Migration LXD vers INCUS

Guide construit pas à pas le 31 août 2026, sur une Ubuntu 24.04 (noyau 6.8) hébergeant 5 conteneurs LXD.
Chaque étape n'est consignée qu'après avoir réellement fonctionné sur la machine.

Les noms d'hôtes, de domaines et de comptes de cette page sont des exemples à transposer. Les adresses `192.168.0.x` appartiennent à l'espace privé RFC 1918 et sont conservées telles quelles : elles illustrent la topologie sans identifier personne.

Le fichier `.md` de cette page : ouvrir le `.md`

Contexte

- Hôte : nas-host (`nas-host.lan.local`, 192.168.0.11), Supermicro X11SAE-M, Xeon E3-1230 v5, 31 Gio de RAM
- Départ : LXD installé par snap, pool ZFS `default` dans un fichier de 27,5 Go
- Arrivée : Incus LTS 6.0 (dépôt Zabbly) en paquet Debian, pool ZFS `local` de 100 Go
- Méthode retenue : migration conteneur par conteneur, l'original LXD conservé arrêté comme filet
- Conteneurs : `navidrome`, `squid-proxy`, `languagetool`, `pi-hole`, `web-srv`

Décisions prises avant de commencer

Question	Choix	Raison
Branche du dépôt	`lts-6.0`	Deux raisons. D'abord, `stable` n'est supporté que jusqu'à la sortie suivante (~1 mois) alors qu'une LTS tient 5 ans. Ensuite et surtout : Incus 7.0 exige un noyau ≥ 6.12, nas-host est en 6.8.

Question	Choix	Raison
Zabbly plutôt que le paquet Ubuntu	Zabbly `lts-6.0`	Ubuntu fige la base en 6.0.0 ; Zabbly suit les correctives amont — 6.0.6 au moment de l'installation.
Nom du pool Incus	`local`, pas `default`	Le zpool de LXD s'appelle déjà `default` — ZFS refuserait un doublon.
Emplacement du pool	Fichier sur le NVMe, 100 Go	Le NVMe n'a qu'une partition ; le pool porte le rootfs des conteneurs, donc MySQL — pas de disque mécanique.

Prérequis vérifiés

- Export complet des 5 conteneurs, archives testées au `tar tzf`
- 335 Go libres sur `/`, conteneurs totalisant ~9 Gio
- IP statiques dans le netplan de chaque conteneur (elles voyagent avec lui)
- Résolveur de secours ajouté partout, pour que la migration du Pi-hole ne coupe rien
- Configuration complète relevée : profils `default` et `macvlan`, périphériques disque, drapeaux

---

Étape 1 — Vérifier la clé du dépôt

Cette commande ne modifie rien : elle affiche la clé sans l'importer.

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

Résultat attendu

```
pub   rsa3072 2023-08-23 [SC] [expires: 2030-08-17]
      4EFC 5906 96CB 15B8 7C73  A3AD 82CC 8797 C838 DCFD
uid                      Zabbly Kernel Builds <info@zabbly.com>
```

Pourquoi. On s’apprête à autoriser un dépôt tiers dont les paquets s’installeront en root. Cette clé est ce qui garantit leur provenance. 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 README de `github.com/zabbly/incus`. Comparer l’empreinte à elle-même ne vérifierait rien.

Notes

- Les messages `directory '/home/hostadmin/.gnupg' created` sont normaux à la première utilisation de `gpg` par cet utilisateur.
- Options de `curl` : `-f` échoue sur erreur HTTP, `-s` silencieux, `-S` affiche quand même les erreurs, `-L` suit les redirections.
- Si `gpg` manque : `sudo apt install gnupg`.

En cas d’écart — s’arrêter et vérifier sur le dépôt GitHub de Zabbly avant toute chose.

Étape 2 — Ajouter le dépôt

Aucune de ces commandes n’installe quoi que ce soit.

2.1 — Créer le répertoire des trousseaux apt

```
sudo mkdir -p /etc/apt/keyrings/
```

Emplacement standard depuis Debian 11 pour les clés de dépôts tiers. Aucune sortie.

2.2 — Y déposer la clé vérifiée à l’étape 1

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

Même URL qu’à l’étape 1 : on télécharge maintenant ce qu’on vient de valider. Aucune sortie.

