

# GodwinOTT technical overview

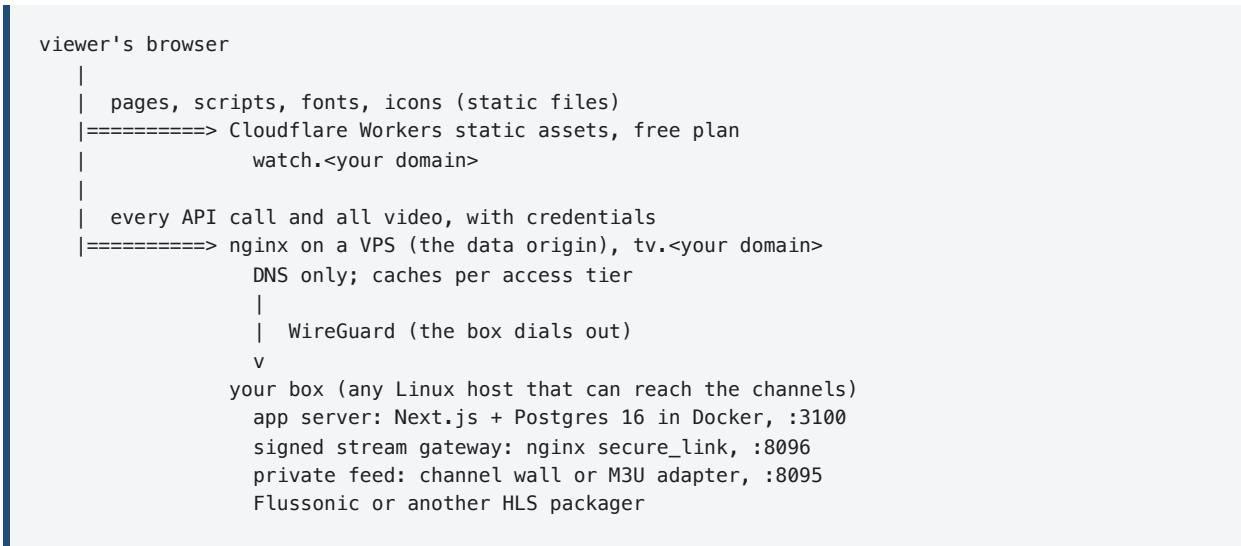
For your engineer: how GodwinOTT is built, what runs where, what you need to provide, and where its limits are.

OCTOBER 2026

## The short version

One Next.js 16 codebase (React 19, strict TypeScript, Tailwind 4) is built two ways. The static build is plain files served by Cloudflare Workers static assets on the free plan, at your watch domain. The server build is a Next.js standalone Node server in Docker beside Postgres 16 on your own machine. Between the two sits an nginx data origin on a VPS, linked to your machine by WireGuard. Video comes from your own HLS packager and never touches Cloudflare.

## Architecture



## What runs where

| Where                 | What                                        | Notes                                                                                                                       |
|-----------------------|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Cloudflare, free plan | The static front end at watch.<your domain> | No code of ours runs per request. Hashed scripts and fonts are cached for a year.                                           |
| VPS with public IPv4  | nginx as data origin at tv.<your domain>    | DNS only in Cloudflare, never proxied. Terminates TLS (certbot), caches by access tier, forwards to the box over WireGuard. |

| Where    | What                                                                | Notes                                                                                                          |
|----------|---------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Your box | App server in Docker, published only on <code>127.0.0.1:3100</code> | Postgres 16 in the same compose file. Idled at about 120 MB in testing; ffmpeg adds about 100 MB to the image. |
| Your box | Signed stream gateway, nginx <code>secure_link</code> , port 8096   | Checks the signature and expiry on every playlist and segment request.                                         |
| Your box | Private feed service, port 8095                                     | Serves the channel list as JSON, refreshed every 5 seconds.                                                    |
| Your box | HLS packager                                                        | The first deployment runs Flussonic. Any packager that writes one HLS folder per channel works.                |

The box dials out to the VPS over WireGuard, so it needs no open port. A Cloudflare tunnel ( `cloudflared` ) is also set up for the `streams` hostname.

The page head asks for data before the app's scripts arrive, which is why first paint is quick from far away. Measured from India against a US data origin 250 ms away on 29 September 2026: first paint 0.1 s on every page, home hero picture 1.5 to 1.9 s on a first visit.

