Ship & operate · Runbooks

Runbook: the Hukamnama each morning

9 min read · Updated 22 November 2026 · Verified against `a16ec9a` on 2026-11-22

> *The owner's decisions for Hukamnama · Sri Harmandir Sahib: reading /health, what to do on a held day, approve, hold and resend, correcting a day after it is published, and the deploy freeze.*

https://docs.gurbanisoul.com/process/runbooks/hukamnama/

What the owner does, and when, for the daily Hukamnama · Sri Harmandir Sahib. How the service works is **the Hukamnama service**; the Render service, its secrets, backup and push key are **the service runbook**.

There are **two services, each with its own state** (checked 2026-11-22):

	Service	`/health`	Who reads it
Production	`sggs-hukamnama-prod`	`https://api.gurbanisoul.com/health` (the Worker passes it on)	the website, the public API; sending is off
Staging	`sggs-hukamnama-staging`	`https://api-staging.gurbanisoul.com/health`	the staging website, TestFlight builds (sending is on)

Each decides every morning on its own, so a held day is held on both, and every owner decision below is made **on each service**: staging first, then production.

The rule behind every step: **late is acceptable, wrong is not.** A held day is shown to readers as `pending`; nothing unproved is ever served, pushed or written.

How you hear about a problem

- **Your phone (ntfy), first.** When `HUKAMNAMA_ALERT_NTFY_URL` is set, every new service alert is posted to your phone the moment it is raised: a hold, or a morning with nothing published, as **urgent**; a late or changed page as high. Each says what readers see and what to run. Setup is in the **service runbook**. GitHub's scheduled checks can run hours late (13 Nov 2026's 07:00 check ran at 12:34), so do not wait for the issue.
- **The sibling alert.** With `HUKAMNAMA_PEER_URL` set, staging and production read each other's `/health` every 5 minutes, 05:15–14:15 IST, and send `peer_down` to the phone when the other is unreachable, reports `ok=false` or has stopped polling (3 failed checks in a row, once a day). This is the timely "the service is down" signal.

- **The dead-man issue.** `hukamnama-deadman.yml` reads both services' `/health`, scheduled at 05:37, 07:17 and 14:23 IST (in practice 6-9 h later: GitHub's cron). When the service is unreachable, the poller is stale (no poll for 15 minutes inside 05:00–14:00 IST), nothing is published by 07:00 IST, or the service raised any other alert today, it opens one issue titled "Hukamnama service: problem on (Amritsar)" (with `[production]` for production), and later runs comment on it. It also posts to your phone (the same topic), which covers the service being down. It never restarts or redeploys anything. Each environment's job is skipped while its URL variable (`HUKAMNAMA_STAGING_URL`, `HUKAMNAMA_PROD_URL`) is unset.
- **The morning check.** `hukamnama-morning.yml` reads each website page, the JSON feed and the embed at 07:15 and 10:00 IST and compares the Gurmukhi each shows with the API's `range_sha256` for the date shown; a mismatch opens one issue per Amritsar date. Each environment's job runs only when its site and API variables are set.
- **`/health`.** The service's own alerts are always listed there, phone or not.

```
curl -s https://api.gurbanisoul.com/health | python3 -m json.tool          # production
curl -s https://api-staging.gurbanisoul.com/health | python3 -m json.tool  # staging
```

Field	Read it as
`commit`	the running build; equals the deployed SHA after `hukamnama-deploy`
`amritsar_date`, `status_today`, `decision_today`	today in Amritsar, `published` or `pending`, and the stored decision (`null` until one is made)
`last_tick`, `last_tick_action`	the last poll and what it did (`waiting`, `published`, `held`, `watching`, `idle`, …)
`alerts_today`	today's alerts, one per kind (the list is in `services/hukamnama/README.md`, "Polling and alerts")
`paused`	`true` while the owner's `pause` is on: every day reads as `pending` and nothing is sent
`scripture_sha256`	the fingerprint of the corpus the service reads
`push`	`sending`, device counts by mode, and today's send states

The owner's tools

Run on the service: `render ssh <service-id>`, then from the service's working directory:

```
python -m hukamnama show    <date>                                  # the decision and
what is served
python -m hukamnama approve <date> --first-line <id> --last-line <id> --by <you> --reason "
<why>"
python -m hukamnama hold    <date> --by <you> --reason "<why>"
python -m hukamnama resend  <date> --by <you> --reason "<why>"
python -m hukamnama correct <date> --first-line <id> --last-line <id> --by <you> --reason "
<why>"
python -m hukamnama pause   --by <you> --reason "<why>"               # and resume, the
same way
```

`<date>` is the **Amritsar** date, `YYYY-MM-DD`. Every one of these except `show` writes the `audit` table with your name and reason. `approve` and `hold` overwrite the day's decision; an automatic publish never replaces an owner decision afterwards. `correct` works only on a published day (use `approve` for a held or undecided one). A change made this way reaches the snapshot files within one poll interval.

