

Kedebah Commerce

New Deployment Architecture, Performance Benefits & Scaling Strategy

Executive Brief — Top Up Pharmacy Go-Live Platform

Purpose: Explain why the new containerised Commerce stack is faster and more resilient, what it means for multi-outlet POS (Top Up + merged Lena / Express), and exactly what to adjust when traffic grows — without requiring engineering jargon.

One-sentence summary: We separated the public website edge, the Finance API, background jobs, realtime websockets, and cache into dedicated containers; tuned PHP and Nginx for concurrent tills; and protect PostgreSQL with PgBouncer — so more outlets can trade at once without the old “everything fights for one process” pattern.

1. Why this architecture change

Commerce is no longer a single-tenant light load. Top Up Pharmacy now runs as the survivor on new infrastructure, with Lena and Express Care merged into the same tenant footprint. That means more concurrent POS sessions, heavier stock and invoice APIs, and background work (alerts, email, tenant maintenance) that must not slow the till.

The previous pattern mixed web traffic, jobs, and supporting services more tightly. Under peak outlet activity, request latency and queue backlog rose together. The new deployment (**kedebah-commerce-deploy**) isolates those concerns and sizes each layer for concurrent use.

2. How the new stack is organised

Users hit **topup.kedebah.com** (and the finance portal). Only the **edge** proxy is public. Everything else communicates on a private Docker network.

Layer	Role
Edge (Nginx + TLS)	Public entry; routes SPA, API, storage, websockets
Finance API	Core Commerce / POS / inventory / invoices API
Commerce SPA	Browser POS & inventory UI
Redis	Cache & sessions (less load on the database)
Reverb	Realtime inventory / commerce events
Dedicated workers	Queues & schedules (jobs off the web path)
Auth / Onboarding / Email / SMS	Supporting services, also containerised
PostgreSQL + PgBouncer (host)	Shared DB; pooling so connections do not explode

Diagram (simplified): Internet → Edge → Finance / Commerce / Reverb; workers consume queues in the background; all apps → PgBouncer → PostgreSQL.

3. Business & operational benefits

Faster under concurrent use

- **~216 concurrent PHP request slots** across services (Finance alone up to **96** workers) — many tills and browsers can be served at once.

- **OPcache + JIT** keeps PHP bytecode hot in memory — less CPU wasted recompiling the same Laravel code on every request.
- **Nginx keep-alive** and a high-capacity edge reduce connection churn at peak.
- **Redis** for cache/sessions reduces repetitive database work.

More stable when busy

- Web traffic no longer shares the same process as email, stock backfill, or scheduled alerts — **background work cannot starve the till**.
- **PgBouncer** pools DB connections so raising PHP capacity does not open hundreds of raw Postgres connections.
- Workers recycle periodically (memory hygiene); config/route caches warm on start.

Easier to grow deliberately

- Clear knobs: raise PHP workers, scale a worker container, or add a Finance API replica — without redesigning the product.
- Same compose project for deploy, rebuild, and rollback — fewer “works on one server, mystery on another” incidents.

4. Capacity we ship by default

Sized for multi-outlet POS go-live. Approximate RAM budget for this Commerce stack: **14–18 GB** on a large shared host.

Service	PHP workers (max)	Approx RAM
Finance (main API)	96	~6.0 GB
Commerce SPA host	32	~2.0 GB
Auth	32	~2.0 GB
Onboarding	24	~1.5 GB
Email / Tenant / SMS	16 each	~1.0 GB each
Redis (cache/session)	—	Capped (e.g. 2 GB)

5. Current host snapshot (live environment)

Snapshot from the production Ubuntu host running Top Up Commerce (shared machine). Figures rounded for executive readability.

Metric	Observed	Reading
Memory	62 GB total; ~20 GB used; ~35 GB available	Comfortable headroom for Commerce; host is shared with other apps
Swap in use	~14 GB of 31 GB	Indicates past or ongoing memory pressure on the host overall — watch closely as POS load grows
CPU	~96% idle; load ~1.4–2.7	Not CPU-bound at this moment; latency issues more likely DB/query or concurrency saturation
Kedebah-relevant processes	Docker (Commerce stack), php-fpm / queue workers, PgBouncer, Postgres (tenant_topup_pharmacy, kedebah_v2)	Commerce path is active and healthy in this sample

Important context for leadership: This host also runs other platforms (e.g. MySQL, Redis, Node/Python apps, AI/LLM-related processes). Commerce capacity planning must reserve RAM for those neighbours. Long-term, a dedicated Commerce host (or moving co-tenants off) unlocks higher Finance worker counts safely.

