MAH Founder Stack Starter

Build your own stack in about an hour. Free tier. No credit card.

2026-04-29

MAH Founder Stack Starter

Build your own stack in about an hour. Free tier. No credit card.

Melbourne AI Hub · 2026-04-29

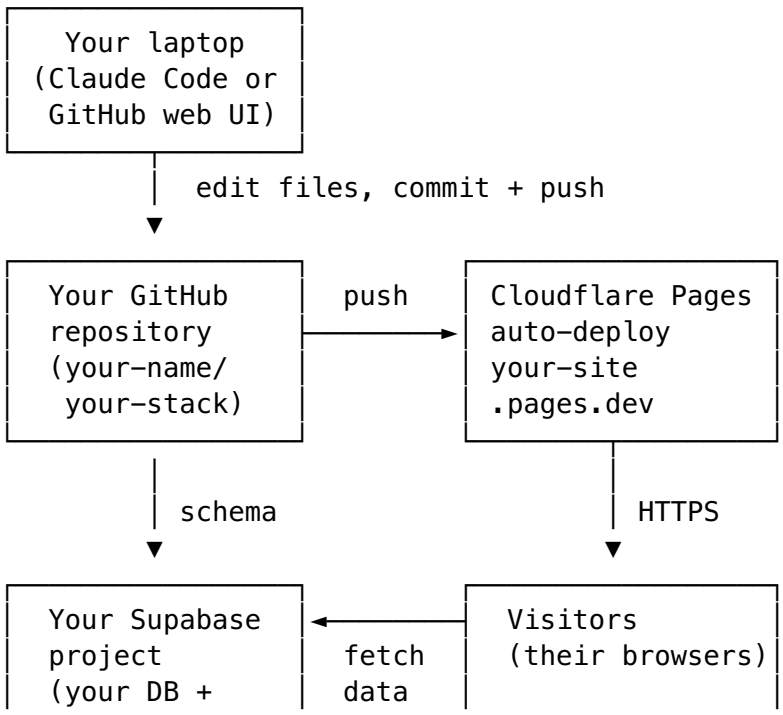
Overview

This is the **founder stack starter handbook**. By the end of it you will have a small, real website live on the internet — at a URL you control, backed by a database you own, deployed automatically every time you push code, all on free tiers.

Time: about one hour, beginning to end. **Cost:** $0. No credit card required to finish chapters 02–07. **Prerequisites:** a laptop, a web browser, and an email address.

You don't need to know any code. You don't need to know how a server works. You don't need to install anything except (optionally) Claude Code in chapter 06 if you'd like an AI agent to help you make changes.

What you'll have at the end



Three accounts (GitHub, Supabase, Cloudflare), all free, all yours. One template you fork. One page you visit (Cloudflare Pages dashboard) to connect them. From that moment on, every edit you push to GitHub deploys itself automatically.

Why this shape

This is the same architecture that powers `melbourneaihub.com.au`, miniaturised. We chose it deliberately:

- **Static HTML at the front** so you can read every line of what runs in the browser, no build step, no bundler, no magic.
- **Supabase at the back** so authentication, a real Postgres database, and Edge Functions all come from a single managed service that has a generous free tier.
- **Cloudflare Pages in the middle** so deploys are automatic on every push, the URL is HTTPS by default, and you don't have to learn what nginx is on day one.

When you later want to learn how to run your own server (the path MAH itself uses in production), the appendix in chapter 09 walks you there. But that's optional, and it's strictly easier once you've already shipped something on the free path.

Reading order

Chapter	What you'll do	Time
01	Sign up for GitHub, Supabase, Cloudflare. No credit card.	10 min
02	Fork the `mah-stack-starter` template into your own GitHub repo.	5 min
03	Create your Supabase project. Run the starter migrations.	10 min
04	Connect your fork to Cloudflare Pages. Site goes live.	5 min
05	(Optional) Buy a custom domain or stick with `*.pages.dev`.	0–10 min
06	Make your first edit. Push. Watch it auto-deploy.	10 min
07	Wire up magic-link login so a real visitor can sign in.	10 min
08	Add a real feature: a contact form that saves to your database.	15 min
09	Submit your live URL and repo to request Tier 2 (MAH collaborator).	5 min
10	Optional next steps: own VPS, own bot, contribute to MAH itself.	—

You can stop after chapter 06 and still have a real working website. Chapter 07 is the moment you can say you've shipped the full stack — frontend, backend, auth — end to end. Chapter 08 makes the stack do something useful; that's where "I built a thing" turns into "I built a thing that helps me."

What this handbook is not

- **Not a Cloudflare tutorial.** We use 3 buttons in the Cloudflare dashboard. If Cloudflare changes its UI, the principles still apply.
- **Not a deep dive on Supabase or Postgres.** We use the SQL editor once, to paste in a migration. You don't need to know SQL to follow it.
- **Not a JavaScript bootcamp.** The starter has plain HTML and a small amount of vanilla JS. The agent does most of the typing if you want it to. You're the operator, not the typist.

When you get stuck

- Tell your AI agent (Claude Code, ChatGPT, Cursor) what you're trying to do and paste the error or screenshot. They are unreasonably good at this stuff.
- Drop a question in the MAH Telegram group.
- Open the Path B handbook (`book-07-founder-onboarding/`) to see how MAH's own production stack works — the shapes are the same.

When you're ready, jump to `01-accounts.md`.

Accounts You Need (All Free)

You will sign up for three services. All three have free tiers that are enough to finish this handbook with room to spare. **You do not need to enter a credit card** to sign up for any of them.

#	Service	What it does for you	Cost on free tier	Sign-up
1	**GitHub**	Stores your code. Every change you make is a snapshot you can roll back to.	$0 forever	https://github.com/signup
2	**Supabase**	Your backend: Postgres database, authentication, email sending.	$0 forever (500 MB DB, 50 MB storage)	https://supabase.com/dashboard/sign-up
3	**Cloudflare**	Hosts your website. Every push to GitHub auto-deploys.	$0 forever (500 builds/month, unlimited bandwidth)	https://dash.cloudflare.com/sign-up

Aim to have all three signed up before you start chapter 02. Plan ~10 minutes total.

1. GitHub — 4 minutes

1. Go to https://github.com/signup.
2. Use the email you'd like associated with your code work.
3. Pick a username. **It will appear in `git log` forever, on every commit you make**, so choose something you don't mind being public. (`olu-mah`, `stephen-research`, your own name — all fine.)
4. Verify your email when GitHub asks.
5. Done. You don't need to set up SSH keys yet — chapter 02 uses the web "Use this template" button which doesn't need keys.

> **Heads-up.** GitHub will offer Copilot, paid plans, etc. during signup. All of it is optional. The free tier is enough.

2. Supabase — 3 minutes

1. Go to https://supabase.com/dashboard/sign-up.
2. Sign up with **GitHub** if you can — it makes future single-sign-on easier. Email signup is also fine.
3. You'll land on the dashboard. **Don't create a project yet** — chapter 03 walks through that with the right settings.

> **Heads-up.** Supabase will ask if you want to add an organisation or upgrade to Pro during signup. Skip both. The free "Personal" org is enough.

3. Cloudflare — 3 minutes

1. Go to https://dash.cloudflare.com/sign-up.
2. Sign up with the email + password of your choice.
3. Verify your email.
4. You'll land on the Cloudflare dashboard. **Don't add a site or buy a domain yet** — chapter 04 walks through Cloudflare Pages, which is a different flow.

> **Heads-up.** Cloudflare's free tier is the most generous of the three. 500 builds/month is enough for ~16 deploys/day, and bandwidth is uncapped on the free tier.

