Ship & operate · Runbooks

Runbook: Hukamnama service

14 min read · Updated 22 November 2026 · Verified against `14c89ad` on 2026-11-12

The Hukamnama service on Render: the staging soak with sending off, its one-time setup, the morning dead-man check, the fetch archive, and what to do when a morning goes wrong.

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

The Hukamnama service (`services/hukamnama/`) locates the daily Hukamnama from Sri Harmandir Sahib inside the pinned corpus. Before any notification is sent it runs a **14-day soak on staging with sending off**: it polls SGPC every morning, decides, and keeps every page it saw. The soak measures the real publish minute and the hs.sgpc.net ↔ sgpc.net lag, and checks that at least 13 of 14 mornings resolve automatically and correctly against a manual check. Sending is now on for staging: its `/health` reported `push.sending: true` on 2026-11-12, and a push reached a TestFlight device on 2026-11-11. Staging reaches TestFlight builds only (**Push**).

The owner's daily decisions (a held day, approve, hold, resend, corrections) are in **the Hukamnama each morning**; how the service works is **the Hukamnama service**.

The service on Render

Setting	Value
Name	`sggs-hukamnama-staging`
Type, runtime	Web service, Docker
Repository, branch	`Algorythmos-AI/sggs-platform`, `integration`
Dockerfile, context	`services/hukamnama/Dockerfile`, repository root (`.`)
Region	Singapore (closest to Amritsar and to the owner)
Plan	**Starter** (always on; a free instance sleeps and would miss the morning)
Disk	1 GB mounted at `/var/data` (state SQLite + fetch archive)
Health check path	`/health`
Auto-deploy	**Off**. Deploys come only from `hukamnama-deploy.yml`.
Environment	The image sets `HUKAMNAMA_DB`, `HUKAMNAMA_STATE`, `HUKAMNAMA_ARCHIVE`. For the R2 backup, set the four values in State backup; without them the service runs unbacked.

A service with a disk runs **one instance** and restarts on deploy (a few seconds of downtime). The freeze in `hukamnama-deploy.yml` keeps deploys out of 23:00–04:30 UTC (04:30–10:00 IST).

Cost: Starter plus the 1 GB disk, about US$7–8 a month.

One-time setup (owner)

1. `render login` (browser), then `render workspace set` to the Algorythmos workspace.
2. Create the service with the settings above (dashboard, or the CLI command in the PR that added this runbook).
3. Copy the service's **Deploy Hook** (Settings → Deploy Hook) into the GitHub environment `staging` as `RENDER_DEPLOY_HOOK_HUKAMNAMA_STAGING`. Never paste it into chat or a file.
4. Set the repository variable to the service URL: `gh variable set HUKAMNAMA_STAGING_URL --repo Algorythmos-AI/sggs-platform --body https://sggs-hukamnama-staging.onrender.com`
5. Dispatch `hukamnama-deploy` (Actions → hukamnama-deploy → Run) outside the freeze. It finishes when `/health.commit` equals the deployed SHA.
6. `api-staging.gurbanisoul.com` is a custom domain on the service (Render → Custom Domains) with a Cloudflare CNAME to `sggs-hukamnama-staging.onrender.com`, **DNS only** (grey cloud) so Render issues and renews the certificate. TestFlight builds read the Hukamnama from that name. Done 2026-11-10.
7. Turn on the state backup (next section).

State backup: Litestream to R2

The decision and audit state (`/var/data/hukamnama-state.sqlite`) is replicated continuously to the R2 bucket `sggs-hukamnama-staging` (account *Company-Domains*, Asia-Pacific) by **Litestream**[1], pinned in the Dockerfile by version and sha256. The scripture DB is not backed up (it comes from the dataset pin), and neither is the fetch archive.

`entrypoint.sh` runs the service under `litestream replicate` (`litestream.yml`) when R2 credentials are present, and plainly when they are not. Credentials without the other two values stop the service at start, so a typo cannot leave it running unbacked.

