DOCUMENTATION CELLULE-PRO

Guide Administrateur

Manuel opérationnel DSI Cellule-PRO

cellule.ai · pro.cellule.ai
2026-05-18

Table des matières

Vue d'ensemble du dashboard

Fig. 1 : Vue d'ensemble du dashboard

0.1 Cellule-PRO — Guide Administrateur

Public : DSI, admin sys, intégrateur en charge du déploiement Cellule-PRO sur site client (Cabinet PME ou Corporate multi-sites).

Prérequis lecteur : Linux/Docker basique, réseau IP basique. Pas besoin de Python ni de connaissance LLM.

État : squelette créé session 2026-04-21 (chantier RAG Phase 6). Sections à compléter au fil des phases.

Sommaire

---

0. Aperçu du dashboard

Le dashboard administrateur est l'interface unique de pilotage du cluster. Tous les réglages (effectifs, fédération inter-sites, flags runtime, MoE, branding) y sont accessibles. Aucune ligne de commande n'est nécessaire pour l'usage quotidien.

0.1 Vue d'ensemble

Page d'accueil. Un seul écran pour comprendre l'état du cluster : capacité actuelle (postes employés contributeurs, serveurs proxy, sites connectés), configuration (effectifs déclarés, profil MoE, topologie réseau auto-détectée), et raccourcis vers les pages les plus utilisées.

0.2 Fédération — connexion inter-sites

Cellule-PRO supporte un déploiement multi-sites (siège + agences, plusieurs bâtiments d'un campus). Chaque site héberge un pool, et les pools s'authentifient mutuellement via signature Ed25519. Cette page liste les sites bondés et permet d'en ajouter un nouveau via token de rattachement à usage unique.

Page Fédération

Fig. 2 : Page Fédération

Page Répartition workers

Fig. 3 : Page Répartition workers

Page Topologie

Fig. 4 : Page Topologie

0.3 Répartition workers — effectifs déclarés

Permet de déclarer le nombre d'employés par site/succursale. Le pool virtuel utilise cette information pour calibrer ses limites internes (file d'attente, queue par employé) et afficher l'écart entre effectifs annoncés et workers réellement contributeurs.

0.4 Topologie

Vue cartographique de la latence inter-sites mesurée en continu. Le type de réseau est auto-détecté (LAN local <5 ms, métro multi-bâtiments 5-30 ms, WAN inter-villes >30 ms) et conditionne certains comportements du cluster (fanout RAG, smart routing).

0.5 Paramètres avancés

Toute option pilotable est ici, organisée par catégorie. Chaque paramètre dispose d'une description orientée DSI métier (pas de jargon LLM imposé) et d'une recommandation par profil. Les modifications du master sont propagées automatiquement aux satellites du cluster via gossip signé Ed25519.

0.6 Documentation

Les guides admin/API/sécurité sont accessibles directement depuis le dashboard, exportables en PDF brandé en un clic (cette page elle-même est issue de cette fonctionnalité).

---

1. Docker

1.1 Image

- Registre : `celluleai/pool-private:latest` (Docker Hub privé) ou `registry.cellule.ai/pool-private:lat` (registre auto-hébergé, cf. `project_todo_registry_distribution`)
- Tag release : `celluleai/pool-private:v<version>`
- Base : Debian slim + Python 3.12 + PostgreSQL client + pgvector runtime

Page Paramètres avancés

Fig. 5 : Page Paramètres avancés

Page Documentation

Fig. 6 : Page Documentation

1.2 docker-compose minimal

Référence : `src/cellule_pro/appliance/quickstart/docker-compose.yml` (pool unique, dev/POC) et `src/cellule_pro/appliance/cabinet/` (3-pool RAID LAN).

Pour un déploiement Cabinet 3-pool LAN :

```yaml
services:
  pg:
    image: pgvector/pgvector:pg16   # pgvector extension obligatoire
    volumes:
      - iamine-pg:/var/lib/postgresql/data
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password

  pool-a:
    image: celluleai/pool-private:latest
    depends_on: [pg]
    environment:
      CELLULE_ENTERPRISE: "1"
      POOL_NAME: cabinet-pool-a
      # ...autres env vars §1.3
    ports:
      - "18081:8080"

  pool-b: { ... }   # clone de pool-a avec POOL_NAME=cabinet-pool-b, port 18082
  pool-c: { ... }   # POOL_NAME=cabinet-pool-c, port 18083

  entry:
    image: nginx:alpine
    depends_on: [pool-a, pool-b, pool-c]
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "18080:8080"   # entrée unifiée cluster

volumes:
  iamine-pg:
```

`nginx.conf` type : voir `src/cellule_pro/appliance/entry/nginx.conf`.

1.3 Variables d'environnement

Critiques :

Variable	Valeur exemple	Description
`CELLULE_ENTERPRISE`	`1`	Active les plugins PRO (sinon = mode communautaire)
`POOL_NAME`	`cabinet-site1-pool-a`	Identifiant pool unique dans la fédération

Variable	Valeur exemple	Description
DATABASE_URL	postgresql://user:pwd@pg:5432/iamine	PostgreSQL + pgvector
IAMINE_EMBEDDER_PATH	~/.cache/iamine-embedder	Chemin MiniLM-L6-v2 pré-téléchargé (384d)

Fédération Ed25519 :

Variable	Description
FEDERATION_SELF_PRIVKEY_PATH	Clé privée Ed25519 locale pour signer mutations
FEDERATION_SELF_ATOM_ID	ID pool dans la table federation_peers

RAG project_documents : aucune env var requise — les constantes sont verrouillées dans document_rag.py (CHUNK_SIZE_TOKENS=512, MAX_DOCUMENT_SIZE_BYTES=50MB). Modification = re-compile, pas config.

À compléter : LICENSE_KEY, FORWARDING_ENABLED, tuning perfs.

1.4 Volumes persistants

Volume	Contenu	Critique
iamine-pg	PostgreSQL data (projets, memories, chunks, audit)	**OUI** — backup quotidien impératif
iamine-embedder-cache	Modèle MiniLM téléchargé (~90 MB)	Non (re-téléchargeable)
iamine-uploads	Binaires documents uploadés (storage_url)	Moyenne (ephemeral-acceptable + ré-uploadable)
iamine-logs	Journaux applicatifs	Non (rotation systemd)

Backup critique : iamine-pg vers stockage externe (Z2 / NAS / S3). Tests live DB du chantier RAG supposent un dump/restore fidèle — vérifier que pgvector est installé côté restore.

1.5 Quickstarts

- **Cabinet PME** (3 pools LAN RAID mono-site) — cf. src/cellule_pro/appliance/quickstart/
- **Corporate** (N pools multi-sites WAN) — cf. src/cellule_pro/appliance/cabinet/ (nommage hérité, générique)

1.6 Premier boot — wizard d'installation (chantier #16)

Au **premier démarrage** d'un pool Cellule-PRO (aucune row dans cluster_setup_state), toutes les routes non-whitelist sont **bloquées** par le middleware wizard_gate :

- Client curl / API → `503 JSON {error:"cluster_setup_pending", setup_url:"/setup/ui"}`
- Navigateur (Accept text/html) → `307 redirect /setup/ui`

Le tableau de bord admin n'est **pas accessible** tant que le wizard n'est pas complété. Seuls `/v1/status` (healthcheck Docker), `/docs`//`redoc` (Swagger) et `/setup/*` restent ouverts.

Les 7 écrans

1. **Identité** — nom entreprise, secteur (AVOCAT/GRAPHISME/HOPITAL/ COMPTABLE/ARCHITECT logo optionnel (PNG/JPG/SVG 500 KB, embedded dans la DB en BYTEA).
2. **Topologie** — liste dynamique des succursales. Un site = minimum requis. Advisory 2 pools LAN pour redondance RAID (cf. §4.5).
3. **Hardware** — workers CPU dormants activés + nombre de serveurs GPU/NPU + URL hub provisioning optionnelle.
4. **Modèles LLM** — sélection tiers (fast 9B / reasoning 35B / code 30B). Les wheels sont téléchargés **après** l'installation.
5. **Sécurité** — rétention audit log (6/12/24/120 mois ou illimité). Clés Ed25519 fédération générées automatiquement au premier boot.
6. **Compte admin** — email + mot de passe (12 caractères min).
7. **Récap + Go** — preview des defaults du secteur qui seront appliqués (retention, audit grade, template description), bouton de confirmation final.

