BUILD VS BUY: VENDOR INDEPENDENCE

# Rent the commodity. Own the moat.

Build-versus-buy is the wrong binary. The real decision is what to own and what to rent, and the cost of getting it backwards is a business that cannot move when a provider changes a price, a region, or a term. This briefing is the decision framework: what a lock-in actually costs, how to keep the underlying model swappable by design, and exactly where the major agent platforms make you a tenant of your own capability.

FOR

**CTO, CIO,** Head of Architecture, CDO, procurement

COMPANION BRIEFINGS

**05 Sovereignty,** 01 Operating model

GROUNDED IN

**The open-source control plane,** the platform survey

DIRECT LINE

**teams@cohorte.co**

02 THE LOCK-IN TRAP

# The switching cost is invisible until the day you need to switch.

A team picks one vendor's agent platform because it is the fastest way to ship. The agents work. A year later the provider triples the price of the tier they built on, or restricts a model to certain regions, or changes a term the legal team cannot accept. Now the business discovers what it actually signed up for: not a tool, a dependency.

> Nobody decides to be locked in. They decide to ship fast, and lock-in is what they find when they try to leave.

The trap is structural, not careless. Proprietary platforms make the on-ramp frictionless precisely because the off-ramp is expensive: the agents are defined in a format only that platform runs, the orchestration logic lives in their console, the data has accumulated in their store, and the governance such as it is, is theirs. None of it ports. The convenience that won the pilot is the same convenience that holds the business captive at renewal.

This is not an argument against buying. It is an argument against buying the wrong layer. Some things should be rented from a vendor, gladly. Others, the few that actually differentiate the business, should never sit somewhere a provider can hold them hostage. Telling the two apart is the whole discipline, and it is the subject of the next page.

03 THE THESIS

# You cannot outsource your moat.

The model is a commodity. Frontier capability is converging, prices are falling, and whatever model leads this quarter will be matched next quarter. Renting it is the right call. What you must not rent is the layer that makes your AI yours: your organised context, your governance, your operating model.

**Rent what is converging. Own what differentiates.** Raw model capability, elastic compute, and undifferentiated infrastructure are commodities; paying a vendor to run them is leverage, not risk. But the context your agents reason over, the way you govern them, and the operating model that ties the two together are the only things a competitor cannot simply buy a copy of. Put those on someone else's proprietary rails and you have handed your differentiator to a landlord.

The independence that matters is not "build everything yourself." It is the ability to keep operating when a provider changes its mind. That ability comes from owning the layer that orchestrates and governs, and keeping the commodity underneath it swappable. Own the moat; rent the rest; design the seam between them so the rented part can be replaced without touching the owned part.

> Rent what is converging. Own what differentiates. Design the seam so you can swap the first without losing the second.
>
> THE BUILD-VERSUS-BUY DECISION, RESTATED

04 THE DECISION

# What to own. What to rent.

The test is simple to state and clarifying to apply. Does this layer differentiate you, and is it durable? If yes, own it. If it is converging toward a commodity and switching is cheap, rent it. Most build-versus-buy arguments are really disagreements about which column a given layer belongs in.

OWN IT

## Durable & differentiating

*The layers a competitor cannot buy a copy of*

- **Your organised context** and the patterns learned from a year of use.
- **The governance layer:** permissions, approvals, the audit trail.
- **The orchestration logic** that defines how your agents actually work.
- **The operating model** tying context, governance and people together.

RENT IT

## Converging & swappable

*Commodities where the vendor carries the cost*

- **The frontier model.** Capability is converging; price is falling.
- **Elastic compute** and undifferentiated storage.
- **Managed infrastructure** you would otherwise babysit.
- **Commodity tooling** with real exit options and open formats.

**The error is almost always the same:** teams pour effort into a thin wrapper over a model they do not control, and rent the governance and orchestration that should have been the moat. They build the commodity and rent the differentiator. The columns above are reversible only on paper; in practice, the owned column is where a year of compounding work is hard to undo, so it is exactly where you want to be the owner.

05 WHAT LOCK-IN COSTS

# The switching cost has five parts.

"Lock-in" sounds abstract until you price the exit. The cost of leaving a proprietary agent platform is the sum of five concrete line items, and a vendor who is good at retention has quietly maximised each one. Knowing the anatomy is how you negotiate, and how you avoid signing up for it in the first place.

