The First 90 Days

A standalone playbook for a fractional or new CTO's first 90 days — the 30/60/90 framework, listening tour, scorecards, and the scope-of-work and decision templates that make an engagement work from day one.

Roger Stringer · rogerstringer.com

June 20, 2026

Contents

Why the First 90 Days Decide Everything 3
Before Day One 4
The Listening Tour 5
Reading the Codebase & Architecture 6
Reading the Team 7
The 30-Day Diagnosis 8
Quick Wins vs. Deep Fixes 9
The 30/60/90 Plan 10
Speaking to the Board & Founders 11
Setting Up How You Work 12
The 90-Day Review 13
Templates & Checklists 14
About Roger 16

Why the First 90 Days Decide Everything

Whether you're a brand-new full-time CTO or a fractional one walking into your third client this quarter, the first ninety days decide how the whole engagement goes — and most people spend them doing the wrong things. They arrive, feel the pressure to prove value, and start *shipping* on day three: rewriting the thing that annoyed them in the interview, reorganizing the team, announcing a new architecture. It feels productive. It's how engagements quietly fail.

The first ninety days are not for fixing. They're for *earning the right* to fix — and for understanding the system well enough that your fixes are the right ones. You're walking into a business with history you don't have, decisions that look wrong but had reasons, and a team watching to see whether you're another consultant who'll reorganize everything and leave. Move too fast and you torch your credibility on a change that turns out to be naive. Move too slow and you look passive while the founders wonder what they're paying for. The art is the sequence: understand deeply, prove value early and small, then earn the mandate for the deep work.

This guide is the playbook for that sequence — expanded from a chapter in the Fractional CTO Field Guide (https://rogerstringer.com/guides/the-fractional-cto-field-guide) into the full thing, with the templates to run it. The arc:

- **Before day one** — the agreements that make or break the engagement before it starts.
- **The first 30 days** — a structured listening tour, reading the codebase and the team, ending in a written diagnosis.
- **Days 30–60** — quick wins to build trust, and the 30/60/90 plan that sequences the real work.
- **Days 60–90** — setting up how you'll work, and a review that resets scope for the next quarter.

The through-line, which the rest of the guide keeps returning to: **your job in the first ninety days is to become the person whose judgment the founders trust on the hard calls.** Everything else — the diagnosis, the quick wins, the plan — is in service of that. Earn the trust and the mandate for the real work follows. Skip it and you'll spend the engagement fighting for permission. Let's start before you've even begun.

Before Day One

The most expensive mistakes in a CTO engagement are made before it starts — in the things everyone assumed and nobody wrote down. Walk in without these four agreements and you'll spend the engagement discovering that you and the founder meant different things.

Scope: what are you actually responsible for? "Be our CTO" is not a scope. Are you owning the existing product's reliability, building a new thing, leading the team, advising the founder — all of it? Fractional especially: are you two days a week of hands-on technical leadership, or one day of strategic advice? Pin it down in writing, because the gap between "advisor" and "owner" is where resentment grows — you thinking you're advising, them thinking you're accountable.

Authority: what can you decide versus recommend? Responsibility without authority is a trap. If you're accountable for engineering outcomes but can't hire, fire, set technical direction, or say no to a feature, you're holding the blame without the levers. Establish explicitly: what you decide alone, what you decide with the founder, what's the founder's call that you advise on. Get this wrong and you'll be blamed for outcomes you couldn't control.

Success: what does good look like in 90 days, and in a year? Ask the founder directly: "Ninety days from now, what has to be true for you to feel this is working?" Their answer is gold — it tells you what they actually care about (often different from the job description) and gives you a target to aim the first quarter at. Write it down; you'll measure the 90-day review against it.

Reporting: who do you answer to, and how often? The CEO? The board? A founder who's also the lead engineer (a delicate one)? And what's the cadence — a weekly 1:1, a monthly board update? The relationship that isn't scheduled doesn't happen, and a CTO the founder never hears from is a CTO the founder stops trusting.

For a fractional engagement, all of this belongs in the scope-of-work document (there's a template at the end of the guide). For a full-time role, it belongs in an explicit first-week conversation, written up afterward so there's a shared record. Either way: the conversation that feels slightly awkward to have up front is the one that prevents the painful conversation in month four. Now, day one.

The Listening Tour

Day one starts with your mouth shut and your ears open. The first two weeks are a **listening tour** — a deliberate, structured round of conversations whose entire purpose is to understand the business, the system, and the people before you form a single opinion out loud. The discipline is hard precisely because you'll see things you want to fix immediately. Note them; don't act on them yet. You don't know why they're that way.

