CELLULE-PRO DOCUMENTATION

Administrator Guide

Cellule-PRO DSI operations manual

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

Contents

Dashboard overview page

Figure 1: Dashboard overview page

0.1 Cellule-PRO — Administrator Guide

Audience: IT director, sysadmin, integrator in charge of deploying Cellule-PRO on a customer site (SMB firm or multi-site Corporate).

Reader prerequisites: basic Linux/Docker, basic IP networking. No Python or LLM knowledge required.

Status: skeleton created session 2026-04-21 (RAG Phase 6 chantier). Sections to be completed phase by phase.

Table of contents

---

0. Dashboard overview

The administrator dashboard is the single control surface for the cluster. All settings (headcount, inter-site federation, runtime flags, MoE, branding) are accessible from it. No command line is needed for daily operations.

0.1 Overview page

The home screen. A single view to grasp the cluster's state: current capacity (employee endpoints contributing compute, proxy servers, connected sites), configuration (declared headcount, MoE profile, auto-detected network topology), and shortcuts to the pages used most often.

0.2 Federation — inter-site connection

Cellule-PRO supports multi-site deployments (HQ + branches, several buildings on a campus). Each site hosts a pool, and pools authenticate each other via Ed25519 signatures. This page lists bonded sites and allows adding a new one through a single-use attachment token.

Federation page

Figure 2: Federation page

Worker balancing page

Figure 3: Worker balancing page

Topology page

Figure 4: Topology page

0.3 Worker balancing — declared headcount

Lets you declare the number of employees per site/branch. The virtual pool uses this information to calibrate its internal limits (queue size, per-employee queue) and to display the gap between declared headcount and workers actually contributing compute.

0.4 Topology

Cartographic view of the inter-site latency, measured continuously. The network type is auto-detected (local LAN <5 ms, multi-building metro 5-30 ms, inter-city WAN >30 ms) and drives certain cluster behaviours (RAG fanout, smart routing).

0.5 Advanced settings

Every tunable option lives here, organised by category. Each setting has a description aimed at the IT business owner (no LLM jargon imposed) and a recommendation per profile. Master changes are automatically propagated to cluster satellites via Ed25519-signed gossip.

0.6 Documentation

The admin/API/security guides are reachable straight from the dashboard, exportable as a branded PDF in one click (this very page is produced by that feature).

---

1. Docker

1.1 Image

- Registry: `celluleai/pool-private:latest` (private Docker Hub) or `registry.cellule.ai/pool-private:la` (self-hosted registry, see `project_todo_registry_distribution`)
- Release tag: `celluleai/pool-private:v<version>`
- Base: Debian slim + Python 3.12 + PostgreSQL client + pgvector runtime

Advanced settings page

Figure 5: Advanced settings page

Documentation page

Figure 6: Documentation page

1.2 Minimal docker-compose

Reference: `src/cellule_pro/appliance/quickstart/docker-compose.yml` (single pool, dev/POC) and `src/cellule_pro/appliance/cabinet/` (3-pool LAN RAID).

For a 3-pool LAN cabinet deployment:

```yaml
services:
  pg:
    image: pgvector/pgvector:pg16   # pgvector extension required
    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
      # ...other env vars in §1.3
    ports:
      - "18081:8080"

  pool-b: { ... }   # clone of pool-a with 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"   # unified cluster entry point

volumes:
  iamine-pg:
```

Sample `nginx.conf`: see `src/cellule_pro/appliance/entry/nginx.conf`.

1.3 Environment variables

Critical:

Variable	Sample value	Description
`CELLULE_ENTERPRISE`	`1`	Enables PRO plugins (otherwise = community mode)

Variable	Sample value	Description
POOL_NAME	cabinet-site1-pool-a	Unique pool identifier in the federation
DATABASE_URL	postgresql://user:pwd@pg:5432/iamine	PostgreSQL + pgvector
IAMINE_EMBEDDER_PATH	~/.cache/iamine-embedder	Path to pre-downloaded MiniLM-L6-v2 (384d)

Ed25519 federation:

Variable	Description
FEDERATION_SELF_PRIVKEY_PATH	Local Ed25519 private key used to sign mutations
FEDERATION_SELF_ATOM_ID	Pool ID in the federation_peers table

RAG project_documents: no env var required — the constants are hard-coded in document_rag.py (CHUNK_SIZE_TOKENS=512, MAX_DOCUMENT_SIZE_BYTES=50MB). Changing them = recompile, not config.

To complete: LICENSE_KEY, FORWARDING_ENABLED, perf tuning.

1.4 Persistent volumes

Volume	Content	Critical
iamine-pg	PostgreSQL data (projects, memories, chunks, audit)	**YES** — daily backup mandatory
iamine-embedder-cache	Downloaded MiniLM model (~90 MB)	No (re-downloadable)
iamine-uploads	Uploaded document binaries (storage_url)	Medium (ephemeral-acceptable + re-uploadable)
iamine-logs	Application logs	No (systemd rotation)

Critical backup: iamine-pg to external storage (Z2 / NAS / S3). The RAG live-DB tests assume a faithful dump/restore — verify that pgvector is installed on the restore target.

