Admin serveurs et applis ...

- Contabo lepont34.fr
  - Sécurisation et performance server
  - sécurisation - plan 2
- Principes d'admin et sécurité

Contabo lepont34.fr

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

sécurisation - plan 2

Context

Suite à la session du 2026-05-13 (migration /srv/docker, nginx natif, SSL). Le serveur est stable mais des tâches de sécurité restent à faire. L'objectif
de cette session : hardening SSH, fail2ban, nettoyage volumes Docker orphelins, script de backup MySQL.

Problème immédiat identifié : conflit dans la config SSH. Le drop-in 99-relyo-hardening.conf a PermitRootLogin prohibit-password mais le fichier
principal a PermitRootLogin no. La règle OpenSSH "première occurrence gagne" fait que le drop-in l'emporte → prohibit-password effectif au lieu de no.
C'est pour ça que le serveur répond à une tentative root avec un prompt password.

Étape 1 — Corriger le conflit SSH (2 min)

Fichier : /etc/ssh/sshd_config.d/99-relyo-hardening.conf

Changer PermitRootLogin prohibit-password → PermitRootLogin no.

Contenu final du fichier :
PubkeyAuthentication yes
PasswordAuthentication no
KbdInteractiveAuthentication no
ChallengeResponseAuthentication no
PermitRootLogin no

Puis : sudo systemctl reload ssh

Côté client (local) : dans ~/.ssh/config sur le poste nicolas@asrock, vérifier que l'alias contabo02 a bien User nicolas (pas User root).

Vérification : ssh -v contabo02 doit montrer Permission denied immédiat sans prompt password si User est root, ou connexion directe si User est nicolas.

Étape 2 — Fail2ban (20 min)

Installation

sudo apt install -y fail2ban

Configuration /etc/fail2ban/jail.local

[DEFAULT]
bantime = 10m
findtime = 10m
maxretry = 5
bantime.increment = true
bantime.factor = 1
bantime.formula = ban.Time * (1<<(ban.Count if ban.Count<20 else 20)) * banFactor
ignoreip = 127.0.0.1/8 ::1

[sshd]
enabled = true
port   = ssh
filter   = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = 1h

Les bans progressifs (bantime.increment) doublent la durée à chaque récidive : 1h → 2h → 4h → ... → plusieurs semaines.

Rapport quotidien (cron)

Script /usr/local/bin/fail2ban-report.sh :
#!/bin/bash
echo "=== Fail2ban report $(date) ==="
sudo fail2ban-client status sshd
sudo fail2ban-client status sshd | grep "Banned IP"

Cron dans /etc/cron.d/fail2ban-report : exécution à 8h00 chaque matin, sortie loguée dans /var/log/fail2ban-report.log.

Note AbuseIPDB : nécessite une clé API. À activer seulement si Nicolas a un compte AbuseIPDB. Sinon, on le laisse de côté.

Vérification : sudo fail2ban-client status sshd doit montrer le jail actif.

    │ Note AbuseIPDB : nécessite une clé API. À activer seulement si Nicolas a un compte AbuseIPDB. Sinon, on le laisse de côté.   │
    │   │
    │ Vérification : sudo fail2ban-client status sshd doit montrer le jail actif.
                        │
    │   │
    │ ---   │
    │ Étape 3 — Nettoyage volumes Docker orphelins (5 min)
                        │

```
    |                                                
```

```
                        |
    |     exit 1
|
    | fi                                                                                  |
    |                                                                                     |
    | docker exec "$CONTAINER" mysqldump -u root -p"${MYSQL_ROOT_PASSWORD}" --all-
databases \                                                                             |
    |     > "$BACKUP_DIR/prod_${DATE}.sql"
                  |
    |                                                                                     |
    | # Garder seulement les 7 derniers dumps
                   |
    | ls -t "$BACKUP_DIR"/prod_*.sql | tail -n +8 | xargs -r rm
                    |
    |                                                                                     |
    | echo "Backup OK: $BACKUP_DIR/prod_${DATE}.sql"
                          |
    |                                                                                     |
    | Le MYSQL_ROOT_PASSWORD est lu depuis l'env du container (passé via .env.vps). À vérifier.
                                          |
    |                                                                                     |
    | Nettoyage immédiat : supprimer /tmp/prod-dump.sql qui contient des données de prod en
clair.                                                  |
    |                                                                                     |
    | Vérification : exécuter le script manuellement, vérifier que le fichier SQL est créé.
                                  |
    |                                                                                     |
    | ---                                                                                 |
    | Ordre d'exécution
      |
    |                                                                                     |
    | 1. Correction SSH + reload (sans risque, ne coupe pas les sessions actives)
                                  |
    | 2. Fail2ban install + config
        |
    | 3. Nettoyage volumes (après confirmation docker volume ls)
                          |
    | 4. Script backup MySQL (+ suppression /tmp/prod-dump.sql)
                           |
    |                                                                                     |
    | Ce qu'on ne fait pas cette session
          |
    |                                                                                     |
    | - Migration Odoo/Bookstack (chantier séparé)
                  |
    | - AbuseIPDB (nécessite clé API externe)
                |
```