Render → service → Environment (values typed there only; never in chat, a file or a PR):

Variable	Value
`LITESTREAM_ACCESS_KEY_ID`	Access Key ID of the R2 Account API token `hukamnama-staging-litestream` (Object Read & Write, this bucket only)
`LITESTREAM_SECRET_ACCESS_KEY`	its Secret Access Key

Variable	Value
`HUKAMNAMA_R2_ENDPOINT`	the token page's S3 endpoint, `https://<account id>.r2.cloudflarestorage.com`
`HUKAMNAMA_R2_BUCKET`	`sggs-hukamnama-staging`

Use **Save only**, then deploy with `hukamnama-deploy` outside the freeze. After the deploy, the service log shows Litestream's `replicating to` line, and objects appear under `hukamnama-state/` in the bucket within a minute.

Restore. A missing state file is restored from R2 before the service starts (`restore-if-db-not-exists`); an existing file is never overwritten. To restore by hand into a scratch path and compare: `render ssh <service-id>` → `litestream restore -config /etc/litestream.yml -integrity-check quick -o /tmp/restored.sqlite /var/data/hukamnama-state.sqlite`. A rotated or revoked token stops replication but not the service; the log shows the error.

Public snapshot: the bucket the edge Worker reads

Not set up yet (staging or production). The service can write each published day as a file to an R2 bucket, which the edge Worker (`services/hukamnama-edge`) serves as the public API. To turn it on for one service, the owner:

1. Creates the bucket (`hukamnama-public-staging` for staging, `hukamnama-public` for production; the names in the Worker's `wrangler.toml`).
2. Creates an R2 API token scoped to **that bucket only**, Object Read & Write. It is a different token from the backup's.
3. Adds three values to the service's environment on Render: `HUKAMNAMA_PUBLIC_BUCKET`, `HUKAMNAMA_PUBLIC_ACCESS_KEY_ID`, `HUKAMNAMA_PUBLIC_SECRET_ACCESS_KEY` (`HUKAMNAMA_R2_ENDPOINT` is already there for the backup). Use the token's S3 Access Key ID and Secret, as for Litestream.

With the bucket name set and a key missing, the service refuses to start and names what is missing. After the next deploy the first poll writes every published day; `public snapshot: wrote N` in the log confirms it. A failed write is logged (`public snapshot sync failed`) and tried again after the next poll; it never stops polling or sending. `HUKAMNAMA_MEANING=on` adds the English translation layer to the files, and removing it takes the layer out on the next poll.

Behind the edge: whose address a request is limited by

The device routes and every read are rate-limited per client address (writes 20 a minute, reads 120 a minute, each with its own budget; a connection idle for 15 seconds is dropped). Which address that is depends on two settings:

Setting	Effect
neither (staging today)	the first `X-Forwarded-For` entry, as Render's proxy sets it

Setting	Effect
`HUKAMNAMA_ORIGIN_SECRET`	only a request carrying the same value in `X-Origin-Auth` may name its address (`X-Client-IP`, set by the edge Worker from what Cloudflare saw); any other request is limited by its socket's own address, so a caller cannot pick a bucket by sending a header
`HUKAMNAMA_ORIGIN_REQUIRED=on` as well	`/v1/*` answers `403` to anything that did not come through the edge; `/health` stays open for Render and the dead-man check

Set the same secret on the Worker (`npx wrangler secret put ORIGIN_SECRET`) and on Render, the Worker first. Turn `HUKAMNAMA_ORIGIN_REQUIRED` on only when every caller goes through the edge: the iOS app and the website's page function call this host directly on staging today. Required without a secret refuses everyone.

