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.

⚠ Trois lignes de ce document dépassent 84 caractères et s'enrouleront dans le PDF sans signe visible : les trois lignes de cron — la sauvegarde du serveur de musique, l'image de la clé d'amorçage et le relevé du tableau de bord. Cron n'accepte aucune 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 : l'inventaire des données à protéger et les machines qui les portent, 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, les douze scripts de tirage écrits en entier, les moyens de savoir que tout cela fonctionne encore, et le tableau de bord qui les relève chaque nuit, ses deux scripts compris. 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.

Quatre 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 ce filtre est pourtant écrit deux fois, dans le module et dans le script qui le tire, et qu'élargir l'un sans l'autre ne transporte rien de plus — sans

qu’aucune erreur ne le dise : même section. 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.

Ce qui est protégé

L’inventaire des données, avant tout mécanisme. C’est cette liste qui permet de repérer un oubli — pas la liste des scripts, qui ne peut par construction que décrire ce qui existe déjà.

Donnée	Où elle vit	Volume
Sites web — contenu réel : blog, photos, webdav	`web-srv` : `/var/www`, disque `sdb2` monté depuis l’hôte	~59 Go
Bases MySQL (6)	`web-srv` : `/var/lib/mysql`	~68 Mo en vidages
Agendas et carnets d’adresses (Radicale)	`web-srv` : `/var/lib/radicale/collections`	faible
Configurations de service	`/etc/` des trois machines : bind, samba, nfs, icecast2, ices2, rsyncd	faible
Crontabs	`/var/spool/cron/crontabs/` de nas-host et du conteneur web	faible
Scripts	`/home/hostadmin/scripts`, `/home/webadmin/scripts`, `/home/user/Scripts`	faible
Bibliothèque musicale — métadonnées	nas-host : MusicIP, MusicMagic, bliss	faible
Configurations d’applications du poste	poste-bureau : filezilla, puddletag, quodlibet	faible
Machines virtuelles VirtualBox	poste-bureau : `/home/user/VirtualBox_VMs`	important
Systèmes des conteneurs eux-mêmes	pool ZFS `local` de nas-host	~7,4 Go en archives
Bibliothèque audio	nas1	important

Donnée	Où elle vit	Volume
Photographies originales	nas1 et dépôt chiffré hors site	important

La surface réelle est plus petite que l’inventaire des sites. Neuf sites sont déclarés dans le panneau d’hébergement, six sont actifs, mais trois seulement hébergent des fichiers :

Site	Actif	Nature
`blog.example.com`	✓	contenu
`photos.example.com`	✓	contenu (Piwigo)
`dav.example.com`	✓	contenu (WebDAV)
`lt.example.com`	✓	proxy inverse seul → LanguageTool
`musique.example.com`	✓	proxy inverse seul → Navidrome
`routeur.example.com`	✓	proxy inverse seul → admin du routeur
`cumulus.example.com`	✗	désactivé
`ha.example.com`	✗	désactivé

✓ Un proxy inverse ne porte aucun fichier à sauvegarder, seulement une configuration Apache — que les modules de configuration emportent déjà. Confondre les deux catégories fait croire à six sites à protéger là où il y en a trois, et le blog est le seul dont la perte serait irremplaçable.

Les arbitrages d’importance, pour ne pas les redécouvrir :

- `photos.example.com` est une vitrine reconstructible. Les photographies originales sont sauvegardées hors site ; le site n’en est qu’une présentation.

- `webdav` est une expérimentation. Son contenu existe déjà sur le NAS et, en bonne partie, hors site. Ce n’est pas une donnée unique.

- La bibliothèque audio n’est pas encore déposée hors site — le ménage des fichiers n’est pas terminé. Elle est couverte par nas1 et par le disque externe chiffré.

- Délibérément non couverts : les vidéos, pour leur volume ; les systèmes d’exploitation des hôtes, qui se reconstruisent depuis ce wiki.

⚠ Un service retiré doit sortir de cet inventaire en même temps que de la crontab. Un serveur de statistiques autonome a vécu ici en doublon d’une extension du blog, avec sa base, son module et ses copies, longtemps après avoir cessé d’être utile. Tant qu’une ligne reste dans le tableau, elle donne à la sauvegarde une étendue qu’elle n’a plus.

Les machines et le stockage de destination

Machine	IP	Rôle
nas-host	`192.168.0.11` (eno2)	Hôte Incus et NAS. Porte nas1/nas2, Samba, NFS, Icecast2. Orchestre toutes les sauvegardes.
`web-srv`	`192.168.0.6`	Conteneur : Apache/ISPConfig, MariaDB, Radicale, BIND, proxy inverse
`pi-hole`	`192.168.0.5`	Conteneur : DNS
`squid-proxy`	`192.168.0.7`	Conteneur : mandataire
`navidrome`	`192.168.0.8`	Conteneur : diffusion musicale
`languagetool`	`192.168.0.10`	Conteneur : correction linguistique
poste-bureau	`192.168.0.106`	Poste de travail Linux Mint

⚠ La seconde carte réseau de l’hôte n’est pas disponible pour autre chose. `eno1` (`192.168.0.2`) porte le réseau macvlan des conteneurs, et c’est elle qui permet à l’hôte de joindre ses propres conteneurs — donc de les sauvegarder. La réaffecter couperait la chaîne entière sans toucher à un seul script.

Un seul disque, deux noms

Le volume de destination est un disque unique (`/dev/sda1`, ext4, 12,7 To) monté une seule fois, sur `/media/nas1-14TB`. Il contient deux répertoires, `nas1` et `nas2`, que deux montages liés exposent comme s’il s’agissait de deux disques :

Point de montage	Réalité
`/media/nas1-14TB`	`/dev/sda1`, ext4, 12,7 To — le montage réel
`/media/nas1`	montage lié de `/media/nas1-14TB/nas1`
`/media/nas2`	montage lié de `/media/nas1-14TB/nas2`

Les lignes correspondantes de `/etc/fstab` :

```
UUID=4788d777-…  /media/nas1-14TB  ext4  defaults,acl  0 2
/media/nas1-14TB/nas1  /media/nas1  none  bind
/media/nas1-14TB/nas2  /media/nas2  none  bind
```

Les anciennes lignes qui montaient `nas1` et `nas2` par UUID sont commentées : vestiges de l'époque où c'étaient deux disques.

 Ces deux noms viennent d'une époque révolue, et ils trompent encore. Écrire « sur nas2 » ne met rien à l'abri de ce qui menace nas1 — voir *Fragilités connues*, où la conséquence est développée. Le nommage a survécu au matériel qui le justifiait ; c'est la forme la plus discrète de fausse sécurité, parce qu'elle ne repose sur aucune erreur technique.

Note d'entretien — la réserve ext4. `mkfs.ext4` réserve 5 % du volume à root, soit 652 Gio sur 13 To. Cette marge est justifiée sur une partition système, où elle permet à root de réparer un disque plein ; elle est sans objet sur un volume de pures données. Ramenée à 1 % :

```
sudo tune2fs -m 1 /dev/sda1
```

Cela rend 522 Gio tout en laissant 130 Gio à l'allocateur, ce qui évite la fragmentation. ⚠ Ne jamais appliquer cela à `/`.

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_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
`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.

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	Copie du mensuel, plus l’image de la clé de démarrage	Continu, versions conservées 30 jours	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 au-delà de trente jours : les versions antérieures expirent, et une compromission du compte les atteindrait aussi
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

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 que trente jours — l'erreur, la corruption, le rançongiciel —, la coupure physique des disques le protège sans limite de durée.

- Ce que les disques n'ont pas — la fraîcheur —, le nuage l'a.

⚠ Une fenêtre de conservation ne vaut que ce que vaut la fréquence

d’inspection. Trente jours de versions antérieures ne protègent que les erreurs remarquées dans le mois. Une archive à laquelle on ne touche jamais est justement celle dont on découvrira le problème trop tard — d’où le niveau débranché, qui n’a pas d’échéance.

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. Une réplication continue 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.

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.

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.*

Mais le filtre est écrit DEUX fois

Le filtre du module dit ce qu'on peut demander. Le script de tirage, lui, garde le sien, qui dit ce qu'on demande. Les deux existent, et rien ne signale leur désaccord.

Le cas se pose dès qu'on ajoute un fichier à ce qui est sauvegardé — par exemple `/etc/apache2/.htpasswd-routeur`, qui protège un mandataire. Élargir le module :

```
exclude = lost+found/ **
include = rsyncd.conf apache2/ .htpasswd-routeur
```

⚠ Le répertoire doit être nommé en plus du fichier : le `**` exclut tout, donc rsync ne descend pas dans `apache2/` si on ne l'ouvre pas.

Le démon servait alors bien le fichier. Et la sauvegarde ne le recevait toujours pas : le script portait encore `--include='rsyncd.conf' --exclude='**'`, et jetait ce que le démon lui offrait. Le tirage se terminait à `échecs : 0`.

Les deux lignes doivent être élargies ensemble — celle du module, et celle du script :

```
--exclude='lost+found/' --include='rsyncd.conf' --include='apache2/' \
  --include='apache2/.htpasswd-routeur' --exclude='**'
```

Dans un script, l'ordre d'écriture fait loi — contrairement au fichier du démon, qui assemble ses règles par type. Les `--include` doivent précéder le `--exclude='**'`, sinon ils ne sont jamais atteints.

Et la démonstration que le serveur commande

Le même jour, une correction trop large a fait réclamer `apache2/` par le script qui tire la configuration de nas-host — machine dont le module, lui, est resté filtré à `rsyncd.conf`.

Rien n'est arrivé. Le répertoire de destination ne contient toujours que `rsyncd.conf`. Le client a demandé, le démon n'a pas servi. C'est la preuve, faite par accident, que c'est bien le filtre du module qui borne — et non celui du script.

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.

Ce module a son propre article. Ce qui l’emploie — la tâche nocturne qui pousse les photos depuis le NAS, son garde-fou, et la galerie qui les reçoit — est décrit dans Piwigo. Les neuf lignes ci-dessus sont reproduites ici parce qu’elles portent tout le propos de cette section ; la déclaration qui fait foi reste celle du fichier complet.

⚠ Une tâche qui pousse avec `--delete` doit vérifier sa source avant d’agir. Un répertoire source démonté se présente comme un répertoire vide, et la synchronisation propage cette vacuité à la destination — dans le cas présent, une galerie entière. C’est le seul endroit du parc où ce risque existe, précisément parce que c’est le seul module en écriture.

Les modules du conteneur web

Le conteneur expose treize modules. Leur déclaration complète — `/etc/rsyncd.conf` en entier, prêt à recopier — figure dans Serveur web — Apache multi-PHP et ISPConfig, section *Le démon rsync*. C'est là qu'on en a besoin : pendant qu'on rebâtit cette machine. Un fichier de configuration n'est écrit en entier que dans un seul article ; les autres y renvoient, faute de quoi deux copies finissent par diverger et c'est toujours celle qu'on ne relit pas qui est fausse.

Ce qui compte ici est ce que chaque module expose, pour savoir ce qui est sauvegardé et ce qui ne l'est pas :

Module	Ce qu'il expose
`blog`	les fichiers du site WordPress
`webdav`	le partage WebDAV de `dav.example.com`, répertoire parent compris — donc son fichier de mots de passe `.htdigest`
`photos`	les fichiers du site Piwigo, galeries comprises
`gallerie`	en écriture — le téléversement des photos, pas une sauvegarde
`cumulus`	les fichiers d'un site désactivé, conservés
`mysql`	les vidages de bases produits par le conteneur
`scripts_NS2`	les scripts du conteneur
`radicale_users`	configuration et comptes du serveur de calendriers
`radicale_data`	les calendriers et carnets d'adresses eux-mêmes
`rsync_NS2`	`/etc/`, filtré à `rsyncd.conf` et `apache2/.htpasswd-routeur`
`crontab_NS2`	la crontab du compte applicatif, filtrée
`www-full`	`/var/www/` en entier, sans filtre
`log_NS2`	le journal des tâches nocturnes du conteneur

Quatre remarques sur ces modules.

⚠ `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.

Les modules que nas-host expose pour lui-même — sa propre configuration, ses scripts — sont décrits dans la page de l'hôte, qui en a besoin pour se reconstruire. Les scripts qui tirent tout cela, en revanche, sont sur cette page : section *Les scripts de tirage*.

Les modules du poste de travail

Le poste sert sept modules, tous en lecture seule, et c'est le seul démon du parc dont la déclaration ne figure dans aucune page de machine.

Module	Chemin servi	Tiré par
`scripts_Beelink`	`/home/user/Scripts`	`backup_scripts_poste-bureau.sh`
`rsync_Beelink`	`/etc/`, filtré côté serveur sur `rsyncd.conf`	`backup_config_poste-bureau.sh`
`filezilla_Beelink`	`/home/user/.config/filezilla`	`backup_config_poste-bureau.sh`
`puddletag.local_Beelink`	`/home/user/.local/share`, filtré	`backup_config_poste-bureau.sh`
`puddletag.config_Beelink`	`/home/user/.config`, filtré	`backup_config_poste-bureau.sh`
`quodlibet.config_Beelink`	`/home/user/.config`, filtré	`backup_config_poste-bureau.sh`
`virtualbox_Beelink`	`/home/user/VirtualBox_VMs`	`backup_virtualbox_poste-bureau.sh`

⚠ Quatre de ces modules servent le même répertoire parent, `/home/user/.config`, chacun filtré sur ce qui le concerne. C'est la forme la plus économique — un seul chemin, plusieurs vues — et c'est aussi celle où élargir un filtre de trop ouvre le répertoire entier. Le filtre est ce qui tient le module, pas le

chemin.

 Le chemin de `filezilla_Beelink` a été faux pendant des années, pointant vers `~/.filezilla` alors que le logiciel avait déménagé sa configuration vers `~/.config/filezilla` plusieurs versions plus tôt. Le module échouait à chaque nuit, la destination gardait une copie figée d'avant le déménagement, et rien ne le disait. C'est l'examen annuel des modules qui l'a trouvé — voir *Savoir que ça fonctionne*.

Le module du conteneur Pi-hole

Le filtre DNS n'expose qu'un seul module, et il ne sert pas des fichiers de configuration vivants : il sert une archive que le conteneur fabrique lui-même chaque nuit, dix minutes avant d'être tiré.

Module	Ce qu'il sert
`config_pihole`	les archives de configuration déposées dans `/var/backups/pihole`

 Pourquoi une archive plutôt que les fichiers eux-mêmes. La configuration du filtre tient dans un fichier de réglages et une base SQLite en cours d'usage. Copier une base pendant qu'elle est ouverte donne une copie dont rien ne garantit la cohérence. L'outil d'export livré avec le filtre produit une archive cohérente avec elle-même, et ne retient que la part personnelle — ni l'historique des requêtes, ni les dizaines de milliers de domaines téléchargés, qui se régénèrent d'une commande. C'est le raisonnement qui fait préférer un vidage de base à une copie de `/var/lib/mysql`.

La fabrication est déclenchée depuis l'hôte, pas par une tâche interne au conteneur. La raison est le mode de défaillance : si le conteneur ne démarre pas, une tâche interne n'écrit aucun rapport — le journal ne contient pas une erreur, il ne contient rien, et l'absence est ce qui se repère le plus difficilement. Lancée de l'extérieur, la tâche échoue bruyamment dans le journal de l'hôte à l'heure prévue.

⚠ Ce que l'archive ne contient pas. Elle ne connaît que ce que le filtre

connaît. La configuration du relais qui chiffre la sortie DNS lui échappe entièrement ; le script la copie donc à côté, dans un sous-dossier que le même module emporte. Toute pièce installée sur cette machine hors du filtre appelle le même geste, sans quoi la sauvegarde couvre moins que ce qu'elle paraît couvrir.

Le script de fabrication, le démon du conteneur et sa surcharge sont écrits au complet dans Pi-hole : c'est là qu'on rebâtit cette machine.

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`.

Le script complet — `/home/webadmin/scripts/backup_mysql.sh` et sa tâche planifiée — figure dans Serveur web — Apache multi-PHP et ISPConfig, section *Le vidage des bases*, pour la même raison que ci-dessus. Ce qui suit est ce qu'il faut comprendre de son fonctionnement pour juger si la sauvegarde tient réellement.

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.

