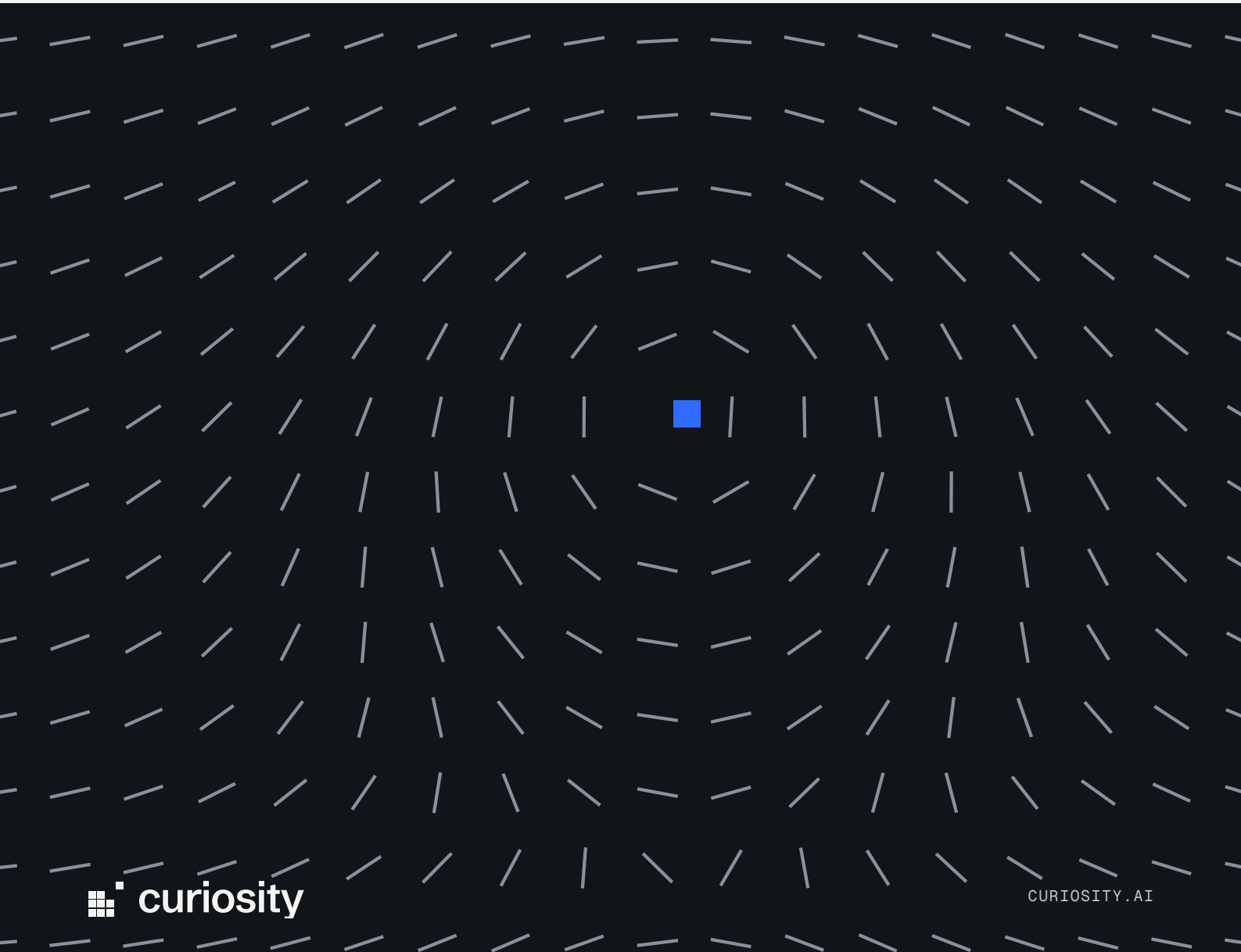


The vendor lock-in you always wish you had.

Every AI platform locks you into something. This paper shows where that lock-in should land, and how to test a vendor's exit before you sign. Written for the people who choose.

TOPIC	FOR	DATE
Platform choice and lock-in	CIOs and enterprise architects	October 2026



Lock-in is not optional. Where it lands is.

