E_la gestion du SWAP du VPS par Claude

REX 260803 — Session Claude sur le VPS Jux

Session du 2026-08-02 (soir) au 2026-08-03 (matin). Quatre chantiers, tous appliqués et vérifiés. Le sujet principal de cette page — le swap — est traité en partie 1 ; les autres chantiers suivent car ils s'expliquent mutuellement (l'allègement JVM est ce qui a fait apparaître la question du swap).

#	Chantier	État	Gain / résultat
1	Swap saturé à 95 %	Résolu	4 Go → 8 Go, `swappiness` 60 → 10
2	Allègement JVM Komga + StirlingPDF	Appliqué	Komga −292 Mio (−41 %), Stirling borné à 2 Go
3	OCR StirlingPDF impossible sur PNG	Résolu	Cause trouvée + script de contournement
4	Connexion Claude Code sur le VPS Alteris	Résolu	Identifiants recopiés depuis le VPS Jux

---

1. Swap saturé — le vrai diagnostic

Symptôme

Swap à **3,8 / 4,0 Go (95 %)**, alors que ~4,6 Gi de RAM restaient disponibles. Il se remplissait nuit après nuit et ne se vidait jamais.

Ce que ce n'était PAS

Un manque de mémoire. `vmstat 2 3` montrait `si`/`so` ≈ 0 : **aucun thrashing**, rien ne ramait. C'est le point contre-intuitif de ce diagnostic — un swap plein n'est pas synonyme de machine à genoux.

Le risque réel était ailleurs : il ne restait que **253 Mio de swap libre**. En cas de pic, l'OOM killer aurait commencé à tuer des containers.

Cause

`vm.swappiness = 60` (valeur par défaut). Pendant les batchs de nuit — backup 2h, scan Komga 4h (230 lignes de scan cette nuit-là), scans Audiobookshelf 4h et Navidrome 4h30 — le noyau préférait pousser les pages anonymes des applications vers le swap plutôt que de réclamer son cache de pages.

Une fois en swap, ces pages **n'en ressortent jamais d'elles-mêmes** : elles n'y reviennent que si le processus les retouche. D'où une accumulation qui ne fait que croître, nuit après nuit.

Fixes appliqués

a) `vm.swappiness = 10`, persisté dans `/etc/sysctl.d/99-swap.conf`. Le noyau réclame désormais son cache de pages avant de swapper. C'est ce qui empêche l'accumulation de recommencer.

b) Second fichier de swap `/swapfile2` **de 4 Go** → swap total **8 Go**, persisté dans `/etc/fstab` (sauvegarde : `/etc/fstab.bak-260803`), validé par `sudo swapon -a` pour garantir un redémarrage sûr.

Méthode choisie délibérément : ajouter un second fichier **évite tout** `swapoff`, donc tout risque. Un `swapoff -a` aurait dû recharger 3,8 Go dans 4,6 Go disponibles — faisable mais serré, plusieurs minutes d'I/O intense, et un pic pendant l'opération aurait pu faire tuer un container.

État après intervention

	Avant	Après
Swap total	4,0 Go	**8,0 Go**
Swap libre	253 Mio	**4,2 Go**
`vm.swappiness`	60	**10**
Disque libre	39 Go	35 Go

Ce qui reste

Les **3,8 Go de pages froides restent en swap**. C'est assumé : elles ne coûtent rien en performance et se libéreront au fil des redémarrages. Un `docker restart` libère instantanément le swap du container concerné.

Top détenteurs au 2026-08-03 : **komga 954 Mio**, yourls-db 389, stirling-pdf 354, Immich-SERVER 212, Immich-DB 197, mealie 189, Kavita 158, jellyfin 131.

Le redémarrage de Komga rendra donc presque un giga à lui seul.

Commandes de diagnostic

```
swapon --show                    # taille et occupation de chaque zone
vmstat 2 3                       # si/so : y a-t-il un swap ACTIF ?
cat /proc/sys/vm/swappiness
```

Swap par container :

```
cat /sys/fs/cgroup/system.slice/docker-$(docker inspect NOM --format
'{{.Id}}').scope/memory.swap.current
```

Swap par processus :

```
for p in /proc/[0-9]*; do s=$(awk '/^VmSwap:/{print $2}' $p/status 2>/dev/null); \
  [ -n "$s" ] && [ "$s" -gt 0 ] && echo "$s $(tr -d '\0' < $p/cmdline | cut -c1-60)"; done |
sort -rn | head
```

À surveiller

Le lendemain matin, après une nuit de batchs : si le swap utilisé a peu bougé, `swappiness=10` a fait son travail. S'il remonte malgré tout, la piste suivante est de réduire les consommateurs réels plutôt que de rejouer sur les réglages noyau.

