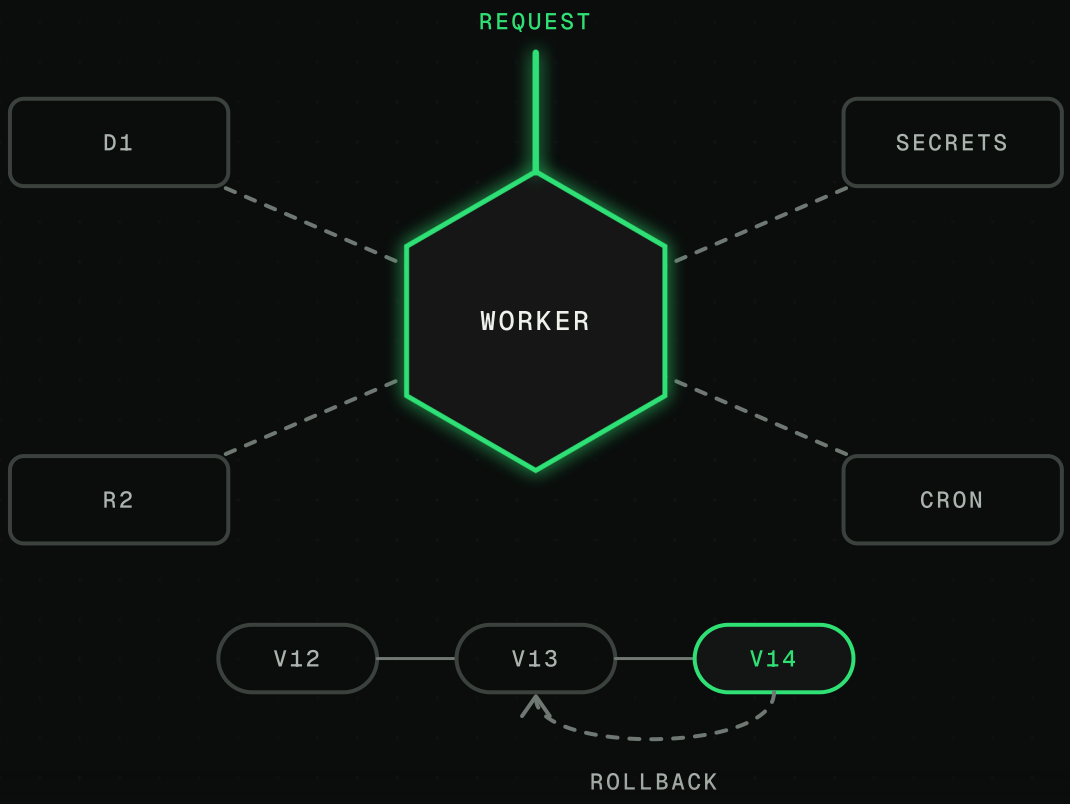


CHECKLIST · FREE

Cloudflare Workers Production Checklist

A 45-check production review for apps on Cloudflare Workers, D1 and R2, with a bindings inventory and a deploy and rollback runbook.

HOW TO USE THIS CHECKLIST

Confirm your Workers app is production-ready, with every binding and secret owned and a rollback path you have rehearsed.

Forty-five verifiable checks, a bindings and secrets inventory, a deploy and rollback runbook, and an observability and limits reference for running an app on Cloudflare Workers, D1, and R2.

WORKING METHOD

01 Work the eight checklist sections in order for a first review, then rerun sections 5 to 8 before every significant release.

02 Treat a check as done only when you hold its evidence: command output, a dashboard reading with a UTC time, or a log line.

03 Fill in the bindings and secrets inventory from the Wrangler configuration and wrangler secret list, never from memory.

04 Replace every bracket in the runbook with your own version IDs, thresholds, and names before the release starts.

05 Reread the linked Cloudflare pages for commands, limits, and retention on the day you use them. They change between releases.

INSIDE

- ✓ 45 production checks
- ✓ Bindings and secrets inventory
- ✓ Deploy and rollback runbook
- ✓ Observability and limits reference

BEFORE YOU START

A Worker rollback changes which code version serves traffic. It does not change D1 data or schema, R2 objects, or other bound resources.

Limits, log retention, and Time Travel windows vary by plan and change. Look them up for your account, do not rely on remembered numbers.

Commands and config keys were read from developers.cloudflare.com on 2026-11-22. Check your installed Wrangler version supports them.

This checklist covers operating on Workers. Moving an app onto Workers from another host needs a separate migration plan.

CONTENTS

01	1. Configuration and environments	6 CHECKS
02	2. Bindings and secrets	6 CHECKS
03	Bindings and secrets inventory	TABLE
04	3. D1	7 CHECKS
05	4. R2 and static assets	6 CHECKS
06	5. Deploys and rollback	6 CHECKS
07	Deploy and rollback runbook	RUNBOOK
08	6. Scheduled and background work	5 CHECKS
09	7. Observability	5 CHECKS
10	Observability and limits reference	TABLE
11	8. Limits, cost and security	4 CHECKS

01 / 11 CHECKLIST

1. Configuration and environments

Make the Wrangler configuration file the single description of each environment, so a deploy never changes production in a way nobody reviewed.

