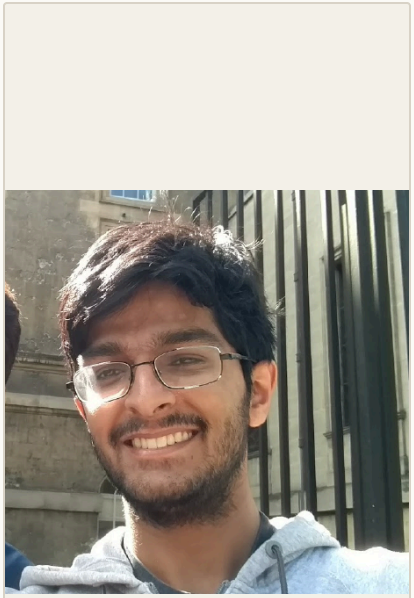
Nori Agentic

Software *Factories*



Intro

CEO of Nori Agentic. Nori is your software factory factory. We make it so you can run agents in the cloud and automate all the grunt work out of your company.







Why this talk?

Whenever you get in a room to listen to someone speak, you should ask: why this talk? And as the speaker, my job is to explain why I deserve 10 minutes of time.

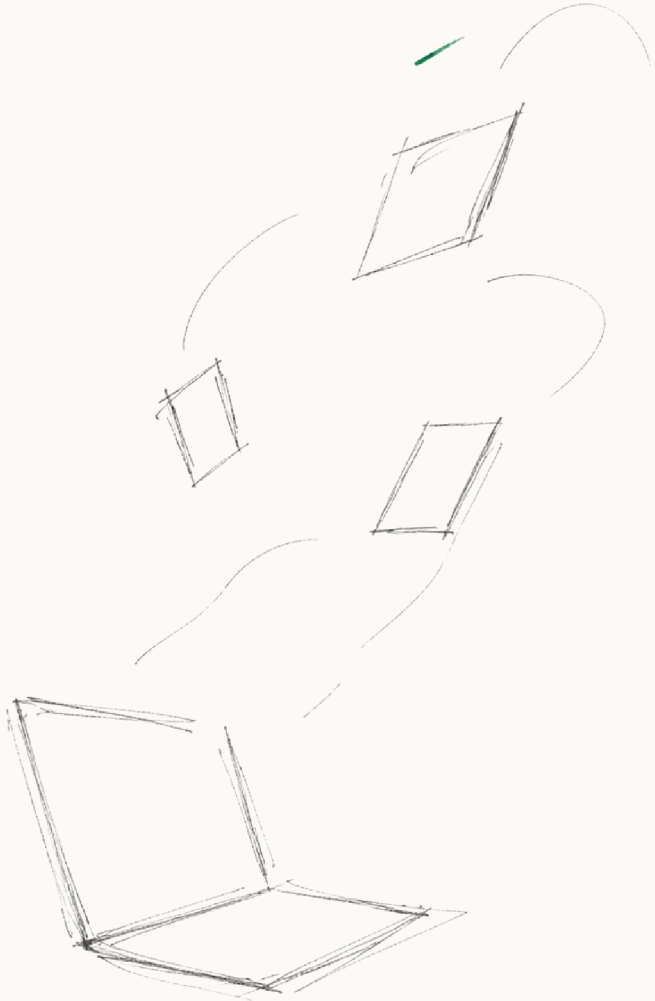
The short answer is, I feel like I've seen the future.

Nori started life as a lab, trying to figure out what it means to run an AI native business. We built one of the first open-source TUIs built on ACP. We built a powerful skills manager to let a whole team share and switch skills. We built infra to run agents in the cloud. And with that last one, the company totally transformed.



We use cloud agents to write 90% of our code and do 80–90% of our ops — it's not an AI employee, it's, without exaggeration, an AI army.

So this talk is important because it's a way to bring the future to the present. My hope is that at the end you'll understand a bit more of the 'art of the possible'.

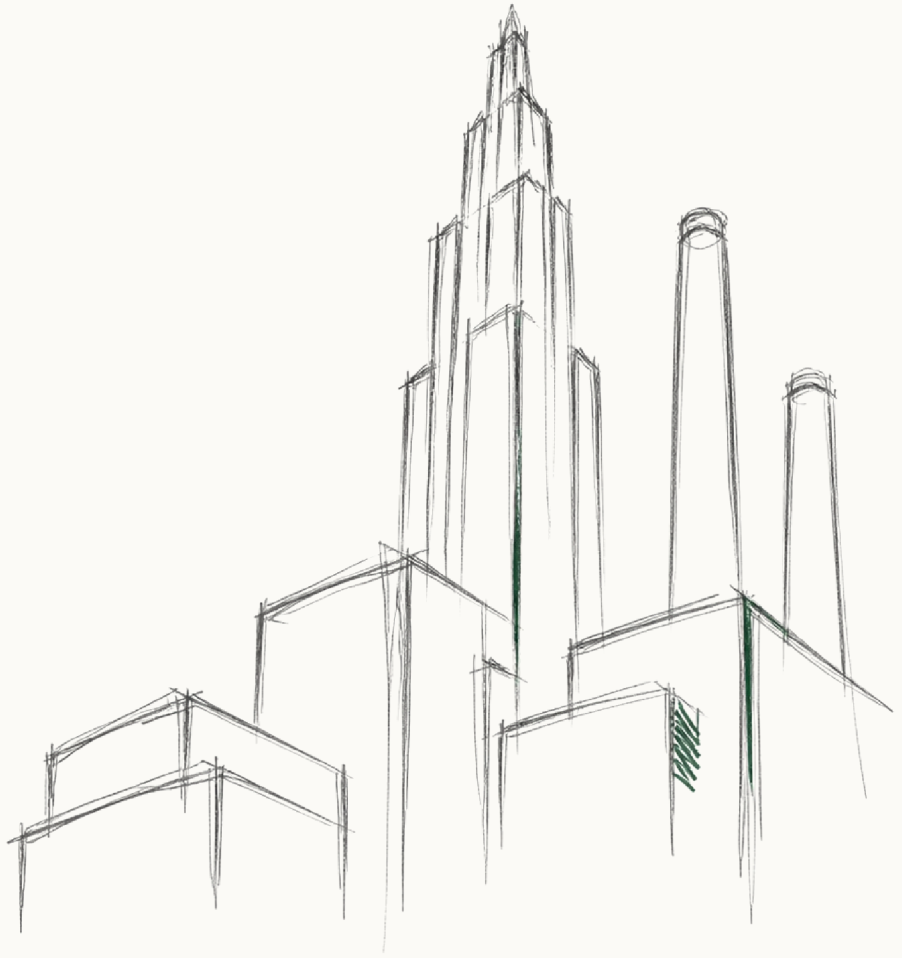


Agenda

Here's how we're going to accomplish that goal. We're going to deep dive on this new buzzword, and walk through:

- What a software factory is
- How to get started building one
- How to avoid common mistakes that grind the gears to a halt

What is a Software Factory?

