

INCUS – Hôte et conteneur & serveur NAS

Guide de reconstruction. Il décrit comment rebâtir nas-host tel qu'il fonctionne, du système nu jusqu'aux conteneurs en service.

△ Treize lignes de ce document dépassent 84 caractères et s'enrouleront dans le PDF sans signe visible : douze des treize lignes de cron, et la ligne `mp3fs` d'`/etc/fstab`. Ni cron ni `fstab` n'acceptent de continuation – les recopier depuis le fichier `.md`, jamais depuis le PDF. Toutes les autres commandes longues ont été coupées par continuation `\`.

Le fichier `.md` de cette page : [ouvrir le .md](#)

Ce que cette page couvre : rebâtir de bout en bout une machine qui cumule deux rôles – serveur de fichiers pour un réseau domestique, et hôte de conteneurs système. Du matériel nu jusqu'aux conteneurs en service, avec les sauvegardes qui vont avec.

Contexte : Ubuntu Server 24.04 sur un Supermicro X11SAE-M, Incus 6.0 LTS du dépôt Zabbly, cinq conteneurs en macvlan, un disque de données de 12,7 To partagé en SMB pour les postes Windows et en NFS pour les postes Linux. Les adresses, noms d'hôtes et identifiants de disques sont des exemples : les transposer à son propre parc.

Trois choses valent le détour même sans suivre le guide de bout en bout. Pourquoi un hôte ne peut pas joindre ses propres conteneurs macvlan, et le montage à deux cartes réseau qui lève la difficulté – section *Réseau*. Le réglage Samba d'une seule ligne sans lequel les fichiers de verrouillage des suites bureautiques s'accumulent indéfiniment, jusqu'à rendre des documents illisibles en écriture pendant des années – section *Partages Samba*. Et le fait que le filtrage des modules rsync s'applique côté serveur et non côté client, ce qui décide de ce qu'un mot de passe compromis donnerait réellement – section

Sauvegardes.

Convention : chaque bloc de commandes indique la machine où il s'exécute. Un bloc unique se copie tel quel ; des blocs séparés signifient qu'une édition ou une décision intervient entre eux.

Ce que cette machine fait

nas-host porte deux rôles qu'on aurait pu séparer sur deux machines, et qui partagent ici le même matériel.

Serveur de fichiers. Un disque de 12,7 To, partagé en SMB pour les postes Windows et en NFS pour les postes Linux, avec un miroir MP3 transcodé à la volée pour les lecteurs qui ne lisent pas le FLAC.

Hôte de conteneurs. Cinq conteneurs Incus en macvlan, chacun avec sa propre adresse sur le réseau domestique : un serveur web, un Pi-hole, un mandataire Squid, un serveur de musique et un correcteur linguistique.

S'y ajoutent des services propres à l'hôte : diffusion audio par Icecast, serveur de musique Lyrion, indexation MusicIP, partage DLNA, surveillance de l'onduleur, et la synchronisation hors site vers Proton Drive.

Cette cohabitation a une conséquence structurelle : l'hôte ne peut pas devenir un système immuable de type IncusOS, puisque la moitié de ce qu'il fait vit en dehors des conteneurs.

Matériel et contraintes

Élément	Valeur
Carte mère	Supermicro X11SAE-M, chipset C236
Processeur	Intel Xeon E3-1230 v5, 4 cœurs / 8 threads
Mémoire	31 Gio

Élément	Valeur
Système	NVMe Samsung 970 EVO Plus, 465 Go
Données	WD 12,7 To en SATA
Disque web	Seagate 1,8 To en SATA, dédié au conteneur web
Démarrage	BIOS hérité, <code>/boot</code> sur une clé USB

La contrainte de démarrage gouverne toute l'installation. La carte mère possède un connecteur M.2 mais ne sait pas démarrer dessus en BIOS hérité, et sa mise à jour de BIOS est impossible. La solution est une clé USB rapide qui porte `/boot` et GRUB ; le système lui-même vit sur le NVMe.

Cette clé doit rester branchée en permanence. C'est le point de fragilité de la machine : si elle lâche, nas-host ne démarre plus. Elle est pour cette raison imagée et répliquée hors site – et non sur un disque de la machine qui, elle, ne démarrerait plus pour aller le lire.

L'image couvre le périphérique entier, table de partition et secteur d'amorçage compris : copier seulement `/boot` donnerait une clé pleine de fichiers justes qui ne démarrerait pas. La procédure, le rafraîchissement automatique au changement de noyau et la compression sont décrits dans la page [Sauvegarde](#).

Prérequis pour rebâtir :

- une carte NVMe pour le système
 - une clé USB 3.1 de 16 Go pour `/boot` et GRUB
 - une clé USB d'installation d'Ubuntu Server 24.04
-

Installer le système

Au partitionnement, trois décisions :

1. Une partition unique sur le NVMe, montée sur `/`

2. La clé USB configurée comme `/boot`

3. GRUB installé sur la clé USB, pas sur le NVMe

Ajouter le serveur OpenSSH quand l'installateur le propose : la machine n'a pas d'écran en service normal.

Dans le BIOS ensuite, régler l'ordre de démarrage sur la clé USB.

Neutraliser cloud-init

Sur `[nas-host]` :

```
sudo touch /etc/cloud/cloud-init.disabled
```

Sur `[nas-host]` :

```
echo "network: {config: disabled}" | sudo tee \  
  /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
```

Pourquoi. cloud-init réécrit la configuration réseau quand il croit voir une instance neuve. Sur une machine dont les adresses sont fixes et dont les conteneurs dépendent, une réécriture silencieuse au démarrage est exactement ce qu'on ne veut pas.

Réseau – deux cartes, et c'est délibéré

```
eno1    192.168.0.2/24    parent du macvlan, porte la route par défaut
eno2    192.168.0.11/24   adresse d'administration de l'hôte
```

Un hôte ne peut pas joindre ses propres conteneurs macvlan à travers l'interface parente. Les paquets ne remontent jamais la pile réseau locale : c'est une propriété du macvlan, pas un défaut de configuration.

Ici, les conteneurs sont enfants d'`eno1`, et l'hôte leur parle depuis `eno2`. Le trafic sort par une carte, traverse le commutateur, revient par l'autre.

C'est ce montage qui rend les sauvegardes possibles. Toutes les sauvegardes nocturnes sont des tirages rsync de l'hôte vers les conteneurs. Sans la seconde carte, elles ne fonctionneraient pas.

⚠ Conséquence : `eno2` n'est pas disponible pour un passthrough PCI vers une machine virtuelle. La céder supprimerait le 192.168.0.11 de l'hôte et couperait sa voie vers les conteneurs.

Configuration

Sur `[nas-host]` :

```
sudo nano /etc/netplan/50-static-public-ip.yaml
```

Contenu :

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eno1:
```

```
dhcp4: no
dhcp6: no
addresses: [192.168.0.2/24]
nameservers:
  addresses: [192.168.0.5, 192.168.0.1]
routes:
  - to: default
    via: 192.168.0.1
eno2:
  dhcp4: no
  dhcp6: no
  addresses: [192.168.0.11/24]
  nameservers:
    addresses: [192.168.0.5, 192.168.0.1]
```

Sur `[nas-host]` :

```
sudo chmod 600 /etc/netplan/50-static-public-ip.yaml
sudo netplan apply
ip -br addr show eno1 eno2
```

Le second résolveur est le routeur, et il n'est pas décoratif. Le premier est le Pi-hole, qui tourne dans un conteneur de cette machine : l'arrêter priverait l'hôte de résolution de noms, y compris pour se réparer. □ Le repli ne peut pas être un résolveur public. Le routeur détourne vers le filtre toute requête DNS adressée hors du réseau local : une adresse extérieure renverrait donc à la machine en panne. Seule une adresse locale replie réellement.

△ □ Ce repli ne sert que les machines à adresse fixe. Les appareils qui reçoivent leurs résolveurs par DHCP n'en ont qu'un seul et n'ont aucune seconde adresse à essayer. Pour eux, la manœuvre de dépannage est de couper la redirection DNS au routeur – *DNS Director* → *Enable* → *OFF* → *Appliquer* –, ce qui leur rend le résolveur du routeur.

Base du système

Sur `[nas-host]` :

```
sudo apt install -y acl ntpsec zfsutils-linux rsync
```

Paquet	Rôle
<code>acl</code>	permissions étendues sur les partages
<code>ntpsec</code>	synchronisation horaire
<code>zfsutils-linux</code>	le pool de stockage d'Incus
<code>rsync</code>	le démon qui sert les sauvegardes

Sur `[nas-host]` – fuseau horaire :

```
sudo timedatectl set-timezone America/Toronto  
timedatectl
```

△[] Changer le fuseau d'une machine déplace ses tâches cron, dont les heures s'interprètent dans ce fuseau. Sur `nas-host` qui porte treize lignes de tâches planifiées, toutes nocturnes, ce réglage se fait à l'installation et ne se retouche plus.

`America/Montreal` est un lien de compatibilité vers `America/Toronto`, avec les mêmes règles d'heure avancée. Le nom canonique est préférable : il existe même dans une image allégée.

Stockage

Structure

Le disque de 12,7 To est monté une seule fois, sur `/media/nas1-14TB`. Les deux points de partage `/media/nas1` et `/media/nas2` sont des montages liés de deux de ses sous-répertoires.

