

# Building AI factories that deliver.

Writing software got cheap. Running it inside a company did not. This paper describes an AI factory that turns requests into internal tools people use, and the foundation, floor and process it needs to ship them.

|              |                      |              |
|--------------|----------------------|--------------|
| TOPIC        | FOR                  | DATE         |
| AI factories | CIOs and heads of IT | October 2026 |

# A factory is judged by what it ships.

|    |                                                                                         |    |
|----|-----------------------------------------------------------------------------------------|----|
| 01 | <b>Writing software got cheap</b><br>What coding agents changed, and what they did not. | 03 |
| 02 | <b>A line, not a lab</b><br>The six stations a request passes through.                  | 06 |
| 03 | <b>The foundation decides what ships</b><br>What every tool inherits from underneath.   | 09 |
| 04 | <b>The factory floor</b><br>A Studio for every team, run one way.                       | 13 |
| 05 | <b>How the work comes through</b><br>Sudo and curio: agents build, a person approves.   | 16 |
| 06 | <b>Features per day</b><br>What our own changelog measured.                             | 19 |
| 07 | <b>Start with one tool</b><br>How to pick it, and what to ask.                          | 22 |

IN ONE PARAGRAPH

Coding agents made software cheap to write, so many companies now plan to replace bought tools with their own. Most of what a bought tool gives you is not its screens but what sits under them: data, permissions, hosting, upgrades, an audit trail. An AI factory delivers when those are built once, as a foundation, and every new tool is a small change on top that a person approves.

# 01

## Writing software got cheap

---

What coding agents changed, and what they did not

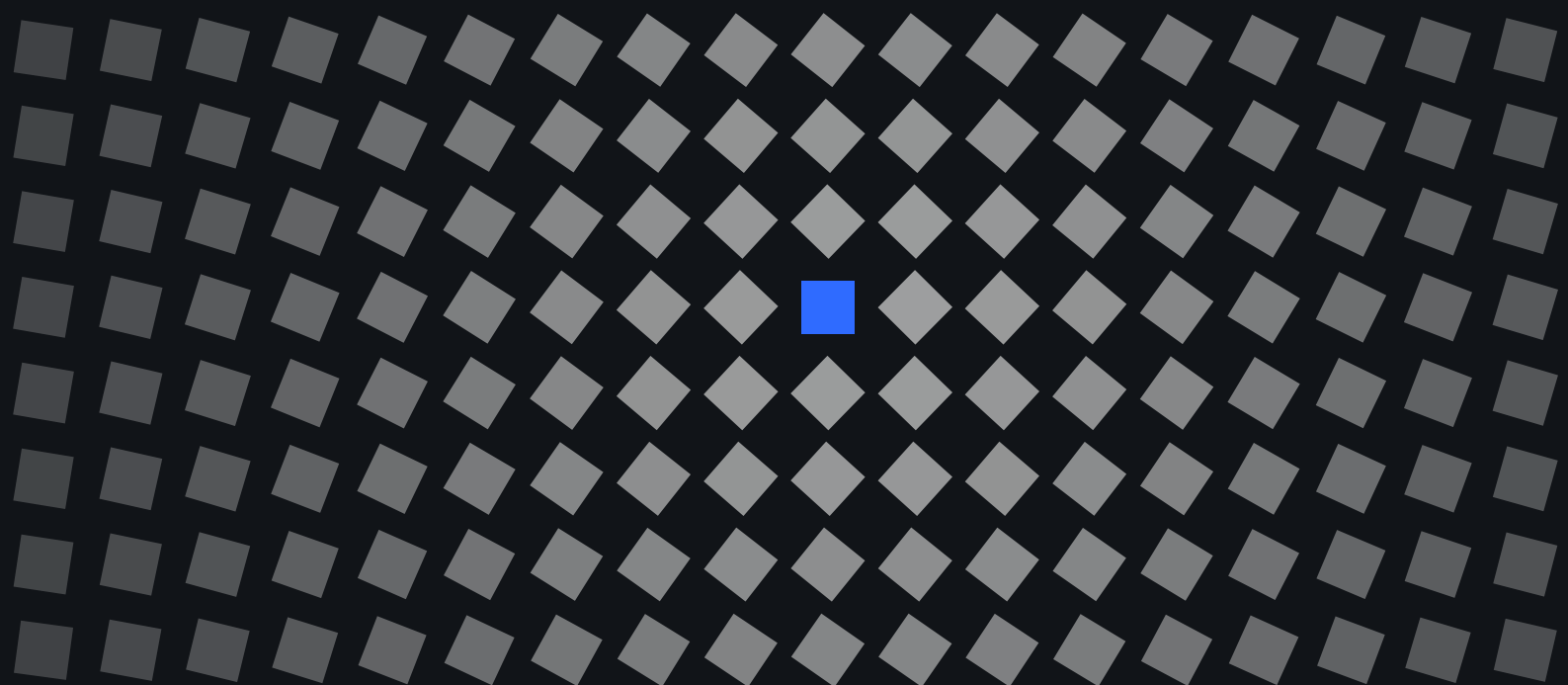
---

What a licensed tool bundles beside its screens

---

Which tools to rebuild, and which to keep

---



## 1.1

## The cost moved from writing to owning

# A coding agent writes a working screen in minutes. Who sees which record, where the data lives and who keeps the tool running cost what they always did.

In 2025 and 2026 coding agents became an ordinary way to write software. A developer describes a change, the agent edits the code, runs the tests and proposes the change for review.

That changes the arithmetic behind buying software. A licensed tool was cheaper than one of your own because writing software was slow.

Barry McCardel, who runs Hex, wrote in October 2026: "I think it is true that a lot of software categories will effectively die over the next couple of years." Many, he added, "will combine and be reborn into new shapes."

We agree, with one condition. The new shapes still need the data, permissions and controls the old tools came with.

