AI-Powered Development

FROM IDEA TO PRODUCTION

The complete all-in-one guide to build your own website, app, or startup from scratch. Master AI agents and tools, integrate payments, deploy your server, and learn marketing tricks to grow organically.

apifreellm.com

Version 1.0 — February 2026

Table of Contents

1. Introduction

> *You are reading this right now because somewhere, somehow, a combination of marketing strategies, SEO techniques, and smart positioning brought this course to your screen. That alone should tell you something: the tactics in this guide actually work. And yes, you will learn every single one of them.*

The World Has Changed

Software development is no longer what it used to be. A few years ago, building a web application required months of work, a team of developers, and a significant budget. Today, a single person with the right tools and the right knowledge can build and launch a production-ready application in days, not months.

This course was written by professionals who work in the industry and use AI agentic tools daily to build real products. We don't teach theory from a textbook. We teach what actually works in the real world, right now.

Why This Course Exists

AI and LLMs are now better and faster executors than humans. They can write code, debug it, refactor it, and deploy it at a speed no human can match. But there's something they fundamentally lack: **ideas**.

> AI models are extraordinary executors, but they are not innovators. They don't see a gap in the market and think "I could build something to fix that." That's your job. Your role is to be the architect of ideas. The AI is your builder.

When you give poor instructions to an LLM, it will still produce something. It won't stop to explain what would actually be better. The quality of what you build is directly proportional to the quality of your knowledge. This course bridges that gap.

This is not just a development course. This is a complete blueprint for going from an idea to a live, revenue-generating product. Including extremely effective marketing and SEO tactics — the same techniques that brought you to this very page.

2. The AI-First Development Stack

> *The tools you choose will define how fast you move. In this chapter, we break down the AI coding agents available today, help you pick the right one, and teach you the prompting strategies that separate amateurs from professionals.*

> **Note:** This guide is written using Windows, but all the tools and steps work identically on macOS and Linux. Google Antigravity, Claude Code, and every other application discussed in this course are fully cross-platform. If you're on a Mac or Linux machine, simply follow the same steps — the interfaces and workflows are the same.

The AI Agent Landscape

The AI coding tools space has exploded. But not all tools are created equal. Some are glorified autocomplete engines. Others are full autonomous agents that can read your entire codebase, plan a strategy, execute changes across multiple files, run tests, and fix errors on their own. Understanding the difference is critical.

There are two fundamental categories of AI coding tools:

- **Inline assistants** — These sit inside your editor and suggest code as you type. Think of the original GitHub Copilot. They are reactive: they wait for you to write, then try to guess what comes next. Useful, but limited.
- **Agentic tools** — These are a completely different beast. You give them a task in natural language, and they autonomously plan, write, edit, debug, and iterate. They don't just suggest a line of code. They build features. This is where the real power is.

Here are the tools that matter right now:

Claude Code (by Anthropic)

Claude Code is, in our experience, the most powerful AI coding agent available today. It's a terminal-based agent that operates directly in your project directory. You give it instructions in natural language, and it reads your files, writes code, runs commands, creates commits, and fixes errors autonomously. It has full access to your filesystem and shell, which makes it incredibly effective for real development work. You can install it via npm (`npm install -g @anthropic-ai/claude-code`) or use it as a VS Code extension, which we'll discuss shortly.

Google Antigravity

Antigravity is a development environment by Google that brings AI-powered chat and agent capabilities directly into your workflow. Think of it as an intelligent workspace where you can interact with AI agents through chat interfaces, and from each agent conversation you can open

an integrated VS Code editor. This is key: Antigravity gives you the conversational AI layer, while VS Code provides the code editing power. The combination is seamless — you discuss what to build with the agent, then jump straight into the code without switching windows.

Cursor

Cursor is a fork of VS Code with AI deeply integrated. It has both inline suggestions and an agentic "Composer" mode where you can describe changes across multiple files. The UI is familiar if you already use VS Code, and it supports multiple models (Claude, GPT, etc.). It's a solid tool, especially if you prefer a fully visual workflow. However, its agentic capabilities, while good, are not as autonomous as Claude Code. You'll often need to review and approve individual changes step by step.

> Our recommended setup: **Google Antigravity + Claude Code as a VS Code extension.** Use Antigravity's agent chats to plan and discuss your work, then open the integrated VS Code and let Claude Code handle the heavy execution. This gives you the best of both worlds: intelligent conversation for planning, and autonomous code execution for building.

Choosing & Configuring Your Agent

Installing and running an agentic tool is the easy part. Getting the most out of it requires understanding how it works, picking the right subscription, and setting it up correctly.

The Claude subscription: start with Pro

Claude Code requires an Anthropic account. We recommend starting with the **Claude Pro subscription**, which is the entry-level paid tier. It's affordable and gives you access to Claude Code with all its features. It's the perfect starting point to experiment, learn the workflow, and build your first simple project.

Be aware, however, that the Pro plan has **limited usage**. If you use Claude Code intensively, you'll hit the usage cap relatively quickly. For learning and small projects, it's more than enough. But if you already know you want to use Claude Code as your primary development tool and plan to work on it for hours every day, consider upgrading to one of the higher-tier plans (such as Max) from the start. The higher plans offer significantly more usage, which means fewer interruptions and a smoother workflow when building larger projects.

The model: Claude Opus 4.6

We currently recommend using **Claude Opus 4.6** as your primary development model. It is, right now, the best model for building websites and applications. It understands complex architectures, writes clean and production-ready code, handles multi-file changes with remarkable accuracy, and rarely needs corrections on the first attempt. When working with Claude Code, set Opus 4.6 as your default model. The difference in output quality compared to smaller models is immediately noticeable.

Setting up Antigravity

From this point forward, this guide proceeds using **Google Antigravity** as our primary development environment. Here's how to get everything ready.

Step 1: Create your project folder. Before opening Antigravity, create a folder on your computer where your first project will live. For example, on Windows you might create `C:\Projects\my-first-app`, or on Mac/Linux `~/Projects/my-first-app`. This is the folder that Antigravity will open as a workspace. It can be empty for now — we'll fill it with code later in the course.

Step 2: Install and launch Antigravity. Download Google Antigravity, install it, and open it. Sign in with your Google account when prompted.

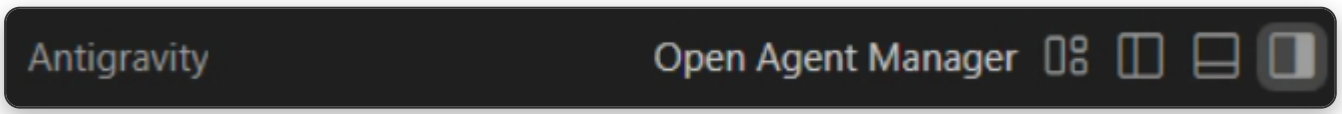
During the installation process, Antigravity will ask you to configure a few policies. For a fast, autonomous development workflow, we recommend selecting **custom configuration** and setting these options:

- **Terminal Execution Policy** → Always Proceed
- **Review Policy** → Always Proceed
- **JavaScript Execution** → Always Proceed

With these settings, Antigravity's agents will be able to execute commands, write files, and run code autonomously without asking for confirmation at every step. This is what enables the fast, fluid development experience we aim for in this course.

> **Security warning:** When you allow agents to proceed autonomously, both Antigravity and Claude Code can execute actions on your system without manual approval. This means you need to be mindful of what you expose them to. Do not point agents at codebases that may contain malware, and be cautious with links or resources from untrusted sources. In the era of AI, there are new attack vectors — malicious actors can craft content specifically designed to trick LLMs into executing harmful commands. Always keep an eye on what your agents are doing. We recommend the "Always Proceed" setup for speed, but stay vigilant and review the output regularly.