Defaults secteur appliqués au complete

Secteur	retention_months_min	audit_grade	template
AVOCAT	120 (10 ans)	strict	“Dossier client :”
HOPITAL	240 (20 ans)	iso27001	“Patient :”
COMPTABLE	120 (10 ans fiscal)	strict	“Dossier comptable :”
ARCHITECTURE / INGENIERIE	60	basic	“Projet :”
GRAPHISME	24	basic	“Livrable client :”
CONSEIL	60	strict	“Mission :”
AUTRE	12	basic	“”

Admin-first : un choix explicite de l'admin (ex : `retention=12` sur AVOCAT) est toujours respecté. Les defaults remplissent uniquement les champs laissés NULL. Un warning est tracé en DB (`cluster_setup_state.sector_defaults_applied` JSONB) pour audit. Les obligations légales (archive avocat 10 ans, dossier médical 20 ans) restent de la responsabilité du client.

Persistance de l'état wizard

- Côté serveur : table `cluster_setup_state` single-row (CHECK id=1).
- Côté client : `localStorage` restaure l'état en cas de refresh accidentel (hors mot de passe admin, toujours effacé avant persist).

Comportement post-complete

- `completed_at` set en DB → cache middleware invalidé → routes applicatives débloquées immédiatement.

- Tentative `POST /setup/step/*` après complete → `409 Conflict` avec message “use re-open flow (chantier #17 licence)”.
- Compte admin créé dans `admin_users` (email + password hashé en Argon2id, chantier sécu 2026-04-27 PM).

Mot de passe admin oublié — reset CLI air-gap Doctrine air-gap : pas d’envoi email externe. L’admin DSI réinitialise le mot de passe directement dans le conteneur du pool via :

```
docker exec -it pool-z2-h python -m cellule_core admin-reset-password \
    --email=admin@example.local
# prompt: New password for admin@example.local: ********
# prompt: Confirm password: ********
# OK: password reset for admin@example.local
```

Le helper : - demande le nouveau password sur stdin (getpass, pas d’écho terminal) - hash en Argon2id via `passlib.hash.argon2` - met à jour `admin_users.password_hash` - aucune sortie réseau, fonctionne en air-gap absolu

Permissions requises : accès SSH au host + droit `docker exec` sur le conteneur pool. Aucun port HTTP exposé pour ce flux (vs les classiques “Forgot password” web qui ouvrent une attaque surface).

Migration des passwords legacy Au boot du pool, si la table `admin_users` contient encore des password en clair (image antérieure au chantier sécu 2026-04-27 PM), le pool re-hash automatiquement en Argon2id. L’opération est idempotente (skip si `$argon2` détecté en préfixe). Les admins existants peuvent continuer à se logger avec le même password — la migration est silencieuse.

Re-ouvrir le wizard Non disponible en v1. Prévu chantier #17 licence : le re-open nécessitera une nouvelle clé de licence valide + double confirmation admin. Cette contrainte protège l’infrastructure contre reconfiguration accidentelle.

Pour forcer un reset **en mode dev uniquement** : supprimer la row `cluster_setup_state` via SQL direct et redémarrer le pool. À ne jamais faire en production.

---

2. Worker

2.1 Qu’est-ce qu’un worker

Un worker = machine qui exécute un LLM (ou un batch d’embedding). Peut être un poste employé dormant (CPU) ou un serveur dédié (NPU/GPU). Principe : “compute dormant valorisé”. Cf. `project_vision_dormant_compute_pitch`.

Deux rôles complémentaires :

Rôle	Hardware typique	Charge
Worker LLM	poste dev / serveur GPU	inférence (chat completions, résumé)
Worker embedding	poste dormant CPU	MiniLM 384d batch (RAG project_documents, smart routing)

Un même poste peut servir les deux. Le pool central dispatche selon la disponibilité et la charge courante. **Decision #6 chantier RAG verrouillée** : le compute d'embedding est dispatché par le pool central, jamais local-only — c'est ce qui permet de valoriser les CPU dormants pour le RAG sans toucher aux workers LLM.

2.2 Provisioning

Le dashboard admin (chantier #14 Worker Provisioning) génère des **snippets prêts à coller** dans ton orchestrateur :

1. Ouvrir le hub admin → onglet "Workers"
2. Cliquer "Ajouter un worker" → choisir OS + hardware (CPU / GPU)
3. Copier les 3 snippets générés :
   - `config.json` (avec token de provisioning usage unique)
   - `install.sh` (téléchargement wheel + dépendances)
   - `start.sh` (commande de lancement + systemd unit)
4. Coller dans Ansible / Intune / Jamf / script manuel

Le provisioning code expire après usage ou 24h, peu importe lequel arrive en premier. Cf. `project_chantier_14_worker_provisioning`.

2.3 Services systemd

Unit file type `/etc/systemd/system/iamine-worker.service` :

```
[Unit]
Description=Cellule-PRO Worker
After=network.target

[Service]
Type=simple
User=iamine
WorkingDirectory=/opt/iamine
ExecStart=/opt/iamine/venv/bin/python -m iamine worker --auto
Restart=on-failure
RestartSec=10
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
```

Activation :

```
sudo systemctl daemon-reload
sudo systemctl enable --now iamine-worker
sudo systemctl status iamine-worker
sudo journalctl -u iamine-worker -f
```

Important : toujours lancer via systemd, jamais en shell direct (cf. `feedback_z2_proxy_systemd_check` — process orphelins = crash silencieux au reboot).

2.4 Troubleshooting

Worker ne se connecte pas

- Vérifier `systemctl status iamine-worker` (doit être `active (running)`)
- Vérifier la connectivité pool (port WebSocket ouvert, certificat TLS)
- `journalctl -u iamine-worker -n 100 --no-pager`

Process orphelin (cf. `feedback_z2_proxy_systemd_check`) Symptôme : le worker tourne en ligne de commande au lieu de systemd → crash silencieux au reboot. **Toujours lancer via systemd**, jamais en shell direct. Vérifier avec `ps -ef | grep iamine` qu'aucune instance n'est orpheline.

Document uploadé jamais indexé (RAG)

1. Vérifier `indexed_status` via GET /documents : `pending` / `skipped` / `failed`
2. Si `skipped` : mime_type hors whitelist (TXT / MD / PDF / DOCX / ODT) ou taille > 50 MB, ou texte extrait vide
3. Si `failed` : embedder indispo → `pip install sentence-transformers` + pré-télécharger MiniLM (`IAMINE_EMBEDDER_PATH` env var, défaut `~/.cache/iamine-embedder`)
4. Si `pending` depuis > 10 min sur un doc < 10 MB : pool central saturé → vérifier métriques CPU + queue
5. Pour extraction PDF/DOCX/ODT : libs soft-dep — installer manuellement si nécessaire (`pip install pypdf python-docx odfpy`)

Pool mort entre upload et fin d'ingest Si le pool hôte tombe entre le `POST /documents` et la fin de l'indexation background, le document reste en `indexed_status='failed'` avec binaire non traité. **V1 ne fait pas d'auto-recovery** : l'opérateur doit ré-uploader depuis la source d'origine, ou depuis un backup externe (Z2 / NAS client), puis renvoyer un `POST /documents` avec le même `content_hash` (l'idempotence DB supprime les anciens chunks orphelins si présents et ré-indexe).

Pour éviter ce scénario en prod multi-pool : garantir que l'upload atteint son pool hôte avant que le client quitte la session (ACK à l'upload, pas à la fin d'ingest).

2.5 Cycle de vie wheels — auto-upgrade zero-touch

Doctrine : DSI bump le pool, le parc workers se met à jour seul. Aucune intervention Admin requise sur les postes employés.

Mécanisme :

1. À chaque connexion d'un worker, le pool compare la version du wheel `cellule-pro` côté worker (`importlib.metadata.version`) avec celle embarquée dans son image.
2. Si le worker est en retard, le pool envoie un `welcome.upgrade` + pousse `command:self_update` (gated par cooldown 10 min/worker pour éviter les boucles si un worker reboot mal).
3. Le worker exécute :

   ```
   pip install --upgrade --index-url http://<pool>/pypi/ \
               --trusted-host <pool> cellule-pro
   ```

   Le `--trusted-host` est requis car le PyPI local du pool est servi en HTTP LAN — la doctrine §2.2 (zéro adhérence Internet runtime) interdit un certificat TLS publiquement signé sur ce service.
4. Le worker fait un `_restart_self()` atomique (PowerShell `start "" /B cmd /c` avec redirect stdio) et boote le nouveau wheel.
5. Au prochain handshake, les versions matchent — pas de re-push.