Chris Gregori put the other half plainly in January 2026: "code has become cheap, but software remains incredibly expensive."

---

### CODING AGENT

A model that reads a code base, edits files, runs commands and tests, and proposes a change for a person to review.

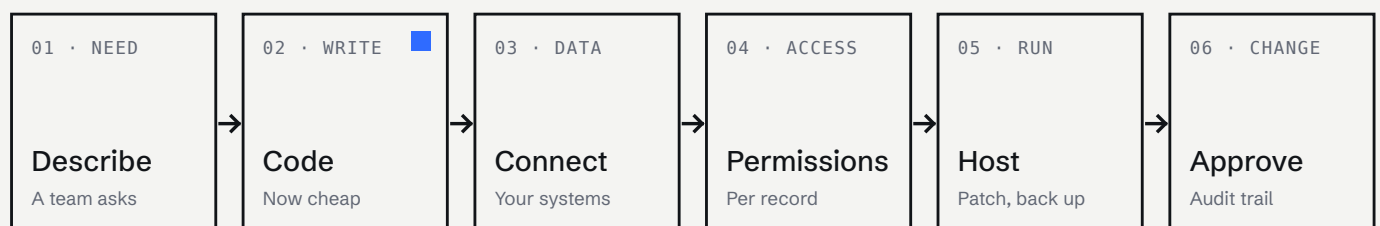


FIG. 1.1 · WHAT AN INTERNAL TOOL COSTS, STEP BY STEP. ONE STEP GOT CHEAP.

TABLE 1.1

# What a licensed tool bundles

| PART                 | WHAT THE LICENSE PAYS FOR         | ON A SHARED FOUNDATION             |
|----------------------|-----------------------------------|------------------------------------|
| Screens and workflow | The vendor's idea of your process | Generated for your process         |
| Data model           | A schema you adapt to             | Your records, typed in a graph     |
| Integration          | Connectors, often priced extra    | Written once, shared by every tool |
| Access control       | The vendor's roles                | Synced from your source systems    |
| Hosting              | Their cloud, their region         | Your servers, your region          |
| Upgrades             | A release cycle you wait for      | A change you approve this week     |

READ IT THIS WAY

The middle column is what a license pays for. Only the first row got cheap. The other five are a foundation: build them once, and every tool on top gets them.

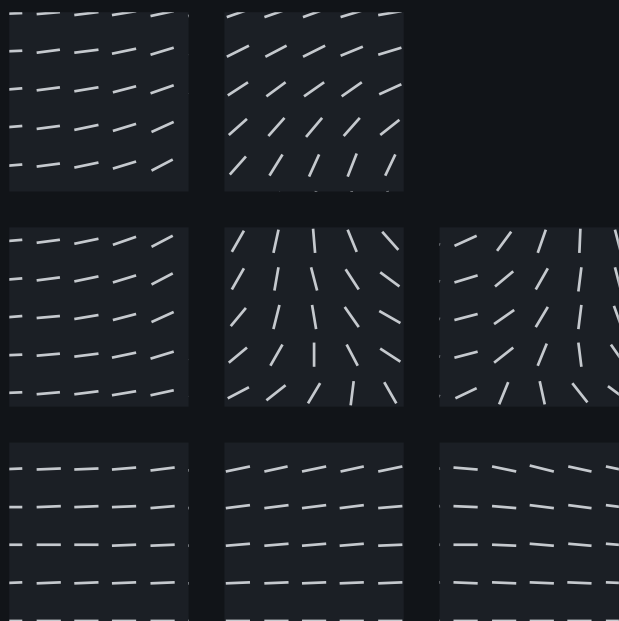
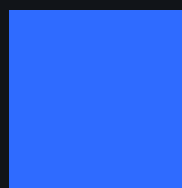
## 1.2 Rebuild what a team uses, not what a vendor sells

The first tools worth rebuilding are the ones a team uses for one workflow and pays for per seat: a review, a tracker, a form, a report. They sit beside your systems of record, not in place of them.

Systems of record stay. The ERP, the PLM and the ticket system own the data; the factory builds what people do with it.

LEGACY SOFTWARE

In this paper: software a company keeps because replacing it was too slow, not because it fits the work.