Configuration file

3 CHECKS

- [ ] 01 Run wrangler deploy --dry-run for each environment in CI; expect a clean build from the committed Wrangler configuration, with no setting that lives only in the dashboard.
- [ ] 02 Compare dashboard variables and routes against the configuration file; expect no differences, because wrangler deploy overrides dashboard variables and routes unless keep_vars is set.
- [ ] 03 Read compatibility_date and compatibility_flags in the configuration; expect a deliberate date, changed only after reading the compatibility flags page and retesting the Worker.

Environments

3 CHECKS

- [ ] 04 List every named environment under env and the Worker each one deploys; expect production and staging as separate Workers that use separate databases and buckets.
- [ ] 05 Check each environment block for its own vars and bindings; expect none missing, since both are non-inheritable keys that must be repeated per environment.
- [ ] 06 Confirm workers_dev and preview_urls are set explicitly for production; expect false for both, unless a public workers.dev address or Version URLs are intended and protected.

02 / 11 CHECKLIST

2. Bindings and secrets

Prove every binding points at the right resource and every secret exists where the code expects it, without a value ever reaching the repository.

Bindings

3 CHECKS

- [ ] 07 Compare the bindings declared for production in the Wrangler configuration against the inventory table; expect each database_id and bucket_name to be the production resource.
- [ ] 08 Run wrangler types --check in CI; expect exit code 0, proving the generated Env types match the bindings and variables declared in the configuration.
- [ ] 09 Search staging configuration for production database IDs and bucket names; expect none, and remember that a Version URL runs an uploaded version against production resources.

Secrets

3 CHECKS

- [ ] 10 Run wrangler secret list for each environment; expect every name in the inventory present, and no secret value stored under vars or committed to the repository.
- [ ] 11 List required secret names under secrets.required in the configuration; expect wrangler deploy to fail with a clear error when one is not set on the Worker.
- [ ] 12 Confirm .dev.vars and .env files are covered by .gitignore and absent from git history; expect only one of the two formats in use for local development.

03 / 11

REFERENCE

Bindings and secrets inventory

One row per binding, variable, secret, and deploy credential: what it is, where it is set, who owns it, and how to rotate or recreate it.

Copy this table into your operations document and replace every bracket. Build the rows from the Wrangler configuration and from wrangler secret list for each environment, then delete the example rows you do not use. Record names only. A secret value never belongs in this table: Cloudflare hides secret values in Wrangler and the dashboard once they are set, so the system that issued the secret is the place to get a new one. A row is complete when a second person could rotate or recreate the item using only what the row says.

BINDING OR SECRET	TYPE AND ENVIRONMENT	WHERE IT IS SET	OWNER	HOW TO ROTATE OR RECREATE
[DB]	D1 database binding, production	d1_databases in env.production: database_name [NAME], database_id [ID]	[OWNER]	Restore with Time Travel inside the retention window; otherwise create, apply migrations, import [EXPORT FILE]
[DB]	D1 database binding, staging	d1_databases in env.staging: a different database_id [ID]	[OWNER]	Create with wrangler d1 create, then run migrations apply with --remote
[BUCKET]	R2 bucket binding, production	r2_buckets in env.production: bucket_name [NAME]	[OWNER]	Recreate bucket, reapply lifecycle and CORS from [FILES], restore objects from [COPY LOCATION]
[ASSETS]	Static assets binding, every environment	assets.directory [DIR] and assets.binding in the configuration	[OWNER]	Rebuilt from the repository and uploaded on every deploy
[QUEUE]	Queue producer and consumer, production	queues.producers and queues.consumers, with dead_letter_queue [DLQ NAME]	[OWNER]	Recreate the queue, redeploy, replay messages from [SOURCE OF TRUTH]

BINDING OR SECRET	TYPE AND ENVIRONMENT	WHERE IT IS SET	OWNER	HOW TO ROTATE OR RECREATE
[PUBLIC_BASE_URL]	Plain variable, one value per environment	vars inside each environment block	[OWNER]	Edit the configuration and deploy; not sensitive
[THIRD_PARTY_API_KEY]	Secret, production	wrangler secret put with the production environment selected	[OWNER]	Issue a new key at [PROVIDER], put it, verify a request, revoke the old key
[SESSION_SIGNING_KEY]	Secret, different value in each environment	wrangler secret put per environment; listed in secrets.required	[OWNER]	Generate a new value, put it, expect [EFFECT ON ACTIVE SESSIONS]
CLOUDFLARE_API_TOKEN	Deploy credential, CI only	CI secret store; token created from the Edit Cloudflare Workers template	[OWNER]	Create a new token, update the CI secret, run a deploy, delete the old token
[R2_S3_ACCESS_KEY]	R2 API token for an S3 client, [ENVIRONMENT]	R2 API tokens page; scoped to bucket [NAME] with [PERMISSION]	[OWNER]	Create a new token, copy the secret key once, swap it, revoke the old token

3. D1

Keep schema changes repeatable, keep a restore point you have actually used, and know which queries are doing the most work.