⚠ Ne jamais lancer ce script en root. Ses identifiants sont dans le `~/.my.cnf` du compte du conteneur. Lancé par root, il écrirait des vidages appartenant à root dans le répertoire de ce compte — que l'exécution suivante, redevenue normale, ne pourrait plus remplacer. La sauvegarde se figerait alors sur la version produite ce jour-là, avec des succès sincères à chaque nuit suivante.

 C'est le même motif que partout ailleurs sur cette page : le dégât n'est pas l'échec, c'est le succès qui ne fait plus rien. Un `chown` corrige l'accident ; encore faut-il savoir qu'il a eu lieu.

Deux protections de forme complètent les décisions ci-dessus : un compteur d'échecs par base avec `exit $ERREURS`, pour qu'un échec sur une base parmi six ne se perde pas dans le succès des autres ; et les marqueurs datés de début et de fin, qui donnent l'heure et la durée — deux secondes en régime normal, ce qui rend toute dérive immédiatement lisible.

Les scripts de tirage

Ils vivent tous dans `/home/hostadmin/scripts/rsync/auto/` sur le serveur de sauvegarde, et sont lancés par la crontab du compte `hostadmin` — reproduite dans la page de l'hôte. Une seule exception : l'image de la clé d'amorçage, qui exige root et part d'un fichier de `/etc/cron.d/`.

Les lignes longues sont coupées ici par des continuations `\`, et la fonction `r` est écrite sur trois lignes plutôt qu'une. Ce sont les deux seules libertés prises avec le texte réel : sur la machine, plusieurs lignes font deux cents caractères. Rien d'autre ne diffère — mêmes options, mêmes chemins écrits en entier, dans le même ordre. Les formes sont rigoureusement équivalentes ; celles-ci se recopient sans se casser dans un PDF ou une fenêtre étroite.

Le squelette commun

Tous suivent la même forme, et il vaut mieux la comprendre une fois que dix :

```
#!/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)"
echo ""
echo "nom_du_module"
r <options> <source> <destination>
echo ""

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

Ce que chaque ligne apporte :

- La ligne 2 charge le garde-fou de destination — le veto décrit juste après.

- La fonction `r` enveloppe chaque `rsync` : elle signale l'échec avec les deux derniers arguments, c'est-à-dire la source et la destination, et incrémente un compteur. Sans elle, le code de sortie du script serait celui de sa dernière commande, et un échec au milieu passerait inaperçu.

- L'étiquette `echo` avant chaque tirage nomme le module dans le journal commun, où quatre scripts écrivent la même nuit.

- Les deux marqueurs `===` encadrent l'exécution. ⚠ C'est sur eux qu'on cherche, jamais sur le mot « erreur » : un journal de rsync est plein de noms de fichiers comme `error.log.3.gz` ou `awstats…errors404.html`, qu'une recherche par mot confond avec des messages.

```
grep -E "^===" /var/log/backup/$(date +%F).log
```

- `exit $ERREURS` : le code de sortie est le nombre d'échecs.

Le garde-fou de destination — un fichier, pas dix copies

 Une destination ne « manque » jamais devant rsync. `/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 soixante gigaoctets sans se plaindre — remplissant `/` et fabriquant une sauvegarde qui n'est pas sur le disque de sauvegarde. Un chemin mal nommé, lui, est simplement créé.

Dans les deux cas l’écriture réussit. C’est pourquoi le contrôle doit avoir lieu avant d’écrire, et pourquoi il vaut veto.

`/home/hostadmin/scripts/rsync/auto/verifier_montages.sh`, au complet :

```
#!/bin/bash
# Garde-fou commun aux tâches de sauvegarde de nas-host.
# Se charge en tête de chaque script par la ligne :
#     . /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
#
# Motif : /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.
#
# Un « exit » dans un fichier chargé par « . » arrête le script appelant :
# c'est ce qui permet à ce contrôle de valoir veto, et non simple avis.

for MONTAGE in /media/nas1 /media/nas2; do
    if ! mountpoint -q "$MONTAGE"; then
        echo "ARRÊT : $MONTAGE n'est pas monté — aucune sauvegarde."
        exit 1
    fi
done
```

La ligne à poser en tête de chaque script, immédiatement après le shebang :

```
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
```

Le fichier est en `644`, volontairement non exécutable : il n’a pas vocation à être lancé seul, et un `chmod +x` inviterait à le faire.

 Pourquoi un fichier commun plutôt que dix copies : le jour où un troisième disque s’ajoute, il y a une place à corriger, et aucun script ne peut « oublier » la correction.

⚠ Il exige que les DEUX disques soient montés, même pour les neuf scripts qui

n’écrivent que sur `nas1`. C’est plus strict que nécessaire, et c’est délibéré : une sauvegarde qui ne tourne pas se voit dans le journal du lendemain ; une sauvegarde écrite au mauvais endroit ne se voit jamais.

Le veto se vérifie, il ne se suppose pas. Le comportement d’`exit` dans un fichier chargé par `.` a été éprouvé dans les deux sens :

```
V=/home/hostadmin/scripts/rsync/auto/verifier_montages.sh
bash -c ". $V ; echo 'PASSE — la sauvegarde aurait lieu'"
mkdir -p /tmp/pas-un-montage
sed 's|/media/nas1|/tmp/pas-un-montage|' $V > /tmp/gf-test.sh
bash -c ". /tmp/gf-test.sh ; echo 'PASSE — ne doit PAS apparaitre'"
echo "code de sortie : $?"
```

Disques montés : `PASSE`. Disque absent simulé : le message `ARRÊT`, aucun `PASSE`, et code de sortie `1`.

Le garde-fou de source — seulement là où il a un sens

Le garde-fou ci-dessus protège la destination. Un seul script protège aussi sa source, et c’est celui qui tire les sites : leur source est distante, vue à travers le démon du conteneur, et `mountpoint` ne peut rien dire d’un montage qui est sur l’autre machine.

Le seuil a donc été mesuré avant d’être écrit — `rsync --list-only` rend toujours une ligne pour le répertoire lui-même :

```
2 lignes ou plus = le module a du contenu
1 ligne          = module vide  (/var/www non monté dans le conteneur)
0 ligne          = module injoignable  (conteneur arrêté, mot de passe refusé)
```

 Le même test attrape deux pannes de natures différentes, et toutes deux mènent au même désastre avec `--delete`.

Un module refusé compte comme un échec, pas comme un succès silencieux : le script sort en `1` et sa ligne de bilan le dit. Un tirage sauté qui ne se voit

pas serait exactement la panne muette que tout ce dispositif traque.

L’inventaire

Ce tableau sert à s’y retrouver ; les scripts eux-mêmes sont écrits en entier ci-dessous, et c’est de là qu’on recopie.

Script	Ce qu’il tire	Quand
`backup_mysql-ns2.sh`	les vidages de bases du conteneur web	00 h 15
`backup_web-ns2.sh`	les fichiers des sites, et `/var/www` en entier	00 h 20
`backup_pihole.sh`	l’archive de configuration du filtre DNS — s’exécute dans le conteneur	00 h 35
`backup_scripts.sh`	les scripts du conteneur web et ceux de nas-host	00 h 40
`backup_config.sh`	les configurations des trois machines, les calendriers, le journal	00 h 45
`backup_scripts_poste-bureau.sh`	les scripts du poste de travail	00 h 50
`backup_config_poste-bureau.sh`	les configurations d’applications du poste	00 h 55
`backup_logiciels.sh`	les configurations d’applications de nas-host	01 h 00
`backup_lyrion.sh`	la configuration du serveur de musique — copie locale, par root	01 h 05
`upload_photos_…sh`	pousse les photos vers la galerie — voir Piwigo	01 h 30
`backup_virtualbox_poste-bureau.sh`	les machines virtuelles du poste	le 1er, 02 h 00
`backup_incus_export.sh`	l’export mensuel des conteneurs	le 1er, 03 h 00
`backup_cle_amorcage.sh`	l’image de la clé d’amorçage — section suivante	04 h 00, par root

⚠ `upload_photos_photos-webadmin-ca.sh` est le seul qui POUSSE, et le seul qui ne charge pas le garde-fou commun — il n’écrit sur aucun des deux disques de sauvegarde, il lit `nas1` et écrit dans la galerie. Il porte donc son propre contrôle, en ligne : `mountpoint -q /media/nas1` et un test de source non vide. Son texte complet est dans Piwigo, qui décrit ce qu’il alimente.

Les scripts, au complet

`backup_mysql-ns2.sh` :

```
#!/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)"
echo ""
r -avz --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::mysql/ \
  /media/nas1/Backups/web-srv_MySQL_db

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

⚠ Il ne fabrique rien : il tire. Le vidage est produit dans le conteneur à minuit, et ce script vient le chercher un quart d’heure plus tard. Le lancer sans que l’autre ait tourné ne teste que la moitié de la chaîne.

`backup_web-ns2.sh` — celui qui porte les deux garde-fous :

```
#!/bin/bash
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
# Tire les sites de web-srv (192.168.0.6) vers les disques de sauvegarde.
# Chaque tirage est un miroir : --delete propage les suppressions.

P=/home/hostadmin/scripts/rsync/auto/rsync_pass
SRV=test@192.168.0.6
ERREURS=0

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

```
# Garde-fou propre à ce script — un module vide ou injoignable n'est
# jamais tiré. rsync --list-only rend toujours une ligne pour le
# répertoire lui-même :
#   2 lignes ou plus = le module a du contenu
#   1 ligne          = module vide (/var/www non monté dans le conteneur)
#   0 ligne          = module injoignable (conteneur arrêté, mot de passe)
# Dans les deux derniers cas, --delete viderait la sauvegarde.
tirer() {
    MODULE="$1"; shift
    N=$(rsync --list-only --password-file "$P" \
              "$SRV::$MODULE/" 2>/dev/null | wc -l)
    if [ "$N" -lt 2 ]; then
        echo "ARRÊT : module $MODULE vide ou injoignable — non tiré."
        ERREURS=$((ERREURS+1))
        return
    fi
    rsync "$@" || {
        echo "ÉCHEC : ${*: -2:1} -> ${*: -1}"
        ERREURS=$((ERREURS+1))
    }
}

tirer photos \
  -avz --stats --delete --exclude='quota.*' --password-file "$P" \
  "$SRV::photos/" /media/nas1/Backups/web-srv_www/photos

tirer blog \
  -avz --stats --delete --exclude='quota.*' --password-file "$P" \
  "$SRV::blog/" /media/nas1/Backups/web-srv_www/blog

tirer webdav \
  -avz --stats --delete --exclude='quota.*' --password-file "$P" \
  "$SRV::webdav/" /media/nas1/Backups/web-srv_www/webdav

tirer www-full \
  -avz --stats --delete --exclude='quota.*' --exclude='lost+found/' \
  --password-file "$P" \
  "$SRV::www-full/" /media/nas2/Backups/web-srv_www-full_path

# Modules désactivés, conservés pour mémoire :
#   cumulus  — ancien site
#   matomo   — retiré le 7 septembre 2026, site supprimé
```

```
#   web      — ancienne adresse 192.168.0.100
#   webmail  — jamais réactivé

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

Le commentaire final n’est pas décoratif. Quatre modules ont existé et n’existent plus ; sans cette liste, chacun redeviendrait un jour une question — « pourquoi celui-là n’est-il pas sauvegardé ? ».

`backup_scripts.sh` :

```
#!/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)"
echo ""
echo "scripts_NS2"
r -avz --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::scripts_NS2/ \
  /media/nas1/Backups/Scripts/web-srv
echo ""
echo "scripts_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --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
```

 C’est ce script qui met les autres à l’abri. Les douze scripts décrits sur cette page vivent dans `/home/hostadmin/scripts/`, et c’est le module `scripts_nas-`

`host` qui les sauvegarde. Une reconstruction les récupère donc du disque de sauvegarde plutôt que de cette page — laquelle sert quand le disque, lui aussi, a disparu.

`backup_config.sh` :

```
#!/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)"
echo ""
echo "Crontab_NS2"
r -avz --stats --delete --exclude='lost+found/' \
  --include='webadmin' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::crontab_NS2/ \
  /media/nas1/Backups/Config/web-srv/crontab
echo ""
echo "Crontab_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='hostadmin' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::crontab_nas-host/ \
  /media/nas1/Backups/Config/nas-host/crontab
echo ""
echo "rsync_NS2"
r -avz --stats --delete --exclude='lost+found/' \
  --include='rsyncd.conf' --include='apache2/' \
  --include='apache2/.htpasswd-routeur' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::rsync_NS2/ \
  /media/nas1/Backups/Config/web-srv/etc
echo ""
echo "rsync_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='rsyncd.conf' --exclude='**' \
```

```
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::rsync_nas-host/ \
  /media/nas1/Backups/Config/nas-host/etc
echo ""
echo "samba_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='smb.conf' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::samba_nas-host/ \
  /media/nas1/Backups/Config/nas-host/
echo ""
echo "nfs_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='exports' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::nfs_nas-host/ \
  /media/nas1/Backups/Config/nas-host/
echo ""
echo "Radical_NS2"
r -avz --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::radicale_users/ \
  /media/nas1/Backups/Config/web-srv/Radicale
echo ""
echo "Radical_data_NS2"
r -avz --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::radicale_data/ \
  "/media/nas1/Backups/Carnets d'adresses/Radical_server_data"
echo ""
echo "icecast2_config_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='icecast.xml' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::icecast2_config_nas-host/ \
  /media/nas1/Backups/Config/nas-host/icecast2
echo ""
echo "ices2_config_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='ices2_*_config.xml' --exclude='**' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::ices2_config_nas-host/ \
```

```
  /media/nas1/Backups/Config/nas-host/ices2
echo ""
echo "ices2_metadata_nas-host"
r -avz --stats --delete --exclude='lost+found/' \
  --include='metadata_*' --exclude='*.pid' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::ices2_metadata_nas-host/ \
  /media/nas1/Backups/Config/nas-host/ices2_metadata
echo ""
echo "config_pihole"
r -avz --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.5::config_pihole/ \
  /media/nas1/Backups/Config/pi-hole
echo ""
echo "log_NS2"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.6::log_NS2/ \
  /media/nas1/Backups/Config/web-srv/log

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

⚠ Deux de ces tirages écrivent dans le même répertoire — `samba_nas-host` et `nfs_nas-host` visent `Config/nas-host/`, chacun filtré à son seul fichier. Voir *Fragilités connues* : ce qui rend ce partage possible est aussi ce qui y laisse un fichier mort quand on retire un tirage.

Le chemin des carnets d’adresses porte une espace et doit donc rester entre guillemets. Une recherche par expression régulière qui s’arrête au premier blanc le tronque — et fait croire à une destination manquante.

`backup_scripts_poste-bureau.sh` :

```
#!/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)"
echo ""
echo "scripts_Beelink"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::scripts_Beelink/ \
  /media/nas1/Backups/Scripts/Beelink/Scripts
echo ""

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

`backup_config_poste-bureau.sh` :

```
#!/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)"
echo ""
echo "rsync_Beelink"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::rsync_Beelink/ \
  /media/nas1/Backups/Config/Beelink/etc
echo ""
echo "filezilla_Beelink"
r -avz --recursive --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::filezilla_Beelink/ \
  /media/nas1/Backups/Configuration_logiciels/[.]filezilla/
echo ""
```

```
echo "puddletag.local_Beelink"
r -avz --recursive --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::puddletag.local_Beelink/ \
  /media/nas1/Backups/Configuration_logiciels/puddletag/[.]local/share/
echo ""
echo "puddletag.config_Beelink"
r -avz --recursive --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::puddletag.config_Beelink/ \
  /media/nas1/Backups/Configuration_logiciels/puddletag/[.]config/
echo ""
echo "quodlibet.config_Beelink"
r -avz --recursive --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::quodlibet.config_Beelink/ \
  /media/nas1/Backups/Configuration_logiciels/quodlibet/[.]config/
echo ""

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

`backup_logiciels.sh` :

```
#!/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)"
echo ""
echo "musicip_nas-host"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::musicip_nas-host/ \
  /media/nas1/Backups/Configuration_logiciels/musicip/MusicIP/
echo ""
```

```
echo "MusicMagic_nas-host"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::MusicMagic_nas-host/ \
  /media/nas1/Backups/Configuration_logiciels/musicip/[.]MusicMagic/
echo ""
echo "bliss_nas-host"
r -avz --stats --delete \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.11::bliss_nas-host/ \
  /media/nas1/Backups/Configuration_logiciels/[.]bliss/