# 02

## A line, not a lab

---

What we mean by an AI factory, and what NVIDIA means

---

The six stations a request passes through

---

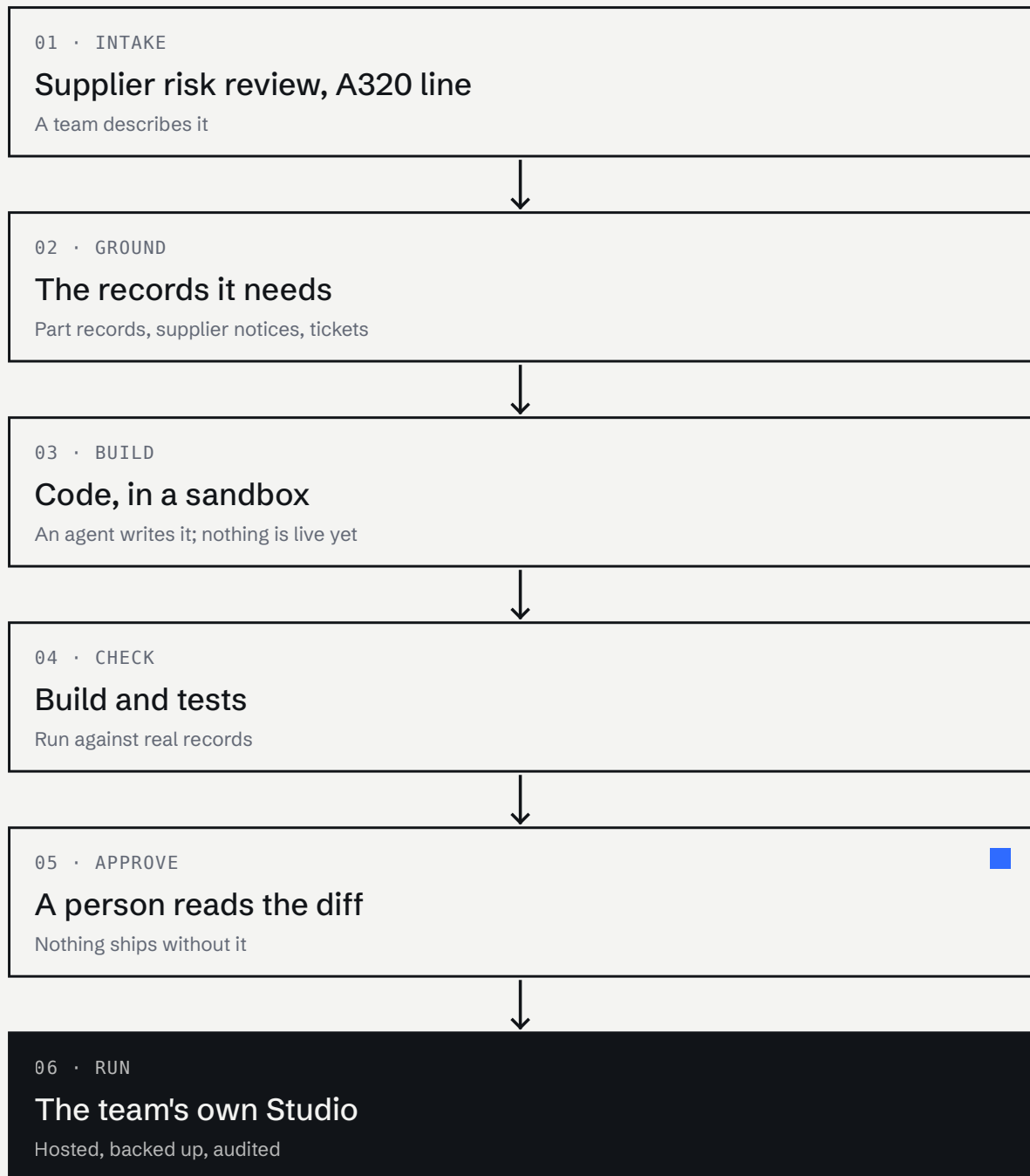
What a demo skips, and why it does not ship

---

FIGURE 2.1

# Six stations, one line

A request enters as a sentence and leaves as a tool people use. Each station has one job and one owner.



02

Grounding is a station, not a prompt: records are read through the permission check.

05

The marked step. It answers the auditor: who decided, and what changed.

06

Running is most of a tool's life. The factory runs every tool the same way.

2.2

## Two meanings of one word

NVIDIA uses "AI factory" for a data center whose "primary product is intelligence, measured by token throughput." That factory makes answers.

We use the word for the line that makes software: requests in, internal tools out, week after week. You can run ours on hosted models or on your own GPUs.

What makes it a factory rather than a lab is repetition. A lab builds a demo once. A factory builds the tenth tool the way it built the first, with the same checks, and keeps all ten running.

A demo generated from a prompt usually has stations 01 and 03: a request and some code. Stations 02, 04, 05 and 06 decide whether it ships. Andrej Karpathy described the gap in his June 2025 talk on how software is changing: his own app ran as a demo on his laptop in a few hours, and making it real took a week.

So judge a factory by what reached production and is still in use, not by what it can prototype in an afternoon.

---

### AI FACTORY

In this paper: a repeatable line that turns a team's request into internal software running in production.

---

### STATION

One step on the line with one job and one owner: intake, ground, build, check, approve, run.

---

### KEY POINT

**Judge a factory by what it ships, not by what it demos.**

# 03

## The foundation decides what ships

---

What every tool on the line needs from underneath

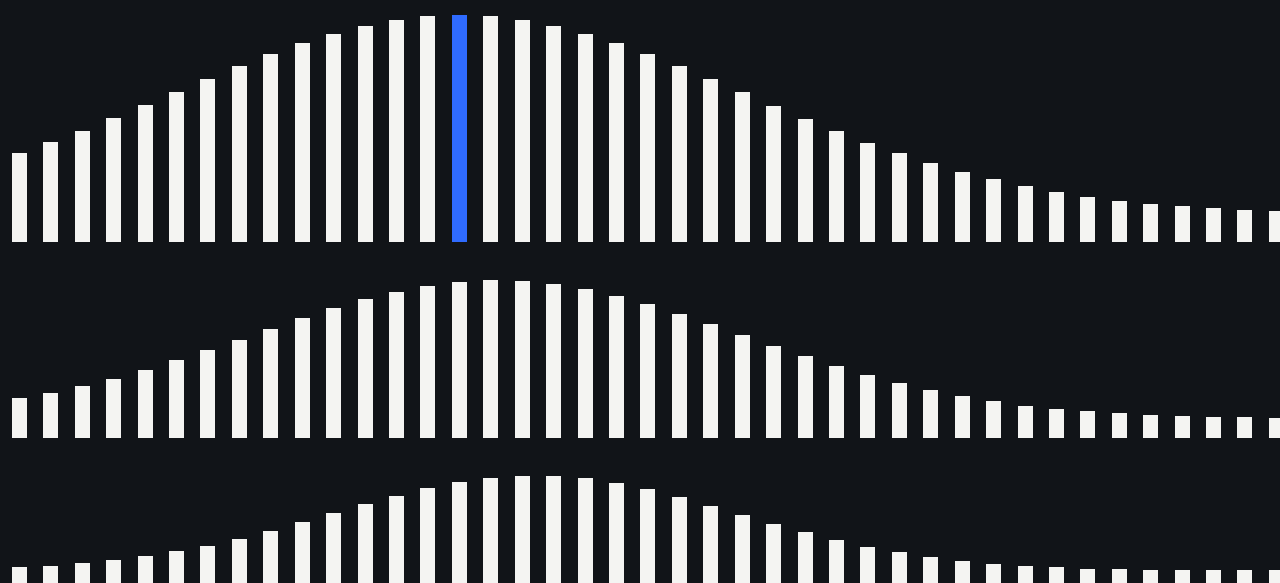
---

Why the tenth tool costs less than the first

---

Curiosity Studio as the foundation

---



## 3.1

## Every tool inherits its foundation

# A generated tool is as trustworthy as what it stands on. Build the data, the permissions and the controls once, and every tool inherits them.