Who to talk to, roughly in order:

- **The founders / CEO** — what's the business actually trying to do, what keeps them up at night, what do they think is wrong with engineering. Their fears are your real brief.
- **Every engineer, one on one.** Not a group meeting — individuals, privately. They know where the bodies are buried, and they'll tell you things in a 1:1 they'd never say in a standup.
- **Adjacent functions** — product, sales, support, ops. They experience engineering's output from the outside, and their complaints ("every release breaks something," "it takes months to get anything") are diagnostic.
- **Whoever was technical before you** — the previous lead, the founder who wrote the first version, the agency that built it. The history is in their heads.

What to ask. Open questions that invite the truth, not yes/no:

- "Walk me through how things work here — how does a change get from idea to production?"
- "If you could fix one thing about how we build software, what would it be?"
- "What's everyone afraid to touch?" (Always revealing — it's where the risk lives.)
- "What's working well that I shouldn't break?" (As important as the problems. You'll be tempted to change things that are quietly load-bearing.)
- "Who really knows how [the critical system] works?" (Key-person risk, mapped in one question.)

How to hold it. Take real notes — you're pattern-matching across a dozen conversations, and the patterns are the diagnosis. When the same fear or complaint comes up from three different people, that's a finding, not an anecdote. And resist the urge to fix things mid-tour; the credibility you'd spend by half-understanding a problem and pronouncing on it is credibility you'll want later. "Tell me more" is the most powerful thing you can say in week one.

The listening tour gives you the human and business map. Running in parallel, you need the technical one — a structured read of the system you've inherited.

Reading the Codebase & Architecture

While the listening tour maps the people, you're reading the system — fast, structured, and without trying to fix anything yet. You're not auditing for a buyer (that's the Technical Due Diligence (https://rogerstringer.com/guides/technical-due-diligence) guide); you're orienting, building the mental model you need to lead. The goal is to know, within two weeks, where the system is solid, where it's fragile, and where the risk concentrates.

A fast structured read, roughly in this order:

Get it running yourself. Clone the repo, follow the setup docs, get it running locally. This is the single most diagnostic hour of your first week: a project that runs in ten minutes tells you one thing about the team's discipline; a project that takes two days and an engineer's help tells you a very different one. The onboarding experience *is* a measurement.

Trace one real request end to end. Pick a core user action — a signup, a checkout, the main thing the product does — and follow it through the code from entry to database and back. You'll learn the architecture faster from one real path than from any diagram, and you'll feel where it's clean and where it's a thicket.

Read the shape, not every line. You're looking for structure and signals, not reviewing code. How is it organized? Are there tests, and do they run? How does it get deployed? What's the git history like — steady and reviewed, or chaotic? Where are the `TODO`s and the apologetic comments clustered? The shape tells you most of the story.

Find the load-bearing risks. Two questions matter more than the rest: *what's the thing that, if it broke, takes the business down?* and *what's the thing only one person understands?* The intersection — critical and known by one person — is your highest-priority risk, and you'll often have heard about it in the listening tour too.

Note, don't fix. Same discipline as the listening tour. You will see code that makes you wince. Write it down, understand why it's that way before you judge it (the wince-inducing thing is sometimes the pragmatic call that kept the company alive), and save the fixing for after you've earned the mandate.

By the end you should be able to draw the architecture on a whiteboard from memory, name the three biggest technical risks, and explain how a change reaches production. That's the technical half of the picture. The other half is the people who build it.

Reading the Team

Engineering outcomes are mostly a function of the team, not the code, so reading the people well is at least as important as reading the system — and harder, because you're judging humans fast and you'll be wrong if you're careless. The goal isn't to rank everyone in week two; it's to understand the team's real capability, where it's strong, where it's thin, and what's quietly at risk.

What you're trying to read:

Capability and range. Who can be handed an ambiguous problem and trusted to come back with the right thing, and who needs a well-specified ticket? Both are valuable — a team is mostly the second kind — but you need to know the mix, because it determines how much you can delegate and where the gaps are. Watch how people talk about their work, not just credentials: the engineer who explains *why* they made a tradeoff is showing you judgment.

Morale and trust. Is the team energized or beaten down? Do they trust each other and their leadership, or is there resignation in the room? Low morale is both a finding and a constraint — a demoralized team can't absorb a big change, so it shapes what you can do in the first quarter. The listening-tour 1:1s are where you sense this; people will tell you, directly or in what they don't say.

