OpenD Administrator Guide

OpenD Documentation

Release 0.2

Generated 2026-11-01

Contents

Hosted architecture: push to publish

Self-hosting

- Run OpenD with Docker Compose
- Run OpenD on Railway
- Environment variable reference
- Console settings, environment variables and the secret key
- Store blobs in a folder or an S3-compatible bucket
- Connect GitHub from the admin console
- Connect GitHub: webhooks and the GitHub App
- One GitHub App for the site, the editor and builds
- Make a site public or private
- Let readers choose the version they read
- Release lines by major or minor version
- Set up reader sign-in with OIDC
- Let readers sign in with GitHub
- Run the web editor

Publish to a docs site someone else operates

Run your site from the admin console

- Change the site's look from the admin console
- Serve your site at your own domain
- Choose a PDF template

Readers and administration

- Manage organizations, entitlements and readers
- Admin API reference
- Docs analytics and reader feedback

Embedded help

- Context-sensitive help in your apps
- Framing, CSP and cookies
- Troubleshoot embedded help

PDF publications

- Per-reader filtering in PDFs
- PDF rendering and limits

Operations

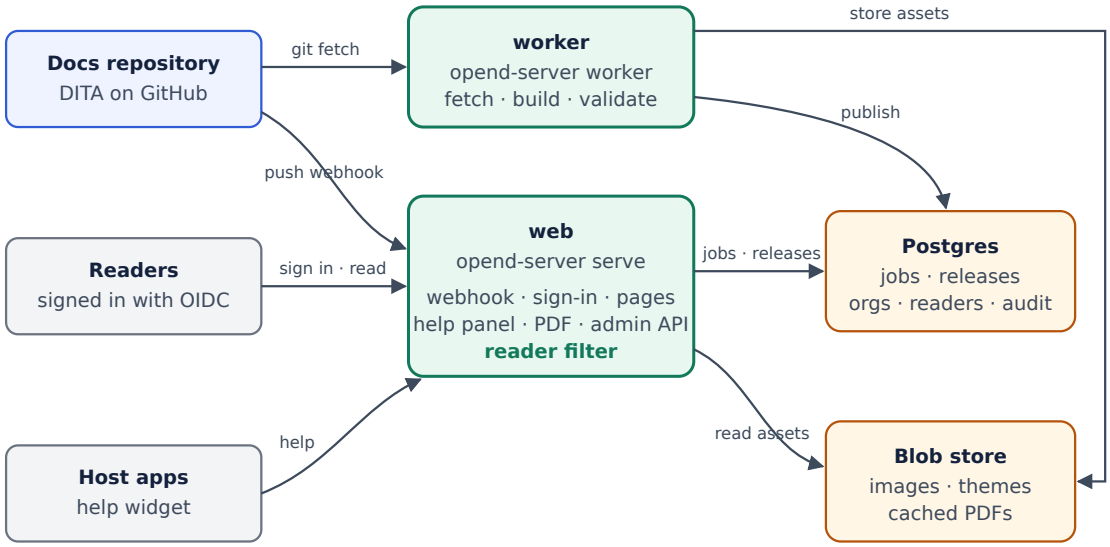
- Back up and restore
- Upgrade OpenD

Validation diagnostics

Hosted architecture: push to publish

A push to the docs repository becomes live, filtered docs through four parts: a web service, a build worker, Postgres and a blob store.

Web, worker, Postgres and a blob store



The parts

web (`opend-server serve`)
: Receives GitHub webhooks, signs readers in, serves pages, the help panel, PDFs and the admin API. Stateless: run as many replicas as you like.

worker (`opend-server worker`)
: Claims build jobs, fetches the exact pushed commit, builds and validates it, uploads images and the theme, indexes it for search, and publishes. Several workers can share the queue.

Postgres
: The job queue, release snapshots and their search indexes, which release is live for each release line, organizations, entitlements, readers, webhook deliveries and the audit log.

Blob store
: Images and themes for each commit, and cached PDFs. A local folder when web and worker share a disk, otherwise any S3-compatible bucket.

An optional fifth part, the web editor (`opend-server editor serve`), runs on its own host. It lets authors who do not use Git save to their own branch and publish through a pull request, committing through the GitHub App, so content still reaches readers only by a push. See Run the web editor.

One image runs every role: `serve`, `worker`, `migrate`, `seed`, `reconcile`, `reindex` and `editor`. It is configured only with environment variables; see the environment variable reference.

From push to live

1. GitHub sends a push webhook to `/webhooks/github`. The web service checks its signature, ignores deliveries it has already seen, and queues a build if the pushed branch or tag is one of the repository's release refs. It answers 202 only once the job is safely stored.
2. A worker claims the job (it wakes on a notification, and also polls every five seconds), fetches that one commit, and runs the same build and validation as `opend validate`.

3. If validation finds an error, the build fails: the diagnostics are stored with the job, the commit gets a failed `opend/build` status, and the previous release stays live.
4. Otherwise the worker stores the images and theme, saves the release snapshot, and makes it live for its release line, unless a newer build already went live.
5. Infrastructure problems, such as a failed fetch, are retried with growing delays, up to five attempts.

Serving readers

- A signed-in reader's organization gives their products, features and version. The version's major number picks the live release; the reader filter does the rest.
- Filtered views and pages are cached per entitlement set, not per user, and cleared at once when an admin changes entitlements or a new release goes live.
- Pages are sent with `Cache-Control: private, no-store` and a strict Content Security Policy, and run no scripts.
- The site serves one docs repository. If several are registered, `OPEND_SITE_REPO` says which.

Multi-tenancy, with many sites in one deployment, is planned for a later release.

Self-hosting

Run OpenD with Docker Compose

The smallest install: the app, the build worker and Postgres on one host, from the Compose file in the repository.

Before you begin

You need Docker with Compose, a checkout of the OpenD repository, and a host name readers can reach over HTTPS (terminate TLS in a reverse proxy in front of port 3000).

1. Create the configuration from the examples.

   ```
   cp deploy/.env.example deploy/.env
   cp deploy/seed/seed.example.yaml deploy/seed/seed.yaml
   ```

2. Edit `deploy/.env`.
   Set at least `PUBLIC_URL` and a random `SESSION_SECRET`. For publishing set `GITHUB_WEBHOOK_SECRET`; for reader sign-in, the `OIDC_*` variables; for the admin API, `OPEND_ADMIN_TOKEN`. Change `POSTGRES_PASSWORD`. See the environment variable reference.

   > **Tip:** Generate secrets with `openssl`: `openssl rand -hex 32` prints 64 random hexadecimal characters.

3. Edit `deploy/seed/seed.yaml` with your docs repository, organizations, entitlements and readers.
   See Manage organizations, entitlements and readers for the format.

4. Build and start the stack.

   ```
   docker compose -f deploy/docker-compose.yml up -d --build
   ```

   Compose starts Postgres, runs the `migrate` service once to create the schema, then starts `app` on port 3000 (or `OPEND_PORT`) and `worker`. Both share the `blobs` volume at `/data/blobs`.

5. Load the seed file.

   ```
   docker compose -f deploy/docker-compose.yml run --rm migrate seed /seed/seed.yaml
   ```

   Seeding is idempotent: everything is upserted, so run it again after each change.

6. Check health: `curl http://localhost:3000/healthz` returns `{"status":"ok"}`.
7. Connect GitHub and your identity provider.
   See Connect GitHub: webhooks and the GitHub App and Set up reader sign-in with OIDC.

Results

Readers sign in at `PUBLIC_URL` and get the live release for their version's line. The first push to a release ref publishes it.

To evaluate without an identity provider, set `NODE_ENV=development` and `OPEND_DEV_LOGIN=1`, then sign in at `/auth/dev` as any seeded reader. Never do this with real content.

The web editor is an optional Compose profile, `editor`. See Run the web editor.

Run OpenD on Railway

One Railway project runs one OpenD instance: a web service, a worker, Postgres and a bucket.

About this task

On Railway the web service and the worker are separate services that cannot share a disk, so blobs go to a Railway Bucket. Railway has deprecated `railway.json`, so apply the settings below per service, in the dashboard or through the Railway API. See Railway's documentation for the platform itself.

1. Create a project and add Postgres and a Bucket.
2. Add two services from this repository, `web` and `worker`, both building `deploy/Dockerfile`, and configure them.

Setting	web	worker
Source	this repository, branch `main`	same
Dockerfile path	`deploy/Dockerfile`	same
Start command	`node dist/bin.js serve`	`node dist/bin.js worker`
Pre-deploy	`node dist/bin.js migrate`	none
Healthcheck	`/healthz`, timeout 120 s	none
Restart policy	on failure, 5 retries	always
Watch paths	`packages/**`, `deploy/Dockerfile`, `package.json`, `pnpm-lock.yaml`	same
Public domain	yes, port 3000	no

   A start command replaces the image's entry point, so always give the full `node dist/bin.js <command>`. Watch paths keep docs-only commits from redeploying the app: docs reach readers through the worker, not through a redeploy.

3. Set the variables on both services, using reference variables for Postgres and the bucket.

```
DATABASE_URL=${{Postgres.DATABASE_URL}}
S3_BUCKET=${{Bucket.BUCKET}}
S3_ENDPOINT=${{Bucket.ENDPOINT}}
S3_ACCESS_KEY_ID=${{Bucket.ACCESS_KEY_ID}}
S3_SECRET_ACCESS_KEY=${{Bucket.SECRET_ACCESS_KEY}}
S3_REGION=${{Bucket.REGION}}
PUBLIC_URL=https://<web domain>
SESSION_SECRET=<openssl rand -hex 32>
OPEND_ADMIN_TOKEN=<openssl rand -hex 32>
GITHUB_WEBHOOK_SECRET=<openssl rand -hex 32>
NODE_ENV=production
```

   Replace `Bucket` with your bucket service's name. Railway buckets use virtual-hosted addressing, so leave `S3_FORCE_PATH_STYLE` unset. Add the `OIDC_*` variables for reader sign-in, and `OPEND_EMBED_SECRET` and `OPEND_EMBED_ORIGINS` to embed help in your apps.

4. Deploy, and wait for `web` to report healthy.
   Its pre-deploy step creates the schema. The worker retries until the schema exists.

5. Register the docs repository, and add organizations and readers, through the admin API.

```
curl -X POST "$PUBLIC_URL/admin/api/repos" -H "authorization: Bearer $OPEND_ADMIN_TOKEN" \
  -H 'content-type: application/json' \
  -d '{"fullName":"acme/docs","sourceUrl":"https://github.com/acme/docs.git","releaseRefs":
["main","v*"]}'
```

   See Manage organizations, entitlements and readers.

6. Publish: point a GitHub webhook at `$PUBLIC_URL/webhooks/github`, or trigger a build directly.

```
curl -X POST "$PUBLIC_URL/admin/api/repos/1/builds" -H "authorization: Bearer
$OPEND_ADMIN_TOKEN" \
  -H 'content-type: application/json' -d '{"ref":"main","sha":"<full commit sha>"}'
```

Results

A public demo can set `NODE_ENV=development` and `OPEND_DEV_LOGIN=1` so visitors pick a sample reader. Never do this with real content.

To add the web editor, create a third service from the same Dockerfile with the start command `node dist/bin.js editor serve` and its own domain. See Run the web editor.

Tagged releases are published as images to `ghcr.io/desktopmachineshop/opend`, so a service can deploy an image tag instead of building from the repository.

Environment variable reference

Every setting of `opend-server`, with its default and the rules it is checked against at start-up.

How settings are read

- All settings are environment variables; nothing is baked into the image. Give the web service and the worker the same values.
- Every command (`serve`, `worker`, `migrate`, `seed`) checks the whole set at start-up and refuses to start with a list of every problem.
- An empty value counts as unset. So does an unresolved reference variable on Railway.
- Flags are on when set to `1` or `true`, and off otherwise.
- The web editor has its own settings, listed in Run the web editor.
- Some settings can instead be changed by staff in the admin console (sign-in, the GitHub connection, editor links). A variable that is set always wins over the console, which then shows the setting read-only. See Console settings, environment variables and the secret key.

> **Note:** The server reads its settings once, at start-up. After you change a variable, restart (or redeploy) both the web service and the worker.

Core

Variable	Default	Meaning and rules
`DATABASE_URL`	required	Postgres connection string, such as `postgres://opend:secret@db:5432/opend`.
`PUBLIC_URL`	`http://localhost:3000`	Where readers reach the site. Must be a URL; a trailing slash is removed. It is the OIDC callback base, the audience of embed tokens, and the base of absolute links in PDFs and the JSON API. When it starts with `https://`, cookies are marked `Secure`.
`PORT`	`3000`	Port to listen on: an integer from 0 to 65535.
`HOST`	`0.0.0.0`	Interface to listen on.
`SESSION_SECRET`	required	Signs session cookies. At least 32 characters. Changing it signs every reader out. Without `OPEND_SECRET_KEY`, the key that seals stored secrets is derived from it, so changing it also makes those secrets unreadable until they are entered again.
`OPEND_SECRET_KEY`	derived from `SESSION_SECRET`	Seals the secrets the admin console stores (client secrets, the GitHub App's private key and webhook secret) with AES-256-GCM. At least 32 characters. Set it to rotate `SESSION_SECRET` freely; give the web, worker and editor services the same value. See Console settings, environment variables and the secret key.
`NODE_ENV`	`development` (the image sets `production`)	With `production`, the server refuses to start with `OPEND_DEV_LOGIN`, and sign-in refuses an `http://` OIDC issuer.
`OPEND_SITE_REPO`	none	The repository the site serves, as `owner/name`, when more than one is registered. Without it, the site works only while exactly one repository is registered.
`OPEND_ADMIN_TOKEN`	none	Bearer token for the admin API. At least 24 characters. Without it the admin API does not exist. The admin console at `/admin` does not need it: staff readers sign in.