Toggle admin — `CELLULE_AUTO_SELF_UPDATE` (env var pool) : - Default `true` (doctrine §2.6 ON pour UX livrable). - `false` pour les sites qui veulent un push manuel (par ex. zone pré-prod isolée, validation manuelle imposée par audit).

Permissions — la ScheduledTask `cellule-worker` créée par l'one- liner installer (`/v1/install/worker.ps1` ou `worker.sh`) tourne en SYSTEM (Windows) / root (Linux). Le pip install peut donc écrire dans `C:\Program Files\Cellule\venv\` (Windows) ou `/opt/cellule/venv/` (Linux) sans erreur de permission. Un worker relancé manuellement par un user (Start-Process interactif) ne pourra pas se mettre à jour — utiliser `Start-ScheduledTask cellule-worker` ou `systemctl start cellule-worker` à la place.

Bootstrap — un parc qui tourne en wheel < 0.2.3 ne possède pas le fix `--trusted-host` dans son code self_update local : pip refusera l'index HTTP et l'upgrade fail silencieux. Premier upgrade manuel one-time requis :

Windows (PowerShell Admin) :

```
& "C:\Program Files\Cellule\venv\Scripts\python.exe" -m pip install \
  --upgrade --index-url http://<pool>:18095/pypi/ \
  --trusted-host <pool> cellule-pro
Start-ScheduledTask -TaskName cellule-worker
```

Linux :

```
sudo /opt/cellule/venv/bin/pip install --upgrade \
  --index-url http://<pool>:18095/pypi/ \
  --trusted-host <pool> cellule-pro
sudo systemctl restart cellule-worker
```

À partir de 0.2.3+, tous les upgrades suivants sont zero-touch.

Validation côté pool (logs) :

```
Worker Thor-7c8a outdated: v0.2.4 → v0.2.5
Worker Thor-7c8a: pushing self_update (v0.2.4 is 1 patches behind)
```

Validation côté worker (logs) :

```
Self-update: pip install --upgrade cellule-pro...
Self-update done, running discovery before restart...
Spawning decoupled relaunch via cellule-restart.bat
* CELLULE-PRO v0.2.5  ← nouvelle version au boot
```

2.6 Worker macOS (Apple Silicon M1/M2/M3/M4)

> **Statut rc92w** : code-ready, validation E2E en attente d'un Mac physique (1er POC client ou rental cloud Mac, ~30 min). Le path d'installation est implémenté et testé unitairement (12 tests pytest) mais aucun Mac n'a encore tourné un worker en production. Si tu es notre premier client Mac, signale-le pour qu'on documente la première install live.

Pré-requis : - Mac Apple Silicon (arm64) — M1, M2, M3, M4 (Pro/Max/Ultra inclus) - macOS 11 (Big Sur) ou plus récent - Compte admin local (sudo) - Connexion réseau au pool sur le LAN entreprise

Intel macOS x86_64 non supporté — parc rare en entreprise post-2020. Si tu en as, contacte support pour évaluer un chantier Phase 2 (~1h).

Installation one-liner (terminal macOS) :

```
curl -fsSL http://<pool-vip>:18095/v1/install/worker.sh | sudo bash
```

Le script auto-détecte `Darwin arm64` et : 1. Pull le runtime Python 3.12 portable (`python-build-standalone` aarch64-apple-darwin) depuis le pool — pas besoin de Homebrew 2. Crée un venv `/usr/local/opt/cellule/venv`, installe `cellule-pro` depuis `/pypi` du pool (air-gap strict) 3. Pull le wheel `llama-cpp-python` arm64 (Metal + Apple Accelerate inclus par défaut, pas de fallback CPU) 4. Génère le launchd plist `/Library/LaunchDaemons/com.cellule.worker.plist` et l'active via `launchctl bootstrap system` 5. Le service redémarre à chaque reboot, auto-restart si crash (équivalent `Restart=on-failure` systemd)

Ressources locales utilisées (différentes de Linux) :

Ressource	Path
Config	`/usr/local/etc/cellule/worker.json`
Install (venv + Python)	`/usr/local/opt/cellule/`
Modèles GGUF	`/usr/local/var/lib/cellule/models/`
Logs	`/Library/Logs/cellule-worker.log`
Plist launchd	`/Library/LaunchDaemons/com.cellule.worker.plist`

Statut + commandes utiles :

```
# Statut du service
sudo launchctl print system/com.cellule.worker

# Logs live
tail -f /Library/Logs/cellule-worker.log
```

```
# Restart manuel
sudo launchctl kickstart -k system/com.cellule.worker

# Arrêt
sudo launchctl bootout system/com.cellule.worker
```

Architecture UMA et `CTX_AUTO_CAP_BY_VRAM` :

Apple Silicon utilise une **mémoire unifiée** (UMA) : la même RAM sert au CPU et au GPU. Le worker Mac advertise sa RAM totale comme `vram_gb` (c'est physiquement ce que llama-cpp Metal a accès). Le pool applique le cap `CTX_AUTO_CAP_BY_VRAM` (cf. §1.x flag) en conséquence :

Mac	RAM/VRAM UMA	Cap typique pour modèle 4B Q4
M1 base	8 GB	ctx 8 192 (cap nécessaire)
M1 Pro / M2 base	16 GB	ctx 32 768
M2 Max	32-96 GB	ctx 131 072 (plafond, no-cap)
M3/M4 Ultra	64-192 GB	ctx 131 072 (plafond)

Le tier hardware reporté par `detect_hardware_class()` est `apple-m` (distinct de `gpu-small`/`gpu-large`), permettant à l'admin DSI d'assigner un modèle spécifique adapté à Apple Silicon via la page Modèles.

Désinstallation :

```
curl -fsSL http://<pool-vip>:18095/v1/install/worker-uninstall.sh | sudo bash
```

Stoppe le service launchd, retire le plist, supprime le venv + modèles GGUF (libère plusieurs GB sur disque). Logs préservés sous `/Library/Logs/` pour audit (clean manuel : `sudo rm /Library/Logs/cellule-work`

---

3. Proxy

3.1 Entry point unifié

Un seul DNS/IP → tout le cluster. Masque la topologie physique (N pools fédérés) derrière une URL unique.

- **Listen** : `:18080` (configurable via docker-compose ports)
- **Routage** :
  - `/v1/admin/*` → sticky session par IP client (cohérence cookies)
  - tout le reste → round-robin least_conn avec failover auto
- **Failover** : `max_fails=3, fail_timeout=30s` retire un pool malade de l'upstream pendant 30 s avant de re-tenter
- **WebSocket** : upgrade `Upgrade`/`Connection` préservé pour les events live du dashboard

Config de référence : `src/cellule_pro/appliance/entry/nginx.conf`. Ajuster les upstreams à ta topologie (ports `:18081/82/83` en général).

3.2 TLS et certificats

Recommandation : TLS terminé en **edge**, pas dans le container nginx interne. Options :

- **Cloudflare Tunnel** (testé prod cellule.ai, cf. `reference_cloudflare_tunnel_master`) — cert auto, rotation gérée
- **Reverse-proxy client** (Traefik, Caddy, nginx système) — avec Let's Encrypt auto-renew
- **VPN privé** (WireGuard, Tailscale) — pas de cert public, trafic chiffré niveau réseau

L'entry nginx interne peut rester en HTTP `:18080` si le trafic ne transite jamais par Internet sans couche TLS en amont. Sinon, ajouter `listen 8443 ssl` + certificat mounted.

3.3 WebSocket upgrade

Déjà présent dans `nginx.conf` de référence :

```
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

location /v1/admin/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade    $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_read_timeout 300s;
}
```

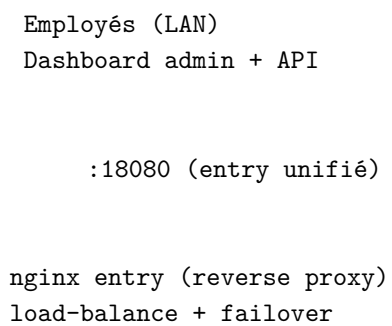
Timeouts : `proxy_read_timeout 300s` tolère les inférences longues sans couper la connexion.