```
/dev/sda1 —→ /media/nas1-14TB
      |
      |— nas1 —(bind)—→ /media/nas1
      |— nas2 —(bind)—→ /media/nas2
```

△□ Espace et sort communs. Les deux « disques » partagent la même capacité et disparaissent ensemble si le disque tombe. Cette structure vient du remplacement de deux disques plus anciens par un seul grand, en conservant les chemins que les scripts et les partages connaissaient.

△□ Ne jamais écrire un `rsync` de `/media/nas1` vers `/media/nas1-14TB`. `rsync` ajoute le nom du répertoire source à la destination : la copie écrirait dans `/media/nas1-14TB/nas1`, qui est la source elle-même. Deux scripts de migration écrits du temps des disques séparés sont devenus des pièges le jour de cette réorganisation.

Points de montage

Sur `[nas-host]` :

```
sudo mkdir -p /media/nas1 /media/nas2
sudo chmod 770 /media/nas1 /media/nas2
sudo mkdir -p /media/nas1/Audio/sub_mp3-128
```

Relever l'identifiant du disque

Sur `[nas-host]` :

```
sudo blkid /dev/sda1
```

L'UUID relevé est propre au système de fichiers : il change à chaque reformatage. C'est lui qu'on inscrit dans `fstab`, jamais `/dev/sda1`, qui dépend de l'ordre de détection des disques.

Le fichier `fstab`

Sur `[nas-host]` :

```
sudo nano /etc/fstab
```

Les lignes qui concernent les données :

```
UUID=<uuid-du-l4To> /media/nas1-14TB ext4 defaults,acl 0 2  
/media/nas1-14TB/nas1 /media/nas1 none bind  
/media/nas1-14TB/nas2 /media/nas2 none bind
```

⚠ L'ordre compte. Le montage lié ne peut pas précéder le montage du disque qui le porte. `fstab` est traité dans l'ordre du fichier.

L'option `acl` sur le montage physique est indispensable : les permissions étendues des partages en dépendent, et un montage sans elle les rendrait inopérantes sans message d'erreur.

Sur `[nas-host]` :

```
sudo mount -a  
df -hT | grep media
```

Miroir MP3 transcodé à la volée

`mp3fs` présente une arborescence FLAC comme une arborescence MP3, transcodée à la lecture. Rien n'est stocké : le miroir ne coûte que du calcul, au moment où un fichier est lu.

Sur `[nas-host]` :

```
sudo apt install -y mp3fs
```

Ligne à ajouter dans `/etc/fstab`, après celles des montages liés :

```
mp3fs#/media/nas1/Audio/Musique/ /media/nas1/Audio/sub_mp3-128/ -ogainmode=1,fuse  
allow_other,ro,bitrate=128 0 0
```

⚠ Cette ligne ne se coupe pas – une entrée de `fstab` tient sur une seule ligne. Elle s’enroule donc dans le PDF sans signe visible : la recopier depuis le fichier `.md`, jamais depuis le PDF.

Ce montage apparaît deux fois dans `mount`, sous `/media/nas1/Audio/sub_mp3-128` et sous `/media/nas1-14TB/nas1/Audio/sub_mp3-128`. C’est normal et non un défaut : la racine est en propagation partagée, donc un montage effectué sous un chemin lié se voit aussi par l’autre. Il n’y a bien qu’un seul processus `mp3fs`.

Partages NFS – pour les postes Linux

Sur `[nas-host]` :

```
sudo apt install -y nfs-kernel-server
```

Sur `[nas-host]` :

```
sudo nano /etc/exports
```

Contenu :

```
/media/nas1 \  
    192.168.0.0/24(rw,sync,no_subtree_check,all_squash,anonuid=1000,anongid=150)  
/media/nas2 \  
    192.168.0.0/24(rw,sync,no_subtree_check,all_squash,anonuid=1000,anongid=150)
```

```
192.168.0.0/24(rw, sync, no_subtree_check, all_squash, anonuid=1000, anongid=150)
/home/hostadmin \
192.168.0.0/24(rw, sync, no_subtree_check, all_squash, anonuid=1000, anongid=150)
```

`/etc/exports` accepte la continuation par barre oblique inverse. Les trois entrées s'écrivent aussi bien sur une ligne chacune ; la forme coupée est retenue ici pour qu'elles se recopient sans risque depuis un PDF.

Sur `[nas-host]` :

```
sudo exportfs -ra
sudo systemctl restart nfs-kernel-server
sudo exportfs -v
```

Le rôle d'`all_squash`. Toute écriture venant du réseau devient propriété de l'utilisateur `hostadmin` (1000) et du groupe `sambashare` (150), quel que soit l'utilisateur du poste client. C'est ce qui garantit que les fichiers créés depuis un portable Linux portent la même identité que ceux créés par les services de la machine – et qu'ils restent accessibles aux partages Samba.

Les attributs étendus `user.*` ne traversent pas NFS. Sans conséquence pour l'usage courant, mais tout outil qui les lit ou les écrit doit tourner sur `nas-host`, pas sur un poste où le partage est monté.

Le côté client est décrit ailleurs. Monter ces exports sur un poste, puis relier les dossiers du profil au partage : [Accéder aux partages du NAS depuis un poste Linux](#).

Partages Samba – pour les postes Windows

Le groupe commun

Toutes les machines du réseau doivent employer le même identifiant de groupe pour que les permissions se correspondent d'une machine à l'autre. Le choix est ici `150`.

Sur `[nas-host]` :

```
sudo apt install -y samba
```

Sur `[nas-host]` – vérifier le GID attribué puis le corriger :

```
getent group sambashare  
sudo groupmod -g 150 sambashare
```

Si le groupe portait un autre numéro et que des fichiers l'utilisaient déjà, il faut les reprendre :

```
sudo find / -xdev -group <ancien-gid> -exec chgrp -h sambashare {} \;
```

Le compte générique Windows

Sur `[nas-host]` :

```
sudo useradd -s /bin/false -d /dev/null -g sambashare virtuel
```

`-s /bin/false` et `-d /dev/null` : ce compte sert uniquement à Samba, il ne doit jamais pouvoir ouvrir une session.

Sur `[nas-host]` – les mots de passe Samba, qui sont distincts de ceux du système :

```
sudo smbpasswd -a hostadmin
```

```
sudo smbpasswd -a virtuel
```

Les services qui accèdent aux fichiers

Chaque service qui lit ou écrit dans les partages doit appartenir au groupe commun.

Sur `[nas-host]` :

```
for u in hostadmin virtuel user squeezeboxserver minidlna; do
    sudo usermod -a -G sambashare "$u"
done
getent group sambashare
```

⚠ Le `-a` est essentiel. Sans lui, `usermod -G` remplace tous les groupes secondaires de l'utilisateur au lieu d'en ajouter un.

Configuration

Sur `[nas-host]` :

```
sudo nano /etc/samba/smb.conf
```

Section globale :

```
[global]
workgroup = WORKGROUP
server string = %h server (Samba, Ubuntu)
server role = standalone server
server min protocol = SMB2_02
delete readonly = yes
obey pam restrictions = yes
unix password sync = yes
passwd program = /usr/bin/passwd %u
```

```
pam password change = yes
map to guest = bad user
log file = /var/log/samba/log.%m
max log size = 1000
logging = file
panic action = /usr/share/samba/panic-action %d
usershare allow guests = yes
vfs objects = acl_xattr
map acl inherit = yes
store dos attributes = yes
```

⚠ `server min protocol = SMB2_02` n'est pas facultatif. Le réglage par défaut des installations anciennes autorisait SMB1 et l'authentification LANMAN – le protocole qu'ont exploité EternalBlue et WannaCry, et un algorithme qui découpe le mot de passe en deux moitiés de sept caractères sans sel. Tout appareil en service aujourd'hui négocie SMB2 ou SMB3.

⚠ `delete readonly = yes` répare un défaut qui met des années à se voir. LibreOffice et Word créent leurs fichiers de verrouillage avec l'attribut DOS « lecture seule ». Sans ce réglage, Samba refuse de les effacer à la fermeture du document – fidèle à la sémantique DOS, où un fichier en lecture seule ne se supprime pas. Le verrou devient alors permanent, et le document s'ouvre ensuite en lecture seule ou en copie. Les verrous s'accumulent indéfiniment.

`store dos attributes = yes` range les attributs DOS dans un attribut étendu de chaque fichier plutôt que dans les bits de permission Unix. C'est ce qui préserve la date de création Windows.

Les partages

```
[Serveur - Dossier home]
comment = Répertoire home de nas-host
path = /home/hostadmin
read only = no
write list = hostadmin
create mask = 0775
directory mask = 0770
```

```
[Documents des parents]
comment = Documents des parents
path = /media/nas1/Documents/
write list = hostadmin, user, virtuel
create mask = 0775
directory mask = 0775
```

```
[Documents des enfants]
comment = Documents des enfants
path = /media/nas1/Documents/LesEnfants
write list = @smbashare
create mask = 0775
directory mask = 0775
```

```
[Scan]
comment = Numérisations
path = /media/nas1/Documents/Numérisations
read list = @smbashare
write list = hostadmin, user, virtuel
create mask = 0775
directory mask = 0775
```

```
[Downloads sur nas1]
comment = Téléchargements
path = /media/nas1/Download
write list = @smbashare
create mask = 0775
directory mask = 0775
```

```
[nas1]
comment = Disque nas1
path = /media/nas1
read list = @smbashare
write list = hostadmin
create mask = 0775
directory mask = 0775
```

```
[nas2]
comment = Disque nas2
path = /media/nas2
write list = @smbashare
create mask = 0775
```