Optional, for later — Anthropic, OpenRouter, a domain

You don't need any of these to finish chapters 02–07. Coming back for them is fine.

- **Anthropic** (https://console.anthropic.com) if you want to use Claude Code as your AI agent in chapter 06. Free trial credits are enough to make several edits. Cursor (https://cursor.com) is the alternative.

- **OpenRouter** (https://openrouter.ai) — one API key, every LLM. You don't need this until you want to add AI features to your own site.
- **A custom domain** — not needed. Cloudflare Pages gives every project a free `*.pages.dev` URL. Chapter 05 covers domains if you want one later.

Quick checklist before chapter 02

- [ ] GitHub account created, email verified.
- [ ] Supabase account created, email verified.
- [ ] Cloudflare account created, email verified.
- [ ] You can log into all three dashboards in your browser.

That's it. No SSH keys yet. No `.env` files yet. No code yet. Move on to `02-fork.md`.

Fork the Starter Template

In this chapter you will create your own copy of the **mah-stack-starter** template repository. After this step, the code is yours — your GitHub username on every commit, your repo name, your right to do whatever you want with it.

Time: 5 minutes. No terminal needed. All steps in the GitHub web UI.

What "use this template" means

GitHub has two ways to copy a repository:

Action	What it does	Use when
Fork	Creates a copy that remembers it came from the original. History is preserved.	You want to contribute back to the original.
Use this template	Creates a fresh repo with the same files, but **no shared history**. Yours, fully independent.	You want to start your own project.

For this handbook you want **Use this template**. You're not contributing back to MAH; you're starting your own thing.

Step 1 — Open the starter

Visit the starter repository in your browser:

```
https://github.com/teddyscleaningserviceaus-design/mah-stack-starter
```

(Once the starter is officially homed in the `Melbourne-ai-hub` org, the URL will redirect — bookmarks still work.)

You'll see the starter's `README.md` rendered on the front page. Skim it if you like, but you don't need to read it now — this handbook covers everything in the README.

Step 2 — Click "Use this template"

Near the top right of the page, find the green **"Use this template"** button. Click it. Choose **"Create a new repository"**.

GitHub takes you to a "Create a new repository from template" form.

Step 3 — Name your repository

Fill in:

Field	Suggestion
Owner	Your own GitHub username (the dropdown defaults to it).
Repository name	Something short, lowercase, no spaces. e.g. `olu-stack`, `frost-research`, `my-first-stack`. You can rename later.
Description	Optional. One sentence. e.g. "My first MAH-pattern stack."
Public / Private	**Public.** Cloudflare Pages free tier deploys public repos most simply. Private is supported but adds setup steps.
Include all branches	Leave **unchecked**. We only want `main`.

Click **"Create repository from template"**.

Step 4 — You now own a repo

GitHub takes you to your new repository. The URL will be:

```
https://github.com/<your-username>/<your-repo-name>
```

Bookmark it. This URL is going to come up a lot in the next chapters.

You should see roughly this file tree on the front page:

```
.env.example
.github/
  workflows/
    deploy.yml
.gitignore
LICENSE
README.md
build.sh
```

```
favicon.svg
index.html
js/
  site.js
styles.css
supabase/
  migrations/
    0001-members.sql
    0002-eoi.sql
vercel.json
wrangler.toml
```

About 13 files in total. If you see a different shape (a new file the starter has added since this handbook was written, etc.) read its README and adapt the steps below — the principles are the same even if file names drift.

What each file does, briefly:

File / dir	Purpose
`index.html`	Your home page. Edit this freely.
`styles.css`	All the styling.
`js/site.js`	Magic-link form handler + Supabase status check.
`supabase/migrations/`	SQL files you'll paste into your Supabase project in chapter 03.
`build.sh`	Substitutes your env vars into HTML at deploy time. Don't edit.
`.env.example`	List of env-var names your app expects. Reference; you don't fill it.
`wrangler.toml`	Cloudflare Pages config metadata.
`vercel.json`	Vercel config (alternative host; ignore if using Cloudflare Pages).
`.github/workflows/`	Optional GitHub Action — most founders won't need it.
`favicon.svg`	The little icon next to your tab title. Replace with your own anytime.
`README.md`, `LICENSE`	Don't need to read; reference and licence info.
`.gitignore`	Tells git what files to never commit (mostly local secrets).

Step 5 — One small "make it yours" edit (optional but recommended)

While you're already in the GitHub web UI, take 30 seconds to make your first edit, just to feel the workflow:

1. Click `index.html` in the file list.
2. Click the pencil icon (top right of the file content) to edit.
3. Find the line that says `<title>{{SITE_NAME}}</title>` and change it to `<title>My First Stack</title>` (or anything else).

4. Scroll down. **Commit changes.** Choose “Commit directly to the `main` branch.” Click **Commit changes** again.

You just made a real commit. It’s in your repo’s history forever (or until you choose to delete it). This was your first push to your own infrastructure.

What you have now

- A GitHub repository, in your name, that you own outright.
- A complete starter codebase ready to deploy.
- An empty `.env` you’ll fill in chapter 03.
- A `wrangler.toml` and a `vercel.json` so chapter 04 can connect to Cloudflare Pages or Vercel without you writing config.

You haven’t deployed anything yet. The repo is just sitting on GitHub. That changes in the next two chapters.

What’s next

- **Chapter 03 — Supabase**: spin up the backend, copy two values into your `.env`, run the starter migrations.
- **Chapter 04 — Host**: connect Cloudflare Pages to your fork. The site goes live.

Troubleshooting

- **“Use this template” is greyed out.** You’re probably not signed in, or you’re on a fork instead of the canonical starter. Make sure the URL says `teddyscleaningserviceaus-design/mah-stack-starter` (or its current home) and you’re logged in.
- **The new repo is empty.** Refresh the page. Sometimes GitHub takes 10–20 seconds to populate a from-template repo.
- **You picked Private and want Public.** Settings → General → Danger Zone → “Change visibility”. One-click toggle.

Create Your Supabase Project

Supabase is the backend half of your stack: database, authentication, and serverless functions, all in one managed service. In this chapter you will create a project, copy two values into your repository’s environment file, and apply the starter database schema.

Time: 10 minutes. **Cost:** $0. No credit card needed.

What is Supabase, in plain English

Supabase wraps a real **Postgres** database (the same database used by banks, scientific data warehouses, and most production web apps) with a friendly dashboard and a few extra services around it:

Supabase feature	What it gives you	Replaces
Postgres	Tables, rows, SQL — your real database	A dedicated database server
Auth	Email + password, magic-link, OAuth — built-in	Auth0, Cognito, Clerk
Storage	File uploads with policies	Direct S3 + a permission layer
Edge Functions	Deno + TypeScript serverless code, called from web	A small backend server
Realtime	Live row-change notifications (websockets)	Pusher, Pubnub

You will use the first two today. Storage, Edge Functions, and Realtime are there when you want them.

Step 1 — Create the project

1. Open https://supabase.com/dashboard in your browser. Sign in.
2. Click **"New project"** (green button, top right or middle of the page depending on whether you have any projects yet).
3. Fill in:
   - **Project name:** something short. e.g. `olu-stack`, `frost-research`. This is purely for your own dashboard, not exposed publicly.
   - **Database password:** click the "Generate a password" button and **save the result somewhere you'll find it again** (your password manager, a notes app — *not* in the repo). You won't need this for normal app use, but you'll need it if you ever connect to the database directly with `psql`.
   - **Region:** pick the one closest to where most of your visitors will be. **Sydney (`ap-southeast-2`)** if you're in Australia like MAH. Region cannot be changed later — choose carefully.
   - **Pricing plan: Free.** Don't add a card.
4. Click **"Create new project"**. Supabase takes 2–3 minutes to spin up. Use this time to read on, or get a coffee.

