Kaloko help

How to set up Kaloko, walk your app with an agent, review what it built and keep a changelog and docs from the result. The pictures come from Kaloko's own accepted runs, so they show the screens as they are today; these pages are themselves living docs published with Kaloko.

Tutorials

- Your first run in ten minutes: install, sign in, walk a flow, share it and review it
- Design a flow with your agent: live HTML steps, versions and the design reference

Review and approve

- Review a run: go through the screens of a run on the canvas
- Approve or return a run: say yes to the steps that are right and send back the rest
- Review only what changed: review mode, its keys and approvals carried over
- Read what's new: the changelog with before and after pictures
- Hand over a signed acceptance record: proof of who accepted what, the evidence package, and how to check them

Walk and check

- Write a scenario: the flow, its steps and what each step must show
- Add criteria and check packs: automatic checks, questions for the evaluator, packs
- Check the health of your app: SEO, accessibility and performance apart from acceptance, adopting a check, alerts
- Walk in more browsers and on devices: the default sizes, WebKit, Firefox, phones, tablets, dark mode
- Walk Android, iOS and desktop apps: emulators, simulators, Electron, macOS and Windows
- Record a walkthrough video: recordings and traces of each step, one video of the run
- Hide personal data in captures: masks, automatic detection, fake values, the check before sharing
- Run Kaloko in CI and comment on pull requests: walk each preview deployment
- Compare with a baseline and ignore what always changes: diffs without clocks and carousels
- Check pages on a schedule (hosted runs): Kaloko's browsers open your pages every day or week
- Monitor SEO and answer engines every week: stacks from per-language sitemaps, the seo, aeo and locale packs, site checks, a weekly GitHub workflow, results for a dashboard

Team and access

- Invite people and guests: members, guests on single projects, readers from your domain
- Roles and API tokens: who may do what, tokens for agents and CI
- Set up company sign-in: sign-in domains, two-step verification, SAML, OIDC and SCIM
- Connect Jira, Linear, Slack and Teams: issues from steps and channels per project

Changelog and docs

- Publish a changelog with pictures: draft the entry, record the release, share it
- Publish living docs and catch stale sections: guides whose pictures keep up with the app
- Export docs and the changelog: Markdown, Docusaurus, VitePress, MkDocs, HTML, PDF or JSON

Reference

• CLI commands: every command of kaloko

• Scenario file: every key of a scenario

• Criteria, checks and packs: every check type and what each pack adds

• Health checks: every pack and its checks, why each matters, its severity and where it applies

• Configuration file: kaloko.config.yml

• MCP tools: what agents can do over MCP

• Supplier summary for quality assurance: one page for a supplier assessment

• Webhooks: events, their payloads and how to verify the signature

• Health API for dashboards: health counted per locale, step and check, and what changed since the last run

Explanation

• How Kaloko maps to your applications: organizations, products, projects, modules, environments and versions

• How verdicts are decided: checks, the evaluator and people

• Security and your data: what leaves your machine and who sees it

• Quality evidence for your QMS: what a record proves, signatures, traceability, how it fits ISO 9001, ISO 13485, IEC 62304, ISO 27001 and GxP, and known gaps

How Kaloko maps to your applications

One repository is rarely one app. A monorepo holds several apps, one app spans several repositories, an agency has a project per client, and every team names its environments differently. Kaloko uses a small set of concepts that you map to your situation once.

The concepts

| Concept | What it is | Typical mapping | |---|---|---| | Organization | your company or agency account | one per company | | Product | something released and versioned as a whole, with its own changelog and docs | a web shop, an admin, a mobile app, a package | | Project | a unit of work and access: who sees it and who reviews it | a client, a team, a module, a whole app | | Module | a part of a product with its own section in the docs and the changelog | checkout, account, billing | | Scenario and step | a flow and its screens | belongs to a project; may name a product and a module | | Environment | where a flow runs | local, preview, staging, production, production-cz | | Version | a released version of a product | from git tags, package.json, your release tool or kaloko release |

Products are optional

Without products, a project is the product and its changelog is the project's changelog. Most teams start there. When you need more, list products under products: in kaloko.config.yml, and scenarios name theirs with product: and module:. Nothing has to be moved.

A project can cover several products, and a product can be worked on in several projects. Docs and changelog entries refer to steps, so they know their product and module from the scenario.

Versions belong to products

An environment only says where a run happened. A run is tied to a product version, and the docs for version 2.4 show the accepted runs of 2.4. The pictures come from the environment the product names as its reference (reference_env), for example staging for an internal tool or production-cz for a Czech-only portal.

Four common set-ups

1. One app, one team. No products; the project is the app and has one changelog. This is the default.

2. A monorepo with several apps. One product per app; projects per team or per app.

3. An agency. A project per client and a product per deliverable. Client guests read only the changelog and docs of their product.

4. A large app with modules. One product with modules per team. Each team keeps its module's docs, and the release has one combined changelog with a section per module.

kaloko init proposes a product for each app it finds: every folder under apps/ with its own package.json, and the folders of your workspaces (npm, Yarn or pnpm) and packages/ that run as an app (a dev, start or serve script). Shared libraries are left out. kaloko products sync puts them on Kaloko before their first release, and Products in the app's menu lists every product you read.



The Products page · en · desktop

Removing a product

A product you no longer ship is archived, not deleted. Run kaloko products remove <product>, or open its changelog, go to the settings at the end and choose Archive product. Only an admin of the product's projects can do it (for a product of the whole organization, an organization admin).

An archived product disappears from Products, and its changelog, docs and public page answer "not found". Its releases and docs stay stored. kaloko products sync will not create it again and tells you it is archived, so also delete it from products: in kaloko.config.yml. If you change your mind, kaloko products restore <product> brings it back as it was. kaloko products list names the archived ones.

Every piece works alone

You can use only the changelog, only the docs, only one module's docs, only the pictures for a docs site you already have (kaloko export), or only the "What's new" e-mail. Nothing requires the whole set.

How verdicts are decided

Every criterion on the canvas ends as pass, partial, fail or waiting for a person. Three kinds of judge take part, in a fixed order: checks that count, an AI evaluator for what needs judgement, and people, who have the last word.

Checks that count come first

Whatever can be measured is checked in the live page while the step is captured: a text is there, a field has a label, the address after the step matches, a colour equals a design token, the page loads under a time limit. Each result keeps its evidence, such as the element that matched or the measured value, so you can see why it passed or failed. These checks run without any AI and never cost an evaluator question.

An evaluator for judgement

Criteria such as "the error message tells the person what to do" go to a semantic evaluator. The default is JEV by TypeSafe. It reads a reduced outline of the page, not the screenshot, and returns a score between 0 and 1. Above the pass threshold (0.85 by default) the criterion passes, below the fail threshold (0.15) it fails, and in between it is marked for a person to decide. The thresholds are set in kaloko.config.yml.

Criteria marked vision: true, and answers in the unsure band, get a second look from a vision evaluator that sees the screenshot. It also says in one sentence what decided it; that sentence shows as "Why" on the canvas.

The evaluator runs with your own key in any plan, or through Kaloko from the Team plan up, within a monthly number of questions. Answers are kept, so evaluating an unchanged screen again asks nothing new. Without either, these criteria stay "not evaluated" until a reviewer decides. kaloko calibrate compares the evaluator with your reviewers' verdicts and suggests thresholds.

On a site with thousands of pages you can cap the paid questions of one evaluation (evaluate.budget.calls). Ordinary steps are asked first, then the pages of each stack, starting with those that fail a check or were checked longest ago; what does not fit is "not evaluated" and waits for the next evaluation. A model of your own, such as one in Ollama or LM Studio, can answer first for free and hand only its unclear answers on to JEV. See the config file.

Triage: what kind of problem it is (experimental)

Verdicts answer your criteria. Triage answers a different question: what looks wrong on this screen, whether or not a criterion asked about it. Once an admin switches it on in Settings -> Security -> Experiments, kaloko evaluate shows every screen, and a few frames of its recording, to Clef, a model on Cloudflare. Clef names the kind of problem: overlapping elements, cut-off text, a raw translation key, a placeholder, an error, an endless spinner, a layout that jumps. The page itself adds overflow it measured.

The findings appear on the canvas under "Possible problems" and never change a verdict. A reviewer turns a right one into an ordinary comment with one click, marks it right, or says it is not a problem. Those answers tell us whether triage saves you time; Settings shows what they add up to so far. Each screen counts one question of the monthly allowance.

Health: advisory, like triage

The checks of packs (SEO, accessibility, performance, usability, console, error pages) answer whether the app is in good shape, not whether the work does what the brief asked. They are evaluated the same way, but they make up the run's health and never change a step's or a run's verdict, the review queue, carried approvals or the acceptance record. A team that wants one of them to decide adopts it as a criterion; a pack can also stay in acceptance as a whole. See Check the health of your app.

People decide

A reviewer approves a step, returns it with a note, or decides a criterion the evaluator was unsure about. At the

end they accept or return the whole run. Every decision carries a name.

• Required approvers (Team plan and up): a run counts as accepted only after the named people approved it.

• Approvals carry over. When a new run is shared, a step whose screenshots and criteria did not change since the last accepted run, and whose checks still pass, keeps the approval a person gave it, with who and when. Only the rest waits in review mode. Anyone can revoke a carried approval. A carried approval never accepts the run and never counts for required approvers.

• Agents never approve for people. Decisions made by an agent or a token are marked as theirs and do not count as a person's.



Review mode · en · desktop

Why in this order

Counting is cheap and exact, so it goes first. Judgement is expensive and sometimes wrong, so it only handles what counting cannot, and it says when it is unsure. People see the evidence and the reason next to every verdict, and their decision is the one that is recorded as acceptance.

Quality evidence for your QMS

This page is for quality, validation and release people whose team works under a quality management system: ISO 9001, ISO 13485, IEC 62304, ISO 27001, or GxP in pharma and medical devices, where the rules are strictest. Kaloko does not make you compliant or certified, and nobody has certified Kaloko against any of these. It gives you evidence: a record you can check, the data your own documents need, and an honest list of what it does and does not do, so your QMS can use it and you can validate that use your way.

What Kaloko is for here

Kaloko walks the screens of a web or mobile application against criteria you wrote, records what it saw (screenshots, page state, results), and collects what people decided about it: approvals, returns, comments and the acceptance of a run. From that it makes a record signed by the service that anyone can check later.

It is not your eQMS. It does not manage controlled documents, deviations, CAPA, training or change requests, and it does not release anything by itself. AI evaluators may suggest results, but their verdicts are advisory: the decisions that count are people's.

What Kaloko records, and what you can rely on

A record holds:

• the run, scenario and environment, with the scenario's version, the SHA-256 of the scenario file as walked and the base URL

• every criterion as it was walked (its text, not today's), its result in each language and screen size, and who assessed it: a person, a deterministic check, or an AI model (named, and marked advisory unless a person decided)

• requirement references (refs:) on the scenario, its steps and criteria

• the records the run serves and why it was walked, when the run names them (see below)

• the SHA-256 of every screenshot and of the run manifest

• decisions on steps and on the run, with the files attached to a step decision (name, size, SHA-256, who and when) and earlier decisions on the step; approvals carried over from earlier runs with where they came from, and approvals by tokens and agents listed apart from people's

• electronic signatures and the reasons for changes, when you sign in Kaloko

• the versions of the CLI, Node.js, the operating system, the browsers and the service

• results that are not acceptance criteria (for example health checks), marked as advisory

The evidence package adds the trace matrix, the run's audit trail, the screenshots and the files attached to step decisions (decisions/<step>/), with a manifest of every file's SHA-256 signed by the same key.

What you can rely on:

• It can be checked without us. kaloko signoff --verify checks a record or a package against the public key at /.well-known/kaloko-signoff.json, offline and without an account. A changed byte fails the check.

• Old records keep verifying. Retired keys stay published.

• The formats hold still. qawalk.evidence.v1 and kaloko.signoff.v1 only gain fields within a version; a breaking change makes a new version and the old one stays readable.

• People and machines are kept apart. An AI result nobody decided is marked advisory, and tokens and agents never count as people or sign.

Two ways to use it

Kaloko as the execution record, signed in your eQMS. This is the default and needs no setup. Kaloko records what was walked and who decided what; your own tooling pulls the evidence (kaloko evidence, the evidence

package or the evidence.ready webhook), turns it into your documents and your people sign those in your eQMS. Every evidence document says so plainly: acceptance is none, accepted or returned, and signing is none, so a generator can write "executed in Kaloko, approved in our eQMS" without guessing. Your system then covers the electronic signature, its meaning, the re-authentication, the link between the signed document and the evidence (keep the record's SHA-256 in it) and how long you keep the package.

Signing in Kaloko. On Business and Enterprise an admin can turn on electronic signatures in Settings -> Security, for every project or only some. A person who accepts a run (and, if you choose, approves a step) then picks what the signature means from your organization's own list, confirms their second step again, and the record shows their printed name, e-mail, the time in UTC and in their time zone, and the meaning. The signer sees the meanings in their language (Reviewed, Approved and Verified come translated; your own get a Czech label after a bar, QA release | Uvolnění QA), and the record keeps the label they saw, its id and the language. The signature also covers the decision itself: its note and the SHA-256 of the files attached to it. Changing a signed decision asks for a reason; the signature stays in the record, marked as withdrawn, with who, when and why. Tokens and agents never sign. With signatures off, nothing changes: people accept and approve as before.

Traceability

Write your requirement IDs into the scenario, as your QMS names them:

```
refs: [URS-12]
nodes:
- id: release-form
refs: [FS-3.4]
criteria:
- id: AC1
text: The batch number is shown before release
refs: [URS-12.1, RA-7]
```

Kaloko does not interpret them. The trace matrix has one row per requirement, criterion, step, language and screen size, with the screenshot and its SHA-256, the result, who assessed it and the decisions on the step and the run. It comes per run and per release, as JSON (qawalk.evidence.v1, the stable contract for generators) and CSV.

Each row also says who performs the step and in which part of the process: the step's session: (the role, such as the QC expert) and its lane (the sub-process, such as data collection). Without lanes: in the scenario, each session is a lane. So a validator can filter the matrix to one role in one sub-process, and the canvas opens on a lane with #lane=<number> (lanes count from 0) or on one role's steps with #session=<key>.

To show a step's screenshot next to the test step in your own documentation portal, the portal fetches it with a read token limited to the project: GET /api/v1/orgs/<org>/runs/<run>/steps/<step>/screen?locale=en&viewport=desktop&size=thumb&sha256=<hash>. Keep the run, the step and the screenshot's SHA-256 from the trace matrix. If the run holds a different file, the answer is 409, never a newer picture. Kaloko pages cannot be embedded in other sites, so the portal shows the image it fetched and links to the step on the canvas, where readers sign in with their own account.

One scenario, many records

A scenario is written once and walked again for each change: a dry run on local for change request CC-2026-001, a formal run on staging for the same request, later a smoke run on production for CC-2026-002. Name the record and the purpose when the run starts:

```
kaloko start --scenario qa/batch.yml --env staging --ref CC-2026-001 --purpose formal
```

In CI, KALOKO_REFS=CC-2026-001 (several separated by commas or spaces) and KALOKO_PURPOSE=formal do the same. A run takes up to 20 references of up to 80 characters: letters, digits and . _ - / : # +. The purpose is dry_run, formal or smoke; it is a label for people and portals and changes nothing about how the run is decided.

The canvas shows both in the run's header. Your portal lists the runs of a record with a read token, without parsing scenario files or run names: GET /api/v1/orgs/<org>/runs?ref=CC-2026-001 returns each run with its

verdict, environment, refs and purpose; &purpose=formal keeps the formal ones. The evidence carries them as run.refs and run.purpose, and the evidence of a release takes the same ref and purpose to keep only those runs.

How it fits your framework

The clauses are where auditors usually look for this kind of record. Your QA decides what counts as evidence.

| Framework | How Kaloko helps | What stays with you | |---|---|---| | ISO 9001 | Signed records that cannot change unnoticed, with an audit trail (7.5); verification records of design and development: runs against criteria, people's decisions, requirement refs (8.3.4); the acceptance record says which criteria, which version and who accepted (8.6) | Document control, keeping records in your own archive, the release decision itself | | ISO 13485 | Material for validating software used in your QMS: the supplier summary, a pinned CLI, our change control, tool versions in every record (4.1.6); legible, identifiable records protected by signatures and hashes (4.2.5); verification and validation results traced to requirements (7.3.6, 7.3.7) | Validating your use of Kaloko by risk, retention and archive, the verification and validation plan | | IEC 62304 | Traceability from requirement to test to result in the trace matrix; software system test records: what was tested, on which version, with which tools, when and who decided; input for the release record | The development plan, safety classification, risk management, residual anomalies and the release itself | | ISO 27001 | Acceptance criteria and their results as a record of security testing in development and acceptance (A.8.29); acceptance tied to a version, a PR and people for change management (A.8.32); a package an auditor can verify offline | The ISMS, risk assessment, Statement of Applicability and your change process | | GxP (21 CFR Part 11, EU GMP Annex 11) | See the section below | Validation (CSV or CSA), SOPs, training, archiving, and your reading of each requirement | | ISO 26262, IEC 61508 | A supporting tool: records of UI checks as additional evidence | Tool qualification and the safety case; Kaloko is not a safety verification tool on its own |

Controls in place today

• Access: roles per organization and project, restricted projects and guests; two-step verification can be required for everyone (Business); session lifetime and idle timeout (Business); company sign-in, SCIM and an IP allowlist (Enterprise). API tokens are scoped, expire and can be revoked.

• Audit trail: security-relevant actions, run verdicts, signatures, deleted step decisions and record downloads go to an append-only audit log with who, when, from where and through what (browser, token or agent). Step decisions and criterion verdicts are kept with their author and time. The run's audit trail in the evidence package combines both. Entries about a run's decisions are kept as long as the run; other entries for two years.

• Record integrity: Ed25519 signatures over canonical JSON, SHA-256 of every file, public keys published, retired keys stay published so older records keep verifying.

• Retention: on Business and Enterprise, accepted runs (and the runs you keep) stay while the plan lasts, and so do their records. Other runs expire after the plan's retention. A person with the right role can still delete a run (it stays in the trash for 7 days, and the deletion is in the audit log). Keep the evidence packages you need in your own system.

• Backups: the database can be restored to any minute of the last 30 days; files changed in the last 30 days are also copied nightly to a separate store.

• Time: service times are UTC from the service's clock, kept in sync by the hosting provider. Walk times come from the clock of the machine that walked.

Our change control

The CLI is released to npm with semantic versions and a changelog; you choose when to update and can pin a version. Data formats are versioned (qawalk.run.v1, qawalk.evidence.v1, kaloko.signoff.v1): a version only gains fields, and a breaking change makes a new version while the old one stays readable. The service is updated continuously; each record names the service version that signed it.

GxP: 21 CFR Part 11 and EU GMP Annex 11

GxP is the strictest profile Kaloko is built for. Kaloko is not a validated system and makes no claim of Part 11 or Annex 11 compliance; here is what it gives you for the requirements people ask about first.

Requirement	What Kaloko gives you	What your validation has to settle
Part 11 §11.10 (controls for closed systems)	Access by role, an audit trail with who, when and through what, authority checks on accepting and signing, accurate copies of records (JSON, PDF, the package)	Whether these controls are enough for your intended use; SOPs and training
Part 11 §11.50 (signature manifestations)	Printed name, time with time zone and the meaning on every signature in the record	Your list of meanings
Part 11 §11.70 (signature and record linking)	The signature is bound to the run and to the decision (its note and attached files by SHA-256); the signed record cannot change unnoticed	Keeping the record's SHA-256 in documents you sign elsewhere
Part 11 §11.200 (signature components)	The person's session plus a fresh second step of their own (a passkey or an authenticator app) at signing	Whether that pair meets your reading of §11.200; there are no passwords
Annex 11 §4 (validation)	The supplier summary, a pinned CLI, versioned formats and our change control	Validating your use
Annex 11 §9 (audit trails)	The audit log and the run's audit trail in the package	Reviewing the audit trail as your SOPs require
Annex 11 §12 (security)	Roles, required two-step verification, session limits, company sign-in, IP allowlist	User management and access reviews
Annex 11 §14 (electronic signature)	Optional signatures with meaning and re-authentication, withdrawn only with a reason	Deciding where you sign: in Kaloko or in your eQMS

How it maps to a validation approach:

Your activity	What Kaloko gives you
Supplier assessment	The supplier summary, the changelog and the trust page
Installation (IQ)	A pinned CLI version from npm (kaloko@x.y.z); every record names the CLI, Node.js, browser and service versions it ran with
Operation (OQ)	Your own scenario against a page you control, with known passes and known failures: deterministic checks must catch the failures, records and packages must verify, and a changed file must fail the check
Performance or UAT (PQ)	Your real scenarios with refs:, walked on the validation environment; the record or the package is the executed evidence, people's decisions are the review
Risk-based assurance (CSA)	Deterministic checks for high-risk criteria, AI criteria only as advisory support with a person deciding, the trace matrix for coverage

Known gaps

• No certification or validation of Kaloko by anyone, and no claim of compliance with any of the frameworks above.

• No SOC 2 report yet; the preparation is described on the trust page.

• Signing in Kaloko uses the person's session (e-mail link or company sign-in) plus a fresh second step of their own (a passkey or an authenticator app). There are no passwords. The signature is made by the service on the person's behalf, not with the person's own certificate, and it is not a qualified electronic signature under eIDAS.

• You cannot hold the service at a version; only the CLI can be pinned.

• The time source is not validated, and records carry no timestamp from an outside authority yet.

• Retention follows the plan: there is no legal hold, and the PDF is not PDF/A. Archive the package yourself.

• The audit log keeps entries other than decisions for two years. It only gains entries through the service, but nothing outside the service seals it; the signed package is what you can check independently.

• Editing a comment's text keeps no earlier version, and changing a decision that is not signed asks for no reason.

• Masking personal data changes what the screenshot shows. Decide whether runs you keep as evidence mask or not.

• There is no ready-made supplier audit package yet; we answer questionnaires and describe our change control, backups and restore on request.

• AI evaluators run at outside providers and their models change; their verdicts stay advisory.

Questions to settle with your QA

- Which framework your evidence has to serve, and which clauses it supports.
- Which path you take: sign in Kaloko, or sign the generated documents in your eQMS.
- Which meanings your signatures use, and whether step approvals are signed too.
- Where the evidence packages are archived and for how long.
- Which criteria may be assessed by AI at all, and who decides them.
- How you treat service updates you cannot postpone.

Security and your data

Kaloko runs where your code and your agent are: the CLI walks, captures and checks on your machine or in your CI. The service at kaloko.app stores what you share, shows the canvas and collects decisions. This page explains what travels where. Where the data is stored and which companies process it is on the trust page, which is kept up to date.

What leaves your machine

Nothing until you share. kaloko walk, evaluate and preview work locally, and the local canvas opens from disk. kaloko share uploads the run: screenshots, the captured HTML, criteria with their evidence, and recordings where your plan keeps them. Files travel by content hash, so a capture that did not change is not sent again.

Traffic data is opt-in. With traffic.gsc or traffic.ga4 in the config, the CLI asks Google's Search Console and Analytics APIs for numbers per page, straight from your machine or CI, signed in as a service account whose key you name with an environment variable. The key and the access token are never written into the run or uploaded, and are removed from any message that would show them. The answers are cached in the run folder's .cache on your machine. The numbers per page (clicks, impressions, CTR, position, sessions) end up in the stack's items.json, so they go to kaloko.app with the run when you share it. A CSV export under traffic.file is read locally like any other file.

What the evaluator sees

For text criteria the evaluator gets a reduced outline of the page with e-mail addresses and tokens masked, never the raw HTML. A screenshot goes to the vision evaluator only for criteria marked vision: true and for answers the text evaluator was unsure about. With your own key the request goes from your machine to that provider; without one, it goes through Kaloko. When you configure your own model (kind: openai_compatible), the page outline goes only to the address you wrote in base_url, such as Ollama on your own computer.

Secrets stay out

Secrets are never written in kaloko.config.yml. The config names an environment variable of your choice (password_env: STAGING_PASSWORD), and the value is read from the environment, the project .env or ~/.config/kaloko/.env. Values of those variables are removed from captured HTML, page text and console output before anything is stored. Sessions saved with kaloko auth save stay on your machine and are never uploaded.

Personal data stays hidden

Masks hide personal data in everything a step stores, not only in the screenshot: the captured HTML, the outline for the evaluators, traces, recordings and captured e-mails. With pii: redact Kaloko finds e-mails, phone numbers, birth numbers, IBANs and addresses by itself; pii: pseudonymize puts consistent fake values in their place. An environment with pii: required gets no run without it, and kaloko share refuses a run that still holds an unmasked e-mail or phone number. See Hide personal data in captures.

Production is read-only

An environment marked as production is always read-only in the CLI: public pages as a visitor, no sign-in to back offices, no data created. A scenario must declare readonly: true to run there.

Runs expire

Every shared run has an expiry date, 30 days by default, at most your plan's retention, and is deleted after it. kaloko keep keeps a run for good; on Business and Enterprise accepted runs do not expire while the plan lasts. From the Team plan up you can download a run as a ZIP at any time with kaloko export.

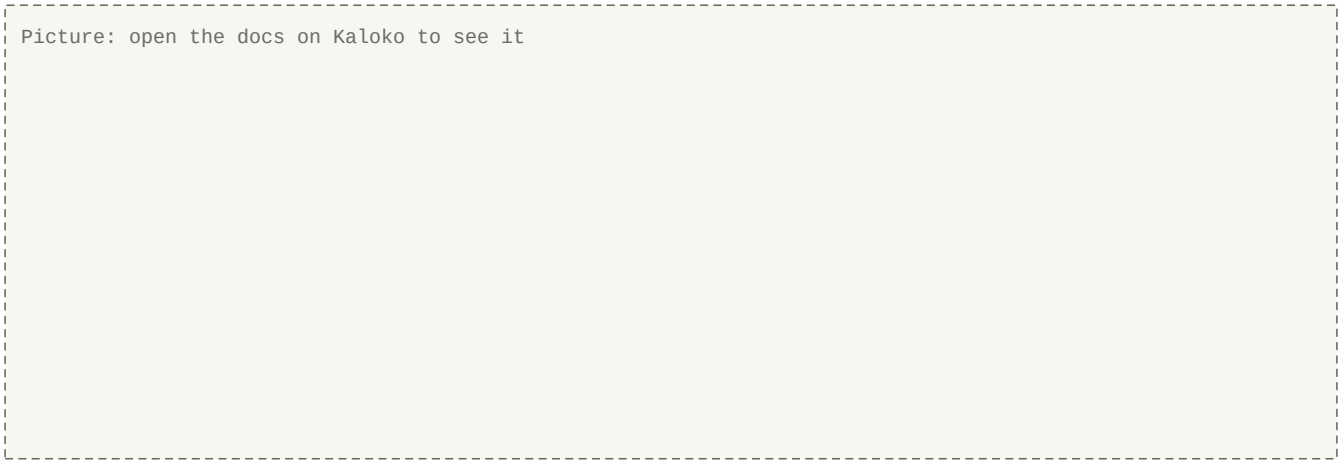
How files are served

Uploaded files are served from a separate domain under short-lived signed links. Captured HTML is shown in a sandbox without scripts.

Who gets in

People sign in with a one-time e-mail link or with Google, Microsoft or GitHub, and can add a passkey or an authenticator app as a second step. Someone who signs in with Google, Microsoft or GitHub before they belong to any organization can create one; the provider has already verified their address. A passkey counts only when it asks for a PIN, a fingerprint or the face. Enterprise adds company sign-in over SAML or OIDC and SCIM.

Roles decide what a person may do: a Viewer reads, comments, approves and returns; a Designer or a Developer / Tester also uploads; an Admin also manages members, tokens and settings. Viewers are free in every plan. API tokens carry their own scopes and expiry, and the CLI's token from kaloko login lasts at most 30 days.



API tokens in Settings · en · desktop

Questions a security review asks

The trust page answers where data lives, how it is encrypted, the list of subprocessors and where the SOC 2 preparation stands. For a vulnerability report or the data processing agreement, write to the contact given there.

Hand over a signed acceptance record

When a client or an auditor asks who accepted a release, give them the acceptance record of the run. It lists who approved what and when, the results of the criteria and a fingerprint of every screenshot, and Kaloko signs it. Records are part of the Business and Enterprise plans.

Download it

1. Open the accepted run and its Acceptance record panel.



The acceptance record of an accepted run · en · desktop

2. Choose Download PDF for people, or JSON for a system that keeps records. The PDF carries the same JSON inside it.

A run that is not accepted has no acceptance record. Accept it first, or ask the person who has to.

Sign the acceptance, if your organization asks for it

Where an admin turned on electronic signatures (Settings -> Security), accepting a run opens a short dialog: pick what your signature means, check your name, and confirm your second step if Kaloko asks for it again. The record then shows your name, e-mail, the time with your time zone and the meaning. Changing a signed decision later asks why, and the reason stays in the record. With signatures off, you accept as usual and nothing is asked.

Hand over the evidence package

For your own documentation, or when an auditor wants everything in one place, download the evidence package: open the run's Evidence panel and choose Evidence package (ZIP), or run kaloko signoff <run-id> --package. It works for any shared run, accepted or not, and holds the signed record (JSON and PDF), evidence.json, the trace matrix (trace.csv, trace.json), the run's audit trail, the screenshots and a signed list of every file with its SHA-256. Trace matrix (CSV) downloads the matrix alone. If your own tools write the final documents, read Quality evidence for your QMS for what the evidence says and what your system has to cover.

Check one you received

Anyone can check a record without an account. Developers run kaloko signoff --verify record.pdf; it checks the signature against Kaloko's public key and says whether anything was changed. If a single character of the record was edited, the check fails. The same command checks an evidence package (kaloko signoff --verify package.zip): every file must match the signed list, and a file added later is named too.

Add criteria and check packs

Each step lists its criteria: what the screen has to meet. Kaloko knows three kinds, and a good scenario uses the cheapest one that can answer the question.

Countable: a deterministic check

Anything you can count or read from the page is checked in the page itself, with the evidence attached:

```
criteria:
- id: AC1
text: The cart shows two items.
check: { type: deterministic, assert: count, selector: ".cart-item", min: 2, max: 2 }
- id: AC2
text: Paying ends on the thank-you page.
check: { type: deterministic, assert: url_matches, pattern: "/thanks$" }
- id: AC3
text: The Pay button uses the brand colour.
check: { type: deterministic, assert: style_equals, selector: "button.pay", property:
background-color, token: color.brand.primary }
```

Other checks read text, attributes, the HTTP status, overflow on a phone or a value the page exposes (js_equals). The full list is in checks and packs. A step script never has to leave a marker for a criterion to find.

A question: a semantic check

What needs judgement goes to the evaluator as a question about the page:

```
- id: AC4
text: The primary call to action is clear.
check: { type: semantic, question: Does the page show one clear primary call to action?,
scope: main }
```

The evaluator reads a reduced outline of the page and answers with a score. A clear yes passes, a clear no fails, and anything in between goes to a person. Add vision: true to the check when the look is the point (layout, imagery, "looks like the brand"): the screenshot then gets a second look from a vision model. Keep it to the steps that need it.

A person decides: a manual check

```
- id: AC5
text: Legal approved the wording of the consent.
check: { type: manual }
```

A manual criterion stays "not evaluated" until a reviewer decides it on the canvas.

Check packs

A pack adds a fixed set of checks to a step, so common checks are not written by hand. They are health checks: they run on every walk and show apart on the canvas, but they do not decide the step's verdict (the design pack is the exception). See Check the health of your app.

```
nodes:
- id: home
packs: [seo, a11y, perf, console]
```

| Pack | What it checks | |---|---| | seo | one H1, title and description, canonical, hreflang, Open Graph, indexing, html lang | | usability | purpose, one primary action, link texts, unlabelled fields | | a11y | alt texts, names, labels, headings, contrast, landmarks, the Tab walk and visible focus | | perf | LCP, CLS, TTFB and page weight measured in the walk | | console | whether console errors touch what the step is for | | errors | error pages: a real 4xx or 5xx status, a way back, no stack trace | | app | app screens: labels on controls, touch target sizes, cut-off

text, crashes | | design | an implementation against the accepted design: pixels, structure, tokens, contrast |

To change a limit, write your own criterion with the check's number as an AC id (for example AC401 for the LCP check HC401); yours wins and, being your own, it counts for acceptance. kaloko health adopt <step>:<check> writes such a criterion for you, with from: HC401 so it is clear where it came from. How verdicts come together is explained in how verdicts are decided.

Approve or return a run

Your decision is what turns "the agent says it is done" into "it is done". You decide step by step, then for the whole run.

Decide each step

1. Open a step and read its criteria. A green criterion was checked by Kaloko; one that needs a person says so, and that judgement is yours.



A step with its criteria and comments · en · desktop

2. Under the comments, choose Approve when the step is right. When it is not, write what should change and choose Reject: "the total jumps when the price loads", not "fix this". The agent reads the note and works from it.

Decide the run

1. When every step is approved, choose Accept run. The run becomes the picture of this version: the changelog and the docs show its screens from now on.

2. If something has to change first, choose Return run. Whoever shared it gets your notes; when they share the next run, the comment in the pull request points to it.



The run on the canvas · en · desktop

Next time is shorter

In the next run, steps whose screens did not change keep your approval. You only look at what is new or different, and the canvas says which approvals were carried over and from which run.

Agents can comment and suggest, but only a person accepts a run.

Walk in more browsers and on devices

By default a run captures every step in Chromium, in each locale and viewport of the scenario. You can add browser engines, device presets, dark mode and a few more settings. Every combination becomes its own capture, so add only what the task is about: a Safari bug needs WebKit on the affected steps, not every setting on every step.

Default capture setup

A new scenario (from kaloko init, kaloko draft or the template) captures two sizes:

| Name | Size | Why | |---|---|---| | desktop | 1280 × 800 | a common laptop | | mobile | 360 × 800 | the narrowest common phone width, where whatever does not fit shows first |

Every step is captured as a whole page, top to bottom. There is no tablet by default, because most tasks do not need one and every size adds captures. Numbered marks on the checked elements (capture: { annotate: true }) are off too.

To change it, edit viewports: in the scenario. A tablet goes in as tablet: [744, 1133] (an iPad mini held upright); at 900 or wider most sites already show their desktop layout. capture: { full_page: false } on a step keeps only the first screen. A scenario you already have keeps its sizes, so its baselines stay comparable; one without viewports: is captured at 1280 × 800 and 390 × 844, as it always was.

In the scenario

```
viewports: [iphone-se, iphone-15, pixel-8, ipad-landscape, laptop]
browsers: [chromium, webkit, firefox]
color_schemes: [light, dark]
reduced_motion: [no-preference, reduce] # optional
forced_colors: [none, active] # optional, Windows high contrast
display_modes: [browser, standalone] # optional, the page as an installed web app
```

Defaults for the whole project go under walk: in kaloko.config.yml. The command line wins over the scenario, and the scenario over the config.

For one run

```
npx kaloko start --scenario qa/flows/checkout.yml --env staging \
--browsers chromium,webkit --color-schemes light,dark --viewports
iphone-se,pixel-8,desktop-hd
npx kaloko walk # one pass per engine and device
npx kaloko walk --browsers webkit # only the WebKit captures of the run
```

Device presets

npx kaloko devices lists the presets: phones (iphone-se, iphone-15, pixel-8, galaxy-s24…), foldables (fold-folded, fold-open), tablets (ipad-portrait, ipad-landscape…), computers (laptop, desktop-hd, desktop-wide) and tv-1080p. Add -landscape to turn a phone, foldable or tablet sideways (iphone-15-landscape). A preset sets the size, pixel ratio, touch and the device's user agent. It catches layout bugs; a real device catches the rest.

Install the engines

WebKit (Safari's engine) and Firefox are downloaded once per machine or CI job:

```
npx kaloko browser install --browsers webkit,firefox
```

npx kaloko doctor lists the engines your scenarios need and says how to install a missing one. On Linux CI add --with-deps for the system libraries.

What to expect

• A few checks run only in Chromium: names from the accessibility tree, visible focus and layout shifts, and on WebKit also the Tab walk. On other engines they say "not evaluated" with the reason and do not make the step partial.

• Contrast is checked per colour scheme, so a dark-mode contrast bug fails on the dark capture.

• A run refuses more than 400 combinations.

On the canvas the browser, colour scheme and other settings sit next to the language switcher. The step detail of a device preset draws the device frame, and Beside: shows the same step in another browser or scheme next to it.