Gaps. What's missing — a role, a skill, a seniority level? No one who owns the infrastructure? No one senior enough to mentor? A team of seniors with nobody who enjoys the grindy maintenance? Gaps are some of your clearest early findings, because they translate directly into a hire or a re-org you can recommend.

Key-person risk. The single most important thing to map: who, if they left tomorrow, would take irreplaceable knowledge with them? Every early-stage company has at least one, usually the person who wrote the original system. That's not a criticism of them — it's a risk to the business, and de-risking it (documentation, pairing, spreading the knowledge) is often one of your highest-value early moves.

Two cautions. **Judge fairly and slowly — you're seeing a snapshot.** The quiet engineer might be the most reliable; the loud one might be coasting. Hold your read loosely for the first month and let evidence accumulate. And **the team is reading you back, harder than you're reading them.** They're deciding whether you're worth trusting, and the way you run the listening tour — curious, respectful, not pronouncing — is the first impression that determines whether they'll follow you when you do start changing things.

Two weeks in, you've got the business, the system, and the people. Time to turn all those notes into something useful: a diagnosis.

The 30-Day Diagnosis

Around the 30-day mark, you stop gathering and start synthesizing. The listening tour, the codebase read, the team read — all those notes — get turned into a single written **diagnosis**: your honest assessment of where things stand and what actually matters. Writing it down is the point. The discipline of putting it on paper forces you from impressions to a position, and the document becomes the thing you align the founders around.

What the diagnosis contains:

The current state, honestly. Where the business is, what engineering's job is right now, what's working and what isn't — in plain language, without sugar-coating or doom. Founders can tell when they're being managed; they trust the assessment that names the real problems *and* credits what's genuinely good.

The risks, ranked. The three to five things that could actually hurt the business — the single-person dependency, the system that can't scale past the next milestone, the security hole, the team gap. Ranked, because "everything is a problem" is useless; the founders need to know what to worry about *first*.

What's load-bearing and must not break. Explicitly call out what's working that you intend to protect. This signals maturity — you're not the consultant who torches everything — and it guards you against your own early instinct to change things that are quietly holding the company up.

The short list of what matters. Not a backlog — the three or four things that, if you got them right over the next quarter, would matter most. This is the seed of the 30/60/90 plan, and keeping it short is the hard part. Founders are used to engineering producing infinite to-do lists; a CTO who can say "these three things, in this order" is showing exactly the judgment they hired you for.

A few principles for the document itself: **keep it short** (a few pages, readable by a non-technical founder), **lead with business impact** not technical detail (the Fractional CTO Field Guide (https://rogerstringer.com/guides/the-fractional-cto-field-guide)'s "speak in outcomes" rule starts here), and **separate observation from recommendation** so the founder can trust your facts even where they debate your conclusions.

Then you walk the founders through it. This conversation is a milestone: the first time you demonstrate that you understand their business better than they expected, and where you earn the mandate for what comes next. Done well, the founder's reaction is relief — *finally, someone who gets it.* That reaction is the whole point of the first thirty days. With the diagnosis agreed, you can start acting — carefully, starting with wins.

Quick Wins vs. Deep Fixes

With the diagnosis agreed, you finally get to *do* things — and the sequencing of what you do first is its own skill. The temptation is to charge at the biggest problem. The wiser move is to bank a few **quick wins** first, because in the early engagement, demonstrated value buys you the mandate for the hard, slow work that follows.

Why quick wins first. You're still building trust. A visible improvement in week four or five — something the team or the founders actually feel — proves you can execute, not just assess, and it earns you the room to then disappear into a three-month deep fix without the founders wondering what they're paying for. Quick wins are how you finance credibility.

What makes a good quick win: high visibility, low risk, genuinely useful — not theater. The deploy that took an hour now takes five minutes. The flaky test suite everyone ignored now passes reliably. The painful local setup now works in one command. The status page that finally tells support what's down. These are real — they make the team's life better and they're felt immediately — but they're contained, so a misstep doesn't blow up. Avoid the fake win (a cosmetic change that impresses no one who matters) and the disguised deep fix (the "quick" thing that turns out to need a month).