GitHub

Variable	Default	Meaning and rules
`GITHUB_WEBHOOK_SECRET`	none	The secret set on the repository's or GitHub App's webhook. The webhook secret of the app connected in the admin console is accepted as well, since that app signs its own deliveries with it; without either, `/webhooks/github` answers 503. In managed hosting, the control plane re-signs forwarded deliveries with this value.
`GITHUB_APP_ID`	none	The GitHub App's id, for private repositories and commit statuses. Set together with `GITHUB_APP_PRIVATE_KEY`. Without it, **Connect GitHub** in the admin console creates and stores an app instead (Connect GitHub from the admin console). Not together with `OPEND_GITHUB_BROKER_URL`.
`GITHUB_APP_PRIVATE_KEY`	none	The App's private key, PEM, PKCS#1 as GitHub issues it or PKCS#8. Literal `\n` sequences are turned into line breaks, so the key fits on one line. Set together with `GITHUB_APP_ID`.
`GITHUB_API_URL`	`https://api.github.com`	API base URL, for GitHub Enterprise Server, such as `https://github.example.com/api/v3`. Must be a URL.
`OPEND_GITHUB_BROKER_URL`	none	Managed hosting only: the control plane's token broker, such as `https://control.example/broker/v1`. The worker, poller and editor get installation tokens from it (`read` for builds, `write` for the editor and commit statuses) instead of a GitHub App of their own, and the admin console's repository picker lists what it returns. Set by the hosting provider, together with `OPEND_GITHUB_BROKER_SECRET`.
`OPEND_GITHUB_BROKER_SECRET`	none	This instance's bearer token for the broker. At least 16 characters.
`OPEND_HOSTING_URL`	none	Managed hosting only: the hosting provider's site API, such as `https://control.example/site/v1`. With it the admin console's **Custom domain** page sets the site's domain and shows the DNS records to add (Serve your site at your own domain); without it the page explains that the domain is `PUBLIC_URL`. Independent of how the site reaches GitHub. Set by the hosting provider, together with `OPEND_HOSTING_TOKEN`. When neither is set, a broker at `…/broker/v1` implies `…/site/v1` with the broker secret.
`OPEND_HOSTING_TOKEN`	none	This site's bearer token for the site API. At least 16 characters.

Variable	Default	Meaning and rules
`OPEND_GITHUB_INSTALL_URL`	none	Managed hosting: the shared GitHub App's installation page, such as `https://github.com/apps/opend/installations/new`. The admin console's **Connect GitHub** button links to it.

Reader sign-in

Variable	Default	Meaning and rules
`OIDC_ISSUER`	none	The OpenID Connect issuer URL. Requires `OIDC_CLIENT_ID` and `OIDC_CLIENT_SECRET`.
`OIDC_CLIENT_ID`	none	The client id registered with the provider.
`OIDC_CLIENT_SECRET`	none	The client secret.
`OIDC_SCOPES`	`openid email profile`	Scopes to request, space-separated. Keep `email`: readers are matched by verified email.
`OIDC_DISPLAY_NAME`	`your company account`	How the sign-in page names the OIDC provider: **Sign in with *name***. At most 60 characters. With `GitHub`, the button shows GitHub's mark.
`OIDC_ICON`	`key` (`github` when the display name is GitHub)	The OIDC button's icon: `key` or `github`. Managed hosting signs readers in with GitHub through its own OIDC provider and sets `github`.
`GITHUB_OAUTH_CLIENT_ID`	none	Adds **Sign in with GitHub**. The client id of the GitHub App (or an OAuth App). The same name and value as the web editor's, so one GitHub App serves both. Set together with `GITHUB_OAUTH_CLIENT_SECRET`.
`GITHUB_OAUTH_CLIENT_SECRET`	none	The matching client secret.
`GITHUB_WEB_URL`	`https://github.com`	Where GitHub sign-in sends readers, for GitHub Enterprise Server. Must be a URL.
`OPEND_GITHUB_COLLABORATOR_ORG`	off	An org slug. A GitHub user with write, maintain or admin access to the site's repository, whose primary email GitHub has verified and no reader uses yet, becomes a reader of this org on first sign-in. Needs GitHub sign-in and `GITHUB_APP_ID`, and the repository registered with its installation id. The org must exist.
`OPEND_GITHUB_COLLABORATOR_STAFF`	off	Flag. Readers added by `OPEND_GITHUB_COLLABORATOR_ORG` are staff: they see everything, conditions marked. Needs `OPEND_GITHUB_COLLABORATOR_ORG`.
`OPEND_DEV_LOGIN`	off	Flag. Lists every registered reader under **Demo readers** on the sign-in page, where anyone can sign in as any of them, for evaluation without an identity provider. Refused when `NODE_ENV=production`. Staff can turn it off (and on again) in the admin console, but never on without this flag.

Variable	Default	Meaning and rules
`EDITOR_PUBLIC_URL`	none	The web editor's address, such as `https://edit.docs.example`. Signed-in staff readers get an **Edit this page** link on topic pages, opening the topic's source file in the editor. The editor still decides who may edit. Must be a URL. Without it, the address set in the admin console (**Editor**) is used.
`OPEND_SOURCE_LINKS`	off	Flag. Adds **View source on GitHub** to topic pages of a repository cloned from `https://github.com`, pointing at the built commit. Every viewer sees it, anonymous visitors of a public site too, so turn it on only when the repository is public or every reader may read it.

Blob storage

Variable	Default	Meaning and rules
`BLOB_DIR`	`./data/blobs` (the image sets `/data/blobs`)	Folder for images, themes and PDFs when no bucket is configured. Web and worker must share it.
`S3_BUCKET`	none	Bucket name. Setting it switches to S3-compatible storage, and then `S3_ENDPOINT`, `S3_ACCESS_KEY_ID` and `S3_SECRET_ACCESS_KEY` are required too: all four or none.
`S3_ENDPOINT`	none	Service URL, such as `https://s3.us-east-1.amazonaws.com`. Must be a URL.
`S3_ACCESS_KEY_ID`	none	Access key id.
`S3_SECRET_ACCESS_KEY`	none	Secret access key.
`S3_REGION`	`auto`	Region. Most S3-compatible services accept `auto`.
`S3_FORCE_PATH_STYLE`	off	Flag. Address the bucket as `/<bucket>/<key>` instead of `<bucket>.<host>`, for MinIO and similar services.
`S3_PREFIX`	none	Prepended to every key, such as `opend/`, to share a bucket.

Embedded help

Variable	Default	Meaning and rules
`OPEND_EMBED_SECRET`	none	Shared secret host apps sign embed tokens with. At least 32 characters. Set together with `OPEND_EMBED_ORIGINS`; without both, embedding is off.
`OPEND_EMBED_ORIGINS`	none	Comma-separated origins allowed to frame the help panel and call the JSON API, such as `https://console.example.com,http://localhost:5173`. Each must be `http` or `https` with no path or trailing slash.
`OPEND_MCP_URL`	none	The address of the managed MCP connector (one for every site), shown on the admin console's **API & AI** page so staff can connect AI assistants. Managed hosting sets it; see Public API and AI assistants.

Fleet mode

Read only by `opend-server fleet serve` and `opend-server fleet worker`, which run many sites from one deployment (Run many sites from one deployment). The fleet also reads `DATABASE_URL`, `PORT`, `HOST`, `BLOB_DIR` and the `S3_*` variables for itself and every site. Each site's other settings come from its registry entry, not from the environment.

Variable	Default	Meaning and rules
`OPEND_FLEET_KEY`	none (required)	Seals each site's settings in the registry. At least 32 characters. Changing it makes stored sites unreadable.
`OPEND_FLEET_TOKEN`	none (required)	Bearer token for the fleet admin API. At least 32 characters.
`OPEND_FLEET_HOST`	none (required)	The fleet's own host name, where the admin API answers. No site may use it.
`OPEND_FLEET_MAX_SITES`	`200`	How many site apps (docs and editor counted separately) stay running at once. The least recently used are closed beyond it and restart on their next request.
`OPEND_FLEET_REDIRECTS`	(none)	Hosts under the fleet's domain answered with a permanent redirect instead of a site, keeping the path and query: comma-separated `host=https://origin` pairs, such as `www.opend.app=https://opend.app`.
`OPEND_FLEET_WORKER_CONCURRENCY`	`2`	Build jobs the fleet worker runs at once, across all sites. One site never runs two at a time.

Analytics and feedback

See Docs analytics and reader feedback for what is recorded.

Variable	Default	Meaning and rules
`OPEND_ANALYTICS`	on	On unless set to `0`, `false`, `off` or `no`. Off records no page views, searches or feedback and hides the "Was this page helpful?" form. The analytics admin routes still answer with what was recorded before.
`OPEND_ANALYTICS_RETENTION_DAYS`	`180`	Days to keep analytics rows: a whole number from 1 to 3650. The worker deletes older rows when it starts and every six hours after, even with analytics off. `opend-server purge-analytics` does it on demand.

PDF printing

Read by the PDF renderer when it first starts Chromium.

Variable	Default	Meaning and rules
`OPEND_CHROMIUM_PATH`	the browser Playwright installed (the image sets `/usr/bin/chromium`)	Path of the Chromium binary that prints PDFs.
`OPEND_CHROMIUM_SANDBOX`	off	Set to exactly `1` to run Chromium with its own sandbox, where the runtime allows the namespaces it needs.

Docker Compose only

`POSTGRES_PASSWORD` (default `opend`) and `OPEND_PORT` (default `3000`) are read by `deploy/docker-compose.yml`, not by the server.

Console settings, environment variables and the secret key

Which settings the admin console stores, how environment variables override them, how changes reach every service, and how stored secrets are protected.

Environment variables win

Each group of settings comes from its environment variables when they are set, and from the admin console otherwise. A group never mixes the two: setting `OIDC_ISSUER` replaces the whole OIDC setup stored in the console. The console shows such a group as **Set by environment**, read-only; remove the variables to manage it in the console.

Setting	Variables that override it
Company sign-in (OIDC)	`OIDC_ISSUER`, `OIDC_CLIENT_ID`, `OIDC_CLIENT_SECRET`, `OIDC_DISPLAY_NAME`, `OIDC_ICON`
The GitHub App (tokens, statuses)	`GITHUB_APP_ID` and `GITHUB_APP_PRIVATE_KEY`, or `OPEND_GITHUB_BROKER_URL` (managed hosting)
GitHub sign-in	`GITHUB_OAUTH_CLIENT_ID` and `GITHUB_OAUTH_CLIENT_SECRET`
Webhook secret	`GITHUB_WEBHOOK_SECRET`
Editor address	`EDITOR_PUBLIC_URL`
View source on GitHub	`OPEND_SOURCE_LINKS`
Demo readers	Only ever on where `OPEND_DEV_LOGIN` allows them; the console can turn them off

Site access, the docs repository, organizations and readers live in their own tables, as before, and the admin API changes the same records.

Changes apply without a restart

Every change sends a Postgres notification. The web service (every replica), the worker and the editor reload their settings when it arrives, and re-read them at least once a minute in case a notification is missed. Sign-in providers are rebuilt only when their own settings change.

The secret key

Client secrets, the GitHub App's private key and its webhook secret are sealed with AES-256-GCM before they reach the database, bound to their names so one cannot be swapped for another. No page, API route, log line or audit entry ever contains one; the console only says whether each is set.

The key comes from `OPEND_SECRET_KEY` when it is set. Otherwise it is derived from `SESSION_SECRET` with HKDF under a label of its own, so it is never the session key itself. Either way, every service that reads the settings needs the same value: give the worker and the editor the web service's `SESSION_SECRET` (or `OPEND_SECRET_KEY`).

> **Important:** Without `OPEND_SECRET_KEY`, changing `SESSION_SECRET` makes the stored secrets unreadable. Nothing breaks silently: the console warns about them, and the settings behave as if unset until you enter the client secret again or reconnect GitHub. Set `OPEND_SECRET_KEY` before you store secrets if you rotate session secrets.

Store blobs in a folder or an S3-compatible bucket

Built images, themes and cached PDFs live in a blob store that the web service and the worker must both reach.

About this task

By default the blob store is a local folder, `BLOB_DIR`. That only works when the app and the worker share it, as in the Compose file's `blobs` volume. When they run on different hosts, as on Railway, or with app replicas on several hosts, use a bucket.

Keys look like `repos/<repo id>/<commit>/images/architecture.svg` for files from the docs tree, and `pdf/<release id>/<hash>/<name>.pdf` for cached PDFs. Nothing is ever overwritten with different content, so no invalidation is needed.

1. Create a bucket and an access key with read and write access to it.
   AWS S3, Cloudflare R2, MinIO and Railway Buckets all work.
2. Set the same four variables on the web service and the worker.

```
S3_BUCKET=opend-blobs
S3_ENDPOINT=https://s3.us-east-1.amazonaws.com
S3_ACCESS_KEY_ID=...
S3_SECRET_ACCESS_KEY=...
```

3. Optional: set `S3_REGION` (default `auto`), `S3_FORCE_PATH_STYLE=1` for MinIO-style endpoints, and `S3_PREFIX` to share a bucket.
4. Restart both services.
   The start-up log line shows `"blobs":"s3"`.
5. Rebuild the live releases, so their images and themes are uploaded to the bucket.
   Trigger a build of each release ref's current commit with the admin API, or push.

Results

Either all four required variables are set or none are; anything in between stops the server at start-up. The bucket can stay private: readers never fetch from it directly.

Connect GitHub from the admin console

Connect GitHub creates a GitHub App for your site with every setting filled in, stores its credentials encrypted, and installs it on your docs repository, so builds, private repositories, GitHub sign-in and the web editor work without environment variables.

