Sauvegarde — Stratégie & Méthode

Guide de reconstruction. Il décrit le système de sauvegarde d'un petit parc de machines : ce qu'il protège, comment il fonctionne, et surtout ce qu'il ne couvre pas.

Ce que cette page couvre : le mécanisme commun à toutes les sauvegardes du parc — un démon rsync sur chaque machine, un serveur qui tire —, les quatre niveaux de protection et leurs angles morts, et les moyens de savoir que tout cela fonctionne encore. Remettre les données en place fait l'objet de la page Restauration.

Contexte : un serveur qui cumule le rôle de NAS et d'hôte de conteneurs, un conteneur web sous ISPConfig, un poste de travail. Les adresses, noms d'hôtes et chemins sont des exemples à transposer.

Trois choses valent le détour même sans monter le même système. Que le filtrage des modules rsync s'applique côté serveur et non côté client — ce qui décide de ce qu'un mot de passe volé donnerait vraiment : section *Le filtrage*. Que les angles morts des niveaux hors site doivent être complémentaires plutôt que superposés, ce qui est le seul raisonnement qui fasse tenir l'ensemble : section *Les quatre niveaux*. Et qu'une tâche qui réussit à ne rien faire est plus difficile à repérer qu'une tâche qui échoue : section *Savoir que ça fonctionne*.

Convention : chaque bloc de commandes indique la machine où il s'exécute.

---

1. Le principe : tirer, jamais pousser

Chaque machine à sauvegarder expose des modules par un démon rsync, sur le port

873, déclarés dans son `/etc/rsyncd.conf`. Le serveur de sauvegarde va les chercher.

```
serveur ──rsync://──► conteneur web    ──► /media/nas1/Backups/
serveur ──rsync://──► lui-même         ──► /media/nas1/Backups/
serveur ──rsync://──► poste de travail ──► /media/nas1/Backups/
```

La syntaxe `::` désigne le démon, pas SSH :

```
rsync -avz --stats --delete --password-file
/home/hostadmin/scripts/rsync/auto/rsync_pass test@192.168.0.6::photos/
/media/nas1/Backups/web-srv_22.04_www/photos
```

Trois propriétés découlent de ce choix, et ce sont elles qui justifient tout le reste.

1. Tout passe par le réseau. Aucun accès au stockage des conteneurs depuis l'hôte n'est nécessaire. □□ C'est ce qui a permis de migrer l'hôte d'un gestionnaire de conteneurs à un autre sans réécrire un seul chemin de sauvegarde : les modules ne savent pas où vivent les données, ils savent seulement les servir.

2. Les modules sont en lecture seule. Le démon ne peut rien écrire. ⚠□ Conséquence à connaître avant d'en avoir besoin : le chemin de restauration n'est pas l'inverse du chemin de sauvegarde. On ne restaure pas en renversant une commande — voir la page Restauration.

3. Le trafic 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 fermé ; à ne pas reproduire au-delà.

Le contrôle d'accès tient en trois couches, et aucune ne suffit seule :

Couche	Ce qu'elle fait
`hosts allow = 192.168.0.11`	Seul le serveur de sauvegarde peut ouvrir une connexion

Couche	Ce qu’elle fait
`auth users = test`	Un compte propre au démon, sans existence sur le système
`secrets file` + `--password-file`	Le mot de passe, côté serveur et côté client

□□ Le compte `test` n’est pas un compte Unix. Le démon tient sa propre liste ; ce nom n’ouvre aucune session, aucun shell, et n’apparaît pas dans `/etc/passwd`.

⚠□ Le fichier de mots de passe doit être en `600`, sans quoi rsync refuse de s’en servir. C’est une des rares protections que l’outil impose plutôt que de suggérer.

---

2. Les quatre niveaux, et ce que chacun ne couvre pas

C’est la section la plus utile de cette page. Une sauvegarde qu’on croit plus large qu’elle ne l’est vaut moins qu’une sauvegarde dont on connaît les limites.