Schema and migrations

3 CHECKS

- [ ] 13 Run wrangler d1 migrations list with --remote against the production database; expect no unapplied migrations, or only the ones named in the current release note.
- [ ] 14 Check that every schema change is a file in the migrations folder and a row in the d1_migrations table; expect no hand-made schema edits in production.
- [ ] 15 Review each pending migration against the Worker version now serving traffic; expect additive changes first, so a code rollback still runs on the new schema.

Recovery and load

4 CHECKS

- [ ] 16 Run wrangler d1 time-travel info for the production database before each migration; expect the current bookmark saved in the release note with its UTC time.
- [ ] 17 Rehearse wrangler d1 time-travel restore on a non-production database; expect an in-place overwrite and a printed previous bookmark that lets you undo the restore.
- [ ] 18 Schedule wrangler d1 export with --remote to storage you control; expect a dated SQL file, taken off-peak because a running export blocks other database requests.

- [ ] 19 Open the database's Metrics tab and run wrangler d1 insights; expect rows read close to rows returned for the top queries, or an index added.

05 / 11 CHECKLIST

4. R2 and static assets

Confirm which buckets are public and how, that old objects expire on purpose, and that the Worker and its static files answer on the right hostnames.

R2 access and lifecycle

4 CHECKS

- [ ] 20 Run wrangler r2 bucket dev-url get for every production bucket; expect the r2.dev URL disabled, since that endpoint is rate limited and meant for non-production traffic.
- [ ] 21 Run wrangler r2 bucket domain list for each bucket; expect custom domains only on buckets meant to be public, and none on buckets that hold private files.
- [ ] 22 Run wrangler r2 bucket lifecycle list for each bucket; expect a deliberate expiry rule for temporary prefixes and a rule that aborts incomplete multipart uploads.
- [ ] 23 Review R2 API tokens in the dashboard; expect object-level permissions scoped to named buckets for applications, and no admin-level token inside application configuration.

Static assets and routing

2 CHECKS

- [ ] 24 Request a missing path and a deep client-side route on the production hostname; expect the status and body your not_found_handling and run_worker_first settings describe.
- [ ] 25 Open Settings, then Domains & Routes for the production Worker; expect every production hostname attached as a custom domain or route that the configuration also declares.

06 / 11 CHECKLIST

5. Deploys and rollback

Make every release traceable to a commit, observable while it rolls out, and reversible by someone who has done the reversal before.

Release path

3 CHECKS

- [ ] 26 Confirm CI deploys with CLOUDFLARE_API_TOKEN held in the CI secret store and limited to one account; expect no token value anywhere in the repository.
- [ ] 27 Pass --message and --tag to every wrangler deploy or wrangler versions upload; expect wrangler versions list to show a commit reference against each version.

- [ ] 28 Run wrangler deployments status after each release; expect the intended version ID at 100 percent, or the planned split while a gradual deployment is running.

Gradual rollout and rollback

3 CHECKS

- [ ] 29 For risky changes, upload with wrangler versions upload and split traffic with wrangler versions deploy; expect errors compared per version before the percentage is raised.
- [ ] 30 If the Worker serves hashed static assets, decide on version affinity before splitting traffic; expect a Cloudflare-Workers-Version-Key rule, or a written choice to skip gradual rollouts.
- [ ] 31 Rehearse wrangler rollback in staging with an explicit version ID and --message; expect the earlier version at 100 percent, with D1 data and R2 objects unchanged.

07 / 11

RUNBOOK

Deploy and rollback runbook

The ordered path for a production release on Workers with D1, the conditions that trigger a rollback, and the parts a rollback leaves untouched.

Fill in every bracket before the release. What a Worker rollback does NOT do, according to the Cloudflare documentation: it does not change resources connected to the Worker, which means bindings and the data in D1, KV, and R2 stay exactly as they are. An applied D1 migration stays applied, and rows written by the new version stay written. Cloudflare warns that errors can occur when code from a prior version meets data whose structure has changed. A rollback is also refused when a Durable Object class lifecycle change sits between the two versions, or when the target version binds to an R2 bucket, KV namespace, or queue that no longer exists. Only recent versions are eligible, and old versions can be deleted under version retention. The rollback page does not say how secrets, cron triggers, or routes behave, so verify those in staging and in the dashboard instead of assuming.

01 Before the release, run wrangler deployments status and record [CURRENT VERSION ID]. Write the rollback triggers: error rate above [ERROR THRESHOLD] for [MINUTES], p95 latency above [LATENCY THRESHOLD], or any data integrity alert. Name [DECISION OWNER].

02 If the release includes a D1 migration, confirm the code in [CURRENT VERSION ID] still works with the new schema. If it does not, split the change: ship an additive migration first and remove old columns in a later release.