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

`backup_lyrion.sh` — le seul qui ne tire rien : le serveur de musique vit sur l'hôte, la copie est donc locale et aucun module n'intervient.

```
#!/bin/bash
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
# Sauvegarde de la configuration de Lyrion Music Server.
#
# Lyrion tourne directement sur nas-host : la copie est locale, aucun module
# rsync n'est nécessaire.
#
# DOIT TOURNER EN ROOT, depuis /etc/cron.d/. Les répertoires internes de
# certaines extensions sont en 0701 : le propriétaire a tout, les autres
# peuvent traverser mais PAS lister. rsync doit énumérer avant de copier,
# donc le compte hostadmin n'en verrait rien — et la sauvegarde contiendrait une
# extension amputée de son binaire et de ses bibliothèques. Restaurée, une
# telle extension se charge et échoue, ce qui est pire qu'une extension
# franchement absente.
#
# Ces mêmes modes arrêtent tout lecteur venu d'ailleurs, y compris la
# réplication hors site, qui lit par le partage NFS. L'export écrase
# l'identité des clients vers uid 1000 / gid 150 : les copies sont donc
# déposées avec la lecture ouverte au groupe 150, et rattachées à ce groupe.
# « --chmod » et « --chown » ne touchent QUE la copie ; la source garde ses
# modes, et le serveur de musique n'en sait rien.
#
```

```
# Deux périmètres, et un seul critère : ce qui ne se reconstruit pas.
#   prefs/                  réglages, lecteurs, favoris, historique d'écoute
#   cache/InstalledPlugins/ le code des extensions — un dépôt amont peut
#                           disparaître, et Lyrion efface sa propre liste
#                           d'extensions au démarrage s'il trouve le
#                           répertoire vide.
# Le reste de cache/ est exclu : une réanalyse le refabrique vraiment.
#
# persist.db est une base SQLite que le serveur tient ouverte en permanence.
# Ses transactions récentes vivent dans le fichier -wal et ne sont pas encore
# dans le .db : une copie du seul .db peut être incohérente. On demande donc
# la copie à SQLite lui-même — même principe que le vidage MySQL.
#
# Elle pèse 71 Mo et se réécrit EN ENTIER chaque nuit : rsync n'y peut rien,
# c'est un fichier neuf à chaque passage. Les autres bases de prefs/ ne sont
# pas traitées ainsi — elles appartiennent à des extensions qui ne les
# tiennent pas ouvertes en permanence.

set -u

PREFS=/var/lib/squeezeboxserver/prefs
PLUGINS=/var/lib/squeezeboxserver/cache/InstalledPlugins
DEST=/media/nas1/Backups/Configuration_logiciels/lyrion
GROUPE=150
ERREURS=0

# Sans ce refus, un lancement à la main sous hostadmin copierait tout SAUF les
# répertoires en 0701, en signalant des erreurs qu'on prendrait volontiers
# pour un incident passager.
if [ "$(id -u)" -ne 0 ]; then
    echo "REFUS : doit tourner en root — voir /etc/cron.d/backup-lyrion"
    exit 1
fi

r() {
    rsync "$@" || { echo "ÉCHEC : ${*: -2:1} -> ${*: -1}";
ERREURS=$((ERREURS+1)); }
}

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

if [ ! -d "$PREFS" ]; then
```

```
    echo "ÉCHEC : $PREFS introuvable"
    echo "=== $(basename "$0") — fin $(date -Is) — échecs : 1"
    exit 1
fi

mkdir -p "$DEST"

# 1. Copie cohérente de la base des écoutes.
#    Écriture sous un nom temporaire, renommée seulement en cas de succès :
#    une copie tronquée ne doit jamais porter le nom d'une copie valide.
#    SQLite écrit sous root : le groupe et le mode se posent après coup,
#    pour que ce fichier suive la même règle que ceux de rsync.
if command -v sqlite3 >/dev/null; then
    if sqlite3 "$PREFS/persist.db" ".backup '$DEST/persist.db.tmp'"; then
        mv "$DEST/persist.db.tmp" "$DEST/persist.db"
        chgrp "$GROUPE" "$DEST/persist.db"
        chmod g+r "$DEST/persist.db"
        echo "OK     persist.db  ($(du -h "$DEST/persist.db" | cut -f1))"
    else
        rm -f "$DEST/persist.db.tmp"
        echo "ÉCHEC : copie SQLite — la copie précédente est conservée"
        ERREURS=$((ERREURS+1))
    fi
else
    echo "ÉCHEC : sqlite3 absent — installer le paquet sqlite3"
    ERREURS=$((ERREURS+1))
fi

# 2. Le reste de prefs/, sans les fichiers vivants de la base.
r -av --delete \
  --chmod=Dg+rX,Fg+r --chown=":$GROUPE" \
  --exclude='persist.db' \
  --exclude='persist.db-wal' \
  --exclude='persist.db-shm' \
  "$PREFS/" "$DEST/prefs/"

# 3. Le code des extensions.
#    Un répertoire disparu est le symptôme même de l'incident que cette
#    sauvegarde existe pour couvrir : il compte comme un échec.
if [ -d "$PLUGINS" ]; then
    r -av --delete \
      --chmod=Dg+rX,Fg+r --chown=":$GROUPE" \
```

```
      "$PLUGINS/" "$DEST/InstalledPlugins/"
else
    echo "ÉCHEC : $PLUGINS introuvable"
    ERREURS=$((ERREURS+1))
fi

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

Sa tâche planifiée ne vit pas dans la crontab mais dans `/etc/cron.d/backup-lyrion`, parce qu'elle tourne sous root :

```
# Sauvegarde de la configuration de Lyrion Music Server.
# Tourne en root : les répertoires internes de certaines extensions sont en
# 0701, et le compte hostadmin ne peut pas les énumérer.
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
MAILTO=""

05 01 * * * root /home/hostadmin/scripts/rsync/auto/backup_lyrion.sh >>
/var/log/backup/$(date +\%F).log 2>&1
```

```
sudo chown root:root /etc/cron.d/backup-lyrion
sudo chmod 644 /etc/cron.d/backup-lyrion
```

01 h 05 tient dans le creux entre la dernière tâche de 01 h 00 et le téléversement des photos de 01 h 30 — et bien après 00 h 15, ce qui respecte la contrainte de la section *Ordonnancement* : le journal du jour doit être créé par une tâche ordinaire, pas par une tâche root.

 Le mot « cache » dans le second chemin est un piège, et c'est la décision qui porte tout ce script. L'intuition veut qu'une extension perdue se retélécharge — ce qui suppose trois choses simultanément vraies : que le dépôt d'origine existe encore, que son adresse soit connue, et que la liste des extensions installées ait survécu. Les trois échouent indépendamment, et deux d'entre elles ont déjà échoué ensemble sur cette machine : un dépôt tiers a disparu, et

le serveur réécrit sa propre liste au démarrage — trouvant le répertoire vide, il y inscrit une liste vide et efface ainsi la mémoire de ce qui était installé.

 La distinction utile n'est donc pas « réglages contre cache », mais « ce qui se reconstruit contre ce qui ne se reconstruit pas ». Le reste de `cache/` — pochettes, bibliothèque analysée, plusieurs gigaoctets — est vraiment refabriqué par une réanalyse, et reste dehors.

⚠ Les droits `0701` ne sont pas une curiosité : ils décident de l'utilisateur du script, et de la lisibilité de la copie. Ce que root peut énumérer, un lecteur venu du partage ne le peut pas : sans le `--chmod` et le `--chown` de ce script, la réplication hors site s'arrête sur ces répertoires, et rien ne le signale hors de son propre message d'erreur — la sauvegarde locale, elle, se déclare réussie. Mesuré sous `hostadmin`, le répertoire des extensions annonçait 121 Mo ; copié par root, il en fait 173. Les 52 Mo d'écart sont exactement ce que `hostadmin` ne pouvait pas énumérer. Le tirage aurait signalé ses erreurs — le compteur fait son travail —, mais la sauvegarde aurait contenu un tiers de code en moins.

⚠ À la restauration, les fichiers `-wal` et `-shm` d'une ancienne session doivent être retirés avant de redémarrer le serveur : laissés à côté d'une base restaurée, SQLite tenterait de leur appliquer des transactions qui ne correspondent plus à elle. La procédure complète est dans Restauration.

`backup_virtualbox_poste-bureau.sh` :

```
#!/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)"
echo ""
```

```
echo "virtualbox_Beelink"
r -av --stats --delete --exclude='lost+found/' \
  --password-file /home/hostadmin/scripts/rsync/auto/rsync_pass \
  test@192.168.0.106::virtualbox_Beelink/ \
  /media/nas2/Backups/virtualbox_dirty/Beelink
echo ""

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

`-av` sans `z` ici, contrairement aux autres : les images de disque des machines virtuelles sont déjà compressées, et la compression au vol coûterait du temps de calcul pour rien.

`backup_incus_export.sh` — le seul qui ne tire rien : il fabrique, sur place :

```
#!/bin/bash
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
# Export mensuel de tous les conteneurs Incus.
# Ne purge l'ancien que si TOUS les exports ont réussi.

DEST="/media/nas1/Backups/incus-exports"
GARDE=2
REP="$DEST/$(date +%Y-%m)"

echo "===== Export Incus — $(date -Is)"

# Garde-fou : une liste vide ne veut pas dire « aucun conteneur »,
# elle veut dire « Incus n'a pas répondu ». Sans ce contrôle, la boucle
# ne tourne pas, le compteur d'erreurs reste à zéro, et le script
# annonce un succès complet avec zéro conteneur exporté.
CONTENEURS=$(incus list --format csv -c n)
if [ -z "$CONTENEURS" ]; then
    echo "ÉCHEC : aucun conteneur listé — Incus répond-il ?"
    echo "===== Terminé — $(date -Is)"
    exit 1
fi

mkdir -p "$REP" || { echo "Impossible de créer $REP"; exit 1; }
```

```
ERREURS=0
for c in $CONTENEURS; do
    echo "--- $c"
    if incus export "$c" "$REP/$c.tar.gz"; then
        echo "OK   $(du -h "$REP/$c.tar.gz" | cut -f1)"
    else
        echo "ÉCHEC pour $c"
        ERREURS=$((ERREURS + 1))
    fi
done

if [ "$ERREURS" -ne 0 ]; then
    echo "$ERREURS échec(s) — purge annulée par précaution."
    exit 1
fi

# Ne garder que les $GARDE répertoires les plus récents
cd "$DEST" || exit 1
ls -1d */ 2>/dev/null | sort -r | tail -n +$((GARDE + 1)) | while read -r vieux;
do
    echo "Purge de $vieux"
    rm -rf "./$vieux"
done

echo "Conservés :"
ls -1d "$DEST"/*/
df -h "$DEST"
echo "===== Terminé — $(date -Is)"
```

⚠ Il découvre les conteneurs plutôt que de les nommer — `incus list` —, donc un conteneur ajouté est exporté le mois suivant sans qu'on y pense. Même raisonnement que la découverte des bases dans le vidage MySQL.

 Et pour la même raison, il refuse de continuer sur une liste vide. Une liste vide ne veut pas dire « aucun conteneur », elle veut dire « Incus n'a pas répondu ». Sans ce contrôle, la boucle ne tourne pas, le compteur d'erreurs reste à zéro, et le script annonce un succès complet avec zéro conteneur exporté. ⚠ Un compteur d'échecs ne voit jamais l'absence de tentative — c'est le même défaut que le vidage MySQL évite de la même façon.

Le garde-fou se place avant la création du répertoire mensuel : quand Incus ne répond pas, rien ne doit être créé. Et il s'éprouve sans exporter cinq conteneurs, en remplaçant la liste par du vide dans une copie :

```
sed 's|incus list --format csv -c n|printf ""|' \
  /home/hostadmin/scripts/rsync/auto/backup_incus_export.sh > /tmp/exp-test.sh
bash /tmp/exp-test.sh ; echo "code de sortie : $?"
```

Attendu : le message d'échec, aucune ligne d'export, et code de sortie `1`.

⚠ `incus export` ne capture pas les disques attachés depuis l'hôte. L'export du conteneur web ne contient donc pas les 64 Go des sites, qui vivent sur un périphérique bloc monté sur `/var/www`. C'est le tirage du module `www-full` qui les porte, et personne d'autre.

⚠ Ses marqueurs diffèrent des autres : `===== Export Incus` et `===== Terminé`, sans ligne `échecs :`. La recherche `grep -E "^==="` les attrape quand même, mais le bilan chiffré manque. C'est un écart connu, pas un défaut à corriger dans l'urgence.

Ce qui n'est écrit par personne

 Tout ce qui est dans `Backups/` n'est pas sauvegardé pour autant. Plusieurs répertoires y ont été déposés à la main, ou l'ont été par un tirage aujourd'hui retiré. Ils ne bougent plus, et rien ne le dit.

Répertoire	Pourquoi il est là
`web-srv_22.04_Exportation contenu WordPress`	archive figée — la version *est* l'information
`lxd-vm_22.04_images`, `lxd-images_pre-migration`	exports d'avant la migration du gestionnaire de conteneurs, plus l'image d'une machine virtuelle de 2023 — figés en une fois, à ne jamais purger

Répertoire	Pourquoi il est là
`virtualbox_dirty/N73`	conservé volontairement : la clé Windows associée peut resservir
`Config/nas-host/bind`	reste d'un service retiré du conteneur web
`Configuration_logiciels/Sauvegardes manuelles/`	neuf dépôts faits à la main, certains depuis 2013

 La règle qui en découle : un répertoire de sauvegarde qui n'apparaît dans aucun script est une archive, pas une sauvegarde. Les deux se ressemblent trait pour trait dans un explorateur de fichiers.

D'où la séparation physique :

```
/media/nas1/Backups/Configuration_logiciels/Sauvegardes manuelles/
```

Tout ce qui a été déposé à la main y est descendu. Ce qui reste à la racine de `Configuration_logiciels/` est exactement ce que les scripts écrivent chaque nuit — `[.]bliss`, `[.]filezilla`, `musicip`, `puddletag`, `quodlibet` — et la distinction se lit sans ouvrir un seul script.

Un dépôt manuel peut avoir l'air maintenu. `[.]jxplorer` porte les crochets de la convention `[.]nom`, exactement comme deux répertoires réellement tirés — alors que rien ne l'a touché depuis plus de dix ans.

⚠ La condition à préserver, et elle n'est écrite nulle part ailleurs : aucun tirage ne doit jamais viser `Configuration_logiciels/` lui-même. Tous visent un sous-répertoire nommé. Un `--delete` sur le parent effacerait les dépôts manuels dès la nuit suivante — et ils ne sont, par définition, sauvegardés nulle part.

Et les services que personne ne tire

Le tableau ci-dessus recense des répertoires qui ont l'air vivants et ne le sont plus. Le trou symétrique est plus difficile à voir : un service dont la configuration n'a de répertoire nulle part. Il n'apparaît dans aucune liste, il ne vieillit sur aucun disque, et rien ne le distingue d'un service correctement

couvert tant qu’on ne cherche pas.

Le partage DLNA vers les téléviseurs — exclusion délibérée. Le service qui l’assure aujourd’hui n’est pas celui d’hier. Le précédent portait une configuration longuement ajustée, et un module la tirait ; le service actuel se configure en quelques lignes, et le remonter de zéro coûte moins que d’entretenir une sauvegarde de plus. Le module de l’ancien a disparu avec lui, rien n’a pris sa place, et c’est voulu.

 C’est écrit ici pour une seule raison : dans deux ans, l’absence de ce module ressemblera trait pour trait à un oubli. Une exclusion décidée et une exclusion subie ont exactement la même apparence sur un disque — seule la trace écrite les sépare.

Le serveur de musique de l’hôte — le trou qu’on vient de boucher. Sa configuration n’était couverte par aucun module ni aucun script, et le trou n’avait pas été choisi : il a été découvert, à l’occasion d’un incident, après des années. Il fait aujourd’hui l’objet de `backup_lyrion.sh`, plus haut sur cette page.