Step 2 — Find your project ref, URL, and anon key

When the project is ready, the dashboard takes you to its home. The URL in your browser will look like:

```
https://supabase.com/dashboard/project/abcdefghijklmnopqrst
                                       ↑
                                       this is your "project ref"
```

That 20-character string is your **project ref**. You'll see it again.

Then in the left sidebar, click the gear icon → **"Project Settings"** → **"API"**. You're looking at three values:

Value	What it is	Used in
Project URL	`https://<project-ref>.supabase.co`	Frontend (browser) calls Supabase
Project API Keys → `anon` key	Public-safe key. Long string starting with `eyJ…`.	Frontend code
Project API Keys → `service_role` key	Secret. Bypasses all security rules. **Never commit this.**	Edge Functions / server-side scripts

For chapters 03–07 you only need the **Project URL** and the **anon key**. The service-role key stays in the dashboard until you write your first Edge Function (chapter 09 advanced path).

Step 3 — Copy the URL and anon key into your repo

Back in your GitHub repository, you should see a file called `.env.example`. That file lists the environment variable *names* your app expects. We're going to make a real `.env` file in your repo with the actual *values*.

> **Wait — `.env` files have secrets. Should I really put one in a public repo?**
>
> Good instinct. The starter's `.gitignore` lists `.env` so it never gets committed accidentally. The values we're about to put in it (`anon` key and project URL) are **safe to expose** — they're called "anon" because they're meant to be exposed; the database's row-level security policies are what actually protect your data. We'll *never* put the `service_role` key in `.env` for browser code.

For this chapter we'll set the values via the **Cloudflare Pages dashboard** in chapter 04 instead of a local `.env` file — that's cleaner and matches the "no terminal needed" promise. Just keep these two values in front of you for the next chapter:

```
SUPABASE_URL=https://<your-project-ref>.supabase.co
SUPABASE_ANON_KEY=eyJ...                                    (the long
anon-key string)
```

Paste them into your password manager or a scratchpad. Don't email them to yourself or paste into chat. (They're "anon" / public-safe, but hygiene matters.)

Step 4 — Run the starter migrations

The starter repo includes SQL files in `supabase/migrations/` that set up:

- A `members` table with row-level security so each user can only see their own row.
- An `eoi_submissions` table for "expression of interest" form data.
- The standard auth-related triggers Supabase needs to wire up the `auth.users` ? `members` relationship.

You're going to apply these by **pasting the SQL into the Supabase SQL editor**.

1. In the Supabase dashboard, left sidebar → **"SQL Editor"** (the icon that looks like a chevron-and-text).
2. Click **"+ New query"**.
3. In a new browser tab, open your GitHub repo's first migration file: `supabase/migrations/0001-members.sql`. Click the file → **"Raw"** to see the unrendered text.
4. Copy the full file content.
5. Paste it into the Supabase SQL editor.
6. Click **"Run"** (bottom right). You should see "Success. No rows returned." in green.
7. Repeat for `0002-eoi.sql`.

If you get a red error message:

- **"already exists"** — the migration ran before, ignore it.
- **"permission denied"** — you ran an `ALTER ROLE` that needs higher privileges; the starter's migrations don't, so this almost certainly means you copied the wrong file.
- Anything else — paste the error to your AI agent and ask what it means.

Step 5 — Confirm the schema is in place

Left sidebar → **"Table Editor"**. You should see:

- `members` with columns `id`, `email`, `created_at`, `tier`.
- `eoi_submissions` with columns `id`, `email`, `note`, `created_at`.

Click `members` → **"Insert"** → **"Insert row"**, fill in any test email, click "Save". A row should appear. (Delete it after — it's just a smoke test.)

The schema works. The database is yours.

What you have now

- A Supabase project in your account, in your region, on the free tier.
- Two values (URL + anon key) ready to paste into Cloudflare Pages.
- The starter's tables created with row-level security enabled.

What's next

- **Chapter 04 — Host**: connect your GitHub fork to Cloudflare Pages, paste the two Supabase values as environment variables, watch your site go live at a `*.pages.dev` URL.

Troubleshooting

- **Project stuck "Setting up…" for >5 min.** Refresh the dashboard. If still stuck, the free-tier region may be temporarily congested; try delete + recreate in a different region.

- **Can't find the API keys page.** It's at `https://supabase.com/dashboard/project/<your-ref>/settings/api`.
- **The anon key is huge (>200 chars).** That's normal. It's a JWT. Copy the whole thing.
- **Accidentally exposed your `service_role` key.** Project Settings → API → "Roll service role key" (it's a button at the bottom). Costs nothing, takes 5 seconds. Anything that was using the old key needs re-setting.

Host on Cloudflare Pages

In this chapter you connect your GitHub repository to **Cloudflare Pages**. After this step, your site is live on the public internet at a free `*.pages.dev` URL with HTTPS. Every future push to GitHub will auto-deploy.

Time: 5 minutes. **Cost:** $0. No credit card needed.

What Cloudflare Pages does

Cloudflare Pages is a service that watches a GitHub repository and, every time you push a commit, builds the site and deploys it to a global CDN. You don't run a server. You don't write deploy scripts. You don't think about TLS certificates. The Pages dashboard does all of it.

The free tier is generous:

- 500 builds per month (≈16 deploys per day)
- Unlimited bandwidth
- Unlimited requests
- Unlimited static-asset storage
- Custom domains supported (we cover that in chapter 05)
- Workers / Edge Functions: 100k requests/day free (we don't use them in this chapter)

You will not run out of free tier on this handbook.

Step 1 — Open Cloudflare Pages

1. Sign into https://dash.cloudflare.com.
2. In the left sidebar, click **"Workers & Pages"**.
3. Click the **"Pages"** tab at the top.
4. Click **"Create application"** → **"Pages"** tab → **"Connect to Git"**.

> If you see a **"Connect GitHub"** button, you haven't authorised Cloudflare to read your GitHub yet. Click it. GitHub will ask which repos to grant access to — pick **"Only select repositories"** and choose your starter fork specifically. (You can grant all repos if you prefer; single-repo access is just stricter and a good habit.)

Step 2 — Pick your repository

After authorising GitHub, Cloudflare shows a list of your repos. Find your fork (e.g. `your-name/olu-stack`) and click **"Begin setup"**.

Step 3 — Configure the build

You'll see a "Set up builds and deployments" form. The starter has a tiny build script (`build.sh`) that injects your Supabase config into the HTML at build time — that's why the Build command below is `bash build.sh` rather than blank. Fill in:

Field	Value
Project name	Cloudflare suggests one based on your repo name. Keep it or rename. This becomes your URL: `<project-name>.pages.dev`.
Production branch	`main` (already filled).
Framework preset	**None.** (The dropdown defaults to None for plain HTML.)
Build command	**`bash build.sh`** — runs the substitution script that injects your Supabase env vars into `index.html`.
Build output directory	Leave blank (or type `/`). The whole repo is the site.
Root directory	*Leave at default (the repo root).*

Step 4 — Set environment variables

Scroll down to **"Environment variables (advanced)"**. Click **"Add variable"** twice and paste in the two values you saved from chapter 03:

Variable name	Value	Type
`SUPABASE_URL`	`https://<your-project-ref>.supabase.co`	Plaintext
`SUPABASE_ANON_KEY`	`eyJ…` (the full anon key from Supabase Project Settings → API)	Plaintext

> The anon key is safe in plaintext. We'll use **Encrypted** type for the service-role key when/if you add Edge Functions in the future.

