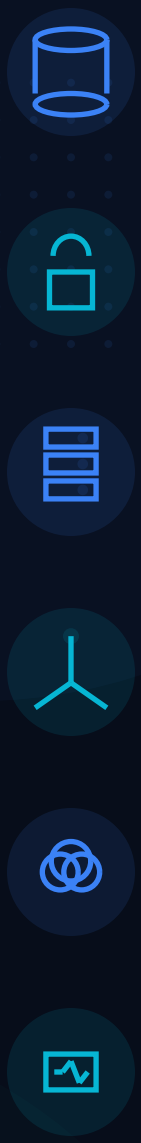
TECH BUILT RIGHT

What's Actually Running Your App

A Non-Technical Guide to the Infrastructure You Didn't Know You Had



By Adam | Tech Built Right

techbuiltright.com

For AI Founders Building the Future

Contents

You Built Something. Now Let's Talk About What's Under It.

You used AI tools to build an app. Maybe Cursor, maybe Bolt, maybe Replit. It works. People are using it. That is a real accomplishment and you should feel good about it.

But here is the thing nobody tells you: the tools that helped you build it do not explain what they built. There is an entire layer of infrastructure underneath your app that you never explicitly chose, probably do not fully understand, and almost certainly are not monitoring.

That is not a criticism. It is just reality. And it is fine, as long as you know it exists.

This guide will walk you through the major pieces of infrastructure that keep your app running. No jargon. No lectures. Just a plain explanation of what each thing does, what can go wrong, and one thing you can check right now to see where you stand.

You do not need to become an infrastructure engineer. You just need to know enough to ask the right questions and make informed decisions about your product.

Let's get into it.

CHAPTER 01

Databases

Where all your users' information actually lives, and what happens when it is not protected.

Every app that stores information, user accounts, messages, orders, AI conversation history, anything, uses a database. Think of it as a giant, organized filing cabinet that your app reads from and writes to thousands of times per minute.

When you signed up for Supabase, Firebase, PlanetScale, or MongoDB, you got a database. When your AI coding tool set one up for you, it made choices about how your data is structured, how it is accessed, and whether it is backed up.

The problem is that most AI built apps have a database that works but is not protected. There are no automatic backups. There is no plan for what happens if the database fills up. There is no restriction on who can access it directly.

REAL SCENARIO

A founder's Supabase database had no row level security enabled. Any logged in user could read every other user's data by changing one number in the URL. It was live for four months before anyone noticed.

SELF CHECK

- Can you log in to your database provider's dashboard right now?
- Do you know if automatic backups are turned on?
- Do you know who else has the database password or connection string?

CHAPTER 02

Authentication

The AAA framework: who gets in, what they can do, and what they did.

In infrastructure, identity and access is governed by a framework called AAA: Authentication, Authorization, and Accounting.

Authentication is how your app knows who someone is. Authorization is what that person is allowed to do. Accounting is the record of what they actually did: every login, every action, every failed attempt.

Most AI built apps handle authentication well through providers like Clerk, Auth0, or Supabase Auth. But authorization and accounting are almost always missing. Are there permission levels? If someone steals a session token, how long before it expires? And if something goes wrong, can you trace who did what and when? Without an audit trail, you cannot investigate breaches, prove compliance, or even understand how a billing discrepancy happened.

REAL SCENARIO

An app used Clerk for login but never set up role based access. Every user had admin privileges. A free tier user discovered they could access the billing dashboard and modify other accounts' subscription status. Because there was no accounting layer, the founder had no audit trail to determine how many accounts were affected or for how long.

SELF CHECK

- Do you know what authentication provider your app uses?
- Are there different user roles with different permissions?
- Do you have logs that show who did what and when inside your app?

CHAPTER 03

Hosting

Where your app physically runs, and why it matters more than you think.

Hosting is where your application code actually runs. When a user opens your app, their browser sends a request to a server somewhere, that server runs your code, and sends back the result. That server is your hosting.