Check the health of your app

A client asks for a new checkout. The run comes back red because the order page has no meta description, which nobody asked for. Kaloko keeps such checks apart: the checks of packs (SEO, accessibility, performance, usability, console, error pages, app screens) make up the health of the app. They run on every walk, but they never decide whether a step or a run passes. Acceptance stays with the criteria from the brief.

| | Acceptance | Health | |---|---|---| | The question | does the work do what was asked? | is the app in good shape? | | Where the criteria come from | the brief, the ticket, people | packs, automatically | | Part of the verdict | yes | no, advisory only | | Mainly for | the person who approves | the agent before review; the team over time |

Turn it on

Health comes from the packs of a step, as before:

```
nodes:
- id: home
packs: [seo, a11y, perf, console]
```

The design pack is the exception: it checks the work against the design your team accepted, so it always counts for acceptance. Health checks lists every pack and what each of its checks looks at.

When the scenario is walked on a design environment, the packs that only code can satisfy (seo, aeo, perf, console, errors) are off: a designer's HTML has no real server, meta tags or error statuses. To keep one on, list it under health: { design_packs: [...] }. The scenario's page in the app shows which packs are on or off.

Read it

After a walk, kaloko walk and kaloko evaluate print one line with the counts. The details:

```
npx kaloko health # the current local run
npx kaloko health --remote # the shared run on kaloko.app
npx kaloko health --json # for scripts and agents
```

Each finding has a severity. Error means people or search engines are shut out: a page excluded from indexing, no viewport on phones, console errors, a button without a name, a keyboard trap. Warning is everything else, such as a title that is too long or a slow page. Unsure means the evaluator could not tell. The list also says what is new or fixed since the previous run of the scenario, and how to fix each check.

On the canvas, Health has its own tab. In the run panel, Health sits beside Summary: the counts, the change since the previous run and the findings by pack or by step; a click opens the step on its Health tab, beside Criteria, which holds the acceptance criteria only. Health checks have their own ids (HC103), so they never look like acceptance criteria (AC1, AC2…). Run lists show the counts as a small badge, and the project's Health page draws the trend over time.

--fail-on error makes kaloko health exit with 1 when there are errors. Use it only where the team wants health to stop a pipeline; nothing else in Kaloko fails because of health.

Make one check count for acceptance

Sometimes a health check is part of the brief after all ("every product page has a meta description of at most 155 characters"). Adopt it, as it is or in your own words:

```
npx kaloko health adopt product:HC103 --text "Every product page has a meta description of at most 155 characters"
```

The check is written into the scenario file as the criterion AC103 of that step, with from: HC103, and counts for its verdict from the next walk. The step's health no longer lists HC103. Older runs that still call the check AC103 accept that id too. On the canvas, Adopt as criterion on a finding does the same for people with the Developer / Tester role; the agent writes it into the scenario with kaloko health pull.

To keep a whole pack in acceptance, as it was before health existed, name it in the scenario:

```
health:
acceptance: [seo]
```

Hear when it gets worse

Hosted runs and runs shared from CI compare their health with the previous run of the scenario. When there is a new error, Kaloko tells the scenario's followers in the inbox and by mail, posts to Slack and Teams channels that take "Health got worse", and sends the webhook event health.worse. Each schedule can alert on new errors (the default), on new warnings too, or not at all: Settings -> Integrations -> Hosted runs.

Set up company sign-in

For organization admins. Kaloko works with the sign-in your company already uses: personal sign-in on every plan, an organization-wide second factor from Business, and your identity provider with SCIM on Enterprise.

Sign-in on every plan

People sign in with a one-time link sent by e-mail, or with Google, Microsoft or GitHub. Each person can add a passkey or an authenticator app as a second step in their account settings.



The sign-in page · en · desktop

On Business an admin can require the second step for the whole organization and set limits for sessions, under Settings -> Security. People without a second step are asked to add one before they continue.

Verify your domain first

Company sign-in and SCIM act only on addresses from verified domains.

1. Open Settings -> People and add your domain under Sign-in domains.

2. Add the TXT record Kaloko shows to your DNS (name _kaloko.<domain>, value kaloko-verify=…), or prove the domain by signing in with Google Workspace or Microsoft 365. A _qawalk record added before the rename still counts.

3. Choose Verify now. Until the domain is verified, nobody joins through it.

Connect your identity provider (Enterprise)

1. Open Settings -> Security -> Company sign-in.

2. Choose SAML (Okta, Entra ID, Google Workspace and others) and paste your provider's metadata, or OIDC with the issuer, client ID and secret. Kaloko's SP metadata, with its certificate, is on the same page.

3. Choose Test connection. It shows what the provider sent and whether Kaloko would accept it; nobody is signed in by the test.

4. Enforce it. People from your verified domains then sign in only through your provider. Admins with a passkey or an authenticator app keep a way in for the day the provider is down.

Optional settings on the same page:

• Encrypted assertions: create the organization's key pair; a new pair keeps the old one working until you remove it.

• Single logout: signing out at the provider signs people out of Kaloko, and the other way round.

• Sign-in from the app tile of your provider, off until you turn it on, with the page people land on.

• Your provider's second step can count as Kaloko's, for sign-ins through your provider.

Guests from other companies keep signing in as before and see only the projects they were invited to.

Provision people with SCIM (Enterprise)

Paste the SCIM base URL and token from Settings into Okta or Entra ID. Assigned people join with their domain's default role, pushed groups map to projects and roles, and removing someone ends their sessions, revokes their tokens and takes them off every project. SCIM never makes anyone an admin.

A SCIM token is valid for a year. Settings shows until when, and "Renew for a year" keeps the same token, so you change nothing in the identity provider. Admins get a reminder in the inbox 30 days before it expires.

Limit where people come from (Enterprise)

Under Settings -> Security you can allow IP ranges for the app and, separately, for the API, the CLI and MCP. The audit log can be streamed to a signed webhook, Splunk or a Datadog-compatible endpoint.

Where your data lives and who processes it: Security and data.

Compare with a baseline and ignore what always changes

Before a release you want one answer: what looks different from the version the team accepted, and is it on purpose. A good run of a scenario becomes its baseline, and every later run is compared with it.

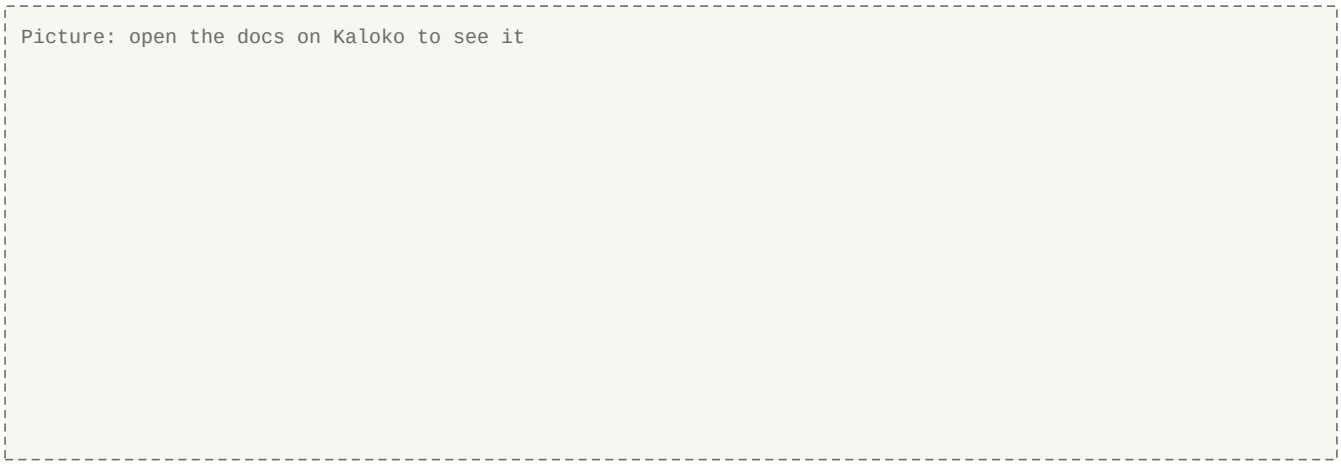
Set the baseline

On the canvas, open a run you are happy with, choose Accept run and then Set as baseline. Each scenario has one baseline, and it moves only when someone sets a new one. From the terminal:

```
npx kaloko reference --scenario checkout --baseline --run <run-id>
npx kaloko reference --scenario checkout # show the current one
```

Compare a run

On the canvas, choose Compare and the baseline (or the previous run, or any other run of the scenario). Each step gets a badge with the share of pixels that changed, and criteria whose verdict or evidence changed are marked.



A run compared with its baseline · en · desktop

In the step detail you can look at the change four ways: Side by side, Diff, Swipe (drag a line across the two screenshots) and Onion skin (fade one into the other). G shows one step in every language and screen size at once.

The same comparison as text, for an agent or the release notes:

```
npx kaloko compare # the current run against the baseline, else the previous run
npx kaloko compare --run <id> --baseline <id> --json
```

Ignore what always changes

Clocks, ads, carousels and "3 minutes ago" change on every run and drown the real differences. Leave them out of the pixel comparison with ignore: on the step:

```
nodes:
- id: home
ignore:
- { selector: ".ticker", note: live prices }
- { rect: [0, 620, 1280, 180], viewport: desktop }
```

You do not have to find them by hand:

```
npx kaloko walk --stability 3 # three screenshots of each capture a moment apart
npx kaloko stability # what moved, compared with earlier runs of the scenario
```

```
npx kaloko stability --apply # writes the suggestions into ignore:
```

Look at the screenshots before you apply them: a region that changed because the page really changed is not noise.

Reviewers can mark noise too. In the step detail on kaloko.app, Mark a changing region and draw a rectangle on the screenshot. It shows dashed on every run of the flow, and the canvas diff skips it at once. Drawing takes the Developer / Tester or Admin role, a run rather than a draft, and the Screenshot view; otherwise the button is greyed out and says why. To put those regions into the scenario, run:

```
npx kaloko stability --pull
```

Walk the affected steps again afterwards: a capture records the regions it ignored.

Connect Jira, Linear, Slack and Teams

For organization admins on the Team plan and up. Once connected, people file a returned step as an issue straight from the canvas, and each project tells its own chat channel what happened.

Jira or Linear

1. Open Settings -> Integrations and connect Jira Cloud (sign in with Atlassian, or paste an API token) or Linear (sign in, or paste an API key).

2. Map each Kaloko project to a Jira project and issue type, or to a Linear team. A default row covers projects you do not list.

3. For Jira, add the webhook Kaloko shows to your Jira site, so a finished issue closes its comment in Kaloko at once. Without it, Kaloko checks every 30 minutes.

People then choose New issue on a step. The issue gets the screenshot, the criterion and the reviewers' notes, with a link back.

A scenario that walks a site for health (SEO, accessibility) can also open one issue per new finding by itself, close it when the finding is fixed and comment when many more or fewer pages are hit. A Developer / Tester switches it on per scenario: Health API for dashboards, section Findings.

GitHub Issues or Plane for findings

Findings can go to GitHub Issues and Plane too; issues from steps stay with Jira and Linear.

• GitHub: the Kaloko GitHub app opens issues on the repositories it is installed on once you accept its Issues permission on GitHub. For other repositories, paste a fine-grained token under GitHub Issues for findings (only the repositories you sync to, Issues: read and write).

• Plane: paste the workspace slug and an API key under Plane (Profile settings -> Personal access tokens). Leave the address empty for Plane Cloud; a self-hosted Plane must be on a public https address.

Kaloko checks the token or the key before it stores it and never shows it again. Then list, per Kaloko project, the repositories and Plane projects its scenarios may send findings to (a default row covers the rest); a Developer / Tester can pick only from that list. Which scenario sends which findings where is set per scenario, with the repository, the Plane project and routes per check or pack.

Slack or Teams channels

1. In Settings -> Integrations, add a channel for a project: Add to Slack for Slack, or the address of a Teams Workflows flow for Microsoft Teams.

2. Tick the events the channel wants: new run, returned, accepted, comment, mention, release recorded, waiting for approval, health got worse.

3. Set quiet hours if the channel should not hear from Kaloko at night; messages wait until they end.

A restricted project only reaches channels that name it. The organization-wide Slack channel keeps working.

Ask approvers in Slack and Teams

When a run waits for the required approvers of its scenario, each of them gets the e-mail and the inbox item. With Slack connected, Kaloko also writes to every approver whose e-mail address belongs to someone in your Slack workspace. The message links straight into review mode of the run.

• Teams workflows cannot write to one person, so in Teams the approvers are named in the channels that take Waiting for approval. Slack channels with that event name them too.

• If nobody has decided after 24 hours, each approver gets one reminder. Change the hours or turn the reminder off in Settings -> Integrations -> Ask approvers in Slack and Teams.

• Quiet hours there hold the direct messages until the morning of each approver's own time zone.

• If Slack was connected before Kaloko could write to people, that settings block says so. Connect Slack again once.

• Guests who turned approval requests off for a project are not written to.

Export docs and the changelog

Your docs site may already have its own build, or a client wants the guide as a PDF. kaloko export writes the docs and the changelog with their pictures as plain files, so nothing stays locked in Kaloko.

Pick a format

```
npx kaloko export docs --format docusaurus --out website/docs
```

| --format | What you get | Plan | |---|---|---| | md | Markdown with an assets/ folder of pictures, for any static site | every plan | | json | the changelog as structured JSON | every plan | | docusaurus | a Docusaurus folder with front matter, categories and sidebars | Business and up | | vitepress | a VitePress folder with its config | Business and up | | mkdocs | a MkDocs folder with mkdocs.yml | Business and up | | html | offline HTML: a page per guide, the changelog and a print page | Business and up | | pdf | one PDF, printed by the browser | Business and up |

The step references become ordinary image files, in the language and screen size of the export. A step whose recording plays on Kaloko comes with it: a player with the picture as its poster and the MP4 and WebM files in assets/ (MkDocs links the picture to the recording). The PDF has the picture only.

Choose what goes in

- --what docs or --what changelog exports only one of them; the default is both.
- --version 2.4.0 exports the docs of a release instead of the latest.
- --lang cs exports one language.
- --product shop picks a product when the config has several.
- --out names the folder or file to write.

Published or local

By default the export takes what was published with kaloko docs publish. --local takes the files in your repository instead, without publishing them, which is handy for checking a change before it goes out.

Keep a docs site in step

An existing site keeps its own build. Either export into its folder and commit the result, or run the export in the site's CI before it builds:

```
npx kaloko export docs --format vitepress --what docs --out docs-site/guide
```

The token in CI needs only read access; see Roles and API tokens.

Download from the docs page

Readers who prefer a file can use Download on the docs or changelog page. It offers the same formats; on plans below Business only Markdown and JSON are available there too.

Every export opens without Kaloko. For the source of the pages and how to keep them current, see Publish living docs and catch stale sections.

Hide personal data in captures

Walking a copy of production, or reading production pages, puts real names and e-mail addresses in front of the browser. Kaloko can hide them in everything a step stores: the screenshot, the captured HTML, the page outline the evaluators read, traces, recordings and captured e-mails. Checks still run against the real page, so a criterion like "the summary shows the name from the form" keeps its verdict.

Mask what you know

List the elements that hold personal data under capture.mask, on the scenario or on a single step:

```
capture:
mask: [".customer-name", "[data-qa=email]", ".payer"]
```

The scenario's masks and the step's add up. In the screenshot each element gets a dark striped box. In the HTML its text becomes [hidden] padded to the same length ([skryto] in Czech runs), so the layout of the copy holds. Form fields (hidden ones too), mailto: and tel: links, title, alt and data-* attributes are covered as well. Attributes nobody sees, such as data-*, change only in the stored HTML: the page itself never notices.

Let Kaloko find it

pii: redact on the scenario turns on detection for every step. It finds e-mail addresses, phone numbers (+420 and 00420 numbers, and the Czech 603 123 456 grouping next to a word like Telefon, Mobil or Kontakt), birth numbers (rodné číslo, with the checksum; the old nine-digit form and numbers without the slash only next to a label like RČ), IBANs and Czech account numbers, dates of birth next to a label, addresses with a postcode, and names in greetings ("Dobrý den, Jano,"). Amounts such as 250 000 000 Kč and order numbers are left alone. Form fields are recognised by their type, autocomplete and name. Values in URLs count too, also when they are encoded (jan%40firma.cz). capture: { mask: auto } does the same for one step.

Detection also reads text in open shadow DOM and in frames from the same origin. It cannot read closed shadow roots, frames from another origin, or text drawn into images and canvases: mask those elements by selector.

Values that look like personal data but are not, such as your support line, go under pii.allow in kaloko.config.yml. * matches any text:

```
pii:
allow: ["info@acme.cz", "*@support.acme.cz", "+420 800 100 200"]
```

Fake values instead of boxes

pii: pseudonymize replaces personal data with made-up values: eva.svobodova42@example.com, +420 000 731 205, Petr Novák. One value gets the same stand-in everywhere in the run, and in every later run of the scenario, so screenshots compare pixel by pixel and approvals carry over. Two different values never share a stand-in. The screens stay readable for a client, and you can still see that the name typed into the form reached the summary and the confirmation e-mail. Fake phone numbers have 000 after the country code and fake birth numbers month 00, so nobody takes them for real ones.

The stand-in is picked by a hash of the value, the scenario and the project. It keeps far too little of the value to lead back to it. If you want a secret in the mix, set KALOKO_PII_SALT (the same one on every machine, or the stand-ins differ between your laptop and CI).

Values masked on one step are replaced on the later ones too, also when the next step runs in a new kaloko process: the run keeps them in .pii-known.json in its folder, which stays on your computer. Short values are not carried over, so a masked 2024 or Praha does not turn into a placeholder everywhere else.

E-mails

The recipients of a captured e-mail (To and Cc) are hidden whenever a step masks anything. For the greeting

and the payer's details, mask their elements on the e-mail step (capture: { mask: [".greeting", ".payer"] }) or rely on detection. A text-only e-mail is covered by detection and by the values masked earlier in the run.

Require it on an environment

```
environments:
production:
base_url: https://shop.acme.cz
pii: required
```

With pii: required, kaloko start creates no run unless the scenario says pii: redact or pseudonymize, or you start with --pii redact. pii: redact or pii: pseudonymize on the environment turns hiding on for every run there.

The check before sharing

kaloko share reads what it is about to upload. In a run that hides personal data (pii on the run, the scenario or the environment, or mask: auto), an unmasked e-mail, phone number, birth number or IBAN stops the upload, and Kaloko names the step and the element:

```
Not shared: the run holds 1 unmasked personal data value(s):
summary (cs, desktop) <p.contact> email ja….cz
```

Mask it and capture the step again, or add it to pii.allow when it is not personal data. kaloko share --allow-pii uploads anyway. A run that does not hide personal data is uploaded, with the same list as a warning.

The site's own contacts do not count: e-mails on the domain of the environment, e-mails and phone numbers in the page footer, and those of an Organization or ContactPoint in JSON-LD. Neither do sample addresses like you@company.com, placeholder text and numbers in SVG drawings. To check the site's contacts like any other value, set pii: { site_contacts: false } in kaloko.config.yml. Screenshots and videos are not read: what covers them is the mask.

On the canvas

A run that hides personal data says so next to the environment ("Personal data hidden"). In the step detail the masked places are outlined in purple, with a note that a dark striped box is on purpose, not a bug in the app.

Check pages on a schedule (hosted runs)

The marketing site changes several times a week, and none of it goes through your pull requests. A hosted run opens the pages of a read-only scenario on Kaloko's own browsers every day or week, takes full-page screenshots in every language and screen size, checks what can be read from the page and shares the run under your name. Nobody has to start the CLI. Hosted runs are part of the Business plan (2,000 loaded pages a month) and Enterprise (10,000).

1. Prepare a read-only scenario

The scenario's steps open pages. To start from the pages themselves:

```
npx kaloko draft https://example.com/ https://example.com/pricing
```

2. Schedule it

```
npx kaloko schedule add --scenario qa/flows/public.yml --env production --every day --at 5
npx kaloko schedule add --scenario qa/flows/public.yml --env production --every week
--weekday mon
```

--at is the hour in UTC. --every manual creates a schedule that runs only when you start it.

Pages behind a gate work too. HTTP Basic credentials of the environment are sent once and kept encrypted. For signed-in pages, a person saves a session with npx kaloko auth save --env production --account member, and the schedule uses it with --account member.

3. Manage it

```
npx kaloko schedule list # next and last run, pages used this month
npx kaloko schedule run <id> # run it now
npx kaloko schedule pause <id>
npx kaloko schedule resume <id>
npx kaloko schedule remove <id>
```

Settings -> Integrations on kaloko.app shows the same schedules, with the next run and how many health results the last run found new and fixed.

Run after a deploy

A schedule can also start when your site is deployed. Create its trigger URL and secret:

```
npx kaloko schedule hook <id> # prints the URL and the secret, once
```

or in Settings -> Integrations -> Hosted runs -> Run after a deploy. Keep the secret in your CI or hosting secrets, never in the repository. A second call makes a new secret and the old one stops working; --remove removes the trigger.

Every call that carries the secret starts the run at the next quarter hour. Calls before the run starts make one run; a call during a walk gets its own run after it. At most 30 calls an hour per schedule are accepted, and each one is in the audit log.

From any CI job, after the deploy step:

```
curl -fsS -X POST "$KALOKO_HOOK_URL" -H "Authorization: Bearer $KALOKO_HOOK_SECRET"
```

Vercel. Vercel reports every deployment to GitHub, so a small workflow calls the trigger when production is up. Add the repository secrets KALOKO_HOOK_URL and KALOKO_HOOK_SECRET, then commit .github/workflows/kaloko-after-deploy.yml:

```
name: Kaloko after deploy
on: deployment_status
jobs:
kaloko:
```

```
if: github.event.deployment_status.state == 'success' && github.event.deployment.environment
== 'Production'
runs-on: ubuntu-latest
steps:
- run: curl -fsS -X POST "$KALOKO_HOOK_URL" -H "Authorization: Bearer $KALOKO_HOOK_SECRET"
env:
KALOKO_HOOK_URL: ${{ secrets.KALOKO_HOOK_URL }}
KALOKO_HOOK_SECRET: ${{ secrets.KALOKO_HOOK_SECRET }}
```

Netlify. Site configuration -> Notifications -> Deploy notifications -> Add notification -> Outgoing webhook. Event "Deploy succeeded", the trigger URL as the URL, the secret as the "JWS secret token". Kaloko checks Netlify's signature.

Your own sender. Instead of the bearer token you can sign the body: send X-Kaloko-Signature: sha256=<hex>, the HMAC-SHA256 of the request body with the secret as the key.

Read the result

The run appears among the others, compared with the scenario's baseline, with the same canvas, comments and approvals. Checks of selectors, texts, titles and meta tags are decided. Criteria that need the full walk say "not evaluated" and give the reason.

Checks of packs (SEO, accessibility) land in the run's health, not in its verdict. When a run has a new error-level finding compared with the previous one, the schedule tells the scenario's followers, the channels that take "Health got worse" and the health.worse webhook. Each schedule can alert on new errors (the default), on new warnings too, or not at all: in Settings -> Integrations -> Hosted runs, or with --health-alert error|warning|off on kaloko schedule add. More in Check the health of your app.

Stack steps: many pages of one template

A stack step stands for a family of pages, every product page for example. A hosted run lists its pages from items.urls, items.sitemap or items.robots (gzipped sitemaps and sitemap indexes included). match, exclude, limit and rotate: week work as in kaloko walk, so both pick the same pages in the same week.

Each page is opened once, in the step's first screen size, and read without a screenshot. Checks that need only the page's elements (selectors, texts, titles, meta tags) are decided; the index fields title, H1, description, word count and JSON-LD types are filled in; everything else says "not evaluated" with the reason. The HTTP status of a page stays empty. On the canvas the step is a stack with its index, as after a walk.

A big stack is read over several quarter-hour rounds, at most 150 pages per round. kaloko schedule list and Settings -> Integrations show how far it got, and the run is shared once every page has been read. While it runs, Run now, a deploy trigger or a saved change waits for it and starts afterwards; pausing the schedule ends it. One run reads at most 1,000 stack pages, and from the sitemaps at most 5 × limit matching addresses per language (20,000 at most). If a round fails or gets nowhere (a sitemap server that is down), the next round tries again; after three such rounds in a row the run is given up and the schedule waits for its next date.

Sitemaps may live on another host, a CDN for example. Kaloko fetches them without the environment's password or cookies, and only when the name resolves to public addresses, after every redirect too. Pages are opened only on the environment's own addresses; others in the sitemap are left out with a note. match and exclude may be up to 300 characters, without backreferences or lookaround.

A stack with items.variants: hreflang groups the language versions of a page as kaloko walk does: the list, the week's window and the ids come from source_locale, and every other language opens each page's hreflang alternate. Before the pages, the run reads each source page's HTML head for its hreflang. Each such read counts as one page, both in the monthly budget and in the 150 a round reads, so a stack of 100 products in three languages takes 400 pages: 100 reads and 300 pages. A page without its translation is marked in the stack index and counted in HC1005, as after a walk. The reads stay on the environment's own addresses, also after a redirect, and carry the schedule's password or cookies there, the same ones the hosted browser gets. When the hreflang names production while the run walks staging, each alternate's path is opened on the

language's staging address. A source page that redirects off the environment is not followed there: unlike kaloko walk, the run reports it as a missing translation (cannot read the page).

One address per language

When the environment has a base_url per language, the hosted run picks it the way kaloko walk does: the language's own address, otherwise default.

```
environments:
production:
base_url: { default: "https://www.example.com", cs: "https://www.example.cz/cs" }
readonly: true
```

What hosted runs cannot do

They only open pages and read them. Nothing is clicked, typed, submitted or created, so steps that do more than open a page are left out and named. Native apps, local or private addresses and credentials sent as headers also need the full walk, and so do stack steps that list their pages with crawl, script or file. Edit links from items.resolve are not filled in by hosted runs.

For those, export the schedule as a GitHub Actions workflow that walks the scenario with the full CLI:

```
npx kaloko schedule export github --scenario qa/flows/checkout.yml --env staging --every week
```

The workflow names the secrets it needs.

How pages are counted

Every loaded page counts: each step in each language and screen size. A scenario of 10 pages in 2 languages on desktop and mobile loads 40 pages per run, so a daily run uses about 1,200 pages a month.

Each page of a stack counts once, when it is read. A weekly stack of 100 pages in 3 languages uses about 1,300 pages a month. When the month's budget runs out halfway through a stack, the run waits up to a day for more (a new month, a higher limit), then shares the pages read so far and names the rest as not walked.

Invite people and guests

A run is only useful once the people who decide have seen it. Bring colleagues into the organization, invite a client to a single project, or let everyone from your company domain read along. Viewers are free on every plan, so nobody has to ration who may look.

Invite a colleague to the organization

1. Open Settings -> People and choose Invite to the organization.



Members in Settings · en · desktop

2. Enter the e-mail address and pick a role. A Viewer reads, comments and approves; the other roles are explained in Roles and API tokens.

3. The person gets an e-mail with a link and joins the organization with it.

The list in Settings shows every member with their role and when they were last seen, and pending invitations with who sent them and when they expire.

If the invitation cannot be delivered, the pending invitation says Address bounced: check the spelling and the person who sent it gets an inbox item. Kaloko does not send it again: revoke it and invite the right address.

Invite someone to one project

Clients, freelancers and colleagues from other teams often need one project, not the whole organization.

1. Open the project and choose Share.

2. Type the e-mail address, also one outside your organization, and pick a role.

3. Choose Invite. The invitation is valid for 14 days.

An address outside your organization joins as a guest. Guests see only the projects they were invited to: not the other projects, not the people of the organization, not settings or billing. The Share dialog also shows who already has access and whether the project is open to every member or restricted to the people listed.

On Business you can set a date when a guest's access ends; on that day they leave the project and get an e-mail. Guests with the Designer or Developer / Tester role are part of Enterprise; on other plans a guest is a Viewer.

Let your company domain sign in

Instead of inviting colleagues one by one, add your company domain under Settings -> People -> Sign-in domains. Prove you control it with the TXT record Kaloko shows; until the domain is verified, nobody joins through it. Anyone with an e-mail on a verified domain can then sign in with the domain's default role (Viewer unless you choose another). Public mailbox domains such as gmail.com cannot be claimed.

Readers from your domain

On Business, the Readers from our domain switch in the same place lets everyone who signs in from a verified domain read the changelog and docs of every product, also where the project is restricted. They join as Viewers and take no seat. See Publish a changelog with pictures.

Who takes a seat

Viewers never do. Designers, developers / testers and admins take one seat each, once for the whole organization, members and guests alike. An invitation with such a role holds a seat until it is accepted or revoked.

Publish a changelog with pictures

Your accepted runs already hold every screen of a release. Kaloko turns CHANGELOG.md in your repository into a page with before and after pictures that clients, support and marketing can read. Kaloko's own changelog is made this way and is public: kaloko.app/p/sinfin/kaloko/changelog.

Set it up once

```
npx kaloko init --docs=changelog
```

It adds CHANGELOG.md (Keep a Changelog) and the changelog block in kaloko.config.yml. A repository with several apps lists them under products:; npx kaloko products sync puts them on Kaloko before their first release.

Put pictures in the entry

A picture names a step instead of a file:

```
- The pricing page compares every plan in one table.
![Before](kaloko:KALOKO-public/pricing?version=prev) ![After](kaloko:KALOKO-public/pricing)
```

KALOKO-public is the scenario id and pricing the step id. Without ?version= the reference shows the newest accepted screen; ?version=prev shows the one from the release before.

Draft the entry

```
npx kaloko changelog draft # prints the Unreleased entry
npx kaloko changelog draft --apply # writes it into CHANGELOG.md
```

The draft collects the pull requests and commits since the last release, groups them into Added, Changed, Fixed and Removed, and adds before and after pictures of the screens accepted runs show differently. Rewrite the lines for people: what they can do now, one change per line. --since <tag> drafts from another release.

Record the release

```
npx kaloko release 2.4.0
```

Unreleased becomes 2.4.0, the release is recorded with the accepted runs that are its pictures, and the command prints the changelog page. Without a version, Kaloko reads it from git tags, package.json, release-please or changesets.

When release-please, changesets or semantic-release make your releases, they keep writing CHANGELOG.md and Kaloko only reads it. npx kaloko ci github --release adds a workflow that records every published GitHub release.

Who reads it



The Products page · en · desktop

The changelog is under Products in the app. Everyone who sees one of the product's projects reads it, guests included. On Business and Enterprise an admin of those projects can choose on the page who else reads it:

• Readers from our domain: everyone who signs in from a verified domain of your organization.

• Anyone with the link: a public page with an Atom feed. Search engines index it only when you allow it.

Readers can also ask for a "What's new" e-mail on the changelog page (Business and up): each new release arrives with its pictures in their language.

How readers use the page: Read what's new. To keep guides current the same way, see Publish living docs.

Publish living docs and catch stale sections

Guides in your repository can show the screens of your accepted runs instead of pasted screenshots, so their pictures never go stale. When a screen changes after the text was written, Kaloko tells you which section to check. The pages you are reading now are made this way: kaloko.app/p/sinfin/kaloko/docs.

Set up the folder

```
npx kaloko init --docs=full
```

It creates docs/ with the four Diátaxis folders, each with a template:

Folder	The reader wants
tutorials/	to learn by doing, start to finish
how-to/	to get one task done
reference/	to look a fact up
explanation/	to understand why

index.md is the start page. Front matter is optional: title (else the first heading) and order (the position in the sidebar). Link pages with relative .md links.

Write a page

Pictures are step references, as in the changelog:

```
1. Choose **Sign in** in the top bar.
![The sign-in page](kaloko:KALOKO-public/login)
```

A translation sits next to its original: pay.cs.md beside pay.md, with the languages listed in the config (docs.languages: [en, cs], originals first). Pictures follow the language of the text.

Some things are easier to watch than to read: a drag and drop, a dialog that opens, a language switch. When the accepted run has a recording of the step (kaloko walk --record), the picture shows a play button; a click or a tap plays the recording without sound and the picture comes back at the end. Nothing plays by itself. Add ?media=video to say that a step needs its recording (kaloko docs check warns when the run has none), or ?media=image to keep just the picture:

```
![Translate a category](kaloko:DOC-console/category-edit?media=video)
```

Recordings that your docs or changelog play are kept for as long as they play them, and they count in storage.

Show one part of a screen

A whole screen is often too much when the text talks about one button. Name the parts a guide needs in the scenario, under the step's details:, and every walk takes them in each language and screen size, masked like the screenshot:

```
- id: cart
title: Cart
path: /cart
details:
- id: add-to-cart
name: { en: Add to cart button, cs: Tlačítko Přidat do košíku }
selector: "#add"
- id: summary
name: Order summary
selector: .summary .total
context: section
```

Kaloko shows the element with the part of the page around it and draws a ring around the element. Without context that part is the nearest section or card of at least 480 × 240 px (narrower on a phone); context: card, context: section, context: page (one screen around it) or a CSS selector of an ancestor choose it yourself, and padding adds room (16 px by default). A long section is cut to one screen around the element. states: [error] takes a detail only in some states of the step.

In the text, add the id after #:

```
2. Choose **Add to cart**.
```

```
![Add to cart](kaloko:shop/cart#add-to-cart)
```

The other parameters work as before (kaloko:shop/cart?locale=cs#add-to-cart). The canvas lists the details of a step under Step attachments on the Page tab, each with its reference. Until an accepted run has the detail, readers see the whole screen and kaloko docs check warns.

One action per list item with its picture right under it also makes the page a walkthrough: readers choose Walk through it step by step and see one step at a time, and can tick I tried it.

Publish

```
npx kaloko docs publish # the docs of latest
npx kaloko docs publish --version 2.4.0 # the docs of a release (Business)
```

Readers get a version selector, language and screen switches and search, and every picture opens its step on the canvas. kaloko release publishes the docs folder for its version on its own unless you pass --no-docs. Who reads the docs follows the same rules as the changelog: see Publish a changelog with pictures.

Catch stale sections

When you publish, Kaloko remembers the screen behind every reference. When a later accepted run shows a different screen, the section says "The screen changed in 2.4 — check the text." to the people who write the docs (Business).

```
npx kaloko docs check # against the published docs; needs a token
npx kaloko docs check --offline # only the local files
```

Errors: broken references, steps the scenario files do not have, references without an accepted picture and stale sections. Warnings: missing or outdated translations, changelog items that people will see but that have no picture, media=video references whose accepted run has no recording of the step, and #detail references that the scenario does not declare or that no accepted run has yet. The command exits with 1 on errors; --strict also on warnings, which suits CI.

For each stale section, open the step, compare it with the text and rewrite what no longer fits; the note clears at the next kaloko docs publish. If the text still fits, kaloko docs publish --reviewed clears the notes without a change. Use it only after you have read them.

To get the docs out as files for another site, see Export docs and the changelog.

Read what's new

Every release of a product has an entry in its changelog, written for the people who use it and illustrated with the screens it changed. The pictures come from runs someone accepted, so they show what was approved.

Find it

1. Open Products in the main menu and choose Changelog at the product, or open the link someone sent you. Each product has its own changelog; its newest release is on top.

2. The versions at the top jump to their release. Everything below a version is what changed in it, grouped as Added, Changed, Fixed and Removed.

Compare before and after

1. A change to a screen shows the screen before and after on top of each other. Drag the slider to move the split.

2. Side by side puts the two pictures next to each other (on a phone, one above the other).

3. Language and Screen switch every picture to another language or screen size, when the run captured them.

4. Open the flow opens the whole flow on the canvas, if you may see the run.

Keep up

• E-mail me what's new sends you the entry of every new release, with its pictures, in your language. The link at the bottom of each mail stops them.

• Download gives you the changelog as JSON, Markdown, offline HTML or PDF, for release notes or a client report.

• A public changelog also has a feed for feed readers.

Record a walkthrough video

A screenshot shows where a step ended. A recording shows how it got there: the spinner that never stopped, the menu that opened twice. Record when the screenshot cannot show what a reviewer has to judge: animations, loading states, drag and drop, a flaky step, or a demo for a client.

Record a walk

```
npx kaloko walk --record # each step, from the previous screen to its capture
npx kaloko walk --record flow # the whole walk per language, a chapter per step
npx kaloko walk --record failed # only steps whose script or checks fail
npx kaloko walk --record --record-quality low --steps checkout
```

To record by default, set it in the scenario or in kaloko.config.yml:

```
capture:
record: { mode: failed, quality: low, max_seconds: 45 }
nodes:
- id: checkout
record: true # this step always; record: false never
```

Quality is low, medium (the default) or high. Recordings are much larger than screenshots, so do not record every walk.

