Nexus HUB | PLATFORM PLAYBOOK

# White-label the LeadConnector API docs

Put the LeadConnector API documentation on your own subdomain, in your own colors, under your own product name — without forking it, copying it, or maintaining a single page of it. One Cloudflare Worker does the work. You do not need to write code: you fill in four values, paste a prompt into a browser assistant, and approve two changes.

Built from live Nexus Hub coaching sessions — five live calls a week with Charles · www.nexushub.club

**CASE STUDY**

This method is running in production at marketplace.getpinnacle.ai — a full LeadConnector developer portal, fully rebranded, live. That URL is the proof, not a step: it appears nowhere in the prompt or the configuration below. Everywhere you see a gold value, your own domain goes in.

**HOW TO READ THIS GUIDE**

Anything shown like `docs.yourdomain.com` is **yours to replace**. Every other value — including the LeadConnector address and the placeholder IP — is fixed and should be copied exactly as written.

Sections 1 to 5 are the whole job for a non-technical reader. Sections 6 onward are reference material for whoever maintains it later.

## 1. Fill in your four values

Write these down before you start. They are the only decisions you make, and every later step refers back to them.

| Value | What to use | Notes |
|---|---|---|
| Docs to proxy | `https://marketplace.leadconnectorhq.com/docs/` | Fixed. Do not change. |
| Name to replace | `LeadConnector` | Fixed. Capitals matter. |
| **Your subdomain** | `docs.yourdomain.com` | Where the docs will live. The domain must already be on Cloudflare. |
| **Your live site** | `https://yourdomain.com` | The assistant reads your colors, fonts and logo off this page. |
| **Your product name** | `Your Brand` | Exactly as it should read on the page, capitals included. |
| **Worker name** | `yourbrand-docs-proxy` | Lowercase, hyphens only. Any name works; this one reads well in Cloudflare. |

Then open three browser tabs and leave them open: the LeadConnector docs site, your own website, and your Cloudflare dashboard on the zone that owns your subdomain. Sign into Cloudflare first. The assistant works inside the session you are already signed into and never asks for a password.

## 2. Before you paste anything

Read the vendor's terms of service and developer agreement. Three things matter: whether reverse-proxying or rebranding their documentation is addressed at all, whether they sell a white-label or custom-domain tier, and what their trademark language says about removing their name from a rendered page. If a paid white-label tier exists, buying the right is almost always the correct answer instead of engineering around its absence. Technical feasibility is not permission.

Two consequences follow either way. This creates a duplicate of the vendor's content, so the proxy tells search engines not to index it. And your users will now file support tickets with you for a product you do not control.

## 3. The master prompt

This is the whole job for most people. Copy everything in the block below, swap the four gold values for your own, and paste it into your browser assistant.

| Claude browser extension | ChatGPT browser extension |
|---|---|
| Paste the prompt in one piece. It works the phases in order and pauses at the stop conditions on its own. | Paste it one phase at a time, and re-paste the FILL THIS IN table at the top of each phase so your values are never lost. |
| Ask it to report findings as it goes so you can follow along without reading code. | Ask for the full Worker file back after each phase, so you never lose progress between messages. |

