidc_app_name }}" : lier la règle d'accès à l'application"
  ansible.builtin.command:
    argv: >-
      {{ authentik_curl_base + ['-X', 'POST', '-H', 'Content-Type: application/json',
        '--data-binary', (oidc_binding_payload | to_json),
        authentik_api ~ '/policies/bindings/' ] }}
  vars:
    oidc_binding_payload:
      target: "{{ oidc_app_pk }}"
      policy: "{{ authentik_allow_policy_pk }}"
      order: 0
  when: oidc_app_created is not skipped

# 4. Les valeurs que l'application doit connaître.
- name: "{{ oidc_app_name }}" : retenir les valeurs de connexion"
  ansible.builtin.set_fact:
    oidc_client_id: "{{ oidc_provider.client_id }}"
    oidc_client_secret: "{{ oidc_provider.client_secret }}"
    # Adresse de l'émetteur : l'application y trouve la découverte automatique de la
configuration.
    oidc_issuer: "https://{{ authentik_host }}/application/o/{{ oidc_app_slug }}/"
  no_log: true

```

Une page d'application appelle ce fichier de tâches ainsi :

```

# roles/nextcloud/tasks/main.yml (extrait)
- name: Créer la connexion SSO de Nextcloud
  ansible.builtin.include_tasks: ../../authentik/tasks/oidc_app.yml
  vars:
    oidc_app_name: Nextcloud
    oidc_app_slug: nextcloud
    oidc_redirect_uris:
      - matching_mode: strict
        url: "https://{{ client_subdomains.nextcloud }}.{{ client_domain

```

```
}}/apps/user_oidc/code"
```

Les adresses OIDC dont l'application a besoin sont toutes construites à partir de l'adresse de l'émetteur :

Usage	Adresse
Découverte automatique	<code>https://sso.penhoat.example/application/o/<slug>/.well-known/openid-configuration</code>
Autorisation	<code>https://sso.penhoat.example/application/o/authorize/</code>
Jeton	<code>https://sso.penhoat.example/application/o/token/</code>
Informations sur l'utilisateur	<code>https://sso.penhoat.example/application/o/userinfo/</code>
Déconnexion	<code>https://sso.penhoat.example/application/o/<slug>/end-session/</code>

Le premier compte du client

Le compte `akadmin` sert à l'exploitation par Nébuleuse. Le client a besoin de son propre compte, créé à partir de `client_admin_email`. Son mot de passe n'est volontairement pas défini ici : il est choisi par le client, par un lien de réinitialisation envoyé par le support.

```
# roles/authentik/tasks/first_user.yml

- name: Chercher le compte du client
  ansible.builtin.command:
    argv: "{{ authentik_curl_base + [authentik_api ~ '/core/users/?username=' ~
client_admin_email] }}"
  register: authentik_user_lookup
  changed_when: false

- name: Créer le compte du client s'il est absent
  ansible.builtin.command:
    argv: >-
    {{ authentik_curl_base + ['-X', 'POST', '-H', 'Content-Type: application/json',
    '--data-binary', (authentik_user_payload | to_json),
    authentik_api ~ '/core/users/'] }}
  vars:
    authentik_user_payload:
      username: "{{ client_admin_email }}"
      name: "Administrateur {{ client_name }}"
```

```
email: "{{ client_admin_email }}"
is_active: true
when: (authentik_user_lookup.stdout | from_json).results | length == 0
```

Dans `main.yml`, ce fichier est inclus à la fin du rôle :

```
# roles/authentik/tasks/main.yml (fin du fichier)
- name: Créer le premier compte du client
  ansible.builtin.include_tasks: first_user.yml
```

Vérification

```
# Sur la VM : les trois conteneurs sont démarrés et la base est « healthy ».
docker ps --filter name=authentik --format '{{.Names}}\t{{.Status}}'

# L'API répond et reconnaît le jeton : doit afficher l'utilisateur "akadmin".
curl -sSk --resolve sso.penhoat.example:443:127.0.0.1 \
  -H @/root/.authentik-api-header \
  https://sso.penhoat.example/api/v3/core/users/me/ | jq '.user.username'

# Une connexion OIDC existe et la découverte automatique répond (exemple : Nextcloud).
curl -sSk --resolve sso.penhoat.example:443:127.0.0.1 \
  https://sso.penhoat.example/application/o/nextcloud/.well-known/openid-configuration | jq
'.issuer'
```

Dans l'interface, on se connecte sur `https://sso.penhoat.example/if/admin/` avec `akadmin` et le mot de passe d'amorçage. Les rubriques **Applications** et **Providers** doivent lister chaque application commandée.

Pièges

1. **Flux d'authentification à la place du flux d'autorisation** : l'utilisateur se reconnecte en boucle au lieu d'être renvoyé vers l'application. Le provider doit utiliser un flux dont le slug commence par `default-provider-authorization-`.
2. **Mappings de scope absents** : l'erreur « Application requested scopes not configured » apparaît. Les trois mappings `openid`, `profile` et `email` doivent être attachés au provider.

3. « **Accès refusé** » **après une connexion réussie** : dans le déploiement qui sert de base à ce TP, une application sans règle liée refusait l'accès. La règle `return True` est liée à chaque application pour l'autoriser à toute personne connectée.
4. **sub_mode inadapté** : c'est l'identifiant de l'utilisateur transmis à l'application. La valeur par défaut `hashed_user_id` convient à la plupart des applications. Une application qui attend un UUID (PaperMerge, par exemple) échoue avec une erreur de format : utiliser `user_uuid`. Les valeurs possibles sont `hashed_user_id`, `user_id`, `user_uuid`, `user_username`, `user_email` et `user_upn`.
5. **Utilisateur introuvable côté application** : certaines applications font correspondre l'utilisateur d'Authentik à un compte qu'elles ont déjà. Le nom d'utilisateur et l'adresse e-mail doivent alors être identiques de part et d'autre, et la connexion doit se faire avec ce compte, pas avec `akadmin`.
6. **Adresse de retour différente** : l'adresse déclarée dans Authentik et celle utilisée par l'application doivent être identiques, y compris pour un `/` final. L'erreur affichée mentionne une URI de redirection invalide.
7. **Compter avec `.count`** : la liste des résultats d'Authentik est dans `.results`, et le nombre d'éléments dans `.pagination.count`. Un `.count` direct provoque l'erreur « 'dict object' has no attribute 'count' ». Le rôle compte avec `.results | length`.
8. **Jeton visible dans la liste des processus ou dans les journaux** : le passer en argument de `curl` l'expose. D'où le fichier d'en-tête en mode `0600` et les `no_log: true`.
9. **Modifier un provider existant** : le rôle ne met pas à jour un provider déjà créé. Si les adresses de retour changent, il faut les modifier dans l'interface, ou supprimer le provider pour qu'il soit recréé.
10. **Premier démarrage long** : les migrations de la base peuvent prendre plusieurs minutes. Le rôle attend jusqu'à cinq minutes avant d'abandonner.
11. **Versión de Redis** : sur une version d'Authentik antérieure à 2025.10, le serveur et le worker exigent un conteneur Redis, absent du fichier Compose de cette page.
12. **Jetons signés avec le secret client (HS256)** : sans clé de signature sur le provider, Authentik signe avec le secret client. Une application qui attend du RS256 refuse alors le jeton. Le rôle associe donc la clé « authentik Self-signed Certificate » à chaque provider.

Pour aller plus loin

- **Remplacer le jeton d'amorçage** par un compte de service dédié, aux droits limités à la gestion des providers, puis supprimer le jeton d'amorçage une fois le déploiement terminé.
- **Configurer l'envoi d'e-mails** (SMTP) pour que les liens de réinitialisation et les notifications partent directement d'Authentik.
- **Déclarer la configuration par « blueprints »** : Authentik sait charger des fichiers de description, une alternative à l'API qui offre une vraie mise à jour déclarative.

Page suivante : **Nextcloud**.

Nextcloud

En bref**Niveau** : intermédiaire**Durée** : 45 minutes**Prérequis** : Traefik et Authentik en place (pages précédentes)**Niveau de test** : reconstitué

Nextcloud, c'est quoi ?

Nextcloud est une plateforme de fichiers, d'agenda et de contacts auto-hébergée : l'équivalent libre d'un espace de stockage en ligne, avec synchronisation sur ordinateur et téléphone. C'est l'application phare du catalogue, et celle qui sert ici de modèle aux autres : un fichier Docker Compose, des labels Traefik, une connexion SSO créée par le rôle Authentik.

Élément	Rôle
Conteneur Nextcloud	Serveur web (Apache et PHP) : interface, API, synchronisation
Conteneur cron	Même image, exécute les tâches de fond toutes les cinq minutes
PostgreSQL	Base de données (comptes, partages, métadonnées)
Redis	Cache et verrouillage des fichiers
Application <code>user_oidc</code>	Module Nextcloud qui délègue la connexion à Authentik

Utilisateur → Traefik → Nextcloud → redirection vers Authentik → retour connecté

Concrètement, le rôle fait cinq choses dans l'ordre : déposer le fichier Compose, démarrer les conteneurs, attendre la fin de l'installation automatique, créer la connexion SSO dans Authentik, puis déclarer cette connexion dans Nextcloud avec la commande `occ`.

Cette page traite le déploiement automatisé d'une instance Nextcloud par Nébuleuse. L'installation manuelle et les commandes `occ` d'usage courant sont décrites dans le livre **Nextcloud** de la base de connaissance.

Prérequis

1. Traefik publie `cloud.penhoat.example` (variable `client_subdomains.nextcloud`) et Authentik répond sur `sso.penhoat.example`.
2. `nextcloud` figure dans `nebuleuse_apps` et dans `nebuleuse_catalogue`.
3. Deux secrets existent dans le Vault du client : `vault_nextcloud_db_password` et `vault_nextcloud_admin_password` (page « Inventaire et variables »).

Arborescence du rôle

```
roles/nextcloud/  
├─ defaults/main.yml  
├─ tasks/main.yml  
└─ templates/  
    └─ docker-compose.yml.j2
```

Variables