Traces come with every web step

Each walked web step keeps a trace: clicks and typing, network requests with status and timing, and console messages. It is small and on by default; --no-trace turns it off.

```
npx kaloko video list # what was recorded and traced, with sizes
npx kaloko video trace --step checkout # the step's trace as a timeline in the terminal
```

On the canvas the step detail shows Video next to the screenshot, and the Technical tab shows the trace. Click a row and the video jumps there. Videos play on: once you play one, the next step's video starts by itself when it ends. Replay with video moves to the next step on its own as well. A step that needs your decision waits for you, and a step without video shows its screenshot for three seconds.

Make one video of the run

```
npx kaloko video export --format mp4 # or webm, gif
npx kaloko video export --locale cs --steps cart,payment --lang cs --out checkout.mp4
```

The export puts the recordings in flow order with a title card before each step (its title and verdict). It is the file to send to a client or attach to a release note.

Tools you need

Web and Electron recordings need nothing but Chrome. With ffmpeg installed you also get MP4 for Safari, app recordings (Android, the iOS Simulator, macOS) and the walkthrough export. npx kaloko video tools says what this machine has. Walks in WebKit and Firefox are neither recorded nor traced.

Sharing

Recording works on every plan. kaloko share uploads recordings with the run on Business and Enterprise; on Free and Team they stay in the local preview, and the screenshots and traces are shared. Shared recordings are kept for 30 days (Enterprise sets its own period in Settings -> Storage). A recording over 25 MB stays local: lower the quality or max_seconds.

Review a run

A run is one pass of an agent through a flow: every screen it reached, in every language and screen size it walked, with the checks it made. You review it in the browser; there is nothing to install.

Open the run

1. Open the link from the pull request, the issue or the e-mail. The run opens on the canvas: every step of the flow as a map, with arrows for the way between them.



The map of a run · en · desktop

2. If you would rather work through a list, press I. The same run becomes a list of steps with their results, and you can search it and filter by status.



The run as a list · en · desktop

Look at a step

1. Click a step. Its detail shows the screenshot, the criteria with what was found, and the comments people left on it. <- and -> move to the previous and next step.



A step with its comments · en · desktop

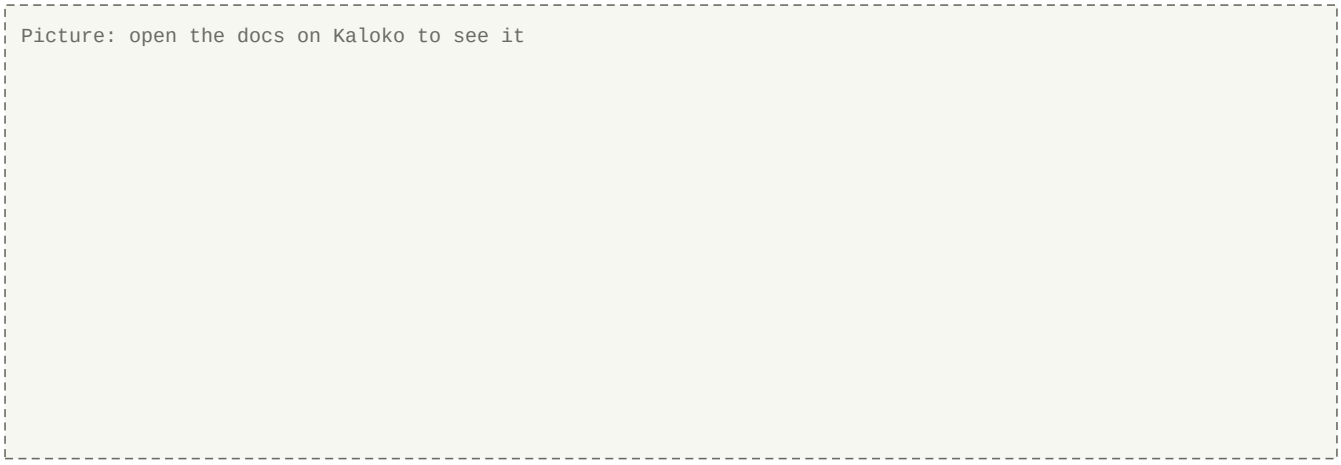
2. Something looks wrong? Write a comment on the step. All open comments of the run sit in the comments panel, so nothing gets lost between steps.



The comments panel · en · desktop

Walk it like a user

Press R to replay the run: one screen at a time, in the order a person would see them. It is the quickest way to show the flow to someone who has never seen the canvas.



Replay · en · desktop

When you know what you think, approve or return the run.

Review only what changed

Review mode shows one step at a time, starting with the steps that need a person: failed and returned steps, criteria the AI was unsure about, steps that changed since the last accepted run and steps with open comments. Steps that did not change keep the approval someone gave them before.

Open review mode

1. Open the review link from the pull request or the e-mail (it ends in #review), or open the run and press Q.



Review mode · en · desktop

2. The step you are deciding sits next to the accepted run, so you see what moved.

Decide with the keyboard

| Key | What it does | |---|---| | J / K | next and previous step | | A | approve the step | | R | return the step with a note | | C | comment | | G | the step in every language and screen size |

When every step is decided, accept or return the whole run.

Look closer

In the large view you can compare the two runs side by side, as a difference, with Swipe (drag a divider across both screenshots) or with Onion skin (this run faded over the other). To point at a problem, draw a box, an arrow or a line on the screenshot while you write the comment, or paste a picture.

Approvals carried over

A carried approval names who gave it and in which run. If you think the step needs another look, revoke it and the step waits again. A carried approval never accepts the run for you.

Roles and API tokens

Every person in an organization has a role, and every token acts for a person with no more rights than that role. Agents and CI use tokens; they never take a seat.

Roles

| Role | May do | Seat | |---|---|---| | Viewer | read runs and designs, comment, approve and return | free | | Designer | also work on Kaloko Design drafts, versions, branches and token sets | one seat | | Developer / Tester | also upload and manage runs and their own tokens | one seat | | Admin | also manage members, domains, all tokens and the organization's settings | one seat |

A project can have its own project admin, who invites people to that project, changes their roles and can restrict or open it. Developers / testers and admins can create projects; whoever creates one becomes its admin. Change a member's role in Settings -> People.

Create a token in the app

1. Open Settings -> API tokens.



API tokens in Settings · en · desktop

2. Name the token after where it lives, for example "laptop CLI" or "GitHub Actions".

3. Tick what it may do. Reading is always included.

• comment and approve: comments and verdicts, as you

• design: kaloko sync of design drafts

• upload runs: kaloko share from the CLI or CI

• everything my role allows

4. Pick how long it lives: 30 days, 90 days or a year.

5. Optionally limit it to chosen projects; it then sees only those and cannot create new ones.

6. Copy the token once and put it into the project's .env as KALOKO_TOKEN.

The list shows each token's owner, role, prefix, last use and expiry. Revoke ends a token at once.

Sign in from the terminal

On your own machine there is a shorter way:

```
npx kaloko login
```

The browser opens; you approve the code, choose the organization and what the token may do. The page shows where the request came from, which terminal asked and when: approve it only if you just ran kaloko login

yourself, and decline a link someone sent you. The token is valid for at most 30 days and is saved to ~/.config/kaloko/.env. --print prints it instead, --no-open shows the link without opening a browser.

Tokens for CI

Create a token with the upload runs scope, ideally limited to the project, and store it as a KALOKO_TOKEN secret in your CI. npx kaloko ci github and ci gitlab write workflows that read it; see the CLI reference.

Tokens for dashboards

A dashboard that pulls health once a week needs less than reading: pick read:health. Such a token reads the health of runs and the newest run of each scenario, and gets 403 for everything else, screenshots included. What it can call is in Health API for dashboards.

Rules an admin can set

Under Settings -> Security an admin can allow only admins to create tokens with every right, turn off tokens limited to projects, and decide whether guests may create tokens at all. A guest's token sees only their projects. On Enterprise, service accounts hold tokens for automation that belongs to no person.

Run Kaloko in CI and comment on pull requests

In CI, Kaloko runs the same commands as on your laptop. The usual setup walks what a pull request changed on its preview deployment (Vercel, Netlify, Cloudflare Pages or any review app) and puts the canvas link into a comment on the pull request.

1. Write the workflow

```
npx kaloko ci github # writes .github/workflows/kaloko.yml
npx kaloko ci gitlab # writes kaloko.gitlab-ci.yml; include it from .gitlab-ci.yml
npx kaloko ci github --print # show it, write nothing
```

--scenario a.yml,b.yml limits it to some scenarios. The command also adds a preview environment to kaloko.config.yml with base_url: ${KALOKO_PREVIEW_URL}. A scenario that lists environments: has to include preview, or kaloko start refuses it; the command tells you which ones.

2. Add the token

Create a token with the tester role in Settings -> API tokens and add it to the repository's CI secrets as KALOKO_TOKEN. On GitLab, add KALOKO_GITLAB_TOKEN too: a project access token with the api scope, so Kaloko can write the note on the merge request. Commit the workflow.

What the workflow does

On every successful preview deployment of a pull request:

1. It installs the project and the browser (npx kaloko browser install --with-deps), plus WebKit and Firefox when a scenario asks for them.

2. npx kaloko ci preview-url --wait 600 --github-env finds the preview address, the pull request and its base branch and passes them on as KALOKO_PREVIEW_URL, KALOKO_PR and KALOKO_BASE.

3. For each scenario, kaloko affected --fail-if-none skips it when the change does not reach any of its steps. Otherwise it runs:

```
npx kaloko start --scenario qa/flows/checkout.yml --env preview
npx kaloko walk --ephemeral --affected
npx kaloko share --pr
```

share --pr adds one comment per scenario to the pull request and updates it on later pushes. Uploads go by content, so screenshots that did not change are not sent again.

The Kaloko check on the pull request

With the Kaloko GitHub App (an admin installs it in Settings -> Integrations -> GitHub, Team and up), every pull request in a connected repository gets one check named Kaloko on its newest commit. It turns green when the runs shared for that commit pass, or when the change reaches no scenario. It waits while a scenario the change reaches has no run yet, and turns red when one fails. The runs can come from this workflow, from a coding agent in the cloud or from a laptop: what counts is the commit. A run of uncommitted changes does not count.

To block merging until it is green, require the status check Kaloko in the repository's branch protection. Kaloko never changes branch protection itself.

• Modes (Settings, per repository): off, manual (only after the label kaloko:run or /kaloko walk), automatic (the default) and strict (every scenario, a review required, a time limit).

• One pull request: the labels kaloko:skip, kaloko:all, kaloko:review, the comment /kaloko skip <reason> (this commit only) or the buttons on the check. Only people with the access your admin chose may skip, never apps or bots, and the check names who skipped it and why.

• In the repository: the github: section of kaloko.config.yml on the base branch can make the check stricter,

never looser.

• npx kaloko ci plan tells the check exactly which scenarios a commit reaches. The generated workflow sends it before walking.

• No check appears? Settings -> Integrations -> GitHub lists the repositories and says why, for example a repository that is not in the app's installation yet.

Good to know

• walk --affected needs the git history: check out with fetch-depth: 0 on GitHub or GIT_DEPTH: "0" on GitLab.

• --trigger pull_request runs on every push instead of the deployment event and waits for the preview.

• kaloko ci preview-url prints only the address, so it works in any CI: export KALOKO_PREVIEW_URL="$(npx kaloko ci preview-url)".

• A preview behind Vercel's deployment protection needs a VERCEL_AUTOMATION_BYPASS_SECRET secret and the access.headers block of the preview environment, which the generated config has commented out.

• Runs shared from CI also compare their health (the advisory checks of packs) with the previous run of the scenario and alert when there is a new error. npx kaloko health --fail-on error stops the job on health errors, if the team wants that.

• No comment on the pull request? Usually there was no preview deployment, or the token is missing. The job log says which.

When release-please, changesets or semantic-release publishes your releases, npx kaloko ci github --release adds a second workflow that records each release on Kaloko. All options are in the CLI reference.

Monitor SEO and answer engines every week

A translation release goes out on Thursday. On Monday someone notices that the Polish product pages lost their structured data and half of the Greek magazine has no meta description. A weekly walk of every public page would have said so on Friday morning. This page sets one up, with Squared as the example: www.squared.one in 20 languages and www.vyvolej.to in Czech, Slovak and Ukrainian, about 400 pages per language.

The plan:

• one stack step per content family (e-shop, magazine, FAQ, the rest), each reading the pages from the site's per-language sitemaps;

• lanes for the content families, so the canvas shows one row per family;

• sessions for the kinds of visitors whose pages you check (a visitor from search, a signed-in club member);

• the seo and aeo packs on every page, advisory, never part of a verdict;

• a GitHub Actions workflow that walks it every week and shares the run;

• the results read back through the API or MCP for a dashboard.

1. Find the sitemaps

The quick way is to let Kaloko draft steps 1 and 2 from the site itself:

```
npx kaloko init --seo https://www.squared.one
```

It reads robots.txt, takes the languages from the sitemap addresses (or from the first path segment of the pages when the site has one sitemap), samples each language's pages and writes qa/flows/seo-squared-one.yml: one rotating stack step per content family with a match for every language, the seo and aeo packs, and the production environment in a new kaloko.config.yml. Families are paired across languages by the sitemaps' hreflang links, then by the same segment, then by size. The comment at the top of the file lists the pairs it guessed by size; check those before the first walk. --families 6 takes more families, --limit 100 more pages a week, --packs seo,aeo,a11y adds accessibility. When the site cannot be reached, nothing is written. The rest of this page explains what the draft contains, so you can sharpen it.

Squared lists one gzipped sitemap per language in robots.txt:

```
Sitemap: https://www.squared.one/sitemaps/en_US/sitemap.xml.gz
Sitemap: https://www.squared.one/sitemaps/de/sitemap.xml.gz
Sitemap: https://www.squared.one/sitemaps/es/sitemap.xml.gz
…
```

items: { robots: true } reads those lines. When a sitemap's address names the walked language as a segment (/sitemaps/de/…), only that language's sitemaps are read, so the German walk gets the German pages. en_US finds en_us or en-us; en does not pick en-us. A site with one sitemap for all languages gets all of it, and match picks the language. You can also name the file yourself: sitemap: '/sitemaps/{locale}/sitemap.xml.gz'. Gzipped sitemaps and sitemap indexes work either way.

A staging behind a password works too. kaloko walk reads robots.txt and the sitemaps with the environment's access (its headers, HTTP Basic, client certificates and the walk's cookies), on the environment's own addresses only. A sitemap on another host, a CDN for example, or one that redirects there is read without them. Each file has 60 seconds and 60 MB.

2. Split the pages into content families

Each family is a stack step with its own match or exclude. Look at the first path segment of each language's sitemap first: on Squared even the e-shop is translated (/fi/tuotteet/, /pl/sklep/, /bg/foto-produkti/), and the magazine is /de/zeitschrift/, /es/revista/, /fi/lehti/. One regex with every translation keeps one step per family for all languages:

```
schema: qawalk.scenario.v1
id: SEO-weekly-squared
```

```
title: Weekly SEO and answer-engine monitor (squared.one)
kind: flow
locales: [en_US, de, es, fr, it, nl, pt]
viewports: { desktop: [1280, 800] }
environments: [production]
readonly: true
page: landing
health: { site: true }
lanes: [E-shop, Magazine, FAQ, Club and other pages]
sessions:
visitor: { title: Visitor from search }
nodes:
- id: eshop
kind: stack
title: E-shop
lane: 0
session: visitor
packs: [seo, aeo, locale]
items:
robots: true
match:
'^/([a-z]{2}/)?(eshop|tuotteet|sklep|trgovina|katastima|pood|veikals|parduotuve|webshop|prod
use|foto-produkti)/'
limit: 400
fields:
- { id: title, from: title }
- { id: http, from: http }
- { id: jsonld_types, label: Structured data }
- { id: og_title, label: OG title, from: og:title }
- { id: canonical, from: canonical }
criteria:
- { id: AC1, text: Product pages carry Product and BreadcrumbList data, check: { type:
deterministic, assert: jsonld_present, types: [Product, BreadcrumbList] } }
- id: magazine
kind: stack
title: Magazine
lane: 1
session: visitor
packs: [seo, aeo, locale]
items:
robots: true
match:
'^/([a-z]{2}/)?(magazine|zeitschrift|revista|rivista|lehti|magazyn|revija|magazin|periodiko|
ajakiri|zurnals|zurnalas|spisanie)/'
fields: [{ id: title, from: title }, { id: words, from: words }, { id: jsonld_types }]
- id: faq
kind: stack
title: FAQ
lane: 2
session: visitor
packs: [seo, aeo, locale]
items: { robots: true, match: '^/([a-z]{2}/)?faq/' }
fields: [{ id: title, from: title }, { id: jsonld_types }]
- id: pages
kind: stack
title: Club and other pages
lane: 3
session: visitor
packs: [seo, aeo, locale]
```

```
items: { robots: true, exclude:
'/(eshop|tuotteet|sklep|trgovina|katastima|pood|veikals|parduotuve|webshop|produse|foto-prod
ukti|magazine|zeitschrift|revista|rivista|lehti|magazyn|revija|magazin|periodiko|ajakiri|zur
nals|zurnalas|spisanie|faq)(/|$)' }
fields: [{ id: title, from: title }, { id: jsonld_types }]
edges: []
```

Quote a source that contains {locale} inside { … }, or YAML reads the braces as a map.

When one regex with every translation gets too long, give match (or exclude) one regex per language. Kaloko tries the walked locale, then its language (en for en_US), then default:

```
items:
robots: true
match:
default: '^/([a-z]{2}/)?eshop/'
fi: '^/fi/tuotteet/'
pl: '^/pl/sklep/'
bg: '^/bg/foto-produkti/'
```

A match map without the walked locale and without default stops the walk with an error, so a forgotten language never takes every page of the site. An exclude map without the locale drops nothing.

match and exclude always apply before limit, for every source. When the pages come from your own script (items: { script: steps/seo-items.mjs }), one script can serve every family: it gets node (id, title, lane, lane_title, session) and items.params:

```
items: { script: steps/seo-items.mjs, params: { family: magazine } }
```

```
// steps/seo-items.mjs
export default async ({ locale, node, params, url }) => {
const res = await fetch(url(`/api/sitemap/${params.family}?locale=${locale}`))
return (await res.json()).map((p) => ({ path: p.path, title: p.title }))
}
```

Before the first full walk, check the counts with npx kaloko walk --limit 3 and look at the numbers per step. A step with 0 pages usually means a translated segment is missing from its regex.

One page in all its languages

By default each language reads its own list, so the Czech product /eshop/obaly-na-fotky/ and its Slovak translation /sk/eshop/obaly-na-fotky-nahodna-farba/ are two pages of the stack, and each language samples different products. To see the translations as language versions of one page, let the stack take its list in one language and follow each page's hreflang to the others:

```
items:
robots: true
match: '^/eshop/'
variants: hreflang
source_locale: cs
limit: 100
rotate: week
```

The list, match, limit and rotate come from source_locale (by default the scenario's first locale). In every other language Kaloko reads each chosen page and walks the address its <link rel="alternate" hreflang> gives for that language, or the Link header, or the sitemap's xhtml:link when the page names none. The page keeps the id of the source page, so the stack index has one row per product with a screenshot in every language, and every language checks the same products in a given week. The row shows the source page's address, title and fields when the run walks the source language too; a run of other languages only (--locales sk) shows what the first walked language found.

A page without a version in a language does not disappear quietly. The stack index marks it with that language, and HC1005 counts it: there is no hreflang for the language, the page lists only x-default (which is no translation), the hreflang for the language points back to the page itself, the page cannot be read, or the translation answers an error (error 404). Pages whose Slovak hreflang all lead to one address (usually a listing

or the home page) count too. kaloko export findings lists each of them as a row of that language at the source page, where the hreflang belongs. Whether the translation links back is HC125 of the seo pack.

Kaloko reads the HTML as the server sends it, without running scripts, with the environment's access (its headers and HTTP Basic, client certificates, the walk's cookies) on the environment's own addresses only. When none of the alternates is on the host the page was read from, the page names another site's addresses: staging or a preview whose hreflang names production, also with one domain per language. Kaloko then opens each alternate's path on the walked language's address from the environment.

When your own script lists the pages per language instead, give each page a key that is the same in every language (for example the path of the source page). Pages with one key are one page of the stack.

Hosted runs (kaloko schedule add) group language versions the same way, with the same pages in the same week. Reading a source page for its hreflang counts as a page of the monthly budget there; see Hosted runs.

3. Choose the sessions

A session is one kind of visitor; every step names one. For a public SEO monitor one is enough: visitor, signed out, the way a search engine sees the site. When signed-in pages matter (the club area for members), add a session with an account, save its sign-in once with npx kaloko auth save --env production --account member, and give those steps session: member. Keep signed-in steps out of the sitemaps step: search engines never see them.

4. What the packs check

The seo pack checks title, description, canonical, hreflang, Open Graph, robots, the H1 and the viewport, and now also that JSON-LD is on the page (HC112) and that every block is valid JSON (HC113). The aeo pack adds cheap signals for AI search and assistants: FAQPage or QAPage data where the page asks and answers (HC901), a short answer-first paragraph under the H1 (HC902), /llms.txt at the site root, fetched once per walk (HC903), a meta description (HC904), and the page as GPTBot gets it without JavaScript (HC905): one more request per page, without cookies, only where robots.txt lets GPTBot in. It fails when the title, canonical, H1, JSON-LD or most of the text appear only after scripts run; a site or WAF that turns GPTBot away shows as not evaluated, not as a failure. To check another crawler, write the criterion yourself with assert: nojs_parity and user_agent: claudebot (or googlebot, perplexitybot, a whole user agent string). Every check is advisory. Leave the errors pack out of these steps: it is for error pages, and on normal pages its soft-404 and explanation checks (HC601, HC605) stay not applicable. When the team decides that product pages must carry their structured data, adopt the check or write your own criterion, as AC1 above does. The details are in Criteria, checks and packs.

What one page cannot show, the stack as a whole can. After the walk the seo pack compares the pages of one language with each other: shared titles, descriptions and H1s (HC121–HC123), canonicals that point elsewhere (HC124), hreflang the other language version does not answer (HC125), and sitemap pages that redirect, answer an error or say noindex (HC126). The findings count pages, and items.json lists the addresses, so they can go to whoever fixes them in groups.

The locale pack watches the language versions: html lang, the page's own hreflang, the language of the text (an untranslated page in a language version) and the decimal separator of prices (12,50 ? in German, $12.50 in English), always against the walked locale.

health: { site: true } adds the checks of the whole site, once per run: robots.txt (typos, blocked start pages, blocked AI crawlers), the sitemaps robots.txt lists (reachable, valid, absolute same-host addresses, sane lastmod), /llms.txt, and http:// to https:// and the other host name (with or without www) in one redirect. When the site blocks AI crawlers on purpose, set health: { site: { ai_crawlers: false } }. kaloko health gives the site checks a line per site.

5. Size the walk

Each page takes about 1–2 seconds and stores about four files, and a shared run holds at most 20,000 files.

Squared has about 400 pages per language, so seven languages (2,800 pages, about 11,000 files) fit into one run. Split the languages into scenario files that share the same steps:

| File | Languages | |---|---| | qa/flows/seo-squared-west.yml | en_US, de, es, fr, it, nl, pt | | qa/flows/seo-squared-north-east.yml | da, sv, fi, pl, sl, hr, el | | qa/flows/seo-squared-baltic-south.yml | et, lv, lt, hu, ro, bg | | qa/flows/seo-vyvolej.yml | cs, sk, uk (environment with base_url: https://www.vyvolej.to) |

walk.concurrency in kaloko.config.yml walks several languages at once; 3 or 4 keeps a weekly run of a few thousand pages within an hour or two.

Cover a big catalogue in turns

When every language together has more pages than one run can hold, walk a part of each family every week. rotate: week takes limit pages starting at a different place each ISO week, so the whole list is covered after pages ÷ limit weeks:

```
items: { robots: true, match: '^/([a-z]{2}/)?eshop/', limit: 100, rotate: week }
```

With 400 product pages per language this covers the e-shop every four weeks at a quarter of the cost. The walk says which part it takes (rotating sample, week 2026-W41 — pages 101…200 of 400, all covered in 4 weeks). The same week gives the same pages, so a run you repeat on Tuesday checks what Monday checked.

rotate: run counts runs instead of weeks. Kaloko keeps the count and the time each page was last checked in .cache/stack/ of the output folder (tmp/kaloko), so it works where that folder stays between runs (your laptop, a self-hosted runner). A fresh GitHub Actions checkout starts from zero every time: use rotate: week there.

steps/<step>/items.json in the run then has a rotation section per language (the window, how many pages have been checked so far and how many never) and each page its last_checked: the previous time Kaloko checked it, or null.

Keep a big run small

A shared run holds 300 MB as stored. HTML and JSON travel gzip-compressed, so the page snapshots count for about a tenth of their size on disk; the screenshots are what adds up. Stack pages are captured whole, so a typical long page stores a screenshot of about 600 KB and a thumbnail of about 180 KB (the thumbnail is as tall as the page). That fills a run at roughly 400 pages.

An SEO family needs the page, not a picture of it. capture: { screenshot: thumb, full_page: false } keeps only a 640 px thumbnail of the first screen, about 25 KB, which is enough to recognise the page in the stack:

```
items: { robots: true, match: '^/([a-z]{2}/)?eshop/', limit: 100, rotate: week }
capture: { screenshot: thumb, full_page: false }
```

capture: { full_page: false } alone keeps a sharp first screen (about 70 KB), and capture: { screenshot: thumb } alone a thumbnail of the whole page. Leave the defaults on families where someone reviews the whole page or a semantic check reads what is below the fold.

When a run comes close to the limit, kaloko share names the largest steps, so you know which family to trim, and says so when a stack captured whole takes most of the room.

6. Walk it every week on GitHub Actions

```
npx kaloko schedule export github --scenario qa/flows/seo-squared-west.yml --env production
--every week --at 5
```

The command writes a workflow that walks the scenario on Monday at 5:00 UTC, evaluates it and shares the run. It needs the secret KALOKO_TOKEN (Tester role, upload scope). Export one workflow per scenario file, at different hours, and raise timeout-minutes in the workflow for the big ones. When your config says that every scenario names its own project (a comment, or a product with projects:), give the scenario project: seo or another of them; kaloko validate warns when it is missing, and a share without it can end with HTTP 404.

Without GitHub Actions, kaloko schedule add walks the same scenario on Kaloko's side (Business plan). It reads stacks from sitemaps and robots.txt, takes a base_url per language and decides the checks that need only the page's elements, without screenshots; each stack page counts against the monthly budget of hosted pages. Structured data, answer-engine and locale checks stay "not evaluated" there, so for the full SEO and AEO packs

the CLI on GitHub Actions is still the right place. More in Check pages on a schedule.

7. Read the results into a dashboard

Every shared run has its health: findings by severity, pack and step, and what is new or fixed since the previous week. For a dashboard:

• GET /api/v1/orgs/<org>/scenarios/<id>/latest/health/aggregate?by=locale,step,check counts the newest run per language, content family and check, with the most affected pages and how many results are new or fixed since last week. …/latest/health/diff lists them page by page, each with the week it first failed. A token with the read:health scope reads these and nothing else, which suits a dashboard that pulls once a week. The shapes are in Health API for dashboards; MCP has get_health_aggregate and get_health_diff.

• The health.changed webhook tells you after each run when something is new or fixed.

• GET /api/v1/orgs/<org>/health?project=<project> or MCP get_health { project } gives the trend of the project; get_health { scenario } the newest run of one scenario.

• MCP get_stack_items { run_id, step, status?, criterion?, query? } lists the pages of one step with their fields (jsonld_types, og_title), status and failing checks, for example every e-shop page that fails HC112.

• In the run folder, steps/<step>/items.json holds the same per page, and its structured_data counts pages per schema.org type, pages without structured data and pages with invalid JSON-LD. kaloko health --json prints the findings of the local run.

A weekly chart of "pages without structured data" and "new health errors" per content family is usually enough. When a number jumps, open the run on the canvas and filter the stack by the failing check.

8. Hand the fixes to whoever fixes them

A finding is half the work; the editor still has to find the page in the CMS. Give every page its way back:

• an item script returns edit_url (the page in the CMS) and content_ref (the record a content API knows it by) next to path;

• or the stack gets a resolver that looks them up for the pages walked this week:

```
items: { robots: true, match: '^/([a-z]{2}/)?eshop/', limit: 100, rotate: week, resolve:
steps/resolve.mjs }
```

```
// steps/resolve.mjs
export default async (item, { locale }) => {
const res = await
fetch(`https://cms.example.com/api/pages?path=${encodeURIComponent(item.path)}&locale=${loca
le}`)
const page = res.ok ? await res.json() : null
return page ? { edit_url: `https://cms.example.com/pages/${page.id}/edit`, content_ref:
`page:${page.id}` } : {}
}
```

A page the resolver fails on, or does not answer for within 15 seconds, is walked anyway, without the link, and the walk says how many. The page detail on the canvas then shows Edit in CMS, and the stack's CSV carries both columns. For the whole list:

```
npx kaloko export findings --format csv --out fixes.csv
npx kaloko export findings --format json --locale de --step eshop
```

Each row is one failing check on one page in one language: the URL, the check, its evidence, edit_url, content_ref and the command that walks that page again (kaloko walk --steps eshop --items <page> --locales de). The JSON goes straight to an agent with a content API, for Squared the MCP tools that propose, apply and verify a change. After the fix, that one command checks the page again without walking the rest, also when a rotating stack has moved on to other pages; such a walk does not count as a week of the rotation.

Work through findings, not pages

Two hundred product pages without structured data in Polish are one problem in the product template, not two

hundred. Health -> Findings of the scenario (the link next to each scenario on the health page) groups what is open by check, site, language and step: one row per problem, with its severity, how many pages it hits, a few of them with their edit link, and the week it first appeared. Acknowledge a finding when someone is on it, or ignore it with a reason ("old blog, moving to the new CMS in November"); the decision stays with it from week to week, and a finding is fixed once none of its pages fails. A dashboard or an agent reads the same through GET …/scenarios/<id>/findings or MCP list_findings.

If the team works in Jira, Linear, GitHub Issues or Plane, a Developer / Tester can let the scenario open one issue per new finding at or above a severity (PUT …/scenarios/<id>/findings/sync { "tracker": "jira", "min_severity": "error" }). Kaloko then comments when the number of pages changes a lot and closes the issue when the finding is fixed. Routes split the work: technical checks (canonical, hreflang, structured data) to the developers' GitHub repository, content checks (descriptions, thin pages, low click-through) to the marketers' Plane project. With traffic data (section 9) each issue says how many clicks and impressions the pages get. The details are in Health API for dashboards.

9. Fix the pages people actually see first

Two hundred missing descriptions are a long list. Search Console knows which of those pages people see in search, and Kaloko can use that. Export the Pages table from Search Console (Performance, Pages, Export, CSV), put it next to the scenario, and name it:

```
traffic:
file: traffic/search-console-pages.csv
```

The export goes stale, so once the weekly run works, let it ask Search Console itself. Create a service account in Google Cloud, add its e-mail to the property as a user, put the path of its JSON key in GOOGLE_APPLICATION_CREDENTIALS (a repository secret in GitHub Actions), and replace the file with the property:

```
traffic:
gsc: { property: sc-domain:example.com }
```

Every run then asks for the clicks, impressions, CTR and position of every page over the last 28 days, once per run. ga4: { property: "123456789" } adds sessions and key events by landing page the same way.

What changes:

• kaloko export findings puts the findings on the most seen pages first and adds clicks, impressions, ctr, position, sessions and impact to each row, so the content team starts at the top of the CSV;

• the page index on the canvas has Impressions and Clicks columns; click one to sort by it;

• the seo pack adds HC127 to each stack: pages seen often in search but clicked less than half as often as the site's pages at similar positions. The title and description in the results are the usual suspects.

The formula and every key are in Configuration file. Data from other SEO tools (ranked keywords, AI visibility) is not joined yet.

Walk Android, iOS and desktop apps

Kaloko walks native apps the way it walks web pages: every step becomes a screenshot with its UI tree, checked criteria and a place on the canvas. The scenario says which platform it is, and the config says which app to open.

1. Name the app per environment

```
# kaloko.config.yml
environments:
staging:
app:
android: { package: com.acme.shop.staging, apk: build/app-staging.apk, reset: true }
ios: { bundle: com.acme.shop, app: build/Shop.app, device: iPhone 17 }
desktop:
app:
electron: { main: dist/main.js }
# desktop: { name: Notes } # a macOS app by its window
# windows: { name: contoso, path: dist/Contoso.exe }
```

2. Set the platform in the scenario

```
platform: android # android | ios | electron | desktop | windows
devices: [pixel-8-api-34, pixel-7-api-33]
color_schemes: [light, dark]
nodes:
- id: product
kind: screen
path: shop://product/42 # a deep link
packs: [app]
```

Steps are reached by deep links or by a step script (app.tap(…), app.type(…)). Every locale starts the app fresh in that language, and reset: true clears its data first.

What each platform can do

Platform	Driven by	Notes
Android	adb: emulator, USB phone or a phone over Wi-Fi	an emulator that is not running is started; dark mode and reduced motion are set per capture
iOS Simulator	xcrun simctl	taps and the UI tree need idb; without it Kaloko reads the text from the screen
iPhone and iPad	devicectl and a screenshot tool	capture only, text read from the screen
Electron	Playwright	the window is a web page: every web check applies
macOS app	the app's window and accessibility tree	capture only; your tool, AppleScript or a person moves between screens
Windows app	UI Automation through PowerShell	taps and typing; walk it on Windows

The app pack checks labels on controls, touch target sizes, cut-off text and crashes.

Screens from other tools

Maestro, Appium, XCUITest, Detox or a person can take the screenshot, and Kaloko lays it out with its UI dump:

```
npx kaloko capture --step checkout --image shot.png --tree hierarchy.xml --device "Pixel 8" --density 2.625
```

--tree reads uiautomator or Appium XML, idb ui describe-all --json and Maestro's hierarchy JSON. Without a tree, --ocr reads the text from the image on macOS. A whole folder of screenshots or a test report goes in with npx kaloko import.

Check the machine first

```
npx kaloko doctor
```

It says which app tools are missing on this machine (adb, simctl, idb, ffmpeg) and how to install them. On macOS, kaloko doctor --fix asks for the Screen Recording and Accessibility permissions desktop captures need.

Write a scenario

A scenario is the flow Kaloko walks: a YAML file with the screens, the way between them and what each screen has to meet. Task flows usually sit next to the task (docs/tasks/TASK-123/qa/scenario.yml), long-lived flows such as the checkout in qa/flows/.

Let the agent write it

With the skill installed, ask your agent instead of typing YAML:

```
Read the task in docs/tasks/TASK-123, write its acceptance plan and a Kaloko scenario, and validate it.
```

The agent writes two files: ACCEPTANCE.md for people (goal, steps, criteria AC1, AC2…) and the scenario for the runner. When the intent changes, it updates the plan first and then the scenario.

The parts of the file

```
schema: qawalk.scenario.v1
id: TASK-123-checkout
title: Checkout
kind: task # task | flow
product: shop # optional, with module:
locales: [en, cs]
viewports:
desktop: [1280, 800]
mobile: [360, 800]
environments: [local, staging]
nodes:
- id: cart
kind: screen
title: Cart
path: /cart
purpose: The visitor sees what they are buying and can go on to pay.
execution:
strategy: playwright # playwright | agent | manual
script: steps/cart.mjs
criteria:
- id: AC1
text: The page has exactly one main heading.
check: { type: deterministic, assert: count, selector: h1, min: 1, max: 1 }
edges:
- { from: cart, to: payment, label: continue }
```

• Step ids (cart) stay stable: links, comments, approvals and docs pictures refer to them.

• purpose is one sentence about what the screen is for. The evaluator reads it.

• execution says who moves to the screen: a Playwright script, an agent that captures with kaloko capture, or a person.

• Other node kinds: decision for a branch, email for a message from the mailbox, stack for many pages of one template.

Every key is in the scenario file reference. To add checks, see Add criteria and check packs.

Check it

```
npx kaloko validate qa/flows/checkout.yml
npx kaloko validate qa/flows/checkout.yml --coverage
```

validate checks the schema and the references between steps. --coverage compares the criteria with the

acceptance plan and lists what the plan asks for and the scenario does not check.

Start from public pages

For a marketing site or documentation, start from a draft:

```
npx kaloko draft https://example.com/ https://example.com/pricing
```

Kaloko fetches the pages and writes a read-only production scenario and its plan, with the page title, H1 and description as the purpose and the seo, a11y, perf and console packs. It also adds a step for an address that does not exist, so the 404 page is checked too. Then sharpen each purpose and add what the page really has to meet.

