A Two-Dimensional Taxonomy of Cloud Service Integrations: Modeling Architectural Coupling and Portability Friction in Managed Infrastructures

Abstract

Modern software engineering increasingly relies on managed cloud services to accelerate delivery and reduce operational overhead. However, the adoption of these services introduces varying degrees of architectural coupling and vendor lock-in. This paper presents a formal, two-dimensional taxonomic framework to classify cloud services along two orthogonal axes: **Architectural Coupling** ($\mathcal{C}_{arch}$) and **Exit Friction** ($\mathcal{F}_{exit}$). By evaluating core Amazon Web Services (AWS) offerings through this systematic model, we transition cloud selection from a heuristic-based practice to a quantifiable software engineering discipline. This taxonomy enables system architects to mathematically assess the long-term structural risks and migration overhead associated with cloud-native system designs. We then extend the model to the question of **European digital sovereignty**: using Donella Meadows' twelve leverage points, we rank the interventions that could most efficiently move EU governments and enterprises off US hyperscalers, and we derive several economic consequences, including a *lock-in floor*: policy acting on data transport alone reaches on average 18 percent of the measured exit tax, and its per-service reach falls below 30 percent precisely on the deeply locked services that carry the policy problem, and the identification of architectural lock-in as a negative externality that justifies disclosure-and-standards intervention over subsidy.

---

1. Introduction

The discipline of software engineering strives to manage complexity through abstraction and modularity. In cloud-native systems, this modular boundary often extends beyond local codebase components to external, third-party managed services. While managed services provide high-level abstractions (serverless execution, global content distribution, automated database scaling), they challenge classic software design principles regarding decoupling and portability.

Historically, decisions to adopt managed cloud services have been guided by subjective "best practices" or binary arguments regarding "vendor lock-in." Replacing dogmatic assertions with formal, auditable classification is the aim here. We propose a framework that evaluates cloud services based on two distinct structural characteristics:

1. The physical location of the integration boundary (the database wire protocol vs. the application runtime compilation).
2. The availability of equivalent external protocol specifications.

By mapping services along these dimensions, engineering organizations can systematically calculate the technical debt, migration risk, and architectural agility of their systems.

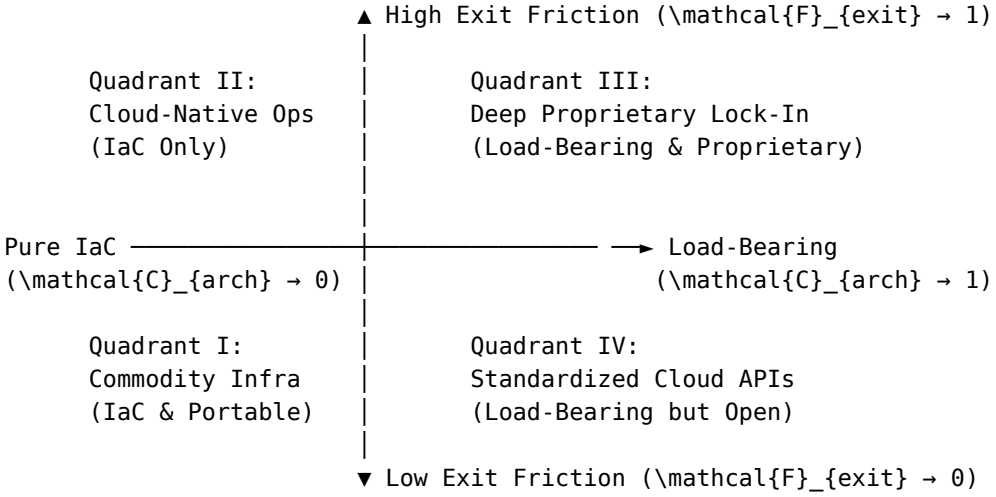
Three literatures border this exercise, and the delta against each locates the contribution. Qualitative surveys of cloud vendor lock-in (Opara-Martins, Sahandi and Tian 2016 is the most cited) document the phenomenon and practitioner perceptions without a scoring model; the portability governance work around the SWIPO codes of conduct and the EU Data Act addresses the contractual and data-transport layer, which §7.2 shows is bounded at a minority share of the exit tax; and the switching-cost economics descending from Klemperer (1995) and Farrell and Klemperer (2007) treats the switching cost as a scalar primitive. This paper's contribution is the decomposition of that scalar into an auditable, service-level production function, plus the mapping from the resulting scores to intervention leverage.

2. Theoretical Framework and Definitions

To formalize the classification, we define a cloud dependency D within an application system S as a tuple:

$$D = \langle \mathcal{C}_{arch}, \mathcal{F}_{exit} \rangle$$

Where $\mathcal{C}_{arch}$ represents the degree of **Architectural Coupling** and $\mathcal{F}_{exit}$ represents the degree of **Exit Friction**.



2.1 Dimension 1: Architectural Coupling ($\mathcal{C}_{arch}$)

Architectural Coupling measures how deeply an application's compile-time or runtime dependencies are embedded in a service's interface, regardless of who owns that interface: interface ownership and openness are priced separately, by $\mathcal{P}_{spec}$ (§2.3). An open protocol can be load-bearing (cache calls inline throughout business logic) and a proprietary one peripheral (a provisioning API touched only by deployment tooling). We define this on a continuous scale $[0, 1]$:

- **Infrastructure-Only Coupling ($\mathcal{C}_{arch} \to 0$):**

The integration lives at the Infrastructure-as-Code or deployment-orchestration layer (e.g., Terraform, CloudFormation), or reaches the application only through a mediation layer it already owns (a data-access layer over a standard SQL wire protocol, a mounted POSIX filesystem). The application binary remains unaware of its execution environment.

- **Application-Level Coupling ($\mathcal{C}_{arch} \to 1$):**

The application source code directly imports provider-specific software development kits (SDKs), compiles against proprietary runtime interfaces, or conforms to provider-defined event-handler structures. The application cannot compile or execute without resolving these dependencies, or scatters service calls inline through business logic at many call sites. Two consequences of this anchoring recur in §3: RDS (0.10) scores far below ElastiCache (0.60) although both speak open wire protocols, because SQL typically sits behind a data-access layer while cache calls sit inline at their call sites; and ECS (0.30) scores below EKS (0.65) because task definitions are deployment-layer artifacts while Kubernetes estates embed API objects, Helm charts, operators, and sidecars into the application's delivery fabric.

2.2 Dimension 2: Exit Friction ($\mathcal{F}_{exit}$)

Exit Friction measures the computational, financial, and labor overhead required to replace a given service with an equivalent alternative (either self-hosted or hosted by an alternative cloud provider). We define this on a continuous scale $[0, 1]$:

- **Low Exit Friction ($\mathcal{F}_{exit} \to 0$):**

The service implements an open, widely adopted industry standard or a commoditized protocol. Migration requires zero application code modifications, focusing instead on data replication, network routing configuration, and DNS updates.

- **High Exit Friction ($\mathcal{F}_{exit} \to 1$):**

The service utilizes a highly proprietary data model, security paradigm, or execution model. No direct open-source equivalent exists. Escaping the service necessitates a fundamental redesign of database schemas, application business logic, or organizational authorization structures.

2.3 A Composite Model for Exit Friction

Rather than assigning $\mathcal{F}_{exit}$ by intuition, we decompose it into three measurable sub-factors and combine them with a non-linear model. This makes each score auditable and reproducible.

- **Data Gravity ($\mathcal{D}_{grav} \in [0, 1]$):** the mass of stateful data that must physically cross network boundaries, governed by egress cost and transfer time.
- **Protocol Proprietary-ness ($\mathcal{P}_{spec} \in [0, 1]$):** the degree of provider ownership over the interface specification. An open or de facto standard (SQL, Redis, the S3 API) approaches 0 a purely custom API (DynamoDB, KMS) approaches 1.
- **Migration Friction ($\mathcal{M}_{fric} \in [0, 1]$):** the structural difficulty of translating schemas, configurations, and architectural patterns, net of available tooling.