1.5 Quickstarts

- **SMB cabinet** (3-pool LAN RAID, single site) — see src/cellule_pro/appliance/quickstart/
- **Corporate** (N pools, multi-site WAN) — see src/cellule_pro/appliance/cabinet/ (legacy name, still generic)

1.6 First boot — install wizard (chantier #16)

On the **first start** of a Cellule-PRO pool (no row in `cluster_setup_state`), every non-whitelisted route is **blocked** by the `wizard_gate` middleware:

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

The admin dashboard is **not reachable** until the wizard is completed. Only `/v1/status` (Docker healthcheck), `/docs`/`redoc` (Swagger) and `/setup/*` remain open.

The 7 screens

1. **Identity** — company name, sector (LAW/DESIGN/HEALTHCARE/ ACCOUNTING/ARCHITECTUR optional logo (PNG/JPG/SVG 500 KB, embedded in DB as BYTEA).
2. **Topology** — dynamic list of branches. One site is the minimum required. Advisory 2 LAN pools for RAID redundancy (see §4.5).
3. **Hardware** — dormant CPU workers enabled + number of GPU/NPU servers + optional provisioning hub URL.
4. **LLM models** — tier selection (fast 9B / reasoning 35B / code 30B). Wheels are downloaded **after** install.
5. **Security** — audit log retention (6/12/24/120 months or unlimited). Federation Ed25519 keys are auto-generated at first boot.
6. **Admin account** — email + password (12 char min).
7. **Recap + Go** — preview of sector defaults that will be applied (retention, audit grade, template description), final confirmation button.

Sector defaults applied at completion

Sector	retention_months_min	audit_grade	template
LAW	120 (10 years)	strict	“Client matter:”
HEALTHCARE	240 (20 years)	iso27001	“Patient:”
ACCOUNTING	120 (10 fiscal years)	strict	“Accounting file:”
ARCHITECTURE / ENGINEERING	60	basic	“Project:”
DESIGN	24	basic	“Client deliverable:”
CONSULTING	60	strict	“Engagement:”
OTHER	12	basic	“”

Admin-first: an explicit admin choice (e.g. `retention=12` in LAW) is always honoured. Defaults only fill fields left NULL. A warning is recorded in the DB (`cluster_setup_state.sector_defaults_applied` JSONB) for audit. Legal obligations (10-year lawyer archive, 20-year medical record) remain the customer’s responsibility.

Wizard state persistence

- Server-side: `cluster_setup_state` single-row table (CHECK id=1).
- Client-side: `localStorage` restores state in case of accidental refresh (admin password excluded, always wiped before persist).

Post-completion behaviour

- `completed_at` set in DB → middleware cache invalidated → application routes unblocked immediately.
- `POST /setup/step/*` after completion → `409 Conflict` with the message "use re-open flow (chantier #17 license)".
- Admin account created in `admin_users` (email + Argon2id-hashed password, security chantier 2026-04-27 PM).