You might be on Vercel, Netlify, Railway, Render, Fly.io, Heroku, or AWS. Each has different pricing models, scaling behavior, and limitations. Some scale automatically. Some do not. Some charge you per request. Some charge you per hour whether anyone is using your app or not.

The most common problem is not knowing which one you are on or what it costs at scale. A hosting bill that is $20 per month at 100 users can become $2,000 per month at 10,000 users depending on the provider and configuration.

> **REAL SCENARIO**
>
> A founder on Vercel's hobby plan hit the serverless function execution limit during a product launch. The app started returning errors to every user. They did not know the limit existed because the AI tool that deployed it never mentioned it.

SELF CHECK

- Do you know exactly where your app is hosted?
- Do you know what your monthly hosting bill is, and what triggers it to increase?
- If your app got 10x traffic tomorrow, do you know what would happen?

CHAPTER 04

Load Balancers

The traffic cops that keep your app from melting under pressure.

When more than a handful of people use your app at the same time, a single server cannot handle all the requests. A load balancer sits in front of multiple servers and distributes incoming traffic across them so no single server gets overwhelmed.

If you are on a managed platform like Vercel or Railway, you probably have some form of load balancing built in. If you are on AWS or a VPS, you may need to configure it yourself. Either way, it is the reason your app does not crash every time more than 50 people use it at once.

For most early stage apps, the built in load balancing from your hosting provider is sufficient. It becomes a real concern when you hit thousands of concurrent users, run long AI inference calls, or need to maintain persistent connections like websockets.

WHY IT MATTERS

Without load balancing, a traffic spike does not slow your app down gradually. It crashes it entirely. Every user gets an error, not just the extra ones.

SELF CHECK

- Does your hosting provider handle load balancing automatically?
- Have you ever tested your app under concurrent load?

CHAPTER 05

Serverless Compute

Code on demand. Great until the bill arrives.

Serverless means your code runs only when someone triggers it, and you pay only for the time it runs. No always on server, no idle costs. Vercel functions, AWS Lambda, Cloudflare Workers, these are all serverless.

For AI apps, serverless is common because AI inference calls are bursty: nothing for 10 seconds, then 50 requests at once. Serverless scales up instantly to handle the burst, then scales back to zero.

The catch is cost unpredictability. Each function execution has a cost. If your AI feature makes three API calls per user interaction, and each call triggers a serverless function, your costs multiply fast. A function that costs $0.001 per run costs $1,000 at a million runs. That can happen in a week if your app goes viral.

The other catch is cold starts. When a serverless function has not been called recently, it takes longer to respond the first time. For AI apps where users expect instant responses, this delay can feel broken.

SELF CHECK

- Do you know if your app uses serverless functions?
- Do you know the per execution cost of your AI inference calls?
- Have you set a spending limit or alert on your hosting provider?

CHAPTER 06

CI/CD Pipelines

How code changes get from your laptop to your live app, safely.

CI/CD stands for Continuous Integration and Continuous Deployment. In plain terms, it is the automated process that takes code changes and pushes them to your live app. When you push code to GitHub and your app updates on Vercel a few minutes later, that is a CI/CD pipeline.

The "CI" part runs checks before deploying: does the code compile, do the tests pass, are there any obvious errors. The "CD" part actually deploys it. Together, they are the safety net between writing code and breaking your app for real users.

Most AI built apps have the deployment part but not the testing part. Code goes straight from your editor to production with no checks. That means every change is a gamble. It works until it does not, and when it does not, your users find out before you do.

REAL SCENARIO

A founder used Cursor to fix a bug and deployed immediately. The fix worked, but it also changed a database query that broke the signup flow. No tests caught it. They lost 3 days of new signups before a user emailed them about it.

SELF CHECK

- Do you know how updates get from your code to your live app?
- Is there a staging or test environment, or does everything go straight to production?
- Has a bad deploy ever broken something for users?

CHAPTER 07

Monitoring & Alerting

If your app breaks right now, how would you know?

