Cloudflare Cowboy · Demo Deep Dive

Antoni K Pestka · cloudflarecowboy.com

Companion to the talking-points sheet. The implementation detail behind each demo, plus crisp answers to likely hard questions. Pull from here only when a technical screener pushes; the talking-points sheet is for the skim.

1. Turnstile · server-side verification and "block on fail"

- The token from the widget is worthless on its own. It only becomes trustworthy when **your server** POSTs it, together with your **secret key**, to Cloudflare's siteverify endpoint (`https://challenges.cloudflare.com/turnstile/v0/siteverify`) and gets back `success: true`. The secret key lives in a server-side binding (`env.TURNSTILE_SECRET`) and never ships to the browser. A page that renders the widget but never calls siteverify is theater: a bot can POST your endpoint directly with no token at all.
- "Block on fail" is not a separate feature you install. It is your handler gating the business action on the verify result: if `success` is not true, return 403 and do not create the account / accept the form / process the payment. In the demo, "fail mode" swaps in Cloudflare's documented always-fail test secret (`2x0000...`) so siteverify genuinely returns `success: false` and the function returns `outcome: "blocked"`, which is the real blocked-bot path, not a fake one.
- Hardening worth naming: pass the client IP (`CF-Connecting-IP`) as `remoteip`; treat tokens as single-use and short-lived (about 300s), so verify once server-side then discard; check the `hostname` (and optional `action` / `cdata`) in the siteverify response to bind the token to the form you expected, which stops token replay across endpoints.

2. AI-crawler control

- The demo carries a signature registry (GPTBot, ClaudeBot, Google-Extended, CCBot, PerplexityBot, and more) mapping each to operator, category, and a recommended policy. With `?enforce=1` it returns the real response: HTTP **402** for license-required (with a `Link: rel="license"` header pointing at terms and an `X-Robots-Tag: noai`), or **403** for block.
- It deliberately separates **training crawlers** (GPTBot, license or block) from **user-directed fetches** (ChatGPT-User, allow). Blocking the latter breaks a real person's assistant lookup, so policy is per-class.
- Doing it right: User-Agent alone is spoofable, so production leans on Cloudflare's verified bot signals and bot management (signed verification, ASN, behavior), not just string matching. The dashboard "Block AI bots" toggle plus pay-per-crawl is the no-code version; the Worker is the custom policy layer.

3. Grounded trust assistant (AutoRAG)

- Grounding is enforced two ways, not one. The AI Search system prompt says "answer only from sources, otherwise refuse," **and** the Function independently checks the top retrieval score against a threshold (about 0.3) and looks for the configured refusal sentinel. So even weak, tangential retrieval that clears the model still gets turned into an explicit refusal. The refusal is the trust feature: it never invents a compliance posture.
- Citations are de-duplicated by filename and link to the actual source doc the buyer can open.
- Two real operational notes: the AI Search **similarity cache must stay OFF** (a cache hit returns empty data and causes a false refusal), and Workers AI free tier is **10,000 neurons/day**, so the most common and the should-refuse questions are **pre-answered from the corpus with zero AI spend**, with a graceful "daily limit reached, resets at 00:00 UTC" fallback for novel questions.

4. Provable data residency (D1)

- There are two D1 databases created with location hints (EU = Western Europe / Dublin, US = Eastern North America). The Worker selects the binding (`RESIDENCY_EU` vs `RESIDENCY_US`) by jurisdiction, so residency is decided **at write time**, not asserted in a policy document. The demo toggles country with `?country=DE`.

- Be precise: residency here means **primary storage of customer records**. Reads must route to the same regional store, and backups inherit the region. Non-customer operational metadata (for example aggregated monitoring) may still be processed elsewhere, so do not overclaim. Edge cases like the UK post-Brexit should be a configurable mapping, not hard-coded.

5. Rate limiting

- On Cloudflare **Workers**, this is the native Rate Limiting binding (`env.LIMITER.limit({ key })`). This site runs on **Pages**, where that binding is not supported and the config key is rejected, so I built a **D1 sliding-window** limiter instead: insert one row per hit, `COUNT` rows in the trailing window, return 429 when the count exceeds the limit (5 per 10s here), and opportunistically delete old rows so the table does not grow. Same observable behavior, different mechanism.
- Doing it right: make the limit decision at the **top of the handler**, before any expensive work. Key by client IP (`CF-Connecting-IP`) or by user/route. Trade-off: a write-per-request on D1 is fine at this scale; at high volume the native binding or a Durable Object counter is the better tool. Per-IP limits are weak against distributed bots, so pair them with Turnstile and bot management.