Niveau	Quoi	Quand	Où
Quotidien	Données des services, copie cohérente	Chaque nuit	`/media/nas1/Backups/`
Mensuel	Export complet des conteneurs	Le 1er, deux mois conservés	`/media/nas1/Backups/incus-exports/`
Hors site	Miroir du mensuel, plus la clé de démarrage	Continu	Proton Drive
Figé	État d’avant une migration	Ponctuel	À ne jamais purger

Aucun ne suffit seul. Le quotidien restaure des données, le mensuel reconstruit un serveur, le hors site survit à la perte de la machine, le figé permet le retour en arrière.

Et voici ce que chacun laisse passer :

Niveau	Ne couvre pas
Quotidien	La perte du serveur lui-même — les copies y sont
Mensuel	Le contenu des disques montés depuis l'hôte : `incus export` ne les capture pas
Hors site	L'erreur et la malveillance : le miroir n'a aucune révision, une archive corrompue est répliquée telle quelle
Hors site	Tout ce qui n'est pas dans son périmètre : les fichiers des sites, la bibliothèque audio
Disques externes	L'écart depuis la dernière copie manuelle — jusqu'à deux mois
Disques externes	Un sinistre au domicile pendant qu'ils y sont branchés
Tous	Les systèmes d'exploitation : reconstruits depuis le wiki, pas restaurés

2.1 Le recoupement qui fait tenir l'ensemble

C'est le raisonnement à retenir, et il vaut au-delà de ce parc : les angles morts des niveaux hors machine doivent être complémentaires, pas superposés.

- Ce que le nuage ne porte pas — les fichiers des sites, l'audio —, les disques externes le portent.
- Ce que le nuage ne protège pas — l'erreur, la corruption, le rançongiciel, qu'un miroir réplique fidèlement —, la coupure physique des disques le protège absolument.
- Ce que les disques n'ont pas — la fraîcheur —, le nuage l'a.

Il ne reste qu'un seul point où les deux échouent ensemble : un sinistre au domicile, disques présents, sur des données modifiées depuis la dernière copie manuelle. Risque réel mais étroit, et résultat d'un arbitrage assumé plutôt que d'un oubli.

⚠ Deux copies au même endroit ne font pas deux niveaux. Un miroir continu vers le nuage et une sauvegarde locale protègent du même sinistre — la panne

matérielle — et d’aucun autre. C’est le débranchement qui crée le second niveau, pas la duplication.

---

3. Anatomie d’un module

Tous les modules suivent la même forme. La connaître dispense de relire chacun.

```
[nom_du_module]
uid = compte
gid = groupe
path = /chemin/servi
exclude = lost+found/ **
include = fichier-a-servir
hosts allow = 192.168.0.11
comment = Ce qui s'affiche dans la liste des modules
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt
```

`uid` et `gid` décident de ce que le démon peut lire. Un module qui sert un site déclare le compte de ce site ; un module qui sert `/etc` déclare `root`.

`comment` n’est pas décoratif. Il s’affiche quand on liste les modules d’une machine, et c’est ce qui rend l’inventaire lisible :

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

□□ Écrire des libellés descriptifs plutôt que « `Backup <nom du module>` », qui ne fait que répéter le nom. La liste ci-dessus est le seul endroit où l’on voit tout le parc d’un coup d’œil ; autant qu’elle dise quelque chose.

⚠□ Aucun redémarrage n’est nécessaire après une modification. Le démon relit sa configuration à chaque connexion entrante. Un `systemctl restart` après édition ne fait que couper les transferts en cours.

4. Le filtrage vit côté serveur, pas côté client

C’est le point le plus important de cette page, et le plus facile à manquer.

Le défaut, tel qu’il existait ici. Plusieurs modules déclaraient `path = /etc/` sans aucun filtre, et le tri était fait côté client, dans les scripts, par des `--include` sur la ligne de commande.

Ça fonctionne — les scripts ne récupèrent que ce qu’ils demandent. Mais :

```
rsync --list-only --password-file=… test@192.168.0.11::rsync_nas-host/
```

…retournait tout `/etc/`. Deux cents entrées, dont `shadow`, `gshadow`, `sudoers`, le fichier de mots de passe du démon lui-même, et les répertoires `ssh/` et `letsencrypt/`.