Forgotten admin password — air-gap CLI reset Air-gap doctrine: no external email. The IT admin resets the password directly inside the pool container 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
```

The helper: - prompts for the new password on stdin (getpass, no terminal echo) - hashes it with Argon2id via `passlib.hash.argon2` - updates `admin_users.password_hash` - no network output, works in fully air-gapped environments

Required permissions: SSH access to the host + `docker exec` rights on the pool container. No HTTP port is exposed for this flow (vs. the classic web "forgot password" that opens an attack surface).

Migrating legacy passwords At pool boot, if `admin_users` still contains plain-text passwords (image predating the 2026-04-27 PM security chantier), the pool re-hashes them automatically with Argon2id. The operation is idempotent (skips if the `$argon2` prefix is detected). Existing admins keep logging in with the same password — the migration is silent.

Re-opening the wizard Not available in v1. Planned for chantier #17 license: re-opening will require a new valid license key + double admin confirmation. This guard protects the infrastructure from accidental reconfiguration.

To force a reset **in dev mode only**: drop the `cluster_setup_state` row via direct SQL and restart the pool. Never do this in production.

---

2. Worker

2.1 What is a worker

A worker is a machine that runs an LLM (or an embedding batch). It can be a dormant employee endpoint (CPU) or a dedicated server (NPU/GPU). Principle: "monetize dormant compute". See `project_vision_dormant_compute_pitch`.

Two complementary roles:

Role	Typical hardware	Workload
LLM worker	dev workstation / GPU server	inference (chat completions, summaries)
Embedding worker	dormant CPU endpoint	MiniLM 384d batch (RAG project_documents, smart routing)

A single endpoint can serve both. The central pool dispatches based on availability and current load. **RAG chantier Decision #6 locked**: embedding compute is dispatched by the central pool, never local-only — this is what enables harvesting dormant CPUs for RAG without disturbing LLM workers.

2.2 Provisioning

The admin dashboard (chantier #14 Worker Provisioning) generates **ready-to-paste snippets** for your orchestrator:

1. Open the admin hub → "Workers" tab
2. Click "Add a worker" → choose OS + hardware (CPU / GPU)
3. Copy the 3 generated snippets:
   - `config.json` (with single-use provisioning token)
   - `install.sh` (wheel download + dependencies)
   - `start.sh` (launch command + systemd unit)
4. Paste into Ansible / Intune / Jamf / manual script

The provisioning code expires after first use or 24h, whichever comes first. See `project_chantier_14_worker_prov`

2.3 systemd services

Sample unit file `/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: always launch via systemd, never directly from a shell (see `feedback_z2_proxy_systemd_check` — orphan processes = silent crash on reboot).

2.4 Troubleshooting

Worker won't connect

- Check `systemctl status iamine-worker` (must be `active (running)`)
- Verify pool connectivity (WebSocket port open, TLS certificate)
- `journalctl -u iamine-worker -n 100 --no-pager`

Orphan process (see `feedback_z2_proxy_systemd_check`) Symptom: the worker is running from a shell command line instead of systemd → silent crash on reboot. **Always launch via systemd**, never directly. Verify with `ps -ef | grep iamine` that no orphan instance is running.

Uploaded document never indexed (RAG)

1. Check `indexed_status` via GET /documents: `pending` / `skipped` / `failed`
2. If `skipped`: mime_type outside the whitelist (TXT / MD / PDF / DOCX / ODT) or size > 50 MB, or extracted text empty
3. If `failed`: embedder unavailable → `pip install sentence-transformers` + pre-download MiniLM (`IAMINE_EMBEDDER_PATH` env var, default `~/.cache/iamine-embedder`)
4. If `pending` for > 10 min on a < 10 MB doc: central pool saturated → check CPU + queue metrics
5. For PDF/DOCX/ODT extraction: soft-deps — install manually if needed (`pip install pypdf python-docx odfpy`)

Pool dies between upload and end of ingest If the host pool goes down between `POST /documents` and the end of background indexing, the document remains in `indexed_status='failed'` with the binary unprocessed. **V1 has no auto-recovery**: the operator must re-upload from the original source or from an external backup (Z2 / NAS), then re-issue `POST /documents` with the same `content_hash` (DB idempotence removes orphan chunks if any and re-indexes).

To avoid this scenario in multi-pool prod: ensure the upload reaches its host pool before the client closes the session (ACK on upload, not on end-of-ingest).

---

3. Proxy

3.1 Unified entry point

A single DNS/IP → the whole cluster. Hides the physical topology (N federated pools) behind one URL.

- **Listen**: `:18080` (configurable via docker-compose ports)
- **Routing**:
  - `/v1/admin/*` → sticky session per client IP (cookie consistency)
  - everything else → least_conn round-robin with auto failover
- **Failover**: `max_fails=3, fail_timeout=30s` removes a sick pool from the upstream for 30 s before retrying
- **WebSocket**: `Upgrade`/`Connection` headers preserved for live dashboard events

Reference config: `src/cellule_pro/appliance/entry/nginx.conf`. Tune the upstreams to your topology (typically `:18081/82/83`).

3.2 TLS and certificates

Recommendation: terminate TLS at the **edge**, not inside the internal nginx container. Options:

- **Cloudflare Tunnel** (battle-tested in cellule.ai prod, see `reference_cloudflare_tunnel_master`) — auto-cert, managed rotation
- **Customer reverse-proxy** (Traefik, Caddy, system nginx) — with Let's Encrypt auto-renew
- **Private VPN** (WireGuard, Tailscale) — no public cert, network- layer encrypted traffic

The internal entry nginx can stay HTTP `:18080` if traffic never crosses the Internet without an upstream TLS layer. Otherwise, add `listen 8443 ssl` + a mounted certificate.

3.3 WebSocket upgrade

Already present in the reference `nginx.conf`:

```
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;
}
```

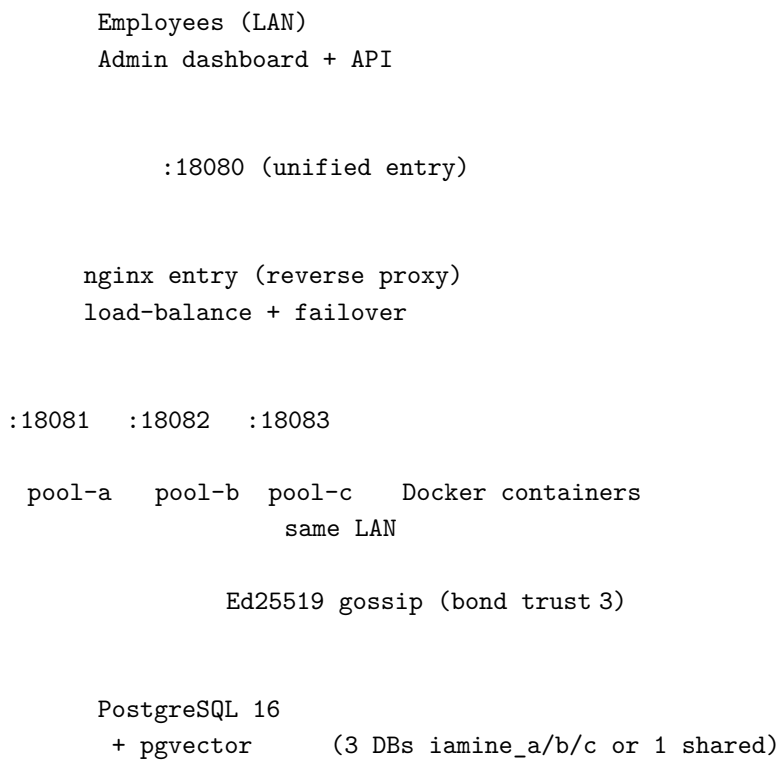
Timeouts: `proxy_read_timeout 300s` tolerates long inferences without dropping the connection.

---

4. Topology

4.1 SMB cabinet — single-site 3-pool LAN RAID

Minimal configuration for a law firm / SMB:



Properties: - 3 pools = software RAID: losing 1 pool does not break the service - Same LAN → gossip latency <1 ms, fast convergence - nginx entry hides the physical topology, single client DNS - Daily PostgreSQL external backup (to Z2 / NAS)

Refs: `feedback_enterprise_lan_raid_minimum`, `CELLULE_PRO_ARCHITECTURE.md`, `appliance/cabinet/README.md`.

4.2 Corporate — multi-site WAN

For a large multi-site enterprise (factory, regional branches):

- Each site = 1 or several pools (local LAN)
- Sites connected by private VPN or secure tunnel (see `feedback_enterprise_multisite_wan`)
- Cross-site Ed25519 federation with trust 3
- Worker `home_pool_id`: mobile (reassignable across sites) vs on-site (locked to the site), see `project_chantier_13_proxy_homing`
- Tolerated gossip latency: up to ~200 ms (LAN = WAN secured by architecture — `feedback_lan_wan_archit`

Chantier #10 + RAG gap: cross-pool propagation of chunks/memories/documents is not wired in v1 (only `moe_local_config` + `project_creation` propagate). Commercial blocker for Corporate — top-priority follow-up. See `project_todo_propagation_extend_project_tables.md`.

Reference: `feedback_enterprise_multisite_wan`.

4.3 Ed25519 federation

Each pool owns a unique Ed25519 keypair. The **bond** (mutual recognition) is the only way for two pools to gossip.

Key generation (at install time):

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

Initial bond (one-shot, via dashboard or admin CLI): - Each pool publishes its pubkey - The admin manually adds each peer's pubkey to `federation_peers` - Default trust = 1 on add, rises to 3 after 24h of stability

Signature: every propagated mutation carries an Ed25519 signature. A pool refuses an unsigned mutation or one signed by a peer not bonded with trust 3. See `chantier #10 invariant`.

Rotation / revocation: no dedicated v1 API. On key compromise → back up the config, regenerate, rebond manually (brief gossip downtime acceptable).

4.4 Ports and firewall

Port	Service	Exposed to
`18080/tcp`	unified nginx entry (API + admin)	LAN clients (or public TLS via reverse-proxy)
`18081-83/tcp`	internal Docker pools	**internal only** (never public)
`5432/tcp`	PostgreSQL	**internal only** (never public)
`8080/tcp`	pool.py inside the container	**internal only**
WebSocket `/ws`	dashboard live events	via `:18080` (upgrade by nginx)

Sample firewall: - INGRESS: allow `:18080` from LAN clients (or Cloudflare tunnel) - INTERNAL: allow ports 18081-83 + 5432 only between Docker containers (bridge network) - EGRESS: pool needs to reach `https://cellule.ai/pypi` (updates) if enabled — otherwise no egress required (air-gap possible)

4.5 Critical multi-host guidelines (discovered live 2026-04-22)

Rules validated on the Z2 + Gladiator cluster (2 physical machines, 4 pools, LAN) before customer rollout. Extracted from `project_preconisations_note_technique_mise_en_place`.

P1. `POOL_URL` **must be LAN/WAN-addressable** **Non-negotiable rule**: each pool advertises a URL reachable by **all other pools** in the cluster. A Docker-internal hostname (`http://pool-a:8080`) only works intra-host; multi-site = broken.

Bad (bug): `POOL_URL=http://pool-a:8080` **Good**: `POOL_URL=http://pool-a.entreprise.local:18091` (internal DNS FQDN) or as a fallback `POOL_URL=http://10.0.1.5:18091` (LAN IP).

Impact of forgetting: peers store a non-routable URL in their `federation_peers`, the handshake may pass but later gossip silently fails. RAG chunks no longer propagate cross-host.

P2. Ed25519 keys on a persistent volume `IAMINE_FED_KEY_DIR` must point to a **named** Docker volume (not a bind mount on `/tmp`). Losing `self_ed25519.key` = a new `atom_id` on restart = ALL existing bonds break and require manual re-bond.

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

P3. Reciprocal bonds + trust=3 promotion required for gossip

- `POST /v1/federation/admin/register` with `reciprocate=true` (default auto-bond) creates both rows in `federation_peers` in one call.
- Initial bond is trust=1. Application-level gossip filters at `trust 3` (`MIN_TRUST_LEVEL` in core.py).
- **Explicit manual promotion** by the admin via `POST /v1/federation/peers/{atom_id}/promote?token=$A` body `{"target_level":3}` — opt-in security.

Auto-promotion after N stable days = upcoming admin toggle (see `feedback_admin_toggle_everything`).

P4. `IAMINE_BOND_PEERS` **format** CSV `host:port` or full URLs. Multi-site example:

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

Consumed by the entrypoint at boot: it waits for the pool to be healthy then issues `POST /v1/federation/admin/register?reciprocate=true` for each listed peer.

P5. Admin-toggle env vars

Variable	Values	Effect
`IAMINE_FED`	`active` / `observe` / `off`	Federation master switch (default via entrypoint = `active`)
`IAMINE_FED_GOSSIP_ENABLED`	`true` / `false`	Application-level project_* gossip (default `true`)