**Four edits, then paste.** In the FILL THIS IN table below, replace `docs.yourdomain.com`, `https://yourdomain.com`, `Your Brand` and `yourbrand-docs-proxy`. Leave everything else exactly as written — the rest of the prompt is what keeps the assistant from breaking your code samples or your DNS.

```
# MASTER PROMPT — White-Label a Third-Party Docs Portal onto My Own Domain with a
Cloudflare Worker

## YOUR ROLE

You are acting as a platform engineer with browser access. You will set up a Cloudflare
Worker
that reverse-proxies a third-party documentation site onto a subdomain I control, and
rebrands
it (colors, fonts, logo, favicon, product name) to match my brand — without forking the
upstream site.

Work autonomously through the phases below. Do not ask me to confirm each step; ask only
when
you hit one of the explicit STOP conditions. Report findings as you go, and give me a full
summary plus caveats at the end.

---

## FILL THIS IN BEFORE RUNNING

| Variable | Value |
|---|---|
| `ORIGIN_URL` — the docs site to proxy | `https://marketplace.leadconnectorhq.com/docs/` |
| `PROXY_HOST` — the subdomain to serve it on | docs.yourdomain.com |
| `BRAND_SOURCE_URL` — a live site with my brand | https://yourdomain.com |
| `VENDOR_NAME` — the name to replace in prose | `LeadConnector` |
| `BRAND_NAME` — the name to replace it with | Your Brand |
| `WORKER_NAME` — leave blank to auto-derive | yourbrand-docs-proxy |

I will have open, or will open for you: a tab on `ORIGIN_URL`, a tab on `BRAND_SOURCE_URL`,
and a tab on my Cloudflare dashboard for the zone that owns `PROXY_HOST`. I am already
logged into Cloudflare.

---

## STOP CONDITIONS — ask me before proceeding

1. **Before creating the DNS record or the Workers route.** Show me exactly what you're
about
   to create and wait for my go-ahead. Everything before that point is read-only recon plus
a
   Worker that isn't yet bound to my domain, so proceed freely up to there.
2. If the zone already has a Workers route that your new route would shadow, tell me what
it
```

````
   is and what will stop firing on this subdomain.
3. If you need to install anything, download a file, or spend money.
4. If the origin requires authentication or sets session cookies — proxying an
authenticated
   app has cookie-domain, `SameSite`, and CSRF implications outside this task's scope. Stop
   and explain.

---

## PHASE 0 — Legal gate (do this first, and actually do it)

Before any engineering, look at the vendor's Terms of Service / developer agreement and
report to me, in plain language:

- Whether reverse-proxying, mirroring, or rebranding their docs is addressed at all.
- Whether they sell a white-label or custom-domain tier — if they do, say so plainly,
because
  buying the right is usually the correct answer instead of engineering around its absence.
- Any trademark language relevant to removing their name from the rendered page.

Then state your read and let me decide. Technical feasibility is not permission. If the
terms
clearly forbid it, tell me that and stop.

Also note for me: this creates a duplicate of the vendor's content, so I will probably want
`X-Robots-Tag: noindex` on the proxy (you'll add it in Phase 4), and my users will now file
support tickets with me for a product I don't control.

---

## PHASE 1 — Interrogate the origin

Run these in the console of the `ORIGIN_URL` tab and report the answers. Each one changes
the plan.

1. **Framework and version:** `document.querySelector('meta[name=generator]')?.content`
2. **Is it a SPA?** Look for a React/Vue root and hydration. This is the single most
   consequential finding — see Phase 4.
3. **Base path.** e.g. Docusaurus `baseUrl`. If the site is built to live at `/docs/`, that
   prefix is **compiled into the client router bundle**. You cannot serve it at `/`.
4. **Asset URL shape.** Root-relative (`/docs/assets/...`) is the easy case. Note that the
   filenames carry content hashes that rotate on every upstream deploy — **never key any of
   your code to an asset filename.**
5. **Theme CSS custom properties:**
```js
   [...document.styleSheets].flatMap(s => { try { return [...s.cssRules] } catch { return
[] } })
````
````

```
     .filter(r => r.selectorText === ':root')
     .flatMap(r => [...r.style].filter(p => p.startsWith('--'))
       .map(p => `${p}: ${r.style.getPropertyValue(p).trim()}`))
```
6. **Hardcoded colors that ignore those variables.** Check the hero, navbar, footer, and buttons with `getComputedStyle`. Gradients are the usual offender and must be overridden by selector, not by variable.
7. **Response headers:**
```js
   (await fetch(location.href)).headers.forEach((v,k)=>console.log(k,'=',v))
```
   - `X-Frame-Options` / CSP `frame-ancestors` — if either is restrictive, **an iframe wrapper
     is impossible and reverse proxy is the only option.** Expect this.
   - The CSP itself. If there's no separate `style-src`/`script-src`, your injected inline `<style>` and `<script>` will be allowed. If there is, you'll need to rewrite the CSP header in the Worker — tell me before you weaken anything.