Before you begin

- You are a staff reader (Run your site from the admin console).
- The site's `PUBLIC_URL` is its real address: GitHub sends webhooks and sign-ins there.
- `GITHUB_APP_ID` is not set. With it, the site uses that app and the console shows it read-only.

1. Open **Admin** > **GitHub**, check the app's name (unique on GitHub, at most 34 characters), optionally give the GitHub organization that should own it, and choose **Connect GitHub**.
2. Choose **Continue to GitHub**, then **Create GitHub App** on GitHub.
   The page sends GitHub an app manifest: the webhook `$PUBLIC_URL/webhooks/github` (push events), the sign-in callbacks `$PUBLIC_URL/auth/github/callback` and, when the editor's address is set, `$EDITOR_PUBLIC_URL/auth/github/callback`, the setup URL `$PUBLIC_URL/admin/github/installed`, and the permissions in One GitHub App for the site, the editor and builds. The app is private to its owner.

   GitHub sends you back with a one-time code. The site checks that the request carries a signed state from your own session (valid for an hour), exchanges the code for the app's id, client secret, private key and webhook secret, and stores them encrypted.
3. Install the app on your docs repository when GitHub asks, and choose **Install**.
   GitHub returns to the setup URL with the installation id. The site asks GitHub, as the app, whether that installation is its own before it stores it.

   Installed on exactly one repository, the app is all the site needs: it finds `opend.config.yaml` on the default branch, registers the repository with that folder and the branches its `site` block names, queues the builds, and takes you to the **Overview**. The next step is then done for you (Set the site up from opend.config.yaml).
4. On **Docs source**, pick the repository and choose **Use this repository**.
   The repository is registered with its installation, and the first build is queued. From now on every push to a release branch or tag builds, and each commit shows an `opend/build` status.

Results

- The web service, the worker and the editor all read the stored app from the database, so one connection configures them all, and a new connection applies without a restart.
- Readers get **Sign in with GitHub**, matched by a verified email. Turn it off on the **Sign-in** page if you prefer.
- **Disconnect GitHub** removes the stored credentials. Delete the app on GitHub too if you no longer need it.
- For GitHub Enterprise Server, set `GITHUB_WEB_URL` and `GITHUB_API_URL` first.

- Adding the editor later? Add `$EDITOR_PUBLIC_URL/auth/github/callback` to the app's callback URLs on GitHub; the console lists the URLs the app needs.

The manual route (create the app yourself and set `GITHUB_APP_ID` and the other variables) still works; see One GitHub App for the site, the editor and builds.

Connect GitHub: webhooks and the GitHub App

A push webhook tells your OpenD server to build. A GitHub App adds private repositories and commit statuses.

About this task

The quickest way is **Connect GitHub** in the admin console, which creates and installs the GitHub App, webhook included, without any variables: see Connect GitHub from the admin console. This page is the manual route.

A public repository needs only the webhook. Add the GitHub App when a docs repository is private, or when you want each commit to show whether it published.

1. Set `GITHUB_WEBHOOK_SECRET` on the web service to a random string.

   > **Tip:** Generate secrets with `openssl`: `openssl rand -hex 32` prints 64 random hexadecimal characters.

2. Add a webhook to the docs repository: payload URL `$PUBLIC_URL/webhooks/github`, content type **application/json**, the same secret, and **Just the push event**.
   GitHub's ping delivery gets `{"status":"pong"}`.

3. Register the repository, with the branches and tags that publish.

   ```
   curl -X POST "$PUBLIC_URL/admin/api/repos" -H "authorization: Bearer $OPEND_ADMIN_TOKEN" \
     -H 'content-type: application/json' \
     -d '{"fullName":"acme/docs","sourceUrl":"https://github.com/acme/docs.git","releaseRefs":
   ["main","v*"],"docsPath":"."}'
   ```

   `fullName` must match the repository name GitHub sends. Pushes to other refs, branch deletions and other events are acknowledged and ignored.

4. Optional: create a GitHub App for private repositories and commit statuses.

   1. Create the App with repository permissions **Contents: Read-only** and **Commit statuses: Read and write**. You can use the App's own webhook, subscribed to push events, instead of per-repository webhooks.
      The web editor needs more permissions; see Run the web editor.
   2. Generate a private key, and set `GITHUB_APP_ID` and `GITHUB_APP_PRIVATE_KEY` on the worker (and the web service, which shares its configuration).
   3. Install the App on the docs repository and note the installation id from the installation's URL.
   4. Register the repository again with `"installationId": 12345678`.
      Send every field again: registering replaces the repository's settings, and omitted ones go back to their defaults.

Results

- The webhook checks the `X-Hub-Signature-256` signature in constant time and answers 401 to a bad one. Each delivery id is recorded, so a redelivered webhook does nothing. The signature does not cover the delivery id, so a push body is also only acted on once: an old push resent under a new delivery id cannot queue an old commit and roll a release back. To publish an older commit on purpose, request a build of it through the admin API.
- For a private repository, the worker fetches with an installation token sent as an HTTP header, never in the URL, so it cannot leak into logs. The token is only sent when the `sourceUrl` is an `https` URL on `github.com`, or on the Enterprise host that `GITHUB_API_URL` names, so register private repositories with their `https://` clone URL.
- With the App installed, every build sets an `opend/build` commit status: pending, then success (`Published 0.1`), failure with the first errors, or error when the build could not run.

- For GitHub Enterprise Server, set `GITHUB_API_URL`.

One GitHub App for the site, the editor and builds

The exact settings of a GitHub App that signs readers and authors in, lets the web editor commit and open pull requests, and sends push webhooks for builds.

Let the admin console create it

Connect GitHub in the admin console registers an app with exactly these settings through GitHub's app manifest flow, and stores its credentials encrypted, so none of the variables below are needed: see Connect GitHub from the admin console. Create the app by hand only when you prefer to keep its credentials in environment variables.

App settings

Setting	Value
GitHub App name	Any name not yet taken on GitHub, such as `Acme docs`
Homepage URL	`$PUBLIC_URL`
Callback URLs	`$PUBLIC_URL/auth/github/callback` (reader sign-in) and `$EDITOR_PUBLIC_URL/auth/github/callback` (author sign-in)
Request user authorization (OAuth) during installation	Off
Setup URL	Empty (an app made by **Connect GitHub** uses `$PUBLIC_URL/admin/github/installed`, which stores the installation)
Webhook	Active; URL `$PUBLIC_URL/webhooks/github`; secret: the value of `GITHUB_WEBHOOK_SECRET`
Repository permissions	Contents: Read and write. Pull requests: Read and write. Administration: Read-only. Commit statuses: Read and write (the editor only reads them; builds report to them). Checks: Read-only. Metadata: Read-only.
Account permissions	Email addresses: Read-only
Subscribe to events	Push
Where can this GitHub App be installed?	Only on this account

After creating it, generate a client secret and a private key, install the App on the docs repository, and note the installation id from the installation's URL. Register the repository with that `installationId`.

A pre-filled registration link

GitHub opens the registration form pre-filled from query parameters. The repository's script prints the link for your two hosts; open it while signed in to GitHub, set the webhook secret (the one value GitHub does not take from a link), and create the App.

```
node scripts/github-app-url.mjs https://docs.example.com https://edit.docs.example.com
# for an organization's App:
node scripts/github-app-url.mjs https://docs.example.com https://edit.docs.example.com "Acme
docs" --org acme
```

Which service gets which variable

Variable	Docs site (web)	Worker	Editor
`GITHUB_APP_ID`, `GITHUB_APP_PRIVATE_KEY`	Yes	Yes	Yes
`GITHUB_OAUTH_CLIENT_ID`, `GITHUB_OAUTH_CLIENT_SECRET`	Yes, for reader sign-in	No	Yes, for author sign-in
`GITHUB_WEBHOOK_SECRET`	Yes	No	No
`EDITOR_PUBLIC_URL`	Optional: **Edit this page** links	No	Required
`PUBLIC_URL`	Required	Same as web	Recommended: refuses to share the docs host

See Let readers sign in with GitHub, Run the web editor and Connect GitHub: webhooks and the GitHub App.

Make a site public or private

A private site needs every reader to sign in. A public site also serves anonymous visitors, filtered for a set of public entitlements you choose.

About this task

The access mode belongs to the registered repository and is kept in the database. Staff change it in the admin console or the admin API, unless the docs repository declares it in the `site` block of its `opend.config.yaml` (`access` and `publicEntitlements`): then the build that goes live on the newest release line applies it, and the console shows it read-only (Set the site up from opend.config.yaml). Either way, switching to public opens exactly the public set, and every cached page is dropped at once. New repositories are private.

Who	Private site	Public site
Anonymous visitor	Sent to the sign-in page	The public entitlements
Signed-in reader	Their organization's entitlements	Their entitlements together with the public ones, when both are for the same release line; otherwise only their own
Reader whose organization has no entitlements	"No access yet"	The public entitlements
Staff reader	Everything, conditions marked	Everything, conditions marked

1. Decide what anonymous visitors may see, as an entitlement set: products, features and the version whose docs they get.
   Without a public set, a public site is fail-closed: anonymous visitors see only content with no product, feature or release condition, on the newest release line.

   The version may be the keyword `latest`: the newest live release line at the time of each request, such as `0.23` with release lines by minor version. When 0.24 goes live it becomes the default at once, with no admin call, and when the newest line is retired the next one takes over. The line's name is used as the version, so content conditioned on `release` is filtered for 0.23 (that is, 0.23.0). With nothing live yet, `latest` serves nothing. Only the public set takes `latest`; organizations name a real version.

2. Switch the repository to public through the admin API.

```
H="authorization: Bearer $OPEND_ADMIN_TOKEN"
curl "$PUBLIC_URL/admin/api/repos" -H "$H"          # find the repository's id
curl -X PUT "$PUBLIC_URL/admin/api/repos/1/access" -H "$H" -H 'content-type:
application/json' \
  -d '{"access":"public","publicEntitlements":{"products":["core"],"features":
[],"version":"0.1"}}'
# or "version":"latest" to follow the newest live release line
```

   Every replica drops its cached pages at once. The next anonymous request gets the public view instead of the sign-in page.

3. To go back, send `{"access":"private"}`.
   The public set is kept for next time; send `"publicEntitlements": null` to clear it.

4. Or set the mode in a seed file, next to the repository.

```
repos:
  - fullName: acme/docs
    sourceUrl: https://github.com/acme/docs.git
    access: public
    publicEntitlements: { products: [core], features: [], version: '0.1' }
```

   A seed without `access` leaves the mode as it is.

5. Or set both in the admin console, under **Access** on the **Site & access** page.
   Type `latest` in **Release version** to follow the newest live line. See Run your site from the admin console.

Results

- **Everything follows the same rules.** Pages, navigation, search, images under `/_assets/`, the logo, the theme and PDFs are all filtered for the viewer's set. Hidden topics look missing.
- **Caching.** Pages are cached per entitlement set. Anonymous visitors share their own cache entries, apart from readers', and staff have theirs. The account controls, edit links and feedback form are added to each response after the cache.
- **Why the union only on the same line.** The public set opens one release line. A reader on another line keeps their own entitlements, so the public set never opens gated content of a line it was not meant for.
- **Feedback** needs sign-in. Anonymous page views and searches are counted without an organization. The embedded help panel is unchanged: it always uses the reader from the host app's signed token.
- **The help JSON API** answers anonymous requests on a public site with exactly this public view, with no token and no embedding set up. See Help JSON API reference.
- Anonymous visitors see **Sign in** in the header (and in the menu on narrow screens), which returns them to the same page. See Set up reader sign-in with OIDC and Let readers sign in with GitHub.

Let readers choose the version they read

The version picker beside the site title decides which version of the docs readers are served: their own, one fixed version, one they choose from a list you set or from every live release line, or no picker at all.

About this task

Each reader normally gets the version their organization is entitled to, and anonymous visitors of a public site get the version of the public set. The version picker belongs to the registered repository and is kept in the database, like the access mode: staff change it, unless the docs repository declares `versionPicker` in the `site` block of its `opend.config.yaml` (Set the site up from opend.config.yaml), which then sets it with each build of the newest release line. New repositories have it off.

Mode	Version served	Beside the site title
`off`	The reader's own	The release label, as without a picker
`reader`	The reader's own	That version, for example **Version 1.4**
`fixed`	The fixed version, for everyone	That version
`list`	The one the reader chose; at first, their own	A menu of the listed versions and the reader's own, newest first
`live`	The line the reader chose; at first, their own line (at their own version), or the newest line when theirs is not live	A menu of every live release line, newest first

1. Open **Site & access** in the admin console at `/admin/site`.
   Only staff readers can open the console. See Run your site from the admin console.
2. Under **Version picker**, choose what readers can pick: **Off**, **Reader's version**, **Fixed version**, **Choose from a list** or **Every live version**.
   Every live version (`live`) needs no list: the menu is the repository's live release lines, such as **0.23**, **0.22** and **0.21** with release lines by minor version. Publishing 0.24 adds it to the menu, and retiring a line removes it, with nothing to change here. A version is matched by its line, so a cookie or request for 0.22.1 picks 0.22. A reader on a live line keeps their own version there (0.22.3 rather than 0.22), so release conditions within the line still apply to them as before; on another line they get the line's name as their version.
3. For **Fixed version**, type the version everyone gets in **Fixed version**, such as `2.1`. For **Choose from a list**, type the versions to offer in **Versions in the list**, separated by commas, such as `2.1, 2.0, 1.4`.
   The hint under the field lists the release lines already published. You can list any version, such as a minor release within a line: content conditioned on `release` is filtered for exactly that version.
4. Select **Save version picker**.
   Every replica drops its cached pages at once, so the next page shows the new picker. The change is recorded in the audit log as `repo.versions`.
5. Or set it through the admin API.

```
H="authorization: Bearer $OPEND_ADMIN_TOKEN"
curl -X PUT "$PUBLIC_URL/admin/api/repos/1/versions" -H "$H" -H 'content-type:
application/json' \
  -d '{"mode":"list","versions":["2.1","2.0","1.4"]}'
# or {"mode":"live"}, {"mode":"fixed","version":"2.1"}, {"mode":"reader"}, {"mode":"off"}
```

6. Or set it in a seed file, next to the repository.

```
repos:
  - fullName: acme/docs
    sourceUrl: https://github.com/acme/docs.git
    versionPicker: { mode: list, versions: ['2.1', '2.0', '1.4'] }
    # or versionPicker: { mode: live }
```

   A seed without `versionPicker` leaves the picker as it is.