Criteria, checks and packs

A criterion is one thing a step must meet. It has an id (AC1), a text for people and exactly one check. How to add them to a scenario: Add criteria and packs. How the results turn into a verdict: How verdicts are decided.

Kinds of check

| type | When to use it | Options | |---|---|---| | deterministic | anything countable or exact: presence, counts, text, the URL, styles, values in the page | assert and its options (below) | | semantic | judgement: is the copy clear, is there one primary action | question (English, yes = met), true and false (what counts as yes and no), scope (CSS selector sent to the evaluator), about: console (read the console errors instead of the page), vision: true (also show the screenshot) | | manual | neither works; a reviewer decides on the canvas | note |

Numbers, dates and counts belong in deterministic checks, never in a semantic question. Ask one thing per question.

Deterministic checks (assert)

Common options: selector (CSS), pattern (a regex) with flags (i, m, s), value, min, max, all (every match must pass, not just one).

Page content

| assert | Passes when | |---|---| | exists | selector matches at least one element | | absent | selector matches nothing | | count | the number of matches is between min and max | | text_matches | the element's visible text matches pattern | | text_absent | no visible text matches pattern | | text_equals | the whole text equals value (whitespace collapsed) | | attr_equals | attribute attr equals value | | attr_matches | attribute attr matches pattern | | no_overflow | the page does not scroll sideways | | no_console_errors | the console has no errors |

Address, metadata and links

| assert | Passes when | |---|---| | title_matches | the page title matches pattern | | meta_matches | the content of the selector meta tag matches pattern | | link_matches | the href of the selector link matches pattern | | url_matches | the URL after the step matches pattern; part picks href, path, query, hash, host or origin | | url_equals | the URL equals value (a path or a whole URL); ignore_query drops ?… and #… | | http_status | the document's status is between min and max (default 200–299), or equals value | | link_home | the page links back to the home page | | links_ok | every same-origin link answers below 400 (max links per page, pattern skips paths, follow outside read-only environments) | | hreflang_pairs | at least min languages, each version answers 200 and links back |

Structured data and answer engines

| assert | Passes when | |---|---| | jsonld_present | the page has a <script type="application/ld+json">; with types: [Product, BreadcrumbList] its structured data contains each of them (graphs, lists and nested values such as offers count, and so does microdata) | | jsonld_valid | every JSON-LD block parses as JSON; a page without JSON-LD passes. A failure names the block and the parse error | | aeo_faq_schema | a page that asks and answers (two or more details > summary, dl > dt, or h2–h4 ending with "?") has FAQPage or QAPage structured data. A page without questions is not evaluated | | aeo_answer_first | the first paragraph after the H1 is between min and max characters (default 40–320): a short answer before the detail | | llms_txt | /llms.txt of the site answers 200 with text, not an HTML page. One request per site and walk | | nojs_parity | the page's raw HTML, fetched without JavaScript, holds what the rendered page shows: the title and canonical match, the meta description, H1 and every JSON-LD type are there, and the text of the body is at least ratio of the rendered one (default 0.3; judged when the rendered page has 200 characters or more). user_agent: fetches as a crawler: gptbot, oai-searchbot, claudebot, perplexitybot, googlebot, bingbot or a whole user agent string |

Without JavaScript (nojs_parity). Many AI crawlers read the HTML the server sends and never run scripts. The check makes one more GET per page, read in Node without a browser (at most 500,000 characters, 15 s), and compares it with the page the walk captured. Titles and descriptions are compared as people read them (–, », no-break spaces and the like decoded). The text counts the whole body on both sides,

hidden parts too, so a part that Next.js streams into a hidden element counts as it does after rendering. With user_agent: it comes like that crawler: without cookies, with the bot's user agent and the walk's Accept-Language, and only after robots.txt allows it. A page robots.txt closes to the bot is not fetched and shows "blocked by robots.txt for GPTBot" as not evaluated; so does every page when robots.txt answers 5xx, because crawlers then fetch nothing. When the site turns the request away (401, 403, 429 or 503, a challenge page, a redirect to another origin such as a sign-in), the check is not evaluated either and says who was turned away: the bot's user agent (the browser's gets the page), the missing cookies of a sign-in or cookie gate, or everyone. That is not a rendering problem. Without user_agent: it uses the walker's browser user agent and session and does not read robots.txt. The evidence lists what is missing without JavaScript, for example GPTBot without JavaScript: no H1 (rendered: "Photo books"); JSON-LD Product only after JavaScript; body text 0 of 1840 characters (0 %), at least 30 % expected. A title in another language says so, with both lang values. A page longer than the limit is cut, and its JSON-LD is then not judged. Each page and user agent is fetched once per walk (the walk keeps a small summary, not the page) and robots.txt once per site and bot. Compare landing steps: the raw HTML is the page as it loads, not after the step's clicks.

Every capture also records structured_data: the number of JSON-LD blocks, the invalid ones, the schema.org types and the recognized ones (Product, Offer, AggregateOffer, BreadcrumbList, FAQPage, QAPage, Article, BlogPosting, NewsArticle, Organization, WebSite). A stack step's items.json counts pages per type, pages without structured data and pages with invalid JSON-LD (structured_data at the top of the index and in the step's stack).

Language versions (the locale pack uses these)

They compare the page with the locale it was walked in (cs, de, en_US). A locale that is not a language tag (an ISO 639-1 language such as cs; cz is a country) makes them not applicable.

| assert | Passes when | |---|---| | locale_lang | html lang names the language of the walked locale (de-DE for de); a different region is mentioned, not failed | | locale_hreflang | the page's own hreflang names it with its language, every code is valid (en-GB, not en_UK), and where addresses carry a language (/de/…) the code agrees. A page without hreflang is not evaluated | | locale_text | the page text reads as the walked language. Kaloko counts common words of 26 languages (most of Europe and Turkish); pages under min words (default 40), other languages and close pairs (Danish, Norwegian, Swedish; Croatian, Slovenian) are not judged | | locale_prices | prices with decimals use the locale's separator (12,50 ? in de, $12.50 in en). Pages without such prices are not evaluated |

Styles and values in the page

| assert | Passes when | |---|---| | style_equals | the computed CSS property equals value or a design token; op compares numbers (>=), tolerance allows px rounding | | style_matches | the computed property matches pattern | | js_equals | one read-only JavaScript expression in the page returns value (or matches pattern, or compares with op) |

Accessibility (the a11y pack uses these)

| assert | Passes when | |---|---| | a11y_images_alt | images have alternative text | | a11y_names | links and buttons have an accessible name | | a11y_labels | form fields have labels | | a11y_headings | heading levels are not skipped | | a11y_contrast | text meets WCAG AA contrast | | a11y_main | the page has a main landmark | | a11y_unique_ids | element ids are unique | | a11y_focus_order | Tab order follows the page and reaches every control | | a11y_focus_visible | focus is visible with at least 3 : 1 contrast | | a11y_keyboard_trap | Tab never gets stuck | | a11y_skip_link | a skip link or the main content within three Tab stops | | a11y_ax_names | controls have a name in the browser's accessibility tree | | a11y_landmarks | landmarks are complete and repeated ones labelled | | a11y_aria | ARIA roles and attributes are valid |

Performance, apps and design

| assert | Passes when | |---|---| | perf_lcp, perf_cls, perf_ttfb, perf_weight | largest contentful paint, layout shift, time to first byte or page weight stays within max | | app_labels | every app control has an accessible label | | app_touch_targets | touch targets are at least 48 dp (Android) or 44 pt (iOS) | | app_no_truncation | no text is cut off with an ellipsis | | app_no_crash | the app did not crash or stop responding | | design_pixels | the screen differs from the design reference in at most ratio of pixels (default 0.03) | | design_structure | headings,

landmarks, actions and fields match the design reference | | uses_tokens | colours, fonts, sizes, spacing and radii come from the token set |

Packs

packs: [seo, a11y] on a step adds a fixed set of checks. They are health checks: advisory, shown apart from acceptance and never part of the step's verdict. The design pack always counts for acceptance; other packs do when the scenario lists them under health: { acceptance: [seo] }. Every check of every pack, with why it matters and its severity, is in Health checks.

On a design environment (kind: design) the steps are your own HTML served from a folder, so the packs that only code can satisfy (seo, aeo, perf, console, errors) are off there. Keep one on with health: { design_packs: [console] } in the scenario. a11y, usability and locale run on designs too.

Health checks have HC ids, acceptance criteria AC ids. The number is the check's own, so HC401 (LCP as a health check) and AC401 are the same check in its two roles: a pack in acceptance adds AC401, and adopting HC401 gives the step the criterion AC401 with from: HC401. A criterion of your own that takes a check's place replaces it and counts for acceptance, which is how you change a limit (AC401 with max: 4000) or adopt one check (kaloko health adopt <step>:<check>). Runs and scenario files from before 7 Oct 2026 call health checks AC101…; Kaloko reads those as HC101… and accepts the old ids in commands.

| Pack | Ids | What it adds | |---|---|---| | seo | HC101–HC113, HC121–HC127 | one H1, title 10–70 and description 50–160 characters, canonical, hreflang for every language, Open Graph, not noindex, viewport meta, html lang, no overflow, no console errors, JSON-LD present (HC112) and valid (HC113); on a stack step also the checks across its pages (below) | | usability | HC201–HC204 | the purpose is clear on the first screen, one primary action, labelled fields, link texts that say what they do | | a11y | HC301–HC314 | every accessibility check above; the walk also presses Tab through the page, without clicking or typing | | perf | HC401–HC404 | LCP within 2.5 s, CLS below 0.1, TTFB within 800 ms, page weight within 3 MB (measured in the walk's browser) | | console | HC501 | the evaluator reads the console errors and says whether they touch what the step accepts | | errors | HC601–HC606 | for error pages: a real 4xx/5xx status, a link home, navigation or search, no stack trace, a plain message with a way forward, no console errors. HC601 and HC605 apply only to error pages (a step with expect_status: 404 or a page answering 4xx/5xx); elsewhere they are not applicable | | app | HC701–HC705 | labels, touch targets, no truncated text, no crash, a clear purpose and next action | | design | AC801–AC803 | the implementation against the design reference: pixels, structure, tokens | | aeo | HC901–HC905 | for answer engines (AI search, assistants): FAQPage or QAPage where the page asks and answers, an answer-first paragraph of 40–320 characters after the H1, /llms.txt at the site root, a meta description, and the page without JavaScript as GPTBot gets it (HC905, nojs_parity) | | locale | HC1001–HC1004 (HC1005 on stacks with items.variants, below) | for multilingual sites: html lang, the page's own hreflang, the language of the text and the decimal separator of prices match the walked locale. Only in scenarios that list their locales |

The a11y pack covers part of WCAG 2.2 AA. It does not replace an audit with a screen reader.

The aeo pack holds cheap signals that can be read from the page, not a ranking in any AI assistant. Hosted runs check its meta description (HC904) and show the other aeo checks, and HC112 and HC113, as "not evaluated"; kaloko walk and the scheduled GitHub workflow evaluate them all. HC905 needs a live page: kaloko recheck leaves it as the walk found it. Ids HC9xx belong to aeo, HC10xx to locale; hosted runs show the locale checks as "not evaluated" too.

Across the pages of a stack (seo pack)

A stack step with the seo pack also compares its pages with each other once they are walked. Nothing is fetched again: the checks read the title, description, H1, canonical, robots meta, hreflang and HTTP status of each capture.

| Id | Fails when | |---|---| | HC121 | two or more pages of one language share a title. Pages that answer an error, pages that name another canonical, and one address reached twice are left out | | HC122 | … share a meta description | | HC123 | … share an H1 | | HC124 | a page names another address as its canonical (pages pointing to one address form a cluster), or a canonical points to a walked page that answers an error or says noindex. On staging or a preview whose canonicals name the production host, the canonical counts as the page itself when

only the host differs | | HC125 | a page with hreflang does not name itself, or a language version the stack walked answers an error or does not link back. Alternates outside the stack are counted, not fetched (hreflang_pairs fetches them); a missing x-default is mentioned, not failed | | HC126 | stacks read from a sitemap (items.sitemap, items.robots): a page answers other than 200, redirects or says noindex | | HC127 | with traffic data only (traffic in the config or the scenario): a page with at least 100 impressions has a CTR below half the site's median CTR at similar positions (1–3, 4–10, 11–20). Revise its title and description (Configuration file) | | HC1005 | stacks with items.variants: hreflang, with or without a pack: a page of the source locale has no version of its own in a walked locale (no hreflang for it, only x-default, the hreflang points back to the page itself, the page cannot be read, or the translation answers 4xx/5xx: error 404), or several pages lead to one version. The walk reads the hreflang when it looks for the language versions |

They are health findings of the stack step, counted in pages. steps/<step>/items.json keeps them under cross_page with the groups and their addresses (at most 20 groups of 10), and every affected page lists the ids in cross_fails. A check across pages cannot be adopted as a criterion of one page. Orphan pages and click depth are not checked yet.

Site checks (health.site)

Some things belong to the whole site, not to a page. With health: { site: true } in the scenario, the walk checks each site it walks once per run (one per origin of the environment's base URLs) and writes site.json into the run:

| Id | Fails when | |---|---| | HC1101 | /robots.txt does not answer 200, is an HTML page, or has lines crawlers ignore (a typo such as Disalow, a rule before any User-agent) | | HC1102 | robots.txt blocks / or the start path of a walked language for every crawler, Googlebot or Bingbot. An error on the environment named production, a warning elsewhere | | HC1103 | robots.txt blocks GPTBot, OAI-SearchBot, ChatGPT-User, PerplexityBot, ClaudeBot, Google-Extended or CCBot from the site or a walked start path. Checked on production only | | HC1104 | a sitemap robots.txt lists (or /sitemap.xml when it lists none) does not answer, is not a urlset or sitemap index, is cut off, has relative addresses or pages on a host other than the sitemap's or the site's (robots.txt may point to a sitemap on another host), more than 50,000 addresses, or lastmod values that are not dates or lie in the future. Sitemap indexes are followed two levels deep, at most 50 files | | HC1105 | /llms.txt does not answer 200 with text | | HC1106 | http:// does not redirect straight to https:// on the same host (a temporary 302 or 307 is mentioned). Not evaluated on a non-default port | | HC1107 | the other host name (example.com for www.example.com and back) answers itself or its first redirect goes elsewhere. Subdomains have no other form and are not evaluated; a non-default port is kept |

The checks make their own requests, which is why they are off unless the scenario asks. They send the environment's gate (headers, HTTP Basic) to the site's own origin only and follow redirects themselves, read at most 10 MB of an answer and stop after two minutes per site. An answer of 401 (a sign-in gate) makes a check not applicable, not a finding. Environments that need a client certificate (mTLS) are not checked. Options instead of true:

```
health:
site:
paths: [/, /cs/, /eshop/] # paths robots.txt must leave open (default: / and each walked
language's start path)
ai_crawlers: false # blocking AI crawlers is the site's decision: skip HC1103
sitemaps: 20 # most sitemap files to read per site (default and most 50)
environments: [production] # walks on other environments skip the site checks (default:
every environment)
```

kaloko health shows a line per site and its findings under the pack site, as site:<host>:HC1104; the run manifest keeps the results as site. Site checks cannot be adopted as criteria.

Severity of a health finding

A failed check is an error when people or search engines are shut out: HC107 (noindex), HC108 (viewport), HC110 (overflow), HC111, HC501 and HC606 (console errors), HC301–HC303, HC305, HC310 and HC312 (names, labels, contrast, keyboard trap), HC601 and HC604 (soft 404, stack trace), HC701 and HC704 (app labels, crash), HC1102 (robots.txt blocks the walked start pages, on production). Every other failed check is a warning, and an answer the evaluator was unsure about is unsure.

CLI commands

Every command of the kaloko CLI with what it does. Run them with npx kaloko <command> in a project that has the package installed. npx kaloko help prints the same list with every option.

Options that work everywhere: --run <id|dir> (or KALOKO_RUN) picks the run, --config <file> the config file, --json prints data for scripts (Machine output below), and KALOKO_AGENT=<name> labels what an agent writes. kaloko help <command> explains one command, kaloko help export run one sub-command.

Set up a project

| Command | What it does | |---|---| | kaloko init | Creates kaloko.config.yml, installs the skill for Claude Code or Codex (--agent), detects the framework, dev server, languages and main routes, drafts a first scenario; --docs=off\|changelog\|full sets up the changelog and the docs folders. --project, --project-name and --product name the project on Kaloko. In a repository with several sites it lists them and guesses no routes: --site picks one, --base-url reads the pages from the running site (sitemap, then the home page links), --skeleton writes a scenario with a TODO | | kaloko init --seo <site> | A weekly SEO and answer-engine monitor of a public site: reads robots.txt and the sitemaps, finds the languages and the biggest content families (the first path segment, translated per language) and writes qa/flows/seo-<site>.yml with one rotating stack step per family. --families, --limit (pages per family, language and week), --packs seo,aeo,a11y,perf, --env, --out, --force | | kaloko login | Signs this machine in through the browser and saves a token (at most 30 days) to ~/.config/kaloko/.env; --print prints it instead | | kaloko doctor | Checks Node, Chrome, the config, environments, browser ports, the token, the evaluator key and app tools; --fix asks macOS for capture permissions | | kaloko draft <url…> | Drafts a first plan and a read-only scenario for public pages, with the seo, a11y, perf and console packs | | kaloko validate <file…> | Checks scenarios against the schema and their references; --coverage compares the criteria with the acceptance plan; --json lists the problems per file | | kaloko scenarios | Lists the scenarios the config matches | | kaloko ci github\|gitlab | Writes a workflow that walks what each pull request affects on its preview deployment and comments on it | | kaloko ci github --release | Writes a workflow that records every release your release tool publishes | | kaloko ci preview-url | In CI: finds the preview URL, the pull request and its base | | kaloko ci plan | In CI or a coding agent's sandbox: tells the Kaloko check on the pull request which scenarios this commit reaches |

Walk and capture

| Command | What it does | |---|---| | kaloko start | Creates a run of --scenario on --env and makes it current; --locales, --viewports, --browsers, --color-schemes, --devices and more choose what to capture; --states limits the step states, --token-modes the token modes of a design; --pii redact or pseudonymize hides personal data in every capture; --ref CC-2026-001 (repeat it, or KALOKO_REFS in CI) names the records the run serves, --purpose dry_run, formal or smoke (KALOKO_PURPOSE) says why it was walked | | kaloko walk | Runs the Playwright steps and captures them, then evaluates; --affected, --record, --heal, --stability N, --concurrency N, --ephemeral | | kaloko capture --step <id> | Captures the current page for a step; --state empty as one of its states, --crop attaches an element, --image imports a screen taken by another tool | | kaloko open --step <id> | Opens the step's URL in the shared browser | | kaloko mail --step <id> | Captures an e-mail from the environment's mailbox, or a saved message with --file | | kaloko browser start\|stop\|status | One shared browser per environment that agents, scripts and people drive | | kaloko browser install | Downloads the browser engines (--browsers chromium,webkit,firefox) | | kaloko devices | Lists the device presets you can use as viewports | | kaloko session reset | Starts as a new visitor: clears cookies and storage in the shared browser | | kaloko auth save | A person signs in once (SSO, MFA) and the walk reuses the session; kaloko auth list shows saved sessions | | kaloko affected | Says which steps the changed files reach, and why; --fail-if-none exits 1 when none (--check is its older name) | | kaloko import <path> | Lays out screenshots, Playwright reports, traces or a walkthrough as a run | | kaloko status | Shows what the current run has captured |

Results and review

| Command | What it does | |---|---| | kaloko evaluate | Runs the deterministic checks and the semantic evaluator for all captures; --recheck after a scenario change, --budget N for at most N paid evaluator calls | | kaloko build | Writes run.json and the local preview | | kaloko preview | Builds the preview and prints its path | | kaloko share | Uploads the run to kaloko.app; --pr comments on the pull request or merge request. In a run that hides personal data, an unmasked e-mail, phone number, birth number or IBAN stops the upload with the step and element (--allow-pii uploads anyway); other runs upload with a warning | | kaloko review | What a person still has to decide on a shared run, and the review-mode link | | kaloko health | The health of a run (advisory checks of packs) and what changed since the previous run; --remote for the shared run, --fail-on error to make errors exit 1 | | kaloko health adopt <step>:<check> | Makes a health check an acceptance criterion of the step in the scenario file, as is or with --text | | kaloko health pull | Writes the checks people adopted on kaloko.app into the scenario file | | kaloko feedback | Open comments from reviewers, returned steps first | | kaloko comment | Comments, replies, resolves and reopens feedback from the terminal | | kaloko compare | A text diff of two runs: statuses, verdicts, scores and evidence | | kaloko stability | Finds regions that change on every run; --apply adds them to the scenario, --pull brings in regions drawn on kaloko.app | | kaloko calibrate | Compares the semantic evaluator with human labels and suggests thresholds | | kaloko signoff <run-id> | Downloads the signed acceptance record of an accepted run; --package downloads the evidence package of any shared run (record, trace matrix, audit trail, screenshots, signed manifest); --verify <file> checks a record or a package without a token | | kaloko evidence <run-id> | The evidence of a run, or of a release with --release <version> [--product p] (Business; --ref and --purpose keep only its runs of one record or purpose): acceptance and signing state, criteria as walked with requirement references, results, screenshot hashes, decisions, signatures, audit trail; --csv prints the trace matrix, --json the whole document | | kaloko video export | Turns the run's recordings into one video; kaloko video list, trace and tools show recordings, traces and tools | | kaloko screens | Searches the newest screenshot of every step across your flows | | kaloko inbox | Your notifications on kaloko.app |

Shared runs

| Command | What it does | |---|---| | kaloko projects | Projects of the organization | | kaloko projects create <slug> | Creates a project with the name people see (--name "ECO Podcasty"), optionally restricted or part of a product; an existing project is refused, not renamed | | kaloko projects rename <slug> | Changes the name people see; the project's admin does this | | kaloko projects sync | Makes Kaloko match project:, project_name: and product: of the config: creates the project when it is missing, renames it when the name differs, adds it to the product | | kaloko runs | Shared runs, the newest per scenario; --all lists every run, page by page to the last; --limit N stops after N and --cursor goes on from there; --ref CC-2026-001 lists every run of that record, --purpose formal only the formal runs | | kaloko trash <run-id…> | Moves runs to the trash; kaloko restore <run-id> brings one back | | kaloko prune | Keeps the newest runs per scenario and environment and lists the rest for the trash; --yes moves them | | kaloko keep <run-id> | A kept run never expires and is never pruned; --off releases it | | kaloko archive <scenario-id> | Hides a retired scenario from the list; its runs stay | | kaloko scenario move <scenario-id> --project <slug> | Moves a scenario to another existing project with all its runs, design drafts and screens, like Scenario settings on the scenario's page in the app. The move wins over project: in the config and the scenario file: later shares of the scenario land in the new project too, until you move it back | | kaloko public | Makes a shared run readable without signing in | | kaloko export run\|steps | run downloads a shared run as a zip; steps --scenario <id> the newest capture of every step (the flags --run and --scenario still work) | | kaloko export findings | The fix list of a local run: every failing criterion and health check per page and language with its evidence, the page's edit_url and content_ref and the walk that checks it again; with traffic in the config also the page's clicks, impressions and sessions and an impact, highest first; --format csv\|json, --locale, --step, --out |

Jira, Linear and hosted runs

| Command | What it does | |---|---| | kaloko issue create | An issue in Jira or Linear from a returned step or a failing criterion; kaloko issue list shows them | | kaloko schedule add | Kaloko opens the pages of a read-only scenario every day or week and shares the run; --health-alert error\|warning\|off says when worse health alerts

the team | | kaloko schedule list\|run\|pause\|resume\|remove | Manages the schedules | | kaloko schedule hook <id> [--remove] | A URL and secret that start the hosted run after a deploy or from CI; the secret is shown once | | kaloko schedule export github | A GitHub Actions workflow that walks the scenario on a schedule with the full CLI |

Products, changelog and docs

| Command | What it does | |---|---| | kaloko products sync | Puts the products of the config on Kaloko without recording a release | | kaloko products list | The products you read on Kaloko, and those of the config that are not there yet | | kaloko products remove <product> | Archives a product: it leaves Products, its changelog and docs answer 404, its releases and docs are kept | | kaloko products restore <product> | Brings an archived product back with its releases and docs | | kaloko changelog draft | Writes the Unreleased entry from merged pull requests and changed screens; --apply writes it into CHANGELOG.md | | kaloko release [<version>] | Records a release with its changelog entry, pictures and docs | | kaloko docs publish | Sends the docs folder to Kaloko as the docs of a version | | kaloko docs check | Broken references, stale sections, missing translations and changelog items without a picture; exits 1 on errors | | kaloko export docs | The docs and changelog as Markdown (the default), Docusaurus, VitePress, MkDocs, HTML, PDF or JSON with --format; kaloko export --format <f> still works |

Design

| Command | What it does | |---|---| | kaloko sync | Exchanges design revisions with the shared draft (--pull, --push) | | kaloko design lint | Checks the design folder against the portable HTML rules | | kaloko design import figma | Turns Figma frames into design steps and variables into tokens | | kaloko history | Versions of the flow and revisions of a step | | kaloko spec [step] | What to build for an approved design step: elements, styles mapped to tokens, states; --state empty for one step state | | kaloko drift [step] | How the implementation differs from the approved design; exits 1 while it differs | | kaloko reference | Shows or sets the design reference and the baseline | | kaloko tokens pull\|push\|check\|validate | The project's design tokens (DTCG) | | kaloko branch list\|create\|merge | Branches of the design draft | | kaloko export live\|design | live the live files of a version into a folder (the handoff), design a version in open formats; --live and --design still work |

Machine output

With --json, a command that prints data writes one JSON document to stdout and nothing else; progress, notes and warnings go to stderr. It works on start, status, evaluate, build, share, validate, scenarios, runs, projects, review, feedback, inbox, auth list, browser status, ci preview-url, affected, prune, devices, doctor and the other commands whose help names it. kaloko share --json prints the run id and its URL:

```
url=$(npx kaloko share --json | jq -r .url)
```

When a command fails with --json, stderr ends with one line like {"error":{"message":"Cannot reach https://kaloko.app (ENOTFOUND).","code":"unreachable","next":"…"}}. code is usage, unreachable, no_token, unauthorized, failed or the service's own reason (such as plan).

Commands from a newer kaloko

kaloko help --json prints every command with its sub-commands and options as JSON ({"version":"…","commands":{"<name>":{"flags":[…],"subcommands":{…}}}}); an agent can read it instead of the help text.

When you type a command, sub-command or option this kaloko does not know, it looks at /cli/latest.json on the service it talks to (kaloko.app, or KALOKO_URL): the commands of the newest released kaloko, kept for a day in ~/.cache/kaloko. Only when that version is newer than yours and has exactly what you typed, it says so instead of guessing a typo:

```
There is no command "projects" in this kaloko.
kaloko projects is in kaloko 0.16.0 (you have 0.15.0). Update: npm i -D kaloko@latest
```

The exit code stays 2, and with --json the line is the error's next. Offline it waits at most 1.5 s and prints what it printed before. KALOKO_NO_UPDATE_CHECK=1 or --offline turns the check off. kaloko doctor also says when a newer kaloko is available.

Exit codes

| Code | Meaning | |---|---| | 0 | Done | | 1 | Checks found problems, or the command failed (docs check, drift, design lint, affected --fail-if-none and validate use 1 for their findings) | | 2 | Usage error: an unknown command or option, a missing argument | | 3 | The Kaloko service cannot be reached: offline, DNS, a refused or broken connection. An error answer from the service is 1 | | 4 | No token, or one that is invalid, expired or revoked: run kaloko login |

Durations

--timeout (mail), --wait (ci preview-url), --expires (share) and --older-than (prune) take a number in their own unit, as before (seconds for the first two, days for the others), or a number with s, m, h or d: --wait 10m, --expires 72h, --older-than 14d.

kaloko help prints all of this with every option.

Configuration file

kaloko.config.yml sits in the root of the repository and tells the CLI where your flows run. kaloko init writes a commented one; kaloko doctor checks it. Its format is qawalk.config.v1, and schema, org and environments are required.

```
schema: qawalk.config.v1
org: acme
project: shop
project_name: E-shop
scenarios: [qa/flows/*.yml]
environments:
local:
base_url: http://localhost:3000
staging:
base_url: https://staging.acme.test
access:
basic: { user_env: STAGING_USER, password_env: STAGING_PASSWORD }
accounts:
member: { user: qa@acme.test, password_env: MEMBER_PASSWORD }
production:
base_url: https://acme.com
readonly: true
blocked_paths: ["/account/**"]
walk:
browsers: [chromium]
share:
expires_days: 30
```

Top-level keys

| Key | What it holds | |---|---| | schema | always qawalk.config.v1 | | org | the organization's slug on kaloko.app | | project | the Kaloko project for runs (its slug, such as eco-podcasty); default: the repository's name. A scenario someone moved to another project (in the app or with kaloko scenario move) stays there: the move wins over this key | | project_name | the name people see on Kaloko (ECO Podcasty). The first kaloko share creates a missing project with it; kaloko projects sync renames an existing one. A share never renames, it only says when the names differ | | product | the slug of a product on Kaloko this project is part of; kaloko projects sync adds the project to it | | scenarios | glob patterns of scenario files | | output | the run folder (default tmp/kaloko) | | environments | where flows run (below) | | browser | channel (a Playwright channel such as chrome; empty = the bundled Chromium) and port of the shared browser (default 9333) | | walk | defaults for every run: concurrency (languages at once, 1–16), browsers, color_schemes, reduced_motion, forced_colors, display_modes | | capture | defaults for every scenario: record, trace | | evaluate | budget.calls: at most this many paid evaluator calls per evaluation (below) | | evaluators | the semantic evaluator jev, your own model, and the vision evaluator (below) | | pii | allow: values that look like personal data but are not (a company contact), * for any text; neither masked nor stopped by kaloko share | | share | url (default https://kaloko.app), token_env (default KALOKO_TOKEN), expires_days (default 30) | | docs | changelog and docs settings (below) | | products | what you release and version as a whole (below) | | traffic | Search Console and GA4 numbers per page, to rank findings by traffic (below); a scenario's traffic replaces it |

Environments

Each key under environments is a name you use with kaloko start --env.

| Key | What it holds | |---|---| | kind | app (default) or design (a folder of live HTML steps) | | base_url | the address, or one per language with a default fallback; ${VAR} is filled from the environment (review apps) | | readonly | true for production: the CLI only reads, never signs in to back offices, never creates data | | pii | redact or pseudonymize: every run hides personal data; required: no run is created unless the scenario or kaloko start

--pii hides it | | blocked_paths | path globs a read-only run never opens | | allowed_post_paths | path globs a read-only run may still POST to (a cart) | | may_create_data | false forbids creating data (default true) | | access | protection in front of the whole environment: basic (user_env, password_env), headers (service tokens), client_certificate, origins that receive them | | accounts | people who sign in, keyed like the scenario's account: user or user_env, password_env, totp_env, storage_state (a sign-in saved by kaloko auth save) | | mail | the mailbox: provider (mailpit, script, none), url, user_env, password_env, address, script, options, auth | | app | the app under test per platform: android, ios, electron, desktop, windows | | serve, fixtures, default_token_mode | design environments: the folder of steps, JSON fixtures, the default token mode (tokens_mode, its older name, still works) | | token_modes | design environments: the token modes every step is rendered in (default every mode of the set; false only default_token_mode) | | origins | design environments: the addresses the design loads fonts, stylesheets and images from (https://fonts.googleapis.com); the organization allows them in Settings -> Security | | html_view | experimental: keep a static HTML view of every capture |

Evaluators

| Key | What it holds | |---|---| | evaluators.jev.api_key_env | the variable with your own evaluator key (default TYPESAFE_API_KEY) | | evaluators.jev.hosted | without a local key, evaluate through your plan's allowance (default true); false uses only your key | | evaluators.jev.thresholds | pass and fail scores | | evaluators.jev.model, base_url, max_state_chars, signals, concurrency | finer settings, rarely needed | | evaluators.vision | the second evaluator that looks at the screenshot (provider anthropic, key in ANTHROPIC_API_KEY); false turns it off | | evaluators.<name> | your own text evaluator: kind: openai_compatible, base_url, model, optionally api_key_env, concurrency (default 2), max_state_chars (default 24000; the page text is cut to it), timeout_ms, response_format (json_schema, json_object or none) | | evaluators.text | which evaluator answers the semantic questions first: jev (default) or the name of your own one | | evaluators.escalate_to | jev: answers of your own model that are not clear enough for a verdict are asked again of JEV. Only this way round | | evaluate.budget.calls | uncached JEV calls per evaluation (your key or through Kaloko), escalations included; calls to your own model are not counted. Steps go first, then each stack's sample, chosen by priority (pages failing a check, then those checked longest ago). What does not fit stays not_evaluated with a note; kaloko evaluate --budget N overrides it | | evaluators.triage | experimental: Clef names the kind of problem on each screen and in recording frames, advisory only. model (clef or the faster clef-flash), keep (shown from this probability, default 0.7), frames_per_step (0 skips recordings), cascade (let Clef settle criteria the text evaluator is unsure about); false turns it off |

A local model in Ollama answers first and hands the unclear answers to JEV:

```
evaluate:
budget: { calls: 300 }
evaluators:
ollama:
kind: openai_compatible
base_url: http://localhost:11434/v1 # LM Studio: http://localhost:1234/v1
model: qwen3:8b
text: ollama
escalate_to: jev
```

Your own evaluator is used only when text names it, and the page text goes only to its base_url. Its verdicts are marked "evaluated by your own model" on the canvas and in the pull request comment.

Docs and products

| Key | What it holds | |---|---| | docs.dir | docs sources (default docs) | | docs.changelog | the changelog file (default CHANGELOG.md) | | docs.style | keep-a-changelog or release-notes | | docs.version_source | auto, git-tag, package.json, release-please, changesets, semantic-release, pyproject, cargo, gemspec, manual | | docs.languages | languages of the docs; the first is the originals' | | docs.framework | auto, docusaurus, vitepress, mkdocs, nextra, plain | | docs.publish | who reads: members, domain or public (the last two need an admin) | | docs.images | viewport and locale (reader or a language) of the pictures | | products.<slug> | name (text or per language), version (source, path, pattern), docs, project or projects, modules, reference_env,

default_scenario, publish |

Without products, the project is the product.

Traffic data

traffic joins search and analytics numbers onto the pages a run walks, so a missing description on the best-selling product comes before the same mistake on a page nobody finds. It is off until you add it, and Kaloko asks Google for nothing unless gsc or ga4 is set.

```
traffic:
file: qa/traffic/search-console-pages.csv # an export from Search Console or GA4
gsc:
property: sc-domain:acme.com # or https://www.acme.com/
credentials_env: GOOGLE_APPLICATION_CREDENTIALS
ga4:
property: "123456789"
low_ctr: { min_impressions: 100, ratio: 0.5 }
```

| Key | What it does | |---|---| | file | one or more CSV exports, relative to the file that names them: Search Console's Pages table or a GA4 report by landing page. Kaloko reads English and Czech headers (Page, URL, Landing page, Stránka, Vstupní stránka; Clicks/Kliknutí, Impressions/Zobrazení, CTR, Position/Pozice, Sessions/Relace, Conversions or Key events/Konverze) and numbers as either language writes them (1 234, 1,234, 3,5 %) | | gsc.property | the Search Console property. Kaloko signs in as a service account, asks searchAnalytics.query for clicks, impressions, CTR and position by page over the 28 days that end two days ago (days changes the length), and caches the answer per run | | ga4.property | the GA4 property id. Sessions and key events (conversions) by landing page over the 28 days that end yesterday, through the Data API | | credentials_env | the variable with the path of the service account's JSON key, or the JSON itself (a CI secret). Default GOOGLE_APPLICATION_CREDENTIALS. Add the service account's e-mail to the Search Console property as a user and to GA4 as a viewer | | low_ctr | thresholds of HC127 (below): pages with at least min_impressions (default 100) whose CTR is below ratio (default 0.5) times the site's median |

Pages are matched by address: the host without www., the path without a trailing slash and the query without tracking parameters (utm_*, gclid, …); the scheme and the hash do not matter. A GA4 path (/cs/eshop/chair) matches that path when the stack walks one host; a stack across several hosts leaves such rows out and says so in stack.traffic.note. A walk on a host the data has no rows for at all (staging, a preview) finds production numbers by the path when the data names it on one host only. A CTR column without % signs is read as percentages when any value in it is above 1, otherwise as fractions. When a page is both in an export and in the API answer, the API's numbers are used, not the sum. The API window is counted from the start of the run, so every build of one run uses the same numbers.