03 Run wrangler d1 time-travel info [DATABASE] and save the bookmark with its UTC time as [PRE-MIGRATION BOOKMARK]. Do this even though Wrangler's documentation says migrations apply captures a backup.

04 Run wrangler d1 migrations apply [DATABASE] --remote with the production environment selected. If a migration errors, the docs say it is rolled back and the previous successful migration remains applied: stop the release and fix the migration.

05 Run wrangler versions upload with --message [COMMIT] and --tag [RELEASE TAG]. Record [NEW VERSION ID]. This creates a version that is not deployed. For a low-risk change you may run wrangler deploy instead and continue at step 8.

06 Start the split: wrangler versions deploy [NEW VERSION ID]@[PERCENT]% [CURRENT VERSION ID]@[REMAINDER]% -y --message [NOTE]. Confirm the split with wrangler deployments status.

07 Watch for [OBSERVATION MINUTES]: run wrangler tail --version-id [NEW VERSION ID] --status error, and compare errors and CPU time in the Worker's Metrics view against the previous release.

08 If every trigger stays quiet, promote with wrangler versions deploy [NEW VERSION ID]@100% -y, confirm with wrangler deployments status, and request your smoke-test URLs on the production hostname.

09 If a trigger fires, [DECISION OWNER] calls the rollback and runs wrangler rollback [CURRENT VERSION ID] --message [REASON]. This creates a new deployment at 100 percent for that version, replacing both sides of a split. Allow time for it to reach every location.

10 Verify the rollback: wrangler deployments status shows [CURRENT VERSION ID] at 100 percent, wrangler tail --status error is quiet, and smoke-test URLs pass. Check cron schedules and routes in the dashboard still match the configuration.

11 Decide on data separately. If the old code fails against the migrated schema, ship a forward-fix migration. Run wrangler d1 time-travel restore [DATABASE] --bookmark [PRE-MIGRATION BOOKMARK] only when [DECISION OWNER] accepts losing every write since the bookmark: it overwrites the database in place and cancels queries in flight. Save the previous bookmark it prints.

12 Hold for [OBSERVATION MINUTES] against normal metrics, then close with a note: trigger, UTC times, version IDs, bookmarks used, data lost or repaired, and what must change before the next attempt.

08 / 11 CHECKLIST

6. Scheduled and background work

Make sure scheduled jobs and queue consumers run when expected, fail visibly, and can run twice without harm.

Cron Triggers

3 CHECKS

- [ ] 32 Compare triggers.crons in the configuration with the schedules listed for the Worker in the dashboard; expect identical entries, written for UTC, using day names instead of numbers.
- [ ] 33 Open View events under Trigger Events in the Worker's Settings after a scheduled time passes; expect a recent invocation for every cron expression in the configuration.
- [ ] 34 Read each scheduled handler for idempotency; expect a repeated or skipped run to be safe, and no assumption that a failed cron invocation is retried.

Queues

2 CHECKS

- [ ] 35 Check every queue consumer in the configuration for dead_letter_queue; expect one set, because messages that reach the retry limit are otherwise deleted permanently.
- [ ] 36 Confirm each dead-letter queue has a consumer or a scheduled review; expect failed messages handled before the queue's retention period removes them.

09 / 11

CHECKLIST

7. Observability

Be able to answer three questions within minutes: is it failing now, what failed earlier, and which version did it.

Logs

3 CHECKS

- [ ] 37 Check observability.enabled and head_sampling_rate for production; expect logs on with a sampling rate chosen on purpose, then find a fresh request under the Worker's Observability tab.
- [ ] 38 Log one JSON object per event with a request identifier and no secrets or personal data; expect to filter Workers Logs by field instead of text search.
- [ ] 39 Run wrangler tail with --status error during a release; expect a live stream, knowing real-time logs are not stored and can sample under high traffic.

Errors and metrics

2 CHECKS

- [ ] 40 Throw a test exception in staging while running wrangler tail; expect a stack trace that names original source files, proving source maps were uploaded with the version.
- [ ] 41 Open the Worker's Metrics view and select Errors; expect a named owner who reviews exceeded-resources and thrown-exception counts after each release and every week.

10 / 11

REFERENCE

Observability and limits reference

Where to look for each operational question, what the source can and cannot tell you, and which limits to look up for your plan.

Numbers are left out of the limits rows on purpose. Cloudflare's limits pages list different values for the Workers Free and Workers Paid plans, some limits depend on the zone's plan, and the values change. Open the page named in each row, write the figures for your plan next to your current usage, and date the note. Each limits page links a form for requesting an increase where one is possible.