Results

- **Only the version changes.** Products and features still come from the reader's entitlements (and the public set on a public site), so a reader who switches to 2.0 sees 2.0's content for the products they hold. Release conditions and the release line follow the version served.

- **Listing a version opens it.** Every reader may switch to any listed version, whatever their own. On a public site that includes anonymous visitors: with no public set, they then see content gated only by release, never by product or feature. A fixed version is served to everyone in the same way.
- **A choice is only a request.** It is kept in the `opend_version` cookie by the picker's links (`/_version?v=2.0&returnTo=/page.html`) and checked on every request: a version the picker no longer offers is ignored, and the reader is back on their own. A listed version whose release line has no live release is not offered.
- **Release lines.** A version served is mapped to its release line by the repository's setting, by major or by minor version. See Release lines by major or minor version.
- **Everything follows it.** Pages, navigation, search, images and PDFs are served for the version chosen, and cached per version. The staff view moves to the chosen version's release line. The embedded help panel, token requests to the help JSON API and the public API are unchanged: they always use the reader's own version. Anonymous requests to the help JSON API on a public site follow the picker through their `version` parameter, as a visitor's choice does (see Help JSON API reference).
- **Defaulting to the newest version.** On a public site, set the public set's version to `latest` (see Make a site public or private): anonymous visitors start on the newest live line, and with `live` the menu offers the others. Each new line becomes the default the moment it goes live.
- The picker is a list of links in a `<details>` menu, so it works without scripts, and sits in the **Menu** panel on narrow screens. What readers see is described in Choose the version you read.

Release lines by major or minor version

Each release line has one live release. Lines group releases by major version, or by minor version for products whose customers are each on a specific 0.x minor.

About this task

A reader is served the live release of their own line, worked out from the version they are served: their organization's version, or the one the version picker gives them. By default a line is a major version: `2.1` and `2.4` readers both read line `2`, whose live release is the newest 2.x build. A product released as 0.21, 0.22 and 0.23 needs a line per minor version instead, or every 0.x reader gets the newest 0.x docs.

Setting	Line of release 0.23.1	A 0.22 reader gets
`major` (default)	`0`	The newest 0.x build
`minor`	`0.23`	The live 0.22 build, or "No documentation has been published for release 0.22.x yet"

1. Choose lines by minor version for the repository: in the admin console, on the **Site & access** page under **Release lines**, select **By minor version** and **Save release lines**; or through the admin API.

   ```
   H="authorization: Bearer $OPEND_ADMIN_TOKEN"
   curl -X PUT "$PUBLIC_URL/admin/api/repos/1/release-lines" -H "$H" -H 'content-type:
   application/json' \
     -d '{"releaseLines":"minor"}'
   ```

   Or give `releaseLines` when registering the repository with `POST /admin/api/repos`, or in a seed file next to the repository: `releaseLines: minor`. Or let the repository say so, in the `site` block of its `opend.config.yaml`: `site: { releaseLines: minor }`. The first build that goes live switches the lines, and the console then shows the setting read-only (Set the site up from opend.config.yaml).

   The change applies at once, in one transaction, to what is already live: each live release moves to the line its `release` belongs to. Nothing that is not live comes back by itself: after switching from `major` to `minor`, line `0` becomes the line of its live release (say `0.23`), and older minors go live again with their next build (request one with `POST /admin/api/repos/:id/builds`). Switching back merges lines, keeping the newest build of each. A build that finishes during the switch publishes to a line of the new setting. The change is audited as `repo.release-lines`,

and every replica drops its caches. The console's form does the same, audited under the staff reader's name; saving the grouping already in place changes nothing. See Run your site from the admin console.

2. Publish each minor from its own branch, for example `docs/0.23` with `releaseRefs` `["docs/*"]`, and `release: "0.23.0"` in that branch's `opend.config.yaml`.
   See Document several releases with release branches. `GET /admin/api/repos/1/releases` lists the live line of each.

Results

With the version picker

A version served through the version picker maps to its line the same way: on a `minor` repository, a reader who chooses `0.21` is served the live 0.21 build. A list offers only the versions whose line is live, so a line not yet built, or retired, is left out of the menu, and a choice of it falls back to the reader's own version. The `live` mode offers every live line with no list at all. See Let readers choose the version they read.

A public site on the newest version, with older ones to choose

For product docs published per minor, where anonymous visitors should land on the newest version and may switch to older ones, set this once:

```
curl -X PUT "$PUBLIC_URL/admin/api/repos/1/access" -H "$H" -H 'content-type: application/json' \
  -d '{"access":"public","publicEntitlements":{"products":["core"],"features":
[],"version":"latest"}}'
curl -X PUT "$PUBLIC_URL/admin/api/repos/1/versions" -H "$H" -H 'content-type: application/json'
\
  -d '{"mode":"live"}'
```

With 0.21, 0.22 and 0.23 live, anonymous visitors get 0.23 at every address, with a menu of 0.23, 0.22 and 0.21. When the 0.24 build goes live it becomes the default and joins the menu, and a retired line leaves it, with no admin call. Signed-in readers start on their own organization's line and can switch the same way. To pin the versions instead, give a version such as `"0.23"` and `{"mode":"list","versions":["0.23","0.22","0.21"]}`, and update both after each release.

Set up reader sign-in with OIDC

Readers sign in through any OpenID Connect provider. OpenD matches them to the readers you registered by verified email.

1. Register a web application with your identity provider, with the redirect URI `$PUBLIC_URL/auth/callback`.
2. Set the provider details on the web service.

   ```
   OIDC_ISSUER=https://login.example.com
   OIDC_CLIENT_ID=...
   OIDC_CLIENT_SECRET=...
   # OIDC_SCOPES=openid email profile   (the default)
   # OIDC_DISPLAY_NAME=Acme SSO          ("Sign in with Acme SSO" on the sign-in page)
   ```

3. Register each reader by email, in their organization.
   See Manage organizations, entitlements and readers.
4. Restart the web service and sign in at `$PUBLIC_URL/auth/login`.
   The sign-in page lists every configured provider: **Sign in with GitHub** (see Let readers sign in with GitHub), **Sign in with** your OIDC provider, and, with `OPEND_DEV_LOGIN` outside production, **Demo readers**. With exactly one provider and no demo readers it goes straight to that provider. The OIDC flow starts at `/auth/oidc`.

Results

How sign-in works:

- The flow is the authorization code flow with PKCE, state and nonce.
- A reader already linked to the provider's issuer and subject is signed in. Otherwise, a *verified* email that matches a registered reader links that reader to the subject on first sign-in. An unverified email never matches.
- Anyone else sees "No access" and is not signed in.
- The session is a signed, `HttpOnly` cookie holding only the reader id, valid for 12 hours. Entitlements are re-read through a short cache that is cleared as soon as an admin changes them.
- A reader whose organization has no entitlements yet sees "No access yet".
- Pages show the reader's name and **Sign out** in the header, and in the menu on narrow screens. Signing out is a form post carrying the session's CSRF token; opening `/auth/logout` only asks.
- On a private site every page sends signed-out visitors to the sign-in page, which returns them to the page afterwards. On a public site they read the public view and can sign in from the header. The return address is always a path on the same site. See Make a site public or private.

Let readers sign in with GitHub

Readers can sign in with their GitHub account, matched to a registered reader by a verified email address. The web editor's GitHub App provides the credentials.

Before you begin

A GitHub App (or an OAuth App) with the docs site's callback URL. One GitHub App can serve the docs site, the web editor and the build webhook; see One GitHub App for the site, the editor and builds.

1. Add `$PUBLIC_URL/auth/github/callback` to the App's callback URLs.
   A GitHub App takes up to ten callback URLs, so the editor's `$EDITOR_PUBLIC_URL/auth/github/callback` stays next to it. For a GitHub App, give it the account permission **Email addresses: Read-only**.
2. Set `GITHUB_OAUTH_CLIENT_ID` and `GITHUB_OAUTH_CLIENT_SECRET` on the web service: the same values as the editor's.
3. Let your own team in without registering each person: set `OPEND_GITHUB_COLLABORATOR_ORG` to an organization slug, and `OPEND_GITHUB_COLLABORATOR_STAFF=1` to make them staff.
   Then anyone with write, maintain or admin access to the docs repository becomes a reader of that organization the first time they sign in with GitHub. The server asks GitHub through the App installation, so the repository must be registered with its `installationId`, and `GITHUB_APP_ID` set. Their primary email must be verified on GitHub and not already belong to a reader. Each such reader is recorded in the audit log as `reader.provision`.
4. Restart the web service and open `$PUBLIC_URL/auth/login`.
   The sign-in page lists **Sign in with GitHub**, and any other configured provider. With exactly one provider and no demo readers, it goes straight to it.

Results

- The flow is GitHub's OAuth web flow with a random `state` and PKCE (`S256`), held in a signed, `HttpOnly` cookie for ten minutes. It asks for `read:user user:email` (OAuth Apps); a GitHub App uses its permissions instead.
- The server reads the account's addresses from GitHub and uses only verified ones, the primary first. The first that matches a registered reader links that reader to the GitHub account, and later sign-ins match the account itself, even after its email changes. Another GitHub account with the same email cannot take the reader over.
- GitHub sign-in and OIDC are separate links: a reader may use both.
- Anyone else sees "No access" and is not signed in.

Run the web editor

The web editor is an optional service on its own host. It signs authors in with GitHub or your company's identity provider, and commits for them through your GitHub App.

Before you begin

- A running OpenD server, with the docs repository registered and the GitHub App installed on it (`installationId` set). **Connect GitHub** in the admin console does both (Connect GitHub from the admin console); the manual route is Connect GitHub: webhooks and the GitHub App.

- A host name for the editor that is not the docs site's, such as `edit.docs.example.com`. Cookies ignore ports, so in production a different port on the same host is refused.

1. Give the GitHub App the permissions the editor needs, and accept the change on the installation.

Repository permission	Access	For
Contents	Read and write	Reading files, saving commits, merging the main branch in
Pull requests	Read and write	Opening, updating and merging pull requests
Administration	Read	Reading branch protection and rulesets
Commit statuses	Read (keep Read and write for build statuses)	Showing required checks
Checks	Read	Showing required checks
Metadata	Read	Checking a developer's permission on the repository

   Without Administration read the editor still works, but it cannot see the review rules, so it never merges by itself: a maintainer merges each pull request on GitHub.

2. Set up at least one way to sign in.
   - **GitHub**, for developers: create an OAuth app (or use the GitHub App's own client id and secret) with the callback URL `$EDITOR_PUBLIC_URL/auth/github/callback`. Anyone with write, maintain or admin permission on the docs repository gets an account the first time they sign in.
   - **Company OIDC**, for other authors: register a client with your identity provider, with the redirect URI `$EDITOR_PUBLIC_URL/auth/oidc/callback`. It can be the same provider as reader sign-in, but use a separate client. These authors must be added before they can sign in.

3. Set the editor's variables.
   The editor reads its own settings and checks them all at start-up. It shares `DATABASE_URL` with the other services, and with it everything the admin console stores: once GitHub is connected there, the editor uses the same GitHub App for commits and author sign-in without variables of its own. With only `SESSION_SECRET` it derives a cookie key of its own, so it never signs cookies with the docs site's key.

   The smallest set, with GitHub connected in the admin console: `DATABASE_URL`, `SESSION_SECRET` (the docs site's value), `EDITOR_PUBLIC_URL` and `PUBLIC_URL`.

Variable	Default	Meaning and rules
`DATABASE_URL`	required	The same Postgres database as the docs site.
`EDITOR_PUBLIC_URL`	required	Where authors reach the editor. Must not share a host with `PUBLIC_URL`; in production the host names must differ.
`SESSION_SECRET`	see the meaning	The docs site's value. The editor derives its own cookie key from it under a separate label (so the two never share a key), and opens the settings and secrets the admin console stored with it. Set this or `EDITOR_SESSION_SECRET`.
`EDITOR_SESSION_SECRET`	derived from `SESSION_SECRET`	Signs the editor's session cookie instead of the derived key. At least 32 characters, and not the docs site's `SESSION_SECRET`.
`OPEND_SECRET_KEY`	none	The docs site's value, when it sets one: the key stored secrets are sealed with. Without it the key comes from `SESSION_SECRET`.
`GITHUB_APP_ID`	the connected app	The GitHub App that commits and opens pull requests. Without it, the editor uses the app connected in the admin console (**Integrations** > **GitHub**) or the token broker. Set together with `GITHUB_APP_PRIVATE_KEY`.
`GITHUB_APP_PRIVATE_KEY`	the connected app	The App's private key, as for the worker.
`OPEND_GITHUB_BROKER_URL`	none	Managed hosting: the control plane's token broker. The editor asks it for `write` tokens. Set by the hosting provider, together with `OPEND_GITHUB_BROKER_SECRET`.
`OPEND_GITHUB_BROKER_SECRET`	none	This instance's bearer token for the broker.
`GITHUB_OAUTH_CLIENT_ID`	the connected app's	Turns on GitHub sign-in. Without it, the client id of the app connected in the admin console is used. Set together with `GITHUB_OAUTH_CLIENT_SECRET`.
`GITHUB_OAUTH_CLIENT_SECRET`	none	The OAuth client secret.
`EDITOR_OIDC_ISSUER`	none	Turns on company sign-in. Needs `EDITOR_OIDC_CLIENT_ID` and `EDITOR_OIDC_CLIENT_SECRET`. Managed hosting sets all three to its own sign-in service.
`EDITOR_OIDC_CLIENT_ID`	none	The editor's OIDC client id.
`EDITOR_OIDC_CLIENT_SECRET`	none	The editor's OIDC client secret.
`EDITOR_OIDC_SCOPES`	`openid email profile`	Scopes to request.