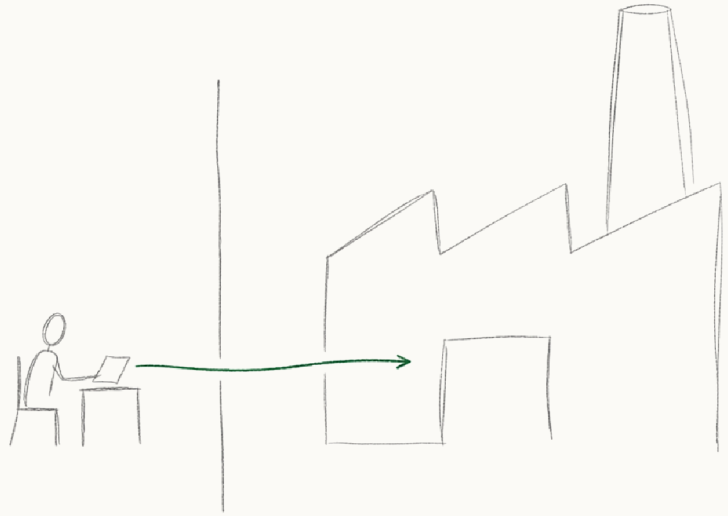


History of the term

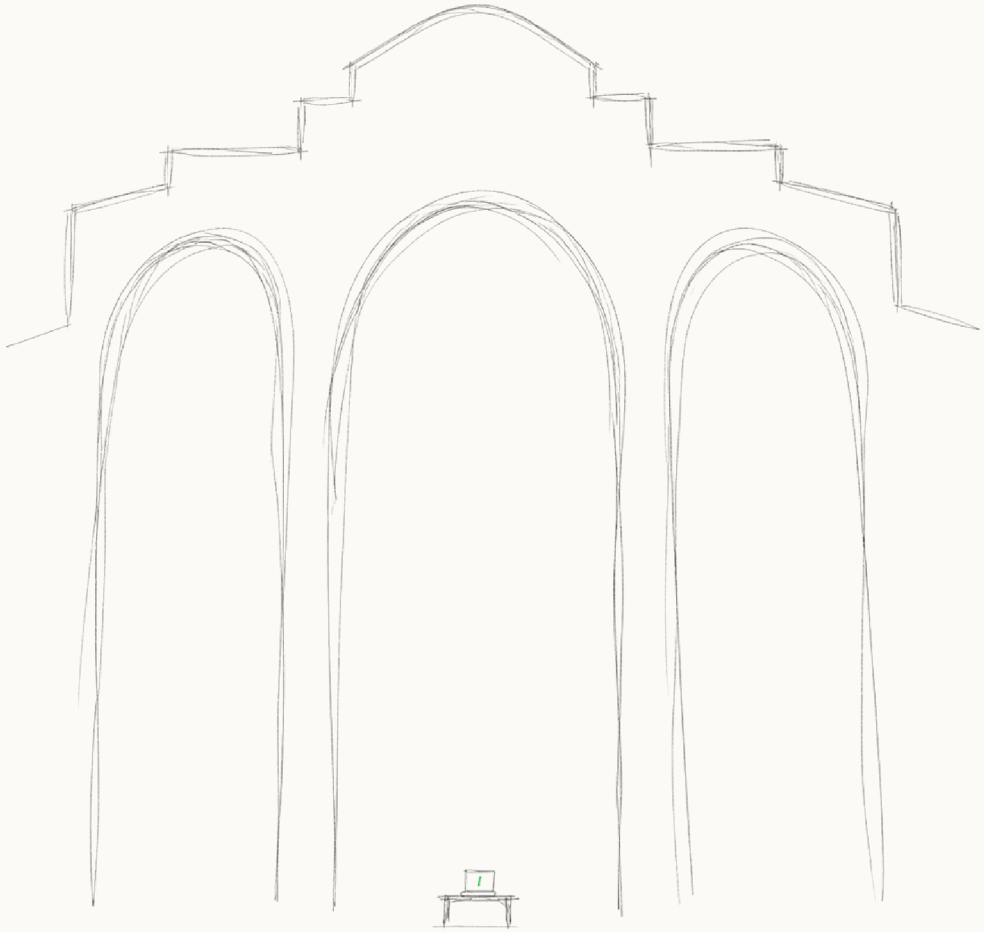
The term originates in operations research, but was repurposed earlier this year by a company called StrongDM. They wrote:

> *Humans define intent: what the system should do, the scenarios it needs to handle, the constraints that matter. After that, the agents take it from there – they generate code, validate it against real-world behavior, and iterate until it converges, without hand-tuning or human review.*

The point of the software factory, as defined by StrongDM, is to pull the human out of the process of doing.



But over time the term degraded, became more about marketing. Now you'll talk to someone, and they'll be like "I have a software factory" and you'll get all excited and then it turns out they're just using Claude Code. So what is a 'real' software factory, separate from the hype?



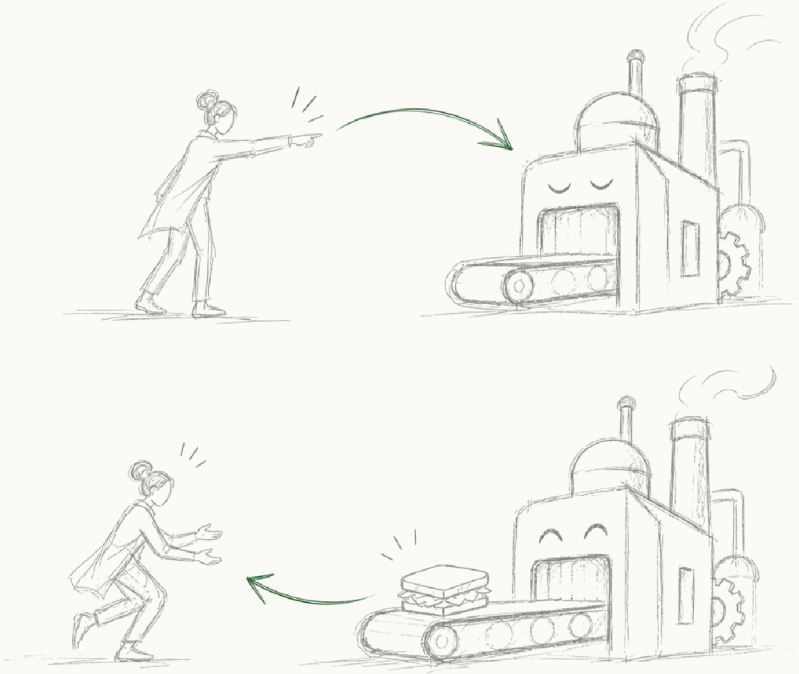
Software factories change your company from being reactive to proactive

REACTIVE

When you use Claude, it's fundamentally reactive. You have to put in something for it to do anything. *"Claude, make me a sandwich."*

PROACTIVE

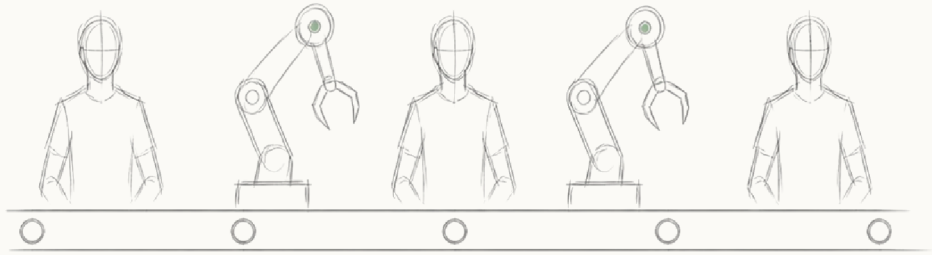
My thesis is that an effective software factory is fundamentally proactive. It comes to you with a sandwich. *"Hey, you're hungry, I made you this."*



This is the massive shift.