A simple weighted average is inadequate because code-level lock-in is non-linear: if *either* the protocol is proprietary *or* migration tooling is absent, the code is effectively trapped regardless of dataset size. We therefore combine $\mathcal{P}_{spec}$ and $\mathcal{M}_{fric}$ through a logical-OR ("noisy-OR") term and add data gravity linearly:

$$\mathcal{F}_{exit} = w \cdot \mathcal{D}_{grav} + (1 - w) \cdot \underbrace{\left[\, 1 - (1 - \mathcal{P}_{spec})(1 - \mathcal{M}_{fric}) \,\right]}_{\text{logical lock-in } \mathcal{L}}$$

The model descends from the author's 2025 *IaaS Dependency Scorecard* [Fermigier 2025b], a linear additive score with an explicit alternatives term; the present form replaces the linear combination with the noisy-OR (which the scorecard's own worked cases motivate: either a proprietary protocol or absent tooling suffices to trap the code) and moves the alternatives assessment into the anchoring of $\mathcal{M}_{fric}$ (Annex C). Section 8 reports the rank concordance between the two models. The lock-in term $\mathcal{L}$ spikes toward 1.0 whenever either input is high, capturing the binary character of code-level dependency. We set $w = 0.30$ as a calibrated judgment, assigning 70% of the weight to logical/code lock-in and 30% to physical data transport, reflecting that in modern enterprises code rewrites dominate the cost of data transit; §8 sweeps w over $[0.10, 0.50]$ and perturbs every judgment score, and reports which conclusions survive. Every $\mathcal{F}_{exit}$ value in this paper is computed from this formula; the full input decomposition appears in **Annex A**.

---

3. Taxonomic Classification Matrix

By evaluating thirty AWS services through this 2D framework, we segment the catalog into four distinct quadrants, each representing a unique risk-reward profile for software systems. The sample

was chosen before scoring as the most widely adopted services per major catalog category (compute, block and object storage, network, load balancing, relational and NoSQL data, warehousing, streaming, interactive query, integration, identity, security, observability, IaC and autoscaling, ML, CDN), and the adoption claim is externally anchored: the sample covers sixteen of the top twenty AWS services in the 2018 practitioner-survey ranking [Jefferson Frank 2018] (omissions: EFS, SES, CloudTrail, Glacier) and fourteen of the top twenty by measured spend on the Vantage cost leaderboard as of March 2026 [Vantage 2026] (omissions, all niche analytics and security services plus physical transport: OpenSearch, MSK, GuardDuty, Config, EMR, Direct Connect), and telemetry puts Lambda alone at 65 percent of AWS customers [Datadog 2025]. The scope is a single vendor, AWS as the market leader, with the Azure and GCP analogues queued as replication work (§8). Quadrant assignment splits each axis at the 0.5 midpoint; all $\mathcal{F}_{exit}$ values are computed from the composite model of §2.3.

AWS Service	Architectural Coupling ($\mathcal{C}_{arch}$)	Exit Tax ($\mathcal{F}_{exit}$)	Taxonomic Quadrant	Primary Interface / Protocol
Amazon Aurora	0.15	0.49	Quadrant I*	MySQL/ PostgreSQL Wire (Aurora extensions)
Amazon Athena	0.25	0.47	Quadrant I*	Standard SQL (Trino) / Glue Catalog
Amazon ECS	0.30	0.41	Quadrant I	Proprietary Task Definitions
Amazon Route 53	0.05	0.40	Quadrant I	Standard DNS / Alias Routing
Amazon CloudFront	0.10	0.35	Quadrant I	Standard HTTP / CDN Semantics
Amazon RDS	0.10	0.28	Quadrant I	Standard SQL Wire Protocols
Amazon VPC	0.00	0.20	Quadrant I	Standard IPv4/ IPv6 Networking
Amazon EC2	0.05	0.16	Quadrant I	POSIX / Linux / x86_64
Amazon EBS	0.10	0.73	Quadrant II	POSIX Block Device / Proprietary Volume Format
AWS IAM	0.05	0.72	Quadrant II	Proprietary RBAC / ABAC Policies
AWS KMS	0.10	0.70	Quadrant II	Proprietary Cryptographic Envelope

AWS Service	Architectural Coupling ($\mathcal{C}_{arch}$)	Exit Tax ($\mathcal{F}_{exit}$)	Taxonomic Quadrant	Primary Interface / Protocol
Amazon CloudWatch	0.40	0.67	Quadrant II*	Proprietary Telemetry APIs (OTel escape)
Amazon Redshift	0.15	0.64	Quadrant II	PostgreSQL-derived SQL Dialect
AWS Secrets Manager	0.35	0.63	Quadrant II	Proprietary Secrets API
AWS CloudFormation	0.05	0.61	Quadrant II	Proprietary IaC Template Language
AWS Auto Scaling	0.30	0.61	Quadrant II	Proprietary Scaling Policies
AWS ELB	0.25	0.51	Quadrant II*	Standard L4/L7 Semantics / Proprietary Integrations
Amazon DynamoDB	0.90	0.90	Quadrant III	Proprietary NoSQL HTTP API
Amazon Cognito	0.70	0.77	Quadrant III	Proprietary Identity Pools (partial OIDC)
Amazon SageMaker	0.75	0.77	Quadrant III	Proprietary ML Pipeline SDK
AWS Step Functions	0.95	0.70	Quadrant III	Amazon States Language (ASL)
Amazon EventBridge	0.85	0.68	Quadrant III	EventBridge Schema / Bus API
Amazon Kinesis	0.65	0.67	Quadrant III	Proprietary Streaming API (Kafka-adjacent)
AWS Lambda (Standard)	0.80	0.66	Quadrant III	Lambda Runtime Handler API
Amazon API Gateway	0.55	0.62	Quadrant III*	Proprietary Route & Authorizer Definitions
Amazon SNS	0.55	0.56	Quadrant III*	Proprietary Pub/Sub API
Amazon SQS	0.50	0.47	Quadrant IV*	Proprietary HTTP Poll API (visibility-timeout semantics)

AWS Service	Architectural Coupling ($\mathcal{C}_{arch}$)	Exit Tax ($\mathcal{F}_{exit}$)	Taxonomic Quadrant	Primary Interface / Protocol
Amazon S3	0.70	0.43	Quadrant IV*	De Facto Standard S3 API
Amazon EKS	0.65	0.12	Quadrant IV	Standard Kubernetes API
Amazon ElastiCache	0.60	0.10	Quadrant IV*	Open-Source Redis/Memcached

Boundary services. Applying the model recalibrates two services relative to a purely intuitive placement. **Amazon Route 53** and **Amazon SQS** both sit near the 0.5 frontier ($\mathcal{F}_{exit} = 0.40$ and 0.47): although each carries proprietary integrations (Alias records, AWS-specific queue semantics), open standards and mature tooling keep their aggregate exit tax below the midpoint, pulling them into the low-friction half of the plane. The composite model thereby separates *perceived* lock-in from *measured* lock-in. Asterisks in the quadrant column mark the nine assignments whose quadrant survives fewer than 90% of the score-noise draws of §8; frontier-adjacent services like Route 53 that are noise-stable but flip under extreme weights carry no mark and are discussed in §8′s weight sweep.