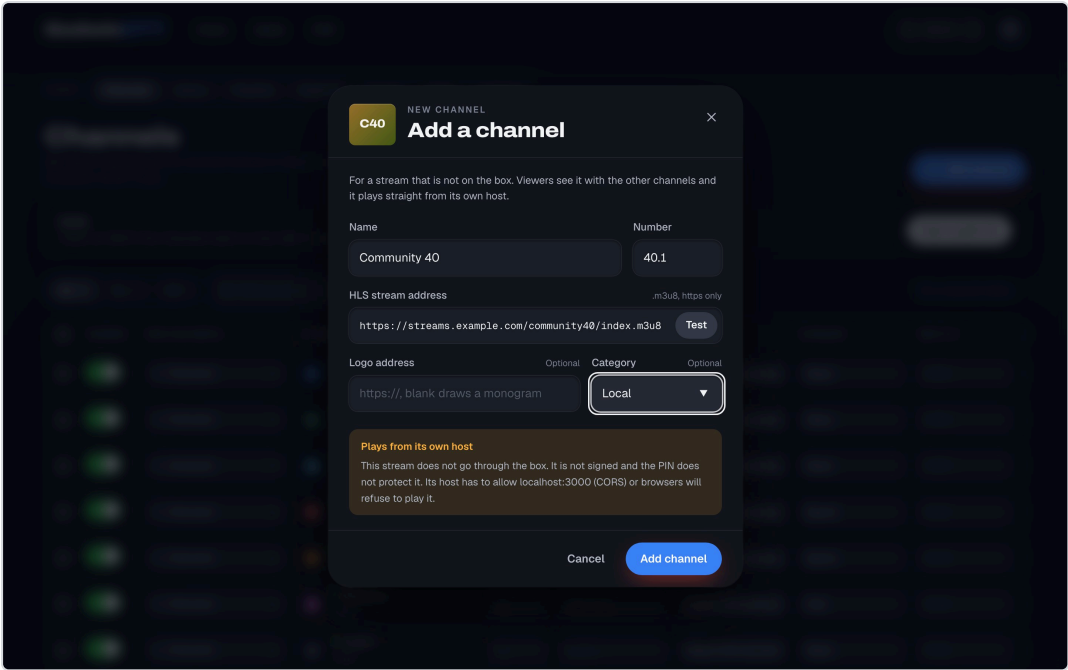
## What never touches Cloudflare

Video, playlists, live thumbnails, uploads and every API call go to the data origin. Cloudflare's terms restrict serving video through its proxy, and the platform uses only the free plan, so the two are kept apart. The static build step ( `scripts/finish-static.mjs` ) fails if an API route, server code or an `.env` file lands in the output, or if no data origin is set.

## The feed contract

The app reads the lineup from a JSON contract called `BoxFeed` every 5 seconds. With Flussonic the feed comes from its channel wall. Without Flussonic, an adapter builds the same contract from an M3U lineup plus XMLTV. The contract follows Flussonic's layout, so another packager needs the adapter and matching segment names. However many viewers are watching, the box sees one feed request every 5 seconds.

A hand added channel skips the box entirely: the admin gives an https `.m3u8` address and the Test button fetches it from the server before it is saved.



Adding an outside channel by its HLS address, with a Test before it is saved.

Guide

XMLTV is pulled every 30 minutes from `EPG_XMLTV_URL` , or uploaded by an admin (up to 20 MB). Import limits: 30,000 programmes, 50 MB, 60 seconds, 8 days ahead, 12 hours per programme. A 51 channel file of 1,300 programmes imports in well under a second.

On demand packaging

Uploads (MP4, MOV, WebM, MP3, M4A, AAC, WAV) are chunked and resumable, 20 GB per file by default. An upload is refused if it would leave less than 20 GB free, counting what running uploads still need. ffmpeg packages to HLS one job at a time, at idle priority, with at most 4 threads, so the GPU stays with the live channels. H.264 files up to 1080p and 10 Mbps are copied in seconds. Anything else is encoded with libx264 at CRF 21 with a 6 Mbps cap.

Playlist channels

A playlist channel writes a live HLS playlist over segments that already exist. Nothing is re-encoded. The schedule is planned 7 days ahead in the channel's time zone, across daylight saving changes. Limits: uploaded items only, no mixing audio with video, web only, at most 50 such channels.

