Piwigo

Référence : Piwigo 16.4 — Ubuntu 24.04, PHP 8.3, ISPConfig

Guide de reconstruction. Il décrit comment rebâtir une galerie photo privée, du site vide jusqu'à la synchronisation nocturne qui l'alimente.

⚠ Une seule ligne de ce document dépasse 84 caractères et s'enroulera dans le PDF sans signe visible : la ligne de cron du téléversement nocturne. Cron n'accepte aucune continuation — la recopier depuis le fichier `.md`, jamais depuis le PDF.

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

Ce que cette page couvre : installer et configurer Piwigo sur un serveur auto-hébergé administré par ISPConfig, l'alimenter depuis un NAS plutôt que par le navigateur, et fermer les accès que l'application ne ferme pas d'elle-même.

Contexte : Piwigo 16.4 dans le conteneur `web-srv` sous Ubuntu 24.04.5, servi par Apache et PHP 8.3 en FPM, base MariaDB 10.11. La galerie compte 17 642 photos réparties en 547 albums, 471 mots-clés et 4 comptes, pour 50 Go d'originaux et 8,7 Go de vignettes calculées. Elle est entièrement privée : rien n'y est visible sans authentification. Les adresses, noms de domaine et noms de comptes sont des exemples : les transposer à son propre parc.

Trois choses valent le détour même sans suivre le guide de bout en bout.

Piwigo protège ses pages, pas ses fichiers. Dans sa configuration par défaut, n'importe qui peut télécharger une photo d'un album privé en devinant son adresse, et cela n'a rien d'un défaut d'installation — section *Fermer l'accès direct aux fichiers*.

Le correctif que l'on trouve le plus souvent en forum pour ce problème n'en est pas un. Il déplace l'adresse sans ajouter le moindre contrôle, ce que seule la lecture du code source révèle — même section, sous-section *La fausse bonne*

idée.

Un champ de métadonnées écrit par les téléphones Samsung interrompt la synchronisation avec une erreur de base de données incompréhensible, et l'outil habituel de diagnostic ne voit rien du problème parce qu'il fait un travail que PHP ne fait pas — section *Dépannage*.

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

Ce que cette galerie est, et comment elle est alimentée

Piwigo est une galerie photo qui se sert d'un répertoire du disque comme source. Elle n'impose pas de déposer les photos par son interface : elle sait lire une arborescence existante et en tirer ses albums.

C'est le modèle retenu ici, et il gouverne tout le reste :

```
NAS (source des photos)
  │
  │  rsync nocturne, module en écriture
  ▼
galleries/ du site web
  │
  │  synchronisation depuis l'administration de Piwigo
  ▼
base de données — albums, mots-clés, dates
```

La source fait autorité, jamais la copie. Le répertoire `galleries/` du serveur est un miroir : tout ce qu'on y modifie directement est défait à la synchronisation suivante. Un fichier à renommer se renomme sur le NAS.

Deux conséquences pratiques. La première est que la galerie ne se répare pas côté serveur — elle se répare à la source. La seconde est que le lien qui relie les deux est un accès en écriture vers la racine web d'un site en ligne, ce qui mérite d'être traité avec soin ; c'est l'objet d'une section entière.

Le conteneur qui héberge tout cela — Apache, ISPConfig, le multi-PHP, les certificats, le démon rsync — est décrit dans Serveur web — Apache multi-PHP et ISPConfig. Cette page-ci suppose ce serveur en état de marche et ne décrit que la galerie.

Trois répertoires de la racine web n'appartiennent pas à Piwigo et ne doivent pas inquiéter : `stats/`, écrit par ISPConfig pour les statistiques de fréquentation du site, `error/` pour ses pages d'erreur, et `webdav/` s'il est activé. Ils vivent à côté de l'application, jamais dans `galleries/`, et n'entrent donc dans aucune synchronisation.

Le site dans ISPConfig

La galerie est un site comme un autre dans le panneau : Sites → Ajouter un site, avec son propre nom de domaine, son propre compte système et sa propre base de données.

Les réglages qui comptent, dans l'onglet Domaine :

```
Nom de domaine          photos.example.com
Version PHP             PHP 8.3
Mode PHP                PHP-FPM
```

⚠ PHP-FPM, pas Fast-CGI. Le choix se fait ici une fois pour toutes et il conditionne la section suivante : sous FPM, les réglages de PHP ne vivent pas là où la plupart des guides les cherchent.

⚠ Piwigo 16 exige PHP 8.2 au minimum. PHP 7.4 continue de fonctionner en

apparence, mais Piwigo signale le risque de vulnérabilités, et surtout rien ne vérifie ce minimum au démarrage : l'application ne s'en aperçoit qu'en atteignant une fonction précise, parfois des mois plus tard, au milieu d'une opération.

Savoir quelle version sert réellement la galerie

`php -v` ne répond pas à la question. Il donne la version de la ligne de commande, qui n'a aucun rapport : sous ISPConfig, chaque site choisit la sienne. La réponse est dans le nom du répertoire qui porte le pool FPM du site.

`[dans web-srv]` :

```
ls -1 /etc/php/*/fpm/pool.d/web2.conf
```

Le chemin retourné contient le numéro de version — ici `/etc/php/8.3/fpm/pool.d/web2.conf`.

`[dans web-srv]` — et la preuve définitive, qui montre quel processus détient le socket du site :

```
ss -xlp | grep web2.sock
```

Attendu : `users:(("php-fpm8.3",...))`.

Le répertoire des sockets porte un nom de version qui n'est pas la bonne, et c'est délibéré : ISPConfig les range tous au même endroit, celui de sa version par défaut d'installation. Voir `/var/lib/php8.1-fpm/web2.sock` pour un site en PHP 8.3 n'est donc pas le signe d'une configuration en retard. L'explication complète est dans Serveur web, section *Les trois pièges du multi-PHP*.