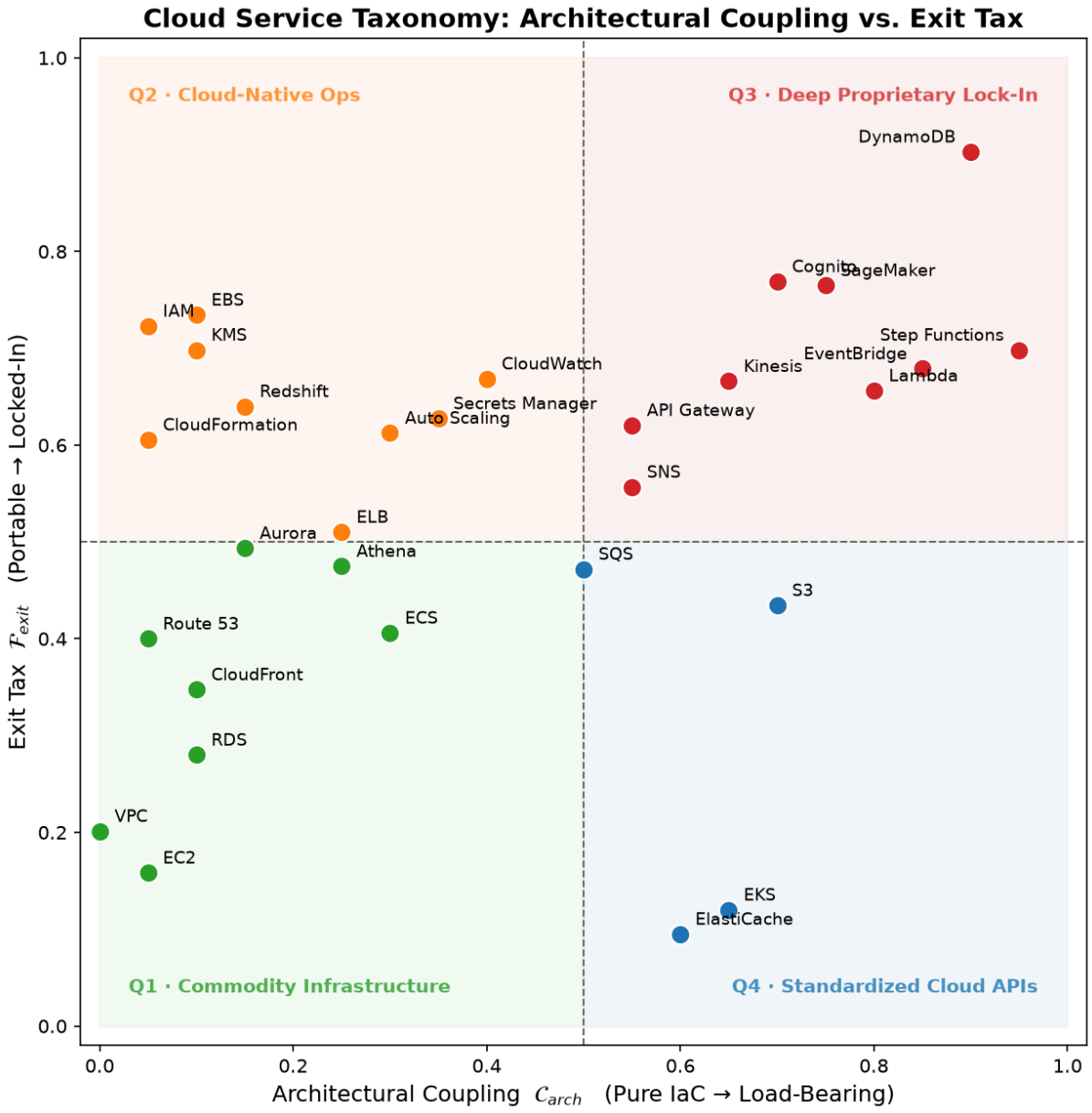


Figure 1. The thirty scored services plotted on the $(\mathcal{C}_{arch}, \mathcal{F}_{exit})$ plane, colored by quadrant. Generated from `notes/services.csv` by `verify/plot_quadrants.py`, which recomputes each $\mathcal{F}_{exit}$ from the §2.3 formula and asserts agreement with the tabulated values.

A recency check on the anchors separates discourse from money, and gives the sample a third, negative anchor: what we exclude. We assembled a corpus of six editorial "top AWS services" rankings published between July 2025 and July 2026 (archived verbatim, with sources and canonicalization rules, in `data/hype_corpus.json`) and counted, for every service in the union of the corpus, the spend top 25, and the sample, how many of the six lists mention it (Figure 2). Three regimes emerge. The canon (EC2, S3, RDS, Lambda, VPC, IAM, SNS) appears in all six lists, with EC2, S3, and RDS at the top of both measured rankings; seven of the 2018 survey's top ten are still in the spend top 25 eight years later, and this persistence of the head is itself an observable consequence of the switching-cost regime the paper models. The plumbing (Direct Connect, OpenSearch, MSK, GuardDuty, Config, EMR, CloudTrail, DocumentDB, Backup) sits inside the spend top 25 yet collects at most one editorial mention in six: where the money is, the discourse is thinnest, and seven of these nine are among the sample's named top-twenty omissions. The hyped

tail runs the other way: the GenAI-era items (Bedrock, Amazon Q, Trainium, Kiro, S3 Vectors) appear in three of the four lists dated 2026, in neither list dated 2025, and in neither measured ranking; Bedrock's spend is growing fast from a base that has yet to reach the top 25. The sample is drawn from the first two regimes and deliberately excludes the third: scoring services whose migration tooling does not yet exist would substitute editorial judgment for the measurements the model wants. One caveat keeps the figure honest: the spend axis undercounts cheap ubiquitous services (Lambda serves 65 percent of AWS customers [Datadog 2025] and ranks 24th by dollars; IAM and SNS are near-free and spend-unranked; VPC is absent from the 2018 survey's product list), so spend anchors the head of the adoption distribution and says little about the free tail.

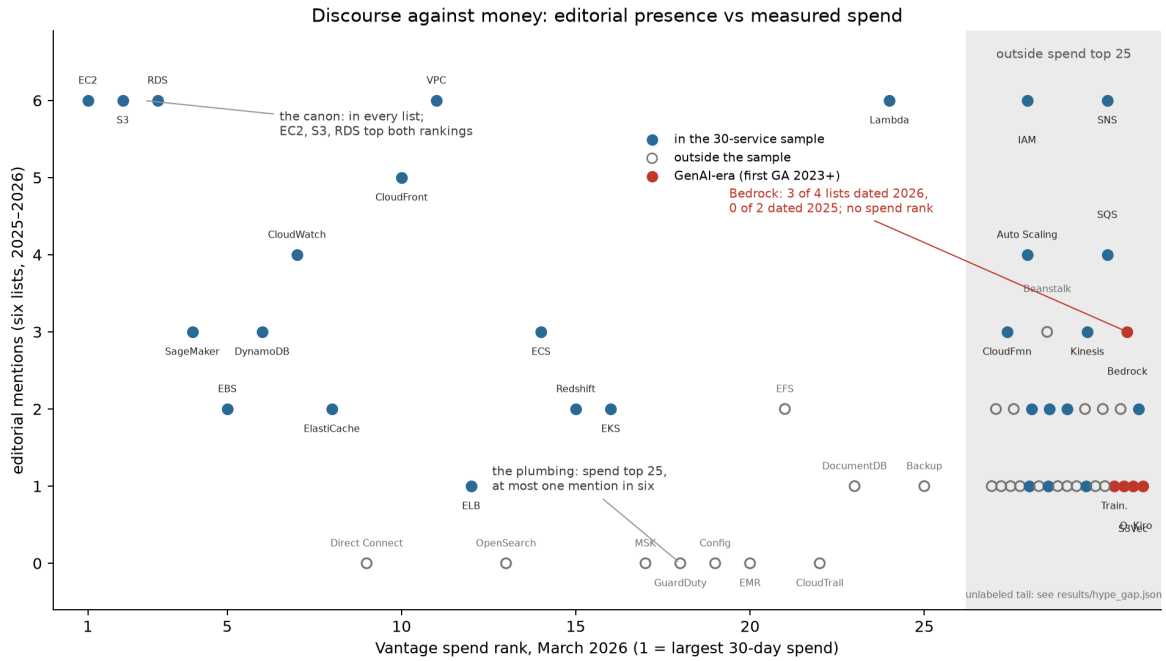


Figure 2. Editorial presence (mentions across six "top AWS services" lists, 2025–2026) against measured spend rank (Vantage, March 2026). Filled points are in the thirty-service sample; red marks GenAI-era services (first GA 2023 or later); the shaded band holds services outside the spend top 25, with the one- and two-mention tail unlabeled (full table in `verify/results/hype_gap.json`). Generated by `verify/hype_gap.py` from the archived corpus `data/hype_corpus.json`; the script asserts each regime claim as stated in the text.

---

4. Deep-Dive Analysis of Taxonomic Quadrants

4.1 Quadrant I: Commodity Infrastructure (Low $\mathcal{C}_{arch}$, Low $\mathcal{F}_{exit}$)

Quadrant I services represent the standard virtualized hardware layer. These services isolate application logic from the underlying hardware hosting provider.

- **Amazon RDS (PostgreSQL/MySQL):** Because RDS provisions standard open-source engines, the application code connects via open, standardized protocols. Replacing RDS with a self-hosted instance or a managed equivalent on another cloud involves no code modifications. Exit friction is limited to data migration velocity (bounded by network bandwidth and write-throughput during database synchronization).
- **Amazon EC2:** Virtual machines execute compiled binaries targeting standard architectures (e.g., x86_64 or ARM64). The application code interacts only with standard operating system APIs. Porting workloads off EC2 requires zero code changes.