Requirements

You provide, or we provide on your behalf and in your name:

| Item  | Detail                                                                                                                                                                                                                                           |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A box | Your head end, or any Linux host that can reach the channels. The first deployment runs Ubuntu 24.04. It needs Docker, nginx built with <code>secure_link</code> , WireGuard, <code>cloudflared</code> , ffmpeg, and about 2 GB of memory spare. |

| Item                  | Detail                                                                                                            |
|-----------------------|-------------------------------------------------------------------------------------------------------------------|
| Postgres 16           | Ships in the Docker compose file. Nothing to install separately.                                                  |
| An HLS origin         | Flussonic, or any packager writing one HLS folder per channel, through the M3U adapter.                           |
| A VPS                 | Public IPv4 and enough bandwidth for your audience. Runs nginx, WireGuard and certbot.                            |
| A Cloudflare account  | Free plan. Your domain's DNS lives there. Nothing needs a paid Cloudflare feature.                                |
| A Google OAuth client | Required, because admins must sign in with Google. Google consent review is outside our control and is not timed. |
| Stripe or Razorpay    | Optional. Only if you charge. Test keys work.                                                                     |
| A player licence      | See "The player licence" below.                                                                                   |
| A private repository  | For your <code>client.json</code> and your secrets. Never in the product repository.                              |
| Optional              | A Google Drive API key for Drive sync, an XMLTV source for the guide.                                             |

## Bandwidth

Each viewer pulls about 1.8 Mbps from the data origin. The box sends each channel to the VPS once, however many people watch. That is why the VPS is where your bandwidth bill sits, and why nothing autoplays: the first deployment's box has about 31 Mbps of upload.

A rough guide, by our own arithmetic from the 1.8 Mbps figure (an estimate): one viewer hour is about 810 MB, so 1 TB of VPS transfer covers about 1,200 viewer hours, and 100 people watching at once need about 180 Mbps from the VPS.

## Provisioning from one client.json

Each client is described by one file, `client.json`, kept in that client's private repository. It holds a short key, the display name, four domains (zone, watch, data, streams), the super admin emails, brand seeds, Cloudflare and Google details, the VPS, WireGuard and box addresses, provider choices (feed, billing, DRM, player), auth mode and time zone.

`scripts/new-client.mjs` checks every field and names each bad one before it writes anything. It writes `deploy/.env` with every secret generated by `openssl rand`, kept across reruns so a rerun never breaks a live install. It also writes `.env.static` for the front end and `stack.env` for the delivery stack. Anything still unknown comes out as `FILL_IN`. A test checks that the output passes the production config check.

The order of work:

1. Fill in and render `client.json`.

2. DNS: `tv` as an A record to the VPS, DNS only; `streams` as a CNAME to the box's tunnel; `watch` left for the Worker.
3. Create the tunnel and WireGuard keys, add the public keys, render again.
4. Render the delivery stack from its separate private repository.
5. VPS: WireGuard, the nginx site, a certificate from certbot.
6. Box: gateway, feed, tunnel, then the Docker compose build and a health check.
7. Static site: copy `.env.static` and `brand/`, run `npm run deploy:static`.
8. First super admin: register the OAuth origins and the one redirect URI, sign in with Google.
9. Branding, billing and DRM on their screens. Each ships off or neutral.
10. Check from outside: Cloudflare serves the pages, CORS from the data origin is right, health answers, a forged stream link gets 403, the Google redirect is correct.
11. Write down rollback for the image, the static site ( `wrangler rollback` ), the database and the stack.

The docs time only two steps: containers healthy after about half a minute, and a new Worker domain that may take a minute. Our own estimate is about a working day for someone experienced, once your machines and accounts are ready. DNS propagation, certificates and Google's consent review can stretch that, and none of them are in our hands.

Icons, the install manifest and the 1200 by 630 share image are fixed at build time from the `brand/` folder, because browsers fetch them before any script runs. Every other piece of branding is set at run time on the Branding screen.

## Security posture

### Signed play links

Every play link carries an expiry and a signature, checked by nginx `secure_link` at the VPS and again at the box. Links last 6 hours, or 15 minutes for members only content, and renew 3 minutes before expiry without a reload under `hls.js`. The signature is MD5 because that is what nginx checks natively. The project's own docs are plain about it: it signs a link, it protects no secret. Tests check the signatures against openssl vectors.