Changer la version de PHP d'une galerie existante

Le vrai risque d'un changement de version n'est pas la version elle-même : c'est un module présent d'un côté et absent de l'autre. Piwigo dépend de `exif`

pour lire les métadonnées et de `gd` pour calculer les vignettes ; leur absence ne se voit qu'à la première synchronisation.

`[dans web-srv]` — comparer avant de basculer :

```
php8.3 -m > /tmp/m83 ; php8.4 -m > /tmp/m84 ; diff /tmp/m83 /tmp/m84
```

Les lignes précédées de `<` sont les modules absents de la nouvelle version — ce sont ceux à installer.

Après la bascule dans le panneau, compter une minute : ISPConfig n'écrit pas au moment du clic, c'est sa tâche cron qui régénère le pool et recharge le service. Vérifier ensuite la galerie, l'administration, une synchronisation complète — c'est elle qui sollicite `exif` et `gd` — et le chargement des albums depuis l'application mobile.

⚠ Une erreur 503 juste après une bascule de version ne vient pas de Piwigo. Elle signifie qu'Apache ne joint pas le socket PHP-FPM du site, presque toujours parce que le service de la nouvelle version n'a pas été rechargé. Le diagnostic complet, et la cause racine qui le fait revenir à chaque changement, sont dans l'article du serveur web.

Les limites d'envoi

Elles se posent dans Sites → le site → onglet Options → Custom php.ini settings :

```
upload_max_filesize = 200M
post_max_size = 200M
memory_limit = 400M
allow_url_fopen = On
max_execution_time = 300
```

Le raisonnement derrière ces nombres :

- `memory_limit` au double de la taille des fichiers traités, parce que le redimensionnement d’une image consomme bien plus que le poids du fichier. Le rapport de deux pour un n’est pas arbitraire.

- `post_max_size` doit rester supérieur ou égal à `upload_max_filesize`, sinon la limite annoncée n’est pas celle qui s’applique.

- ⚠ `post_max_size` plafonne l’envoi entier, pas chaque fichier. Sur un dépôt par lots — une galerie, typiquement — c’est le lot qui bloque en premier, jamais la photo isolée.

- `max_execution_time` à 300 secondes parce que la synchronisation d’une bibliothèque volumineuse dépasse largement les 30 secondes par défaut, et s’interrompt sans message clair quand elle les atteint.

⚠ Ne pas ajouter `FcgidMaxRequestLen` ici. C’est une directive de `mod_fcgid`, l’ancien mode d’exécution, et elle n’a strictement aucun effet sous PHP-FPM. Elle se retrouve pourtant dans beaucoup de configurations, recopiée d’un guide écrit pour l’autre mode. ISPConfig l’écrit fidèlement dans le pool, PHP l’ignore, et elle donne l’impression durable qu’un réglage est en place.

`[dans web-srv]` — vérifier ce qui a réellement été écrit, une minute après avoir enregistré :

```
grep -E 'php_admin' /etc/php/8.3/fpm/pool.d/web2.conf
```

C’est le fichier que PHP-FPM lit vraiment. Le champ du panneau n’est qu’une intention tant que le pool n’a pas été régénéré.

Installer Piwigo

La base de données

Elle se crée dans le panneau — Sites → Base de données → Ajouter — avec son propre utilisateur, distinct de celui des autres sites.

```
Nom de la base       c1photos
Utilisateur          un compte dédié à cette base
Jeu de caractères    utf8
```

Un compte de base par application. Le mot de passe de la galerie ne doit pas ouvrir la base du blogue : c'est ce qui borne les dégâts d'une faille applicative à une seule application.

Les fichiers

`[dans web-srv]` :

```
cd /var/www/clients/client1/web2/web
```

`[dans web-srv]` :

```
wget https://piwigo.org/download/dlcounter.php?code=latest -O piwigo.zip
unzip piwigo.zip
```

L'archive se décompresse dans un sous-répertoire `piwigo/` : en déplacer le contenu à la racine du site, puis retirer le répertoire vide et l'archive.

`[dans web-srv]` — rendre les fichiers au compte du site :

```
chown -R web2:client1 /var/www/clients/client1/web2/web
```

⚠ Ce `chown` n'est pas facultatif. `wget` et `unzip` lancés en root laissent des

fichiers appartenant à root ; le pool FPM tourne sous `web2` et ne pourrait ni écrire sa configuration, ni calculer ses vignettes, ni recevoir de mise à jour.

L’assistant

Ouvrir `https://photos.example.com/install.php` dans un navigateur et répondre aux questions : hôte de base `localhost`, nom de la base, utilisateur, mot de passe, puis le compte administrateur de la galerie.

L’assistant écrit un seul fichier, qu’il faut connaître parce que c’est lui qui porte les identifiants :

`[dans web-srv]` :

```
cat /var/www/clients/client1/web2/web/local/config/database.inc.php
```

```
<?php
$conf['dblayer'] = 'mysqli';
$conf['db_base'] = 'c1photos';
$conf['db_user'] = '<utilisateur de la base>';
$conf['db_password'] = '<mot de passe>';
$conf['db_host'] = 'localhost';

$prefixeTable = 'piwigo_';

define('PHPWG_INSTALLED', true);
define('PWG_CHARSET', 'utf-8');
define('DB_CHARSET', 'utf8');
define('DB_COLLATE', '');

?>
```

Le préfixe des tables est `piwigo_`. Toutes les requêtes de dépannage de cette page s’en servent ; sur une installation qui aurait choisi un autre préfixe, les adapter.

La configuration locale

Tout ce qui n’a pas de case dans l’interface se règle dans un seul fichier.