---

4. Topologie

4.1 Cabinet PME — LAN mono-site 3-pool RAID

Configuration minimale pour un cabinet avocat / PME :



```
:18081   :18082   :18083

 pool-a   pool-b  pool-c    Docker containers
                    même LAN

                gossip Ed25519 (bond trust 3)


     PostgreSQL 16
      + pgvector      (3 DBs iamine_a/b/c ou 1 DB partagée)
```

Propriétés : - 3 pools = RAID software : perte d'1 pool ne casse pas le service - Même LAN → latence gossip < 1 ms, cohérence rapide - nginx entry masque la topologie physique, un seul DNS client - PostgreSQL backup externe quotidien (vers Z2 / NAS)

Ref : `feedback_enterprise_lan_raid_minimum` + `CELLULE_PRO_ARCHITECTURE.md` + `appliance/cabinet/README.md`.

4.2 Corporate — WAN multi-sites

Pour une grande entreprise multi-sites (usine, antennes régionales) :

- Chaque site = 1 ou plusieurs pools (LAN local)
- Sites reliés par VPN privé ou tunnel sécurisé (cf. `feedback_enterprise_multisite_wan`)
- Fédération Ed25519 cross-sites avec trust 3
- Workers `home_pool_id` : mobile (réassignable cross-sites) vs on-site (locked au site) cf. `project_chantier_13_`
- Latence gossip tolérée : jusqu'à ~200 ms (LAN = WAN sécurisé architecturalement — `feedback_lan_wan_architecture_equivalent`)

Gap chantier #10 + RAG : la propagation cross-pool des chunks / memories / documents n'est pas câblée v1 (seuls `moe_local_config` + `project_creation` propagent). Bloquant commercial Corporate — follow-up prioritaire. Cf. `project_todo_propagation_extend_project_tables.md`.

Référence : `feedback_enterprise_multisite_wan`.

4.3 Fédération Ed25519

Chaque pool a une paire Ed25519 unique. Le **bond** (reconnaissance mutuelle) est la seule façon pour 2 pools de gossiper.

Génération clé (à l'install) :

```
python -m cellule_pro.federation generate-keypair \
    --out /var/lib/cellule/federation_keys/
# Produit federation_self.privkey + federation_self.pubkey
```

Bond initial (une fois, via dashboard ou CLI admin) : - Chaque pool publie sa pubkey - Admin ajoute manuellement la pubkey de chaque pair dans `federation_peers` - Trust par défaut = 1 à l'ajout, monte à 3 après 24h stable

Signature : chaque mutation propagée inclut une signature Ed25519. Un pool refuse une mutation non-signée ou signée par un pair non bondé trust 3. Cf. `invariant chantier #10`.

Rotation / révocation : pas d'API v1 dédiée. En cas de compromission clé → backup config, re-générer, re-bond manuel (bref downtime gossip acceptable).

4.4 Ports et firewall

Port	Service	Exposé à
`18080/tcp`	nginx entry unifié (API + admin)	clients LAN (ou TLS public via reverse-proxy)
`18081-83/tcp`	pools Docker internes	**interne seul** (jamais public)
`5432/tcp`	PostgreSQL	**interne seul** (jamais public)
`8080/tcp`	pool.py dans container	**interne seul**
WebSocket `/ws`	live events dashboard	via `:18080` (upgrade par nginx)

Firewall type : - ENTRÉE : autoriser `:18080` depuis LAN client (ou tunnel Cloudflare) - INTERNE : autoriser les ports 18081-83 + 5432 uniquement entre les containers Docker (réseau bridge) - SORTIE : pool doit joindre `https://cellule.ai/pypi` (updates) si activé — sinon aucune sortie requise (air-gap possible)

4.5 Préconisations critiques multi-host (découvertes en live 2026-04-22)

Règles validées sur cluster Z2 + Gladiator (2 machines physiques, 4 pools, LAN) avant déploiement client. Issues de `project_preconisations_note_technique_mise_en_place`.

P1. `POOL_URL` doit être LAN/WAN-addressable **Règle non-négociable** : chaque pool advertise une URL que **tous les autres pools** du cluster peuvent joindre. Hostname Docker interne (`http://pool-a:8080`) ne fonctionne que intra-host ; multi-sites = cassé.

Mauvais (bug) : `POOL_URL=http://pool-a:8080` **Bon** : `POOL_URL=http://pool-a.entreprise.local:18091` (FQDN DNS interne) ou à défaut `POOL_URL=http://10.0.1.5:18091` (IP LAN).

Impact oubli : les pairs enregistrent URL non-routable dans leur `federation_peers`, handshake peut passer mais gossip ultérieur échoue silencieusement. Chunks RAG ne se propagent plus cross-host.

P2. Clés Ed25519 sur volume persistant `IAMINE_FED_KEY_DIR` doit pointer vers un volume Docker **nommé** (pas un bind mount sur `/tmp`). La perte de `self_ed25519.key` = nouvelle `atom_id` au redémarrage = rupture de TOUS les bonds existants et re-bond manuel obligatoire.

Bon : `IAMINE_FED_KEY_DIR=/var/lib/iamine/fed-keys` mounted sur volume Docker `fedkeys_X`.

P3. Bonds réciproques + promotion trust=3 requis pour gossip

- `POST /v1/federation/admin/register` avec `reciprocate=true` (défaut auto-bond) crée les 2 rows dans `federation_peers` en 1 call.
- Au bond initial trust=1. Le gossip applicatif filtre à `trust 3` (`MIN_TRUST_LEVEL` en core.py).

- **Promotion manuelle explicite** par admin via `POST /v1/federation/peers/{atom_id}/promote?token=$A` body `{"target_level":3}` — opt-in sécurité.

Auto-promotion après N jours stables = toggle admin à venir (cf. `feedback_admin_toggle_everything`).

P4. `IAMINE_BOND_PEERS` format CSV `host:port` ou URLs complètes. Exemple multi-sites :

```
IAMINE_BOND_PEERS=pool-paris.entreprise.local:18091,pool-lyon.entreprise.local:18091
```

Consommé par l'entrypoint au boot : attend que le pool soit healthy, puis POST `/v1/federation/admin/register` pour chaque peer listé.

P5. Env vars admin-toggle

Variable	Valeurs	Effet
`IAMINE_FED`	`active` / `observe` / `off`	Master switch fédération (défaut via entrypoint = `active`)
`IAMINE_FED_GOSSIP_ENABLED`	`true` / `false`	Gossip applicatif project_* (défaut `true`)
`CELLULE_DEBUG_INGEST`	`1` / unset	Endpoint debug `ingest_inline` (défaut unset = 404)
`ADMIN_PASSWORD`	string fort	Rotation impérative avant prod, stocker en vault

P6. DBs séparées par pool (jamais partagées) Doctrine : chaque pool a sa propre DB. Le gossip chantier #1 réplique les rows Ed25519-signées cross-pool. Partager la DB = SPOF, contradit la promesse RAID N-pool. WAN multi-sites physiquement impossible en DB partagée (latence).

4.6 HA failover & auto-orchestration cross-pool

Doctrine `feedback_pool_virtuel_orchestre_jamais_manuel` : le pool virtuel décide seul de migrer / rebasculer les workers et proxies en fonction de la **charge** ET de la **panne**. Aucun déplacement manuel par l'admin. Le rôle DSI = définir la politique via UI ; jamais bouger une pièce à la main.

4.6.1 Failover réactif (panne d'un pool) Hystérésis 3 tiers anti-flap appliquée par chaque proxy à son pool primary :

Tier	Seuil	Comportement
PRIMARY_OK	< 90s sans reconnect	Retry primary toutes les 30s, log silencieux
FAILING	90s à 5min	Reste sur primary (anti-flap), event audit-log

Tier	Seuil	Comportement
DOWN	5min en mode auto	Bascule sur peer bondé Ed25519 via `peer_pool_urls` reçus au handshake

Au retour du primary, **sticky return en 60s** (cycle `_sticky_primary_check`) ramène le proxy chez lui. Aucun job perdu : les peers bondés acceptent les proxies migrés via la mécanique de trust cross-pool.

