Debugging

- Inspect Cf-Cache-Status
- My Worker runs on every request
- My Worker runs even after the first request
- My cache hit rate dropped after a deployment
- My cache still serves old content after a deployment
- My cache never updates after content changes
- Two callers receive each other's cached responses
- Cf-Cache-Status: UPDATING appears constantly
- Cf-Cache-Status: UPDATING never appears
- Cf-Cache-Status: STALE appears unexpectedly
- My response is larger than the size limit
- I need more visibility

If caching is not behaving the way you expect, the `Cf-Cache-Status` response header is the first place to look. Every response carries it, and its value tells you exactly what happened for that request.

:::note Workers Caching is **your Worker's cache**. When debugging, look at your Worker's traces and responses. Zone-level cache controls and dashboards operate on a separate cache and will not affect what Workers Caching stores or serves. :::

Inspect `Cf-Cache-Status`

Send two requests to the same URL and compare the headers:

```
curl -I https://my-worker.example.workers.dev/api/users/42
curl -I https://my-worker.example.workers.dev/api/users/42
```

Match the status value against the scenarios below.

My Worker runs on every request

`Cf-Cache-Status` is not present. Check that your wrangler version is 4.69.0 or above, and that wrangler.toml or wrangler.jsonc has `cache.enabled = true` for that worker.

`Cf-Cache-Status` is `MISS` on every request, or `DYNAMIC`, or `BYPASS`. Caching is not storing anything, or a bypass rule is firing.

Check your `Cache-Control` header. The response must carry directives that make it cacheable:

- `public, max-age=N` — cached in Cloudflare and browsers for `N` seconds.

A response with `Cache-Control: private` or `no-store` is not stored, and `Cf-Cache-Status` is `BYPASS`.

A response with `Cache-Control: no-cache` *is* stored, but Cloudflare treats every subsequent request as stale and consults your Worker before serving. The exact `Cf-Cache-Status` depends on whether `stale-while-revalidate` is also set:

- With just `Cache-Control: no-cache`, every subsequent request triggers an inline revalidation. `Cf-Cache-Status` is `REVALIDATED` if your Worker returns `304 Not Modified` (body served from cache), or `EXPIRED` if your Worker returns a fresh `200` (body replaced).
- With `Cache-Control: no-cache, stale-while-revalidate=N`, the cached body is served immediately and the Worker runs in the background. `Cf-Cache-Status` is `UPDATING` for the SWR window.

If you wanted long-lived cache hits, use `max-age` instead. Refer to `no-cache` is not a bypass.

If the response carries **no** `Cache-Control` header at all, behavior depends on the status code: Workers Caching applies RFC 9111 heuristic freshness and caches default-cacheable status codes for a heuristic TTL — for example, `200` is cached for 2 hours and `404` for 3 minutes. For the full table of default TTLs, refer to Responses with no `Cache-Control` header are still cached in the configuration reference. If you do not want any of these defaults to apply, set `Cache-Control` explicitly on the response.

Check the request method. Only `GET` and `HEAD` requests are cached. Everything else is `BYPASS`. `GET` and `HEAD` requests for the same URL share the same cache entry — refer to Cache keys for how Cloudflare handles populating the cache from either method.

Check for automatic bypass conditions. Cloudflare bypasses the cache when:

- The response includes a `Set-Cookie` header.
- The request includes an `Authorization` header, unless the response explicitly sets `Cache-Control: public`, `must-revalidate`, or `s-maxage`.

If your Worker unconditionally sets `Set-Cookie` (for example, a session cookie on every response), the response is never cached. Either remove the cookie from cacheable responses, or separate cookie-setting and cacheable responses into different routes.

Check the status code. Workers Caching follows RFC 9111. Responses with status codes that are not cacheable by default (for example, `401`, `403`, `500`) are not stored unless you explicitly mark them with cacheable directives.

A few status codes are never cached, even with explicit `Cache-Control`:

- `520`–`526` are treated as Cloudflare failsafe responses and are never stored.
- `206 Partial Content` returned by your Worker is not stored. Workers Caching handles `Range` requests itself by fetching the full body from your Worker and slicing from the cached entry — if your Worker returns its own `206`, that response is treated as uncacheable. Return a full `200` instead. Refer to `Range` requests.

My Worker runs even after the first request

`Cf-Cache-Status` is `MISS` on the first request but still `MISS` on subsequent requests.

The cache is likely partitioned. The cache key includes the request path, the target entrypoint, and the invocation's `ctx.props`. Two requests that look the same to you may produce different cache keys if any of these differ.

Common causes:

- The URL path or query string differs between requests (even trailing slashes matter).
- The calling Worker passes different `ctx.props` for each request — for example, a different user ID.
- Requests are hitting different named entrypoints of the same Worker.

Cloudflare does not currently expose the cache key composition, so you cannot see the computed key directly. Instead, walk through the components listed in Cache keys and verify each one is the same for both requests.

My cache hit rate dropped after a deployment

This is expected with the default configuration. By default, the Worker version is part of the cache key, so each new version starts from a cold cache and cannot reuse the previous version's cached responses. The first requests after a deploy are misses while the new version's cache fills, then the hit rate recovers.

If you deploy frequently and your responses rarely change between deployments, enable `cache.cross_version_cache` to share cached responses across versions and avoid resetting the cache on every deploy. The trade-off is that cache-affecting changes no longer apply immediately — see below.