> **Why these specific names?** The starter's `build.sh` reads `SUPABASE_URL` and `SUPABASE_ANON_KEY` from the build environment. If you name your variables anything else, the substitution won't happen and your site will load with the placeholder strings still in it — which makes the backend status badge show "not configured." So: spell them exactly as shown.

Click **"Save and Deploy"**.

Step 5 — Wait 30–60 seconds

Cloudflare runs the (no-op) build, copies the files to its CDN, and provisions a TLS cert. You'll see live build logs roll past. When it's done, you'll see a green **"Success!"** banner with your URL:

`https://<your-project-name>.pages.dev`

Click it. **Your site is live.**

It should look like the starter's placeholder page — probably the `<title>` you edited in chapter 02, plus a "Hello from your stack" heading and a Supabase status indicator.

Step 6 — Sanity-check the Supabase wiring

The starter's home page includes a small "Backend status" badge that runs in your browser, calls Supabase using your `SUPABASE_URL` and `SUPABASE_ANON_KEY`, and reports whether it could read the `members` table.

If you see "Backend connected ✓" — chapter 03 + 04 are wired correctly, move on to chapter 06.

If you see "Backend unreachable" or a red error:

- Open your browser's developer tools (right-click → Inspect → Console).
- The error message will be either:
  - **"Failed to fetch"** — `SUPABASE_URL` is wrong. Check for typos or trailing slash.
  - **"Invalid API key"** — `SUPABASE_ANON_KEY` is wrong. Re-copy from Supabase → Project Settings → API → `anon public` key.
  - **"row-level security policy violated"** — chapter 03's migrations didn't run cleanly. Re-paste them in the SQL editor.
- Fix the env var in Cloudflare Pages → your project → **Settings** → **Environment variables** → edit → **Save and redeploy**. The next build runs immediately.

What you have now

- A live HTTPS website at `<project-name>.pages.dev`.
- Auto-deploy on every push to `main`. (Test it: open a file in GitHub, edit one character, commit. Within 60 seconds the live site updates. Or wait for chapter 06 — same idea, more deliberate.)
- A working backend connection to your Supabase project.
- Zero servers, zero certbot, zero ssh.

What's next

- **Chapter 05 — Domain (optional):** point a custom domain at your Pages site. Skip this if `*.pages.dev` is fine for now (it always will be — many real production sites stay on `*.pages.dev`).

- **Chapter 06 — First edit:** the muscle memory of edit → commit → push → live. With or without an AI agent.

Troubleshooting

- **"Build failed."** Open the build log. The starter has no build, so a fail almost always means a typo in `wrangler.toml` (e.g. you edited it). Restore from the template's original.
- **The URL says "Site not found."** Cloudflare can take 1–2 minutes for first DNS to propagate. Wait. Refresh.
- **Cloudflare keeps asking me to upgrade to Pro.** Ignore. The free tier is enough for this handbook and far beyond.
- **You want Vercel instead.** The starter has a `vercel.json` too — same idea, different dashboard. https://vercel.com/new, "Import Git Repository", same env vars, deploy.

A Custom Domain (Optional)

This chapter is **optional**. Your site is already live at `<project-name>.pages.dev` from chapter 04 — that URL is HTTPS, has a valid TLS cert, works on every device, and will keep working forever. Many real production sites stay on `*.pages.dev` indefinitely.

Read this chapter only when:

- You want a memorable URL like `frostresearch.au` or `your-name.com`.
- You already own a domain and want to point it at your site.

If neither applies, skip to **chapter 06 — First edit**. Come back here later.

Time: 5–15 minutes (mostly waiting for DNS). **Cost:** $5–$15/year for the domain registration. **No card needed if you skip this chapter.**

Two situations

A. You don't own a domain yet

Buy one through Cloudflare itself — it sells domains at-cost (no markup, unlike most registrars):

1. https://dash.cloudflare.com → **"Domain Registration"** in the left sidebar.
2. Search the name you want. Cloudflare shows what's available.
3. `.com` is usually $9–$10/year. `.au` is ~$13/year. Many alternative TLDs (`.dev`, `.app`, `.xyz`) are similar. Pick one.
4. Add to cart, check out (yes, this step does need a card — it's the only step in this whole handbook that does, and only if you do this chapter).
5. The domain will appear in your Cloudflare dashboard immediately. DNS for it lives at Cloudflare automatically.

B. You already own a domain

If your domain is registered elsewhere (Namecheap, GoDaddy, Hostinger, etc.), you have two options:

- **Easiest** — transfer the domain to Cloudflare. Same dashboard does it; transfer takes 5–7 days; saves you money in the long run because Cloudflare doesn't markup.
- **Faster** — keep the registration where it is, but point its nameservers at Cloudflare. This brings the DNS records into the Cloudflare dashboard while leaving the registration alone. Search the Cloudflare dashboard for "Add a site" and follow the wizard.

For the rest of this chapter we assume your domain is now manageable inside the Cloudflare dashboard, however it got there.

Step 1 — Attach the domain to your Pages project

1. https://dash.cloudflare.com → **"Workers & Pages"** → click your Pages project (e.g. `olu-stack`).
2. Click the **"Custom domains"** tab.
3. Click **"Set up a custom domain"**.
4. Type the domain you want, e.g. `frostresearch.au` or `www.frostresearch.au`. Click **"Continue"**.
5. Cloudflare offers to add the DNS record automatically (because you manage the domain in Cloudflare). Click **"Activate domain"**.

That's it. Within 30 seconds, `https://frostresearch.au` (or whatever you chose) routes to your Pages site, with a valid TLS cert provisioned automatically.

Step 2 — Decide if you want apex or `www`

Choice	Pros	Cons
Apex only (`frostresearch.au`)	Cleaner. What people will type.	None for static sites on Cloudflare.
`www` only (`www.frostresearch.au`)	Older convention; some users type `www.` reflexively.	The bare apex won't work without a redirect.
Both, apex canonical (recommended)	Best of both. `www.` redirects to apex.	One extra config step (covered below).

To do "both, apex canonical": add **both** custom domains in the Pages custom-domains UI. Then Cloudflare → your domain → **"Rules"** → **"Page Rules"** → add a rule: `www.frostresearch.au/*` → "Forwarding URL (301 Permanent Redirect)" → `https://frostresearch.au/$1`. Save.

Step 3 — Wait, then test

DNS changes take 1–60 seconds inside Cloudflare's network. Test in your browser. If you don't see your site:

- Open an incognito/private window — your normal window may have a stale DNS cache.
- Run `dig frostresearch.au` in a terminal, or visit https://dnschecker.org/#A/frostresearch.au to see the propagation state across the world.

Step 4 — Update your starter's site config

Open `wrangler.toml` in your repo and set the `name` field if it's still the default. (If it already matches your Pages project name from chapter 04, nothing to change.)

If your starter's HTML mentions a hard-coded URL (e.g. for canonical links, Open Graph tags, etc.), update those to your new custom domain. Search the repo for `pages.dev` and replace where appropriate.

Commit the changes. Cloudflare auto-deploys.

What you have now

- Your site at the URL of your choosing, served over HTTPS.
- A renewable annual cost of $5–$15 instead of free.
- A canonical brand presence — useful for marketing, business cards, email signatures.

What's next

- **Chapter 06 — First edit:** make a real change to your site, commit, push, watch it auto-deploy. Whether you stayed on `*.pages.dev` or added a custom domain, the workflow is the same.

Troubleshooting

- **"Domain already in use"** — your domain is attached to a different Pages project. Detach it from the old one first (Custom domains → remove).
- **TLS cert pending for >5 min** — Cloudflare's automatic cert provisioning sometimes hits rate limits. Wait 10 min. If still pending, detach + re-attach the custom domain.
- **You bought a domain through GoDaddy / Hostinger and don't want to move it** — point its nameservers at Cloudflare instead (see "Add a site" in the Cloudflare dashboard). The registration stays where it is; only DNS moves. No transfer fees.

