THE VELOCITY-GOVERNANCE GAP

Why Companies Ship Faster Than They Can Govern — and What It Is Costing Them

The Hidden Cost of Cloud, AI, and Security Debt in Modern Software Delivery

A Business Consulting Report on Cloud Cost Waste,
AI Spend Governance, and Security-by-Delivery

Prepared August 2026
In partnership with Bithost — Security, Cloud & AI Engineering
www.bithost.in

Table of Contents

Executive Summary

Companies are moving faster than ever. Very few are managing the cost, security, and infrastructure debt that speed creates.

Every organization in this report's audience recognizes the same story, even if the details differ. A project starts. Developers build. DevOps deploys. A demo happens, a stakeholder is impressed, and the team moves on to the next priority. What nobody schedules is a return trip: the staging server that never gets switched off, the database seeded with test data that never gets dropped, the load balancer still routing to nothing, the API key that was supposed to be temporary. Months later, someone in finance asks a question nobody can answer with confidence: “Why is the cloud bill so high?” In 2026 there is a second version of the same question, arriving faster and for larger sums: “Why is the AI bill so high?”

This report argues that both questions have the same root cause. It is not that cloud is expensive, and it is not that AI is expensive. It is that technology delivery now happens faster than technology governance. Provisioning a server, spinning up a database, or generating an API key takes seconds. Tracking who owns it, what it costs, when it should be retired, and whether it is secure takes discipline that most delivery teams were never resourced or incentivized to apply. The gap between how fast organizations can create infrastructure and AI access, and how slowly they notice, own, and retire it, is where the money — and the risk — quietly accumulates.

What the evidence shows

29%
of IaaS/PaaS cloud spend is wasted globally in 2026 — the first increase in five years (Flexera 2026 State of the Cloud Report)

$4.44M
average global cost of a data breach in 2025; $10.22M in the United States (IBM Cost of a Data Breach Report 2025)

28.6M
hardcoded secrets — API keys, tokens, credentials — exposed on public GitHub in 2025, up 34% year over year (GitGuardian, 2026)

92%
average CPU headroom paid for but unused across production Kubernetes clusters, which run at roughly 8% CPU utilization (Cast AI, 2026)

81%
year-over-year growth in leaked AI-service credentials as agentic tooling and API keys proliferate (GitGuardian, 2026)

$670K
additional average breach cost attributable to ungoverned “shadow AI” usage (IBM, 2025)

These are not edge cases. They are the median condition of technology organizations moving at the pace the market now demands. Flexera's 2026 survey of 753 cloud decision-makers found that estimated wasted cloud spend rose to 29% of IaaS and PaaS budgets, reversing five consecutive years of improvement — and attributed the reversal directly to the surge of AI workloads outrunning cost forecasting and governance. GitGuardian's fifth annual secrets-sprawl study recorded the largest single-year jump in exposed credentials it has ever measured, with AI-related leaks growing more than twice as fast as the broader trend. IBM's

twentieth annual breach-cost study shows that where AI and cloud data sprawl across multiple, loosely governed environments, both the cost and the duration of a breach climb sharply.

The pattern behind the numbers

Across cloud infrastructure, AI usage, and security, the report identifies one recurring pattern: resources are trivially easy to create and structurally hard to retire, track, or own. A developer can spin up a staging environment, request a database, or generate an API key in under a minute, with no approval workflow, no expiration date, and no named owner attached. The organization pays for the convenience twice — once in wasted spend, and once in the risk that an unmonitored, unpatched, unowned resource becomes the easiest way in for an attacker or the least defensible line on next quarter's budget review.

What this report covers

- The delivery chain — from business requirement to production — and precisely where governance breaks down at each handoff.
- Cloud cost waste: idle compute, orphaned storage, oversized instances, Kubernetes overprovisioning, networking and observability costs, and the organizational blind spots that let them persist.
- AI cost waste: unrestricted API keys, uncontrolled model selection, agentic and “vibe coding” workflows, and real, documented incidents ranging from four-figure surprises to eight-figure ones.
- Security during delivery: how secrets, credentials, and access controls are compromised specifically because security is treated as a final audit rather than a continuous discipline.
- A practical framework — covering ownership, visibility, guardrails, and culture — for closing the governance gap without slowing delivery down, including a 90-day roadmap organizations can start on immediately.
- In the final section only, how Bithost's cloud, security, and AI engineering services map to each part of this framework for organizations that want hands-on help closing the gap.

The core recommendation

> **Governance is not the opposite of speed.**
>
> The organizations that control cost and risk best are not the ones that slow delivery down with approval committees. They are the ones that build ownership, expiration, and visibility into the delivery pipeline itself — so that speed and accountability travel together instead of trading off against each other. The remainder of this report explains, with evidence, exactly where that discipline is missing today and how to put it back.

How to read this report

Readers pressed for time can get the core argument from the Executive Summary, Section 6 (root cause), and Section 8 (the framework) alone. Readers building a business case for a governance program will get the most value from Section 7's financial modeling and Section B's answers to likely objections. Readers responsible for a specific domain — cloud, AI, or security — can go directly to Sections 3, 4, or 5 respectively, each of which

stands on its own with its own evidence and sourcing. Section 10, on Bithost, is intentionally isolated at the end for readers who want to evaluate the diagnosis and framework entirely on their own merits before considering who might help implement them.

1. Introduction: The Velocity-Governance Gap

1.1 The central thesis

Every technology organization is under the same pressure: ship faster. Faster releases, faster experiments, faster proof-of-concepts, faster adoption of the newest AI model. Speed is not optional — it is close to the only competitive advantage that consistently matters in software markets. Cloud platforms and AI APIs were built to serve that pressure. A developer can provision a virtual machine, a managed database, a Kubernetes cluster, or a frontier-model API key in the time it takes to read this paragraph. That is the entire value proposition of modern infrastructure: near-zero friction between an idea and a running system.

The problem this report investigates is what happens after that moment of near-zero friction. Creating a resource takes seconds. Tracking who owns it, why it exists, what it costs, when it should be retired, and whether it is configured securely takes discipline — discipline that has to be designed into an organization's delivery process on purpose. It does not happen by default, and in most organizations it currently does not happen at all. The result is a structural mismatch: the rate at which technology gets created has outpaced the rate at which it gets governed. This report calls that mismatch the velocity-governance gap, and it is the lens through which every subsequent section — cloud waste, AI waste, and security exposure — should be read.