Curiosity Studio is the foundation we built for this. Connectors bring records from your systems into one typed graph: tickets, part records, maintenance logs, supplier notices. More than 70 systems connect, and a connector SDK in C# or Python writes the rest.

Who may see a record is synced from the source system and stored in the same graph. Every read is checked per record at query time. A tool generated on Studio cannot show anyone a record the source would not.

Search and AI run on the same graph, in one runtime.

The model is your choice: 19 providers are built in, among them open weights on your own servers. Studio runs on premises or in a private cloud, with EU data residency.

---

#### CONTEXT GRAPH

Records from separate systems connected into one typed, permission-aware graph that people and models query.

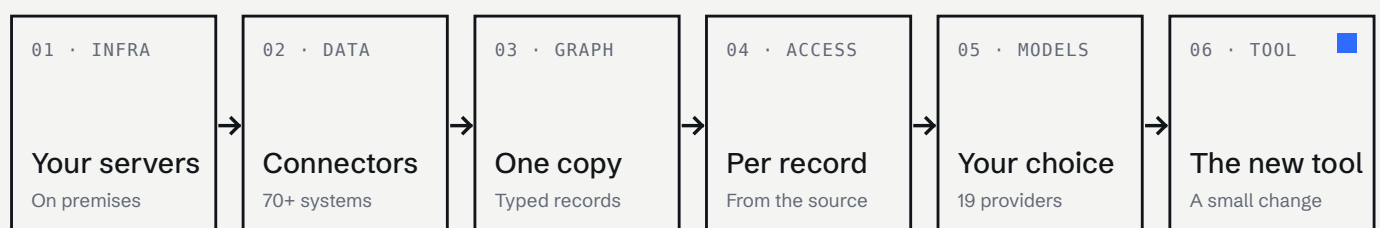


FIG. 3.1 · WHAT A TOOL ON STUDIO INHERITS, FROM THE BASE UP.

FIGURE 3.2

# The factory, from the ground up

Read it from the bottom. Each layer is built once and used by every layer above it. Most new tools change only the top of the pyramid.

