

260611_procédure Komga-PDF par Claude sur StirlingPDF

Présentation

La procédure Komga-PDF est un script de compression de fichiers PDF pour réduire considérablement la taille du stockage des bibliothèques d'ouvrages de la collection de Julien.

L'acquisition de fichiers issus de bases de données Internet accumule des fichiers PDF en haute résolution (images surtout). Ces fichiers sont synchronisés entre les NAS Synology et un disque dur kDrive Infomaniak de 6 To.

Contexte

Des points de montage Rclone relient les répertoires kDrive et les containers Docker installés sur le VPS OVH juxjux.ovh. Ces containers lisent les PDF à la volée — permettant ainsi de construire une infrastructure de très forte densité de connaissances digitales (epub, pdf, images, vidéos, musiques...) avec une solution serveur d'entrée de gamme (10 €/mois tout inclus).

La lecture réseau est multi-support : PC, tablette, téléphone. Elle est aussi nomade, séquentielle et pressée. La compacité des fichiers PDF devient donc nécessaire pour diffuser rapidement sur les réseaux. Un magazine ou un ouvrage informatique ne doit pas peser 130 Mo sur un écran de 10 pouces.

Procédure

- Les nouveaux PDF sont déposés dans le dossier Syncthing : `Syncthing/komga-pdf/` depuis n'importe quelle machine (PC Elio+Jux, PC Nexte, Ubuntu, Xiaomi...)

- Syncthing synchronise automatiquement vers le VPS : `/home/debian/Documents/komga-pdf/`
- Le service `komga-watch` (systemd VPS) détecte l'arrivée via inotifywait local
- Dès détection, `komga_compress_vps.py` compresse le PDF via StirlingPDF et dépose le résultat dans `/home/debian/Documents/komga-compressed/`
- Julien range manuellement les PDF compressés vers sasnexte puis dans le dossier thématique de son choix
- Les originaux ne sont jamais supprimés du VPS — Julien valide et conserve si besoin (haute résolution souhaitée)

Architecture technique

Élément	Chemin / URL
Dépôt (toutes machines)	<code>Syncthing/komga-pdf/</code>
Dossier VPS (Syncthing)	<code>/home/debian/Documents/komga-pdf/</code>
Destination VPS	<code>/home/debian/Documents/komga-compressed/</code>
API compression	<code>https://spdf.juxjux.ovh/api/v1/misc/compress-pdf</code>
Scripts VPS	<code>/home/debian/komga_compress_vps.py</code> + <code>/home/debian/komga_watch_vps.sh</code>
Logs	<code>journalctl -u komga-watch -f</code> (sur le VPS) + journal persistant <code>/home/debian/logs/komga_compress.log</code>

Scripts

Le service tourne entièrement sur le VPS — plus de dépendance Ubuntu ni de SSH persistant.

Fichier	Rôle
<code>komga_compress_vps.py</code>	Compression : liste les PDFs locaux (extension insensible à la casse), envoie à StirlingPDF, écrit dans <code>komga-compressed/</code> de façon atomique. Idempotent. Vérifie que la réponse est bien un PDF et met à l'écart un fichier après 3 échecs.
<code>komga_watch_vps.sh</code>	Surveillance : traite le backlog au démarrage, puis boucle <code>inotifywait -t 600</code> . Le délai impose un rebalayage périodique, indispensable car les événements survenant pendant une compression sont perdus (voir REX 2026-08-23).
<code>/etc/systemd/system/komga-watch.service</code>	Service system (pas user), actif au boot, <code>Restart=always</code> .

Gérer le service (VPS)

```
sudo systemctl status komga-watch  
sudo systemctl restart komga-watch  
journalctl -u komga-watch -f
```

Retour d'expérience (2026-08-23) — Deux fichiers bloqués sans la moindre trace

Deux PDF déposés le 22/08 stagnaient dans `komga-pdf/` sans **aucune ligne au journal** : ni erreur, ni tentative. Le service était pourtant `active`, et deux autres fichiers étaient passés normalement le matin même à 06:13.