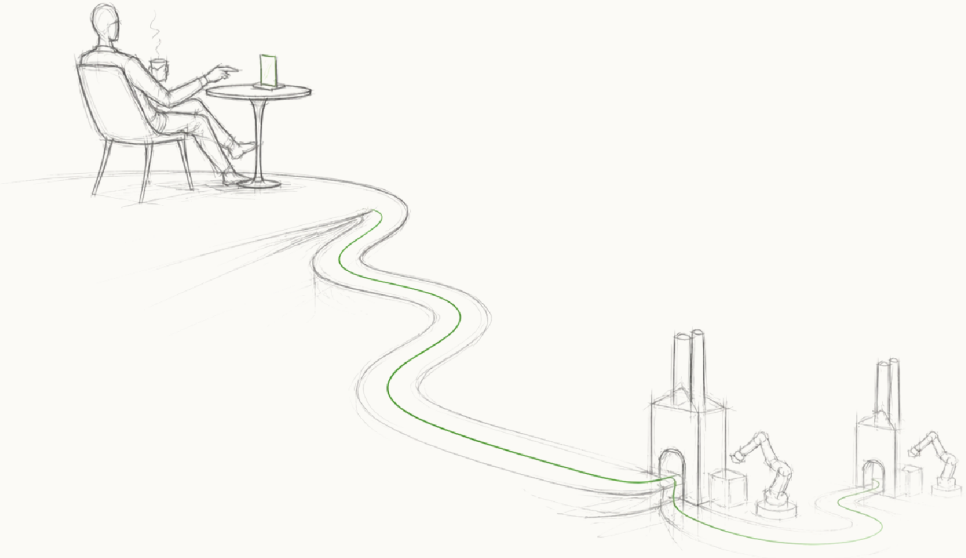
The software factory is less about agents, and more about the relationship between people and agents.

The agents give me the work, instead of the other way around.



Impacts

We ship more. We ship better. But the main thing I want to convey is how much this reduces stress. If you set up your loops right, suddenly you're on rails. You don't have to remember, wait, shit, did I do that thing?



How to build it

Hopefully I've convinced you these are worth building. So how might you go about it?

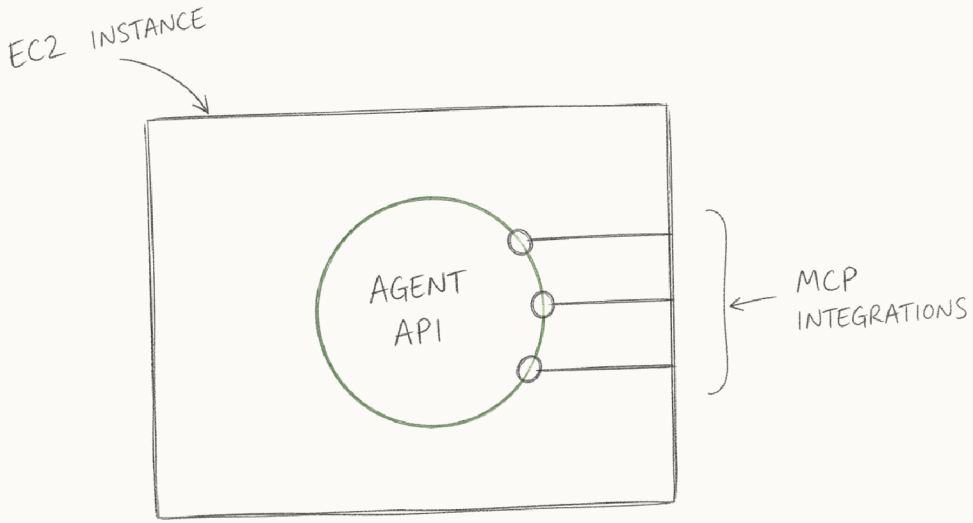


Let's start with a single automation

An agent that, every day, gives me a summary in Slack of what happened in the company GitHub.

You can do with with an EC2 instance, an agent API, and some MCP integrations.

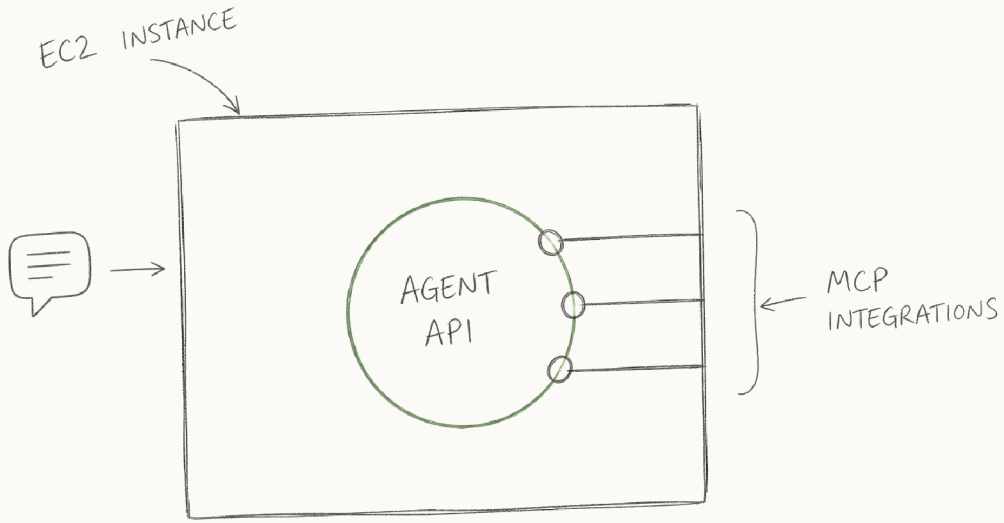
For this first basic automation you don't even really need an outward facing server.



Let's add another!

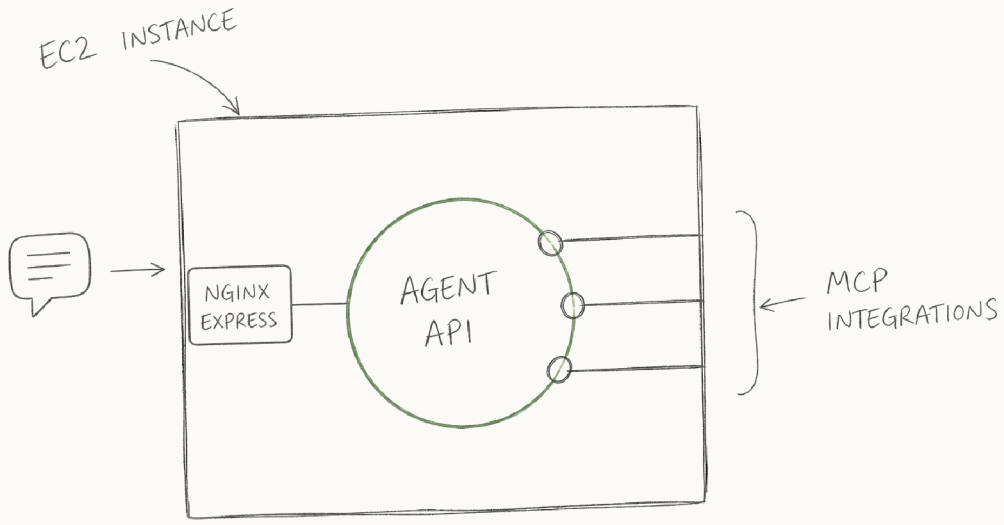
Say we want to add another automation. We want to have a Q&A system where people can ask questions about the product from Slack.

Immediately, there are a lot of interesting problems that didn't pop up with the first one.