│ - Restriction sudo pour samuel (à décider avec samuel)
                    │
│ - Audit secrets git relyo (chantier séparé)

Bilan de la session — tout est fait :

Tâche	Statut
SSH : PermitRootLogin no effectif	✅
Fail2ban + bans progressifs + rapport 8h	✅
Rebuild images Docker (prod + staging)	✅
Migration médias (25 fichiers) vers nouveau volume	✅
Suppression volumes orphelins easyapps-prod_*	✅
Script backup MySQL + cron 2h	✅

Il reste pour les prochaines sessions :
- Migration Odoo 16 + Bookstack depuis 82.66.134.226
- Restreindre samuel du groupe sudo
- Audit secrets dans le code git relyo
- Vérification renouvellement auto certbot

Je mets la mémoire à jour.

Principes d'admin et sécurité

Bonjour .. je voudrais valider les bonnes pratique en matière d'administration de serveur hébergé (vps ou baremetal) - Si on est 2 admins - Doit-t-on proscrire l'accès en root, même par clé rsa ?- si oui, doit-t-on avoir un compte par admin pour les logs et la traçabilité ?- dans ce cas, il sont juste sudoers, pas root .. Est-ce correcte ?

Voici les bonnes pratiques en matière d’administration de serveurs (VPS ou bare metal), surtout quand vous êtes plusieurs administrateurs :

---

1. Accès en root

À proscrire absolument, même par clé RSA.

- **Pourquoi ?** Même avec une clé RSA, un accès direct en root expose le serveur à des risques majeurs (erreur humaine, fuite de clé, attaque ciblée).
- **Alternative :** Utilisez toujours un compte utilisateur standard, puis `sudo` pour les commandes nécessitant des privilèges élevés.

---

2. Un compte par admin pour la traçabilité

Oui, chaque admin doit avoir son propre compte.

- **Pourquoi ?**
  - **Traçabilité** : Chaque action est liée à un utilisateur spécifique, ce qui facilite l’audit et la responsabilité.
  - **Sécurité** : Limite l’impact d’une compromission (une clé ou un mot de passe volé ne donne accès qu’à un seul compte).
- **Configuration :**
  - Chaque compte doit être ajouté au groupe `sudo` (ou `wheel` selon la distribution) pour pouvoir utiliser `sudo`.
  - Exemple de commande pour ajouter un utilisateur au groupe sudo :

bash

```
usermod -aG sudo nom_utilisateur
```

---

3. Configuration des droits sudo

Oui, les admins doivent être sudoers, pas root.

- **Fichier de configuration :** `/etc/sudoers` (à éditer avec `visudo`).
- **Exemple de ligne pour un admin :**
  plaintext

  ```
  nom_utilisateur ALL=(ALL:ALL) ALL
  ```

- **Bonnes pratiques :**
  - Limitez les commandes sudo si possible (principe du moindre privilège).
  - Activez les logs des commandes sudo (`/var/log/auth.log` ou `/var/log/secure` selon la distribution).

---

4. Autres recommandations

- **Clés SSH uniquement** : Désactivez l'authentification par mot de passe (`PasswordAuthentication no` dans `/etc/ssh/sshd_config`).
- **Mots de passe forts** : Même pour les comptes sudoers, utilisez des mots de passe robustes ou des clés SSH.
- **Mises à jour** : Maintenez le système et les outils (OpenSSH, sudo) à jour.
- **Sauvegardes** : Sauvegardez régulièrement les fichiers de configuration critiques (`/etc/sudoers`, `/etc/ssh/sshd_config`).

Résumé de votre cas

- **Accès root direct** : □ Non, même avec clé RSA.
- **Un compte par admin** : □ Oui, pour la traçabilité.
- **Sudoers uniquement** : □ Oui, avec configuration dans `/etc/sudoers`.