2.3 — Relever les deux valeurs dont le dépôt a besoin

```
. /etc/os-release && echo $VERSION_CODENAME
```

```
dpkg --print-architecture
```

Résultat obtenu sur nas-host : `noble` et `amd64`.

Les afficher plutôt que de les glisser dans une substitution de commande permet de voir ce qu’on écrit réellement dans le fichier.

2.4 — Écrire le fichier de dépôt

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

Contenu :

```
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
```

Format « deb822 », plus lisible que l’ancienne ligne `deb [signed-by=…] URL suite composant`. Chaque champ est explicite : provenance, version d’Ubuntu, architecture, et quelle clé authentifie les paquets.

`lts-6.0` dans l’URL est le choix de branche. C’est cette seule ligne à modifier

en 2029, en fin de support.

Vérification

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

Pourquoi 6.0 et non 7.0 — le piège du noyau

La 7.0 LTS apporte les conteneurs applicatifs OCI, les pilotes Linstor et TrueNAS, un écouteur S3 intégré, les sauvegardes incrémentales de VM, des ensembles d'adresses réseau et diverses fonctions de grappe.

Rien de tout cela ne concerne un hôte unique portant cinq conteneurs système en macvlan sur ZFS.

Et surtout, la documentation est catégorique : *« The minimum supported kernel version is 6.12. »* nas-host tourne sur 6.8. Obtenir un noyau conforme imposerait `linux-generic-hwe-24.04` — un changement de noyau sur un serveur en production dont le `/boot` est sur une clé USB, pour des fonctionnalités inutilisées.

La 7.0 retire aussi CGroupV1 et xtables. Sans conséquence ici (cgroup v2 déjà actif, macvlan sans pare-feu géré par Incus), mais confirme qu'elle vise des systèmes plus récents.

Leçon à retenir : croiser les prérequis de la version visée avec le noyau en place, avant de choisir la branche.

Étape 3 — Rafraîchir apt et vérifier d'où viendra le paquet

3.1 — Mettre à jour la liste des paquets

```
sudo apt update
```

Une ligne doit mentionner `pkgs.zabbly.com`. Ce qui compte, c'est l'absence d'un message `NO_PUBKEY` ou « signatures couldn't be verified », qui signalerait un `Signed-By` mal dirigé.

3.2 — Regarder d'où viendra le paquet, avant de l'installer

```
apt policy incus
```

⚠ C'est l'étape qui a tout changé. Premier résultat obtenu :

```
Candidat : 6.0.0-1ubuntu0.3+esm3
     1:7.0.1-ubuntu24.04-...   500   pkgs.zabbly.com
     6.0.0-1ubuntu0.3+esm3     510   esm.ubuntu.com/apps/ubuntu
```

Ubuntu distribue son propre paquet `incus`, et l'abonnement Ubuntu Pro donne au dépôt ESM une priorité de 510, supérieure au 500 par défaut d'un dépôt tiers. Un `apt install incus` aurait installé le paquet Ubuntu — silencieusement, sans erreur.

Apt choisit d'abord par priorité, ensuite seulement par numéro de version. L'« epoch » `1:` en tête de la version Zabbly la placerait pourtant devant, mais la priorité tranche avant.

Prendre l'habitude de `apt policy` avant toute installation depuis un dépôt tiers.

3.3 — Donner la priorité à Zabbly

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

Contenu :

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

- `Pin: origin` désigne le nom d’hôte du dépôt, pas son champ « Origin » — confusion classique dans 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.
- `Package: *` couvre tout le dépôt sans énumérer `incus`, `incus-client`, `incus-ui-canonical`. Le dépôt ne contenant qu’Incus, la portée reste étroite.

3.4 — Revérifier

```
sudo apt update
```

```
apt policy incus
```

Résultat attendu :

```
Candidat : 1:6.0.6-ubuntu24.04-202603272003
     1:6.0.6-ubuntu24.04-...   600   pkgs.zabbly.com/incus/lts-6.0
     6.0.0-1ubuntu0.3+esm3     510   esm.ubuntu.com/apps/ubuntu
```

6.0.6 contre 6.0.0 : six versions correctives d’écart avec le paquet Ubuntu. C’est le bénéfice concret du dépôt Zabbly sur la même ligne LTS.

Étape 4 — Installer

4.1 — Vérifier l'origine de chaque paquet

```
apt policy incus-ui-canonical
```

Même réflexe que pour `incus` : candidat attendu depuis `pkgs.zabbly.com` à 600.