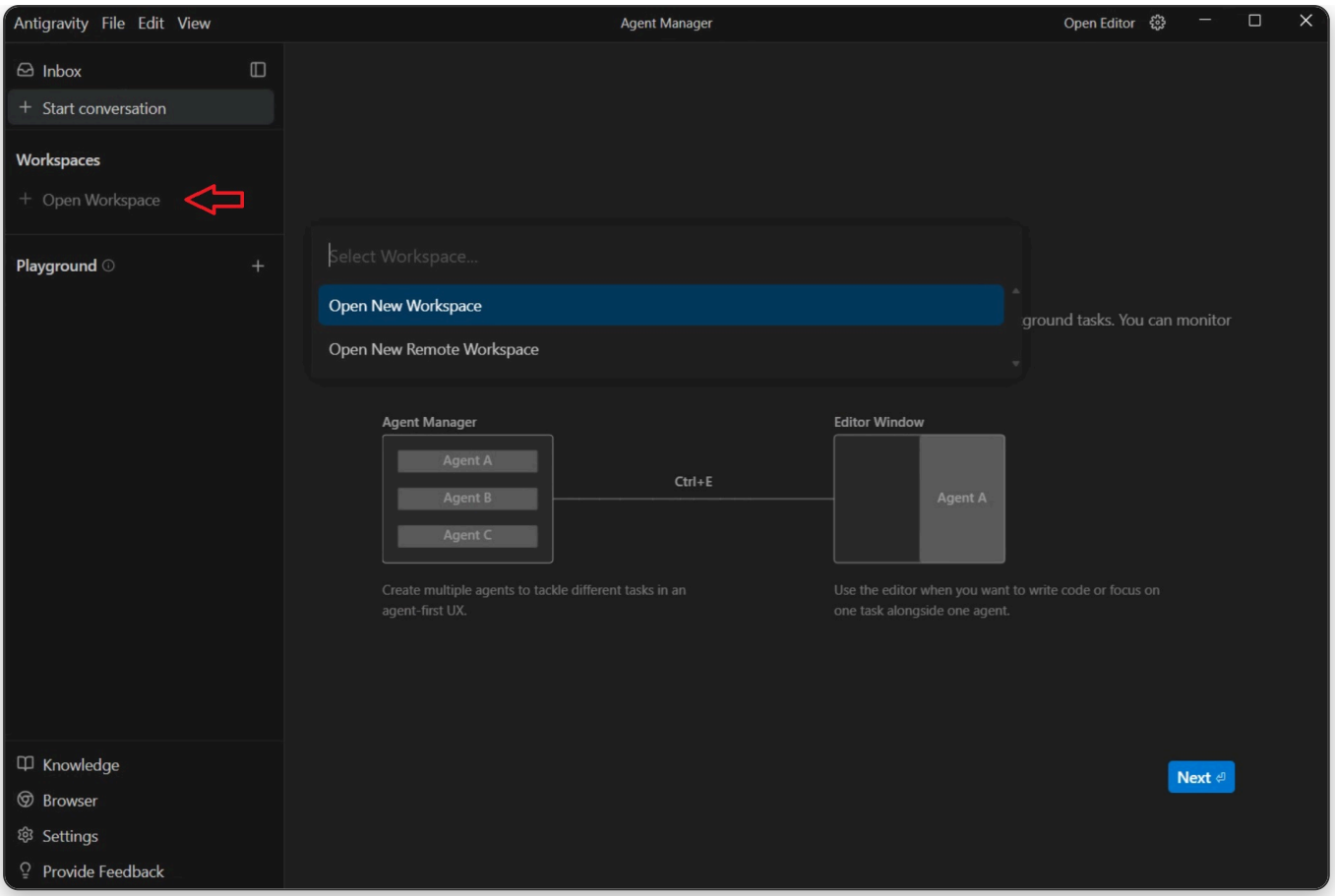
Step 3: Open the Agent Manager. Once Antigravity is running, click **"Open Agent Manager"** in the top bar of the window.



The "Open Agent Manager" button in Antigravity's top bar.

The **Agent Manager** is the central hub of Antigravity. This is where you manage your projects, start AI conversations, and open your development workspaces.

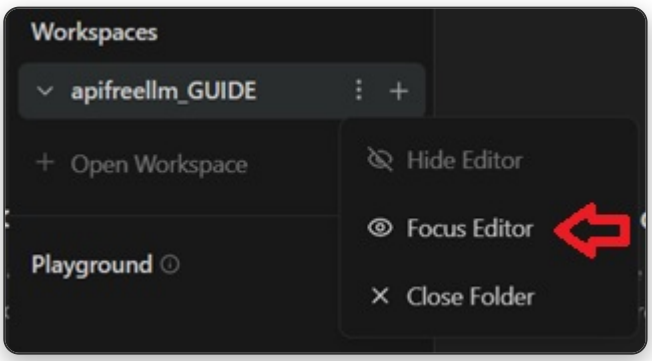
Step 4: Create your first workspace. In the Agent Manager, look at the left sidebar. Under the **"Workspaces"** section, click **"+ Open Workspace"**. A dropdown will appear — select **"Open New Workspace"**.



Click "+ Open Workspace" in the left sidebar, then select "Open New Workspace".

Antigravity will ask you to select a folder. Navigate to the project folder you created in Step 1 and select it. Your new workspace will appear in the left sidebar under "Workspaces", with the name of the folder you selected. Think of each workspace as an individual development session — one per project. You can create as many workspaces as you need, and switch between them from the Agent Manager at any time.

Step 5: Open the editor. Once your workspace is created, you'll see its name in the sidebar. Click the **three vertical dots** (⋮) next to the workspace name, then select **"Focus Editor"**. This opens the full VS Code environment for that workspace, where you'll write and edit your code.

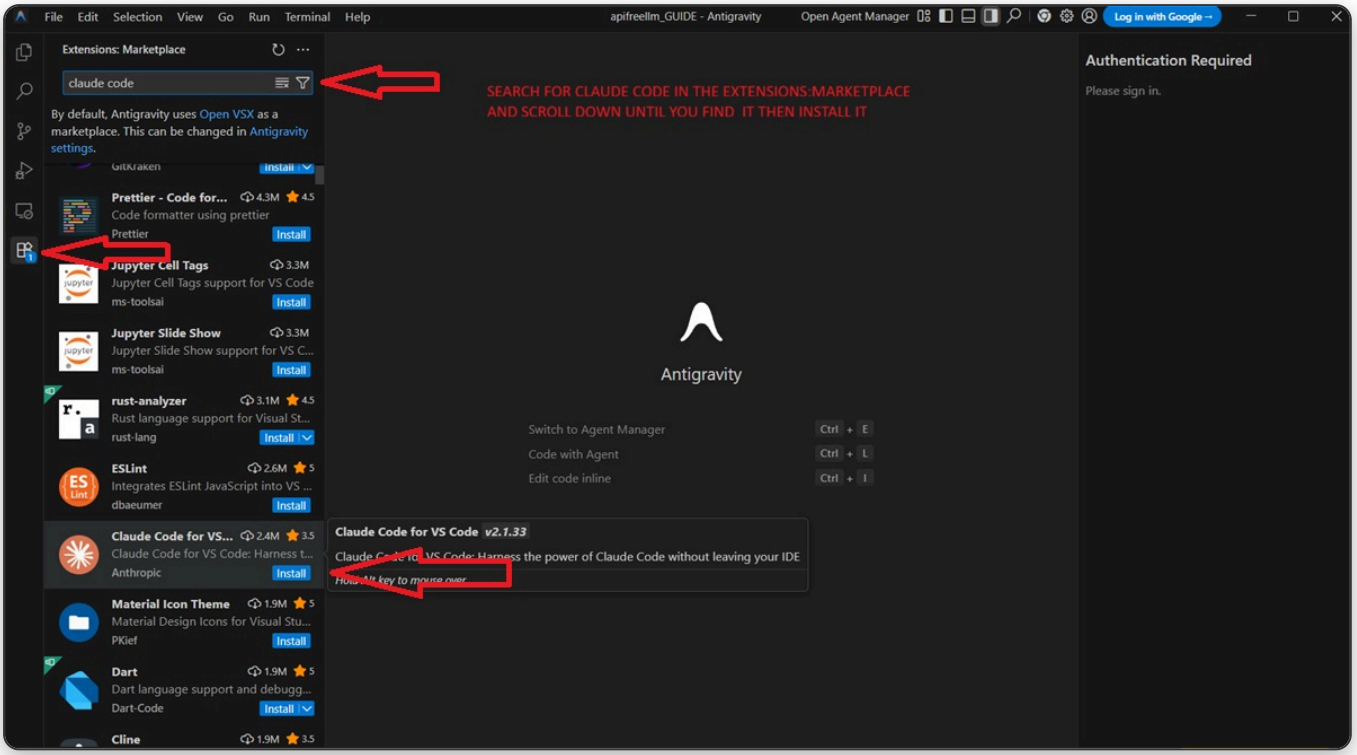


Click the three dots next to your workspace name and select "Focus Editor".