 Le filtre client dit ce qu’on demande, pas ce qu’on pourrait demander. Le mot de passe du démon protégeait l’accès, donc il n’y avait pas d’escalade — mais une fuite de ce seul mot de passe aurait livré tout `/etc/` au lieu d’un fichier de configuration. La différence entre les deux situations ne se voit dans aucun journal et n’apparaît dans aucun test fonctionnel : les sauvegardes marchaient parfaitement dans les deux cas.

Le correctif, deux lignes dans le module :

```
exclude = lost+found/ **
include = rsyncd.conf
```

Après quoi le même `--list-only` ne rend plus que deux lignes.

⚠ L’ordre de ces deux lignes dans le fichier n’a aucune importance. Le démon

assemble les règles par type de paramètre, dans un ordre fixe où `include` précède toujours `exclude`. La première règle qui correspond l'emporte : `rsyncd.conf` est retenu avant que le `**` ne rejette le reste. C'est contre-intuitif pour qui a l'habitude des filtres en ligne de commande, où l'ordre d'écriture fait loi.

□□ Un effet de bord instructif. En élargissant un motif de `ices2_*_config.xml` à `ices2*config.xml`, un fichier a été récupéré que l'ancien motif ne pouvait pas atteindre — il exigeait quelque chose entre `ices2_` et `_config.xml`. Ce fichier n'avait jamais été sauvegardé, et rien ne le signalait : un filtre trop étroit ne produit pas d'erreur, il produit un silence.

La règle qui en découle, et qui vaut pour tout module : *réduire ce qu'un composant peut faire, pas seulement ce qu'il fait.*

---

5. L'exception : le seul module inscriptible

Un module du parc porte `read only = false`. Il sert à téléverser des images vers la galerie d'un site, pas à sauvegarder.

```
[gallerie]
uid = web2
gid = client1
path = /var/www/clients/client1/web2/web/galleries
hosts allow = 192.168.0.11
comment = Upload photos.example.com
read only = false
auth users = test
secrets file = /etc/rsyncd.scrt
```

⚠□ Son chemin est à l'intérieur de la racine web d'un site en ligne. Un module inscriptible pointant là où le serveur web sert des fichiers mérite d'être connu pour ce qu'il est : ce qui borne le risque n'est pas le module mais le

`hosts allow`, qui n’autorise qu’une seule machine à ouvrir la connexion.

⚠️ Ne pas s’en servir comme chemin de restauration. Il ne vise qu’un répertoire, et l’employer ainsi ferait de la seule brèche voulue un outil général.

---

6. Le fichier complet du conteneur web

Treize modules. Écrits en entier, parce qu’un tableau récapitulatif ne se recopie pas sans faute.

`/etc/rsyncd.conf` :

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

[blog]
uid = web1
gid = client1
path = /var/www/clients/client1/web1/web
hosts allow = 192.168.0.11
comment = Backup blog.example.com
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[webdav]
uid = web1
gid = client1
path = /var/www/clients/client1/web1/webdav/laflaf
hosts allow = 192.168.0.11
comment = Backup blog.example.com/webdav
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[photos]
uid = web2
```

```
gid = client1
path = /var/www/clients/client1/web2/web
hosts allow = 192.168.0.11
comment = Backup photos.example.com
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[gallerie]
uid = web2
gid = client1
path = /var/www/clients/client1/web2/web/galleries
hosts allow = 192.168.0.11
comment = Upload photos.example.com
read only = false
auth users = test
secrets file = /etc/rsyncd.scrt

[cumulus]
uid = web6
gid = client1
path = /var/www/clients/client1/web6/web
hosts allow = 192.168.0.11
comment = Backup cumulus.example.net
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[mysql]
uid = webadmin
gid = webadmin
path = /home/webadmin/backups/mysql_DB
hosts allow = 192.168.0.11
comment = Backup mysql
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[scripts_NS2]
uid = webadmin
gid = webadmin
path = /home/webadmin/scripts/
```