4.2 — Simuler avant d'installer

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

Le `--dry-run` ne modifie rien : apt calcule et affiche ce qu'il ferait. Sur une machine en production, c'est le réflexe qui évite les surprises.

Trois choses à regarder dans la sortie :

- Ce qui sera installé — 4 paquets Zabbly (`incus`, `incus-base`, `incus-client`, `incus-ui-canonical`) et une trentaine de dépendances Ubuntu.
- `0 à enlever` — la ligne capitale. Si apt proposait de retirer un paquet, s'arrêter.
- Absence de `qemu-system-x86` — le support des VM n'est pas installé. À ajouter le jour où une VM sera nécessaire.

À propos de `dnsmasq-base` : c'est la bibliothèque et le binaire, pas le paquet `dnsmasq` qui lance un service écoutant sur le port 53. Incus ne s'en sert que pour ses réseaux gérés — inutilisé en macvlan. Aucun conflit avec un Pi-hole sur le réseau.

4.3 — Installer

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

4.4 — Vérifier le démon

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

Résultat attendu, et il surprend :

```
Active: inactive (dead)
TriggeredBy: ● incus.socket
```

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. Au repos, il ne consomme rien. LXD, lui, garde son démon en permanence — différence de conception, pas défaut.

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

Celui-ci doit être `active (listening)`.

4.5 — Vérifier que LXD n'a pas bougé

```
lxc list -c ns
```

Les conteneurs LXD tournent toujours. Deux démons indépendants cohabitent sans se connaître.

Étape 5 — Autoriser son utilisateur

5.1 — Ajouter l'utilisateur au groupe d'administration

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

Le paquet crée deux groupes : `incus` (accès restreint) et `incus-admin` (contrôle complet). C’est le second qu’il faut.

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

5.2 — Se déconnecter et se reconnecter

```
exit
```

Le changement de groupe n’est lu qu’à l’ouverture de session. Dans le shell courant, `incus` répondrait `permission denied` sur le socket.

5.3 — Confirmer

```
id
```

`incus-admin` doit figurer dans la liste des groupes.

5.4 — Première commande Incus

```
incus version
```

Sans `sudo`. Elle réveille le démon par activation de socket.

```
Version du client : 6.0.6
Server version: 6.0.6
```

Les deux lignes doivent afficher la même version.

Topologie réseau — à comprendre avant d’initialiser

```
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 (SSH)
lxdbr0 10.248.34.1/24    DOWN — pont LXD créé par défaut, jamais utilisé
```

Deux cartes physiques sur le même sous-réseau, et c’est délibéré.

Un hôte ne peut normalement pas joindre ses propres conteneurs macvlan à travers l’interface parente : les paquets ne remontent jamais la pile réseau locale. Ici, les conteneurs sont enfants macvlan 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 fait fonctionner les sauvegardes rsync de l’hôte vers `192.168.0.6`. Sans la seconde carte, elles seraient impossibles.

⚠ Conséquence : la seconde carte n’est *pas* disponible pour un passthrough PCI vers une VM. La céder supprimerait le 192.168.0.11 de l’hôte et couperait la voie vers les conteneurs.

Deux règles qui en découlent pour l’initialisation :

1. Le profil macvlan d’Incus doit pointer vers `parent: eno1`, comme celui de LXD.
2. Refuser la création d’un pont réseau, sous peine de reproduire le `lxdbr0` inutile.

Étape 6 — Tester la compatibilité LXD ↔ Incus

Étape décisive : son résultat détermine la stratégie de migration.

6.1 — Exposer l'API de LXD sur la boucle locale

```
lxc config set core.https_address 127.0.0.1:8443
```

`127.0.0.1` uniquement : rien n'est exposé sur le LAN.

6.2 — Créer un jeton de confiance

```
lxc config trust add --name incus-migration
```

LXD affiche un jeton base64 à usage unique, valable deux semaines.

6.3 — Déclarer LXD comme dépôt distant auprès d'Incus

```
incus remote add lxd-local https://127.0.0.1:8443 --token=LE_JETON
```

Incus génère un certificat client, affiche son empreinte et demande confirmation. Répondre `o`.

