

Ship & operate · Runbooks

Runbook: move production onto the API functions

🕒 3 min read · Updated 25 September 2026 · Verified against [4391ce0](#) on 2026-09-26

Moving production onto the per-context API functions, gated on the performance baseline and the data canary.

<https://docs.gurbanisoul.com/process/runbooks/services-production/>

Staging answers every bounded context with its own Vercel function inside the web project (ADR-0011). Steps 0 and 1 below are merged on integration and ship with the next release: production answers /api with all, Git previews are off, and deploy-production checks every function and the golden contract on the unaliased deployment before promoting it. Render keeps deploying as the rollback target until step 4. Every production deployment already carries the six functions, so moving production is a routing change, released through the normal pipeline, with no new hosting.

Prerequisites

- A green staging deploy with every context routed (deploy-staging → verify: every function ready at the commit, every probe ok, the golden contract PASS through the gateway).
- A spending limit on the Vercel team (Settings → Billing → Spend Management) so usage-based function pricing can never surprise the plan.

Steps

0. **Previews.** The committed frontend/vercel.json names no functions (the deploy adds them with the generated files), so Vercel's Git preview builds carry none and their /api follows the production rule. Once production is on vercel, that rule points at a function a Git preview does not have: switch Git previews off (git.deploymentEnabled: false in frontend/vercel.json) in the same release, and review web changes on staging.
1. **Move production to the functions, whole API first.** In gateway/routes.json set production.api_platform to vercel and drop production.api, keeping production.services empty, so every /api path is answered by all (the single API's role, same database, same code). Run `python3 tools/gen_gateway.py` and release it. Verify: `curl -sI https://gurbanisoul.com/api/meta | grep -i x-service` shows all; `make verify-prod`; `make canary`; `python3 scripts/ci/verify_functions.py https://gurbanisoul.com --commit <sha> - -env production`.

2. **Hold the budget.** From the same place the baseline was taken: `python3 tools/perf_baseline.py --base https://gurbanisoul.com --from "<same vantage>" --compare docs/perf/baseline-2026-09.json` must PASS (p95 within +10 % per context; cold starts excluded by its warm-up).
3. **Split the contexts.** Add the contexts to `production.services`, regenerate, release, and verify as in step 1: each context's probe shows its own X-Service (e.g. `knowledge`), and step 2's budget holds per context. *Decision 2026-09-26 (owner)*: under the one-version policy every release is also an App Store update, so all five contexts move in **one** release (after the app is live) instead of five. Each is still gated on its own: `verify_functions --env production` proves every function ready at the commit with exactly its contexts, the golden contract runs through the new rules before promotion, and one context rolls back by removing it (below); the instant lever is `vercel rollback` to the previous deployment.
4. **Retire Render** after a full release cycle on the functions with the canary green: suspend and then delete the `sggs-knowledge-base` Render service, remove `RENDER_DEPLOY_HOOK` and the `deploy-api / verify-api` jobs from `deploy-production.yml`, and point the data canary's and the uptime check's API origin at the site (the functions have no separate origin).

Roll back

- One context: remove it from `production.services`, regenerate, release — all answers it again.
- The whole API: set `production.api_platform` back to `render` with its `api` origin, regenerate, release. The Render service keeps running until step 4, so this is always available until then.