```
directory mask = 0775
```

```
[submp3_128]  
comment = Miroir MP3  
path = /media/nas1/Audio/sub_mp3-128  
read list = @sambashare  
write list = hostadmin  
create mask = 0775  
directory mask = 0775
```

△ Ces partages se chevauchent, et le droit obtenu dépend de la porte empruntée. `[nas1]` expose `/media/nas1` en entier, ce qui contient déjà `Documents/`, `Download` et le miroir MP3. Un même fichier atteint par `[nas1]` n'est modifiable que par `hostadmin`, alors qu'il l'est aussi par `user` et `virtuel` s'il est atteint par `[Documents des parents]`. C'est un héritage à connaître avant de diagnostiquer un refus d'écriture : vérifier d'abord par quel partage le poste est entré.

Un partage sans `read only` explicite est en lecture seule – c'est le défaut de Samba. C'est la `write list` qui rouvre l'écriture, nominativement. La `read list`, elle, n'apporte rien dans cette configuration : le partage étant déjà en lecture seule pour tous, elle ne peut que retirer ce que la `write list` accorde.

Sur `[nas-host]` – vérifier avant de recharger, sans exception :

```
testparm -s
```

Sur `[nas-host]` :

```
sudo systemctl restart smbd nmbd  
systemctl is-active smbd nmbd
```

`testparm -s` n'affiche que ce qui diffère du défaut. Une sortie vide après un changement est donc un bon signe, pas un échec. Pour voir les valeurs réellement en vigueur, il faut `testparm -s -v`.

Permissions étendues

Sur `[nas-host]` :

```
sudo setfacl -Rm g:sambashare:rwX /media/nas1 /media/nas2
sudo setfacl -Rm d:g:sambashare:rwX /media/nas1 /media/nas2
sudo setfacl -Rm d:o:rx /media/nas1 /media/nas2
getfacl /media/nas1
```

Option	Effet
<code>-R</code>	récuratif
<code>g:sambashare:rwX</code>	le groupe commun peut lire et écrire
<code>X</code> majuscule	droit d'exécution sur les répertoires seulement, pas sur les fichiers
<code>d:</code>	règle par défaut, héritée par tout ce qui sera créé ensuite

△ Le `d:` est ce qui fait tenir l'ensemble dans la durée. Sans les règles par défaut, la commande corrige l'existant et rien de plus : chaque fichier créé le lendemain échapperait au groupe commun.

En cas de refus d'écriture inexpliqué, lire `getfacl` et non `ls -l`. Une ACL comporte une entrée `mask::` qui plafonne les droits effectifs de toutes les entrées de groupe. Un `mask::r-x` laisse `group:sambashare:rwX` s'afficher intact tout en le ramenant à la lecture – `getfacl` le signale par un `#effective:` en bout de ligne, `ls -l` ne le montre jamais.

Le groupe qui se perd, et le bit qui l'en empêche

Les ACL décident de qui a le droit, pas de quel groupe porte un fichier neuf. Or un fichier créé prend par défaut le groupe principal de celui qui l'écrit – et le groupe principal d'un compte de service n'est pas `sambashare`. Chaque logiciel qui réécrit un fichier le fait donc sortir du groupe commun, en silence.

□ Le bit `setgid` sur les dossiers renverse la règle : un fichier créé dans un dossier qui le porte prend le groupe du dossier, quel que soit son auteur. Le

groupe cesse de se perdre, et la remise en état périodique devient inutile.

Sur `[nas-host]` :

```
sudo find /media/nas1 -xdev -type d -exec chmod 2775 {} +
```

Le `2` de tête est le `setgid` ; `775` est le mode habituel d'un dossier partagé. L'épreuve tient en trois commandes, et elle est sans appel : le fichier doit sortir au groupe du dossier, pas au groupe principal du compte.

Sur `[nas-host]` :

```
cd /media/nas1
touch essai.tmp; stat -c '%a %U:%G %n' essai.tmp; rm -f essai.tmp
```

△ Le `setgid` agit à la création, donc il ne couvre pas tout. Une copie crée un fichier et en hérite. Un déplacement sur le même système de fichiers n'est qu'un renommage – même inode, groupe d'origine conservé, `setgid` sans effet. Et `cp -p`, `cp -a` et `rsync -a` réimposent délibérément le propriétaire et le groupe de la source, donc ils l'annulent aussi. Un dossier de préparation d'où les fichiers migrent vers le partage doit donc porter le `setgid` lui aussi, sans quoi tout ce qui en sort arrive avec le mauvais groupe.

L'ordre des trois opérations n'est pas libre

□ `chmod` réécrit le masque des ACL. Sur un fichier qui en porte, les bits du groupe ne désignent pas le droit du groupe propriétaire mais le masque – le plafond de toutes les entrées nommées. Un `chmod` passé après un `setfacl` rabote donc en silence les droits qu'on vient d'accorder : l'entrée reste inscrite, visible dans `getfacl`, et sans effet.

L'ordre est donc propriétaire, puis modes, puis ACL, jamais l'inverse.

Et `chmod` ne peut pas tout : il n'atteint jamais l'entrée `group::`. Un `group::rwx` hérité d'un ancien `chmod -R 774` survit à un `chmod 664` et fait remonter le masque

au `setfacl` suivant. Seul `setfacl` corrige cette entrée.

Sur `[nas-host]` – remettre une arborescence d’aplomb, dans cet ordre :

```
cd /media/nas1
sudo find . -xdev -name '.Trash*' -prune -o -exec chown -h hostadmin:sambashare {} +
sudo find . -xdev -name '.Trash*' -prune -o -type d -exec chmod 2775 {} +
sudo find . -xdev -name '.Trash*' -prune -o -type f -exec chmod 664 {} +
sudo find . -xdev -name '.Trash*' -prune -o -type d \
    -exec setfacl -m g:sambashare:rwX -m d:g:sambashare:rwX {} +
sudo find . -xdev -name '.Trash*' -prune -o -type f \
    -exec setfacl -m g::rw,g:sambashare:rw {} +
```

⚠ `-xdev` n’est pas une précaution de style. Il empêche `find` de franchir une frontière de système de fichiers, ce qui écarte d’un coup le miroir transcodé et tout autre montage placé dans l’arborescence – un montage en lecture seule ferait échouer les commandes, et le parcourir coûterait des heures. Les corbeilles `.Trash-1000` sont écartées séparément : leur mode `700` est voulu. Le `-h` de `chown` porte sur les liens symboliques eux-mêmes plutôt que sur leurs cibles, qui peuvent être ailleurs.

Sauvegarder les droits avant d’y toucher

□ Sur une arborescence de dizaines de milliers de fichiers, une erreur de mode ne se rattrape pas à la main. Un relevé complet se prend en une commande et restaure propriétaires, modes et ACL à l’identique.

Sur `[nas-host]` :

```
sudo sh -c "find /media/nas1 -xdev -print0 \
    | xargs -0 getfacl -p > /root/acl-nas1.txt"
```

La redirection passe par `sh -c` parce qu’un `sudo commande > /root/fichier` échoue : c’est le shell appelant, sans droits sur `/root`, qui traite le `>`.

Le retour arrière :

```
sudo setfacl --restore=/root/acl-nas1.txt
```

Retirer des ACL

```
sudo setfacl -bR /media/nas1/dossier  
sudo setfacl -x g:sambashare /media/nas1/dossier  
sudo setfacl -k /media/nas1/dossier
```

`-b` retire toutes les entrées étendues, `-x` une entrée nommée, `-k` les seules règles par défaut. C'est ce qu'il faut pour remettre en état un sous-dossier qu'un traitement récursif a touché par erreur – une corbeille, par exemple.

Sauvegardes

Le principe

Toutes les sauvegardes sont des tirages : nas-host va chercher les données là où elles sont, par le protocole du démon rsync, sur le port 873. Rien n'est poussé, et aucun accès au stockage des conteneurs depuis l'hôte n'est nécessaire.

```
nas-host —rsync://→ conteneur web    → /media/nas1/Backups/  
nas-host —rsync://→ lui-même          → /media/nas1/Backups/  
nas-host —rsync://→ poste de travail → /media/nas1/Backups/
```

⚠ Le protocole du démon rsync n'est pas chiffré. Le mot de passe est protégé par défi-réponse, mais les données circulent en clair. Compromis assumé sur un réseau domestique ; à ne pas reproduire au-delà.

Le démon

Sur `[nas-host]` :

```
sudo nano /etc/rsyncd.conf
```

Le fichier complet, onze modules :

```
log file = /var/log/rsync.log

[scripts_nas-host]
uid = hostadmin
gid = hostadmin
path = /home/hostadmin/scripts
hosts allow = 192.168.0.11
comment = Scripts de nas-host
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[musicip_nas-host]
uid = hostadmin
gid = hostadmin
path = /home/hostadmin/logiciels/MusicIP
exclude = lost+found/
hosts allow = 192.168.0.11
comment = Configuration de MusicIP
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[MusicMagic_nas-host]
uid = hostadmin
gid = hostadmin
path = /home/hostadmin/.MusicMagic
exclude = lost+found/
hosts allow = 192.168.0.11
comment = Base MusicMagic
read only = true
```