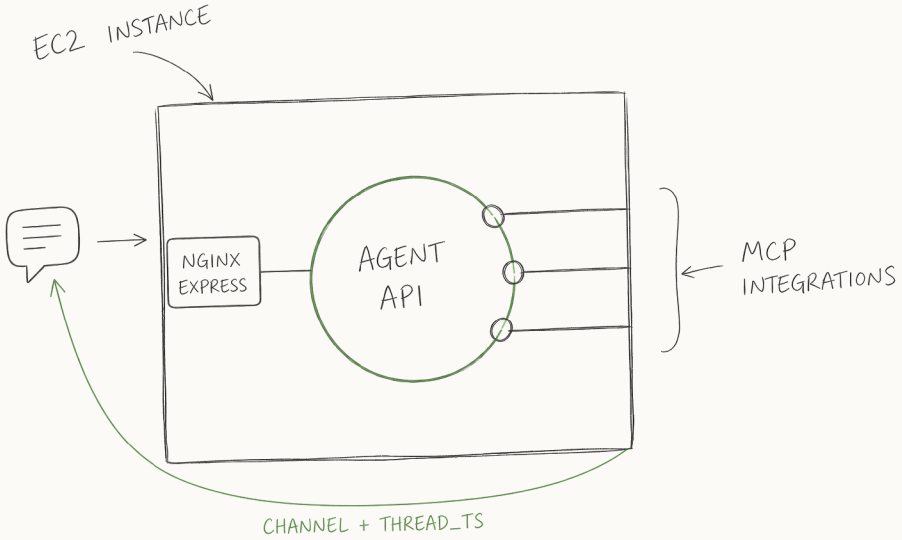
Now we do need some sort of server. So you run a TypeScript Express app on EC2 behind a reverse proxy like nginx, and configure the EC2 security group to accept HTTPS requests.

When a request comes in, it hits the server. The server spins up a Claude Agent SDK session with the request prompt.



You'll need a Slack app set up, but let's say you have that already. The request needs to carry the channel and root thread timestamp (`thread_ts`), so the agent knows where to post the result.

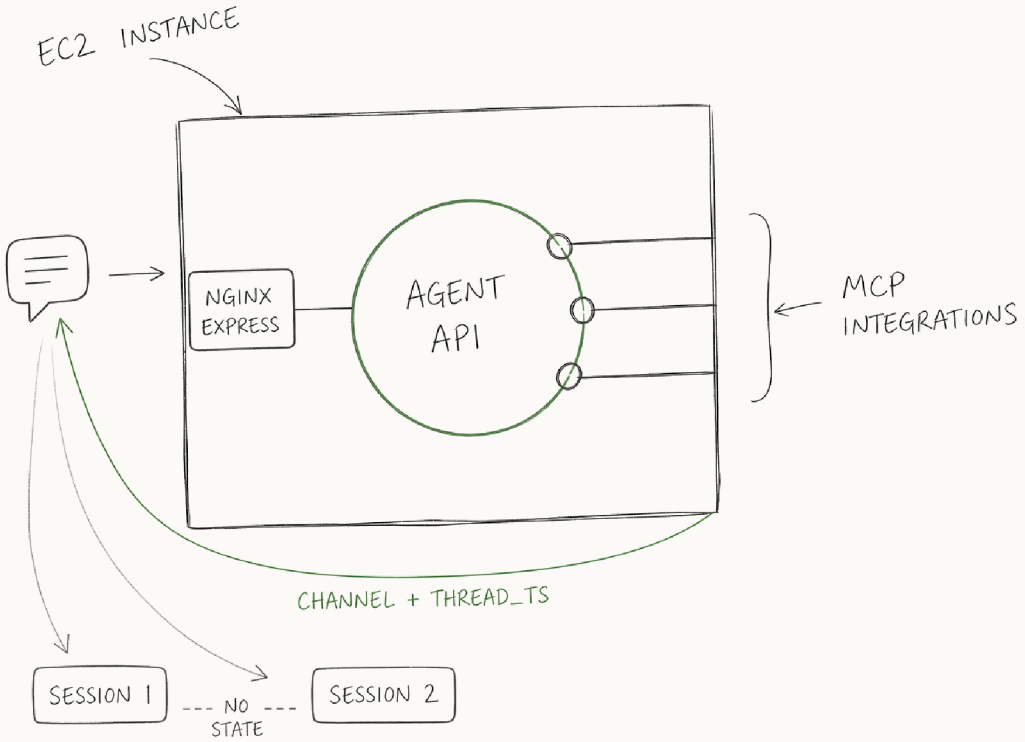
Those and the prompt get passed to the Claude Agent SDK. Claude looks at GitHub and Slack — which we already had before — and uses the channel and thread timestamp to respond.



Let's add another!

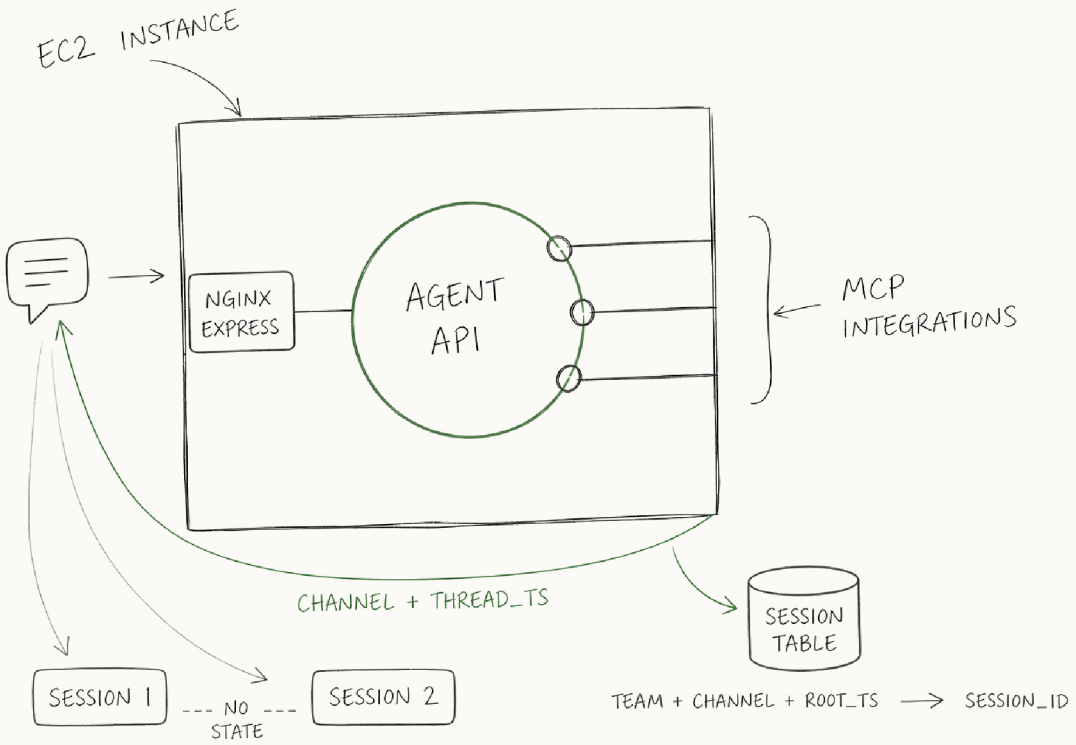
Your team wants to know why, every time you talk to the agent in Slack, it doesn't remember the previous conversation. There's no state!

So what if we wanted to make these resumable?

