



BEYOND PORTABILITY

Lock-In 2.0: From the Mainframe to the Cloud, and the Dependence We Build Ourselves

A Practitioner's Perspective on Vendor Dependence in Enterprise Cloud Transformation

Giovanni Leonardi

November 2016

Contents

1. Executive Summary	3
2. A Familiar Villain in New Clothes	3
3. Why the Last Lesson Did Not Transfer	4
4. What Lock-In 2.0 Actually Is	4
5. The Forces That Hold the Pattern in Place	5
6. The Gap Between the Business Case and the Reality	6
7. Is Lock-In Even the Right Frame?	6
8. Toward Deliberate Dependence	7

Lock-In 2.0: From the Mainframe to the Cloud, and the Dependence We Build Ourselves

Transformation · No. 250

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

"Dependence is not the exception in this industry; it is the standing condition."

1. Executive Summary

Vendor lock-in is not a new anxiety. A generation of technology leaders learned it with the mainframe and relearned it with the great enterprise software suites, and most of us carry the scars of a renewal negotiation conducted from a position of no leverage. What is new is the shape the dependency now takes. The move to public cloud is often sold, in part, as an escape from lock-in — a world of commodity infrastructure, elastic and interchangeable. The reality I have observed is close to the opposite. Cloud has not abolished lock-in; it has changed its character, from something contractual and visible into something architectural and quietly cumulative.

This essay argues three things. First, that the dependency of the cloud era is more insidious than its predecessors precisely because we choose it willingly, one convenient managed service at a time, rather than signing up for it in a single conspicuous contract. Second, that the structural forces sustaining this pattern — data gravity, the economics of managed services, the skills market, the pressure to move fast — are strong enough that exhortations to "stay portable" are largely wishful. And third, that the honest response is not to pretend we can avoid dependence but to choose it deliberately: to treat exit cost as a number we manage rather than a principle we invoke. The organisations that will fare best are not the ones that avoid lock-in but the ones that know exactly how much of it they have bought, and why.

2. A Familiar Villain in New Clothes

Every experienced technology leader has a lock-in story, and most of them are old. Mine begin, as many do, with the mainframe: the proprietary everything, the maintenance line on the budget that only ever went up, the sure knowledge that the cost of leaving exceeded the cost of staying, forever. Then came the enterprise application suites, which taught the same lesson in a different dialect. The database, the ERP, the middleware — each arrived promising to be the last integration you would ever need, and each became a set of foundations you could not move without demolishing the house.

So when the industry began, a few years ago, to talk about cloud as a liberation — infrastructure as a utility, drawn from the wall like electricity, switched from one supplier to another at will — those of us with long memories were entitled to a degree of scepticism. The utility metaphor is seductive and, at the level of raw compute, not entirely false. A virtual machine is close enough to a commodity. If all you rent is undifferentiated capacity, you can, in principle, take your workloads elsewhere.

But almost nobody rents only undifferentiated capacity, and the providers have arranged matters so that you would be foolish to. The value, and the trap, lie one layer up.

3. Why the Last Lesson Did Not Transfer

There is a puzzle worth sitting with. The people leading cloud programmes today are, very often, the same people who lived through the enterprise-software lock-in of the previous decade. They know the story. They negotiated the punishing renewals; they costed the migrations that never happened because the exit was too expensive to contemplate. If any cohort should have been inoculated against walking into dependency, it is this one. And yet here we are.

Why did the lesson not transfer? Partly because the new dependency does not look like the old one. The mind that is braced for a bad contract is not braced for a bad architecture, and the two feel nothing alike in the moment. Nobody sits in a renewal meeting when they adopt a managed queue; they simply merge a pull request. The absence of a contractual flashpoint removes the very trigger that would have made a scarred veteran wary.

Partly, too, because the cloud narrative was explicitly framed as the cure for the last disease. Lock-in was the named villain the cloud was sold to defeat, and it is genuinely hard to stay vigilant against a risk when the whole rationale of your programme is that you have already escaped it. The framing disarmed the very instinct that experience should have sharpened. We were looking for lock-in in the shape it last took, and it arrived in a shape we had not learned to fear.