Mode pilotable : `CELLULE_FAILOVER_MODE=auto|manual` (défaut : `auto`). Mode `manual` désactive la bascule automatique tier 3 — admin doit la déclencher explicitement (`force_migration` via UI). Failsafe 24h en mode manuel : si l'admin n'a pas répondu après 24h, bascule auto pour sauver la disponibilité.

4.6.2 Migration orchestrée (rebalancing actif) Plus large que le simple failover : le balancer émet des ordres `worker_migration_orders` selon la charge observée et l'allocation employees, sans attendre une panne. Mécanisme :

1. `worker_balancing.balancer.compute_balancing_plan(pool)` calcule la répartition cible (employees × tier × pool).
2. `emit_migration_orders` insère un row signé Ed25519 par mouvement souhaité.
3. `migration_executor` (loop 5s) progresse chaque ordre dans la state machine `issued → draining → completed` (ou `rolled_back` si timeout 90s).
4. À l'étape `draining`, l'executor envoie au proxy la commande WS `cmd: migrate_pool` avec `target_pool_url`.
5. Le proxy ACK draining → drain ses jobs in-flight (graceful 60s) → ACK drained → bascule en changeant son `pool_url` (migration permanente) → reconnecte au target.
6. Côté pool source, l'executor détecte le worker disconnecté → marque l'ordre `completed` (et update `worker_proxy_registry.current_pool_id`).

Toggle global : `balancer_config.executor_loop_enabled` (défaut `true`). Pilotable via `PUT /v1/admin/dashboard/b`
OFF = aucune migration orchestrée n'est exécutée (les ordres restent `issued`, l'admin peut les `reject` manuellement).

4.6.3 Master / satellite — failover catalogue modèles Pour les **mutations de catalogue** (upload `.gguf`, assignment hardware_class, delete), un seul pool est *master* à un instant t. Les autres = *satellites* (read-only catalogue).

Si le master tombe, les satellites **ne s'auto-promeuvent jamais** : - Les satellites continuent à servir le catalogue existant en lecture - L'inférence sur les workers locaux + cross-pool continue normalement - Les uploads et changements d'assignment renvoient HTTP 409 avec hint `"écrivez sur le master"` - L'admin DSI promeut explicitement un satellite via UI dashboard (`POST /v1/admin/dashboard/models/promote-master`) — bouton avec confirmation, signature Ed25519, audit trail, email aux admins.

Au retour du master historique : il bootstrap **satellite par défaut** (anti-reclamation). L'admin re-promeut explicitement s'il veut le master à nouveau "pinned" sur le site historique.

Doctrine : zéro magie, admin pilote, audit trail. Évite les split-brain catalogue (deux masters simultanés = deux catalogues qui divergent = réconciliation manuelle obligatoire). Cf. invariants `project_section_5_4_phases_g_h_i_complete`.

4.6.4 Test du failover en pré-prod Procédure de validation avant rollout client multi-sites :

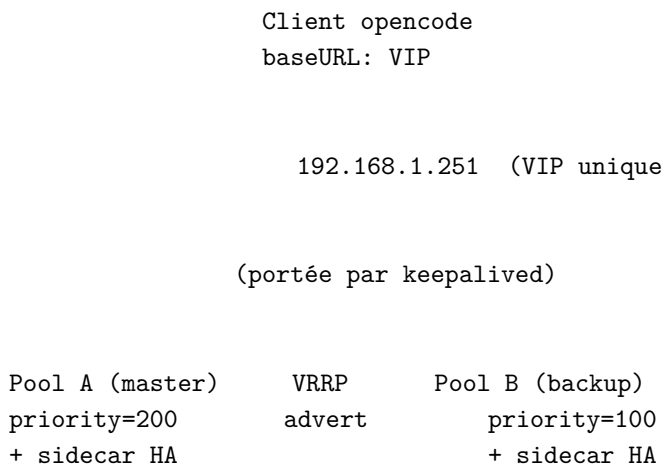
1. Préparer 2 pools fédérés bondés (trust 2 réciproque).
2. Stop physique du pool primary d'un proxy : `docker stop pool-X`
3. Observer les logs du proxy : `tail -f ~/.cellule/proxy-*.log`
4. À T+5min : transition `FAILING → DOWN (outage 300s, mode=auto)` puis `Connexion au pool ws://...peer.../ws (fallback)`.
5. Vérifier sur le pool peer : `curl http://peer:18095/v1/status` doit afficher les workers migrés.
6. Restart le primary : à T+60s, sticky return automatique.

4.7 Pool virtuel HA — VIP keepalived (chantier rc62)

> **Problème résolu** : avant rc62, un client API (opencode, Cursor, OpenWebUI) devait pointer son `baseURL` sur l'IP d'un pool concret (ex : `http://192.168.1.199:18095`). Si ce pool tombait, le client était bloqué — il fallait reconfigurer manuellement chaque poste pour pointer un autre pool. Le forward inter-pool M7a couvre les cas "pool sain mais saturé / modèle absent", mais pas "pool down".

Solution : une **adresse IP virtuelle unique** (VIP) partagée entre tous les pools du LAN, portée par `keepalived` (VRRP). Un seul pool tient la VIP à la fois (le "master VIP", choisi par priorité). Si le master tombe, un pool backup prend la VIP en moins de 3 secondes — le client ne voit aucune coupure.

4.7.1 Architecture



Chaque pool embarque un **sidecar** `keepalived-pool` (même image Docker, entrypoint dédié). Le sidecar tourne en `network_mode: host` + `cap_add: NET_ADMIN` pour pouvoir binder la VIP sur l'interface physique. Au boot, il fetch dynamiquement sa configuration via l'endpoint local `/v1/cluster/vip-self` du pool — pas d'env var manuelle, c'est le wizard qui pilote.

4.7.2 Activation au wizard first-boot Étape **Réseau** du wizard (`/setup/ui`), section **Pool virtuel HA** :

1. Cocher *"Activer le pool virtuel HA"*
2. Choisir le mode :
   - **Premier pool du cluster** : saisir `vip_address` (IP libre du subnet, ex `192.168.1.251`) + masque CIDR du LAN (24, 22, 16…). Le mot de passe VRRP est auto-généré.
   - **Rejoindre un cluster existant** : saisir l'URL d'un pool déjà installé (ex `http://192.168.1.199:1809` Le wizard fetch la config VIP du master via `/v1/cluster/vip-info` (admin auth) — pas de re-saisie manuelle.

Le sidecar lance keepalived dès que la config est en DB.

4.7.3 Activation côté compose Le sidecar est inscrit dans le profile Docker Compose `vip`, **désactivé par défaut**. Pour les déploiements simples mono-pool sans HA, laissez le profile inactif. Pour HA, activez via :

```
COMPOSE_PROFILES=vip docker compose up -d
# ou ajoutez dans .env :  COMPOSE_PROFILES=vip
```

4.7.4 Vérification opérationnelle Sur le pool master :

```
ip -br -4 addr show | grep <VIP>
# attendu : votre interface UP <IP_host>/24 <VIP>/24
```

Sur les pools backup, la VIP n'apparaît PAS dans `ip a` — c'est normal.

Test bascule (sur le pool master) :

```
docker stop pool-<host>-h
# Attendez ~3-8s
ip -br -4 addr show | grep <VIP>   # plus rien
```

Sur un pool backup :

```
ip -br -4 addr show | grep <VIP>   # la VIP est apparue
```

Recovery automatique : `docker start pool-<host>-h` → après ~30s, la VIP revient sur le master initial (priority 200 > 100, preempt).

4.7.5 Pré-requis réseau

- LAN switché plat (VRRP utilise multicast `224.0.0.18`). Marche par défaut sur tous les switches L2 standard.
- VIP libre dans le subnet au moment du setup (pas pingable).
- Tous les pools du cluster doivent partager le **même** `vip_vrid` et le **même** `vip_auth_pass` (le wizard les propage automatiquement en mode "rejoindre cluster").
- VRID unique sur le LAN si plusieurs clusters VRRP cohabitent (rare).