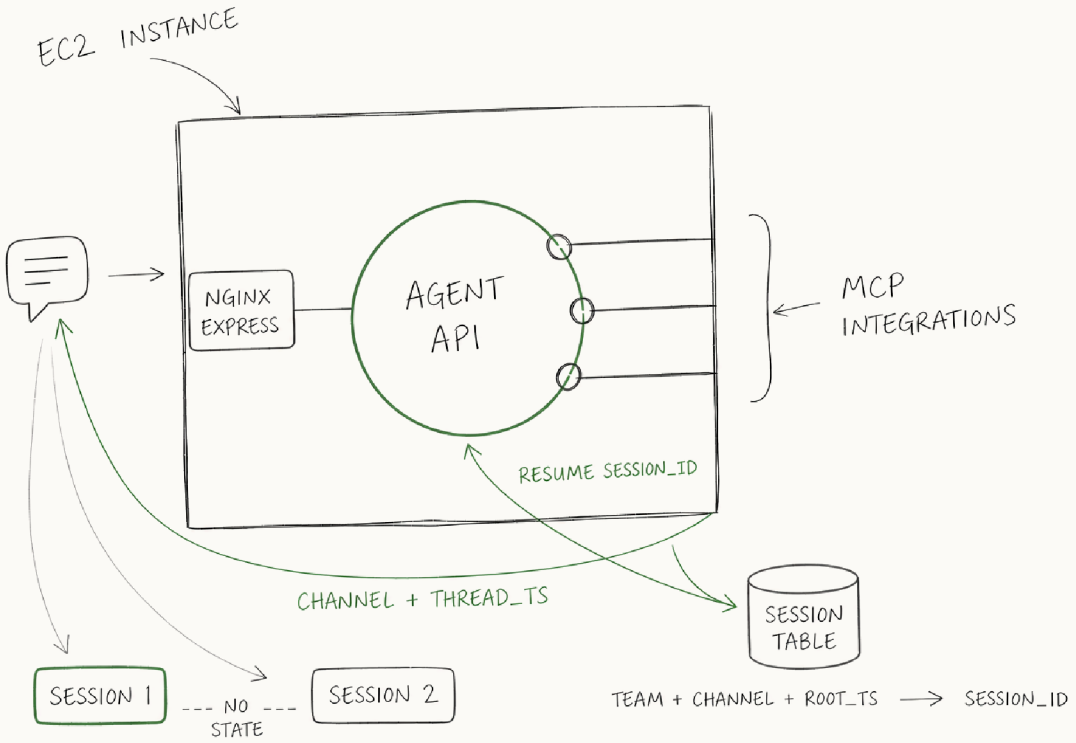


One way is to add a light database to the EC2 instance. Every time a server request comes in, it stores a unique ID in this database — maybe the Slack thread timestamp.

The Claude Agent SDK gives you a session ID, so you map the Slack workspace, channel, and thread timestamp to the Claude session in this table.

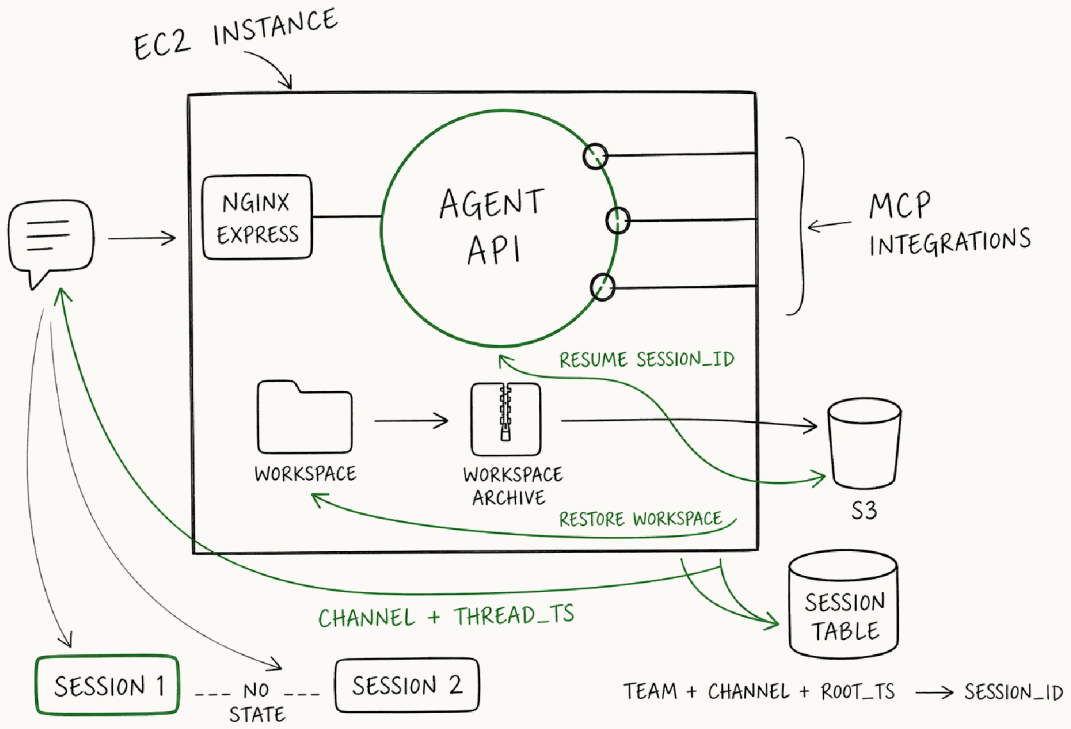


In the future, when a new request comes in, the server checks the table first. If the thread is already there, the server resumes the Claude Agent SDK session instead of starting from scratch.



Right now we're assuming the Claude Agent SDK is handling the full resume logic. But the SDK only handles the transcript. What if you want to capture changed files or things like that?

The simplest way to handle that is to zip up the whole workspace and store it, then restore it when needed. So you need an object store like S3 too.

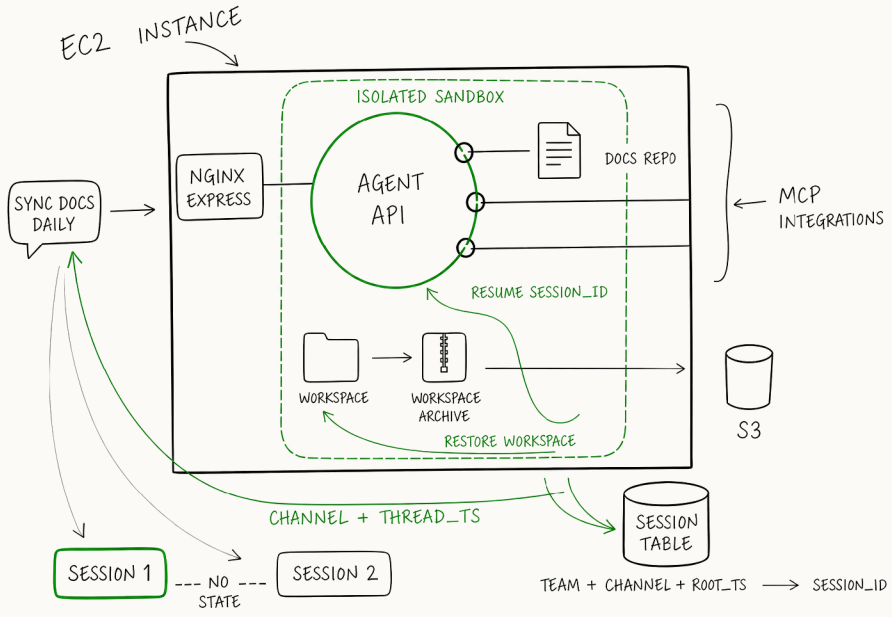


Let's add another!

