25_StirlingPDF

RCH-20260802-0001 — Echec OCR sur images PNG — deux refus cumules d'ocrmypdf (canal alpha + DPI absent)

Date	2026-08-02
Service	StirlingPDF
Contexte	Julien signale une erreur HTTP 500 lors d'un OCR sur une image PNG via StirlingPDF. L'interface web n'affiche aucun detail exploitable, le diagnostic passe par les logs du container.
Outils utilisés	docker logs stirling-pdf, API /api/v1/misc/ocr-pdf, Pillow, tesseract --list-langs
Requête	`POST /api/v1/misc/ocr-pdf (multipart fileInput, languages=[fra], ocrType=Force-OCR) sur PNG RGBA, puis sur variantes aplaties et converties, avec mesure docker stats en parallele.`

Résultats

Deux erreurs distinctes et cumulees dans ocrmypdf 17.4.0, revelees seulement dans les logs : (1) UnsupportedImageFormatError — The input image has an alpha channel ; (2) une fois l'alpha retire, DpiError — Input file is an image, but has no resolution (DPI) in its metadata. StirlingPDF ne transmet pas l'option --image-dpi a ocrmypdf, donc tout PNG sans DPI dans ses metadonnees echoue. Deux contournements valides en test : voie interface web (Convert > Image to PDF puis OCR sur le PDF, HTTP 200) et voie scriptee (aplatissement de l'alpha sur fond blanc + DPI force a 300, HTTP 200). Texte reconnu restitue a l'identique sur une page de test en francais. Mesure memoire : pic a 894 Mio pour une seule page A4 300 dpi.

Points clés

- L'interface web ne remonte pas la cause : le diagnostic exige docker logs stirling-pdf
- Les deux erreurs sont sequentielles — corriger l'alpha seul ne suffit pas, le DPI bloque ensuite
- Convertir l'image en PDF avant l'OCR neutralise les deux problemes sans outil local
- 6 langues tesseract installees : fra, eng, deu, por, chi_sim, osd
- ocrmypdf et tesseract sont des process Python hors JVM : -Xmx ne les borne pas, seul le plafond cgroup du container s'applique
- La limite de 1 Go prevue pour l'allegement du 2026-08-02 aurait provoque un OOM kill pendant l'OCR — portee a 2 Go

Limites

Mesure realisee sur une page A4 300 dpi generee synthetiquement ; un document multi-pages ou un scan haute resolution consommera davantage. Le pic de 894 Mio est un plancher, pas un maximum.

Localisation des ressources

URLs	["https://spdf.juxjux.ovh", "https://spdf.juxjux.ovh/swagger-ui/index.html"]
Chemins	["D:\\Syncthing\\Jux_univers\\Jux-scripts\\Stirling-OCR\\ocr_image.py", "D:\\Syncthing\\Jux_univers\\Jux-scripts\\Stirling-OCR\\README.md", "/home/debian/stirling-compose-new.yml", "/home/debian/stirling-compose-new.yml.orig1g"]
IDs	["stack Portainer 89", "container stirling-pdf", "page BookStack 211"]
Auteurs / Source	—
Métadonnées	{"ocrmypdf": "17.4.0", "tesseract": "5.3.4", "endpoint_ocr": "POST /api/v1/misc/ocr-pdf", "pic_memoire_ocr": "894 MiB / page A4 300 dpi", "limite_memoire_retenue": "2g"}

Tags

#StirlingPDF #OCR #ocrmypdf #tesseract #PNG #alpha #DPI #memoire #Docker

Suite / Pistes

Verifier apres l'application du compose allege (2026-08-02 19h) que compress-pdf ET l'OCR fonctionnent toujours avec memory=2g et -Xmx512m.

RCH-20260530-0001 — Mise en place API StirlingPDF pour compression PDF automatisée (workflow geo_loud)

Date	2026-05-30
Service	StirlingPDF
Contexte	Étape 3 du workflow geo_loud : automatiser la compression des 127 PDFs lourds (>75 Mo) déplacés dans \\NASMAISON\foxy\geo_loud\. Objectif : appeler l'API StirlingPDF depuis le PC, écraser chaque fichier avec sa version compressée, mettre à jour la DB SQLite. Fichier de test : 000209_L'inde.pdf (314 Mo).
Outils utilisés	StirlingPDF API REST (JWT Bearer) | curl.exe | plink (PuTTY) pour SSH nginx | Python 3.14 + requests | SQLite nas_geographie.db
Requête API	`POST https://spdf.juxjux.ovh/api/v1/misc/compress-pdf — multipart/form-data champ fileInput, header Authorization: Bearer <token>`

Résultats

Auth JWT résolue. nginx corrigé (client_max_body_size 500M). Test de compression réussi sur 000209_L'inde.pdf : 314 Mo → 311.7 Mo (0.7% de gain — PDF déjà optimisé en images JPEG). Script `nas_geo_compress.py` opérationnel pour traitement en batch de tous les fichiers.