01	Something always builds up Waiting and building it yourself are lock-in too.	03
02	The model is a setting Why the model is the last thing to lock into.	06
03	Your systems stay the system of record Nothing migrated in, so nothing to migrate out.	08
04	What you build lives in your git Schemas, endpoints and agents as files you hold.	11
05	The lock-in worth having What stays sticky, said plainly.	13
06	Test the exit before you sign Six questions for any vendor, and three steps.	15
.	Glossary	21

IN ONE PARAGRAPH

Any system that is useful collects your work: records joined across systems, rules for who sees what, tools people rely on. So you will be locked into something. Make it your own context, on your own servers, in files you can read. Keep the model, the cloud and the vendor replaceable.

01

Something always builds up

Why waiting and building it yourself are lock-in too

The five layers where your work collects

The one test that tells good lock-in from bad

1.1

Lock-in is the work you would have to redo

Every platform you run collects your work. The question is not whether you are locked in. It is what you are locked into.

Ask a CIO about vendor lock-in and you hear about an exit that cost more than the entry. A migration that took a year. A renewal price that rose because leaving cost more than paying.

So the instinct is to avoid commitment. Wait for the market to settle, or build it yourself from open frameworks.

Neither avoids lock-in. Waiting locks you into the backlog you have today. Building it yourself locks you into code your own team now has to run, connectors and permissions included.

Any system that is useful collects something: records joined across systems, rules for who sees what, tools people now rely on.

So the useful question is where that work lands. In a vendor's cloud, in one provider's model, in a sandbox you cannot export? Or on your servers, in files you can read, under the permissions you already run?

LOCK-IN

The cost of leaving a system, counted in the work you would have to redo.

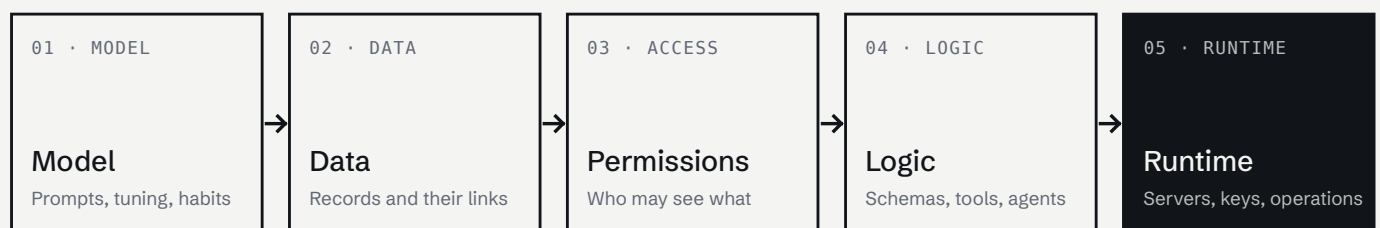


FIG. 1.1 · FIVE LAYERS WHERE YOUR WORK COLLECTS. THIS PAPER TAKES THEM IN TURN.

TABLE 1.1

Where lock-in lands, layer by layer

LAYER	WHAT BUILDS UP	LOCK-IN TO AVOID	LOCK-IN WORTH HAVING
Model	Prompts, tuning, habits	One provider's roadmap	Any model, as a setting
Data	Records and their links	Moved into their cloud	A synced copy, your servers
Permissions	Who may see what	A second set of grants	Synced from each source
Logic	Schemas, tools, agents	Only inside their sandbox	Files in your git remote
Runtime	Servers, keys, operations	Their cloud, their key	Your servers, your key
Evaluation	Proof that it works	A services phase first	A free install, your data

READ IT THIS WAY

The middle column is where leaving costs you your own assets. The right column is where leaving costs only the work you chose to build. Every row has a cost. Only one kind is a hostage.