Variable	Default	Meaning and rules
`EDITOR_REPO`	`OPEND_SITE_REPO`	The registered repository to edit, such as `acme/docs`. Not needed when only one is registered.
`OPEND_SITE_REPO`	none	Used when `EDITOR_REPO` is unset.
`EDITOR_BASE_BRANCH`	`main`	The branch authors' work is based on and published to.
`PUBLIC_URL`	none	The docs site, if set: the editor refuses to start on the same host.
`PORT`	`3001`	Port to listen on. The image sets `3000`.
`HOST`	`0.0.0.0`	Interface to listen on.
`GITHUB_API_URL`	`https://api.github.com`	For GitHub Enterprise Server.
`GITHUB_WEB_URL`	`https://github.com`	Where GitHub sign-in sends authors, for GitHub Enterprise Server.
`NODE_ENV`	`development`	Set `production` for real use.

> **Tip:** Generate secrets with `openssl`: `openssl rand -hex 32` prints 64 random hexadecimal characters.

4. Start the editor.
   - **Docker Compose**: `docker compose -f deploy/docker-compose.yml --profile editor up -d` starts the `editor` service on port 3001 (or `OPEND_EDITOR_PORT`). Put it behind your reverse proxy on its own host name.
   - **Railway**: add a third service, `editor`, from the same repository and `deploy/Dockerfile`, with the start command `node dist/bin.js editor serve`, the health check `/healthz` and its own public domain on port 3000. It needs no pre-deploy step: the web service runs the migrations. See Run OpenD on Railway.
   - **Anywhere else**: run `opend-server editor serve` from the image.
5. Add the authors who sign in with their company account.

```
opend-server editor add-user writer@example.com --name "Wendy Writer"
opend-server editor add-user dev@example.com --github dev-login
opend-server editor list-users
opend-server editor disable-user writer@example.com
```

   These commands need only `DATABASE_URL`. With Compose, run them as `docker compose -f deploy/docker-compose.yml run --rm migrate editor add-user …`.

   - `add-user` registers an email address, with the name used on commits. `--github` also lets that GitHub account in, for developers without write access. Adding a disabled editor again turns them back on.
   - Company accounts are matched by verified email at first sign-in, then by the provider's subject.
   - `disable-user` stops someone signing in and ends their sessions at once.

Results

Authors sign in at `EDITOR_PUBLIC_URL`. See Edit docs in the browser for what they can do.

- **Edit this page.** Set `EDITOR_PUBLIC_URL` on the docs site too, and signed-in staff readers see **Edit this page** under each topic, opening `/edit?path=topic file` in the editor. The editor still asks them to sign in as an author. One GitHub App can serve both hosts; see One GitHub App for the site, the editor and builds.
- **Separate accounts.** Editors are not readers. A reader session never opens the editor, and an editor session grants nothing on the docs site.

- **Commits.** Each save is a commit to the author's branch `editor/<user-id>/<name>`. Authors who signed in with GitHub commit as themselves: the editor keeps their GitHub user access token, sealed with the same key as the admin console's secrets, and GitHub signs their commits (Verified). Other authors' commits are made by the App, with the author's name and email and a `Co-authored-by` line. The editor refuses to write to any other branch, including the main branch.
- **Requiring signed commits.** If branch protection or a ruleset requires signed commits on the base branch or on `editor/**`, only authors who signed in with GitHub can save and publish; authors who use company sign-in are asked to sign in with GitHub. For that, GitHub sign-in must use a GitHub App's client id and secret (the app connected in the admin console, or a `GITHUB_OAUTH_CLIENT_ID` starting with `Iv`): an OAuth app's tokens are never kept. A secret key is needed too (`SESSION_SECRET` or `OPEND_SECRET_KEY`), and the App needs Administration read so the editor can see the rule. Managed hosting signs authors in with OIDC, which carries no GitHub token, so those authors cannot save to such repositories. Tokens are kept per session, refreshed before they expire, and deleted on sign-out and by `disable-user`.
- **Preview as an org.** Any signed-in editor can preview their branch as an org that has entitlements. Each preview is tied to that editor, one commit and the org's entitlements at that moment, expires after 30 minutes, and is logged. There are no editor roles yet.
- **Audit.** Every sign-in, save, merge from the main branch, conflict resolution, publish, merge, request for a developer and preview is recorded in the `editor_audit` table, with the session, user, commit and pull request. Each author sees their own on the **Activity** page.
- **Large repositories.** The preview rebuilds the whole docs project in memory. That suits hundreds of topics; a repository too large for GitHub's recursive tree listing is refused.

Publish to a docs site someone else operates

When another team or a provider runs the OpenD server, connecting your docs repository takes a few details and one webhook.

About this task

On a managed site you are usually a staff reader yourself, and do all of this in the admin console at `/admin` (Run your site from the admin console):

- **GitHub** says **Managed by your hosting provider**. Its **Connect GitHub** button installs the provider's GitHub App on the repositories you choose; there is no app to create and no key to handle.
- Installed on one repository, that is all: the site finds `opend.config.yaml`, connects the repository and builds it, and the `site` block of the file sets up access, the version picker and release lines (Set the site up from opend.config.yaml). With several, **Docs source** lists them: pick yours and the first build starts. Pushes reach the site through the provider, already verified.
- **Custom domain** serves the site at your own domain: add the DNS records it lists, and it switches over once they have propagated (Serve your site at your own domain).
- Readers and staff sign in with **Sign in with GitHub**, which goes through the provider's sign-in service; the web editor signs authors in the same way.

Otherwise, ask the operator to do these steps for you.

1. Send the operator your repository details.
   - The repository's full name and clone URL, such as `acme/docs` and `https://github.com/acme/docs.git`.
   - The branches and tags that publish, such as `main` and `v*`.
   - The folder inside the repository that holds `opend.config.yaml`, if it is not the root.
   - Whether the repository is private. Private repositories need the operator's GitHub App installed on them.
2. Add the webhook the operator gives you, or install their GitHub App.
   The webhook sends push events, as JSON, to the site's `/webhooks/github` with a secret the operator shares with you.
3. Tell the operator which organizations should read the docs, with their products, features and release, and who their readers are.
4. Push.
   With the GitHub App installed, the commit shows an `opend/build` status: pending, then `Published 0.1`, or a failure naming the first errors.

Results

A push that fails validation never goes live; the previous version stays up. Run `opend validate` locally or in CI to catch problems first, and ask the operator of your docs site for the full list of a failed build's diagnostics.

If the operator also runs the web editor, ask them to add the authors who do not use Git. See Edit docs in the browser.

Run your site from the admin console

Staff readers run the docs site in the browser at `/admin`: who can read it, its custom domain, which repository it is built from, readers and organizations, the site's look, sign-in, the GitHub connection and the web editor. No server access or environment variables needed.

About this task

Only signed-in staff readers can open the console. Anyone else gets a `404`; visitors who are not signed in are sent to the sign-in page first. Staff readers see an **Admin** link next to their name on every docs page.

The console works without JavaScript. Every change is a form protected by a token tied to your session, is refused from other sites, and is written to the audit log as `staff:your email`. A change applies at once on every server, without a restart.

1. Become the first staff reader with a one-time setup link.
   The first time, nobody is staff yet. Whoever operates the server mints a setup link, which works once, for 30 minutes, and only while no staff reader exists:

   ```
   opend-server admin-link                    # on Railway: run it in the web service's shell
   # or, with the admin token:
   curl -X POST "$PUBLIC_URL/admin/api/setup-links" -H "authorization: Bearer
   $OPEND_ADMIN_TOKEN" \
     -H 'content-type: application/json' -d '{"minutes":30}'
   ```

   Open the link, enter your email address and name, and you are signed in as a staff reader of the `staff` organization (or your existing reader is made staff). Use the address you will sign in with from then on, with GitHub or your company account.

   On a managed site, sign in at `https://opend.app/login` instead and choose **Open admin** next to the site: the account panel signs you in to the console through `/admin/handoff`, with a link that works once, for a minute. The site's owner is made staff the first time; other people must already be staff.

   The console's **Overview** opens, with what is left to set up.

2. Connect GitHub on the **GitHub** page.
   Self-hosted sites create their own GitHub App in a few clicks (Connect GitHub from the admin console). Managed sites install the provider's app with **Connect GitHub**.

3. Choose the docs repository on the **Docs source** page.
   Pick it from the repositories the GitHub App can access, or give a public repository's `https://` clone URL. Set the branches and tags that publish (such as `main, v*`) and the folder that holds `opend.config.yaml`. **Build now** queues a build of every matching branch and tag; **Rebuild now** does it again later. Choosing another repository keeps the site's access settings and readers.

   When the GitHub App can access only one repository, the site does this step itself as soon as the app is installed, and the **Overview** says so; when it cannot (several repositories or docs folders), the **Overview** says why. See Set the site up from opend.config.yaml.

4. Decide who can read the site on the **Site & access** page.
   Settings the docs repository declares in the `site` block of its `opend.config.yaml` show **Set by opend.config.yaml** and are read-only; change them in the repository.

   Private or **Public**, and for a public site the products, features and release anonymous visitors see, chosen from those the published release declares in `opend.config.yaml`. See Make a site public or private. The site title comes from the repository and is shown read-only. The release version may be `latest`, the newest live release line. **Release lines** groups releases by major or by minor version; switching moves the live releases to their new lines at once, and older minor versions come back only with their next build (see Release lines by major or minor version). The overview's **Live releases** card links to it. **Version picker** sets what readers can choose beside the site title, including **Every live version**; see Let readers choose the version they read.

5. Optional: serve the site at your own domain on the **Custom domain** page.

On a managed site, enter a domain such as `docs.example.com` and add the DNS records the page lists at your DNS provider; it shows whether each has propagated and when it was last checked. The site keeps its own address until they are in place, then switches by itself, and the old address keeps working. **Remove domain** goes back at once. A self-hosted site's domain is its `PUBLIC_URL`, so the page only explains that. See Serve your site at your own domain.

6. Add organizations and readers on the **People** page.
   An organization's products, features and release decide what its readers see. Readers sign in with an account whose verified email is the one you give. Make teammates staff here; you cannot remove your own staff role, yourself or your own organization.

7. Optional: set the site theme and PDF template on the **Appearance** page.
   A colour preset or your brand colour, fonts, corners, reading width, colour scheme and custom CSS for the site, the help panel and PDFs, and the template every PDF prints with. See Change the site's look from the admin console.

8. Choose how readers sign in on the **Sign-in** page.
   Sign in with GitHub comes with the connected GitHub App and can be turned off. Company sign-in (OIDC) takes the issuer URL, client id and client secret from your identity provider; register `$PUBLIC_URL/auth/callback` there as the redirect URI. Demo readers can be turned off where the server allows them; they never appear in production.

9. Set up the web editor on the **Editor** page.
   Give the editor's address to turn on **Edit this page** links, choose whether **View source on GitHub** links show, and add authors who sign in with company sign-in. The editor service itself needs only `DATABASE_URL`, `SESSION_SECRET`, `EDITOR_PUBLIC_URL` and `PUBLIC_URL`; see Run the web editor.

10. Optional: under **API & AI**, create API keys for your own tools and AI assistants.
    Each key has a name, what it can do (read, propose changes, publish, builds), whose view of the docs it reads (everything, one org, or public) and an expiry. The key is shown once; revoke it here at any time. On a managed site the page also shows the address to connect AI assistants. See Public API and AI assistants.

Results

Settings set by environment variables are shown as **Set by environment** and cannot be changed in the console; see Console settings, environment variables and the secret key. Secrets you enter (client secrets) are stored encrypted and never shown again: the console only says whether one is set.

Change the site's look from the admin console

The admin console's **Appearance** page sets the theme of the docs site, the help panel and PDFs, and the template every PDF prints with. Changes apply to every page at once, with no rebuild and no change to the repository.

About this task

The repository can style the site too, with `theme` and `pdf` in `opend.config.yaml` (Theme the site, the panel and PDFs, Choose a PDF template). What you set on the **Appearance** page loads after the repository's theme and replaces its PDF settings, so it wins. A badge on each card says where the current look comes from: **Set here**, the repository, or the default.

1. Open **Admin** > **Appearance**.

2. Under **Site theme**, choose the look and select **Save theme**.
   - **Colour preset**: **Default** (OpenD's emerald, or the repository's own colours), **Ocean**, **Indigo**, **Violet**, **Teal**, **Rose**, **Amber** or **Graphite**. Each sets an accent and matching neutral greys for the light and the dark scheme, all with AA contrast.
   - **Custom accent colour**: your brand colour, in place of the preset's. Without a dark-scheme accent, a lighter tint is used in dark mode. Accents need at least 3:1 contrast against the page background; text on buttons picks white or black, whichever reads better.
   - **Body font** and **Heading font**: system sans-serif, humanist sans-serif, serif, rounded or monospace, from the fonts on the reader's device. No web fonts are loaded.
   - **Corners** (square, default or round), **Reading width** (narrow, default, wide or full width) and **Colour scheme** (follow the reader's system, always light or always dark).

