Skynet — Architecture and Deployment Design

For: Agon, Sead, Sven (CTO), Vince. **Companion to:** the approved functional design (what), the workflow source map (how work flows and where each step comes from), the delivery plan (in which order).

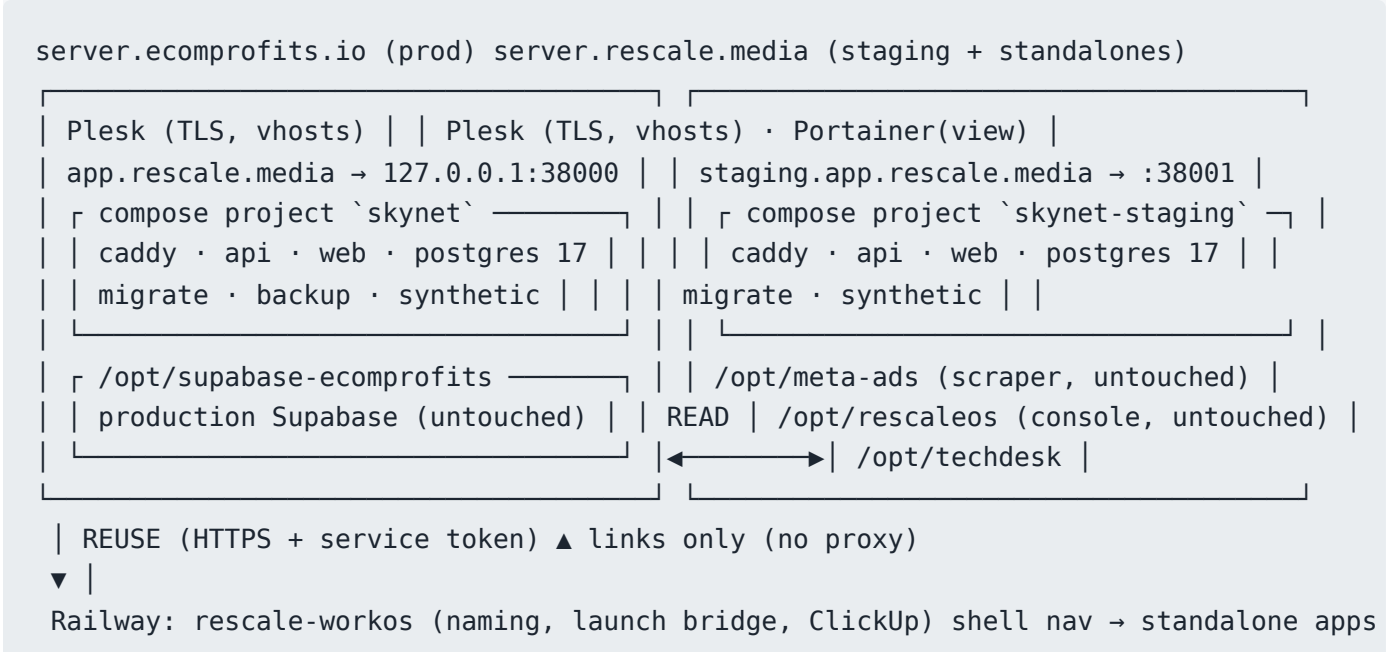
Supersedes the technical plan v5 §2–§4 and the implementation PRD for everything about host, topology, routing, auth scope, staging, deploy, backup and rollback. Contracts, stack and monorepo discipline carry over unchanged.

---

1. Decisions this design is built on

1. **Host.** The unified app runs on `server.ecomprofits.io` (48 cores, 250 GB RAM, 1.2 TB free; Plesk + Portainer; the production ecomprofits Supabase lives here). "We can migrate to a new server anytime" — nothing in this design is host-specific beyond a compose file and a vhost.
2. **Standalone apps stay standalone.** meta-ads-scraper and RescaleOS keep running on `server.rescale.media`. Their functionality is merged into Skynet natively, module by module, in later slices; each standalone is retired only when its replacement is accepted. No sidecar mounting, no cross-host proxying. The only allowed pre-merge change set is bridge track B while S1 runs.
3. **rescale-workos stays on Railway** and is reused over an API (naming assembler, launch bridge). **ecomprofits is read-only** for Skynet, always.
4. **Staging** runs on `server.rescale.media`; **production** on `server.ecomprofits.io`. A change reaches production only after the staging smoke gate passes.
5. **CI/CD and alerting** replicate the ecomprofits pattern (GitHub Actions → GHCR → SSH → `docker compose pull/up` → Telegram), with one improvement: the whole env file comes from one CI secret.
6. **Delivery = 100 % complete.** No calendar dates; slices are a sequence with gates. S1 exit gate is §12.
7. Backups: Skynet has its own backup path and sweep on the production box, independent of the ecomprofits pipeline. Agon owns Plesk vhosts and TLS on both boxes.