⚠ Une sauvegarde absente ne se signale jamais. Aucune erreur, aucun journal, aucun symptôme : rien n’indiquait que ce service n’était pas couvert. C’est le mode de défaillance que l’inventaire des données, en tête de cette page, existe pour attraper — et il ne l’attrape que si on le relit. Un service qui s’ajoute au parc n’ajoute pas sa propre ligne à cet inventaire ; quelqu’un doit le faire.

Le cas inverse : un module qui sauvegardait le vide

Un module exposait `/var/vmail` — la racine des boîtes de courriel du conteneur web — et aucun script ne le tirait. La crainte, chaque fois qu’on le rencontrait, était qu’il contienne un historique de messages sans copie ailleurs. Personne n’osait y toucher.

Il ne contenait rien. Quatre fichiers, 9,5 Ko : le squelette créé par le panneau d’hébergement à l’installation. Aucune boîte, aucun message. Le module

exposait un répertoire vide depuis des années, et sa seule production était l’hésitation qu’il provoquait.

 Vérifier a coûté une commande ; ne pas vérifier avait coûté des années d’incertitude. C’est le rapport habituel entre les deux, et il vaut d’être noté : devant un répertoire dont on ne sait pas ce qu’il contient, la mesure est presque toujours moins chère que la prudence.

Quatre vérifications avant de démonter la pile entière, et chacune répondait à une question différente :

Question	Ce qu’on a regardé
Du courrier arrive-t-il encore ici ?	l’enregistrement MX public du domaine, qui pointe ailleurs
Du courrier est-il parti d’ici ?	l’absence de tout `status=sent` au journal du serveur
Les sites s’en servent-ils ?	ils passent par le SMTP du fournisseur, pas par le `sendmail` local
Le panneau le régénérerait-il ?	l’option correspondante était déjà décochée

Vingt et un paquets ont été purgés — serveur SMTP, serveur IMAP, analyseur antivirus, filtre antipourriel et leurs dépendances —, puis les répertoires de configuration et de données retirés, et le module supprimé de `rsyncd.conf`.

 La file d’attente contenait 1437 messages de `root` jamais livrés, empilés depuis des années : des sorties de tâches cron adressées à un système de courriel qui ne fonctionnait pas. C’est le symptôme d’une règle énoncée ailleurs sur cette page — une sortie envoyée quelque part où personne ne la lit équivaut à une sortie jetée.

⚠ Le bénéfice n’est pas la place — 56,7 Mo. C’est la surface d’attaque et l’entretien : un analyseur de formats exposé à des données hostiles, un serveur IMAP et POP3 ouvert sans aucune boîte derrière, et une mise à jour quotidienne de signatures antivirus pour protéger un courrier inexistant.

Deux conventions de nommage

❌ Jamais de numéro de version de système

Un nom de répertoire de sauvegarde ne porte jamais la version du système de la machine sauvegardée. Sinon chaque montée de version oblige à réécrire des scripts qui n’ont rien à voir avec elle — et à renommer des répertoires pendant que des tâches nocturnes tournent.

Le parc en portait huit ; ils ont été renommés. ⚠️ Le renommage et la réécriture des scripts tiennent dans la même fenêtre : si une tâche partait entre les deux, rsync recréerait le nom manquant et retirerait tout le contenu une seconde fois.

Les seules exceptions sont les archives figées du tableau ci-dessus : aucun script actif ne les écrit, et la version *y est* l’information.

🔒 Les crochets de `[.]nom`

Plusieurs destinations portent des crochets : `[.]bliss`, `[.]filezilla`, `[.]MusicMagic`, `[.]config`.

Ce n’est pas une faute de motif shell. C’est une convention délibérée : les crochets rendent visible dans la sauvegarde ce qui est caché à la source. Un répertoire `.filezilla` disparaîtrait de l’affichage d’un explorateur ou d’un `ls` ordinaire, et on oublierait qu’il est sauvegardé.

⚠️ Avant de « corriger » une bizarrerie qui dure depuis des années, demander pourquoi elle est là. Sa durée est une information.

La clé de démarrage — imager un

périphérique plutôt qu’un dossier

Ce serveur ne sait pas démarrer sur son disque système : sa carte mère est en BIOS hérité et ne reconnaît pas le NVMe comme périphérique d’amorçage. `/boot` vit donc sur une clé USB, qui doit rester branchée en permanence.

⚠ C’est le point de fragilité de la machine, et il a une propriété désagréable : une copie déposée sur son propre disque serait inutilisable, puisqu’il faudrait démarrer pour la lire. Cette sauvegarde-là n’a de sens que hors site.

Le disque entier, pas la partition

```
sudo lsblk -o NAME,SIZE,TYPE,MOUNTPOINTS,MODEL
```

La clé porte deux partitions : une minuscule d’un mégaoctet, et celle montée sur `/boot`. L’image doit couvrir le périphérique entier, pas la seconde partition.

Pourquoi cette petite partition d’un mégaoctet compte plus que sa taille ne le laisse croire. Elle porte le cœur de GRUB — le code que le BIOS charge après le secteur d’amorçage. Copier seulement `/boot` donnerait une partition pleine de fichiers justes sur une clé qui ne démarrerait pas, et le diagnostic serait déroutant : tout serait là, rien ne fonctionnerait. La table de partition et le secteur d’amorçage sont dans le même cas — ils n’existent nulle part ailleurs.

Lecture seule, puis remise à zéro des blocs libres

Deux gestes qui se renforcent, à faire avant la copie.

```
sudo mount -o remount,ro /boot
```

Imager un système de fichiers monté en écriture peut capturer un état incohérent. `/boot` ne sert à rien en fonctionnement normal — il n’est lu qu’au démarrage et écrit qu’à l’installation d’un noyau — donc la bascule est indolore.

```
sudo zerofree -v /dev/sdc2
```

⚠ Sans cette étape, la compression ne sert presque à rien. Un `dd` copie tous les secteurs, y compris ceux qui ne contiennent plus que d'anciennes données effacées — et ces octets-là ne se compressent pas. `zerofree` remet à zéro les blocs marqués libres, sans toucher à aucun fichier existant, et il exige justement un montage en lecture seule.

Ses trois nombres méritent d'être lus : blocs remis à zéro, blocs libres, blocs au total. Sur cette machine, `796363/3620518/3776512` — soit 14,8 Go libres sur 15,5, dont 3,3 Go contenaient encore d'anciennes données. C'est exactement ce qui aurait résisté à la compression.

L'image

```
sudo dd if=/dev/sdc bs=4M status=progress \
  | gzip -c > /chemin/vers/boot-usb-AAAA-MM-JJ.img.gz
```

```
sudo mount -o remount,rw /boot
```

⚠ Ne pas oublier le remontage en écriture. Une partition `/boot` laissée en lecture seule ne se remarque pas — jusqu'à la prochaine mise à jour de noyau, qui échoue alors sans rapport apparent avec ce qu'on a fait des semaines plus tôt.

Le gain est considérable, et mesuré : 274 Mo de contenu réel dans une clé de 15 Go donnent une image compressée de 261 Mio — 272 833 688 octets — contre 15,5 Go pour un `dd` nu. Cinquante-neuf fois plus petit, pour exactement les mêmes secteurs. Ça compte doublement quand ce fichier part hors site.

Vérifier souvent, agir rarement

Une image prise une fois vieillit : `/boot` change à chaque nouveau noyau, donc à chaque mise à jour automatique. Une image de six mois ne contient plus le noyau

sur lequel la machine démarre.

La réponse n’est pas de la refaire périodiquement — ce serait réécrire 15 Go pour rien la plupart du temps — mais de vérifier chaque nuit et de n’agir qu’au changement.

L’empreinte porte sur les noms, tailles et dates des fichiers de `/boot`, pas sur leur contenu : lire 274 Mo chaque nuit pour ne rien trouver serait du gaspillage, et ces trois attributs suffisent. Un noyau installé, un initrd régénéré ou un `grub.cfg` réécrit les modifient tous.

```
find /boot -type f -printf '%p %s %T@\n' | sort | sha256sum | cut -d' ' -f1
```

Le script complet, à déposer dans le répertoire des scripts de sauvegarde :

```
#!/bin/bash
. /home/hostadmin/scripts/rsync/auto/verifier_montages.sh
# Image de la clé USB d'amorçage de nas-host, rafraîchie automatiquement.
#
# nas-host ne sait pas démarrer sur son NVMe en BIOS hérité : /boot vit sur une
# clé USB. Si elle lâche, la machine ne démarre plus — et une copie déposée
# sur son propre disque serait inaccessible, puisqu'il faudrait démarrer pour
# la lire. D'où une image hors site.
#
# PRINCIPE : vérifier chaque nuit, n'agir que si /boot a changé. Un nouveau
# noyau change /boot ; une nuit ordinaire, non. L'image suit donc les noyaux
# sans intervention et sans réécrire 15 Go pour rien.
#
# DOIT TOURNER EN ROOT : remontage, zerofree et dd l'exigent.
#
# Version 1.1.0 — 8 septembre 2026
#
# 1.1.0 — deux ajouts après un essai de lancement manuel.
#   a) Refus explicite hors root. Sans lui, un lancement sans sudo échouait
#      au remontage, et le filet de sécurité annonçait alors que /boot était
#      resté en lecture seule — alors qu'il n'y avait jamais été mis. Un
#      filet qui produit un faux signal est pire qu'aucun filet.
#   b) Progression de dd à l'écran quand la sortie est un terminal, silence
```

```
#      sous cron. Le même script sert donc aux deux usages sans réglage.

set -u

# --- Doit tourner en root : mount, zerofree et dd l'exigent -----------------
if [ "$(id -u)" -ne 0 ]; then
    echo "ÉCHEC : ce script doit tourner en root."
    echo "        sudo $0"
    exit 1
fi

CLE=/dev/sdc          # le disque ENTIER : table de partition + amorçage + /boot
PART=/dev/sdc2        # la partition ext4 montée sur /boot
DEST="/media/nas1/Backups/Sauvegarde-OS/Clé usb d'amorçage de nas-host"
EMPREINTE=/var/lib/boot-image.empreinte
GARDER=3              # nombre d'images conservées

NOM="boot-usb-$(date +%F).img.gz"
ERREURS=0

# Progression à l'écran en lancement manuel, silence sous cron.
if [ -t 1 ]; then DD_ETAT=progress ; else DD_ETAT=none ; fi

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

# ------------------------------------------------------------------------------
# 1. Faut-il refaire l'image ?
# ------------------------------------------------------------------------------
# L'empreinte porte sur les noms, tailles et dates des fichiers de /boot —
# pas sur leur contenu. Lire 274 Mo chaque nuit pour ne rien trouver serait
# du gaspillage, et ces trois attributs suffisent : un noyau installé, un
# initrd régénéré ou un grub.cfg réécrit les modifient tous.
ACTUELLE=$(find /boot -type f -printf '%p %s %T@\n' 2>/dev/null \
           | sort | sha256sum | cut -d' ' -f1)

if [ -z "$ACTUELLE" ]; then
    echo "ÉCHEC : impossible de lire /boot — est-il monté ?"
    echo "=== $(basename "$0") — fin $(date -Is) — échecs : 1"
    exit 1
fi

if [ -f "$EMPREINTE" ] && [ "$(cat "$EMPREINTE")" = "$ACTUELLE" ]; then
```

```
    echo "/boot inchangé — aucune image à refaire."
    echo "=== $(basename "$0") — fin $(date -Is) — échecs : 0"
    exit 0
fi

echo "/boot a changé depuis la dernière image — nouvelle copie."
[ -d "$DEST" ] || { echo "ÉCHEC : $DEST introuvable"; exit 1; }

# ------------------------------------------------------------------------
# 2. Filet de sécurité : /boot doit TOUJOURS redevenir inscriptible
# ------------------------------------------------------------------------
# Sans ce piège, une erreur au milieu du traitement laisserait /boot en
# lecture seule. Rien ne le remarquerait avant la prochaine mise à jour de
# noyau, qui échouerait alors sans rapport apparent avec ce script.
remettre_rw() {
    mount -o remount,rw /boot 2>/dev/null
    if mount | grep -q ' /boot .*[(,]ro[,)]'; then
        echo "ALERTE : /boot est resté en LECTURE SEULE — le remettre à la main
:"
        echo "         sudo mount -o remount,rw /boot"
    fi
}
trap remettre_rw EXIT

# ------------------------------------------------------------------------
# 3. Lecture seule, puis remise à zéro des blocs libres
# ------------------------------------------------------------------------
# Deux raisons de passer en lecture seule, et elles se renforcent :
#   - imager un système de fichiers monté en écriture peut capturer un état
#     incohérent ;
#   - zerofree l'exige, et refuse de travailler autrement.
if ! mount -o remount,ro /boot; then
    echo "ÉCHEC : impossible de passer /boot en lecture seule"
    exit 1
fi

# zerofree ne touche à AUCUN fichier : il remet à zéro les blocs marqués
# libres, pour que gzip les réduise à presque rien. Sans lui, l'image
# compressée porterait encore les anciennes données effacées.
if ! zerofree "$PART"; then
    echo "ÉCHEC : zerofree sur $PART"
    ERREURS=$((ERREURS + 1))
```

```
    exit 1
fi

# ------------------------------------------------------------------------
# 4. L'image, du disque ENTIER
# ------------------------------------------------------------------------
# Du disque entier, pas de la seule partition : c'est ce qui rend la clé
# amorçable. La table de partition et la petite partition d'amorçage BIOS —
# celle qui porte le noyau de GRUB — n'existent nulle part ailleurs.
#
# Écriture sous un nom temporaire, renommé seulement en cas de succès : une
# image tronquée ne doit jamais porter le nom d'une image valide.
TMP="$DEST/.$NOM.tmp"

if dd if="$CLE" bs=4M status="$DD_ETAT" | gzip -c > "$TMP"; then
    mv "$TMP" "$DEST/$NOM"
    echo "$ACTUELLE" > "$EMPREINTE"
    echo "OK     $NOM  ($(ls -lh "$DEST/$NOM" | awk '{print $5}'))"
else
    rm -f "$TMP"
    echo "ÉCHEC  dd/gzip — l'image précédente est conservée"
    ERREURS=$((ERREURS + 1))
fi

# ------------------------------------------------------------------------
# 5. Ne garder que les dernières
# ------------------------------------------------------------------------
# Uniquement si tout a réussi : sur un échec, on ne touche pas à ce qui
# existe. Un script de sauvegarde qui détruit sur une erreur est pire que
# celui qui ne fait pas le ménage.
if [ "$ERREURS" -eq 0 ]; then
    ls -1t "$DEST"/boot-usb-*.img.gz 2>/dev/null \
      | tail -n +$((GARDER + 1)) | while read -r vieille; do
        rm -f "$vieille" \
          && echo "RETIRÉ $(basename "$vieille") — au-delà des $GARDER
conservées"
    done
fi

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

Quatre décisions valent l’explication.

⚠ Le piège `trap` remet toujours `/boot` en écriture, y compris si le script s’arrête en cours de route. Sans lui, une erreur laisserait la partition en lecture seule et rien ne le remarquerait avant la prochaine mise à jour de noyau. Il vérifie même que le remontage a réussi, et le dit sinon.

⚠ L’image s’écrit sous un nom temporaire, renommée seulement en cas de succès. Une image tronquée ne doit jamais porter le nom d’une image valide.

⚠ L’empreinte n’est enregistrée qu’après une image réussie. Si la copie échoue, l’empreinte reste l’ancienne et le script réessaiera la nuit suivante — au lieu de croire le travail fait.

Le ménage des anciennes n’a lieu que si tout a réussi. Sur un échec, on ne touche à rien.

Ce script a besoin de root

`mount`, `zerofree` et `dd` l’exigent. Il ne peut donc pas vivre dans la crontab du compte de sauvegarde comme les autres. `/etc/cron.d/` convient bien : le fichier est explicite, il nomme l’utilisateur, et il est sauvegardé avec le reste de `/etc`.

`/etc/cron.d/image-cle-amorcage` :

```
# Image de la clé USB d'amorçage de nas-host.
# Ne fait quelque chose qu'au changement de /boot — donc au changement de noyau.
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
MAILTO=""

00 04 * * * root /home/hostadmin/scripts/rsync/auto/backup_cle_amorcage.sh >>
/var/log/backup/$(date +\%F).log 2>&1
```

04 h 00 n’est pas un choix arbitraire. Les autres tâches nocturnes s’enchaînent de 00 h 15 à 01 h 30, et le 1er du mois jusqu’à 03 h 30. Une nuit ordinaire,

celle-ci ne fait que lire `/boot` et repartir en une seconde ; mais la nuit d'un nouveau noyau elle occupe onze minutes de lecture disque, et il vaut mieux qu'elle ne les prenne pas pendant qu'une autre copie travaille.

Le `%` doit être échappé dans une ligne de cron — sans la barre oblique inverse, cron le lit comme une fin de commande et la redirection tombe à côté. C'est l'erreur classique des lignes de cron qui datent leur journal.

Mise en place

Le script doit être exécutable, et le fichier de cron doit appartenir à root.