`CELLULE_DEBUG_INGEST`	`1` / unset	`ingest_inline` debug endpoint (default unset = 404)
`ADMIN_PASSWORD`	strong string	Mandatory rotation before prod, store in a vault

P6. Per-pool DBs (never shared) Doctrine: each pool has its own DB. Chantier #1 gossip replicates Ed25519-signed rows cross-pool. Sharing a DB = SPOF, contradicts the N-pool RAID promise. Multi-site WAN is physically impossible with a shared DB (latency).

4.6 HA failover & cross-pool auto-orchestration

Doctrine `feedback_pool_virtuel_orchestre_jamais_manuel`: the virtual pool decides on its own when to migrate / fail over workers and proxies based on **load** AND **outage**. No manual move by the admin. The IT director's role = define policy via the UI; never move a piece by hand.

4.6.1 Reactive failover (pool outage) 3-tier anti-flap hysteresis applied by each proxy to its primary pool:

Tier	Threshold	Behaviour
PRIMARY_OK	<90s without reconnect	Retry primary every 30s, silent log
FAILING	90s to 5min	Stay on primary (anti-flap), audit-log event
DOWN	5min in auto mode	Fail over to a bonded peer via the `peer_pool_urls` received during the handshake

When the primary returns, **sticky return in 60s** (cycle `_sticky_primary_check`) brings the proxy back home. No job is lost: bonded peers accept migrated proxies through the cross-pool trust mechanism.

Driver: `CELLULE_FAILOVER_MODE=auto|manual` (default: `auto`). Mode `manual` disables the automatic tier-3 failover — the admin must trigger it explicitly (`force_migration` via UI). 24h failsafe in manual mode: if the admin has not responded after 24h, automatic failover engages to preserve availability.

4.6.2 Orchestrated migration (active rebalancing) Broader than mere failover: the balancer issues `worker_migration_orders` based on observed load and the employee allocation, without waiting for a failure. Mechanism:

1. `worker_balancing.balancer.compute_balancing_plan(pool)` computes the target distribution (employees × tier × pool).
2. `emit_migration_orders` inserts an Ed25519-signed row per desired move.
3. `migration_executor` (5s loop) walks each order through the state machine `issued → draining → completed` (or `rolled_back` on 90s timeout).
4. On the `draining` step, the executor sends the `cmd: migrate_pool` WS command with `target_pool_url` to the proxy.
5. The proxy ACKs draining → drains its in-flight jobs (60s graceful) → ACKs drained → switches its `pool_url` (permanent migration) → reconnects to the target.
6. On the source pool, the executor detects the disconnected worker → marks the order `completed` (and updates `worker_proxy_registry.current_pool_id`).

Global toggle: `balancer_config.executor_loop_enabled` (default `true`). Steerable via `PUT /v1/admin/dashboard/` OFF = no orchestrated migration runs (orders stay `issued`, the admin can `reject` them manually).

4.6.3 Master / satellite — model catalogue failover For **catalogue mutations** (`.gguf` upload, hardware_class assignment, delete), a single pool is the *master* at any given time. The others are *satellites* (read-only catalogue).

If the master dies, satellites **never auto-promote**: - Satellites keep serving the existing catalogue read-only - Inference on local + cross-pool workers keeps working - Uploads and assignment changes return HTTP 409 with the hint `"write to the master pool"` - The IT admin explicitly promotes a satellite via the dashboard UI (`POST /v1/admin/dashboard/models/promote-master`) — button with confirmation, Ed25519 signature, audit trail, email to admins.

When the historical master comes back, it bootstraps as a **default satellite** (anti-reclamation). The admin re-promotes explicitly if they want the master pinned back to the historical site.

Doctrine: zero magic, the admin drives, audit trail. Avoids catalogue split-brain (two simultaneous masters = two diverging catalogues = mandatory manual reconciliation). See `project_section_5_4_phases_g` invariants.

4.6.4 Pre-prod failover testing Validation procedure before a multi-site customer rollout:

1. Prepare two federated pools with reciprocal trust 2.
2. Physically stop a proxy's primary pool: `docker stop pool-X`
3. Watch the proxy logs: `tail -f ~/.cellule/proxy-*.log`
4. At T+5min: transition `FAILING → DOWN (outage 300s, mode=auto)` then `Connecting to ws://...peer.../(fallback)`.
5. Verify on the peer pool: `curl http://peer:18095/v1/status` must show the migrated workers.
6. Restart the primary: at T+60s, automatic sticky return.

4.7 HA virtual pool — keepalived VIP (chantier rc62)

> **Problem solved**: before rc62, an API client (opencode, Cursor, OpenWebUI) had to point its `baseURL` at a concrete pool's IP (e.g. `http://192.168.1.199:18095`). If that pool went down, the client was stuck — every workstation had to be manually reconfigured to point at another pool. The M7a inter-pool forward covers "healthy but saturated / model missing", but not "pool down".

Solution: a **single virtual IP** (VIP) shared between every pool on the LAN, owned by `keepalived` (VRRP). Only one pool holds the VIP at a time (the "VIP master", chosen by priority). If the master dies, a backup pool grabs the VIP within 3 seconds — the client sees no outage.

4.7.1 Architecture