```
auth users = test
secrets file = /etc/rsyncd.scr

[icecast2_config_nas-host]
uid = root
gid = root
path = /etc/icecast2/
hosts allow = 192.168.0.11
comment = Configuration d Icecast
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[ices2_metadata_nas-host]
uid = hostadmin
gid = hostadmin
path = /home/hostadmin/ices2/
hosts allow = 192.168.0.11
comment = Metadonnees ices2
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[ices2_config_nas-host]
uid = root
gid = root
path = /etc/
exclude = lost+found/ **
include = ices2*config.xml
hosts allow = 192.168.0.11
comment = Configuration des flux ices2
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[bliss_nas-host]
uid = hostadmin
gid = hostadmin
path = /home/hostadmin/.bliss
include= regex-rules/** settings
exclude=lost+found/ **
hosts allow = 192.168.0.11
```

```
comment = Configuration de BlissHQ
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[rsync_nas-host]
uid = root
gid = root
path = /etc/
exclude = lost+found/ **
include = rsyncd.conf
hosts allow = 192.168.0.11
comment = Configuration du demon rsync
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[samba_nas-host]
uid = root
gid = root
path = /etc/samba/
exclude = lost+found/ **
include = smb.conf
hosts allow = 192.168.0.11
comment = Configuration de Samba
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[nfs_nas-host]
uid = root
gid = root
path = /etc/
exclude = lost+found/ **
include = exports
hosts allow = 192.168.0.11
comment = Exports NFS
read only = true
auth users = test
secrets file = /etc/rsyncd.scr

[crontab_nas-host]
```

```
uid = root
gid = root
path = /var/spool/cron/crontabs/
exclude = lost+found/ **
include = hostadmin
hosts allow = 192.168.0.11
comment = Crontab de hostadmin
read only = true
auth users = test
secrets file = /etc/rsyncd.scr
```

△ Le filtre serveur n'est pas décoratif. Cinq modules pointent sur `/etc` ou sur un répertoire plus large que ce dont on a besoin, et six tournent avec `uid = root`. Le couple `exclude = lost+found/ **` et `include = <fichier>` restreint ce que le démon accepte de servir. Il vit côté serveur, n'est pas transmis au client, et ne se contourne pas.

△ Ne jamais écrire `include = ***` en pensant filtrer. Les `include` étant évalués avant les `exclude`, `***` retient absolument tout – y compris ce que la ligne `exclude` suivante prétend écarter. Un module ainsi configuré n'a aucun filtre, alors qu'il en affiche deux. Pour ne retirer qu'un répertoire, un `exclude` seul suffit ; c'est la forme employée ci-dessus pour `musicip_nas-host` et `MusicMagic_nas-host`.

`bliss_nas-host` emploie pourtant `regex-rules/***`, et ce n'est pas la même chose. Le motif interdit est `include = ***` seul, qui retient tout. Appliqué à un sous-répertoire nommé, `***` veut dire « ce répertoire et tout son contenu » – c'est la forme correcte pour servir une arborescence précise et rien d'autre. △ Ce module écrit par ailleurs ses deux directives sans espace autour du `=` et dans l'ordre inverse des autres : `rsyncd` lit les deux formes de façon identique, et le fichier est reproduit ici tel qu'il est sur la machine.

△ Pas de commentaire en fin de ligne dans ce fichier. `rsyncd` ne reconnaît `#` comme marque de commentaire qu'en début de ligne. Un `hosts allow = 192.168.0.11 #ici votre IP` fait lire `#ici`, `votre` et `IP` comme trois noms d'hôtes de plus. Ils ne résolvent pas, donc ils sont ignorés et le filtrage semble fonctionner – par accident, pas par conception.

Le `comment` de chaque module s'affiche dans la liste que rend `rsync --list-only rsync://192.168.0.11/`. Le recopier d'un module à l'autre sans l'adapter produit un inventaire trompeur le jour où on cherche ce qui est sauvegardé.

`rsyncd` évalue les `include` avant les `exclude`, quel que soit leur ordre dans le fichier. Le `**` exclut donc tout ce que l'`include` n'a pas retenu.

⚠ Ne pas confondre avec les `--include` des scripts. Ceux-là sont côté client : ils décident de ce qu'on demande, pas de ce que le serveur accepte de donner. Un module sans filtre serveur reste entièrement lisible par quiconque atteint le démon avec le mot de passe.

□ La démonstration se fait sur ce module même. Qu'un script réclame `apache2/` au module `rsync_nas-host`, et rien n'arrive : le répertoire de destination ne contient toujours que `rsyncd.conf`. Le client a demandé, le démon n'a pas servi – son `include` ne le permet pas.

⚠ Le revers, et il coûte du temps : élargir le filtre du module ne suffit pas non plus. Le script porte le sien, et les deux doivent être élargis ensemble – sinon le démon sert le fichier, le client le jette, et le tirage se termine à `échecs : 0`. Le cas est détaillé dans [Sauvegarde](#).

Le mot de passe

Sur `[nas-host]` :

```
sudo nano /etc/rsyncd.scr
```

Une ligne, `utilisateur:motdepasse` :

```
test:<mot-de-passe>
```

Sur `[nas-host]` :

```
sudo chmod 600 /etc/rsyncd.scr
```

```
sudo systemctl enable --now rsync
```

Le fichier jumeau côté client, sans nom d'utilisateur, ne contient que le mot de passe :

```
nano /home/hostadmin/scripts/rsync/auto/rsync_pass
```

```
chmod 600 /home/hostadmin/scripts/rsync/auto/rsync_pass
```

rsync refuse un fichier de mots de passe lisible par d'autres. C'est une protection, pas une chicane.

Aucun redémarrage n'est nécessaire après modification de `rsyncd.conf` : le démon relit sa configuration à chaque connexion entrante. C'est aussi pourquoi un module cassé n'empêche jamais le démon de tourner – et pourquoi il peut échouer en silence pendant des années.

Durcir le démon

Sur `[nas-host]` :

```
sudo mkdir -p /etc/systemd/system/rsync.service.d
```

Sur `[nas-host]` :

```
printf '%s\n' '[Service]' 'ProtectSystem=full' 'PrivateDevices=yes' \
'NoNewPrivileges=yes' \
| sudo tee /etc/systemd/system/rsync.service.d/durcissement.conf
```

Sur `[nas-host]` :

```
sudo systemctl daemon-reload && sudo systemctl restart rsync
systemctl show rsync -p ProtectSystem -p PrivateDevices -p NoNewPrivileges
```

Option	Effet
<code>ProtectSystem=full</code>	<code>/usr</code> , <code>/boot</code> et <code>/etc</code> en lecture seule pour ce service
<code>PrivateDevices=yes</code>	aucun accès aux périphériques physiques
<code>NoNewPrivileges=yes</code>	interdit d'acquérir des privilèges en cours d'exécution

⚠ Ne jamais ajouter `ProtectHome=`. C'est la première option que suggèrent tous les guides de durcissement, et elle masquerait `/home` au service – or cinq modules servent `/home/hostadmin`. Ils échoueraient tous d'un coup.

Un *drop-in* plutôt qu'une modification de l'unité fournie par le paquet : celle-ci serait remplacée à la prochaine mise à jour.

La forme des scripts

Chaque script de sauvegarde suit le même moule :

```
#!/bin/bash
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
ERREURS=0
r() {
    rsync "$@" || { echo "ÉCHEC : ${*: -2:1} -> ${*: -1}"; ERREURS=$((ERREURS+1));
}
}

echo "=== $(basename "$0") – début $(date -Is)"

r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::scripts_nas-host/ \
  /media/nas1/Backups/Scripts/nas-host

echo "=== $(basename "$0") – fin $(date -Is) – échecs : $ERREURS"
exit $ERREURS
```

□ La deuxième ligne est un veto, pas une formalité. `/media/nas1` et `/media/nas2`

sont des points de montage : démontés, les chemins existent toujours comme simples répertoires du disque système, et rsync y écrirait sans se plaindre – remplissant `/` et fabriquant une sauvegarde qui n’est pas sur le disque de sauvegarde. Le fichier chargé vérifie les deux montages et arrête le script appelant s’il en manque un. Son texte complet est dans **Sauvegarde**.

La fonction `r()` corrige un défaut structurel. Le code de sortie d’un script est celui de sa dernière commande. Un script qui enchaîne treize `rsync` dont le douzième échoue se termine malgré tout en succès. Le compteur rend l’échec visible et exploitable.

⚠ Un compteur d’échecs ne voit pas l’absence de tentative. Si la liste sur laquelle une boucle travaille est vide – parce que le service interrogé ne répond pas – la boucle ne s’exécute pas, le compteur reste à zéro, et le script annonce un succès complet. Tout script qui construit sa liste dynamiquement doit vérifier qu’elle n’est pas vide avant de commencer.

Les marqueurs de début et de fin ne sont pas cosmétiques. Les tâches écrivent toutes dans le même journal quotidien ; sans eux, rien ne permet de savoir où commence et finit chacune, ni d’attribuer une suppression au bon script.

La crontab

Sur `[nas-host]` :

```
sudo mkdir -p /var/log/backup
sudo chown hostadmin:hostadmin /var/log/backup
```

⚠ Le `chown` est indispensable. Les tâches tournent sous `hostadmin`, qui ne peut pas écrire dans `/var/log`. Un répertoire resté à `root` produirait un « Permission denied » dans la redirection elle-même – donc une erreur sans destination, invisible.

Sur `[nas-host]` :