Your First Edit

This is the chapter that earns the rest of the handbook. After this, you have done the **edit → push → live** loop end-to-end, with your name on the commit and your stack auto-deploying. The first time it works it feels small. By the tenth time it feels like a superpower.

Time: 10 minutes. **Cost:** $0.

Two ways to make the edit

Pick whichever fits your comfort level. Both produce identical results.

Way	Pros	Cons
A. GitHub web UI	Zero install. Edit straight in the browser, click commit, done.	Limited to small edits one file at a time.
B. Claude Code on your laptop	The agent does the typing. Best for "change four files at once" or "find every place that says X and update".	One-time install of Claude Code + clone the repo.

If you've never edited code before, start with **Way A** to feel the loop. Come back for Way B in 10 minutes when you want more leverage.

Way A — GitHub web UI

We're going to change the homepage hero — the big text at the top of your starter's `index.html`.

1. Go to your repo on GitHub: `https://github.com/<your-username>/<your-repo>`.
2. Click `index.html`.
3. Click the pencil icon at top right of the file viewer (the "Edit this file" button).
4. Search the file (Ctrl/Cmd-F) for `Hello from your stack` (or whatever the starter's hero text currently says — could be `{{HERO}}` or a placeholder).
5. Replace the text with something of your own. e.g. `Frost research, built in the open.` or `<your name>'s first stack — shipped 2026-04-29`.
6. Scroll down. **Commit changes.**
   - **Commit message:** "first hero edit"
   - **Commit directly to the `main` branch** (the default).
7. Click **Commit changes** again to confirm.

GitHub takes you back to the file. Done. You just made a real commit.

Verify the auto-deploy

Open Cloudflare Pages dashboard → your project → **Deployments** tab.

Within 5 seconds of your commit, a new deployment row appears with status "Building…". 30–60 seconds later it flips to "Success".

Open your site URL (`<project>.pages.dev` or your custom domain). Force a hard refresh:

- **macOS:** Cmd-Shift-R
- **Windows / Linux:** Ctrl-Shift-R

Your new hero text is live. **You shipped a change to a website you own.** That's the moment.

Way B — Claude Code on your laptop

If you'd like to make bigger edits with an AI agent doing the typing:

B1. Install Claude Code (3 minutes)

Open a terminal (Mac: Spotlight → "Terminal"; Windows: Win → "PowerShell").

```
npm install -g @anthropic-ai/claude-code
```

If you don't have `npm`: install Node.js first from https://nodejs.org (Long-Term Support version). Restart your terminal after the install.

Then in your terminal:

```
claude-code login
```

It opens your browser to log in. You'll need an Anthropic console account (https://console.anthropic.com) — free tier credits cover several edits.

B2. Clone your repo (1 minute)

```
cd ~                          # or wherever you keep projects
git clone https://github.com/<your-username>/<your-repo>.git
cd <your-repo>
```

B3. Run Claude Code in your repo

```
claude-code
```

The agent starts. Type a prompt:

> Change the hero subtitle in index.html to "My second edit, this time with an agent." Then commit it with a one-line message and push.

The agent will:

1. Read `index.html` to find the hero subtitle.
2. Edit the file.
3. Show you a diff.
4. Ask if it should commit. Type `y`.

5. Run `git add`, `git commit`, `git push`.

Within 60 seconds Cloudflare Pages auto-deploys. Refresh your site — new subtitle, no terminal commands typed by hand.

What you've actually learned

The full deploy loop is just three steps:

1. **Edit** a file (in browser or with an agent).
2. **Commit + push** to `main` on GitHub.
3. **Cloudflare Pages auto-deploys** — you don't do anything.

Every change to a real production website MAH or any other small site ever makes follows this loop. The shapes scale up — bigger sites add a review step before merging to `main`, run automated tests, deploy to a preview URL first — but the spine is identical.

Things to try while you're warm

Pick one. Each is ~5 minutes. Each ships a real visible change to your live site.

- **Change the page title.** `<title>...</title>` in `index.html`. This is what shows in the browser tab.
- **Change the favicon.** Replace `favicon.svg` (or `favicon.ico`) with any SVG. Search "favicon generator" if you want to design one.
- **Add your name to the footer.** Find the `<footer>` in `index.html`, edit the text inside.
- **Add a second page.** Copy `index.html` → `about.html`. Edit its content. Cloudflare Pages will serve it at `/about.html` automatically.

Each of these gives you a slightly more confident understanding of "I own this and can change anything."

What's next

- **Chapter 07 — Add magic-link login**: connect the form on your site to your Supabase project so a real visitor can sign in. This is the full-stack moment.

Troubleshooting

- **My commit didn't trigger a deploy.** Check Cloudflare Pages → Settings → Builds & deployments → make sure "Production branch" is `main` (not `master` or something else).
- **The site shows my old text after refresh.** Force-refresh again (Cmd-Shift-R / Ctrl-Shift-R). If still stuck, Cloudflare is caching; wait 60 seconds or disable cache in dev tools temporarily.
- **The agent edited the wrong file.** Tell it. "That edit was in the wrong file — please revert and edit `index.html` instead." It will.

- **Claude Code says "could not authenticate"** — `claude-code login` again; the token may have expired.

Add Magic-Link Login

Until now your site has had a frontend (Cloudflare Pages) and a backend (Supabase) but nothing connecting them with real user accounts. This chapter wires up **magic-link authentication** — the sign-in flow where a visitor enters their email, gets a one-time link in their inbox, clicks it, and is signed in.

After this chapter you can say, honestly, that you've built and deployed a full-stack web app — frontend, backend, auth — end to end.

Time: 10 minutes. **Cost:** $0.

What "magic link" means

Instead of asking visitors to invent and remember a password, you ask for their email and Supabase emails them a one-click sign-in link. Benefits:

- No passwords to store, hash, or have stolen.
- Email is already most users' identity anyway.
- Built into Supabase auth out of the box; you write almost no code.

The same flow MAH uses for `members.html`. Same primitives, same Supabase function under the hood.

Step 1 — Confirm Supabase auth email is configured

1. Supabase dashboard → your project → left sidebar → **"Authentication"** → **"Providers"**.
2. Make sure **"Email"** is **enabled** (it is by default on new projects).
3. Click into **"Email"**. The default settings work for testing:
   - **"Confirm email"** ON
   - **"Secure email change"** ON
   - **"Mailer Autoconfirm"** OFF (we want users to click the link)
4. Below, look at **"SMTP Settings"**. By default, Supabase uses its built-in mail service which is **rate-limited to ~3 emails/hour**. That's fine for testing. For a real launch, swap in AWS SES or another SMTP provider — Supabase docs walk through it.

Save if you changed anything.

Step 2 — Set the redirect URL

When a user clicks the magic link in their email, Supabase needs to know where to bounce them after authenticating.

1. Authentication → **"URL Configuration"**.

2. **“Site URL”** — set to your live URL (e.g. `https://your-project.pages.dev` or your custom domain from chapter 05).
3. **“Redirect URLs”** — add the same URL plus `/*` to be safe: `https://your-project.pages.dev/*`.
4. Save.

If you skip this step, magic links will work but redirect users back to `localhost:3000` (Supabase’s dev default) — broken in prod.

