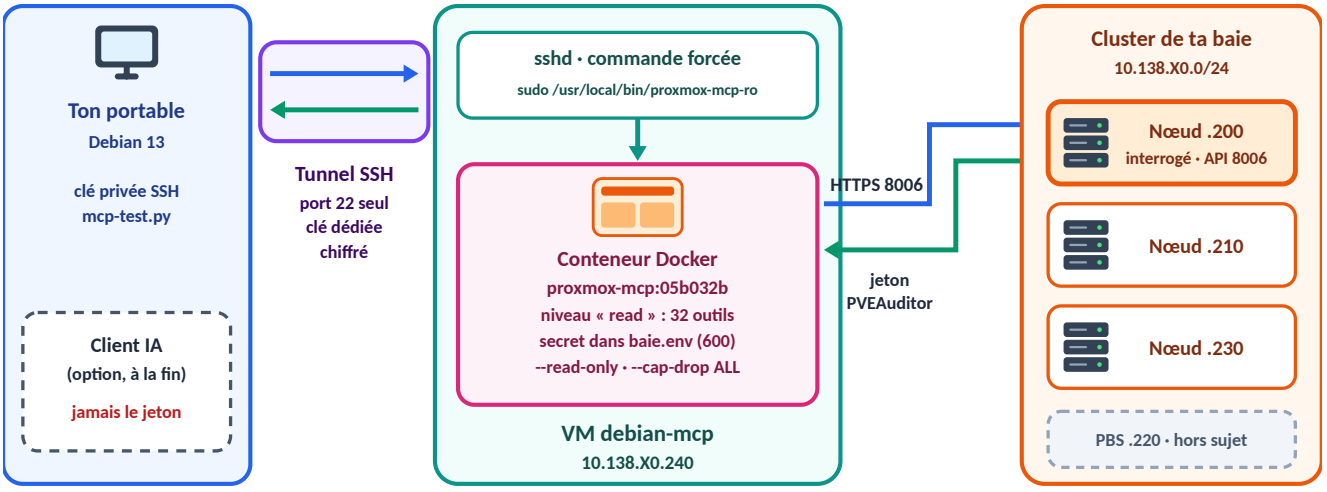
BRANCHER UNE IA EN LECTURE SEULE SUR TON CLUSTER PROXMOX (MCP)

Lié à la fiche B2-36 « Brancher une IA sur son infrastructure : le protocole MCP » · durée : 1 h 30 à 2 h · aucun compte IA nécessaire pour réussir.

CE QUE TU VAS FAIRE

Dans la fiche B2-36, tu as vu qu'une IA peut agir sur une infrastructure grâce au protocole MCP, et pourquoi on commence toujours par la **lecture seule**. Ce guide te fait **refaire la même chose sur le cluster de ta baie** : un programme posera des questions à ton Proxmox (« quels nœuds ? quelle mémoire libre ? ») sans qu'il soit **possible de rien modifier**. À la fin, tu le **prouveras** avec trois tests et les journaux de Proxmox.



Une seule porte d'entrée (SSH), un seul programme lancé (le serveur MCP), un seul jeton (lecture seule). Les flèches bleues sont les requêtes, les vertes les réponses.

RÈGLES À RESPECTER

1. ***Tu travailles uniquement sur le cluster de ta baie, jamais sur celui d'un camarade ni sur un serveur de production.***
2. ***Niveau `read` uniquement : tu ne changes jamais la valeur de `PROXMOX_RISK_LEVEL`.***
3. ***Le secret du jeton s'écrit dans un seul fichier (`baie.env`). Jamais dans un chat, un mail, un dépôt Git, une capture d'écran ou ton dossier de preuves.***

AVANT DE COMMENCER

- Le cluster de ta baie fonctionne : trois nœuds Proxmox en `10.138.X0.200`, `.210` et `.230` (et le PBS en `.220`, dont on ne se sert pas ici). `X0` est le troisième nombre de l'adresse de ta baie : `10` pour la baie 1, `20` pour la baie 2, etc. Tu as le compte `root` sur un nœud.
- Ton portable est sous **Debian 13**, avec `ssh` et `python3` (déjà installés : vérifie avec `ssh -V` et `python3 --version`).
- Tu vas créer une **VM Debian 13 « debian-mcp »** sur ton cluster (2 vCPU, 2 Go de RAM, 20 Go de disque), avec un **accès à Internet** : elle télécharge des paquets et du code.
- Tu as lu la fiche B2-36, au moins les chapitres sur les trois rôles (hôte, client, serveur) et sur les sept verrous.