```
crontab -e
```

```
MAILTO=""
15 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_mysql-ns2.sh >>
/var/log/backup/$(date +%F).log 2>&1
20 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_web-ns2.sh >>
/var/log/backup/$(date +%F).log 2>&1
35 00 * * * /usr/bin/incus exec pi-hole -- bash /root/scripts/backup_pihole.sh >>
/var/log/backup/$(date +%F).log 2>&1
40 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_scripts.sh >>
/var/log/backup/$(date +%F).log 2>&1
45 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_config.sh >>
/var/log/backup/$(date +%F).log 2>&1
50 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_scripts_poste-
bureau.sh >> /var/log/backup/$(date +%F).log 2>&1
55 00 * * * bash /home/hostadmin/scripts/rsync/auto/backup_config_poste-bureau.sh
>> /var/log/backup/$(date +%F).log 2>&1
00 01 * * * bash /home/hostadmin/scripts/rsync/auto/backup_logiciels.sh >>
/var/log/backup/$(date +%F).log 2>&1
30 01 * * * bash /home/hostadmin/scripts/rsync/auto/upload_photos_photos-
webadmin-ca.sh >> /var/log/backup/$(date +%F).log 2>&1
00 02 1 * * bash /home/hostadmin/scripts/rsync/auto/backup_virtualbox_poste-
bureau.sh >> /var/log/backup/$(date +%F).log 2>&1
00 03 1 * * bash /home/hostadmin/scripts/rsync/auto/backup_incus_export.sh >>
/var/log/backup/$(date +%F).log 2>&1
30 03 1 * * find /var/log/backup/ -name '*.log' -mtime +180 -delete
30 04 * * * cd /home/hostadmin/scripts/tableau-de-bord && bash releve-tableau-de-
bord.sh >/dev/null && python3 build_tableau-de-bord.py >/dev/null
```

□ Les scripts que ces lignes appellent ne sont pas décrits ici. Ils vivent dans /home/hostadmin/scripts/rsync/auto/ et sont écrits en entier dans **Sauvegarde**, section *Les scripts de tirage*, avec le garde-fou commun que tous chargent en deuxième ligne. Cette page porte la machine et ses tâches planifiées ; l'autre porte ce qu'elles exécutent.

□ Cette crontab n'est pas l'inventaire des tâches de la machine, et c'est un piège. Deux tâches de plus, absentes de cette liste, tournent chaque nuit depuis /etc/cron.d/, sous root, parce qu'elles exigent des privilèges que le

compte `hostadmin` n'a pas :

```
/etc/cron.d/backup-lyrion      01:05  configuration du serveur de musique  
/etc/cron.d/image-cle-amorcage 04:00  image de la clé d'amorçage
```

Elles n'apparaissent nulle part ici, et ne se révèlent que dans le journal. Pour inventorier ce qui s'exécute réellement, il faut regarder les deux endroits :

```
sudo ls -l /var/spool/cron/crontabs/  
ls -l /etc/cron.d/
```

△ Et root apporte une contrainte que rien ne rappelle : aucune tâche sous root ne doit être la première de la nuit. Le journal quotidien est créé par la première tâche qui écrit dedans. Créé par root, les tâches suivantes sous `hostadmin` ne pourraient plus y ajouter leurs lignes, et perdraient leur trace sans erreur visible – la redirection échouant là où personne ne la lit. La première tâche est à 00:15 sous `hostadmin` : l'ordre tient, et c'est à vérifier avant d'avancer l'heure d'une tâche root.

L'ordre des lignes n'a aucune importance pour cron, qui lit le fichier entier et déclenche selon l'heure. Elles sont rangées ici par heure d'exécution, ce qui rend lisible l'enchaînement de la nuit ; le fichier réel peut porter le même contenu dans un autre ordre.

Deux de ces tâches ne sont pas des sauvegardes. À 1 h 30, `upload_photos_photos-webadmin-ca.sh` pousse les photos du NAS vers la galerie du serveur web – c'est le seul transfert de cette machine qui va dans ce sens, et le seul qui écrive chez un autre. Son fonctionnement, son garde-fou et le module inscriptible qu'il emploie sont décrits dans [Piwigo](#). Il partage le journal des autres tâches, d'où sa présence ici.

À 4 h 30, la dernière ligne ne sauvegarde rien : elle lit. Elle relève l'état de la machine – âge des sauvegardes sur les deux disques, tâches de la nuit retrouvées ou manquantes, services, montages, espace disque, températures – et

en fabrique une page HTML déposée sur le disque de sauvegarde. C'est le tableau de bord, décrit en entier dans `guide-tableau-de-bord-nas-host.md` : les deux scripts y figurent verbatim.

△ □ L'heure n'est pas arbitraire. 4 h 30 vient après la dernière tâche de la nuit et après l'image de la clé d'amorçage de 4 h : le relevé lit une nuit terminée. Avancé avant 4 h, il signalerait chaque matin comme manquantes des tâches qui n'ont simplement pas encore tourné.

□ □ Le `&&` entre les deux commandes est délibéré. Si le relevé échoue, le formatage ne s'exécute pas et la page conserve le contenu de la veille – visiblement daté d'hier – plutôt que d'afficher une journée à demi mesurée qui aurait l'air d'un relevé valide. Le `cd` préalable est nécessaire parce que les deux scripts écrivent leurs fichiers à côté d'eux-mêmes.

Cette ligne n'écrit rien dans `/var/log/backup/`, d'où le `>/dev/null` : sa sortie est un rapport de terminal, pas un journal, et la page HTML est elle-même le résultat conservé.

△ □ Une ligne de cron ne se coupe pas. Cron n'accepte aucune continuation par barre oblique inverse : chaque tâche tient sur une seule ligne, aussi longue soit-elle. Dans un PDF elle s'enroule sans signe visible – la recopier depuis le fichier `.md`, jamais depuis le PDF.

△ □ Le `\%` est obligatoire. Dans une crontab, `%` marque une fin de ligne et bascule le reste vers l'entrée standard de la commande. Un `date +%F` non échappé casse la ligne de façon parfaitement déroutante.

△ □ `bash` et non `sh`. Le préfixe `sh` court-circuite le `#!/bin/bash` du script et l'exécute sous `dash`. Un script écrit pour bash mais lancé par `sh` échoue uniquement en cron, jamais à la main – le pire des modes de défaillance.

△ □ L'ordre de `>> fichier 2>&1` compte. Le `2>&1` renvoie le canal d'erreur là où pointe déjà le canal de sortie. Inversé, les erreurs partiraient vers l'ancienne destination – or ce sont elles qui importent.

Tester une tâche dans les conditions de cron

Un script qui fonctionne dans un shell interactif peut échouer en cron, dont l'environnement est presque vide.

Sur `[nas-host]` :

```
env -i HOME=/home/hostadmin PATH=/usr/bin:/bin \
/home/hostadmin/scripts/rsync/auto/backup_config.sh
```

Le point sensible est `HOME` : sans lui, tout outil qui cherche un fichier de configuration dans le répertoire personnel échoue.

Incus

Choisir la branche

Question	Choix	Raison
Dépôt	Zabbly <code>[lts-6.0]</code>	Ubuntu fige sa base en 6.0.0 ; Zabbly suit les correctives amont sur la même ligne LTS
Version	6.0, pas 7.0	Incus 7.0 exige un noyau ≥ 6.12 ; Ubuntu 24.04 est en 6.8
Nom du pool	<code>[local]</code>	
Emplacement	fichier creux de 100 Go sur le NVMe	Le pool porte le système de fichiers des conteneurs, donc MySQL – pas sur un disque mécanique

Ce que la 7.0 apporte – conteneurs OCI, écouteur S3, sauvegardes incrémentales de VM, fonctions de grappe – ne concerne pas un hôte unique portant cinq conteneurs système en macvlan.

Vérifier la clé du dépôt

Sur `[nas-host]` – cette commande n'importe rien, elle affiche :

```
curl -fsSL https://pkgs.zabbly.com/key.asc | gpg --show-keys --fingerprint
```

Empreinte attendue :

```
4EFC 5906 96CB 15B8 7C73 A3AD 82CC 8797 C838 DCFD
```

Pourquoi. On s'apprête à autoriser un dépôt tiers dont les paquets s'installeront en root. La vérification n'a de valeur que parce que l'empreinte de référence vient d'une source indépendante du téléchargement – elle est publiée dans le dépôt GitHub de Zabbly. Comparer l'empreinte à elle-même ne vérifierait rien.

Ajouter le dépôt

Sur `[nas-host]` :

```
sudo mkdir -p /etc/apt/keyrings/  
sudo curl -fsSL https://pkgs.zabbly.com/key.asc -o /etc/apt/keyrings/zabbly.asc
```

Sur `[nas-host]` :

```
sudo nano /etc/apt/sources.list.d/zabbly-incus-lts-6.0.sources
```

```
Enabled: yes  
Types: deb  
URIs: https://pkgs.zabbly.com/incus/lts-6.0  
Suites: noble  
Components: main  
Architectures: amd64  
Signed-By: /etc/apt/keyrings/zabbly.asc
```

Donner la priorité au dépôt

△ C'est l'étape qu'on ne devine pas. Ubuntu distribue son propre paquet `incus`, et un abonnement Ubuntu Pro donne au dépôt ESM une priorité de 510, supérieure au 500 par défaut d'un dépôt tiers. Sans épinglage, `apt install incus` installerait la version d'Ubuntu – silencieusement, sans erreur.

Sur `[nas-host]` :

