

iOS app

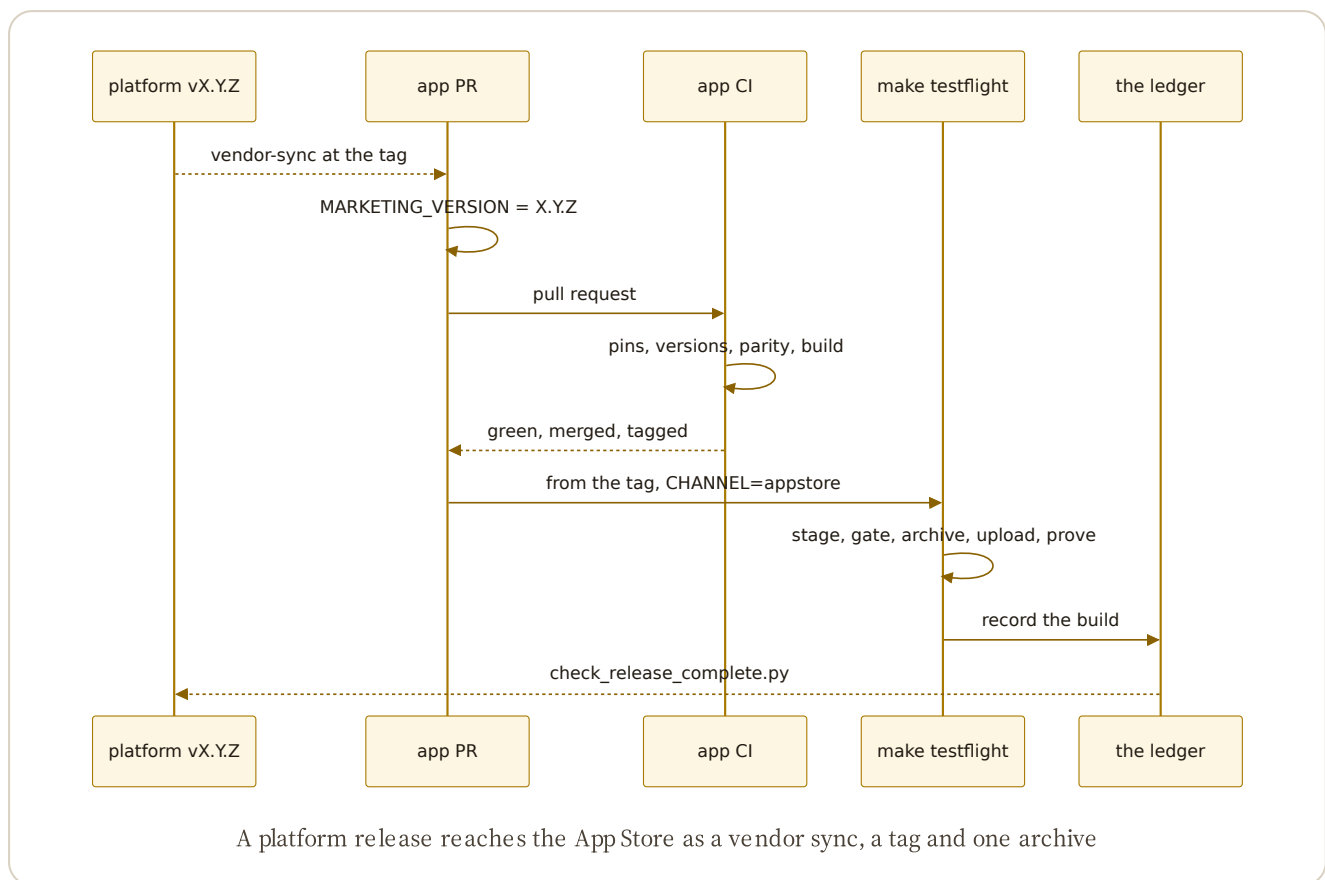
How the app takes a release

🕒 3 min read · Updated 27 September 2026 · Verified against [417c92f](#) on 2026-09-25

A platform release reaches the app as a reviewed vendor sync: the contract at the tag, one version number, an App Store archive from the tag, a ledger entry, and the completeness check.