---

2. Allègement JVM Komga + StirlingPDF

Appliqué le 2026-08-02 à 22h, après vérification du backup quotidien (BILAN 5/5 OK).

Service	Avant	Après	Résultat
Komga (stack 7)	`-Xmx4g -Xms2g -XX:MaxRAMPercentage=70.0`, limits 4500M / reservations 3000M, 707 Mio	`-Xmx1g`, limits 1500M / reservations 256M	**415 Mio (−292, −41 %)**
StirlingPDF (stack 89)	JVM sans limite, 722 Mio	`-Xmx512m`, limits 2g	785 Mio — **pas de réduction**

Bilan hôte : swap 3,7 → 2,2 Go juste après l'opération, mémoire disponible 4,4 → 4,6 Gi. L'essentiel du gain vient de Komga : c'est le `-Xms2g` qui pré-allouait 2 Go de heap au démarrage.

Pourquoi StirlingPDF ne baisse pas

À dire franchement plutôt que de maquiller le résultat : le process `java` conserve **799 Mio de RSS** malgré `-Xmx512m`. Le heap n'est qu'une partie du total (metaspace, code cache, piles de threads), et Stirling lance en plus LibreOffice (`soffice.bin`, ~98 Mio), `unoserver` et `Xvfb` pour ses conversions.

Le bénéfice obtenu n'est pas une baisse mais un **plafond** : la consommation ne peut plus déborder, alors qu'elle était non bornée.

La limite portée de 1 Go à 2 Go — décision critique

Le plan initial prévoyait 1 Go. Il a été corrigé **avant** application, grâce au chantier OCR mené le même jour : l'OCR fait tourner ocrmypdf et tesseract comme processus **Python, hors JVM**. `-Xmx` ne les borne pas — seul le plafond cgroup du container s'applique.

Pic mesuré : **894 à 969 Mio pour une seule page A4 300 dpi**. Un plafond de 1 Go aurait provoqué un OOM kill dès le premier OCR. La version 1 Go est conservée en `/home/debian/stirling-compose-new.yml.orig1g` mais **ne doit pas être utilisée**.

Vérifications après déploiement

- Komga répond 200, API `/api/v1/libraries` renvoie bien les bibliothèques
- StirlingPDF : `compress-pdf` OK (workflows komga-pdf et nas_geo_compress) **et** OCR OK (page A4 300 dpi, pic 969 Mio, aucun OOM kill, aucun redémarrage)

Piège de déploiement

Les projets compose s'appellent `7` **et** `89` — les IDs de stack Portainer — et non `komga` / `stirlingpdf`. La procédure documentée initialement aurait échoué dessus.

```
sudo docker compose -p 7 -f /var/lib/docker/volumes/portainer_data/_data/compose/7/docker-compose.yml up -d
```

---

3. OCR StirlingPDF impossible sur les images PNG

Symptôme

HTTP 500 sur tout OCR d'image PNG. L'interface web n'affiche **aucun détail exploitable** — le diagnostic passe obligatoirement par `docker logs stirling-pdf`.

Deux refus cumulés d'ocrmypdf 17.4.0

1. `UnsupportedImageFormatError: The input image has an alpha channel.` — tout PNG avec transparence
2. Une fois l'alpha retiré, un **second** blocage apparaît : `DpiError: Input file is an image, but has no resolution (DPI) in its metadata.` StirlingPDF ne transmet pas `--image-dpi` à ocrmypdf

Le point à retenir : les erreurs sont **séquentielles**. Corriger la transparence seule ne suffit pas, le DPI bloque juste après — ce qui explique pourquoi aucun réglage de l'interface ne pouvait s'en sortir.

Solution sans outil (interface web)

Convert → **Image to PDF**, puis OCR sur le PDF obtenu. Les deux problèmes disparaissent puisque l'entrée n'est plus une image. Validé en test, HTTP 200.

Solution scriptée

`Jux-scripts/Stirling-OCR/ocr_image.py` — aplatit l'alpha sur fond blanc et force 300 dpi avant envoi.

```
py "D:\Syncthing\Jux_univers\Jux-scripts\Stirling-OCR\ocr_image.py" mon_image.png
```

Accepte un fichier ou un dossier. Options : `-l fra+eng`, `-o sortie.pdf`, `--dpi 400`, `--skip-text`. Testé depuis le PC Windows **et** depuis le VPS ; texte restitué à l'identique.

Chemin sur le VPS : `/home/debian/Documents/Jux_univers/Jux-scripts/` (et non `~/Syncthing/...`).