`[dans web-srv]` :

```
nano /var/www/clients/client1/web2/web/local/config/config.inc.php
```

Le fichier complet en service :

```
<?php

/* Importation des métadonnées */

$conf['use_exif_mapping'] = array(
  'date_creation'        => 'DateTimeOriginal',
  );

$conf['use_iptc'] = true;
$conf['use_iptc_mapping'] = array(
  'author'          => '2#080',
  'name'            => '2#005',
  'comment'         => '2#120',
  'keywords'        => '2#025',
  );

/* Protection des originaux : les liens passent par action.php,
   qui vérifie les permissions. */

$conf['original_url_protection'] = 'all';

/* Envoi des courriels par SMTP */

$conf['smtp_host']     = 'smtp.protonmail.ch:587';
$conf['smtp_user']     = '<adresse d expedition>';
$conf['smtp_password'] = '<mot de passe d application>';
$conf['smtp_secure']   = 'tls';
```

```
?>
```

⚠ Jamais `config_default.inc.php`. Ce fichier porte les valeurs livrées avec Piwigo et il est remplacé à chaque mise à jour : tout ce qu'on y écrit disparaît sans avertissement, au moment précis où l'on pense avoir mis l'application à jour. `local/config/config.inc.php` est le seul endroit prévu pour les surcharges, et il survit aux mises à jour.

Le greffon LocalFiles Editor permet d'éditer ce fichier depuis l'administration, sans accès au serveur. Il dépose une copie `config.inc.bak.php` à côté à chaque enregistrement — à retirer, puisqu'elle contient le mot de passe SMTP dans un répertoire servi par le serveur web.

Ce qui est en fichier, et ce qui est en base

C'est la distinction qui décide de ce qu'une restauration ramène, et elle n'est écrite nulle part dans la documentation.

Dans `config.inc.php` — restauré avec les fichiers du site : le mapping des métadonnées, la protection des originaux, les paramètres SMTP.

En base de données — restauré avec le vidage MySQL, et seulement avec lui : l'inscription libre des visiteurs, le verrouillage de la galerie, les tailles de vignettes activées, les permissions d'albums, les comptes.

`[dans web-srv]` — lire les réglages qui vivent en base :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT param, LEFT(value,60) AS valeur FROM piwigo_config \
      WHERE param IN ('allow_user_registration','gallery_locked', \
      'email_admin_on_new_user','obligatory_user_mail_address') \
      ORDER BY param;"
```

La barre oblique inverse en fin de ligne fonctionne aussi à l'intérieur des guillemets doubles : bash supprime la barre et le saut de ligne avant de passer la chaîne à `mysql`. C'est ce qui permet de couper une requête longue sans la

déformer — et ce qui la rend recopiable depuis un PDF, où une ligne trop longue s'enroulerait sans marque visible.

⚠ Un paramètre absent de cette table n'est pas un paramètre absent. `original_url_protection` ne s'y trouve pas — il ne vit que dans le fichier. Chercher un réglage uniquement en base, ou uniquement en fichier, mène à conclure qu'il n'est pas posé alors qu'il l'est.

Le mot de passe root de MariaDB est rangé dans `/home/webadmin/.my.cnf`, en mode 600, et c'est ce fichier que lit le script de vidage nocturne. `--defaults-file` évite d'écrire un mot de passe sur la ligne de commande, où il serait visible dans `ps` par tout compte de la machine pendant la durée de l'exécution, et conservé dans l'historique du shell.

Les métadonnées

Piwigo peut remplir ses champs à partir de ce que les fichiers portent déjà. C'est ce qui permet de garder les mots-clés et les légendes dans les photos plutôt que dans une base qu'il faudrait ressaisir un jour.

La documentation officielle des correspondances : https://piwigo.org/doc/doku.php?id=user_documentation:metadata

La date vient de l'EXIF, champ `DateTimeOriginal` — l'heure de la prise de vue, celle qu'écrit l'appareil.

Tout le reste vient de l'IPTC, dont les champs portent des numéros :

Champ Piwigo	IPTC	Nom courant
`author`	`2#080`	By-line
`name`	`2#005`	Object Name
`comment`	`2#120`	Caption-Abstract
`keywords`	`2#025`	Keywords

Après toute modification de ces correspondances, resynchroniser les répertoires

: les photos déjà en base ne sont pas relues d’elles-mêmes.

⚠ Ne jamais mapper `comment` sur le champ EXIF `UserComment`. C’est la correspondance qui vient naturellement à l’esprit, et elle casse la synchronisation dès qu’une photo de téléphone entre dans la bibliothèque. La raison, qui vaut d’être comprise, est expliquée dans la section *Dépannage*. La légende vient donc de l’IPTC `2#120`, le champ légitime pour cet usage — et le seul que Piwigo nettoie et convertit correctement avant de l’écrire en base.

Extensions et thèmes

Les greffons en service

Greffon	Ce qu’il fait
LocalFiles Editor	édition de `config.inc.php` depuis l’administration
SmartAlbums	albums dynamiques construits par critères
ShareAlbum	partage d’un album par lien
private_share	partage d’une photo précise, à durée limitée
check_uploads	vérification du répertoire `upload/` — voir *Pièges connus*
language_switch	choix de la langue par le visiteur
two_factor	second facteur d’authentification sur les comptes
TakeATour	visite guidée, livrée avec Piwigo

`two_factor` mérite d’être nommé parce qu’il change la nature de la protection : sur une galerie déjà placée derrière une authentification Apache, il ajoute un facteur au compte Piwigo, pas au verrou du serveur. Les deux se cumulent et couvrent deux risques différents — le mot de passe partagé du site d’un côté, le compte nominatif de l’autre.

Un greffon désactivé n’est pas un greffon absent. Piwigo garde sa ligne en base avec l’état `inactive`, et son répertoire sur le disque. Pour savoir ce qui tourne réellement :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT id, state, version FROM piwigo_plugins ORDER BY state, id;"
```

⚠ Un greffon présent sur le disque sans aucune ligne dans cette table n’a jamais été activé. Piwigo ne l’inscrit qu’à la première activation. C’est la seule façon de distinguer « installé et désactivé » de « déposé et jamais mis en service ».

⚠ Méfiance envers les greffons présentés comme des dépendances. Certains bibliothèques partagées d’anciennes versions — `GrumPluginClasses` en est l’exemple — sont décrites partout comme requises par plusieurs greffons, alors que ceux qui en dépendaient ont depuis été réécrits. Vérifier en désactivant plutôt qu’en recopiant la liste d’un guide ancien.

