

Socle

- [Socle système et Docker Engine](#)
- [Bastion Teleport](#)

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 écoutera.

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**.