What's a deep fix, and how to treat it. The re-architecture, the platform migration, the team restructure, paying down the load-bearing tech debt — these are real and necessary, and they're slow, risky, and easy to get wrong while you're still learning the system. Deep fixes go on the 30/60/90 plan with proper sequencing; they don't get started in week four on a hunch. The cardinal error is leading with a deep fix — announcing a six-month rewrite before you've earned trust or fully understood why the current thing exists. That's how good CTOs burn their mandate in month one.

The sequencing instinct: quick wins in the first 30–60 days to establish that you deliver, deep fixes scheduled deliberately for once you've got the credibility and the understanding to do them safely. You're not avoiding the hard work — you're earning the right to do it well. And the discipline of distinguishing the two, honestly, is itself a signal to the founders that you know the difference, which is more than the last person did.

With wins banked and the deep work understood, you turn the diagnosis into an actual plan.

The 30/60/90 Plan

The diagnosis says what matters; the **30/60/90 plan** says when it happens and who owns it. It's the document that turns your assessment into a sequenced commitment — the thing you and the founders agree to, and the thing you'll be measured against at the 90-day review. Keep it concrete and keep it short; a plan nobody can hold in their head is a plan nobody follows.

The structure is exactly what the name says — three horizons:

Days 0–30 (mostly behind you): understand and stabilize. The listening tour, the diagnosis, the first quick wins, any urgent fire that genuinely couldn't wait. By the time you write the plan this is largely done; it's in the document so the arc is legible.

Days 30–60: prove and prepare. The bulk of the quick wins land here, and you lay the groundwork for the deep work — the spike that de-risks the migration, the hire you start recruiting for, the documentation that defuses the key-person risk. Visible progress plus quiet preparation.

Days 60–90: execute the first deep work. The first real, substantial improvement — the thing from the diagnosis that actually moves the business — gets underway and ideally shows early results before the 90-day mark. Not necessarily finished, but underway and visibly on track.

What makes the plan actually work, rather than be a deck nobody opens:

- **Each item has an owner and a rough date.** "Improve reliability" is a wish; "Jana leads moving deploys to Coolify by end of week 6" is a plan. Ownership and dates are what separate the two.
- **It's framed in outcomes, not tasks.** "Cut deploy time from an hour to under ten minutes," not "set up a CI pipeline." The founders care about the outcome; the task is your business. (Straight from the Fractional CTO Field Guide (https://rogerstringer.com/guides/the-fractional-cto-field-guide)'s outcomes rule.)
- **It's honest about capacity.** A plan that assumes the team can do three big things at once is a plan that fails and costs you credibility. Sequence ruthlessly; under-promise against what the team can actually absorb while keeping the lights on.
- **It leaves room for the unknown.** Something will catch fire in week seven. A plan packed to 100% has no slack to absorb it. Plan to roughly 70% of capacity and the surprises don't derail everything.

The plan is a living document — you'll revisit it at the 90-day review and reset it for the next quarter — but committing to it now is what makes the engagement feel directed rather than reactive. Founders sleep better when there's a plan with their name on the outcomes. Which is really about communication — the skill that determines whether all this work actually lands.

Speaking to the Board & Founders

You can do everything else right and still fail the engagement if you can't communicate it. A CTO spends a surprising amount of the first ninety days translating — turning technical reality into something founders and boards can understand, trust, and act on. The engineers who never learn this stay engineers; the ones who do become the technical leader the business actually relies on.

The core skill is **speaking in outcomes and risk, not architecture.** Leadership doesn't care that you're migrating to a queue-based system; they care that "orders will stop occasionally failing silently during traffic spikes." They don't want to hear about test coverage; they want "we can now ship changes without the fear that we've broken checkout." Every technical thing you do has a business translation, and your job is to do the translating so they never have to. (The Fractional CTO Field Guide (https://rogerstringer.com/guides/the-fractional-cto-field-guide) calls this speaking in outcomes; in the first ninety days it's how you build the founders' trust in your judgment.)

What this looks like in practice:

- **Lead with the 'so what.'** Start with the impact, then the detail if they want it. "We have a risk that could take the site down during a launch, and here's my plan to remove it" — not a fifteen-minute architecture lecture that buries the point.
- **Quantify in their terms.** Money, time, risk, customer impact. "This saves the team a day a week," "this is the thing most likely to cause an outage," "this is why features take a month instead of a week." Numbers in their currency, not yours.
- **Bring problems with proposed answers.** "Here's a risk, here's what I recommend, here's what it costs" — not "here's a problem, what do you want to do?" You're hired for judgment; show it. A board trusts the CTO who arrives with a recommendation and the reasoning behind it.
- **Be honest about bad news, early.** The instinct to hide the slipping timeline or the nasty risk until you have a fix is exactly wrong — founders forgive problems surfaced early far more than ones sprung late. Early honesty is the foundation of the trust the whole engagement runs on.