Step 3 — The starter’s sign-in form

Your starter’s `index.html` already has a sign-in section. Find it — search for `id="signin-form"` or `<form` near the bottom of `index.html`. It looks roughly like:

```
<form id="signin-form">
  <label for="email">Email</label>
  <input type="email" id="email" name="email" required />
  <button type="submit">Send me a magic link</button>
</form>
<p id="signin-status" hidden></p>
```

The corresponding JavaScript lives in `js/site.js`. Search for `signin-form` there. The handler looks like:

```
import { createClient } from "https://esm.sh/@supabase/supabase-
        js@2";

const supabase = createClient(window.SUPABASE_URL,
        window.SUPABASE_ANON_KEY);

document.getElementById("signin-form").addEventListener("submit",
        async (e) => {
  e.preventDefault();
  const email = e.target.email.value;
  const status = document.getElementById("signin-status");

  const { error } = await supabase.auth.signInWithOtp({
    email,
    options: {
      emailRedirectTo: window.location.origin + "/",
    },
  });

  status.hidden = false;
  status.textContent = error
    ? `Couldn't send link: ${error.message}`
    : "Check your inbox for a sign-in link.";
});
```

That’s the whole magic-link wiring. ~15 lines of code.

If your starter is missing this form (different version), tell your agent: “Add a magic-link sign-in form to index.html plus the JS handler in js/site.js, using `supabase.auth.signInWithOtp` and reading `window.SUPABASE_URL` / `window.SUPABASE_ANON_KEY` from the page.” It will produce the equivalent.

Step 4 — Test it end-to-end

1. Open your live site in an incognito/private window (so you don’t re-use any cached state).
2. Type a real email address you control.
3. Click **“Send me a magic link”**.
4. The status text changes to “Check your inbox…”
5. Open your email. You should see a Supabase email from `noreply@mail.app.supabase.io` (until you swap the SMTP) titled something like “Confirm your sign-up” with a button “Confirm your mail”.
6. Click the link. It bounces you back to your site, signed in.

How do you *know* you’re signed in? The starter has a small `<div id="auth-status">` that shows your email after sign-in. Or open browser dev tools → Console → run:

```
const {
  data: { user },
} = await supabase.auth.getUser();
console.log(user);
```

If it prints your email and a UUID, you’re authenticated.

Step 5 — See yourself in the database

Supabase auto-creates an `auth.users` row on sign-up.

1. Supabase dashboard → **“Table Editor”** → in the schema dropdown (top left of the table list), switch from `public` to **`auth`**.
2. Click `users`. You should see a row with your email, your `id` (UUID), and a `confirmed_at` timestamp.

Then switch back to `public` and click `members`. **There’s no row for you yet.** That’s normal — the starter’s `auth.users` → `public.members` trigger only fires when an auth row is created *while* the trigger exists. If you signed up *before* running migrations, you’ll need to manually insert your `members` row. Easiest: in the SQL editor:

```
insert into public.members (id, email)
select id, email from auth.users
where email = 'your-email@example.com'
on conflict do nothing;
```

Run. Refresh the `members` table. You’re in.

What you've built

Let's count the pieces:

- **Frontend:** static HTML on Cloudflare Pages CDN, free tier.
- **Backend:** Supabase Postgres, free tier, with row-level security.
- **Auth:** magic-link via Supabase, no passwords, no JWT plumbing.
- **Deploy pipeline:** every git push auto-deploys.

That's a real production-shape stack. The same shape that powers MAH's own members area at `melbourneaihub.com.au/members.html`. You built it.

What's next

- **Chapter 08 — Request Tier 2 access.** When you're ready to contribute to MAH itself (write access to the canonical repo), submit your live URL + repo URL via the members-area form. A founder reviews and adds you as a collaborator on GitHub. (Eventually the MAH AI Console will assess this automatically — for now it's a manual review.)
- **Chapter 09 — Optional advanced paths.** Your own VPS, your own Telegram bot, contribute to MAH's actual codebase, payments, newsletter pipeline.

Troubleshooting

- **No email arrives.** Supabase's built-in mailer is rate-limited. Wait 20 min and retry, or set up SES/SMTP. Also: check spam folder.
- **"Email not authorized"** — your Site URL in Auth settings doesn't match where the click came from. Re-check Step 2.
- **The redirect from email lands on `localhost:3000`** — Site URL not set. Same fix as above.
- **Magic link works but I can't read the `members` table.** RLS is protecting you (working as designed — the starter's policy is "users can only see their own row"). Either (a) you don't have a members row yet (run the insert from Step 5), or (b) the SELECT policy doesn't match your auth method. Ask your agent: "explain why `select * from members` returns 0 rows when I'm signed in." It will walk you through the policy.

Add a Feature — A Working Contact Form

Up to chapter 07 you built a stack that exists. This chapter makes it **do something useful** — you'll add an "Expression of Interest" (EOI) form to your homepage, where any visitor can leave their email and a short note. The submissions land in your Supabase database where only you (or anyone you grant access) can read them.

This is the smallest possible "real feature." After this you'll understand the loop you'll repeat for everything else: front-end form → Supabase Insert → row in your database → you read it later.

Time: 15 minutes. **Cost:** $0. **Useful immediately.**

What you'll have at the end

- A form on your homepage with two fields: email + a "tell me what you're working on" note.
- Submissions saved to the `eoi_submissions` table in your Supabase project (the table is already there from chapter 03's migration — you set it up without realising).
- A "Thanks, we'll be in touch" confirmation when the visitor submits.
- Visibility into submissions via the Supabase Table Editor (or, later, via a private members-only page on your site).

Step 1 — Confirm the table exists

The `0002-eoi.sql` migration you ran in chapter 03 created `public.eoi_submissions`. Quick sanity check:

1. Supabase dashboard → **"Table Editor"**.
2. Click `eoi_submissions`. You should see four columns: `id`, `email`, `note`, `source`, `created_at`.
3. If the table isn't there — re-run `0002-eoi.sql` from your repo's `supabase/migrations/` folder via the SQL editor.

Step 2 — Confirm the RLS policy lets visitors insert

We want anyone (signed in or not) to be able to submit an EOI. We do **not** want them to be able to read other people's submissions. The migration in chapter 03 set this up; let's verify.

1. Supabase dashboard → **"Authentication"** → **"Policies"** → `eoi_submissions`.
2. You should see three policies:
   - `eoi_anon_insert` — INSERT, role `anon, authenticated`, with check `true`. (Anyone can insert.)
   - `eoi_tier2_read` — SELECT, role `authenticated`, with policy scoped to `tier2/tier3` members. (Only verified members can read.)
   - `eoi_service_role` — ALL, role `service_role`. (Service-role bypasses; used by Edge Functions.)
3. If they're missing — re-run `0002-eoi.sql`.

This RLS shape is what makes the public form safe: a malicious visitor can't read or modify the existing submissions, only insert their own.

Step 3 — Add the form to your homepage

Open `index.html` (in the GitHub web UI or via Claude Code). Add this block somewhere that makes sense — probably below the existing sign-in section:

```
<section id="contact" class="content">
  <h2>Let us know what you're working on</h2>
```

```
  <p>
    Drop your email and a one-line note. We read everything; we
        won't spam you.
  </p>
  <form id="eoi-form" class="signin-form">
    <label for="eoi-email" class="visually-hidden">Email</label>
    <input
      type="email"
      id="eoi-email"
      name="email"
      placeholder="you@example.com"
      required
    />
    <input
      type="text"
      id="eoi-note"
      name="note"
      placeholder="What are you building or curious about?"
    />
    <button type="submit">Send</button>
  </form>
  <p id="eoi-status" class="signin-status" hidden></p>
</section>
```

Save and commit.