6.4 — Le verdict

```
incus list lxd-local:
```

Résultat obtenu : les cinq conteneurs listés avec leurs IP, leur type et leur nombre d'instantanés.

Les API LXD et Incus sont donc restées compatibles, malgré la divergence des deux projets depuis 2023 — et malgré le fait que `incus remote add` ne documente plus de protocole `lxd`. La migration progressive est possible.

Si ce test avait échoué, la solution de repli était `lxd-to-incus`, qui migre tout l'hôte d'un coup — acceptable uniquement parce que les exports complets avaient été faits et vérifiés au préalable.

À ne pas oublier en fin de migration : refermer l'API de LXD avec `lxc config unset core.https_address`.

Étape 7 — Initialiser Incus

```
incus admin init
```

Sans `sudo` (l'utilisateur est dans `incus-admin`). Réponses données :

Question	Réponse	Pourquoi
Would you like to use clustering?	`no`	Hôte unique.
Configurer un nouveau pool de stockage ?	`yes` (défaut)	
Name of the new storage pool [default=default]	`local`	⚠ Surtout pas `default` : le zpool de LXD porte déjà ce nom, ZFS refuserait le doublon.
Storage backend	`zfs` (défaut)	Même moteur que LXD : instantanés et transferts optimisés.
Créer un nouvel agrégat ZFS ?	`yes` (défaut)	
Utiliser un périphérique bloc vide existant ?	`no` (défaut)	Le NVMe n'a qu'une partition ; pas de repartitionnement en production.
Size in GiB of the new loop device [default=30GiB]	`100`	Fichier creux : ne consomme que ce qui est écrit.
Créer un pont réseau local ?	`no`	⚠ Sinon Incus fabrique un `incusbr0` inutile, jumeau du `lxdbr0` qu'on veut voir disparaître.

Question	Réponse	Pourquoi
Utiliser un pont ou une interface existante ?	`no` (défaut)	Le profil `macvlan` sera créé à part, pour reproduire la structure de LXD.
Serveur disponible sur le réseau ?	`no` (défaut)	Sera ouvert plus tard, lié explicitement à l'IP d'administration.
Mettre à jour les images en cache ?	`yes` (défaut)	
Imprimer un preseed YAML ?	`yes`	Trace exacte, et fichier de réinitialisation réutilisable.

Preseed obtenu

```
config: {}
networks: []
storage_pools:
- config:
    size: 100GiB
  name: local
  driver: zfs
profiles:
- devices:
    root:
      path: /
      pool: local
      type: disk
  name: default
```

Un `incus admin init --preseed < fichier.yaml` reproduirait cette configuration sans aucune question — utile pour un second hôte.

Vérifications

```
incus storage list
```

```
sudo zpool list
```

C’est ici que la cohabitation devient visible :

```
default  27.5G  11.9G  15.6G   ← LXD
local    99.5G   622K  99.5G   ← Incus
```

```
incus profile show default
```

Ne doit contenir que le périphérique `root` pointant vers `pool: local`. Pas de carte réseau.

Étape 8 — Recréer le profil `macvlan`

8.1 — Créer le profil

```
incus profile create macvlan
```

8.2 — Y ajouter la carte réseau

```
incus profile device add macvlan eth0 nic nictype=macvlan parent=eno1
```

Décomposition, car cette commande porte toute la topologie réseau du parc :

Élément	Rôle
`macvlan`	le profil auquel on ajoute
`eth0`	le nom du périphérique tel que le conteneur le verra — celui que les netplan configurent
`nic`	le type de périphérique Incus
`nictype=macvlan`	le conteneur obtient sa propre MAC directement sur le réseau physique
`parent=eno1`	l’interface hôte à laquelle il s’accroche

⚠ `parent=eno1` est le point critique. Avec `eno2`, les conteneurs resteraient

joignables depuis le LAN, mais l'hôte ne pourrait plus les atteindre — et les sauvegardes rsync tomberaient sans erreur visible.

8.3 — Comparer avec l'original

```
incus profile show macvlan
```

```
lxc profile show macvlan
```

Les deux blocs `devices` doivent être identiques. C'est ce contrôle qui garantit qu'un conteneur copié retrouvera exactement le même réseau.

Étape 9 — Migrer un conteneur (modèle)

Séquence appliquée à `navidrome` en premier, puis répétée pour les autres. Commencer par le conteneur le moins critique.