For board updates specifically: short, outcome-framed, honest about risk, and consistent in format so they can track progress quarter over quarter. A board that gets a clear, steady technical read from you stops worrying about engineering — which is the entire job of the update.

The meta-point: communication isn't separate from the technical work, it *is* the leadership half of the job. The first ninety days are where you establish that you're someone whose read on technical reality the founders can rely on without having to verify it themselves. Earn that, and the rest of the engagement gets dramatically easier. Which brings us to making the way you work durable.

Setting Up How You Work

Somewhere in the second month you shift from *assessing* the team to *running* it — and the operating rhythm you establish now is what lets the engagement function without you holding every thread by hand. This is especially true for a fractional CTO, who isn't there every day: the cadences have to carry the team when you're not in the room.

The pieces of a working rhythm:

Cadences — the standing meetings that create predictability. A weekly 1:1 with the founder (or your reporting line), a team cadence that fits how they work (weekly planning plus a lightweight daily sync, for most small teams), and a monthly or quarterly step back for direction. The goal isn't more meetings — it's the *minimum* set that keeps everyone aligned, so decisions and blockers surface on a schedule instead of randomly. A fractional CTO especially needs these load-bearing, because they're how the team stays coordinated on the days you're not there.

Decision rights — who decides what, made explicit. You established your own authority before day one; now do it for the team. Who can deploy? Who approves an architecture change? What needs your sign-off versus what should the team just *do* without waiting for you? Clear decision rights are what let a team move fast without you as the bottleneck — and the bottleneck is the failure mode fractional leaders fall into, where everything waits for their two days a week. Push decisions down as far as they'll safely go.

How work flows — the path from idea to production. You learned this path in the listening tour; now you shape it. How does work get prioritized, picked up, reviewed, shipped? You don't need heavy process — small teams die under too much of it — but you need *enough* that work doesn't depend on everyone remembering how things go. Just enough structure that the system runs without heroics.

The handoff principle — build it to outlast you. The mark of a good engagement, fractional or full-time, is that the team gets *better at running itself*, not more dependent on you. Every cadence and decision right you set up should make the team more autonomous, not less. The fractional CTO who makes themselves indispensable has failed; the one who makes the team able to run without them has succeeded — and, paradoxically, gets kept around longer for it.

With the rhythm in place, the team runs on a system instead of on your constant attention. Which sets up the last milestone of the ninety days: the review.

The 90-Day Review

At ninety days you close the loop. The 90-day review is where you demonstrate what the engagement has delivered, honestly assess what's left, and reset the scope for the next quarter — and it's a milestone you should run deliberately, not let drift past. It's the moment the founders consciously decide the engagement is working, which for a fractional CTO is also the moment they decide to keep paying for it.

What the review covers:

Against the success criteria from before day one. Remember the question you asked the founder up front — "what has to be true in ninety days for this to be working?" This is where you answer it. Walk through what you committed to in the 30/60/90 plan and what actually happened, plainly: what landed, what's in progress, what changed and why. Measuring against the criteria *they* set is what makes the review credible rather than self-congratulatory.

Honest about what didn't happen. Something on the plan slipped — it always does. Name it, explain why, and say what you're doing about it. A review that claims everything went perfectly is a review nobody believes; the one that owns the misses is the one that builds trust for the next quarter. The 70/30 Engineer (https://rogerstringer.com/guides/the-70-30-engineer) guide's ownership principle applies to leadership too: you own the outcomes, including the ones that didn't land.

The reset — what the next quarter looks like. The first ninety days were about understanding, stabilizing, and proving value. The next quarter is usually about the deeper work the mandate now permits. Reset the plan: a new 30/60/90, informed by everything you've learned, with the founders' input on priorities. The engagement isn't a one-time setup; it's a rolling cycle of plan, execute, review, reset — and the 90-day review is the first turn of that wheel.

The relationship check. Implicitly or explicitly, both sides are deciding whether to continue and on what terms. Is the scope still right? Has the need grown or shrunk? For a fractional engagement, is two days a week still the right amount, or has the company outgrown it (toward full-time) or stabilized past it (toward advisory)? Better to have this conversation openly at ninety days than to let a mismatch quietly sour over the following months.