---

## PHASE 2 — Extract my brand tokens

From the `BRAND_SOURCE_URL` tab, capture and report:

- All `:root` custom properties (same snippet as above).
- Resolved semantic tokens via `getComputedStyle(document.documentElement).getPropertyValue(...)`:
  `--primary`, `--background`, `--foreground`, `--muted`, `--muted-foreground`, `--border`, `--ring`.
- **Actually rendered** fonts, not stylesheet intent:
  `getComputedStyle(document.querySelector('h1')).fontFamily` and the body equivalent, plus weights.
- Logo URLs (light and dark variants if they exist) and the favicon URL.

Then derive a full ramp from my primary color, because design systems rarely ship one and doc themes usually want `primary` plus dark/darker/darkest/light/lighter/lightest. Also pick
a dark-mode background and surface color from the darkest end of my ink scale.

---

## PHASE 3 — Cloudflare recon

- Get the Account ID and Zone ID.
- **List existing Workers routes before doing anything.** Cloudflare resolves the most specific route, so a new `PROXY_HOST/*` route will silently override any `*.mydomain.com/*`

```
  wildcard on this subdomain. Report what would stop firing. (STOP condition #2.)
- Check DNS for an existing record on the subdomain, and note the conventions already in
use
  in this zone (naming, placeholder IPs, comments) so the new resources match.
- Note the Workers plan and limits.

---

## PHASE 4 — Build the Worker

Write a **module-syntax** Worker (`export default { async fetch(...) }`). Build it in
layers
and test after each. Known traps are listed — do not rediscover them.

**Proxy core**
- Rewrite the URL to the origin host; preserve method, path, query, headers, body.
- Delete `accept-encoding` on the outbound request so `HTMLRewriter` sees decompressed
HTML,
  and strip `content-encoding` / `content-length` when you rebuild the response.
- Only run `HTMLRewriter` on `text/html`; stream everything else through untouched.
- Add `x-proxied-by: WORKER_NAME` to every response. This is how you prove which route
fired.
- Add `X-Robots-Tag: noindex` unless I've told you otherwise.

**Path strategy**
- Serve the site under the origin's own base path and **302 the root to it**
  (`https://PROXY_HOST/` → `https://PROXY_HOST/docs/`).
- ⚠️ Do **not** try to rewrite `baseUrl` inside the hashed JS bundles. It's a maintenance
trap
  and one upstream deploy breaks it. If serving at the bare root matters to me, tell me the
  tradeoff and let me decide.

**CSS injection**
- Inject a `<style>` into `<head>` via `HTMLRewriter`. Override the theme's CSS variables,
  then the hardcoded selectors you found in Phase 1, then a dark-mode block for whatever
  attribute the theme uses (`[data-theme='dark']` in Docusaurus).
- Load my webfonts and set the base/heading font families.
- ✅ CSS injection survives hydration. This layer is stable.

**Text rebranding — read this carefully, it is where the project actually fails**
- ⚠️ On a SPA, `HTMLRewriter` text edits **do not survive**. React hydrates, detects a
  mismatch between your rewritten server HTML and its own render, throws recoverable errors
  (#418/#423/#425), discards the server markup, and re-renders the vendor's text.
- So the client-side re-brander is not a nice-to-have, it is the actual mechanism. Make it
  robust. It must re-apply on `DOMContentLoaded`, on `load`, on every `MutationObserver`
  mutation, **and** on a short polling interval (~250ms for the first 5s) to cover the
  hydration window. A single run plus a naive observer is not enough — verify it survives.
```

```
- ⚠️ Guard the queued/debounce flag so it always resets. A flag that latches `true`
silently
  disables every subsequent re-run and the rebrand dies after first paint.
- **Never rebrand inside `SCRIPT`, `STYLE`, `PRE`, `CODE`, `TEXTAREA`, `NOSCRIPT`.** API
  endpoints, sample payloads, and curl commands must survive untouched.
- **Match case-sensitively** so lowercase hostnames like `api.vendor.com` inside code
samples
  are never touched.
- Rebrand `document.title` explicitly.

**Favicon and logo**
- ⚠️ If the site uses react-helmet or similar, it **re-renders the `<link rel="icon">`
after
  hydration** and restores the vendor icon. Server-side rewriting alone will not hold.
Enforce
  the favicon from the same client-side loop as the text rebrand. Setting
`type="image/png"`
  alongside `href` is a handy signal that your script actually ran.
- Getting brand images to load, in order of preference:
  1. Fetch them at the edge from a normal static host I control and cache them. Try this
first.
  2. ⚠️ If the brand source is a preview/staging host, expect it to block cross-origin
image
     loads **and** to return its SPA `<!DOCTYPE html>` shell to server-side fetches.
Diagnose
     by fetching the bytes and printing the first eight as hex — `3c 21 44 4f 43 54 59 50`
     means you got HTML, not a PNG.
  3. Fallback: downscale the marks to ~64px via a canvas, embed them as base64 in the
Worker,
     and serve them from Worker-owned routes (`/__brand/mark-light.png`, `/__brand/mark-
dark.png`,
     `/__brand/favicon.png`) with `content-type: image/png` and a long immutable cache
header.
     Keep the total script comfortably small.
- Swap the navbar logo via CSS with separate light/dark rules.

---

## PHASE 5 — Deploy

Prefer **Wrangler** (`wrangler.toml` with `name`, `main`, `compatibility_date`, `routes`;
`wrangler deploy`; `wrangler tail` for logs). Produce the files even if you deploy another
way,
so I own a maintainable copy.

⚠️ The dashboard's Quick Edit editor runs in a **cross-origin iframe** and cannot be driven
programmatically. Don't waste time on it.
```