What you get:

• every page of a stack has traffic: { clicks, impressions, ctr, position, sessions, conversions } in items.json, and the page index on the canvas gets Impressions and Clicks columns you can sort by;

• kaloko export findings adds those columns and impact, and lists the findings with the highest impact first. impact = weight × (impressions + 10 × (clicks + sessions) + 100 × conversions), where the weight is 3 for an error or a failed acceptance criterion and 1 for a warning. A page without numbers has impact 0;

• a stack with the seo pack gets HC127: pages with many impressions and a CTR below half the site's median CTR at similar positions (1–3, 4–10, 11–20, each with at least five pages to compare), which usually means the title and description in the results need a rewrite. Without traffic data the check is not there at all.

A source that cannot be read is reported on stderr and left out; the run still builds. The service account key never leaves your machine and is never written into the run; see Security and your data.

Secrets

Secrets never go into the config. Wherever one is needed, the config names an environment variable (password_env: STAGING_PASSWORD, { env: CF_ACCESS_CLIENT_SECRET }). Kaloko reads the value

from:

1. the environment of the process,
2. .env next to kaloko.config.yml,
3. ~/.config/kaloko/.env.

A variable already set wins over the files. Values are removed from captured HTML, page text and console output. kaloko doctor says which named variables are missing. The token for sharing is KALOKO_TOKEN unless share.token_env says otherwise; see Roles and tokens.

Health API for dashboards

A weekly SEO walk over a few thousand pages ends as one run with hundreds of findings. A dashboard does not want to download it and count again. These endpoints answer with the counts already made: per language, step (a stack step is one page template), check or pack, with the pages that are hit most, and what changed since the run before.

A token for the dashboard

Create a token with the read:health scope in Settings -> API tokens (on Enterprise also as a service account). It reads the endpoints on this page and the rest of health (GET …/health of a run, a scenario or a project, MCP get_health); every other request answers 403. A token with the read scope works too.

```
curl -H "Authorization: Bearer $KALOKO_TOKEN" \
"https://kaloko.app/api/v1/orgs/acme/scenarios/seo-weekly/latest/health/aggregate?by=locale,
step"
```

Endpoints

| Request | What it answers | |---|---| | GET /api/v1/orgs/<org>/scenarios/<id>/latest | The newest shared run of a scenario you can see: its id, date, verdict, health counts and the links below. ?environment=production picks the newest run on one environment | | GET …/scenarios/<id>/latest/health/aggregate | The aggregate of that run, in one call | | GET …/scenarios/<id>/latest/health/diff | The diff of that run, in one call | | GET /api/v1/orgs/<org>/runs/<run>/health/aggregate | Counts per group with the most affected pages, and the diff's counts | | GET /api/v1/orgs/<org>/runs/<run>/health/diff | New, fixed and still open results, each with the run that first saw it | | GET /api/v1/orgs/<org>/scenarios/<id>/findings | Open results grouped into findings by check, site, language and step, with their status (below) |

MCP has the same as get_health_aggregate, get_health_diff (with run_id, or scenario and environment) and list_findings.

What is counted

One result is one page in one language for one check. A page walked on several viewports counts once, with its worst verdict. A stack step counts each of its pages; a plain step is its own page. A result is pass, fail, needs_review (the evaluator was unsure) or not_evaluated. Checks that do not apply to a page are left out.

Health is advisory: these numbers never change a step's or a run's verdict.

Aggregate

GET …/health/aggregate takes:

| Parameter | Meaning | |---|---| | by | Comma-separated: locale, step, check, pack. Default locale,step,check. Empty groups everything into one | | locale, step, check, pack | Only results with this value | | top | Affected pages per group, 0 to 20 (default 5); failing pages come before unsure ones | | limit | Groups in the answer, 1 to 5000 (default 500), the most failing first |

```
{
"schema": "kaloko.health-aggregate.v1",
"run": { "id": "2026-10-12-seo-weekly-a1b2c3", "scenario": "seo-weekly", "title": "Weekly
SEO", "project": "web", "environment": "production", "started_at":
"2026-10-12T05:00:00.000Z", "canvas":
"https://kaloko.app/acme/runs/2026-10-12-seo-weekly-a1b2c3/#health", "aggregate_url": "…",
"diff_url": "…" },
"by": ["locale", "step", "check"],
"filter": {},
"totals": { "results": 16800, "pass": 15910, "fail": 812, "needs_review": 64,
```

```
"not_evaluated": 14, "pages": 2800 },
"checks": { "HC112": { "text": "The page carries structured data (JSON-LD)", "pack": "seo",
"level": "warning" } },
"diff": { "previous": { "id": "2026-10-05-seo-weekly-d4e5f6", "started_at":
"2026-10-05T05:00:00.000Z" }, "new": 14, "fixed": 3, "still_open": 859, "unchecked": 0,
"worse": 1, "expired": 0, "open": 876 },
"group_count": 168,
"truncated": false,
"groups": [
{
"id": "locale=pl;step=eshop;check=HC112",
"locale": "pl", "step": "eshop", "check": "HC112",
"results": 400,
"counts": { "pass": 210, "fail": 190, "needs_review": 0, "not_evaluated": 0 },
"affected": 190,
"top": [
{ "step": "eshop", "locale": "pl", "check": "HC112", "page": "sklep-kubek", "path":
"/pl/sklep/kubek", "url": "https://www.example.com/pl/sklep/kubek", "verdict": "fail",
"evidence": "no JSON-LD block on the page" }
]
}
]
}
```


• id of a group is made of its values, so the same group has the same id every week: use it as the key in your dashboard.

• affected is fail plus needs_review.

• checks names every check of the run with its text, pack and level (error or warning when it fails).

• diff is the summary of the diff below; previous is null for the first run of a scenario in an environment.

Diff

GET …/health/diff compares the results that fail or are unsure with what the scenario had open after its previous shared run in the same environment and project. Runs of the same scenario in another project are never compared.

A result is fixed only when this run checked it and it passed. A result this run did not check stays open and is listed as unchecked: a page outside this week's rotating sample (rotate), a language left out, a check the evaluator could not decide. It keeps its first_seen, and the next run that checks it decides. A result that goes 8 runs in a row or 60 days without a check is dropped: it is listed once more with "expired": true and counted in expired.

| Parameter | Meaning | |---|---| | state | new, fixed, still_open or unchecked; without it, all four lists | | locale, step, check, pack | Only results with this value | | limit, offset | Paging within each list: up to 2000 (default 200) |


```
{
"schema": "kaloko.health-diff.v1",
"run": { "id": "2026-10-12-seo-weekly-a1b2c3", "…": "…" },
"previous": { "id": "2026-10-05-seo-weekly-d4e5f6", "started_at": "2026-10-05T05:00:00.000Z"
},
"counts": { "new": 14, "fixed": 3, "still_open": 859, "unchecked": 0, "worse": 1, "expired":
0 },
"open": 876,
"truncated": false,
"new": {
"total": 14,
"items": [
{ "id": "eshop|pl|HC112|sklep-kubek", "step": "eshop", "locale": "pl", "check": "HC112",
"page": "sklep-kubek", "url": "https://www.example.com/pl/sklep/kubek", "severity":
```

```
"warning", "text": "The page carries structured data (JSON-LD)", "first_seen": { "run":
"2026-10-12-seo-weekly-a1b2c3", "at": "2026-10-12T05:00:00.000Z" } }
]
},
"fixed": { "total": 3, "items": [] },
"still_open": { "total": 859, "items": [] },
"unchecked": { "total": 0, "items": [] }
}
```


• id of a result is step|locale|check|page. It stays the same while the step, language, check and page keep their ids, so you can follow one page across weeks. page is null for a plain step.

• severity is error or warning for a failing check and info when the evaluator was unsure. A still open result whose severity changed has was with the earlier one; worse counts those that got worse.

• first_seen is the run where the result started failing and has not passed in any run since. A result that was fixed and came back starts again.

• open counts what is open after this run: failing or unsure now, plus unchecked.

• A site with more than 50,000 open results in one run keeps the worst 50,000 and says truncated: true; the rest count as unchecked, never as fixed.

Findings

Eight hundred failing pages are usually a dozen problems: one template in one language misses its structured data. GET /api/v1/orgs/<org>/scenarios/<id>/findings groups what the scenario has open into findings: one check on one site in one language on one step. The fingerprint is check|site|locale|step, where the site is the origin of the page's URL (https://www.example.com). It stays the same from week to week, so a finding is counted, decided and tracked once, however many pages it hits. The read:health token reads findings too (MCP list_findings).

Parameter	Meaning
environment	The environment; without it, the newest one with recorded health
locale, step, check	Only findings with this value
status	open, acknowledged, ignored or fixed
limit, offset	Paging: up to 1000 (default 200)


```
{
"schema": "kaloko.health-findings.v1",
"scenario": "seo-weekly", "project": "web", "environment": "production",
"run": { "id": "2026-10-12-seo-weekly-a1b2c3", "started_at": "2026-10-12T05:00:00.000Z",
"canvas": "…" },
"page": "https://kaloko.app/acme/health/seo-weekly?environment=production",
"counts": { "open": 9, "acknowledged": 2, "ignored": 1, "fixed": 3 },
"filter": {}, "total": 15, "offset": 0, "limit": 200,
"findings": [
{
"id": "5f0c2a9e1b7d4c33",
"fingerprint": "HC112|https://www.example.com|pl|eshop",
"check": "HC112", "text": "The page carries structured data (JSON-LD)", "pack": "seo",
"site": "https://www.example.com", "locale": "pl", "step": "eshop",
"severity": "warning", "status": "acknowledged", "reason": null,
"status_by": "jana@example.com", "status_at": "2026-10-12T09:14:00.000Z",
"count": 190, "unchecked": 0, "fixed": 4,
"samples": [
{ "page": "sklep-kubek", "url": "https://www.example.com/pl/sklep/kubek", "edit_url":
"https://cms.example.com/pages/812/edit" }
],
"first_seen": { "run": "2026-09-28-seo-weekly-77aa01", "at": "2026-09-28T05:00:00.000Z" },
"last_seen": { "run": "2026-10-12-seo-weekly-a1b2c3", "at": "2026-10-12T05:00:00.000Z" },
"issue": { "tracker": "jira", "key": "WEB-412", "url":
"https://acme.atlassian.net/browse/WEB-412", "state": "open" }
}
```

```
]
}
```

• id is a short hash of the fingerprint; use it in links and in PATCH.

• severity is the worst of the finding's results; count is the pages it affects now. unchecked counts pages the newest run did not check (they stay open), fixed the pages the newest run fixed.

• samples lists up to 10 pages with the edit_url their stack item names, so a content team goes straight to the CMS (Find what fails on a site).

• first_seen is the earliest first seen of its results. last_seen is the newest run that checked it and still saw it fail; for a fixed finding it is the run before the fix.

• status is fixed once none of its pages fails; otherwise what a person set (acknowledged, ignored with a reason) or open. A decision stays with the fingerprint: an ignored finding that comes back is still ignored.

• With traffic: on the stack (Fix the pages people actually see first), a finding has traffic (clicks and impressions added up over its pages that have numbers, and how many pages those are) and impact, and each sample its own traffic. Without traffic data neither key is there.

• issue.key is WEB-412 in Jira or Linear, acme/website#87 in GitHub and MKT-12 in Plane.

• Findings are worst first, the fixed ones last.

To acknowledge a finding or ignore it, send PATCH …/scenarios/<id>/findings/<finding>[?environment=] with { "status": "acknowledged" }, { "status": "ignored", "reason": "Old blog, moving to a new CMS in November" } or { "status": "open" }. Anyone who may comment on the scenario's runs may do it; a token needs the comment scope (a read:health token never changes anything). The app shows the same table on Health -> Findings of each scenario, with buttons for the same three choices.

Issues for findings

A scenario can open an issue for each new finding in Jira, Linear, GitHub Issues or Plane. A Developer / Tester switches it on with PUT …/scenarios/<id>/findings/sync:

```
{ "tracker": "jira", "min_severity": "error", "environment": "production" }
```

In Jira and Linear the issue goes to the project or team the scenario's project maps to in Settings -> Integrations (Connect trackers and channels). GitHub and Plane take the destination in the setting itself, and routes send some findings to another tracker than the scenario's:

```
{
"tracker": "github", "min_severity": "warning",
"github": { "repository": "acme/website", "labels": ["seo"], "assignee": "jana" },
"plane": { "project": "MKT", "labels": ["content"] },
"routes": [{ "tracker": "plane", "checks": ["HC127", "HC9*"], "packs": ["aeo"] }]
}
```

• github: the repository (owner/repo), labels and an optional assignee. Kaloko opens the issue as the Kaloko GitHub app when the app is installed on the repository and has the Issues permission, otherwise with the fine-grained token an admin connected (Issues: read and write on that repository).

• plane: the project identifier (MKT) and labels that already exist in the project.

• The repository and the Plane project must be on the list an organization admin allowed for the scenario's project in Settings -> Integrations (a project without its own row uses the default row). Anything else is refused with 403 before GitHub or Plane is asked, so the answer says nothing about whether it exists. When an admin takes a destination off the list, no new issues go there; the ones that exist are still followed.

• routes: at most 20, in order. The first route that names the finding's check (HC127, or a prefix such as HC1*) or its pack (seo, aeo, …) decides; every other finding goes to tracker. Each tracker a route names must be connected. When the routes change, an issue that exists stays where it is and is followed there.

Saving checks the repository, the assignee, the project and the labels with the tracker and says what is wrong.

After each shared run of the scenario Kaloko creates one issue per new finding at or above min_severity (error, warning or info), never two for one fingerprint and none for an ignored one. The issue says which check fails

where, how many pages, up to ten of them with their edit links and, with traffic:, the clicks and impressions of each page and of the whole finding with its impact; it links back to the finding. When the number of affected pages changes by at least three and a quarter, the issue gets a comment; when the finding is fixed, a comment and the issue is closed (Jira: the first transition into Done; Linear: the team's first completed state; GitHub: closed as completed; Plane: the project's first completed state). If it comes back, the same issue gets a comment and is opened again where the tracker allows it (Plane: the first unstarted state).

A run creates at most 20 issues, the worst first, and makes at most 60 calls to the trackers within 20 seconds, of them at most 25 requests to Plane, each counted (its API takes 60 a minute; closing an issue also reads the project's states once per run); the rest follow with the next run. A finding the tracker refuses (a field it does not accept) keeps the error in the list and the others go on. When a tracker refuses the connection, asks Kaloko to slow down or does not answer, the run stops for that tracker and the next run tries again; findings routed to another tracker go on. A create that never reached the tracker (a connection that could not be renewed, a Linear team that is not there, a Plane address that is refused) lets the finding try again with the next run. If a tracker does not say clearly whether it created an issue (a timeout or a server error), the finding shows the issue as being created, with the reason in issue.error, and Kaloko does not open another one: check the tracker, and if nothing is there, choose Try again at the finding on the findings page or send POST …/scenarios/<id>/findings/<finding>/retry[?environment=] (Developer / Tester; once the issue has waited two minutes, so a create still on its way is never cut off); the next run creates it. Saving the setting also lets go of such claims older than an hour. A scenario whose trackers take no new issues (a destination taken off the list, a mapping removed) still comments on, closes and reopens the issues it opened.

The setting belongs to the scenario in its project: moved to another project, the scenario stops opening issues until someone switches it on there. { "tracker": null } switches it off; GET on the same address shows the setting and the trackers the organization can sync to.

Runs shared before

The counts of a run are made once, after it is shared. A run shared before this API existed is counted on each request instead, which can take a few seconds for a large run; its diff compares it with the run before it and says "recorded": false. The first run shared after that starts the history of its scenario.

When something changes

The health.changed webhook sends the diff's counts and the first new and fixed results after every shared run where something moved: Webhooks.

Health checks

Every pack and every check it runs, listed from the same code the CLI runs, so the list matches the version you install. Health checks are advisory: the canvas shows them on the Health tab, and they never change the verdict of a step or a run. When a check matters to your team, adopt it as a criterion (kaloko health adopt <step>:<check>); from then on it counts for acceptance.

A step turns packs on with packs: [seo, a11y] in the scenario file (Add criteria and packs). The kinds of check, their options and the ids are in Criteria, checks and packs.

Severity of a failed check: an error when people or search engines are shut out, a warning otherwise. An answer the evaluator was unsure about shows as unsure.

SEO basics (seo)

What search engines read from a page: the title, description, canonical, hreflang, Open Graph, robots, viewport and structured data. On a stack step it also compares the pages with each other.

Off on design environments, where the steps are your own HTML served from a folder. To keep it on there, list it in health: { design_packs: [...] } in the scenario.

Id	Check	Why it matters	Severity	Applies to
HC101	Exactly one H1	Search engines and screen readers take the H1 as the name of the page; none or several make it unclear.	warning	every step with the pack
HC102	Title between 10 and 70 characters	The title is the blue link in search results; too short says nothing, too long gets cut.	warning	every step with the pack
HC103	Meta description between 50 and 160 characters	The meta description is the snippet under the link; without it search engines pick random text.	warning	every step with the pack
HC104	Canonical link present	Without a canonical URL, copies of the page (parameters, filters) compete with each other in search.	warning	every step with the pack
HC105	hreflang for every language	hreflang tells search engines which language version to show to whom.	warning	scenarios with more than one locale
HC106	Open Graph title and description	Shared links on social networks and chats use Open Graph for their preview.	warning	every step with the pack
HC107	Not excluded from indexing	A noindex robots tag keeps the page out of search results entirely.	error	every step with the pack
HC108	Viewport meta for phones	Without the viewport tag phones show a zoomed-out desktop page.	error	every step with the pack
HC109	html lang attribute set	Screen readers pick the voice and search engines the language from <html lang>.	warning	every step with the pack
HC110	No horizontal overflow	Content wider than the screen makes phones scroll sideways and breaks the layout.	error	every step with the pack
HC111	No console errors	Console errors often mean a feature silently does not work.	error	every step with the pack
HC112	Structured data (JSON-LD) present	Structured data tells search engines and AI assistants what the page is (a product with a price, an article, a FAQ) and earns rich results.	warning	every step with the pack
HC113	Every JSON-LD block is valid JSON	A block that does not parse is ignored whole: the price, breadcrumbs or FAQ in it never reach search results.	warning	every step with the pack
HC121	Titles are unique within a language	Pages with the same title look like duplicates to search engines, which then pick one and hide the others.	warning	stack steps, across their pages
HC122	Meta descriptions are unique within a language	A description copied across pages tells searchers nothing about the one they are about to open.	warning	stack steps, across their pages
HC123	H1 headings are unique within a language	Two pages with one H1 compete for the same search and confuse people who land on the wrong one.	warning	stack steps, across their pages
HC124	Every page is its own canonical, and no canonical points to an error or noindex page	A canonical that points elsewhere asks search engines to drop the page; pointing at an error or noindex page drops both.	warning	stack steps, across their pages
HC125	hreflang alternates answer back: each language version links to the others and to itself	Search engines ignore hreflang that is not confirmed from the other side, so the wrong language shows up in results.	warning	stack steps, across their pages
HC126	Pages listed in the sitemap answer 200 without a redirect and are indexable	A sitemap is the list of pages you want found; redirects, errors and noindex in it waste the crawl and erode trust in the whole list.	warning	stacks read from a sitemap
HC127	Pages seen often in search are not clicked far less than the site's other pages at similar positions	Searchers see the page but pick another result: the title and description in the results do not promise what they are looking for.	warning	stacks with traffic data

Answer engines (aeo)

What AI search and assistants pick up: FAQ structured data, a short answer under the H1, llms.txt, a meta description, and the page as GPTBot gets it without JavaScript. Cheap signals from the page, not a ranking in any assistant.

Off on design environments, where the steps are your own HTML served from a folder. To keep it on there, list it in health: { design_packs: [...] } in the scenario.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC901 | Questions and answers on the page have FAQPage or QAPage structured data | Answer engines and rich results pick up questions and answers much more reliably when the page marks them as FAQPage or QAPage. | warning | every step with the pack | | HC902 | A short answer-first paragraph follows the H1 (40 to 320 characters) | AI assistants quote the first clear answer they find; a short paragraph right under the heading is the one they take. | warning | every step with the pack | | HC903 | llms.txt is reachable at the site root | llms.txt gives language models a short map of the site: what it offers and which pages to read. | warning | every step with the pack | | HC904 | Meta description present | Without a meta description, search engines and assistants write their own summary from random text. | warning | every step with the pack | | HC905 | Without JavaScript, GPTBot gets the title, canonical, H1, structured data and most of the text | Most AI crawlers do not run JavaScript. What appears only after scripts run does not exist for them, even if the page looks complete in a browser. | warning | every step with the pack |

Accessibility (a11y)

The part of WCAG 2.2 AA that can be measured on the page, a walk with the Tab key (without clicking or typing) and the accessibility tree. It does not replace an audit with a screen reader.

Runs on design environments too.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC301 | Images have alternative text | Without alt text, blind visitors and search engines do not know what an image shows. | error | every step with the pack | | HC302 | Links and buttons have an accessible name | An icon button without a name is read as just "button". | error | every step with the pack | | HC303 | Form fields have labels | Screen reader users hear only "edit text" for a field without a label. | error | every step with the pack | | HC304 | Heading levels are not skipped | People navigate by headings; a jump from H2 to H4 hides the structure. | warning | every step with the pack | | HC305 | Text meets WCAG AA contrast | Low contrast text is hard to read in sunlight and for people with weaker sight. | error | every step with the pack | | HC306 | The page has a main landmark | Screen reader users jump straight to <main> to skip the navigation. | warning | every step with the pack | | HC307 | Element ids are unique | Duplicate ids break labels, anchors and scripts that look elements up by id. | warning | every step with the pack | | HC308 | Tab order follows the page and reaches every control | People who use a keyboard move with Tab; focus that jumps around or skips a button leaves them lost or stuck. | warning | every step with the pack | | HC309 | Keyboard focus is clearly visible (at least 3:1 contrast) | Without a visible focus ring, keyboard users do not know where they are on the page. | warning | every step with the pack | | HC310 | No keyboard trap | A widget that keeps Tab inside (an embed, a custom menu) traps keyboard users for good. | error | every step with the pack | | HC311 | A skip link or an early way into the main content | Without a skip link, keyboard and screen reader users tab through the whole menu on every page. | warning | every step with the pack | | HC312 | Controls have a name in the accessibility tree | The accessibility tree is what screen readers read; a control without a name there is announced as just "button". | error | every step with the pack | | HC313 | Landmarks are complete and labelled | Screen reader users jump between header, navigation, main and footer; duplicates without names all sound the same. | warning | every step with the pack | | HC314 | ARIA roles and attributes are valid | Wrong ARIA is worse than none: screen readers trust it and announce things that are not there. | warning | every step with the pack |

Usability (usability)

Questions for the evaluator, picked by the page type of the step (page: landing, work, list, settings, form, doc), and labelled form fields.

Runs on design environments too.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC201 | The purpose of the page is clear within the first screen | Visitors decide within seconds whether they are in the right place. | warning | page types landing, doc | | HC202 | One clear primary action | Several equally loud buttons make people hesitate; none leaves them without a next step. | warning | page type landing | | HC203 | Form fields have visible labels | A placeholder disappears as people type; they forget what the field was. | warning | every step with the pack | | HC204 | Links and buttons say what they do | "Click here" or "More" does not say where it leads, especially out of context or to a screen reader. | warning | every step with the pack | | HC205 | The heading says what the screen is for | People switch between screens all day; a vague heading makes them check where they are. | warning | page types work, list, settings, form | | HC207 | Each item in the list says what it is and how to open it | A row with only an id or a date makes people open items one by one to find the right one. | warning | page type list | | HC208 | Each setting says what it changes and how it is saved | People leave settings unsure whether the change took effect. | warning | page type settings | | HC206 | It is clear how to send the form and what happens next | When the send button or the next step is unclear, people submit twice or give up. | warning | page type form | | HC209 | The text can be scanned: headings and short paragraphs | Readers of documentation scan for their question; a wall of text hides the answer. | warning | page type doc |

Performance (perf)

Lab values measured in the walk's browser: LCP, CLS, TTFB and page weight. Real visitors on slower phones may see other numbers.

Off on design environments, where the steps are your own HTML served from a folder. To keep it on there, list it in health: { design_packs: [...] } in the scenario.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC401 | Largest contentful paint within 2.5 s | LCP is how long people wait before the page looks ready; Google uses it for ranking. | warning | every step with the pack | | HC402 | Cumulative layout shift below 0.1 | Content that jumps while loading makes people tap the wrong thing. | warning | every step with the pack | | HC403 | Time to first byte within 800 ms | A slow first byte delays everything else on the page. | warning | every step with the pack | | HC404 | Page weight within 3 MB | Heavy pages are slow and expensive on mobile data. | warning | every step with the pack |

Console (console)

The evaluator reads the console errors of the step and says whether they touch what the step accepts or are noise from third-party scripts.

Off on design environments, where the steps are your own HTML served from a folder. To keep it on there, list it in health: { design_packs: [...] } in the scenario.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC501 | Console errors do not touch what the step accepts | Separates real failures from third-party noise (analytics, extensions). | error | every step with the pack |

Error pages (errors)

For 404, 410, 500 and maintenance pages: a real error status, a way back, and no technical detail leaking out. Meant for steps that are error pages.

Off on design environments, where the steps are your own HTML served from a folder. To keep it on there, list it in health: { design_packs: [...] } in the scenario.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC601 | Answers with an error status (4xx/5xx), not a soft 200 | A missing page that answers 200 gets indexed as real content and hides broken links from monitoring. | error | error pages only (expect_status, or a page answering 4xx/5xx) | | HC602 | Links back to the home page | A dead end loses the visitor. | warning | every step with the pack | | HC603 | Navigation or search stays available | People on an error page want to find what they came for. | warning | every step with the pack | | HC604 | No stack trace, SQL error or framework debug page | Stack traces leak paths, versions and

queries that help attackers. | error | every step with the pack | | HC605 | Explains the error in plain language and offers a way forward | A calm explanation keeps people on the site instead of leaving. | warning | error pages only (expect_status, or a page answering 4xx/5xx) | | HC606 | No console errors | Scripts failing on the error page often break its navigation too. | error | every step with the pack |

Language versions (locale)

For multilingual sites: html lang, the page's own hreflang, the language of the text and the decimal separator of prices match the walked locale. Only in scenarios that list their locales.

Runs on design environments too.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC1001 | html lang matches the walked locale | Screen readers pick the voice and search engines the language from html lang; a German page marked as English is read with an English accent. | warning | every step with the pack | | HC1002 | The page's own hreflang names it with the walked locale, codes are valid and agree with the addresses | A wrong own hreflang or a code such as en_UK makes search engines skip the whole set of language versions. | warning | every step with the pack | | HC1003 | The text is in the language of the walked locale | An untranslated page in a language version loses its visitors and looks like duplicate content of the source language. | warning | every step with the pack | | HC1004 | Prices use the decimal separator of the walked locale | 12.50 ? reads oddly to a German or Czech customer and 12,50 $ to an American; it looks like the shop was not made for them. | warning | every step with the pack | | HC1005 | Every page has its own language version (hreflang) in each walked locale | A page without its translation sends visitors of that language to the wrong one, and search shows them whatever is left; several pages pointing to one version usually mean a fallback to a listing or the home page. | warning | stacks with items.variants: hreflang |

App screens (app)

Native apps on Android, iOS and desktop: labelled controls, touch targets, cut-off text, crashes and a clear next step.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC701 | Every control has an accessible label | VoiceOver and TalkBack read unlabeled controls as "button". | error | every step with the pack | | HC702 | Touch targets at least 48 dp (Android) / 44 pt (iOS) | Small targets are missed, especially by thumbs and older people. | warning | every step with the pack | | HC703 | No truncated texts (…) | Cut text hides prices, names or instructions, often only in longer languages. | warning | every step with the pack | | HC704 | The app did not crash or stop responding | A crash or an "app not responding" dialog ends the session. | error | every step with the pack | | HC705 | The purpose of the screen and the next action are clear | On a small screen, people need to see at once what to do next. | warning | every step with the pack |

Design reference (design)

The implementation against the accepted design version: pixels, structure and tokens. The only pack that always counts for acceptance, so its ids are AC.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | AC801 | Looks like the design reference (under 3 % of pixels differ) | Visible differences from the accepted design mean it was built differently than agreed. | acceptance criterion | every step with the pack | | AC802 | Same structure as the design reference: headings, landmarks, actions and form fields | Missing headings, actions or fields change what people can do, even if it looks close. | acceptance criterion | every step with the pack | | AC803 | Colors, fonts, sizes, spacing and radii come from the token set | Raw values drift from the design system and make later changes expensive. | acceptance criterion | every step with the pack |

Site checks (site)

Once per site and run, not per step: robots.txt, the sitemaps it lists, llms.txt and the redirects to https and to one host name. Off unless the scenario sets health: { site: true }, because they make requests of their own. They

cannot be adopted as criteria.

| Id | Check | Why it matters | Severity | Applies to | |---|---|---|---|---| | HC1101 | robots.txt answers 200 and every line parses | Crawlers skip lines they do not understand, so a typo such as "Disalow" silently opens or closes part of the site; a server error stops crawling altogether. | warning | once per site | | HC1102 | robots.txt does not block the walked start pages for search engines | A Disallow that covers the home page or a language root removes the site from search, often after a staging setting reached production. | error on production, warning elsewhere | once per site | | HC1103 | robots.txt does not block AI crawlers (GPTBot, OAI-SearchBot, ChatGPT-User, PerplexityBot, ClaudeBot, Google-Extended, CCBot) | GPTBot, ClaudeBot, PerplexityBot and others read robots.txt; blocked, the site does not appear in AI answers and assistants cannot open its pages for users. | warning | once per site, on production only | | HC1104 | Every sitemap answers, is valid XML, and lists absolute same-host addresses with sane lastmod dates | A broken or cut-off sitemap, relative addresses or another host make search engines ignore it, and made-up lastmod dates teach them to ignore those too. | warning | once per site | | HC1105 | llms.txt is reachable at the site root | llms.txt gives language models a short map of the site: what it offers and which pages to read. | warning | once per site | | HC1106 | http:// redirects to https:// in one hop | Each extra redirect costs time on every first visit, and a page served over http:// splits links and rankings between two addresses. | warning | once per site | | HC1107 | The other host name (with or without www) redirects in one hop | When both example.com and www.example.com answer, search engines see two copies of the site; a chain of redirects slows every visit that comes the other way. | warning | once per site |

MCP tools

Kaloko's MCP server lets an agent read runs, feedback, designs, the changelog and docs, and answer reviewers. It is part of the Team plan and up. Capturing, evaluating and uploading stay in the CLI.

Connect

The server is https://kaloko.app/mcp (MCP over Streamable HTTP). Authenticate with an organization API token in the Authorization header; how to create one: Roles and tokens.

```
claude mcp add --transport http kaloko https://kaloko.app/mcp --header "Authorization: Bearer $KALOKO_TOKEN"
```

Other agents (Codex, Cursor) take the same URL and header as an HTTP MCP server. A read-only token is enough to read; tools that write need a token with that scope, as noted below. A read:health token (for a dashboard) sees get_health, get_health_aggregate, get_health_diff and list_findings only. Screenshots and files are not sent over MCP: the answers carry their URLs, which the same token can fetch.

list_runs, list_notifications and list_comments answer one page at a time. When more match than limit, the answer has next; call the tool again with cursor set to it. The REST API pages the same lists with ?cursor= and names the next page in the X-Next-Cursor and Link headers.

Runs and review

| Tool | What it does | |---|---| | get_started | where the organization stands, its sample run and the next steps | | add_sample_run | adds the sample run, or returns the one there is | | list_projects | projects of the organization with their slugs | | upsert_project | creates a project with its name, or renames it (the project's admin), and adds it to a product; like kaloko projects create, rename and sync | | list_runs | shared runs, newest first, filtered by scenario, project, verdict, author, environment, branch, ref (a record the run serves, such as CC-2026-001) or purpose (dry_run, formal, smoke); cursor for the next page | | get_run_summary | one run: statuses per step and language, criteria that need attention with evidence and reasons, screenshot URLs, the canvas link, the flow's lanes and sessions (roles) and each step's lane and session | | get_stack_items | the pages of a stack step with their status and failing criteria | | get_step_assets | the newest capture of every step of a scenario at stable URLs (Business); state picks a step state | | get_step_recording | a step's video and trace; state picks a step state | | get_criteria_verdicts | reviewers' verdicts on manual or overridden criteria | | get_review_queue | what a person still has to decide in a run, and the review link to pass on | | decide_step | the agent's own approve or return on a step; marked as an agent's, never counted as a person's | | revoke_carried_approval | takes back an approval carried over to a step that should be looked at again | | get_signoff_record | the signed acceptance record of an accepted run (Business) | | get_evidence | the evidence of a run or a release for your own documents: acceptance and signing state, criteria as walked with requirement references, results, screenshot hashes, decisions, signatures, audit trail and the trace matrix (Business) | | get_health | the health of a run, the newest run of a scenario, or the trend of a project: findings by severity, pack and step, and what is new or fixed since the previous run; advisory, never a verdict | | get_health_aggregate | health of a run (or a scenario's newest run) counted for a dashboard: per locale, step, check or pack how many results pass, fail or are unsure, the most affected pages, and how many are new or fixed since the previous run; a read:health token may call it | | get_health_diff | per page, locale and check what is new, fixed or still open since the previous run, with the run that first saw it; a read:health token may call it | | list_findings | the open results of a scenario grouped into findings by check, site, locale and step: severity, pages affected, sample pages with their CMS edit link, first and last seen, status (open, acknowledged, ignored with the reason, fixed) and the tracker issue; read-only, a read:health token may call it | | adopt_health_check | turns a health check of a step (HC103) into the acceptance criterion AC103 of the scenario, as is or with new text (Developer / Tester); kaloko health pull writes it into the scenario file | | get_affected_steps | which steps of a scenario the changed files reach, and why | | list_notifications | the token owner's inbox |

Feedback and issues

| Tool | What it does | |---|---| | get_feedback | the run's verdict, approved steps, open returns and comments | | list_comments | open threads across the organization, with replies, pins and drawn regions | | add_comment | a comment on a step, a run or a whole flow, or a reply | | resolve_comment | closes a thread with a note on what changed | | reopen_comment, edit_comment, delete_comment | reopen a thread; edit or delete your own comment | | get_findings | experimental triage findings of a run (the kind of problem on each screen) and the answers people gave | | answer_finding | marks a finding right, not a problem, or turned into a comment | | get_triage_report | how triage does in your organization: useful findings per kind, how well its probabilities hold, findings turned into comments, time to the first review with and without findings | | list_ignore_regions | regions people drew to leave out of pixel diffs | | add_ignore_region | suggests such a region | | create_issue | a Jira or Linear issue from a step, with the screenshot, criterion and notes | | list_issues | issues created from steps, with their state | | list_schedules, run_schedule | hosted runs and their page budget; start one now |

Design

| Tool | What it does | |---|---| | get_draft | which revision each step of the shared draft shows, and who changed it | | get_step | the handoff of a step: live files, tokens, page structure, open comments; each capture names its step state and token mode | | get_step_spec | what to build for a step as approved: elements, styles mapped to tokens, states, contrast; state for an empty, error or other step state | | get_step_drift | how an implementation run differs from the design reference, in one step state (state) | | get_step_history | versions and revisions of a step with their authors | | list_step_changes | steps that changed between two references or since the previous accepted run | | get_references, get_tokens | the design reference and baseline; the project's token set | | get_draft_feedback | comments on the draft | | get_version_notes, set_version_notes | read or rewrite what changed in a version | | publish_version, set_reference, set_step_status, add_draft_comment | publish the draft as a version, set the reference, set a step's status, comment on the draft |

Screens, changelog, docs and products

| Tool | What it does | |---|---| | search_screens | the screen library: the newest screenshot of every step, searched by words, project, product, language or status | | list_products | the products you read, with their newest release, docs and readers | | get_changelog | a product's visual changelog, optionally with the pictures resolved | | draft_changelog_entry | what accepted runs since the last release show differently, as a draft entry with before and after references | | resolve_step_image | one step reference (kaloko:<scenario>/<step>) as an image URL, with the step's recording when it plays | | get_docs | a product's published docs: versions, languages, the sidebar, or one page | | docs_check | what needs attention in the published docs: broken references, stale sections, missing translations |

Errors come back as tool results marked as errors (an unknown run, a missing argument). A token that is not valid gets HTTP 401; an organization below Team is told the MCP server is not in its plan.