Les repères de ce guide. Chaque étape porte une pastille qui dit où taper les commandes, et chaque bloc de code a le même code couleur : NŒUD PROXMOX VM DEBIAN-MCP TON PORTABLE. Les valeurs **surlignées** sont à remplacer par les tiennes. Le cadre vert « **Tu dois voir** » te dit comment savoir que l'étape est réussie : ne passe jamais à la suivante sans l'avoir obtenu.

Étape	Ce que tu fais	Où	Ce que tu obtiens
1	Créer un jeton en lecture seule	NŒUD PROXMOX	L'identité `mcp-ro@pve!lecture` et son secret
2	Préparer la VM debian-mcp	VM	Debian à jour, Docker, utilisateur `mcp`
3	Construire l'image Docker, code relu	VM	Image `proxmox-mcp:05b032b`
4	Secret, script de lancement, sudo limité	VM	Un seul script autorisé, secret en 600
5	Clé SSH et commande forcée	PORTABLE VM	Une clé qui ne sait lancer que le serveur MCP
6	Tester sans IA	PORTABLE	Liste des outils, nœuds, outil d'écriture absent
7	Prouver par les journaux de Proxmox	NŒUD VM	Des GET seulement, une écriture refusée (403)
+	Brancher un client IA (option)	PORTABLE	Des questions en langage naturel

Ma feuille de valeurs

Élément	Valeur type	Ma valeur
Adresse de ma baie	`10.138.X0.0/24`	
Nœud interrogé par le serveur MCP	`10.138.X0.200`	
Adresse de la VM debian-mcp	`10.138.X0.240/24`	
Passerelle et DNS de la VM	comme tes autres VM	
Nom du nœud qui héberge debian-mcp, et son VMID	**NOM-DU-NOEUD** et **VMID**	

1 CRÉER L'ACCÈS EN LECTURE SEULE

SUR UN NŒUD PROXMOX

Connecte-toi en `root` sur le nœud `10.138.X0.200` (SSH, ou console depuis l'interface web). Dans un cluster, les utilisateurs et les jetons sont partagés entre les nœuds : **tu fais cette étape une seule fois**, sur un seul nœud.

Sur le nœud, en root

```
set +H     # empêche bash d'interpréter le « ! » du nom de jeton
pveum user add mcp-ro@pve --comment "Lecture seule pour MCP"
pveum acl modify / --users mcp-ro@pve --roles PVEAuditor
pveum user token add mcp-ro@pve lecture --privsep 1
pveum acl modify / --tokens 'mcp-ro@pve!lecture' --roles PVEAuditor
```

Le rôle **PVEAuditor** n'a que des droits d'audit, c'est-à-dire de lecture. Avec `--privsep 1`, les droits réels du jeton sont l'**intersection** de ceux de l'utilisateur et de ceux du jeton : c'est pourquoi on donne le rôle aux deux. Si l'un des deux était trop large, l'autre limiterait encore.

TU DOIS VOIR

Après la 3e commande, un tableau avec les lignes `full-tokenid` (`mcp-ro@pve!lecture`), `info` et `value`. La valeur de `value` (un identifiant du type `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`) est le **secret : il n'est affiché qu'une seule fois**. Copie-le tout de suite dans un fichier temporaire de ton portable, protégé et hors de tout dossier partagé ; tu l'utiliseras à l'étape 4, puis tu supprimeras ce fichier.

Vérification : `pveum acl list` affiche deux lignes avec le rôle `PVEAuditor` sur le chemin `/`, l'une pour l'utilisateur `mcp-ro@pve`, l'autre pour le jeton `mcp-ro@pve!lecture`.

SI TU PERDS LE SECRET

Il est impossible de le relire. Supprime le jeton (`pveum user token remove mcp-ro@pve lecture`), recrée-le avec la 3e commande, puis redonne-lui le rôle avec la 4e.

2 PRÉPARER LA VM DEBIAN-MCP

SUR LA VM

Crée la VM comme d'habitude sur ton cluster (Debian 13, installation *serveur* : serveur SSH et utilitaires usuels, pas de bureau). Donne-lui l'adresse fixe `10.138.X0.240/24`, avec la passerelle et le DNS de tes autres VM. Connecte-toi ensuite depuis ton portable : `ssh TON-COMPTE@10.138.X0.240`.

Sur la VM, avec ton compte administrateur

```
sudo apt update && sudo apt -y upgrade
sudo apt install -y docker.io git curl fail2ban unattended-upgrades
echo 'APT::Periodic::Update-Package-Lists "1"; APT::Periodic::Unattended-Upgrade "1";' \
  | sudo tee /etc/apt/apt.conf.d/20auto-upgrades
sudo systemctl enable --now docker fail2ban
sudo adduser --disabled-password --gecos "" mcp
```

L'utilisateur `mcp` n'a **pas de mot de passe** (il ne se connectera que par clé) et il n'est **pas dans le groupe `docker`** : un membre de ce groupe peut lancer un conteneur qui monte le disque de la machine, ce qui équivaut à être `root`. On donnera donc à `mcp` un `sudo` limité à un seul script (étape 4).