> **The central claim of this report**
>
> The problem is not that cloud is expensive. The problem is not that AI is expensive. The problem is that technology delivery happens faster than technology governance — and every dollar of hidden waste and every unnecessary security exposure in the chapters that follow traces back to that one structural gap.

1.2 A familiar story

Picture a mid-size software team building a new customer-facing feature. A business stakeholder needs to see it work before greenlighting the budget for a full rollout. Developers build a working version. DevOps stands up a staging environment to host it: a few virtual machines, a managed database seeded with realistic test data, a load balancer, a couple of storage volumes, logging and monitoring wired in because that is good practice. The demo happens. The stakeholder is impressed. The project either moves into a longer build phase or, just as often, gets quietly deprioritized while everyone moves on to the next sprint, the next incident, the next quarter's roadmap.

Nobody sends an email to shut the staging environment down, because shutting it down was never anyone's assigned job. The virtual machines keep running at 2 a.m. on a Sunday exactly as they did at 2 p.m. on the day of the demo. The database keeps accepting automated backups. The snapshots taken “just in case” before a risky change keep accumulating, each one billed independently of whether anyone will ever restore from it. The load balancer keeps a health check running against an application nobody is using. The unattached storage volume that was cloned for a one-off data migration test sits in the account, invisible in the console unless someone specifically goes looking for orphaned resources. None of this shows up as a single alarming line item. It shows up as a slow, compounding creep in the monthly invoice.

Three, six, twelve months later, someone in finance or in an executive review notices that the cloud bill has grown faster than the business has. They ask engineering leadership a simple question that turns out to be remarkably hard to answer: “Why is the cloud bill so high?” Answering it requires reconstructing a history that nobody recorded — which environment belongs to which project, which project is still active, and who has the authority to turn any of it off. This is not a hypothetical. It is, by a wide margin, the most common way that organizations first discover they have a cloud cost problem, and it is documented at scale: Flexera's 2026 State of the Cloud Report finds that managing cloud spend has now been the top-ranked challenge for cloud decision-makers for four years running, ahead of security and software licensing.

1.3 The same pattern, arriving faster, in a new domain

Everything described above is now repeating itself with AI, except compressed into weeks instead of months and denominated in far less forgiving units. A developer wants to try a new model, or wire an AI feature into an internal tool, or give an agent access to a coding assistant. They generate an API key. In the overwhelming majority of organizations, that key ships with no spending cap, no model restriction, no expiration date, and no owner recorded anywhere central. Usage starts small — a few test prompts, a demo, a proof of concept. Then an automated job starts calling it on a schedule. Then someone builds an agent that calls it in a loop. Then the key finds its way into a public repository, a shared Slack channel, or a client-side configuration file, and someone outside the organization starts using it too.

The organization does not find out from a dashboard. It finds out from an invoice, and by the time that invoice arrives the number can be shocking: this report documents real, publicly reported incidents ranging from a few hundred dollars from an unattended coding-agent session, to $18,000 overnight after a single Google Cloud key leaked, to $80,000 in a single week from one employee's unsupervised experimentation, up to a reported $500 million in a single month at one large enterprise where nobody had configured a spending limit at all. Each of these is examined in detail in Section 4. What they share is the mechanism, not the dollar amount: an access credential was created faster than the guardrail meant to bound its use.

1.4 Why this report, and how to use it

This report is written for the people who will be asked “why is the bill so high” and need a credible, evidence-based answer — CFOs and finance business partners who own the budget; CTOs, VPs of Engineering, and Heads of Platform who own the infrastructure; and CISOs and security leaders who own the risk. It is deliberately not a generic cloud cost or AI security whitepaper. It is built around one specific, verifiable claim: that cost, security, and infrastructure debt in modern software organizations are three symptoms of the same underlying condition, and that condition can be diagnosed and treated with the same set of governance practices.

Sections 2 through 5 build the evidence base: the shape of the modern delivery chain, and a detailed, sourced account of cloud cost waste, AI cost waste, and security risk. Sections 6 and 7 connect those findings back to root cause and put a number on what the gap is actually costing a representative organization. Section A places that diagnosis in an industry-specific context. Sections 8 and 9 turn the diagnosis into a practical

framework and a 90-day starting roadmap that any technology organization can begin executing without waiting for a large program to be approved. Section B addresses the objections any such program will realistically face, and Section C surveys the tooling landscape at a category level. Section 10, and only Section 10, describes how Bithost's services map onto that framework for organizations that want an experienced partner to help close the gap rather than closing it entirely with internal resources.

1.5 A note on what this report is not

This report is not an argument that cloud spend or AI spend should be minimized. Cloud infrastructure and AI capability are, for the large majority of the organizations this report is written for, a source of genuine competitive advantage, and an organization that under-invests in either out of excessive caution will lose to competitors who do not. The argument throughout this report is narrower and, the evidence suggests, considerably less controversial once stated plainly: spend and access should be a deliberate choice, made with visibility into its cost and risk, rather than an accidental byproduct of the absence of a decision. Nothing in the framework that follows asks an organization to spend less on cloud or AI. It asks an organization to know, with confidence, what it is spending on and why — which is a substantially lower bar, and one every organization in this report's audience is capable of clearing.

2. The Modern Delivery Chain

To see where governance breaks down, it helps to lay the delivery chain out explicitly, from the moment a business need is identified to the moment software is running in front of real customers. In most organizations, this chain looks something like the sequence below, and — critically — responsibility for cost, ownership, and security shifts at every arrow without anyone formally handing it off.

2.1 Business requirement to production

Stage	What happens	Who typically owns cost/risk here
Business requirement	A stakeholder identifies a need: a feature, an experiment, an integration, an internal tool.	Business owner — rarely tracks technical cost
Developer	Code is written, often against a personal or shared cloud account and a personal or shared AI API key for speed.	Individual developer — no budget authority
Application	A working application or prototype exists, frequently with test data, debug flags, and permissive access left in for convenience.	Nobody formally — the app is “not yet in production”
DevOps / Platform	Infrastructure is provisioned to run and demo the application: compute, storage, networking, sometimes a database clone.	DevOps team — provisions but does not always own the bill
Cloud	The provisioned infrastructure begins accruing cost on a running-hourly or per-request basis, independent of whether it is used.	Finance sees the aggregate bill, not the resource
AI services	API keys are generated for models, embeddings, and agent tooling, usually without spend caps or usage attribution.	Whoever created the key — often untracked after that
Staging	A staging or demo environment is stood up to validate the work before a go/no-go decision.	Ambiguous — treated as temporary, run as permanent
Production	If approved, the workload graduates to production, but staging is rarely decommissioned in the same step.	Production owner — staging is left behind

2.2 Where governance breaks down at each handoff

The table above looks orderly. In practice, every arrow in that chain is a place where accountability can be silently dropped, for entirely understandable reasons rooted in how teams are incentivized and measured.