2. System boundaries and coexistence



Five boundaries, each with one rule:

- **Skynet ↔ ecomprofits Postgres:** read-only role `skynet_readonly`, account-scoped queries, one view (`skynet_creative_daily`); no writes, ever. Skynet's own Postgres is a separate container; it never shares the Supabase instance.
- **Skynet ↔ workos:** HTTPS to Railway with a Skynet service token; workos owns ClickUp; Skynet writes only human-owned fields through the bridge.
- **Skynet ↔ scraper / RescaleOS:** links in the shell nav, nothing else, until each merge slice. Their logins remain separate; bridge track B may add temporary features in those standalone repos while S1 runs.
- **Skynet ↔ the box:** hard resource limits (§9) so the Supabase workload is never starved.
- **Skynet ↔ its own future:** every table carries `workspace_id`; queue rows, outbox and R2 keys carry it; credentials are per workspace. This is what makes later merges and SaaS hardening additive instead of rewrites.

3. Repository (monorepo `Rescale-Media/skynet`)

```
apps/api/ Python 3.13 · FastAPI — core (auth, workspaces, creative codes, bindings, li
apps/web/ React 19 · Vite · TS · Tailwind — shell + Production pages + admin
packages/contracts/ JSON Schema + TS types + Python models + fixtures (auth, creative
db/migrations/unified/<schema>/NNN_*.sql forward-only SQL; runner `apps/api/migrate.py
infra/ compose files (prod, staging, dev), Caddyfile, backup + synthetic scripts, secr
docs/ ARCHITECTURE, decisions/ (ADRs), runbooks/, checklists/
services/ imported code for later slices only — NOT part of the S1 runtime:
 scraper/ transcriber/ agent-runtime/ workos/ (scrubbed, no secrets)
```

Imports keep history (`git subtree`); the workos tree is a scrubbed copy (its history contains credentials). The `services/*` packages are built in CI for tests only; no image is published from the monorepo for them until their merge slice (ADR-006).

4. Stack (per package)

PACKAGE	RUNTIME / FRAMEWORK	IMAGE	NOTES
`apps/api`	Python 3.13 · FastAPI · psycopg 3 · pytest	`ghcr.io/rescale-media/skynet-api`	Base = static-ads backend; `core` module added; supervisord runs `uvicorn` + the job worker (readback, storyboard queue); no Postgres inside the container any more
`apps/web`	React 19 · Vite · TypeScript · React Router · Tailwind v4	`ghcr.io/rescale-media/skynet-web` (Caddy + built SPA)	The one gateway container: serves the SPA, proxies `/api/*`
`postgres`	`postgres:17` + pgvector	official	Schemas `core`, `production`; later `meta_ads` (S2), `studio` (S3)
`migrate`	`skynet-api` image, entrypoint `migrate`	—	One-shot before every deploy
`backup`	`postgres:17` client + cron	—	Nightly `pg_dump -Fc` (§10)
`synthetic`	`curlimages/curl` + cron	—	Health checks → Discord/Telegram (§11)
Model access	OpenRouter default; direct vendor optional; Claude subscription runner = internal binding inside Studio (S3)	—	Per-workflow bindings, spend guard

5. Runtime topology on `server.ecomprofits.io` (production)

```
/opt/skynet/ git checkout of the monorepo (main), compose project `skynet`
 infra/compose/compose.prod.yml name: skynet · network skynet-net (internal) · volumes
.env.production written by CI from secret SKYNET_ENV_B64 (never edited on the box)
```

SERVICE	PORT (HOST)	VOLUMES	PURPOSE
`caddy` (`skynet-web`)	`127.0.0.1:38000:80`	—	`/` → SPA; `/api/*` → `api:8000`; `X-Request-Id`; 512 MB uploads; auth rate-limit; websocket/SSE passthrough; 1 h read timeout for streams
`api`	internal 8000	`skynet_api_tmp`	FastAPI core + production, job worker
`postgres`	`127.0.0.1:38432:5432` (admin tools only)	`skynet_pg_data`	Skynet's own database
`migrate`	—	—	profile `ops`; run by CI before `api` is recreated
`backup`	—	`skynet_pg_data` (ro), `skynet_backups` → host path (§10)	nightly dump
`synthetic`	—	—	5-min checks