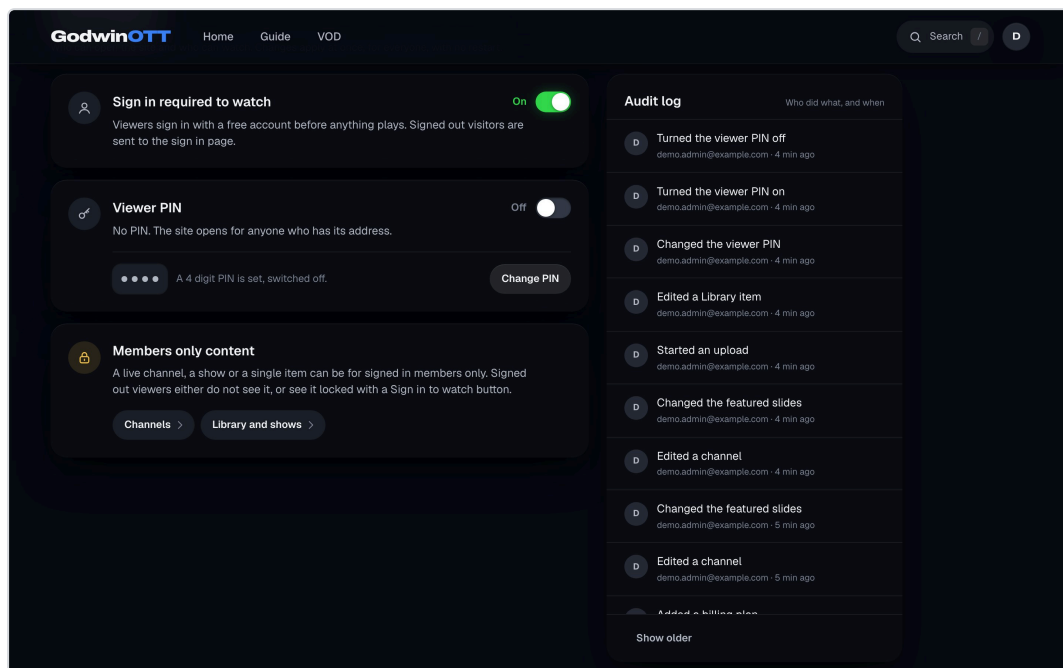
### Trusted origins

Only the public watch URL and the data origin are trusted. Writes from anywhere else get 403. Production has no default origin, so a misconfigured install answers 503 instead of quietly pointing at another client's settings.

### Admins

Admins and super admins are listed in the server environment ( `SUPER_ADMIN_EMAILS` for super admins) and nowhere else. No screen can promote anyone. An admin must be listed, must have an address verified by Google, and must be on a session created by a Google sign in. A password

account registered at an admin's address is an ordinary viewer, so nobody can squat an admin address. Changes to access, deletion, branding, billing and DRM need a sign in within the last 12 hours. Every admin write goes into an audit log, which never stores keys.



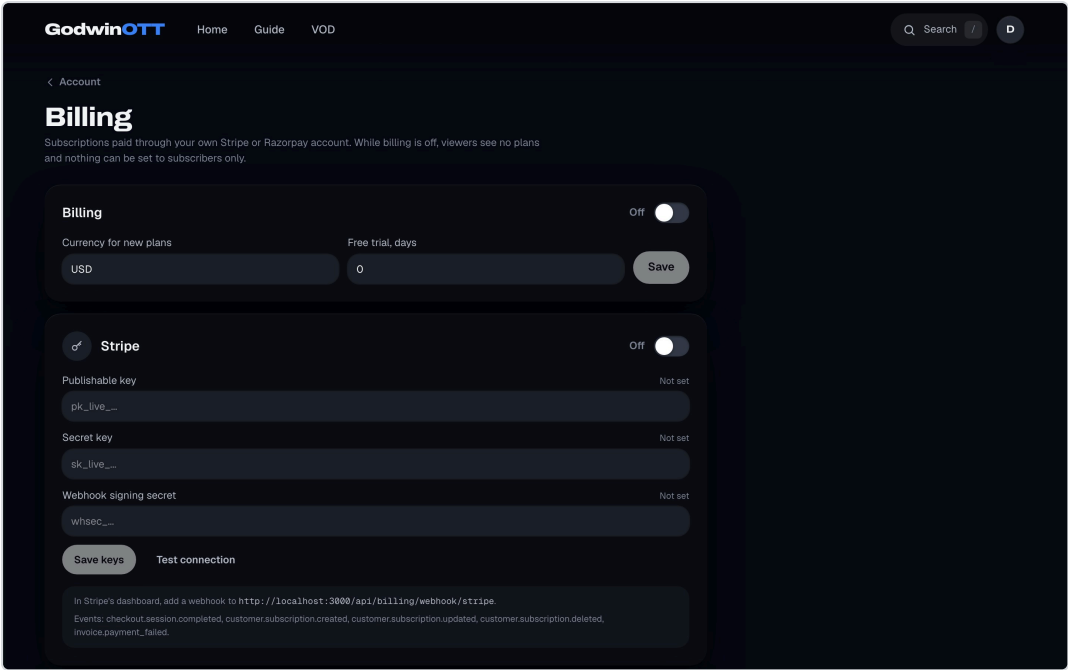
The audit log beside the Access settings records who changed what, and when.