```
chmod +x /home/hostadmin/scripts/rsync/auto/backup_cle_amorcage.sh
sudo chown root:root /etc/cron.d/image-cle-amorcage
sudo chmod 644 /etc/cron.d/image-cle-amorcage
```

⚠ Cron ignore un fichier de `/etc/cron.d/` qui n'appartient pas à root ou qui est inscriptible par le groupe — et il l'ignore en silence. Aucune erreur, aucun journal : la tâche ne s'exécute simplement jamais. C'est une panne particulièrement pénible à diagnostiquer, parce que tout *semble* installé.

Amorcer l'empreinte, juste après avoir fabriqué la première image à la main :

```
sudo sh -c 'find /boot -type f -printf "%p %s %T@\n" \
            | sort | sha256sum | cut -d" " -f1 \
            > /var/lib/boot-image.empreinte'
```

⚠ Sans ce geste, la nuit suivante refait quinze gigaoctets pour rien. Le fichier d'empreinte n'existe pas encore, le script en conclut que `/boot` a changé, et il recopie une image qu'on vient tout juste de faire.

Vérifier, en lançant le script à la main :

```
sudo /home/hostadmin/scripts/rsync/auto/backup_cle_amorcage.sh
```

Il doit refuser de travailler :

```
=== backup_cle_amorcage.sh — début 2026-09-08T12:01:00-04:00
/boot inchangé — aucune image à refaire.
=== backup_cle_amorcage.sh — fin 2026-09-08T12:01:00-04:00 — échecs : 0
```

C’est le bon essai à faire, et il est gratuit. Il prouve d’un coup que le script est exécutable, qu’il lit `/boot`, qu’il trouve son fichier d’empreinte et qu’il sait s’arrêter — sans rien écrire ni immobiliser la machine onze minutes.

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.

Deux tâches vivent ailleurs que dans la crontab

 La crontab du compte de sauvegarde n’est pas l’inventaire des tâches de la machine, et c’est un piège. Deux tâches tournent depuis `/etc/cron.d/`, sous root, parce qu’elles exigent des privilèges que ce compte n’a pas. Elles n’apparaissent nulle part dans la crontab, et ne se révèlent que dans le journal.

```
/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
```

Pour inventorier ce qui s’exécute réellement, il faut donc 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 — qui tournent sous un compte ordinaire — 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 de la nuit est à 00:15 sous le compte de sauvegarde : l’ordre tient, et c’est à vérifier avant d’avancer l’heure d’une tâche root.

⚠ Cron ignore en silence un fichier de `/etc/cron.d/` qui n’appartient pas à root, qui est inscriptible par le groupe, dont le nom contient un point, ou qui ne se termine pas par un saut de ligne. Aucune erreur, aucun journal : la tâche ne s’exécute simplement jamais. Le contrôle qui tranche est de redémarrer cron et de lire ce qu’il dit — une ligne mal formée s’y signale par un `ERROR` nommant le fichier.

```
sudo systemctl restart cron
sleep 3 ; sudo journalctl -u cron --since "1 min ago" --no-pager | tail
```

La réplication hors site n’a pas d’heure, et c’est voulu

Le niveau hors site ne figure pas dans la crontab : `incus-exports/` est répliqué par un outil qui observe le système de fichiers, avec verrou et anti-rebond, et qui passe en plus une fois par nuit sur l’arborescence entière.

 La passe complète est la garantie ; l’observateur n’est qu’une optimisation. C’est le bon sens de cette répartition : une couche événementielle peut manquer un changement — un événement perdu, un redémarrage au mauvais moment, un dépassement de la file du noyau — et rien ne le signalerait. La passe complète, elle, ne dépend d’aucun événement : elle compare. Le pire cas est donc vingt-quatre heures de retard, jamais une absence.

Il n’y a par conséquent aucun ordonnancement à coordonner avec les tâches de la nuit. Un export mensuel produit à 03:00 part quand il part ; il sera dehors au plus tard le lendemain.

⚠ La conservation est de trente jours, en mode révision. Une archive corrompue ou effacée par erreur reste récupérable pendant un mois. Au-delà, la version saine a disparu en ligne — c’est le niveau, et non le mécanisme, qui porte cette limite.

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.

Le démon est un point d’appui unique, et silencieux

 Un `killall rsync` lancé en root sur l’hôte arrête 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.

⚠ Et rien ne les relance. `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 reste hors service sur les deux machines sans qu'aucun signal ne l'indique.

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.

Deux surcharges dans le même répertoire, et il ne faut pas les confondre

La surcharge de relance ci-dessus n’est pas la seule que porte ce service. Une seconde, posée à l’installation, le durcit : `ProtectSystem=full`, `PrivateDevices=yes`, `NoNewPrivileges=yes`. Elle rend `/usr`, `/boot` et `/etc` en lecture seule pour ce service seulement, ce qui est sans effet sur son travail — tous les modules étant déclarés `read only = true`, le démon n’écrit nulle part — mais borne ce qu’une faille dans rsync permettrait d’atteindre. Les commandes et le tableau des trois options sont dans la page de l’hôte, qui est l’endroit où cette machine se rebâtit.

 Le raisonnement est celui des filtres de `/etc` : réduire ce qu’un composant *peut* faire, et pas seulement ce qu’il fait. Un service qui n’écrit jamais n’a aucune raison d’en garder le droit.

⚠ Ce sont deux fichiers distincts dans `/etc/systemd/system/rsync.service.d/`, et c’est délibéré. Rien n’empêcherait de les fondre en un seul, mais les réunir ferait qu’une erreur en modifiant l’un emporterait l’autre — et l’un des deux est ce qui relève le démon après un arrêt accidentel. Vérifier ce qui est réellement actif se fait sur le service, pas sur les fichiers :

```
systemctl show rsync \
```

```
   -p ProtectSystem -p PrivateDevices -p NoNewPrivileges -p Restart
```

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.

🖴 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.

🗑 Plusieurs tirages écrivent dans le même répertoire avec `--delete`

Deux tirages de configuration visent aujourd’hui `Backups/Config/nas-host/` — `smb.conf` et `exports` —, 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 configurations s’effacent mutuellement.

🗑 Et le même comportement a un revers. Qu’un tirage visant ce répertoire soit commenté, et son fichier reste — protégé de la suppression par les filtres des tirages voisins, qui l’excluent. Il vieillit sur place indéfiniment, sans que rien n’indique qu’il n’est plus rafraîchi. Un cas de ce parc y a séjourné trois

ans avant d’être retiré à la main.

 Ce qui fait tenir le partage d’un répertoire est exactement ce qui y laisse un fichier mort indéfiniment. Retirer un tirage n’est donc pas fini tant que son fichier n’a pas été retiré à la main.

La détection d’échec ne dépend plus d’une lecture humaine — mais elle dépend encore d’un registre

Ce qui était vrai et ne l’est plus. Les journaux existaient, et rien n’y signalait un échec. Le cas concret : un vidage de base qui échoue, le script qui conserve correctement la version de la veille, le serveur qui la tire sans broncher. Tout paraissait normal, et la sauvegarde vieillissait en silence jusqu’à ce que quelqu’un ouvre le journal.

Un relevé nocturne lit désormais ces journaux à la place de l’œil humain, contrôle l’âge des répertoires de destination, et énonce un verdict sur une page — voir *Savoir que ça fonctionne*. Le cas du vidage figé est précisément celui qu’il attrape : le fichier ne change plus, son répertoire vieillit, et l’âge est mesuré indépendamment de ce que le journal raconte.

 Mais le relevé ne vaut que ce que vaut son registre, et c’est là que la fragilité s’est déplacée. Il ne cherche que les tâches et les répertoires qu’on lui a nommés. Une tâche ajoutée à la crontab sans être ajoutée au registre du relevé n’est surveillée par rien, exactement comme avant — à ceci près que la page affiche maintenant un bilan rassurant qui ne la concerne pas.

⚠ Ajouter une tâche de sauvegarde est donc un geste en deux temps, et le second est facile à oublier : la ligne de cron, puis l’entrée dans le registre. Le premier seul produit une sauvegarde qui fonctionne et que personne ne surveille ; le second seul produit une alerte quotidienne pour une tâche qui n’existe pas. Des deux oublis, c’est le premier qui est dangereux, parce qu’il est silencieux.

Et la lecture humaine n’a pas disparu, elle s’est déplacée : quelqu’un doit ouvrir la page. Une page qu’on cesse de consulter ramène exactement à la

situation d’avant, avec en plus l’illusion d’être couvert.

Savoir que ça fonctionne

Le relevé nocturne, et la page qu’il fabrique

Tout ce qui suit dans cette section se faisait à la main. Une tâche de 4 h 30 le fait désormais chaque nuit et en dépose le résultat sur une page. Les contrôles manuels restent décrits ici — ils servent à vérifier le relevé lui-même, et à travailler le jour où il ne tourne plus.

Le dispositif tient en deux scripts qui ne partagent aucun travail :

Script	Ce qu’il fait
`releve-tableau-de-bord.sh`	mesure et juge — lit les journaux de la nuit, l’âge des répertoires de destination, les services, les montages, l’espace disque, les prisons `fail2ban` des deux machines et le trafic d’Internet vers le seul service public sans authentification ; écrit un fichier de valeurs
`build_tableau-de-bord.py`	met en forme seulement — fabrique la page HTML et tient l’historique des trente derniers jours

 Toute la connaissance du parc vit dans le premier, aucune dans le second. Ce qui compte comme une alerte, quel âge est toléré, quelle tâche doit avoir tourné : le script de mesure décide, le formateur ne fait que rendre. Changer un seuil ne demande donc jamais de toucher au HTML, et refaire la présentation ne peut pas altérer un verdict.

Ce que le relevé cherche dans les journaux est une absence, pas une erreur. Chaque script de tirage écrit un marqueur de début et un marqueur de fin dans le journal du jour. Une tâche qui échoue laisse un marqueur de fin avec un compte d’échecs ; une tâche qui n’a pas démarré — parce que le garde-fou de montage l’a interrompue avant tout, par exemple — ne laisse rien du tout. C’est cette absence-là que le relevé sait nommer, et c’est elle qu’une lecture

humaine rate le plus facilement : on repère une ligne d’erreur, on ne repère pas une ligne manquante.

Deux familles de contrôles, et elles sont complémentaires : ce que les journaux racontent, et ce que le disque montre. Un répertoire de destination trop vieux est une alerte même si le journal de la nuit est parfait — c’est exactement le cas du vidage de base figé.

La ligne de crontab, sur nas-host, sous le compte qui porte les tâches :

```
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
```

⚠ 4 h 30 vient après la dernière tâche de la nuit, et ce n’est pas un détail : avancé avant, le relevé dénoncerait chaque matin comme manquantes des tâches qui n’ont simplement pas encore tourné. Un tableau de bord qui crie tous les jours cesse d’être lu en une semaine. Et 4 h 30 plutôt que 4 h 15 parce que l’image de la clé d’amorçage part à 4 h et occupe onze minutes les nuits où un nouveau noyau a changé `/boot` — la marge est mince, elle est mesurée.

Le `&&` entre les deux commandes est délibéré. Si la mesure échoue, la mise en forme ne s’exécute pas : 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. La date affichée est donc elle-même un contrôle.

⚠ Cette ligne renvoie sa sortie vers le néant, alors que cette page l’interdit ailleurs — et l’exception est justifiée, pas tolérée. La règle vise les tâches dont la sortie est la seule trace de ce qu’elles ont fait. Ici, la sortie n’est pas la trace : le résultat est un fichier, et ce fichier porte sa propre date. Une tâche qui n’a pas tourné laisse une page datée de la veille, visible du premier coup d’œil, là où un journal aurait demandé qu’on aille l’ouvrir. Le rapport de terminal, lui, n’a rien à faire dans le journal des sauvegardes, qu’il noierait.

 Et la distinction va plus loin que la redirection : cette tâche ne produit

pas d’événement, elle livre une lecture. Les journaux qu’elle dépouille existent déjà et gardent, eux, la trace de ce qui s’est passé pendant la nuit. Lui ouvrir un journal à elle reviendrait à consigner l’acte de lire.

Son seul mode de défaillance visible est donc la page elle-même, restée à la date de la veille. Et l’objection habituelle — « si personne ne la regarde, personne ne le verra » — ne tient pas ici : une page de veille que personne n’ouvre n’a aucune raison d’exister. Le jour où c’est vrai, le problème n’est plus la redirection.

Le dispositif complet — les deux scripts au long, le lanceur du poste de travail et son raccourci de bureau — est écrit en fin de page, section *Le tableau de bord*.

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.

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.

Le gain : un module mort ne se signale d’aucune autre façon. Passé sur l’ensemble des modules du parc, ce contrôle en a trouvé deux — un répertoire disparu après le déménagement d’un service, un chemin de configuration changé par une mise à jour — tous deux en échec silencieux depuis des années.

Le message à savoir lire : `@ERROR: chroot failed`

C’est la réponse qu’un module rend quand son chemin n’existe plus. Le démon s’enferme par `chroot` dans le chemin du module avant de servir quoi que ce soit ; si le répertoire a disparu, l’enfermement échoue et le service s’arrête là.

 Le message ne parle pas du chemin, et c’est ce qui le rend opaque la première fois. Il décrit l’opération qui a échoué, pas sa cause. Devant lui, la question à poser est toujours la même : *ce chemin existe-t-il encore sur la machine qui sert le module ?*

Le cas rencontré : un module servant `/etc/bind` sur l’hôte, longtemps après que le DNS eut déménagé dans un conteneur. Aucune donnée perdue — la destination n’avait même jamais été créée — mais le module a échoué chaque nuit dans l’intervalle, sans que rien d’autre qu’une ligne de journal ne le dise.

Trois contrôles qui n’en sont pas

Trois gestes couramment pris pour des vérifications, et qui ne prouvent rien.

 Un bilan à `échecs : 0` mesure l’exécution, pas la destination. Un script relancé après un renommage rend zéro qu’il écrive au bon endroit ou non, puisque rsync crée ce qui manque. Le contrôle qui tranche cherche les noms qui ne devraient plus exister :

```
find /media/nas1/Backups /media/nas2/Backups -maxdepth 2 -name '*22.04*'
```

 `rsync --list-only` ne descend pas dans les sous-répertoires sans `-r`. Un contrôle bâti là-dessus rend exactement la même sortie que le filtre serve le fichier attendu ou non — et cette sortie a l'air complète.

 `ls` ne montre pas les fichiers cachés. Un répertoire de sauvegarde qui paraît vide peut contenir un `.htpasswd` ou un `.htdigest` — précisément les fichiers qu'on tient à sauvegarder. `ls -la`, ou mieux, `find` :

```
find /media/nas1/Backups/Config -type f -printf '%p  %s o  %TY-%Tm-%Td\n'
```

 Ce que les trois ont en commun : chacune répond fidèlement à une question qui n'est pas celle qu'on croit poser, et aucune ne se plaint. Devant un contrôle rassurant, la question à se poser est : *cette commande pourrait-elle distinguer le cas où tout va bien du cas où rien ne va ?*

L'essai de restauration, deux fois l'an

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

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.

Le tableau de bord — le dispositif au complet

Ce qui suit bâtit de zéro la page décrite plus haut. Deux scripts sur le serveur, un lanceur sur le poste de travail, et rien d'autre : ni service, ni

base de données, ni dépendance réseau.

 Le partage du travail entre les deux scripts est la décision structurante, et tout le reste en découle. Le relevé mesure et juge — c'est lui qui sait ce qu'est une alerte. Le constructeur met en forme et ne juge rien. Un seuil se change donc sans jamais toucher au HTML, et la présentation se refait sans risque d'altérer un verdict. Et regarder le fichier de données avant la page localise une panne du premier coup.

Avant de commencer

Trois conditions, toutes vérifiables en une commande.

Sur `[nas-host]` :