- Business requirement → Developer: the requirement rarely specifies a budget, a data-sensitivity classification, or an expected lifetime for the resulting infrastructure. “Build a demo” does not usually come with “and turn it off by such-and-such date.”
- Developer → Application: developers are measured on shipping working software, not on the ongoing cost or security posture of the infrastructure that supports it. A developer under deadline pressure will reach for the fastest path — a hardcoded credential, a wide-open security group, the largest available instance size “just to be safe” — because none of those shortcuts are visible to the metrics the developer is judged on.

- Application → DevOps: DevOps teams are typically optimizing for reliability and speed of deployment, not for cost efficiency of every environment they stand up. Provisioning generous, over-specified infrastructure is the path of least resistance because under-provisioning risks an outage that DevOps will be blamed for, while over-provisioning risks a cost overrun that finance will be blamed for — an asymmetry that predictably biases every decision toward excess.
- DevOps → Cloud: once a resource exists in a cloud account, the cloud provider has no incentive to ask whether it is still needed — the provider is paid whether or not the resource is used. Idle, forgotten, and duplicate resources are invisible unless an organization builds its own process to go looking for them.
- Cloud → AI services: AI providers, in the current market, are optimized for frictionless adoption — minimizing signup friction and, in many cases as documented in Section 4, minimizing the friction of setting spending controls too, since default configurations frequently ship with alerts rather than hard caps.
- Application → Staging: staging environments are treated organizationally as temporary but architecturally as permanent — they use the same durable, always-on infrastructure patterns as production, with none of the same decommissioning discipline.
- Staging → Production: the promotion of an approved feature to production is a well-defined, usually automated step. The corresponding retirement of the staging environment that validated it is, in most organizations, not defined at all.

2.3 Speed versus control: a false choice

The natural response to this analysis is to assume the fix is more approval gates — a change advisory board for every environment, a finance sign-off for every API key, a security review before every deploy. That response is understandable and almost always counterproductive. Approval gates slow delivery without necessarily improving governance, because they are typically applied uniformly regardless of risk, and because developers under deadline pressure will find the path around a gate that feels like pure friction rather than genuine protection.

The organizations that manage this well, examined in more detail in Section 8, do not add friction at the point of creation. They add default expiration, default ownership tagging, default budget ceilings, and default security baselines that travel with a resource automatically from the moment it is created — so that speed and accountability are properties of the same action rather than a trade-off between two competing ones. A staging environment that is born with a 14-day expiration date and an owner tag does not require anyone to remember to clean it up. An AI API key that is born with a monthly spend ceiling and a named service owner cannot silently become a $500,000 surprise. The remainder of this report documents, in detail, exactly how much is currently being lost because that default does not yet exist in most organizations — and what it takes to build it.

2.4 Instrumenting the chain: what “good” looks like

It is easier to recognize the delivery chain's governance gaps once there is a concrete picture of what their absence looks like in practice, in contrast to what a well-instrumented chain looks like. The table below contrasts the two directly, stage by stage, and functions as a preview of the specific practices developed fully in Section 8.

Stage	Ungoverned default (most organizations today)	Governed default (Section 8's target state)
Developer creates a resource	No required owner, tag, or budget; whatever is fastest to type	Owner, cost center, and expected lifetime required before creation succeeds
DevOps provisions an environment	Copied from a previous environment, sized generously “to be safe”	Provisioned from a governed template with appropriate default sizing
An AI API key is generated	No spend cap, no expiration, often shared across a whole team	Per-identity key with a hard spend ceiling and default expiration
Staging is stood up for a demo	Same durable, always-on pattern as production, no end date	Time-boxed by default, with a low-friction renewal path if still needed
A feature graduates to production	Automated and well-defined	Automated and well-defined — and staging teardown is included as part of the same workflow
Cost or risk needs to be reviewed	Manual, ad hoc, usually triggered by a surprising invoice or incident	Continuous and automatic, visible to the owning team without a special request

Every row on the right side of that table is achievable with tooling that exists today, described at a category level in Section C, and every row is a direct answer to a finding documented in Sections 3 through 5. The gap between the two columns is, in a very literal sense, the subject of this entire report.

2.5 Two velocities, measured

It is useful to think of the delivery chain as having two distinct velocities running through it simultaneously, and the gap between them is what this report has been calling the velocity-governance gap since Section 1. The first is delivery velocity: how quickly an idea becomes a running system, typically measured in minutes for provisioning a resource or generating an API key. The second is governance velocity: how quickly that same system acquires an owner, a cost attribution, and a security review, typically measured, in organizations without the practices in Section 8, in months — if it happens at all. Every category of waste and risk documented in Sections 3 through 5 exists in the space between those two velocities, for exactly as long as the gap between them persists. Closing the gap does not require slowing delivery velocity down to match governance velocity, which is the mistake Section 2.3 warned against. It requires speeding governance velocity up to match delivery velocity — automating ownership, cost visibility, and security baselines so they are established in minutes as well, at the same moment the resource itself is created.

3. Cloud Cost Waste

Roughly three of every ten cloud dollars spent in 2026 deliver no business value — and the share is rising for the first time in five years.

3.1 The scale of the problem

Flexera's 2026 State of the Cloud Report, based on a survey of 753 cloud decision-makers, found that estimated wasted spend on infrastructure- and platform-as-a-service climbed to 29% of total cloud spend, up from 27% the prior year — the first increase after five consecutive years of decline. The report attributes the reversal directly to the cost complexity introduced by AI workloads and newer cloud services, which are harder to forecast and rightsize than the workloads organizations had spent five years learning to optimize. Independent analysis of the same underlying data estimates the wasted share at well over $100 billion globally in 2026 alone. Separately, 76% of large enterprises now spend more than $5 million a month on public cloud, and 17% of organizations exceeded their public cloud budget over the past year — meaning that even a few points of unmanaged waste translates into a material line item at the scale most mid-size and large organizations now operate.

