



BEYOND THE ESCAPE NARRATIVE

Vendor Lock-In 2.0: We Escaped the Mainframe and Walked Into the Cloud

A Practitioner's Perspective on Cloud Dependency in Enterprise Transformation

Giovanni Leonardi

September 2016

Vendor Lock-In 2.0: We Escaped the Mainframe and Walked Into the Cloud

Transformation · No. 248

Originally written September 2016 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

"We did not escape lock-in. We refinanced it."

The Freedom We Were Sold

For the better part of a decade, the case for moving to the public cloud has rested on a quiet promise that almost nobody states aloud but nearly everybody believes: that we are, at last, escaping the vendor. The mainframe years taught a generation of technology leaders exactly what dependency felt like — the annual licence true-up, the proprietary middleware, the hardware refresh you could not refuse, the systems integrator whose day rate seemed to rise in perfect step with your inability to do without them. So when the hyperscalers arrived with consumption billing and the language of no capital, no commitment, it read to many boards like liberation.

I have sat through enough migration steering committees these past two years to recognise the pattern, and I have come to believe we have misread it. We did not escape lock-in. We refinanced it.

The argument I want to make is narrow and, I think, uncomfortable: the move from the licensed estate to the public cloud does not reduce dependency on a single supplier. It changes the form that dependency takes — from something you could see on a balance sheet to something buried in your architecture, where it is far harder to price and far harder to leave.

What the Old Lock-In Actually Was

It is worth being precise about the thing we were trying to get away from, because the escape only makes sense if you name the cage. The lock-in of the licensed era was contractual and capital. It lived in perpetual licences already paid for, in multi-year support agreements, in the sunk cost of hardware sitting in a data centre you also paid to cool. It was expensive and it was rigid — but it had one redeeming quality that we undervalued at the time. It was visible. It had a renewal date. It had a line in the budget with a number next to it. A CFO could look at the Oracle or IBM relationship and know, more or less, what leaving would cost and when the decision fell due.

That visibility is precisely what we have given up.

The New Lock-In Is Architectural

The organisations moving fastest into the cloud are not the ones reducing their dependency on a single supplier. They are the ones changing where that dependency lives — from the contract, where governance can see it, to the architecture, where it usually cannot.

The new dependency does not announce itself at renewal. It accumulates, decision by reasonable decision, every time a team reaches for a managed service to go faster. And going faster is the whole point — nobody adopts a hosted queue or a serverless function or a proprietary data warehouse to be difficult; they do it because rebuilding those capabilities in-house would be indefensible. But each of those choices is also a quiet statement about who you can no longer leave.

Four forces are doing the real work here, and none of them appears in a procurement framework:

- **Managed-service entanglement.** The moment your application logic assumes a specific provider's queue, function runtime, or identity layer, portability stops being an architectural property and becomes a rewrite. The convenience and the lock-in are the same feature; you cannot take one without the other.
- **Data gravity.** Compute is portable in a way that petabytes are not. Once the data lives with the provider, everything that wants to be near the data follows it there, and the centre of gravity of your entire estate quietly relocates onto someone else's platform.
- **Egress economics.** It costs little to move data in and conspicuously more to move it out. This is not an accident of pricing; it is a moat with a meter on it. The bill you would face to repatriate at scale is the real exit cost, and almost no migration business case models it.
- **Skills concentration.** Within two years of a serious cloud programme, your best engineers think fluently in one provider's primitives. That fluency is an asset until the day you want a second option, at which point it is a dependency wearing the costume of a capability.

Why the Textbook Answer Fails

The procurement-era remedy for lock-in was dual sourcing: keep a second supplier credible, and the threat of switching disciplines the first. It is sound advice for commodities, and it is nearly useless here.

You cannot dual-source an architecture the way you dual-source a hardware supplier. The "second source" for a deeply integrated cloud estate is not another vendor's catalogue — it is a parallel implementation of your own systems, maintained at cost, forever, against the day you might need it. Almost no organisation will fund that, and the few that try discover they are paying twice to move half as fast, which is its own kind of failure. The abstraction layers sold as insurance — the promise that a portability framework will let you lift and shift between providers — tend to deliver the lowest

common denominator of every platform and the distinctive advantages of none. You end up paying a premium to not use the very services you moved to the cloud to get.

The honest conclusion is that lock-in here is not a defect to be engineered away. It is a structural property of buying capability rather than building it. The question is not whether to accept it. The question is whether you accepted it on purpose.

Choosing Your Dependency Deliberately

That reframing is the whole of my advice, and it is more practical than it sounds.

1. Separate the reversible from the irreversible. Some dependencies are cheap to unwind — a compute layer, a container runtime, a storage tier with modest volumes. Others are effectively permanent — the data platform, the identity fabric, the services your core transactions run through. Treat these two categories completely differently. Spend your portability budget only where reversal is both plausible and worth it, and stop pretending the rest is optional.
2. Price the exit before you sign the entrance. Every material adoption decision should carry an estimate — however rough — of what leaving would cost in egress, rebuild, and retraining. You will rarely act on it. That is not the point. The point is that the number exists, it is written down, and the dependency is on the register where governance can see it rather than discovered years later by an architect with bad news.
3. Concentrate deliberately, don't drift. If you are going to be dependent — and you are — be dependent on purpose, on a platform chosen for reasons you can defend, not on whichever services each team happened to reach for under delivery pressure. Accidental lock-in has all the cost of the deliberate kind and none of the benefit.

We are not, it turns out, in the business of avoiding vendors. We never were. We are in the business of choosing which ones to depend on, understanding what that choice costs, and making it with our eyes open rather than closed. The mainframe taught us to fear dependency. The cloud is teaching us the more useful lesson: that the dependency you chose deliberately, and priced honestly, is a strategy — and the one you backed into while congratulating yourself on your freedom is just the old trap in newer, more comfortable clothes.