If you must deploy from the browser, use the dashboard's own same-origin API with my session:

```
PUT /api/v4/accounts/<ACCOUNT_ID>/workers/scripts/<WORKER_NAME>
multipart/form-data:
  metadata (application/json):
    { "main_module": "worker.js", "compatibility_date": "<YYYY-MM-DD>",
      "compatibility_flags": [], "bindings": [], "observability": { "enabled": true } }
  worker.js (application/javascript+module): <source>
```

⚠️ Notes that will otherwise cost you an hour:
- `main_module` (not `body_part`) and the `application/javascript+module` MIME type are what
  make it a module worker. Get either wrong and you'll see confusing validation errors.
- **A `GET` on that same endpoint returns a multipart envelope, not raw JS.** Parse it with
  `response.formData()` before editing, or you'll re-upload the envelope and get
  `SyntaxError at worker.js:1:2`.
- If you're editing an existing script, check how the injected client script is assembled
  (it's often an array of string literals joined together). Inserting raw newlines into a
  string literal breaks it. Match the existing structure exactly, and parse-check your
  generated code with `new Function(src)` before uploading.

Then, **after asking me** (STOP condition #1):

```
POST /api/v4/zones/<ZONE_ID>/dns_records
  { "type":"A", "name":"<subdomain>", "content":"192.0.2.1",
    "proxied":true, "ttl":1, "comment":"Placeholder for <WORKER_NAME>" }

POST /api/v4/zones/<ZONE_ID>/workers/routes
  { "pattern":"<PROXY_HOST>/*", "script":"<WORKER_NAME>" }
```

`192.0.2.1` is TEST-NET-1 and is never contacted — Workers routes only fire on **proxied** hostnames, so the orange cloud must be on. Explain this to me rather than picking a real IP.

---

## PHASE 6 — Verify (and beware two caching traps)

⚠️ **Deploys take 30–60 seconds to propagate, and during that window different requests hit different versions.** Poll until you see the new build consistently before drawing any conclusion. Several of my earlier debugging detours were caused by testing too soon.

⚠️ **The browser caches the proxied HTML.** A plain reload shows you a stale page. Load with a
changed query string, and for a specific asset use `fetch(url, { cache: 'reload' })` on the
**exact plain URL** — probing with `?bust=` tests a different cache key and will fool you into
thinking it's fixed.