The compose files never reference mutable `latest` at deploy time; they read `IMAGE_TAG`, `API_IMAGE_DIGEST` and `WEB_IMAGE_DIGEST` from `.env.release`, so staging and production run the exact images the build job produced. Deploy-time variables live in two files on each host: `.env.production` / `.env.staging` (the full application environment, written by CI from `*_ENV_B64`) and `.env.release` (image selection for that deploy only). Operators never edit either by hand.

Plesk: vhost `app.rescale.media` → proxy `127.0.0.1:38000`, Let's Encrypt, HSTS, `client_max_body_size 512m`. Caddy trusts `X-Forwarded-Proto` only from `127.0.0.1`. Portainer is a viewer; host-side `docker compose` owns every container; nothing is ever edited in the Plesk/Portainer GUI (both rewrite mounts and env — the RescaleOS lesson).

Rules that keep the ecomprofits stack safe on the same box: separate compose project and network (no shared network with `supabase-*`); separate Postgres (never the Supabase instance); published ports only on `127.0.0.1` and only the two above; resource limits (§9); Skynet never touches `/opt/supabase-ecomprofits`; deploys never run `docker system prune`, `down -v`, or anything project-wide.

---

6. Staging on `server.rescale.media`

Same compose file, project `skynet-staging`, port `127.0.0.1:38001`, own network `skynet-staging-net`, own volumes, `.env.staging` from secret `SKYNET_STAGING_ENV_B64`, vhost `staging.app.rescale.media`. Staging reads ecomprofits with the same read-only role (the only shared dependency; it cannot write). Staging does not run scrapers or agents; it proves the unified app, migrations, the read layer, the loop, and the deploy itself. Staging keeps the existing box's backup sweep (Plesk `private/pgdump` on `rescale.media`) for its dumps.

---

7. CI/CD — the ecomprofits pattern, applied

`ci.yml` (PR + main; path-filtered jobs, all required on `main`): `contracts` · `api` (pytest) · `web` (lint + build) · `secrets` (gitleaks with the workos custom patterns) · test-only builds of `services/*`.

`deploy.yml` (push to `main`):

1. **build** → GHCR `skynet-api`, `skynet-web`; tags `main-<sha>` + `latest`; the job records the pushed **image digests** as outputs (`API_DIGEST`, `WEB_DIGEST`).
2. **deploy-staging** (SSH to `server.rescale.media`, `/opt/skynet-staging`): `git reset --hard origin/main` → decode `SKYNET_STAGING_ENV_B64`, validate (non-empty, required keys, no `ANTHROPIC_API_KEY=`), keep `.bak-<ts>`, atomic write → write `IMAGE_TAG=main-<sha>`, `API_IMAGE_DIGEST`, `WEB_IMAGE_DIGEST` into `.env.release` → `docker compose run --rm migrate` → `pull` → `up -d` → wait `/api/healthz`.
3. **smoke-staging** (`infra/smoke/run.sh`): login with a seeded user; `/`, `/api/auth/me`, `/production` 200 with one cookie; `/api/version` `{sha, migration_head}` — `sha == git rev-parse HEAD` and `migration_head == ledger head`; one production call (`GET /api/products`); one read-layer call (`GET /api/performance?code=<known>`). Any failure stops the run.
4. **deploy-prod** (SSH to `server.ecomprofits.io`, `/opt/skynet`, gated by `vars.DEPLOY_PROD_ENABLED`): identical steps with `SKYNET_ENV_B64`, reusing the **exact**

`API_IMAGE_DIGEST` and `WEB_IMAGE_DIGEST` that staging ran; pre-migrate `pg_dump` into `skynet_backups/pre-deploy/`; refuses if disk free < 20 GB.

5. **notify** → Telegram (build / staging / smoke / prod result, actor, sha, run link) — the ecomprofits message format.

Migrate-step guarantees: one transaction per file; checksum ledger `core.schema_migrations`; refuses a changed checksum; idempotent; forward-only and additive by rule (`docs/MIGRATIONS.md`); a destructive migration needs an ADR and a manual dispatch.