Les thèmes

Deux thèmes sont en service : modus, celui que les comptes utilisent, et smartpocket, vers lequel Piwigo bascule automatiquement sur les écrans de téléphone.

Le thème n’est pas un réglage global. Piwigo le range dans les préférences de chaque utilisateur :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT DISTINCT theme FROM piwigo_user_infos;"
```

⚠ Ne rien insérer dans les gabarits d’un thème. C’est la manière évidente d’ajouter un code de suivi ou un pied de page — et une mise à jour du thème écrase le fichier, en silence. Ce qui doit survivre passe par un thème enfant, ou par un greffon prévu pour ça.

Alimenter la galerie depuis le NAS

C’est la partie qui demande le plus de soin, parce qu’elle ouvre un accès en écriture vers la racine web d’un site en ligne.

Le principe

Le NAS pousse ses photos vers le répertoire `galleries/` du site par le démon rsync du serveur web, puis Piwigo lit ce répertoire et met sa base à jour. Deux étapes distinctes : le transfert des fichiers ne crée aucune photo dans la galerie tant que la synchronisation n'a pas été lancée.

Les deux modules rsync

Le serveur web expose deux modules qui concernent cette galerie, et ils ne servent pas à la même chose :

- `photos`, en lecture seule, sur la racine du site. C'est par lui que le NAS vient tirer la sauvegarde du site.
- `gallerie`, en écriture, sur `galleries/` seulement. C'est par lui que les photos arrivent.

Leur déclaration complète figure dans `/etc/rsyncd.conf` du conteneur, écrit en entier dans Serveur web, section *Le démon rsync*. Un seul fichier, un seul endroit où le corriger.

⚠ `gallerie` est le seul module en écriture de tout le parc, et il pointe dans la racine web d'un site accessible depuis Internet. Ce qui borne le risque n'est pas le module lui-même mais son `hosts allow`, limité à la seule machine qui a le droit de s'en servir. Ce filtre vit côté serveur, n'est pas transmis au client et ne se contourne pas — contrairement aux options de la ligne de commande d'un script, qui décident seulement de ce qu'on demande.

⚠ Le protocole du démon rsync n'est pas chiffré. L'authentification l'est, les données non. C'est un compromis acceptable sur un réseau domestique maîtrisé, à ne pas reproduire au-delà : sur un réseau qu'on ne contrôle pas, ce transfert passe par SSH.

Le script de téléversement

Il vit sur le NAS, pas sur le serveur web.

Sur `[nas-host]` :

```
nano /home/hostadmin/scripts/rsync/auto/upload_photos_photos-webadmin-ca.sh
```

```
#!/bin/bash
# Téléverse les photos de nas1 vers la galerie Piwigo (photos.example.com).
# Miroir : toute suppression côté nas1 se propage à la galerie.

PASS=/home/hostadmin/scripts/rsync/auto/rsync_pass
SOURCE=/media/nas1/Photographies/Photos/
CIBLE=test@192.168.0.6::gallerie

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

# Garde-fou : ne rien synchroniser si la source n'est pas là.
# /media/nas1 est un montage lié : démonté, le chemin existe toujours
# et rsync verrait une source vide, que --delete propagerait à la galerie.
if ! mountpoint -q /media/nas1; then
    echo "ARRÊT : /media/nas1 n'est pas monté — aucune synchronisation."
    exit 1
fi

if [ -z "$(ls -A "$SOURCE" 2>/dev/null)" ]; then
    echo "ARRÊT : source vide ou illisible — aucune synchronisation."
    exit 1
fi

rsync -avz --stats --delete --delete-excluded \
      --password-file "$PASS" \
      --exclude='quota.*' \
      --exclude='Picasa.ini' \
      --exclude='.dtrash' \
      --exclude='.trashed*' \
      --exclude='.piwigo_import_tree' \
      --exclude='.comments' \
```

```
      --exclude='*.[mM][pP]4' \
      --exclude='*.[aA][vV][iI]' \
      --exclude='*.[mM][oO][vV]' \
      --exclude='*.[mM][kK][vV]' \
      "$SOURCE" "$CIBLE"
CODE=$?

echo "=== $(basename "$0") — fin $(date -Is) — code de sortie : $CODE"
exit $CODE
```

Sur `[nas-host]` — le fichier de mot de passe, qui ne contient que le mot de passe, sans nom d'utilisateur :

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

Sur `[nas-host]` :

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

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

Le garde-fou, et pourquoi il existe

C'est le passage à ne pas sauter.

`--delete` fait de la cible un miroir exact de la source : ce qui disparaît d'un côté disparaît de l'autre. C'est exactement ce qu'on veut d'une synchronisation — et c'est un piège dès que la commande part toute seule.

`/media/nas1` est un montage lié. S'il n'est pas monté au moment où la tâche se déclenche, le chemin `/media/nas1/Photographies/Photos/` existe toujours : c'est un répertoire vide sur le disque système. rsync fait alors précisément ce qu'on lui demande — il synchronise une source vide vers une cible pleine, avec `--delete`. À une heure du matin, sans témoin, la galerie se vide.

Le script ne peut pas distinguer « le disque est démonté » de « on a tout supprimé volontairement ». Les deux tests règlent la question autrement : le premier vérifie que le chemin mène bien au système de fichiers attendu, le second attrape tout le reste — un répertoire renommé, un droit perdu, un chemin corrigé de travers.

Une troisième protection existe et n'est pas retenue ici. `--max-delete=N` fait abandonner la synchronisation si elle s'apprête à supprimer plus de N fichiers. Elle est excellente sur une bibliothèque stable, et nuisible pendant une réorganisation : un déplacement d'album compte comme des suppressions, et un seuil qui bloque le travail légitime finit par être retiré — au mauvais moment. À poser quand la source est stabilisée, pas avant.

⚠ Avant toute première exécution qui suit un changement de chemin, faire un essai à blanc. Le `-n` montre ce qui serait supprimé sans rien toucher :