```
sudo nano /etc/apt/preferences.d/zabbly-incus
```

```
Package: *  
Pin: origin pkgs.zabbly.com  
Pin-Priority: 600
```

- `Pin: origin` désigne le nom d'hôte du dépôt, pas son champ « Origin » – confusion classique de la documentation d'apt.
- `600` dépasse le 510 d'ESM. Ne pas dépasser 1000 : au-delà, apt s'autorise à *rétrograder* des paquets déjà installés.

Sur `[nas-host]` – vérifier d'où viendra le paquet, avant de l'installer :

```
sudo apt update  
apt policy incus
```

Le candidat doit venir de `pkgs.zabbly.com` avec la priorité 600. Prendre l'habitude de `apt policy` avant toute installation depuis un dépôt tiers.

Installer

Sur `[nas-host]` – simuler d’abord :

```
sudo apt install --dry-run incus incus-ui-canonical
```

Trois choses à regarder : les paquets Zabbly qui seront installés, la ligne « 0 à enlever » – s’arrêter si apt propose de retirer quoi que ce soit – et l’absence de `qemu-system-x86`, le support des machines virtuelles n’étant pas installé par défaut.

Sur `[nas-host]` :

```
sudo apt install incus incus-ui-canonical
```

Sur `[nas-host]` :

```
systemctl status incus.socket --no-pager
```

`incus.service` apparaît `inactive (dead)`, et c’est normal. Incus utilise l’activation par socket : systemd écoute sur `/var/lib/incus/unix.socket` et ne démarre le démon qu’à la première connexion d’un client. C’est `incus.socket` qui doit être `active (listening)`.

Autoriser son utilisateur

Sur `[nas-host]` :

```
sudo usermod -aG incus-admin hostadmin
```

Le paquet crée deux groupes : `incus` pour un accès restreint, `incus-admin` pour le contrôle complet.

Sur `[nas-host]` – se déconnecter et se reconnecter, le changement de groupe n'étant lu qu'à l'ouverture de session :

```
exit
```

Sur `[nas-host]` :

```
id
incus version
```

Les versions client et serveur doivent correspondre.

Initialiser

Sur `[nas-host]` :

```
incus admin init
```

Question	Réponse
Clustering	☐ no
Nouveau pool de stockage	☐ yes
Nom du pool	☐ local
Moteur	☐ zfs
Nouvel agrégat ZFS	☐ yes
Périphérique bloc existant	☐ no
Taille en Gio	☐ 100
Créer un pont réseau local	☐ no
Utiliser un pont existant	☐ no
Serveur disponible sur le réseau	☐ no
Mettre à jour les images en cache	☐ yes
Imprimer un preseed YAML	☐ yes

⚠ Refuser le pont réseau. Sinon Incus fabrique un `incusbr0` qui ne servira jamais, les conteneurs étant tous en macvlan.

Le preseed imprimé est à conserver : `incus admin init --preseed < fichier.yaml`

reproduit cette configuration sans aucune question.

Sur `[nas-host]` :

```
incus storage info local
incus profile show default
```

Le profil `default` ne doit contenir que le périphérique `root` pointant vers `pool:local`, sans carte réseau.

▲ `incus storage info`, et non `incus storage list`. La liste ne montre aucune colonne d'espace dans cette version : on peut la lire dix fois sans jamais voir combien il reste. C'est `info` qui donne l'espace total et l'espace employé du pool.

Et pour savoir où un conteneur est réellement monté, ne pas demander à ZFS. `zfs list -o mountpoint` rend `legacy` pour ces datasets – Incus les monte lui-même, sans point de montage enregistré. ▲ Une variable qui contient le mot `legacy` ne le dit pas : les commandes suivantes portent alors sur un chemin relatif inexistant, et un `find | wc -l` rend zéro, qui ressemble trait pour trait au résultat attendu. Le chemin réel se lit dans `/proc/mounts` :

```
sudo grep -n 'nom-du-conteneur' /proc/mounts
```

Le profil réseau

Sur `[nas-host]` :

```
incus profile create macvlan
incus profile device add macvlan eth0 nic nictype=macvlan parent=en01
incus profile show macvlan
```

Élément

`eth0`

Rôle

le nom que le conteneur verra – celui que son netplan configure

Élément	Rôle
<code>nictype=macvlan</code>	le conteneur obtient sa propre MAC sur le réseau physique
<code>parent=enol</code>	l'interface hôte à laquelle il s'accroche

△ □ `parent=enol` est le point critique. Avec `eno2`, les conteneurs resteraient joignables depuis le réseau, mais l'hôte ne pourrait plus les atteindre – et les sauvegardes nocturnes tomberaient sans erreur visible.

L'interface web

Sur `[nas-host]` :

```
incus config set core.https_address 192.168.0.11:8443
```

△ □ Lier à l'adresse d'administration, jamais à `:8443` tout court, sur une machine dont le port 443 est redirigé depuis Internet.

L'authentification se fait par certificat client, pas par mot de passe : l'API donne un contrôle équivalent à root. Le client en ligne de commande le fait sans qu'on s'en aperçoive ; le navigateur exige une manipulation manuelle, et son ordre n'est pas évident :

1. Onglet « Browser certificate » → Generate. C'est cette page qui *crée* le certificat.
2. Télécharger le `.pfx`, l'importer dans Firefox : `about:preferences#privacy` → Certificats → Afficher les certificats → Vos certificats → Importer. Mot de passe vide si aucun n'a été défini.
3. Redémarrer Firefox complètement.

4. Rouvrir l'URL et répondre aux deux dialogues : le choix du certificat, puis l'avertissement `SELF_SIGNED_CERT` qui concerne le certificat *du serveur*.

5. Seulement maintenant, l'onglet « Identity trust token » devient utilisable. Générer un jeton et le déclarer :

```
incus config trust add incus-ui
```

Un jeton est à usage unique : la première soumission consomme l'opération en attente, qu'elle aboutisse ou non. En cas d'échec, régénérer plutôt que réessayer.

Créer un conteneur

Sur `[nas-host]` :

```
incus launch images:ubuntu/24.04 mon-conteneur --profile default --profile macvlan
```

Sur `[nas-host]` :

```
incus config set mon-conteneur boot.autostart true
```

Les limites de mémoire et de processeur

Un conteneur sans limite déclarée peut prendre toute la mémoire de la machine, et rien ne le signale : les autres ralentissent, puis le noyau tue un processus, chez eux ou chez lui.

Le plafond se pose d'après une mesure, jamais d'après l'intuition. Sur `[nas-host]` – relever ce que chaque conteneur consomme :

```
for c in $(incus list --format csv -c n); do
  printf "%-14s limite %-10s courante %s\n" "$c" \
    "$(incus config get $c limits.memory)" \
    "$(incus info $c | awk '/Mémoire \{(courante\)/ {print $NF}')"
done
free -h
```

△□ Une consommation courante est une photographie, pas un maximum. Le plafond se pose à trois ou cinq fois la valeur relevée : il sert à empêcher qu'un emballement affame les autres, pas à serrer au plus près. Trop juste, il fait tuer le service par le noyau sans un mot dans le journal de ce service – c'est l'hôte qui tue, pas l'application qui échoue. Deux précisions évitent de sous-estimer le besoin : un plafond n'est pas une réservation, donc la somme des plafonds peut dépasser sans risque la mémoire de la machine ; et ce qu'un service lit par projection en mémoire y est compté, ce qui porte le besoin d'un service à gros index bien au-delà de la taille de son propre processus.

Sur `[nas-host]` – poser les plafonds :

```
incus config set mon-conteneur limits.memory 1GiB
incus config set mon-conteneur limits.cpu 2
```

□□ La limite s'applique à chaud, et se vérifie de l'intérieur : `incus exec mon-conteneur -- free -h` doit annoncer le plafond comme mémoire totale. C'est ce qui distingue un plafond réellement appliqué d'une valeur seulement inscrite dans la configuration.

Retirer ce que l'image apporte sans qu'on le demande

`open-vm-tools` est installé sur toutes les images Ubuntu Server, par recommandation du méta paquet `ubuntu-server`. C'est l'agent invité des machines virtuelles VMware : il n'a aucun objet dans un conteneur, et c'est lui qui

déclenche le `udevadm trigger` d'où vient tout le bruit des installations.

Sur `[nas-host]` :

```
incus exec mon-conteneur -- apt-get -y purge open-vm-tools
incus exec mon-conteneur -- apt-get -s autoremove --purge | grep -E
"^(Purg|Remv)"
```

Lire la liste avant d'appliquer – elle diffère d'un conteneur à l'autre :

```
incus exec mon-conteneur -- apt-get -y autoremove --purge
```

△ `--purge` n'est pas optionnel sur un `autoremove` : sans lui, apt garde les fichiers de configuration et fabrique autant de paquets en état `rc`.

△ Une montée de version ne retire rien. `do-release-upgrade` met à niveau ce qui est présent ; `open-vm-tools` ressortirait simplement dans sa version suivante. Le retrait est un geste explicite, et il vaut mieux le faire tôt : ce qui est retiré n'a pas à être migré.

Trois pièges d'`incus exec`

△ `incus exec` ne lance pas d'interpréteur de commandes. Un `*`, un tube ou une redirection écrits après `incus exec <conteneur> --` sont traités par le shell de l'hôte, pas par celui du conteneur. `incus exec web -- du -sh /var/www/*` cherche un fichier littéralement nommé `*`, puisque `/var/www` n'existe pas sur l'hôte. Pour donner un motif au conteneur : `bash -c "..."`.

Les `Failed to write 'change' to '/sys/.../uevent': Permission denied` ne sont pas des erreurs. Ils viennent d'un `udevadm trigger` lancé par un script post-installation : dans un conteneur non privilégié, `/sys` est en lecture seule. Les chemins le disent mieux que n'importe quelle explication – ils nomment le contrôleur SATA et les disques de l'hôte. C'est un message d'erreur qui prouve que le confinement fonctionne.

△ `incus file push` ne dépose pas le fichier en `root:root`. Sans `--uid 0 --gid 0`, il arrive avec le propriétaire par défaut du conteneur – `ubuntu:ubuntu` – et le service qui devait le lire l’ignore. Les deux options font partie de la commande, pas de ses raffinements :