Rollback: `infra/deploy/rollback.sh <sha>` — reset the checkout to `<sha>`, pin/pull `skynet-api:main-<sha>` + `skynet-web:main-<sha>`, `up -d`; restore the pre-migrate dump if the ledger head is newer than the target. Drilled on staging before the first production deploy.

---

8. Auth, identity, permissions (S1 scope)

Skynet issues RS256 access + refresh tokens (`rs_access` 12 h, `rs_refresh` 30 d; HttpOnly, Secure, SameSite=Lax, exact-host cookie domain), refresh rotation with family revocation, logout via `token_version`; claims `{sub, email, name, ws, ws_slug, roles[], tv, typ, exp}`; roles as data: `core.permissions` (fixed catalogue, action-level), `core.roles` (six seeded system roles `admin, strategist, media_buyer, editor, researcher, viewer` + custom roles per workspace from S5), `core.workspace_member_roles` (a member may hold several roles; effective permissions = union of allows, no deny rules); every route gate checks a permission, never a role name; the JWT carries `roles[]` only and `/api/auth/me` returns the effective `permissions[]`; `roles[]` remains in JWTs for compatibility, membership display and legacy mapping only; Skynet UI and route gating use `/api/auth/me.permissions[]` as the canonical authorization surface. JWKS at `/api/auth/jwks`; service tokens (`typ=service`) for workos and internal jobs. Legacy static-ads users are imported and invited once; the ported Production API verifies the unified token (Bearer / `?token=` / cookie) — the `?token=` bridge is a documented S1 limitation (ADR-005) removed when asset URLs become signed. **The standalone scraper and console keep their own logins until their merge slices.** CSRF = origin check; CORS none. Contract detail: day-0 lock §4.1 (unchanged).

---

9. Resource protection on the production box

Current host facts: 48 cores, 250 GB RAM (197 GB available), 1.2 TB free, load ≈ 0.1; Supabase db limit 160 GB, pooler 16 GB.

Skynet is budgeted as a **bounded tenant** on this box: **≤ 12 cores, ≤ 32 GB RAM**, with hard limits and soft reservations. Limits are enforced in the host-side compose files with runtime-supported settings (`cpus`, `mem_limit`, `mem_reservation`, `pids_limit`, `restart`) — not Swarm-only `deploy.resources`.

SERVICE	CPUS	MEM_LIMIT	MEM_RESERVATION	PIDS_LIMIT	NOTES
api	6	12 GB	8 GB	512	uvicorn + job worker; bounded thread pools only
postgres	4	12 GB	8 GB	256	Skynet DB only
caddy	1	1 GB	256 MB	128	proxy only
backup	0.5	512 MB	128 MB	64	off-hours, `nice` + `ionice`
synthetic	0.5	512 MB	128 MB	64	curl only

Skynet Postgres tuning, conservative for a shared host: `shared_buffers = 3GB`, `effective_cache_size = 8GB`, `work_mem = 16MB`, `maintenance_work_mem = 512MB`, `max_connections = 80`, `max_wal_size = 4GB`.

S1 app-level concurrency caps: `MAX_CONCURRENT_IMAGE_JOBS = 4`, `MAX_CONCURRENT_STORYBOARD_JOBS = 2`, `MAX_CONCURRENT_EXPORT_JOBS = 4`; queue backpressure instead of unbounded threads or per-request fan-out.

Host-protection rules:

- Skynet never runs Playwright, scraping, browser automation or proxy workloads on this box in S1.
- Backups run with lowered CPU and I/O priority (`nice` + `ionice`) and never overlap deploy migrations.
- Every Skynet service uses `restart: unless-stopped`; an OOM kill restarts that container only — never a compose-wide recovery.
- `mem_reservation` stays below `mem_limit` so the kernel has reclaim room before hard kills; `pids_limit` prevents runaway forks.
- Production deploys abort if host free disk < 20 GB; alerts at < 100 GB.

Alert thresholds: host load > 24 for 10 min · available RAM < 32 GB · swap > 2 GB · disk free < 100 GB · Skynet Postgres restart / OOM · backup duration over its window · two consecutive synthetic failures.

S2 review trigger: before any scraper or runtime workload lands on this box, this table is re-issued with separate limits for those services and a fresh pressure test.