```
{
echo "--- compte dans incus-admin ?"
id -nG | tr ' ' '\n' | grep -x incus-admin || echo NON
echo "--- python3"
python3 --version 2>&1
echo "--- dossier cible"
ls -d "/media/nas1/Backups/Tableau de bord - nas-host" 2>&1
}
```

Le compte `hostadmin` doit appartenir au groupe `incus-admin` : le relevé interroge les conteneurs et `fail2ban` à travers `incus exec`, sans quoi il faudrait le faire tourner en root. Le dossier cible doit exister et lui appartenir.

S'il manque, le créer :

```
mkdir -p "/media/nas1/Backups/Tableau de bord - nas-host"
```

Le relevé

Sur `[nas-host]` — ouvrir l'éditeur :

```
mkdir -p /home/hostadmin/scripts/tableau-de-bord && \
nano /home/hostadmin/scripts/tableau-de-bord/releve-tableau-de-bord.sh
```

Contenu :

```
#!/bin/bash
# Relevé de l'état des sauvegardes et des services de nas-host.
#
# Mesure, affiche à l'écran, et écrit UN fichier de données — donnees.tsv,
# dans le dossier du tableau de bord. Rien d'autre, nulle part ailleurs.
#
# La page est fabriquée par un second script qui ne fait que mettre en forme
# ce fichier. Le jugement — ce qui compte comme une alerte — vit ici, à un
# seul endroit.

set -u
export LC_ALL=C.UTF-8

DOSSIER="/media/nas1/Backups/Tableau de bord - nas-host"
TSV="$DOSSIER/donnees.tsv"

JOURNAL_DIR=/var/log/backup
JOURNAL="$JOURNAL_DIR/$(date +%F).log"
[ -f "$JOURNAL" ] || JOURNAL=$(ls -1t "$JOURNAL_DIR"/*.log 2>/dev/null | head -1)

JOUR=$(date +%d)
AUJOURDHUI=$(date -d "$(date +%F)" +%s)

# nom|rythme|heure prévue
TACHES="
backup_mysql-ns2.sh|quotidien|00:15
backup_web-ns2.sh|quotidien|00:20
backup_pihole.sh|quotidien|00:35
backup_scripts.sh|quotidien|00:40
backup_config.sh|quotidien|00:45
backup_scripts_poste-bureau.sh|quotidien|00:50
backup_config_poste-bureau.sh|quotidien|00:55
backup_logiciels.sh|quotidien|01:00
backup_lyrion.sh|quotidien|01:05
upload_photos_photos-webadmin-ca.sh|quotidien|01:30
```

```
backup_virtualbox_poste-bureau.sh|mensuel|02:00
backup_incus_export.sh|mensuel|03:00
backup_cle_amorcage.sh|quotidien|04:00
"

# jours tolérés|chemin
TEMOINS="
1|/media/nas1/Backups/web-srv_MySQL_db
2|/media/nas1/Backups/Config/web-srv/log
2|/media/nas1/Backups/Config/pi-hole
2|/media/nas1/Backups/Configuration_logiciels/lyrion
35|/media/nas1/Backups/incus-exports
90|/media/nas1/Backups/Sauvegarde-OS/Clé usb d'amorçage de nas-host
"

SERVICES="icecast2 ices2-chromecast lyrionmusicserver minidlna smbd nmbd
          nfs-server incus zfs-zed"

if [ ! -d "$DOSSIER" ]; then
    echo "ÉCHEC : $DOSSIER introuvable"
    exit 1
fi
: > "$TSV" || { echo "ÉCHEC : impossible d'écrire $TSV"; exit 1; }

# Les champs sont séparés par une tabulation. « $* » les joint avec le
# premier caractère d'IFS, d'où le réglage local.
emit() {
    local IFS=$'\t'
    printf '%s\n' "$*" >> "$TSV"
}

# Les boucles « while read » alimentées par un tube tournent dans un
# sous-processus : une variable de comptage incrémentée dedans serait perdue
# à la sortie. Les alertes vont donc dans le fichier, et le total se compte
# à la fin en le relisant.
alerte() { emit ALERTE "$1"; }

# printf complète en OCTETS, pas en caractères : un « é » en occupe deux et
# décale la colonne. En locale UTF-8, ${#texte} compte bien les caractères.
col() {
    local texte="$1" large="$2" n
    n=${#texte}
```

```
    printf '%s' "$texte"
    while [ "$n" -lt "$large" ]; do printf ' '; n=$((n + 1)); done
}

ligne4() {
    col "$1" 36; col "$2" 7; col "$3" 12; printf '%s\n' "$4"
}

emit META genere "$(date -Is)"
emit META journal "$(basename "$JOURNAL")"

echo "RELEVÉ DU $(date '+%F %H:%M')   journal : $(basename "$JOURNAL")"
echo

# ------------------------------------------------------------------------
# 1. Les tâches de la nuit
# ------------------------------------------------------------------------
# On part du registre des tâches ATTENDUES, pas de ce que le journal
# contient. C'est le seul moyen de voir une tâche arrêtée avant sa première
# ligne — le cas d'un disque non monté, où le script sort sur « ARRÊT : »
# sans laisser ni début ni fin.
echo "== TÂCHES"
ligne4 NOM HEURE ÉTAT DÉTAIL
echo "$TACHES" | while IFS='|' read -r nom rythme heure; do
    [ -n "$nom" ] || continue

    if [ "$rythme" = mensuel ] && [ "$JOUR" != "01" ]; then
        ligne4 "$nom" "$heure" "mensuelle" "prévue le 1er"
        emit TACHE "$nom" "$heure" mensuelle "prévue le 1er"
        continue
    fi

    if [ "$nom" = backup_incus_export.sh ]; then
        deb=$(grep -c '^===== Export Incus' "$JOURNAL" 2>/dev/null)
        fin=$(grep -c '^===== Terminé' "$JOURNAL" 2>/dev/null)
        n=$(grep -c '^ÉCHEC pour' "$JOURNAL" 2>/dev/null)
    else
        deb=$(grep -c -- "^=== $nom — début" "$JOURNAL" 2>/dev/null)
        fin=$(grep -c -- "^=== $nom — fin" "$JOURNAL" 2>/dev/null)
        n=$(grep -- "^=== $nom — fin" "$JOURNAL" 2>/dev/null \
            | sed -E 's/.*(échecs|code de sortie) : //' | tail -1)
    fi
```

```
    n=${n:-0}

    if [ "$deb" -eq 0 ] && [ "$fin" -eq 0 ]; then
        ligne4 "$nom" "$heure" "ABSENTE" "aucune trace dans le journal"
        emit TACHE "$nom" "$heure" absente "aucune trace dans le journal"
        alerte "$nom n'a laissé aucune trace dans le journal"
    elif [ "$fin" -eq 0 ]; then
        ligne4 "$nom" "$heure" "INACHEVÉE" "début sans fin"
        emit TACHE "$nom" "$heure" inachevee "début sans fin"
        alerte "$nom a commencé sans se terminer"
    elif [ "$n" != "0" ]; then
        ligne4 "$nom" "$heure" "ÉCHEC" "$n échec(s)"
        emit TACHE "$nom" "$heure" echec "$n échec(s)"
        alerte "$nom : $n échec(s)"
    else
        ligne4 "$nom" "$heure" "ok" ""
        emit TACHE "$nom" "$heure" ok ""
    fi
done

# ------------------------------------------------------------------------
# 2. Fraîcheur des destinations
# ------------------------------------------------------------------------
# Le journal dit si la tâche a tourné ; il ne dit pas si elle a rapporté
# quelque chose. Une source figée produit un succès sincère sur une
# sauvegarde qui vieillit. On regarde donc la date du fichier le plus
# récent — celle du répertoire ne veut rien dire, rsync ne la touche que si
# le contenu change.
#
# L'âge se compte en jours de CALENDRIER : un écart en secondes afficherait
# « 0 j » pour un fichier d'hier soir et « 1 j » pour un fichier d'hier midi.
echo
echo "== FRAÎCHEUR DES DESTINATIONS"
col DESTINATION 44; col DERNIER 12; col ÂGE 8; echo ÉTAT
echo "$TEMOINS" | while IFS='|' read -r tolere chemin; do
    [ -n "$chemin" ] || continue
    court=${chemin#/media/nas1/Backups/}

    if [ ! -d "$chemin" ]; then
        col "$court" 44; col "-" 12; col "-" 8; echo "ABSENT"
        emit TEMOIN "$court" - - "$tolere" absent
        alerte "destination absente : $court"
```

```
        continue
    fi

    horo=$(find "$chemin" -type f -printf '%T@\n' 2>/dev/null \
           | sort -rn | head -1)
    if [ -z "$horo" ]; then
        col "$court" 44; col "-" 12; col "-" 8; echo "VIDE"
        emit TEMOIN "$court" - - "$tolere" vide
        alerte "destination vide : $court"
        continue
    fi

    date_lis=$(date -d "@${horo%.*}" +%F)
    jours=$(( (AUJOURDHUI - $(date -d "$date_lis" +%s)) / 86400 ))

    if [ "$jours" -gt "$tolere" ]; then
        col "$court" 44; col "$date_lis" 12; col "${jours} j" 8
        echo "TROP VIEUX (toléré : $tolere j)"
        emit TEMOIN "$court" "$date_lis" "$jours" "$tolere" vieux
        alerte "$court : $jours jours, toléré $tolere"
    else
        col "$court" 44; col "$date_lis" 12; col "${jours} j" 8; echo "ok"
        emit TEMOIN "$court" "$date_lis" "$jours" "$tolere" ok
    fi
done

# ---------------------------------------------------------------------
# 3. Services
# ---------------------------------------------------------------------
echo
echo "== SERVICES"
for s in $SERVICES; do
    etat=$(systemctl is-active "$s" 2>&1)
    col "$s" 24; echo "$etat"
    emit SERVICE "$s" "$etat"
    [ "$etat" = active ] || alerte "service $s : $etat"
done

echo
echo "-- conteneurs"
while IFS=, read -r nom etat; do
    [ -n "$nom" ] || continue
```

```
    col "$nom" 24; echo "$etat"
    emit CONTENEUR "$nom" "$etat"
    [ "$etat" = RUNNING ] || alerte "conteneur $nom : $etat"
done < <(incus list --format csv -c ns 2>/dev/null)

# « Actif » ne veut pas dire « en ondes ». ices2 peut tourner sans être
# connecté à Icecast : c'est la panne déjà rencontrée. On lit donc les
# points de montage réellement publiés, et on ne nomme aucune station —
# un seul dongle, les quatre unités rtl-fm s'excluent.
echo
echo "-- diffusion"
FLUX=$(curl -s --max-time 5 http://localhost:8000/status-json.xsl 2>/dev/null)
for m in /Radio_FM.ogg /Chromecast.ogg; do
    if echo "$FLUX" | grep -q "$m"; then
        col "$m" 24; echo "en ondes"
        emit FLUX "$m" "en ondes"
    else
        col "$m" 24; echo "ABSENT DU SERVEUR"
        emit FLUX "$m" absent
        alerte "flux $m non connecté à Icecast"
    fi
done
N_FM=$(systemctl list-units 'rtl-fm-*.service' --state=active \
       --no-legend --no-pager 2>/dev/null | wc -l)
col "unités rtl-fm actives" 24; echo "$N_FM (attendu : 1)"
emit RTLFM "$N_FM"
[ "$N_FM" = 1 ] || alerte "unités rtl-fm actives : $N_FM au lieu de 1"

# systemctl --failed sort en code 0 même sans rien à dire : on teste donc
# le contenu, pas le code de sortie. Une section muette serait ambiguë.
echo
echo "-- unités en échec"
ECHECS_UNITS=$(systemctl --failed --no-legend --no-pager)
if [ -n "$ECHECS_UNITS" ]; then
    echo "$ECHECS_UNITS" | sed 's/^/   /'
    while read -r u reste; do
        [ -n "$u" ] || continue
        emit UNITEECHEC "$u"
        alerte "unité en échec : $u"
    done <<< "$ECHECS_UNITS"
else
    echo "   aucune"
```

```
fi

# ------------------------------------------------------------------------
# 4. fail2ban — sur l'hôte ET dans le conteneur web
# ------------------------------------------------------------------------
# Deux machines portent des prisons, pour des raisons différentes : le
# conteneur sert des sites publics, l'hôte reçoit le SSH exposé depuis
# Internet. Les interroger ensemble est le seul moyen de n'en pas oublier
# une — ce relevé n'a longtemps connu que le conteneur, et rien ne le disait.
#
# L'écart entre les deux compteurs est le signal : des échecs qui montent
# avec zéro blocage n'est pas une panne du filtre, c'est une fenêtre de
# comptage mal réglée. Les deux se corrigent à des endroits différents.
#
# Sur l'hôte, le socket de fail2ban est en 0700 root. Ce script tourne sous
# hostadmin et passe donc par une règle sudo étroite, déposée dans
# /etc/sudoers.d/tableau-de-bord. Le « -n » interdit toute demande de mot de
# passe : sous cron, une invite resterait suspendue jusqu'au délai.
#
# Registre : machine|prison|note. « nas-host » désigne la machine locale.
PRISONS="
nas-host|sshd|SSH exposé
web-srv|wordpress-login|échecs de connexion
web-srv|wordpress-honeypot|ancienne adresse, 1 essai
web-srv|apache-401|
web-srv|sshd|réseau local seulement
"

echo
echo "== FAIL2BAN"
col MACHINE 12; col PRISON 20; col ÉCHECS 10
col BLOCAGES 10; col ACTUELS 10; echo NOTE
echo "$PRISONS" | while IFS='|' read -r machine j note; do
    [ -n "$j" ] || continue
    if [ "$machine" = nas-host ]; then
        # Les « </dev/null » ne sont pas décoratifs : incus exec LIT
        # son entrée standard. Sans ce garde, le premier appel avale le
        # reste du registre servi par le tube, et les prisons suivantes
        # disparaissent sans la moindre erreur.
        sortie=$(sudo -n fail2ban-client status "$j" 2>/dev/null </dev/null)
    else
        sortie=$(incus exec "$machine" -- fail2ban-client status "$j" \
```

```
                2>/dev/null </dev/null)
    fi

    # Une sortie vide ne veut pas dire « zéro échec », elle veut dire que
    # fail2ban n'a pas répondu. Sans ce contrôle, un service arrêté
    # s'afficherait « 0 échec, 0 blocage » — soit exactement l'apparence
    # d'une machine parfaitement protégée.
    if [ -z "$sortie" ]; then
        col "$machine" 12; col "$j" 20; col "-" 10; col "-" 10; col "-" 10
        echo "INJOIGNABLE"
        emit JAIL "$machine" "$j" - - - injoignable
        alerte "prison $machine/$j : injoignable"
        continue
    fi

    ech=$(echo "$sortie" | grep 'Total failed' | sed 's/.*:[[:space:]]*//')
    ban=$(echo "$sortie" | grep 'Total banned' | sed 's/.*:[[:space:]]*//')
    act=$(echo "$sortie" | grep 'Currently banned' | sed 's/.*:[[:space:]]*//')
    ech=${ech:-0}; ban=${ban:-0}; act=${act:-0}
    col "$machine" 12; col "$j" 20; col "$ech" 10
    col "$ban" 10; col "$act" 10; echo "$note"
    emit JAIL "$machine" "$j" "$ech" "$ban" "$act" "$note"
    # Un ou deux échecs sans blocage est le régime NORMAL : le seuil de la
    # prison reste loin, et tout va bien. Le signal utile est un compteur qui
    # monte franchement sans que rien ne soit jamais banni — dans ce cas, la
    # fenêtre de comptage est mal réglée.
    if [ "$ech" -ge 10 ] && [ "$ban" -eq 0 ]; then
        alerte "prison $machine/$j : $ech échecs, aucun blocage"
    fi
done

# ---------------------------------------------------------------------------
# 5. LanguageTool joignable depuis Internet
# ---------------------------------------------------------------------------
# Six sites sont servis par le 443 public, et cinq sont gardés : le blog est
# public par vocation, la galerie demande un mot de passe au niveau d'Apache,
# la diffusion musicale et l'administration du routeur ont chacune leur
# authentification. LanguageTool est le SEUL qui accepte et traite du texte
# sans aucune authentification.
#
# C'est donc le seul pour lequel un visiteur d'Internet est une information.
# On compte la journée d'HIER, complète : à 4 h 30 le jour en cours ne
```