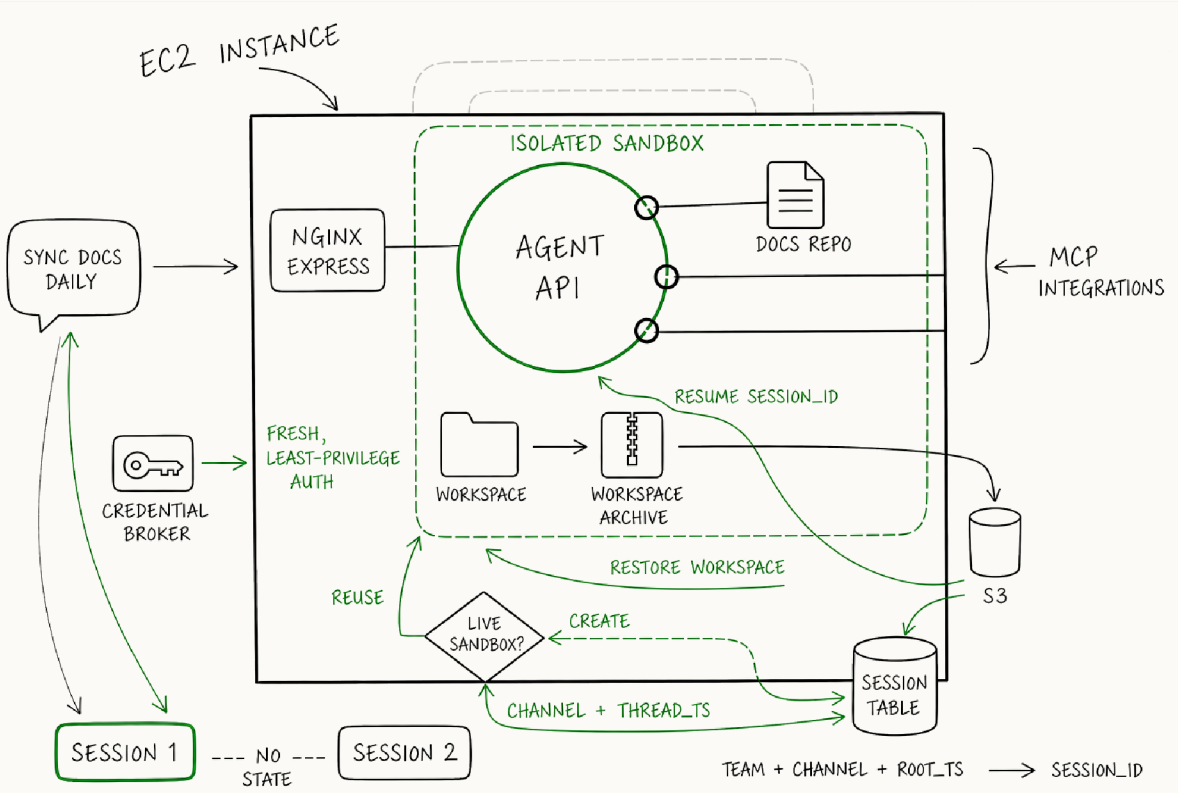
Lets say you want to automatically keep your docs up to date. Every day you want an agent to go through and make sure that the external docs match the internal repo.

This is the first thing we have that is actually writing to something. So now we need to make sure these agents aren't stepping over each other.

One way is to automatically set up git worktrees, but you probably want more isolation than that. Whenever the server spins up, it creates an isolated folder or a sandbox — maybe even a Modal Sandbox.



We didn't talk about this before, but you have to inject valid, least-privilege credentials into all these different sandboxes — especially if they're running on different machines — and make sure your scopes are right.



Let's add...another?

This is all pretty hard! And there's way more:

- Ensure proper scoping and RBAC
- Control cost and token budgets across many runs
- Deduplicate events so retries don't repeat side effects
- Make sure your system cant get agent-jacked
- Coordinate dependencies between automations
- Handle queues and backpressure when triggers spike
- Audit what every agent did and why
- Decide retention rules for transcripts and artifacts
- Route notifications to the right people
- Handle integrations that don't support MCP
- Detect and quarantine bad automation output

Get an infra provider

This is the brief sales pitch part: most teams don't really want to own all that crap.

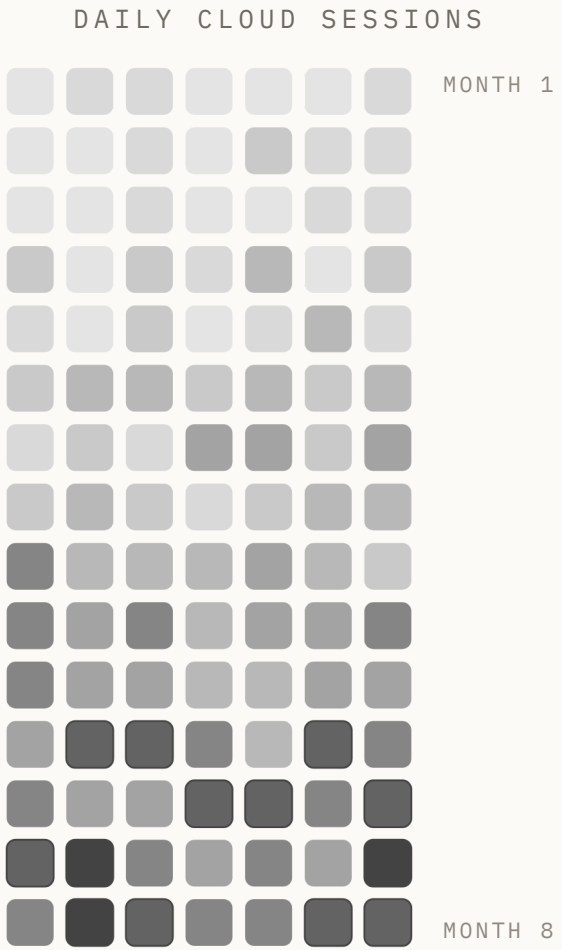
So get a vendor to do all the boring stuff.



`start a free trial → noriagentic.com`

How do you build a sustainable factory?

Alright, say you're using Nori or you've built your own. Let me share a bit of scar tissue, things we've learned from 8 months of running our own factory.

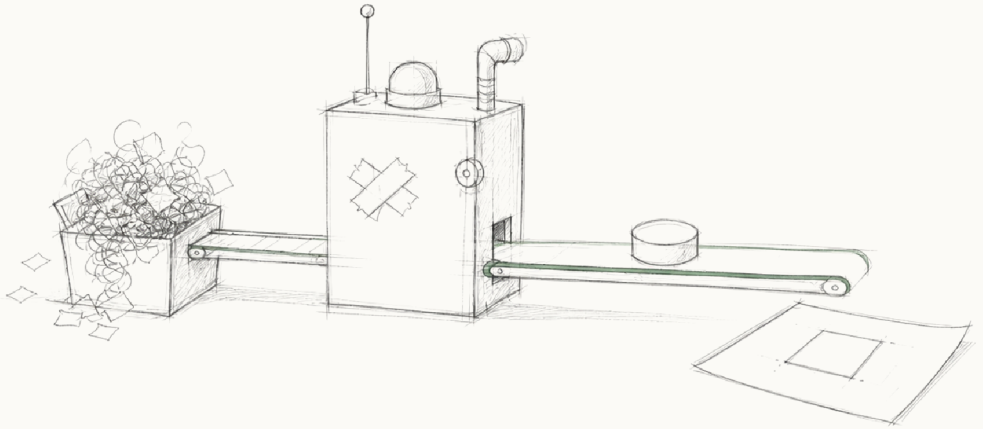


The big problems with long running factories:

Noise: your factory produces a bunch of garbage you have to sift through — the agents aren't perfect and may lack context, capability to verify, or (big one) taste.

Wear and Tear: it gets worse over time, because agents do not like deleting things, ever. They keep building up cruft that destroys their context.

Drift: the factory is producing widgets, but now we need bobbles. Building things is way less important than knowing what to build.