Vérifie la VM, puis ses accès réseau :

Sur la VM

```
id mcp
sudo fail2ban-client status sshd
curl -s -o /dev/null -w "%{http_code}\n" https://github.com
curl -sk -o /dev/null -w "%{http_code}\n" https://10.138.X0.200:8006/
```

TU DOIS VOIR

- `id mcp` : `uid=…(mcp) gid=…(mcp) groupes=…(mcp)`, avec **un seul groupe** et surtout pas `docker`.
- `fail2ban-client` : « Status for the jail: sshd ».
- Les deux `curl` : `200`. Un échec sur le premier signifie que la VM n'a pas Internet (passerelle, DNS) ; sur le second, une mauvaise adresse de nœud ou un réseau qui ne passe pas.

À SAVOIR

Si `sudo` n'est pas installé sur ta VM : `su -`, puis `apt install -y sudo` et `usermod -aG sudo TON-COMPTE`. Déconnecte-toi, reconnecte-toi, et reprends l'étape.

3 CONSTRUIRE L'IMAGE DOCKER, APRÈS AVOIR RELU LE CODE

SUR LA VM

Le serveur MCP est un programme open source (`akmalovaa/proxmox-mcp`) écrit par quelqu'un d'autre. Tu ne le lances donc pas à l'aveugle : tu **fixes un commit précis**, tu **relis ce qu'il peut faire**, puis tu construis l'image toi-même. On n'utilise jamais l'étiquette `:latest`, qui change sans prévenir.

Sur la VM, avec ton compte administrateur

```
cd ~
git clone https://github.com/akmalovaa/proxmox-mcp.git proxmox-mcp-src
cd proxmox-mcp-src
git checkout --detach 05b032b
git log -1 --format='%h  %s'
```

Le commit `05b032b` est celui de la version **2.3.0**. Audite maintenant le code : trois comptages suffisent à vérifier que les appels qui modifient quelque chose sont réservés à des outils qui n'existeront pas au niveau « read ».

Sur la VM, dans proxmox-mcp-src

```
grep -rn '@tool(tier="read"' src | wc -l
grep -rn '@tool(tier="lifecycle"\|@tool(tier="all"' src | wc -l
grep -rn '\.post(\|\.put(\|\.delete(' src | wc -l
sed -n '118,119p' src/proxmox_mcp/tools/vms.py
```

TU DOIS VOIR

- `05b032b Add streamable-http transport, health endpoints, tier/annotation split` pour le `git log`.
- Les trois comptages : **32** outils de lecture, **18** outils d'écriture (niveaux « lifecycle » et « all »), et **18** appels qui écrivent : chaque écriture correspond à un outil d'un niveau élevé.
- Les deux lignes de `vms.py` : l'outil `stop_vm`, déclaré avec `tier="lifecycle"`.

Construis l'image (compte quelques minutes : le build télécharge Python, `uv` et les dépendances) :

Sur la VM, dans proxmox-mcp-src

```
sudo docker build -t proxmox-mcp:05b032b .
sudo docker image ls proxmox-mcp
```

TU DOIS VOIR

À la fin du build, le nom `proxmox-mcp:05b032b` ; puis `docker image ls` affiche une ligne avec le tag `05b032b`. Un avertissement « legacy builder is deprecated » n'est pas une erreur. Une erreur de téléchargement (`ghcr.io`, `docker.io`) indique un problème d'accès à Internet ou de DNS sur la VM.

4 SECRET, SCRIPT DE LANCEMENT ET SUDO LIMITÉ

SUR LA VM

4.1 · Le fichier du secret

Sur la VM

```
sudo install -d -m 700 -o mcp -g mcp /home/mcp/mcp-proxmox
sudo -u mcp nano /home/mcp/mcp-proxmox/baie.env
sudo chmod 600 /home/mcp/mcp-proxmox/baie.env
```

Dans nano, écris ces six lignes **sans guillemets ni espaces autour du `=`**, puis enregistre (Ctrl+O, Entrée, Ctrl+X). Le nom du jeton est `lecture`, pas `mcp-ro@pve!lecture` : l'utilisateur et le nom du jeton sont deux variables distinctes.