## Accounts

Better Auth, script password hashing, sessions in Postgres. Five sign in attempts a minute per address. Blocking an account works at once, even against the 5 minute session cookie cache.

## Billing keys

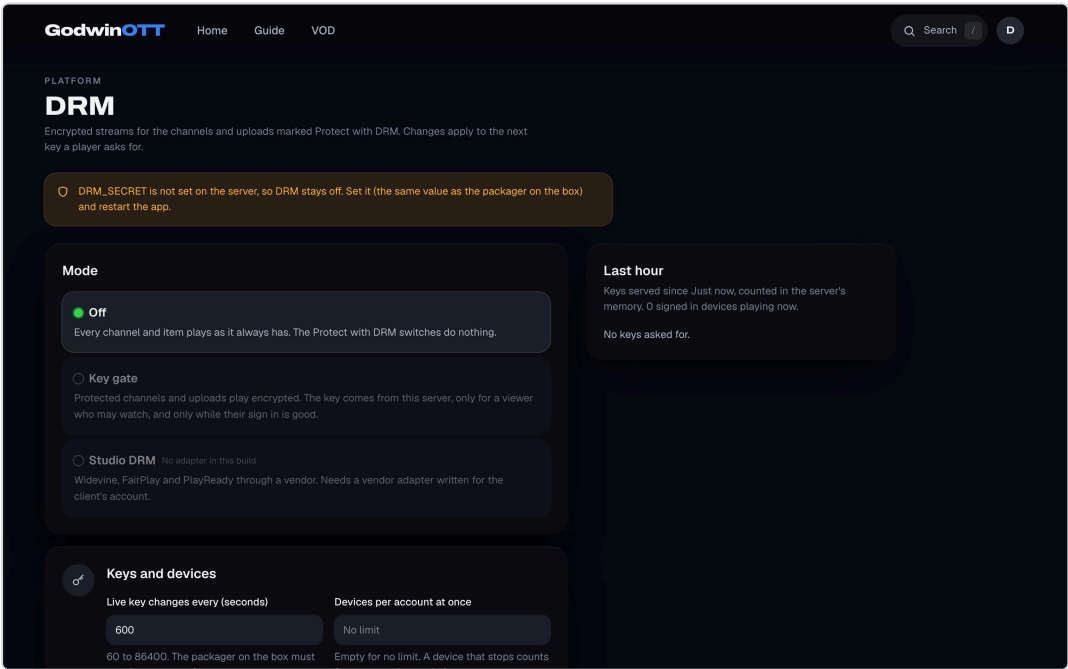
Stripe and Razorpay keys are encrypted with AES-256-GCM under a key derived by HKDF from `BILLING_KEY`. The Billing screen shows only the last four characters. Checkout is always the provider's hosted page, so card details never reach the platform. Webhooks are signature checked, applied once, and out of order events are skipped. There is no payment SDK; the calls are plain REST.



Billing stays off until provider keys are entered, and stored keys are shown only by their last four characters.

The DRM key gate, and its limits

With the key gate on, the box encrypts segments with AES-128. Keys are derived from a secret, never stored, and rotate every period on live channels (60 to 86,400 seconds, default 600). The key route checks a token bound to the content, its expiry and the session, rechecks membership and blocking, and enforces the device limit. Safari gets a per viewer playlist. Without the DRM secret set, no DRM code runs at all, and tests confirm that DRM off plays exactly as before.