29%	76%	8%
of IaaS/PaaS spend wasted in 2026, reversing 5 years of decline (Flexera)	of large enterprises spend over $5M/month on public cloud (Flexera)	average CPU utilization across production Kubernetes clusters (Cast AI, 2026)

Sources: Flexera 2026 State of the Cloud Report; Cast AI 2026 State of Kubernetes Optimization Report.

3.2 Compute: idle, oversized, and always-on by default

The single largest category of cloud waste is compute that is either idle or provisioned far larger than the workload requires. Industry benchmarking puts the median cloud instance at roughly 7–12% CPU utilization, and idle compute together with overprovisioned instances account for an estimated 60% of all cloud waste — the largest, and often the easiest, category to address, because much of it carries no meaningful restoration risk once identified.

Development and staging environments running 24/7

The most common and most preventable source of idle compute is non-production infrastructure that runs on a permanent, always-on schedule when it is only ever used during a working week. Industry data shows that roughly 44% of total cloud spend covers non-production resources, and that these resources are frequently needed only during a 40-hour work week — meaning they sit idle for the remaining 128 hours, or 76% of every week, while still billing at the full rate. A staging environment stood up for a two-week sprint and left running for a year is not an unusual event; it is close to the median outcome absent an explicit decommissioning step.

Oversized instances

Separately from idleness, resources that are actively used are routinely provisioned larger than necessary. Teams under deadline pressure default to “size up to be safe” because the personal and team-level cost of an outage from under-provisioning is far more visible and immediate than the diffuse, delayed cost of overprovisioning showing up on next month's invoice. This asymmetry, discussed further in Section 3.9, is one of the most consistent behavioral drivers of cloud waste across every category in this chapter.

3.3 Storage: orphaned disks, snapshots, backups, and unused IPs

Storage waste tends to be less visible than compute waste because it rarely appears as a single alarming line item — it accumulates in small increments that are individually cheap and collectively significant.

- Orphaned disks and volumes: when a virtual machine is terminated, its attached storage volume is frequently not deleted along with it, either because the deletion is a separate manual step or because the volume is deliberately preserved “just in case” and then never revisited. These detached volumes continue to bill at full storage rates indefinitely.
- Unused IP addresses: static/elastic IP addresses reserved for a resource that has since been terminated or resized continue to incur an hourly charge on most major cloud providers precisely because they remain reserved and unused — providers charge for the scarcity they are holding, not for traffic.
- Snapshots: pre-change snapshots taken “just in case” before a risky deployment, database migration, or infrastructure change are cheap individually and rarely have an automatic expiration policy attached. Over months and years, snapshot storage frequently grows to exceed the size of the live data it was meant to protect, with no team actively deciding to keep any specific one.
- Backups without retention policy: automated backup jobs are, correctly, treated as a safety-critical default that nobody wants to be responsible for disabling. Without an explicit retention and expiration policy, backup storage grows monotonically for the life of the account, and the absence of a policy is itself rarely flagged as a decision anyone made — it is simply what happens when nobody decides otherwise.
- Unused or underutilized databases: managed database instances provisioned for a project that was later shelved, or sized for a peak load that never materialized, are one of the more expensive idle categories because database instance pricing scales with both compute and storage commitments simultaneously.

3.4 Kubernetes and container waste

Kubernetes has become the default compute layer for a large share of production workloads — 80% of organizations now run it in production, according to the CNCF's annual survey — and it has become one of the largest single sources of avoidable cloud spend precisely because of how its resource model works. Kubernetes schedules workloads based on the CPU and memory a team requests for a pod, not on what that pod actually consumes. Teams routinely set requests well above real usage to avoid the risk of throttling or out-of-memory crashes, and once those values are set in a deployment manifest, they are rarely revisited.

Metric	2026 benchmark	Source
Average CPU utilization, production clusters	~8% (down from 10% the prior year)	Cast AI, 2026 State of Kubernetes Optimization Report
Average memory utilization	~20%	Cast AI, 2026
CPU overprovisioning	69% (up from 40% year over year)	Cast AI, 2026
Memory overprovisioning	79%	Cast AI, 2026
Average GPU utilization	~5% (or ~23% by a separate estimate)	Cast AI, 2026 / Harness, 2025
Container costs attributable to idle resources	83%, split 54% overprovisioned infrastructure / 29% oversized requests	Datadog State of Cloud Costs
Organizations running full cost chargeback to teams	~14%	CNCF 2024 Annual Survey

The chargeback figure at the bottom of that table is arguably the most important line in it. When only a small minority of organizations attribute Kubernetes cost back to the team whose workloads generate it, the cost becomes a shared abstraction: real on the finance report, invisible to the engineers who control the resource requests that actually drive it. Without that link, overprovisioning has no natural correction mechanism — nobody feels the cost of the padding they added, so nobody has a reason to remove it.

3.5 Networking: load balancers, NAT gateways, and data transfer

Networking infrastructure is a particularly persistent form of waste because, unlike a virtual machine, it rarely appears as an obvious idle resource in a console view — a load balancer with zero healthy targets still looks like a normally configured load balancer.

- Load balancers left routing to decommissioned or scaled-to-zero backends continue to bill hourly regardless of whether they are serving any traffic.
- NAT gateways, billed both hourly and per gigabyte processed, are frequently provisioned once per environment and never reviewed again, even after the workloads that justified their throughput have shrunk or moved.
- Cross-availability-zone and cross-region data transfer is one of the least visible categories of cloud cost precisely because it is priced per gigabyte in a way most engineers do not see reflected in their architecture diagrams — a design that routes traffic between zones for no functional reason can silently become a recurring five- or six-figure annual cost.
- Logging and monitoring pipelines that forward every log line and metric at full fidelity, indefinitely, from every environment including staging and development, are one of the fastest-growing cost categories as organizations instrument more of their systems — ingestion and retention costs scale with volume, and volume rarely gets revisited once a pipeline is wired up.

3.6 Organizational waste: the categories that spreadsheets alone will not fix

Every category above is a technical waste pattern with a technical fix: rightsize the instance, delete the orphaned volume, add a lifecycle policy to the snapshot. The categories in this section are organizational, and they are the reason the technical fixes do not stick without a governance layer around them.