Every morning (automatic)

- The poller runs every minute 05:00–09:00 IST, then every 10 minutes until 14:00 IST.
- The two services watch each other (`HUKAMNAMA_PEER_URL`, `peer.py`): every 5 minutes from 05:15 to 14:15 IST each reads the other's `/health`, and after 3 failed checks in a row it sends a `peer_down` phone alert, once a day. This is the timely dead-man check.
- `hukamnama-deadman.yml` is the backstop, for staging and production: it checks `/health` at 05:37, 07:17 and 14:23 IST *as scheduled*. GitHub's cron ran it 6-9 h late every day in November 2026, so do not rely on its times. On a problem it opens one issue per Amritsar date and service (later runs comment on it). It never restarts anything.
- Each distinct page fetched is kept at `/var/data/archive/<amritsar date>/` (gzip) with an `index.ndjson` of every fetch. Day folders older than 60 days are pruned.

The morning check (automatic)

`hukamnama-morning.yml` runs at 07:15 and 10:00 IST. From outside it reads what a visitor reads (the page in English, Punjabi and French, the JSON feed's newest entry, the embed), hashes the Gurmukhi lines each one shows and compares that with the `range_sha256` the API published for the date shown. Once the API has published today's Hukamnama, every surface must show today's. A mismatch opens one issue per Amritsar date, and later runs comment on it.

It is silent until its variables are set (repository variables):

Environment	Variables
staging	`HUKAMNAMA_STAGING_SITE_URL` (the staging site's origin); the API is `HUKAMNAMA_STAGING_URL`
production	`HUKAMNAMA_PROD_SITE_URL` and `HUKAMNAMA_PROD_API_URL`

Run it by hand: `python3 scripts/ci/hukamnama_morning.py <site origin> <api origin>`.

Production deploys (owner, by hand)

Production never deploys on a push. After a release, dispatch `hukamnama-deploy-production` with the release tag:

```
gh workflow run hukamnama-deploy-production.yml --repo Algorythmos-AI/sggs-platform --
ref main -f tag=v1.7.0
```

Write the real tag in place of `v1.7.0` (`gh release list` shows the latest). `--ref main` is required: the `production` environment only accepts runs from `main`, so a run dispatched from `integration` is rejected before its first step. That also means the workflow can be used only once a release has carried it to `main`.

It refuses a tag that is not `vX.Y.Z` on `main`, runs the service's tests on the tag, and then, before asking Render for anything, proves every day the running service has published against the database the tag pins. If a dataset bump moved or changed a published day's lines the run stops with the day named: correct or re-approve that day on the running service first. For the very first deploy, when there is no service to read, add `-f first_deploy=true`.

Setup, once: the Render service with auto-deploy off, its deploy hook as the secret `RENDER_DEPLOY_HOOK_HUKAMNAMA_PROD` in the `production` environment, and the repository variable `HUKAMNAMA_PROD_URL`. If the service refuses direct calls (`HUKAMNAMA_ORIGIN_REQUIRED=on`), also set `HUKAMNAMA_PROD_READ_URL` to the public API origin, where its published days can be read.

The staging deploy runs the same proof on every push.

Looking at the soak

```
curl -s https://sggs-hukamnama-staging.onrender.com/health | python3 -m json.tool
curl -s https://sggs-hukamnama-staging.onrender.com/v1/hukamnama?limit=14 | python3 -m
json.tool
render ssh <service-id>              # then: ls /var/data/archive; python -m hukamnama show
2026-11-10
```

Copy a day's archive off the box for the replay tests with `render ssh <service-id> -- tar -C /var/data/archive -czf - 2026-11-10 > hk-2026-11-10.tgz`.

Phone alerts (ntfy)

A held day only reaches readers once you approve it, so every new alert is posted to your phone the moment the service raises it, through **ntfy**[2] (free app, no account). A hold, a closed window with nothing published, a stopped push or a failed dead-man check is **urgent** (sounds through the lock screen and Focus, if you allow it); a late or changed page is **high**. Each alert says what happened, what readers see, and the command to run; tapping it opens this runbook. It carries the date, Ang and reason only, never a device token or reader data.