```
# roles/nextcloud/defaults/main.yml  
  
# Version majeure de Nextcloud : remplacer par la dernière version stable.  
# Nextcloud ne permet pas de sauter une version majeure lors d'une mise à jour.  
nextcloud_version: "32"  
  
# Dossier de travail sur la VM.  
nextcloud_dir: /opt/nebuleuse/nextcloud  
  
# Nom de domaine de l'application.  
nextcloud_host: "{{ client_subdomains.nextcloud }}.{{ client_domain }}"  
  
# Secrets : définis dans le Vault, jamais en clair ici.  
nextcloud_db_password: "{{ vault_nextcloud_db_password }}"  
nextcloud_admin_password: "{{ vault_nextcloud_admin_password }}"  
  
# Compte administrateur local, réservé à l'exploitation. Les utilisateurs du client
```

```
# se connectent par le SSO.
nextcloud_admin_user: nebuleuse-admin

# Réseaux d'où Nextcloud accepte l'en-tête X-Forwarded-For. Ici : les réseaux privés de
# Docker,
# où seul Traefik peut joindre l'application.
nextcloud_trusted_proxies: "172.16.0.0/12"

# Limites PHP : taille maximale d'un fichier envoyé et mémoire d'un processus.
nextcloud_upload_limit: 10G
nextcloud_memory_limit: 1G

# Réseau Docker partagé avec Traefik.
nextcloud_network: "{{ docker_proxy_network }}"

# Nom du fournisseur OIDC déclaré dans Nextcloud (apparaît sur le bouton de connexion).
nextcloud_oidc_provider_name: Authentik
```

Le fichier Docker Compose

```
# roles/nextcloud/templates/docker-compose.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.
# Il contient des secrets : accès réservé à root.

# Variables communes au serveur et au conteneur cron (ancrage YAML réutilisée plus bas).
x-nextcloud-env: &nextcloud_env

# Base de données.
POSTGRES_HOST: db
POSTGRES_DB: nextcloud
POSTGRES_USER: nextcloud
POSTGRES_PASSWORD: "{{ nextcloud_db_password }}"

# Cache et verrouillage.
REDIS_HOST: redis

# Lues uniquement à la première installation : créent le compte administrateur local.
NEXTCLOUD_ADMIN_USER: "{{ nextcloud_admin_user }}"
NEXTCLOUD_ADMIN_PASSWORD: "{{ nextcloud_admin_password }}"
NEXTCLOUD_TRUSTED_DOMAINS: "{{ nextcloud_host }}"

# Nextcloud est derrière Traefik : il doit construire ses adresses en HTTPS
```

```

# et faire confiance à l'en-tête X-Forwarded-For envoyé par le proxy.
OVERWRITEPROTOCOL: https
OVERWRITEHOST: "{{ nextcloud_host }}"
OVERWRITECLIURL: "https://{{ nextcloud_host }}"
TRUSTED_PROXIES: "{{ nextcloud_trusted_proxies }}"
PHP_UPLOAD_LIMIT: "{{ nextcloud_upload_limit }}"
PHP_MEMORY_LIMIT: "{{ nextcloud_memory_limit }}"

services:
  app:
    image: nextcloud:{{ nextcloud_version }}-apache
    container_name: {{ client_name }}_nextcloud
    restart: unless-stopped
    environment: *nextcloud_env
    volumes:
      # Code, configuration et données des utilisateurs.
      - {{ nextcloud_dir }}/html:/var/www/html
    depends_on:
      - db
      - redis
    networks:
      - {{ nextcloud_network }}
      - internal
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network={{ nextcloud_network }}"
      - "traefik.http.routers.nextcloud-{{ client_name }}.rule=Host(`{{ nextcloud_host }}`)"
      - "traefik.http.routers.nextcloud-{{ client_name }}.entrypoints=websecure"
      - "traefik.http.services.nextcloud-{{ client_name }}.loadbalancer.server.port=80"
      # Redirige les adresses de découverte CalDAV et CardDAV vers le service DAV de
      Nextcloud.
      # "$$" : un "$" littéral dans un fichier Compose.
      - "traefik.http.routers.nextcloud-{{ client_name }}.middlewares=nextcloud-dav-{{
client_name }}"
      - "traefik.http.middlewares.nextcloud-dav-{{ client_name
}}.redirectregex.regex=https://(.*)/.well-known/(?:(card|cal)dav)"
      - "traefik.http.middlewares.nextcloud-dav-{{ client_name
}}.redirectregex.replacement=https://${1}/remote.php/dav"
      - "traefik.http.middlewares.nextcloud-dav-{{ client_name
}}.redirectregex.permanent=true"

```

cron:

```
image: nextcloud:{{ nextcloud_version }}-apache
container_name: {{ client_name }}_nextcloud_cron
restart: unless-stopped
# Script fourni par l'image : lance "cron.php" toutes les cinq minutes.
entrypoint: /cron.sh
environment: *nextcloud_env
volumes:
  - {{ nextcloud_dir }}/html:/var/www/html
depends_on:
  - db
  - redis
# Pas besoin d'être joignable : réseau interne uniquement.
networks:
  - internal
```

db:

```
image: postgres:16-alpine
container_name: {{ client_name }}_nextcloud_db
restart: unless-stopped
environment:
  POSTGRES_USER: nextcloud
  POSTGRES_DB: nextcloud
  POSTGRES_PASSWORD: "{{ nextcloud_db_password }}"
volumes:
  - {{ nextcloud_dir }}/db-data:/var/lib/postgresql/data
networks:
  - internal
```

redis:

```
image: redis:7-alpine
container_name: {{ client_name }}_nextcloud_redis
restart: unless-stopped
networks:
  - internal
```

networks:

```
{{ nextcloud_network }}:
  external: true
```

```
internal: {}
```

Seul le serveur Nextcloud est sur le réseau `proxy`. La base, Redis et le conteneur cron restent sur un réseau privé au projet. Redis n'a donc pas de mot de passe : aucun autre conteneur de la VM ne peut l'atteindre.

Déploiement

```
# roles/nextcloud/tasks/main.yml

- name: Créer les dossiers de Nextcloud
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory
    mode: "0755"
  loop:
    - "{{ nextcloud_dir }}"
    - "{{ nextcloud_dir }}/db-data"
  # Le dossier "html" est créé par Docker : il appartient ensuite à l'utilisateur www-data
  # du conteneur (UID 33). Ne pas le modifier avec chown.

- name: Déposer le fichier Docker Compose de Nextcloud
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ nextcloud_dir }}/docker-compose.yml"
    # Le fichier contient des secrets : root uniquement.
    mode: "0600"
  no_log: true

- name: Démarrer Nextcloud
  community.docker.docker_compose_v2:
    project_src: "{{ nextcloud_dir }}"
    state: present

# Au premier démarrage, l'image installe Nextcloud toute seule (plusieurs minutes).
# "occ status" ne renvoie "installed": true qu'une fois l'installation terminée.
- name: Attendre la fin de l'installation de Nextcloud
```

```

ansible.builtin.command:
  argv:
    - docker
    - exec
    - --user
    - www-data
    - "{{ client_name }}_nextcloud"
    - php
    - occ
    - status
    - --output=json
  register: nextcloud_status
# On retire les espaces pour ne pas dépendre de la mise en forme du JSON.
until: >-
  nextcloud_status.rc == 0 and
  '"installed":true' in (nextcloud_status.stdout | replace(' ', ''))
retries: 60
delay: 10
changed_when: false

# Réglages complémentaires, rejouables sans effet de bord.
- name: Régler le fonctionnement de Nextcloud
  ansible.builtin.command:
    argv: >-
      {{ ['docker', 'exec', '--user', 'www-data', client_name ~ '_nextcloud',
        'php', 'occ', 'config:system:set'] + item }}
  loop:
    # Région par défaut des numéros de téléphone.
    - ['default_phone_region', '--value=FR']
    # Fenêtre de maintenance : les tâches lourdes tournent à partir de 1 h (UTC).
    - ['maintenance_window_start', '--type=integer', '--value=1']
  changed_when: false

- name: Utiliser le conteneur cron pour les tâches de fond
  ansible.builtin.command:
    argv: [docker, exec, --user, www-data, "{{ client_name }}_nextcloud", php, occ,
background:cron]
  changed_when: false

```

Connexion SSO

Cette partie du rôle appelle le fichier de tâches de la page Authentik, puis déclare le résultat dans Nextcloud. L'adresse de retour `/apps/user_oidc/code` est celle de l'application `user_oidc`.

```
# roles/nextcloud/tasks/main.yml (suite)

- name: Créer la connexion SSO de Nextcloud dans Authentik
  ansible.builtin.include_tasks: ../../authentik/tasks/oidc_app.yml
  vars:
    oidc_app_name: Nextcloud
    oidc_app_slug: nextcloud
    oidc_redirect_uris:
      - matching_mode: strict
        url: "https://{{ nextcloud_host }}/apps/user_oidc/code"

- name: Lister les applications Nextcloud installées
  ansible.builtin.command:
    argv: [docker, exec, --user, www-data, "{{ client_name }}_nextcloud", php, occ, app:list,
--output=json]
  register: nextcloud_apps_list
  changed_when: false

- name: Installer l'application user_oidc si elle est absente
  ansible.builtin.command:
    argv: [docker, exec, --user, www-data, "{{ client_name }}_nextcloud", php, occ,
app:install, user_oidc]
  when: "'user_oidc' not in nextcloud_apps_list.stdout"

# La commande crée le fournisseur, ou le met à jour s'il existe déjà.
# --unique-uid=0      : l'identifiant Nextcloud est le nom d'utilisateur, pas un hachage.
# --mapping-uid=...  : champ du jeton utilisé comme identifiant.