Installing Claude Code as a VS Code extension

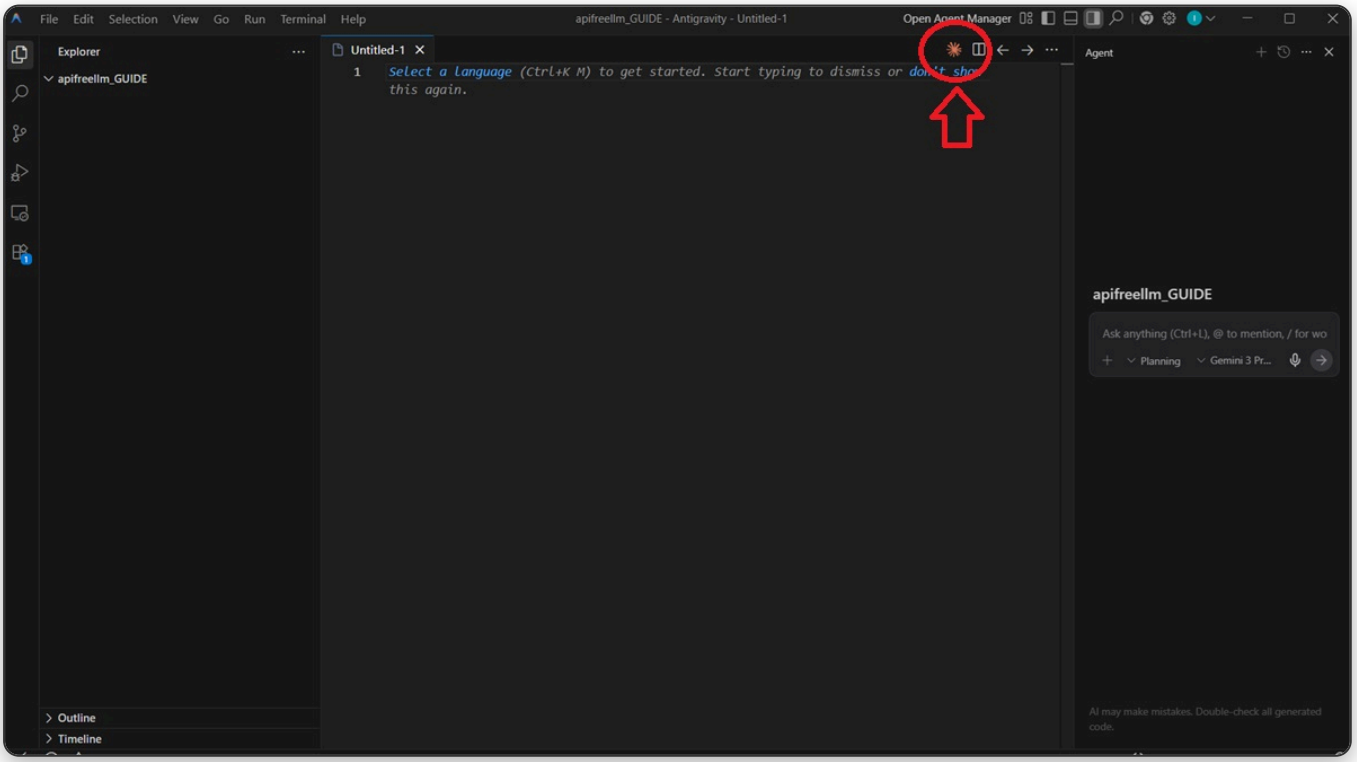
Now that you have the editor open, it's time to install Claude Code. Since the editor is based on VS Code, you have access to the extensions marketplace. Click the **Extensions icon** in the left sidebar

(or press `Ctrl+Shift+X`). In the search bar, type **"claude code"**. You may need to scroll down through the results — look for **"Claude Code for VS Code"** by Anthropic. Once you find it, click the **Install** button.



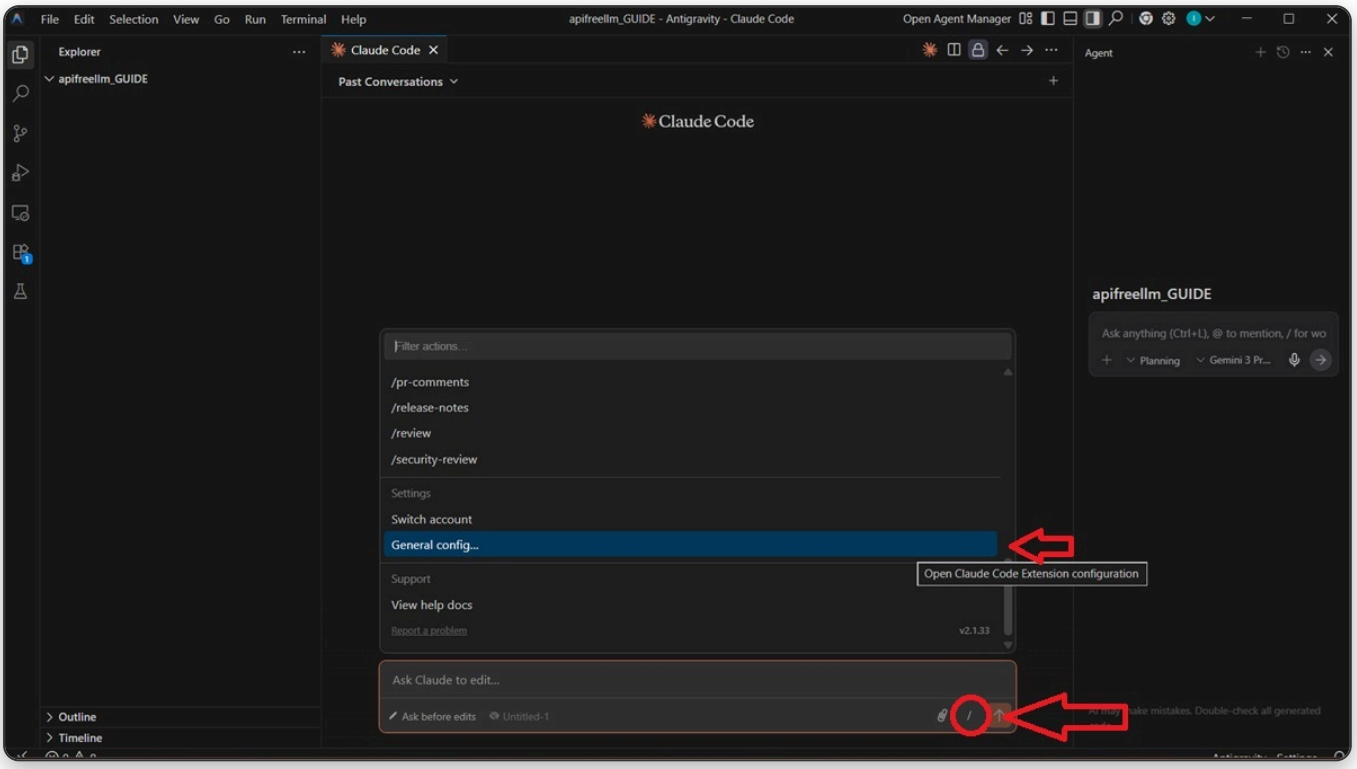
Search for "claude code" in the Extensions marketplace, scroll down to find it, and click Install.