1.2

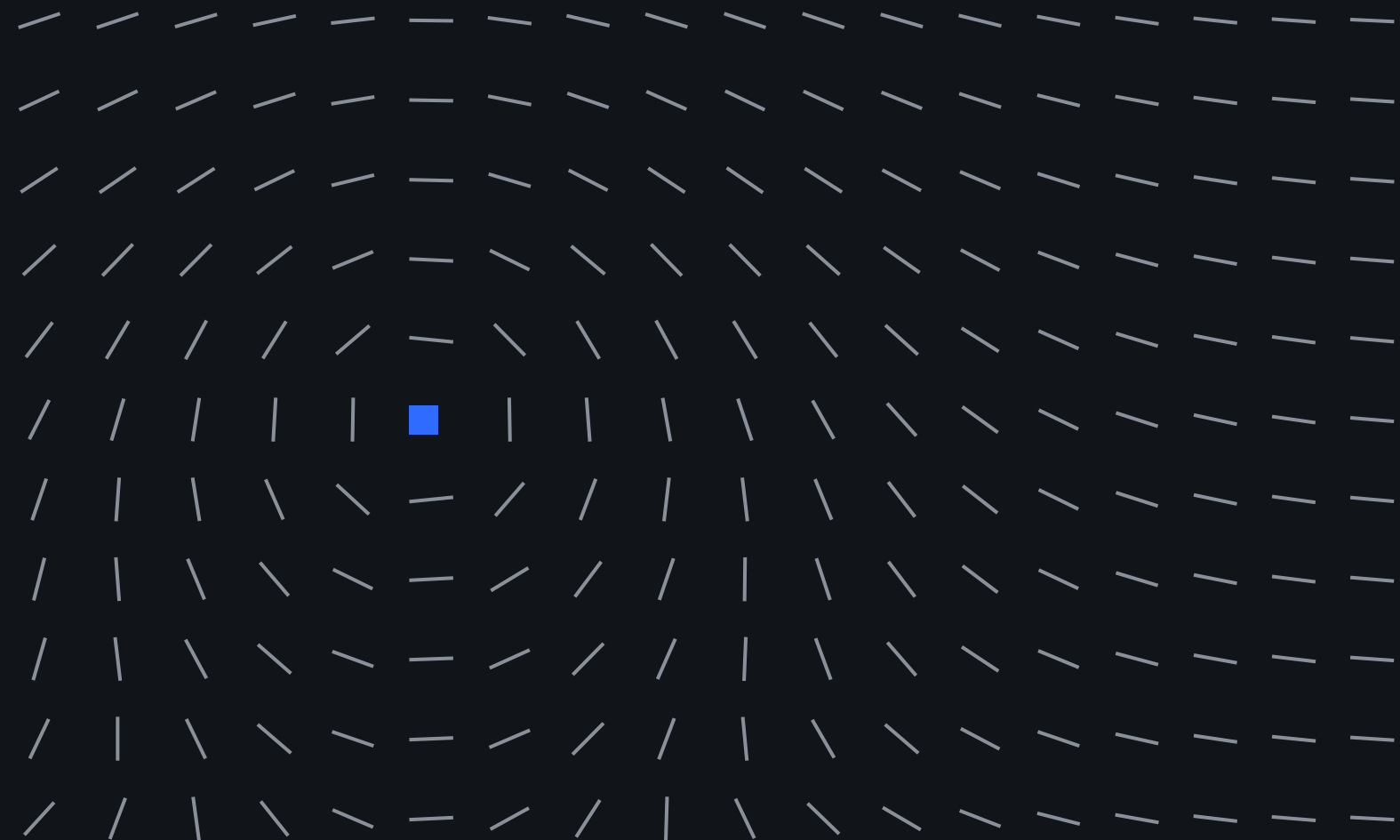
The test: would you stay if leaving were free?

Good lock-in passes one test. You would stay even if leaving cost nothing, because the system does its job. You stay for the value, not because your data, your rules or your code are stuck inside.

Bad lock-in fails it. You stay because leaving means a migration, a second set of permissions to rebuild, and tools that only ran in someone else's environment.

ASK IT OF EVERY LAYER

For each row of Table 1.1, ask what you would lose on the day you leave. Your work, or your assets? The answer is the lock-in you are buying.



02

The model is a setting

Why the model is the last thing to lock into

Four ways to run a model, from a key to your own servers

What stays the same when the model changes

2.1

Swap the model, keep the rest

New models arrive every few months, from different providers, in different regions. A platform built around one model inherits that model's roadmap, price and regions. A platform built around your context treats the model as configuration.