- name: Déclarer Authentik comme fournisseur de connexion
  ansible.builtin.command:
    argv:
      - docker
      - exec
      - --user
```

```
- www-data
- "{{ client_name }}"_nextcloud"
- php
- occ
- user_oidc:provider
- "{{ nextcloud_oidc_provider_name }}"
- "--clientid={{ oidc_client_id }}"
- "--clientsecret={{ oidc_client_secret }}"
- "--discoveryuri={{ oidc_issuer }}.well-known/openid-configuration"
- "--unique-uid=0"
- "--mapping-uid=preferred_username"
- "--scope=openid email profile"

changed_when: false
no_log: true
```

Vérification

```
# Sur la VM : quatre conteneurs "Up".
docker ps --filter name=nextcloud --format '{{.Names}}\t{{.Status}}'

# Nextcloud est installé et en version attendue.
docker exec --user www-data alpha_nextcloud php occ status

# L'adresse publique répond et annonce une instance installée.
curl -sS https://cloud.penhoat.example/status.php | jq '{installed, maintenance, version}'

# Le fournisseur OIDC est déclaré.
docker exec --user www-data alpha_nextcloud php occ user_oidc:provider

# La redirection CalDAV fonctionne : l'adresse "Location" doit finir par /remote.php/dav.
curl -sSI https://cloud.penhoat.example/.well-known/caldav | grep -i '^location'
```

Dans le navigateur, la page `https://cloud.penhoat.example` doit proposer le bouton « Authentik ». Après connexion, l'espace fichiers du compte s'affiche. Le compte local reste accessible par `https://cloud.penhoat.example/login?direct=1`.

Pièges

1. **Variables d'installation sans effet après coup** : `NEXTCLOUD_ADMIN_*`, les paramètres de base et les domaines approuvés ne sont lus qu'à la première installation. Pour les changer ensuite, il faut passer par `occ` (par exemple `config:system:set trusted_domains 1 --value=...`).
2. « **Accès via un domaine non approuvé** » : le nom utilisé n'est pas dans `trusted_domains`. Tous les noms par lesquels l'application est atteinte doivent y figurer.
3. **Adresses en `http://` ou boucle de redirection** : `OVERWRITEPROTOCOL`, `OVERWRITEHOST` ou `TRUSTED_PROXIES` manquent. Les journaux d'accès ne montrent alors que l'adresse de Traefik.
4. **Utilisateurs créés avec un identifiant haché** : sans `--unique-uid=0`, Nextcloud nomme les comptes d'après une empreinte, illisible dans les partages et les journaux.
5. **Erreur sur l'adresse de retour** : l'adresse déclarée dans Authentik doit être exactement `https://<domaine>/apps/user_oidc/code`.
6. **Premier utilisateur SSO sans droits d'administration** : le compte du client est créé à sa première connexion, comme simple utilisateur. Pour le promouvoir : `occ group:adduser admin <identifiant>`.
7. **Tâches de fond jamais exécutées** : sans conteneur cron, le mode « AJAX » ne s'exécute que lorsqu'un utilisateur ouvre l'interface. Le conteneur cron et la commande `background:cron` règlent ce point.
8. **Gros fichiers refusés** : Traefik ne limite pas la taille, mais PHP oui. Les variables `PHP_UPLOAD_LIMIT` et `PHP_MEMORY_LIMIT` fixent les limites ; les clients de synchronisation découpent déjà les envois.
9. **Mise à jour de version** : Nextcloud n'accepte de monter que d'une version majeure à la fois. Passer de 31 à 33 impose de passer par 32.
10. **Secret client visible brièvement** : `--clientsecret` est passé en argument de `docker exec` et apparaît pendant quelques secondes dans la liste des processus de la VM, accessible à root seulement. Avec `no_log: true`, il n'apparaît pas dans les journaux d'Ansible.
11. **Nextcloud ne joint pas le SSO** : l'erreur survient à l'échange du code de connexion. Le mécanisme d'alias réseau de Traefik, décrit dans sa page, évite ce détour par l'adresse publique.

Page suivante : **Immich**.

Immich

En bref

Niveau : avancé

Durée : 45 minutes

Prérequis : Traefik et Authentik en place ; 4 Go de mémoire sur la VM

Niveau de test : reconstitué

Immich, c'est quoi ?

Immich est une galerie de photos et de vidéos auto-hébergée : sauvegarde automatique depuis le téléphone, albums partagés, recherche par visage, par lieu ou par contenu. C'est l'alternative libre aux services de photos en ligne. Il se compose de plusieurs conteneurs qui travaillent ensemble.

Élément	Rôle
Serveur Immich	Interface web, API, envoi et traitement des fichiers
Machine learning	Reconnaissance de visages et recherche par contenu
PostgreSQL (image Immich)	Base de données, avec les extensions de recherche vectorielle
Redis (Valkey)	File des tâches de fond et cache
Fichier <code>immich.json</code>	Configuration déclarative, dont la connexion SSO

Téléphone ou navigateur → Traefik → Serveur Immich → Machine learning, PostgreSQL, Redis

Concrètement, la page suit le même modèle que Nextcloud : un fichier Compose et des labels Traefik. Deux différences. Le SSO est déclaré dans un fichier de configuration lu au démarrage, et non par une commande après coup. Et le premier compte administrateur est créé par le rôle, pour éviter qu'un inconnu s'inscrive le premier.

Prérequis

1. Traefik publie `photos.penhoat.example` (variable `client_subdomains.immich`) et Authentik répond sur `sso.penhoat.example`.
2. `immich` figure dans `nebuleuse_apps` et dans `nebuleuse_catalogue`.
3. Deux secrets existent dans le Vault du client : `vault_immich_db_password` (déjà prévu) et `vault_immich_admin_password` (à ajouter).
4. La VM dispose d'au moins 4 Go de mémoire : la reconnaissance de visages est gourmande.

`vault_immich_admin_password` complète la liste de la page « Inventaire et variables ». Valeur d'exemple : `FICTIF-admin-motdepasse-6`.

Arborescence du rôle

```
roles/immich/  
├─ defaults/main.yml  
├─ tasks/main.yml  
└─ templates/  
    ├─ docker-compose.yml.j2  
    └─ immich.json.j2
```

Variables

```
# roles/immich/defaults/main.yml  
  
# Version d'Immich : "v2" suit la version majeure 2. Immich publie souvent des changements  
# incompatibles : lire les notes de version avant toute montée de version.  
immich_version: "v2"  
  
# Image PostgreSQL d'Immich. Elle embarque les extensions de recherche vectorielle.  
# Le tag change à chaque version : le reprendre dans le docker-compose.yml officiel  
# de la version d'Immich choisie.  
immich_postgres_image: "ghcr.io/immich-app/postgres:14-vectorchord0.4.3-pgvector0.2.0"  
  
# Dossiers de travail sur la VM.  
immich_dir: /opt/nebuleuse/immich  
  
# Bibliothèque : photos et vidéos des utilisateurs. C'est le dossier qui grossit.
```

```
immich_library_dir: "{{ immich_dir }}/library"

# Nom de domaine de l'application.
immich_host: "{{ client_subdomains.immich }}.{{ client_domain }}"

# Secrets : définis dans le Vault, jamais en clair ici.
immich_db_password: "{{ vault_immich_db_password }}"
immich_admin_password: "{{ vault_immich_admin_password }}"

# Réseaux d'où Immich accepte l'en-tête X-Forwarded-For (réseaux privés de Docker).
immich_trusted_proxies: "172.16.0.0/12"

# Réseau Docker partagé avec Traefik.
immich_network: "{{ docker_proxy_network }}"
```

Le fichier Docker Compose

```
# roles/immich/templates/docker-compose.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.
# Il contient des secrets : accès réservé à root.

# Variables communes au serveur (ancrage YAML : un seul endroit à modifier).
x-immich-env: &immich_env
  DB_HOSTNAME: database
  DB_USERNAME: immich
  DB_DATABASE_NAME: immich
  DB_PASSWORD: "{{ immich_db_password }}"
  REDIS_HOSTNAME: redis
  # Active la lecture de la configuration déclarative.
  IMMICH_CONFIG_FILE: /config/immich.json
  IMMICH_TRUSTED_PROXIES: "{{ immich_trusted_proxies }}"
  TZ: Europe/Paris

services:
  server:
    image: ghcr.io/immich-app/immich-server:{{ immich_version }}
    container_name: {{ client_name }}_immich_server
    restart: unless-stopped
```

environment: *immich_env

volumes:

- # Bibliothèque des utilisateurs.
- {{ immich_library_dir }}/data
- # Configuration déclarative, en lecture seule.
- {{ immich_dir }}/immich.json:/config/immich.json:ro
- /etc/localtime:/etc/localtime:ro

depends_on:

- redis
- database

networks:

- {{ immich_network }}
- internal

labels:

- "traefik.enable=true"
- "traefik.docker.network={{ immich_network }}"
- "traefik.http.routers.immich-{{ client_name }}.rule=Host(`{{ immich\_host }}`)"
- "traefik.http.routers.immich-{{ client_name }}.entrypoints=websecure"
- # Port d'écoute du serveur Immich.
- "traefik.http.services.immich-{{ client_name }}.loadbalancer.server.port=2283"

machine-learning:

image: ghcr.io/immich-app/immich-machine-learning:{{ immich_version }}

container_name: {{ client_name }}_immich_ml

restart: unless-stopped

volumes:

- # Modèles téléchargés au premier usage : évite de les retélécharger à chaque démarrage.
- {{ immich_dir }}/model-cache:/cache

Joint uniquement par le serveur, sur le réseau privé.

networks:

- internal

redis:

image: docker.io/valkey/valkey:8-bookworm

container_name: {{ client_name }}_immich_redis

restart: unless-stopped

networks:

- internal

database:

```

image: {{ immich_postgres_image }}
container_name: {{ client_name }}_immich_db
restart: unless-stopped
environment:
  POSTGRES_USER: immich
  POSTGRES_DB: immich
  POSTGRES_PASSWORD: "{{ immich_db_password }}"
  # Détecte la corruption silencieuse des données.
  POSTGRES_INITDB_ARGS: "--data-checksums"
volumes:
  # La base doit être sur un disque local, jamais sur un partage réseau.
  - {{ immich_dir }}/db-data:/var/lib/postgresql/data
networks:
  - internal

networks:
  {{ immich_network }}:
    external: true
  internal: {}

```

Comme pour Authentik et Nextcloud, seul le serveur est sur le réseau `proxy`. Le service de machine learning, Redis et la base ne sont joignables que depuis le projet.

La configuration déclarative

Immich se configure dans son interface d'administration, mais accepte aussi un fichier JSON lu au démarrage. Ce fichier rend la configuration reproductible pour chaque client. En contrepartie, une fois le fichier actif, les réglages qu'il définit ne sont plus modifiables dans l'interface.