```
# contient presque rien. Le journal de la veille peut avoir été mis de côté
# par la rotation, d'où la lecture des deux fichiers.
#
# Une seule requête suffit à alerter. Le seuil n'est pas là pour filtrer du
# bruit, il n'y en a pas : ce service ne reçoit que les textes de la maison.
# Le jour où quelqu'un d'autre le trouve, mieux vaut le savoir le lendemain
# qu'au centième visiteur.
#
# ATTENTION : l'adresse du routeur apparaît pour tout ce qui vise le nom
# public DEPUIS la maison — le retour en épingle masque la machine
# d'origine. Elle compte donc comme locale, ce qui est correct ici.
echo
echo "== LANGUAGETOOL DEPUIS INTERNET"
HIER=$(date -d yesterday +%d/%b/%Y)
JX=/var/log/apache2/other_vhosts_access.log
N_LT=$(incus exec web-srv -- sh -c "grep -h '\[$HIER' $JX $JX.1 2>/dev/null" \
         </dev/null \
       | awk '$1 ~ /^lt\.webadmin\.ca:/ &&
              $2 !~ /^(192\.168\.0\.|10\.6\.0\.|127\.0\.0\.1)/' \
       | wc -l)
echo "$N_LT requête(s) venue(s) d'Internet hier"
emit LT "$N_LT"
if [ "$N_LT" -gt 0 ]; then
    alerte "LanguageTool : $N_LT requête(s) venues d'Internet hier"
fi

# ------------------------------------------------------------------------
# 6. Espace et certificat
# ------------------------------------------------------------------------
# Une seule jauge : nas1 et nas2 sont des montages liés du même disque.
echo
echo "== ESPACE"
read -r taille utilise libre pct < <(df -h --output=size,used,avail,pcent \
                                        /media/nas1 | tail -1)
pct=${pct%\%}
echo "$taille total, $utilise utilisés, $libre libres — $pct %"
emit ESPACE "$taille" "$utilise" "$libre" "$pct"
[ "$pct" -ge 85 ] && alerte "disque à $pct %"

echo
echo "== CERTIFICAT *.example.com"
FIN=$(echo | openssl s_client -connect 192.168.0.6:443 \
```

```
        -servername blog.example.com 2>/dev/null \
      | openssl x509 -noout -enddate 2>/dev/null | cut -d= -f2)
if [ -n "$FIN" ]; then
    DATE_FIN=$(date -d "$FIN" +%F)
    RESTE=$(( ( $(date -d "$FIN" +%s) - $(date +%s) ) / 86400 ))
    echo "expire le $DATE_FIN  —  $RESTE jours restants"
    emit CERT "$DATE_FIN" "$RESTE"
    [ "$RESTE" -le 30 ] && alerte "certificat : $RESTE jours restants"
else
    echo "LECTURE IMPOSSIBLE"
    emit CERT - -
    alerte "certificat : lecture impossible"
fi

# ------------------------------------------------------------------------
# 7. Le verdict
# ------------------------------------------------------------------------
NB=$(grep -c '^ALERTE' "$TSV")
echo
if [ "$NB" -eq 0 ]; then
    echo "== RIEN À REGARDER"
else
    echo "== $NB POINT(S) À REGARDER"
    grep '^ALERTE' "$TSV" | cut -f2- | sed 's/^/   • /'
fi
echo
echo "données : $TSV"
```

Éprouver :

Sur `[nas-host]` :

```
bash /home/hostadmin/scripts/tableau-de-bord/releve-tableau-de-bord.sh
```

Les douze tâches doivent apparaître, les deux mensuelles marquées « prévue le 1er » les jours où elles ne sont pas dues.

Le constructeur

Sur [nas-host] — ouvrir l’éditeur :

```
nano /home/hostadmin/scripts/tableau-de-bord/build_tableau-de-bord.py
```

Contenu :

```
#!/usr/bin/env python3
"""Fabrique la page du tableau de bord à partir de donnees.tsv.

Ce script ne mesure rien et ne juge rien : il met en forme. Le jugement —
ce qui compte comme une alerte — est décidé par le relevé, et arrive ici
sous forme de lignes ALERTE.

Il tient aussi l'historique : chaque passage ajoute l'état du jour à
historique.tsv, qui alimente la bande de trente jours. C'est de la tenue de
registre, pas de la mesure — et ça évite de relire trente journaux à chaque
fois.

La page est autonome : aucune requête sortante, aucune police distante,
aucune bibliothèque.
"""

import html
import sys
from datetime import date, timedelta
from pathlib import Path

DOSSIER = Path("/media/nas1/Backups/Tableau de bord - nas-host")
SOURCE = DOSSIER / "donnees.tsv"
HISTORIQUE = DOSSIER / "historique.tsv"
SORTIE = DOSSIER / "tableau-de-bord.html"

JOURS_BANDE = 30
JOURS_GARDE = 60


def lire(chemin):
```

```
    """Range les lignes du TSV par type : {'TACHE': [[...], ...], ...}"""
    donnees = {}
    for ligne in chemin.read_text(encoding="utf-8").splitlines():
        if not ligne.strip():
            continue
        champs = ligne.split("\t")
        donnees.setdefault(champs[0], []).append(champs[1:])
    return donnees


def historiser(d):
    """Enregistre l'état du jour et rend {tâche: {jour: état}}.

    Relancer le script le même jour REMPLACE les lignes du jour plutôt que
    de les empiler : le lanceur peut être cliqué dix fois sans fausser la
    bande. Au-delà de JOURS_GARDE, les lignes sont oubliées.
    """
    meta = dict(tuple(m) for m in d.get("META", []))
    jour = meta.get("genere", "")[:10] or date.today().isoformat()
    limite = (date.today() - timedelta(days=JOURS_GARDE)).isoformat()

    lignes = []
    if HISTORIQUE.exists():
        for l in HISTORIQUE.read_text(encoding="utf-8").splitlines():
            if l.strip():
                c = l.split("\t")
                if len(c) == 3 and c[0] >= limite and c[0] != jour:
                    lignes.append(c)

    for nom, _heure, etat, _detail in d.get("TACHE", []):
        lignes.append([jour, nom, etat])

    lignes.sort()
    HISTORIQUE.write_text(
        "\n".join("\t".join(l) for l in lignes) + "\n", encoding="utf-8")

    hist = {}
    for j, nom, etat in lignes:
        hist.setdefault(nom, {})[j] = etat
    return hist
```

```
def e(texte):
    return html.escape(str(texte), quote=True)


def pastille(texte, classe):
    return f'<span class="chip {classe}">{e(texte)}</span>'


ETATS_TACHE = {
    "ok": ("ok", "ok"),
    "mensuelle": ("mensuelle", "neutre"),
    "absente": ("absente", "crit"),
    "inachevee": ("inachevée", "crit"),
    "echec": ("échec", "crit"),
}
ETATS_TEMOIN = {
    "ok": ("à jour", "ok"),
    "vieux": ("trop vieux", "crit"),
    "absent": ("absent", "crit"),
    "vide": ("vide", "crit"),
}
# Une tâche mensuelle non due ce jour-là n'est pas un trou : elle se rend
# comme une case vide, au même titre qu'un jour sans relevé.
CASES = {
    "ok": "c-ok",
    "echec": "c-crit",
    "absente": "c-crit",
    "inachevee": "c-crit",
}


def bande(hist, nom):
    """Trente cases, du plus ancien à gauche au jour même à droite."""
    largeur = JOURS_BANDE * 12 - 2
    out = [f"<svg class='bande' viewBox='0 0 {largeur} 10' "
           "preserveAspectRatio='none' role='img'>"]
    for k in range(JOURS_BANDE):
        j = (date.today() - timedelta(days=JOURS_BANDE - 1 - k)).isoformat()
        etat = hist.get(nom, {}).get(j)
        out.append(f"<rect class='{CASES.get(etat, 'c-vide')}' x='{k * 12}' "
                   f"y='0' width='10' height='10' rx='2'>"
                   f"<title>{j}</title></rect>")
```

```
    out.append("</svg>")
    return "".join(out)


# ATTENTION : une valeur CSS ne doit JAMAIS être coupée en deux lignes. Une
# chaîne entre guillemets contenant un saut de ligne est invalide, la
# déclaration est rejetée en silence, et le texte retombe sur la police par
# défaut du navigateur.
CSS = """
:root{
  --ground:#dfe6e8; --surface:#fff; --surface-2:#eaf1f2; --line:#c2ced1;
  --ink:#14191b; --ink-2:#4d585c; --ink-3:#79868b;
  --accent:#16626d; --ok:#2c6e4a; --warn:#8a6100; --crit:#9d2f2f;
  --ok-bg:#dcecE3; --warn-bg:#f6edd8; --crit-bg:#f7e0de;
  --f-titre:"Ubuntu Condensed","DejaVu Sans Condensed",system-ui,sans-serif;
  --f-texte:"Ubuntu","Noto Sans","DejaVu Sans",system-ui,sans-serif;
  --f-mono:ui-monospace,"DejaVu Sans Mono","Liberation Mono",monospace;
}
@media (prefers-color-scheme:dark){:root:not([data-theme=light]){
  --ground:#0d1113; --surface:#191f21; --surface-2:#222a2c; --line:#333d40;
  --ink:#e7edee; --ink-2:#a4b1b5; --ink-3:#78868b;
  --accent:#5fb9c6; --ok:#63b98a; --warn:#d3a441; --crit:#e0736f;
  --ok-bg:#17301f; --warn-bg:#2c2416; --crit-bg:#341d1d;
}}
*{box-sizing:border-box}
body{margin:0;background:var(--ground);color:var(--ink);
     line-height:1.5}
.wrap{max-width:1000px;margin:0 auto;padding:26px 20px 56px;
      display:flex;flex-direction:column;gap:20px}
h1{font-size:34px;margin:0;letter-spacing:-.01em}
.entete{display:flex;flex-wrap:wrap;align-items:baseline;gap:8px 16px;
        border-bottom:2px solid var(--ink);padding-bottom:12px}
.entete .quand{margin-left:auto;font-size:13px;
               color:var(--ink-2)}
.entete .quand.vieux{color:var(--warn);font-weight:600}
.verdict{border-radius:4px;padding:16px 20px;border:1px solid}
.verdict.calme{background:var(--ok-bg);border-color:var(--ok);color:var(--ok)}
.verdict.alerte{background:var(--crit-bg);border-color:var(--crit)}
.verdict h2{font-size:22px;margin:0}
.verdict ul{margin:10px 0 0;padding-left:20px;color:var(--ink)}
.verdict li{margin:3px 0}
section.bloc{background:var(--surface);border:1px solid var(--line);
```

```
            border-radius:4px;overflow:hidden}
section.bloc>h2{font-size:19px;margin:0;
               padding:12px 18px;background:var(--surface-2);
               border-bottom:1px solid var(--line)}
.corps{padding:16px 18px;display:flex;flex-direction:column;gap:22px}
h3{font-size:11px;letter-spacing:.12em;
   text-transform:uppercase;color:var(--ink-3);margin:0 0 10px}
table{width:100%;border-collapse:collapse;font-size:14px}
th{text-align:left;font-size:10.5px;
   letter-spacing:.1em;text-transform:uppercase;color:var(--ink-3);
   padding:0 10px 6px 0;border-bottom:1px solid var(--line)}
th.num{text-align:right}
td{padding:7px 10px 7px 0;border-bottom:1px solid var(--line)}
tr:last-child td{border-bottom:none}
td.mono{font-size:13px;white-space:nowrap}
td.num{font-variant-numeric:tabular-nums;
       text-align:right;white-space:nowrap}
td.plein{width:100%}
.chip{display:inline-block;font-size:10.5px;
      letter-spacing:.06em;text-transform:uppercase;padding:2px 7px;
      border-radius:3px;border:1px solid}
.chip.ok{color:var(--ok);background:var(--ok-bg);border-color:var(--ok)}
.chip.crit{color:var(--crit);background:var(--crit-bg);
           border-color:var(--crit)}
.chip.warn{color:var(--warn);background:var(--warn-bg);
           border-color:var(--warn)}
.chip.neutre{color:var(--ink-3);background:transparent;
             border-color:var(--line)}
.duo{display:grid;grid-template-columns:repeat(auto-fit,minmax(260px,1fr));
     gap:22px}
.grand{font-size:44px;line-height:1;
       font-variant-numeric:tabular-nums}
.note{font-size:13px;color:var(--ink-3)}
footer{font-size:12.5px;color:var(--ink-3);border-top:1px solid var(--line);
       padding-top:12px}

/* Jauge et bande sont dessinées en SVG et non avec un fond CSS : un objet
   de contenu s'imprime toujours, un arrière-plan non — les navigateurs les
   suppriment par défaut, et la case « imprimer les arrière-plans » de leur
   boîte de dialogue prime sur print-color-adjust. */
.jauge{display:block;width:100%;height:14px}
.j-fond{fill:none;stroke:var(--accent);stroke-width:1}
```

```
.j-plein{fill:var(--accent)}
.bande{display:block;width:100%;height:13px}
.c-ok{fill:var(--ok)}
.c-crit{fill:var(--crit)}
.c-vide{fill:var(--line)}

@media print{
  :root{--ground:#fff; --surface:#fff; --surface-2:#f0f0f0;}
  :root{--line:#b8c4c7; --ink:#000; --ink-2:#333; --ink-3:#555;}
  *{print-color-adjust:exact; -webkit-print-color-adjust:exact}
  body{background:#fff}
  .wrap{max-width:none;padding:0}
  section.bloc,table,.verdict,.duo>div{break-inside:avoid}
}
"""



def construire(d, hist):
    meta = dict(tuple(m) for m in d.get("META", []))
    alertes = [a[0] for a in d.get("ALERTE", [])]
    p = []
    A = p.append

    A("<!doctype html>")
    A('<html lang="fr"><head><meta charset="utf-8">')
    A('<meta name="viewport" content="width=device-width,initial-scale=1">')
    A("<title>Tableau de bord — nas-host</title>")
    A(f"<style>{CSS}</style></head><body><div class='wrap'>")

    A("<div class='entete'><h1>Tableau de bord — nas-host</h1>")
    A(f"<div class='quand' id='quand' data-genere='{e(meta.get('genere',''))}'>"
      f"relevé du {e(meta.get('genere','?')[:16].replace('T',' '))}</div>")
    A("</div>")

    if alertes:
        A("<div class='verdict alerte'>")
        n = len(alertes)
        A(f"<h2>{n} point{'s' if n > 1 else ''} à regarder</h2><ul>")
        for a in alertes:
            A(f"<li>{e(a)}</li>")
        A("</ul></div>")
    else:
```

```
    A("<div class='verdict calme'><h2>Rien à regarder</h2></div>")

# --- Sauvegardes ---
A("<section class='bloc'><h2>Sauvegardes</h2><div class='corps'>")
A("<div><h3>La nuit écoulée</h3><table><tr><th>Tâche</th>"
  "<th class='num'>Prévue</th><th>État</th><th>Détail</th></tr>")
for nom, heure, etat, detail in d.get("TACHE", []):
    libelle, classe = ETATS_TACHE.get(etat, (etat, "neutre"))
    A(f"<tr><td class='mono'>{e(nom)}</td><td class='num'>{e(heure)}</td>"
      f"<td>{pastille(libelle, classe)}</td><td>{e(detail)}</td></tr>")
A("</table></div>")

A("<div><h3>Trente jours</h3>"
  "<p class='note'>Du plus ancien à gauche au jour même à droite. Une "
  "case pâle signifie qu'aucun relevé n'existe pour ce jour — "
  "l'historique se remplit à partir d'aujourd'hui. Les tâches "
  "mensuelles n'y marquent que le 1er.</p><table>")
for nom, _h, _e, _d in d.get("TACHE", []):
    A(f"<tr><td class='mono'>{e(nom)}</td>"
      f"<td class='plein'>{bande(hist, nom)}</td></tr>")
A("</table></div>")