La cause — une course perdue sur les événements inotify

`komga_watch_vps.sh` appelait `inotifywait` **une seule fois par tour de boucle**, puis lançait la compression. Pendant toute la durée du traitement, plus personne n'écoutait le dossier.

Horodatage	Événement
06:13:07	<code>inotifywait</code> rend la main (dépôt de « Le Grand Livre... »)
06:13:10	le script Python liste le dossier — 2 PDF trouvés
06:13:10 → 06:15:16	2 min 06 s de compression, aucune écoute inotify
06:15:16	<code>inotifywait</code> redémarre — les événements de la fenêtre sont perdus

Les deux fichiers « Les secrets de... » sont arrivés dans cette fenêtre. Leur `close_write` et leur `moved_to` sont tombés dans l'angle mort, et le `glob()` du script était déjà passé. Ils sont devenus **invisibles pour de bon** — jusqu'au dépôt d'un autre fichier, qui aurait relancé un balayage complet.

Preuves relevées au moment du diagnostic :

- `inotifywait` (PID 1902285) en attente depuis 32 min, soit exactement depuis 06:15:16
- mtime du dossier à 06:45:22 — une **suppression** (les originaux déjà traités), non surveillée donc sans événement
- les deux fichiers parfaitement valides : `%PDF-` ... `%%EOF`, 148,4 et 51,4 Mio

Le défaut est structurel et se reproduit dès que plusieurs fichiers arrivent ensemble — précisément le cas où la compression dure le plus longtemps.

Le correctif — un délai sur inotifywait

```
inotifywait -q -t 600 -e close_write,moved_to "$SOURCE_DIR"
```

`-t` rend la main au bout de 600 s même sans événement, ce qui garantit un **rebalayage périodique**. Le traitement étant idempotent — et désormais silencieux quand il n'y a rien de neuf — un balayage à vide ne coûte rien. Plus aucun fichier ne peut rester orphelin : au pire 10 minutes de latence.

Vérification du correctif, en reproduisant le bug

Le scénario a été rejoué volontairement. Un PDF a été déposé dans le dossier par **lien matériel** (`ln`), qui ne produit qu'un `IN_CREATE` — événement **non surveillé** par le script. C'est l'équivalent exact d'un événement perdu.

Horodatage	Constat
07:51:09	démarrage du guetteur, balayage initial
07:51:56	dépôt du fichier — aucune ligne au journal , donc aucun événement émis
08:01:10	le rebalayage périodique le trouve : « 1 PDF(s) à traiter (sur 1 présents) »

600 secondes pile après le démarrage. Avec l'ancien script, ce fichier serait resté indéfiniment invisible.

La compression a ensuite échoué en HTTP 400 — le PDF de test faisait 193 octets sans table `xref`, StirlingPDF le refuse à raison. Cet échec a validé trois autres correctifs du même coup :

- le compteur de tentatives s'est incrémenté (`ERREUR (1/3)`) et la signature a été mémorisée dans `~/komga_compress_state.json`
- chaque ligne est bien apparue dans le journal persistant
- **rien n'a été écrit en destination** : ni fichier corrompu, ni `.part` orphelin