```

# roles/immich/templates/immich.json.j2
{
  "server": {
    "externalDomain": "https://{{ immich_host }}"
  },
  "passwordLogin": {
    "enabled": true
  },
  "oauth": {

```

```

"enabled": true,
"issuerUrl": "{{ oidc_issuer }}",
"clientId": "{{ oidc_client_id }}",
"clientSecret": "{{ oidc_client_secret }}",
"scope": "openid email profile",
"signingAlgorithm": "RS256",
"buttonText": "Se connecter avec Authentik",
"autoRegister": true,
"autoLaunch": false,
"storageLabelClaim": "preferred_username"
}
}

```

Le format JSON n'accepte pas les commentaires : les réglages sont donc expliqués ci-dessous. Le fichier contient le secret client du SSO, d'où le mode `0600` à son dépôt.

Réglage	Effet
<code>server.externalDomain</code>	Adresse publique, utilisée dans les liens de partage
<code>passwordLogin.enabled</code>	Garde la connexion par mot de passe : le compte administrateur local reste utilisable à côté du SSO
<code>oauth.issuerUrl</code>	Adresse de l'émetteur : Immich y trouve la découverte automatique de la configuration
<code>oauth.signingAlgorithm</code>	Algorithme de signature des jetons attendu (voir les pièges)
<code>oauth.autoRegister</code>	Crée le compte Immich à la première connexion par le SSO
<code>oauth.autoLaunch</code>	Si <code>true</code> , la page de connexion redirige directement vers le SSO
<code>oauth.storageLabelClaim</code>	Champ du jeton utilisé comme étiquette de stockage de l'utilisateur

Déploiement

```

# roles/immich/tasks/main.yml

- name: Créer les dossiers d'Immich
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory

```

```

    mode: "0755"
loop:
  - "{{ immich_dir }}"
  - "{{ immich_library_dir }}"
  - "{{ immich_dir }}/model-cache"
  - "{{ immich_dir }}/db-data"

# La connexion SSO est créée avant le fichier de configuration : elle fournit
# l'identifiant et le secret qui y sont inscrits.
- name: Créer la connexion SSO d'Immich dans Authentik
  ansible.builtin.include_tasks: ../../authentik/tasks/oidc_app.yml
vars:
  oidc_app_name: Immich
  oidc_app_slug: immich
  oidc_redirect_uris:
    # Connexion depuis le navigateur.
    - matching_mode: strict
      url: "https://{{ immich_host }}/auth/login"
    # Association d'un compte SSO depuis les réglages de l'utilisateur.
    - matching_mode: strict
      url: "https://{{ immich_host }}/user-settings"
    # Application mobile.
    - matching_mode: strict
      url: "app.immich:///oauth-callback"

- name: Déposer la configuration d'Immich
  ansible.builtin.template:
    src: immich.json.j2
    dest: "{{ immich_dir }}/immich.json"
    # Le fichier contient le secret client : root uniquement.
    mode: "0600"
  register: immich_config
  no_log: true

- name: Déposer le fichier Docker Compose d'Immich
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ immich_dir }}/docker-compose.yml"
    mode: "0600"

```

```

no_log: true

- name: Démarrer Immich
  community.docker.docker_compose_v2:
    project_src: "{{ immich_dir }}"
    state: present

# Immich ne relit son fichier de configuration qu'au démarrage.
- name: Redémarrer Immich si la configuration a changé
  community.docker.docker_compose_v2:
    project_src: "{{ immich_dir }}"
    state: restarted
  when: immich_config.changed

# Le premier démarrage applique les migrations de la base : cela peut durer quelques minutes.
- name: Attendre que l'API d'Immich réponde
  ansible.builtin.command:
    argv:
      - curl
      - -fsS
      - -k
      - --resolve
      - "{{ immich_host }}:443:127.0.0.1"
      - "https://{{ immich_host }}/api/server/ping"
  register: immich_ping
  until: immich_ping.rc == 0
  retries: 60
  delay: 5
  changed_when: false

- name: Savoir si Immich a déjà un administrateur
  ansible.builtin.command:
    argv:
      - curl
      - -sS
      - -k
      - --resolve
      - "{{ immich_host }}:443:127.0.0.1"
      - "https://{{ immich_host }}/api/server/config"

```

```

register: immich_server_config
changed_when: false

# Le premier compte créé dans Immich devient administrateur : le rôle le crée
# avant que quiconque puisse s'inscrire.
- name: Créer le compte administrateur du client
  ansible.builtin.command:
    argv: >-
    {{ ['curl', '-sS', '-k', '--fail-with-body',
        '--resolve', immich_host ~ ':443:127.0.0.1',
        '-X', 'POST', '-H', 'Content-Type: application/json',
        '--data-binary', (immich_admin_payload | to_json),
        'https://' ~ immich_host ~ '/api/auth/admin-sign-up'] }}
  vars:
    immich_admin_payload:
      email: "{{ client_admin_email }}"
      password: "{{ immich_admin_password }}"
      name: "Administrateur {{ client_name }}"
  when: not (immich_server_config.stdout | from_json).isInitialized
  no_log: true

```

Vérification

```

# Sur la VM : quatre conteneurs "Up".
docker ps --filter name=immich --format '{{.Names}}\t{{.Status}}'

# L'API répond : doit afficher {"res":"pong"}.
curl -sS https://photos.penhoat.example/api/server/ping

# L'instance est initialisée et le SSO est activé.
curl -sS https://photos.penhoat.example/api/server/config | jq '{isInitialized,
oauthButtonText}'

# La découverte de l'émetteur répond et indique l'algorithme de signature (attendu : RS256).
curl -sS https://sso.penhoat.example/application/o/immich/.well-known/openid-configuration \
| jq '{issuer, id_token_signing_alg_values_supported}'

```

Dans le navigateur, la page `https://photos.penhoat.example` doit afficher le formulaire de connexion avec le bouton « Se connecter avec Authentik ». La connexion SSO avec l'adresse e-mail du client doit ouvrir le compte administrateur créé par le rôle, sans en créer un second.

Pièges

1. **Le premier compte créé devient administrateur** : si Immich est exposé avant la création du compte, la première personne qui s'inscrit, ou qui se connecte par le SSO, prend ce rôle. Le rôle crée donc l'administrateur dès le démarrage.
2. **Réglages verrouillés dans l'interface** : avec le fichier `immich.json`, les rubriques qu'il définit sont en lecture seule. Toute modification passe par le fichier, puis un redémarrage.
3. **Algorithme de signature** : Immich attend des jetons signés en RS256. Authentik ne signe en RS256 que si le provider porte une clé de signature. C'est pourquoi la page Authentik associe la clé de signature par défaut à chaque provider.
4. **Trois adresses de retour** : le navigateur, les réglages utilisateur et l'application mobile en utilisent chacun une. Sans la troisième, la connexion fonctionne dans le navigateur mais échoue sur le téléphone.
5. **Image PostgreSQL** : le tag doit correspondre à la version d'Immich. Une image PostgreSQL ordinaire ne contient pas les extensions de recherche, et le serveur ne démarre pas. Un changement de version majeure de PostgreSQL impose une migration, pas un simple changement de tag.
6. **Base sur un partage réseau** : PostgreSQL y est lent et peut se corrompre. Le dossier `db-data` reste sur un disque local.
7. **Mémoire et premier lancement** : le service de machine learning télécharge ses modèles au premier usage et consomme plusieurs Go. Les premières tâches de reconnaissance sont longues, sans que ce soit une panne.
8. **La bibliothèque grossit vite** : elle croît avec chaque photo et vidéo envoyées. La surveillance de l'espace disque et la sauvegarde du dossier `library` et de la base sont à prévoir, la sauvegarde de la base seule ne suffit pas.
9. **Montées de version** : Immich publie régulièrement des changements incompatibles. Il faut lire les notes de version et sauvegarder avant de changer `immich_version`.
10. **Compte SSO et compte existant** : Immich rattache la connexion SSO au compte qui porte la même adresse e-mail. L'adresse du compte d'Authentik doit donc être identique à celle du compte Immich.
11. **Immich ne joint pas le SSO** : la découverte de l'émetteur échoue depuis le conteneur. Le mécanisme d'alias réseau de Traefik, décrit dans sa page, évite ce détour par l'adresse publique.

Page suivante : **Vaultwarden**.

Vaultwarden

En bref

Niveau : intermédiaire

Durée : 30 minutes

Prérequis : Traefik en place

Niveau de test : reconstitué

Vaultwarden, c'est quoi ?

Vaultwarden est un gestionnaire de mots de passe auto-hébergé, compatible avec les applications et extensions Bitwarden. Chaque utilisateur y range ses identifiants dans un coffre chiffré, accessible depuis le navigateur, l'ordinateur et le téléphone. Un seul conteneur suffit : le serveur et sa base de données SQLite.

Élément	Rôle
Conteneur Vaultwarden	Serveur, interface web et API compatible Bitwarden
Dossier <code>data</code>	Base SQLite, pièces jointes, clés et journal
Journal <code>vaultwarden.log</code>	Lu par Fail2ban pour bloquer les échecs de connexion

Navigateur ou application → Traefik → Vaultwarden

Concrètement, Vaultwarden est l'application la plus simple du catalogue à déployer, et la seule qui ne passe pas par le SSO.

Le coffre est chiffré avec le mot de passe principal de l'utilisateur, qui ne quitte jamais son appareil. Ce mot de passe ne peut pas être délégué à Authentik : le serveur ne pourrait plus rien déchiffrer sans lui. Le rôle ne crée donc pas de connexion SSO pour Vaultwarden. Certaines versions proposent une connexion SSO optionnelle, non activée ici.

Prérequis

1. Traefik publie `vault.penhoat.example` (variable `client_subdomains.vaultwarden`).
2. `vaultwarden` figure dans `nebuleuse_apps` et dans `nebuleuse_catalogue`.
3. Le secret `vault_vaultwarden_admin_token` existe dans le Vault du client (page « Inventaire et variables »).

Arborescence du rôle