Points clés

- **Auth imposée :** malgré `DOCKER_ENABLE_SECURITY=false` dans la stack Docker, StirlingPDF impose un compte admin au premier lancement. Compte créé : **admin / Motdepasse18!**
- **Endpoint de login :** `POST /api/v1/auth/login` avec body JSON `{"username":"admin","password":"Motdepasse18!"}` → réponse : JWT dans `.session.access_token`, valable 24h
- **Utilisation du token :** header `Authorization: Bearer <token>` sur chaque appel API
- **nginx :** client_max_body_size par défaut = 1 Mo → erreur 413 sur gros fichiers. Correction : ajout de `client_max_body_size 500M;` dans `/etc/nginx/sites-enabled/spdf.juxjux.ovh` (via plink/SSH), puis `sudo nginx -s reload`
- **Endpoint compression :** `POST /api/v1/misc/compress-pdf` — multipart avec champ `fileInput` — retourne le PDF compressé directement dans le corps de la réponse (binary)
- **Gain variable :** PDFs avec images JPEG déjà compressées → gain faible (<1%). PDFs texte/vectoriel ou images PNG → gain potentiellement important (30–70%)
- **PowerShell inadapté aux gros fichiers :** concaténation de byte arrays → OutOfMemoryException sur 314 Mo. Utiliser curl.exe ou Python requests
- **Spec API :** `/v3/api-docs` retourne le HTML de la SPA React. Documentation disponible dans la Swagger UI interactive : `https://spdf.juxjux.ovh/swagger-ui/index.html` (après login)

Limites

- Token JWT expire après 24h — pour des batchs très longs, prévoir un re-login automatique
- La compression StirlingPDF n'expose pas de paramètre de niveau dans cette version — niveau par défaut uniquement
- Le fichier transite en RAM sur le VPS pendant le traitement — fichiers >500 Mo potentiellement problématiques selon la RAM disponible

Script nas_geo_compress.py

Emplacement : `C:\Users\eliob\.claude\nas_geo_compress.py`
Lancement : `py C:\Users\eliob\.claude\nas_geo_compress.py`
Dépendances : `pip install requests` (sqlite3 et os natifs Python)

