Sécurisation et performance server

Le diagnostic est complet. Voici ce que j'ai trouvé.

Diagnostic — ce qui rame et pourquoi

Load average : 6.04 sur 6 CPU → le serveur est saturé.

Problème n°1 — Staging tournant en parallèle du prod (le principal coupable)

Tu as deux environnements complets qui tournent simultanément :
- /opt/easyapps/prod/ → prod
- /opt/easyapps/staging/ → staging

Chacun a son propre MySQL, Redis, Celery workers, API Python. Résultat :

Service	Instances actives	RAM consommée
MySQL (mysqld)	3	~1.4 Go
Redis	3	~215 Mo
Celery workers	26	~2.5 Go
API (run_hosted.py)	4	~380 Mo

Problème n°2 — Celery sans limite de concurrence

Le docker-compose.hosted.yml lance Celery sans --concurrency. Par défaut, Celery ouvre autant de workers que de CPU. Sur 6 CPU × 2 envs + overhead = 26
processus Python qui tournent en permanence.

Problème n°3 — Health check qui spamme le CPU

```
healthcheck:
  test: ["CMD", "python", "-c", "import json, urllib.request; ..."]
  interval: 10s    # ← toutes les 10 secondes
```

Ça spawne un nouveau process Python toutes les 10s. J'en ai vu deux tourner à 100% CPU au

moment du diagnostic.

Problème n°4 — Pas de swap

Si la RAM sature → OOM killer, pas de filet de sécurité.

Ce que je te recommande (par ordre de priorité)

1. Éteindre staging si personne ne s'en sert activement → gain immédiat ~40% de charge
2. Limiter la concurrence Celery à 2-4 workers max
3. Allonger l'interval du health check à 30s minimum
4. Ajouter du swap (2Go minimum)

Voilà, j'ai ce qu'il faut pour te donner une réponse claire avant d'écrire la moindre ligne.

Situation réelle du déploiement

Ce qui est dans git (sera écrasé à chaque deploy) :
- docker-compose.vps.yml ✗
- docker-compose.hosted.yml ✗
- lib.sh, tous les scripts release ✗

Ce qui est gitignore (persistera) :
- .env.staging, .env.production ✓
- Fichiers non trackés ✓

Point important : les fichiers staging.target.env et production.target.env n'existent pas — les scripts de release automatisés ne sont pas encore
configurés. Le staging a probablement été lancé à la main via docker compose up. Ça simplifie les choses.

La bonne approche

Créer un fichier override gitignored par env, et modifier la commande de lancement pour l'inclure :

/opt/easyapps/staging/app/current/docker-compose.staging-limits.yml ← gitignored, persistant
/opt/easyapps/prod/app/current/docker-compose.prod-limits.yml ← gitignored, persistant

Docker Compose supporte plusieurs -f : la commande devient :
docker compose -f docker-compose.vps.yml -f docker-compose.staging-limits.yml up -d

Les limits sont locales au serveur, pas dans le repo de relyo. Si relyo refait un git pull et relance, il doit juste utiliser la même commande (qu'on
peut documenter dans un README local).

Tu veux que je :

1. Crée les deux fichiers override (staging + prod) avec les limits appropriées
2. Les ajoute au .gitignore
3. Te montre la commande exacte pour (re)lancer chaque env avec les limits

Les valeurs que je propose :

Service	Staging	Prod
MySQL	0.5 CPU / 512 Mo	1 CPU / 1 Go
Redis	0.2 CPU / 128 Mo	0.3 CPU / 256 Mo
API	0.5 CPU / 512 Mo	1 CPU / 1 Go
Worker Celery (--concurrency=2)	0.5 CPU / 512 Mo	1 CPU / 1 Go
Beat	0.1 CPU / 128 Mo	0.2 CPU / 256 Mo
Frontend	0.2 CPU / 128 Mo	0.3 CPU / 256 Mo

Revision #1
Created 2026-05-13 09:43:53 UTC by Nicolas Farrie
Updated 2026-05-13 16:24:23 UTC by Nicolas Farrie