Les autres défauts corrigés dans la même passe

#	Défaut	Correctif
1	<code>glob("*.pdf")</code> ignorait les <code>.PDF</code> — régression par rapport à l'ancien <code>find -iname</code>	comparaison sur <code>suffix.lower()</code>
2	<code>write_bytes()</code> écrivait le fichier final directement dans un dossier Syncthing : propagation possible d'un PDF incomplet	écriture en <code>.part</code> puis <code>os.replace()</code> , atomique
3	aucune validation de la réponse : une page d'erreur HTML renvoyée en HTTP 200 était enregistrée en <code>.pdf</code>	vérification de la signature <code>%PDF-</code>
4	<code>timeout=300</code> trop juste — un fichier de 148 Mio prend environ 3 min	porté à 900 s
5	si la compression ne gagnait rien, le résultat était écrit quand même	l'original est recopié tel quel, ce qui clôt proprement le traitement
6	un fichier en échec était retenté sans fin — grave avec le rebalayage toutes les 10 min, soit 144 envois par jour	mémoire des échecs dans <code>~/komga_compress_state.json</code> , mise à l'écart après 3 tentatives
7	<code>get_token()</code> sans protection : Stirling indisponible = traceback, lot entier perdu	message d'erreur explicite, sortie propre
8	<code>mkdir(exist_ok=True)</code> sans <code>parents=True</code>	corrigé
9	aucun journal persistant — journald seul, tourné régulièrement	<code>/home/debian/logs/komga_compress.log</code> , en plus de stdout

La mise à l'écart après trois tentatives s'appuie sur la **signature** du fichier (taille + date de modification) : redéposer une version corrigée portant le même nom relance donc le traitement normalement.

Déblocage des deux fichiers et mesure mémoire

Fichier	Original	Compressé	Gain
---------	----------	-----------	------

Les secrets de la photo lifestyle — Baptiste Dulac	148,4 Mio	30,1 Mio	-80 %
Les secrets de la série photo — Frédéric Landragin	51,4 Mio	38,6 Mio	-25 %

Traitement en 4 min 05 s. **Pic mémoire de `stirling-pdf` : 1 426 Mio sur les 2 Go du plafond (71 %)**, aucun OOM kill — c'est le plus gros fichier jamais passé dans la chaîne. Troisième confirmation que porter la limite à 2 Go était le bon choix : le plafond de 1 Go envisagé le 2026-08-02 aurait tué le container.

Sauvegardes et versionnement

- Versions précédentes conservées sur le VPS : `komga_compress_vps.py.bak-260823` et `komga_watch_vps.sh.bak-260823`
- Les scripts de production sont désormais **versionnés dans `Jux-scripts/Komga-PDF/`** sous leurs noms de production. Jusqu'ici ils n'existaient que sur le VPS, hors de tout périmètre Syncthing — exactement le scénario qui a fait perdre les MCP Python avec le profil Windows `eliob`
- Les anciens `komga_compress.py` et `komga_watch.sh` du même dossier correspondent à l'architecture Ubuntu abandonnée en juin 2026 (pilotage par SSH depuis Ubuntu, destination `/mnt/sasnexte/Komga/pdf_compressed`). Ils ne doivent plus servir de référence

Retour d'expérience (2026-08-22) — Test de bout en bout depuis le poste `julie`

Première validation complète de la chaîne depuis le redémarrage du service le 2026-07-08, et **premier passage depuis le PC Windows `julie`** (entré dans le maillage Syncthing le 2026-08-10). Aucune intervention manuelle : dépôt du fichier dans `D:\Syncthing\komga-pdf\`, tout le reste s'est enchaîné seul.

Chronologie mesurée — 1 min 50 s entre le dépôt sur le VPS et le fichier compressé :

Étape	Horodatage (UTC)
Syncthing dépose le <code>.tmp</code> sur le VPS, inotify déclenche	11:24:50
Fin de l'attente de 3 s, le script détecte le PDF	11:24:53
StirlingPDF rend le fichier compressé	11:26:40

Étape	Horodatage (UTC)
Retour du fichier compressé sur le poste julie (Syncthing)	11:26

Résultat de compression :

Fichier	Original	Compressé	Gain
La Revue du Vin de France - Septembre 2026.pdf	129,2 Mo	55,4 Mo	-57%

Même taux que le numéro de juin 2026 du même magazine (-57 %) — la compression est reproductible d'un numéro à l'autre pour une source identique.