Scenario file

A scenario is a YAML file (usually in qa/flows/) that describes one flow: its steps, the arrows between them and what each step must meet. Its format is qawalk.scenario.v1; kaloko validate <file> checks a file against it. How to write one step by step: Write a scenario.

```
schema: qawalk.scenario.v1
id: WEB-12-checkout
title: Checkout
locales: [en, cs]
viewports: { desktop: [1280, 800], mobile: iphone-15 }
environments: [local, staging]
nodes:
- id: cart
kind: screen
title: { en: Cart, cs: Košík }
path: /cart
purpose: The shopper sees what they are buying and can continue.
execution: { strategy: playwright, script: steps/cart.mjs, instructions: Add one product and
open the cart. }
criteria:
- { id: AC1, text: Shows the total, check: { type: deterministic, assert: exists, selector:
.cart-total } }
- id: payment
kind: screen
title: Payment
path: /checkout/payment
packs: [a11y]
edges:
- { from: cart, to: payment }
```

Top-level keys

schema, id, title, nodes and edges are required.

| Key | What it holds | |---|---| | schema | always qawalk.scenario.v1 | | id | the scenario's id (lowercase letters, digits, dashes); a ticket in it (WEB-12-…) links the issue | | title | a name for people | | kind | task (default) or flow | | archived | true hides a retired scenario from the list; its runs stay reachable | | source | path to the acceptance plan (ACCEPTANCE.md), used by kaloko validate --coverage | | version | a number you raise when the flow changes | | locales | languages to walk, e.g. [en, cs] | | viewports | names with a size ([1280, 800]), a device preset (iphone-15, ipad-landscape), or a list of preset names | | browsers | chromium (default), webkit, firefox | | color_schemes | light, dark | | reduced_motion | no-preference, reduce | | forced_colors | none, active (Windows high contrast) | | display_modes | browser, standalone (an installed web app) | | devices | app scenarios: the emulators, simulators or phones to walk on | | token_modes | design environments: which modes of the token set to render ([brand-a, brand-b]), or false for the environment's default_token_mode only; not set: every mode | | environments | environments of the config the scenario may run on | | readonly | true for a scenario that only reads; required on a read-only environment | | platform | web (default), android, ios, electron, desktop, windows | | health | acceptance: [seo]: packs whose checks count for acceptance in this scenario; without it pack checks are advisory health (Check the health of your app). site: true (or { paths, ai_crawlers, sitemaps, environments }) adds the site checks once per run: robots.txt, sitemaps, llms.txt, redirects (Criteria, checks and packs). design_packs: [console]: code-only packs (seo, aeo, perf, console, errors) that still run on design environments, where they are off otherwise | | traffic | Search Console and GA4 numbers for this scenario's pages; replaces the config's traffic (Configuration file) | | tags | free labels | | refs | requirement references the scenario verifies, as your tracker or QMS writes them ([URS-12, FS-3.4]); they reach every criterion in the trace matrix (quality evidence) | | nodes | the steps (below) | | edges | arrows between steps (below) | | lanes | labels of the rows on the canvas, index = lane number (a link ending in #lane=<number> opens that row) | | issue | the ticket: number, repo (owner/name), key (Jira or Linear, ENG-42), url, title | | sessions | people in the flow, each with title, optional

mailbox, account, blocked_on | | project | the Kaloko project for its runs; overrides the config. When the config asks every scenario for its own project (a comment, or a product with projects:), kaloko validate and kaloko start warn when it is missing | | product, module | which product and module of products: the flow shows, for the changelog and docs | | capture | capture quality for every step (below) | | pii | redact or pseudonymize: personal data hidden in every capture, found automatically (how-to) |

Step keys (nodes)

Every step needs id, kind and title.

| Key | What it holds | |---|---| | id | the step id, used in links, --steps and step references (kaloko:<scenario>/<step>) | | kind | screen, email, external (a third-party screen), decision, stack (many pages of one template) | | title | text, or one per language ({ en: Cart, cs: Košík }) | | path | the address relative to base_url, or one per language; a deep link in app scenarios; {random} makes an address that cannot exist | | purpose | one sentence about what the step is for; the evaluator reads it | | refs | requirement references of the step; they reach its criteria in the trace matrix | | lane | the row on the canvas | | readonly | this step only reads | | execution | strategy (playwright, script, agent, manual), script (path to the step script), instructions (required), requires (e.g. browser) | | mail | e-mail steps: subject and to | | criteria | what the step must meet (below) | | session | which person from sessions does this step | | affects | file globs that shape the step, for kaloko walk --affected, or always | | packs | ready-made sets of health checks: seo, usability, a11y, perf, console, errors, app, aeo, locale, and design, which counts for acceptance | | expect_status | the HTTP status the step should answer with, e.g. 404 (or a list). A status of 400 or more marks an error page: the errors pack checks HC601 (soft 404) and HC605 (the explanation) only on error pages, that is with expect_status or on a 4xx/5xx answer | | capture | capture quality for this step only | | record | true records this step even when the walk does not; false never | | ignore | regions left out of pixel comparisons: { selector } or { rect: [x, y, w, h], viewport }, optionally limited by viewports, locales, with a note | | items | stack steps: where the pages come from (urls, file, sitemap (gzipped too; sitemap: robots or robots: true reads the sitemaps of /robots.txt for the walked locale) with match/exclude/limit, crawl, script). match and exclude are a regex or a map by locale ({ default: '^/eshop/', fi: '^/fi/tuotteet/' }) and apply before limit for every source; rotate: week or rotate: run takes limit pages from a different place each week or run, so a big list is covered in turns; variants: hreflang (with source_locale) takes the list in one locale and walks each page's hreflang alternate in the others, so a page and its translations are one page of the stack and a missing translation is reported (HC1005), in kaloko walk and hosted runs alike; params go to script, which also gets node (id, title, lane, lane_title, session); a script or a JSON list may give each page edit_url (the page in the CMS) and content_ref (its record), and resolve: steps/resolve.mjs (default export (item, ctx) => ({ edit_url, content_ref })) adds them to the walked pages, for the stack index and kaloko export findings; on a design environment the variants are folders of the step, named by urls or found as subfolders with an index.html | | fields | stack steps: up to 12 columns of the stack index (id, label, from: title, h1, description, canonical, lang, words, http, jsonld_types, a meta tag such as og:title or meta:robots, a CSS selector, selector@attribute, count:selector) | | details | named parts of a screen step for guides: { id, name, selector }, optionally context (section, card, page or a CSS selector), padding and states; shown in docs with kaloko:<scenario>/<step>#<id> (Publish living docs) | | states | other states the step is captured in, e.g. [empty, error, loading]; any name works (below) |

Criteria

Each criterion has an id (AC1, AC12b), a text for people, an optional highlight (a CSS selector to outline on the screenshot), optional refs (requirement IDs it verifies, free text) and exactly one check: deterministic, semantic or manual. Every check type and option is listed in Criteria, checks and packs.

environments: [production] limits a criterion to some environments, states: [empty] to some step states. On the other ones it shows as not applicable and does not make the step partial.

Step states

A cart is not one screen: it can be empty, fail to load or still be loading. states: on a step captures each of them

next to the default one:

```
- id: cart
kind: screen
title: Cart
path: /cart
states: [empty, error, loading]
execution: { strategy: playwright, script: steps/cart.mjs, instructions: Open the cart. }
criteria:
- { id: AC1, text: Shows the total, check: { type: deterministic, assert: exists, selector:
.cart-total } }
- { id: AC2, text: Says the cart is empty, states: [empty], check: { type: semantic,
question: Does the page say the cart is empty? } }
```

• A walk runs the step script once per state and passes the state as state ('default' first). The script puts the page into that state: an empty cart, a mocked API error, a slow response.

• A design step renders cart/empty.html for the state empty (per language cart/cs/empty.html). Without that page Kaloko opens cart/index.html?state=empty, so one page can switch its states itself.

• Each state has its own screenshots and verdicts. The canvas gets a State switcher next to the language, review mode and the grid show the state you pick, and a step without that state shows its default screen.

Edges

| Key | What it holds | |---|---| | from, to | step ids | | label | text on the arrow, e.g. the choice at a decision | | kind | main (default) or alt for an alternative path |

Capture

| Key | What it holds | |---|---| | format | webp (default) or png (lossless, for print) | | scale | 1 or 2 (device pixels of the screenshot) | | quality | WebP quality from 0.3 to 1 | | mask | personal data hidden in every output of the step: CSS selectors, auto (detection), or both; the scenario's and the step's add up (how-to) | | annotate | numbered marks on the elements of criteria with highlight | | full_page | the whole page (default, stack pages too) or, with false, the first screen: several times smaller on long pages | | screenshot | thumb keeps only the small thumbnail (640 px wide, as tall as the captured page) as the step's picture: for stacks of hundreds of pages whose checks read the page, not the picture, the smallest together with full_page: false; default full | | record | true, step, flow, failed or off; see kaloko walk --record | | trace | keep a trace of actions, requests and console per step (default true) |

A step's capture wins over the scenario's, and the scenario's over the config's; masks are the exception and add up.

Supplier summary for quality assurance

A one-page summary your QA can attach to a supplier assessment. It describes Kaloko as it is today; the details are in Quality evidence for your QMS and on the trust page.

| | | |---|---| | Product | Kaloko: a CLI (kaloko on npm) that walks the screens of an application against written criteria, and the service at kaloko.app that stores the runs, collects people's decisions and signs records | | Supplier | Sinfin, contact on the trust page | | Intended use in quality-managed and regulated work | Executing and recording UI checks, collecting review decisions, producing verifiable records and evidence for the customer's own quality system | | Not intended for | Document control, deviations, CAPA, training records or releasing a product by itself | | Validation status | Not validated or certified by anyone; no claim of compliance with ISO 9001, ISO 13485, IEC 62304, ISO 27001, 21 CFR Part 11 or EU GMP Annex 11. The customer validates its own use | | Hosting and data | Cloud service; where data is stored and which subprocessors handle it is on the trust page | | Security assurance | SOC 2 being prepared, no report yet | | Access control | Roles per organization and project; required two-step verification and session limits (Business); company sign-in over SAML or OIDC, SCIM and an IP allowlist (Enterprise); scoped, expiring API tokens | | Audit trail | Append-only log with who, when, from where and through which client; a run's audit trail is part of its evidence package | | Records | Signed with Ed25519 over canonical JSON; SHA-256 of every screenshot and of the run manifest; verifiable offline with kaloko signoff --verify against the published public key | | Electronic signatures | Optional, off by default: meaning from the customer's list, a fresh second step, printed name, e-mail, time with time zone, bound to the run; changes need a reason and stay recorded. Tokens and agents never sign | | AI | AI evaluators can suggest results; their verdicts are advisory and named with the model; people decide | | Evidence | Per run and release, as JSON (qawalk.evidence.v1) and CSV, and as a signed zip package | | Retention | Accepted runs stay while a Business or Enterprise plan lasts; other runs expire after the plan's retention. Database restorable to any minute of the last 30 days | | Change control | Semantic versions and a public changelog for the CLI; versioned data formats that only gain fields within a version; the service is updated continuously and every record names the service version | | Known gaps | No certification; no SOC 2 report; no passwords (signatures rely on the sign-in plus a second step); the service cannot be held at a version; time source not validated; no legal hold |

Webhooks

Kaloko can call your URL when something happens to a run: it is shared, accepted or returned, a step is approved or returned, someone comments or resolves a thread, its evidence is ready, or its health changes. Your docs pipeline can rebuild from the newest captures, and a QMS portal can show the verdict and the open comments next to its test cases without asking Kaloko every minute. Webhooks are part of the Business plan and up.

Set one up

An organization admin adds the URL in Settings -> Integrations -> Webhooks and ticks its events and projects. Each webhook has its own signing secret, shown next to it. An organization can have up to 10 webhooks.

A webhook takes every open project (and runs without a project), or only the projects you tick. Events of a restricted project reach only the webhooks that name it, so a client's portal hears about its own project and nothing else. A guest's comment counts for the project of the run they commented on.

The same settings work over the API with an admin token:

| Request | What it does | |---|---| | GET /api/v1/orgs/<org>/webhooks | Your webhooks with their events, projects and secrets | | POST /api/v1/orgs/<org>/webhooks | Adds one: { "url", "events": [...], "projects": [...], "all_projects": true } | | PATCH /api/v1/orgs/<org>/webhooks/<id> | Changes its events or projects; what you leave out stays | | DELETE /api/v1/orgs/<org>/webhooks/<id> | Removes it | | GET /api/v1/orgs/<org>/webhooks/<id>/deliveries | The newest deliveries: event, state, tries, last answer |

In projects, an empty string stands for runs without a project. Without projects and all_projects, a new webhook takes every open project.

Events

| Event | When | |---|---| | run.shared | A run was shared (uploaded and finished) | | run.accepted | A run was accepted | | run.returned | Someone returned the whole run, with an excerpt of their note | | review.step_changed | A step was approved or returned, or its decision was revoked: deleted, resolved, or a carried approval withdrawn | | comment.added | A new comment on a run: a thread, a reply, or a returned step with its note | | comment.resolved | A thread was resolved, in Kaloko or by its Jira or Linear issue being done | | comment.changed | A comment was edited or deleted, or its thread reopened (change says which) | | evidence.ready | A run's or a release's evidence can be fetched | | health.worse | The health of a scheduled or CI run got worse than the walk before it | | health.changed | A shared run has health problems that are new or fixed since the run before it |

Approving a step is not a comment, so it raises only review.step_changed. Returning a step raises both: the decision and the thread that explains it.

What a delivery looks like

Each delivery is a JSON POST with these headers:

| Header | Value | |---|---| | Kaloko-Event | The event, e.g. comment.added | | Kaloko-Signature | t=<unix time>,v1=<hex HMAC-SHA256 of "<t>.<body>" with the webhook's secret> | | User-Agent | Kaloko-Webhooks/1 |

The same values also go out as QAwalk-Event and QAwalk-Signature for receivers written before the rename.

The body is an envelope around the event's data:

```
{
"id": "evt_Xb3kQ9vT2mLp",
"type": "comment.added",
"created_at": "2026-10-09T10:14:03.512Z",
"org": "acme",
```

```
"data": {
"run": {
"id": "2026-10-09-checkout-a1b2c3",
"scenario": "checkout",
"project": "shop",
"canvas": "https://kaloko.app/acme/runs/2026-10-09-checkout-a1b2c3/",
"api": "https://kaloko.app/api/v1/orgs/acme/runs/2026-10-09-checkout-a1b2c3"
},
"step": "payment",
"locale": "en",
"viewport": "desktop",
"thread_id": "k3Jd9QpZr",
"comment_id": "k3Jd9QpZr",
"kind": "comment",
"author": { "email": "pam@acme.example", "via": "session" },
"excerpt": "The total shows 0 Kč after the coupon. Ask [email] about the rounding rule.",
"resolved": false,
"at": "2026-10-09T10:14:03.401Z",
"url":
"https://kaloko.app/acme/runs/2026-10-09-checkout-a1b2c3/#step=payment&comments=step&thread=
k3Jd9QpZr",
"comments_api":
"https://kaloko.app/api/v1/orgs/acme/runs/2026-10-09-checkout-a1b2c3/comments"
}
}
```

Answer with any 2xx within 5 seconds. Kaloko sends after it has answered the person who made the change, so a slow receiver never holds anyone up.

Verify the signature

Compute the HMAC over the timestamp, a dot and the raw body exactly as it arrived, compare it in constant time and refuse old timestamps:

```
import { createHmac, timingSafeEqual } from 'node:crypto'

export function verify (rawBody, header, secret, toleranceSeconds = 300) {
const { t, v1 } = Object.fromEntries(header.split(',').map((p) => p.split('=')))
if (!t || !v1 || Math.abs(Date.now() / 1000 - Number(t)) > toleranceSeconds) return false
const expected = createHmac('sha256', secret).update(`${t}.${rawBody}`).digest('hex')
return expected.length === v1.length && timingSafeEqual(Buffer.from(expected),
Buffer.from(v1))
}
```

```
import hashlib, hmac, time

def verify(raw_body: bytes, header: str, secret: str, tolerance: int = 300) -> bool:
parts = dict(p.split("=", 1) for p in header.split(","))
t, v1 = parts.get("t"), parts.get("v1")
if not t or not v1 or abs(time.time() - int(t)) > tolerance:
return False
expected = hmac.new(secret.encode(), f"{t}.".encode() + raw_body,
hashlib.sha256).hexdigest()
return hmac.compare_digest(expected, v1)
```

Retries and repeats

When your endpoint does not answer 2xx, Kaloko tries twice more within a few seconds, then again over about a day at growing intervals (from a few minutes up to 12 hours); after that it gives up. Each try is signed afresh, but the event id stays the same, so keep the ids you have handled and skip repeats. The same change is never sent

twice as two events: one comment is one comment.added.

Settings shows the last answer of each webhook and its recent deliveries (delivered, will try again, gave up), and the API lists them under …/deliveries. Deliveries are listed for 30 days.

Payloads

Payloads carry ids, states, short excerpts and links. Screenshots, attachments and whole comments stay behind the API: fetch them with your own token from the api and comments_api URLs. An excerpt is one line of at most 140 characters, with e-mail addresses replaced by [email].

Every event about a run has run: id, scenario, project (or null), canvas (the run in Kaloko) and api. Times are ISO 8601 in UTC. via says how a person acted: session (in the app), token (CLI, CI, MCP), agent or integration (Jira, Linear).

run.returned

```
{
"run": { "id": "…", "scenario": "checkout", "project": "shop", "canvas": "…", "api": "…",
"environment": "staging", "verdict": "returned" },
"by": { "email": "ada@acme.example", "via": "session" },
"at": "2026-10-09T11:02:41.118Z",
"note": "Checkout total is wrong on the coupon step."
}
```

review.step_changed

```
{
"run": { "id": "…", "scenario": "checkout", "project": "shop", "canvas": "…", "api": "…" },
"step": "payment",
"locale": "en",
"viewport": "desktop",
"state": "returned",
"by": { "email": "ada@acme.example", "via": "session" },
"at": "2026-10-09T10:20:12.006Z",
"comment_id": "Qm2xYt7Lc",
"note": "Total shows 0 Kč",
"url": "https://kaloko.app/acme/runs/…/#step=payment",
"review_queue_api": "https://kaloko.app/api/v1/orgs/acme/runs/…/review-queue"
}
```

state is approved, returned or revoked. comment_id names the decision; a withdrawn carried approval has "carried": true instead. locale and viewport appear when the decision was made on one capture. note comes with returned steps.

comment.added, comment.resolved, comment.changed

| Field | Meaning | |---|---| | step | The step id, or null for a comment on the whole run | | locale, viewport | The capture the comment was made on, when there is one | | thread_id | The thread's first comment; equal to comment_id for a new thread | | comment_id | The comment the event is about | | kind | comment, reply or returned (a returned step's note) | | change | comment.changed only: edited, deleted or reopened | | author | Who wrote the comment: email, via and, for agents, agent | | by | Who resolved, edited, deleted or reopened it | | excerpt | The start of the comment; left out when it was deleted | | resolved | Whether the thread is resolved now | | url | The thread on the canvas | | comments_api | All comments of the run, for your token |

The url opens the step with its comments and the thread in view: https://kaloko.app/<org>/runs/<run>/#step=<step>&comments=step&thread=<thread>. A comment on the whole run links to …/runs/<run>/#comments=run.

run.shared, run.accepted, evidence.ready, health.worse

run.shared and run.accepted carry the run with its steps and where to fetch the newest captures and the changes since the previous accepted run (steps_url, changes_url). evidence.ready names the run or release and the URLs of its evidence, package and trace matrix: Quality evidence. health.worse carries the run, the previous

one and the new findings: Check the health of your app.

health.changed

Sent after every shared run of a scenario whose health checks moved against the previous run in the same environment: something fails (or the evaluator is unsure) that did not before, or something that failed is fine now. It counts per page, language and check, so it suits a weekly SEO walk over thousands of pages. A first run, and a run where nothing moved, send nothing.


```
{
"run": { "id": "2026-10-12-seo-weekly-a1b2c3", "scenario": "seo-weekly", "title": "Weekly
SEO", "environment": "production", "started_at": "2026-10-12T05:00:00.000Z", "canvas":
"https://kaloko.app/acme/runs/2026-10-12-seo-weekly-a1b2c3/#health" },
"previous": { "id": "2026-10-05-seo-weekly-d4e5f6", "started_at": "2026-10-05T05:00:00.000Z"
},
"counts": { "new": 14, "fixed": 3, "still_open": 120, "unchecked": 0, "worse": 1, "expired":
0 },
"open": 134,
"new": [
{ "id": "eshop|de|HC112|sklep-kubek", "step": "eshop", "locale": "de", "check": "HC112",
"text": "The page carries structured data (JSON-LD)", "page": "sklep-kubek", "url":
"https://shop.example/de/eshop/kubek", "severity": "warning", "first_seen": { "run":
"2026-10-12-seo-weekly-a1b2c3", "at": "2026-10-12T05:00:00.000Z" } }
],
"fixed": [],
"aggregate_url":
"https://kaloko.app/api/v1/orgs/acme/runs/2026-10-12-seo-weekly-a1b2c3/health/aggregate",
"diff_url":
"https://kaloko.app/api/v1/orgs/acme/runs/2026-10-12-seo-weekly-a1b2c3/health/diff"
}
```


new and fixed list the first 20; the rest is behind diff_url. The fields are explained in Health API for dashboards.

Design a flow with your agent

In this tutorial you design the screens of one flow as live HTML with your own agent, let the team comment on them, publish a version and make it the design that every implementation is checked against. You need a project where kaloko init has run and a scenario for the flow, for example qa/flows/checkout.yml.

Add a design environment

A design environment is a folder the CLI serves while it walks. Add it to kaloko.config.yml:

```
environments:
design: { kind: design, serve: design }
```

The scenario stays the one development and QA use: same step ids, same criteria. A design is checked against the acceptance criteria too, before anyone builds it.

Let the agent draw the steps

Ask your agent, with whatever design skill your team uses:

> Design the checkout flow into design/ with our brand skill: one folder per step of qa/flows/checkout.yml, > shared CSS in design/_shared, tokens in design/tokens.json. Then run kaloko walk and kaloko sync.

Each step becomes design/<step>/index.html: plain HTML, CSS and JS that open from disk. Links between steps are relative links such as ../payment/index.html.

Walk, lint and sync

```
npx kaloko start --scenario qa/flows/checkout.yml --env design
npx kaloko design lint
npx kaloko walk
npx kaloko sync
```

design lint checks the folder against the portable HTML rules: relative URLs, no CDNs, no storage, predictable content. walk renders only the steps that changed, and sync saves them into the shared draft and takes the steps others changed.



The draft of a design flow · en · desktop

One template, several variants

Some steps come in variants: an e-mail for each reason support can pick, a landing page for each city, a card for each plan. Make such a step a stack step in the scenario and give each variant a folder of its own:

```
- id: credit
kind: stack
```

```
title: Credit from support
items: { urls: [refund, late, damaged] }
```

```
design/credit/index.html an overview, optional
design/credit/refund/index.html
design/credit/late/index.html
design/credit/damaged/index.html
```

Without items.urls naming the folders, every subfolder with an index.html is a variant. walk captures each variant in every viewport, and on the canvas the step opens a list of its variants with their results. Click one to see it live. design lint reports a variant folder without its index.html.

Web fonts and a prototype that remembers

Live steps load files only from the design folder. If your designs take fonts from Google Fonts, Adobe Fonts or your company CDN, an organization admin adds those addresses in Settings -> Security -> Fonts and assets of designs. List them on the environment too:

```
environments:
design: { kind: design, serve: design, origins: [https://fonts.googleapis.com,
https://fonts.gstatic.com] }
```

design lint then tells you which of them the organization does not allow yet. Stylesheets, fonts and images load from those addresses; scripts never do.

A cart filled on one step can show on the next: ask the agent to keep it in the prototype state (window.kaloko.state). Every step still shows its own sample data when someone opens it alone, and Start over in the presentation empties the cart again.

Comment together

The team opens the draft on the canvas, clicks through the live steps and pins comments to elements. Everyone sees who else is looking. Your agent reads the comments with kaloko feedback, changes what they ask for and syncs again.



A live design step · en · desktop

Empty carts and brand themes

A screen people only see in its happy state gets built without the others. List them on the step, and the agent draws a page for each one next to the step's page:

```
- id: cart
states: [empty, error, loading]
```

design/cart/empty.html is the empty cart, design/cart/error.html the one that failed to load. Each state is a capture of its own with its own comments and verdicts, and the canvas gets a State switcher next to the language.

A token set with several modes, such as brand-a and brand-b, renders every step in each mode, with the mode's tokens as CSS custom properties, and the canvas gets a Token mode switcher. tokens_mode in the environment stays the default mode; token_modes: [brand-a, brand-b] picks which modes to render. Light and dark alone follow color_schemes: [light, dark] as before.

Publish a version

```
npx kaloko share --name "Checkout v2"
```

This publishes a version of the whole flow, such as "v2 · 3 changed, 17 unchanged". Unchanged steps are stored once. share --notes-preview shows the notes on what changed before you publish.



A published version · en · desktop

Make it the reference

Once a person has accepted the version, make it the design reference (Team plan and up):

```
npx kaloko reference --design --version 2
```

Turn on packs: [design] for the implementation steps of the scenario. On Business and up, every implementation run is then checked against the reference: pixels, structure, tokens and contrast, and kaloko drift names the token to use where the code differs.

Coming from Figma

If the design already lives in Figma, kaloko design import figma <file-url> --scenario qa/flows/checkout.yml turns its frames into design steps and its variables into tokens. A folder of frames exported from Figma works without a token.

Your first run in ten minutes

By the end of this tutorial one flow of your app is on the canvas at kaloko.app, shared, with every screen in every language and screen size your scenario names. You need Node 20 or newer, Chrome, and an app running locally.

Install the CLI

In the root of your project:

```
npm install --save-dev kaloko
npx kaloko init --agent claude --org <your-org>
```

Use --agent codex for Codex, --agent all for both, or leave it out to drive the CLI yourself. init writes kaloko.config.yml and copies the skill to your agent. It also reads the project: the framework, the dev server's command and URL, the locales and the main routes. From those it drafts a first scenario, qa/flows/main-pages.yml, so the first walk works without editing.

To give the project its name on Kaloko right away, add --project eco-podcasty --project-name "ECO Podcasty". The slug goes into addresses; the name is what people see in the app.

If the repository holds several sites or apps (a monorepo, a Rails app with several Folio sites), init lists them and drafts no scenario, because the routes of the whole repository would mix the sites. Pick one with --site, and if it is running, give its address with --base-url. The pages then come from its sitemap or its navigation:

```
npx kaloko init --site eco-podcasty --base-url http://localhost:3000
```

If you have no organization yet, create one first; you become its admin. With Google, Microsoft or GitHub it takes one step: sign in, name the organization and you are in, no e-mail link to wait for.

Sign this machine in

```
npx kaloko login
npx kaloko doctor
```

login opens the browser, you approve the code and choose what the token may do. It is saved to ~/.config/kaloko/.env. doctor checks Node, Chrome, the config, the environments and the token, and says what to fix next to every x.

Signed in, npx kaloko projects sync creates the project with the name from the config. Without it, your first kaloko share creates it.

Walk the flow

Start your dev server, then:

```
npx kaloko start --scenario qa/flows/main-pages.yml --env local
npx kaloko walk
npx kaloko preview
```

start creates a run, walk opens every step in the browser and captures it, and runs the checks as it goes. It ends with the verdicts: what failed and what a person should look at. preview prints the path of the local canvas, which opens from disk.

With the skill installed you can skip the typing and ask the agent instead: "Walk the main pages with Kaloko on local and show me what fails." It runs the same commands.

Share it

```
npx kaloko share
```

The run goes to kaloko.app and the command prints its link. On a branch with a pull request, share --pr also puts the link in a comment on the pull request.

Look at it and decide

Open the link. The run is a map of its steps; click one to see the screenshot, the criteria and what was found.



The map of a run · en · desktop

Approve the steps that are right, return the rest with a note, and accept the run when you are happy with it. Your agent reads the notes with kaloko feedback.

Where next

- Write a scenario for the flow you care about most.
- Review a run shows the canvas in more detail.
- Run it in CI on every pull request.
- The CLI reference lists every command.

Changelog: Kaloko

0.22.0

Released 2026-10-20

Added

• A Health checks page in the help lists every pack and each of its checks: what it looks at, why it matters, its severity and where it applies, in English and Czech. It is written from the same code the CLI runs, and the Health tab on the canvas links to it.

• The scenario page in the app shows which health packs the latest run had on and off, and how to switch one with packs: in the scenario file.

• Light or dark, your pick: the account menu in the app and on the canvas has a theme switch (System, Light, Dark). The choice stays on this device and applies before the page paints, so nothing flashes white at night.

• The account menu holds who you are, the language of Kaloko, the theme, a link to the Kaloko website and Sign out. On a phone the same items sit at the bottom of the menu.

• Signed-in people reach the website again: "Kaloko website" in the account menu and the footer leads to the home page instead of back to your organization.

• A product you no longer ship can be archived: kaloko products remove <product>, or Archive product at the end of its changelog settings. It disappears from Products, its changelog and docs answer "not found", and its releases and docs stay stored. kaloko products sync reports it as archived instead of creating it again, and kaloko products restore <product> brings it back.

• The help page on browsers and devices describes the default capture setup and how to change it.

• kaloko scenario move <scenario-id> --project <slug> moves a scenario to another project with all its runs, design drafts and screens. Admins do the same in the app, in Scenario settings on the scenario's page, after a confirmation. The move wins over project: in the config, so later shares land in the new project too.

Changed

• On a design environment the code-only packs (seo, aeo, perf, console, errors) are off, because a designer's HTML has no real server, meta tags or error statuses. health: { design_packs: [console] } in the scenario keeps one on. The run says which packs it turned off (packs_off in run.json).

• Design versions are listed under Design drafts on the home page and no longer among the runs.

• The canvas explains Live, Inspect and Dev mode in a line each: Live runs your own uploaded HTML, Inspect styles shows an element's styles and tokens, and Dev mode is the handoff to development. Kaloko does not write code from a design.

• The canvas shows the same main links as the app, in the same order: Scenarios, Runs, Screens, Projects, Products.

• The language of Kaloko moved from the canvas bar and the app header into the account menu, so it no longer sits next to the run's own locale switch.

• The menu button on a phone is round with three lines in the app too, like on the canvas.

• Someone who is a guest or a viewer in another organization can now create their own. Only a full member elsewhere (Designer, Developer / Tester or Admin) is asked to sign in instead. Sign-in goes to the organization where you are a full member first; the others are in the organization switcher.

• New scenarios from kaloko init, kaloko draft and the template capture mobile at 360 × 800 instead of 390 × 844, next to desktop at 1280 × 800. Existing scenarios keep their sizes and baselines. A tablet is added only when you ask for one (tablet: [744, 1133]).

• The scenario and runs lists no longer look the same. A scenario row starts with its trend, the number of runs and the last result; the runs list is split by day, and each run starts with its time and who shared it.

• Branch and commit moved from their own column to a grey line under the scenario name, with a long branch cut short (hover shows it whole). Both lists now fit a 1280 px screen without a sideways scrollbar.

• The scenario list shows the project as a plain label. The project picker in every row is gone, so a stray click no longer moves a scenario with all its runs.

• Return and Approve on a step have their own bar under the comment box, so a comment no longer sits one click away from a decision. In review mode you decide in the bottom bar only; the side panel is for comments.

• In Live mode the comment box says that comments go on an element of the page and starts picking one; pinning and drawing stay on the screenshot.

• "Mark a changing region" no longer disappears on drafts, for viewers and reviewers, or on a step without a screenshot. The button stays, disabled, and says why.

Fixed

• The Screens page follows dark mode: tiles, picture frames and the status labels used light colours on a dark page.

• On the Screens page, the "not checked" label no longer hides off screen.

• The canvas viewport switch offers only the viewports the run has: no "Both" for a run captured on one, and Tablet when the run has a tablet (with three or more viewports the first choice is "All").

• The gate of an environment (HTTP Basic, service-token headers) no longer follows a redirect to another host. Podcast audio that a gated page sends on to S3 now plays instead of failing with HTTP 400, and an identity provider or a link checked on another host never sees the gate. Redirects within the gate keep it, as before.

0.21.3

Released 2026-10-15

Fixed

• In a step's detail, opening an AI-evaluated or failed criterion no longer squeezes its explanation into a narrow column with the technical detail spilling over it; everything sits under the criterion's title again.

0.21.2

Released 2026-10-13

Changed

• Stack pages now report console errors. Before, the check passed on every page, so stack steps with no_console_errors or the console pack may fail after this update where they passed.

• kaloko doctor --json now lists the token's scopes in the token check, so a CI script can check that the token may make a run public (full), not only share it (upload), before it spends twenty minutes walking.

Fixed

• Console errors on the shared browser stay with the capture they belong to. A capture used to list every error the browser had seen on the same page, so one failed request from yesterday's test showed up in every run after it. Now kaloko capture counts only errors from the tab it captures, logged after the run started and after the last capture on that tab, and a walk counts what a page logged since the previous step on it was captured.

• The shared browser's console log starts empty with every kaloko browser start and keeps only its last few megabytes, so a browser left running for a week no longer piles up a log nobody reads.

0.21.1

Released 2026-10-11

Added

• Findings can open issues in GitHub Issues and Plane, not only in Jira and Linear. Connect them in Settings -> Integrations (the Kaloko GitHub app with the Issues permission, or a fine-grained token; a Plane API key), then

name the repository or the Plane project in PUT …/scenarios/<id>/findings/sync. One issue per finding, a comment when the number of pages changes a lot, closed when it is fixed and reopened when it comes back. Admins list which repositories and Plane projects each Kaloko project may send findings to.

• Routes per check or pack in the findings sync: send canonical, hreflang and structured data to the developers' repository and thin content or low click-through to the marketers' Plane project, from one scenario.

• With traffic: on the stack, findings carry the clicks, impressions and impact of their pages, and so do the issues the sync opens.

Fixed

• A tracker that timed out while creating an issue for a finding could get a second one on the next run. The finding now waits and says the issue may exist; after a look at the tracker, Try again on the findings page lets Kaloko create it with the next run.

• Evaluating the same run again no longer sends a step's recording to triage a second time. On a busy machine the first sampled frame sometimes came out empty, so the second evaluation picked other frames and missed the cache. Kaloko now waits until a frame is really drawn and remembers the frames it chose for each recording, so they are the same every time.

0.21.0

Released 2026-10-11

Changed

• Stack pages are now captured whole by default, like every other step; until now a stack kept the first screen only. This is a changed default, and stack runs grow several times: a typical long page stores a screenshot of about 600 KB instead of about 70 KB, and a thumbnail of about 180 KB instead of about 25 KB, because the thumbnail is as tall as the page. Vision checks now see the whole page too, not just what fits on the first screen. To keep the old size, set capture: { full_page: false } on the stack; capture: { screenshot: thumb, full_page: false } is the smallest and suits big monitors that need the page facts rather than the picture.

• When a stack captured whole takes most of a run close to the per-run limit, kaloko share names it and suggests full_page: false; a run over the limit lists full_page: false among the ways to trim it.

• Captures finish finite animations, and most captures show endless ones (a spinner, a pulsing badge) at their first frame. On pages with animations the first run after the update can differ from the runs before it, even though the page did not change.

Fixed

• Before a capture Kaloko waits until what scrolling changed on fixed and sticky parts is back as it was at the top (a header that floats while scrolling, a search panel fading in), or has stopped changing for a second, at most 3 s. Full-page captures no longer show a sticky header caught in its floating state with a half-visible panel.

• A full-page capture no longer catches CSS transitions halfway: resizing the page for the capture restarted them on some sites (a search panel fading out again over the breadcrumbs). Transitions end at once and finite animations finish for the capture.

• A page of a stack has its own Health tab, as a step has: its health checks with the evidence for that page, findings first, and Adopt for the stack step. Before, health checks of stack pages were only in the run's Health panel and the page listed them among its acceptance criteria.

0.20.1

Released 2026-10-11

Added

• Hosted runs group the language versions of a stack page by hreflang (items.variants: hreflang), as kaloko walk does: the same pages in the same week in every language, one row per page in the stack index, and a missing translation marked on the page and counted in HC1005. Reading a source page for its hreflang counts as one

hosted page. kaloko validate no longer warns about such stacks in hosted runs.

Fixed

• kaloko walk reads the sitemaps and robots.txt of a stack with the environment's access (its headers, HTTP Basic, client certificates and the walk's cookies), so a staging behind a password no longer ends with an empty stack and HTTP 401. A sitemap on another host, or one that redirects there, is still read without them. Each file has 60 seconds and 60 MB.