Duplicate environments

It is common for two teams working on adjacent problems to independently provision near-identical staging, testing, or demo environments because neither team knew the other's existed. Without a central inventory, duplication is not a mistake anyone makes on purpose — it is the predictable result of many teams each optimizing locally for their own speed.

Temporary infrastructure that becomes permanent

Every environment in this chapter started life described as temporary: a demo environment, a proof of concept, a load-testing cluster, a one-off migration script's supporting infrastructure. None of it was provisioned with malicious intent or even carelessness — it was provisioned to solve a real, immediate problem. The failure mode is not the creation. It is the absence of a mechanism that converts “temporary” into an actual, enforced expiration rather than a label that quietly becomes inaccurate.

Resources without owners

Cloud consoles do not require an owner field to be populated before a resource can be created, and most organizations do not enforce one through policy either. The result, at scale, is an inventory in which a meaningful share of running resources cannot be traced to a specific person or team without manual investigation — which means nobody can be asked whether the resource is still needed, and by default, nothing happens.

Resources without expiration dates

The absence of a default expiration is arguably the single highest-leverage gap in most organizations' cloud governance, because it is the one gap that, if closed, would automatically remediate a large share of every other category in this chapter without requiring anyone to make an active decision. A staging environment born with a 14-day time-to-live does not need a human to remember to delete it.

Ungoverned cloud accounts, projects, and subscriptions

As organizations grow, the number of cloud accounts, projects, and subscriptions in active use tends to grow faster than any central function's ability to track them — a new account for a hackathon, a new subscription for a proof of concept with a vendor, a personal developer account used for “just testing.” Each one represents a completely separate billing surface, IAM boundary, and security perimeter, and each one that exists outside a central inventory is, by definition, a blind spot.

3.7 Why individual teams do not notice

It is worth being explicit about why this waste persists in organizations staffed by capable, well-intentioned engineers, because the answer matters directly for the fix in Section 8. Four structural factors recur across nearly every organization examined for this report:

1. Cost is separated from the decision that creates it. The engineer who provisions a resource is almost never the person who sees the resulting invoice, and the person who sees the invoice almost never has the technical context to know which specific resource drove it.
2. The cost of underprovisioning is immediate and personal; the cost of overprovisioning is delayed and diffuse. An outage caused by an undersized instance generates a page, an incident review, and reputational cost to the engineer on call tonight. An oversized instance generates a slightly larger number on a monthly invoice that nobody individually is asked to explain.
3. Waste accumulates in small increments. No single orphaned volume, unused IP, or forgotten snapshot is expensive enough on its own to trigger an investigation. It is only the sum, reviewed rarely and usually only at a budget cycle boundary, that becomes alarming — by which point the specific decisions that produced it are long forgotten.
4. Non-production environments carry no natural forcing function to be reviewed. Production infrastructure is watched constantly because customers depend on it. Staging, development, and demo infrastructure is watched only as long as someone is actively using it — and then, structurally, by nobody at all.

3.8 Case vignette: the forgotten staging environment

A composite, representative example

A product team builds a new feature to demo for a Q2 board meeting. DevOps stands up a staging environment: three application servers, a managed database seeded from a production snapshot, a load balancer, an object storage bucket for uploaded test files, and full logging and monitoring, matching production configuration so the demo behaves realistically. The board meeting goes well. The feature is deprioritized for two quarters while the team focuses elsewhere. Nobody owns “turn off the Q2 demo environment” as a task, because it was never assigned to anyone — it was implicitly assumed to be temporary. Eleven months later, a cost review flags an unexplained monthly infrastructure line item. Reconstructing its origin — which project, which team, whether it is safe to delete — takes a platform engineer the better part of two days, because no tag, ticket, or record connects the running resources back to the original demo request. The direct cost of eleven idle months is real money; the two days spent reconstructing the history is a second, less visible cost that repeats every time this pattern occurs anywhere in the organization.

3.9 A closer look: the mechanics of overprovisioning

It is worth pausing on why “size up to be safe” is such a durable habit, because the framework in Section 8 only works if the underlying behavior is understood rather than simply criticized. When an engineer provisions a database or a compute instance, they are almost always doing so under real uncertainty about future load, with real memory of a past incident where undersizing caused a painful outage. Sizing up removes that specific, personally remembered risk at a cost that is diffuse, delayed, and denominated in a currency — next month's invoice — that the engineer will very likely never personally see attributed to their decision. Rational individuals, told to choose between a risk they will be personally blamed for tomorrow and a cost the company

will absorb next quarter, choose to avoid the risk. This is not a character flaw to be corrected through training; it is a predictable response to how the choice is currently structured, and it is why Section 8 focuses on changing the structure — through visible cost feedback and automated rightsizing — rather than asking engineers to individually exercise more restraint.

3.10 Provider-specific patterns worth knowing

While the underlying dynamics described in this chapter are consistent across every major cloud provider, a few patterns are common enough, and specific enough to individual platforms, to be worth naming directly.

- AWS: unattached Elastic Block Store volumes and unreleased Elastic IP addresses are consistently among the largest and easiest-to-reclaim categories of waste, because both continue billing after the instance that justified them is terminated, and neither is deleted automatically by default.
- Azure: orphaned managed disks left behind after a virtual machine is deleted, and unused Reserved Instance or Savings Plan commitments purchased for a workload that was later resized or retired, are recurring patterns — the latter is a particularly stubborn one because the commitment was often made with good intentions to save money, and then outlived the workload it was sized for.
- Google Cloud: idle persistent disks left attached to stopped (but not deleted) virtual machine instances continue to bill at full rate, and it is a common misunderstanding that stopping an instance meaningfully reduces its cost when the attached storage remains the larger share of the bill.
- Across all three: committed-use and reserved-capacity discounts, while genuinely valuable, lock in a specific level of spend regardless of whether usage later falls — Flexera's 2026 data found that fewer than half of organizations use any single commitment-discount program consistently across their estate, meaning most organizations are simultaneously leaving savings unclaimed on stable workloads and locking in excess spend on workloads that later shrank.