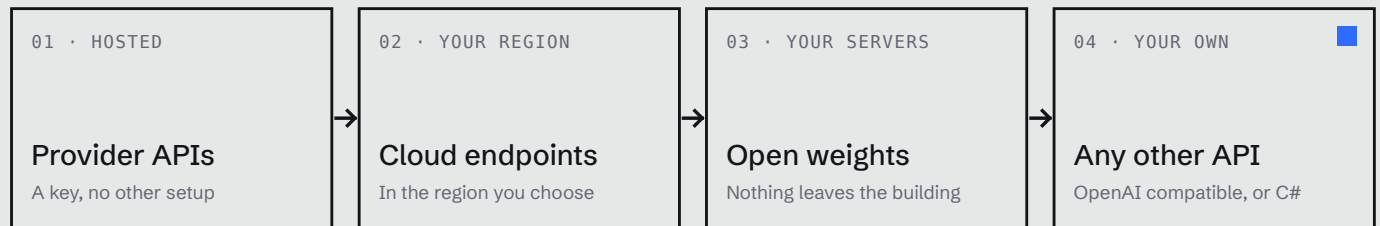


FIG. 2.1 · FOUR WAYS TO RUN THE MODEL. EACH IS A SETTING, AND THE GRAPH IS THE SAME UNDER ALL FOUR.

Curiosity is not the model. It is what gives the model context. Providers built into Studio include OpenAI, Anthropic, Google, Mistral, Azure OpenAI, AWS Bedrock and Ollama for open weights on your own servers.

A model that is not listed still fits. Point Studio at any endpoint that speaks the OpenAI API, or write a provider of your own in C# for any other API.

Changing the model is configuration. The graph, the permissions and the tools your teams built do not change with it.

Permissions are resolved before the model is called. Studio's built in tools only hand the model records the person asking may see, whichever model that is.

KEY POINT

Lock into a model and you inherit its roadmap. Lock into your context and the model becomes a setting.

OPEN WEIGHTS

A model whose weights you can run on your own servers, so no provider sees the request.