4.2 Quadrant II: Cloud-Native Operations (Low $\mathcal{C}_{arch}$, High $\mathcal{F}_{exit}$)

Quadrant II services are characterized by minimal code footprint but profound structural lock-in at the organizational, compliance, and deployment layers.

- **AWS Identity and Access Management (IAM):** While application binaries do not compile IAM-specific logic, the entire access-control matrix, service authorization boundary, and secure execution environment are declared using AWS's proprietary policy engine. Migrating this configuration to an alternative cloud's identity platform (e.g., Azure Active Directory or Google Cloud IAM) cannot be executed via standard migration tools; it requires a manual, complete mapping and re-implementation of the security architecture.
- **AWS Key Management Service (KMS):** Applications rely on KMS for envelope encryption and data-at-rest protection. Because the master cryptographic keys physically reside within AWS hardware security modules (HSMs) and are non-exportable by design, the keys themselves cannot leave. For server-side encrypted storage, exit means re-encrypting data as it is copied out; for client-side envelope encryption the bulk ciphertext can move as-is and only the wrapped data keys must be re-wrapped under the target key-management system. Key import (BYOK) and external key stores bound the exposure without removing the custody coupling.

4.3 Quadrant III: Deep Proprietary Lock-In (High $\mathcal{C}_{arch}$, High $\mathcal{F}_{exit}$)

Quadrant III services represent the highest operational and architectural risk. Applications built in this quadrant are functionally inseparable from the cloud provider.

- **Amazon DynamoDB:** Developers must construct data queries and index lookups strictly around DynamoDB's unique partition key, sort key, and global secondary index (GSI) architecture. The application code must explicitly invoke the DynamoDB SDK. Migrating away from DynamoDB requires a complete redesign of the database schema (often transitioning from DynamoDB's highly specialized single-table design to a relational or document-based model) and a massive rewrite of all data-access code.
- **AWS Step Functions:** The control flow, state transitions, parallel execution branches, and error-handling routines of the application are declared in JSON-based Amazon States Language (ASL). To escape Step Functions, engineers must rebuild the orchestration layer in an entirely different engine, such as Temporal or Apache Airflow, rewriting the core business workflow logic from scratch.

4.4 Quadrant IV: Standardized Cloud APIs (High $\mathcal{C}_{arch}$, Low $\mathcal{F}_{exit}$)

Quadrant IV services leverage the power of managed cloud APIs while maintaining high portability due to the adoption of open-source standards or de facto industry conventions.

- **Amazon S3:** Application code must use specific API calls to manipulate objects (e.g., `PutObject`, `GetObject`). However, the S3 API has achieved such profound industry saturation that it operates as a de facto open standard. Competitors (e.g., Cloudflare R2, Google Cloud Storage) and open-source self-hosted solutions (e.g., MinIO) natively support S3-compliant APIs. Thus, migrating away from Amazon S3 requires data migration, but rarely demands code modification beyond updating the endpoint URL in configuration files.
- **Amazon EKS (Kubernetes):** Code running on EKS targets standard container runtimes and interacts with the standardized Kubernetes API. While deploying EKS requires AWS-specific IaC, the application workloads themselves are highly portable to any CNCF-compliant Kubernetes cluster on-premises or on alternative clouds. The 0.12 prices the Kubernetes interface in isolation: a real EKS estate is additionally entangled through IAM roles for service accounts, load-balancer annotations, and storage and network drivers, an entanglement this framework prices in the IAM,

ELB, EBS, and VPC rows and in the coupling term of §5.2, keeping the EKS row itself about the Kubernetes interface.

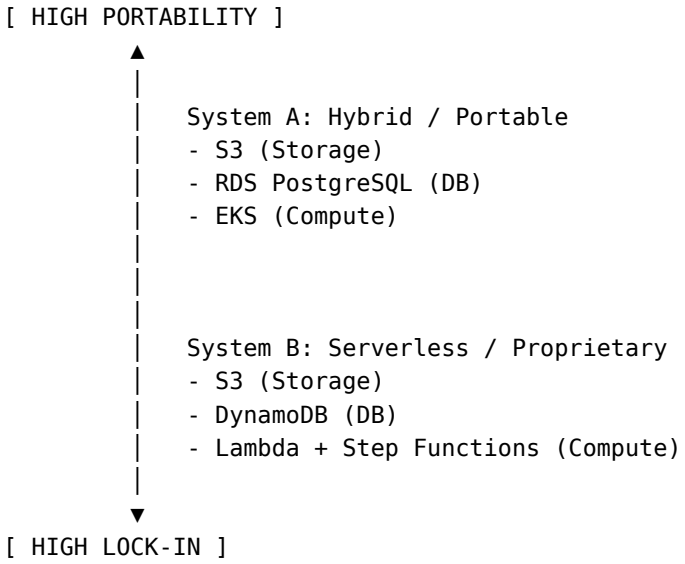
---

5. Architectural Implications & "Escape Velocity"

To conceptualize the trade-offs of these quadrants, we define the **Escape Velocity** ($\mathcal{V}_{esc}$) of a system as the inverse of its mean exit friction:

$$\mathcal{V}_{esc} = \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{F}_{exit}(D_i) \right)^{-1}$$

so that an estate of many portable services keeps a high velocity, while each Quadrant III adoption drags the mean up and the velocity down. (The unnormalized sum remains meaningful as the estate's *aggregate migration burden*, which grows with size even when every component is portable; the two measures answer different questions.)



When designing software systems, architects must navigate the **Velocity vs. Lock-In Paradox**:

- **The Paradigm of Maximum Acceleration (Quadrant III Dominant):**

By leveraging services like DynamoDB, Step Functions, and Lambda, engineering teams bypass boilerplate infrastructure code, accelerating time-to-market. The trade-off is a structurally rigid system with a low Escape Velocity. If the cloud provider changes their pricing, deprecates underlying runtimes, or suffers catastrophic outages, migrating the system requires significant capital expenditure.

- **The Paradigm of Maximum Portability (Quadrant I/IV Dominant):**

By constraining development to services like RDS, EKS, and S3-compatible endpoints, the engineering organization preserves an exceptionally high Escape Velocity. However, this introduces hidden operational costs, as the team must spend valuable engineering cycles managing container orchestration, manual scaling strategies, and complex patching cycles that cloud providers would otherwise automate.

5.1 Mitigation Strategies for Load-Bearing Services

To achieve the speed of Quadrant III and IV services without succumbing to high exit friction, standard software architecture recommends decoupling the application core using structural design patterns, such as **Ports and Adapters (Hexagonal Architecture)**.

By segregating all AWS SDK interactions into isolated "Adapter" modules behind clean interface boundaries ("Ports"), the core domain logic remains purely protocol-agnostic ($\mathcal{C}_{arch} \to 0$). If a migration becomes necessary, only the concrete adapter implementations must be rewritten, isolating the impact of cloud migrations on the core business logic.

5.2 Systemic Compounding: The Inter-Service Coupling Coefficient

The escape-velocity formula above treats exit friction additively, but real systems compound. As managed services trigger and bind to one another (Lambda invoked by EventBridge, orchestrated by Step Functions, persisting to DynamoDB, authorized by IAM), they form a tightly coupled state-machine network. The *systemic* exit tax $\mathcal{F}_{sys}$ of an N-service estate therefore scales super-linearly:

$$\mathcal{F}_{sys} = 1 - \prod_{i=1}^{N} (1 - \mathcal{F}_{exit,i})^{\,1+\alpha(N-1)}$$

where $\alpha \geq 0$ is the **Inter-Service Coupling Coefficient** and $\mathcal{F}_{sys}$ reads each component's exit tax as the chance that this component resists migration, so the aggregate is the chance that *something* in the estate resists. At $\alpha = 0$ the exponent collapses to one and the formula reduces to the independent noisy-OR: the estate can be migrated piecemeal, service by service, though the chance that at least one component resists still grows with estate size, as it should. As α grows, the exponent inflates every individual term to model integrations so entangled they must move together, and the whole estate becomes hostage to its most entangled component. Mandating open standards *at the service boundaries* drives $\alpha \to 0$: coupling inflation disappears and each component can be swapped without disturbing its neighbours. This single parameter, a qualitative device, uncalibrated (§8), is the hinge between the architectural analysis of §1–§4 and the sovereignty question of §6.