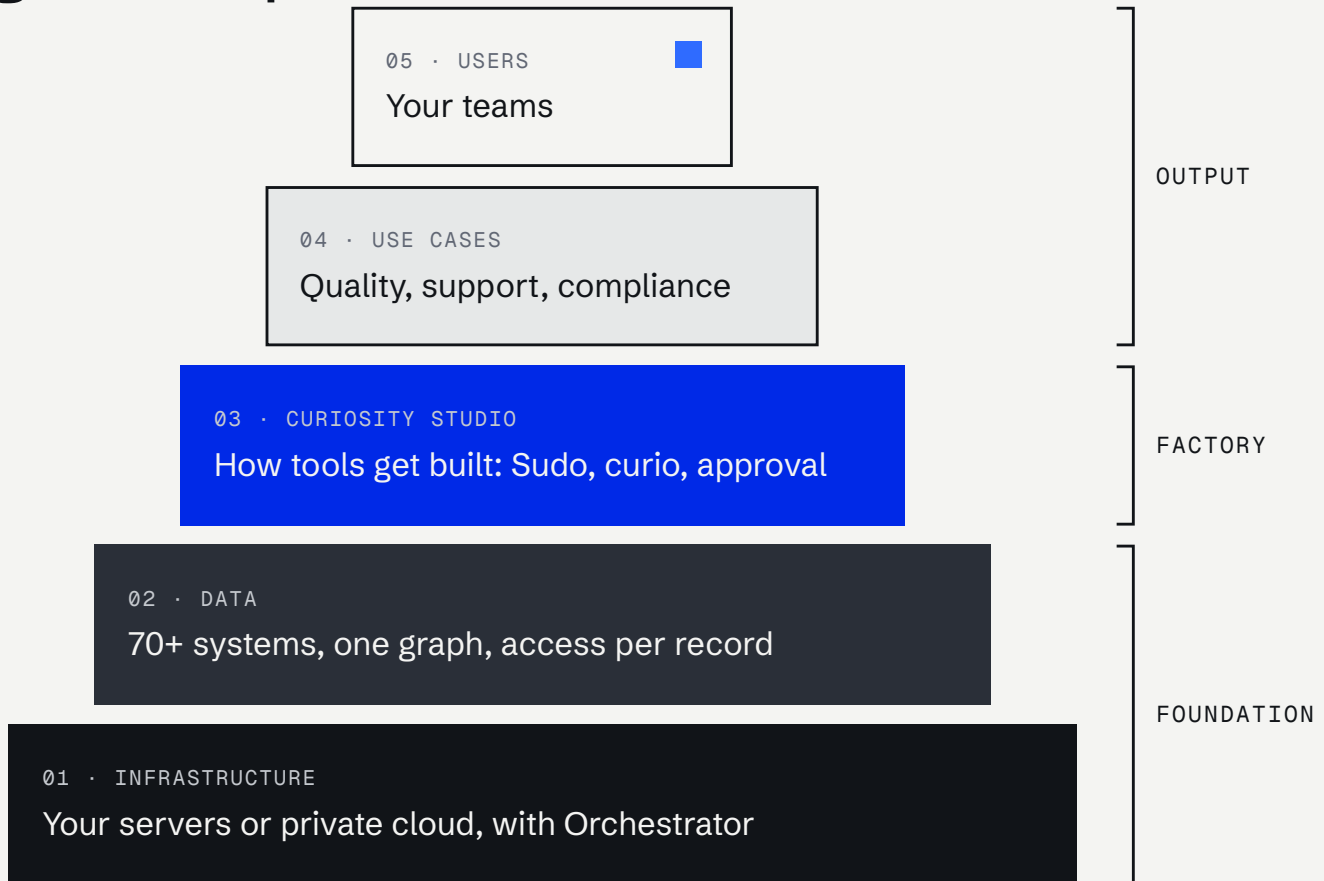


FIG. 3.2 · THE AI FACTORY AS A PYRAMID. THE FOUNDATION IS BUILT ONCE; EVERY TOOL ON TOP INHERITS IT.

## FOUNDATION

Your infrastructure and your data. Orchestrator runs a Studio per team on your servers; connectors keep the graph in step with the systems that own the records.

## FACTORY

Curiosity Studio is where a request becomes a tool. Sudo and curio open the same sandbox, and a person approves each change.

## OUTPUT

The use cases your teams describe, and the people who use the tools. This is the layer each new request adds to.

3.3

# Why the tenth tool costs less than the first

The first tool on a foundation pays for the connectors, the schema and the access groups. The second tool reuses them. By the tenth, a new tool is mostly an endpoint, a screen and an assistant over records that are already there.

Without a foundation the curve runs the other way. Each tool brings its own copy of the data, its own login, its own access check and its own dependencies. Ten tools are ten security reviews and ten things that drift apart. Our paper *The app is the easy part* collects the incidents that follow.

On a foundation, the review of a new tool shrinks to the change itself. Security reviews the platform once, and each tool after that is a diff they can read.

That is what speeds adoption up in a regulated company: not a faster model, but fewer things to approve per tool.

---

## IN STUDIO

Most of a new tool is configuration: endpoints, AI tools, agents and schemas, kept as files under `/workspace` and reviewed as a diff.

---

## SPARK

Our model server for open weights on your own GPUs, behind OpenAI and Anthropic compatible APIs, so Studio can use it like a hosted provider.

---

## KEY POINT

**A foundation turns each new tool into a diff a person can review.**



# 04

## The factory floor

---

A Studio for every team, project or customer

---

Memory for the Studios in use, not every Studio

---

The work around a Studio, done once for all of them

---

## 4.1

# A Studio for every team, one way to run them

A factory turns out tools for many teams. Curiosity Orchestrator gives each team, project or customer its own Studio, with its own data and users, behind one address on servers you control.

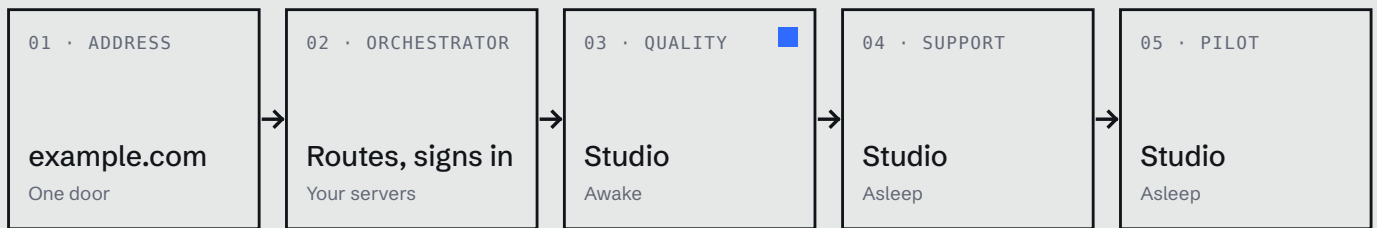


FIG. 4.1 · ONE ORCHESTRATOR, MANY STUDIOS. EACH SLEEPS UNTIL SOMEONE OPENS IT.

Each Studio keeps its own data, users and settings, and a team sees its Studio and nothing else.

A Studio nobody has opened for a while goes to sleep and frees its memory; the next visit wakes it. So you size the server for the Studios in use, not for every Studio.

When one server fills up, add machines. One orchestrator manages the others, and every Studio keeps its address wherever it runs.

**KEY POINT**

**Separate Studios, one place to secure.**

**ORCHESTRATOR**

Hosts Studio instances behind one address. All it needs is Docker on a server.

TABLE 4.1

# The work around a Studio, done once

| JOB          | THE ORCHESTRATOR                      | WITHOUT IT             |
|--------------|---------------------------------------|------------------------|
| Releases     | One Studio first, then all at once    | Each server by hand    |
| Backups      | Download, restore or reset a Studio   | A script per server    |
| Sign in      | Accounts defined once, one sign in    | Accounts per instance  |
| Certificates | Yours, or Let's Encrypt               | Renewed by hand        |
| Capacity     | The least used Studio sleeps first    | Sized for the peak     |
| Monitoring   | CPU, memory, disk and logs per Studio | A dashboard per server |
| Automation   | The console's API, with a token       | Runbooks               |

READ IT THIS WAY

Every row is a runbook a factory would otherwise keep per server. The orchestrator controls Docker on its host, which amounts to full control of that machine, so limit who can sign in to it.

4.2

## Try a release before every team gets it

A factory changes its own floor too. Run a new Studio release on one Studio, check it with that team, then move every other Studio to it in one step. Their data stays where it is.

Developers get their own console access and a number of Studios to experiment in, reset to empty and delete when done, away from production.