Step 4 — Wire the form to Supabase

Now open `js/site.js` and add the handler. Find the bottom of the file (after the magic-link handler) and add:

```
// EOI form — anyone can submit; rows land in eoi_submissions
        table.
const eoiForm = document.getElementById("eoi-form");
const eoiStatus = document.getElementById("eoi-status");
if (eoiForm) {
  eoiForm.addEventListener("submit", async (e) => {
    e.preventDefault();
    const email = e.target.email.value.trim();
    const note = e.target.note.value.trim();

    eoiStatus.hidden = false;
    eoiStatus.textContent = "Sending…";

    const { error } = await
        supabase.from("eoi_submissions").insert({
      email,
      note,
      source: window.location.pathname,
    });

    if (error) {
```

```
      eoiStatus.textContent = `Couldn't send: ${error.message}`;
    } else {
      eoiForm.reset();
      eoiStatus.textContent = "Thanks — we'll be in touch.";
    }
  });
}
```

That's it. **15 lines of JavaScript** to insert a row into a real Postgres database with row-level security enforced.

Commit and push. Wait 60 seconds for Cloudflare Pages to redeploy.

Step 5 — Test it

1. Open your live site in an incognito/private window (so you're anonymous, not signed in).
2. Scroll to the contact section.
3. Type a test email and note. Submit.
4. The status text should read "Thanks — we'll be in touch."
5. Open Supabase dashboard → Table Editor → `eoi_submissions`. **Your row is there.**

If something fails, the status will tell you what — usually:

- **"Could not find table"** — the migration didn't run; redo step 1.
- **"new row violates row-level security policy"** — the `eoi_anon_insert` policy is missing or wrong; redo step 2.
- **"Failed to fetch"** — `SUPABASE_URL` env var is wrong in Cloudflare; redo chapter 04 step 4.

Step 6 — How you read the submissions

Three options, in increasing sophistication:

A. Table Editor (today, 0 effort)

Supabase dashboard → Table Editor → `eoi_submissions`. Sort by `created_at desc`. Refresh whenever you want to see new ones.

B. Daily email digest (~30 min, when you want it)

Add a Supabase Edge Function (`supabase/functions/digest-eoi/index.ts`) that runs once a day via Supabase's scheduled trigger, queries the last 24 hours of submissions, and emails them to your address via SES (or your provider). The starter doesn't include this; chapter 09 of book-04 in MAH's archive (`book-04-integrations/02-aws-ses.md`) walks through the SES SigV4 pattern when you're ready.

C. A private "submissions" page in your site (~45 min)

Once you've upgraded yourself to `tier2` in your `members` table (just update the row in the Supabase Table Editor), the `eoi_tier2_read` policy lets you SELECT. Add a `/submissions.html` page to your site, gated by sign-in, that queries `eoi_submissions` and displays them.

Most founders ship A, then upgrade to B when submissions get frequent enough that they stop checking the dashboard. C is for when you have co-founders who also need access.

What you've actually built

You added a real feature in 15 minutes. The pattern is the same for every feature you'll ever add:

1. **Schema** — does a table exist for this data? If not, write a migration.
2. **Policy** — who can insert / read / update / delete? RLS policy.
3. **Form / UI** — HTML + a small handler that calls `supabase.from('table').insert(...)` or `.select(...)`.
4. **Status** — surface success / error to the user.

That's the whole loop. Subscribe-to-newsletter, contact-form, update-your-profile, like-a-post — all the same shape with different nouns.

Things you could try next, in order of "smallest interesting"

Try this	What you learn
Make the form's "Send" button disabled while submitting	Tiny UX improvement; teaches you DOM state
Add a hidden `honeypot` field that bots fill but humans don't	Spam protection without CAPTCHAs
Add `created_at` formatting so the timestamp looks nice on screen	Date handling
Add a count: "12 founders have already shared what they're building"	A `select count` query
Add an admin-only `/submissions.html` page (option C above)	Auth-gated routes; the bridge to a real members area
Replace the form fields with whatever your project actually needs	Now it's your product

What's next

- **Chapter 09 — Request Tier 2 access.** Now that you've shipped a real feature, the request to get added to MAH itself is more than pro-forma — you have something to point at.
- **Chapter 10 — Optional advanced paths.** AI features, payments, newsletter pipeline, your own VPS.

Troubleshooting

- **The form submits but nothing appears in the table.** Open the browser console — you're probably seeing a 401 or 403, which means the anon-insert policy isn't applied. Re-run `0002-eoi.sql`.
- **Form submits but I can never see the submissions.** RLS is doing its job — anonymous reads are blocked. Either use the Table Editor (which uses the service-role key) or upgrade your `members` row to `tier2` and use option C above.
- **A rude person submitted spam.** Inevitable. Add the honeypot field from the table above. If still bad, add a Cloudflare Turnstile widget (free, ~10 min of work, doc-linked from chapter 10).

Request Tier 2 — MAH Collaborator Access

If you've made it through chapters 02–08, you've shipped a real full-stack site on infrastructure you own. That's **Tier 1**.

This chapter is the path from Tier 1 to **Tier 2** — being added as a collaborator on the canonical Melbourne AI Hub repository so you can contribute changes directly to `melbourneaihub.com.au` itself.

Time: 5 minutes. **Cost:** $0.

Why there's a gate at all

We don't gate Tier 2 to be exclusive. We gate it because:

1. **MAH's codebase is shared infrastructure** — many founders, real members, real payments. A confused commit at the wrong moment can take the live site down or leak member data.
2. **Tier 1 builds the muscle memory you need to contribute safely** — if you've shipped your own stack, you understand "edit → push → deploy" as a real loop, not an abstraction.
3. **The gate is cheap to clear** — once you've finished chapter 07, the request itself is a 5-minute form.

The gate is one of intent, not skill. Anyone willing to stand up their own free-tier stack and ship a real change has earned access.

What "Tier 2" gets you

- Write access (Maintain role) on the GitHub repo `teddyscleaningserviceaus-design/melbourne-ai-hub`.
- Permission to open pull requests against `main`. Other founders review and merge.
- Write access on the dev mirror `Melbourne-ai-hub/Melbourne-ai-hub-dev` if you want to push small edits via the Hermes-on-Telegram path.
- Access to MAH's Supabase project as a read-only dashboard user (Teddy invites). You can *see* the live data; you cannot write to it from the dashboard until Tier 3.

- An invite to the founders' channel of the MAH Telegram bot for internal coordination.

What you submit

A short structured note. Two URLs and one sentence. We'll add a real form in the members area soon; for now, send via Telegram or email.

```
TIER 2 REQUEST

Live URL:   https://<your-project>.pages.dev   (or your custom
domain)
Repo URL:   https://github.com/<your-username>/<your-repo>
Latest commit SHA you'd like reviewed: <short-sha, e.g. 8a70469>
What I changed (one sentence):
  e.g. "Updated the hero subtitle, wired magic-link login,
        and added a note about my horticulture research focus."

(Optional) Anything specific you want to focus on at MAH:
  e.g. "Frost / horticulture domain — interested in
BloomShieldAI."
```

Where to send it

Right now (manual review):

- DM Teddy on Telegram with the structured note above. He'll review on his next pass through messages (usually same day).
- Or post in the MAH Telegram group with `@teddybear tier 2 request` and the same note inline.

Coming soon (AI-assessed):

The MAH members area is gaining an AI Console (it's currently being configured with OpenRouter / Ollama Cloud as model backends). Once it's live, you'll be able to paste your URL + repo into the console; the AI will fetch them, set a unique challenge, verify you actually shipped the change, and flip your tier automatically. The form on this page is the **manual fallback** until that's wired.

We expect the AI assessor to ship within a few weeks of this handbook's launch. Until then, the manual review path is the only path. It's working fine for current founders.

What the reviewer checks

When a founder reviews your Tier 2 request, they look for:

Check	Why
Live URL responds with HTTPS 200	Confirms you actually completed chapter 04.