4. What Lock-In 2.0 Actually Is

The lock-in of the mainframe era was contractual. It lived in the licence, the renewal date, the maintenance agreement. It was unpleasant, but it was legible: you could see it on a page, put a number against it, and brief a board on it. Its very visibility made it, in a strange way, manageable. You knew the day the negotiation was coming.

The dependency of the cloud era is architectural, and that is a different animal. It does not arrive in a contract. It accretes through a thousand small, sensible engineering decisions, each of which is locally correct. The team chooses the provider's managed database rather than running its own, because operating a database well is genuinely hard and the managed service is genuinely good. It adopts the provider's queue, its object store, its identity service, its proprietary data warehouse, its

functions-as-a-service. Every one of those choices is defensible on its own merits. Collectively, they weave the organisation into the fabric of a single supplier so thoroughly that the question "could we leave?" stops having a meaningful answer.

The old lock-in was something you signed. The new lock-in is something you build — commit by commit, service by service, each decision reasonable, the sum of them a wall.

This is why I call it lock-in 2.0. It is not worse in every respect than what came before — the services really are better, and the productivity gains are real. But it is harder to see, harder to price, and much harder to argue against in the moment, because resisting it means asking a good engineer to choose a harder path for the sake of an optionality nobody is currently paying for.

"Portability is a cost you pay continuously for an option you may never exercise. That is precisely why it is the first thing to be cut."

5. The Forces That Hold the Pattern in Place

If this were merely a matter of insufficient discipline, it would be easy to fix. It is not. The pattern persists because a set of structural forces make dependence the path of least resistance, and no amount of architectural good intention fully overcomes them.

The first is data gravity. Data is heavy — not physically, but economically. Once a large and growing body of data lives with a provider, and once the analytics, the pipelines and the applications have arranged themselves around it, moving it becomes an undertaking out of all proportion to the moving of compute. The provider understands this perfectly, which is why ingress is free and egress is not. Your data is welcomed in without charge and invited to leave only at a price. Over time, the centre of gravity of the whole estate shifts to wherever the data sits, and the data does not want to move.

The second is the economics of managed services. The entire commercial logic of the cloud is to sell you not raw capacity but capability — the higher up the stack you buy, the more the provider captures, and the more differentiated, which is to say proprietary, the service becomes. This is not a conspiracy; it is a business model, and a good one. But it means the provider's incentives and the customer's portability are structurally opposed. The provider is rewarded precisely to the extent that you build on the things you cannot easily take with you.

The third is the skills market. Engineers build their careers on specific ecosystems. They accumulate expertise, certifications and instincts in one provider's world, and that human capital becomes its own form of gravity. An organisation that has spent three years hiring and training for one platform cannot casually announce a move to

another; the cost is not only technical but in the knowledge, and the goodwill, of its people.

The fourth is simply pace. The reason most organisations went to the cloud was speed — the promise of moving faster than their own procurement and provisioning cycles allowed. Every hour spent building an abstraction layer to preserve portability is an hour not spent shipping the thing the business actually asked for. When the pressure is to deliver, and it always is, the portability work is what gives.

- Data gravity makes the estate physically reluctant to move.
- The managed-services model aligns the provider's profit with your dependence.
- The skills market turns your own people into a source of platform inertia.
- The pressure for pace ensures portability is always the work that gets deferred.

Set these four forces alongside one another and the conclusion is uncomfortable but clear: lock-in in the cloud era is not an accident or a failure of will. It is the equilibrium the system tends towards. Fighting it with slogans about staying cloud-agnostic is like fighting a current with a strongly worded sign.

6. The Gap Between the Business Case and the Reality

Here is where the pattern becomes a transformation problem rather than merely an architectural one. The business cases I have seen for major cloud programmes lean heavily on the language of agility, optionality and freedom. The board is told that the organisation is buying flexibility — the ability to scale up and down, to avoid capital lock-up, to move with the market. Optionality is, quite often, one of the headline justifications.