Check all of these and report results:

- Root redirects to the base path.
- Homepage: hero, navbar, footer all show my colors and my brand name.
- A deep page renders: sidebar, breadcrumbs, in-page nav.
- Interactive components work (API explorer, request builder, search).
- **Code samples untouched** — the vendor's API hostname still appears correctly inside
  `<code>`. This is the most damaging possible regression; verify it explicitly.
- **Zero occurrences of `VENDOR_NAME` in `document.body.innerText`** on at least two pages.
  Assert this numerically, don't eyeball a screenshot.
- Favicon: correct `href`, correct `type`, and the image actually decodes at the expected size.
- Light and dark mode both correct, including the logo swap.
- **Client-side navigation** between docs pages preserves both the rebrand and the favicon.
  This is the real test of the MutationObserver.
- `curl -I` the proxy and confirm your `x-proxied-by` header.
- Mobile viewport and the mobile nav drawer.

---

## PHASE 7 — Hand off

Give me:

1. The complete Worker source, commented, with a config block at the top holding every value
   I'd need to change.
2. `wrangler.toml`.
3. A list of exactly what you created in my Cloudflare account (worker, DNS record, route).
4. A **synthetic check** I can run on a Cron Trigger: fetch the proxy, assert HTTP 200, assert
   `BRAND_NAME` present and `VENDOR_NAME` absent, alert on failure.
5. An honest caveats list. At minimum cover: which selectors are fragile and will break if the
   vendor redeploys; whether you stripped CSP or `X-Frame-Options` and what that means for
   framing; whether brand assets are embedded and should later be re-pointed at real hosted
   files; what stopped firing because of route precedence; and the legal position from Phase 0.

```
Tell me plainly that this approach is inherently fragile — I am theming someone else's
build
output, and an upstream redeploy can break it. Don't oversell it.
```

**INSIDE NEXUS HUB** A copy-ready version of this prompt sits in the member library, and Charles will run it with you live on a weekly call — www.nexushub.club

## 4. The two things you approve

The assistant runs read-only recon and builds the Worker without touching your live domain. Then it stops and asks. These are the only two changes that go live, and you should read them before saying yes.

**ONE · THE DNS RECORD**

An A record on `docs.yourdomain.com` pointing at **`192.0.2.1`**, with the orange cloud **on**. That address is a reserved test range that is never actually contacted — the record exists only so Cloudflare has something to proxy, and the Worker answers before any server is reached. Copy the IP exactly; do not substitute a real one. Turn the cloud grey and the whole thing stops working.

**TWO · THE WORKERS ROUTE**

A route written as `docs.yourdomain.com/*`, bound to your Worker by name. Before approving, ask what existing routes it shadows. Cloudflare picks the most specific match, so a new rule on this subdomain silently switches off any catch-all rule that used to cover it.

Screenshot: Workers Routes list

or browse files

Workers & Pages → your zone → Settings → Routes, captured *before* the change.

Screenshot: DNS records

or browse files

DNS → Records. Confirm nothing already claims the subdomain.

## 5. Check it before you announce it

Two caching traps first, because both will mislead you. A deploy takes 30 to 60 seconds to spread, and during that window different refreshes hit different versions — wait for it to settle before concluding anything. And your browser caches the proxied page, so a plain reload shows you something stale: add a junk query string to the URL, or use a private window.

- ✓ The bare subdomain redirects to the docs folder.
- ✓ Homepage hero, navbar and footer all show your colors and product name.
- ✓ A deep endpoint page renders: sidebar, breadcrumbs, in-page nav.
- ✓ Search and the interactive API explorer still work.
- ! **Code samples untouched** — the real API hostname still appears inside the code blocks. The most damaging possible regression; check it explicitly.
- ! **Zero occurrences** of the vendor name in visible text, on at least two pages. Count them, do not eyeball a screenshot.
- ✓ Favicon is yours, and the logo swaps correctly in both light and dark mode.
- ✓ Clicking between docs pages keeps the branding and the favicon.
- ✓ Mobile viewport and the mobile nav drawer.