Monitoring is how you see what your app is doing: how many requests it handles, how fast it responds, whether any errors are occurring. Alerting is what tells you when something goes wrong. Together, they are the difference between finding out about a problem in 30 seconds and finding out from an angry user three days later.

Tools like Sentry catch code errors. Datadog or Grafana show performance metrics. Vercel and Railway have basic analytics dashboards. PostHog and Mixpanel track user behavior. Each serves a different purpose.

Most AI built apps have zero monitoring. The founder's first indication that something is wrong is either a blank page when they open the app themselves, or a message from a user saying it is broken. By then, the damage is done. Users who hit errors silently leave. They do not email you.

THE SILENT KILLER

An AI app's OpenAI API key hit its rate limit during peak hours. The app returned empty responses instead of errors. It looked like it was working. Users just thought the AI was bad. Churn spiked for two weeks before anyone investigated.

SELF CHECK

- If your app went down right now, would you get a notification?
- Do you have any error tracking tool installed?
- Can you see how many errors your app had in the last 24 hours?

CHAPTER 08

DNS & CDN

The invisible highways that connect your domain to your app and make it load fast everywhere.

DNS, Domain Name System, is the phone book of the internet. When someone types yourapp.com, DNS translates that into the actual server address where your app lives. It is invisible when it works. When it breaks, your app simply vanishes from the internet.

CDN, Content Delivery Network, stores copies of your app's static files (images, CSS, JavaScript) on servers around the world. When a user in Tokyo loads your app, they get files from a server in Tokyo instead of one in Virginia. It makes your app feel fast everywhere.

Most managed platforms like Vercel and Netlify include a CDN automatically. DNS is usually managed wherever you bought your domain: GoDaddy, Namecheap, Cloudflare, Google Domains. The risk is that these are configured once and forgotten. If the DNS record points to an old server, or if your domain registration expires, your entire app disappears.

REAL SCENARIO

A founder's domain auto renewal failed because the credit card on file expired. The domain lapsed. A bot registered it within hours. Getting it back took weeks and cost $2,000.

SELF CHECK

- Do you know where your domain is registered and when it expires?
- Is auto renewal turned on with a valid payment method?
- Do you know where your DNS records are managed?

CHAPTER 09

What Now?

If you answered "I don't know" to more than a few of those self checks, you are not behind. You are normal. Most AI built apps are in exactly this position.

But now that you know these systems exist, ask yourself two questions:

1. **If any of the issues outlined in this guide happened to your app right now, could you accurately identify what caused it?**

2. **Could you fix it in a short enough window before it severely impacted your business revenue and brand perception?**

For most founders, the honest answer to both is no. And that is not a personal failing. It is the reality of how deep these systems go.

There is a reason companies have dedicated IT departments, and within those departments, specialized infrastructure engineers. Databases, networking, security, observability, cloud architecture: each of these is its own discipline with years of depth. These specializations exist because modern infrastructure evolves so quickly and runs so deep that generalists cannot keep pace. Supporting the business needs of production applications and services, whether in the cloud or on premise, demands focused expertise.

You should not need to become an infrastructure engineer to run your business. But you do need access to someone who already is one.

Tech Built Right exists to bridge that gap. We are an advisory practice built specifically for AI founders who shipped fast and now need to make sure their product can survive real users, real traffic, and real threats.

We do not sell implementation. We do not upsell retainers. We review your stack with 12+ years of infrastructure experience across enterprise healthcare (HIPAA), Fortune scale data centers, and 5G networks, and we tell you exactly what needs attention, in what order, and why.

Book a Free Assessment

30 minutes. No pitch. We review your entire stack, flag the risks by severity, and deliver a prioritized remediation roadmap your team can act on immediately. If everything looks solid, we tell you that too.

- **Security & Threat Assessment**
- **Cloud & Network Architecture**
- **DevOps & CI/CD Strategy**
- **Cost Modeling & Inference Economics**

techbuiltright.com

You built something real. Let's make sure it stays that way.

TECH BUILT RIGHT

Infrastructure & Security Advisory
for AI Founders

techbuiltright.com