03

Your systems stay the system of record

Connect everything, replace nothing

Permissions read from the source, at query time

Your servers, your key, your choice of where

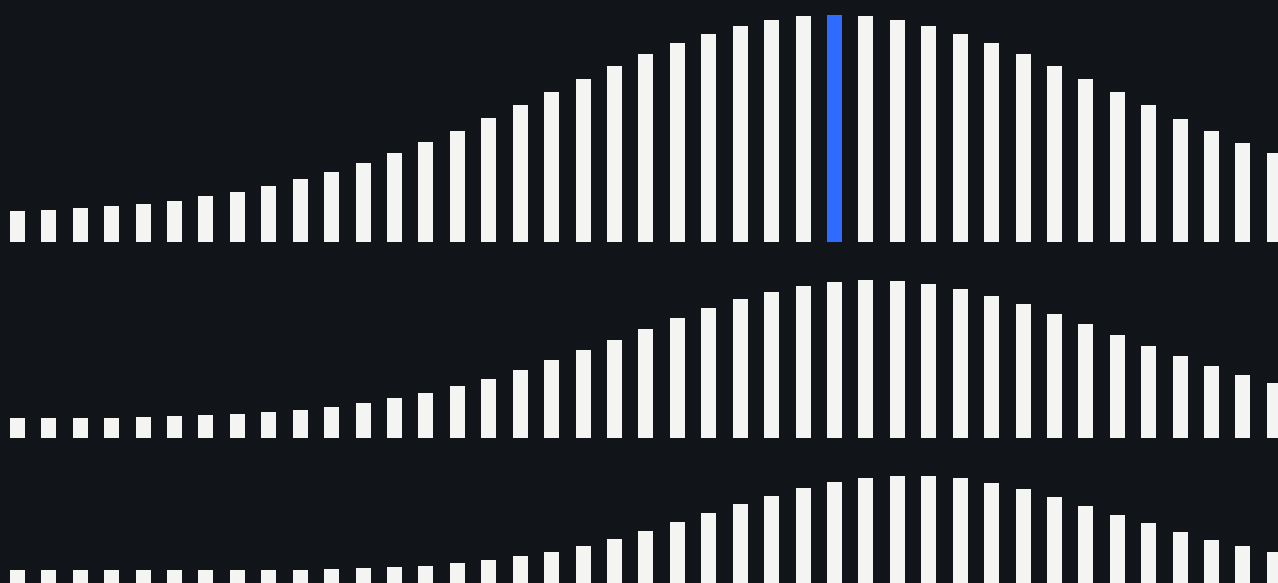
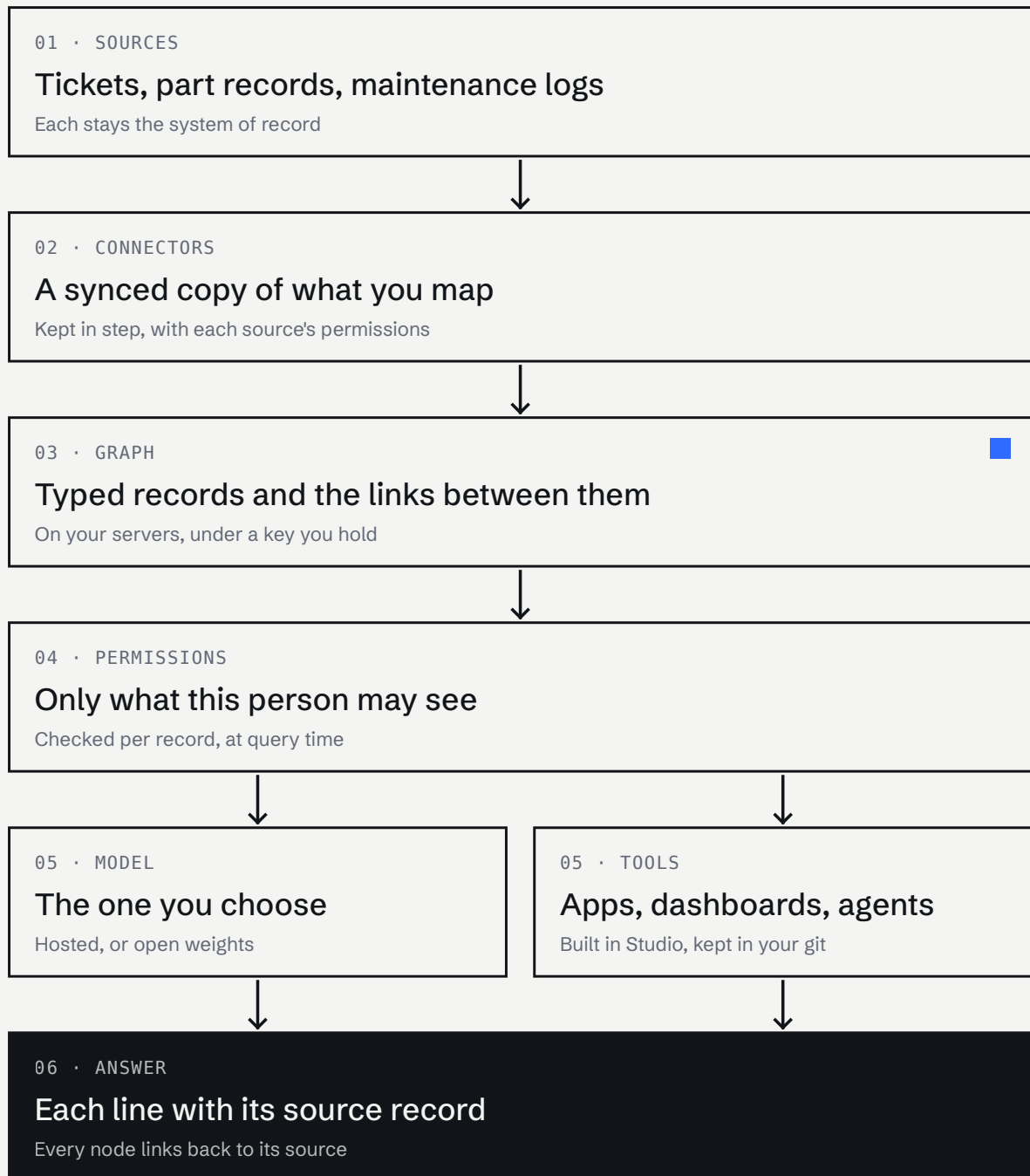


FIGURE 3.1

Record to answer

One path from a record to an answer. Every step runs in your environment.



01

Sources keep their role, their validation and their audit trail. Nothing is switched off.

03

The marked step: the graph runs where you deploy it, under a key you supply.

06

Every answer cites its records, so a reviewer opens the source itself.