4.7.6 Doctrine : ce que la VIP NE fait PAS

- **Workers et proxies n'utilisent PAS la VIP.** Ils continuent à pointer leur pool d'enrôlement et bénéficient de l'hystérésis cross-pool (Phase 1+2, cf §4.6) pour migrer en cas de panne.
- **Pas de load-balancing actif.** keepalived = failover seul, le master fait tout le trafic. Le forward M7a inter-pool reste le mécanisme d'équilibrage charge interne.
- **Pas de cross-WAN.** La VIP est LAN-only (multicast). Pour multi-sites, prévoir une VIP par site + DNS health-check ou anycast côté upstream (hors scope rc62).

4.8 Worker auto-routing par latence (chantier rc81)

Doctrine : le poste de travail employé est déclaré une fois sur la VIP du cluster. Le pool virtuel décide ensuite vers quel pool physique diriger ce worker, selon sa proximité réseau réelle. Aucun geste DSI par poste, aucun choix de pool côté employé.

Cas d'usage : cluster multi-sites (cabinet Paris + Lyon, chaîne hôtelière, industrie multi-usines). Chaque site a son pool local, fédéré via Ed25519. Tous les postes employés (Win / macOS / Linux) pointent sur la VIP centrale via un seul one-liner d'install. Au boot, l'employé Lyon va naturellement être enrôlé sur le pool Lyon (LAN local, latence minimale) sans intervention.

Mécanisme : - Au handshake WebSocket, le master extrait le sous-réseau `/24` du worker (depuis `ws.client.host`) - Il compare avec son propre `federation_self.url` et chaque peer bonded (`trust_level >= 2`) - Si un peer a le même `/24` que le worker (et pas le master courant), un ordre `cmd: migrate_pool` signé Ed25519 est inséré dans `worker_migration_orders` et notifié au worker - Le worker drain ses jobs en cours (max 60s), persiste son `worker.json` sur le nouveau pool, ferme la WebSocket, reconnecte sur le pool cible. Permanent (pas un fallback temporaire).

Flag admin : `WORKERS_AUTO_LATENCY_ROUTING` (catégorie `cluster_routing`, default `false`, opt-in).

Quand activer	Quand laisser OFF
Cluster multi-sites stable, fédération Ed25519 bonded validée	Site unique ou cluster en phase de mise en route
Postes employés répartis sur plusieurs sous-réseaux `/24`	Postes tous dans le même `/24` que le pool master
Vous avez observé que la VIP enrôle des workers loin de leur LAN local	Vous voulez que tous les workers passent par le master pour audit centralisé

Anti-pattern : ne pas tenter de re-router un worker oscillant entre 2 peers proches. Hystérésis prévue côté code : 1 ordre par worker (idempotence par PK `migration_order_id`), nouveau ordre seulement si le worker se reconnecte ailleurs et que le calcul redonne un meilleur peer.

Limitations connues rc81 : - Heuristique `/24` uniquement (pas de RTT ping actif). Suffisant pour LAN/WAN typique entreprise. Une extension future pourra ajouter du ping pour les cas ambigus (sous-réseaux NATés, VPN site-à-site). - Le master reste toujours le master. On déplace seulement les workers, jamais le rôle du pool. Cohérent avec doctrine "le pool virtuel orchestre, jamais de migration manuelle".

Référence visuelle : voir le diagramme `docs/architecture/layer-worker-latency-routing.html`.

4.9 Choix des modèles GGUF — recommandations DSI

Cellule-PRO sert n'importe quel `.gguf` compatible llama.cpp uploadé via le dashboard admin. Mais TOUS les modèles ne se comportent pas pareil côté UX assistant chat. Trois familles à connaître avant de choisir.

4.9.1 Modèles Instruct non-thinking (recommandés POC + prod) Les variants entraînés explicitement pour répondre **directement** sans chain-of-thought visible. Aucun raisonnement inline, aucun pré-amble analytique. Comportement out-of-the-box conforme à un assistant chat.

Liste blanche testée : - `Qwen3-4B-Instruct-2507-Q4_K_M.gguf` (~2.4 GB) — variant officiel non-thinking, 262K contexte natif. Modèle par défaut recommandé pour hardware_class `gpu-small` + `cpu-mid`. - `Qwen3-30B-A3B-Instruct-2507-Q4_K_M.gguf` (~18 GB) — variant non-thinking large pour `gpu-large`. - `Llama-3.x-Instruct` (toutes tailles) — Meta Llama Instruct. - `Mistral-Instruct` (7B, Small, Large) — Mistral AI Instruct. - `Phi-3-mini/medium-Instruct` — Microsoft, petits modèles efficaces.

4.9.2 Modèles thinking (caveat) Variants entraînés pour montrer leur raisonnement, soit avec balises `<think>...</think>` soit en numérotation explicite inline. Cellule-PRO strippe automatiquement les balises `<think>`, mais le **raisonnement inline numéroté** ("1. Analyze the Request :…, 2. Draft Response :…") peut traverser le filtre selon le SYSTEM_PROMPT et la taille du modèle.

Modèles concernés : - `Qwen3-*` (sans `Instruct-2507`) — Qwen3 v1, thinking activé par défaut. - `Qwen3.5-4B/9B-*` (sans `Instruct`) — Qwen3.5, idem. **Particulier** : Qwen3.5-4B en Q4 a tendance à montrer un raisonnement inline numéroté même avec un SYSTEM_PROMPT qui demande explicitement "réponds directement" — c'est dans son entraînement. - `QwQ-*` — modèle reasoning Qwen, thinking par design. - `DeepSeek-R1-*` (et distillations) — reasoning model.

Recommandation : pour un déploiement chat employé classique, **éviter** ces variants comme modèle principal. Les utiliser uniquement pour des cas d'usage où le raisonnement explicite est souhaité (recherche, analyse approfondie, debugging algorithmique).

Si malgré tout vous devez servir un thinking model : - Le strip côté pool retire automatiquement les balises `<think>`. - Filtres regex défensifs sur les patterns numérotés "1. **Analyze :**". - Performance variable selon le modèle.

4.9.3 Modèles base ou fine-tunes propriétaires Variants non-instruct (modèle "raw") ou fine-tunes maison du DSI. Comportement non garanti — peut produire des réponses sans formatage chat, des completions au lieu de réponses, du leak de tokens spéciaux.

Recommandation : à n'utiliser qu'avec un wrapper applicatif (API gateway custom) qui formate proprement les inputs/outputs. **Pas recommandé** comme modèle servi directement aux employés.

4.9.4 Comment choisir pour mon parc

Hardware client typique	hardware_class	Modèle recommandé
Postes Win/Linux 8-16 GB RAM CPU only	`cpu-mid`	Qwen3-4B-Instruct-2507 Q4_K_M
Workstations RTX 2060/3060 6 GB VRAM	`gpu-small`	idem (Q4_K_M tient en 6 GB ctx 4K)

Hardware client typique	hardware_class	Modèle recommandé
Workstations RTX 4090 / A6000 24+ GB VRAM	`gpu-large`	Qwen3-30B-A3B-Instruct-2507 Q4_K_M
Mac M1/M2/M3 16+ GB RAM	`apple-m`	Qwen3-4B-Instruct-2507 Q4_K_M (Metal)
Spécialisé code (DevOps, dev assistant)	aucun (proxy dédié)	Qwen3-Coder-30B-A3B-Instruct

Le DSI peut uploader plusieurs modèles et les assigner par `hardware_class` via la page **Modèles** du dashboard. Le pool route automatiquement chaque worker vers son modèle assigné via §5.4 hardware_class catalogue.

Avant d'uploader un modèle non listé : faire un test chat de contrôle "bonjour" + "écris un email à mon manager" + "résume ce texte" avec un cobaye en mode worker isolé. Si la réponse contient un préambule analytique numéroté, c'est un thinking model — basculer sur un variant Instruct non-thinking.

---

5. Avantages

Pitch opérationnel pour DSI et décideur technique. Angles :

5.1 Frugalité compute

Cellule-PRO tourne sur le hardware **déjà présent** en entreprise : postes employés dormants (CPU), serveurs déjà amortis. Pas d'achat GPU dédié datacenter, pas de facture cloud mensuelle.

Estimation ordre de grandeur (à chiffrer précisément via whitepaper) : empreinte kWh/token ~14× inférieure à un déploiement GPU datacenter dédié (hardware en service, seuls cycles dormants utilisés). Sources à produire : ADEME, EPRI, rapports Anthropic/Google.