9.1 — Lire la configuration source avant de copier

```
lxc config show navidrome
```

Chercher trois choses en particulier :

- Les périphériques disque montés depuis l'hôte — leurs *définitions* suivent la copie, les *données* ne bougent pas. Chemins identiques puisque c'est le même hôte.
- `raw.idmap` — la projection d'identifiants qui permet à un conteneur non privilégié d'accéder aux fichiers de l'hôte avec les bonnes permissions. C'est l'alternative propre au `security.privileged`.
- `boot.autostart` — s'il vaut `true`, il faudra le désactiver côté LXD après validation (étape 9.7).

9.2 — Arrêter l'original

```
lxc stop navidrome
```

On copie à l'arrêt : la copie est alors cohérente au niveau du système de fichiers, sans base de données saisie en pleine écriture.

9.3 — Copier

```
incus copy lxd-local:navidrome navidrome
```

`lxd-local:navidrome` = le conteneur sur le dépôt distant. Le second, sans préfixe, est la destination locale dans Incus.

⚠ Ne pas ajouter `--storage local`. La racine vient déjà du profil `default` d'Incus. L'option crée un périphérique `root` explicite sur l'instance, divergence inutile avec la structure d'origine. (Réparable après coup : `incus config device remove <nom> root`.)

9.4 — Comparer

```
incus config show navidrome
```

Doivent être identiques : les périphériques, `raw.idmap`, les profils, les limites.

Différences normales :

Clé	Observation
`volatile.eth0.hwaddr`	MAC régénérée — préfixe `10:66:6a` (Incus) au lieu de `00:16:3e` (LXD/Xen). Sans conséquence si l'IP est statique dans le netplan ; à corriger si des règles au routeur s'indexent sur la MAC.

Clé	Observation
`volatile.cloud-init.instance-id`	Nouvel identifiant — cloud-init voit une instance neuve et *peut régénérer sa configuration réseau*. D'où l'importance de l'avoir neutralisé au préalable.
`volatile.apply_template: copy`	Normal sur une copie.

9.5 — Démarrer la copie

```
incus start navidrome
```

⚠ Règle absolue : l'original est arrêté depuis 9.2, la copie démarre maintenant. Jamais les deux ensemble — même IP sur le LAN, et corruption si un disque hôte est partagé entre les deux.

9.6 — Vérifier

```
incus exec navidrome -- ip -br addr
```

```
incus exec navidrome -- ls -la /mnt/musique
```

Le second est le contrôle décisif quand `raw.idmap` est en jeu : des propriétaires normaux confirment que la projection a suivi. Des `nobody` ou `65534` signifieraient le contraire.

Puis tester le service lui-même dans un navigateur, pas seulement le réseau.

9.7 — Désactiver le démarrage automatique de l'original

```
lxc config set navidrome boot.autostart false
```

⚠ À ne jamais sauter. Sans cette commande, au prochain redémarrage de l'hôte, LXD et Incus lanceraient chacun leur exemplaire — même IP, conflit garanti, et diagnostic pénible puisque tout aura fonctionné jusque-là.

Le conteneur LXD reste en place, arrêté : c’est le filet de retour immédiat.

Étape 10 — Les conteneurs suivants

La séquence de l’étape 9 se répète. Deux cas rencontrés méritent d’être notés.

`squid-proxy` — le cas nu

```
devices: {}
```

Aucun disque monté, aucun `raw.idmap`. La migration est purement mécanique.

Vérifier que le service écoute ne suffit pas ; vérifier qu’il fonctionne de bout en bout, depuis une autre machine :

```
curl -x 192.168.0.7:3128 -sI https://www.google.com | head -1
```

→ `HTTP/1.1 200 Connection established`

`languagetool` — la clé cloud-init côté instance

Sa configuration LXD portait :

```
cloud-init.network-config: |-
  version: 2
  ethernets:
    eth0:
      addresses: [192.168.0.10/24]
      nameservers:
        addresses:
          - 192.168.0.5
```

C’est la source du `50-cloud-init.yaml` trouvé dans le conteneur. Le travail de

neutralisation avait traité le symptôme à l'intérieur ; voici la cause, un étage au-dessus. Elle ne contenait d'ailleurs que le Pi-hole, sans résolveur de secours.