---

6. From Architecture to Sovereignty: Leverage Points for EU Cloud Independence

The preceding sections establish *where* lock-in lives (the exit-tax model) and *why* it compounds (the coupling coefficient). The stakes are measured at market scale: 80 percent of European business spending on cloud software and services flows to US vendors, some EUR 265 billion a year [Asterès 2025], and prices on that locked base rose 8.7 percent annually over the last three years, an estimated EUR 93 billion a year leaving the continent from the inflation alone [Asterès 2026]. We now turn to the policy question motivating this work: how can European governments and enterprises efficiently migrate off US hyperscalers (AWS, and equivalently Azure and GCP) toward EU-based, EU-controlled cloud services, where "EU-controlled" spans both European commercial providers and EU-governed open-source implementations? The goal is to move the system to a structurally different equilibrium, since trading one hyperscaler for another (AWS → Azure) changes the flag while leaving the exit tax intact.

We frame the answer using Donella Meadows' **twelve leverage points**, the places to intervene in a system, ranked from shallow (parameters) to deep (paradigms). Meadows' central finding bears directly on the problem: the interventions everyone reaches for first (subsidies, taxes, physical capacity) are the *least* effective, while the deep interventions (rules, self-organization, goals, paradigms) are exponentially more powerful, and usually cheaper. Cloud sovereignty policy is currently over-invested in the shallow end.

6.1 The Foundational Distinction: Open Standards ≠ Open Source

Before ranking interventions, one confusion must be dissolved, because it corrupts leverage points at every level. **Open Source is an implementation; an Open Standard is an interface specification.** They are not substitutes:

- **Open Standard (the interface):** a formal, vendor-neutral, royalty-free definition of how systems exchange data (standard SQL, POSIX, HTTP, DNSSEC, OCI container specs, S3-compatible APIs). Under the strict **European Interoperability Framework (EIF v1.0, 2004)**, an open standard must be formalized, independent of any single implementation, and freely implementable without legal or financial restriction.
- **Open Source (the implementation):** one of several possible code bases realizing a specification: PostgreSQL implementing SQL, Linux implementing POSIX, MinIO implementing the S3 API.

Mandating a *specific open-source project* rather than a *standard* re-creates lock-in in a new costume. Two traps recur:

- **The Governance Trap:** Kubernetes is open source, but it is governed by the CNCF, a US-registered 501(c)(6) entity subject to US export-control and sanctions jurisdiction. Standardizing Europe's execution layer on Kubernetes-the-project relocates dependency without removing it.
- **The Complexity Trap:** complex implementations cannot be swapped. Sovereignty rests on *protocol-level simplicity*: only a commoditized, simple standard lets any European vendor or community ship a hot-swappable compliant engine. Complexity is itself a barrier to entry that entrenches whoever already paid to master it.

This distinction maps to $\mathcal{P}_{spec}$ directly: sovereignty is achieved by driving *protocol proprietary-ness* to zero, which any implementation (commercial or open source, European or otherwise) can then serve.

6.2 Mapping Interventions to Leverage Points

The table below places the interventions available to EU policymakers and architects onto Meadows' ladder, with a verdict on each.

#	Meadows Leverage Point	EU Cloud Intervention	Verdict
12	Constants, parameters, subsidies	Cloud subsidies (IPCEI-CIS, Gaia-X funding); egress-fee caps	**Weak.** Funds a locked-in architecture more cheaply; see §7's lock-in floor.
11	Buffers (stabilizing stocks)	Sovereign capacity reserves, multi-cloud "insurance"	**Weak.** A buffer is useless if workloads cannot flow into it.
10	Stock-and-flow structure	Build EU datacenters / a "Euro-hyperscaler"; Chips Act	**Necessary, insufficient.** Geographic sovereignty ≠ architectural sovereignty; a "sovereign region" running proprietary APIs leaves $\mathcal{F}_{exit}$ untouched.
9	Length of delays	Fast-track ratification of de-facto APIs into standards	**Moderate.** Closes the gap between proprietary feature velocity and standards lag.

#	Meadows Leverage Point	EU Cloud Intervention	Verdict
8	Balancing feedback loops	Interoperability mandates with enforcement + portability audits	**Strong if enforced.** The Data Act's switching provisions are this loop; teeth determine strength.
7	Gain of reinforcing loops	Counter the hyperscaler adoption flywheel (student/ startup credits, certification, defaults)	**High-leverage, under-used.** Reducing a runaway positive loop beats strengthening a negative one.
6	Information flows	**Mandatory exit-tax disclosure** ("sovereignty label" using $\mathcal{C}_{arch}, \mathcal{F}_{exit}$)	**High-leverage, cheap.** Restores the missing feedback to the point of decision.
5	Rules of the system	**Specification-first procurement**: buy the standard, never the product	**Deep.** Public procurement is 14% of EU GDP; rules reshape the whole supplier field.
4	Power to self-organize	Royalty-free, *simple* standards → low barrier → competing EU implementations	**Deep.** Simplicity is the enabler of an evolving European implementation market.
3	Goal of the system	Reframe the objective to sovereign-by-default, exit-tax-bounded procurement, displacing cheapest-service-now	**Deepest actionable.** Changing the goal realigns every point below it.
2	Paradigm	**Sovereignty is architectural**: the interface is the sovereign asset, and geographic residency does not confer control	**Deepest.** Reframes data-residency compliance as interface control.
1	Transcend the paradigm	Refuse "we need our own hyperscaler"; aim for a commodity interface *nobody owns*	**The win condition.** Dissolve the hyperscaler category rather than nationalize it.

6.3 Why Current Policy Under-Performs: Over-Investment in Shallow Points

The bulk of visible EU cloud-sovereignty effort (capacity funding, "European hyperscaler" ambitions, datacenter build-outs) clusters at points 12–10, exactly the points Meadows identifies as least effective and most frequently "pushed in the wrong direction." Three failure modes recur:

1. **Subsidy without standards (12):** funding a European provider that re-implements the same proprietary-API paradigm produces a weaker copy of the lock-in it was meant to escape.
2. **Capacity without portability (11, 10):** a sovereign datacenter, or even a hyperscaler's own "European Sovereign Cloud" region, delivers data residency while leaving the architectural exit tax $\mathcal{F}_{exit}$ and coupling α entirely intact. The workload still cannot leave.

3. **The tipped-market trap (10):** attempting to out-feature AWS on its own proprietary terrain fights increasing returns uphill (see §7.5). The market has already tipped; a symmetric competitor cannot win it.

6.4 The Under-Used High-Leverage Interventions

The interventions with the best leverage-to-cost ratio sit in the deep half of the ladder and cost little capital, mostly regulatory will:

- **(6) Mandatory exit-tax disclosure.** The engineer who selects DynamoDB under time-to-market pressure is not the party who pays the migration cost years later. This missing information feedback is, in Meadows' account, the single most common cause of system malfunction. Requiring every public-sector cloud procurement to publish a standardized lock-in disclosure (the $\mathcal{C}_{arch}/\mathcal{F}_{exit}$ scores of this paper as a "sovereignty nutrition label") restores that feedback cheaply and immediately.
- **(5) Specification-first procurement.** Public tenders must specify *standards*, never *products*: "must communicate via standard SQL and deploy as OCI containers," never "must run on Aurora" or "must run on Kubernetes." This is the operational core of enforceable interoperability, and it converts EU procurement's 14%-of-GDP purchasing power into a standards-shaping force.
- **(4) Royalty-free simple standards → self-organization.** Fund the *interface* as public infrastructure (initiatives such as the industry-led **Sovereign European Cloud API (SECA)**) and leave *implementations* to a competitive European market of SMEs and open-source communities. Simplicity is deliberate: the simpler and more royalty-free the standard, the lower the barrier to a compliant "hot-swappable" engine, and the more the ecosystem can self-organize without anyone's permission.
- **(3, 2) Goal and paradigm.** For architects, this concretizes as **de-SaaSification of systemic anchors** (identity on LDAP/OAuth2 rather than proprietary IAM, routing on standard DNSSEC, core data stores on standard SQL) and **commodity-native execution** on POSIX/KVM VMs, OCI containers, or W3C-standardized WebAssembly. Every anchor built on a standard holds α near zero and keeps the cloud host a purely interchangeable utility.