0.20.0

Released 2026-10-10

Added

• Findings: what a scenario has open in health is grouped by check, site, language and step, so two hundred product pages without structured data in Polish are one row, not two hundred. Each finding has its severity, how many pages it hits, up to ten of them with the CMS edit link, and when it first and last appeared. Health -> Findings shows the table per scenario; GET …/scenarios/<id>/findings and MCP list_findings give the same to dashboards and agents (a read:health token reads them too).

• Acknowledge a finding when someone is on it, or ignore it with a reason. The decision stays with the finding from week to week; a finding is fixed once none of its pages fails. Anyone who may comment on the scenario may decide, in the app or with PATCH …/scenarios/<id>/findings/<id>.

• A scenario can open a Jira or Linear issue for each new finding at or above a severity, comment when the number of pages changes a lot, close it when the finding is fixed and open it again when it comes back (PUT …/scenarios/<id>/findings/sync, Developer / Tester). A run opens at most 20 issues; the rest follow with the next run.

• Hosted runs read stack steps: pages listed by items.urls, a sitemap (gzipped and sitemap indexes too) or the sitemaps in robots.txt, with match, exclude, limit and rotate: week picking the same pages as kaloko walk. Each page is read without a screenshot: checks of selectors, texts, titles and meta tags are decided, index fields such as title, H1, description and JSON-LD types are filled in, and the canvas shows the stack with its index. Big stacks are read over several quarter-hour rounds, at most 1,000 pages per run; Run now, deploy triggers and edits wait for a run under way.

• kaloko schedule add takes an environment with a base_url per language; the hosted run opens each language at its own address, otherwise at default, as the walk does.

• Settings -> Integrations and kaloko schedule list show how far a stack run got and how many health results the last run found new and fixed.

• items: { variants: hreflang, source_locale: cs } on a stack step shows the translations of a page as its language versions instead of separate pages. The list, limit and rotate come from the source language, and every other language walks each page's own hreflang alternate, so the stack index has one row per page with a screenshot per language and all languages check the same pages in a given week.

• A page without a translation is reported instead of left out: the stack index marks the missing language, HC1005 counts the pages (no hreflang for the language, only x-default, a hreflang back to the page itself, a translation that answers an error, or several pages sharing one translation), and kaloko export findings lists one row per page and language.

• Traffic data for the fix list: traffic: in kaloko.config.yml or the scenario joins Search Console and GA4 numbers onto the walked pages, from a CSV export (English or Czech headers) or, with a service account key, straight from the Search Console and GA4 APIs, once per run. kaloko export findings then adds clicks, impressions, CTR, position, sessions and an impact score and puts the most seen pages first, the page index on the canvas gets Impressions and Clicks columns, and the seo pack adds HC127 for pages that are seen often in search but clicked less than half as often as the site's pages at similar positions. Nothing is requested unless you configure it.

Changed

• Every page of a stack in a hosted run counts against the monthly budget of hosted pages. When the budget runs out halfway, the run waits up to a day for more, then shares the pages read so far and names the rest as not walked.

• A hosted run with stack steps takes up to 30 languages (10 without).

Security

• Sitemaps and robots.txt read by hosted runs go through the same public-address check as pages, on every redirect and for sitemaps on other hosts too, without the environment's credentials; pages are opened only on the environment's own addresses. match and exclude are checked when a schedule is saved and run in a linear-time matcher.

0.19.0

Released 2026-10-10

Added

• Check nojs_parity: one more request per page fetches the raw HTML without JavaScript and compares it with the rendered page. It fails when the title or canonical differ, when the meta description, H1 or a JSON-LD type appear only after scripts run, or when the raw body text is below 30 % of the rendered one (ratio:). With user_agent: gptbot (or oai-searchbot, claudebot, perplexitybot, googlebot, bingbot, a whole user agent string) it comes like that crawler, without cookies, and a page robots.txt closes to it is reported as blocked, not fetched. A bot the site, a WAF or a gate turns away (403, 429, a challenge page, a redirect to a sign-in) is not evaluated, with a note saying who was turned away. The aeo pack adds it as HC905 for GPTBot. Hosted runs show it as not evaluated; kaloko recheck keeps the walk's result.

• An evaluator budget for large sites: evaluate.budget.calls in kaloko.config.yml, or kaloko evaluate --budget N and kaloko walk --budget N, caps the paid JEV calls of one evaluation. Ordinary steps are asked first, then the pages of each stack, starting with pages that fail a deterministic check and pages checked longest ago. What does not fit stays "not evaluated" with a note, and kaloko evaluate says how many captures it left out. Cached answers cost nothing, so the next evaluation picks up the rest. The same captures always spend the budget on the same pages.

• items.semantic_order: priority on a stack step picks its semantic sample the same way without a budget; by default the sample stays the first pages in walk order.

• Your own text evaluator behind an OpenAI-compatible API, such as Ollama (http://localhost:11434/v1) or LM Studio: an evaluator of kind: openai_compatible with base_url and model, chosen with evaluators.text. It answers the same questions from the same page text. With evaluators.escalate_to: jev, answers it is not sure about go on to JEV, counted in the budget. Its verdicts are marked as your own model's on the canvas, in acceptance records and in the pull request comment. Nothing changes until you configure it, and the page text goes only to your base_url.

• A dashboard can read the health of a run already counted: GET …/runs/<id>/health/aggregate?by=locale,step,check gives pass, fail, unsure and not evaluated per language, step, check or pack, with the pages hit most. Group ids stay the same from week to week. GET …/scenarios/<id>/latest finds the newest shared run of a scenario (?environment=), and …/latest/health/aggregate answers for it in one call. MCP has get_health_aggregate.

• Health over time: GET …/runs/<id>/health/diff (MCP get_health_diff) lists, page by page, what fails or is unsure now and did not in the previous run of the scenario on the same environment, what was fixed and what is still open, each with the run where it first appeared. A page the run did not check (a rotating sample, a language left out) stays open as unchecked instead of counting as fixed. The health page shows new and fixed per scenario, and the scenario's health history has them per run.

• The health.changed webhook tells you after a shared run when something in its health is new or fixed, with the counts and the first results.

• Tokens with the read:health scope read health and the newest run of each scenario and nothing else, for a dashboard that pulls once a week. Create one in Settings -> API tokens (on Enterprise also as a service

account). The fields are described in the new help page "Health API for dashboards".

• kaloko init --seo <site> drafts a weekly SEO and answer-engine monitor from the live site: robots.txt and the sitemaps give the languages, a sample of each language's pages the biggest content families (first path segment), paired across languages by hreflang, the same segment or size. It writes qa/flows/seo-<site>.yml with one rotating stack step per family (match per language, rotate: week, packs seo and aeo; a11y and perf with --packs), the production environment in a new config, and prints the next steps. A site that cannot be reached writes nothing; a site whose pages live on its www (or https) name is followed there.

• Stack pages can link back to their source: an item script or a JSON list gives edit_url and content_ref, or items.resolve: steps/resolve.mjs looks them up for the walked pages (15 s per page). The page detail shows "Edit in CMS", and the stack CSV and MCP get_stack_items carry both.

• kaloko export findings [--format csv|json] [--locale l] [--step s] [--out file]: the fix list of a local run, one row per failing criterion or health check, page and language, with the evidence, edit_url, content_ref and the command that walks that page again. It reads the run and writes nothing into it.

• Pack aeo (answer engines: AI search and assistants), advisory like the other packs: HC901 questions and answers on the page (details/summary, dl/dt, two or more headings ending with "?") come with FAQPage or QAPage structured data, HC902 a short answer-first paragraph follows the H1, HC903 /llms.txt answers at the site root (one request per site and walk), HC904 a meta description is present. Pages without questions show HC901 as not evaluated. Hosted runs check HC904 and mark the others "not evaluated" with one note.

• Stack steps read gzipped sitemaps (sitemap: /sitemaps/{locale}/sitemap.xml.gz, or gzip bytes behind any URL) and follow sitemap indexes two levels deep.

• items: { sitemap: robots } (or robots: true) takes the pages from the sitemaps listed in /robots.txt. When their URLs name the walked locale (/sitemaps/de/…), only that locale's sitemaps are read. match, exclude and limit work as before.

• Stack index fields jsonld_types (the structured data types of each page, for example Product, Offer, BreadcrumbList) and meta tags by name, such as from: og:title, from: twitter:card or from: meta:robots, so items.json carries them for every URL.

• items: { limit: 100, rotate: week } walks a big catalogue in turns: each ISO week (or each run with rotate: run) takes the next limit pages of the list, so every page is checked within a few weeks. The stack index shows per language which part was walked and how many pages were never checked, and each page its last_checked.

• match and exclude of a stack step take a map by language ({ default: '^/eshop/', fi: '^/fi/tuotteet/' }) for sites that translate their URL segments.

• An item script gets the step (node: id, title, lane, session) and items.params, so one script serves several stacks.

• expect_status: 404 on a step says it is an error page. kaloko draft sets it on the drafted 404 step.

• The seo pack checks structured data: HC112 JSON-LD is on the page, HC113 every JSON-LD block is valid JSON (a warning names the block and the parse error). New assert jsonld_present takes types: [Product, BreadcrumbList] for pages that must carry certain types; jsonld_valid is the parse check.

• Every capture records structured_data (blocks, invalid blocks, the schema.org types and the recognized ones: Product, Offer, BreadcrumbList, FAQPage, Article, BlogPosting, Organization, WebSite and a few more). A stack step's items.json and its manifest count pages per type, pages without any structured data and pages with invalid JSON-LD, so a dashboard can chart them week by week.

• A stack step with the seo pack now compares its pages with each other after the walk: shared titles, meta descriptions and H1s within one language (HC121–HC123), canonicals that point elsewhere or to a page that answers an error or says noindex (HC124), hreflang the other walked language version does not answer (HC125), and sitemap pages that answer other than 200, redirect or say noindex (HC126). Advisory like the pack; the findings count pages, and items.json lists the affected addresses in groups.

• Pack locale for multilingual sites: html lang, the page's own hreflang, the language of the text and the decimal separator of prices are checked against the walked locale (HC1001–HC1004). An untranslated page in a language version shows up as HC1003.

• Site checks with health: { site: true } in the scenario, once per site and run: robots.txt parses and does not block the walked start pages (an error on production, a warning elsewhere) or AI crawlers, every sitemap it lists answers and is valid with absolute same-host addresses and sane lastmod dates, /llms.txt is there, and http:// and the other host name (with or without www) redirect in one hop (HC1101–HC1107). kaloko health and the canvas show a row per site; health.site.environments: [production] keeps staging walks from checking.

• capture: { screenshot: thumb } keeps only the 640 px thumbnail of each page as the step's picture: for stacks of hundreds of pages whose checks read the page, not the picture (SEO crawls).

Changed

• kaloko walk --items picks stack pages from every matching page, not only from this week's rotation window, so a fixed page can be walked again in any week; such a walk leaves the rotation state as it was.

• match and exclude apply before limit for every source of a stack step, scripts included.

• The errors pack checks HC601 (soft 404) and HC605 (the explanation) only on error pages: a step with expect_status of 400 or more, or a page that answered 4xx/5xx. On normal pages they are not applicable instead of failing. kaloko validate warns when the pack is on a stack step.

• kaloko validate and kaloko start warn when the config asks every scenario for its own project (a comment, or a product with projects:) and the scenario has no project:.

Fixed

• When kaloko share gets HTTP 404, it says why: which project the run names and which projects the organization has, or that the token belongs to another organization or expired (kaloko login --org <org>). Other commands say the same for a bare "Not found" on an organization or project.

• Changing the Kaloko app's repository access or permissions on GitHub now brings you back to Settings with the repositories read again, instead of an error saying the installation link expired.

• kaloko share checks the run against its limit by the size the service stores (HTML and JSON gzip-compressed), not the size on disk, and when a run is too big it names the largest steps. A 20-language SEO run of 1 400 pages that was refused at 529 MB is about a third of that as stored.

• The personal data check before sharing no longer reads the user part of a URL as an e-mail address (a public Sentry DSN such as https://key@o1.ingest.sentry.io/23 in a meta tag).

0.18.0

Released 2026-10-10

Added

• Every pull request in a repository connected to the Kaloko GitHub App gets one Kaloko check on its newest commit. It waits for the runs shared for that commit, from a coding agent in the cloud, your machine or CI, and turns green when they pass or when the change reaches no scenario. It keeps a stable name, so branch protection can require it before the merge. An optional Kaloko review check follows the decision on those runs.

• Modes per organization and repository in Settings -> Integrations -> GitHub: off, manual, automatic (the default) and strict. The github: section of kaloko.config.yml on the base branch can make the check stricter, never looser.

• One pull request can be controlled with the labels kaloko:skip, kaloko:all, kaloko:review and kaloko:run, with the comments /kaloko skip <reason>, /kaloko walk and /kaloko all, or with the buttons on the check. Only people with the access you choose may skip, never apps or bots, and the check names who skipped it and why.

• kaloko ci plan tells the check which scenarios a commit reaches, following imports like kaloko affected. The workflow from kaloko ci github sends it before walking.

• Settings list the installation's repositories with their mode, last check and what went wrong, in words you can act on, such as a repository that is not in the installation yet.

Changed

• A shared run carries its repository, the project's folder in a monorepo, and whether tracked files differed from

the commit. The pull request is found by commit, so a sandbox without gh still gets its check. A run of uncommitted changes does not count for the commit.

Fixed

• A link to Kaloko from another site (a portal, GitLab, Teams, an e-mail) opens the page for someone who is signed in, instead of asking them to sign in again.

• A link to a step (#step=…, #lane=…, #session=…) still opens that step after signing in.

• Links to Kaloko's public changelog and help now show a preview picture in chat apps and social posts.

• In replay, -> and the Next button follow the diagram: the next screen on the same row, forward before back, instead of whichever connection the run happened to list first.

• Open index on a stack in replay shows the stack's pages; before, they opened hidden behind the replay.

• A step reference shown as an example inside a code block or inline code is no longer resolved as a picture, so kaloko docs check and kaloko docs publish stop warning about examples.

0.17.0

Released 2026-10-09

Added

• Webhooks now tell you about comments and review changes: comment.added, comment.resolved, comment.changed, review.step_changed and run.returned. A QMS portal can show the open comments of a run next to its test cases and link straight to the thread, without polling.

• Each webhook takes the projects you tick, like a Slack or Teams channel. A restricted project's events reach only the webhooks that name it.

• Settings -> Integrations -> Webhooks lists each webhook's recent deliveries. A delivery that fails is tried again for about a day with the same event id. Events, payloads and signature checks are in the help under Webhooks.

• A documentation portal can show a step's screenshot next to its test step: GET /api/v1/orgs/<org>/runs/<run>/steps/<step>/screen with a read token answers the full screenshot or its thumbnail with its SHA-256. Pass the hash you stored as sha256 and a different file answers 409 instead of a newer picture. MCP agents get the same through get_step_screen.

• A run can name the records it serves outside Kaloko, such as a change request in your QMS: kaloko start --ref CC-2026-001 (repeat it for more, or set KALOKO_REFS in CI), and why it was walked with --purpose dry_run, formal or smoke (KALOKO_PURPOSE). The canvas shows both in the run's header.

• GET /api/v1/orgs/<org>/runs?ref=CC-2026-001 lists every run of a record with its verdict, environment and purpose; ?purpose=formal keeps the formal ones. kaloko runs --ref, the MCP tool list_runs and the evidence of a release take the same filters, and the evidence of a run carries run.refs and run.purpose.

• The trace matrix says who performs each step and in which part of the process: every row of trace.csv, trace.json and the evidence names the step's session (the role) and its lane (the sub-process). The evidence and MCP get_run_summary also list the flow's lanes and sessions, so a portal can show them without reading the YAML.

• A canvas link ending in #lane=1 opens the map on that lane with its band highlighted, and #session=qc brings one person's steps forward across all lanes.

Changed

• An invitation code now starts its trial the moment you sign up with it or enter it under Plan & billing. There is no checkout and no card. Plan & billing shows when the trial ends and has an Add a card button. Billing starts only when the trial ends; without a card the organization goes back to Free that day and keeps its runs.

• Organizations that already had the sample run get the current one (with health, requirement references and detail crops) in the same language, once; an organization that deleted its sample does not get it back.

• The sample run in a new organization shows what Kaloko does today: Health finds three small things in the tea shop (a picture without alt text, a missing meta description, a pale note under the delivery options), steps and

criteria carry requirement references, the cart and the payment have crops for user guides, and the keyboard walk leaves focus shots. The guided first review has one more hint, about Health, and points at the requirement and the crop on the way.

Fixed

• Organizations that entered an invitation code but never finished the checkout that followed were left on Free. Their trial now starts for the full length of the code.

• Dragging the map to pan no longer leaves lane headings, step titles and thumbnails highlighted as selected text. Text in the side panels, comments and fields still selects as before.

• On the page that approves kaloko login, picking another organization now reloads its projects, scopes and token lengths, so a project-limited token only ever lists projects of the organization it is for.

• Screenshots of runs shared with the deduplicated upload no longer go missing after an internal storage recount. The recount now leaves every file that points at stored content alone, and unreferenced content is kept for a week before it is cleaned up, so a lost reference can be restored.

0.16.1

Released 2026-10-07

Added

• When you type a command, sub-command or option your kaloko does not know but a newer release has, the CLI says which version has it and how to update (npm i -D kaloko@latest, or npx kaloko@latest …) instead of guessing a typo. It asks kaloko.app/cli/latest.json at most once a day, waits at most 1.5 s and stays quiet offline; KALOKO_NO_UPDATE_CHECK=1 turns it off. kaloko doctor shows when a newer kaloko is available.

• kaloko help --json lists every command with its sub-commands and options, for agents and scripts.

0.16.0

Released 2026-10-07

Added

• Signing in with Google, Microsoft or GitHub when your e-mail has no organization yet now leads to creating one: name it and you are in as its admin, without waiting for an e-mail link. The sign-up page offers the same buttons, and a trial code in its link carries over.

• Trial code links can point to the homepage: https://kaloko.app/?kod=CODE. The visitor first sees what Kaloko is, a slim banner on every page says what the code gives ("You have an invitation: 60 days of Business free, no card."), and the code stays with them until they create an organization, by e-mail or with Google, Microsoft or GitHub. An organization without a subscription starts the offer from Plan & billing with one click.

• Projects from the CLI: kaloko projects create <slug> --name "ECO Podcasty", kaloko projects rename <slug> --name … and kaloko projects sync, all with --json. The config takes project_name: (the name people see) and product: (a product the project is part of); kaloko init --project … --project-name … --product … writes them and, when you are signed in, creates the project on Kaloko. The MCP tool upsert_project does the same for agents.

• kaloko init in a repository with several sites or apps (a monorepo, Rails engines, several Folio sites) lists them instead of guessing routes. --site picks one, --base-url reads its pages from the running site (sitemap, then the links of its home page) and --skeleton writes a scenario with a TODO to fill in.

• Details for guides: name the parts of a screen a guide talks about under details: on a step (an id, a name and a selector). Every walk takes each one in every language and screen size with its section or card around it and a ring around the element, with personal data hidden as on the screenshot. In docs and the changelog, shows it; kaloko docs check warns about a detail the scenario does not declare or no accepted run has yet, exports carry the picture and the MCP tool resolve_step_image takes the detail id.

Changed

• New organizations cannot take an address Kaloko uses or keeps for itself: its routes (hooks, scim, …), words

for likely future pages in English and Czech, two-letter language codes and look-alikes of Kaloko, QAwalk and Sinfin. Existing organizations keep their addresses.

• The first kaloko share of a new project creates it with project_name: from the config, not with its slug. An existing project is never renamed by a share: when the names differ, the share says so and points to kaloko projects sync.

• kaloko init leaves out what is not a page: favicons and other static files, ad and tracking endpoints, API, admin and sign-in routes, routes that need a parameter and, with a running site, anything that does not answer HTML. The drafted scenario says at the top where its pages came from and what was skipped.

• The admin of a project can rename it with their API token (kaloko projects rename); everything else about a project (visibility, people, removal) stays in the app.

• Teammates need this CLI version or newer once kaloko.config.yml has project_name: or product:: older ones reject keys they do not know.

• The focus shots of the keyboard walk show on the Health tab under the focus check and beside the tab order of the shown language and screen. In Step attachments, which are now grouped into details, keyboard focus and crops, each focus shot says "Keyboard focus · HC309" and opens that check.

• The scenario browser leads with the scenario's name (up to two lines) and shows its id as a detail; the runs header names the scenario; the kind reads Task/Flow (Úkol/Průchod).

• A run opens in the reader's language when it has screens in it (an English reader sees the English screens first), and that language leads the locale list; an embedded canvas always follows its page.

Fixed

• Design drafts on the scenarios page follow its filters: a project shows only its own drafts, run filters leave them out, and each draft shows its name and project.

0.15.0

Released 2026-10-07

Added

• Requirement references: refs: on a scenario, a step or a criterion (refs: [URS-12, FS-3.4], free text). They reach the run, the acceptance record and the trace matrix. Runs also record the SHA-256 of the scenario file as walked and the CLI, Node.js, operating system and browser versions.

• Evidence for your own documentation (Business): for any shared run, accepted or not, and for a release. It says whether the run was accepted and whether it was signed in Kaloko, and lists the criteria as walked with who assessed each result (a person, a deterministic check or a named AI model, AI always advisory), screenshot hashes, decisions, signatures, the audit trail and the trace matrix. kaloko evidence <run> | --release <version> (--json, --csv), the MCP tool get_evidence, the API (…/runs/<id>/evidence, …/trace.csv, …/audit.csv, …/evidence?product=&version=) and the format qawalk.evidence.v1 with a JSON schema.

• The evidence package: kaloko signoff <run> --package or Evidence package (ZIP) on the canvas downloads the signed record (JSON and PDF), the evidence, the trace matrix, the audit trail and the screenshots with a signed list of every file. kaloko signoff --verify checks a package as well as records.

• Electronic signatures, optional and off by default (Business): an admin turns them on in Settings -> Security for every project or some. Accepting a run (and, if chosen, approving a step) then asks for the meaning of the signature from the organization's own list and a fresh second step; the record shows the printed name, e-mail, time with time zone and meaning. Changing a signed decision asks for a reason and the signature stays in the record as withdrawn. Tokens and agents never sign.

• The webhook event evidence.ready: a run's evidence is complete (shared, accepted or returned) or a release was recorded, with the state and where to fetch the evidence and the package.

• The acceptance record and its PDF list signatures, traceability, tools and environment, the AI models behind results, and advisory results such as health checks.

• Help: Kaloko in regulated work (GxP) and a supplier summary for quality assurance.

• kaloko health lists the health findings of a run by pack and step, with what is new or fixed since the previous run and how to fix each check; --json for agents, --remote for the shared run, --fail-on error for a team that wants health to stop a pipeline.

• Health checks have their own ids: HC101…HC705 (HC1xx seo, HC2xx usability, HC3xx a11y, HC4xx perf, HC5xx console, HC6xx errors, HC7xx app). Acceptance criteria stay AC. The number is the check's, so a pack kept in acceptance adds AC302 where the health check is HC302. Runs and scenario files written before keep working: health checks stored as AC302 show as HC302, the old ids work in kaloko health adopt, MCP and links, nothing stored is rewritten, and captures stay current.

• kaloko health adopt <step>:<check> [--text "…"] writes a health check into the scenario as an acceptance criterion of the step, as it is or in your words, so it counts from the next walk: home:HC103 becomes AC103 with from: HC103, and the step's health stops listing HC103. On the canvas, Adopt as criterion does the same, and kaloko health pull brings those adoptions into the scenario file.

• Health on the canvas has its own tabs: Health beside Summary in the run panel (counts, change since the previous run, findings by pack or by step) and beside Criteria in the step detail, which now holds acceptance criteria only. Arrow keys move between tabs. Review mode links to the step's Health tab. Plus a health badge on run lists and a health page per project with the trend over time. MCP: get_health and adopt_health_check.

• After a decision the step shows it: the approval panel opens with who approved or returned the step and when, the signature's meaning and time when signed, the note, the attached files and earlier decisions, and Change decision. Approvals are no longer tucked away as resolved comments. The step header and its card on the canvas show the decision too (with a mark when signed), and the run summary shows who accepted the run.

• Signature meanings come in the signer's language. Reviewed, Approved and Verified are translated; an organization adds a Czech label to its own meaning after a bar (QA release | Uvolnění QA). The signature keeps the label the signer saw, its id and the language. The signing dialog fills in the name from the profile, or from company sign-in.

• A signature now also covers the decision: its note and the SHA-256 of the attached files, pin and drawing. Earlier signatures keep verifying as they were made. The acceptance record lists the files attached to each step decision and the step's earlier decisions, and the evidence package holds the files under decisions/<step>/, listed in the signed manifest.

• Requirement references show on the canvas as chips: in the run info, under the step title, beside each criterion and in review mode. The index finds steps by them.

• Hosted runs and runs shared from CI tell you when health gets worse: a new error-level finding notifies the scenario's followers, the Slack and Teams channels that take "Health got worse" and webhooks with health.worse. Each schedule chooses errors (the default), warnings or no alerts, in Settings or with kaloko schedule add --health-alert.

• The agent skill checks health before handing work over and fixes what it can, without hiding findings.

Changed

• The semantic evaluator's prompt version is kaloko-jev-v1 (it was qawalk-jev-v1); answers cached before keep being used.

• Deleting a decision on a step goes into the audit log, and decisions on a run stay in the audit log as long as the run does (other entries still for two years).

• Check packs no longer decide acceptance. The checks of seo, a11y, perf, usability, console, errors and app still run on every walk, but they make up the run's health: shown apart on the canvas, counted by severity, compared with the previous run, and never part of a step's or run's verdict, the review queue, carried approvals or the acceptance record. A run no longer comes back red because of a meta description nobody asked for. The design pack still counts for acceptance. Runs shared earlier keep their verdicts.

• To keep a pack deciding as before, list it in the scenario: health: { acceptance: [seo] }.

0.14.0

Released 2026-10-07

Added

• An admin of a paid organization can start another organization under the same e-mail: Plan & billing has the link, and the sign-up page says the new one gets the same address. Free organizations and code trials without a card do not open the way, so they cannot be chained.

• --json is a contract now: stdout holds only the JSON answer, progress and notes go to stderr, and a failed command ends stderr with one line {"error":{"message":"…","code":"…","next":"…"}}. New on share (the run id and its URL), evaluate, build, validate, scenarios, auth list, browser status and ci preview-url.

• Two more exit codes: 3 when the Kaloko service cannot be reached (no connection at all), 4 when the token is missing, invalid, expired or revoked. 0, 1 and 2 mean what they meant; an error answer from the service other than 401 is still 1, and docs check, drift and design lint still report their findings with 1.

• Lists page through the API: runs, comments, notifications, tokens and the audit log take ?cursor= and name the next page in X-Next-Cursor and Link: <…>; rel="next". Bodies keep their shape. The MCP tools list_runs, list_notifications and list_comments take cursor and answer next.

• kaloko runs --all follows those pages to the last run instead of stopping at 300; --limit N lists at most N and --cursor goes on where a list stopped.

• kaloko export run|steps|docs|live|design as sub-commands, each with its own --help. The flag forms (export --live, --design, --format, --scenario) keep working.

• kaloko affected --fail-if-none: exit 1 when nothing is affected. --check does the same and says it has a new name; workflows written by kaloko ci use the new one.

• Durations take units: --timeout 90s, ci preview-url --wait 10m, share --expires 72h, prune --older-than 14d. A bare number means what it meant before.

• default_token_mode in a design environment names its default token mode; tokens_mode still works.

• A step picture in docs and changelogs plays the step's recording. When the accepted run has a recording (kaloko walk --record), the picture shows a play button; a click or tap plays it without sound, and the picture comes back at the end. Nothing plays by itself. ?media=video in a step reference says the step needs its recording, and kaloko docs check warns when the run has none; ?media=image keeps the picture only. It works in "Walk through it step by step" too.

• Recordings that published docs or a changelog play are kept as long as they play them, instead of expiring after 30 days, and they count in storage.

• Settings -> Security -> Experiments shows how Clef triage does in your organization: useful findings per kind, how well its probabilities hold, findings turned into comments, the time to the first review on runs with findings and without, and triage requests per useful finding. The same numbers come from GET …/findings/stats and MCP get_triage_report.

• Exports carry the recordings: Markdown, Docusaurus, VitePress and offline HTML get a player with the MP4 and WebM in assets/, MkDocs a picture linked to the recording; the PDF keeps the picture. MCP resolve_step_image and GET …/steps/resolve answer video with the MP4 and WebM addresses and the length.

• Required approvers are asked by name. When a run waits for them, each approver whose e-mail is in your Slack workspace gets a direct message with a link straight into review mode, and Slack and Teams channels that take the new event "Waiting for approval" name them (Teams workflows cannot write to one person). One reminder follows after 24 hours if nobody has decided; the hours, the reminder and quiet hours are in Settings -> Integrations, and admins can read and change them through GET and PATCH /api/v1/orgs/<org>/approval-messages. Slack connected earlier needs to be connected once more so Kaloko can look people up.

• Hosted runs start after a deploy. kaloko schedule hook <id> (or Settings -> Integrations -> Hosted runs -> Run after a deploy) makes a trigger URL and a secret, shown once. A POST with the secret as a bearer token, an

HMAC signature of the body (X-Kaloko-Signature) or Netlify's signed deploy notification starts the run at the next quarter hour. The hosted runs guide has copy-paste examples for curl, Vercel and Netlify. Business plan and up, at most 30 triggers an hour per schedule, each one in the audit log.

• An invitation that cannot be delivered says so. When the mail provider reports a hard bounce or a spam complaint, the pending invitation shows "Address bounced: check the spelling" in Settings -> People and in the Share dialog, the person who sent it gets an inbox item, and Kaloko stops sending it. GET /invitations names the mark in bounced.

• Links to the pricing page, the comparisons, the situations, the "for…" pages and every guide show their own card in Slack, LinkedIn, X and other previews: the page's title and description on the Kaloko card, in the page's language. The other pages keep the default card.

Changed

• New API tokens start with kal_. Tokens that start with qwk_ keep working until they expire.

• Domain verification shows the TXT record _kaloko.<domain> with kaloko-verify=…. A _qawalk record added earlier still verifies the domain.

• Names from before the rename print one line on stderr when used: the qawalk command, QAWALK_* variables, qawalk.config.yml and ~/.config/qawalk/.env. They keep working; kaloko doctor lists them with their new names.

• Public docs and changelogs take their language from ?lang=, as the app does. Links, the canonical address, hreflang, the sitemap and llms.txt use it; the former ?language= (docs) and ?locale= (changelog) redirect there, so old links keep working.

0.13.0

Released 2026-10-06

Added

• kaloko <command> --help (or -h) and kaloko help <command> show the help of that one command, with the options every command takes and the exit codes: 0 done, 1 checks found problems or the command failed, 2 a usage error. kaloko --version, -v and kaloko version print the version.

• kaloko start --json prints the run id and its folder, kaloko status --json what each step has captured.

• --org <slug> on every command works with another organization than org: in the config.

• API errors name their reason in code next to error. When the plan does not include something (HTTP 402) the answer links the organization's Plan & billing in billing_url, and the CLI prints that link.

• Webhooks carry Kaloko-Signature and Kaloko-Event next to QAwalk-Signature and QAwalk-Event, with the same values, so receivers written before the rename keep working.

• The trust page names a contact for reporting abuse (phishing, malware) on kaloko.app, files.kaloko.app and design previews at *.kaloko-usercontent.com.

• The trust page explains how Kaloko hides personal data in captures (masks, [hidden] in the stored HTML and the outline evaluators read, pii: redact) and links the how-to. The subprocessor list now names Workers AI at Cloudflare, which Clef triage uses only when an admin turns the experiment on.

• Every guide is also plain Markdown at its own address with .md added (/guides/ci.md, /cs/pruvodci/ci.md), for agents and answer engines. llms-full.txt now carries the pricing table and billing questions, the comparisons, the situations and the trust facts.

• The comparison pages answer the questions people ask about them.

Changed

• An option a command does not take gets a "Did you mean --env?" instead of being skipped quietly. prune, trash, restore, keep, archive, public, release and comment stop before they change anything; other commands say it and go on.

• A usage mistake (unknown command or option, missing argument) exits with code 2 and points to kaloko help

<command>, without the support line.

• runs, prune and screens take --env like the other commands; --environment still works. archive takes --off like keep and public (--undo still works), and prune takes --dry-run, which is what it does without --yes.

• A service account token on a plan without service accounts is answered with HTTP 402 (it was 403): the plan, not the token, has to change.

• Webhook requests say User-Agent: Kaloko-Webhooks/1.

• Pricing: the print-quality row is gone, a row shows that hiding personal data works on every plan, and another that the GitHub check on the pull request comes with Team. The site says plainly that your data stays readable on every plan and that exporting a run as a ZIP comes with Team.

• Terms of service, version 4: a trial started with an invitation code needs no card; if none is added, the organization returns to Free when the trial ends. The terms and the pricing page call the billing settings Plan & billing, as the app does.

• Kaloko's public changelog and help have one address per language, which search engines and link previews now understand. The Czech changelog page says that the release notes are written in English.

• /design and /cs/navrh lead straight to the guide on designing with your agent.

Fixed

• kaloko prune --env preview --yes pruned the runs of every environment: --env was not read. It now prunes only that environment.

• --key=value keeps everything after the first =: --map="Sign in=login" arrives whole.

• Commands run without org: in the config say how to set it instead of failing with a 404 on /orgs/undefined.

• kaloko runs says when the list stops at the newest 300 and how to narrow it.

• kaloko design import --json prints only the JSON on standard output; progress goes to standard error.

0.12.2

Released 2026-10-06

Fixed

• Czech changelog headings (Přidáno, Změněno, Opraveno, Odstraněno, Zastaralé, Bezpečnost) count as Added, Changed, Fixed and the other Keep a Changelog groups, so a Czech CHANGELOG.md is grouped like an English one.

• A form that needed a fresh second sign-in step (inviting someone, changing a setting) is sent again after the step instead of coming back empty. The fields wait in the browser tab, never on the service.

Changed

• Screens on the canvas cards stay sharp: thumbnails are 640 wide and scaled down step by step (one big jump made small text fall apart), and when you zoom in past what a thumbnail holds, the visible cards load the full screenshot. Runs shared before this get the zoom part too.

• Videos play on: once you play a step's video, the next step's video starts by itself in the step detail, and replay with video moves to the next step when a video ends (a decision waits for you, a still step shows its screenshot for three seconds). The next step's video loads while the current one plays.

• A step's video is at least 2.5 seconds long, held on its last frame, so a quick step no longer flashes past. A step where nothing moved keeps no video at all, and the canvas shows its screenshot.

• kaloko.com and www.kaloko.com lead to the same address on kaloko.app.

0.12.1

Released 2026-10-05

Added

• A trial code from us gives an organization's first subscription a longer free trial without a card. Open the sign-up link with the code, or enter it under Plan & billing. Two weeks before the trial ends, admins are asked for a card; without one the organization goes back to Free on its own and keeps its runs.

Changed

• kaloko share refuses unmasked personal data only in a run that hides it (pii on the run, scenario or environment, or mask: auto); other runs are shared with the same list as a warning. The site's own contacts no longer count: e-mails on the environment's domain, e-mails and phone numbers in the page footer, and those of an Organization or ContactPoint in JSON-LD (pii: { site_contacts: false } checks them again). Sample addresses such as you@company.com, placeholder text and SVG drawings are not read as personal data.

• Phone numbers written 603 123 456 count only next to a word like Telefon or Mobil, and amounts, EANs and order numbers no longer read as phone numbers. The nine-digit birth number needs its label (RČ), as does one without the slash.

• pii: pseudonymize gives one value the same stand-in in every run of the scenario, so screenshots and approvals carry over, and two values never share one. KALOKO_PII_SALT adds a secret of your own.

Fixed

• Hidden form fields, value attributes and personal data in URLs (also encoded, jan%40firma.cz) are masked.

• The capture record no longer keeps real data: page title, headings, JSON-LD, microdata, console lines, blocked requests, the URL and check evidence all go through the mask, and a capture whose masked page cannot be read stores nothing.

• Masking no longer changes what the page reacts to: console errors are read before it, data-* attributes are masked only in the stored HTML, and the keyboard walk runs on the real page with masked focus shots.

• Short masked values such as 2024 or Praha are no longer replaced across the whole run (width:100% stays).

• Recordings cover personal data in modal dialogs and popovers, open shadow DOM and same-origin frames, and on a new page from its first frame with everything masked so far. Screenshots of steps mask the same places.

• Values masked in one kaloko process are masked in the next process of the run too.

• Triage (experimental) fits long pages into Clef: two page tiles and smaller recording frames instead of four full tiles, and a request still too large is asked again with half the images. A Cloudflare token in the project that cannot use Workers AI hands triage to Kaloko instead of stopping it, and such a token is used only when evaluators.triage is set in kaloko.config.yml.