3.11 A second vignette: the cluster nobody remembers approving

The orphaned Kubernetes cluster

A platform team spins up a dedicated Kubernetes cluster to load-test a new service ahead of a major product launch, deliberately overprovisioned to simulate peak traffic safely. The launch succeeds. The load-testing cluster is meant to be torn down afterward, but the engineer who built it moves to a different team two weeks later, and the teardown step — which existed only as a line in that engineer's personal notes, not as a ticket or a runbook — is never picked up by anyone else. The cluster, provisioned for peak simulated load, sits at single-digit CPU utilization for the next year, consistent with the 8% average utilization figure documented in Section 3.4, because nothing about its configuration signals that it is disposable rather than a deliberately maintained piece of production infrastructure. It is eventually found not through a review process but by accident, when a cost-allocation exercise for an unrelated initiative surfaces a cluster nobody on the current platform team recognizes.

3.13 Logging, monitoring, and observability: a fast-growing category

Observability spend deserves separate attention because it is one of the few categories in this chapter that is actively growing as a share of the typical cloud bill, rather than simply persisting. As organizations instrument

more of their systems — more services, more granular metrics, more detailed tracing — the volume of telemetry data generated grows correspondingly, and both ingestion and retention are typically priced per unit of data. A common and specific failure mode is forwarding logs and metrics at full, unsampled fidelity from every environment including staging and development, and retaining all of it indefinitely because nobody has set a retention policy and reducing an existing pipeline's scope feels riskier than leaving it alone — the same asymmetry documented in Section 3.9 applied to data retention instead of compute sizing. A useful discipline, directly analogous to the tiered access patterns cloud storage providers already offer, is tiering observability data by age and by environment: full fidelity and fast query access for recent production data, downsampled or compressed storage for older data, and meaningfully shorter retention windows for non-production environments than for production ones.

3.14 Estimating waste by category: an illustrative breakdown

Pulling together the benchmarks cited throughout this chapter, the table below offers an illustrative allocation of where a typical dollar of cloud waste originates, useful for prioritizing a remediation effort rather than treating every category as equally urgent.

Waste category	Illustrative share of total cloud waste	Primary remediation lever
Idle and underutilized compute	~35%	Automated idle-resource detection and scheduled shutdown for non-production (Section 8.2)
Overprovisioned / oversized instances and Kubernetes requests	~25%	Continuous rightsizing against observed usage (Section 8.2, 8.7)
Orphaned storage, snapshots, and unused IPs	~15%	Automated cleanup with lifecycle policies (Section 8.2)
Non-production environments left running indefinitely	~15%	Default expiration / time-to-live enforcement (Section 8.1)
Networking, logging, and observability overhead	~10%	Data tiering and architecture review (Section 3.5, 3.13)

Illustrative allocation synthesized from Flexera 2025/2026, Harness 2025, Datadog State of Cloud Costs, and CNCF FinOps Survey 2024, as referenced throughout this chapter; an organization's own mix will vary by architecture and industry.

3.15 Summary: what cloud waste is actually telling the organization

None of the patterns in this chapter are evidence of incompetence. They are the predictable output of giving capable teams frictionless provisioning power without a corresponding, equally frictionless system of ownership, visibility, and expiration. Section 8 returns to this chapter's findings with a concrete framework — tagging and ownership policy, default expiration, chargeback, and rightsizing automation — built specifically to close the gap this chapter documents, without adding the kind of approval friction that would slow delivery back down.

4. AI Cost Waste

Cloud cost waste took organizations roughly a decade to fully understand. AI cost waste is repeating the same lessons in a fraction of the time, at a fraction of the warning.

4.1 The scale of AI adoption — and of AI-driven waste

Generative AI has moved from experimentation to default infrastructure faster than any prior cloud service category. Flexera's 2026 survey found that every single organization surveyed reported using generative AI in some capacity, with 45% describing their usage as extensive, up from 36% a year earlier. GenAI is now the third most widely used public cloud service overall, at 58% adoption, up from 50% the year before. That adoption curve is precisely what Flexera identifies as the leading cause of cloud waste rising for the first time in five years: AI workloads are harder to forecast, harder to rightsize, and — as this chapter documents — governed with far less maturity than the compute and storage categories organizations spent the last decade learning to control.

The gap in oversight is measurable. Even among large enterprises, only 47% report having a dedicated AI governance team or leader in place, despite 85% saying AI oversight has some designated owner. For small and mid-size organizations — the majority of the market — the picture is typically worse: AI access is granted through the same lightweight, self-service signup flow as any other developer tool, with no equivalent of the budget, procurement, and security review that would traditionally accompany a new six- or seven-figure infrastructure commitment.

4.2 Why AI cost behaves differently from cloud cost

Traditional cloud waste, documented in Section 3, is largely a story of paying for idle capacity — a server running when nobody needs it. AI cost waste is different in an important way: it is usage-based and can scale without any human decision at all. A single support ticket triaged by an AI agent might trigger dozens of model calls as the agent reasons, retrieves context, calls tools, and retries failures. A coding agent working through a large codebase can consume a working session's budget in minutes rather than hours. None of this requires a mistake in the traditional sense — it requires only the absence of a ceiling.

- Token consumption compounds non-linearly. Longer conversation histories, larger retrieved contexts, and multi-step agent reasoning all increase the number of tokens processed per task, and cost scales with tokens processed, not with the value of the outcome.
- Agentic workflows multiply the number of calls per task. Where a single API call once answered a single question, an agent may now make dozens of tool calls, self-corrections, and retries to complete one task — research on agentic workloads has found token consumption for agent tasks can reach thousands of times that of a simple query, and that running the identical task twice can produce up to a 30-fold difference in tokens consumed, with no guarantee that higher spend produces a better outcome.

- Retry loops turn a rate limit into a cost multiplier rather than a safety mechanism. When an API call fails or is throttled, an automated script or agent that is not designed to recognize the failure will frequently retry immediately — turning what should be a pause into a billing accelerant.
- Cost is invisible until the invoice arrives. Unlike a cloud console that shows a running server, most AI usage dashboards report spend with a meaningful lag, and few provide real-time, pre-request enforcement — by the time an alert email fires, a looping agent can already have burned through a significant additional amount.

4.3 Unlimited API keys and missing spending guardrails

The pattern described in Section 1.3 is now well documented across the industry: a developer, team, or entire company is issued an AI API key with no meaningful ceiling on how much it can spend. This is not a hypothetical risk. It is the leading, publicly reported cause of the largest AI cost incidents to date.