```
rsync -avzn --stats --delete --password-file "$PASS" "$SOURCE" "$CIBLE" | tail
-20
```

La tâche planifiée

Sur `[nas-host]` :

```
crontab -e
```

```
30 01 * * * bash /home/hostadmin/scripts/rsync/auto/upload_photos_photos-
webadmin-ca.sh >> /var/log/backup/$(date +\%F).log 2>&1
```

⚠ Une ligne de cron ne se coupe pas. Cron n'accepte aucune continuation par barre oblique inverse. Dans un PDF elle s'enroule sans signe visible — la recopier depuis le fichier `.md`, jamais depuis le PDF.

⚠ Le `\%` est obligatoire. Dans une crontab, `%` marque une fin de ligne et bascule le reste vers l'entrée standard de la commande.

⚠ `bash` et non `sh`. Le préfixe `sh` court-circuite le `#!/bin/bash` du script. Un script écrit pour bash mais lancé par `sh` échoue uniquement en cron, jamais à la main — le pire des modes de défaillance.

L'heure est choisie après les sauvegardes, pas avant. Le tirage du site a lieu plus tôt dans la nuit : les photos déposées à 1 h 30 ne sont donc sauvegardées que la nuit suivante. Ce n'est pas un défaut — la copie qui compte est la source, sur le NAS, elle-même sauvegardée par ailleurs. L'ordre inverse, lui, ferait sauvegarder chaque nuit des fichiers qui existent déjà en double.

La synchronisation dans Piwigo

Le transfert terminé, les fichiers sont sur le disque et invisibles dans la galerie. Il reste à les inscrire en base : Administration → Photos → Synchroniser, en choisissant la synchronisation des répertoires et des métadonnées.

C'est aussi l'étape où se manifestent les problèmes décrits en *Dépannage* : c'est elle qui lit chaque fichier, sollicite `exif` et `gd`, et écrit en base.

Fermer l'accès direct aux fichiers

Cette section décrit le comportement par défaut de Piwigo et ce qu'il faut faire pour le corriger. Elle vaut la lecture même pour qui ne suit pas ce guide.

Le problème

Piwigo protège ses pages, pas ses fichiers. Apache sert `galleries/`, `upload/` et `_data/i/` en statique : une requête sur un de ces chemins ne passe jamais par Piwigo, donc aucune permission d'album ne s'applique.

Sur une galerie pourtant entièrement privée, et sans être connecté :

```
https://photos.example.com/galleries/un-album/20171008_105702.jpg
→ image servie en 5312×2988

https://photos.example.com/_data/i/galleries/un-album-prive/photo-me.jpg
→ vignette servie en 594×594
```

Un nom d'album commençant par un point n'y change rien : c'est une convention du système de fichiers, invisible pour Apache.

Ce n'est pas un défaut d'installation. `$conf['original_url_protection']` vaut `''` d'origine, et le code de Piwigo porte à cet endroit un commentaire de ses propres développeurs annonçant le travail restant à faire sur les répertoires de médias. C'est une tâche en attente, pas un oubli de configuration.

La seule barrière réelle est donc la difficulté de deviner un chemin, et elle est faible. Les noms de fichiers de téléphone sont très prévisibles — `20171008_105702.jpg` est une date suivie d'une heure — et il suffit qu'un nom d'album fuite une fois pour qu'un répertoire entier devienne énumérable par force brute sur l'horodatage.

La fausse bonne idée

Le correctif que l'on trouve le plus souvent consiste à basculer `$conf['derivative_url_style']` sur `2`, ce qui fait passer les vignettes par `i.php` au lieu d'être servies en statique — et permettrait ensuite de bloquer le répertoire.

Vérification faite dans le code source : `i.php` n'effectue aucun contrôle de permission. Il ouvre le fichier et le renvoie. L'adresse `i.php?/galleries/...` répondrait donc tout aussi bien à un inconnu. C'est un déplacement d'adresse, pas une autorisation.

La leçon dépasse ce cas : avant d'appliquer un correctif trouvé en forum, vérifier dans le code qu'il fait bien ce qu'on croit. Un correctif qui change le symptôme sans changer le mécanisme laisse le problème intact et la conviction qu'il est réglé.

Les originaux — deux couches

D’abord la configuration, sinon l’affichage casse. Dans `local/config/config.inc.php` :

```
$conf['original_url_protection'] = 'all';
```

Piwigo cesse alors de publier des liens `galleries/...` et fait passer les originaux par `action.php`, qui vérifie les permissions.

Ensuite le blocage Apache, dans ISPConfig → le site → onglet Options → champ Directives Apache :

```
<Directory /var/www/clients/client1/web2/web/galleries>
    AllowOverride None
    Require all denied
</Directory>

<Directory /var/www/clients/client1/web2/web/upload>
    AllowOverride None
    Require all denied
</Directory>
```

⚠ L’ordre des deux couches n’est pas indifférent. Assouplir côté application avant de verrouiller côté serveur : dans l’autre sens, on se coupe soi-même l’accès entre les deux étapes, avec une galerie qui n’affiche plus rien et une cause difficile à isoler.

`AllowOverride None` complète le dispositif. Un `.htaccess` déposé un jour dans `upload/` par une faille de téléversement resterait sans effet.

Pourquoi le vhost plutôt qu’un `.htaccess`. Les deux fonctionnent, mais un `.htaccess` posé dans un répertoire de données est fragile : un greffon de ménage, une mise à jour ou une restauration partielle peut le faire disparaître, sans que rien ne le signale. Une directive de vhost n’est modifiable que par ISPConfig.

Les vignettes — l’authentification en amont

Les dérivées de `_data/i/` ne passent par aucun contrôle de permission, et le correctif qu’on croirait naturel n’en est pas un. Pour une galerie entièrement privée, le verrou se pose donc plus haut : devant tout le site.

Dans le panneau : Sites → Protection de dossier, chemin `/`, actif ; puis Utilisateur de dossier protégé, avec un identifiant et un mot de passe partagés.

ISPConfig écrit alors, à la racine du site :