Ask the assistant to run this on the finished page and read you the numbers.

```
// Both numbers must be as marked.
({
  vendorInVisibleText: (document.body.innerText.match(/LeadConnector/g) ||
[]).length, // 0
  vendorInCodeSamples: [...document.querySelectorAll('code, pre')]
    .filter(n => n.textContent.includes('leadconnectorhq.com')).length,
// > 0
  title: document.title,
  favicon: document.querySelector("link[rel~='icon']")?.href,
})
```

## 6. Troubleshooting

| Symptom | Cause and fix |
|---|---|
| Your brand shows for a moment, then the vendor name returns | The page redrew itself after loading and discarded the rewritten markup. The client script needs the 250ms poll for the first five seconds, not just one run plus an observer. |
| Rebrand works on first load, dies after clicking a link | The debounce flag latched true. It must reset at the top of the run function, never at the end. |
| Code examples now point at a hostname that does not exist | The rename walked into a code block, or matched case-insensitively. Both guards are mandatory. Fix this before anything else. |
| Favicon reverts to the vendor's | The site re-renders its own icon link after loading. Enforce yours from the same client loop as the text swap. |
| Garbled HTML, or the styling never appears | The `accept-encoding` header was left on the outbound request, or `content-encoding` was kept on the response. |
| Your logo route returns a web page instead of an image | A staging host returned its HTML shell. Check the first eight bytes for `3c 21 44 4f 43 54 59 50` and fall back to embedding the image in the Worker. |
| Something else on your domain stopped working | Route precedence. The new specific route shadowed a wildcard. Check the routes list captured before the change. |
| Upload fails with a syntax error on line one | The multipart envelope returned by a GET was re-uploaded as source. Parse it with `response.formData()` first. |

## 7. Reference: the Worker

From here on is for whoever maintains this later. The assistant produces this file for you — the version below is a reference to check its output against. Every value you would change lives in the config block at the top, and the four gold values are the same four from section 1.

```
/**
 * Docs proxy — white-labels a third-party docs site onto our own domain.
 * Nothing is forked or copied; the upstream response is re-skinned in flight.
 */
const CONFIG = {
  ORIGIN:      'https://marketplace.leadconnectorhq.com',
  BASE_PATH:   '/docs/',            // compiled into the vendor's client bundle
  VENDOR_NAME: 'LeadConnector',     // case-sensitive on purpose
  BRAND_NAME:  'Your Brand',
  WORKER_NAME: 'yourbrand-docs-proxy',
  NOINDEX:     true,
  PRIMARY:     '#YOURHEX',   // your primary brand color
  ACCENT:      '#YOURHEX',   // your accent color
  FONTS:       'https://fonts.googleapis.com/css2?family=…',
};

export default {
  async fetch(request) {
    const url = new URL(request.url);

    // Worker-owned brand assets (see BRAND_ASSETS below).
    if (url.pathname.startsWith('/__brand/')) return serveBrandAsset(url.pathname);

    // The site cannot run at the bare root — send it to its own base path.
    if (url.pathname === '/') {
      return Response.redirect(url.origin + CONFIG.BASE_PATH, 302);
    }

    // Rebuild the request against the origin, preserving everything.
    const sc-camel-origin-url = new URL(url.pathname + url.search, CONFIG.ORIGIN);
    const headers = new Headers(request.headers);
    headers.delete('accept-encoding');   // so HTMLRewriter sees decompressed HTML
```

```
    headers.set('host', new URL(CONFIG.ORIGIN).host);

    const upstream = await fetch(originUrl.toString(), {
      method: request.method,
      headers,
      body: request.method === 'GET' || request.method === 'HEAD'
        ? undefined : request.body,
      redirect: 'manual',
    });

    const out = new Headers(upstream.headers);
    out.delete('content-encoding');      // we changed the body length
    out.delete('content-length');
    out.set('x-proxied-by', CONFIG.WORKER_NAME);   // proves which route fired
    if (CONFIG.NOINDEX) out.set('X-Robots-Tag', 'noindex, nofollow');

    const response = new Response(upstream.body, { status: upstream.status, headers:
out });

    // Everything that is not HTML streams through untouched.
    if (!(out.get('content-type') || '').includes('text/html')) return response;

    return new HTMLRewriter()
      .on('head', { element(el) { el.append(headInjection(), { html: true }); } })
      .transform(response);
  },
};
```