5.2 Complémentarité cloud (pas substitution)

Cellule-PRO ne remplace pas ChatGPT Enterprise sur le raisonnement frontier-class. Elle couvre **~70% des requêtes LLM courantes** : synthèse de mails, classification (urgence, routage interne), traduction interne, extraction d'information, Q/A sur documents internes (chantier RAG).

Les ~30% complexes (raisonnement juridique frontier, analyse stratégique) restent envoyés au cloud via segmentation métier. Les deux cohabitent.

5.3 Souveraineté data

Argument le plus fort segment Cabinet avocat / Santé / Défense : - Aucune donnée ne quitte le LAN client - Compatible attorney-client privilege (envoyer données client à un tiers non-autorisé = faute déontologique en France) - RGPD data segregation par projet (isolation cryptographique cf. `project_chantier_10_invariants`) - Audit trail immutable append-only (`project_audit`, ISO 27001) - Droit à l'oubli : CASCADE DELETE prouvé e2e (test Phase 4 chantier RAG)

Le RAG chantier livré cette session renforce l'argument : la connaissance privée (archives, contrats, mails) devient exploitable sémantiquement **sans quitter le site**.

5.4 Écologie

Valorisation hardware dormant = **zéro achat neuf, zéro coût de fabrication** attribuable à Cellule-PRO. Le parc existant continue à servir, juste plus efficacement.

Chiffrage précis à produire (whitepaper dédié) avec sources ADEME, EPRI, rapports environnementaux cloud providers. Angle : CO2 par token utile vs H100 neuf dédié.

5.5 Simplicité opérationnelle

Doctrine `feedback_compliqué_de_faire_simple` : - Installation Cabinet 3-pool : `git clone` + `docker compose up -d` (cf. `appliance/cabinet/install.sh`) - Defaults intelligents (ports / paths / thème) — modifier uniquement si besoin - Admin en français actionnable, pas de jargon - Hub admin web unifié (pas de CLI obligatoire) — chantier #11 - Onboarding wizard pour les premiers projets - Backup et restore documentés avec commandes exactes (cf. §6) - **HA cross-site sans intervention admin** : un pool tombe, les proxies migrent automatiquement vers les peers bondés (5 min hystérésis anti-flap). Le pool virtuel rééquilibre aussi en fonction de la charge observée, sans attendre une panne. Cf. §4.6 pour la mécanique complète (Phase 1 réactif + Phase 2 orchestré).

---

6. Annexes

6.1 Logs à surveiller

Docker containers :

```
docker logs -f pool-a                    # log live d'un pool
docker logs --since 1h pool-a | grep -i error
docker logs --since 1h entry 2>&1 | tail -50   # nginx access + errors
```

Loggers Python structurés (tags dans le format d'émission) : - `cellule_pro.document_ingest` — pipeline RAG (`skipped` / `failed` / `done`) - `cellule_pro.document_search` — KNN queries - `cellule_pro.project_scope` — schema migrations - `cellule_pro.memory_fact_replication` — gossip chantier #1 - `iamine.routing_embeddings` — MiniLM load status - `uvicorn.access` — HTTP requests

PostgreSQL slow queries (à activer en prod) :

```
log_min_duration_statement = 500   # >500ms
```

Audit applicatif : lire via `GET /v1/projects/{id}/audit?limit=500` (compliance ISO 27001 append-only).

6.2 Troubleshooting fréquent

Symptôme	Première vérif	Fix courant
Pool down (health 503)	`docker ps`	`docker restart pool-X` + vérif logs
Worker muet	`systemctl status iamine-worker`	re-démarrer service, vérif connectivité pool
Latence élevée	CPU + embedder chargé	trop de requêtes parallèles → scale worker idle
Erreur fédération `bad signature`	pubkey pair obsolète	re-bond pair via dashboard
Upload doc pas indexé	`indexed_status` via API	cf. §2.4 “Document uploadé jamais indexé”
Search RAG retourne vide	chunks générés ?	vérif `chunks_count` sur doc, re-ingest si 0
DB “vector extension missing”	image postgres vanilla	utiliser `pgvector/pgvector:pg16`
CASCADE DELETE a raté	contrainte FK désactivée	re-appliquer `ensure_document_rag_schema`

6.3 Runbook incident

Procédures de reprise détaillées : `reference_pra_z2` (template PRA en 8 sections, RTO 4-8h, à adapter au cluster client).

Scénarios couverts : - Perte d’un pool sur 3 → service continue (RAID), re-join après fix - Perte du volume pgvector → restore depuis backup externe (Z2 / NAS), re-migrate schema - Fuite de clé Ed25519 → re-génère keypair + re-bond manuel + rotate tokens dashboard - Base corrompue → `pg_basebackup` restore puis `ensure_*_schema` (migrations idempotentes = safe) - Perte complète hosting pool → documents uploadés en `indexed_status=failed`, ré-upload manuel depuis source client (cf. §2.4 “Pool mort mid-ingest”)

6.4 Références croisées

- `BOUNDARY.md` — contrat d’interface public PRO
- `CELLULE_PRO_ARCHITECTURE.md` — architecture détaillée
- `SECURITY_MODEL.md` — modèle de sécurité
- `API_REFERENCE.md` — référence API pour intégrateur

---

7. Licence Cellule-PRO

The Cellule-PRO license system (chantier #17) is an **offline-verifiable JWT EdDSA** scheme. No phone-home, no telemetry, no kill-switch. The public key of the CELLULE.ai signing authority is embedded in each Docker image release. License issuance happens on David’s air-gapped workstation via `tools/issue_license.py`.

7.1 Lifecycle overview

```
[admin uploads JWT] -> [crypto verify Ed25519]
                    -> [payload schema check]
                    -> [revocation list check]
                    -> [persist to cluster_license table]
                    -> [runtime: status tracked at each request]
```

Five runtime statuses :

Status	Meaning	Mutations
`active`	license valid, `exp` more than 30 days ahead	allowed
`warning_j30`	license valid, less than 30 days before `exp`	allowed + dashboard banner
`grace`	past `exp`, still within `grace_days` window	allowed + WARNING log
`expired`	past `exp + grace_days` — mutations blocked	**refused 402**
`revoked`	`revoked=true` in DB OR client_id in revocation list	**refused 402**
`none`	no license installed (v1 dev opt-in tolerant)	allowed

Reads (GET/HEAD/OPTIONS) are **never** blocked — invariant #2 “grace non-punitive”. A client whose license expired can still read and export their data ; they just cannot create new projects, bond new peers, or ingest new documents until they renew.

7.2 Wizard first-boot upload

At first boot, the install wizard (chantier #16) presents a license screen **before** the identity step. Admins can :

- Drop or paste a `.jwt` file → live client-side preview (tier, expiry, features, limits)
- Click **Activate license** → sends `POST /setup/license`, crypto + schema + revocation checks happen server-side
- OR click **Continue in evaluation mode** → starts Cellule-PRO without a license, with a permanent info banner in the admin dashboard

The wizard never gates pool boot — doctrine v1 opt-in tolerant. A valid license can be uploaded at any time via the admin dashboard.

7.3 Admin endpoints

Method	Path	Purpose
POST	`/v1/admin/license/upload`	Upload / renew license JWT
GET	`/v1/admin/license`	Current status (JWT token NEVER returned)
DELETE	`/v1/admin/license`	Remove license — back to dev opt-in mode
POST	`/v1/admin/license/enforce`	Toggle gate enforcement (see §7.6)

Auth : cookie `admin_token` matching `ADMIN_PASSWORD` env var, or `X-Cellule-User` header matching an email in `admin_users` table.