Once installed, close the Extensions panel and open a new file (or any file in your project). You'll notice a small **Claude Code icon** appearing in the top-right area of the editor — it looks like a small orange symbol. Click it to open the Claude Code chat panel.



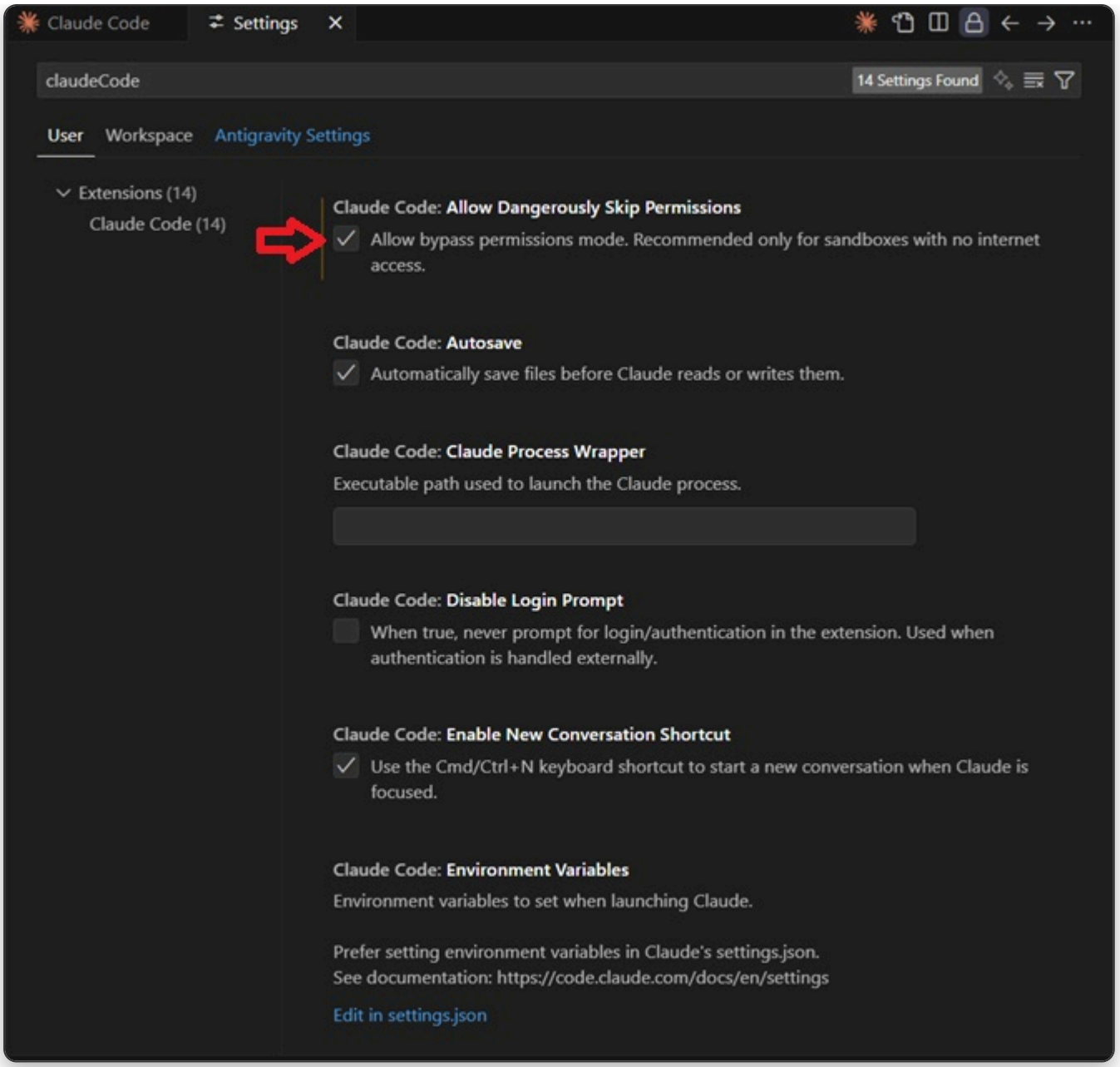
The Claude Code button (circled) appears in the top-right of the editor. Click it to open the Claude Code chat.

Claude Code will ask you to sign in with your Anthropic account (the one with your Pro subscription). After signing in, you need to configure it. In the Claude Code chat panel, click the **/** button at the bottom of the panel. A menu will appear with various commands. Scroll down to the **Settings** section and click **"General config..."** to open the Claude Code extension configuration.



Click the "/" button at the bottom, then scroll to Settings and select "General config..." to open the configuration.

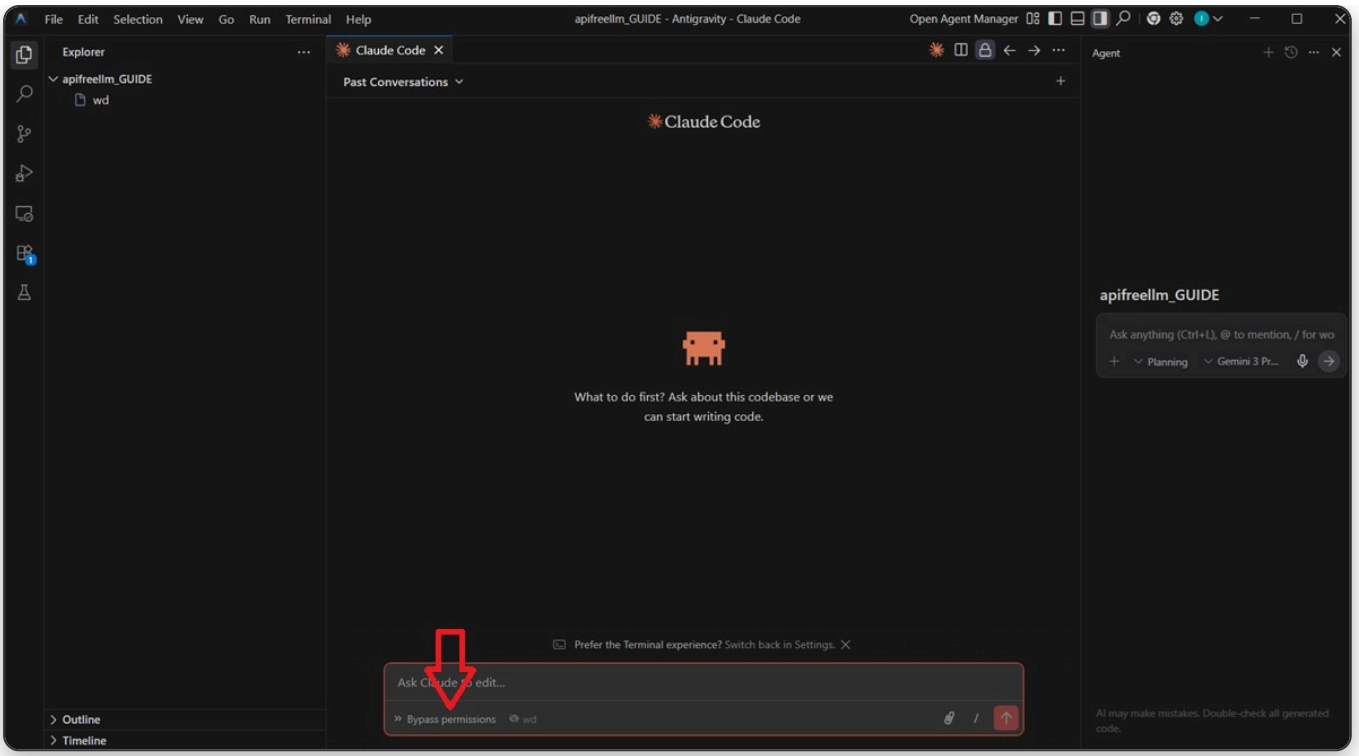
In the configuration panel that opens, look for the option called **"Allow Dangerously Skip Permissions"**. Enable this toggle. Once activated, you will be able to select **"Bypass permissions"** as your permission mode, which allows Claude Code to operate fully autonomously — reading files, writing code, running terminal commands, and making changes without asking for confirmation at each step.