10. Backup and restore

- Nightly `pg_dump -Fc` of Skynet's Postgres by the `backup` sidecar into the host path `/var/lib/skynet-backups/daily/skynet-<date>.dump` (own folder, mode 640), retention 20 days locally; a Plesk Scheduled Task syncs that folder to the Hetzner Storage Box under its own prefix (`skynet/`), independent of the ecomprofits sweep. R2 object protection is handled separately and must be confirmed explicitly (bucket versioning / lifecycle policy is an infra input, not assumed here).
- The backup path is created and owned before the first deploy (`/var/lib/skynet-backups/{daily,pre-deploy}`), and the Scheduled Task runs as `root` or an account explicitly granted read access to that tree — part of the production-box prep card (A0).
- Pre-deploy dumps in `/var/lib/skynet-backups/pre-deploy/` (never auto-pruned; alert at 20 GB).
- Restore drill (`docs/runbooks/RESTORE.md`): restore last night's dump into a scratch container, compare row counts per table, time it; repeated by a second person. Done before the first production deploy and monthly after.

11. Observability and operations

- `/api/healthz` (process + ledger readable), `/api/version` (sha + migration head).
- `synthetic` every 5 min: `/` , `/api/healthz` , `/api/auth/jwks` , the workos bridge `/health` , one read-layer query, and backup freshness (latest daily dump exists and is newer than 24 h); two consecutive failures → Discord + Telegram; history in `core.synthetic_checks` ; `/status` page behind SSO.
- Telegram deploy messages (ecomprofits format); Discord for spend-guard warnings.
- Runbooks: local dev, deploy, rollback, restore, alerts, legacy hosts (what still runs on `server.rescale.media` and Railway, and who to call).
- Working rules: compose is the sole owner of containers; config only via CI secrets; no GUI edits; no prune; every incident gets a note in `docs/runbooks/INCIDENTS.md` .

12. S1 exit gate (what must be true before S2 starts)

1. `app.rescale.media` serves the unified app from `server.ecomprofits.io` ; one login for Skynet itself.
2. Production fully native at parity.
3. Creative-code loop proven with one real ad.
4. Bindings + spend guard + minimal admin.

5. Export package attached to the existing ClickUp Launch card via the workos bridge.
6. Intelligence and Studio entries link to the standalones.
7. Staging + CI/CD + backups + rollback drill + monitoring.

---

13. Later slices — what this design already makes room for

- **S2 Intelligence merge:** scraper engine + transcriber become services in `compose.prod.yml` with their own limits; `meta_ads` schema in Skynet's Postgres (the A4 spike decides the data path: PostgREST over unified PG vs a pg adapter); native Intelligence pages; Reddit VOC (A18); the standalone scraper retired.
- **S3 Studio merge + strategy layer:** agent runtime on Postgres (`studio` schema), night-shift jobs on the stack, Claude subscription binding internal-only, Bian's coexistence agreement first; coverage map, hook session, instinct loop, validated memory.
- **S4 Workspace cutover:** native boards one by one with the ClickUp mirror as rollback; workos moves off Railway into the stack; credentials discontinued; Railway off.
- **S5 SaaS hardening:** RLS per schema, per-workspace keys and quotas, audit log, billing, onboarding, security review. Everything above is built with `workspace_id` so this is additive.

Bridge track B (temporary coexistence). While S1 runs, the standalone scraper and RescaleOS may gain temporary features for Mathew (LFS extraction package, `/lfs`, `/evidence-layer`, PubMed adapter/MCP). They live outside the unified runtime, are tracked as track-B cards in the delivery plan, and are retired by A19/A20/B0. Model boundary: RescaleOS sessions choose Claude models only (`setModel`); non-Claude models reach the loop through the scraper's OpenRouter tiers or an MCP, never inside the agent runtime.

14. Risks specific to this design

RISK	MITIGATION
Skynet load affects the production Supabase	§9 limits; separate Postgres; no scraping on the box in S1; synthetic box-pressure alerts; rollback under 15 min
Two boxes, two deploy targets	One workflow, two SSH steps, identical compose; smoke gate before prod
Standalone apps drift while merges wait	Bridge track B is the only allowed temporary standalone change set; sunset is enforced by A19/A20/B0 and merge slices still start from a fresh code scout
Cross-box confusion for users	The shell labels links clearly ("opens the scraper in a new tab"); comms pack
Workos on Railway is a remote dependency for naming/launch	Bridge calls are idempotent and retried; the loop test covers it; S4 brings workos into the stack