QUESTION	WHERE TO LOOK	WHAT IT TELLS YOU	CONFIRM FOR YOUR ACCOUNT
Is it failing right now?	wrangler tail --status error, or Logs then Live in the dashboard	Live events as JSON; nothing is stored	Sampling under high traffic; a cap on concurrent viewers per Worker
What failed earlier?	Workers Logs: the Worker's Observability tab	Stored, filterable logs, when observability is enabled	Retention for your plan; daily volume cap and sampling after it
Which version is serving?	wrangler deployments status and wrangler versions list	Version IDs, traffic split, messages, and tags	Wrangler lists recent versions and deployments only
Is one version worse than the other?	wrangler tail --version-id [VERSION ID]	Live events filtered to a single version	Version URLs do not appear in Workers Logs, tail, or Logpush
Are errors or CPU time trending up?	The Worker's Metrics view; select Errors for invocation statuses	Requests, errors, subrequests, CPU time, wall time, exceeded resources	How far back the metrics view reaches
Where did a slow request spend time?	Traces, enabled with observability.traces.enabled	Spans for fetch calls, binding calls, and handlers	Docs describe tracing as beta; confirm status, sampling, and billing
How do we keep logs longer?	Workers Logpush (logpush = true) or OpenTelemetry export	Trace events sent to a destination you own	Logpush needs Workers Paid; field truncation limits
Did the scheduled job run?	Worker Settings, Trigger Events, View events	Recent cron invocations	Only recent events are kept; use Workers Logs for history
Is the database healthy?	D1 database Metrics tab; wrangler d1 insights	Read and write queries, rows read and written, latency, size	Metrics retention; insights is marked experimental
Are queue messages failing?	The dead-letter queue and consumer logs	Messages that exhausted their retries	Retry count and how long unconsumed messages persist
Which Workers limits apply?	developers.cloudflare.com/workers/platform/limits	CPU time, memory, subrequests, Worker size, startup time, variables, cron triggers, asset files	Free and Paid columns; request body size follows the zone plan
Which D1 limits apply?	developers.cloudflare.com/d1/platform/limits	Database size, queries per invocation, statement length, bound parameters, query duration, Time Travel window	Per-database size is a hard ceiling; plan for it early

QUESTION	WHERE TO LOOK	WHAT IT TELLS YOU	CONFIRM FOR YOUR ACCOUNT
Which R2 limits apply?	developers.cloudflare.com/r2/platform/limits	Object and upload size, custom domains per bucket, r2.dev throttling	Whether any r2.dev URL still serves traffic
Is spend under control?	Billing notifications in the dashboard, plus limits.cpu_ms in configuration	Alerts on usage, and a ceiling on CPU per invocation	Whether usage notifications exist for your plan and account type

11 / 11 CHECKLIST

8. Limits, cost and security

Know which ceilings you are closest to, put a cap on runaway work, and protect the endpoints most likely to be abused.

Limits and cost

3 CHECKS

- [ ] 42 Read the Workers, D1, and R2 limits pages for your plan; expect the limits your heaviest route approaches written down with current usage beside each.
- [ ] 43 Set limits.cpu_ms deliberately in the production configuration; expect a ceiling that fits the slowest legitimate request, so a runaway loop is stopped early.
- [ ] 44 Confirm in the dashboard whether usage-based billing notifications are available for your account; expect a threshold set, or a scheduled manual usage review with an owner.

Abuse protection

1 CHECKS

- [ ] 45 Add a rate limiting rule on the zone, or a rate limit binding in the Worker, for sign-in and write endpoints; expect a test burst to be refused.

FIELD NOTES

Common mistakes, useful habits, hard limits.

01 **Treating a code rollback as a data rollback**

wrangler rollback puts an earlier version back at 100 percent of traffic. The D1 schema, the rows written since, and every R2 object stay as they are. If the earlier code cannot read the new schema, the rollback itself causes errors. Ship schema changes that both versions tolerate, and keep a Time Travel bookmark from before each migration.

02 **Changing production in the dashboard**

Cloudflare recommends treating the Wrangler configuration file as the source of truth. Variables and routes edited in the dashboard are overridden by the next wrangler deploy unless you opted out, and a workers.dev address disabled only in the dashboard is re-enabled by the next deploy. Make the change in the file and deploy it.

03 **Assuming an environment inherits its bindings**

Named environments inherit some top-level keys, but vars and bindings such as d1_databases, r2_buckets, and queues are not among them. A production block without its own bindings deploys a Worker that has none, and a block that copies staging IDs writes production traffic into staging data. Review each environment block on its own.

PRACTICAL TIPS

wrangler secret put creates a new version and deploys it immediately. During a gradual rollout, use wrangler versions secret put so the change travels with a version you control.

Pass --remote explicitly on every wrangler d1 command meant for production. Never depend on a default to decide between a local and a remote database.

A Version URL is public when enabled and runs against production resources. Disable preview_urls for production, or put Cloudflare Access in front of it.

Write cron days of the week as names such as MON. Cloudflare numbers days from 1 for Sunday, which differs from many other schedulers.

Keep lifecycle and CORS rules for each bucket as JSON files in the repository and apply them with wrangler r2 bucket lifecycle set and cors set.

WHEN NOT TO USE THIS

A static site on Workers with no D1, R2, secrets, or scheduled work. Section 1, the static assets checks, and a look at the limits page are enough.