Check	Why
Repo is yours (commits authored by you)	Confirms you didn't just paste someone else's URL.
At least one substantive commit beyond the template's initial state	Confirms you completed chapter 06 (real edit, not just fork).
Magic-link form is wired and reaches Supabase	Confirms you completed chapter 07 (full-stack proof).

If something's missing, the reviewer replies with what to fix. There's no failure state — only "not yet, here's what to add." Most founders get added to MAH on first review.

After you're added

You'll get a GitHub email: "You've been invited as a collaborator on `teddyscleaningserviceaus-design/melbourne-ai-hub`." Accept.

Then go to `book-07-founder-onboarding/` (the existing 74-page handbook). That's the reference for **how MAH itself works** — its file layout, its deploy script, its do-not-touch surfaces. You can skim it; you don't need to memorise it. Come back to specific chapters when specific problems arise.

Your first contribution can be tiny. A typo fix. A copy edit. Use Hermes via the MAH Telegram group to make it (it'll handle the dev mirror + Cloudflare path); or open a PR against `main` directly if you prefer the GitHub web UI flow.

What's next

- **Chapter 09 — Optional advanced paths.** Once you have Tier 2, there are several directions to go: own VPS (Tier 1's optional appendix), own Telegram bot (the `mah-hermes-deploy` playbook), payments (Stripe wiring), newsletter (the AI content pipeline). Pick the one that matches your project.

Troubleshooting

- **My Tier 1 site went offline.** Cloudflare Pages free tier doesn't go offline unless you delete the project. If `*.pages.dev` is down, Cloudflare itself is having an outage — extremely rare. Check https://www.cloudflarestatus.com.
- **I can't sign into my Supabase project.** Different problem; see Supabase support or chapter 03's troubleshooting section.
- **Reviewer hasn't replied in >48h.** Bump the request. We're a small team; messages occasionally get buried. Once-per-business-day bumps are fine.

Next Up — Optional Advanced Paths

You've shipped your own full-stack site (chapters 02–07) and either requested Tier 2 access or chosen to keep operating sovereignly on your own infrastructure (chapter 08). This final chapter is a **menu**, not a path: pick whichever direction matches the project you actually want to build.

None of these are required. Many founders never go beyond what chapters 02–07 deliver, because that stack is already enough to ship a real product. These are doors, not stairs.

A. Bring up your own VPS (Tier 1.5 / "the server-admin path")

The starter you forked also includes the files needed to deploy on a real virtual server (VPS) instead of Cloudflare Pages. This is what MAH itself uses in production — nginx serving static files from a Lightsail or Hetzner box, certbot for HTTPS, `release.sh` to deploy via SSH and rsync.

Why bother, given Cloudflare Pages already works?

- You learn how a "real" server works — SSH, nginx, the systemd service model, certificate renewal.
- You can run things Cloudflare Pages can't host directly: a long-running Telegram bot, a custom backend daemon, a database in the same machine, etc.
- You stop being dependent on Cloudflare's free tier (which is generous, but is a single point of failure if their policies change).

What you'll need

- $5/month for a Lightsail Ubuntu 22.04 nano (or equivalent on Hetzner / Vultr / DigitalOcean — Hetzner CX22 at €4.51/month is the cheapest credible option).
- A domain (chapter 05 covers the cheap path).
- ~3 hours of patience the first time. Half an hour every subsequent time.

The walkthrough

The starter repo has `docs/tier-2-vps.md` with the step-by-step. It covers:

1. Provisioning the server.
2. The minimum `apt install` (nginx, certbot, rsync).
3. Setting up an SSH key.
4. Running `bootstrap.sh --with-vps` to generate the nginx config and `.env` for server-side use.
5. The `release.sh` deploy command.

You don't have to abandon Cloudflare Pages to do this — many founders keep both, with their VPS used for anything that needs a long-running process.

B. Add a Telegram bot to your stack

If you want an AI agent that can edit your stack via Telegram (the way Hermes edits MAH's dev mirror), the playbook lives at `mah-hermes-deploy/` in the MAH repo (Tier 2 access required to read, but the snapshot at `mah-hermes-deploy/snapshots/2026-04-28/` is public).

This is significant work — you're standing up an LLM gateway on your own server, configuring allowed Telegram users, hardening the OS against the agent escaping its sandbox. **Do not attempt this on day one.** Months in, when you have a sense of what your bot would actually do for you, come back.

A simpler version: just use Claude Code or Cursor on your laptop when you want to make changes. You're effectively "your bot," with full context and zero deployment overhead. That's how 90% of MAH's edits happen even today.

C. Contribute to MAH itself

If you got Tier 2 access in chapter 08, the path forward is the existing 74-page handbook at `book-07-founder-onboarding/`.

That book covers:

- The MAH-specific deploy flow (`release.sh`).
- The do-not-touch surfaces (Stripe webhook signatures, magic-link generation, RLS policies on member data).
- The shared dev mirror at `dev.melbourneaihub.com.au`.
- How Hermes-on-Telegram makes small edits without you touching the terminal.

Pick a real first contribution that scratches your own itch. A copy fix. A typo. A new image on the homepage. A small new content section about your domain expertise. Don't try to refactor the architecture on your first PR.

D. Add payments

If your project will charge for something (workshops, memberships, products), Stripe is the path. MAH uses Stripe Checkout (the hosted-page flow) plus a webhook to `register-workshop` / `stripe-webhook` Edge Functions. The full integration is documented in `book-04-integrations/01-stripe.md` (Tier 2 access required to read).

Two warnings:

1. **Don't put live Stripe keys in your starter until you actually have something to sell.** Test-mode keys are free and behave identically; switch to live the day you're ready.
2. **The Stripe webhook signature check is sacred.** It's the only line of defence against forged payment events. The starter doesn't include this skeleton (yet) — when you add Stripe, copy the pattern from MAH's `stripe-webhook` EF rather than rolling your own.

E. Add an AI feature to your site

You signed up for OpenRouter in chapter 01 (or you didn't and want to now). One API key, every model. The pattern:

1. Create a Supabase Edge Function in your starter (`supabase/functions/ai-chat/`).
2. Read `OPENROUTER_API_KEY` from Edge Function secrets (set via Supabase dashboard → Project Settings → Edge Functions → Secrets).
3. Forward a chat request to OpenRouter; stream the response back to the browser.
4. Wire a chat UI on your homepage to call your EF, not OpenRouter directly. **Never put your OpenRouter key in browser code.**

MAH's own AI Console (`js/members-ai-console.js`) is roughly this shape. The pattern is the same regardless of which AI you eventually use.

F. Add a newsletter

MAH's weekly newsletter is generated by an AI pipeline (`mah newsletter` CLI + `draft-weekly-newsletter` Edge Function) that scrapes ~9 sources, synthesises with an LLM, runs a Drive doc mirror, and broadcasts via AWS SES. Documented in `book-05-content-pipeline/`.

For a starter version, three components:

1. A simple `subscribe-newsletter` Edge Function (the starter ships this skeleton).
2. AWS SES set up with your domain verified (or use Resend for simplicity — similar shape, easier setup).
3. A scheduled Edge Function (Supabase cron) that runs weekly and sends to whoever's subscribed.

G. Stop here

This is a real, valid choice. Many founders need only what chapters 02–07 deliver — a small site, a database, sign-in, the ability to edit and ship. You don't have to keep adding complexity for its own sake. Knowing when you have enough is half the discipline.

Closing note

The whole point of this handbook was to lower the barrier to "I have my own infra and can ship to it." If you've finished chapters 02–07, you've cleared that bar. Everything in this chapter is leverage on top of it.

Build the thing you actually want to build. Come back to this menu when a specific need arises, not before.

Welcome to the club.