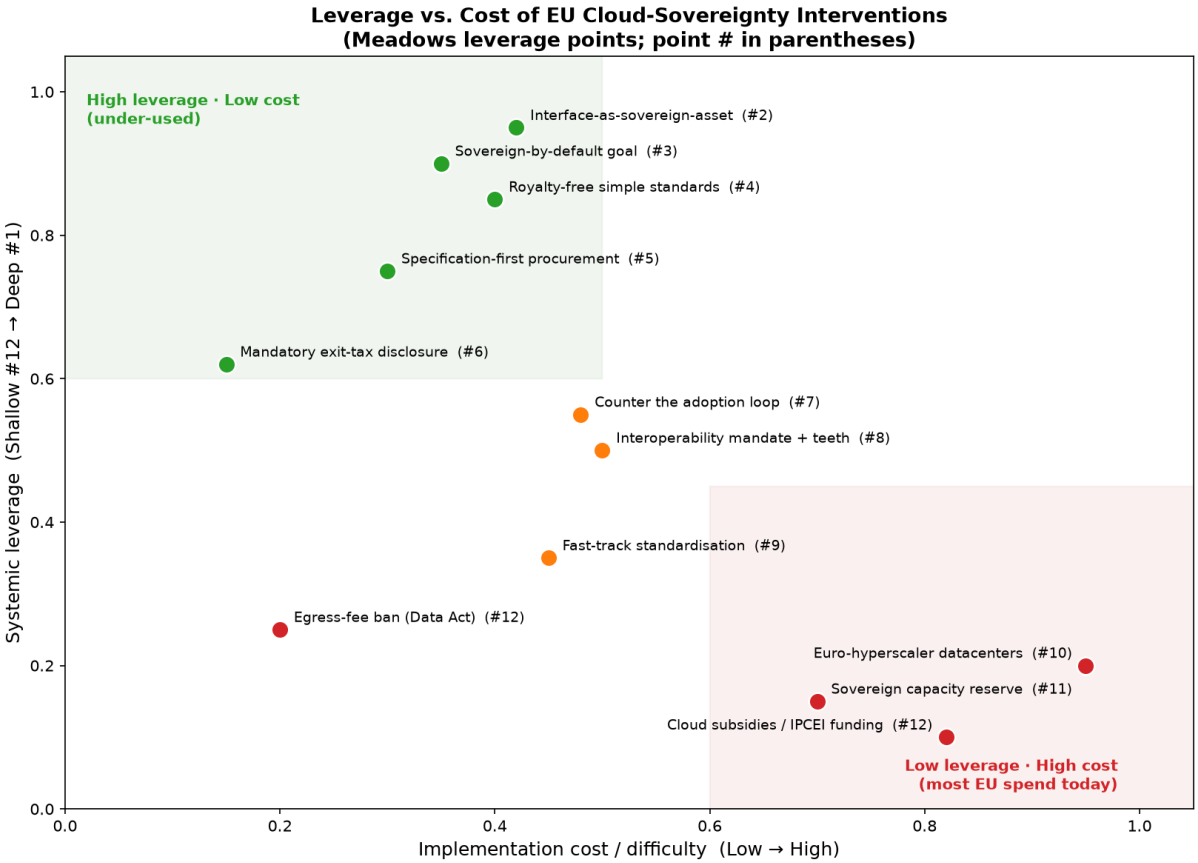


Figure 3. The interventions of §6.2 on a leverage-versus-cost plane (point numbers in parentheses give the Meadows rank). Both coordinates are deliberate editorial estimates for exposition (see the generating script's docstring), and only the ordinal contrast is claimed. The under-used, high-leverage quadrant (upper-left: disclosure, procurement rules, simple standards) is cheap; the corner absorbing most current EU spend (lower-right: subsidies, capacity, datacenters) is both expensive and shallow. Generated by `verify/plot_leverage.py`.

---

7. Economic Implications

The taxonomy also yields economic results: several follow directly from the exit-tax model and its systemic extension.

7.1 Exit Tax Is the Vendor's Pricing-Power Ceiling

In the economics of switching costs (Klemperer; Shapiro & Varian), a rational incumbent can extract rents from a locked-in customer up to (but not beyond) that customer's cost of switching. In our notation, $\mathcal{F}_{exit}$ *is* that ceiling: it is the maximum sustainable price premium a hyperscaler can charge over commodity cost before migration becomes rational. The present value of rents extractable from an installed base is therefore approximately $\sum_i \mathcal{F}_{exit,i}$, discounted; via the coupling coefficient, the systemic bound $\mathcal{F}_{sys}$ compounds it. Architectural lock-in is thus a direct, quantifiable transfer of future economic surplus from buyer to vendor. The 2026 Asterès/Cigref study documents the ceiling being consumed at market scale: 8.7 percent annual price growth on the locked base, justified by bundled AI features whose productivity gains a near-60 percent majority of surveyed organizations rate as theoretical [Asterès 2026].

7.2 The Lock-In Floor: The Bounded Reach of Data-Transport Policy

Because $\mathcal{F}_{exit} = w\,\mathcal{D}_{grav} + (1-w)\,\mathcal{L}$ with $w = 0.30$, a hard lower bound follows immediately:

$$\mathcal{F}_{exit} \geq (1 - w)\,\mathcal{L} = 0.70\,\mathcal{L}$$

Policy that acts only on data movement, most prominently the **EU Data Act's ban on egress fees (in force 2027)**, can influence only the $w\,\mathcal{D}_{grav}$ term, an absolute component capped at $w = 0.30$ on the unit scale; it cannot touch the logical-lock-in mass $\mathcal{L}$. As a *share* of a service's exit tax, that reach is $w\mathcal{D}/\mathcal{F}_{exit}$, which exceeds 30% exactly where data gravity outweighs the interface term ($\mathcal{D} > \mathcal{L}$: seven of the thirty services, all portable-interface data movers, topping out at 75% for RDS) and stays below 30% wherever the interface term dominates, which is every deeply locked service the policy problem is about. On the thirty scored services the mean transport share is 18% (§8). The floor $(1 - w)\mathcal{L}$ survives any $w < 1$ the shares move with the calibration and are swept in §8. For a proprietary service where $\mathcal{L} \to 1$ (DynamoDB, KMS, Step Functions), the exit tax is *structurally floored near 0.70* no matter how cheap or free data transfer becomes. The egress-fee ban is correct but, by the arithmetic of the model, addresses a minority of the problem; the binding constraint is $\mathcal{P}_{spec}$ and $\mathcal{M}_{fric}$, which only standards and procurement rules (points 5–4) can move. **You cannot buy your way out of logical lock-in by subsidizing data transport.**

7.3 Lock-In Is a Negative Externality → Market Failure

The party who incurs lock-in (the developer choosing a proprietary service under delivery pressure) is systematically *not* the party who later pays the exit tax (the organization; and, for critical public infrastructure, society, via correlated systemic risk and foreign-jurisdiction exposure under instruments such as the US CLOUD Act). This temporal and agency gap is a classic negative externality: because switching costs confer market power, the private equilibrium **over-supplies lock-in relative to the social optimum**. The textbook remedy is Pigouvian, though the *instrument choice* matters and connects back to the leverage ladder: pricing lock-in via a tax (point 12) is hard to calibrate and shallow, while internalizing it via disclosure (point 6) and capping it via procurement rules (point 5) is **cheaper and structurally deeper**.

7.4 Standards Are a Public Good → Fund the Interface, Compete on Implementations

An open, royalty-free standard is non-rival and non-excludable, a public good. Private hyperscalers have no incentive to supply it (they profit from proprietary interfaces), and individual European providers would each rather free-ride than fund standardization, so the standard is *under-provided* by the market. This is the economic justification for *public* funding of the interface layer (SECA, EIF bodies) while leaving implementations to competitive private markets. The payoff is rent dissipation: when $\mathcal{P}_{spec} \to 0$, the interface becomes contestable, Bertrand-style competition among interchangeable engines drives price toward marginal cost, and the lock-in rent of §7.1 is transferred from incumbent to buyers. Across the EU cloud market, that transfer is the measurable **"sovereignty dividend"**: (current hyperscaler EU revenue) × (rent fraction), recaptured as consumer surplus. This is the S3-versus-MinIO/R2 dynamic already visible in Quadrant IV, generalized to the whole estate.