A move onto Workers that is still in progress. Finish the migration and its cutover first, then run this checklist against the result.

Workloads built mainly on Durable Objects, Containers, or Workflows. Their deployment and rollback rules differ and are not covered here.

SOURCES · CHECKED 2026-11-22

Cloudflare Workers: Wrangler configuration · Lists inheritable keys and non-inheritable keys (vars, r2_buckets, queues, secrets and other bindings); documents compatibility_date, workers_dev, preview_urls, routes with custom_domain, triggers.crons, observability, keep_vars, limits.cpu_ms, d1_databases, r2_buckets, queues consumers with dead_letter_queue and max_retries, assets with not_found_handling and run_worker_first, and secrets.required. Recommends the configuration file as source of truth: dashboard variables and routes are overridden on the next deploy unless keep_vars is set, and secrets are not deleted by a deploy.
https://developers.cloudflare.com/workers/wrangler/configuration/

Cloudflare Workers: Environments · Environments are defined under env and selected with --env or -e; Cloudflare creates a Worker named top-level-name-environment-name; bindings and vars are non-inheritable and must be defined per environment; secrets are defined per environment.
https://developers.cloudflare.com/workers/wrangler/environments/

Cloudflare Workers: Secrets · wrangler secret put and delete create a new version and deploy it immediately; wrangler versions secret put creates a version without deploying; secret values are not visible in Wrangler or the dashboard after being set; secrets.required makes deploy and versions upload fail when a required secret is missing; use either .dev.vars or .env locally and keep them out of git; do not store sensitive values in vars.
https://developers.cloudflare.com/workers/configuration/secrets/

Cloudflare Workers: Wrangler commands · Documents wrangler deploy (--env, --dry-run, --message, --tag, --keep-vars), versions upload (--message, --tag), versions deploy with version-id@percentage specs and -y, versions list, versions view, deployments list, deployments status, rollback [version-id] with --message, tail with --status and --version-id, secret put, delete, list and bulk, and types --check.
https://developers.cloudflare.com/workers/wrangler/commands/workers/

Cloudflare Workers: Versions and deployments · A version captures code, static assets, bindings and compatibility settings; a deployment sets which versions serve traffic; a new version is created on each code upload or settings or secrets change; state in KV, R2, Durable Objects and D1 is not tracked with versions; only recent versions are retained and deployable.
https://developers.cloudflare.com/workers/configuration/versions-and-deployments/

Cloudflare Workers: Gradual deployments · wrangler versions upload creates a version that is not deployed; wrangler versions deploy splits traffic between versions; by default each request is routed independently, so consecutive requests can be served by different versions.
https://developers.cloudflare.com/workers/configuration/versions-and-deployments/gradual-deployments/

Cloudflare Workers: Rollbacks · A rollback creates a new deployment at 100 percent for the selected version and takes time to reach every location; resources connected to the Worker (bindings, KV, D1, R2) are not changed; errors can occur if the structure of data has changed; rollback is not allowed across a Durable Object class lifecycle change or when the target binds to a deleted R2 bucket, KV namespace or queue; only recent versions are eligible.
https://developers.cloudflare.com/workers/configuration/versions-and-deployments/rollbacks/

Cloudflare Workers: Gradual rollouts and static assets · During a split, HTML from one version can request hashed assets that exist only in the other, producing 404s; the Cloudflare-Workers-Version-Key header gives version affinity and can be set with Transform Rules on a zone.
https://developers.cloudflare.com/workers/static-assets/routing/advanced/gradual-rollouts/

Cloudflare Workers: Version URLs (previews) · Version URLs are public when enabled, are controlled by preview_urls, run an uploaded version against production resources, can be protected with Cloudflare Access, and their logs are not available in Workers Logs, wrangler tail or Logpush.
https://developers.cloudflare.com/workers/configuration/previews/

Cloudflare Workers: workers.dev · Cloudflare recommends running production Workers on a route or custom domain; workers_dev = false disables the subdomain, and disabling it only in the dashboard is undone by the next Wrangler deploy.
https://developers.cloudflare.com/workers/configuration/routing/workers-dev/

Cloudflare Workers: Custom Domains · Custom domains are declared under routes with custom_domain set to true; Cloudflare creates the DNS records and certificates; the dashboard manages them under Settings, Domains & Routes.
https://developers.cloudflare.com/workers/configuration/routing/custom-domains/

Cloudflare Workers: Compatibility dates · compatibility_date opts a Worker into backwards-incompatible runtime changes up to that date; old dates stay supported; review the compatibility flags and test before updating.
https://developers.cloudflare.com/workers/configuration/compatibility-dates/

Cloudflare D1: Wrangler commands · Documents d1 execute, export (--remote, --output, --table, --no-schema, --no-data), time-travel info and restore (--timestamp, --bookmark), migrations create, list and apply (--local, --remote), and the experimental d1 insights; says a backup is captured when applying migrations and that a failing migration is rolled back with the previous successful one remaining applied.
https://developers.cloudflare.com/d1/wrangler-commands/