```
hosts allow = 192.168.0.11
comment = Backup scripts_NS2
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[radicale_users]
uid = root
gid = root
path = /etc/radicale/
hosts allow = 192.168.0.11
comment = Backup radicale users and config file
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[radicale_data]
uid = root
gid = root
path = /var/lib/radicale/collections/
hosts allow = 192.168.0.11
comment = Backup radicale data
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[rsync_NS2]
uid = root
gid = root
path = /etc/
exclude = lost+found/ **
include = rsyncd.conf
hosts allow = 192.168.0.11
comment = Backup rsync_NS2
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[crontab_NS2]
uid = root
gid = root
path = /var/spool/cron/crontabs/
```

```
exclude = lost+found/ **
include = webadmin
hosts allow = 192.168.0.11
comment = Backup crontab_NS2
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[www-full]
uid = root
gid = root
path = /var/www/
hosts allow = 192.168.0.11
comment = Backup full path of www disk
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt

[log_NS2]
uid = webadmin
gid = webadmin
path = /var/log/backup
hosts allow = 192.168.0.11
comment = Journal des taches nocturnes de ns2
read only = true
auth users = test
secrets file = /etc/rsyncd.scrt
```

Quatre remarques sur ce fichier.

⚠ `crontab_NS2` doit être filtré, sinon il sert toutes les crontabs du système, celle de root comprise. Le `include` nomme le seul compte dont la crontab a une valeur propre ; celle de root n’en a pas ici, ses deux lignes étant posées par l’installateur du panneau et donc reproductibles.

⚠ `www-full` recouvre plusieurs autres modules. Il sert `/var/www/` en entier — donc les fichiers de `blog`, `photos` et `webdav` une seconde fois, vers une autre destination. Ce n’est pas un doublon inutile : c’est ce qui fait que la suppression d’un site se propage sans qu’on touche à rien, et c’est aussi ce

qui explique qu'un `--delete` puisse retirer des dizaines de milliers de fichiers d'un coup après un ménage.

□□ `log_NS2` sert le journal des tâches nocturnes du conteneur. Sans lui, savoir si un vidage de base a échoué obligerait à entrer dans le conteneur pour lire un fichier. Exposer son propre journal est ce qui rend la surveillance possible depuis un seul endroit.

□□ Le côté serveur de sauvegarde — ses propres modules, ses scripts de tirage et sa crontab — est décrit dans la page de l'hôte, qui en a besoin pour se reconstruire.

---

7. Le script qui fabrique les vidages de bases

Il tourne dans le conteneur, pas sur le serveur de sauvegarde : c'est là que sont les bases. Le serveur ne fait que tirer le résultat par le module `mysql`.

`/home/webadmin/scripts/backup_mysql.sh` :

```
#!/bin/bash
# Vidage des bases MySQL vers ~/backups/mysql_DB/
# Identifiants dans ~/.my.cnf — jamais sur la ligne de commande.
# En cas d'échec, la sauvegarde précédente est CONSERVÉE.
DEST="/home/webadmin/backups/mysql_DB"
ERREURS=0
echo "=== $(basename "$0") — début $(date -Is)"
mkdir -p "$DEST"
# Découverte des bases, en excluant les schémas système
BASES=$(mysql -N -B -e "SHOW DATABASES" | grep -Ev
'^(information_schema|performance_schema|mysql|sys)$')
[ -z "$BASES" ] && { echo "ÉCHEC : aucune base listée — MySQL répond-il ?"; exit
1; }
for db in $BASES; do
    TMP="$DEST/.$db.sql.tmp"
```