6. Edge cache (Cache API)

- Cache key hygiene matters: the demo builds the key from the request URL with the volatile `reset` flag stripped, so HIT and MISS share the same key. On MISS it renders once, then `cache.put` with a `Cache-Control: public, max-age=...` TTL. Purge is `cache.delete` on that key.
- How it proves the cache is real: the stored payload carries a `renderedAt` timestamp and the `colo` it was served from. The timestamp stays identical on every HIT, which proves you are getting the one stored copy, not a fresh re-render.
- Doing it right: never cache personalized or authenticated responses without varying the key; choose TTL vs freshness deliberately; for whole static pages, a no-code Cache Rule often does the job.

7. Atomic inventory (D1)

- The guard lives **in the WHERE clause**, where the database enforces it atomically, not in application code (a read-then-write in app logic is exactly the race that oversells). You decide bought vs sold out by rows-changed (`meta.changes === 1` is a sale, `0` is sold out), never by a prior SELECT.
- D1 serializes writes, so simultaneous buys cannot double-decrement; the count physically cannot go below zero. Keep the decrement and the order record in one atomic step (or compensate) so you do not decrement then fail to record the sale.

Cross-cutting implementation considerations

- Platform fit: Pages Functions cannot export Durable Objects, and the native Rate Limiting binding is Workers-only, which is why Stampede uses a D1 fallback. Knowing those edges is part of the SE job.
- Secrets: anything sensitive lives in a runtime binding read on the server (`env.X`), never a client-exposed build variable. Health checks report presence and length, never the value.
- AI cost control: free Workers AI is 10,000 neurons/day per account; the trust assistant pre-answers common questions (zero neurons) and degrades gracefully at the cap.
- Local vs deployed: under plain `astro dev` the `/api` Functions return 404; they run under `wrangler pages dev` and in production. Worth saying so a live walkthrough never surprises you.

Hard questions and crisp answers

Conceding a limit and naming the right fix reads as senior.

- **"Isn't User-Agent spoofable? Then the AI-crawler control is not real."** Correct, UA alone is spoofable. That is why production leans on Cloudflare's verified bot signals (signed bot verification, ASN, behavioral signals), not string matching. My registry is the policy layer; the trust signal comes from bot management. The demo shows the decision point and the 402/403 response.
- **"Why D1 and not Durable Objects for the counter and the rate limiter?"** Pages Functions cannot export a Durable Object class, so a DO was not available on this surface. For inventory, a single atomic `UPDATE ... WHERE qty > 0` gives the same race-free guarantee. For a hot, high-throughput counter at scale, a Durable Object (on Workers) or the native rate-limit binding is the better tool. D1 is the right call for correctness and simplicity at this scale.
- **"Isn't request.cf.country spoofable? How accurate is it?"** It is set by Cloudflare at the edge from the connecting IP, not a client header, so a user cannot forge it in the request. It can still be wrong for VPNs and proxies, so for a legal residency guarantee you combine it with the customer's declared region. The demo's point is enforcement at write time, not that geo-IP is perfect.
- **"What breaks at scale?"** The cache, Turnstile, and crawler checks scale trivially because they run at the edge. The weak point is the D1 sliding-window rate limiter: one write per request. At high volume I would switch to the native rate-limit binding or a Durable Object counter. The trust assistant is bounded by Workers AI neurons, handled by pre-baking common answers and moving to the paid tier if needed.
- **"Why Pages instead of Workers?"** Pages gives the static site, the Functions, Git-based CI, and per-branch previews in one place, which fits a content-heavy portfolio. The trade-offs (no DO export, no native rate-limit binding) are exactly why two demos use fallbacks. For a pure API product I would reach for Workers directly.
- **"How do you stop the assistant from hallucinating a compliance answer?"** Two layers. The system prompt restricts it to retrieved sources, and the Function independently checks the retrieval score and a refusal sentinel, so weak retrieval still refuses. Answers carry citations to the source doc. The refusal path is the feature, not a failure.
- **"Are the secrets in the repo safe?"** The committed Turnstile keys are Cloudflare's public, documented test keys, safe by design. Real secrets are runtime bindings (`env.X`), set with `wrangler ... secret put`, never committed and never sent to the client.
- **"How would you productionize this for a real customer?"** Swap test keys for real secret bindings; replace UA matching with bot management; route reads to the same regional store and handle backups; add auth and per-tenant isolation; wire observability (Workers logs and analytics); add CI tests; and use the native rate-limit binding if running on Workers.