Inerte une fois cloud-init désactivé dans le conteneur, mais trompeuse. À retirer après la copie, côté Incus seulement — l'original LXD garde son état, c'est le principe du filet :

```
incus config unset languagetool cloud-init.network-config
```

Contrôle spécifique après démarrage : c'est bien notre netplan qui est en place, pas un `50-cloud-init.yaml` ressuscité par le nouvel `instance-id`.

```
incus exec languagetool -- cat /etc/netplan/50-static-public-ip.yaml
```

Deux leçons sur la vérification des services

Ne pas tester depuis l'hôte

Un `curl` lancé depuis nas-host vers un conteneur macvlan peut geler sans erreur : selon la route empruntée, les paquets partent par `eno1`, l'interface parente, où le macvlan interdit précisément le dialogue hôte↔conteneur. Ils disparaissent sans RST, donc sans « connection refused ».

Tester les services depuis une autre machine du LAN, jamais depuis l'hôte — sous peine de diagnostiquer un problème réseau qui n'existe pas.

Tester le service tel qu'on l'utilise réellement

L'accès direct `IP:port` peut n'avoir jamais fonctionné, indépendamment de la migration. Ce qui compte est le chemin réel : pour LanguageTool,

`https://lt.example.com/v2/languages` à travers le proxy inverse d'Apache sur `web-srv`.

Avant de chercher une panne, se demander : est-ce que ça marchait comme ça avant ?

Lire les journaux du service plutôt que sonder

La preuve la plus solide de la réussite est venue du journal de LanguageTool :

```
Handling POST /v2/check ... Handled request in 159ms; sending code 200
```

Un vrai client servi après la migration vaut mieux que n'importe quelle sonde artificielle.

`pi-hole` — le service dont tout dépend

Mécaniquement le plus simple : `devices: {}`, aucun `raw.idmap`, aucune clé cloud-init. La difficulté est purement opérationnelle : le DNS de toute la maison est coupé pendant la copie (~4 min pour 2,67 Gio). À faire à une heure creuse — 6 h 24 s'est avéré idéal.

Les autres conteneurs ayant un résolveur de secours, seuls les appareils du foyer qui reçoivent le DNS par DHCP sont concernés.

Particularité relevée : `pi-hole` n'avait aucune clé `boot.autostart`. Il revenait au démarrage par le mécanisme de restauration d'état (`volatile.last_state.power`). Pour un service dont tout dépend, mieux vaut l'écrire explicitement du côté Incus :

```
incus config set pi-hole boot.autostart true
```

Vérification en trois temps — et le troisième est celui qu'on oublie :

```
incus exec pi-hole -- ss -lntup | grep :53
```

→ `pihole-FTL` en écoute, UDP et TCP, IPv4 et IPv6.

```
dig @192.168.0.5 www.google.com +short
```

→ des adresses : la résolution fonctionne.

```
dig @192.168.0.5 doubleclick.net +short
```

→ `0.0.0.0` : le blocage fonctionne.

⚠ Sans ce dernier test, un Pi-hole qui résout mais ne filtre plus passerait tous les contrôles de connectivité. On s'en apercevrait des semaines plus tard, en voyant la publicité revenir. Tester la fonction du service, pas seulement sa présence.

Le tableau de bord confirme ensuite que la base de FTL a suivi : historique des requêtes, compteurs et listes intacts.

`web-srv` — le conteneur critique

Le dernier, et le seul qui demande une préparation particulière : conteneur privilégié, disque hôte de 1,7 To, 7 instantanés, cible du 443 public, et proxy inverse pour trois autres conteneurs.

Décision sur les instantanés. Les sept dataient de décembre 2022 – janvier 2023 : `postfix`, `phpmyadmin`, `ISPconfig_clean`, `fullinstall`, `fullinstallwscripts`, `installationcomplete`. Des jalons de construction, pas des points de restauration – et rien des trois années suivantes.

Ils restent conservés à deux endroits : sur l'original LXD gardé en filet, et dans l'archive d'export (`lxc export` les emporte par défaut). Donc :

```
incus copy lxd-local:web-srv web-srv --instance-only
```

Copie plus rapide, pool Incus qui démarre propre.

Vérifier AVANT de démarrer — c'est le seul conteneur où ça compte vraiment :

```
incus config show web-srv
```

Deux clés à confirmer impérativement :

```
security.privileged: "true"
devices:
  web:
    path: /var/www
    source: /dev/disk/by-id/scsi-SATA_...-part2
```

Si le périphérique `web` manquait, Apache démarrerait sur un `/var/www` vide.