Contenu de /home/mcp/mcp-proxmox/baie.env

```
PROXMOX_HOST=10.138.X0.200
PROXMOX_USER=mcp-ro@pve
PROXMOX_TOKEN_NAME=lecture
PROXMOX_TOKEN_VALUE=SECRET-NOTÉ-À-L'ÉTAPE-1
PROXMOX_VERIFY_SSL=false
PROXMOX_RISK_LEVEL=read
```

`PROXMOX_VERIFY_SSL=false` : les nœuds de la baie ont un certificat auto-signé, que le serveur ne peut pas vérifier. C'est acceptable sur un réseau de labo ; en production, on installerait un vrai certificat. `PROXMOX_RISK_LEVEL=read` est **le verrou central** : à ce niveau, les outils d'écriture ne sont pas enregistrés, donc absents pour le modèle. Une fois le secret copié, **supprime ton fichier temporaire** de l'étape 1.

4.2 · Le script de lancement

Sur la VM

```
sudo tee /usr/local/bin/proxmox-mcp-ro >/dev/null <<'EOF'
#!/bin/sh
exec docker run --rm -i \
  --env-file /home/mcp/mcp-proxmox/baie.env \
  --read-only --tmpfs /tmp --cap-drop ALL \
  --security-opt no-new-privileges \
  --pids-limit 100 --memory 256m \
  proxmox-mcp:05b032b
EOF
sudo chown root:root /usr/local/bin/proxmox-mcp-ro
sudo chmod 755 /usr/local/bin/proxmox-mcp-ro
```

Option Docker	Rôle
`-i` (et surtout pas `-t`)	Garde l'entrée standard ouverte : MCP parle par `stdin` / `stdout`. Un terminal (`-t`) corromprait les échanges.
`--rm`	Le conteneur disparaît à la fin de la session : rien ne reste.
`--env-file`	Le secret entre par l'environnement, il n'est jamais dans la commande.
`--read-only`, `--tmpfs /tmp`	Système de fichiers en lecture seule, sauf un `/tmp` en mémoire.
`--cap-drop ALL`, `no-new-privileges`	Aucun privilège Linux, et impossible d'en gagner.
`--pids-limit`, `--memory`	Limites de processus et de mémoire contre les emballements.

Teste le script en local, **avant d'y ajouter SSH** : tu sépares ainsi les pannes de Docker de celles du réseau. Le fichier `init.json` contient le message d'ouverture de session MCP.

Sur la VM

```
cat > ~/init.json <<'EOF'
{"jsonrpc":"2.0","id":1,"method":"initialize","params":{"protocolVersion":"2025-03-
26","capabilities":{},"clientInfo":{"name":"test","version":"0"}}}
EOF
sudo /usr/local/bin/proxmox-mcp-ro < ~/init.json
```

TU DOIS VOIR

Une ligne de JSON qui commence par `{"jsonrpc":"2.0","id":1,"result":…` et contient `"serverInfo"` (le nom et la version du serveur). Si rien ne s'affiche, ou si tu lis une erreur, regarde le tableau de dépannage en fin de guide.

4.3 · Un sudo pour ce script, et rien d'autre

Sur la VM

```
echo 'mcp ALL=(root) NOPASSWD: /usr/local/bin/proxmox-mcp-ro' \
  | sudo tee /etc/sudoers.d/proxmox-mcp-ro
sudo chmod 440 /etc/sudoers.d/proxmox-mcp-ro
sudo visudo -cf /etc/sudoers.d/proxmox-mcp-ro
sudo -l -U mcp
```

TU DOIS VOIR

`/etc/sudoers.d/proxmox-mcp-ro: parsed OK`, puis, pour `sudo -l -U mcp`, une seule commande autorisée : `(root) NOPASSWD: /usr/local/bin/proxmox-mcp-ro`.

5 UNE CLÉ SSH QUI NE SAIT LANCER QU'UN PROGRAMME

SUR TON PORTABLE, PUIS SUR LA VM

Tu crées une clé réservée à cet usage. Laisse la **passphrase vide** (Entrée deux fois) : le client lancera `ssh` tout seul, sans te poser de question. Ce n'est pas dangereux ici, parce que la ligne `authorized_keys` que tu vas écrire limite cette clé à une seule commande.

Sur ton portable

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_mcp_baie -C "mcp-baie-$(whoami)@$(hostname)"
cat ~/.ssh/id_ed25519_mcp_baie.pub
```

La deuxième commande affiche la clé **publique** sur une ligne : `ssh-ed25519 AAAA… mcp-baie-…`. Ne confonds pas : la clé privée (sans `.pub`) **ne quitte jamais ton portable**. Sur la VM, tu vas écrire **une seule ligne** dans le fichier des clés autorisées de `mcp`, en ajoutant devant la clé publique les deux options qui font toute la sécurité :



Une seule ligne, sans retour à la ligne, dans /home/mcp/.ssh/authorized_keys

N°	Morceau	Effet
1	`restrict,`	Désactive d'un coup le terminal, les redirections de ports, l'agent SSH et X11 : cette clé ne sert plus qu'à lancer une commande.
2	`command="…"`	Impose la commande à exécuter, **quelle que soit celle demandée par le client** : il n'y a pas de shell, seulement ce serveur.
3	la clé publique	Identifie ton portable : c'est ce qui rend la clé privée utile.
4	le commentaire	Un nom lisible. Si ce portable est volé, c'est la ligne à supprimer.

Sur la VM, avec ton compte administrateur