<https://docs.gurbanisoul.com/ios/how-the-app-takes-a-release/>

A platform release ends with a tag vX.Y.Z on main (**the delivery pipeline**). It is **complete** only when the App Store binary reports the same number from the same platform commit. Between the two sits a small, fully mechanical flow in the app repository.



1. Vendor the release

make vendor-sync-platform REF=vX.Y.Z re-vendors the platform's golden contract (contract/_meta.json, every contract/golden_*.ndjson) and the contributors roster at the tag, **at the same paths they have upstream**, and records in vendor.lock.json the

source commit, the sha256 of every file and the platform's APP_VERSION at that ref. Vendored files are never edited in the app repository; CI rejects a local edit.

gurbani-soul-ios/scripts/vendor_sync.py · lines 1–20

```
#!/usr/bin/env python3
"""Vendored files from the platform and data repositories, pinned by commit and sha256.

The app is built and tested against inputs other repositories own: the golden contract, the
contributors roster and the brand tokens (platform) and the iOS database builder (sggs-
data).
They are committed here at the same paths they have upstream, so the app, its tests and its
tooling read them unchanged – and vendor.lock.json records where each came from and its
sha256.

python3 scripts/vendor_sync.py check          # every vendored file == its lock hash
(CI)
python3 scripts/vendor_sync.py check --remote # ... and == the file at source@commit
python3 scripts/vendor_sync.py sync platform <ref> # re-vendor from a platform ref
python3 scripts/vendor_sync.py sync data          # re-vendor from
dataset.lock.json's commit

`sync platform` also records the platform's APP_VERSION at that ref: the one-version policy
(scripts/release/check_versions.py) requires this app's MARKETING_VERSION to equal it, so a
binary always ships with the contract of the release it carries the number of.

Stdlib only. GH_TOKEN / GITHUB_TOKEN is sent when set (needed once the sources are private).
"""
from __future__ import annotations
```

Read from the source at build time — [view these lines on GitHub at a386422](#) .

Source: [scripts/vendor_sync.py](#)² in gurbani-soul-ios.

The dataset is a separate pin: dataset.lock.json there names the same sggs-data commit as this platform's, and make vendor-sync-data re-vendors the iOS database builder from that commit.

2. One number, enforced

MARKETING_VERSION in ios/App/project.yml must equal the vendored platform version (scripts/release/check_versions.py there; [ADR-0004](#) here). CURRENT_PROJECT_VERSION stays at the floor 1; every upload passes BUILD=N explicitly and the ledger keeps N monotonic within a marketing version, so a 20-minute archive never discovers Apple's rejection of a reused build number.

3. The pull request and the app's gates

Job	Proves
ci · gates	every vendored file equals its lock entry and its source at the pinned commit (vendor_sync.py check --remote); sggs-data publishes the pinned database; the version invariants; the iOS source, listing and archive gates (ios/tests)

Job	Proves
iOS fidelity gate · parity	both database profiles build; the Swift kit's parity suites pass against the vendored contract on the derived database
iOS fidelity gate · app	the app builds and its unit and UI tests pass on a simulator
security · secrets	no secret in the tree

PRs are squash-merged; the app then tags its own vX.Y.Z.

4. The archive, from the tag

make testflight TEAM_ID=... BUILD=N CHANNEL=appstore UPLOAD=1 is the **only** sanctioned way to build a candidate, run from a clean detached worktree at the tag. It refuses a dirty tree, waits for green CI on the commit, checks the version invariants and the ledger, then stages the database, runs the licence gate, generates a temporary Xcode project, archives, uploads, and proves what was built — the whole chain is **poster 12**. CHANNEL=appstore is declared up front: it makes the Nitnem scholar-review attestation a hard requirement, and only a build recorded with that channel may be submitted (make appstore-preflight).