3.2

Nothing to migrate out

Connecting a system copies what you map. The connector keeps that copy in step, and each node links back to its source. The source stays the system of record, with its own validation and its own audit trail.

That changes what leaving means. Nothing was migrated into Curiosity, so nothing has to be migrated out. Your ticketing system, your SharePoint and your network drives were never switched off.

Permissions follow the same rule. Access is synced from each source system and checked per record when a query runs. There is no second set of grants to drift while you stay, and none to rebuild when you go.

Where the copy lives is your choice. On premises it stays in your environment, and Curiosity holds none of it. You supply the encryption key and the certificate. With an offline license, local models and usage reporting switched off, the deployment can run air gapped.

Your data also trains nothing. Customer data is never used to train shared or public AI models.

SYSTEM OF RECORD

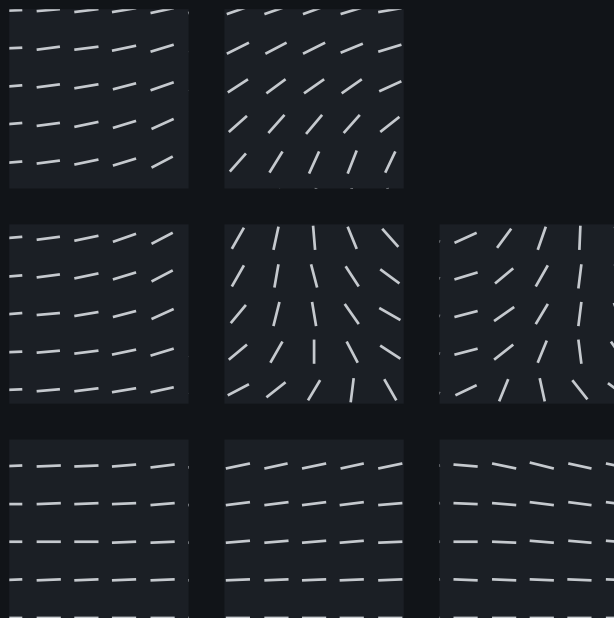
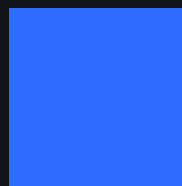
The system where a record is created and owned. Curiosity keeps a synced copy and links back to it.

AIR GAPPED

A deployment with no connection to the internet at all, models included.

KEY POINT

Nothing was migrated in, so nothing has to be migrated out.



04

What you build lives in your git

Schemas, endpoints, tools and agents as files you can review

Your code against a documented API, in C# or Python

Upgrades that compile, or say why they do not

4.1

Your configuration is a repository you hold

Studio commits every configuration change to a git repository and can push it to your remote. What your teams build is a set of files you can review, version and import back.

```
studio-config/      # pushed to your git remote
├─ schemas/         # node types and their links
├─ endpoints/       # your REST endpoints, in C#
├─ ai-tools/        # the tools agents may call
├─ agents/          # agents and their prompts
└─ tasks/           # scheduled jobs
```

LISTING 4.1 · ILLUSTRATIVE LAYOUT, NOT THE ACTUAL FILE NAMES

WHAT YOU WRITE	WRITTEN IN	USED THROUGH
Endpoints, AI tools, agents	C#, inside Studio	REST, Studio, MCP
Connectors	C# or Python	The connector SDK
Your own code	C# or Python	The Library API

4.2

Upgrades compile, or say why

Extensions are kept as source and compiled by Studio. An upgrade compiles them against the new version, so a break shows as a compile error, not a silent failure.

Your own front end can replace the built in one. Push it over git, and Studio builds each commit as a preview.

THE AUDIT TRAIL IS YOURS TOO

Admins choose which actions are logged, then filter and download the trail. The files carry a hash chain you can verify, and convert to CSV or JSON for your own retention.

05

The lock-in worth having

What stays sticky after a year, said plainly

Why the cost of leaving is the value you built

The difference between a switching cost and a hostage

5.1

You will be locked into your context. Good.

After a year you will not want to leave the context you built. That is the lock-in you always wish you had.

Be clear about what stays sticky. After a year your graph joins tickets to part records to supplier notices, across systems that never shared a key.

Enrichment has pulled the identifiers out of field reports. Teams rely on tools they described in a sentence.