• A failure while masking leaves the page as it was.

• Long words without spaces no longer slow down detection.

0.12.0

Released 2026-10-05

Added

• Triage, an experiment an admin switches on in Settings -> Security -> Experiments. kaloko evaluate shows each screen, and a few frames of its recording, to Clef on Cloudflare, which names the kind of problem it sees: overlapping elements, cut-off text, a raw translation key, a placeholder, an error, an endless spinner, a layout that jumps. The canvas lists them under "Possible problems"; one click turns a right one into an ordinary comment, and findings never change a verdict. evaluators.triage in kaloko.config.yml tunes or turns it off.

• MCP tools get_findings and answer_finding.

• Masks hide personal data in everything a step stores, not only in the screenshot. The captured HTML and the page outline the evaluators read get [hidden] of the same length ([skryto] in Czech runs), and traces, recordings, crops, focus shots and the HTML view are covered too. Checks still read the real page, so their verdicts do not change.

• pii: redact on a scenario finds e-mails, phone numbers, birth numbers, IBANs, account numbers, dates of birth,

addresses and names in greetings without a selector, in Czech formats too; capture: { mask: auto } does it for one step. pii.allow in kaloko.config.yml lists values that only look personal, such as your support line.

• pii: pseudonymize replaces personal data with made-up values instead of boxes. One value gets the same stand-in for the whole run, so you can still see a name travel from the form to the summary and the e-mail.

• Captured e-mails hide their recipients (To and Cc, now captured too) whenever a step masks anything; greetings and payer blocks follow the step's masks.

• kaloko share checks the run before it uploads: an unmasked e-mail, phone number, birth number or IBAN stops it, with the step and the element. --allow-pii uploads anyway.

• An environment with pii: required gets no run that would keep personal data; kaloko start --pii redact (or pii: on the scenario or the environment) satisfies it.

• The canvas says "Personal data hidden" on such runs and outlines the masked places in the step detail, so a dark box is not taken for a bug in the app.

Changed

• The masks of a scenario and of its step now add up; before, a step's capture.mask replaced the scenario's.

• kaloko.app serves its own fonts and the canvas you open from disk carries them inside, so no page asks Google Fonts for anything and Google Fonts is off the subprocessor list on the trust page.

0.11.0

Released 2026-10-04

Added

• The approval page of kaloko login says where the request came from (city, country and address), which terminal asked and when, and starts with a plain warning: approve only a sign-in you started yourself, just now. When the request came from another country than the one you are in, the page says so in red.

• SCIM tokens are valid for a year. Settings -> Security -> SCIM shows the date, and "Renew for a year" keeps the same token, so nothing changes in your identity provider. Admins get a reminder in the inbox 30 days before a token expires. Tokens you already have run for a year from this release.

• A company sign-in over SAML set up before "Require a signed response" existed turns it on with one click in Settings -> Security. Kaloko offers the button once it has seen your identity provider sign its responses, so people keep signing in as before.

Changed

• A passkey used as the second step must ask for your PIN, fingerprint or face. A security key that signs without asking no longer counts; your account page names the key and says how to bring it back (set a PIN on it, or add another passkey).

• A run whose title, environment, branch, commit message or another field is too long is refused at the start of the upload with a message that names the field and the limit, instead of an unexpected error halfway through.

Fixed

• A browser that launched but then stopped answering no longer holds kaloko walk, evaluate --recheck or a capture until someone kills it. A new browser must open its first window within 20 seconds or it is closed and started once more, and later windows have the same limit. kaloko browser start, kaloko doctor, kaloko auth save, the PDF export and connecting to the shared browser now have the time limit and the second try too (KALOKO_LAUNCH_TIMEOUT, seconds).

• A js_equals check on a busy machine no longer fails when the page answers a little late: the expression keeps its 2-second limit, and Kaloko waits up to 17 seconds for the page to reply.

Security

• The sign-in cookie can be set only by kaloko.app itself. You stay signed in while it changes over. Your session also gets a new id at every sign-in and right after the second step.

• App and website pages tell the browser which features they never use (camera, microphone, location, payments and the like), and a Kaloko tab opened from another site is cut off from that site's window.

• Billing events from Stripe and pull request events from GitHub are applied once, also when someone sends a captured copy again.

0.10.0

Released 2026-10-04

Added

• A step can list its other states: states: [empty, error, loading]. Walks run the step script once per state (it gets state), a design draws cart/empty.html next to cart/index.html, and each state gets its own screenshots and verdicts. The canvas has a State switcher next to the language, and review mode and the grid show the state you pick. A criterion with states: [empty] is judged on the empty cart only.

• A design whose token set has several modes, such as two brands, is rendered in each of them. Switch modes on the canvas like colour schemes; tokens_mode in the environment stays the default. Light and dark alone still follow color_schemes. Variants of a stack step are captured in every state and mode too.

• kaloko capture --state, kaloko spec --state and a state option on the MCP step tools (get_step, get_step_spec, get_step_drift, get_step_assets, get_step_recording).

• A design step can come in variants of one template, such as an e-mail for each reason support can pick. Make it a stack step and give each variant a folder, design/<step>/<variant>/index.html. kaloko walk captures every variant in every viewport, kaloko sync sends them to the draft, and in a published version the step opens a list of its variants with their results. Each variant opens live. kaloko design lint reports a variant folder without its page, and both design exports include the variant folders.

• Web fonts on live design steps. An organization admin lists the addresses designs may take fonts, stylesheets and images from (Google Fonts, Adobe Fonts, your CDN) in Settings -> Security, and live steps load them instead of falling back to a system font. Scripts still load only from the design itself. List the addresses under origins of the design environment and kaloko design lint says which of them the organization does not allow yet.

• Prototypes remember things between steps: a cart filled on one step shows on the next, in Live mode and in the presentation. The state stays in the viewer's browser tab, every step opened alone still shows its own sample data, and "Start over" in the presentation clears it. Design agents use window.kaloko.state.

Changed

• Large runs are quicker on the canvas. On a run of 500 steps in seven languages on three devices, with the CPU slowed down to a phone's, a zoom step takes 33 ms instead of 133 ms, switching between desktop and mobile 650 ms instead of 910 ms, and the map is drawn once when it opens (it was drawn twice). Scrolling the step index no longer sorts every step on each frame.

• On kaloko.app a large run's canvas data and a scenario's history page come from a cache after the first visit (27 ms instead of 113 ms, 9 ms instead of 230 ms), and the changelog, the products page, the screen library and review mode ask the database far fewer times.

0.9.0

Released 2026-10-04

Added

• After a release, people who read the product see a short "What's new" bar in the app that links its illustrated changelog. It comes once per release and person; open the changelog or close the bar and it stays gone.

• The canvas tells you when the connection drops or Kaloko cannot save. Comments, approvals and verdicts wait and go out once you are back online, or with "Try again"; nothing is posted twice and nothing is lost quietly.

• kaloko doctor --fix installs a missing browser, creates missing folders and fills in org: from your token. An expired token points you to kaloko login.

• Long stack walks show roughly how much time is left.

Changed

• On a phone, review mode puts Approve and Return under your thumb at the bottom of the screen and says what you just decided and which step comes next.

• The first-steps list links straight to where each step is done: your newest run for comments and approvals, with the command to copy for the steps done in the terminal.

• Every error page has a way to write to support, with the address already in the e-mail, and a link to the status page. The app and website footers link the status page too.

• CLI errors say what happened and what to do next (sign in again, wait and retry, check the network) and end with support@sinfin.cz. Losing the connection no longer prints a stack trace.

Fixed

• A browser video encoder that stops responding no longer holds kaloko walk --record: the recording ends with a note and the walk goes on.

• Dark mode is easier to read: form fields have visible edges, the highlighted plan column on the pricing page is readable, and shortcut keys on the canvas's review buttons stand out.

0.8.0

Released 2026-10-04

Added

• Steps and scenarios say what kind of page they are: page: landing, work, list, settings, form or doc. The usability pack asks questions that fit: "one clear primary action" only on landing pages, and on lists, settings, forms and docs a question of their own. Without page, public pages count as landing pages and app screens as work pages, so a signed-in overview is no longer asked for a single call to action.

• A criterion can apply to some environments only: environments: [production] on hreflang pairs, a live demo or the production sitemap. On a local or staging walk it shows as not applicable to that environment and does not make the step partial.

• kaloko validate warns when a scenario walks several languages and a text check matches only one of them, for example pattern: "Plans and payment" without the Czech wording.

• Empty pages in the app say what to do next and give the command with a copy button: a product without a release or docs, its changelog and docs, an organization without runs, the runs and projects lists, the inbox and the screen library. People who only read a product see a plain sentence instead of a command.

• Kaloko's help covers the whole product, in English and Czech: two tutorials, how-to guides from writing a scenario to company sign-in, a reference of every CLI command, scenario key, check, pack, config key and MCP tool, and explanations of how verdicts are decided and how Kaloko maps to your applications.

• The website links Kaloko's own public changelog and help, so you can see both working before you set them up.



The home page · en · desktop

• Company sign-in over SAML can require a signed response: the identity provider must sign the whole response, not only the assertion. New connections start with it on. Connections set up earlier keep working as before, and Settings -> Security recommends turning it on.

Changed

• Error pages say why you got there and how to go on. A missing page offers the overview or another sign-in; a page that needs a role you do not have offers an e-mail to your organization's admins, already written; an expired link says to ask for a new one; an ended session signs you in and brings you back to the same address.

• Unsent text on the canvas survives. The verdict, return and issue notes keep their draft like comments do, and when your session expires while you write, the canvas says so, keeps the text and returns you to the same step after you sign in again.

Fixed

• A browser that starts but never answers no longer leaves kaloko walk, video or a check waiting: each launch has a time limit and one more try (KALOKO_LAUNCH_TIMEOUT in seconds, default 45).

• kaloko init proposes products from npm, Yarn and pnpm workspaces and from packages/ too, not only from apps/; shared libraries without a dev, start or serve script are left out.

• kaloko stability --pull names the button people use on kaloko.app: Mark a changing region.

• A table in published docs can hold a pipe inside a cell, written as \|.

• Hosted runs open only pages whose address leads to the public internet. An address that points into a private network is refused when you schedule the run and again before each run. Saved passwords and cookies are sent only to https addresses.

• Text and attribute checks in hosted runs finish quickly whatever the pattern. Patterns with backreferences or lookahead are reported as not evaluated there, with the reason; kaloko walk still checks them in full.

• Signing in, sign-up, invitation links and the second step have their own limits on repeated attempts. Exports and new API tokens have a generous hourly or daily limit per person.

0.7.1

Released 2026-10-04

Fixed

• kaloko walk of an Android, iOS, desktop or Electron app ends with the verdicts, like a walk of a website; before, the criteria stayed unchecked until you ran kaloko evaluate.

0.7.0

Released 2026-10-04

Added

• Products in the app. The main menu has Products: every product you read with its newest version, its docs, its modules and projects, who reads it and links to its changelog and docs. A project page names the products it belongs to. kaloko products sync puts the products of kaloko.config.yml on Kaloko before their first release, and kaloko products list shows which ones are there.



The Products page · en · desktop

• A trust page at kaloko.app/trust (in Czech at /cs/duvera): where your data lives, how it is encrypted, who can get in, which companies process it and where our SOC 2 preparation stands. Kaloko is not SOC 2 certified yet, and the page says so. The security contact is also in /.well-known/security.txt.



The trust page · en · desktop

• Seat warnings. When 80 % of the paid seats are taken, and again when all are, organization admins see a banner on the home page, in Settings and on the billing page, and get one e-mail per threshold in each billing period. GET /api/v1/orgs/<org>/usage reports the seats too.

• Yearly Enterprise subscriptions can go above their paid seats. Kaloko keeps the highest number of seats in each quarter; after the quarter, Kaloko billing reviews the true-up and invoices the extra seats for the rest of the subscription year or waives them. Seats added in the last quarter are not charged for that year: a week before the renewal the subscription rises to them and the renewal invoice includes them. The billing page shows the current quarter and every earlier true-up.

• Company sign-in over SAML accepts encrypted assertions. Create the organization's key pair under Settings -> Security -> Company sign-in; its certificate is in the SP metadata, and a new key pair keeps the old one working until you remove it.

• Single logout with your identity provider: signing out at the provider signs people out of Kaloko, and signing out of Kaloko can sign them out at the provider too.

• Sign-in from the app tile of your identity provider, off until an admin turns it on, with a default page people land

on.

Fixed

• The keyboard checks of kaloko walk no longer report a trap inside a third-party widget such as an anti-bot check, and judge a link wrapped over two lines and a small radio inside its label as people see them. Tab lists with arrow keys count as one tab stop.

• The canvas works better from the keyboard: every control shows a clear focus ring in light and dark mode, a "Skip to content" link jumps past the top bar, and while a step detail or another overlay is open, Tab stays in it instead of wandering through the hidden canvas. Screen readers get named side panels and dialogs.

• The canvas no longer jumps while it loads.

• A link from a comment notification opens the step with its comments in view, on a phone too.

• On a phone, a live page in the step detail fills the room above the sheet instead of a short strip, an open sheet leaves a strip of the preview that you can tap to lower it, and "Ctrl+Enter sends" shows only with a keyboard.

0.6.0

Released 2026-10-03

Added

• A visual changelog. kaloko changelog draft writes the Unreleased entry of CHANGELOG.md from the pull requests since the last release and adds before and after pictures of the screens your accepted runs show differently. kaloko release <version> records the release with its pictures, and kaloko.app shows it as a page you can share with clients, support or anyone outside development.

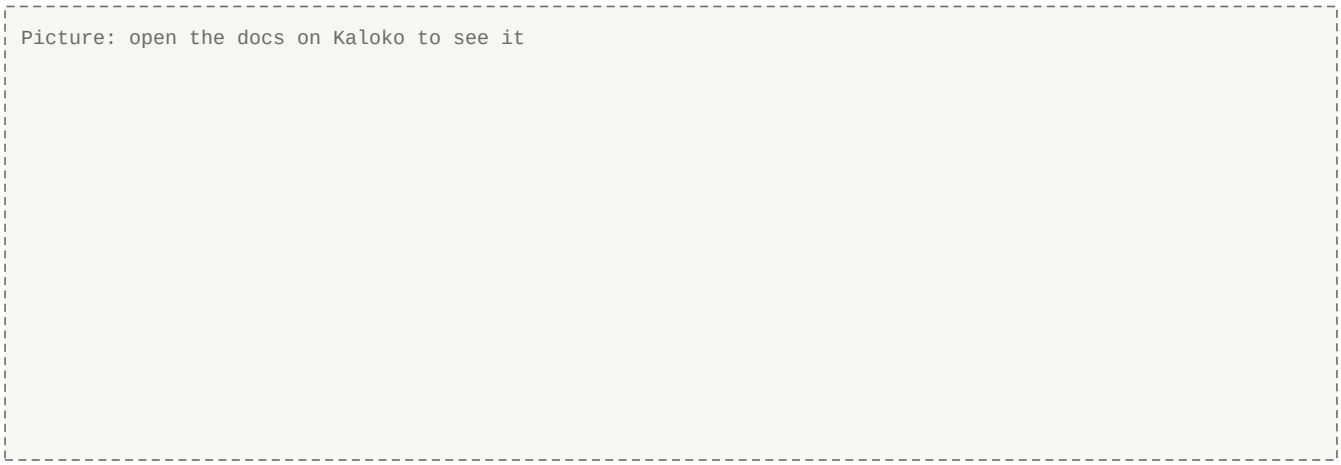
• Step references in Markdown: always shows the newest accepted screen, and ?version=2.4.0 the screen of that release.

• Products and modules in kaloko.config.yml for repositories with several apps; scenarios say which product they show (product:, module:). Without them the project is the product.

• Readers from your domain: everyone who signs in from a verified domain reads the changelogs, also of restricted projects (Business and up).

• kaloko init asks whether to keep a visual changelog, kaloko doctor checks that every product's version can be read, and agents get get_changelog, draft_changelog_entry and resolve_step_image over MCP.

• Docs that keep up with the app. Guides in your docs/ folder show steps instead of screenshots; kaloko docs publish puts them on kaloko.app with a version picker, language and screen switches and search, and every picture opens its step on the canvas.



A guide with its pictures · en · desktop

• When a screen changes after its guide was written, the section says "the screen changed in 2.4 — check the text", and kaloko docs check lists it with broken references, missing translations and changelog items without a picture, so CI can stop on them.

• A guide plays as a walkthrough, one step at a time with its words as the caption, and readers tick off the steps they tried.

• Exports for your own docs site: kaloko export --format md, docusaurus, vitepress, mkdocs, html, pdf or json, with the pictures as files. The docs and changelog pages offer the same downloads.

• A "What's new" e-mail: readers ask for it on a product's changelog and get each new release with its pictures in their language (Business and up).

• kaloko ci github --release records every release that release-please, changesets or semantic-release publishes.

• Issues in Jira and Linear from a returned step or a failing criterion, on the canvas, in review mode or with kaloko issue create. The issue gets the screenshot, the criterion, the notes and a link back, and the comment on the step is resolved when the issue is done. Branches and pull request titles that name an issue link the run to it.

• Slack and Teams channels per project, each with the events it wants and quiet hours. Restricted projects reach only channels chosen for them.

• A short list of first steps on the home page (share, comment, approve, invite, CI, release) until the team has done them.

• Hosted runs: Kaloko opens the pages of a read-only scenario every day or week and shares the run (kaloko schedule add, Business). kaloko schedule export github writes a scheduled workflow for the full walk.

0.5.2

Released 2026-10-03

Added

• The first version of the visual changelog: kaloko changelog draft, kaloko release, step references and the changelog page on kaloko.app with a before and after slider. It is described in full under 0.6.0.

0.5.1

Released 2026-10-03

Added

• Swipe and onion skin compare next to side by side and diff, in the step detail and in review mode.

• Regions to ignore can be drawn by hand on a capture. The canvas diff skips them, and kaloko stability --pull writes them into the scenario.

• A grid (key G) shows one step in every language and screen size at once, also on a phone and in review mode.

• Comments can carry drawn boxes, arrows and lines, and attached images. Agents get the marked region in pixels.

• An accepted run has a signed acceptance record: a PDF with thumbnails and a printable page. kaloko signoff --verify checks the signature.

0.5.0

Released 2026-10-03

Added

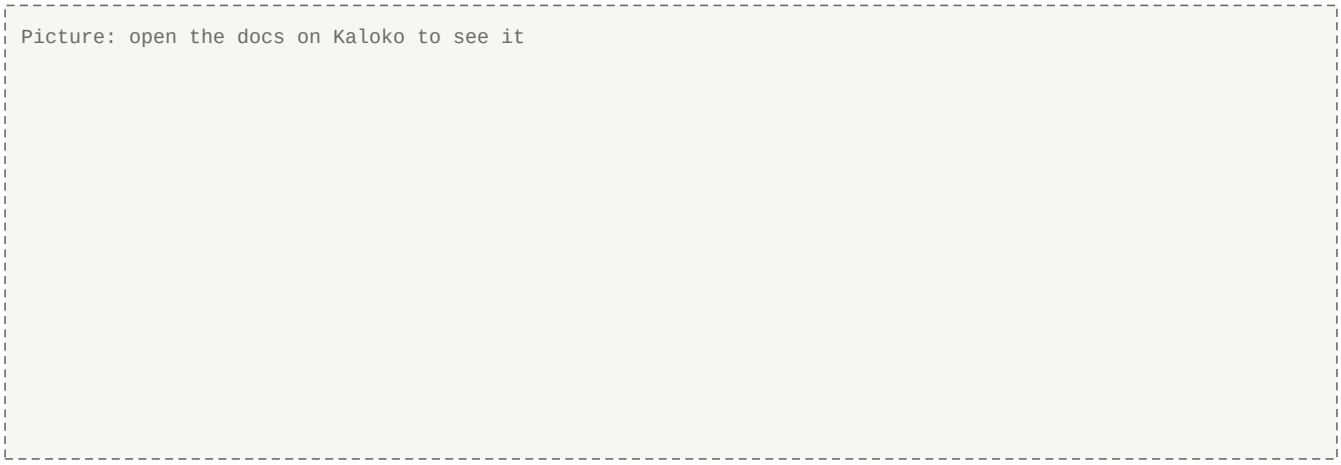
• Checks of the page itself: the address a step ends on, computed styles of an element and values the page exposes, so scenarios no longer need data-qa markers.

• Runs in WebKit and Firefox besides Chromium, on device presets (phones, foldables, tablets, TV), in dark mode, with reduced motion, forced colours or as an installed web app. Every step is captured in each combination.

• App walks on several devices at once: Android emulators, iOS simulators, a real iPhone and Android

WebViews, and Windows apps next to macOS ones.

• One picker on the canvas for language, browser and colour scheme that stays usable with many values.



The canvas of a run · en · desktop

• Workflows written by kaloko ci install the browser engines your scenarios walk.

Fixed

• Design drafts keep the browsers, schemes and device presets of their scope.

• An e-mail captured during a walk in several browsers counts once.

• Language and theme switchers work in the phone menu.

• Ignored regions, imported screenshots and cached evaluator answers carry over between capture variants.

0.4.11

Released 2026-10-03

Added

• Recordings: kaloko walk --record films each step from the previous screen to its capture, or the whole flow with a chapter per step, and keeps a trace of actions, requests and console messages for every step.

• The canvas plays a step's recording and shows its trace.



A recording on the canvas · en · desktop

• kaloko video export turns the recordings into one walkthrough video with a title card per step.

• Shared runs carry their recordings on the Business and Enterprise plans, kept for 30 days unless the organization chooses otherwise.

Fixed

• A recording too large to upload stays in the local preview instead of failing the share.

0.4.10

Released 2026-10-03

Added

• kaloko ci github|gitlab writes a workflow that walks what each pull request changes on its preview deployment and comments on the pull request; kaloko ci preview-url finds the preview address in CI.

• kaloko import lays out screenshots, recordings and traces that an agent or Playwright already produced as a run.

• kaloko walk --affected and kaloko affected walk only the steps that the changed files reach, and say why.

• kaloko walk --heal proposes a new selector when the page lost the old one, and applies it once the step passes.

• Several languages walk at the same time (--concurrency).

• kaloko init recognises the framework, the dev server, languages and main routes, and drafts a first scenario.

• Uploads skip screenshots the service already has and resume after a broken connection.

0.4.9

Released 2026-10-03

Added

• A vision evaluator that looks at the screenshot and says why, keyboard and screen reader checks, and regions that change on every run can be ignored.

• Dev mode in the step detail of a design: box model, tokens, redlines, states and assets.



Dev mode in the step detail · en · desktop

• kaloko spec tells an agent what to build for an approved design step, and kaloko drift what its implementation does differently.

Fixed

• The canvas on phones: a smaller stack badge, readable stack lists, the step header above its tabs and no sideways scrolling.



A stack page on a phone · en · mobile

• Page counts in Czech use the right plural.

0.4.8

Released 2026-10-03

Added

• Review mode: the canvas walks a person through what still needs a decision, with keyboard shortcuts, and kaloko review lists the same for agents.

• Approvals carry over to steps whose screenshots did not change since the last accepted run.

• Every new organization gets a sample run with a guided first review.

• Two-step verification with passkeys or an authenticator app, and an organization rule that requires it.

• An audit log for admins, tokens limited to chosen projects, and company sign-in with SAML or OIDC, SCIM and an IP allowlist on Enterprise.

Fixed

• Plan badges on the pricing page no longer overflow on phones.

0.4.7

Released 2026-10-03

Added

• Sign in with Google, Microsoft or GitHub where the service offers it.



The sign-in page · en · desktop

• kaloko login approves a token in the browser instead of copying it from Settings.

• Invitations: people outside the organization join single projects as guests, and the Share dialog of a project shows who has access.

0.4.5

Released 2026-10-03

Added

• A Designer role and API tokens with chosen scopes (read, comment, design, upload).

• Focus mode shows the preview alone, and the step detail has a layout for phones.

0.4.4

Released 2026-10-02

Fixed

• kaloko evaluate --recheck says that it asks the semantic questions again too.

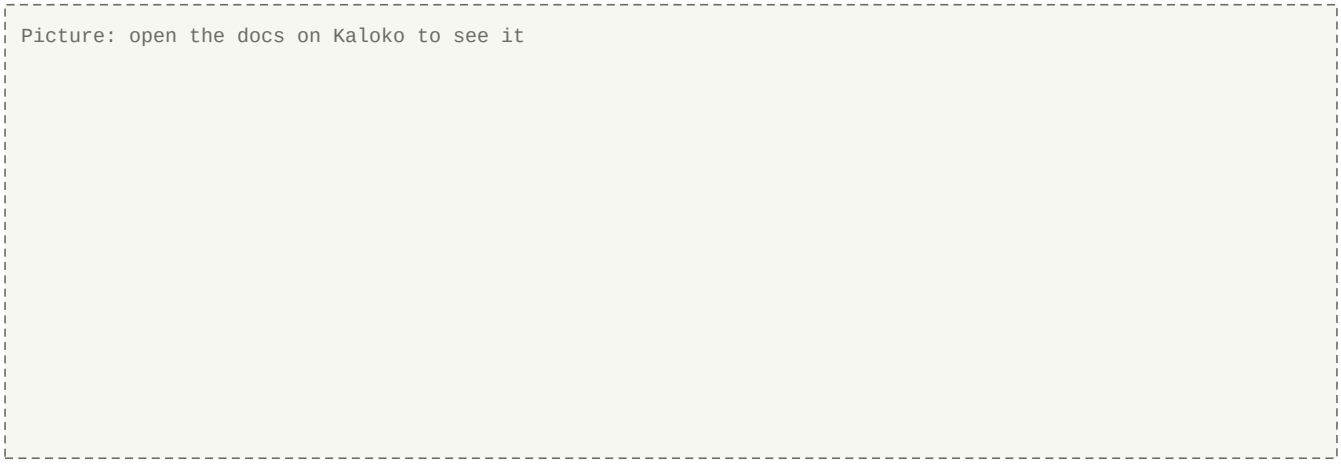
• A lane on the canvas is as tall as its tallest card.

0.4.3

Released 2026-10-02

Fixed

• Signed-in pages that Kaloko's own walk found wanting: labelled fields, real settings tabs, cards on phones and the whole inbox.



Settings · en · desktop

• The canvas has a main landmark, readable badges and a visible compare button, and no longer shifts while loading.

• Pages load their fonts sooner and keep their layout when the font arrives.

0.4.2

Released 2026-10-02

Added

• The Kaloko wordmark with a yellow marker and a new icon.



The home page · en · desktop

• The Czech website lives under /cs/, so each language has its own address.

• Pages of a stack may carry their own purpose for the semantic evaluator.

• Live design revisions open on their own domain.

Fixed

• kaloko doctor prints the config file it actually read.

0.4.1

Released 2026-10-02

Added

• links_ok and hreflang_pairs checks for pages and stacks.

Fixed

• Unknown addresses on kaloko.app answer 404 instead of the sign-in form.

• Every language version of the website answers at its own address.

0.4.0

Released 2026-10-02

Changed

• QAwalk is now Kaloko: the CLI is kaloko (the qawalk command and qawalk.config.yml keep working) and the service lives at kaloko.app.



The landing page · en · desktop

Added

• Kaloko Design presents interactive HTML designs, lints pages that must work from disk, and exports a design in open formats.

0.3.0

Released 2026-10-02

Changed

• The CLI is on npm as kaloko instead of @qawalk/cli. The qawalk command, qawalk.config.yml, the QAWALK_* variables and ~/.config/qawalk/.env keep working, and kaloko init replaces the old skill.

• The app and the website share kaloko.app. Pages on qawalk.com redirect there; the API, MCP, webhooks and file links on the old addresses keep answering.

Added

• A "Group by issue" switch on the lists of runs and scenarios. The lists remember it, and "Latest per scenario" too.

0.2.22

Released 2026-09-30

Fixed

• Large runs open faster on the canvas, and kaloko evaluate needs less memory for big stacks.

0.2.21

Released 2026-09-30

Fixed

• A kept run cannot be trashed, and releasing it gives it at least a week. Kept design versions are not pruned.

• The "no project" filter on runs works, failed step details load again, and an unusual cookie no longer breaks the lists.

• Lists, the home page and scenario history load faster.

0.2.20

Released 2026-09-30

Added

• The project you pick is remembered across the lists and the scenario browser on the canvas.

• Scenarios and runs of one issue, or of one pull request when there is no issue, are listed together.

0.2.19

Released 2026-09-30

Changed

• Shortcuts and help open from a keyboard button in the top bar, and the list now includes <- -> and Esc.

0.2.18

Released 2026-09-29

Changed

• A shorter top bar on the canvas: Projects, Scenarios and Runs, the view switch, and the inbox at the right end.

0.2.17

Released 2026-09-29

Changed

• Projects, scenarios and runs share one layout with the navigation on the left, so switching views does not jump. "All runs" is now "Runs".

• The run verdict has Accept and Return as the main buttons; note, baseline and public sit below them.

0.2.16

Released 2026-09-29

Added

• A projects page: scenarios, runs and how the newest runs stand in each project.

• Console errors in the step detail are highlighted, wrapped and can be copied one by one or all at once.

• Times follow the reader's time zone, set in Settings -> Profile or taken from the browser.

Changed

• Screen size switches say the name and width (Desktop 1440, Mobile 360), and a click anywhere on a card opens the step.

0.2.15

Released 2026-09-29

Changed

• The step detail has three tabs: Criteria, Page and Technical. Criteria that need attention come first; passing ones take one line.

• When only phones are shown, the map shows larger phone screenshots.

0.2.14

Released 2026-09-29

Changed

• Evidence of built-in checks reads as a sentence in Czech or English ("TTFB 950 ms, the limit is 800 ms"); the raw text stays under Technical detail.

0.2.13

Released 2026-09-29

Added

• Every criterion of the built-in packs says why it matters and how to fix a failure, in Czech and English. The walk summary prints the fix for the agent.

0.2.12

Released 2026-09-29

Changed

• The run summary sorts problems by what to do: what failed and why, what a person should decide, and screens that look off.

0.2.11

Released 2026-09-29

Added

• Quick signals on every screen: the evaluator also says whether the page does what it is for, is in the expected language or looks broken. Signals never change a verdict.

• kaloko walk evaluates while it walks and ends with a list of what failed or is unsure.

0.2.10

Released 2026-09-29

Added

• Every criterion says how it was checked, in words: the check as a sentence, or the question the AI was asked with its score and the pass threshold.

• Hosted AI evaluation from the Team plan up, with a monthly allowance shown on the billing page and in kaloko doctor. Your own evaluator key still works.

• Design tokens in the current DTCG format, with aliases, modes and deprecated tokens; kaloko tokens validate.

Fixed

• Manual decisions show the criterion on its own line, with reason and verdict below.

0.2.9

Released 2026-09-29

Added

• Scenarios come first in the navigation, and Runs show the newest run per scenario unless you filter.

• Keep a run so it never expires, prune old runs (dry run first) and archive scenarios you no longer walk, in the app or with kaloko runs, trash, restore, prune, keep and archive.

0.2.8

Released 2026-09-29

Changed

• A claimed sign-in domain lets people join only once it is verified.

• Comments on public runs name nobody, and upload tokens cannot delete runs.

• Below 1500 px the navigation and view options fold into one menu.

Fixed

• The canvas shows the map sooner and loads step details after it; large runs upload faster.

• When the service asks the CLI to slow down, it waits and tries again.

0.2.7

Released 2026-09-29

Added

• Comments on steps, whole runs and whole flows: threads, pins on the screenshot, resolve and reopen. Mentions are mailed at once, the rest goes to the digest, and agents use kaloko comment or MCP.

• Kaloko Design: design drafts as live HTML steps on the canvas, versions, an inspector with token names, comments pinned to elements, edits on the canvas, branches linked to git, and a design pack that checks an implementation against the approved design.

• An HTML view of captured pages (experimental) that can be forked into a design draft.

Changed

• A calmer run view on smaller screens: one top bar, view options in a popover, help behind ?.

Fixed

• A long step path no longer widens its card over the next one.

• macOS desktop captures ask for the Screen Recording and Accessibility permissions they need.

0.2.6

Released 2026-09-29

Added

• Mobile and desktop apps: Android, iOS Simulator, Electron and macOS windows, with UI tree checks, touch target sizes, crash detection and an app pack. kaloko capture --image imports screens from other tools.

• A run index next to the map (key I): every step in a list with filters, search and CSV, and stacks as folders.

0.2.5

Released 2026-09-29

Added

• The HTTP status of every capture, an errors pack for error pages, and {random} in a path for an address that cannot exist. kaloko draft adds a not-found step.

• The run card links the issue next to the pull request.

0.2.4

Released 2026-09-29

Added

• Stacks: one step stands for many pages of one template, taken from a list, a sitemap, a crawl or a script. The canvas shows them as a stack with an index you can filter, search, sort and export as CSV.



A stack opened as a folder · en · desktop

0.2.3

Released 2026-09-28

Added

• acceptDialogs() for step scripts, flags on text checks, kaloko evaluate --recheck after a scenario fix, and kaloko

walk --ephemeral for parallel walks.

Fixed

• kaloko mail reads only messages received since the run started.

• A walk captures fresh screens after --steps, basic auth survives a redirect, and only one walk at a time uses a shared browser.

0.2.2

Released 2026-09-28

Added

• kaloko draft <url…> writes a first scenario and acceptance plan for public pages.

• Check packs a11y, perf and console.

• Live presence on the canvas: who is looking, their cursors, and following someone's view.

• Large flows stay readable: lanes with headers, long lanes wrapped into rows, tidy edges and a zoomed-out view.

• Scenario history with pass rate and criterion stability, and kaloko calibrate to tune evaluator thresholds.

• Print-quality captures for documentation, with masks over personal data and numbered marks.

• The newest capture of every step at a stable address, a list of changes, signed webhooks and kaloko export --scenario for documentation (Business and up).

• GitHub checks on pull requests that complete when the run is accepted or returned.

• A sign-in domain can be proved by signing in with Google Workspace or Microsoft 365, and Settings shows the DNS host with copyable record fields.

Fixed

• The overview follows reviewers' criterion verdicts.

• Public mailbox domains such as gmail.com cannot be claimed.

• A walk stops a session after three steps fail with the same error, instead of trying a wrong password again and again.

• An open menu or popover stays on the capture, signed links in the page lose their credentials, and text check evidence shows the actual match.

• kaloko share says when the plan grants a shorter lifetime than requested.

0.2.1

Released 2026-09-25

Added

• Protected environments: HTTP Basic, service-token headers and client certificates, sent only to the environment's own addresses.

• Accounts per scenario with password and TOTP, and kaloko auth save for a single sign-on a person does once.

• ${VAR} in base_url for review apps, and secrets named in the config are removed from captures.

• Any mailbox through a script adapter, and kaloko mail --file for saved messages.

0.2.0

Released 2026-09-25

Added

• The CLI is on npm (as @qawalk/cli until 0.3.0).

• Plans with a billing page, a 14-day trial and a public pricing page.

• Named sessions: several people in one flow, each in their own browser, shown in swimlanes on the canvas.

• Run verdicts, a baseline run, required approvers and reviewer verdicts for manual criteria.

• Compare a run with the baseline or any earlier run: pixel differences, changed criteria and evidence, and kaloko compare.

• Replay walks the run one step at a time with a branch choice at decisions, and the canvas has a minimap.

• An inbox with notifications, a daily or weekly digest, and Slack.

• kaloko share --pr comments on the pull request; kaloko export and the canvas download a run as a ZIP.

• An MCP server for agents.

• Check packs seo and usability, plan coverage with kaloko validate --coverage, and flaky criteria.

• A public demo run anyone can open without signing in.

• Projects per scenario, and a runs list filtered by project, scenario, verdict, author, environment and branch.

• Sign-in domains verified by a DNS record, token scopes and expiry, a list of your sessions, and a 7-day trash.

• kaloko doctor checks your setup and says what to fix.

• Guides on the website for use cases and agents.

0.1.0

Released 2026-09-23

Added

• The first version. The CLI walks scenarios in Chrome, captures every step, runs the checks and asks an AI evaluator what a check cannot answer. A read-only policy keeps production safe.

• The canvas shows the run as a map of screens with criteria, page metadata (Open Graph, hreflang, structured data, headings) and links to a single step. A click on a criterion highlights it on the screenshot.

• The agent skill for Claude Code and Codex, installed with kaloko init --agent.

• The service: organizations by e-mail domain, sign-in by link, roles and API tokens. kaloko share uploads a run, reviewers approve or comment on each step, and kaloko feedback brings their notes back to the agent.

• The app and the website in Czech and English, with sign-up for new organizations.