### 7.1 The injected head

One font link, one style block, one favicon link, one script. The CSS layer survives the page redrawing itself and is stable; the script exists because text edits are not.

```
function headInjection() {
  return `<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link rel="stylesheet" href="${CONFIG.FONTS}">
<link rel="icon" type="image/png" href="/__brand/favicon.png">
<style>${BRAND_CSS}</style>
<script>${CLIENT_SCRIPT}</script>`;
}

// Replace every colour and font below with your own.
const BRAND_CSS = `
:root{
  --ifm-color-primary: #YOURHEX;
  --ifm-color-primary-dark: #YOURHEX;
  --ifm-color-primary-darker: #YOURHEX;
  --ifm-color-primary-darkest: #YOURHEX;
  --ifm-color-primary-light: #YOURHEX;
  --ifm-color-primary-lighter: #YOURHEX;
  --ifm-color-primary-lightest: #YOURHEX;
  --ifm-font-family-base: 'Your Font',system-ui,sans-serif;
  --ifm-heading-font-family: 'Your Font',system-ui,sans-serif;
}
[data-theme='dark']{
  --ifm-color-primary: #YOURHEX;
  --ifm-background-color: #YOURHEX;
  --ifm-background-surface-color: #YOURHEX;
}
/* Hardcoded gradients ignore the variables — override by selector. */
.hero, .heroBanner, header[class*='hero']{
  background:linear-gradient(160deg, #YOURHEX 0%, #YOURHEX 100%) !important;
}
/* Logo swap, light and dark. */
.navbar__logo img{ content:url('/__brand/mark-light.png'); }
[data-theme='dark'] .navbar__logo img{ content:url('/__brand/mark-dark.png'); }`;
```

## 7.2 The client-side re-brander

This is where the project actually fails if it is done carelessly. Nothing here needs changing — it reads its values from the config block.

```
const CLIENT_SCRIPT = `(function(){
  var VENDOR = ${JSON.stringify(CONFIG.VENDOR_NAME)};
  var BRAND  = ${JSON.stringify(CONFIG.BRAND_NAME)};
  // Never rebrand inside these: API endpoints and curl samples must survive.
  var SKIP = {SCRIPT:1, STYLE:1, PRE:1, CODE:1, TEXTAREA:1, NOSCRIPT:1};
  var queued = false;

  function swapText(root){
    var walker = document.createTreeWalker(root, NodeFilter.SHOW_TEXT, {
      acceptNode: function(n){
        var p = n.parentNode;
        while (p && p.nodeType === 1){
          if (SKIP[p.nodeName]) return NodeFilter.FILTER_REJECT;
          p = p.parentNode;
        }
        // Case-sensitive: leaves api.leadconnectorhq.com alone.
        return n.nodeValue.indexOf(VENDOR) > -1
          ? NodeFilter.FILTER_ACCEPT : NodeFilter.FILTER_REJECT;
      }
    });
    var n, hits = [];
    while ((n = walker.nextNode())) hits.push(n);
    for (var i = 0; i < hits.length; i++){
      hits[i].nodeValue = hits[i].nodeValue.split(VENDOR).join(BRAND);
    }
  }

  function enforceFavicon(){
    var old = document.querySelectorAll("link[rel~='icon']");
    for (var i = 0; i < old.length; i++) old[i].parentNode.removeChild(old[i]);
    var l = document.createElement('link');
    l.rel = 'icon'; l.type = 'image/png'; l.href = '/__brand/favicon.png';
    document.head.appendChild(l);
  }

  function run(){
    queued = false;              // reset FIRST — a latched flag kills every re-run
    try {
      if (document.body) swapText(document.body);
      if (document.title.indexOf(VENDOR) > -1){
        document.title = document.title.split(VENDOR).join(BRAND);
      }
```

```
      enforceFavicon();
    } catch (e) {}
  }

  function schedule(){ if (queued) return; queued = true;
requestAnimationFrame(run); }

  document.addEventListener('DOMContentLoaded', run);
  window.addEventListener('load', run);
  new MutationObserver(schedule).observe(document.documentElement,
    {childList:true, subtree:true, characterData:true});
  var poll = setInterval(run, 250);          // covers the redraw window
  setTimeout(function(){ clearInterval(poll); }, 5000);
  run();
})();`;
```