6. What happens when traffic increases — and what to tweak

Think of capacity as a funnel. Users hit the edge → PHP workers handle requests → each busy worker may need a DB connection via PgBouncer → Postgres executes queries. **The database is usually the true ceiling**, not Nginx.

Stage A — Mild growth (more outlets / more concurrent tills)

- **Symptom:** Occasional slow API calls; FPM status shows workers often busy or “max children reached”.
- **Tweak:** Raise **FINANCE_FPM_MAX_CHILDREN** (e.g. 96 → 112) in deploy **.env**, recreate finance container. Confirm free RAM and PgBouncer limits first.
- **Also check:** Redis memory under cap; edge/proxy timeouts already set for long POS operations (~180s).

Stage B — Queue / job backlog (alerts, email, stock jobs lag)

- **Symptom:** UI feels fine but notifications, mail, or background stock work arrive late.
- **Tweak:** Scale workers — e.g. `docker compose up -d --scale finance-worker=2` (same pattern for email/onboarding if needed).
- **Why it works:** Jobs use separate containers; adding workers increases throughput without changing the website.

Stage C — Sustained API saturation (Finance always at max)

- **Symptom:** Finance FPM saturated even after raising workers; response times climb across outlets.
- **Tweak:** Add a second Finance API replica — `docker compose up -d --scale finance=2`. Edge already load-balances via Docker DNS.
- **Prerequisite:** Enough free RAM (~6 GB+ per extra finance replica) and PgBouncer sized for the extra clients.

Stage D — Database pressure (the most common hard limit)

- **Symptom:** High wait times in PgBouncer; Postgres busy; API slow even with spare PHP workers.
- **Tweak:** Ensure **pool_mode = transaction**; raise **max_client_conn** (e.g. ≥ 600) and **default_pool_size** (e.g. 50–80); apps must connect to **PgBouncer**, not Postgres directly.
- **Product/engineering follow-up:** Optimise heavy endpoints (large stock payloads, N+1 queries) — infra cannot fix an expensive query by itself.

Quick decision table

If you see...	First action	Owner
Tills waiting / API timeouts at peak	Check Finance FPM status → raise FINANCE_FPM_MAX_CHILDREN if maxed	Platform
Jobs / email / alerts delayed	Scale the relevant *-worker container	Platform
DB waits / PgBouncer saturated	Tune PgBouncer pools; review slow queries	Platform + DBA / Eng
Host RAM / swap climbing	Do not raise FPM further; free RAM or move co-hosted apps	Infra / Leadership
Only one slow screen (e.g. huge stock list)	Application optimisation — not more servers first	Engineering

7. Guardrails (so scaling stays safe)

- Budget **~50–70 MB RAM per PHP worker**. More workers without free RAM causes swap and makes the system slower, not faster.
- Every busy PHP worker + every queue worker can hold a DB client — expect **~250–320** potential clients at current defaults; PgBouncer must absorb them.
- After OPcache production mode, **code changes require container recreate** (expected for performance).
- Prefer raising FPM on one Finance service before multi-replica unless RAM allows.

8. Recommended leadership takeaways

1. The new architecture is a deliberate investment in concurrent POS reliability, not a cosmetic Docker move.
2. We already have a clear growth path: PHP workers → more queue workers → Finance replicas → PgBouncer / DB — in that order of cost and complexity.
3. The host is shared and already shows swap use; protecting Commerce performance long-term may mean dedicating resources (or reducing co-tenancy) before simply “turning numbers up”.
4. Some slowness will remain application-level (large reports/stock payloads); those need product/engineering fixes alongside infra.

Prepared for executive walkthrough · Kedebah Commerce / Top Up platform · Reference: kedebah-commerce-deploy (architecture + go-live load tuning).

Document version: July 2026 · For internal use