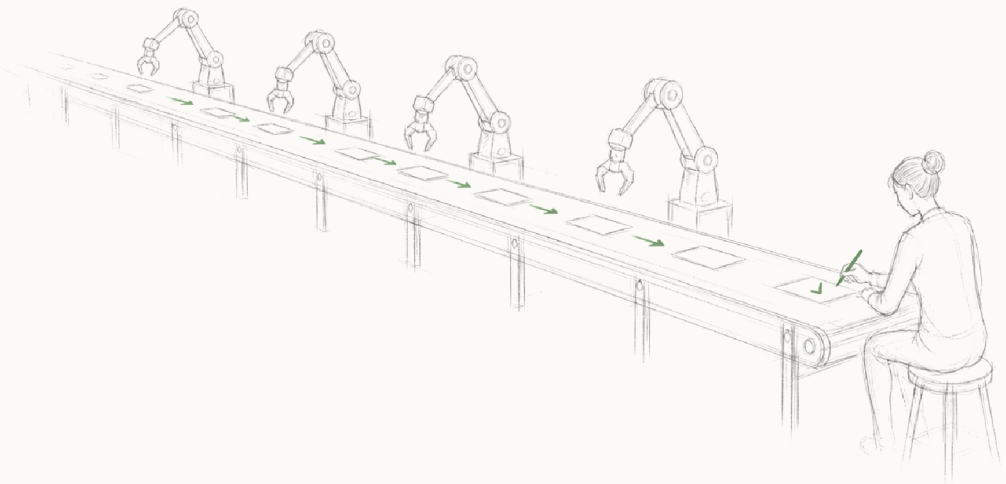
NOISE WEAR DRIFT

Creating a sustainable loop

If there's a single piece of advice I have, it's: don't let perfect be the enemy of the good. People want AI to do 100% of the work. Look at the original StrongDM definition — "without human review." No.

Let the AI do what it's good at — 95% of the work — and put a human in the loop for the last five percent.

That includes for 'agentic memory' -- agents are bad at knowing what to remember.



95% AI **5%** HUMAN

- Don't be afraid to split a long task into alternating AI and human steps, based on what each is good at.
- Focus on verification processes. Use tmux for TUIs and playwright for web. Build dev environments.
- Think about high level organization / swim lanes -- where code goes, plugin architectures, etc.



AI DRAFT | HUMAN REVIEW | AI REVISE | HUMAN APPROVE | AI SHIP

- At the team level, someone needs to own the factory. They need to keep it clean.



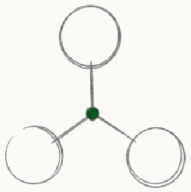
The future

When you use an AI on your computer, with you driving, it's very much just a standard tool. Powerful, but mostly about moving in and around existing structures.

Cloud agents are very transformative to how a company works, because they change the company structure.



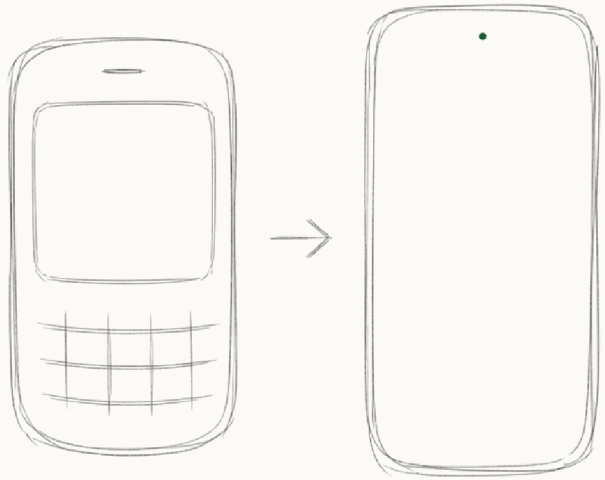
TOOL



SYSTEM

It's a bit like a BlackBerry vs. an iPhone. The BlackBerry is stuck in the world of desktop computers. Its abstractions are all desktop abstractions, down to the clunky keyboard. An iPhone is ~the same technology, but is a 'mobile native' experience.

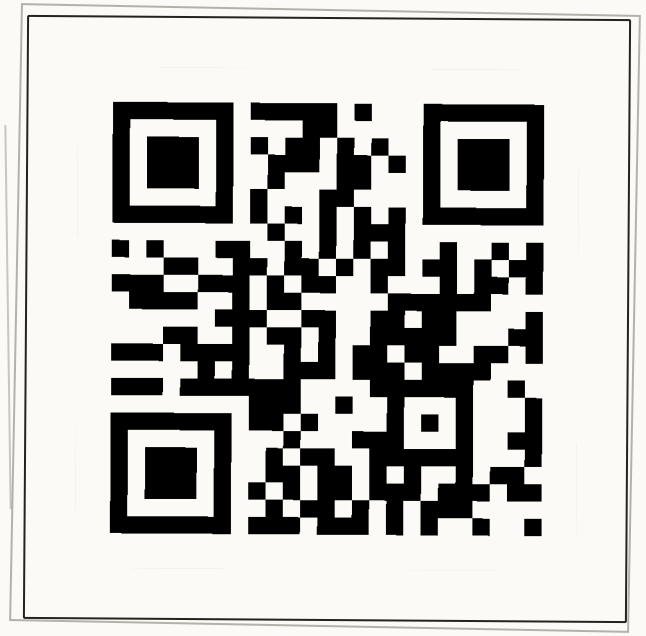
It's the same here. Agents on your laptop are powerful, like a BlackBerry is powerful. But agents in the cloud are transformative the way an iPhone is transformative.



Nori Agentic

Your *software factory* factory

PROMO CODE **AGENTICSNYC**



noriagentic.com

Created with Nori Agentic

Appendix

The vision is something like an assembly line. You want to make a widget. Instead of making the widget, you have a meta process, which constructs the assembly line. And then the assembly line makes the widget.

Instead of a bunch of robot arms, though, you have agents.



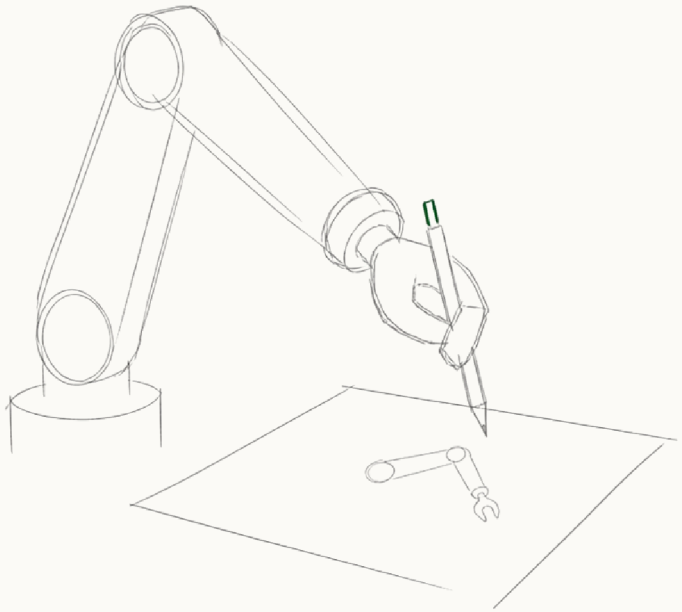
So you'd have a research agent. And a plan agent. And a test-case writing agent. Then you'd write code. And then you'd have some kind of evaluation agent. People built really complicated frameworks to meet this vision of the assembly line.

But this is kinda silly, isn't it? You don't need seventeen different agents to do things. You actually only need one skill file. You can scan that QR code to get a skill file that will do all that.



An open source skill that does the whole framework

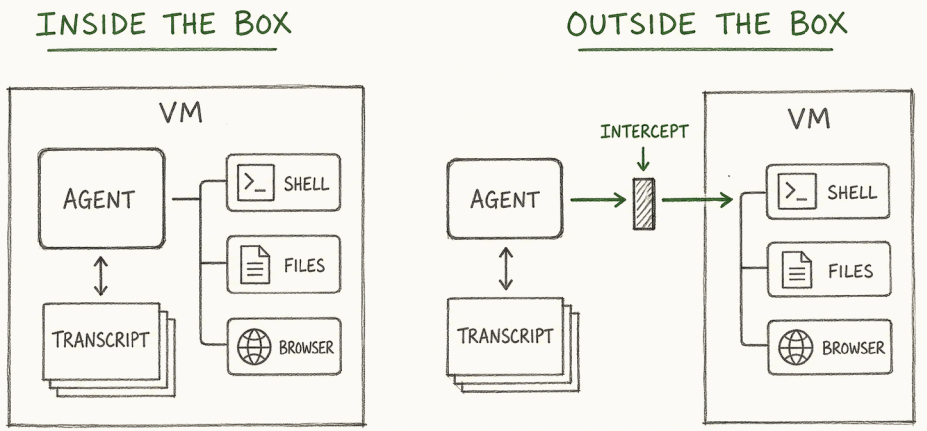
And you can get pretty meta about this. For example, we have an automation that looks through things we do regularly in our codebase, and proposes automations to make them more consistent.