```
incus file push /tmp/fichier.conf --uid 0 --gid 0 \  
    mon-conteneur/etc/quelque-part/fichier.conf  
incus file pull mon-conteneur/etc/quelque-part/fichier.conf -
```

La relecture par `incus file pull` n’est pas décorative. Si le chemin de dépôt était fautif, le dépôt réussirait quand même – en créant un fichier ailleurs.

L’adresse fixe, dans le conteneur

Les adresses vivent dans le netplan de chaque conteneur, pas dans Incus. C’est ce qui les rend indépendantes de l’hôte : elles voyagent avec le conteneur, et la MAC n’a aucune importance.

Sur `[nas-host]` :

```
incus exec mon-conteneur -- nano /etc/netplan/50-static-public-ip.yaml
```

```
network:  
  version: 2  
  renderer: networkd  
  ethernets:  
    eth0:  
      dhcp4: no  
      dhcp6: no  
      addresses: [192.168.0.X/24]  
      nameservers:  
        addresses: [192.168.0.5, 192.168.0.1]  
      routes:  
        - to: default  
          via: 192.168.0.1
```

Sur `[nas-host]` :

```
incus exec mon-conteneur -- chmod 600 /etc/netplan/50-static-public-ip.yaml
incus exec mon-conteneur -- netplan apply
```

⚠ Neutraliser cloud-init dans le conteneur, faute de quoi un fichier `50-cloud-init.yaml` peut réapparaître et reprendre la main sur l'adresse :

```
echo "network: {config: disabled}" | incus exec mon-conteneur -- \
tee /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
```

Sur `[nas-host]` – aligner le fuseau horaire, les images arrivant en `Etc/UTC` :

```
incus exec mon-conteneur -- timedatectl set-timezone America/Toronto
```

Sans cela, un même fichier porte deux heures différentes selon qu'on le regarde depuis l'hôte ou depuis le conteneur – quatre heures d'écart qui font perdre du temps quand on corrèle deux journaux.

⚠ Ne jamais tester un service macvlan depuis l'hôte. Un `curl` lancé depuis `nas-host` vers un conteneur peut geler sans erreur : les paquets partent par l'interface parente, où le macvlan interdit précisément le dialogue hôte↔conteneur, et disparaissent sans message de refus. Tester depuis une autre machine du réseau.

Copier un conteneur

C'est le geste à faire avant toute opération risquée : la copie sert de répétition, là où une erreur ne coûte rien.

Sur `[nas-host]` :

```
incus copy mon-conteneur essai
incus config set essai boot.autostart false
incus config device remove essai <disque-attaché>
incus config get essai boot.autostart
incus config device list essai
incus config device list mon-conteneur
```

▲ □ L'ordre n'est pas négociable. `incus copy` produit une copie arrêtée – rien ne démarre tout seul –, mais elle porte l'adresse de l'original dans son netplan et la déclaration de ses périphériques. Les deux gestes qui suivent la copie viennent donc avant tout démarrage.

□ Deux instances qui montent le même disque ne produisent pas une erreur, mais une corruption. C'est la raison du `device remove`.

▲ □ Et `boot.autostart false` se relit après coup. Prescrit une fois sans être exécuté, il a déjà laissé deux conteneurs démarrer sur la même adresse au redémarrage de l'hôte. Les trois relectures ci-dessus ne sont pas du zèle – la troisième vérifie qu'on a retiré le disque de la *copie* et non de la machine en service.

Changer l'adresse d'un conteneur arrêté – on ne peut pas y ouvrir d'éditeur : le fichier se prépare sur l'hôte, puis se dépose.

Sur `[nas-host]` :

```
nano /tmp/netplan-essai.yaml
```

Contenu, identique à celui de l'original sauf l'adresse – à coller dans l'éditeur ouvert ci-dessus :

```
network:
  version: 2
  renderer: networkd
  ethernets:
```

```
eth0:
  dhcp4: no
  dhcp6: no
  addresses: [192.168.0.4/24]
  nameservers:
    addresses: [192.168.0.5, 192.168.0.1]
  routes:
    - to: default
      via: 192.168.0.1
```

Sur `[nas-host]` :

```
incus file push /tmp/netplan-essai.yaml --uid 0 --gid 0 \
  essai/etc/netplan/50-static-public-ip.yaml --mode 0600
incus file pull essai/etc/netplan/50-static-public-ip.yaml -
```

`--mode 0600` préserve les permissions ; sans lui, netplan se plaint d'un fichier trop ouvert.

Deux choses qu'une copie ne reproduit pas

△ □ Le nom d'hôte. Incus pose le nom de l'instance comme nom d'hôte à chaque démarrage. Une copie nommée autrement ne résout plus son propre nom, Apache s'en plaint, et ISPConfig s'en sert. Il faut le reposer après chaque redémarrage de la copie :

```
incus exec essai -- hostnamectl set-hostname mon-conteneur
```

△ □ L'état volatil. Drapeaux, verrous, sockets, processus lancés à la main : tout ce qui vit dans `/run` – un système de fichiers en mémoire – ne traverse pas la copie, qui démarre avec un `/run` vierge. □ Une répétition sur copie ne peut donc pas révéler un obstacle de cette nature : elle part d'un état plus propre que l'original.

Le renommage de l'instance n'est pas nécessaire pour s'y retrouver : tant qu'on passe par `incus exec <nom> -- ...` depuis l'hôte, il n'y a pas d'invite à

confondre.

Un conteneur avec un disque de l'hôte

Le conteneur web monte une partition entière comme `/var/www`.

Sur `[nas-host]` – identifier le disque par son identifiant matériel, jamais par `/dev/sdX` :

```
ls -l /dev/disk/by-id/
```

Sur `[nas-host]` :

```
incus config device add conteneur-web web disk \
  source=/dev/disk/by-id/scsi-SATA_<modèle>-part2 \
  path=/var/www
```

Le faire fonctionner sans privilèges

Un conteneur non privilégié décale les identifiants : ce qui est l'utilisateur 0 à l'intérieur est l'utilisateur 1000000 sur l'hôte. Les fichiers d'un disque écrit avant cette bascule portent des numéros qui ne tombent plus dans la plage attendue.

Sur `[nas-host]` :

```
sudo apt install -y incus-extra
```

Sur `[nas-host]` – le conteneur arrêté, décaler les propriétaires du disque :

```
sudo fuidshift /point/de/montage/du/disque b:0:1000000:1000000
```

△ La taille de plage `1000000` – et non `1000000000` – est ce qui rend l’opération rejouable. Avec la grande plage, les fichiers déjà décalés à 1000000 tombent encore dans la plage source et seraient décalés une seconde fois, vers 2000000. Avec la petite, ils en sortent et l’outil les marque « ne pas changer ». C’est la forme à retenir, y compris pour une première passe.

△ Un attribut immuable interrompt le décalage. `ISPConfig` pose `chattr +i` sur la racine de certains sites, et un fichier immuable refuse `chown` même à root. L’outil s’arrête net en laissant l’arborescence à moitié traitée – d’où l’importance de la forme rejouable ci-dessus. Retirer l’attribut avec `chattr -i` avant de reprendre.

Le mode essai de `fuidshift` imprime une ligne par entrée parcourue, pas par entrée modifiée. Compter les lignes ne mesure rien ; il faut compter celles qui ne se terminent pas par `to -1 -1`.

Pourquoi cette manœuvre n’est pas nécessaire pour le système de fichiers racine du conteneur. `Incus` n’écrit plus les propriétaires décalés sur le disque : il utilise les montages `idmapped`, une fonction du noyau qui applique la translation au moment du montage. Le fichier reste tel quel sur le disque ; seul le conteneur le voit décalé. Mais cette fonction ne s’applique qu’aux sources de type répertoire, pas aux périphériques bloc – d’où le décalage manuel ici.

△ Piège de lecture qui en découle : un `ls -n` de l’hôte montrant des numéros bas dans un système de fichiers de conteneur ne signifie pas que ce conteneur est privilégié. Seule la carte fait foi :

```
incus config get conteneur-web volatile.idmap.current
```

Accéder aux fichiers de l’hôte avec la bonne identité

Quand un conteneur doit lire des fichiers appartenant à l’utilisateur de l’hôte, la projection d’identifiants se règle sur l’instance :

```
raw.idmap: |
  uid 1000 1000
  gid 150 150
```

Cette ligne découpe la correspondance en trois segments : 0 à 999 vont vers 1000000, l'identifiant 1000 traverse tel quel, et 1001 et au-delà reprennent plus loin. Le conteneur porte alors l'identité du compte de l'utilisateur pour ces fichiers-là seulement.

△ □ Monter en lecture seule tout ce qui n'a pas besoin d'être écrit. Un conteneur exposé sur Internet qui porte l'identité de l'utilisateur de l'hôte et monte un répertoire en écriture offre, en cas de faille applicative, un chemin d'écriture vers les données de l'hôte – sans aucune évasion de conteneur. Ce n'est pas un défaut de confinement mais le fonctionnement prévu.