Détails techniques

- Endpoint : `POST /api/v1/misc/ocr-pdf` — multipart `fileInput`, champs `languages` (liste), `ocrType` (`Force-OCR` / `skip-text`), `outputType`
- Langues tesseract installées (6) : `fra`, `eng`, `deu`, `por`, `chi_sim`, `osd`
- Versions : ocrmypdf 17.4.0, tesseract 5.3.4

---

4. Connexion Claude Code sur le VPS Alteris

Symptôme

Au lancement de `claude` en SSH (PuTTY / CMD / PowerShell), Claude Code demande de se connecter et affiche une URL. Impossible de la copier : `Ctrl+C`, clic droit, rien ne fonctionnait. La mention verte « copied » s'affichait pourtant.

Cause

Le « copied » venait de Claude Code **tournant sur le VPS** : il copiait l'URL dans le presse-papier du serveur Linux, qui n'a aucun lien avec celui du PC Windows à travers SSH. Le presse-papier Windows restait donc vide, d'où le collage à blanc dans le navigateur.

À noter au passage : dans un terminal SSH, `Ctrl+C` envoie un signal d'interruption au programme — il peut tuer le processus de login au lieu de copier.

Cause réelle du re-login

Le fichier `~/.claude/.credentials.json` d'Alteris avait `expiresAt: 0` — jeton invalide. Le VPS Jux, lui, disposait d'identifiants sains (compte Pro, refresh token valide).

Solution retenue

Recopier les identifiants depuis le VPS Jux, commande lancée **depuis Alteris** :

```
scp debian@51.77.141.54:.claude/.credentials.json ~/.claude/.credentials.json && chmod 600 ~/.claude/.credentials.json
```

Vérifié : 504 octets, permissions 600, jeton complet. Claude Code ne redemande plus de login et rafraîchit le token seul au démarrage (l'access token expiré n'est pas un problème, le refresh token est valide).

Conséquence : les deux VPS partagent désormais le même jeu d'identifiants. Si l'un redemande un login, refaire le même `scp` depuis celui qui fonctionne encore.

Alternative si les deux tombent : `claude setup-token` depuis le PC Windows (le navigateur s'ouvre en local), puis `export CLAUDE_CODE_OAUTH_TOKEN=...` dans le `~/.bashrc` du serveur.

---

5. Découverte annexe — port PostgreSQL 5432 bloqué sur Alteris

`publier_recherche.py` échoue sur sa partie PostgreSQL : `Connection timed out (10060)`.

- ufw est **actif** sur Alteris **sans règle pour 5432**
- Le container `alteris_postgis` tourne et `pg_isready` répond en local
- Vérifié : le port est injoignable depuis le PC **et** depuis le VPS Jux

Contournement sans toucher au pare-feu — exécuter le SQL en local sur Alteris :

```
pscp fichier.sql debian@79.137.14.202:/tmp/
sudo docker cp /tmp/fichier.sql alteris_postgis:/tmp/i.sql
sudo docker exec alteris_postgis psql -U alteris_admin -d alteris_geo -f /tmp/i.sql
```

Rouvrir le port exposerait à nouveau une base de données directement sur internet. Décision à prendre sciemment, surtout après l'incident du 2026-06-09.

Fichiers, sauvegardes et rollbacks

Élément	Chemin	Rôle
Compose Komga (avant)	`/var/lib/docker/volumes/portainer_data/_data/compose/7/docker-compose.yml.bak-260802`	Rollback Komga
Compose Stirling (avant)	`/var/lib/docker/volumes/portainer_data/_data/compose/89/docker-compose.yml.bak-260802`	Rollback Stirling
Version Stirling 1 Go	`/home/debian/stirling-compose-new.yml.orig1g`	**Ne pas utiliser** (OOM sur OCR)
fstab (avant)	`/etc/fstab.bak-260803`	Rollback swap
Réglage swappiness	`/etc/sysctl.d/99-swap.conf`	`vm.swappiness = 10`
Second swap	`/swapfile2` (4 Go)	Marge de swap
Script OCR	`Jux-scripts/Stirling-OCR/ocr_image.py` + `README.md`	Contournement OCR images

Rollback JVM : restaurer le `.bak-260802` de la stack et redéployer avec `sudo docker compose -p {7|89} -f ... up -d`.

Références

- Page **177** — 00_Récapitulatif : section « 260802 - Allègement JVM Komga + StirlingPDF »
- Page **211** — REX StirlingPDF : entrée `RCH-20260802-0001` (également en base `recherches` sur Alteris)
- Page **208** — 25_StirlingPDF
- `CLAUDE.md` du projet : sections « Swap », « Alléger Komga », « Alléger StirlingPDF », StirlingPDF/OCR

Revision #2
Created 2026-08-03 06:15:03 UTC by Julien
Updated 2026-08-03 06:17:37 UTC by Julien