```
    if mysqldump --single-transaction --quick \
                 --routines --triggers --events \
                 --databases "$db" > "$TMP"; then
        mv "$TMP" "$DEST/$db.sql"
        echo "OK     $db  ($(ls -lh "$DEST/$db.sql" | awk '{print $5}'))"
    else
        rm -f "$TMP"
        echo "ÉCHEC  $db — la sauvegarde précédente est conservée"
        ERREURS=$((ERREURS + 1))
    fi
done
echo ""
# Retrait des vidages dont la base n'existe plus.
# Uniquement si tout a réussi : sur une liste partielle, on effacerait
# des sauvegardes valides.
if [ "$ERREURS" -eq 0 ]; then
    for f in "$DEST"/*.sql; do
        [ -e "$f" ] || continue
        db=$(basename "$f" .sql)
        if ! printf '%s\n' "$BASES" | grep -qx "$db"; then
            rm -f "$f" && echo "RETIRÉ $db.sql — la base n'existe plus"
        fi
    done
fi
if [ "$ERREURS" -eq 0 ]; then
    echo "Toutes les bases ont été sauvegardées dans $DEST"
else
    echo "$ERREURS base(s) en échec — voir ci-dessus."
fi
echo "La sauvegarde sera complète lorsque nas-host tirera le module « mysql »."
echo "=== $(basename "$0") — fin $(date -Is) — échecs : $ERREURS"
exit "$ERREURS"
```

Cinq décisions valent d’être expliquées, parce qu’aucune n’est évidente.

1. Il découvre les bases plutôt que de les nommer. `SHOW DATABASES` moins les schémas système. Une base ajoutée est sauvegardée dès le lendemain sans qu’on y pense, et une base supprimée cesse d’elle-même d’être vidée. ⚠ Un script qui liste les bases en dur échoue en silence le jour où l’on en ajoute une — et c’est la sorte d’oubli qu’on découvre en cherchant une sauvegarde qui n’existe

pas.

2. Il écrit d’abord dans un fichier temporaire, puis renomme. Si `mysqldump` échoue à mi-parcours, la version de la veille reste intacte. Écrire directement sur la destination remplacerait une bonne sauvegarde par une mauvaise, ce qui est pire que de ne pas sauvegarder du tout.

3. Il refuse de continuer si la liste des bases est vide. Une liste vide ne veut pas dire « rien à sauvegarder », elle veut dire « MySQL ne répond pas ». Sans ce garde-fou, le script se terminerait avec succès en n’ayant rien fait.

4. Il nettoie les vidages orphelins — mais seulement si tout a réussi. Un fichier dont la base n’existe plus resterait indéfiniment, et le `--delete` du tirage le recopierait fidèlement puisqu’il existe encore à la source. ⚠ Le garde-fou est le cœur de cette partie : si MySQL rendait une liste partielle, le nettoyage effacerait des sauvegardes parfaitement valides. Un script de sauvegarde qui détruit sur une erreur est pire que celui qui ne nettoie pas.

5. Les identifiants sont dans `~/.my.cnf`, jamais sur la ligne de commande. Un mot de passe passé en argument est visible par tout compte de la machine dans la liste des processus, le temps de l’exécution.

⚠ Ce fichier d’identifiants existe et doit être en `600`. Cette page dit qu’il existe, où il est et quelles permissions il porte — jamais ce qu’il contient. La même règle vaut pour le fichier de mots de passe du démon rsync.

---

8. Ordonnancement

Les tirages sont espacés plutôt que simultanés : ils lisent le même disque de destination, et les faire se chevaucher ne ferait que les ralentir mutuellement.

⚠ L’ordre relatif compte plus que les heures elles-mêmes. Le vidage des bases est produit dans le conteneur à 00:00 ; le serveur le tire à 00:15. Quinze

minutes de marge. Si les bases grossissent au point que le vidage dépasse ce délai, le serveur tirerait un fichier en cours d'écriture — et l'obtiendrait sans erreur, tronqué. C'est à surveiller quand la durée du vidage s'allonge, pas le jour où la restauration échoue.

Les journaux vont dans un fichier daté par jour, purgé au-delà de six mois :