Enable the "Allow Dangerously Skip Permissions" toggle in the Claude Code settings, then select "Bypass permissions".

> **Important:** With bypass permissions enabled, Claude Code will execute actions on your system without manual approval. This is essential for a fluid development workflow, but it comes with responsibility. Be cautious about the code you ask it to run, and never point it at untrusted repositories or suspicious URLs. AI agents can be exploited through prompt injection — malicious content hidden in files or websites that tricks the agent into executing harmful commands. Always review what Claude Code is doing, especially when working with external resources.

We also recommend enabling the setting that makes **"Bypass permissions" the default selection** for new chats, so you don't have to select it manually every time you start a new conversation. Now close Claude Code and reopen it (by clicking the Claude Code icon button again). From this point on, you'll see the **"Bypass permissions"** option available in the permission selector button at the bottom of the Claude Code chat panel. Click it to activate it.



Select "Bypass permissions" from the button indicated in the screenshot.

With Bypass permissions active, Claude Code will execute commands, create files, edit code, and run terminal operations completely on its own — without stopping to ask for your approval at every step. This is what allows you to **automate 100% of the development workflow**: you describe what you want to build, and Claude Code builds it autonomously from start to finish. No more clicking "Accept" on every single file change or command execution. You give the instructions, and the agent handles the rest.

Managing your usage smartly

One thing you should know from the start: every interaction with Claude Code consumes **tokens**, and your subscription has a usage cap. The good news is that you can monitor exactly how much you've used. In the Claude Code chat panel, click the **/** button — among the available commands you'll find your **current usage**, showing how many tokens you've consumed and how much capacity you have left.

Not all models consume tokens at the same rate. **Claude Opus 4.6** is the most intelligent and capable model, but it also consumes the most tokens per interaction. Smaller models like **Sonnet** or **Haiku** are less powerful but have higher usage caps and consume significantly fewer tokens. Our recommendation: use **Opus for complex tasks** that require deep reasoning, multi-file changes, or architectural decisions — this is where its intelligence makes a real difference. For simpler tasks like quick fixes, small edits, or straightforward questions, switch to a lighter model to preserve your Opus tokens for when they truly matter.

There's another strategy to save your Claude tokens: **use Antigravity's built-in agent** for questions that don't require Claude's level of intelligence. Antigravity supports multiple models, but we recommend using **Gemini** for these quick questions — since Antigravity is a Google product, Gemini has the highest usage cap and is also the fastest model available on the platform. Need a quick CSS reminder? Want to know the syntax for a Git command? Curious about how a library

works? Ask Antigravity instead of Claude Code. This way, you keep your Claude tokens for the heavy development work where they make the biggest impact.

As you can see in the earlier screenshot, we recommend keeping the **Claude Code chat on the left** and **Antigravity's agent chat on the right**. This side-by-side layout gives you instant access to both agents at all times.

> The smart workflow: **Opus for building, lighter models for quick tasks, Antigravity for general questions.** This way you maximize your Claude subscription value. If you're running low on your Claude usage cap, remember that you always have Antigravity's Gemini agent right next to you for simpler questions — use it to save your Claude tokens for the development tasks that truly need them.

Essential configuration

Regardless of how you install Claude Code, these configuration steps will dramatically improve your results:

- **Use a CLAUDE.md file** — Place a `CLAUDE.md` file in your project root. This file is automatically read by Claude Code at the start of every session. Use it to describe your project structure, coding conventions, tech stack, and any rules the agent should follow. Think of it as onboarding documentation for your AI developer.
- **Keep your project organized** — AI agents work dramatically better with clean, well-structured codebases. If your code is a mess, the agent will produce messy output. Good folder structure, clear naming conventions, and consistent patterns make a massive difference.
- **Use version control** — Always work with Git initialized. This gives you a safety net. If the agent makes a mistake, you can revert instantly. It also allows the agent to create commits for you, which is surprisingly useful for tracking what changed and why.

Essential Tools: Git & GitHub CLI

Before we start building anything, there are two tools that need to be installed on your system: **Git** and the **GitHub CLI (gh)**. These are fundamental for any modern development workflow, and they're what allow your AI agents to manage your code repositories autonomously.

Why Git and GitHub CLI matter

Git is the version control system that tracks every change in your project. It's your safety net: if Claude Code makes a mistake, you can revert instantly. It also allows the agent to create commits, manage branches, and keep a clean history of your project's evolution — all automatically.

GitHub CLI (gh) is a command-line tool that gives direct access to GitHub from the terminal. This is the key to full automation: once `gh` is installed and authenticated, Claude Code can create repositories, push code, manage pull requests, configure repository settings, set up GitHub Actions for deployment, add secrets, and much more — all from its terminal, without you having to open GitHub in a browser.

Installing Git and GitHub CLI

The simplest way to install these tools is to **ask Claude Code or Antigravity to do it for you**. Simply tell your agent: *"Install Git and the GitHub CLI on my system."* The agent will detect your operating system and run the appropriate installation commands. On Windows, it will typically use `winget` or download the installers; on macOS, it will use `brew`; on Linux, `apt` or your system's package manager.

If you prefer to install them manually, you can download Git from the official website and the GitHub CLI from its GitHub releases page. But letting the AI handle it is faster and avoids common installation mistakes.

Authenticating GitHub CLI

After installation, you need to log in so that `gh` can access your GitHub account. Again, you can simply ask your agent: *"Log me into GitHub CLI."* The agent will run `gh auth login` and guide you through the authentication flow, which typically involves opening a browser link and entering a code. Once authenticated, the agent has full access to your GitHub repositories.

> **This is a game-changer.** With `gh` authenticated, you can tell Claude Code things like: *"Create a new private repository called my-app, initialize the project, and push the code."* Or later: *"Set up a GitHub Action that deploys to my server on every push to main."* The agent handles everything — creating files, configuring secrets, setting up workflows — without you ever leaving the editor. This is what true development automation looks like.

3. Your First Build: Zero to Live

> *Your environment is ready. Claude Code is open, Antigravity is running, Git and GitHub CLI are configured. It's time to build something real. From this point forward, this course shifts from setup to strategy — practical techniques and insights that will give you a real advantage when building with AI agents.*

Never Start From Scratch

This is the single most important piece of advice in this entire course: **never build from scratch.**

Even though Claude Opus is an incredibly advanced and intelligent model, it will make mistakes. Every AI model does. It might misunderstand your project structure, use an outdated library, create an inconsistent file layout, or generate code that doesn't quite fit together when the project grows. Starting from an empty folder means the AI has to make hundreds of decisions with no reference point — and some of those decisions will inevitably be wrong.

Here's the reality: **99.9% of what you want to build already exists** in some form. Whether it's an e-commerce store, a SaaS dashboard, a portfolio site, a social media app, or a booking platform — someone has already built something similar. And many of these projects are open-source, available as templates on GitHub, ready to be cloned and customized.

Your workflow should always start the same way: **search for a template first.** Go to GitHub and look for open-source projects that match what you want to build. Find one that's close to your vision, clone it into your project folder, and then point Claude Code at it. Now instead of building from zero, the agent is modifying, improving, and customizing an existing, working codebase. The difference in quality and speed is enormous.

> **Templates are not cheating — they're strategy.** Professional developers use boilerplates and starter kits all the time. You're not copying someone's product. You're using a structural foundation and building your own unique product on top of it. The AI performs dramatically better when it has existing patterns to follow rather than inventing everything from nothing.

Choosing the Right Stack

If your template search comes up empty and you truly need to start a new project, the technology you choose can make a difference — especially when working with AI agents. That said, there's no single mandatory choice. The best stack is the one that gets you to a working product fastest.