Leaving would mean rebuilding that somewhere else. The joins, the enrichment and the tools are work, and work costs something to redo.

We would rather say so than claim there is no switching cost. Every platform worth running has one.

Look at what that cost is made of. Your records never left their source systems. Your permissions still live where they were set. Your schemas and agents are files in your git remote.

So the cost of leaving is the value you built. It is not your data held hostage.

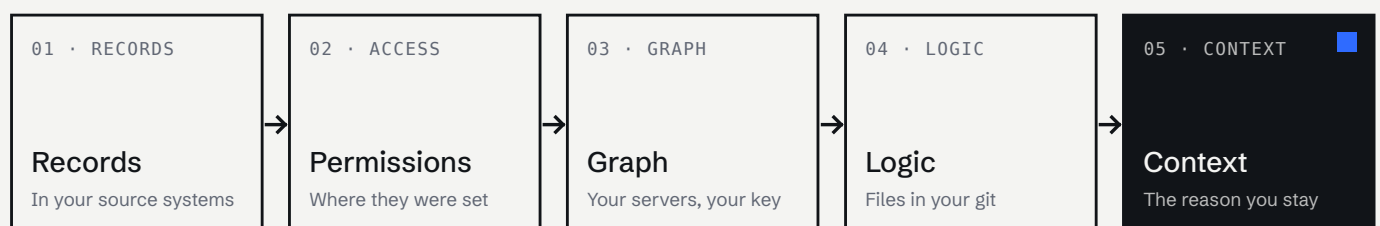
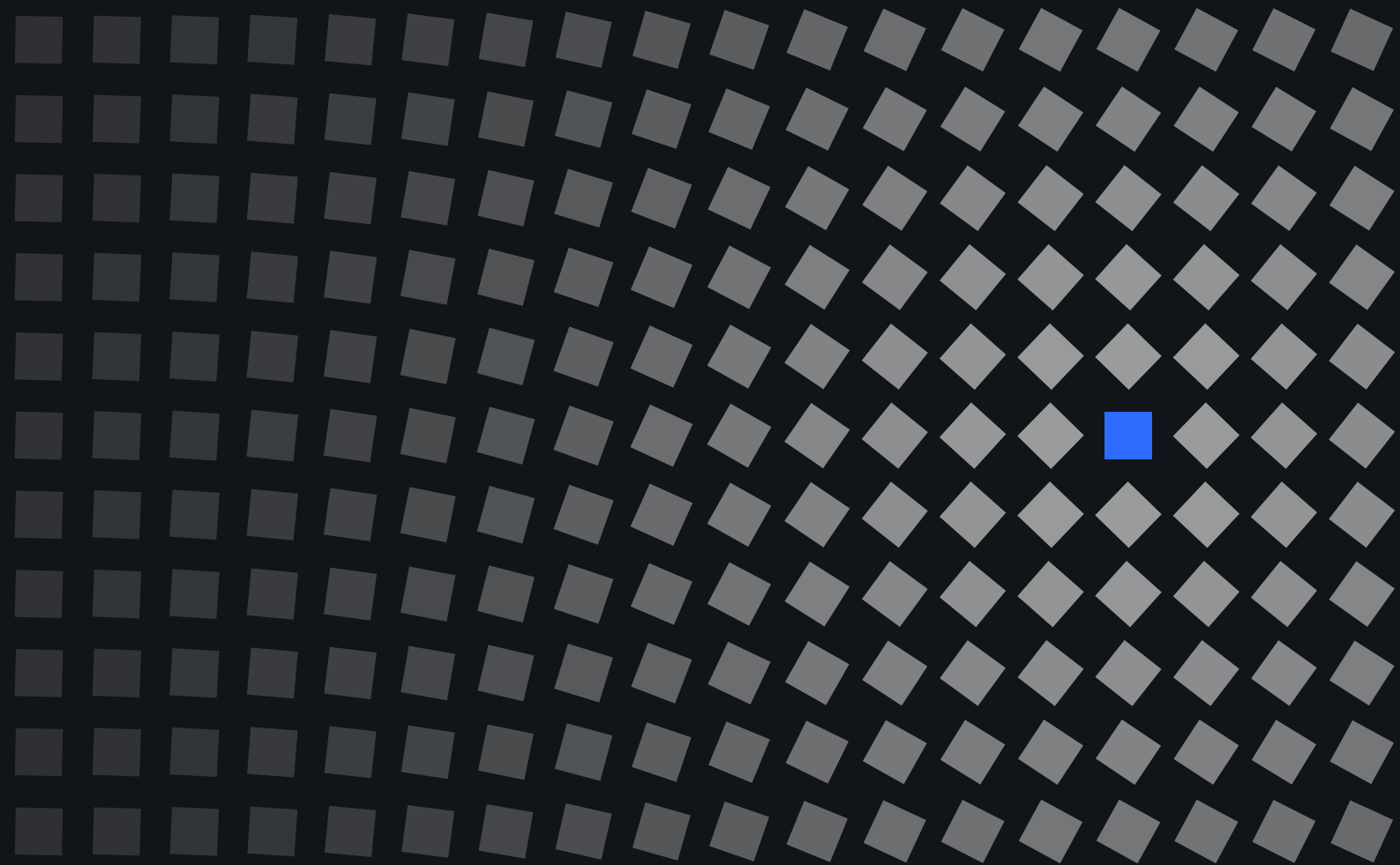