7.5 Increasing Returns → Don't Fight a Tipped Market; Change the Terrain

Cloud platforms exhibit increasing returns to adoption (network effects, learning-by-using, complementary ecosystems). Under increasing returns (Arthur), markets tip early and lock in to a possibly-inferior standard. The implication for EU strategy is decisive: a European entrant *cannot* win a later, symmetric, feature-for-feature contest on the incumbents' proprietary terrain, which is fighting increasing returns uphill, and this is why "build our own hyperscaler" is economically doomed. The only viable move is to **change the terrain to a commodity standard**, whose network effects accrue to *all* implementers rather than a single owner. This is the economic backbone of the paradigm shift (points 2–1): out-scaling the hyperscalers does not win sovereignty, whereas commoditizing the interface beneath them does.

8. Robustness and limitations

The scores are calibrated judgments, so the suite in `verify/robustness.py` measures how much rides on them. Perturbing every input ($\mathcal{C}_{arch}$, $\mathcal{D}_{grav}$, $\mathcal{P}_{spec}$, $\mathcal{M}_{fric}$) by ± 0.15 over 2,000 seeded draws at $w = 0.30$: twenty-one of thirty services keep their exact quadrant, and twenty-four of thirty keep their friction half, in at least 90% of draws; the nine exceptions are SQS (33%), Aurora (52%), SNS (58%), Athena (62%), ELB (63%), API Gateway (67%), S3 (80%), ElastiCache (84%), and CloudWatch (85%). Sweeping w over $[0.10, 0.50]$: the deep corners never move, sixteen services flip their quadrant somewhere in the range, and none flips inside $[0.24, 0.31]$ around the baseline. The mechanism at the high extreme is transparent: at $w = 0.5$ every zero-gravity service has $\mathcal{F}_{exit} \leq 0.5$ and drops out of the high-friction half, which is how even KMS and Step Functions cross at the endpoint; the §7.2 floor arithmetic is exact at the calibrated $w = 0.30$ and weakens mechanically as w grows. The classification of the extremes, which carries the paper's architectural and policy conclusions, is robust; individual fragile assignments are marked with an asterisk in §3 and Annex A, the mark meaning quadrant stability below 90% under score noise; exactly the nine services above carry it. The suite also computes the §7.2 transport shares: mean 18%, per-service range 0–75%, above 30% for exactly the seven services with $\mathcal{D} > \mathcal{L}$. A cross-method check adds stability evidence of a different kind: the Spearman rank correlation between this model's exit taxes and the author's 2025 Dependency Scorecard (a linear model with an explicit alternatives term, scored a year earlier) is 0.60 over the eighteen common services, same author, so a consistency exhibit and never an independent validation; the largest divergences (CloudWatch, RDS, VPC) sit exactly where operational rebuild effort and interface portability come apart, the distinction the two-axis model was built to make.

The limitations that remain are structural. The sample is one vendor; the Azure and GCP analogues are queued as replication work, and the framework transfers unchanged. The inputs are single-coder judgments made auditable by Annex A; a two-coder protocol with agreement statistics is the next measurement step, together with anchoring $\mathcal{D}_{grav}$ and $\mathcal{M}_{fric}$ to observables (egress price times dataset benchmarks; documented migration case studies). The weight w is set by judgment and swept rather than estimated. The adoption anchors span 2018 to 2026, and the discourse-versus-spend check of §3 (Figure 2) bounds the recency risk: the head of the distribution persisted, and the named gaps are the plumbing services and the GenAI-era tail, both queued for the next revision. The inter-service coupling coefficient α is a modeling device, uncalibrated. Figure 3′s cost coordinates are editorial estimates, disclosed in its caption. And the leverage ladder ranks leverage; political feasibility is a separate axis this paper does not score.

9. Conclusion

Software engineering as a science requires moving beyond qualitative evaluations of cloud infrastructure. By categorizing cloud dependencies along the dual axes of Architectural Coupling ($\mathcal{C}_{arch}$) and Exit Friction ($\mathcal{F}_{exit}$), we establish a predictable model for system design.

This taxonomic framework demonstrates that vendor lock-in varies along a multi-dimensional spectrum. Understanding where each AWS service sits on that spectrum empowers engineering organizations to make calculated, rational architecture decisions that balance the immediate benefits of cloud-native development against the long-term agility and survival of their software systems.

Extended to the European sovereignty question, the same model yields a clear strategic ordering. Read through Meadows' leverage points (§6), the most efficient path off the US hyperscalers is the least visible one. Subsidies, a state-built "Euro-hyperscaler," and data-residency compliance are the shallow, expensive interventions. The high-leverage, low-cost moves are informational and

regulatory: disclose the exit tax at the point of procurement (point 6), buy standards rather than products (point 5), and fund simple royalty-free interfaces that a competitive European implementation market can self-organize around (point 4). All of these rest on the deepest paradigm shift (points 2–1): sovereignty is a property of the system's interfaces, and geographic residency alone does not confer it. The economics (§7) reinforce this ordering: the exit tax is the vendor's pricing-power ceiling; the model's lock-in floor shows that data-transport policy reaches on average 18 percent of the measured exit tax and none of the interface term that dominates the deeply locked services; lock-in is a genuine negative externality that justifies information-and-rules intervention over subsidy; and because cloud markets exhibit increasing returns, sovereignty is won by *commoditizing the interface* beneath the incumbents. The sovereign asset is the interface itself; the datacenter is incidental.

References

(Reference list subject to a final full-text verification pass.)

- Arthur, W. B. (1989). Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal* 99(394), 116-131.
- Asterès (2025). *Technological dependence on American software and cloud services: an assessment of the economic consequences in Europe.* Study commissioned by Cigref, May 2025.
- Asterès (2026). *From technological dependency to economic capture: the cost of cloud and software services inflation to Europe.* Study commissioned by Cigref, May 2026.
- European Commission (2004). *European Interoperability Framework for pan-European eGovernment services*, v1.0.
- European Union (2023). Regulation (EU) 2023/2854 on harmonised rules on fair access to and use of data (Data Act). *Official Journal*, 22 December 2023.
- Datadog (2025). *State of Containers and Serverless.* datadoghq.com.
- Farrell, J., & Klemperer, P. (2007). Coordination and lock-in: competition with switching costs and network effects. In *Handbook of Industrial Organization*, vol. 3, 1967-2072.
- Fermigier, S. (2025a). *AWS Alternatives (European or Open Source).* Unpublished draft, July 2025.
- Fermigier, S. (2025b). *The IaaS Dependency Scorecard: A Practical Model for Assessing Vendor Lock-in Risk.* Unpublished draft, 2025.
- Jefferson Frank (2018). *AWS Careers and Hiring Guide: product usage survey.* jeffersonfrank.com.
- Klemperer, P. (1995). Competition when consumers have switching costs. *Review of Economic Studies* 62(4), 515-539.
- Meadows, D. (1999). *Leverage Points: Places to Intervene in a System.* The Sustainability Institute.
- Meadows, D. (2008). *Thinking in Systems: A Primer.* Chelsea Green.
- Opara-Martins, J., Sahandi, R., & Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration. *Journal of Cloud Computing* 5, 4.
- Shapiro, C., & Varian, H. R. (1998). *Information Rules: A Strategic Guide to the Network Economy.* Harvard Business School Press.
- SWIPO AISBL (2020). *Codes of Conduct for Data Portability and Cloud Service Switching* (IaaS and SaaS).
- Vantage (2026). *AWS Cost Leaderboard* (spend aggregated across Vantage users, 30-day window, March 2026). leaderboard.vantage.sh.

Annex A. Computed Exit-Tax Table

The table below lists the full input decomposition for all thirty services and the resulting $\mathcal{F}_{exit}$ score. Each score is derived from the composite model of §2.3:

$$\mathcal{F}_{exit} = w \cdot \mathcal{D}_{grav} + (1 - w) \cdot \mathcal{L}, \qquad \mathcal{L} = 1 - (1 - \mathcal{P}_{spec})(1 - \mathcal{M}_{fric}), \qquad w = 0.30$$

Rows are ordered by descending exit tax. $\mathcal{L}$ is the intermediate logical-lock-in term. Quadrants marked * are boundary cases discussed in §3.