### 7.3 Brand assets

Fetch your marks at the edge from a static host you control and cache them — try that first. If your brand source is a preview or staging host, expect it to block cross-origin image loads and to return its HTML shell to server-side fetches. The fallback below downscales the marks to about 64px, embeds them as base64 and serves them from Worker-owned routes.

```
// Base64 PNGs, ~64px, kept small so the script stays well under limits.
const BRAND_ASSETS = {
  '/__brand/mark-light.png': 'your base64 here',
  '/__brand/mark-dark.png':  'your base64 here',
  '/__brand/favicon.png':    'your base64 here',
};

function serveBrandAsset(pathname) {
  const b64 = BRAND_ASSETS[pathname];
  if (!b64) return new Response('Not found', { status: 404 });
  const bin = atob(b64);
  const bytes = new Uint8Array(bin.length);
  for (let i = 0; i < bin.length; i++) bytes[i] = bin.charCodeAt(i);
  return new Response(bytes, {
    headers: {
      'content-type': 'image/png',
      'cache-control': 'public, max-age=31536000, immutable',
      'x-proxied-by': CONFIG.WORKER_NAME,
    },
  });
}
```

## 8. Deploying and monitoring

Ask for these files even if the assistant deploys another way, so you own a maintainable copy. The dashboard's Quick Edit editor cannot be driven programmatically — do not let anyone spend time on it.

```
# wrangler.toml
name = "yourbrand-docs-proxy"
main = "src/worker.js"
compatibility_date = "2025-01-01"

[observability]
enabled = true

[[routes]]
pattern   = "docs.yourdomain.com/*"
zone_name = "yourdomain.com"

# Deploy, then watch live logs:
#   wrangler deploy
#   wrangler tail
```

Put this on a Cron Trigger so an upstream change surfaces as an alert rather than a customer email. It checks availability and styling; the text rebrand happens in the browser, so the assertion in section 5 remains the real test.

```
export default {
  async scheduled(event, env, ctx) {
    const res  = await fetch('https://docs.yourdomain.com/docs/', { cf: { cacheTtl:
0 } });
    const html = await res.text();
    const problems = [];
    if (res.status !== 200)                problems.push('status ' + res.status);
    if (!html.includes('Your Brand'))      problems.push('brand name missing');
    if (html.includes('>LeadConnector'))   problems.push('vendor name visible');
    if (problems.length) {
      await fetch(env.ALERT_WEBHOOK, {
        method: 'POST',
        headers: { 'content-type': 'application/json' },
        body: JSON.stringify({ text: 'Docs proxy check failed: ' + problems.join(',
') }),
      });
    }
  },
};
```

## 9. Caveats and maintenance

This approach is inherently fragile. You are theming someone else's build output, and an upstream redeploy can break it. Worth saying plainly rather than overselling it.

- **Fragile selectors.** The hero gradient and navbar logo rules are pinned to the vendor's current page structure. A redesign breaks them silently. Nothing is keyed to their hashed filenames, which is the one thing that would break on every deploy.
- **Headers.** Note whether any security header was rewritten or stripped, and what that means for who can now embed your page.
- **Embedded assets.** If your logo and favicon were embedded in the Worker to get it live, re-point them at real hosted files once you have somewhere to put them.
- **Route precedence.** Record what stopped firing when the new route was added.
- **Legal.** Keep the terms-of-service read on file with the build, and revisit it if the vendor launches a white-label tier.
- **Support.** Tickets about a product you do not control now arrive at your desk. Staff for it.