FIG. 5.1 · WHAT BUILDS UP, AND WHERE EACH PART IS HELD. FOUR PARTS YOU KEEP. THE FIFTH IS WHY YOU STAY.



06

Test the exit before you sign

Six questions to ask any vendor, us included

The answers to worry about

Three steps to test it on your own data

TABLE 6.1

Six questions for any vendor's exit

ASK THE VENDOR	THE ANSWER TO WORRY ABOUT	CURIOSITY'S ANSWER
Can we change the model?	A new contract	A setting
Where does our data live?	In our cloud	Your servers, your key
Who decides who sees what?	A second set of grants	Your source systems
Where is what we built?	In our environment	Your git remote
Can we try it first?	After a services phase	Free, no time limit
What happens if we leave?	A migration project	Your sources never moved

READ IT THIS WAY

Ask these in the evaluation, not at the renewal. A vendor who answers from the middle column is selling you lock-in on your assets. Ask us too.

6.2

Try it before anyone signs anything

The developer version of Studio is free for up to ten test users, with no time limit for development use. It installs on one machine with one command, and your data stays on that machine.

So your own engineers can run the evaluation, on a real system, before a contract or a services phase.

ONE COMMAND

On macOS and Linux, `curl -fsSL https://curiosity.ai/install.sh | bash` starts Studio in a container. Windows has an installer of its own.

Three steps to test it on your data

01

Install the free developer version on one machine you control.

02

Connect one system your teams already argue about.

03

Change the model, then check that nothing else changed.

BEFORE YOU START

- ☐ Which model would you want to replace first, and why?
- ☐ Which source system holds the permissions that matter?
- ☐ Where is the git remote for what your teams build?
- ☐ Who needs to read the audit trail, and in what format?

AIRBUS

“By intelligently enhancing our search efficiency, Curiosity lets Airbus technical support quickly find information across millions of documents.”

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

Lock-in is not optional.

Something always builds up. Choose where it lands.

02

Keep the model a setting.

When the model changes, your context should not.

03

Own what you build.

Records in your systems, logic in your git, keys on your servers.

Terms used in this paper

AIR GAPPED

A deployment with no connection to the internet, models included.

CONTEXT GRAPH

Records from many systems and the typed links between them, served to people and models.

OPEN WEIGHTS

A model you can run on your own servers, so no provider sees the request.

PERMISSION-AWARE

Results filtered by what the user may see in the source system, at query time.

CONNECTOR

The part that copies mapped records from a source system and keeps them in step.

LOCK-IN

The cost of leaving a system, counted in the work you would have to redo.

OPENAI COMPATIBLE

An endpoint that speaks the OpenAI API, as many inference servers and gateways do.

SYSTEM OF RECORD

The system where a record is created and owned, and which stays authoritative.

Stay because it works. Not because you cannot leave.

Install the free developer version on your own machine and connect one system. Or book a demo, and we run it against data shaped like yours, with the model you choose.

curiosity.ai/request-demo

DATA RESIDENCY

EU, on premises or private cloud

COMPLIANCE

GDPR

MEMBER

KI Bundesverband