- **Custom CSS**: rules loaded last on every page, in the help panel and in PDFs, up to 50,000 characters. Override the `--opend-*` properties listed in Theme the site, the panel and PDFs, or style the page's classes.

Pages link the theme as `/_site-theme.css`, with its version in the address, so browsers fetch the new one at once. **Reset the theme** goes back to the repository's look.

3. Under **PDF template**, choose a template and select **Apply template**.
   Standard, **Report**, **Compact** or **Handbook**; see Choose a PDF template. Applying a template replaces the PDF settings with its own and keeps your print CSS.
4. Adjust the **PDF settings** and select **Save PDF settings**.
   Page size, font, orientation, margins, text size, cover and cover note, contents, chapter breaks, page numbers, running header and footer, and **Print CSS** loaded last in every PDF. PDFs also follow the site theme, always in the light scheme. **Reset the PDF template** goes back to the repository's `pdf` settings.
5. Check the result with the links under **Check the result**: the docs site and each publication's PDF, as you see them. Every PDF is printed again after a change, the first time someone downloads it.

Results

Each save is audited as `settings.appearance.theme` or `settings.appearance.pdf`. Tools can read and set both with `GET` and `PUT /admin/api/appearance`; see Admin API reference.

Serve your site at your own domain

On a managed site, staff set a custom domain such as `docs.example.com` on the admin console's **Custom domain** page and add the DNS records it shows. A self-hosted site's domain is its `PUBLIC_URL`.

About this task

On a managed site the hosting provider attaches the domain and switches the site over once your DNS is ready. Until then the site keeps its own address, so sign-in and links keep working, and the site's own address (such as `acme.opend.app`) keeps working afterwards too. Owners can do the same from their account panel at `https://opend.app/account`.

A subdomain such as `docs.example.com` is the easiest choice: it needs one `CNAME` record. A domain belongs to one site at a time.

1. Open **Admin** > **Custom domain**.
   The page is also linked from the **Overview** and from **Site & access**. Only staff readers can open it.
2. Enter the domain, without `https://`, and select **Add domain**.
   The status is **Being attached**. In a few minutes it changes to **Waiting for DNS** and the page lists the records to add.
3. Add each record at your DNS provider, exactly as shown.
   Your DNS provider is where you bought the domain, or where its DNS is hosted. The table gives each record's type, name and value; click a name or value to select it for copying. There is a `CNAME` that points the domain at the site and a `TXT` record named `_railway-verify.`*`your domain`* that proves you own it.

   The **Propagated** column shows **Pending** until the provider sees a record, then ✓ **In place**. **Last checked** says when it last looked; it checks every few minutes.
4. Wait for the records to propagate.
   Usually minutes, sometimes a few hours. Nothing else is needed: the site switches to the domain by itself.

   The status is **Live**. Links, sign-in and the editor use the new domain; the site's own address keeps working, so existing links still reach it.

Results

To use another domain, enter it and select **Change domain**: the new one starts again with its own records, and the site stays at its own address until they are in place. **Remove domain** puts the site back on its own address at once;

delete the DNS records afterwards. Keep the records while the domain is live: removing them takes it offline.

Needs attention means the provider could not attach the domain. Check its spelling and that no other service uses it, then remove it and add it again; if it keeps failing, ask the operator of your docs site. A suspended site cannot change its domain until it is running again.

What to do next

Self-hosted sites

A site on your own server has no provider to attach domains (no `OPEND_HOSTING_URL`): the **Custom domain** page shows its address and explains this, with no form. To move it, point the domain at your server, serve it with HTTPS from your reverse proxy or hosting platform, set `PUBLIC_URL` to the new address and restart. Register the new sign-in callback with your identity provider and GitHub App first. See Environment variable reference.

Choose a PDF template

A PDF template sets the page size, margins, text size, cover, contents and running header and footer of every publication. Pick one of four built-in templates with `pdf` in `opend.config.yaml`, and change any of its settings.

Built-in templates

Template	What it gives you
`standard`	The default. A4 portrait, normal margins, 10.5 pt text, a title page with the title in the accent colour, contents, each chapter on a new page, the publication title in the header and page numbers in the footer.
`report`	US Letter with wide margins, a banner title page (a full-width band in the accent colour), the site title in the header and `{title} · Release {release}` in the footer.
`compact`	Dense A4 for printing: narrow margins, 9.5 pt text, no title page (a title block opens the first page) and chapters that run on.
`handbook`	Pocket-sized A5 in a serif face, with a plain left-aligned title page, no header and the publication title in the footer.

Settings

Setting	Values	What it does
`template`	`standard`, `report`, `compact`, `handbook`	The built-in template the other settings change. Default `standard`. Every other setting defaults to the template's own value.
`pageSize`	`A4`, `Letter`, `Legal`, `A5`	The paper size.
`orientation`	`portrait`, `landscape`	Which way up the page is.
`margins`	`normal`, `narrow`, `wide`	Page margins: 20/18 mm, 14/12 mm or 26/25 mm (top and bottom / sides).
`fontSize`	`small`, `normal`, `large`	Body text of 9.5, 10.5 or 11.5 points.
`font`	`default`, `system`, `humanist`, `serif`, `rounded`, `mono`	The body font. `default` follows the site theme. Only fonts installed on the server that prints the PDF are used.
`cover`	`classic`, `banner`, `minimal`, `none`	The title page. `none` puts a title block at the top of the first page instead.
`coverNote`	text	An extra line on the cover, such as `Confidential`.
`toc`	`true`, `false`	Print the contents after the cover.
`chapterBreaks`	`true`, `false`	Start every top-level chapter, and the first page after the contents, on a new page.
`header`	text	Text at the top of every page; empty for none. `{title}` (the publication), `{site}` (the site title), `{release}` and `{date}` (when the PDF was printed) are filled in.
`footer`	text	Text at the foot of every page, before the page number, with the same placeholders.
`pageNumbers`	`true`, `false`	Print `page / pages` at the foot of every page.

Example

```
pdf:
  template: report
  pageSize: A4          # the report layout, on A4 paper
  coverNote: Confidential: for customers only
  footer: '{title} · {date}'
```

Quote text that starts with `{`: unquoted, YAML reads it as a mapping. A setting the template does not know, or a value it does not allow, is reported as `invalid-config`. Print CSS goes in the project `theme`; see Theme the site, the panel and PDFs.

On a hosted site, staff can choose a template on the admin console's **Appearance** page instead. Saved there, it replaces these settings for every release until it is reset; see Change the site's look from the admin console.

Readers and administration

Manage organizations, entitlements and readers

An organization's entitlements decide what its readers see. Load them from a seed file, or change them through the admin API.

About this task

- An **organization** has a slug and a name.
- Its **entitlements** are the products and features it holds and the version it runs, exactly the fields of a preview context. Without entitlements, its readers see "No access yet".
- A **reader** belongs to one organization and is identified by email. Sign-in links the email to the identity provider's account the first time.
- A **staff** reader (you and your team) sees everything the build publishes, on the newest release line or their organization's, with each condition marked, as in the previewer. Content removed at build time stays removed. Only the admin API (`PUT /admin/api/readers/:id/staff`) and seed files set it: entitlement sync from billing or a CRM never grants or revokes it, and a reader cannot set it.

1. Write a seed file.

```
repos:
  - fullName: acme/docs              # owner/name as GitHub sends it
    sourceUrl: https://github.com/acme/docs.git
    # installationId: 12345678       # GitHub App installation, for private repos
    releaseRefs: [main, 'v*']        # branches and tags that publish (default [main])
    docsPath: .                      # docs root inside the repository (default .)

orgs:
  - slug: acme
    name: Acme Corp
    entitlements:
      products: [core]
      features: []
      version: '0.1'                 # picks the 0.x docs
    readers: [ann@acme.example]
  - slug: globex
    name: Globex
    entitlements:
      products: [hosted, embed]
      version: '0.1'
    readers:
      - email: gus@globex.example
        name: Gus
  - slug: docs-team                  # your own team
    name: Docs team
    readers:
      - email: owner@example.com
        staff: true                  # the unfiltered staff view
```

2. Load it with the `seed` command.

```
opend-server seed /seed/seed.yaml
# with Docker Compose:
docker compose -f deploy/docker-compose.yml run --rm migrate seed /seed/seed.yaml
```

Everything is upserted in one transaction, so the file can be applied again after each change. Seeding never deletes anything.

3. For day-to-day changes, use the admin API.

```
H="authorization: Bearer $OPEND_ADMIN_TOKEN"
# Change what an organization may see
curl -X PUT "$PUBLIC_URL/admin/api/orgs/acme/entitlements" -H "$H" -H 'content-type:
application/json' \
  -d '{"products":["core","embed"],"features":[],"version":"0.1"}'
# Add a reader, or move one to another organization
curl -X POST "$PUBLIC_URL/admin/api/readers" -H "$H" -H 'content-type: application/json' \
  -d '{"orgSlug":"acme","email":"bo@acme.example","name":"Bo"}'
# Give a reader the staff view (reader id from GET /admin/api/readers)
curl -X PUT "$PUBLIC_URL/admin/api/readers/7/staff" -H "$H" -H 'content-type:
application/json' \
  -d '{"staff":true}'
```

Results

Entitlement changes take effect at once on every replica: the server is notified through Postgres and clears its caches. Every change through the API or a seed is written to the audit log. Values are tokens (letters, digits, `_`, `.`, `:`, `-`), and the version is a version number such as `0.1`. Use the same product and feature values the docs declare in `opend.config.yaml`.

Admin API reference

Every route of the admin API: docs repositories, builds, organizations, readers, the audit log and docs analytics.

Basics

- Base path `/admin/api`, JSON in and out.
- Every request needs `Authorization: Bearer $OPEND_ADMIN_TOKEN`. A missing or wrong token gets 401 `{"error":"unauthorized"}`. The token is compared in constant time.
- Without `OPEND_ADMIN_TOKEN`, none of these routes exist.
- An invalid body gets 400 `{"error":"invalid request","issues":[...]}`, one issue per problem.
- Every change is written to the audit log with the actor `admin-token`.

Staff readers can do most of this in the browser instead, in the admin console at `/admin`; see Run your site from the admin console. Its changes are audited with the actor `staff:email`.

Repositories

Route	What it does
`GET /admin/api/repos`	Lists registered repositories: `id`, `fullName`, `sourceUrl`, `installationId`, `releaseRefs`, `docsPath`, `access`, `publicEntitlements`, `versionPicker`, `releaseLines` and `declared`: the `site` block of the repository's `opend.config.yaml` as last applied (`{settings, ref, sha, release, appliedAt}`), or `null`.
`POST /admin/api/repos`	Registers a repository, or replaces the settings of the one with the same `fullName`. An optional `releaseLines` (`major` or `minor`) works as `PUT …/release-lines`; leaving it out keeps the current setting. 201 with the repository.
`PUT /admin/api/repos/:id/access`	Makes the repository's site private or public. Body `{"access": "public", "publicEntitlements": {"products": [], "features": [], "version": "2.1"}}`. Leaving out `publicEntitlements` keeps the current set; `null` clears it. Every cached page is dropped at once, on every replica. 200 with the repository; 404 `{"error":"no such repo"}`. See Make a site public or private.
`PUT /admin/api/repos/:id/release-lines`	Groups the repository's releases into lines by `major` (the default) or `minor` version. Body `{"releaseLines": "minor"}`. Live releases move to their new lines at once; where several land on one line, the newest build stays live. 200 with the repository; 404 `{"error":"no such repo"}`. See Release lines by major or minor version.
`PUT /admin/api/repos/:id/versions`	Sets the site's version picker. Body `{"mode": "off"}`, `{"mode": "reader"}`, `{"mode": "fixed", "version": "2.1"}` or `{"mode": "list", "versions": ["2.1", "2.0"]}` (1 to 50 versions such as `1.4`). Every cached page is dropped at once, on every replica. 200 with the repository; 404 `{"error":"no such repo"}`. See Let readers choose the version they read.
`GET /admin/api/repos/:id/releases`	The live release for each release line: `releaseId`, `releaseLine`, `release`, `sha`.
`DELETE /admin/api/repos/:id/releases/:line`	Takes a release line off the site, for example one left over from a test build. Readers of that line see that nothing is published, and the staff view and the API fall back to the next newest line. The stored release is kept, and the next successful build for the line puts it back.
`GET /admin/api/export`	Everything about the site that is not rebuilt from Git, as one JSON document for moving it to another deployment: the repository, orgs, entitlements, readers, staff, editor accounts and working branches, settings, sealed secrets, API key hashes and the audit log. Releases, builds, search indexes and analytics are left out. Treat the file as a secret.
`POST /admin/api/import`	Loads an export into a site that has none of that data yet (409 otherwise). Rows keep their ids. Sealed secrets only open if this deployment uses the same `OPEND_SECRET_KEY` (or `SESSION_SECRET`) as the one exported from. Then request builds, or let the poller run, to publish releases.

Body field	Default	Rules
`fullName`	required	`owner/name`, as GitHub sends it in webhooks. Matched without regard to case.
`sourceUrl`	required	The URL the worker fetches from, such as `https://github.com/acme/docs.git`. Must be an `https://` URL; when `NODE_ENV` is not `production`, an absolute local path or a `file:` URL is also accepted. The installation token is only sent to `github.com`, or to the host that `GITHUB_API_URL` names.
`installationId`	`null`	The GitHub App installation id: a positive integer, or `null`.
`releaseRefs`	`["main"]`	Branch and tag names that publish. Not empty. `*` matches any run of characters.
`docsPath`	`.`	The folder holding `opend.config.yaml`, inside the repository: no leading `/` and no `..`.