Setup, once per environment:

1. Make a topic name that nobody can guess (it is the secret: whoever knows it can read the alerts): `python3 -c "import secrets; print('hk-' + secrets.token_urlsafe(18))"`.
2. In the ntfy iPhone app: `+` → that topic on `ntfy.sh` → subscribe. In iOS Settings → ntfy, allow notifications; to let urgent alerts through Sleep, add ntfy to the Focus's allowed apps.
3. Render → the service → Environment: `HUKAMNAMA_ALERT_NTFY_URL = https://ntfy.sh/<topic>` and `HUKAMNAMA_ENV = staging` (or `production`). A malformed value stops the service from starting, as a half-configured backup does.
4. GitHub → Settings → Secrets → Actions: the same URL as `HUKAMNAMA_ALERT_NTFY_URL`, for the dead-man check (it alerts when the service itself is down and cannot post).
5. Test: `printf 'test alert' | HUKAMNAMA_ALERT_NTFY_URL=https://ntfy.sh/<topic> HUKAMNAMA_ENV=staging PYTHONPATH=services/hukamnama python3 -m hukamnama.alerts 2026-11-16 deadman` — your phone should ring within seconds.

Unset, the service runs as before, without phone alerts; the dead-man issue stays the record.

One-tap approve from the alert (held_veto)

A `held_veto` day is one where hs.sgpc.net's page already equals an exact run of corpus lines and only sgpc.net's home-page marquee disagreed (13, 20 and 21 Nov 2026 were held because of how sgpc.net spelled the weekday). With `HUKAMNAMA_APPROVE_SECRET` set on the service, its phone alert names that run (lines, Ang, line count) and carries two buttons:

- **Approve**: an ntfy `http` action that POSTs the signed request to `<service>/owner/approve`.
- **Review**: opens the same request as a page showing the corpus opening line and one "Approve and publish" button (use it where the ntfy app does not run `http` actions).

At the tap the service re-proves everything: the HMAC signature (over the day, the range, its `range_sha256` and the corpus fingerprint), that the day is still held by `auto`, that the range is the one stored with the hold, and that the range rebuilt from the corpus has the same `range_sha256`. Only then does it publish, audited as `owner (phone alert)`, and the push follows. A `held_no_match` day never gets a button, and nor does a day the owner already decided, a day older than 2 days, or a range from before a dataset bump. Use the shell for those.

Setup (owner, once per service): Render → Environment → add `HUKAMNAMA_APPROVE_SECRET` with a long random value (for example the output of `openssl rand -hex 32`, pasted straight into the dashboard, never into a chat or a file).

Choose **Save only**, then deploy. The buttons call the service's own Render URL (`RENDER_EXTERNAL_URL`), not the edge.