The DRM screen with no secret configured: the key gate is offered, and studio DRM is shown as having no adapter in this build.

What it does not do: it does not stop screen recording, and it does not stop a viewer reading the key inside their own browser. It stops a link being passed on and caps devices per account. Studio DRM (Widevine, FairPlay, PlayReady) has an interface and a vendor template, but no vendor is connected and vendor mode is refused. It would need CMAF packaging and a FairPlay certificate. Outside streams, HLS links and YouTube items cannot be protected by either.

## The launch PIN

One shared PIN of 4 to 8 digits, stored with salted script under a pepper, 8 wrong tries per address per 15 minutes, a signed 7 day cookie that stops working when the PIN changes. The docs call it a sharing gate, not security, and so do we.

## Outbound fetches

Custom channel tests, Drive sync and YouTube import refuse private addresses, resolving DNS once, which guards against server side request forgery.

## Personal data stored

Name, email, password hash or Google link, sessions (no IP address), favourites, audit rows, subscriptions and viewing analytics. Analytics keep no IP address: it is hashed under a daily key held only in memory, for rate limits. Anonymous viewers get a random device id. Detailed events are kept 30 days, linkable summaries 90 days, anonymous totals 2 years. Deleting an account cuts its viewing history loose from it.

One gap is open. Billing webhook payloads, which for Stripe include an email address, are not yet pruned when an account is deleted.

## When the box is unreachable

The server keeps the last good feed. If the box stops answering, channels are still listed but marked not fresh, a "Reconnecting" pill appears, and health reports `feed: "stale"` without marking the app unhealthy. Pages never wait on the box.



With the feed interrupted, the player keeps the guide information on screen and says it is reconnecting.

If Postgres is down, settings keep their last good copy. With no copy at all, the access gate fails closed with a 503. If WireGuard is down, the box's own nginx still starts.

Our reading of this (inferred, not tested as a whole): with the box fully unreachable, the pages load from Cloudflare and look right, and nothing plays.

Turning a channel off removes it from the lineup, guide and search within about 10 seconds and refuses new plays. A link already handed out keeps working until it expires, for up to six hours.

## Backups and rollback

A nightly `pg_dump`, kept two weeks, which you should copy off the box. Uploaded media is backed up separately. After a restore, accounts deleted since the dump are deleted again. An update tags the running image first, so rollback is one command, and database migrations only ever add.

## Test coverage

48 Vitest files with about 545 test cases. The largest suites cover on demand (37), admin (32), Drive (30), billing (28), access tiers (28), the on demand viewer (24), DRM (22), access and PIN (20), featured (19) and branding (19). Eleven suites run against a scratch Postgres when `TEST_DATABASE_URL` is set; some skip without ffmpeg. ESLint and strict TypeScript throughout.

Some things the tests check, by name: play link signatures match openssl vectors; billing off leaves every access rule byte for byte as before; DRM off plays exactly as before; the playlist planner handles daylight saving and gaps between slots; two upload chunks for one offset never both land. Analytics were benchmarked on 3.5 million seeded events, with every route answering under 30 ms at the 95th percentile.

## The player licence

The live player and the on demand video player play through Ultimate Video Player, a commercially licensed component, rebranded so viewers never see the vendor, driving hls.js (Safari plays HLS natively). The licence held today belongs to the first deployment. Each new client needs its own licence, or needs the plain hls.js player for those two pages, which is not built yet. The project's provisioning notes estimate it at roughly a few days of work. The shorts feed and the audio player already use hls.js directly, so they are not affected.

This is the first thing to decide, before anything is promised to your viewers.

## Other limits worth knowing

- One server process. Running more than one would need the guide schedule behind a database lock.
- No native mobile or TV apps; it is a web app with an install manifest and no service worker.
- No email sender, so no verification or password reset emails.
- No casting, no captions.
- Analytics report in one time zone ( `ANALYTICS_TIME_ZONE` , default America/New\_York), which must sit a whole number of hours from UTC all year.
- Abandoned partial uploads are not cleaned up automatically.
- The privacy page is a draft, and the delivery stack lives in a separate private repository.
- A small amount of code is specific to the first deployment (a quiz guide, and Drive conventions built around its daily shows).