Mesure mémoire — marge plus étroite qu'attendu

C'est le point neuf de ce test. Le fichier de 129 Mo est le plus gros passé dans la chaîne à ce jour (précédent record : 96,6 Mo le 2026-06-13). Consommation du container `stirling-pdf`, relevée toutes les 15 s :

t+	Mémoire
15 s	701 Mio
30 s	1001 Mio
45 s	1,036 Gio
60 s	1,069 Gio (pic)
75 s	1,047 Gio
90 s	1,039 Gio (terminé)

- Pic à **1,069 Gio sur le plafond de 2 Gio** posé le 2026-08-02 → il reste ~45 % de marge
- **Le plafond de 1 Go initialement envisagé aurait tué ce traitement en OOM kill.**
La décision de le porter à 2 Go (motivée alors par l'OCR, qui monte à ~969 Mio) se trouve validée une seconde fois, pour une raison indépendante : la compression d'un gros PDF
- La consommation ne suit pas la taille du fichier de façon linéaire — elle plafonne vers 1 Gio et s'y maintient. Mais **une compression concurrente d'un OCR ferait dépasser les 2 Gio**. Ne pas lancer les deux en parallèle, et ne pas redescendre la limite

Point d'attention — aucun log persistant (résolu le 2026-08-23)

Les logs partent uniquement dans journald (`komga_watch_vps.sh` écrit sur stdout, capté par systemd). **Le journal a été tourné depuis le démarrage du service** : tous les traitements de juin et juillet sont irrécupérables, `journalctl -u komga-watch` ne montrait rien avant ce test.

Pour conserver un historique consultable, ajouter une redirection vers un fichier dans le script, ou poser un `journalctl --unit komga-watch` en rotation propre. Non fait à ce jour.

Corrigé le 2026-08-23 : le script Python écrit désormais chaque ligne dans `/home/debian/logs/komga_compress.log` en plus de stdout. Voir le REX du 2026-08-23 ci-dessus.

Retour d'expérience (2026-06-13)

— Refactorisation VPS

Problème de l'architecture Ubuntu : le service tournait sur Ubuntu avec un SSH persistant vers le VPS. Deux défauts structurels :

- `inotifywait` ne détecte pas les fichiers déjà présents au redémarrage du service (backlog silencieux)
- Impossible à contrôler depuis Windows/Claude Code

Solution : service systemd sur le VPS lui-même. inotifywait local, pas de SSH persistant. Le script traite le backlog à chaque démarrage avant d'entrer dans la boucle de surveillance.

Résultats de compression (2026-06-13) :

Fichier	Original	Compressé	Gain
Beaux Arts - Juin 2026.pdf	75.8 Mo	61.1 Mo	-19%
Connaissance des Arts - Juin 2026.pdf	57.3 Mo	35.2 Mo	-39%
La Revue du Vin de France - Juin 2026.pdf	93.3 Mo	39.9 Mo	-57%
Livre lacto fermentation.pdf	96.6 Mo	19.6 Mo	-80%
Monde Gourmand N°93 - Juin 2026.pdf	41.2 Mo	17.4 Mo	-58%

Points techniques :

- Dossier VPS avec **D majuscule** : `/home/debian/Documents/` (Syncthing sensible à la casse)
- Race condition inotifywait/Syncthing : attente 3s après événement, script idempotent
- Python : `/usr/bin/python3` (système, `requests` 2.28.1 disponible)

- StirlingPDF auth : JWT via `POST /api/v1/auth/login` → `session.access_token`, valable 24h
- Fichiers `sync-conflict` Syncthing filtrés automatiquement par le script

Retour d'expérience (2026-06-12)

Résultats de compression :

Fichier	Original	Compressé	Gain
Monde Gourmand N°93 - Juin 2026.pdf	42 Mo	17 Mo	-58%
Connaissance des Arts - Juin 2026.pdf	57 Mo	35 Mo	-39%

Revision #10

Created 2026-06-11 06:12:37 UTC by Julien

Updated 2026-08-23 08:03:00 UTC by Julien