```
### ISPConfig folder protection begin ###
AuthType Basic
AuthName "Members Only"
AuthUserFile /var/www/clients/client1/web2/web/.htpasswd
require valid-user
### ISPConfig folder protection end ###
```

⚠ Ces identifiants Apache s’ajoutent devant les comptes Piwigo, ils ne les remplacent pas. Chacun garde son compte nominatif dans la galerie ; le mot de passe du site est partagé et ne dit rien de qui entre.

⚠ Ne pas bloquer `_data/` en entier. `_data/combined/` contient les feuilles de style et les scripts agrégés que le navigateur doit charger : les refuser donne une galerie sans mise en forme.

Cette protection-ci repose bien sur un `.htaccess`, contrairement à celle des originaux, et c’est une dépendance à connaître plutôt qu’un défaut. Le fichier vit à la racine du site, hors de `galleries/` — la synchronisation nocturne ne peut donc pas l’emporter, et les greffons de ménage ne regardent pas là. Mais il exige `AllowOverride All` sur ce répertoire dans le vhost, et il ne survit pas à une restauration du site qui l’aurait omis.

Le fichier de mots de passe est déposé dans la racine web servie. Apache le refuse par une règle globale, qu’il vaut mieux constater que supposer :

```
grep -A2 'FilesMatch' /etc/apache2/apache2.conf
```

Le bloc attendu refuse tout fichier dont le nom commence par `.ht` :

```
<FilesMatch "^\.ht">
    Require all denied
</FilesMatch>
```

L’application mobile Piwigo NG gère l’authentification Basic : Réglages → Identification HTTP → activer, puis les identifiants Apache, en plus du compte de la galerie.

Durcissement complémentaire

Fermer l’inscription libre. Configuration → Options → Permissions → décocher « Permettre l’enregistrement des utilisateurs ». Sans quoi n’importe qui crée un compte. Ce réglage vit en base, pas dans un fichier de configuration : il ne revient pas avec une restauration des seuls fichiers.

Réduire les tailles de dérivées. Configuration → Tailles de photo. Les tailles en service ici sont Carré, Miniature, XXS, S, M et L ; XL et XXL sont décochées. Elles ne servent qu’à l’affichage de très grands écrans, et chacune multiplie le nombre de fichiers calculés — 8,7 Go de vignettes pour 50 Go d’originaux avec six tailles seulement.

⚠ Décocher une taille ne supprime pas les fichiers déjà calculés. Les purger par Outils → Maintenance.

Vérifier

Le site étant sous authentification Apache, un appel sans identifiants renvoie `401` avant d’atteindre la règle de répertoire. Le test doit donc être lu avec ça en tête :

```
curl -s -o /dev/null -w '%{http_code}\n' -u UTILISATEUR_APACHE \
```

```
https://photos.example.com/galleries/un-album/une-photo.jpg
```

Code	Interprétation
`403`	la directive fonctionne
`200`	l'image est servie : la directive n'est pas active
`401`	identifiants refusés ; le test ne prouve rien

Subtilité qui simplifie le test. Avec `Require all denied`, aucune authentification ne peut jamais satisfaire la règle. Apache ne réclame donc pas de mot de passe et refuse directement en `403` — y compris avec des identifiants faux. Un `403` obtenu avec de mauvais identifiants prouve que la règle a tranché avant l'authentification. Pour écarter l'hypothèse d'un serveur qui répondrait `403` partout, appeler la racine du site avec les mêmes identifiants faux : elle doit répondre `401`.

`[dans web-srv]` — vérifier la syntaxe avant tout rechargement, une erreur coupant tous les sites du conteneur et pas seulement celui-ci :

```
apache2ctl configtest
```

Tester depuis une fenêtre de navigation privée, jamais depuis une session déjà authentifiée. C'est la seule façon de voir ce que voit un visiteur.

Ce que la sauvegarde ne couvre pas

Trois choses vivent hors du répertoire du site et hors de la base de Piwigo. Une sauvegarde qui ne couvre que ces deux-là les perd.

Les directives Apache du site — celles qui bloquent `galleries/` et `upload/` — sont stockées dans la base d'ISPConfig et écrites dans `/etc/apache2/sites-available/`. Un site recréé dans le panneau repartirait sans elles, avec ses originaux ouverts au premier venu. Vérifier que la base d'ISPConfig est sauvegardée.

La protection de dossier est dans la même base, et son fichier `.htpasswd` dans la racine du site.

Les réglages de la galerie qui vivent en base — inscription libre, tailles de dérivées, permissions d'albums, comptes — ne reviennent qu'avec le vidage MySQL. Restaurer les fichiers seuls donne une galerie qui affiche les bonnes photos avec les mauvaises permissions.

La stratégie complète est décrite dans Sauvegarde — stratégie et méthode, et la marche à suivre pour remonter un site dans Récupération, restauration, reconstruction.

Pièges connus

Le greffon check_uploads

Il liste les fichiers de `upload/` absents de la base et propose « Les supprimer tous ». Tout `.htaccess` de protection y figure en permanence, puisque le greffon ne connaît que les fichiers enregistrés en base.

⚠ Ne jamais cliquer ce bouton sans avoir lu la liste. Il effacerait la protection sans avertissement. C'est l'une des raisons du choix des directives de vhost, qu'aucun greffon ne peut atteindre.

Les entrées `buffer/*.chunk` sont, elles, de vrais déchets : des morceaux d'envois web interrompus. Pour les nettoyer sans toucher au reste, quand aucun envoi n'est en cours :

```
find /var/www/clients/client1/web2/web/upload/buffer -name '*.chunk' -delete
```

« Chemins anormalement dupliqués »

Le tableau de bord annonce un ou plusieurs chemins dupliqués et renvoie vers check_uploads, qui affiche « Tout semble bon ». Les deux contrôles ne regardent

pas la même chose : le greffon analyse le répertoire `upload/`, le message du tableau de bord vient d'un contrôle d'intégrité qui porte sur la base.

Trouver le chemin en cause :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT path, COUNT(*) AS n, GROUP_CONCAT(id) AS ids \
      FROM piwigo_images GROUP BY path HAVING n > 1;"
```

Un chemin de fichier étant unique sur un disque, cette requête ne devrait jamais rien rendre. Si elle rend quelque chose, c'est que le classement par défaut de MySQL ignore la casse et rassemble deux chemins que le système de fichiers tient pour distincts. Le défaut de l'outil est ici ce qui révèle le problème.