Tool	Effect for readers	Effect on push
`approve`	the range is published at that moment, with its `range_sha256`; the snapshot file is written (or rewritten)	`asap` devices get it then. Approved after its Amritsar midnight, it is still pushed (with its own date) until **05:00 IST**, unless the new day is already published; later than that it is never pushed
`hold`	the day reads as `pending`; its snapshot file is deleted	nothing new is queued; a notification already delivered stays on the phone
`correct`	the published range is replaced; the day keeps its first publication time and gains `corrected_at`; the snapshot is rewritten	nothing is queued by itself: follow it with `resend` the same Amritsar day
`pause`	every day reads as `pending` everywhere and the snapshot drops every day file; `resume` restores them	nothing is sent while paused; queued sends wait
`resend`	none	re-queues every device that already had (or failed) that day's push, and clears a tripped circuit breaker (audited); the same collapse id replaces the first notification on the phone

`resend` does not check the date itself: use it on the same Amritsar day, or a reader's lock screen gets an old day's notification.

A held day

`held_no_match` (the page does not equal any corpus run) or `held_veto` (sgpc.net shows a different Ang).

1. Open hs.sgpc.net and the sgpc.net marquee and note the Ang and the opening words.
2. Find the shabad in the corpus by that Ang (the Knowledge Base reader, or `/api/ang/<N>`, which lists each line with its `id`). A `held_no_match` alert often already names the nearest run of line ids and the words that differ.

3. Decide whether the difference is the source's or ours. The corpus is proven against the printed Bir character for character, so a one-word difference is normally SGPC's spelling; compare ShabadOS as well. **Never change the corpus to match SGPC**: a real fault in the corpus is flagged for review in sggs-data, never fixed here.
4. If the shabad is certain, `approve` its range as the service would have served it: when the shabad opens its composition, the first line is the first of its printed heading lines in the corpus (the lines with its `comp_id` just before the first verse); otherwise the first verse, or a Vaar unit's salok label line when the page prints it. The last line is where the shabad ends (its numbered marker, such as `॥ 2॥3॥`, even when the composition carries on). Put the reason in words someone can audit later (for example "SGPC page variant in one word").
5. If you are not certain, leave it held. Readers see `pending`, the website and the app show the most recent day under its own date, and nothing is wrong.

For a `held_veto`, the alert names the Ang or the first opening word that differs; check which page is right first, then approve that range, or leave the day held. With `HUKAMNAMA_APPROVE_SECRET` set, the alert names that range and has **Approve** and **Review** buttons that publish exactly it (re-checked against the corpus at the tap); see the service runbook.

After a day is published

`changed_after_publish` or `veto_after_publish` means a source changed within 2 hours of an automatic publish. The service never applies the change.

- **SGPC corrected its page** and the new range is right: `correct` the day to the new range (the snapshot is rewritten and the change is recorded beside the day), then, the same Amritsar day, `resend` so phones get the corrected day in place of the first notification.
- **The published day is wrong and the right one is not yet clear:** `hold` it. Readers see `pending` until you approve the right range. When nothing at all should be shown, `pause` stops every day and every send until `resume`, without a deploy.
- **The source flickered and the published range is right:** do nothing; the alert stays on that day's `/health` and the dead-man issue.

Whatever you change on a published day (`correct`, `hold`, `pause`), the service clears the website's CDN copy itself when the purge is configured. A `cdn_purge_failed` alert means it could not: clear it by hand with `vercel cache dangerously-delete --tag hukamnama` on the web project.

Close the day's dead-man issue with a line saying what you decided.

Other alerts

Alert	Action
`late`, `not_published`	Open hs.sgpc.net. If SGPC has not published, there is nothing to do. If it has, look at `last_tick_action` and the other alerts.
`hs_layout_changed`, `hs_parse_failing`	SGPC changed its page. Keep the day's fetch archive (`/var/data/archive/<date>/`), fix the parser in a pull request with that page as a fixture. Approve the day by hand meanwhile.
`push_breaker`, `push_auth_failed`, `push_payload_too_large`, `device_flood`	The service runbook, Push.
The poller is stale or the service unreachable	Render → service → Logs and Events; restart it. Its state is on the disk.

Deploys and the freeze

- **Staging** changes only through `hukamnama-deploy.yml`: on a push to `integration` that touches `services/hukamnama/**`, `dataset.lock.json` or the workflow, or by dispatch. Render auto-deploy is off.
- **Production** changes only through `hukamnama-deploy-production.yml`, which you dispatch from `main` with a release tag after the release (**the service runbook**). A push never deploys it, and the website release (`deploy-production.yml`) does not either, so after a release the service keeps the previous tag until you dispatch it.
- **Freeze: 23:00–04:30 UTC (04:30–10:00 IST)**, the hours SGPC publishes and the poller decides. A deploy in that window fails the workflow's freeze step (both workflows); dispatch it again after 10:00 IST. The `override_freeze` input exists for an emergency fix only. The service runs one instance with a disk, so a deploy restarts it for a few seconds.
- **A dataset bump** (`dataset.lock.json`) also deploys the service. If the corpus fingerprint changed, the service proves every published day in the new corpus at start and, when all are byte-identical there, re-pins them and carries on. It refuses to start only when a day does not prove, and names it: fix that day with `correct` or `hold`, confirm with `python -m hukamnama repin`, and restart. Both deploy workflows run the same proof first (`scripts/ci/hukamnama_preflight.py` reads the running service's published days and proves each in the commit's database), and the production one proves them again once the new build answers. Never delete the state file to get past it: it holds every decision and the audit trail.