Claude Code, like all LLMs, is fundamentally better at writing **web-based code**: HTML, CSS, JavaScript, and TypeScript. This is the language of the internet, and it's what these models have been trained on the most. The closer your project stays to web technologies, the better the AI will perform.

For **web applications**, **Next.js** is an excellent choice if you can find it. It's a modern React framework with built-in server-side rendering, which is critical for SEO. Claude Code works exceptionally well with Next.js projects: it understands the file-based routing, API routes, server components, and the entire ecosystem. You'll find an enormous number of Next.js templates on GitHub for virtually any type of application.

For **desktop applications** (executables for Windows, macOS, or Linux), **Electron** is a great option. Electron lets you build desktop apps using HTML, CSS, and JavaScript — the same web technologies Claude excels at. Since the UI is essentially a web page rendered inside a native window, the AI can build beautiful, functional desktop applications with the same ease as building a website.

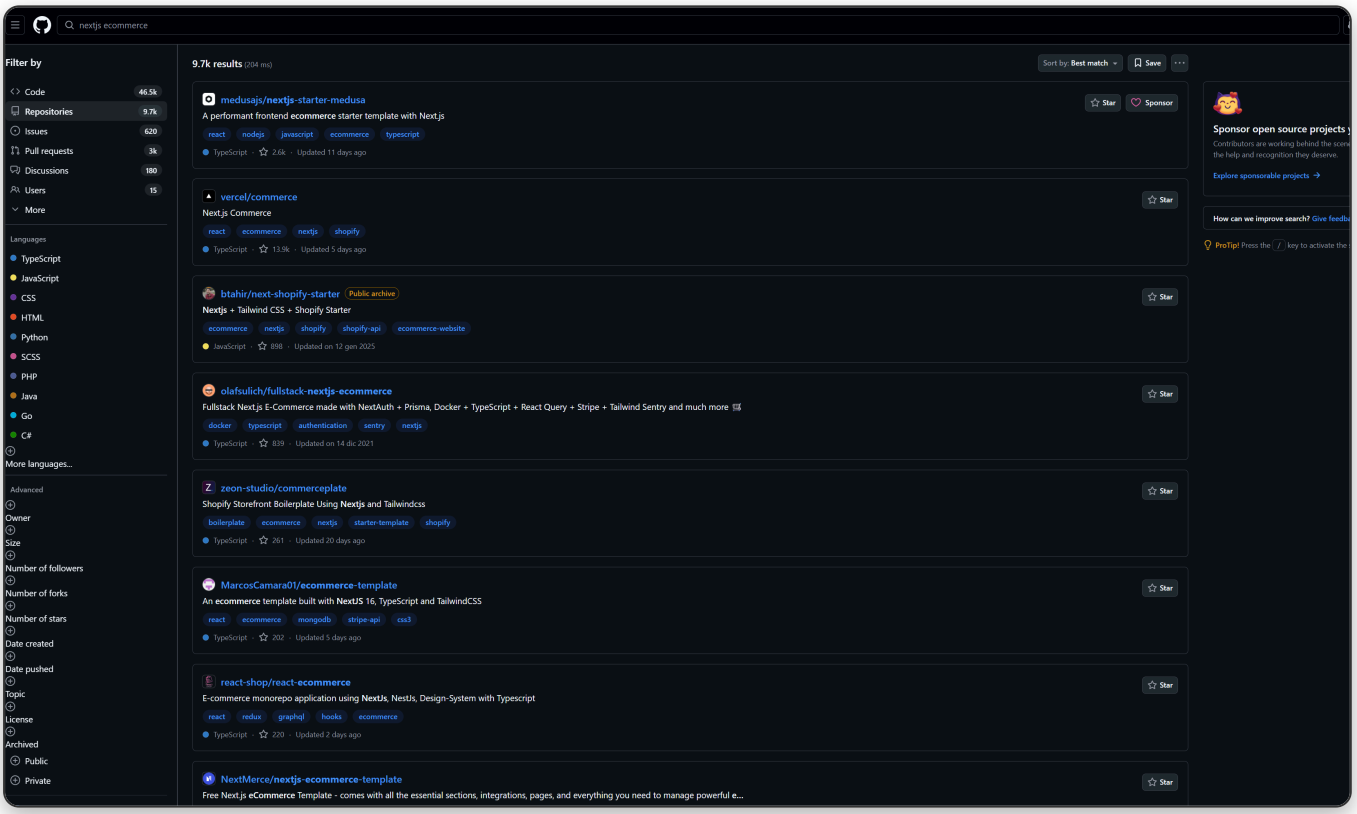
But here's the important nuance: **a well-built template in an older stack is almost always better than starting from scratch with a modern one.** You might find a PHP, jQuery, or Laravel project that's exactly what you need — complete, well-structured, battle-tested, with all the features already implemented. In that case, use it. AI agents can work with any technology, and the time you save from having a solid, ready-made foundation far outweighs the theoretical advantages of a newer framework. Claude Code can understand and modify PHP, Python, Ruby, or any other codebase just fine.

The priority is simple: **find the best template available.** If you find two templates of similar quality and one uses Next.js while the other uses PHP, go with Next.js. But if the PHP template is more complete, more feature-rich, and better maintained — pick that one without hesitation. Quality and completeness of the starting point matter more than stack modernity.

> **The rule of thumb:** use whatever you find that gets you closest to your goal. Aim for modern web technologies (Next.js, Electron, React Native) when available, but never reject a great template just because it uses an older stack. The time saved from a solid foundation is always more valuable than stack purity.

Practical example: finding a template

Let's say you want to build an e-commerce store. Instead of telling Claude Code *"Build me an e-commerce site from scratch"*, open GitHub and search for something like **"nextjs ecommerce"**. You'll find dozens of ready-made projects with product listings, shopping carts, checkout flows, and payment integration already built.



A quick GitHub search for "nextjs ecommerce" already shows several promising templates.

One important thing to keep in mind: many templates on GitHub are **freemium projects** — the core is open-source and free, but some features or premium versions require payment. Always check the repository's README and license before committing to a template. Avoid anything that requires paid subscriptions or hidden costs that you don't need.

Looking at the search results, we can spot **fullstack-nextjs-ecommerce** — and it's an excellent starting point. Let's analyze why. The project uses Next.js with TypeScript, which is exactly what we want for AI-assisted development. But what really stands out is what's already integrated: **Stripe**

for payments, **PostgreSQL with Prisma** as the database, and **NextAuth for authentication**. These are three of the most critical — and most error-prone — parts of any web application.

Having **Stripe already integrated** is particularly valuable. Payment processing involves webhooks, session management, error handling, and edge cases that can be surprisingly tricky to get right. When something goes wrong with payments, your customers are the ones who suffer — failed charges, duplicate payments, or broken checkout flows are the kind of bugs that destroy trust and cost you real money. Starting with a template that already has a working Stripe integration saves you from these headaches.

The project also uses **PostgreSQL** as its database, which is a solid choice. Postgres is reliable, well-documented, offers slightly better security defaults compared to alternatives like MySQL, and can be easily hosted on Amazon AWS, Hetzner, or similar providers — we'll cover server setup and deployment in the next chapter. The fact that **NextAuth** is already wired up means user authentication is handled out of the box, another complex piece you don't have to build from scratch.

This is the power of starting with the right template: instead of spending days (or weeks) setting up payments, database, and authentication — all things that are easy to get wrong — you start with a project where these critical systems already work. You can then focus entirely on customizing the product to your vision: changing the design, adding your products, modifying the business logic, and building the features that make your store unique.

Once you've found your template, the next step is simple: **download it and place it in your workspace folder.** Open that folder with Claude Code (or Antigravity) and start working. You can tell Claude Code to modify the template directly — change the branding, add new pages, rework the layout, integrate new features — or you can use the template as a **reference** for your own project. Even if you're building something slightly different, having the template in the same workspace means Claude Code can look at it, study the code patterns, and use them as a foundation for what it writes.