# 05

## How the work comes through

---

Sudo builds in a sandbox, a person approves

---

curio opens the same sandbox to coding agents

---

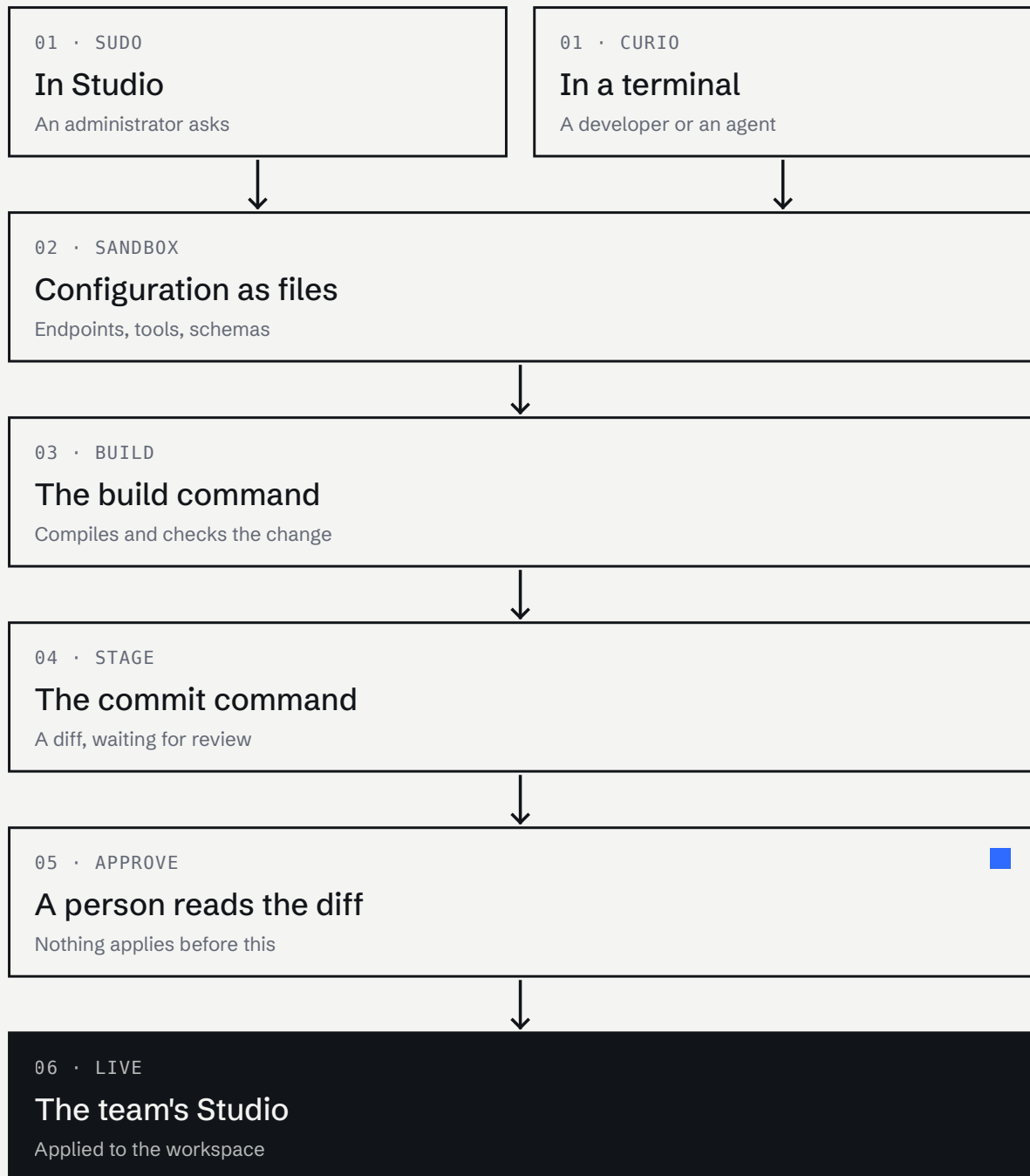
Two applications, each built in an evening

---

FIGURE 5.1

# One path for all

Every change to a Studio goes through the same sandbox and the same approval, whoever starts it.



**01**  
Sudo and curio open the same sandbox. A curio session is a Sudo conversation.

**02**  
The sandbox has no network route out and no write access to your data.

**05**  
The marked step. The diff is the review: a person sees exactly what will change.

5.2

# Two applications, each built in an evening

In October 2026 a coding agent built a non-conformance application on Studio during a dinner with a prospective customer. The first commit, our application template, was at 20:08. The application was deployed and tested at 21:11, and the last commit of the evening was at 22:18.

It held 259 synthetic NCRs, access per program and per division, a workflow from raised to closed, and an assistant that works through the records as the person asking.

Earlier that month an agent built an aviation market intelligence app in one evening. It reads 248 news feeds, knows which companies and aircraft each article mentions, and writes a daily brief.

Neither went to production that evening. They show how much of the work the foundation already holds: the agent wrote a connector, endpoints, a front end and an assistant, and inherited the rest.

---

## APPLICATION TEMPLATE

A starting repository for a Studio application: one skill per layer and a reference implementation of each.