```
opencode client
 baseURL: VIP


  192.168.1.251  (single VIP)
```

```
                    (carried by keepalived)


    Pool A (master)      VRRP      Pool B (backup)
    priority=200        advert         priority=100
    + HA sidecar                       + HA sidecar
```

Each pool ships a `keepalived-pool` **sidecar** (same Docker image, dedicated entrypoint). The sidecar runs in `network_mode: host` + `cap_add: NET_ADMIN` so it can bind the VIP on the physical interface. At boot it dynamically fetches its configuration from the local `/v1/cluster/vip-self` endpoint of the pool — no manual env var, the wizard drives.

4.7.2 Activation in the first-boot wizard Wizard's **Network** step (`/setup/ui`), section **HA virtual pool**:

1. Tick *"Enable HA virtual pool"*
2. Choose mode:
   - **First pool of the cluster**: enter `vip_address` (a free IP in the LAN subnet, e.g. `192.168.1.251`) + the LAN CIDR mask (24, 22, 16…). The VRRP password is auto-generated.
   - **Join an existing cluster**: enter the URL of an already installed pool (e.g. `http://192.168.1.199:180` The wizard fetches the master's VIP config via `/v1/cluster/vip-info` (admin auth) — no manual re-entry.

The sidecar starts keepalived as soon as the config is in the DB.

4.7.3 Compose-side activation The sidecar lives in the Docker Compose `vip` profile, **disabled by default**. For simple single-pool deployments without HA, leave the profile inactive. For HA, activate via:

```
COMPOSE_PROFILES=vip docker compose up -d
# or add to .env:  COMPOSE_PROFILES=vip
```

4.7.4 Operational checks On the master pool:

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

On backup pools, the VIP is NOT in `ip a` — that's normal.

Failover test (on the master pool):

```
docker stop pool-<host>-h
# Wait ~3-8s
ip -br -4 addr show | grep <VIP>   # nothing
```

On a backup pool:

```
ip -br -4 addr show | grep <VIP>   # the VIP appeared
```

Automatic recovery: `docker start pool-<host>-h` → after ~30s, the VIP returns to the original master (priority 200 > 100, preempt).

4.7.5 Network prerequisites

- Flat switched LAN (VRRP uses `224.0.0.18` multicast). Works by default on every standard L2 switch.
- VIP free in the subnet at setup time (not pingable).
- All cluster pools must share the **same** `vip_vrid` and the **same** `vip_auth_pass` (the wizard propagates them automatically in "join cluster" mode).
- Unique VRID on the LAN if several VRRP clusters coexist (rare).

4.7.6 Doctrine: what the VIP does NOT do

- **Workers and proxies do NOT use the VIP.** They keep pointing at their enrolment pool and benefit from cross-pool hysteresis (Phase 1+2, see §4.6) to migrate on failure.
- **No active load-balancing.** keepalived = failover only, the master takes all the traffic. The M7a inter-pool forward remains the internal load-balancing mechanism.
- **No cross-WAN.** The VIP is LAN-only (multicast). For multi-site, plan one VIP per site + DNS health-check or anycast on the upstream (out of scope for rc62).

4.8 Latency-based worker auto-routing (chantier rc81)

Doctrine: the employee workstation is declared once on the cluster VIP. The virtual pool then decides which physical pool to route that worker to, based on actual network proximity. No per-workstation gesture from IT, no pool choice on the employee side.

Use cases: multi-site cluster (Paris + Lyon law firm, hotel chain, multi-factory industry). Each site has its local pool, federated via Ed25519. All employee endpoints (Win / macOS / Linux) target the central VIP via a single install one-liner. At boot, the Lyon employee is naturally enrolled on the Lyon pool (local LAN, minimal latency) without intervention.

Mechanism: - At the WebSocket handshake, the master extracts the worker's `/24` subnet (from `ws.client.host`) - It compares it with its own `federation_self.url` and every bonded peer (`trust_level >= 2`) - If a peer shares the worker's `/24` (and is not the current master), an Ed25519-signed `cmd: migrate_pool` order is inserted in `worker_migration_orders` and notified to the worker - The worker drains its in-flight jobs (max 60s), persists its `worker.json` to the new pool, closes the WebSocket, and reconnects on the target pool. Permanent (not a temporary fallback).

Admin flag: `WORKERS_AUTO_LATENCY_ROUTING` (`cluster_routing` category, default `false`, opt-in).

When to enable	When to keep OFF
Stable multi-site cluster, validated bonded Ed25519 federation	Single site or cluster still in setup
Employee endpoints split across several `/24` subnets	All endpoints on the same `/24` as the master pool

When to enable	When to keep OFF
You have observed the VIP enrolling workers far from their local LAN	You want every worker to route through the master for centralised audit

Anti-pattern: do not attempt to re-route a worker oscillating between two close peers. Hysteresis is built in: 1 order per worker (idempotent on PK `migration_order_id`), a new order only if the worker reconnects elsewhere and the computation yields a better peer.

Known rc81 limitations: - `/24` heuristic only (no active RTT ping). Sufficient for typical enterprise LAN/WAN. A future extension can add ping for ambiguous cases (NATed subnets, site-to-site VPN). - The master always remains the master. Only workers move, never the pool's role. Consistent with the doctrine "the virtual pool orchestrates, never manual migration".

Visual reference: see the diagram `docs/architecture/layer-worker-latency-routing.html`.

4.9 GGUF model selection — IT director recommendations

Cellule-PRO serves any `.gguf` compatible with llama.cpp uploaded via the admin dashboard. But not all models behave the same on the assistant chat UX. Three families to know about before picking.

4.9.1 Instruct non-thinking models (recommended POC + prod) Variants explicitly trained to answer **directly** without visible chain-of-thought. No inline reasoning, no analytical pre-amble. Out-of-the-box behavior matches a chat assistant.

Tested whitelist: - `Qwen3-4B-Instruct-2507-Q4_K_M.gguf` (~2.4 GB) — official non-thinking variant, 262K native context. Recommended default model for `gpu-small` + `cpu-mid` hardware classes. - `Qwen3-30B-A3B-Instruct-2507-Q4_K_M.gguf` (~18 GB) — large non-thinking variant for `gpu-large`. - `Llama-3.x-Instruct` (any size) — Meta Llama Instruct. - `Mistral-Instruct` (7B, Small, Large) — Mistral AI Instruct. - `Phi-3-mini/medium-Instruct` — Microsoft, efficient small models.

4.9.2 Thinking models (caveat) Variants trained to expose their reasoning, either with `<think>...</think>` tags or as explicit numbered inline analysis. Cellule-PRO automatically strips `<think>` tags, but **inline numbered reasoning** ("1. Analyze the Request:…, 2. Draft Response:…") may pass through filters depending on SYSTEM_PROMPT and model size.

Affected models: - `Qwen3-*` (without `Instruct-2507`) — Qwen3 v1, thinking on by default. - `Qwen3.5-4B/9B-*` (without `Instruct`) — Qwen3.5, same. **Particular**: Qwen3.5-4B in Q4 tends to show inline numbered reasoning even with a SYSTEM_PROMPT requesting "answer directly" — it's in its training. - `QwQ-*` — Qwen reasoning model, thinking by design. - `DeepSeek-R1-*` (and distillations) — reasoning model.

Recommendation: for a standard employee chat deployment, **avoid** these variants as the primary model. Use them only for use cases where explicit reasoning is desired (research, deep analysis, algorithmic debugging).

If you must serve a thinking model anyway: - Pool-side strip removes `<think>` tags automatically. - Defensive regex filters on numbered "1. **Analyze:**" patterns. - Variable performance depending on model.

4.9.3 Base models or proprietary fine-tunes Non-instruct ("raw" model) variants or in-house IT-director fine-tunes. Behavior not guaranteed — may produce unformatted chat replies, completions instead of answers, leaks of special tokens.

Recommendation: use only with an application wrapper (custom API gateway) that properly formats inputs/outputs. **Not recommended** as a model served directly to employees.

4.9.4 How to choose for my fleet

Typical client hardware	hardware_class	Recommended model