This is a crucial point: **always give Claude Code something to reference.** When the agent has a solid, working codebase to look at, it writes significantly better code. It follows the same patterns, uses the same conventions, and produces output that is consistent and reliable. When it has nothing to reference and has to generate everything from scratch, that's when mistakes happen — inconsistent file structures, wrong library versions, code that doesn't fit together. A template acts as an anchor that keeps the AI grounded and producing high-quality, coherent results.

> **What if the template doesn't have payments or a database?** That's fine too. This course includes ready-made Stripe integration files that you can give directly to Claude Code to integrate payments into any project. For the database, we recommend PostgreSQL or whatever database the template already uses — don't fight the template's choices unless there's a strong reason. Same goes for authentication: if the template uses NextAuth, Clerk, or any other auth system, work with it. The goal is to leverage what's already built, not to replace everything.

4. From Code to Production

> *Your product is built. Now it needs to accept payments, run on a server, and be fast and secure for users worldwide. This chapter covers the essential services that turn your project from local code into a live, production-ready business.*

> **A note about this chapter.** We're keeping things concise and to the point here. Our goal is to tell you **what** you need to do and **why** — not to write lengthy tutorials that waste your time. Any LLM (Claude, Gemini, ChatGPT) can explain the details, walk you through each step, and answer your questions far better than a static page ever could. Whenever something isn't clear, just ask your agent: *"I'm following a course and it says I need to [do X]. Can you explain what this means and walk me through it?"* This is faster, more personalized, and always up to date.

Payments with Stripe

If your product sells anything, you need a payment processor. **Stripe** is the industry standard: reliable, well-documented, and supported by virtually every AI agent because of how widely it's used.

Here's what you need to do:

- **Create a Stripe account** at stripe.com. You'll get **API keys** (a publishable key for the frontend, a secret key for the backend) and access to both **test mode** (fake payments for development) and **live mode** (real money).
- **Set up webhooks** in the Stripe dashboard. Webhooks are URLs on your server that Stripe calls when something happens — a payment succeeds, a subscription renews, a charge fails. This is how your app knows when to activate a subscription, confirm an order, or handle a failure. You need webhooks for both **local testing** (Stripe has a CLI tool that forwards events to your localhost) and **production** (pointing to your live server). Always get everything working locally before going live.
- **Integrate Stripe into your project.** If your template already has Stripe, just plug in your API keys. If not, this course includes ready-made Stripe integration files — drop them in your workspace and tell Claude Code to integrate them. Stripe supports one-time payments and recurring subscriptions, both using hosted checkout pages that handle card validation, 3D Secure, and PCI compliance for you.

One crucial rule: **always verify payments server-side through webhooks**, never trust the frontend. The webhook is Stripe telling your server directly that money actually changed hands — it's the only reliable source of truth.

Hosting: AWS, Hetzner & Beyond

Your app needs to live somewhere. The good news: **AWS gives you a Free Tier** when you sign up — for **one full year**, you get a **t2.micro server** and a **micro database** completely free. That's enough to host your first project while you validate the idea and start getting users.

Don't know how to set up an AWS account or use the Free Tier? Just ask Claude or Antigravity — they'll walk you through every step.

When you outgrow the free tier, here's what you need to know about server sizing:

- **t3.small** is the sweet spot for most apps — good CPU, enough RAM, and reasonable pricing (~$16/month on AWS). The "t" refers to the instance generation; older generations (t2, etc.) can sometimes cost more for less performance, so stick with the latest available.
- **Hetzner** is a European alternative that's *significantly* cheaper — you can get performance comparable to a t3.small for around **$4/month**. That's roughly 4× cheaper than AWS for similar specs.
- **Not every app needs a server.** If your project is a static site or a JAMstack app, you might not need a dedicated server at all. Platforms like Vercel, Netlify, or even Cloudflare Pages can host it for free or nearly free. Always ask your LLM: *"What's the best hosting for my specific app?"*

Our recommendation: **start with AWS Free Tier** for your first year, then evaluate whether Hetzner or another provider makes more sense for your budget and audience location. If most of your users are in Europe, Hetzner's German servers will give you lower latency. If your audience is global or US-based, AWS regions might be a better fit.

> **SSH keys & AI server management.** To let Claude Code connect to and manage your server remotely, you'll need to set up an **SSH key**. Ask Claude to generate one and configure it on your server. Once connected, Claude can deploy code, manage services, troubleshoot issues — all from your terminal. For server management tasks, we recommend using **Opus** for the best results, though Sonnet works too if you want to save tokens (with a slightly higher chance of errors).

Cloudflare: Performance & Protection

Once your app is on a server, you need something sitting in front of it to protect it and speed it up. That's **Cloudflare**. The good news: **the free plan is more than enough** for the vast majority of websites. You don't need a paid plan.

But first, you'll need a **domain name**. We recommend buying one directly from **Cloudflare** or **Namecheap** — both are reliable and fairly priced. Once you have your domain, you'll need to configure the **DNS** to point to the IP address of your AWS or Hetzner server. Just ask your LLM: *"I bought a domain on [Cloudflare/Namecheap]. How do I set up DNS to point to my server at [your IP]?"*It will walk you through it.

Why is Cloudflare so important? It protects your site from **DDoS attacks**, various **security exploits**, and common web vulnerabilities. It also provides a **global cache**, meaning your content gets served from servers close to your users, making your site faster worldwide. Without a service like Cloudflare, you're exposing your site to serious risks — especially now, in the AI era, where anyone can use AI tools to find vulnerabilities, exploit misconfigurations, or even steal data from poorly protected websites. Don't skip this.

> **Quick tip: Flexible SSL mode.** When setting up Cloudflare, you can choose **Flexible** SSL mode. This means the connection between Cloudflare and your server uses plain HTTP, but the connection between Cloudflare and your end users is HTTPS — so visitors see a secure padlock. This saves you from having to configure SSL certificates on your server, which is great for getting started quickly. For production apps that handle sensitive data, you should eventually switch to **Full** mode (encrypted end-to-end). But for your first tests and launches, Flexible is perfectly fine and saves a lot of setup time. Ask your LLM to explain the differences if you're unsure which mode fits your project.

There's more to Cloudflare than just protection. The free plan includes powerful features that many developers don't even know about:

- **Cloudflare Pages** — free static hosting for frontend apps (React, Next.js static export, Vue, etc.). You push to GitHub, Cloudflare builds and deploys automatically. Zero configuration, zero cost. Perfect for landing pages, portfolios, and JAMstack apps.
- **Edge Functions (Workers)** — run serverless code at the edge, close to your users, on the free plan. Great for API routes, redirects, A/B testing, and lightweight backend logic without needing a full server.
- **CDN & Caching** — your static assets (images, CSS, JS) are cached globally, making your site blazing fast from anywhere in the world.

The key question is: **does your app actually need a dedicated server, or can Cloudflare handle it for free?** Ask Claude: *"I'm building [describe your app]. Do I need a dedicated server on AWS/Hetzner, or can I use Cloudflare Pages and Workers for free?"* Claude will analyze your specific case and tell you the best option. You might be surprised how many projects can run entirely on Cloudflare's free tier.

API Keys: Automate Everything

Here's a game-changing tip that most tutorials skip: **create API keys for your hosting providers** and give them to Claude. This lets Claude manage your infrastructure directly from the terminal — no clicking around dashboards, no copy-pasting, no manual work.

- **Cloudflare API Key** — go to your Cloudflare dashboard → My Profile → API Tokens → create a token. With this, Claude can automatically configure DNS records, set up Pages projects, manage Workers, update SSL settings, and much more. Ask Claude: *"Here's my Cloudflare API token. Set up DNS for my domain pointing to my server IP."* Done in seconds.