```
30 03 1 * * find /var/log/backup/ -name '*.log' -mtime +180 -delete
```

⚠ Ne pas mêler les journaux de sauvegarde à ceux d'autres tâches. Deux services de diffusion audio écrivaient autrefois dans ce même répertoire — jusqu'à 5,6 Mo pour une seule journée, noyant les rapports qu'on venait d'y mettre. Un journal qu'on n'ouvre plus parce qu'il est illisible ne sert à rien.

---

9. Le démon est un point d'appui unique, et silencieux

 Un incident réel, qui vaut d'être connu. Un `killall rsync` lancé avec les privilèges root sur l'hôte, pour interrompre une copie, a arrêté les deux démons du parc à la même seconde — celui de l'hôte et celui du conteneur.

Deux enseignements, et le second est le plus gênant.

⚠ Une commande visant un nom de processus, lancée en root sur l'hôte, traverse les conteneurs. Leurs processus ont leur propre espace de noms de PID, mais l'hôte les voit tous. Cela vaut pour `killall`, `pkill`, et tout ce qui cible par nom plutôt que par identifiant.

⚠ Rien n'a été relancé. `killall` envoie un SIGTERM, que le démon traite proprement : code de sortie 0, « Deactivated successfully ». Pour systemd, ce n'est pas une panne mais un arrêt normal. La chaîne de sauvegarde entière était hors service sur les deux machines, sans qu'aucun signal ne l'indique — la panne n'a été découverte qu'en testant autre chose.

La parade, à poser sur chaque machine qui sert des modules — `/etc/systemd/system/rsync.service.d/override.conf` :

```
[Service]
Restart=always
RestartSec=5
```

```
sudo systemctl daemon-reload
```

⚠ `Restart=on-failure` n'aurait rien changé : un SIGTERM produit une sortie propre, pas un échec. Seul `always` relance quel qu'en soit le motif. Un `systemctl stop` explicite reste respecté — systemd distingue l'arrêt voulu de la disparition.

⚠ Ne pas créer ce fichier avec `systemctl edit`. La commande ouvre un fichier ne contenant que des commentaires ; si l'on enregistre sans que systemd y détecte de contenu utile, il supprime le fichier avec un message discret. Le créer à la main est déterministe.

Vérifier que la surcharge est prise en compte :

```
systemctl cat rsync
```

Deux blocs doivent apparaître : l'unité, puis le chemin de la surcharge. Son absence signifie qu'elle n'existe pas.

Le test qui valide vraiment :

```
sudo killall rsync
```

```
sleep 8 ; systemctl is-active rsync
```

□□ Le `sleep` n’est pas de la prudence, il est nécessaire : sans lui on interroge pendant les cinq secondes de `RestartSec` et l’on croit à un échec.

---

10. Fragilités connues

Les écrire vaut mieux que les découvrir. Aucune n’est un défaut à corriger d’urgence ; toutes sont des arbitrages dont il faut connaître le prix.

10.1 □ Deux points de montage, un seul système de fichiers

Ce qui fut historiquement deux disques est aujourd’hui deux points de montage sur une seule partition. Écrire « sur le second » ne met donc rien à l’abri : ni d’une panne matérielle, ni d’une corruption, ni même d’un disque plein — l’espace est commun.

La conséquence est bornée par les disques externes, qui portent les deux séparément, sur deux supports distincts. La répartition qui n’existe plus en ligne existe hors ligne.

10.2 □ Trois modules écrivent dans le même répertoire avec `--delete`

Trois tirages de configuration visent la même destination, chacun avec `--delete`. Ça fonctionne parce que rsync protège de la suppression les fichiers exclus par filtre, et que chaque ligne exclut tout sauf son propre fichier.

⚠□ La sécurité repose entièrement sur ce comportement subtil. Le jour où quelqu’un ajoute `--delete-excluded` en croyant faire le ménage, les trois configurations s’effacent mutuellement.

10.3 □ La détection d’échec dépend d’une lecture

humaine

Les journaux existent, mais rien ne signale un échec. Le cas concret : si un vidage de base échoue, le script conserve correctement la version de la veille, et le serveur la tire sans broncher. Tout paraît normal, et la sauvegarde vieillit en silence jusqu'à ce que quelqu'un ouvre le journal.

Exposer le journal du conteneur par un module — pour qu'il soit lisible depuis le serveur, et sauvegardé comme le reste — améliore les choses sans les régler. La détection reste passive.

---

11. Savoir que ça fonctionne

11.1 Les journaux, chaque semaine