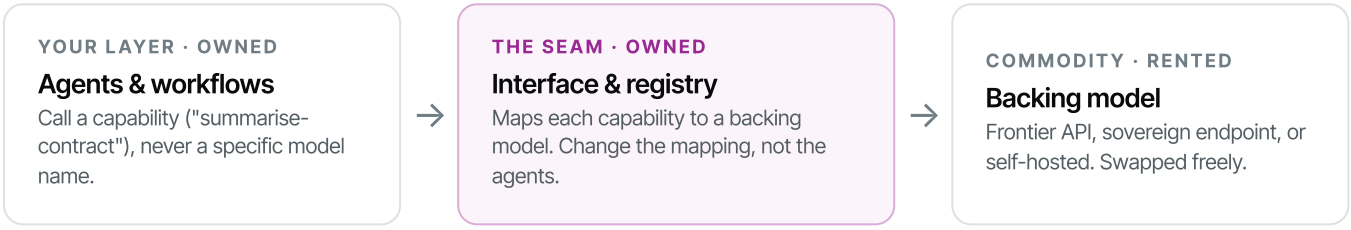
| COST | WHAT IT IS | HOW IT BITES |
|---|---|---|
| **Data gravity** | Context, embeddings and history accumulated in the vendor's store. | Re-ingesting and re-indexing months of knowledge, and often re-earning the freshness you had. |
| **Proprietary format** | Agents and workflows defined in a syntax only that platform runs. | Every agent is rebuilt by hand. The logic does not export; it is re-authored. |
| **Re-verification** | A different model behaves differently, even at the same benchmark score. | Every workflow must be re-evaluated and re-certified before it can be trusted in production again. |
| **Retraining** | Operators and engineers fluent in one console and its quirks. | Productivity drops while the team relearns the tooling and rebuilds its intuition. |
| **Contractual** | Minimums, egress fees, and notice periods in the agreement. | A direct bill for leaving, and a clock that delays the exit you have already decided on. |

**Re-verification is the cost teams forget, and it is often the largest.** Swapping the model underneath an agent is not a config change; a model that scores the same on a public benchmark can fail differently on your work. If the platform makes re-certification cheap and repeatable, switching is survivable. If it does not, the lock-in is the evaluation debt, not the contract. This is why portability and verification are the same problem, addressed on the next page and in Briefing 02.

06 PORTABILITY BY DESIGN

# Make the model swappable before you need to swap it.

Vendor independence is not a contract clause; it is an architecture. Agents call a capability, not a named model. A registry maps that capability to whichever backing model is configured, so the model can change without a single agent changing. The seam is the registry, and it is where independence lives.




*A swap is gated by re-verification: no new backing model reaches production until it re-passes the workflow's evaluation suite.*

This is the same portability architecture set out for regulatory reasons in Briefing 05, here for a commercial one. Because the agents never name a model, a provider's price rise, region restriction or term change becomes a configuration decision rather than a re-platforming project. You move the workload, the agents do not notice, and the re-verification gate confirms the new model still meets the reliability bar before anything ships.

### THE SEAM STAYS YOURS

The interface and registry are part of the owned layer, not the rented one. That is the whole point: the thing that grants independence cannot itself live on a vendor's proprietary rails.

### OPEN BY DEFAULT

The control plane that implements this is open-source. Your engineers can read the seam, not just trust it.

07 WHERE THE PLATFORMS LOCK YOU IN

# The same survey, read for independence.

The major agent platforms are capable and convenient. They are also, by design, places where the differentiating layer becomes theirs. The table reads the published platform survey through one lens: can you keep your moat when you leave? It is not marketing; it is what the architectures permit.

| INDEPENDENCE PROPERTY | RUN ON YOUR OWN INFRASTRUCTURE | SWAP THE MODEL WITHOUT RE-ARCHITECTING | YOU OWN THE GOVERNANCE LAYER |
|---|---|---|---|
| **Microsoft Copilot Studio** | no | partial | no |
| **Salesforce Agentforce** | no | no | no |
| **AWS Bedrock Agents** | partial | partial | no |
| **Google Vertex AI Agents** | partial | partial | no |
| **Open control plane (host-your-own)** | **yes** | **yes** | **yes** |

The pattern is not an accident, and it is not a moral failing on the vendors' part; it is their commercial model. The orchestration runs in their console, the agents are defined in their format, and the governance, the permissions, approvals and audit trail, is a feature of their platform rather than an asset you hold. **Leave, and the moat does not come with you.** The open control plane on the bottom row inverts that: it runs on your infrastructure, keeps the model swappable, and leaves the governance layer in your hands, because it was always yours to begin with.

08 THE BUILD-IT-YOURSELF REALITY

# Own the moat without rebuilding the world.

"Own the differentiating layer" is not a licence to rebuild every component from scratch. That is the opposite mistake: spending the team's scarce attention re-implementing solved problems instead of the few things that are actually yours. Independence is engineered, not maximised.