Win/Linux desktop 8-16 GB RAM CPU only	`cpu-mid`	Qwen3-4B-Instruct-2507 Q4_K_M
Workstations RTX 2060/3060 6 GB VRAM	`gpu-small`	same (Q4_K_M fits in 6 GB ctx 4K)
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)
Code-specialized (DevOps, dev assistant)	none (dedicated proxy)	Qwen3-Coder-30B-A3B-Instruct

The IT director can upload multiple models and assign by `hardware_class` via the **Models** page in the dashboard. The pool automatically routes each worker to its assigned model via §5.4 hardware_class catalog.

Before uploading an unlisted model: run a control chat test "hello" + "write an email to my manager" + "summarize this text" with a guinea-pig worker in isolation mode. If the answer contains a numbered analytical preamble, it's a thinking model — switch to an Instruct non-thinking variant.

---

5. Benefits

Operational pitch for IT directors and technical decision-makers. Angles:

5.1 Compute frugality

Cellule-PRO runs on hardware **already present** in the enterprise: dormant employee endpoints (CPU), already-amortized servers. No dedicated datacenter GPU purchase, no monthly cloud bill.

Order-of-magnitude estimate (to be precisely measured in a whitepaper): ~14× lower kWh/token footprint than a dedicated datacenter GPU deployment (in-service hardware, only dormant cycles consumed). Sources to produce: ADEME, EPRI, Anthropic/Google reports.

5.2 Cloud complement (not substitute)

Cellule-PRO does not replace ChatGPT Enterprise on frontier-class reasoning. It covers ~**70% of common LLM requests**: email summarisation, classification (urgency, internal routing), internal translation, information extraction, Q/A on internal documents (RAG chantier).

The complex ~30% (frontier legal reasoning, strategic analysis) keep going to the cloud through business segmentation. The two coexist.

5.3 Data sovereignty

Strongest argument for the law firm / healthcare / defence segments: - No data leaves the customer LAN - Compatible with attorney-client privilege (sending client data to a non-authorised third party = professional misconduct in France) - GDPR per-project data segregation (cryptographic isolation, see `project_chantier_10_invariants`) - Append-only immutable audit trail (`project_audit`, ISO 27001) - Right to erasure: CASCADE DELETE proven e2e (RAG chantier Phase 4 test)

The RAG chantier delivered this session strengthens the argument: private knowledge (archives, contracts, emails) becomes semantically searchable **without leaving the site**.

5.4 Ecology

Reusing dormant hardware = **zero new purchase, zero manufacturing cost** attributable to Cellule-PRO. The existing fleet keeps serving, just more efficiently.

Precise figures to be produced (dedicated whitepaper) with sources ADEME, EPRI, cloud-provider environmental reports. Angle: CO2 per useful token vs. brand-new dedicated H100.

5.5 Operational simplicity

Doctrine `feedback_compliqué_de_faire_simple`: - 3-pool cabinet install: `git clone` + `docker compose up -d` (see `appliance/cabinet/install.sh`) - Smart defaults (ports / paths / theme) — only edit if needed - Actionable French admin UI, no jargon - Unified web admin hub (no mandatory CLI) — chantier #11 - Onboarding wizard for first projects - Backup and restore documented with exact commands (see §6) - **Cross-site HA without admin intervention**: a pool dies, proxies automatically migrate to bonded peers (5 min anti-flap hysteresis). The virtual pool also rebalances based on observed load, without waiting for a failure. See §4.6 for the full mechanism (Phase 1 reactive + Phase 2 orchestrated).

---

6. Appendix

6.1 Logs to watch

Docker containers:

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

Structured Python loggers (tags appear in the emit format): - `cellule_pro.document_ingest` — RAG pipeline (`skipped` / `failed` / `done`) - `cellule_pro.document_search` — KNN queries - `cellule_pro.project_scope` — schema migrations - `cellule_pro.memory_fact_replication` — chantier #1 gossip - `iamine.routing_embeddings` — MiniLM load status - `uvicorn.access` — HTTP requests

PostgreSQL slow queries (enable in prod):

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

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

6.2 Common troubleshooting

Symptom	First check	Common fix
Pool down (health 503)	`docker ps`	`docker restart pool-X` + check logs
Silent worker	`systemctl status iamine-worker`	restart service, check pool connectivity
High latency	CPU + embedder loaded	too many parallel requests → add idle worker
Federation `bad signature` error	stale peer pubkey	re-bond peer via dashboard
Uploaded doc not indexed	`indexed_status` via API	see §2.4 “Uploaded document never indexed”
RAG search returns empty	chunks generated?	check `chunks_count` on doc, re-ingest if 0
DB “vector extension missing”	vanilla postgres image	use `pgvector/pgvector:pg16`
CASCADE DELETE failed	FK constraint disabled	re-apply `ensure_document_rag_schema`

6.3 Incident runbook

Detailed recovery procedures: `reference_pra_z2` (DRP template in 8 sections, RTO 4-8h, to be tailored to the customer cluster).

Scenarios covered: - Loss of one pool out of three → service continues (RAID), re-join after fix - Loss of the pgvector volume → restore from external backup (Z2 / NAS), re-migrate schema - Ed25519 key leak → regenerate keypair + manual re-bond + rotate dashboard tokens - Corrupted DB → `pg_basebackup` restore then `ensure_*_schema` (idempotent migrations = safe) - Total pool host loss → uploaded documents in `indexed_status=failed`, manual re-upload from customer source (see §2.4 “Pool dies mid-ingest”)

6.4 Cross-references

- `BOUNDARY.md` — public PRO interface contract

- `CELLULE_PRO_ARCHITECTURE.md` — detailed architecture
- `SECURITY_MODEL.md` — security model
- `API_REFERENCE.md` — API reference for integrators
- `CONTRIBUTING.md` — internal contribution rules

---

7. Cellule-PRO license

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:

```python
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`:

```python
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:

```bash
# 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 tolerant, no service interruption). 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).