Le script ajoute automatiquement les colonnes `taille_compresse_octets` et `date_compression` à la table `fichiers` si elles n'existent pas encore.

```
"""
Étape 3 du workflow geo_loud : compression PDF via StirlingPDF.
- Lit les fichiers statut='deplace' dans nas_geographie.db
- Envoie chaque PDF à l'API StirlingPDF
- Écrase le fichier d'origine avec la version compressée
- Met à jour la DB (taille_compresse_octets, date_compression)
```

```
"""
import sqlite3
import requests
import os
from datetime import datetime

DB_PATH = r"C:\Users\eliob\Documents\nas_geographie.db"
GEO_LOUD = r"\\NASMAISON\foxy\geo_loud"
STIRLING_URL = "https://spdf.juxjux.ovh"
STIRLING_USER = "admin"
STIRLING_PASS = "Motdepasse18!"


def get_token():
    r = requests.post(
        f"{STIRLING_URL}/api/v1/auth/login",
        json={"username": STIRLING_USER, "password": STIRLING_PASS},
        timeout=30,
    )
    r.raise_for_status()
    return r.json()["session"]["access_token"]


def init_db_columns(conn):
    cur = conn.cursor()
    cols = {row[1] for row in cur.execute("PRAGMA table_info(fichiers)")}
    if "taille_compresse_octets" not in cols:
        cur.execute("ALTER TABLE fichiers ADD COLUMN taille_compresse_octets INTEGER")
    if "date_compression" not in cols:
        cur.execute("ALTER TABLE fichiers ADD COLUMN date_compression TEXT")
    conn.commit()


def get_pending(conn):
    cur = conn.cursor()
    cur.execute("""
        SELECT id, nom, chemin_geo_loud, taille_octets
        FROM fichiers
        WHERE statut = 'deplace'
```

```
          AND extension IN ('.pdf', '.PDF')
          AND (date_compression IS NULL)
        ORDER BY taille_octets DESC
    """)
    return cur.fetchall()


def compress_file(token, filepath):
    with open(filepath, "rb") as f:
        r = requests.post(
            f"{STIRLING_URL}/api/v1/misc/compress-pdf",
            headers={"Authorization": f"Bearer {token}"},
            files={"fileInput": (os.path.basename(filepath), f, "application/pdf")},
            timeout=600,
            stream=True,
        )
    r.raise_for_status()
    return r.content


def update_db(conn, file_id, taille_compresse):
    conn.execute("""
        UPDATE fichiers
        SET taille_compresse_octets = ?, date_compression = ?
        WHERE id = ?
    """, (taille_compresse, datetime.now().isoformat(), file_id))
    conn.commit()


def fmt_mo(octets):
    return f"{octets / 1_048_576:.1f} Mo"


def main():
    print("Connexion à StirlingPDF...")
    token = get_token()
    print("Token JWT OK")

    conn = sqlite3.connect(DB_PATH)
```

```
init_db_columns(conn)
pending = get_pending(conn)
print(f"{len(pending)} fichier(s) PDF à compresser\n")

if not pending:
    print("Rien à faire.")
    conn.close()
    return

total_avant = sum(r[3] for r in pending)
total_apres = 0
ok = 0
errors = []

for i, (file_id, nom, chemin_geo_loud, taille_avant) in enumerate(pending, 1):
    filepath = chemin_geo_loud or os.path.join(GEO_LOUD, nom)
    if not os.path.exists(filepath):
        print(f"[{i}/{len(pending)}] ABSENT  {nom}")
        errors.append((nom, "fichier absent"))
        continue

    taille_reel_avant = os.path.getsize(filepath)
    print(f"[{i}/{len(pending)}] {nom}")
    print(f"    avant : {fmt_mo(taille_reel_avant)}", end=" ", flush=True)

    try:
        compressed = compress_file(token, filepath)
        taille_apres_compression = len(compressed)
        gain = (taille_reel_avant - taille_apres_compression) / taille_reel_avant * 100

        tmp = filepath + ".tmp"
        with open(tmp, "wb") as f:
            f.write(compressed)
        os.replace(tmp, filepath)

        update_db(conn, file_id, taille_apres_compression)
        total_apres += taille_apres_compression
        ok += 1
        print(f"-> après : {fmt_mo(taille_apres_compression)} (gain {gain:.1f}%)")
```

```
        except Exception as e:
            print(f"-> ERREUR : {e}")
            errors.append((nom, str(e)))
            tmp = filepath + ".tmp"
            if os.path.exists(tmp):
                os.remove(tmp)

    conn.close()

    print(f"\n{'='*50}")
    print(f"Terminé : {ok}/{len(pending)} compressés")
    if ok:
        gain_total = (total_avant - total_apres) / total_avant * 100
        print(f"Volume avant : {fmt_mo(total_avant)}")
        print(f"Volume après : {fmt_mo(total_apres)}")
        print(f"Gain total   : {gain_total:.1f}%  ({fmt_mo(total_avant - total_apres)}
récupérés)")
    if errors:
        print(f"\nErreurs ({len(errors)}) :")
        for nom, msg in errors:
            print(f"  {nom} : {msg}")


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

Localisation des ressources

URLs	https://spdf.juxjux.ovh (UI) | https://spdf.juxjux.ovh/swagger-ui/index.html (API docs, après login)
Chemins	Script : C:\Users\eliob\.claude\nas_geo_compress.py | DB : C:\Users\eliob\Documents\nas_geographie.db | Fichiers source : \\NASMAISON\foxy\geo_loud\ | nginx vhost : /etc/nginx/sites-enabled/spdf.juxjux.ovh
IDs	Stack Portainer : stirlingpdf (ID 89) | Container : stirling-pdf | Port interne : 8090

Métadonnées	{"fichiers_a_traiter": 127, "volume_total_go": 13.577, "test_fichier": "000209_L-inde.pdf", "test_avant_mo": 314, "test_apres_mo": 311.7, "test_gain_pct": 0.7, "nginx_max_body": "500M", "token_duree_h": 24}

Tags

#StirlingPDF #compression #PDF #geo_loud #API #JWT #nginx #Python #NAS #workflow

Suite / Pistes

1/ Lancer le script sur l'ensemble des 127 fichiers — surveiller le gain réel par catégorie de PDF.
2/ Mettre à jour la page BookStack 210 avec les colonnes DB ajoutées (taille_compresse_octets, date_compression) et le bilan de compression une fois le batch terminé.
3/ Si le gain moyen est insuffisant, explorer d'autres niveaux de compression StirlingPDF ou un outil alternatif (Ghostscript direct).
4/ Après compression de tous les fichiers : lancer l'étape 4 (restauration) via nas_geo_loud.py restaurer.

Revision #5
Created 2026-05-30 20:29:06 UTC by Julien
Updated 2026-08-02 08:28:54 UTC by Julien