```
sudo install -d -m 700 -o mcp -g mcp /home/mcp/.ssh
sudo -u mcp nano /home/mcp/.ssh/authorized_keys
sudo chmod 600 /home/mcp/.ssh/authorized_keys
```

Dans nano, colle la ligne suivante en remplaçant `ssh-ed25519 AAAA… mcp-baie-…` par la clé publique copiée sur ton portable :

Contenu de /home/mcp/.ssh/authorized_keys (une seule ligne)

```
restrict,command="sudo /usr/local/bin/proxmox-mcp-ro" ssh-ed25519 AAAA… mcp-baie-toi@portable
```

PIÈGE CLASSIQUE

Si le copier-coller a coupé la ligne en deux, la connexion échoue avec `Permission denied (publickey)`. La ligne doit être **d'un seul tenant** : vérifie qu'il n'y a pas de retour à la ligne caché au milieu de la clé.

6 TESTER SANS IA

SUR TON PORTABLE

Un client MCP n'est rien de plus qu'un programme qui **lance une commande et lui parle ligne à ligne** en JSON. Tu n'as donc pas besoin d'IA pour tester : le script `mcp-test.py` (fourni, reproduit en annexe) joue le rôle du client et affiche ce que l'IA verrait.

6.1 · La clé ne sert qu'à une chose

Sur ton portable

```
ssh -T -i ~/.ssh/id_ed25519_mcp_baie -o IdentitiesOnly=yes mcp@10.138.X0.240 id < /dev/null
```

À la première connexion, ssh demande de confirmer l'empreinte de la VM : réponds `yes`. Tu demandes ici la commande `id`… mais la commande forcée la remplace.

TU DOIS VOIR

Aucune ligne `uid=…`. Ta demande a été ignorée : la clé ne peut pas obtenir de shell. (Un message d'arrêt du serveur MCP, car son entrée est vide, est normal.)

6.2 · Le client de test

Sur ton portable, dans le dossier qui contient mcp-test.py

```
python3 mcp-test.py ssh -T -i ~/.ssh/id_ed25519_mcp_baie \
  -o IdentitiesOnly=yes -o BatchMode=yes mcp@10.138.X0.240
```

Le script lance la commande qui suit son nom, puis déroule cinq échanges. Pour chacun, compare à ce tableau :

N°	Message envoyé	Tu dois voir
1	`initialize`	Le nom et la version du serveur (**2.3.0**) : le client et le serveur se sont présentés.
2	`tools/list`	Une trentaine d'outils (`list_nodes`, `get_vm_status`…) et la ligne **« Outils d'écriture dans la liste : AUCUN »**.
3	`tools/call list_nodes`	Les **trois nœuds de ta baie**, avec leur état : une vraie requête est partie vers l'API Proxmox.
4	`tools/call get_server_info`	Le serveur annonce lui-même sa version et son niveau de risque : `read`.
5	`tools/call stop_vm`	Un refus du type « Unknown tool », puis **« OK : stop_vm n'existe pas pour le modèle »**.

OUTIL REFUSÉ OU OUTIL ABSENT ?

Ce n'est pas la même chose. Un outil **refusé** existe dans la liste : l'IA peut l'appeler en boucle ou chercher un contournement. Un outil **absent** n'existe pas pour elle : elle ne peut même pas le demander. Ici, le niveau `read` retire les outils d'écriture de la liste. Si le script affiche « ATTENTION », ton fichier `baie.env` n'est pas au niveau `read` : corrige-le avant d'aller plus loin.

Fais une capture de cette sortie pour ton dossier de preuves : c'est la preuve n° 1.

7 PROUVER PAR LES JOURNAUX DE PROXMOX

NŒUD PROXMOX, PUIS VM

Dire « c'est verrouillé » ne suffit pas : un administrateur le **prouve avec une source indépendante de l'IA**, ici le journal du proxy web de Proxmox (`pveproxy`), qui note toutes les requêtes faites avec ton jeton.

7.1 · Compter les requêtes du jeton

Sur le nœud 10.138.X0.200, en root