Done well, the 90-day review converts a trial into a relationship. The founders have seen you understand their business, deliver real improvement, communicate honestly, and set up a team that runs. That's the foundation everything after is built on — and it's exactly what the first ninety days were for. The last chapter gives you the templates to run all of this.

Templates & Checklists

Here are the artifacts to run the playbook — copy them, adapt them, make them yours. They're deliberately lightweight; a template you'll actually use beats a comprehensive one you won't.

Scope-of-work one-pager (before day one)

```
# Engagement: [Company] — [Fractional/Full-time] CTO
- Time commitment: [e.g. 2 days/week]
- Scope: [what you own vs. advise on]
- Authority: decide alone […] / decide with founder […] / advise only […]
- Reports to: [name]; cadence: [weekly 1:1 + monthly board]
- 90-day success = [the founder's own answer, verbatim]
- Term & review: [length], reviewed at 90 days
```

Listening-tour question set

```
Founders/CEO:
- What is the business trying to do this year? What's the constraint?
- What worries you about engineering? What would "fixed" feel like?
Engineers (1:1):
- Walk me through how a change gets to production.
- If you could fix one thing about how we build, what?
- What's everyone afraid to touch? What should I NOT break?
- Who really understands [critical system]?
Adjacent (product/sales/support):
- Where does engineering's output let you down?
- What takes too long? What breaks too often?
```

30-day diagnosis outline

```
1. Current state (plain language, the good and the bad)
2. Top risks, ranked (3–5, each with business impact)
3. Load-bearing / do-not-break list
4. The short list: 3–4 things that matter most this quarter
(Keep to a few pages. Lead with impact. Separate fact from recommendation.)
```

30/60/90 plan skeleton

```
Days 0–30  — Understand & stabilize: [items, owners]
Days 30–60 — Prove & prepare: [quick wins + groundwork, owners, dates]
Days 60–90 — First deep work: [the substantial improvement, owner, target]
Each item: outcome-framed, owned, dated. Plan to ~70% of capacity.
```

Red-flag checklist (watch for in the first 30 days)

```
- [ ] One person is the only one who understands a critical system
- [ ] No tests, or tests nobody runs / trusts
- [ ] Deploys are manual, scary, or rare
- [ ] No backups, or backups never tested (see The Boring Stack guide)
- [ ] Security basics missing: secrets in the repo, no access control
- [ ] Team morale visibly low / quiet attrition
- [ ] Founder and you disagree on scope or authority (fix immediately)
- [ ] "We'll rewrite it" is the team's answer to every problem
- [ ] No one can explain how a change reaches production
```

90-day review template

```
1. Success criteria (from day zero) — met / partial / missed, honestly
2. What landed, what's in progress, what slipped (and why)
```

```
3. What I learned that changes the plan
4. Reset: next-quarter 30/60/90
5. Relationship check: is scope/commitment still right?
```

That's the whole playbook, from the agreements before day one to the review that resets for the quarter ahead. The pattern underneath all of it: understand before you act, prove value before you ask for the hard mandate, communicate in outcomes, and build a team that runs without you. Do that in the first ninety days and you don't just survive the engagement — you earn the one after it.

This playbook is the deep version of a chapter in the Fractional CTO Field Guide (https://rogerstringer.com/guides/the-fractional-cto-field-guide); for assessing a system you're inheriting or buying, pair it with Technical Due Diligence (https://rogerstringer.com/guides/technical-due-diligence).

About Roger

I'm Roger Stringer — I build things, break them, and write up what I learned so you don't have to learn it the hard way. These Field Guides come straight out of that work.

Working on something bigger? I take on a handful of fractional CTO (https://rogerstringer.com/guides/the-fractional-cto-field-guide) engagements — helping founders and teams set technical direction, build AI-powered workflows, and actually ship the hard parts. If you're wrestling with the kind of problem this guide covers and want someone in your corner who's done it before, that's exactly what I help with. Drop me a line. (mailto:roger.stringer@hey.com)

And if a guide helped, got something wrong, or you just want to compare notes, I'd love to hear from you:

- **Email** — roger.stringer@hey.com (mailto:roger.stringer@hey.com)
- **X** — @freekrai (https://x.com/freekrai)
- **GitHub** — github.com/freekrai (https://github.com/freekrai)
- **LinkedIn** — linkedin.com/in/rogerstringer (https://www.linkedin.com/in/rogerstringer)

New guides go up as I hit problems worth documenting — follow along wherever suits you.