The Command Center

What it is, and how it creates an operating layer for your business

Generated 17 Jun 2026, 4:50 pm

An explainer for James, from our conversation. Sam McKay, Enterprise DNA.

WHERE THIS CAME FROM

We got onto the operating layer the other day and I said I would write down what I actually mean by it. This is that. It is not a pitch. It is the plain version of how I run several businesses from one place, so you can decide whether the same structure is worth standing up where you are heading.

A Command Center is the thing that creates the operating layer. So I will start with what it is.

WHAT A COMMAND CENTER IS

It is a folder. You own it, you open it in Claude Code, and it becomes the cockpit you run the business from. Nothing to log into, no new app. Just a folder with a particular structure, and an AI worker that reads that structure and does the work inside it.

The reason it works is that three things live in the folder, not in a chat window:

1. **Instructions.** One file the worker reads at the start of every session. It says who you are, how you want work done, and the one right way to do each recurring job. You write it once and it applies every time.
2. **Memory.** The durable facts about your business, written down once so the worker never has to be told twice. Who your customers are, how you like things, the decisions you made last month.
3. **Pipelines and agents.** Your recurring jobs saved as named recipes, and longer-running workers that handle a whole task across your systems.

A normal chat session is clever but it forgets, and it cannot improve. A Command Center does both, because the knowledge lives in the folder. That is the whole difference between using AI ad hoc and running a business on it.

HOW IT CREATES AN OPERATING LAYER

The operating layer is the worker sitting on top of your systems and doing the repeatable work against them, while you stay in command of the decisions.

For thirty years software assumed a human sits at the screen and clicks through the work. The operating layer flips that. You stop operating ten tools by hand. You operate one worker that reaches the ten tools for you. You describe the outcome. It does the legwork between your decisions and brings the work back done, or drafted, for your yes.

The skill you build is not technical. It is the skill of being a clear director. You decide what matters and where the line is. The worker does the rest.

CONNECTING IT TO YOUR REAL DATA

This is the part that makes it more than a writing assistant, and it is the part worth getting right.

The folder on its own only knows what is in the folder. To make it useful for a real operation you connect it to where your business data actually lives: your existing database, your data warehouse, or your data lake. Once it can reach that, it can answer real questions and build real reports off live numbers instead of guessing.

The way you connect it safely:

- **Give it a read-only credential, not the keys to the kingdom.** You create a read-only user on the database and hand the worker that. It can read everything it needs and change nothing.
- **The credential lives in the environment, never in a file you commit.** It sits in a local `.env` file that is kept out of version control. The worker reads it at run time. It never gets pasted into a document or a chat.
- **Queries are parameterized and specific.** The worker does not run loose, open-ended commands against your data. Each connection is a defined pipeline: a named, repeatable way to ask one kind of question, written down and reviewed once.

A pipeline to your data can take two shapes, and often it is both:

- **A skill.** A saved recipe. "Pull this month's revenue by client" becomes one named job that runs the same query the same way every time.
- **An agent.** A worker that strings several steps together. "Read the pipeline, find the deals that stalled, pull each one's history, draft a follow-up" is an agent, not a single query.

Skills are the single, repeatable moves. Agents are the multi-step jobs. You build them up over time, and together they become the secure, defined set of ways the worker is allowed to touch your data. Nothing ad hoc, nothing that writes when it should only read.

HOW YOU RUN IT: THINK LIKE A DEVELOPER

The thing that turns this from a personal trick into a firm-wide asset is one decision: you run it like a developer, with GitHub as the central source of truth.

Here is what that means in practice. The Command Center is a folder, and a folder can be put under version control. You push it to a private GitHub repository. Now:

- **Everyone in the firm runs the exact same Command Center.** Same instructions, same memory, same skills and agents. One person improves a job, commits it, and everyone else pulls it. The operating knowledge of the firm compounds in one place instead of living in ten separate laptops.
- **Every change is tracked.** You can see who changed what and when, and roll back if something was wrong. For a firm that cares about control and record-keeping, that matters.
- **The routing table and the safety rules are shared and enforced for everyone**, not re-invented per person.

This is the bit people skip, and it is the bit that makes the difference at the level of a firm rather than an individual. The operating layer becomes a shared, versioned asset. You are not deploying a tool to people. You are giving them one structured place that already knows how the firm works.

WHERE YOU ACTUALLY RUN CLAUDE CODE

The structure is what matters. The surface you run it on is preference.

I run Claude Code inside Cursor, an editor. It lets me keep the files, the chat, and the work side by side, and it is simply faster for how much I iterate. That is a "how Sam does it" detail, not a requirement. You can run the exact same Command Center from the Claude Code desktop app, or straight from the terminal. Same folder, same worker, same result.

So do not get hung up on the surface. Get the structure right first. The structure is what the template gives you.

THE PART THAT KEEPS IT SAFE

This is the question a serious operator asks first, and rightly. If the worker acts on its own, what stops it acting wrongly?

The answer is a written line, and it is the most important thing in the folder. The worker does the work in the middle. It stops at anything that sends, spends, deletes, or faces a customer. Those it hands back as a draft or a proposal. It drafts the email, you send it. It proposes the invoice, you approve it. The data connection is read-only by design, so it cannot change a record even if asked.

Review is a dial, not a switch. On a brand new job you check everything. After the worker has done it well twenty times you spot-check. The glance never goes to zero, because the cost of an unwatched job going wrong is real. The perimeter is written down once, in plain language, and the worker holds it on every run. The lines are yours to set and yours to move as the evidence comes in.

HOW TO START WITH THE TEMPLATE

I have put together a clean, industry-neutral version of this, set up the way I run mine. It is the zip file that came with this note. The structure is the point, not the contents.

What to do with it:

1. **Download and unzip it** onto your machine.
2. **Open the folder in Claude Code** (in Cursor, the desktop app, or the terminal, whichever you prefer).
3. **Point Claude Code at it and ask it how it could run the business.** Genuinely. Ask it: "given this structure, how would you use this to run the different parts of my business?" Walk through its answer. That conversation is where it clicks, and you will learn more from ten minutes of that than from anything I can write here.
4. **Fill in who you are.** Five honest minutes in the business file changes every answer from generic to yours.
5. **Hand it one real, boring, repeating job.** Watch it come back done. That first handoff is the moment it stops being a chatbot and becomes a worker.

It is a starting point, not a finished system. The point is to get the shape in your hands.

A CALL, IF IT IS USEFUL

If you like what you see, this is a good place for a call. I am happy to walk you through how I actually run it day to day, including the Cursor setup and how I wire the data pipelines, and to talk through what it would look like for the firm you are joining. No pitch. Just a proper look under the hood, founder to founder.

Either way, good to reconnect, and genuinely pleased about the move.

Sam McKay Founder, Enterprise DNA sam.mckay@enterprisedna.co.nz enterprisedna.co