Vérification en couches, du plus bas au plus haut :

```
incus exec web-srv -- ip -br addr
```

```
incus exec web-srv -- df -h /var/www
```

→ doit annoncer 1,7 To : preuve que le disque hôte est monté dans la copie.

```
incus exec web-srv -- systemctl is-active apache2 mysql radicale bind9
```

Puis le démon rsync, avant tout le reste — c'est lui qui sert les sauvegardes nocturnes :

```
incus exec web-srv -- ss -lntp | grep 873
```

```
rsync --list-only rsync://192.168.0.6/
```

Cette dernière commande, lancée depuis l'hôte (la machine qui fait les sauvegardes), teste exactement le chemin de production. Elle a révélé 14 modules là où la crontab n'en exploite que 6 — dont `courriel`, `radicale_users` et `radicale_data`, qui mériteraient d'être sauvegardés eux aussi.

Enfin le navigateur, en n'oubliant pas ce que ce conteneur publie pour les autres :

À vérifier	Ce que ça teste
`https://blog.example.com`, `https://photos.example.com`	les sites eux-mêmes
ISPConfig	l'administration
`https://musique.example.com`	le proxy inverse vers Navidrome
`https://lt.example.com/v2/languages`	le proxy inverse vers LanguageTool
`https://routeur.example.com`	le proxy inverse vers l'admin du routeur
Le cadenas	le certificat wildcard `*.example.com`

Si `musique.example.com` échoue alors que `192.168.0.8` répond, c'est Apache qu'il faut regarder, pas Navidrome.

Étape 13 — Interface web d'Incus

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

⚠ Lier à l'IP d'administration, jamais `:8443` tout court sur une machine dont le 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 CLI le fait sans qu'on s'en aperçoive (`~/.config/incus/`) ; le navigateur exige une manipulation manuelle.

Ordre obligatoire, et il 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. Deux dialogues apparaissent, il faut répondre aux deux :
5. la boîte « X vous demande de vous identifier » → choisir le certificat, De façon permanente, OK
6. la page `SELF_SIGNED_CERT` → Continuer (c'est le certificat *du serveur*, auto-signé)
7. Seulement maintenant, l'onglet « Identity trust token » devient utilisable — son avertissement jaune disparaît. Générer un jeton et le coller :

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

Pièges rencontrés :

- `incus config trust add --name X` n'existe pas. Le nom est positionnel : `incus config trust add X`. Divergence de syntaxe avec LXD.
- Un jeton est à usage unique : la première soumission consomme l'opération en attente, qu'elle aboutisse ou non. Un échec suivi d'une nouvelle tentative donne `No matching certificate add operation found`. Regénérer plutôt que réessayer.
- Les jetons LXD et Incus ont un format identique. Vérifier le champ `addresses` : `127.0.0.1:8443` = LXD, `192.168.0.11:8443` = Incus.
- Le certificat du client CLI n'apparaît pas dans `incus config trust list` : il

> passe par le socket Unix, autorisé par le groupe `incus-admin`. Seuls les clients réseau ont besoin d'un certificat.

Étape 14 — Nettoyage final *(à faire après quelques jours de stabilité)*

Ne rien retirer avant d'avoir vécu au moins une nuit de sauvegardes et un redémarrage complet de l'hôte.

Retirer LXD Mosaic, devenu inutile :

```
sudo snap remove lxdmosaic
```

Retirer le remote et refermer l'API de LXD :

```
incus remote remove lxd-local
```

```
lxc config unset core.https_address
```

Retirer LXD, une fois la confiance acquise :

```
sudo snap remove lxd
```

⚠ Cette commande détruit les conteneurs LXD originaux — le filet de retour — ainsi que le pool `default.img` de 27,5 Go et le pont `lxdbr0`. Les archives d'export sur `/media/nas1/Backups/lxd-images_pre-migration/` restent, elles, la dernière ligne de défense.

Vérifier ensuite que l'espace est bien rendu :

```
df -h /
```

```
zpool list
```

Seul `local` doit subsister.

Étape 15 — Le redémarrage de validation

Ne rien retirer avant d'avoir redémarré l'hôte au moins une fois. C'est ce test qui révèle ce que la migration a laissé passer — et il a effectivement révélé quelque chose.