```
zcat -f /var/log/pveproxy/access.log* \
  | awk '$3 ~ /mcp-ro@pve/ {print $6, $9}' | sort | uniq -c | sort -rn
```

TU DOIS VOIR

Des lignes du type `12 "GET 200` : le nombre de requêtes, la méthode HTTP, le code de réponse. Il n'y a que des **GET** (lecture), aucun `POST`, `PUT` ni `DELETE`. Ces requêtes sont celles du test de l'étape 6 : c'est la preuve n° 2.

7.2 · Tenter une écriture, exprès

Ce test contourne le serveur MCP : il tape **directement l'API Proxmox avec le jeton**, pour vérifier que c'est Proxmox lui-même qui refuse d'écrire. Il tente seulement de modifier la *description* de la VM `debian-mcp`, un changement sans conséquence si, par erreur, il était accepté.

Sur la VM debian-mcp

```
set +H
read -rsp "Secret du jeton : " S; echo
curl -sk -i -X PUT -d "description=test-lecture-seule" \
  -H "Authorization: PVEAPIToken=mcp-ro@pve!lecture=$S" \
  https://10.138.X0.200:8006/api2/json/nodes/NOM-DU-NOEUD/qemu/VMID/config | head -1
unset S
```

TU DOIS VOIR

`HTTP/1.1 403 Permission check failed (…)` : c'est la preuve n° 3. **Si tu lis 200**, le jeton a trop de droits : reprends l'étape 1. Relance ensuite la commande du 7.1 : une ligne `PUT 403` est apparue. Le refus est inscrit dans le journal de Proxmox.

POURQUOI DEUX PROTECTIONS ?

Le niveau `read` empêche l'IA de *demander* une écriture (l'outil est absent). Le jeton `PVEAuditor` empêche Proxmox de l'*exécuter*, même si un jour le serveur était mal configuré. Chaque verrou suppose que le précédent peut céder : c'est la défense en profondeur de la fiche B2-36.

+ OPTION : BRANCHER UN CLIENT IA

SUR TON PORTABLE

Si tu disposes d'un client IA compatible MCP, tu peux lui donner exactement la même commande que celle de `mcp-test.py`. Le principe est identique dans tous les clients : un nom de serveur, la commande `ssh` et ses arguments. Selon ton système, ce sera le fichier de configuration d'une application de bureau ou celui d'un outil en ligne de commande ; consulte l'aide de ton client pour son emplacement.

Bloc de configuration à adapter (format « mcpServers » courant)

```
{
  "mcpServers": {
    "proxmox-baie": {
      "command": "ssh",
      "args": ["-T", "-i", "/home/TON-COMPTE/.ssh/id_ed25519_mcp_baie",
               "-o", "IdentitiesOnly=yes", "-o", "BatchMode=yes",
               "mcp@10.138.X0.240"]
    }
  }
}
```

- Utilise un **chemin absolu** pour la clé : le `~` n'est pas développé dans cette liste.
- Ferme le client avant de modifier le fichier, puis relance-le. Les outils du serveur apparaissent dans sa liste.
- Pose une question en lecture : « *Liste mes nœuds Proxmox et la mémoire libre de chacun.* » Puis essaie une écriture : « *Arrête la VM 100.* » L'assistant doit répondre qu'il n'a pas d'outil pour cela.
- Refais le comptage du 7.1 : les nouvelles requêtes sont toutes des `GET`.

ATTENTION AUX DONNÉES

Les noms de VM, leurs états et leurs configurations quittent ton labo pour le fournisseur de l'IA. Tu ne le fais que sur le cluster de ta baie, jamais sur un environnement réel. Et rappelle-toi : le modèle n'est pas un mécanisme de sécurité. Ce qui protège, ce sont les droits, les filtres et la configuration.

EN CAS DE PROBLÈME

Ce que tu vois	Cause probable	Que faire
`Permission denied (publickey)`	Ligne `authorized_keys` coupée, ou droits trop larges	Une seule ligne. `.ssh` en 700 et `authorized_keys` en 600, propriétaire `mcp`. Essaie avec `ssh -v`.

Ce que tu vois	Cause probable	Que faire
`Host key verification failed`	L'empreinte de la VM n'a pas encore été acceptée	Refais le test 6.1 sans `BatchMode` et réponds `yes`.
`sudo: a password is required`	Fichier sudoers absent, ou chemin différent de celui du `command=`	`sudo visudo -cf /etc/sudoers.d/proxmox-mcp-ro`. Compare les deux chemins, caractère par caractère.
Rien ne s'affiche au test local (4.2)	Fichier `baie.env` absent ou illisible, image absente	Regarde le message d'erreur affiché. `sudo docker image ls proxmox-mcp` doit montrer le tag `05b032b`.
`Unable to find image 'proxmox-mcp:…'`	Le tag du script et celui de l'image diffèrent	Corrige le tag dans `/usr/local/bin/proxmox-mcp-ro`.
Erreur `401` dans `list_nodes`	Secret ou noms erronés	Vérifie `PROXMOX_USER=mcp-ro@pve` et `PROXMOX_TOKEN_NAME=lecture`. Secret perdu : recrée le jeton (étape 1).
Délai dépassé vers le port 8006	Mauvaise adresse de nœud, ou réseau fermé	Rejoue le `curl` de l'étape 2 depuis la VM.
`event not found` après un `!`	bash interprète le point d'exclamation	Tape `set +H` dans la session, puis relance la commande.
`Le serveur s'est arrêté` (mcp-test.py)	La commande lancée a échoué	Lis les messages affichés juste au-dessus (ils viennent de ssh ou de Docker) et reprends le tableau.

DOSSIER DE PREUVES ET FIN DE SÉANCE

Ton dossier contient **trois captures**, sans aucun secret visible : la sortie de `mcp-test.py` (preuve n° 1), le comptage des requêtes du jeton dans le journal (preuve n° 2) et le `403` de l'écriture refusée (preuve n° 3). Ajoute deux phrases qui expliquent, avec tes mots, pourquoi un outil *absent* protège mieux qu'un outil *refusé*.

Pour **révoquer** l'accès quand tu n'en as plus besoin (c'est la bonne habitude, et la première chose à faire si ton portable est perdu) :

Sur la VM : retirer la clé du portable