```
roles/vaultwarden/  
├─ defaults/main.yml  
├─ tasks/main.yml  
└─ templates/  
    └─ docker-compose.yml.j2
```

Variables

```
# roles/vaultwarden/defaults/main.yml  
  
# Version de Vaultwarden : remplacer par la dernière version stable.  
vaultwarden_version: "1.34.3"  
  
# Dossier de travail sur la VM.  
vaultwarden_dir: /opt/nebuleuse/vaultwarden  
  
# Nom de domaine de l'application.  
vaultwarden_host: "{{ client_subdomains.vaultwarden }}.{{ client_domain }}"  
  
# Inscriptions : fermées en fonctionnement normal. À ouvrir temporairement pour créer  
# les premiers comptes (voir la section « Créer les premiers comptes »).  
vaultwarden_signups_allowed: false  
  
# Seules les adresses de ce domaine peuvent s'inscrire quand les inscriptions sont ouvertes.  
vaultwarden_signups_domain: "{{ client_domain }}"  
  
# Panneau d'administration (/admin) : désactivé par défaut. Il donne accès à tous les comptes.  
vaultwarden_admin_enabled: false
```

```
# Jeton du panneau d'administration, stocké dans le Vault sous forme de hachage Argon2.
vaultwarden_admin_token: "{{ vault_vaultwarden_admin_token }}"

# Réseau Docker partagé avec Traefik.
vaultwarden_network: "{{ docker_proxy_network }}"
```

Le fichier Docker Compose

```
# roles/vaultwarden/templates/docker-compose.yml.j2
# Fichier géré par Ansible : toute modification manuelle sera écrasée.
# Il peut contenir un jeton d'administration : accès réservé à root.

services:
  vaultwarden:
    image: vaultwarden/server:{{ vaultwarden_version }}
    container_name: {{ client_name }}_vaultwarden
    restart: unless-stopped
    environment:
      # Adresse publique : utilisée dans les liens des e-mails et par les extensions.
      DOMAIN: "https://{{ vaultwarden_host }}"
      SIGNUPS_ALLOWED: "{{ vaultwarden_signups_allowed | lower }}"
      SIGNUPS_DOMAINS_WHITELIST: "{{ vaultwarden_signups_domain }}"
      # Les comptes existants peuvent inviter d'autres personnes (organisations).
      INVITATIONS_ALLOWED: "true"
      # Journal dans un fichier : c'est lui que lit Fail2ban.
      LOG_FILE: /data/vaultwarden.log
      LOG_LEVEL: warn
      TZ: Europe/Paris
{% if vaultwarden_admin_enabled %}
      # Un hachage Argon2 contient des "$", qui doivent être doublés dans un fichier Compose.
      ADMIN_TOKEN: '{{ vaultwarden_admin_token | replace("$", "$$") }}'
{% endif %}
    volumes:
      - {{ vaultwarden_dir }}/data:/data
    networks:
      - {{ vaultwarden_network }}
    labels:
      - "traefik.enable=true"
```

```

- "traefik.docker.network={{ vaultwarden_network }}"
- "traefik.http.routers.vaultwarden-{{ client_name }}.rule=Host(`{{ vaultwarden_host }}`)"
- "traefik.http.routers.vaultwarden-{{ client_name }}.entrypoints=websecure"
# Port d'écoute de Vaultwarden dans le conteneur.
- "traefik.http.services.vaultwarden-{{ client_name }}.loadbalancer.server.port=80"

networks:
  {{ vaultwarden_network }}:
    external: true

```

Vaultwarden lit l'adresse du visiteur dans l'en-tête `X-Real-IP`, que Traefik renseigne. Les échecs de connexion de son journal portent donc la vraie adresse du visiteur, ce qui permet à Fail2ban de la bloquer.

Déploiement

```

# roles/vaultwarden/tasks/main.yml

- name: Créer les dossiers de Vaultwarden
  ansible.builtin.file:
    path: "{{ item }}"
    state: directory
    mode: "0755"
  loop:
    - "{{ vaultwarden_dir }}"
    - "{{ vaultwarden_dir }}/data"

- name: Déposer le fichier Docker Compose de Vaultwarden
  ansible.builtin.template:
    src: docker-compose.yml.j2
    dest: "{{ vaultwarden_dir }}/docker-compose.yml"
    # Le fichier peut contenir un jeton : root uniquement.
    mode: "0600"
  no_log: true

- name: Démarrer Vaultwarden

```

```
community.docker.docker_compose_v2:
  project_src: "{{ vaultwarden_dir }}"
  state: present

- name: Attendre que Vaultwarden réponde
  ansible.builtin.command:
    argv:
      - curl
      - -fsS
      - -k
      - --resolve
      - "{{ vaultwarden_host }}:443:127.0.0.1"
      - "https://{{ vaultwarden_host }}/alive"
  register: vaultwarden_alive
  until: vaultwarden_alive.rc == 0
  retries: 30
  delay: 5
  changed_when: false
```

Créer les premiers comptes

Les inscriptions sont fermées par défaut : personne ne peut créer de compte sur l'adresse publique. Pour intégrer un client, le support ouvre les inscriptions le temps de créer les premiers comptes, puis les referme.

```
# 1. Ouvrir les inscriptions pour ce client, pour les adresses de son domaine seulement.
ansible-playbook site.yml --limit client-alpha --tags vaultwarden -e
vaultwarden_signups_allowed=true

# 2. Le client crée son compte sur https://vault.penhoat.example (mot de passe principal
choisi par lui).

# 3. Refermer les inscriptions : relancer sans la variable.
ansible-playbook site.yml --limit client-alpha --tags vaultwarden
```

Les comptes suivants sont créés par invitation depuis une organisation, sans rouvrir les inscriptions.

Vérification

```
# Sur la VM : le conteneur est démarré.
docker ps --filter name=vaultwarden --format '{{.Names}}\t{{.Status}}'

# L'adresse publique répond : doit afficher une date ISO.
curl -sS https://vault.penhoat.example/alive

# Le fichier de journal existe (créé au premier événement).
ls -l /opt/nebuleuse/vaultwarden/data/vaultwarden.log
```

Dans le navigateur, `https://vault.penhoat.example` affiche la page de connexion. Si les inscriptions sont fermées, le bouton « Créer un compte » est absent.

Pièges

1. **Inscriptions laissées ouvertes** : une instance publique avec les inscriptions ouvertes accepte n'importe quelle adresse du domaine autorisé, sans vérification. La variable `vaultwarden_signups_allowed` doit revenir à `false` dès les comptes créés.
2. **Jeton d'administration en clair** : le jeton doit être stocké sous forme de hachage Argon2 dans le Vault, pas en clair. Dans le fichier Compose, chaque `$` doit être doublé, ce que fait le modèle avec `replace`.
3. **Panneau d'administration exposé** : il donne accès à tous les comptes. Il reste désactivé tant que `vaultwarden_admin_enabled` n'est pas passé à `true`, et ne doit l'être que le temps d'une intervention.
4. **Mot de passe principal perdu** : le serveur ne peut ni le retrouver ni le réinitialiser, car il ne le connaît pas. Sans lui, le coffre est perdu. Prévenir le client à l'ouverture de son compte.
5. **Extensions qui n'atteignent pas le serveur** : l'adresse du serveur doit être saisie en entier, `https://vault.penhoat.example`, et le certificat doit être reconnu. Les certificats de test de Let's Encrypt sont refusés par les applications.
6. **Journal absent pour Fail2ban** : sans `LOG_FILE`, Vaultwarden écrit sur la sortie standard et le filtre ne trouve rien. Le chemin hôte du journal est `/opt/nebuleuse/vaultwarden/data/vaultwarden.log`.
7. **Sauvegarde** : tout le contenu est dans le dossier `data`. Une copie du dossier à froid, ou une copie de la base avec `sqlite3 ... ".backup"`, suffit. Une copie à chaud du seul fichier de base peut être corrompue.

Page suivante : **Fail2ban**.

Fail2ban

En bref**Niveau** : avancé**Durée** : 45 minutes**Prérequis** : Traefik et au moins une application en place**Niveau de test** : reconstitué

Fail2ban, c'est quoi ?

Fail2ban lit des fichiers de journaux, repère les adresses qui multiplient les échecs (mots de passe erronés, pages interdites) et les bloque quelques instants dans le pare-feu. Il ne remplace pas les mots de passe solides : il réduit le bruit des attaques automatiques et ralentit les tentatives de devinette.

Élément	Rôle
Filtre (<code>filter.d</code>)	Expression régulière qui reconnaît un échec dans une ligne de journal
Jail (<code>jail.d</code>)	Associe un filtre, un fichier de journal, un seuil et une durée de blocage
Action de blocage	Ajoute une règle dans le pare-feu pour l'adresse fautive
Chaîne <code>DOCKER-USER</code>	Point où les règles s'appliquent aux conteneurs

Visiteur fautif → journal Traefik ou Vaultwarden → filtre → jail → règle dans `DOCKER-USER` → adresse bloquée

Concrètement, le rôle installe Fail2ban sur la VM et y dépose deux filtres et deux jails : un pour les erreurs d'authentification vues par Traefik, un pour les échecs de connexion à Vaultwarden. Les jails portent le préfixe `nebuleuse-`.

Les principes de Fail2ban (filtres, jails, commandes de base) sont détaillés dans le livre **Fail2Ban** de la base de connaissance. Cette page ne traite que l'intégration au socle Nébuleuse.

Prérequis

1. Le socle est en place : Docker a créé la chaîne `DOCKER-USER`.
2. Traefik écrit son journal d'accès en JSON dans `/var/log/traefik/access.log` (page Traefik).
3. Pour la jail Vaultwarden : le journal `/opt/nebuleuse/vaultwarden/data/vaultwarden.log` existe (page Vaultwarden).

Pourquoi `DOCKER-USER` ?

Docker publie les ports des conteneurs en les redirigeant avant le pare-feu de la machine. Une règle de blocage placée dans la chaîne habituelle (`INPUT`) ne verrait jamais le trafic destiné à Traefik. La chaîne `DOCKER-USER`, elle, est traversée avant les règles de Docker, et Docker n'y touche jamais : c'est le bon endroit pour bloquer une adresse.

Arborescence du rôle