The owned layer is assembled, not invented. The control plane that orchestrates agents, the router that maps capabilities to models, the permission and audit components, are open-source and already built. You run them on your infrastructure and configure them to your organisation; you do not write a model server or a vector database from first principles. The moat is in the configuration, the context, and the governance you encode, not in re-coding infrastructure that thousands of teams share.

So the honest cost of independence is real but bounded: standing up the open stack, integrating your sources, and building the evaluation suites that make model swaps safe. In return you hold the layer that compounds and stays portable. The alternative, renting the differentiator for convenience, is cheaper this quarter and far more expensive the quarter the provider changes the terms.

**WHAT YOU RUN, NOT WRITE**

Open components for routing, permissions, monitoring and guardrails. Named and detailed in **Briefing 03**.

**THE BOUNDED COST**

Stand up the stack, integrate sources, build the evaluation suites. Then it is yours, and it ports.

**THE COMPANION**

The compounding value of the owned context layer is the subject of **Briefing 07**.

> Independence is not building everything. It is owning the few things a provider could otherwise hold hostage.

09 COMMON MISTAKES

# Four ways independence quietly slips away.

None of these looks like a mistake at the time. Each is a reasonable shortcut that mortgages the future. Each has a correction from the pages above.

MISTAKE 01

### Build the commodity, rent the moat.

Effort spent wrapping a model you do not control, while governance and orchestration sit on the vendor's rails.

**Instead:** own the differentiating layer, rent the converging one. Get the columns right.

MISTAKE 02

### Agents that name a model.

The model identifier is wired through the workflows, so swapping it means rebuilding them.

**Instead:** agents call a capability; a registry maps it to a backing model. The seam stays yours.

MISTAKE 03

### No re-verification path.

Switching is treated as a config change, so a swapped model ships behaving differently and unchecked.

**Instead:** gate every swap on re-passing the evaluation suite. Portability and verification are one problem.

MISTAKE 04

### No exit, priced or planned.

The contract is signed without pricing data egress, format export, or the cost of leaving.

**Instead:** price the five switching costs before you sign, and design the architecture to keep them low.

10 WHAT THIS IS NOT

# Three things this is not.

The argument here is for owning a specific layer, and it is easy to mistake for three nearby positions it rejects.

**Not anti-cloud or anti-vendor.** Renting the model and the compute from a hyperscaler is the right call; they carry the cost of a commodity better than you can. The argument is about which layer to rent, not whether to rent at all.

**Not "build everything."** Re-implementing solved infrastructure is the same error as renting the moat, pointed the other way. You assemble the owned layer from open components; you do not write a model server from scratch.

**Not a one-time procurement decision.** Independence is a property of the architecture, maintained over time: the registry kept current, the evaluation suites kept runnable, the exit kept cheap. It is designed in and then defended, not chosen once at signing.

> The cheapest seat today is the one with the highest cost to leave. Price the exit before you take the seat.

11 REFERENCES

# References. Each claim, anchored.

The companion briefings, the platform survey, and the open stack behind the claims. The full record lives at **teams.cohorte.co/research**.

**The agent-platform survey** (Cohorte, 2026). The independence properties of Microsoft Copilot Studio, Salesforce Agentforce, AWS Bedrock Agents and Google Vertex AI Agents, read for portability and ownership. Companion: Briefing 01.

**The open-source control plane.** context-router, agent-auth, agent-monitor and guardrails implement the owned, host-your-own orchestration and governance layer. github.com/Cohorte-ai.

**Companion: Briefing 05, EU AI sovereignty.** The same portability architecture, set out for regulatory rather than commercial reasons, with the sovereign-cloud gradients.

**Companion: Briefing 02, Verification & evaluation.** The reliability method and evaluation suites that make a model swap safe to ship.

**Companion: Briefing 07, Data & context.** Why the owned context layer is the asset that compounds, and the one that ports.

KEEP THE MOAT, SWAP THE MODEL

# One discovery call. We find where your build-vs-buy line actually sits.

Sixty minutes with our founder. We map your stack onto the own-versus-rent columns, price the switching cost of the platforms you are on or considering, and you leave knowing which layer to own and how to keep the model under it swappable. No deck.

**teams@cohorte.co**

**Cohorte SAS** · Société par actions simplifiée, registered in France · founded September 2022 · Paris & Rabat

The full briefing series, the open-source stack, and the research record at **teams.cohorte.co/trust-and-governance**