```
ls -lh /var/log/backup/
```

⚠ Un journal absent est plus inquiétant qu'un journal contenant une erreur. Le premier veut dire que rien ne s'est lancé ; le second, que quelque chose a essayé.

⚠ Ne jamais juger une sauvegarde à la date de ses fichiers. `rsync -a` préserve la date de la source : un fichier de 2022 dans une copie faite cette nuit est parfaitement normal. La question se pose au journal, pas au système de fichiers.

11.2 L'examen des modules, une fois l'an

Un module dont le chemin a disparu échoue chaque nuit sans rien dire d'autre qu'une ligne dans un journal que personne ne lit. Cette boucle interroge chaque module d'un démon et dit lequel répond :

```
P=/home/hostadmin/scripts/rsync/auto/rsync_pass
for m in $(rsync --list-only --password-file=$P test@192.168.0.6:: | awk '{print
```

```
$1}'); do
  printf '%-26s ' "$m"
  rsync --list-only --password-file=$P "test@192.168.0.6::$m/" >/dev/null 2>&1 &&
echo OK || echo ÉCHEC
done
```

À répéter en changeant l'adresse pour chaque machine. Dix secondes chacune.

Ce contrôle n'est pas théorique. Passé sur trente-quatre modules, il en a trouvé deux morts. L'un pointait vers un répertoire disparu quand un service avait déménagé ; l'autre vers un chemin de configuration qu'une application avait changé plusieurs versions plus tôt, si bien que la destination contenait une copie figée d'avant le déménagement. Tous deux échouaient en silence depuis des années.

11.3 L'essai de restauration, deux fois l'an

Voir la page Restauration. Une sauvegarde jamais restaurée est une hypothèse, pas une sauvegarde.

11.4 Et le cas le plus difficile à voir : ce qui réussit à ne rien faire

Une tâche qui échoue finit par se voir. Une tâche qui réussit à ne rien faire est invisible pour toujours.

Le cas rencontré ici : une tâche planifiée lançait un script toutes les cinq minutes, avec toute sa sortie redirigée vers `/dev/null`. Le programme qu'elle appelait avait été désinstallé, mais le script — déposé dans `/usr/local/bin`, donc n'appartenant à aucun paquet — avait survécu à la purge.

Et il n'échouait pas : il cherchait des fichiers de configuration, n'en trouvait aucun, et un garde-fou l'empêchait d'aller plus loin. Il posait un verrou, ne faisait rien, retirait le verrou, et sortait proprement. Son répertoire de configuration était vide depuis près de quatre ans — de l'ordre de 400 000 exécutions parfaitement réussies et parfaitement inutiles.

Ce que ça apprend, et qui ne figure dans aucun manuel :

- Une purge de paquet n'emporte pas ce qui vit dans `/usr/local/`, ni les tâches planifiées d'un compte de service, ni le compte lui-même.
- `2> /dev/null` sur une tâche planifiée rend toute panne future indétectable. Rediriger la sortie vers un journal, jamais vers le néant.
- L'inventaire à faire n'est donc pas seulement « qu'est-ce qui échoue », mais « qu'est-ce qui s'exécute encore, et pourquoi » :

```
sudo ls -l /var/spool/cron/crontabs/
```

```
ls -l /etc/cron.d/ /etc/cron.daily/
```

Pour chaque entrée, une seule question : *si je supprimais ça, qu'est-ce qui cesserait de fonctionner ?* Une réponse hésitante vaut enquête.

---

12. Où va la suite

Remettre les données en place — une base, un site, un conteneur, une machine entière : page Restauration.

Le côté serveur de sauvegarde — ses propres modules, ses scripts de tirage, sa crontab et le durcissement de son service : page de l'hôte.