Omitted fields are reset to their defaults when a repository is registered again. The access mode and the version picker are not among them: only `PUT /admin/api/repos/:id/access` and `PUT /admin/api/repos/:id/versions` (or a seed file) change them.

Settings the repository declares in the `site` block of its `opend.config.yaml` are its own (Set the site up from opend.config.yaml): a request that would change one gets 409 `{"error":"set by opend.config.yaml (site: access); change it in the docs repository"}`, naming the fields. Sending the value already in place is not a change. Registering the same repository and folder again keeps declared release refs, and a seed file leaves declared fields as they are.

Builds

Route	What it does
`POST /admin/api/repos/:id/builds`	Queues a build of one commit. Body `{"ref": "main", "sha": "<40 hex characters>"}`. 202 with the job; 404 `{"error":"no such repo"}`.
`GET /admin/api/builds`	Recent builds, newest first. Query `repoId` to filter, and `limit` (default 50, at most 200).
`GET /admin/api/builds/:id`	One build, with its diagnostics. 404 `{"error":"no such build"}`.

A build has `id`, `repoId`, `ref`, `sha`, `status` (`queued`, `running`, `succeeded` or `failed`), `attempts`, `error`, `diagnostics`, `releaseId`, `createdAt` and `finishedAt`. Each diagnostic has the `severity`, `code`, `message`, `file`, `line` and `column` described in Validation diagnostics.

Organizations and entitlements

Route	What it does
`GET /admin/api/orgs`	Lists organizations with their `products`, `features` and `version` (`null` without entitlements).
`POST /admin/api/orgs`	Creates or renames an organization. Body `{"slug", "name", "entitlements"?}`. 201 `{"slug"}`.
`PUT /admin/api/orgs/:slug/entitlements`	Replaces the entitlements. Body `{"products": [], "features": [], "version": "0.1"}`. 200 with the result; 404 `{"error":"no such org"}`.
`DELETE /admin/api/orgs/:slug`	Deletes the organization, its entitlements and its readers. 204; 404 if there is no such organization.

A slug is 1 to 63 lowercase letters, digits and dashes, starting with a letter or digit. `products` and `features` default to empty lists and hold tokens; `version` is required and looks like `0.1`, with an optional leading `v`.

Readers

Route	What it does
`GET /admin/api/readers`	Lists readers: `id`, `orgId`, `orgSlug`, `email`, `name`, `staff`, and the linked `oidcIssuer` and `oidcSubject`.
`POST /admin/api/readers`	Adds a reader, or moves an existing email to another organization. Body `{"orgSlug", "email", "name"?, "staff"?}`; leaving out `staff` keeps it. 201 with the reader; 404 `{"error":"no such org"}`.
`PUT /admin/api/readers/:id/staff`	Grants or revokes the staff view. Body `{"staff": true}`. Takes effect on the reader's next request. 200 with the reader; 404 `{"error":"no such reader"}`. See Manage organizations, entitlements and readers.
`DELETE /admin/api/readers/:id`	Removes a reader. 204; 404 `{"error":"no such reader"}`.

Emails are unique without regard to case.

Audit log

Route	What it does
`GET /admin/api/audit`	The latest 100 entries, newest first: `at`, `actor`, `action` and `detail`.

Actions are `repo.upsert`, `repo.access`, `repo.versions`, `build.request`, `org.upsert`, `entitlements.set`, `org.delete`, `reader.upsert`, `reader.staff`, `reader.delete` and, from the `seed` command, `seed.apply`. The admin console adds `settings.oidc`, `settings.signin`, `settings.editor`, `settings.appearance.theme`, `settings.appearance.pdf`, `github.connect`, `github.install`, `github.disconnect`, `editor.user`, `admin.setup-link` and `admin.setup`. Details never hold a secret.

Admin console settings and setup links

Route	What it does
`GET /admin/api/settings`	The settings stored by the admin console (`values`), which stored secrets are set (`secrets`, as `"set"`; never their values), any that this key cannot decrypt (`unreadable`), where each group of settings comes from (`sources`: `env`, `console`, `broker` or `none`) and the GitHub mode.
`POST /admin/api/setup-links`	A one-time link that makes its visitor the first staff reader. Body `{"minutes": 30}` (5 to 1440, default 30). 201 `{"url", "expiresAt"}`; 409 once a staff reader exists. See Run your site from the admin console.
`GET /admin/api/appearance`	The site theme and PDF template set on the console's **Appearance** page: `{"theme", "pdf", "version"}`, each `null` when not set (the repository's look applies). See Change the site's look from the admin console.
`PUT /admin/api/appearance`	Sets either or both. Body `{"theme": {"preset": "ocean", "accent": "#0b6bcb"}, "pdf": {"template": "report", "toc": false}}`: a field left out stays as it is, `null` resets it. `theme` takes `preset`, `accent`, `accentDark`, `font`, `headingFont`, `radius`, `width`, `colorScheme` and `customCss`; `pdf` takes the settings in Choose a PDF template plus `customCss`, and the named template fills in the rest. 200 with the new state; 400 for an unknown value or an accent too faint to read.
`GET /admin/api/api-keys`	Lists public API keys: `id`, `name`, `prefix`, `scopes`, `audience`, `orgSlug`, creation, expiry, last use and revocation times. Never the token. See Public API and AI assistants.
`POST /admin/api/api-keys`	Creates a key. Body `{"name": "Support bot", "scopes": ["read"], "audience": "org", "org": "acme", "expiresInDays": 90}`: `scopes` from `read`, `write`, `publish`, `builds`; `audience` `staff`, `org` (with `org`) or `public`; `expiresInDays` 1 to 3650, or `null` for never (default 90). 201 with the key and its `token`, shown only here.
`DELETE /admin/api/api-keys/:id`	Revokes a key at once. 204; 404 when there is no such active key.

Analytics and feedback

Route	What it does
`GET /admin/api/analytics/summary`	Totals, views by surface, top pages, top searches, searches with no results, and the share of "helpful" answers per page, least helpful first. `enabled` says whether recording is on.
`GET /admin/api/analytics/feedback`	Feedback answers, newest first: `at`, `topicId`, `helpful`, `comment`, `surface`, `orgSlug` and `releaseLine`.
`GET /admin/api/analytics/feedback.csv`	The same list as a CSV download, for a spreadsheet.

Query	Default	Rules
`days`	`30`	Look back this many days, 1 to 3650.
`limit`	`20`; `100` for feedback	Rows per list: at most 1000 in the summary, 10000 for feedback.
`org`	all	Only this organization's readers, by slug.
`topic`	all	Feedback only: one topic id, such as `topics/install`.
`comments`	`0`	Feedback only: `1` lists only answers with a comment.

Comments are reader text: the JSON returns them as written, and the CSV quotes every field and prefixes values that start with `=`, `+`, `-` or `@` with an apostrophe, so a spreadsheet does not run them as formulas. Escape comments before showing them in any web page of your own.

Docs analytics and reader feedback

See which pages readers open, what they search for and fail to find, and which pages they find unhelpful, without recording who they are.

What is recorded

Page views

The topic id, the reader's release line and where it was read: `site` (the docs site), `embed` (the help panel in a host app) or `api` (the JSON help API). Pages a reader cannot see are not counted.

Searches

Searches from the docs site's search page (`site`), the help panel (`embed`) and the JSON help API (`api`): the query, in lower case with spaces tidied, and how many results it found. Email addresses and runs of seven or more digits are replaced with `[email]` and `[number]` before anything is stored. Empty searches are not counted.

Feedback

Whether a page was helpful, and an optional comment of up to 1000 characters.

Every row also carries the reader's organization slug and the time. No row holds a reader's email, name or id, or an IP address, so the numbers cannot be traced back to a person.

Readers who opt out

When the browser sends `DNT: 1` (Do Not Track) or `Sec-GPC: 1` (Global Privacy Control), that reader's page views and searches are not recorded. Feedback is still accepted, because the reader sends it on purpose.

To turn everything off, set `OPEND_ANALYTICS=0`. Nothing is recorded and the feedback form disappears. See Environment variable reference.

The feedback form

Every topic page on the site, and every topic in the help panel, ends with “Was this page helpful? Yes / No”. **No? Tell us what was missing** opens a comment box. The form is plain HTML and works without JavaScript; the reader returns to the same page with a thank-you message.

Only signed-in readers see the form. Answers are accepted only from the form on the reader's own page, and only for topics that reader can see; anything else gets the same “Page not found” answer, so the form cannot be used to find out which hidden topics exist.

Reading the numbers

Use the admin API (see Admin API reference):

- `GET /admin/api/analytics/summary?days=30`: top pages, top searches, searches with no results, and the share of helpful answers per page. Searches with no results show what readers expect to find and cannot; the least helpful pages come first.

- `GET /admin/api/analytics/feedback?topic=topics/install&days=90`: the answers and comments for one page, newest first. Add `comments=1` for comments only.
- `GET /admin/api/analytics/feedback.csv`: the same as a spreadsheet.

```
curl -H "Authorization: Bearer $OPEND_ADMIN_TOKEN" \
  "$PUBLIC_URL/admin/api/analytics/summary?days=30&org=acme"
```

> **Important:** Comments are written by readers. Treat them as untrusted text and escape them before showing them in a web page or chat message of your own.

How long it is kept

Rows older than `OPEND_ANALYTICS_RETENTION_DAYS` (default 180) are deleted by the worker when it starts and every six hours after. To purge now, or with a shorter period:

```
opend-server purge-analytics            # uses OPEND_ANALYTICS_RETENTION_DAYS
opend-server purge-analytics --days 30
```

Analytics rows live in Postgres with everything else, so database backups include them; see Back up and restore.

Embedded help

Context-sensitive help in your apps

Show OpenD help inside your own web app, in a side panel that opens on the help for the current screen and is filtered for the signed-in user.

The help is served live from the docs repository, through the same reader filter as the docs site. A user on release 0.1 with the embed product sees exactly the help their organization is entitled to, no more.

How the pieces fit

1. **Authors tag topics** with context keys using DITA's `resourceid`: `<resourceid appname="opend-console" appid="tenants.detail"/>`. See Tag topics with context keys.
2. **Authors map routes to keys** in `help-contexts.yaml`, so an app can just send its current path. See help-contexts.yaml reference.
3. **The app loads the widget**, tells it the current route, and opens the panel from a help button. See Embed the help widget.
4. **The app's backend signs a short-lived token** for its signed-in user, which the panel exchanges for a session. See Sign embed tokens in your backend.
5. **OpenD resolves the context** for that reader: the key, then its parents, then the app's default, never a topic the reader cannot see. See How a context resolves, and its fallback rules.

Apps that render help in their own UI can call the JSON API instead of showing the panel: see Help JSON API reference. If your docs site is public and the app needs only what any visitor sees, it calls the API anonymously: no token, no secret, no embedding to turn on.

What the panel shows

- The topic, compact, with links that stay in the panel.
- "In this section": the topic's visible children and siblings in the navigation.
- Search over the topics this reader can see, and nothing else.
- "Open in full docs", which opens the page on the docs site in a new tab.

The project theme applies in the panel too.

Turning it on

Embedding is off until you set `OPEND_EMBED_SECRET` and `OPEND_EMBED_ORIGINS` on your OpenD server. Until then nothing under `/embed` exists, and `/api/v1/help` answers only anonymous requests on a public site.

Ask the operator of your docs site to turn on embedding for your app's origins and to share the embed secret with your backend team. A public site's anonymous JSON API needs neither.

To check context help before publishing, use the previewer's help lab.

Framing, CSP and cookies

What protects the help panel inside other sites, and what your app's own Content Security Policy must allow.

Off by default

Nothing under `/embed` exists, and `/api/v1/help` accepts no token, unless both `OPEND_EMBED_SECRET` (at least 32 characters) and `OPEND_EMBED_ORIGINS` are set. Each origin is a scheme, host and optional port, such as `https://console.example.com`, with no path or trailing slash.

The one exception is the help JSON API's anonymous answers on a public site: they are what any visitor of the site reads already, so they are always on, for any origin, and need no secret. A private site answers them with 401. See Help JSON API reference.

Who may frame what

- Panel pages send `frame-ancestors` listing only the allowed origins, so no other site can frame them.
- The docs site itself sends `frame-ancestors 'none'`: it cannot be framed at all.
- The panel runs no inline script. Its policy is `default-src 'none'; script-src 'self'; img-src 'self' https: data:; style-src 'self' 'unsafe-inline'; connect-src 'self'; form-action 'self'; base-uri 'none'`, plus the `frame-ancestors` list.
- Docs pages run no scripts at all.

Your app's Content Security Policy

If your app sends a CSP, allow the docs origin for the script and the frame:

```
script-src 'self' https://docs.example.com;
frame-src https://docs.example.com;
```

Apps that call the JSON API from the browser also need `connect-src https://docs.example.com`.

The panel's session

Inside your app, the panel is a third-party frame. Its session cookie, `opend_embed`, is `HttpOnly`, `Secure`, `SameSite=None` and `Partitioned` (CHIPS): it exists only under your app's site, never as a general session for the docs site, and expires after 60 minutes. The docs site's own session cookie is separate.

Because the cookie must be `Secure`, the docs site has to be served over HTTPS for the panel to keep a session. Browsers that block all third-party cookies, even partitioned ones, cannot keep a panel session; use the JSON API there.

What a host app can and cannot do

- A host app vouches for *who* the user is. Entitlements still come from OpenD, so it cannot show a user more than their organization may see.
- A token naming an email from another organization is refused.
- Anyone with the embed secret can sign in as any registered reader. Keep it in your backend only, and rotate it by changing it on both sides.
- There is one embed secret per deployment for now.
- An app that needs only the public view of a public site needs no secret at all: it calls the JSON API anonymously.