A("<div><h3>Fraîcheur des destinations</h3>"
  "<p class='note'>La date du fichier le plus récent, et non celle du "
  "répertoire : rsync ne touche cette dernière que si le contenu change. "
  "Chaque destination porte son propre âge toléré.</p>"
  "<table><tr><th>Destination</th><th class='num'>Dernier</th>"
  "<th class='num'>Âge</th><th class='num'>Toléré</th>"
  "<th>État</th></tr>")
for court, dat, jours, tolere, etat in d.get("TEMOIN", []):
    libelle, classe = ETATS_TEMOIN.get(etat, (etat, "neutre"))
    A(f"<tr><td class='mono'>{e(court)}</td><td class='num'>{e(dat)}</td>"
      f"<td class='num'>{e(jours)} j</td><td class='num'>{e(tolere)} j</td>"
      f"<td>{pastille(libelle, classe)}</td></tr>")
A("</table></div></div></section>")

# --- Services ---
A("<section class='bloc'><h2>Services</h2><div class='corps'>")
A("<div class='duo'>")

A("<div><h3>Unités</h3><table>")
for nom, etat in d.get("SERVICE", []):
```

```
        c = "ok" if etat == "active" else "crit"
        A(f"<tr><td class='mono'>{e(nom)}</td>"
          f"<td>{pastille(etat, c)}</td></tr>")
    A("</table></div>")

    A("<div><h3>Conteneurs</h3><table>")
    for nom, etat in d.get("CONTENEUR", []):
        c = "ok" if etat == "RUNNING" else "crit"
        A(f"<tr><td class='mono'>{e(nom)}</td>"
          f"<td>{pastille(etat, c)}</td></tr>")
    A("</table></div>")
    A("</div>")

    A("<div><h3>Diffusion</h3>"
      "<p class='note'>Une unité active ne prouve pas qu'un flux est en "
      "ondes : on lit les points de montage réellement publiés par Icecast. "
      "Un seul dongle, donc exactement une unité rtl-fm attendue — quelle "
      "que soit la station.</p><table>")
    for mont, etat in d.get("FLUX", []):
        c = "ok" if etat == "en ondes" else "crit"
        A(f"<tr><td class='mono'>{e(mont)}</td>"
          f"<td>{pastille(etat, c)}</td></tr>")
    for (n,) in d.get("RTLFM", []):
        c = "ok" if n == "1" else "crit"
        A(f"<tr><td class='mono'>unités rtl-fm actives</td>"
          f"<td>{pastille(n + ' (attendu : 1)', c)}</td></tr>")
    A("</table></div>")

    ue = d.get("UNITEECHEC", [])
    A("<div><h3>Unités en échec</h3>")
    if ue:
        A("<table>")
        for (u,) in ue:
            A(f"<tr><td class='mono'>{e(u)}</td>"
              f"<td>{pastille('en échec', 'crit')}</td></tr>")
        A("</table>")
    else:
        A(f"<p>{pastille('aucune', 'ok')}</p>")
    A("</div></div></section>")

    # --- Système ---
    A("<section class='bloc'><h2>Système</h2><div class='corps'>")
```

```
    A("<div class='duo'>")

    for taille, _utilise, libre, pct in d.get("ESPACE", []):
        try:
            large = min(99.0, float(pct) * 0.99)
        except ValueError:
            large = 0.0
        A("<div><h3>Espace</h3>"
          f"<div class='grand'>{e(pct)} %</div>"
          "<svg class='jauge' viewBox='0 0 100 10' "
          f"preserveAspectRatio='none' role='img' aria-label='{e(pct)} %'>"
          "<rect class='j-fond' x='.5' y='.5' width='99' height='9' rx='4.5'/>"
          f"<rect class='j-plein' x='.5' y='.5' width='{large:.1f}' "
          "height='9' rx='4.5'/></svg>"
          f"<p class='note'>{e(libre)} libres sur {e(taille)}. "
          "Une seule jauge : nas1 et nas2 sont des montages liés du même "
          "disque.</p></div>")

    for date_fin, reste in d.get("CERT", []):
        A("<div><h3>Certificat générique</h3>"
          f"<div class='grand'>{e(reste)}</div>"
          f"<p class='note'>jours avant le {e(date_fin)}. Renouvellement "
          "manuel : rien sur la machine ne s'en charge.</p></div>")

    for (n_lt,) in d.get("LT", []):
        A("<div><h3>LanguageTool depuis Internet</h3>"
          f"<div class='grand'>{e(n_lt)}</div>"
          "<p class='note'>requêtes venues d'ailleurs que du réseau local et "
          "du tunnel, hier. Seul des six sites publics à n'exiger aucune "
          "authentification : toute valeur autre que zéro mérite qu'on le "
          "restreigne.</p></div>")
    A("</div>")

    A("<div><h3>fail2ban</h3>"
      "<p class='note'>Trois nombres de portées différentes : échecs et "
      "blocages comptent depuis le démarrage du service, "
      "« actuels » dit ce qui est bloqué à l'instant. Des échecs qui montent "
      "sans aucun blocage signalent un seuil hors d'atteinte.</p>"
      "<table><tr><th>Machine</th><th>Prison</th>"
      "<th class='num'>Échecs</th>"
      "<th class='num'>Blocages</th><th
class='num'>Actuels</th><th>Note</th></tr>")
```

```
    for machine, nom, ech, ban, act, note in d.get("JAIL", []):
        A(f"<tr><td class='mono'>{e(machine)}</td>"
          f"<td class='mono'>{e(nom)}</td><td class='num'>{e(ech)}</td>"
          f"<td class='num'>{e(ban)}</td><td class='num'>{e(act)}</td>"
          f"<td class='note'>{e(note)}</td></tr>")
    A("</table></div></div></section>")

    A(f"<footer>Page fabriquée à partir de {e(SOURCE.name)} et de "
      f"{e(HISTORIQUE.name)}, produits par le relevé sur nas-host. "
      "Aucune requête sortante.</footer>")
    A("</div>")

    A("""<script>
(function(){
  var q=document.getElementById('quand'), iso=q&&q.dataset.genere;
  if(iso){
    var h=(Date.now()-new Date(iso).getTime())/3600000;
    q.textContent='relevé du '+iso.slice(0,16).replace('T',' ')
                +' · '+Math.round(h)+' h';
    if(h>36) q.classList.add('vieux');
  }
})();
</script>""")
    A("</body></html>")
    return "\n".join(p)


def main():
    if not SOURCE.exists():
        sys.exit(f"ÉCHEC : {SOURCE} introuvable — lancer le relevé d'abord.")
    d = lire(SOURCE)
    hist = historiser(d)
    SORTIE.write_text(construire(d, hist), encoding="utf-8")
    print(f"page écrite : {SORTIE}")
    print(f"historique  : {HISTORIQUE} ({len(hist)} tâches suivies)")


if __name__ == "__main__":
    main()
```

Fabriquer la page :

Sur `[nas-host]` :

```
python3 /home/hostadmin/scripts/tableau-de-bord/build_tableau-de-bord.py
```

Le lanceur sur le poste

Sur `[poste-bureau]` — ouvrir l'éditeur :

```
mkdir -p ~/.local/bin ~/.local/share/applications && \
nano ~/.local/bin/tableau-de-bord
```

Contenu :

```
#!/bin/bash
# Ouvre le tableau de bord de nas-host, après avoir tenté de le rafraîchir.
#
# Le rafraîchissement est une commodité, pas une condition : s'il échoue,
# la page s'ouvre avec les données du dernier passage. Son en-tête indique
# l'âge du relevé et passe à l'ambre au-delà de 36 h.

PAGE="/media/nas1/Backups/Tableau de bord - nas-host/tableau-de-bord.html"
CIBLE=hostadmin@192.168.0.11
DISTANT=/home/hostadmin/scripts/tableau-de-bord

if ! ssh -o BatchMode=yes -o ConnectTimeout=5 "$CIBLE" \
        "bash $DISTANT/releve-tableau-de-bord.sh >/dev/null &&
         python3 $DISTANT/build_tableau-de-bord.py >/dev/null" 2>/dev/null
then
    notify-send -i dialog-warning "Tableau de bord" \
        "Rafraîchissement impossible — affichage du dernier passage." \
        2>/dev/null
fi

if [ -f "$PAGE" ]; then
    exec xdg-open "$PAGE"
else
    notify-send -i dialog-error "Tableau de bord" \
        "Page introuvable — /media/nas1 est-il monté ?" 2>/dev/null
```

```
    exit 1
fi
```

Le rafraîchissement suppose une clé SSH vers `hostadmin@192.168.0.11`. Sans elle, la notification s'affiche et la page s'ouvre quand même. Pour l'établir, une seule fois :

```
ssh-copy-id hostadmin@192.168.0.11
```

Sur `[poste-bureau]` — le raccourci :

```
nano ~/.local/share/applications/tableau-de-bord.desktop
```

Contenu :

```
[Desktop Entry]
Type=Application
Name=Tableau de bord — nas-host
Comment=État des sauvegardes et des services
Exec=/home/user/.local/bin/tableau-de-bord
Icon=utilities-system-monitor
Terminal=false
Categories=System;Monitor;
```

Le chemin de `Exec` doit être absolu : le tilde n'est pas développé dans un fichier `.desktop`.

Sur `[poste-bureau]` — rendre exécutable, poser sur le bureau, autoriser :

```
chmod +x ~/.local/bin/tableau-de-bord
cp ~/.local/share/applications/tableau-de-bord.desktop ~/Bureau/
chmod +x ~/Bureau/tableau-de-bord.desktop
gio set ~/Bureau/tableau-de-bord.desktop metadata::trusted true
update-desktop-database ~/.local/share/applications 2>/dev/null
```

⚠ Cinnamon exige les deux dernières opérations. Sans le bit exécutable et sans la marque de confiance, l’icône affiche le nom du fichier et refuse de se lancer au double-clic.

Vérifier

L’essai qui compte n’est pas le lancement manuel mais celui en environnement dépouillé : cron n’a ni le `PATH` de la session, ni ses variables, ni sa locale.

Sur `[nas-host]` :

```
env -i HOME=/home/hostadmin PATH=/usr/bin:/bin /bin/bash -c \
  'cd /home/hostadmin/scripts/tableau-de-bord &&
   bash releve-tableau-de-bord.sh >/dev/null &&
   python3 build_tableau-de-bord.py'
```

Un code de sortie nul et deux lignes de confirmation. Un `command not found` sur `incus`, `curl` ou `openssl` trahirait un `PATH` insuffisant.

Relancer ce test ne fausse rien : l’historique remplace les lignes du jour au lieu de les empiler.

Le tableau surveille sa propre fraîcheur. L’en-tête recalcule l’âge du relevé à l’ouverture et passe à l’ambre au-delà de trente-six heures. Si l’ordonnancement cesse, la page le dit d’elle-même — c’est la seule surveillance dont le dispositif ait besoin pour lui-même.

Régler les seuils

Les âges tolérés sont des valeurs de départ. Une destination qui se plaint sans raison n’a pas un défaut de sauvegarde : elle a un seuil trop serré pour ce qu’elle contient. `incus-exports` est mensuel, et l’archive de la clé d’amorçage ne change qu’au gré du système.

Ils se modifient dans le bloc `TEMOINS` du relevé, un nombre de jours par ligne.

 Une mesure peut exister pour ne rien dire pendant des années, et c’est son intérêt. La dernière section du relevé compte les requêtes venues d’Internet vers le seul site public que rien ne protège par mot de passe. Elle affiche zéro, et affichera zéro tant que personne ne l’aura trouvé. Le jour où ce n’est plus zéro, la décision de fermer ce service se prend sur un fait plutôt que sur une crainte — et elle se prend le lendemain, pas au centième visiteur.

⚠ Ce zéro doit rester visible sur la page, et pas seulement déclencher une alerte. Une mesure qui ne parle que lorsqu’elle se déclenche est indiscernable d’une mesure morte : « aucune alerte » et « le relevé ne mesure plus rien » ont exactement la même apparence. Le chiffre affiché est ce qui prouve que le contrôle tourne encore.

 Un seuil mal posé transforme un tableau de veille en bruit de fond. Le relevé signalait d’abord toute prison affichant des échecs sans aucun blocage — ce qui paraît sensé et ne l’est pas : une prison qui compte deux échecs avec un seuil à cinq fonctionne exactement comme prévu. La règle aurait produit un point à regarder chaque matin, pour rien. Le signal utile est un compteur qui monte franchement sans que rien ne soit jamais banni ; le seuil est donc à dix, et la nuance est écrite dans le script. Les trois nombres du panneau n’ont d’ailleurs pas la même portée : échecs et blocages comptent depuis le démarrage du service — bannissements rétablis depuis la base compris —, tandis qu’« actuels » dit ce qui est bloqué à l’instant. Les comparer terme à terme fait conclure à une panne là où la prison travaille.

 Mais avant de régler un seuil, il faut savoir si le témoin peut mesurer quelque chose. `rsync -a` préserve les dates de la source : la fraîcheur d’un répertoire de configurations miroir ne dit pas si le tirage fonctionne, elle dit quand tu as utilisé le logiciel pour la dernière fois. Un témoin posé sur `Configuration_logiciels/` affichait « ok » parce que deux applications avaient bougé la veille, et serait passé au rouge un mois tranquille où tout allait bien. Il a été retiré : aucun seuil ne pouvait le rendre juste.

 Un témoin de fraîcheur ne vaut que là où le fichier est refabriqué, pas recopié. Les vidages de bases, l’archive du filtre DNS, la copie SQLite du serveur de musique, les exports mensuels : ceux-là portent la date de la tâche.

Un miroir rsync de fichiers qui ne changent pas porte la date de leur dernière modification, et ce n’est pas la même question. Les miroirs de configurations et de scripts n’ont donc aucun témoin : leur exécution se lit dans le registre des tâches, et nulle part ailleurs.

⚠ Retirer un témoin ne retire aucune surveillance de la tâche elle-même : son exécution reste lue dans le registre des tâches, par les marqueurs du journal. Ce qu’on perd est un indicateur qui racontait autre chose que ce qu’on y lisait — et c’est un gain, selon la doctrine des *trois contrôles qui n’en sont pas*.

⚠ Deux tâches ne doivent jamais écrire dans un même répertoire surveillé. Le témoin prend le fichier le plus récent de toute l’arborescence : la plus assidue des deux maintient le témoin au vert pour les deux, et la défaillance de l’autre devient invisible.

De même, le seuil de remplissage du disque est à 85 % et celui du certificat à trente jours, tous deux dans la dernière section du relevé.

Ce qu’il faut savoir avant de modifier

Le registre des tâches est la source de vérité. Ajouter une tâche de sauvegarde à la crontab sans l’ajouter au bloc `TACHES` la rend invisible au tableau — et une tâche absente du registre ne sera jamais signalée absente.

Trois formats de marqueurs cohabitent dans le journal. La forme courante est `=== <script> — fin <date> — échecs : N`. Mais `upload_photos_photos-webadmin-ca.sh` termine par `— code de sortie : N`, et `backup_incus_export.sh` emploie `===== Export Incus` et `===== Terminé`, sans compteur. Un analyseur qui ne chercherait que `— échecs :` déclarerait ces deux-là absentes chaque nuit.

Un journal peut peser plusieurs mégaoctets une nuit de gros transfert, `rsync -v` nommant chaque fichier transféré. Filtrer les lignes `^===` avec `grep` ; ne jamais charger un journal entier en mémoire.

 Une valeur CSS ne se coupe jamais en deux lignes. Une chaîne entre guillemets contenant un saut de ligne est invalide : la déclaration entière est rejetée

sans aucun message, et le texte retombe sur la police par défaut du navigateur.

Jauge et bande sont dessinées en SVG, pas avec un fond CSS. Un objet de contenu s’imprime toujours ; un arrière-plan non — les navigateurs les suppriment par défaut, et la case « imprimer les arrière-plans » de leur boîte de dialogue prime sur `print-color-adjust`.

`printf` du shell complète en octets, pas en caractères. Un « é » en occupe deux et décale la colonne d’autant. En locale UTF-8, `${#texte}` compte bien les caractères : c’est le rôle de la fonction `col` du relevé.

Les boucles alimentées par un tube tournent dans un sous-processus. Une variable de comptage incrémentée dedans est perdue à la sortie. C’est pourquoi les alertes sont écrites dans le fichier de données et comptées ensuite en le relisant.

 Une commande lancée dans une boucle `while read` peut manger le reste du registre. `incus exec` lit son entrée standard — celle-là même qui sert à alimenter la boucle. Sans `