```
sudo -u mcp sh -c ': > /home/mcp/.ssh/authorized_keys'
```

Sur un nœud Proxmox : supprimer le jeton, puis l'utilisateur

```
pveum user token remove mcp-ro@pve lecture
pveum user delete mcp-ro@pve
```

POUR ALLER PLUS LOIN, AVEC L'ACCORD DE TON ENSEIGNANT

Tu peux aussi **restreindre l'accès au port 8006** des nœuds aux seules adresses utiles, avec les variables `ALLOW_FROM`, `DENY_FROM` et `POLICY` du fichier `/etc/default/pveproxy` (puis `systemctl restart pveproxy`). Ce fichier est propre à chaque nœud, à refaire sur chacun. **Attention :** y faire figurer ton portable, la VM debian-mcp et les autres nœuds, sinon tu perds l'accès à l'interface web (le SSH reste possible pour corriger).

OÙ EST QUOI ?

Élément	Où	Ce qu'il contient	Droits
`id_ed25519_mcp_baie`	Portable, `~/.ssh/`	La clé **privée** : ne quitte jamais le portable	600
`authorized_keys`	VM, `/home/mcp/.ssh/`	Clé publique + `restrict` + commande forcée	600
`proxmox-mcp-ro`	VM, `/usr/local/bin/`	La commande `docker run` durcie	755 root
`proxmox-mcp-ro`	VM, `/etc/sudoers.d/`	Une seule règle : `mcp` peut lancer ce script	440
`baie.env`	VM, `/home/mcp/mcp-proxmox/`	**Le secret du jeton**, le niveau `read`	600 mcp
`mcp-ro@pve!lecture`	Cluster Proxmox	Jeton avec rôle `PVEAuditor`, privilèges séparés	—

ANNEXE · LE CLIENT DE TEST `MCP-TEST.PY`

Le fichier est fourni avec ce guide. Le recopier depuis le PDF risque de casser l'indentation, qui compte en Python : prends le fichier, ne le retape pas. Il ne dépend que de la bibliothèque standard de `python3`.

mcp-test.py (listing complet)

```
#!/usr/bin/env python3
# mcp-test.py : rejoue à la main l'échange MCP de la fiche B2-36, sans IA.
# Usage : python3 mcp-test.py <commande qui lance le serveur MCP>
# Exemple : python3 mcp-test.py ssh -T -i ~/.ssh/id_ed25519_mcp_baie -o IdentitiesOnly=yes -o BatchMode=yes
mcp@10.138.10.240
import json
import os
import select
import subprocess
import sys

ECRITURE = ("start_", "stop_", "shutdown_", "reboot_", "suspend_", "resume_",
            "clone_", "migrate_", "create_", "delete_", "rollback_")

if len(sys.argv) < 2:
    sys.exit("Usage : python3 mcp-test.py <commande qui lance le serveur MCP>")

