Ship & operate

The API image

4 min read · Updated 22 November 2026 · Verified against `c5806fd` on 2026-11-22

> *What webapp/Dockerfile builds: the pinned database checked inside the image, one image for every service through SGGS_MODULES, the baked commit, the health check and its CI proof.*

https://docs.gurbanisoul.com/process/api-image/

`webapp/Dockerfile` builds the container that runs the API server (`webapp/serve.py`) with its database. Render builds it for the `sggs-knowledge-base` service, which is kept as the API's rollback target (**environments**); CI builds the same image on every pull request and every push to `integration` and `main` (`web-ci.yml`, job `api-image`), so a broken Dockerfile or dataset pin fails there first. The Hukamnama service has its own image, `services/hukamnama/Dockerfile` (**runbook: Hukamnama service**).

```
docker build -f webapp/Dockerfile -t sggs-backend .   # from the repository root
docker run -p 7777:7777 -e PORT=7777 sggs-backend
```

The build context is the **repository root**, not `webapp/`: `serve.py` opens `../db/sggs.sqlite` relative to `webapp/`, and the image needs `dataset.lock.json` and the fetch script beside it.

What it does, layer by layer

1. **Base**: `python:3.12-slim`. The server is standard-library Python with no pip dependencies; Debian's SQLite has FTS5 and `bm25()`, which search and the analytics routes need.
2. **The database by pin.** `dataset.lock.json` and `scripts/data/fetch_dataset.py` are copied in and the script installs the exact database the lock names from sggs-data, checking its sha256 and size; any mismatch fails the build. The layer then checks the file starts with the SQLite header and makes it read-only (`0444`). Because only the lock and the script are copied before this step, the layer stays cached until the lock changes: a code-only build downloads nothing. (Why the database comes by pin: ADR-0008.)
3. **The server**: `webapp/` and `tools/slice_db.py`, laid out as `serve.py` expects.
4. **One image, any service.** The build argument and environment variable `SGGS_MODULES` defaults to `all`, every bounded context from the full database, which is what the Render rollback service runs. Given one or more contexts (for example `knowledge`), the build cuts that service's slice with `tools/slice_db.py`, which proves

every kept table equal to the full database and every kept FTS index whole, and puts the slice in place of the database at the same path. Render passes a service's environment variables to the build as build arguments.

5. **The commit, baked in.** The build argument `RENDER_GIT_COMMIT` (Render supplies it) becomes `SGGS_COMMIT`, which `serve.py` reports as `/api/health.commit` (or `unknown` when empty). The production deploy waits for that value to equal the SHA it deployed (**runbook: deploy**).
6. **Run.** `SGGS_OPEN_BROWSER=0`; the server listens on `$PORT` (falling back to 7777, the exposed port) and is started with `python serve.py` from `webapp/`.

Health

The image's `HEALTHCHECK` calls `/readyz` every 30 seconds (timeout 5 s, start period 15 s, 3 retries). `/readyz` answers 200 when every table the enabled contexts declare is present, and 503 otherwise; `/healthz` is the plain liveness answer. In split mode (`SGGS_MODULES` not `all`) `serve.py` also refuses to start at all when the database lacks a declared table.

> **Note.** The comment above `HEALTHCHECK` in the Dockerfile still says it probes `/api/meta`; the command it runs is `/readyz`. The command is what counts.

`/api/health` is the full integrity self-test (line and Ang counts, FTS, the verbatim checks, a live verification) and is too heavy for a probe; deploy verification calls it.

How CI proves it

The `api-image` job in `.github/workflows/web-ci.yml`:

1. builds the image exactly as Render does, with `RENDER_GIT_COMMIT` set to the commit under test, before the runner has the database (so the image must fetch the pin by itself);
2. runs it, waits for `/api/meta`, checks that `/api/health.commit` is the commit under test (`scripts/ci/wait_for_deploy.py`), and replays the golden contract over HTTP (`tools/contract_http.py`);
3. builds a second image with `SGGS_MODULES=knowledge` and checks that `/readyz` reports only the `knowledge` context, that `/api/timing/clock` answers 200 while `/api/ang/1` (another service's route) answers 404, and that the sliced database is under 5 MB.

Changing it

- A new table a context reads must be declared in `CONTEXT_TABLES` (`webapp/serve.py`), or the slice drops it.
- Keep the database layer before `COPY webapp/`: moving the code above it would make every code change download the database again.

- The repository's `.dockerignore` is an allowlist: only `webapp/`, `dataset.lock.json`, the fetch script, `tools/slice_db.py` and the Hukamnama service's files reach a build. A file the image needs must be added there, and the database never is: the image takes it only by pin.

Sri Guru Granth Sahib Ji — Knowledge Base wiki. Generated from commit 7783889. Updated 22 November 2026. Verified against c5806fd on 2026-11-22. This copy may be out of date; the page is at https://docs.gurbanisoul.com/process/api-image/