```
incus config device set conteneur-web <périphérique> readonly true
```

Un lien symbolique ne suit pas hors de l'espace de montage. Si le répertoire monté ne contient que des liens vers des fichiers situés ailleurs, il faut monter aussi leur destination, au même chemin que sur l'hôte, sinon les liens pointent dans le vide.

Deux limites du conteneur non privilégié, à connaître avant d'en avoir besoin

△ □ Il ne peut pas créer de fichier de périphérique, même en root. Les capacités sont évaluées contre l'espace de noms propriétaire du système de fichiers – même cause que le `chattr +i` impossible. Conséquence concrète : restaurer une arborescence par `tar` échoue sur les nœuds de `/dev` que contient un environnement confiné jailkit :

```
tar: ./web6/dev/urandom: Cannot mknod: Operation not permitted
```

`tar` signale et poursuit : le reste arrive bien, mais il sort en échec. À savoir

avant de conclure qu'une restauration a échoué.

□ Incus refuse de démarrer un conteneur dont la source d'un disque a disparu. Éprouvé : source supprimée, démarrage refusé. Le scénario redouté – `/var/www` non monté et un tirage nocturne `--delete` propageant le vide jusqu'à la sauvegarde – n'existe pas.

△□ Il reste un cas, et il n'est pas couvert de ce côté : une source présente mais vide. Incus démarre alors sans rien dire. Seul un contrôle « source non vide » dans le script de tirage l'attrape – il est décrit dans [Sauvegarde](#).

L'expérience elle-même a un piège : `incus config device add` valide la source au moment de la déclaration et refuse tout de suite. Pour éprouver le *démarrage*, il faut déclarer le disque vers une source existante, la retirer ensuite, puis tenter de démarrer.

Sauvegarde des conteneurs

Les tirages nocturnes sauvegardent les données des services. Ils ne sauvegardent pas les conteneurs eux-mêmes.

Un script mensuel s'en charge le 1er du mois : `backup_incus_export.sh`, écrit en entier dans [Sauvegarde](#), section *Les scripts de tirage*. Il vit avec les dix autres, et c'est là qu'on le corrige – un seul endroit, pour qu'il n'existe pas deux versions qui divergent.

Ce qu'il faut en comprendre, et qui ne se lit pas dans son code :

- Découverte dynamique – un conteneur ajouté plus tard est sauvegardé sans qu'on y pense.
- Un répertoire par mois – le tri alphabétique équivaut au tri chronologique,

ce qui rend la purge triviale.

- Purge par comptage et non par âge – garantit exactement deux mois même si une exécution a été manquée.
-  Aucune purge si un export a échoué. Supprimer l'ancienne sauvegarde quand la nouvelle vient d'échouer est la façon classique de se retrouver sans rien.
- Sans `--optimized-storage` – plus lent et plus gros, mais restaurable vers n'importe quel stockage, y compris le jour où le pool ZFS serait lui-même le problème.
- Vérification de liste vide – sans elle, un Incus qui ne répond pas produirait un succès avec zéro conteneur sauvegardé.

 Les conteneurs sont exportés en marche. Le système de fichiers peut être saisi en pleine écriture et une base de données n'y sera pas nécessairement cohérente. C'est acceptable parce que la copie cohérente existe ailleurs : le tirage nocturne des données. L'export reconstruit le conteneur, le tirage restaure les données – ni l'un ni l'autre ne suffit seul.

 Un export ne contient pas les disques montés depuis l'hôte. Il capture le volume du conteneur, pas ce qui y est monté. Un serveur web restauré depuis son export démarre, mais ses sites sont vides tant que leur propre sauvegarde n'a pas été remplacée.

Ce que cette machine porte, et ce qu'elle ne protège pas

nas-host est le point de convergence de toutes les sauvegardes du parc : les copies quotidiennes et les exports mensuels atterrissent sur son disque de données, et c'est de là que part la synchronisation hors site.

<code>/media/nas1/Backups/</code>	copies quotidiennes
<code>/media/nas1/Backups/incus-exports/</code>	exports mensuels, deux mois conservés

Et c'est exactement pourquoi ce disque ne suffit pas. Tout converge au même endroit, sur la même machine, sur un seul système de fichiers – `/media/nas1` et `/media/nas2` sont deux montages liés d'un même disque. Une panne matérielle les emporte ensemble.

Les quatre niveaux de protection, leurs angles morts et le raisonnement qui les rend complémentaires sont décrits dans [Sauvegarde](#) – un seul endroit où les tenir à jour.

Services de l'hôte

Ces services ne vivent pas dans des conteneurs. Chacun a son propre article ; l'ordre ci-dessous est celui de leur installation.

Service	Rôle
Lyrion Music Server	serveur de musique du foyer
MusicIP	analyse acoustique et listes de lecture
BlissHQ	métadonnées et pochettes
minidlna	partage DLNA vers les téléviseurs
Icecast et ices2	diffusion audio et repli d'alarme
Onduleur CyberPower	arrêt propre en cas de coupure
Syncthing	propagation de fichiers entre appareils
Synchronisation Proton Drive	copie hors site

BlissHQ est un programme Java lancé depuis son répertoire, sans paquet ni unité systemd. Il n'apparaît donc ni dans `dpkg -l` ni dans `systemctl` – ce qui ne veut pas dire qu'il est absent.

Et une chose présente sur la machine n'est pas ce qu'elle paraît : `/etc/apache2`. Aucun serveur web ne tourne ici et aucun paquet apache n'est installé, pas même en état `rc`. Le répertoire ne contient qu'un `conf-available/javascript-common.conf`, déposé par le paquet `javascript-common` – lui-même tiré par `libjs-jquery` et `libjs-underscore`. Un paquet qui ne dépend pas d'Apache y dépose un fragment au cas où un serveur web existerait. Rien à retirer, rien à surveiller – et trois commandes pour le savoir, alors qu'une phrase suffit à ne pas refaire l'enquête.

Quand une mise à jour refuse de commencer

Un jour, `apt upgrade` s'arrête avant d'avoir traité le moindre paquet :

```
dpkg: erreur fatale irréversible, abandon :  
l'utilisateur système « untel » inconnu dans le fichier des permissions
```

□ Le logiciel nommé dans le message n'a rien à voir avec ce qu'on met à jour, et il peut avoir été retiré de la machine des années plus tôt. `dpkg` tient une liste de dérogations de permissions – des propriétaires et des droits qu'il impose à certains fichiers, différents de ceux que le paquet déclare. Quand un service est purgé, son compte système disparaît, mais la dérogation qui le nomme peut rester. `dpkg` refuse alors de travailler tant qu'il ne peut pas résoudre ce nom – et il refuse avant de traiter quoi que ce soit, bloquant donc l'ensemble des mises à jour, correctifs de sécurité compris.

Voir de quoi il s'agit :

```
sudo dpkg-statoverride --list | grep -i <nom-du-compte>
```

Chaque ligne donne le propriétaire, le groupe, les droits, puis le chemin concerné. C'est ce chemin qui sert à retirer l'entrée, une par une :

```
sudo dpkg-statoverride --remove /chemin/du/fichier
```

Retirer une dérogation ne touche à aucun fichier : elle ne fait qu'effacer une consigne. Les répertoires du service disparu restent donc en place, et sont à traiter séparément.

⚠ La règle qui en découle vaut au retrait, pas à la panne : après avoir purgé un service qui tournait sous son propre compte système, vérifier qu'aucune dérogation ne le nomme encore. Rien ne le signale sur le moment – le défaut dort jusqu'à la mise à jour suivante, où il se présente sous un message qui désigne le mauvais coupable.

Vérifications après reconstruction

Sur `[nas-host]` – le système :

```
uname -r  
ip -br addr show eno1 eno2  
df -hT | grep media  
systemctl --failed
```

Sur `[nas-host]` – les partages :

```
testparm -s -v | grep -iE 'min protocol|delete readonly'  
sudo exportfs -v  
systemctl is-active smbd nmbd nfs-kernel-server rsync
```

Sur `[nas-host]` – les conteneurs :

```
incus list  
sudo zpool list
```

Sur `[nas-host]` – les sauvegardes, en conditions réelles :

```
bash /home/hostadmin/scripts/rsync/auto/backup_config.sh | tail -3
```

La dernière ligne doit se terminer par « échecs : 0 ».

△ Ne jamais juger une sauvegarde à la date de ses fichiers. `rsync -a` préserve la date de la source : le fichier le plus récent d'une sauvegarde porte la date qu'il avait à l'origine, pas celle de la copie. La question se pose au journal de cron, pas aux fichiers.

△ Redémarrer la machine avant de considérer l'installation terminée. C'est ce test qui révèle ce qui a été fait à la main sans être inscrit : une unité activée mais non `enable`, un montage absent de `fstab`, un service qui ne démarre que parce qu'on l'a lancé. Rien de tout cela ne se voit tant qu'on ne redémarre pas.

Ce que ce guide ne couvre pas

- l'installation de chaque service de l'hôte – un article par service
- le conteneur web et sa pile Apache multi-PHP – article dédié
- [le Pi-hole](#), le mandataire Squid, le correcteur linguistique – articles

dédiés

- la montée d'un conteneur vers la prochaine version LTS – article dédié :
rebâtir une machine et en monter une existante sont deux travaux distincts
- la sécurisation de l'accès distant – article dédié