# 1. On lance la commande, comme le fait un hôte MCP : on parle à son stdin / stdout.
serveur = subprocess.Popen(sys.argv[1:], stdin=subprocess.PIPE, stdout=subprocess.PIPE)
fd = serveur.stdout.fileno()
tampon = b""
numero = 0


def envoyer(methode, params=None, notification=False):
    """Écrit UNE ligne de JSON sur stdin du serveur. Une notification n'a pas d'id : aucune réponse n'est
attendue."""
    global numero
    msg = {"jsonrpc": "2.0", "method": methode}
    if params is not None:
        msg["params"] = params
    if not notification:
        numero += 1
        msg["id"] = numero
    ligne = json.dumps(msg, ensure_ascii=False)
    print("→ " + (ligne if len(ligne) < 130 else ligne[:127] + "..."))
    serveur.stdin.write((ligne + "\n").encode("utf-8"))
    serveur.stdin.flush()
    return numero


def ligne_suivante(delai):
    """Lit UNE ligne sur stdout du serveur (None si rien n'arrive dans le délai, '' si le serveur s'est
arrêté)."""
    global tampon
    while b"\n" not in tampon:
        pret, _, _ = select.select([fd], [], [], delai)
        if not pret:
            return None
        bloc = os.read(fd, 65536)
        if not bloc:
```

```
            return ""
        tampon += bloc
    ligne, tampon = tampon.split(b"\n", 1)
    return ligne.decode("utf-8", "replace")


def reponse(id_attendu, delai=40):
    """Attend la réponse qui porte l'identifiant attendu (les autres lignes sont affichées puis ignorées)."""
    while True:
        ligne = ligne_suivante(delai)
        if ligne is None:
            sys.exit("Pas de réponse du serveur au bout de %d secondes." % delai)
        if ligne == "":
            sys.exit("Le serveur s'est arrêté : relis les messages d'erreur affichés plus haut.")
        try:
            msg = json.loads(ligne)
        except ValueError:
            print("   (ligne reçue qui n'est pas du JSON : %r)" % ligne[:100])
            continue
        if msg.get("id") == id_attendu:
            return msg
        print("   (message ignoré : %s)" % (msg.get("method") or "sans méthode"))


def texte(msg):
    """Texte renvoyé par un outil (premier bloc de contenu)."""
    for bloc in (msg.get("result") or {}).get("content", []):
        if bloc.get("type") == "text":
            return bloc.get("text", "")
    return ""


print("== 1. initialize : le client se présente, le serveur répond")
rep = reponse(envoyer("initialize", {"protocolVersion": "2025-03-26", "capabilities": {},
                                     "clientInfo": {"name": "mcp-test", "version": "1"}}))
if "error" in rep:
    sys.exit("Erreur du serveur : %s" % rep["error"])
info = rep["result"].get("serverInfo", {})
print("← serveur : %s, version %s, protocole %s" % (info.get("name"), info.get("version"),
rep["result"].get("protocolVersion")))
envoyer("notifications/initialized", notification=True)

print("\n== 2. tools/list : quels outils le modèle verrait-il ?")
noms, curseur = [], None
while True:
    rep = reponse(envoyer("tools/list", {"cursor": curseur} if curseur else {}))
    noms += [t["name"] for t in rep["result"]["tools"]]
    curseur = rep["result"].get("nextCursor")
    if not curseur:
        break
print("← %d outils : %s" % (len(noms), ", ".join(noms)))
ecr = [n for n in noms if n.startswith(ECRITURE)]
print("   Outils d'écriture dans la liste : %s" % (", ".join(ecr) if ecr else "AUCUN (niveau « read » : ils sont
absents)"))

print("\n== 3. tools/call list_nodes : une vraie requête vers l'API Proxmox")
rep = reponse(envoyer("tools/call", {"name": "list_nodes", "arguments": {}}))
t = texte(rep) or json.dumps(rep.get("error", rep), ensure_ascii=False)
print("← " + (t if len(t) < 600 else t[:600] + " [...]"))

print("\n== 4. tools/call get_server_info : le serveur annonce lui-même son niveau de risque")
rep = reponse(envoyer("tools/call", {"name": "get_server_info", "arguments": {}}))
t = texte(rep) or json.dumps(rep.get("error", rep), ensure_ascii=False)
print("← " + (t if len(t) < 600 else t[:600] + " [...]"))

print("\n== 5. tools/call stop_vm : on essaie d'arrêter une VM qui n'existe pas (VM 999999)")
rep = reponse(envoyer("tools/call", {"name": "stop_vm", "arguments": {"node": "inconnu", "vmid": 999999}}))
t = texte(rep) or json.dumps(rep.get("error", ""), ensure_ascii=False)
print("← " + (t if len(t) < 300 else t[:300] + " [...]"))
if "stop_vm" in noms:
    print("   ATTENTION : l'outil stop_vm EXISTE pour le modèle. Le niveau n'est pas « read » : corrige
PROXMOX_RISK_LEVEL dans le fichier .env.")
else:
    print("   OK : stop_vm n'existe pas pour le modèle. Il ne peut même pas le demander.")

serveur.stdin.close()
try:
    serveur.wait(timeout=10)
except subprocess.TimeoutExpired:
    serveur.kill()
print("\nTerminé : fais une capture de cette sortie pour ton dossier de preuves.")
```