So far, we've assumed a standard Agent SDK is running on the machine. But there are other ways to build this. You can use a harness that runs inside the box, which is what we have now. Or you can build a custom agent that runs outside the box and manipulates the VM through tool calls.

You'd mostly do this if you care a lot about intercepting the agent's actions before they happen.

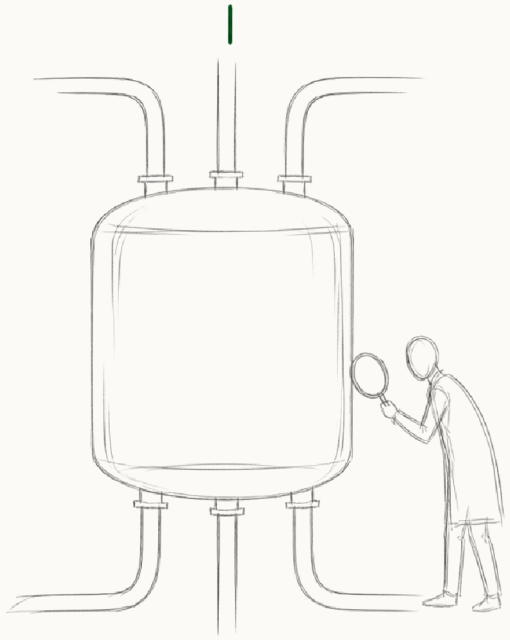
That completely changes how you think about resume: now the transcript lives outside the box too.



Example: applied to a master db

Say you have a Snowflake database meant to be a golden record for your organization. All the data has to be clean — that's the only reason this database exists. Everyone downstream requires it.

How do you keep it clean? Right now it's pretty manual: track down lineage, triage to the right people, evaluate correctness.

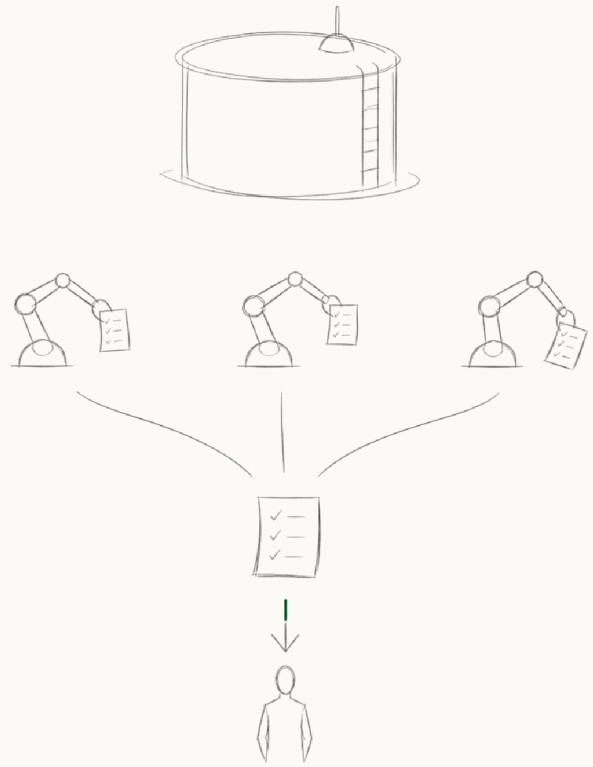


SOURCES MASTER DB HUMAN INSPECTION USE CASES

Or...use agents. Run a daily check of the data coming into the master DB. Evaluate every cell that changed.

For each cell, spin up an agent that runs an exact checklist for that column — what public sources to look at, or which contracts to find in Google Drive.

And finally, human in the loop: surface the change and the analysis to a person who now has everything they need.

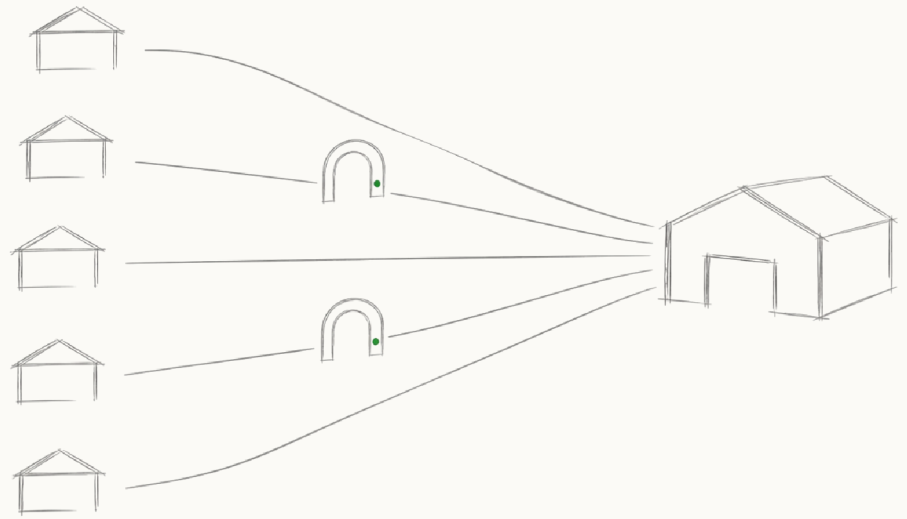


MASTER DB AGENT CHECKS REVIEW PACKET HUMAN

Example: applied to managing a CRM

In some sense a CRM is a master database we're all familiar with, but it generally sits far outside the data org. Our CRM is ~automatic.

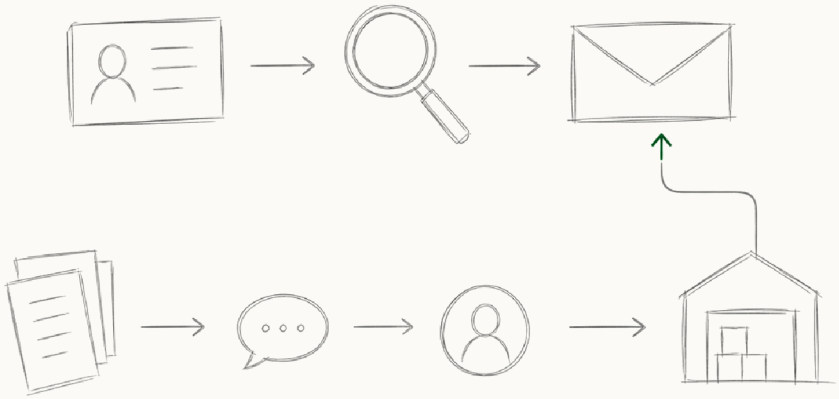
Every day, agents spin up and load subscriber lists from our newsletter, YouTube, Substack, LinkedIn, and Twitter. Some are behind login walls — but the agents can just sign in as actual users.



NEWSLETTER · YOUTUBE · SUBSTACK · LINKEDIN · X

CRM

Agents can do research about people in our CRM and decide who we need to reach out to. They update status themselves based on internal messages, so there's way less fiddling with HubSpot settings. Sellers just say yes/no in Slack and go back to selling.



CONTACT → RESEARCH → DRAFT

TRANSCRIPTS + SLACK → AGENT → CRM