Service	$\mathcal{C}_{arch}$	$\mathcal{D}_{grav}$	$\mathcal{P}_{spec}$	$\mathcal{M}_{fric}$	$\mathcal{L}$	$\mathcal{F}_{exit}$	Quad.
Amazon DynamoDB	0.90	0.70	0.90	0.90	0.99	0.90	III
Amazon Cognito	0.70	0.30	0.80	0.85	0.97	0.77	III
Amazon SageMaker	0.75	0.45	0.75	0.60	0.90	0.77	III
Amazon EBS	0.10	0.85	0.30	0.55	0.69	0.73	II
AWS IAM	0.05	0.10	0.90	0.90	0.99	0.72	II
AWS KMS	0.10	0.00	0.95	0.95	1.00	0.70	II
AWS Step Functions	0.95	0.00	0.95	0.95	1.00	0.70	III
Amazon EventBridge	0.85	0.00	0.80	0.85	0.97	0.68	III
Amazon Kinesis	0.65	0.40	0.60	0.45	0.78	0.67	III
Amazon CloudWatch	0.40	0.35	0.70	0.35	0.80	0.67	II*
AWS Lambda	0.80	0.00	0.75	0.75	0.94	0.66	III
Amazon Redshift	0.15	0.85	0.25	0.40	0.55	0.64	II
AWS Secrets Manager	0.35	0.05	0.75	0.50	0.88	0.63	II
Amazon API Gateway	0.55	0.05	0.70	0.55	0.86	0.62	III*
AWS CloudFormation	0.05	0.00	0.70	0.55	0.86	0.61	II
AWS Auto Scaling	0.30	0.00	0.75	0.50	0.88	0.61	II
Amazon SNS	0.55	0.10	0.55	0.45	0.75	0.56	III*
AWS ELB	0.25	0.05	0.55	0.35	0.71	0.51	II*
Amazon Aurora	0.15	0.70	0.15	0.30	0.41	0.49	I*
Amazon SQS	0.50	0.15	0.40	0.35	0.61	0.47	IV*
Amazon Athena	0.25	0.30	0.25	0.40	0.55	0.47	I*
Amazon S3	0.70	0.90	0.10	0.15	0.23	0.43	IV*
Amazon ECS	0.30	0.00	0.30	0.40	0.58	0.41	I
Amazon Route 53	0.05	0.05	0.10	0.50	0.55	0.40	I
Amazon CloudFront	0.10	0.05	0.25	0.30	0.48	0.35	I
Amazon RDS	0.10	0.70	0.00	0.10	0.10	0.28	I
Amazon VPC	0.00	0.00	0.05	0.25	0.29	0.20	I
Amazon EC2	0.05	0.30	0.05	0.05	0.10	0.16	I
Amazon EKS	0.65	0.05	0.00	0.15	0.15	0.12	IV
Amazon ElastiCache	0.60	0.20	0.00	0.05	0.05	0.10	IV*

Worked example (Amazon S3). $\mathcal{L} = 1 - (1 - 0.10)(1 - 0.15) = 0.235$ $\mathcal{F}_{exit} = 0.30 \times 0.90 + 0.70 \times 0.235 = 0.43$. Despite high data gravity, the open S3 API keeps $\mathcal{L}$ low, holding total exit tax below the midpoint.

Annex B. The alternatives landscape (the $\mathcal{M}_{fric}$ anchor)

The migration-friction scores are anchored to the observable landscape of alternatives, consolidated from the author's 2025 survey [Fermigier 2025a] and extended to the newer sample entries: a mature, widely deployed open-source alternative (better still, a compatible API) pulls $\mathcal{M}_{fric}$ down; the absence of any direct replacement pins it high. The table lists the principal open-source and European alternatives per service; it is also, read column-wise, the option set that the companion research programme's sovereignty measures count.

Service	Open-source alternatives	European alternatives
EC2	OpenStack, Proxmox VE	OVHcloud, Scaleway, Hetzner, IONOS, Exoscale
S3	MinIO, Ceph, Garage	Scaleway, OVHcloud, Exoscale, IONOS
RDS	PostgreSQL (Patroni), MySQL/ MariaDB	OVHcloud, Scaleway, Aiven, IONOS
Aurora	PostgreSQL/MySQL + HA tooling	Aiven, OVHcloud, Clever Cloud
Redshift	ClickHouse, PostgreSQL+Citus, Trino	Exasol, Aiven, OVHcloud Data Platform
DynamoDB	Cassandra; ScyllaDB and MongoDB (source-available, not OSI-open)	OVHcloud (MongoDB), Aiven (Cassandra)
ElastiCache	Redis/Valkey, Memcached	Aiven, OVHcloud, Scaleway
Athena	Trino, Spark SQL	self-hosted Trino on EU IaaS
Kinesis	Apache Kafka, Flink	Aiven, OVHcloud, Clever Cloud (Kafka)
SQS	RabbitMQ, ActiveMQ	Scaleway (SQS-compatible API), CloudAMQP
SNS	NATS, RabbitMQ, MQTT brokers	Scaleway (NATS), CloudAMQP
EventBridge	Kafka, NATS, RabbitMQ	Aiven, OVHcloud, Clever Cloud
Lambda	Knative, OpenFaaS, Nuclio	Scaleway Serverless, Clever Cloud
Step Functions	Temporal, Airflow, Argo Workflows	self-hosted on EU IaaS
EKS	any CNCF-conformant Kubernetes	OVHcloud, Scaleway Kapsule, IONOS, Exoscale
ECS	Kubernetes, Nomad	EU managed Kubernetes (as above)
EBS	Ceph RBD, LINSTOR, Longhorn	OVHcloud, Scaleway, Hetzner volumes
VPC	OpenStack Neutron, Open vSwitch	OVHcloud vRack, Scaleway, Hetzner
ELB	HAProxy, Nginx, Traefik, Envoy	OVHcloud, Scaleway, Hetzner
CloudFront	Varnish/HAProxy stacks	Bunny.net, OVHcloud CDN
Route 53	PowerDNS, Knot DNS, BIND	OVHcloud, Gandi, IONOS, Infomaniak
IAM	none direct; Keycloak, Zitadel, Authentik federate	provider IAMs + a sovereign IdP
Cognito	Keycloak, Zitadel, Authentik, Ory	Zitadel, Ory (hosted)
KMS	OpenBao (Vault fork), softHSM	EU HSM vendors (Utimaco, Thales), provider KMS

Service	Open-source alternatives	European alternatives
Secrets Manager	OpenBao, Infisical	self-hosted on EU IaaS
API Gateway	Kong, APISIX, Tyk	self-hosted on EU IaaS
CloudWatch	Prometheus, Grafana, OpenTelemetry, ELK	Scaleway Cockpit, OVHcloud Logs
CloudFormation	OpenTofu (open-source fork of BUSL-licensed Terraform), Pulumi, Ansible	EU providers ship Terraform/ OpenTofu providers
Auto Scaling	Kubernetes HPA / Cluster Autoscaler	feature of EU managed Kubernetes
SageMaker	Kubeflow, MLflow, Ray	OVHcloud AI, Scaleway

Two regularities in the table carry §6′s argument: where the alternative column is anchored by an open standard or a de facto open API (Kubernetes, SQL, Redis, S3), the European column is crowded; where the decisive layer is a proprietary control plane (IAM, KMS, CloudFormation), the open-source column starts with "none direct" and the European column offers federation at best. The alternatives landscape is the option set; the interface decides whether it is populated.

Annex C. Reproducibility

The source data lives in `notes/services.csv`. The script `verify/plot_quadrants.py` recomputes every $\mathcal{F}_{exit}$ from the formula above, asserts it matches the tabulated value (tolerance 0.01), and renders **Figure 1**. It is self-contained via `uv`:

```
uv run verify/plot_quadrants.py   # regenerates papers/figures/quadrants.png
uv run verify/robustness.py       # score-noise + weight-sweep + concordance gates
(results/robustness.json)
uv run verify/hype_gap.py         # discourse-vs-spend regimes + Figure 2 (data/
hype_corpus.json -> results/hype_gap.json)
uv run verify/plot_leverage.py    # regenerates papers/figures/leverage.png (Figure
3; editorial coordinates)
```