```
roles/fail2ban/  
├─ defaults/main.yml  
├─ handlers/main.yml  
├─ tasks/main.yml  
└─ templates/  
    ├─ filter-nebuleuse-traefik.conf.j2  
    ├─ filter-nebuleuse-vaultwarden.conf.j2  
    └─ jail-nebuleuse.conf.j2
```

Variables

```
# roles/fail2ban/defaults/main.yml  
  
# Adresses jamais bloquées : la VM elle-même et le réseau d'administration.  
# Ajouter ici l'adresse fixe du client si elle est connue.  
fail2ban_ignore_ips:  
  - 127.0.0.1/8  
  - "{{ admin_network_cidr }}"
```

```
# Jail Traefik : trop d'erreurs 401 ou 403 en peu de temps.
```

```
fail2ban_traefik_maxretry: 30
```

```
fail2ban_traefik_findtime: 10m
```

```
fail2ban_traefik_bantime: 1h
```

```
# Jail Vaultwarden : échecs de connexion.
```

```
fail2ban_vaultwarden_maxretry: 5
```

```
fail2ban_vaultwarden_findtime: 10m
```

```
fail2ban_vaultwarden_bantime: 1h
```

```
# Chaîne du pare-feu où les blocages sont ajoutés.
```

```
fail2ban_chain: DOCKER-USER
```

Les filtres

Le journal d'accès de Traefik contient une ligne JSON par requête. Les champs sont triés par ordre alphabétique : `ClientHost` (l'adresse du visiteur) précède donc toujours `DownstreamStatus` (le code de réponse).

```
# roles/fail2ban/templates/filter-nebuleuse-traefik.conf.j2
```

```
# Fichier géré par Ansible.
```

```
[Definition]
```

```
# Une ligne est un échec quand la réponse est 401 (non authentifié) ou 403 (interdit).
```

```
failregex = ^.*"ClientHost": "<HOST>" .*"DownstreamStatus": (?<401|403>).*
```

```
# Format de la date dans le journal JSON.
```

```
datepattern = "time": "%Y-%m-%dT%H:%M:%S
```

```
ignoreregex =
```

```
# roles/fail2ban/templates/filter-nebuleuse-vaultwarden.conf.j2
```

```
# Fichier géré par Ansible.
```

```
[Definition]
```

```
# Mot de passe erroné, ou jeton d'administration erroné. <ADDR> désigne l'adresse du visiteur.
```

```
failregex = ^.*Username or password is incorrect\. Try again\. IP: <ADDR>\. Username:.*$
```

```
^.*Invalid admin token\. IP: <ADDR>.*$
```

```
ignoreregex =
```

Les jails

```
# roles/fail2ban/templates/jail-nebuleuse.conf.j2
# Fichier g  r   par Ansible.

[DEFAULT]
ignoreip = {{ fail2ban_ignore_ips | join(' ') }}
# Blocage par iptables, dans la cha  ne travers  e par le trafic des conteneurs.
banaction = iptables-multiport
chain = {{ fail2ban_chain }}
port = http,https

[nebuleuse-traefik]
enabled = true
filter = nebuleuse-traefik
logpath = /var/log/traefik/access.log
maxretry = {{ fail2ban_traefik_maxretry }}
findtime = {{ fail2ban_traefik_findtime }}
bantime = {{ fail2ban_traefik_bantime }}
{% if 'vaultwarden' in nebuleuse_apps %}

[nebuleuse-vaultwarden]
enabled = true
filter = nebuleuse-vaultwarden
logpath = /opt/nebuleuse/vaultwarden/data/vaultwarden.log
maxretry = {{ fail2ban_vaultwarden_maxretry }}
findtime = {{ fail2ban_vaultwarden_findtime }}
bantime = {{ fail2ban_vaultwarden_bantime }}
{% endif %}
```

T  ches

```
# roles/fail2ban/tasks/main.yml

- name: Installer Fail2ban et iptables
  ansible.builtin.apt:
    name:
```

```

- fail2ban
- iptables          # absent d'une Debian 12 minimale
state: present

- name: Déposer les filtres
  ansible.builtin.template:
    src: "filter-{{ item }}.conf.j2"
    dest: "/etc/fail2ban/filter.d/{{ item }}.conf"
    mode: "0644"
  loop:
    - nebuleuse-traefik
    - nebuleuse-vaultwarden
  notify: Recharger fail2ban

- name: Déposer les jails
  ansible.builtin.template:
    src: jail-nebuleuse.conf.j2
    dest: /etc/fail2ban/jail.d/nebuleuse.conf
    mode: "0644"
  notify: Recharger fail2ban

- name: Activer et démarrer Fail2ban
  ansible.builtin.service:
    name: fail2ban
    state: started
    enabled: true

```

```

# roles/fail2ban/handlers/main.yml

- name: Recharger fail2ban
  ansible.builtin.service:
    name: fail2ban
    state: restarted

```

Vérification

```

# Contrôle que le filtre reconnaît bien des lignes du journal (aucune erreur attendue).
sudo fail2ban-regex /var/log/traefik/access.log /etc/fail2ban/filter.d/nebuleuse-traefik.conf

```

```
# Les jails sont actives.
sudo fail2ban-client status

# Détail d'une jail : compteurs d'échecs et adresses bloquées.
sudo fail2ban-client status nebuleuse-traefik

# La chaîne DOCKER-USER contient les règles de Fail2ban.
sudo iptables -L DOCKER-USER -n
```

Pour provoquer un blocage, depuis une machine qui n'est pas dans `ignoreip`, envoyer plus de requêtes refusées que le seuil :

```
# 35 requêtes avec un mauvais mot de passe : la réponse est 401 à chaque fois.
for i in $(seq 35); do
    curl -s -o /dev/null -u test:test https://cloud.penhoat.example/remote.php/dav
done

# L'adresse de la machine de test doit apparaître dans "Banned IP list".
sudo fail2ban-client status nebuleuse-traefik

# Débloquer l'adresse après le test.
sudo fail2ban-client set nebuleuse-traefik unbanip <adresse>
```

Pièges

1. **Fichier de journal absent** : une jail dont le fichier n'existe pas ne démarre pas. D'où la place de Fail2ban en dernier dans `site.yml`.
2. **Blocage sans effet** : une action qui insère les règles dans `INPUT` ne voit pas le trafic des conteneurs. L'action doit viser la chaîne `DOCKER-USER`.
3. **Adresse de Traefik bloquée à la place du visiteur** : si l'adresse lue dans le journal est celle du proxy et non celle du visiteur, toute la VM est coupée. Le champ `ClientHost` doit contenir l'adresse réelle ; l'adresse du réseau d'administration est exclue par `ignoreip`.
4. **Clients de synchronisation bloqués** : une application de synchronisation (Nextcloud, par exemple) provoque normalement quelques réponses 401. Le seuil de 30 en dix minutes laisse de la marge. Un seuil trop bas bloque un utilisateur légitime.
5. **Blocages perdus après un redémarrage de Docker** : Docker recrée la chaîne `DOCKER-USER` et efface les règles. Fail2ban doit être redémarré après Docker pour réinstaller les blocages en cours.
6. **Expression régulière qui ne reconnaît rien** : `fail2ban-regex` est l'outil de contrôle. Si la date n'est pas reconnue, la ligne est ignorée, même si l'expression est bonne.

7. **Se bloquer soi-même** : en cas de test depuis le poste d'administration, l'adresse peut être bloquée à son tour. `fail2ban-client set <jail> unbanip <adresse>` la libère.

Page suivante : **Déploiement d'un client.**

Exploiter

Déploiement d'un client

En bref

Niveau : intermédiaire

Durée : 45 minutes

Prérequis : tous les rôles des pages précédentes écrits ; une VM Debian 12 vierge

Niveau de test : reconstitué

Le déploiement, c'est quoi ?

Les pages précédentes ont décrit les rôles un par un. Cette page les assemble : le fichier `site.yml` complet, la procédure pour intégrer un nouveau client, et le test qui prouve que le déploiement est reproductible. C'est la page qui répond à l'objectif du projet : un client, une commande.

Élément	Rôle
<code>site.yml</code>	Playbook unique : contrôles, puis rôles dans l'ordre
<code>inventory/host_vars/<client>/</code>	Variables et secrets du client
Étiquettes (<code>tags</code>)	Permettent de rejouer une seule partie
<code>--limit</code>	Restreint l'exécution à un client

Nouvelle commande → dossier `host_vars` → ligne dans l'inventaire → `ansible-playbook site.yml --limit <client>`

Concrètement, déployer un client revient à créer ses fichiers de variables, l'ajouter à l'inventaire, et lancer le même playbook que pour tous les autres. Aucun rôle n'est modifié.

Prérequis

- Le serveur Ansible est en place, avec `.vault_pass` (page 1).
- Les rôles `base-hardening`, `docker-engine`, `teleport-agent`, `traefik`, `authentik`, `nextcloud`, `immich`, `vaultwarden` et `fail2ban` existent dans `roles/`.
- La VM du client est livrée : Debian 12, utilisateur `deploy` autorisé par clé SSH, adresse du réseau d'administration.

4. Les enregistrements DNS du client pointent vers l'adresse publique de la VM.

Les secrets attendus par application

Chaque application commandée exige des secrets dans le Vault. Une table dans les variables communes les liste, pour que le playbook puisse vérifier qu'aucun ne manque avant de commencer.