Renewal example (admin shell) :

```
# 1. Receive the new .jwt from David (encrypted channel)
# 2. Upload it
curl -X POST https://pool.internal/v1/admin/license/upload \
  -H "Content-Type: application/json" \
  -H "X-Cellule-User: admin@corp.local" \
  -d @<(jq -n --rawfile tok new_license.jwt '{token: $tok}')

# 3. Confirm
curl https://pool.internal/v1/admin/license \
  -H "X-Cellule-User: admin@corp.local"
# Expected: {"status": "active", "days_remaining": 365, ...}
```

7.4 Feature flags (route-level gating)

A license JWT declares `features` as a **set**, not a counter. Example payload for a Cabinet tier :

```
{
  "tier": "cabinet",
  "features": ["rag_documents", "balancer", "smart_routing"],
  "limits": {"max_pools": 3, "max_users": 50}
}
```

Inside plugins, routes that need a specific feature guard themselves :

```
from cellule_pro.plugins.license.decorators import require_feature

@app.post("/v1/projects/{pid}/documents")
async def ingest_doc(pid: str, ...):
    await require_feature(pool, "rag_documents")
    # ... rest of handler
```

- License absent (dev opt-in) → feature gate passes silently (v1 tolerant)
- Feature not in license set → HTTP 403 with `feature_not_licensed`
- License `expired` / `revoked` / `invalid` → HTTP 402 (all features denied)

7.5 Limit gates

Same pattern with `require_limit` :

```
from cellule_pro.plugins.license.decorators import require_limit

@app.post("/v1/admin/pools")
async def add_pool(...):
    current = await count_pools(pool)
    await require_limit(pool, "max_pools", current)
    # ... rest
```

- Limit not declared → unlimited (no gate)
- `current_count >= max` → HTTP 402 `limit_reached:max_pools=5>=5`
- Counters are **read at call time** by the caller (no license-side state)

7.6 Admin-toggle `enforce` — emergency override

The `cluster_license.enforce` column is a **BOOLEAN DEFAULT TRUE** (Phase E). When set to `false`, the license gate is fully bypassed — mutations pass regardless of expiration or revocation.

Use cases :

- **Contract dispute** — admin refuses to renew but needs temporary access to export data
- **Urgent migration** — moving to a new instance before the new license is signed
- **Maintenance window** — decoupling ops from license state

Toggle via endpoint :

```
# Disable gate (all mutations pass)
curl -X POST https://pool.internal/v1/admin/license/enforce \
  -H "Content-Type: application/json" \
  -H "X-Cellule-User: admin@corp.local" \
  -d '{"enabled": false}'
# -> {"ok": true, "enforce": false}

# Re-enable (return to normal gating)
curl -X POST https://pool.internal/v1/admin/license/enforce \
  -d '{"enabled": true}' ...
```

Or directly via DB (last-resort, no app access) :

```
UPDATE cluster_license SET enforce = FALSE WHERE id = 1;
```

Every toggle is logged at WARNING level with explicit text (`license.enforce toggled to false`). Doctrine : safety-first default is `true`. The toggle itself is the kill-switch replacement — admin keeps full control, CELLULE.ai has no remote authority over a running cluster.

7.7 Revocation — release-gated list

The file `src/cellule_pro/plugins/license/revoked_clients.json` ships with each Docker image and lists revoked client_ids with metadata :

```
{
  "_schema_version": 1,
  "_last_updated": "2026-04-22",
  "revoked": {
    "evil-corp-001": {
      "reason": "security_incident",
      "revoked_at": "2026-03-15"
    }
  }
}
```

Adding a client to this list requires :

1. Edit `revoked_clients.json` on David's workstation
2. Rebuild Docker image
3. Release the new image to clients (via internal registry or image upgrade on their instance)
4. Clients apply the update → revoked license refused on next boot and runtime `compute_license_status` returns `status=revoked`

Why release-gated (not online CRL) : invariant #1 offline strict — no license-side fetch of external state. A client that never pulls a new image never sees an updated revocation list, but that's by design : CELLULE.ai has no backdoor into running clusters. Trust is anchored in the release supply chain, not in a live revocation service.

The validator enforces ordering : **signature verify first**, then **revocation check**. This prevents information leaks — if the JWT is forged, the response is `jwt_invalid`, never `client_revoked`.

7.8 Generating a new license — David's workflow

On David's air-gapped workstation :

```
# 1. Ensure the CELLULE.ai private key is available (GPG-decrypted
#    from the ceremony backup)
gpg --decrypt cellule_license_privkey.ed25519.gpg > /tmp/priv.key

# 2. Issue a new license
python tools/issue_license.py issue \
  --privkey /tmp/priv.key \
  --client-id cluster-example-001 \
  --tier cabinet \
  --duration-days 365 \
  --output cluster-example-001.jwt

# 3. Verify with the shipped validator
python tools/issue_license.py verify \
  --token cluster-example-001.jwt \
  --pubkey <hex_pubkey>
```

```
# 4. Shred the decrypted private key
shred -uvz /tmp/priv.key

# 5. Send the .jwt to the client via secure channel
#    (signed email + S/MIME, or encrypted physical media)
```

Tier defaults (duration, features, limits) live in `tools/issue_license.py::TIER_DEFAULTS` — adjust per commercial agreement before calling `issue`.

7.9 Ceremony — private key management

The ceremonial generation of the CELLULE.ai signing keypair is the **single most sensitive operation of the project**. Loss of the private key = loss of the ability to issue new licenses (existing ones remain valid until `exp`). Compromise of the private key = any attacker can forge licenses.

Procedure (executed offline once, documented in `tools/generate_cellule_license_keypair.py` docstring) :

1. **Air-gap workstation** — no network, no USB except the one holding the output
2. `python tools/generate_cellule_license_keypair.py` — generates 32-byte Ed25519 keypair, writes public key hex to stdout + private key to `cellule_license_privkey.ed25519`
3. **GPG-encrypt** the private key with a strong passphrase : `gpg --symmetric --cipher-algo AES256 cellule_license_privkey.ed25519`
4. **Three backups, geographically distinct** :
   - David's personal safe (primary)
   - Sealed envelope at notary (legal custodian)
   - Trusted family member in a different city (geo-redundancy)
5. **Wipe the plaintext private key** : `shred -uvz cellule_license_privkey.ed25519`
6. **Replace** `CELLULE_LICENSE_PUBKEY_PROD` constant in `src/cellule_pro/plugins/license/validator.py` with the generated hex pubkey
7. Commit + release new Docker image

Until step 6 is done, the `CELLULE_LICENSE_PUBKEY_PROD` is the 32-byte zero placeholder and the validator rejects **all** licenses (fail-safe). This is why the v1 doctrine is *dev opt-in tolerant* — a pool with no license installed still boots fully functional, so an early release before the ceremony doesn't lock anyone out.

Recovery : if 2 of 3 backups are lost, the remaining copy is used to sign a new release. No Shamir split in v1 — a single custodian can always issue licenses. Future hardening : N-of-M Shamir split with the M custodians (see `project_todo_license_shamir_split` — not yet scheduled).

7.10 Boot check — what you'll see in logs

At each pool startup, the license plugin logs :

```
license: schema ensured=True
license: revoked clients list loaded (N entries)
license boot: active, client=cluster-example-001 tier=cabinet expires_in=342d
```

```
license.routes: 5 endpoints installed
license: routes installed=True
install_license_gate: installed - mutations gated (reads always OK, ...)
license: gate middleware installed=True
```

If no license is installed :

```
license boot: no license installed - dev opt-in mode active
  (v1 tolérant, pas de coupure). Upload via admin dashboard.
```

Warning progression for upcoming expiration :

- 30 days before `exp` → WARNING log + dashboard banner
- Past `exp`, within grace → ERROR log every request that mutates
- Past grace → CRITICAL log + mutations refused

7.11 Troubleshooting

Symptom	Cause	Fix
`license pubkey not initialized` at upload	Ceremony not yet run	Complete the ceremony (see §7.9) and rebuild image
Every license rejected with `signature verification failed`	Mismatch between `CELLULE_LICENSE_PUBKEY_PROD` and the private key used by `issue_license.py`	Ensure they come from the same keypair
Mutations always refused despite valid license	Check `enforce` column in DB	`UPDATE cluster_license SET enforce=true WHERE id=1`
Client shows `revoked` but revocation was reverted	Image still carries old `revoked_clients.json`	Pull latest image and restart pool
Wizard blocks at license screen with “jwt_invalid”	Pubkey placeholder or corrupt JWT	Verify with `python tools/issue_license.py verify --token ... --pubkey ...`

7.12 Commercial framing — “Ceremony”

The expression **ceremony** (borrowed from Certificate Authority and cold-wallet terminology) is used in commercial pitch material. It signals to DSI / CISO / Legal that the key management follows the same operational rigor as a CA root or a crypto-custody cold storage setup. This differentiates Cellule-PRO from typical SaaS license servers and reinforces the sovereignty message (see `project_sales_hook_ceremonie` in memory notes). - `CONTRIBUTING.md` — règles de contribution interne