Troubleshoot embedded help

The usual reasons the help panel stays empty, refuses to sign in, or opens the wrong topic.

The panel is blank, or the browser console says it refused to frame the page

The docs server does not list your page's origin in `frame-ancestors`, or your own CSP blocks the frame.

Cause

Your app's origin is missing from `OPEND_EMBED_ORIGINS`, or is written with a path or trailing slash.

Remedy

Add the exact origin, such as `https://console.example.com`, and restart the server. Also allow the docs origin in your app's `frame-src` and `script-src`.

Cause

Embedding is off: `/embed/opend-help.js` does not return the script.

Remedy

Set both `OPEND_EMBED_SECRET` and `OPEND_EMBED_ORIGINS`.

The panel says "Sign-in failed."

Condition

The token exchange at `/embed/session` answered 401.

Cause

The token is wrong: a different secret, an audience other than the docs site's `PUBLIC_URL` (watch for a trailing slash), a lifetime over 600 seconds, a missing `iat`, or an expired token because your server's clock is off.

Remedy

Decode the token and check each claim against Sign embed tokens in your backend. The server logs `embed token rejected` with the reason.

Cause

`getToken` rejected or returned something other than the bare token, such as JSON.

Remedy

Return the token as plain text, or unwrap it in `getToken`.

The panel says "You do not have access to this help."

Condition

The token is valid, but the exchange answered 403.

Cause

The email is not registered as a reader, or is registered in a different organization from the token's `org`.

Remedy

Register the reader in the right organization. The server logs `embed token for unknown reader or wrong org`.

The panel opens the wrong topic, or "No help here yet"

Condition

The context resolves differently from what you expected.

Cause

The route, the key or the reader's entitlements do not lead where you think.

Remedy

1. Open the previewer's help lab, preview as a reader like the user, and try the same key or path.
2. Check that the widget's `app` matches the `appname` in the topics and `help-contexts.yaml`.
3. Remember that a hidden target falls back to the parent key or the default; see How a context resolves, and its fallback rules.

The panel asks for a token on every page

Condition

The session cookie does not stick.

Cause

The docs site is served over plain HTTP, so the browser drops the `Secure` cookie, or the browser blocks partitioned third-party cookies.

Remedy

Serve the docs site over HTTPS. For browsers that block partitioned cookies, render help from the JSON API instead.

PDF publications

Per-reader filtering in PDFs

A publication is filtered with exactly the same rules as the site. Chapter conditions cover everything in the chapter.

- Conditions on a `chapter`, `part` or `topicref` apply to everything under it. The Administrator Guide's self-hosting chapter has `product="core"`, so a managed customer's copy has no such chapter at all.
- Conditions on topics and elements apply as on the site, paragraph by paragraph.
- A heading whose chapters are all hidden is left out, and so is its contents entry.
- A reader who can see nothing in a publication gets a 404, the same as for a publication that does not exist, and no download link.
- Links to topics in the same publication jump within the PDF. Links to other topics point at the docs site when the reader can see them there, and are plain text otherwise.
- Only images in topics the reader can see are embedded.

Who shares a render

On the hosted site a PDF is printed once per release and entitlement set, then cached in the blob store, so readers with the same products, features and version share one file. A new release gets new files; nothing needs clearing. Because one file serves several organizations, the hosted cover never names an organization.

In the previewer, the cover says "Prepared for" the preview reader's `org`, when the reader has one, so you can tell printouts apart. The staff view prints the first alternative of each conditional key.

PDF rendering and limits

How PDFs are printed, what the renderer needs, and what it cannot do yet.

The renderer

A headless Chromium, driven by `playwright-core`, prints the publication's self-contained HTML. One browser is launched on the first request and reused.

- Each document gets a fresh browser context with JavaScript disabled, and every network request is refused. Authored content therefore cannot make the server fetch a URL.
- At most two documents print at once; others wait. Each has 60 seconds to load and print.
- Pages are A4 with 20 mm top, 18 mm side and bottom margins.

Getting Chromium

- The server image includes Debian's Chromium and sets `OPEND_CHROMIUM_PATH=/usr/bin/chromium`.
- Outside Docker, install a browser with `pnpm exec playwright-core install chromium` in the OpenD checkout, or point `OPEND_CHROMIUM_PATH` at one.
- Chromium's own sandbox is off by default, because unprivileged containers usually cannot create the namespaces it needs. `OPEND_CHROMIUM_SANDBOX=1` turns it on where the runtime allows it.

Limits

- The contents list has links but no page numbers: Chromium does not support `target-counter()`. Use the PDF outline for page-accurate navigation.
- The header and footer also appear on the cover.
- External images (`https:` URLs) are not loaded, because the renderer refuses network requests. Keep images in the project.
- `url()` references in the theme are not resolved.
- The first request for a new release and entitlement set takes a moment, about a second for a small manual. Later requests are served from the cache.

Operations

Back up and restore

Postgres holds everything that cannot be rebuilt. Back it up regularly, and keep a copy away from the database's own disk.

About this task

- **Postgres**: release snapshots, which release is live, organizations, entitlements, readers and their sign-in links, build history and the audit log. Back this up.
- **Blob store**: images and themes, which the worker can upload again by rebuilding from Git, and cached PDFs, which are printed again on demand. Back it up if rebuilding old commits would be inconvenient.
- **The docs themselves** live in Git.

1. Dump the database on a schedule.

   ```
   # Docker Compose
   docker compose -f deploy/docker-compose.yml exec -T db pg_dump -U opend -Fc opend >
   opend-$(date +%F).dump
   # Anywhere else
   pg_dump -Fc "$DATABASE_URL" > opend-$(date +%F).dump
   ```

   On Railway, Postgres volume backups are available, but also schedule a `pg_dump` to the bucket or elsewhere so a copy lives off the volume.

2. Copy the dumps off the host, and test a restore now and then.

Results

To restore, stop the web service and the worker, restore into an empty database, then start them again:

```
pg_restore --clean --if-exists -d "$DATABASE_URL" opend-2026-10-16.dump
```

If blobs were lost too, rebuild the current commit of each release ref through the admin API so the live releases get their images back.

Upgrade OpenD

Run the new image's migrations, then replace the web service and the worker. Both are stateless, so an upgrade is a redeploy.

About this task

Images are published to `ghcr.io/desktopmachineshop/opend`: `edge` and `sha-<commit>` from `main`, and `1.2.3`, `1.2` and `latest` from release tags. Pin a version in production.

1. Back up the database.
   See Back up and restore.
2. Read the release notes for changes to settings.
3. Run the migrations with the new image.
   - Docker Compose: `git pull`, then `docker compose -f deploy/docker-compose.yml up -d --build`. The `migrate` service runs before the app and the worker start.
   - Railway: deploy as usual; the web service's pre-deploy command runs `node dist/bin.js migrate`.
   - Elsewhere: run `opend-server migrate` once before starting the new version.

Migrations are applied in order and recorded in `schema_migrations`. An advisory lock makes concurrent runs safe, and a migration that has already run is skipped.

4. Replace the web service and the worker.
5. If the release notes say so, rebuild the search index with `opend-server reindex`.
   The worker indexes every release it publishes. Releases published before an upgrade that changes the index (or before search indexing existed) are searched in memory, without stemming or prefix matching, until they are reindexed. `opend-server reindex --release <id>` rebuilds one release; the ids are in `/admin/api/repos/:id/releases`. Readers are served throughout.
6. Check `/healthz` and the start-up log line of each service.

Results

Published releases keep working: the release snapshot format is versioned, and the docs are not rebuilt by an upgrade. To pick up rendering improvements in the build itself, push or trigger a build of each release ref.

Validation diagnostics

Every problem code `opend validate`, the previewer, the web editor and the hosted build report, what it means and how to fix it.

Reading a diagnostic

```
topics/install.dita:14:9: error [broken-link] link target not found: setup.dita
```

That is the file, line and column, the severity, the code and a message. An **error** fails validation and stops a hosted build from publishing. A **warning** does not, unless you validate with `--strict`. Codes are stable, so tools can filter on them.

Project files

Code	Severity	Meaning	Fix
`missing-config`	error	`opend.config.yaml` is not in the folder you validated.	Pass the docs root, or create the file.
`invalid-config`	error	The config is not valid YAML, or a field breaks its rules. The hosted build also reports it when `release` is not a version number.	Follow the message, which names the field. See opend.config.yaml reference.
`missing-ditaval`	error	The `buildFilter` file does not exist.	Fix the path, which is relative to the docs root.
`invalid-ditaval`	error	The DITAVAL root is not `<val>`, or a `prop` has an unknown `action`.	Use `include`, `exclude`, `flag` or `passthrough`.
`missing-theme`	error	The `theme` file does not exist.	Fix the path.
`missing-logo`	error	The `logo` file does not exist.	Fix the path.
`duplicate-publication`	error	Two publications have the same file name, so only one could be downloaded at `/pdf/<name>.pdf`.	Rename one bookmap.
`invalid-preview-contexts`	error	`preview-contexts.yaml` is not valid YAML, or a reader is malformed, for example without a `version`. Reported by the CLI only.	See preview-contexts.yaml reference.
`xml-syntax`	error	A topic, map or DITAVAL file is not well-formed XML.	Look at the line and column: an unclosed tag, a stray `&` or `<` (write `&` and `<`, or use CDATA), or an unknown entity.

Maps and topics

Code	Severity	Meaning	Fix
`missing-map`	error	The root map, or a map a `mapref` points at, does not exist, or a `mapref` has no usable `href`.	Fix the path.
`map-cycle`	error	Maps reference each other in a loop. The message shows the chain.	Remove one `mapref`.
`missing-topic`	error	A `topicref` points at a file that does not exist, or that is not a `.dita` or `.xml` file.	Fix the `href`.
`unsupported-topicref`	warning	A `topicref` points at an external URL, or has a `format` other than `dita`. It is skipped.	Link to external pages from a topic with `xref` instead.
`empty-topicref`	warning	A navigation entry has neither an `href` nor a `navtitle`, so it has nothing to show.	Add one, or use `topicgroup` to group without a heading.
`not-a-topic`	error	A file a map references has no topic root element.	Make the root a topic type, or point the map elsewhere.
`missing-id`	error	A topic's root element has no `id`.	Add one; references use it.
`duplicate-id`	error	Two elements in one topic share an `id`.	Rename one, and update references to it.
`unsupported-element`	warning	An element is outside the supported subset. It renders as a plain block.	Use a supported element: see Supported DITA subset.
`conflict-markers`	error	The web editor refused a save, or a conflict resolution, because the text still has conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`), such as those in the hand-edit box when you resolve a conflict.	Keep the text you want from each side and delete the marker lines, then save again.
`unreferenced-topic`	warning	A topic file is in no map, and no published topic reuses content from it. It is usually forgotten.	Add it to a map, reuse a chunk from it, or delete it.

Conditions

Code	Severity	Meaning	Fix
`undeclared-value`	error	A filtering attribute has a value that is not declared in `opend.config.yaml`, or the attribute has no declared values at all. Readers never see that content.	Fix the typo, or declare the value under `attributes`.
`invalid-release`	error	A `release` token is not a version or range. Readers never see that content.	Use `1.4`, `2.0..`, `..1.9` or `1.2..1.9`. See Release range grammar.

Links, images, keys and reuse

Code	Severity	Meaning	Fix
`broken-link`	error	A link points outside the docs root, or at a file, topic or element that does not exist.	Fix the `href`. Same-file links take `#topic-id/element-id`.
`link-to-unpublished`	warning	A link points at a topic that is not in the site navigation, so every reader sees plain text.	Add the target to the site map (with `toc="no"` to keep it unlisted), or remove the link.
`missing-image`	error	An image file, or the file a key for an image names, does not exist.	Fix the path, relative to the topic or map.
`unresolved-key`	warning for `keyref`, error for `conkeyref`	The key is not defined in the keyspace of the map publishing this topic. A `keyref` falls back to the element's own content; a `conkeyref` pulls nothing.	Define the key, or pull the shared key map into this map or bookmap with a `mapref`.
`missing-conref`	error	A `conref` is not of the form `file#topic/element`, its file, topic or element does not exist, or a `conkeyref` key has no local `href`.	Check the ids in the chunks file.
`conref-cycle`	error	Content references pull each other in a loop. The message shows the chain.	Break the loop.
`conref-type-mismatch`	warning	An element pulls a chunk of a different element type, such as a `p` pulling a `note`. The chunk is used as it is.	Make the referencing element the same type as the chunk.
`unsupported-conref`	warning	The element uses `conrefend` or `conaction`. Only the start element is pulled.	Reference each element separately, or wrap the range in one element with an id.

Context-sensitive help

Code	Severity	Meaning	Fix
`invalid-resourceid`	error	A `resourceid` lacks `appname` (the host app) or `appid` (the context key).	Give both.
`duplicate-context`	error	Two topics claim the same app and key without conditions, so the second could never be shown.	Add conditions to tell them apart, or use a different key.
`invalid-help-contexts`	error	`help-contexts.yaml` is not valid YAML, does not match the expected shape, or has a bad route pattern.	See the help-contexts.yaml reference.
`unknown-context`	error	A route or default in `help-contexts.yaml` maps to a key that no topic declares, nor any parent of it.	Add a `resourceid` for the key, or fix the key.

Where diagnostics appear

- In the terminal, from `opend validate` and `opend preview`.
- In the previewer's banner, and at `/_opend/diagnostics.json`.
- As GitHub workflow annotations with `--format github`.
- On the hosted platform: a failing push shows a failed `opend/build` commit status with the first errors, and the full list is in the build's record from the admin API.