Cloudflare D1: Migrations · Migrations are .sql files in the migrations folder (migrations_dir to change it); applied migrations are recorded in the d1_migrations table; the page describes no down migrations.
https://developers.cloudflare.com/d1/reference/migrations/

Cloudflare D1: Time Travel · Time Travel is always on with a plan-dependent retention window; time-travel info returns a bookmark; restore overwrites the database in place, cancels in-flight queries and prints a previous bookmark for undo; it cannot yet clone a database to a new copy.
https://developers.cloudflare.com/d1/reference/time-travel/

Cloudflare D1: Import and export data · wrangler d1 export with --remote and --output writes a .sql file; a running export blocks other database requests; export is not supported for databases with virtual tables.
https://developers.cloudflare.com/d1/best-practices/import-export-data/

Cloudflare D1: Metrics and analytics · The database Metrics tab shows read and write queries, rows read and written, query latency and storage; wrangler d1 insights is an experimental command showing top queries.
https://developers.cloudflare.com/d1/observability/metrics-analytics/

Cloudflare D1: Limits · Lists plan-dependent limits (database size, queries per Worker invocation, Time Travel duration) and fixed limits (statement length, bound parameters, query duration); the per-database size limit cannot be increased; each database processes queries one at a time.
https://developers.cloudflare.com/d1/platform/limits/

Cloudflare R2: Public buckets · Buckets are private by default; public access is by custom domain or the r2.dev subdomain; r2.dev access is rate limited and intended for non-production traffic; WAF rules, caching and access controls require a custom domain.
https://developers.cloudflare.com/r2/buckets/public-buckets/

Cloudflare R2: Object lifecycles · Lifecycle rules can expire objects, transition them to Infrequent Access and abort incomplete multipart uploads; buckets have a default rule for incomplete multipart uploads.
https://developers.cloudflare.com/r2/buckets/object-lifecycles/

Cloudflare R2: API tokens · Permission levels are Admin Read & Write, Admin Read only, Object Read & Write and Object Read only; object permissions can be scoped to specific buckets; the Secret Access Key cannot be viewed again after creation.
https://developers.cloudflare.com/r2/api/tokens/

Cloudflare Workers: Wrangler R2 commands · Documents r2 bucket dev-url get, enable and disable; r2 bucket domain list, add and remove; r2 bucket lifecycle list, add, remove and set; r2 bucket cors list, set and delete.
https://developers.cloudflare.com/workers/wrangler/commands/r2/

Cloudflare R2: Limits · Lists object size, upload size and custom domains per bucket limits; r2.dev access is not intended for production and is throttled when its rate limit is exceeded.
https://developers.cloudflare.com/r2/platform/limits/

Cloudflare Workers: Static assets routing · Cloudflare checks for a matching asset first and otherwise invokes the Worker; run_worker_first accepts true or an array of route patterns; not_found_handling controls responses for unmatched navigation requests.
https://developers.cloudflare.com/workers/static-assets/routing/worker-script/

Cloudflare Workers: Cron Triggers · Cron Triggers are set in triggers.crons, run on UTC, can take several minutes to propagate, and number days 1 (Sunday) to 7 (Saturday) with day-name abbreviations recommended; past events are under Settings, Trigger Events, View events. The page does not state a retry guarantee.
https://developers.cloudflare.com/workers/configuration/cron-triggers/

Cloudflare Queues: Dead letter queues · dead_letter_queue is set on a consumer; without it, messages that reach the retry limit are deleted permanently; messages in a dead-letter queue with no consumer are kept for a limited period.
https://developers.cloudflare.com/queues/configuration/dead-letter-queues/

Cloudflare Queues: Batching and retries · A failed batch is retried in full unless messages were explicitly acknowledged; max_retries has a default; explicit acknowledgement is advised for non-idempotent work.
https://developers.cloudflare.com/queues/configuration/batching-retries/

Cloudflare Workers: Workers Logs · Enabled with observability.enabled and head_sampling_rate; viewed under the Worker's Observability tab; retention depends on plan; JSON logging is recommended so fields are indexed; a daily volume limit triggers sampling.
https://developers.cloudflare.com/workers/observability/logs/workers-logs/

Cloudflare Workers: Real-time logs · wrangler tail and the dashboard Live view stream logs that are not stored; high traffic can cause sampling; the number of concurrent viewers per Worker is capped.
https://developers.cloudflare.com/workers/observability/logs/real-time-logs/

Cloudflare Workers: Metrics and analytics · Worker metrics show requests, errors, subrequests, CPU time and wall time; invocation statuses include exceeded resources and Worker threw exception, viewed by selecting Errors in the Metrics summary.
https://developers.cloudflare.com/workers/observability/metrics-and-analytics/

Cloudflare Workers: Traces · Tracing is enabled with observability.traces.enabled and an optional head_sampling_rate, instruments fetch calls, binding calls and handlers, and is described as beta.
https://developers.cloudflare.com/workers/observability/traces/