When a morning goes wrong

Symptom	Action
Issue: the poller is stale / unreachable	Render dashboard → service → Logs and Events. Restart the service; state is on the disk.
Issue: nothing by 07:00 IST	Open hs.sgpc.net. If SGPC is late, nothing to do. If it is up, look at `/health` alerts.
`held_no_match` / `held_veto`	Check the day by hand against the corpus, then `python -m hukamnama approve …` or `hold …` over `render ssh` (both are audited). A `held_no_match` alert names the nearest corpus run and the differing words when one run is that close (≤3 words). Confirm the difference is the source's, not ours (the corpus is PDF-proven; compare ShabadOS too), then approve that range, adding the heading line before it if the page printed one: `approve <date> --first-line <first> --last-line <last> --by <you> --reason "SGPC page variant: <word>"`. Example: 2026-11-10, Ang 678, SGPC wrote ਨਾਨਕੁ for the corpus's ਨਾਨਕ (lines 29433-29442).
`hs_layout_changed` / `hs_parse_failing`	SGPC changed its page. Save the archive for that day, fix the parser in a PR, add the page as a fixture.
A published day is the wrong shabad	`python -m hukamnama correct <date> --first-line <first> --last-line <last> --by <you> --reason "…"`. The day keeps its first publication time and gains `corrected_at`; the public files, the feeds (as an updated entry) and the pages follow within one poll interval (up to ten minutes when idle). When the CDN purge is configured (`HUKAMNAMA_VERCEL_PURGE_TOKEN`, `HUKAMNAMA_VERCEL_PROJECT`, `HUKAMNAMA_VERCEL_TEAM` on the service), the same sync deletes the `hukamnama` cache tag from Vercel's CDN, so the pages, feeds, embed and sitemap re-render from the corrected answer; without it a past day's page can stay cached for up to a day. A failed purge raises `cdn_purge_failed` and never stops publishing. The API host needs no purge: Cloudflare does not cache the Worker's answers. It sends no push: run `resend` as well if the phones should be told. Only a published day can be corrected; use `approve` for a held one.
Something is badly wrong and nothing should be shown	`python -m hukamnama pause --by <you> --reason "…"`. Within one poll interval every day reads as pending, the public files are removed and nothing is sent; `/health` and `/v1/status` say `paused`. The poller keeps deciding days, so `python -m hukamnama resume --by <you> --reason "…"` restores everything. Both are audited.
Service refuses to start: published days no longer prove	A dataset bump changed the corpus. At start the service proves every published day in the new corpus (same line ids, same `range_sha256`) and, when all prove, re-pins them and carries on (audited as `repin`). It refuses only when a day does not prove, and names it. Fix each named day with `correct` (or `hold`), then `python -m hukamnama repin` to confirm, and restart. The deploy workflows run the same proof before deploying, so this should be caught there first. Never delete the state file to get past it.

Push (APNs)

Sending is off unless the service has `HUKAMNAMA_SEND=on`. To turn it on (owner, once per service):

1. **APNs Auth Key:** Apple Developer → Certificates, Identifiers & Profiles → Keys → +, name it `Gurbani Soul push`, tick **Apple Push Notifications service (APNs)**, choose **Sandbox & Production**, and download the `.p8` once. It is not the App Store Connect API key.
2. **Render → service → Environment → Secret Files:** add `apns.p8` with the file's contents (paste it into the dashboard; never into chat, a ticket or the repository).
3. **Environment variables:** `HUKAMNAMA_APNS_KEY_FILE=/etc/secrets/apns.p8`, `HUKAMNAMA_APNS_KEY_ID=<the key's 10-character ID>`, `HUKAMNAMA_APNS_TEAM_ID=<team id>`, then `HUKAMNAMA_SEND=on`. **Save only**, then dispatch `hukamnama-deploy` outside the freeze.
4. Check `/health`: `push.sending` is `true`. If the key is missing or unusable the service refuses to start (Render keeps the previous instance running and shows the error in Logs).

Which phones a service reaches is decided by the app build: TestFlight builds register with **staging** (`api-staging.gurbanisoul.com`), App Store builds with **production** (`api.gurbanisoul.com`). So staging only ever pushes to TestFlight testers.

Every morning after publishing, `/health.push.today` shows the send states. Alerts: `push_breaker` (more than 5% of the first results were 4xx; the run stopped and nothing was deleted: check the key, topic and environment, then `resend`, which clears the breaker and is audited), `push_auth_failed` (403: the key was revoked or the IDs are wrong; the day pauses for 20 minutes and resumes by itself, the queued devices are not lost), `push_payload_too_large` (also a 20-minute pause), `device_flood` (the new-device cap was hit). To push a corrected day again: `python -m hukamnama resend <date> --by <you> --reason "…"`.

Not yet

The R2 copy of the fetch archive.

Links

1. https://litestream.io/
2. https://ntfy.sh/

Sri Guru Granth Sahib Ji — Knowledge Base wiki. Generated from commit 7783889. Updated 22 November 2026. Verified against 14c89ad on 2026-11-12. This copy may be out of date; the page is at https://docs.gurbanisoul.com/process/runbooks/hukamnama-service/