---

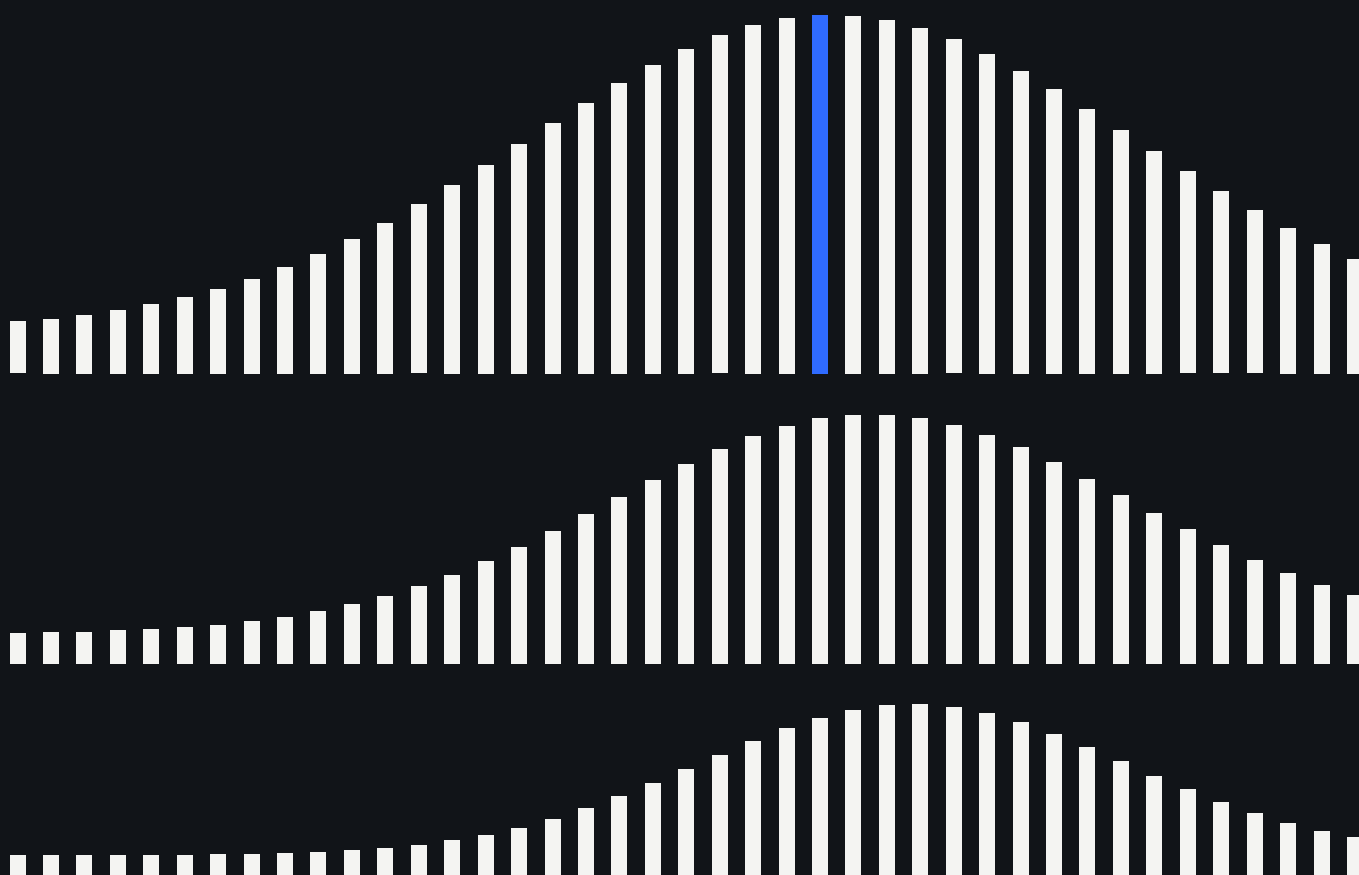
## CURIO

The command line for Studio, for you and your coding agent. Install it from [curiosity.sh](https://curiosity.sh).

---

## KEY POINT

**Agents write the code.  
A person approves  
what ships.**



06

## Features per day, not months per feature

---

What our own changelog counted, 2025 to 2026

---

Who writes the code now, and who reviews it

---

Three days in October, feature by feature

---

TABLE 6.1

# What our changelog counted

| PERIOD                         | WEEKLY RELEASES | ENTRIES A WEEK | FEATURES A WEEK |
|--------------------------------|-----------------|----------------|-----------------|
| June to December 2025          | 29              | 9              | 2               |
| January to June 2026           | 26              | 13             | 3.5             |
| 7 September to 20 October 2026 | 7               | 49             | 17              |

HOW WE COUNTED

From the public changelog at docs.curiosity.ai/changelog: one file per weekly release, one entry per change, each marked Feature, Improved or Fixed. 17 features a week is about 2.4 a day, weekends included. Our internal changelog, rebuilt from commits, shows 7 to 9 entries a week in every year from 2021 to 2025.

6.2

## Who writes the code now

In October 2026, 243 of the 275 commits to Studio's main branch carried a coding agent's attribution. The rest are merges, package updates and small fixes made by hand.

People decide what to build, review every pull request and merge it. The agents read written instructions and skills kept beside the code.

THE SAME PROCESS WE SELL

An agent writes the change, a build checks it, a person approves it. That is how Studio is built, and how Sudo and curio change a workspace.

TABLE 6.2

# Three days in October

| DAY          | WHAT SHIPPED                                              | PULL REQUESTS          |
|--------------|-----------------------------------------------------------|------------------------|
| 18 Oct       | Web search providers, image generation, PowerPoint themes | 3, from 17:10 to 22:00 |
| 22 to 23 Oct | Cluster mode: the primary moves between servers           | 3, in 12 hours         |
| 26 Oct       | Voice control: "Hi Sudo" and dictation                    | 4, from 18:30 to 23:36 |

READ IT THIS WAY

Times are when each pull request was merged into the main branch, Munich time. Cluster mode covers handover, election of a new primary and forced promotion, without reloading the graph. Each row is a changelog entry at [docs.curiosity.ai/changelog](https://docs.curiosity.ai/changelog).

6.3

## Review sets the pace now

When a feature takes a day to write, the backlog stops being the bottleneck. Review becomes the bottleneck, and that is where a factory should be slow.

The same holds for your own factory. The rate at which tools ship is set by how fast people can read and approve diffs.

KEY POINT

The faster code gets written, the more the foundation and the approval matter.