- **AWS Access Keys** — go to AWS IAM → create an access key. With this, Claude can manage EC2 instances, configure security groups, set up RDS databases, manage S3 buckets, and handle your entire AWS infrastructure programmatically. Ask Claude: *"Here are my AWS credentials. Launch a t3.small EC2 instance in us-east-1 and configure the security groups for a web app."*
- **Hetzner API Token** — go to your Hetzner Cloud Console → project → Security → API Tokens → generate one. Claude can then create servers, configure firewalls, manage snapshots, and handle your entire Hetzner infrastructure. Ask Claude: *"Here's my Hetzner API token. Create a CX22 server in Nuremberg with Ubuntu."*

> **Security note on API keys.** These API keys are powerful — treat them like passwords. **Never commit them to Git**, never share them publicly, and never paste them in chat interfaces that you don't trust. Store them in a `.env` file or pass them directly to Claude in your terminal session. If a key gets compromised, revoke it immediately from the provider's dashboard and generate a new one. Also, when creating API tokens, **always use the least privilege principle**: give only the permissions that are actually needed, not full admin access.

The combination of these API keys with Claude is incredibly powerful. You can literally say: *"I have a Next.js app. Deploy it on Cloudflare Pages, set up the custom domain, and configure the DNS — here are my API keys."* And Claude will do the entire thing automatically. Or: *"Create an EC2 instance on AWS, install Node.js and PM2, configure nginx, set up Cloudflare DNS, and deploy my app."* Full infrastructure setup in minutes, not hours.

CI/CD with GitHub Actions

Remember when we set up Git and GitHub CLI back in Chapter 2? This is where it all pays off. With `gh` authenticated, you can tell Claude Code to **push your project to a private GitHub repository**, set up **GitHub Actions**, configure all the necessary **secrets** (your server's SSH key, environment variables, etc.), and create an automatic deployment pipeline — all from its terminal, without you touching GitHub's interface.

The idea is simple: once GitHub Actions is configured, **every time Claude Code pushes code to your repository, it automatically deploys to your server.** No manual SSH, no copying files, no running commands on the server yourself. Claude pushes, GitHub Actions picks it up, connects to your AWS or Hetzner server via SSH, and deploys everything. Fully automated.

Just tell Claude: *"Push this project to a new private repository on GitHub, set up a GitHub Action that deploys to my server on every push to main. Here's my server IP and SSH key."* Claude already knows how to do this — it will create the workflow file, add the SSH key and server IP as GitHub secrets, and configure the entire pipeline. This is exactly why we installed the GitHub CLI earlier.

> **Dynamic IP gotcha.** Static IPs usually cost extra on both AWS and Hetzner, so you'll likely be using a **dynamic IP** to save money. This means every time you stop and restart your server, the IP changes. When that happens, you'll need to update it in **two places**: the Cloudflare DNS record and the `SERVER_IP` GitHub secret. Tell Claude to store the server IP as a `SERVER_IP` secret in GitHub Actions, so when it changes you only need to update one variable. Also tell Claude to remind you to check and update the IP if a deployment fails — a changed IP is almost always the reason.

5. Marketing & SEO Tactics

> *Your product is live. Now the world needs to know it exists. The most effective marketing for indie products is organic — free, creative, and surprisingly powerful when done right. This chapter covers the exact tactics we use.*

Organic Growth Strategies

Let's be honest: **the best marketing doesn't look like marketing.** The internet is saturated with ads, and people have developed a reflex to ignore anything that feels like a promotion. What actually works is **stealth marketing** — content that looks like genuine information, a question, or a recommendation rather than an advertisement. People are naturally curious and love to help with suggestions. Use that.

Reddit is one of the most powerful platforms for this. It's massive, highly indexed by Google, and full of niche communities where your target audience already hangs out. But here's the key: **never post aggressive marketing.** Don't write *"Check out my amazing new site!"* — that will get downvoted, removed, or banned. Instead, write something like:

- *"Can anyone recommend sites similar to [well-known competitor] or [your site]? Looking for alternatives."*
- *"Has anyone tried [your product category]? I found [your site] but curious about other options."*
- Answer questions in relevant subreddits and naturally mention your product where it genuinely helps.

This is how the majority of successful indie marketing works on Reddit. You're not lying — you're framing your product as a discovery within a genuine conversation. People click because they're curious, not because they feel sold to. And here's a bonus: **Reddit posts get indexed by Google**, so a well-placed thread can bring you organic search traffic for months or even years.

You can also **sponsor a Reddit post** for a small budget to boost it temporarily. This pushes it higher in search results, lets Google index it faster, and gives it initial visibility. Ask your LLM how Reddit

post promotion works and what budget makes sense.

> **The funnel trick.** On your homepage or landing page, include something that **attracts people independently of your main product** — a free tool, a trending topic, useful information, or something that's currently popular. This acts as a **funnel**: people arrive for the free/interesting content, discover your product, and a percentage of them convert. Think of it as bait that provides genuine value while leading to what you're selling.

SEO, Content & Distribution

Before you start any marketing, set up your **analytics and tracking**. You need two things:

- **Google Analytics** — add the tracking code to your site (ask Claude to integrate it). There's also a **mobile app** so you can check your traffic from your phone anytime. This shows you total visits, user behavior, traffic sources, and conversion data.
- **Google Search Console** — set this up and connect it to Google Analytics. Search Console specifically tracks your **organic Google traffic**: what search queries bring people to your site, how often you appear in search results, and your click-through rates. Ask your LLM how to set up and connect both tools.

If you have budget available, **Google Ads** can give you an initial boost. Running ads briefly helps build trust with Google's algorithm and drives early traffic while your organic SEO grows. But the costs add up quickly, so treat it as a short-term accelerator, not a long-term strategy. Your LLM can explain Google Ads setup and budgeting in detail.

For SEO itself, start with the basics. Open your browser's **DevTools** (F12), go to **Lighthouse**, and generate a report. This gives you scores for Performance, Accessibility, Best Practices, and **SEO**. Fix what it tells you to fix — and yes, you can ask Claude Code to handle the fixes.

Key SEO actions to take:

- **Add translations** to your pages. Multi-language support dramatically increases your reach. Ask Claude Code to set up internationalization for your project.
- **Add proper meta tags** — title, description, Open Graph tags for social sharing, structured data. These directly affect how your site appears in Google search results.
- **Create and submit a sitemap.** Generate an up-to-date sitemap and upload it to Google Search Console. This tells Google exactly which pages exist on your site and helps them get indexed faster.

SEO requires patience. Don't expect organic traffic overnight — it takes weeks or months for Google to properly index and rank your pages. But when it kicks in, it's **free traffic that keeps coming** without you spending a cent. The clicks you'd otherwise pay hundreds of euros for via Google Ads will come for free through organic search. In the meantime, put in the work on social platforms like Reddit to build initial visibility and backlinks. SEO is a long game, but it's the most valuable marketing investment you can make.

The emerging trend: AI-driven traffic

There's a new and rapidly growing source of traffic that most people aren't paying attention to yet: **LLMs recommending your site.** ChatGPT, Gemini, Claude, and other AI assistants are increasingly being used as search engines. When someone asks *"What's a good site for [your product category]?"*, these models can and do recommend specific websites — and that drives real traffic. A significant portion of our own visits come from ChatGPT and other LLMs.

To take advantage of this, go to your **Cloudflare dashboard** and make sure you **disable the option that blocks AI crawlers**. Many sites block AI bots by default, which prevents LLMs from learning about your content. By allowing AI crawlers to access your site, you're letting these models index your pages, understand what you offer, and potentially recommend you to users in the future. This is essentially **free SEO for the AI era** — and the boost can be enormous.

Think beyond Google. Traditional SEO targets Google search. But AI-driven recommendations are becoming a major traffic source — and this trend is only accelerating. Make sure your site is accessible to AI crawlers, has clear and descriptive content, and is well-structured so LLMs can easily understand what you offer. The sites that position themselves now for AI discovery will have a massive advantage as this channel grows.

Thank You!

Thank you for purchasing this course and for trusting us with your time and money. We genuinely hope you found it valuable and that it helps you build something real.

We'd love to hear from you. **Join our Discord community** at apifreellm.com — it's where we hang out, share updates, and help each other out.

If you have a moment, **write us a review** and tell us what you think. What did you find most useful? What was missing? What could be better? We're genuinely interested in your feedback — this course is a living project and we want to keep improving it based on what **you** actually need.

See you on Discord. Now go build something amazing.