⚠ Ce qui s'est produit

Au redémarrage, `incus list` montrait les cinq conteneurs Incus corrects. Mais `lxc list` montrait :

```
| web-srv | RUNNING | 192.168.0.6 (eth0) |
```

Deux conteneurs sur la même IP, montant le même disque hôte. La commande `lxc config set web-srv boot.autostart false` avait été donnée dans la procédure mais jamais exécutée — et personne ne l'avait vérifiée avant de déclarer la migration terminée.

Correction immédiate :

```
lxc stop web-srv
```

```
lxc config set web-srv boot.autostart false
```

Bilan des dégâts : aucun. Un `find /var/www -newermt '-1 hour' -type f` n'a révélé que des journaux d'accès et un fichier de cache Matomo. Rien des données de Radicale ni du courriel.

Sur le disque partagé : Linux détecte qu'un périphérique bloc est déjà monté et partage le superbloc existant plutôt que d'en créer un second — le deuxième montage se comporte comme un montage lié. La corruption survient quand deux noyaux indépendants écrivent, pas dans ce cas. Le conflit d'adresse IP, lui, était bien réel.

Leçon : après chaque migration, vérifier que `boot.autostart false` a bien été appliqué à l'original, plutôt que de supposer que la commande a été lancée.

```
lxc list
```

Les cinq doivent être `STOPPED`. C'est le seul contrôle qui compte au premier redémarrage.

Le reste du bilan post-redémarrage

```
sudo zpool list
```

Les deux pools doivent être importés. `default` compte encore : c'est le filet de retour.

```
systemctl --failed
```

Ce tamis a révélé deux dettes anciennes, sans rapport avec la migration :

- `lxdui.service` — une interface LXD abandonnée depuis 2022, en échec à chaque démarrage. `sudo systemctl disable --now lxdui.service`
- `postfix@-.service` — `postconf: fatal: open /etc/postfix/main.cf: No such file or directory`, à chaque démarrage depuis au moins juillet. Le `master.cf` daté de

2022 prouve qu’il avait été configuré ; `main.cf` a disparu depuis.

⚠ Conséquence de ce second point : l’hôte ne pouvait plus envoyer de courriel. Or les scripts de sauvegarde commencent par un `echo` que cron expédie par courriel. Les rapports de sauvegarde se perdaient depuis des mois.

Vérifier que les sauvegardes ont bien tourné

⚠ Ne pas se fier aux dates des fichiers sauvegardés. `rsync -a` préserve la date de la source : le fichier le plus récent dans la sauvegarde porte la date qu’il avait sur l’origine, pas celle de la copie. Même piège que la date d’un répertoire, qui ne change que si on y ajoute ou retire une entrée.

La question se pose au cron :

```
journalctl -u cron.service --since "2026-08-31 23:00" --no-pager | grep -i backup
```

(`journalctl` n’analyse pas les dates en français — utiliser le format ISO plutôt que « hier ».)

Remplacer le courriel par un journal

Plus robuste qu’un MTA : rediriger la sortie des tâches vers des fichiers datés.

```
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`.

Puis dans `crontab -e` :

```
MAILTO=""

15 00 * * * sh /home/hostadmin/scripts/rsync/auto/backup_mysql-ns2.sh >>
/var/log/backup/$(date +\%F).log 2>&1
…
30 03 1 * * find /var/log/backup/ -name '*.log' -mtime +180 -delete
```

⚠ Une ligne de cron ne se coupe pas. Cron n’accepte aucune continuation par barre oblique inverse : chaque tâche tient sur une seule ligne. Dans le PDF elle est enroulée 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. C’est l’un des pièges les plus classiques de cron.

- `2>&1` capture aussi les erreurs — sans lui, seuls les succès seraient journalisés.
- `MAILTO=""` dit à cron de ne rien tenter d’envoyer.

Enfin, faire taire le MTA devenu inutile :

```
sudo systemctl disable --now postfix@-.service
```

```
sudo systemctl disable --now postfix.service
```

Tester sans attendre le lendemain — dans le shell, `%F` s’écrit sans échappement :

```
sh /home/hostadmin/scripts/rsync/auto/backup_scripts.sh \
  >> /var/log/backup/$(date +%F).log 2>&1
```

```
cat /var/log/backup/$(date +%F).log
```