What happened	Reported impact	Root cause
Enterprise Claude access with no organizational spending limit configured	Reported at roughly $500 million in a single month	No spending cap set at the organization level for a large, agentic-workflow-heavy workforce
Employee at a fintech company built a personal side project using company AI credits over one week	$80,000 in model usage in seven days	Individual access with no per-user budget ceiling or usage review
Unattended coding-agent session left running overnight	$6,000–$6,531 in charges within hours	No session-level cost cap or automatic termination on idle/looping behavior
Retry-storm bug in a production integration	$72,000 in a single overnight run	Misconfigured retry logic with no circuit breaker on cumulative spend
Leaked Google Cloud API key	A $7 monthly budget resulted in an $18,000 bill overnight	Credential exposed publicly; no key-level rate limit or anomaly detection
Recursive agent tool-call loop (LangChain-based agent)	14,000 redundant tool calls, ~$437	No loop or recursion-depth detection in the agent framework
Company-wide AI budget exhausted early in the fiscal year	Full annual AI coding budget consumed by April	No per-team or per-quarter allocation; usage tracked only in aggregate

Sources: reporting aggregated from Axios, HackerNoon, DEV Community, KuCoin Learn, and independent engineering blogs, 2026. Company names are omitted where the original reporting did not confirm them; details reflect publicly available accounts at time of writing.

The common thread across every incident in that table is not recklessness. It is the absence of a default. Several widely used LLM platforms, at various points in 2026, have shipped with usage alerts — which notify a user after money has already been spent — rather than hard, pre-request spending caps that block a call before it happens. Even where hard caps exist, billing enforcement is frequently not instantaneous, meaning a genuinely runaway process can still spend a meaningful amount in the gap between the threshold being crossed and the system actually stopping.

4.4 Model selection and uncontrolled experimentation

Frontier models carry a meaningful cost premium over smaller, task-appropriate alternatives, and without routing guidance, the default behavior of an ungoverned team is to reach for the most capable model available for every task — including tasks, like classification, simple extraction, or short-answer lookups, that a much smaller and cheaper model would handle just as well. This is a rational individual choice in the absence of any visible cost signal: if there is no budget, no visibility, and no routing logic, using the most powerful model for everything is simply the path of least resistance, and the cost difference compounds across every call, every day, for the life of the workflow.

Uncontrolled experimentation compounds the same problem in a different way. Developers evaluating a new model, a new prompt strategy, or a new agent framework will naturally run the same task many times while iterating — which is exactly the right way to build good AI features, and exactly the wrong thing to leave unmetered. Without a development-tier budget that is deliberately set tighter than production, experimentation cost and production cost blend into a single, unexplained number on the monthly bill.

4.5 Agentic AI: a genuinely new cost category

Agentic workflows — AI systems that plan, call tools, retrieve information, and take multiple sequential actions with limited human review of each step — are the fastest-growing and least governed category of AI spend documented in this report. Because an agent's cost is a function of how many steps it takes to complete a task rather than a fixed price per interaction, agentic cost is inherently less predictable than traditional API usage, and organizations that budget for AI the way they budgeted for a single-call API integration are structurally unprepared for it.

- Hook and recursion loops: several documented incidents in 2026 trace back to automation — a hook script, a retry wrapper, or a chained tool call — that triggers itself again, with no built-in timeout or recursion-depth limit, so the agent continues spending until an external system (or an alarmed human) stops it.
- “Vibe coding” and AI-assisted development: AI coding tools have made it dramatically faster to generate working code, but the same speed that benefits legitimate development also means an unsupervised agent session can consume a meaningful budget in minutes rather than hours — a single engineer running an AI coding assistant intensively can reasonably rack up $500–2,000 a month in usage simply by working effectively, a figure that was not part of most teams' cost models even a year ago.
- AI automation workloads: scheduled jobs, triage bots, and background agents that call AI models on a recurring basis accumulate cost the same way idle cloud infrastructure does — quietly, continuously, and without the kind of single dramatic event that would trigger a review.

4.6 Beyond the API call: embeddings, vector databases, and inference infrastructure

As AI usage matures beyond simple prompt-and-response calls, organizations increasingly build retrieval-augmented systems and self-hosted inference — each of which introduces its own, less obvious cost surface.

- Embeddings and vector databases: generating and storing vector embeddings for a large document corpus, and running a vector database to serve retrieval queries against it, adds a persistent infrastructure cost layer that is easy to underestimate during a proof of concept and easy to leave over-provisioned once the system reaches production scale.
- GPU costs and idle inference capacity: organizations that self-host models to control cost or data residency take on GPU infrastructure that is expensive whether or not it is being used — a GPU with a model loaded into memory but no active requests still consumes power and still bills at the reserved rate, and industry benchmarking puts average GPU utilization across measured workloads at roughly 5–23%, meaning the large majority of provisioned GPU capacity sits idle at any given time. For the first time in nearly two decades, GPU capacity prices have also begun rising rather than falling, removing one of the safety nets — the assumption that waiting will make idle capacity cheaper — that organizations have historically relied on.
- Fine-tuning and model hosting: custom fine-tuning runs and dedicated model hosting introduce both a one-time training cost and an ongoing hosting cost that persists whether or not the fine-tuned model is still being actively used in production — a common failure mode is a fine-tuned model created for a specific evaluation that outlives the evaluation itself.

4.7 Multiple providers, duplicate subscriptions, and shadow AI

As AI tooling has proliferated, so has the number of distinct AI-related subscriptions and API accounts an organization accumulates — often without any central inventory. Different teams independently subscribe to different AI coding assistants, different model providers, and different point-solution AI products that overlap significantly in capability. Each one is billed separately, sized independently, and reviewed by nobody in particular.

This overlaps directly with the security risk documented in Section 5: IBM's 2025 Cost of a Data Breach Report found that breaches involving “shadow AI” — AI tools adopted and used without formal IT approval or oversight — cost organizations an average of $670,000 more than breaches that did not involve shadow AI, driven by longer detection times and broader, harder-to-trace data exposure. The same lack of central visibility that produces duplicate subscriptions and wasted spend is, in a meaningful share of cases, the identical gap that later produces a materially more expensive security incident.

4.8 Why AI cost governance is structurally harder than cloud cost governance

Organizations that have spent years building FinOps discipline around cloud infrastructure often assume the same playbook will translate directly to AI spend. It translates partially, but AI introduces at least three structural differences that make the problem harder, not merely bigger.

5. Attribution is weaker. A cloud resource is tied to an account, a project, and increasingly a mandatory tag. An API key is frequently shared across an entire team or embedded in a service with no per-request record of which human or workflow initiated a given call — as one industry analysis put it, when usage spikes, nobody knows whether it was the machine learning team, an intern, or a runaway job, because the key itself carries no identity.