⚠ Toujours réafficher les lignes une par une avant de conclure, le regroupement n'en montrant qu'une comme représentante :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT id, path FROM piwigo_images WHERE id IN (<ids>);"
```

Le cas type : `PICT2257.jpg` et `Pict2257.jpg` dans le même répertoire — noms distincts pour Linux, identiques pour MySQL, tailles et empreintes différentes. Deux photos réelles, donc, et pas un doublon d'enregistrement : la base est saine et l'alerte techniquement fausse.

Mais le problème sous-jacent est réel. Toute requête de Piwigo cherchant une photo par son chemin ramène les deux enregistrements. C'est le terrain idéal pour qu'une synchronisation attribue des métadonnées au mauvais fichier. Ce genre de collision vient d'une photothèque passée par Windows ou par un logiciel de gestion insensible à la casse.

Avant de corriger, regarder ce que chaque enregistrement porte — le fichier renommé ou supprimé perdra le sien :

```
mysql --defaults-file=/home/webadmin/.my.cnf c1photos \
  -e "SELECT image_id, tag_id FROM piwigo_image_tag \
      WHERE image_id IN (<ids>);"
```

Balayer ensuite toute la photothèque pour savoir si le cas est isolé. Sur la source, pas sur le serveur, qui n'en est qu'un miroir :

```
find /media/nas1/Photographies/Photos \
  | awk '{k=tolower($0); n[k]++; p[k]=p[k] $0 "\n"}
         END {for (k in n) if (n[k]>1) printf "%s----\n", p[k]}'
```

`find` liste fichiers et répertoires — les albums se télescopent de la même façon. `awk` indexe chaque chemin par sa version tout en minuscules, compte les occurrences, et n'affiche que les groupes qui en comptent plus d'une. Aucune sortie signifie qu'il n'y a aucune collision de casse dans toute l'arborescence.

Correction : renommer ou supprimer l'un des deux à la source, puis relancer le téléversement et la synchronisation. Ne rien modifier dans `galleries/` côté serveur — le rsync suivant annulerait le changement. Réappliquer ensuite mots-clés et albums virtuels par Photos → Gérer par lot si le fichier conservé était le renommé.

□ Une image qui se télécharge au lieu de s'afficher

Le `/etc/mime.types` du conteneur ne déclare pas `image/webp`. Les fichiers `.webp` du serveur partent donc en `application/octet-stream` — « octets bruts » —, ce que les navigateurs traitent comme un téléchargement plutôt que comme une image.

⚠□ Le symptôme ne ressemble pas à une erreur. Le fichier existe, Apache le sert, la réponse est `200`, rien n'apparaît au journal — et l'image manque, ou le navigateur propose de l'enregistrer. Une vignette absente fait chercher du côté des dérivées, des droits, du cache : jamais du type déclaré.

Le fichier appartient au conteneur, pas à la galerie : la déclaration et sa

pose sont dans Serveur web, section *Les types MIME*. Ce qui appartient à cette page, c'est le contrôle.

`[dans web-srv]` — sur une dérivée, pas sur un original : `galleries/` est refusé par la directive de vhost, ce qui donnerait `403` sans rien dire du type.

```
curl -sI -k -u UTILISATEUR_APACHE \
  --resolve photos.example.com:443:127.0.0.1 \
  https://photos.example.com/_data/i/<chemin>/<fichier>.webp \
  | grep -i content-type
```

Attendu : `content-type: image/webp`. Un `application/octet-stream` désigne la déclaration manquante, pas la galerie.

Les droits de galleries/ ne sont pas ceux du reste du site

Le répertoire `galleries/` porte des permissions plus ouvertes que ses voisins, et ce n'est pas une anomalie : elles sont recopiées du NAS par la synchronisation, qui préserve les droits de la source.

⚠ Ne pas les corriger côté serveur. Un `chmod` tiendrait sur `galleries/` lui-même — que rsync ne touche pas, la source du script se terminant par une barre oblique — mais serait défait sur tous les sous-répertoires dès la nuit suivante. Le résultat serait une arborescence à deux régimes, moins cohérente que l'état de départ.

Le contenu de ce répertoire est de toute façon refusé par Apache, et les seuls comptes qui pourraient en lire les noms sont des comptes de service. La correction, si elle est jugée nécessaire, se fait à la source ou par une option `--chmod` dans le script de téléversement — pas sur le serveur.

Dépannage

« Incorrect string value » sur la colonne comment

Symptôme, pendant une synchronisation :

```
Fatal error: Uncaught mysqli_sql_exception: Incorrect string value:
'\xE8\x04\x04\x0B\x13\xFF...' for column ...comment at row 5130
```

Cause. Un mapping de `comment` vers le champ EXIF `UserComment` fait lire à Piwigo un champ dont l'en-tête de huit octets annonce l'encodage du reste — `ASCII\0\0\0`, `UNICODE\0`, `JIS\0\0\0\0\0`. Les téléphones Samsung y écrivent des données binaires internes sous un en-tête qui annonce pourtant de l'ASCII :

Octets	Signification
`41 53 43 49 49 00 00 00`	`ASCII\0\0\0` — en-tête EXIF
`4a 4b 4a 4b`	`JKJK` — bloc de détection de scène Samsung
`41 4c 43 53 49 49 46 35`	`ALCSIIF5` — autre bloc de la même famille

Pourquoi exiftool ne voit rien. Il lit l'en-tête, décode selon ce qu'il annonce, et restitue de l'UTF-8 propre. L'extension EXIF de PHP ne fait pas ce travail : elle renvoie les octets bruts, que MySQL en mode strict refuse.

⚠ Une divergence entre exiftool et PHP sur un même fichier n'est pas un bug. Pour reproduire ce que fait une application PHP, il faut scanner avec PHP :