```
# inventory/group_vars/all/secrets_catalogue.yml
# Secrets obligatoires : ceux du socle, puis ceux de chaque application.

nebuleuse_core_secrets:
  - vault_authentik_secret_key
  - vault_authentik_db_password
  - vault_authentik_bootstrap_password
  - vault_authentik_bootstrap_token

# Une entrée par application du catalogue, même sans secret (liste vide).
nebuleuse_app_secrets:
  nextcloud:
    - vault_nextcloud_db_password
    - vault_nextcloud_admin_password
  immich:
    - vault_immich_db_password
    - vault_immich_admin_password
  vaultwarden:
    - vault_vaultwarden_admin_token
  jellyfin: []
  papermerge: []
  homeassistant: []
  matrix: []
```

Le playbook complet

```

# site.yml

- name: Déployer un client Nébuleuse
  hosts: clients
  become: true          # les tâches s'exécutent en root via sudo
  # Un client à la fois : une erreur sur l'un n'interrompt pas la préparation des autres.
  serial: 1

pre_tasks:
  # --- Contrôles : échouer tout de suite plutôt qu'au milieu du déploiement ---
  - name: Vérifier que toutes les applications commandées existent au catalogue
    ansible.builtin.assert:
      that:
        - nebuleuse_apps | difference(nebuleuse_catalogue) | length == 0
      fail_msg: >-
        Application(s) inconnue(s) : {{ nebuleuse_apps | difference(nebuleuse_catalogue) }}
      quiet: true
    tags: always

  - name: Vérifier que le client a bien un domaine
    ansible.builtin.assert:
      that:
        - client_domain is defined
        - client_domain | length > 0
      fail_msg: "La variable client_domain doit être définie pour {{ inventory_hostname }}."
      quiet: true
    tags: always

  - name: Vérifier que les secrets nécessaires existent dans le Vault
    ansible.builtin.assert:
      that:
        # "vars" contient toutes les variables visibles pour ce client, Vault déchiffré
        compris.
        - nebuleuse_secrets_manquants | length == 0
      fail_msg: "Secrets manquants dans le Vault : {{ nebuleuse_secrets_manquants }}"
      quiet: true
    vars:
      nebuleuse_secrets_attendus: >-
        {{ nebuleuse_core_secrets +
          (nebuleuse_apps | map('extract', nebuleuse_app_secrets) | flatten | list) }}

```

```
nebuleuse_secrets_manquants: >-
  {{ nebuleuse_secrets_attendus | reject('in', vars) | list }}
tags: always
```

roles:

```
# 1. Socle : système, pare-feu, Docker et réseau "proxy".
- { role: base-hardening, tags: [base] }
- { role: docker-engine, tags: [docker] }

# 2. Accès administrateur par le bastion.
- { role: teleport-agent, tags: [teleport] }

# 3. Publication web et certificats.
- { role: traefik, tags: [traefik] }

# 4. SSO : toujours présent, il fait partie du socle.
- { role: authentik, tags: [sso] }

# 5. Applications : déployées uniquement si elles figurent dans la commande.
- { role: nextcloud, tags: [nextcloud], when: "'nextcloud' in nebuleuse_apps" }
- { role: immich, tags: [immich], when: "'immich' in nebuleuse_apps" }
- { role: vaultwarden, tags: [vaultwarden], when: "'vaultwarden' in nebuleuse_apps" }

# 6. Protection : en dernier, elle lit les journaux des rôles précédents.
- { role: fail2ban, tags: [fail2ban] }
```

Les contrôles portent l'étiquette `always` : ils s'exécutent même quand un seul rôle est rejoué avec `--tags`. Le premier passage d'un nouveau client doit en revanche se faire sans étiquette.

Intégrer un nouveau client

La procédure est la même pour chaque client. L'exemple prend un troisième client, de numéro 3 : adresses `10.99.0.23` et `192.0.2.23`.

```
# 1. Créer le dossier de variables à partir du client existant le plus proche.
cp -r inventory/host_vars/client-alpha inventory/host_vars/client-<nom>
```

```
# 2. Ajouter la machine à l'inventaire.
# inventory/hosts.yml (extrait)
all:
  children:
    clients:
      hosts:
        client-<nom>:
          ansible_host: 10.99.0.23
```

```
# 3. Éditer vars.yml (domaine, e-mail, applications) puis vault.yml (secrets neufs).
#   Chaque client a des secrets distincts : ne jamais recopier ceux d'un autre.
ansible-vault edit inventory/host_vars/client-<nom>/vault.yml

# 4. Vérifier la connexion et la syntaxe.
ansible client-<nom> -m ping
ansible-playbook site.yml --syntax-check

# 5. Simuler : montre ce qui changerait, sans rien modifier.
ansible-playbook site.yml --limit client-<nom> --check --diff

# 6. Déployer.
ansible-playbook site.yml --limit client-<nom>
```

La simulation (`--check`) s'arrête vite sur un client vierge : plusieurs tâches dépendent de ce que les précédentes ont réellement installé. Elle est surtout utile sur un client déjà déployé, pour prévoir l'effet d'une modification.

Le test d'idempotence

Un déploiement est idempotent quand le rejouer sur un client à jour ne modifie rien. C'est le critère qui distingue un playbook fiable d'un script.

```
# Premier passage : de nombreuses tâches "changed".
ansible-playbook site.yml --limit client-<nom>

# Second passage immédiat : la ligne finale doit afficher "changed=0".
ansible-playbook site.yml --limit client-<nom>
```

Si le second passage signale des changements, la tâche concernée est à corriger : un `command` sans `changed_when`, un modèle qui change à chaque exécution (une date, une valeur aléatoire), ou un module qui ne compare pas l'état réel.

Rejouer une partie seulement

```
# Un seul service : par exemple après un changement de version de Nextcloud.
ansible-playbook site.yml --limit client-alpha --tags nextcloud

# Plusieurs services, dans l'ordre du playbook.
ansible-playbook site.yml --limit client-alpha --tags traefik,sso

# Ajouter une application à un client existant : modifier nebuleuse_apps dans vars.yml,
# puis rejouer son étiquette et celle du SSO (qui crée la connexion).
ansible-playbook site.yml --limit client-alpha --tags sso,immich,fail2ban
```

Vérification globale

```
# Sur la VM : tous les conteneurs sont "Up".
docker ps --format '{{.Names}}\t{{.Status}}'

# Les certificats sont délivrés : chaque adresse répond en HTTPS.
for h in sso cloud photos vault; do
    curl -sS -o /dev/null -w "${h}.penhoat.example : %{http_code}\n" "https://${h}.penhoat.example/"
done

# Le bastion voit la VM (depuis le bastion).
tctl nodes ls

# Les jails Fail2ban sont actives.
sudo fail2ban-client status
```

Pièges

1. **Secret manquant découvert en fin de parcours** : sans le contrôle des secrets, l'erreur « is undefined » survient après plusieurs minutes de déploiement. Le tableau `nebulouse_app_secrets` doit être tenu à jour à chaque nouvelle application.
2. **Rejouer une application sans le SSO** : une application a besoin de la connexion créée par Authentik. Pour la première installation d'une application, inclure l'étiquette `sso`.
3. `--limit` **oublié** : sans lui, le playbook cible tous les clients, l'un après l'autre. `serial: 1` limite les dégâts, mais le temps et les effets d'une erreur s'accumulent.
4. **Secrets copiés d'un client à l'autre** : un secret commun annule l'isolation entre clients. Chaque `vault.yml` doit avoir ses propres valeurs.
5. **Certificats de test restés actifs** : si `traefik_acme_staging` est resté à `true` après les essais, les navigateurs refusent le certificat. La variable est à repasser à `false`, puis le dossier `acme` à vider avant de rejouer l'étiquette `traefik`.
6. **Idempotence qui se dégrade** : une tâche ajoutée sans `changed_when` fait échouer le test d'idempotence. Le test se rejoue après chaque modification de rôle.

Page suivante : **Pièges et dépannage.**

Pièges et dépannage

En bref**Niveau** : intermédiaire**Durée** : 20 minutes de lecture**Prérequis** : avoir lu les pages de rôles**Niveau de test** : reconstitué

Dépanner, c'est quoi ?

Une application qui ne répond pas peut tomber en panne à plusieurs endroits : le DNS, le certificat, Traefik, Authentik, l'application elle-même. Chercher au hasard fait perdre du temps. La méthode la plus sûre consiste à remonter le chemin d'une requête, étape par étape, et à s'arrêter à la première qui échoue.

Étape	Question	Page de référence
1. DNS	Le nom pointe-t-il vers la bonne adresse ?	Traefik
2. Pare-feu	Les ports 80 et 443 sont-ils ouverts et joignables ?	Socle
3. Traefik	Le routeur existe-t-il et le certificat est-il délivré ?	Traefik
4. Application	Le conteneur tourne-t-il et répond-il ?	Page de l'application
5. SSO	La connexion par Authentik aboutit-elle ?	Authentik
6. Blocage	L'adresse du visiteur est-elle bloquée ?	Fail2ban

DNS → pare-feu → Traefik → application → SSO → blocage éventuel

Les commandes de diagnostic

```
# 1. DNS : l'adresse retournée doit être l'adresse publique de la VM.
dig +short cloud.penhoat.example

# 2. Pare-feu : règles de la VM et chaîne des blocages.
sudo ufw status verbose
sudo iptables -L DOCKER-USER -n

# 3. Traefik : journal de démarrage, puis requêtes récentes.
docker logs --tail 50 alpha_traefik
sudo tail -n 20 /var/log/traefik/access.log

# 4. Application : conteneurs, puis journal d'un conteneur précis.
docker ps --format '{{.Names}}\t{{.Status}}'
docker logs --tail 50 alpha_nextcloud

# 5. SSO : l'API d'Authentik répond.
curl -sSk --resolve sso.penhoat.example:443:127.0.0.1 https://sso.penhoat.example/-/health/live/