And then the operating reality manufactures the opposite. The programme that was sold as a purchase of flexibility becomes, in its execution, a steady accumulation of dependency. Nobody lied. The business case was sincere. But the incentives inside delivery all point the other way, and eighteen months later the organisation is more tightly bound to a single supplier than it ever was to its old data centre — and, unlike the data centre, this supplier can change its prices with a quarter's notice.

We approve cloud programmes on a promise of optionality and then spend the next two years engineering it away, one sensible decision at a time. The intent and the outcome are not merely different; they are opposites.

This is the characteristic gap of our era: not a gap between a good plan and poor execution, but a gap between what the transformation was believed to be buying and what the day-to-day mechanics of it actually deliver. It is the gap that the enthusiasm papers over, and it is the one a serious practitioner has to name out loud, however unwelcome that makes them in the room.

7. Is Lock-In Even the Right Frame?

Having made the case for alarm, let me now argue against myself, because the honest position is more nuanced than the alarmist one.

The truth is that some dependence is not merely unavoidable but desirable. The entire proposition of the cloud is that someone else runs the undifferentiated heavy lifting better than you can, and that you are freed to spend your scarce talent on the things that actually distinguish you. To insist on perfect portability is to forgo most of that value — to run everything on the lowest common denominator of services that happen to exist everywhere, and thereby to buy an insurance policy so expensive it defeats the purpose of the journey.

So the mature question is not "how do we avoid lock-in?" — that framing leads either to paralysis or to a self-defeating asceticism. The mature question is: which dependencies are we taking on, how much would each cost us to reverse, and are we being paid enough in capability and speed to justify that price? Lock-in, put this way, is not a sin to be avoided but a cost to be underwritten — like any other risk an organisation knowingly carries.

Posture	How it treats dependence	Where it usually ends
Lock-in denial	Pretends portability is free and achievable	Dependency accrues invisibly; nobody is accountable
Portability absolutism	Refuses proprietary services on principle	Forgoes the value; loses the race on cost and speed
Deliberate dependence	Prices each dependency and chooses knowingly	Dependence is real but bounded, owned and understood

8. Toward Deliberate Dependence

What, then, does a grown-up posture look like? Not a portability mandate, and not a shrug. Something in between, and more demanding than either.

It begins by making exit cost a number rather than a principle. For each significant platform dependency, the organisation should be able to state, roughly, what it would cost and how long it would take to unwind — the way a treasurer can tell you the cost of unwinding a hedge. That number need not be small. It simply needs to be known, owned, and reviewed, so that dependence is a position the organisation holds on purpose rather than a fact it discovers during a price rise.

It continues by grading dependencies deliberately. Not everything deserves the same treatment. The things that are genuinely commoditised and easy to move can be left entirely to the provider's best services without a second thought. The things that sit closest to the organisation's distinctive value, or that hold its most gravitationally heavy data, warrant more caution — a deliberate choice about how much proprietary depth to accept there, made with eyes open, and revisited as the stakes change.

1. Name the dependencies. Maintain an honest register of where the organisation is bound, and to what depth. You cannot manage a risk you refuse to look at.

2. Price the exits. For each material dependency, hold an estimate of the cost and time to reverse it, and keep it current.
3. Grade deliberately. Accept deep, proprietary dependence where the value is high and the data is heavy; insist on shallower, more portable choices where the value is low and the switching stakes are real.
4. Put an owner on it. Dependence with no owner is dependence nobody is managing. Someone senior must hold the portfolio of exit costs as a standing responsibility, not a one-off architecture-review footnote.

None of this is glamorous, and none of it will appear on a transformation programme's headline metrics. But it is the difference between an organisation that has chosen its dependencies and one that has merely accumulated them.

The journey from the mainframe to the cloud has taught the same lesson twice, in two dialects. Dependence is not the exception in this industry; it is the standing condition. What changes, from one era to the next, is only whether we see it coming. The mainframe made us pay for the privilege in a contract we could at least read. The cloud invites us to build the same wall ourselves, brick by convenient brick, and to be surprised by it later. The task of the practitioner is to refuse the surprise — to know the wall is being built, to decide how high to build it, and to keep, always, an honest reckoning of what it would cost to climb back over.