```
nano /root/scan-usercomment.php
```

```
<?php
$dir = $argv[1] ?? '/var/www/clients/client1/web2/web/galleries';
$it = new RecursiveIteratorIterator(
  new RecursiveDirectoryIterator($dir, FilesystemIterator::SKIP_DOTS));
foreach ($it as $f) {
  if (!$f->isFile()) continue;
  if (!preg_match('/\.(jpe?g|tiff?)$/i', $f->getFilename())) continue;
  $exif = @exif_read_data($f->getPathname(), null, true);
```

```
  if (!$exif) continue;
  foreach ($exif as $section => $tags) {
    if (!is_array($tags)) continue;
    foreach (['UserComment','ImageDescription'] as $k) {
      if (!isset($tags[$k]) || !is_string($tags[$k])) continue;
      if (!mb_check_encoding($tags[$k], 'UTF-8')) {
        echo $f->getPathname()." [$section.$k] "
           . bin2hex(substr($tags[$k],0,24))."\n";
      }
    }
  }
}
```

```
php /root/scan-usercomment.php 2>/dev/null | wc -l
```

⚠ Ne pas envoyer la sortie dans `| head -20` pour compter : le script est tué en cours de route par le signal de tuyau fermé, et le décompte est faux.

Correctif : ne pas mapper `comment` sur `UserComment`. Nettoyer les fichiers un par un serait du travail perdu — il n’y a aucun commentaire réel à sauver dans ces champs.

⚠ Deux contournements qui n’en sont pas. Passer la base en `utf8mb4` ne règle rien : ces octets ne sont valides dans aucune variante d’UTF-8. Désactiver le mode strict de MySQL non plus : la valeur serait tronquée silencieusement au premier octet invalide.

Synchronisation qui s’interrompt sans message

C’est `max_execution_time` qui expire. Voir *Les limites d’envoi* : la valeur par défaut de 30 secondes ne tient pas sur une bibliothèque volumineuse, et l’interruption ne produit aucune erreur exploitable.

Les photos arrivent mais n’apparaissent pas

Le transfert et la synchronisation sont deux étapes distinctes. Vérifier d’abord que les fichiers sont bien sur le serveur, puis lancer la

synchronisation depuis l’administration.

Si les fichiers ne sont pas arrivés, le journal du démon rsync le dit :

```
grep gallerie /var/log/rsync.log | tail -5
```

Et côté NAS, le journal de la tâche nocturne :

```
grep -A5 upload_photos /var/log/backup/$(date +%F).log
```

⚠ Une ligne « ARRÊT » dans ce journal est le garde-fou qui a fonctionné, pas une panne : la source n’était pas disponible et la galerie a été préservée. Chercher pourquoi le montage manquait plutôt que de relancer la synchronisation.

Vérifications après reconstruction

`[dans web-srv]` — la version qui sert réellement le site :

```
ss -xlp | grep web2.sock
```

`[dans web-srv]` — les limites appliquées :

```
grep -E 'php_admin' /etc/php/8.3/fpm/pool.d/web2.conf
```

`[dans web-srv]` — le type WebP est déclaré, sans quoi les dérivées de ce format se téléchargent au lieu de s’afficher :

```
grep -nE '^image/webp' /etc/mime.types
```

`[dans web-srv]` — les extensions dont Piwigo dépend :

```
php8.3 -m | grep -E '^(exif|gd|imagick|mysqli)$'
```

`[dans web-srv]` — la syntaxe d’Apache et les directives du site :

```
apache2ctl configtest
grep -A3 'web/galleries' /etc/apache2/sites-available/photos.example.com.vhost
```

Sur `[nas-host]` — le téléversement, en conditions réelles :

```
bash /home/hostadmin/scripts/rsync/auto/upload_photos_photos-webadmin-ca.sh \
  | tail -5
```

Dans un navigateur, en fenêtre privée — les trois réponses qui comptent :

- la racine du site doit demander un mot de passe ;
- une adresse `galleries/...` doit être refusée ;
- une photo d’album privé ne doit pas s’afficher sans connexion.

⚠ Redémarrer le conteneur avant de considérer l’installation terminée. C’est ce test qui révèle ce qui a été fait à la main sans être inscrit : un service lancé mais non activé, un réglage appliqué à chaud. Rien de cela ne se voit tant qu’on ne redémarre pas.

À retenir

- Une application web qui gère des permissions ne protège que ce qui passe par elle. Tout répertoire de médias servi en statique est hors de son contrôle.

- Avant d'appliquer un correctif trouvé en forum, vérifier dans le code source qu'il fait bien ce qu'on croit. Un correctif qui déplace le symptôme laisse le problème intact et la conviction qu'il est réglé.

- Ordre d'application : assouplir côté application avant de verrouiller côté serveur, sinon on se coupe soi-même l'accès entre les deux étapes.

- Un outil de ménage qui compare le disque à une base ne connaît pas l'intention. Placer les règles de sécurité là où ces outils ne regardent pas.

- Une synchronisation avec `--delete` qui part toute seule doit vérifier sa source avant d'agir. Elle ne peut pas distinguer un disque démonté d'une suppression volontaire.

- Tester depuis une fenêtre privée, jamais depuis une session déjà authentifiée.

Ce que ce guide ne couvre pas

- le conteneur, Apache, le multi-PHP, ISPConfig, les certificats et le démon rsync — Serveur web — Apache multi-PHP et ISPConfig

- l'hôte, ses partages et les tâches nocturnes qui s'y exécutent — INCUS — Hôte et conteneur & serveur NAS

- la stratégie de sauvegarde du parc — Sauvegarde — stratégie et méthode

- la remise en service après incident — Récupération, restauration, reconstruction