My cache still serves old content after a deployment

By default a deployment takes effect immediately, because the Worker version is part of the cache key and the new version starts from a cold cache. If you are still seeing responses from a previous version, you have `cache.cross_version_cache` enabled, which shares cached entries across versions. To force a deployment to take effect while keeping `cross_version_cache` on:

- **Call** `ctx.cache.purge({ purgeEverything: true })` after deploying. This is the simplest approach.
- **Tag each cached response with the producing version** using the version metadata binding, then purge that tag on rollback. Refer to Version-specific purging.

My cache never updates after content changes

If your origin data changed but requests still return stale content:

- **Check the TTL.** The response stays cached for `max-age` seconds. You may be looking at a response that is still within its freshness window.
- **Purge the affected responses.** Use `ctx.cache.purge()` with tags or a path prefix to delete specific entries. Refer to Purging the cache.
- **Add tags at write time.** If you did not set `Cache-Tag` headers, you cannot purge by tag. Add tags to your cached responses, deploy, and once new entries are written they become purgeable.

Two callers receive each other's cached responses

This should not happen if you use `ctx.props` for per-caller authorization context. If it does, one of the following is true:

- You are authenticating callers with a header or query parameter that is not part of the cache key. Move the authorization input into `ctx.props`. Refer to Multi-tenant safety with `ctx.props`.
- You are calling a service binding with a user-specific query parameter that is not present. The query string is part of the cache key; make sure each caller's request path actually differs.

`Cf-Cache-Status: UPDATING` appears constantly

`UPDATING` means the response was served from cache while stale and your Worker is running in the background to refresh it. This is expected behavior when using `stale-while-revalidate`.

If you see `UPDATING` more often than you expect:

- Your `max-age` is shorter than your request arrival rate. Every request that arrives after `max-age` elapses triggers a revalidation.
- With `max-age=0, stale-while-revalidate=<large>`, **every** request triggers a revalidation. This is "always serve from cache" behavior, not "don't run the Worker." Refer to Choose TTL and stale-while-revalidate values.

`Cf-Cache-Status: UPDATING` never appears

`UPDATING` is emitted only when **all** of the following are true:

- A cached entry exists and is past its freshness window (stale).
- The response carries `stale-while-revalidate=N`, and the request arrives within `N` seconds of the entry going stale.
- The response does **not** also carry `s-maxage`, `must-revalidate`, or `proxy-revalidate`.

If any of those is false, requests for stale entries fall through to inline revalidation instead, producing `EXPIRED` (Worker returned a fresh body) or `REVALIDATED` (Worker returned `304 Not Modified`).

Common reasons `UPDATING` does not appear:

- **No `stale-while-revalidate` directive on the response.** The default SWR window is `0`, so without an explicit directive every stale request is foreground-revalidated.
- **`s-maxage`, `must-revalidate`, or `proxy-revalidate` is present.** Per RFC 9111 §4.2.4, these directives forbid serving stale content, so Cloudflare disables `stale-while-revalidate` (and `stale-if-error`) when any of them is present. Use `max-age` for the edge freshness window if you want stale-serving to work.
- **The SWR window has elapsed.** If your response uses `max-age=60, stale-while-revalidate=120`, you will see `UPDATING` for requests arriving in the 120 seconds after the entry goes stale. Requests arriving after that revert to inline revalidation.

`Cf-Cache-Status: STALE` appears unexpectedly

`STALE` means Cloudflare served a previously cached response because your Worker errored on the request that would have refreshed it — for example, the Worker threw, timed out, or returned a `5xx` response. This is `stale-if-error` behavior. Refer to Serve stale on error with `stale-if-error`.

If you see `STALE` and did not expect it:

- **Your Worker is failing on cache fills or revalidations.** Check the Workers observability dashboard for errors on the requests that should have produced a fresh

response. The fact that clients see a stale response instead of a `5xx` is masking a real failure.

- **You did not set `stale-if-error` explicitly, and your response does not include `s-maxage`/`must-revalidate`/`proxy-revalidate`.** In that case, Cloudflare's default behavior is to serve stale responses on Worker error indefinitely, as long as the cached entry has not been purged. If you want errors to surface to clients quickly, set `stale-if-error=0` in `Cache-Control`. For details, refer to Serve stale on error with `stale-if-error`.
- **A previously deployed version is being served.** If you deployed a fix but `STALE` keeps appearing, the cached entry from the broken version is still being served on every error. Purge the affected entries to force a fresh fill from the current version.

To distinguish a `STALE` from a normal `HIT` in client-side observability, log `Cf-Cache-Status` alongside the response — `STALE` is the only signal that the Worker is failing and clients are not seeing it.

My response is larger than the size limit

If a response is too large to cache, Cloudflare does not store it. You will see `Cf-Cache-Status: MISS` on every request even though the response otherwise looks cacheable.

For per-plan response size limits, refer to Cacheable size limits. Note that at launch all Workers Caching responses are subject to the Free plan size limit — refer to Response size for details.

I need more visibility

At launch, the primary debugging surfaces are the `Cf-Cache-Status` response header and per-invocation cache-hit information in the Workers observability dashboard.

□□□ #2
□ □□□ □□□ 2026-11-01 05:33:17 UTC
□ □□□ □□□ 2026-11-01 05:33:20 UTC