Cloudflare Workers: Source maps and stack traces · Source maps are controlled by upload_source_maps; source-mapped stack traces are visible when streaming real-time logs and in Tail Workers.
https://developers.cloudflare.com/workers/observability/source-maps/

Cloudflare Workers: Logpush · Workers Logpush is available on the Workers Paid plan, is enabled with logpush = true, and sends Workers trace events to a destination; log and exception fields are truncated past a character limit.
https://developers.cloudflare.com/workers/observability/logs/logpush/

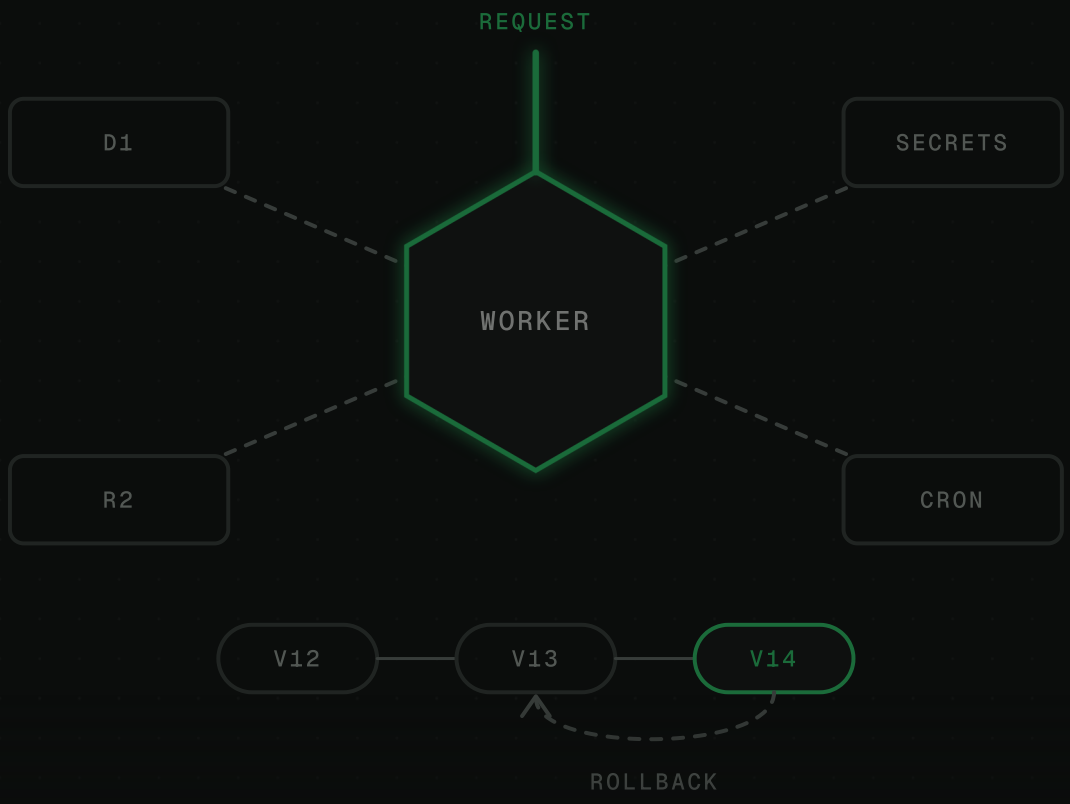
Cloudflare Workers: Limits · Lists per-plan limits for CPU time, subrequests, environment variables, number of Workers, cron triggers and static asset files, fixed limits for memory, Worker size and startup time, and says CPU time can be configured with limits.cpu_ms; links a limit increase request form.
https://developers.cloudflare.com/workers/platform/limits/

Cloudflare Workers: GitHub Actions · CI deploys use CLOUDFLARE_API_TOKEN and CLOUDFLARE_ACCOUNT_ID stored as CI secrets; the token is created from the Edit Cloudflare Workers template and should be scoped to the deploying account; do not store the token in the repository.
https://developers.cloudflare.com/workers/ci-cd/external-cicd/github-actions/

Cloudflare Notifications: Available notifications · A Usage Based Billing notification exists for usage above a set level; its availability depends on plan and account type.
https://developers.cloudflare.com/notifications/notification-available/

Cloudflare WAF: Rate limiting rules · Rate limiting rules define limits for requests matching an expression on a zone; available fields, periods and rule counts vary by plan.
https://developers.cloudflare.com/waf/rate-limiting-rules/

Cloudflare Workers: Rate Limiting binding · The ratelimits binding exposes limit({ key }) returning success; counts are approximate, eventually consistent and separate per Cloudflare location.
https://developers.cloudflare.com/workers/runtime-apis/bindings/rate-limit/



FAISAL KARKOH / RESOURCES

More working files for building with AI.

Prompt packs, checklists, templates and playbooks for shipping reliable AI products. Free to download and reuse in your own work.

LIBRARY
faisalkarkoh.com/resources

WRITING
faisalkarkoh.com/blog

WORK TOGETHER
faisalkarkoh.com/contact