# 07

## Start with one tool

---

How to pick the first tool to rebuild

---

Three steps from license to running tool

---

Four questions to ask any AI factory

---

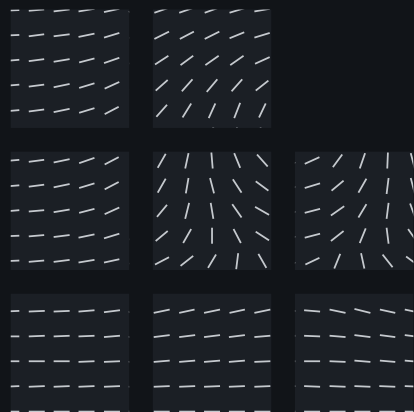


TABLE 7.1

# Pick the first tool by these signs

| SIGN                              | WHY IT MATTERS                          |
|-----------------------------------|-----------------------------------------|
| One team, one workflow            | A clear owner and a clear test          |
| Its data lives in systems you run | The connector is most of the work       |
| Paid per seat, used for one task  | The saving is visible                   |
| Access rules you can name         | They come from the source               |
| A process that changes often      | Approval beats a vendor's release cycle |
| Not a system of record            | Read and enrich before you replace      |

READ IT THIS WAY

The best first tool sits on top of your systems of record, not in place of one. Replacing an ERP is a program. Replacing the review spreadsheet beside it is a pilot.

7.1

## What a pilot proves

A pilot on one tool proves the foundation, not the tool. The connectors, the access groups and the approval path it sets up are what the second tool reuses.

A developer install runs on a laptop in minutes. A pilot connected to your real systems usually takes weeks, most of it spent on the connectors.

# From a license to a running tool

01

Pick one tool a team pays for per seat and uses for one workflow.

02

Connect its sources to Studio and rebuild that workflow on the graph.

03

Run both side by side, then retire the license when the team agrees.

ASK OF ANY AI FACTORY

☐ Where does the data live, and how many copies are there?

☐ Who checks access on each read, and in which system?

☐ Who approves a change before it reaches users?

☐ Who runs the tenth tool, and the hundredth?

# AIRBUS

“At Airbus, we developed the next generation of customer support AI assistance tools with Curiosity, enhancing our technical support’s ability to access and navigate through vast data from multiple sources.”

SERVICES INNOVATION TEAM · AIRLINE SERVICES AT AIRBUS

---

## THE PROBLEM

Forty years of technical data across CRM systems, network drives and engineering databases, in a vocabulary of acronyms and reference numbers that defeated ordinary search.

## WHAT CHANGED

One entry point over those sources, with the engineering references extracted and joined in a graph, and filters by aircraft type, program and reference number.

# Running on infrastructure our customers own.

Across Curiosity's production deployments, on premises and in private clouds.

30TB+

DATA CONNECTED IN PRODUCTION

20,000+

ACTIVE USERS ACROSS DEPLOYMENTS

15%+

EFFICIENCY GAIN IN PRODUCTION WORKFLOWS

70+

ENTERPRISE SYSTEMS CONNECTED

# What to take from this paper

---

01

Writing software got cheap. Owning it did not.

Data, access, hosting and upgrades still cost what they did.

---

02

A factory ships what its foundation allows.

Build it once; every tool on top inherits it.

---

03

Agents build, a person approves, the floor runs it.

Studio, Sudo and curio, Orchestrator.

---

SOURCES

# Where the outside words come from

| TOPIC               | SOURCE                                                      | DATE     |
|---------------------|-------------------------------------------------------------|----------|
| Software categories | Barry McCardel, A new era for Hex, and data, hex.tech       | Oct 2026 |
| Code and software   | Chris Gregori, Code is cheap now. Software isn't.           | Jan 2026 |
| AI factory          | NVIDIA glossary, What is an AI factory?                     | Current  |
| Demo and real       | Andrej Karpathy, Software is changing (again), Y Combinator | Jun 2025 |
| Incidents           | Curiosity, The app is the easy part                         | Oct 2026 |

CURIOSITY'S OWN SOURCES

| TOPIC                | SOURCE                                                    | DATE                 |
|----------------------|-----------------------------------------------------------|----------------------|
| Feature counts       | docs.curiosity.ai/changelog                               | Jun 2025 to Oct 2026 |
| Commits              | Curiosity Studio repository, main branch                  | Oct 2026             |
| NCR application      | blog.curiosity.ai, The era of mockups is over             | Oct 2026             |
| Aviation application | blog.curiosity.ai, An aviation market intelligence app    | Oct 2026             |
| Orchestrator         | curiosity.ai/orchestrator, docs.curiosity.ai/orchestrator | Oct 2026             |
| Sudo and curio       | curiosity.ai/sudo, curiosity.sh                           | Oct 2026             |

# Terms used in this paper

---

## AI FACTORY

A repeatable line that turns a team's request into internal software running in production.

---

## CODING AGENT

A model that edits a code base, runs its tests and proposes the change for review.

---

## CURIO

The command line for Curiosity Studio, for a developer and their coding agent.

---

## SUDO

The admin assistant in Studio. He builds in a sandbox; a person approves.

---

---

## APPLICATION TEMPLATE

A starting repository for a Studio application: one skill per layer and a reference implementation of each.

---

## CONTEXT GRAPH

Records from separate systems in one typed, permission-aware graph that people and models query.

---

## ORCHESTRATOR

Hosts many Studios behind one address, waking each when someone opens it.

---

## SYSTEM OF RECORD

The system that owns a kind of data, such as the ERP for orders or the PLM for parts.

---

# Build the foundation once. Run the factory on it.

Pick one tool your teams pay too much for. Install the free developer version and rebuild it, or book a demo and we build it with you on data shaped like yours.

[curiosity.ai/request-demo](https://curiosity.ai/request-demo)

#### DATA RESIDENCY

EU, on premises or private cloud

#### COMPLIANCE

GDPR

#### MEMBER

KI Bundesverband