5. The ledger

ios/testflight-builds.json is the tracked, append-only source of record for every upload. Each row carries the version and build, the app's source_commit, the platform_commit whose contract the binary vendors, the dataset_commit, the database profile, the channel, the bundled db_sha256, the Xcode and SDK versions and the upload time. It is committed with the release.

gurbani-soul-ios/ios/tools/testflight_ledger.py · lines 1-28

```
#!/usr/bin/env python3
"""
testflight_ledger.py – the source of record for every build uploaded to App Store
Connect, and the gate that keeps a build number monotonic *before* a 20-minute
archive+upload discovers Apple's rejection.

The ledger is ios/testflight-builds.json (tracked). Apple enforces two rules that
this tool mirrors:
- a CFBundleVersion (build number) may not be reused within a marketing version, and
- a marketing version already on TestFlight may not be undercut by a lower one.

Commands
check <version> <build> [--strict]
    Print `next: N` for the version. With --strict, exit 1 when <build> is not
    greater than every build already recorded for <version>, or when <version>
    is lower than the highest version in the ledger. Without --strict a violation
    is a warning and the exit code stays 0 (so a local re-archive with UPLOAD=0
    is never blocked).
next <version>
    Print only the next build number for <version> (1 if the version is new).
record <candidate.json> [--force]
    Append a row from the archive script's candidate-<v>-<b>.json. Refuses a
    duplicate (version, build). Refuses a candidate whose "uploaded" is false
    unless --force (the Transporter path, where the upload happens later).

Exit codes: 0 ok · 1 gate/refusal · 2 usage/IO error.
"""
import argparse, json, sys
```

Read from the source at build time — [view these lines on GitHub at a386422](#) .

Source: [ios/tools/testflight_ledger.py](#)⁴ in gurbani-soul-ios.

6. Complete

scripts/release/check_release_complete.py · lines 1-20

```
#!/usr/bin/env python3
"""check_release_complete.py – prove a release is COMPLETE across all three surfaces.

The one-number policy (docs/engineering/delivery.md "Versioning convention") says web, API
and the
shipped iOS binary must all report the same X.Y.Z from the same released commit.
This is the release-time, network-touching check that proves it – run as the final
step of sggs-release, and by sggs-verify-prod --ios.

For version X.Y.Z it asserts:
1. prod API /api/health.version == X.Y.Z and .commit == the commit tag vX.Y.Z points at
2. prod WEB /api/health.version == X.Y.Z and .commit == that same commit
3. the app's ledger (ios/testflight-builds.json in Algorythmos-AI/gurbani-soul-ios) has a
   channel="appstore" build of X.Y.Z built against that same commit: its platform_commit
   (the platform release whose contract the binary vendors) – or, for builds made before
the
   app had its own repository, its source_commit – equals it (full 40-hex).

Exit 0 if complete, 1 otherwise.
Usage: check_release_complete.py X.Y.Z [--api URL] [--web URL] [--ledger PATH]
"""
import argparse, json, os, subprocess, sys, urllib.error, urllib.request
```

Read from the source at build time — [view these lines on GitHub at c3d9360](#) .

Source: [scripts/release/check_release_complete.py](#)⁶ in this repository.

The check reads the ledger from the app repository's main and asserts that production's web and API report X.Y.Z from the commit the platform tag points at, and that an appstore build of X.Y.Z was built against that same commit. It is the last step of the release skill and of make verify-prod --ios.

Hotfixes

A one-sided fix is never shipped: a web or iOS fix bumps the **patch** everywhere, and the app re-archives at the new tag. The App Store has no binary rollback, so the app repository's hotfix runbook expires the bad TestFlight build, pauses a phased release and ships forward.

Links

1. https://github.com/Algorythmos-AI/gurbani-soul-ios/blob/a386422e90c2c681f9ecc4f7167b4deb6a35fb92/scripts/vendor_sync.py#L1-L20
2. https://github.com/Algorythmos-AI/gurbani-soul-ios/blob/main/scripts/vendor_sync.py
3. https://github.com/Algorythmos-AI/gurbani-soul-ios/blob/a386422e90c2c681f9ecc4f7167b4deb6a35fb92/ios/tools/testflight_ledger.py#L1-L28
4. https://github.com/Algorythmos-AI/gurbani-soul-ios/blob/main/ios/tools/testflight_ledger.py
5. https://github.com/Algorythmos-AI/sggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/scripts/release/check_release_complete.py#L1-L20
6. https://github.com/Algorythmos-AI/sggs-platform/blob/integration/scripts/release/check_release_complete.py

Sri Guru Granth Sahib Ji — Knowledge Base wiki. Generated from commit c3d9360. Updated 27 September 2026. Verified against 417c92f on 2026-09-25.
This copy may be out of date; the page is at <https://docs.gurbanisoul.com/ios/how-the-app-takes-a-release/>