

Ship & operate · Engineering handbook

Delivery, versioning & release

🕒 4 min read · Updated 27 September 2026 · Verified against [f25ab97](#) on 2026-09-25

How work reaches users: the versioning convention, the integration to staging to production loop, delivery tooling and the iOS release.

<https://docs.gurbanisoul.com/engineering/delivery/>

How work reaches users. Read before shipping.

Versioning convention (important)

- **APP_VERSION in webapp/serve.py is the source of truth** for the running build. Bump it on every search-logic/UI release.
- **One number, everywhere.** The web footer, /api/meta, /api/health, the iOS About screen (MARKETING_VERSION) and the App Store all report the same XYZ; they are **never allowed to differ** (check_versions.py gates this repository's 5 strings; the app repository requires its MARKETING_VERSION to equal the platform release it vendors, and its ledger never to lead it). **Every release is re-archived at the same version** and uploaded as iOS X.Y.Z (1) from the release tag vX.Y.Z, even a web-only change — so the App Store number always equals the live site. The **build number restarts at 1 per version** (only a re-upload of the *same* version increments it). A release is *complete* only when scripts/release/check_release_complete.py X.Y.Z passes (prod web+API at the tag commit **and** an appstore ledger entry of XYZ from that commit). Never ship a one-sided hotfix — patch-bump both surfaces.
- The DB's own build version is returned separately as db_version and is **not** shown in the UI. This decoupling is intentional: the data releases on its own cadence (see below).
- Keep README.md, CHANGELOG.md, and MASTER-INDEX.md in step when you bump (scripts/release/bump.py does it).
- **The dataset versions separately.** It is built and versioned in Algorythmos-AI/sggs-data: every build stamps the database's meta.version (served as db_version) with sggs-data's DATASET_VERSION, and dataset.lock.json records it as dataset_version. The database pinned today was built on 2026-09-18, before that rule, so /api/meta reports db_version 1.1.4 while the lock says 1.0.0; the next rebuild makes them one number. This repository serves exactly the object dataset.lock.json pins; a new dataset arrives as a reviewed lock bump, never with an unrelated app change.

Delivery SOP — how work reaches users (read before shipping)

Environments: integration = trunk → auto-deploys **staging** (sggs-staging.vercel.app, SSO-protected; the API runs as Vercel functions inside it). main = **production** (web on the canonical domain **gurbanisoul.com**; www and the legacy *.vercel.app alias 308 → apex; the API runs as Vercel functions inside it from the release after 1.3.9, with the Render single API ...onrender.com kept deploying as the rollback target). Both deploy **only** through CI (deploy-staging.yml / deploy-production.yml); the platforms' own git auto-deploys are OFF and Render auto-deploy is OFF. Never deploy by hand except a documented pipeline-outage hotfix.

The loop (use the tooling below; do not improvise it):

1. **Ship:** branch from integration (fix/..., feat/...), run local gates, open a PR into integration, watch CI green. Corpus/DB change? It happens in sggs-data¹ first (scripture proofs); here it arrives as a reviewed dataset.lock.json bump. You cannot merge — hand the user the exact --admin command (one line, no comments).
2. The user merges → the push auto-deploys **staging**. Review on sggs-staging.vercel.app (open logged in to Vercel).
3. **Release:** preflight → PR integration→main as a **merge commit** (never squash — main must stay a descendant of integration) → the merge runs the gated production deploy → verify with make verify-prod and confirm the vX.Y.Z tag.

Invariants: a deploy is verified by the running **commit** (/api/health.commit), not just the version. The release tag is cut only after a verified deploy. Secrets live in GitHub Environments (production/staging) — never type or echo a value; if one is pasted into chat, treat it as leaked and have the user rotate it. Full detail: docs/process/runbooks/deploy.md, docs/process/branching.md, ADR-0005.

Delivery tooling

Tested scripts that encode the gotchas learned shipping v1.1.x. Prefer them to ad-hoc commands:

Step	Command	Script
Watch a PR's checks to completion	make pr-checks PR=<n>	scripts/ci/wait_pr_checks.sh
Release preflight (trunk green, versions unified, CHANGELOG section)	make release-preflight	scripts/release/release_preflight.sh
Watch the production deploy gate by gate	make watch-deploy [SHA=<sha>]	scripts/release/watch_deploy.sh
Prove what production serves (commit, health, Ang 712 heading, web deployment)	make verify-prod [ARGS="--commit <sha> --version X.Y.Z"]	scripts/ops/verify_prod.py

Step	Command	Script
Install / check the pinned database	<code>make dataset·make dataset-check</code>	<code>scripts/data/fetch_dataset.py</code>

iOS app (Gurbani Soul)

The app lives in [Algorythmos-AI/gurbani-soul-ios](#)². It builds against this repository’s golden contract, vendored at a pinned platform release (`vendor.lock.json` there), and its `MARKETING_VERSION` must equal that release’s `APP_VERSION` — the one-number policy, enforced by its CI. After a release is tagged `vX.Y.Z` here, that repository runs `make vendor-sync-platform REF=vX.Y.Z`, sets `MARKETING_VERSION`, and uploads `X.Y.Z (1)` with `make testflight ... CHANNEL=appstore` (the archive refuses an App Store upload whose contract was not vendored from `vX.Y.Z`). Its ledger records the platform commit each binary was built against; `scripts/release/check_release_complete.py X.Y.Z` reads it from there to prove web, API and the binary report the same number from the same commit. TestFlight/App Store runbooks, plans and the store listing live in that repository.

Links

1. <https://github.com/Algorythmos-AI/sggs-data>
2. <https://github.com/Algorythmos-AI/gurbani-soul-ios>