The Creator Platform Playbook

How to build the one place your audience lives: a fast content site that sells your products and hosts your members — all under your own domain.

Most creators build on rented land — a Substack here, a Gumroad there, a community on Skool or Discord. It works until it doesn't, and when the platform changes the rules, you find out that none of it was really yours. This is the stack for building a home base you actually own: content, products, and membership all on one domain. A live creator site runs exactly this way.

01 **Astro on Vercel** — fast static pages + server routes when you need them

↓

02 **MDX content collections** — write in Markdown, get typed schema-validated pages

↓

03 **Resend newsletter** — the audience you actually own

↓

04 **Stripe Checkout** — take money without a store platform

↓

05 **Magic-link login** — email a signed link — no passwords

↓

06 **Signed session cookie** — Stripe is the source of truth for membership status

↓

07 **Members area** — gate routes in middleware; members get the tools

↓

08 **Admin area** — manage flags and content without a deploy

---

The Build Sequence

1. The idea — own the destination

The goal is one URL that does everything: publishes content for the public, sells products for buyers, and gives members access to the tools they paid for. When you own the stack, you control the rules. No platform

fees on membership, no algorithm deciding who sees your newsletter, no risk of a Terms of Service change wiping out your community.

The tradeoff is that you build and maintain it. This playbook makes that concrete.

2. Framework + hosting — Astro on Vercel

Astro is a website framework built around the idea that most pages don't need JavaScript on every render. Content pages ship as static HTML — fast to load, easy to cache. When a page does need server logic (checking a cookie, hitting a database), you enable server-side rendering for that route only. The result is a site that's fast by default and capable when it needs to be.

Vercel handles deploys: push to GitHub, it builds and ships. Preview URLs for every pull request. Edge middleware for auth checks. No servers to manage.

3. Content — MDX content collections

MDX is Markdown that can import components. Astro's content collections let you point a loader at a folder of `.mdx` files and get back a typed, schema-validated collection you can query with `getCollection()`. Each piece of content has frontmatter (title, date, status, tags) enforced by a Zod schema — so a missing field fails the build rather than producing a broken page at runtime.

The live site uses this pattern for newsletters (letters) and this playbook series. Write a file, push, it appears. No CMS database, no rich-text editor to wrangle.

Why this over a headless CMS: for a solo creator, a folder of Markdown files in a git repo is faster to write in, easier to back up, and trivial to deploy. You can always add a CMS layer later if you need one.

4. Newsletter — Resend

Email is the one channel where the platform can't suppress your reach. Resend handles both transactional email (magic-link sign-in, purchase receipts, download links) and broadcast newsletters from one API and one dashboard. When you publish a new letter, a script calls the Resend API to send it to your list.

Why Resend over Mailchimp or ConvertKit: it's developer-native, the API is clean, and the free tier handles a real list. You're already integrating it for transactional email anyway — keeping it for newsletters means one tool instead of two.

5. Products + checkout — Stripe Checkout

Each product (a course, a download, a membership) has one Stripe Checkout session endpoint. A visitor clicks "buy," your server creates a session, Stripe handles the payment form and confirmation, then fires a webhook back to your server when the payment succeeds. Your server handles the webhook: it marks the user as a member, sends a confirmation email, and unlocks access.

Why Stripe Checkout over embedding a payment form: you don't handle raw card data, which keeps you out of PCI scope. And Stripe's hosted checkout converts well — it already knows how to handle card errors,

3D Secure, Apple Pay.

6. Membership — magic-link login + signed session

No passwords. When a member wants to sign in, they enter their email, your server emails them a one-time link signed with a secret key (HMAC). They click it, the server verifies the signature and the expiry, and sets a signed session cookie. Done.

The key design decision: Stripe is the source of truth for membership status, not a separate users table. When a session is verified, your middleware checks Stripe to confirm the email has an active subscription. No sync issues, no "why can't I log in?" support tickets from users whose access was manually toggled somewhere.

Why magic links: fewer passwords in the world is a good thing. Members don't forget their credentials, and you don't store password hashes.

7. Stored data — Supabase (Postgres)

Supabase provides the Postgres database. Subscriber records (email, newsletter opt-in status), feature flags, and admin data all live here. Supabase's free tier handles real data volumes, and its JavaScript SDK integrates cleanly with Astro's server routes.

The key distinction: Supabase holds the platform's operational data, while Stripe remains the source of truth for membership status. A member's record in Supabase says they exist; Stripe says whether their subscription is active. The middleware checks both.

8. The members area — gate routes in middleware

Astro middleware runs before a page renders. You check the session cookie, verify it, confirm Stripe membership status, and either serve the page or redirect to the sign-in flow. Members who are current see the members area and the tools it includes. Everyone else gets redirected.

The members area is where you deliver the core value: included tools, community access, archived content. The gate is the cookie check — everything behind it is a regular Astro page.

9. A little admin — password-gated admin area

A simple admin area behind HTTP Basic Auth (handled in middleware) lets you manage feature flags, check subscriber counts, trigger a newsletter send, or update content — without a deploy. It's not a CMS; it's a utility belt for common tasks that would otherwise require a code change.

Why not build a full CMS: a handful of admin pages covering the real repetitive tasks is faster to build and easier to maintain than a general-purpose CMS. Build what you actually need.

What it costs to run

Near-free to start: Vercel Hobby is free, Supabase's free tier handles real data volumes, Resend's free tier covers a few thousand emails per month, Stripe has no monthly fee (2.9% + 30¢ per transaction). The whole stack can run at $0/month while you're building your first 100 subscribers and first few hundred dollars in revenue.

The first upgrades as it grows:

- **Vercel Pro (~$20/mo)** when you hit Hobby compute limits, need more team members on the project, or want more build minutes.
- **Supabase Pro (~$25/mo)** past the free tier's row or storage cap, or when automated daily backups become non-negotiable.
- **Resend paid (~$20/mo)** once your list size or monthly send volume passes the free tier limit.

Payments scale with revenue automatically — Stripe's percentage fee means you pay more only when you earn more, which is the right shape for a solo creator.

If you want to build this alongside people doing the same thing, the Workshop is where that happens.

KEEP GOING

Pick one step. Do it today. Ship the thing — then build the next one.

MORE PLAYBOOKS — FREE

The Tools I Use	paulguardado.com/playbooks/the-tools-i-use
The Niche Content Site Playbook	paulguardado.com/playbooks/niche-content-site-playbook
The AI Tool (Micro-SaaS) Playbook	paulguardado.com/playbooks/ai-tool-playbook