# 6. Blocage : adresses bloquées par jail.
sudo fail2ban-client status nebuleuse-traefik
```

Symptômes courants

Symptôme	Cause probable	Vérification et correction
Le navigateur affiche une erreur de certificat	Certificat non délivré, ou mode <code>staging</code> actif	<code>traefik_acme_staging</code> doit valoir <code>false</code> ; examiner le journal de Traefik pour l'erreur ACME
Erreur 404 de Traefik	Le routeur n'existe pas	Labels <code>traefik.enable</code> et <code>Host(...)</code> absents ou erronés dans le fichier Compose
Erreur 502 ou 504	Traefik n'atteint pas le conteneur	Mauvais port dans <code>loadbalancer.server.port</code> , conteneur hors du réseau <code>proxy</code> , label <code>traefik.docker.network</code> manquant
Boucle de connexion sur le SSO	Mauvais flux sur le provider	Le provider doit utiliser le flux d'autorisation (page Authentik)

Symptôme	Cause probable	Vérification et correction
« Redirect URI invalide »	Adresse de retour différente	Comparer caractère pour caractère l'adresse du provider et celle de l'application
« Accès refusé » après connexion	Pas de règle d'accès liée à l'application	Lier la règle <code>nebuleuse-allow-all</code> à l'application
L'application ne joint pas le SSO	Détour par l'adresse publique	Alias réseau de Traefik pour <code>sso.<domaine></code>
Visiteur légitime bloqué	Seuil trop bas, ou adresse partagée	<code>fail2ban-client set <jail> unbanip <adresse></code> , puis relever le seuil ou ajouter à <code>ignoreip</code>
Le déploiement échoue sur une variable <code>vault_...</code>	Secret absent du Vault	Contrôle des secrets en début de <code>site.yml</code>
SSH refusé après le socle	Pare-feu ou <code>ListenAddress</code>	Se connecter par la console de l'hébergeur et vérifier <code>sshd -T</code>
Le disque se remplit	Journaux ou bibliothèque Immich	<code>docker system df ; du -sh /opt/nebuleuse/*</code> ; vérifier la rotation du journal Traefik

Où sont les journaux ?

Élément	Emplacement
Journal d'accès de Traefik	<code>/var/log/traefik/access.log</code>
Traefik, applications, Authentik	<code>docker logs <conteneur></code>
Vaultwarden	<code>/opt/nebuleuse/vaultwarden/data/vaultwarden.log</code>
Fail2ban	<code>/var/log/fail2ban.log</code>
Ansible	Sortie du terminal ; pour garder une trace : <code>-vv</code> , ou une redirection vers un fichier

Si le déploiement échoue en cours de route

Un déploiement qui s'arrête au milieu laisse la VM dans un état intermédiaire, mais rejouable : les rôles sont idempotents.

1. Lire le message de la **première** tâche en erreur : les suivantes en découlent souvent.

2. Corriger la cause (variable, secret, DNS, connectivité).
3. Rejouer le playbook sur ce client, avec la même commande.
4. Si l'erreur persiste dans un rôle précis, le rejouer seul avec `--tags` et `-vv` pour afficher le détail des appels.

```
# Rejouer un seul rôle avec le détail de chaque tâche.  
ansible-playbook site.yml --limit client-alpha --tags sso -vv
```

Remettre une application à zéro

Cette opération supprime les données de l'application : elle n'est à faire que sur une VM d'essai, ou après une sauvegarde.

```
# Sur la VM : arrêter l'application et supprimer ses conteneurs.  
cd /opt/nebuleuse/nextcloud && docker compose down  
  
# Supprimer ses données (IRRÉVERSIBLE).  
sudo rm -rf /opt/nebuleuse/nextcloud/html /opt/nebuleuse/nextcloud/db-data  
  
# Depuis le serveur Ansible : redéployer.  
ansible-playbook site.yml --limit client-alpha --tags nextcloud
```

Les variables d'installation (compte administrateur, base de données) ne sont lues qu'au premier démarrage. Pour les changer, il faut soit passer par l'outil du service (`occ`, interface d'administration), soit repartir de zéro comme ci-dessus.

Pour aller plus loin

- **Sauvegardes** : aucun rôle de sauvegarde n'est décrit ici. Il faudrait sauvegarder les dossiers `/opt/nebuleuse/<application>` de chaque client, ainsi que le dépôt Git et le mot de passe du Vault du serveur Ansible.
- **Supervision** : la page de la base de connaissance sur la supervision (à écrire) permettrait de surveiller les certificats, l'espace disque et les conteneurs de chaque VM.
- **Mises à jour** : les versions d'images sont figées dans les variables des rôles. Un rythme de mise à jour (mensuel, par exemple) et une procédure de retour arrière restent à définir.

Page suivante : **Fiche Atelier Penhoat.**

Clients

Fiche Atelier Penhoat

En bref
Niveau : référence
Durée : 3 minutes de lecture
Prérequis : aucun
Niveau de test : illustratif

Atelier Penhoat, c'est quoi ?

L'Atelier Penhoat est une très petite entreprise artisanale, premier client de l'offre Nébuleuse (client « alpha »). Elle utilise trois services : un espace de fichiers partagé, une galerie de photos de ses réalisations et un gestionnaire de mots de passe pour l'équipe.

Cette fiche rassemble ce qu'un technicien doit savoir avant d'intervenir sur ce client. Les valeurs sont celles utilisées dans les pages techniques du livre.

Rubrique	Valeur
Client	Atelier Penhoat
Identifiant dans l'inventaire	client-alpha
Identifiant court (conteneurs)	alpha
Domaine	penhoat.example
Contact administrateur	admin@penhoat.example
Adresse d'administration	10.99.0.21
Adresse publique	192.0.2.21

Atelier Penhoat → penhoat.example → VM client-alpha → Nextcloud, Immich, Vaultwarden

Services commandés

Service	Adresse	Rôle pour le client
---------	---------	---------------------

Authentik (SSO)	sso.penhoat.example	Connexion unique à Nextcloud et Immich
Nextcloud	cloud.penhoat.example	Fichiers, devis et plans partagés, agenda
Immich	photos.penhoat.example	Photos des réalisations, albums partagés avec les clients de l'atelier
Vaultwarden	vault.penhoat.example	Mots de passe de l'équipe

Fichiers de configuration

```
# inventory/host_vars/client-alpha/vars.yml
client_name: alpha
client_domain: penhoat.example
client_admin_email: admin@penhoat.example

nebulouse_apps:
  - nextcloud
  - immich
  - vaultwarden

client_subdomains:
  sso: sso
  nextcloud: cloud
  immich: photos
  vaultwarden: vault
```

Les secrets du client sont dans `inventory/host_vars/client-alpha/vault.yml` (chiffré). Aucun secret n'est reproduit dans cette fiche.

Points d'attention

1. **Photos volumineuses** : Immich est l'application qui grossit le plus. L'espace disque de la VM doit être surveillé, et une VM avec 4 Go de mémoire au minimum est nécessaire pour la reconnaissance de visages.
2. **Mots de passe principaux** : le client est prévenu à l'ouverture de son compte Vaultwarden qu'un mot de passe principal perdu ne peut pas être récupéré.

3. **Accès SSH** : l'administrateur passe par le bastion ; aucune connexion directe depuis Internet.

Commandes utiles

```
# Depuis le serveur Ansible : rejouer tout le déploiement pour ce client.  
ansible-playbook site.yml --limit client-alpha
```

```
# Depuis le bastion : se connecter à la VM.  
tsh ssh deploy@client-alpha
```

Page suivante : **Fiche Cabinet Kerjean.**

Fiche Cabinet Kerjean

En bref
Niveau : référence
Durée : 3 minutes de lecture
Prérequis : aucun
Niveau de test : illustratif

Cabinet Kerjean, c'est quoi ?

Le Cabinet Kerjean est un cabinet de petite taille, deuxième client de l'offre Nébuleuse (client « beta »). Il manipule des documents confidentiels et n'utilise que deux services : un espace de fichiers et un gestionnaire de mots de passe. Il n'a pas commandé la galerie de photos.

Cette fiche rassemble ce qu'un technicien doit savoir avant d'intervenir sur ce client, et sert de modèle pour les suivantes.

Rubrique	Valeur
Client	Cabinet Kerjean
Identifiant dans l'inventaire	client-beta
Identifiant court (conteneurs)	beta
Domaine	kerjean.example
Contact administrateur	admin@kerjean.example
Adresse d'administration	10.99.0.22
Adresse publique	192.0.2.22

Cabinet Kerjean → kerjean.example → VM client-beta → Nextcloud, Vaultwarden

Services commandés

Service	Adresse	Rôle pour le client
---------	---------	---------------------

Authentik (SSO)	sso.kerjean.example	Connexion unique à Nextcloud
Nextcloud	cloud.kerjean.example	Dossiers clients, échange sécurisé de documents
Vaultwarden	vault.kerjean.example	Mots de passe du cabinet

Fichiers de configuration

```
# inventory/host_vars/client-beta/vars.yml
client_name: beta
client_domain: kerjean.example
client_admin_email: admin@kerjean.example

# Immich n'est pas commandé : le rôle n'est pas déployé et aucun sous-domaine photos n'existe.
nebuleuse_apps:
  - nextcloud
  - vaultwarden

client_subdomains:
  sso: sso
  nextcloud: cloud
  vaultwarden: vault
```

Les secrets du client sont dans `inventory/host_vars/client-beta/vault.yml` (chiffré), avec des valeurs distinctes de celles du premier client. Les secrets propres à Immich n'y figurent pas.

Points d'attention

1. **Données sensibles** : les sauvegardes et la durée de conservation sont à convenir avec le client. L'accès à la VM reste tracé par le bastion.
2. **Immich non commandé** : si le client le commande plus tard, il suffit d'ajouter `immich` à `nebuleuse_apps`, un sous-domaine et deux secrets, puis de rejouer les étiquettes `sso`, `immich` et `fail2ban`.
3. **Inscriptions Vaultwarden** : fermées par défaut. Elles sont ouvertes le temps d'intégrer l'équipe du cabinet, puis refermées.

Commandes utiles

```
# Depuis le serveur Ansible : rejouer tout le déploiement pour ce client.
```

```
ansible-playbook site.yml --limit client-beta
```

```
# Depuis le bastion : se connecter à la VM.
```

```
tsh ssh deploy@client-beta
```

Page suivante : retour à la page **Présentation** du livre, ou premier livre de l'étagère **Base de connaissance**.