6. Cost per unit of work is far less predictable. A virtual machine's hourly cost is fixed regardless of what runs on it. An agent's cost for a given task varies with context length, number of reasoning steps, and number of tool calls — documented as ranging up to 30-fold for functionally identical tasks — which means historical spend is a much weaker predictor of future spend than it is for traditional infrastructure.
7. The tooling is younger. FinOps for cloud has a decade of mature tooling, established job titles, and cross-industry benchmarks behind it. FinOps for AI is, by the FinOps Foundation's own account, only now emerging as a distinct top forward-looking priority — which means most organizations are building governance practices for AI spend from a much earlier starting point than they had for cloud.

> **The mechanism that matters, not the dollar figure**
>
> Every incident cited in this chapter, from a few hundred dollars to hundreds of millions, traces back to the same missing control: a spending ceiling that was never configured, at the individual, team, or organizational level, before access was granted. The dollar figures vary enormously by scale of usage. The root cause does not vary at all.

4.9 The economics of tokens, explained for a non-technical reader

Understanding why AI cost behaves so differently from a typical software subscription requires understanding, at a basic level, what is actually being metered. Large language models process text — both what is sent to them and what they generate in response — as small chunks called tokens, roughly corresponding to word fragments. Providers charge per token processed, with output tokens typically priced several times higher than input tokens because generating text is more computationally expensive than reading it. A single request to a capable model, given a large amount of context — a long document, a big chunk of source code, an extended conversation history — can involve hundreds of thousands of tokens, and an agentic workflow that makes dozens of such requests to complete one task can multiply that figure many times over within a single task, let alone a single day of use.

This matters for a governance conversation because it means AI cost does not scale the way most finance and engineering leaders are used to budgeting for software. A traditional SaaS subscription costs the same whether an employee uses it intensely or barely at all. AI API cost scales directly and immediately with how much text is processed, which means usage that looks identical from the outside — “the team is using the coding assistant” — can vary in cost by an order of magnitude depending on the size of the codebase being worked on, the length of the conversation, and how many autonomous steps an agent takes to complete a task. Budgeting for AI the way an organization budgets for a fixed-price software license is, structurally, the wrong model, and it is one of the more common reasons AI spend surprises finance teams who assumed their existing software-budgeting process would simply extend to cover it.

4.10 Signs an organization already has an AI cost governance problem

Because AI spend is new enough that many organizations have not yet built a mental model for what “normal” looks like, the checklist below is offered as a practical, non-technical diagnostic. Answering yes to more than

two or three of these is a strong signal that the patterns documented earlier in this chapter are already present, whether or not they have yet produced a visible incident.

- Nobody can currently list every AI API key or account in active use across the organization without contacting each team individually.
- At least one AI-related invoice in the past year was materially higher than anyone expected before it arrived.
- Developers can generate a new API key without any approval step, budget check, or expiration date attached.
- There is no established process for deciding which model tier is appropriate for a given task — developers choose based on personal preference or habit.
- Nobody has reviewed, in the past quarter, which AI subscriptions or API accounts are still actively used versus quietly abandoned.
- Agentic workflows or scheduled jobs that call an AI model run without a defined per-run or per-day cost ceiling.

4.11 GPU economics in more detail

For organizations that self-host models rather than calling a provider's API — whether for cost control, data residency, or latency reasons — GPU capacity introduces a cost dynamic that is genuinely unlike anything most infrastructure teams have budgeted for before. Unlike general-purpose compute, where idle capacity is cheap and prices have reliably fallen year over year for two decades, GPU capacity is both expensive to leave idle and, as of early 2026, rising in price rather than falling — a structural break from the pattern every prior generation of infrastructure cost planning was built on. A GPU with a large model loaded into memory continues to draw power and continues to bill at its reserved rate whether or not it is actively serving a request, and because loading a large model back into memory from cold storage can take meaningful time, teams are structurally incentivized to keep GPUs warm and idle rather than scale them down between bursts of demand — which is precisely why measured GPU utilization sits as low as it does across the industry.

The practical consequence for a governance program is that GPU capacity planning cannot be treated as a simple extension of the compute rightsizing practices described in Section 3.2. It requires its own review cadence, its own ownership, and — for most mid-size organizations — a genuine build-versus-rent decision, revisited regularly, about whether self-hosting a given model is still cheaper than the equivalent usage-based API spend once idle GPU time is properly accounted for.

4.12 What good AI cost governance looks like in practice

As a counterpart to the warning signs in Section 4.10, it is worth describing the positive state a governance program is working toward, concretely enough that progress against it is measurable rather than aspirational.

- Every AI API key is issued to a named individual, team, or automated workflow — never shared anonymously — and carries a spending ceiling configured at issuance, not added later.

- Usage is visible, broken down by team and by workflow, in a dashboard that engineering leads check the way they already check application performance dashboards — not a report finance requests quarterly.
- A lightweight, documented process exists for choosing which model tier fits a given task, and it is the path of least resistance rather than a guideline developers have to remember to follow.
- Development and experimentation usage is metered separately from production, so a burst of legitimate testing does not read as an unexplained anomaly, and does not draw down the same budget silently.
- When usage does spike unexpectedly, the organization finds out within minutes through an enforced ceiling or an active alert — not weeks later, on an invoice.

None of these five properties require an organization to slow down its AI adoption. Every one of them is achievable with tooling that already exists, described at a category level in Section C, and organizations that reach this state consistently report that their AI spend becomes not just safer but more predictable — a secondary benefit that matters just as much to a CFO as the risk reduction does.

5. Security During Delivery

"We need to ship before the deadline" and "we need to secure what we ship" are treated as competing priorities in most delivery organizations. The evidence suggests that treating them as competing is precisely what makes both outcomes worse.

5.1 The delivery-versus-security tension, and why it is a false one

Security work that is postponed until "after launch" is, in practice, security work that competes for attention with whatever the next deadline is — and the next deadline almost always wins, because it is scheduled and the security review is not. This is not a statement about any individual team's priorities; it is a structural description of how most delivery organizations allocate attention. The consequence, documented throughout this chapter, is that the shortcuts a team takes under deadline pressure to hit a demo or a launch date do not disappear once the deadline passes. They persist, quietly, as exactly the kind of exposure that turns into an incident.

5.2 Secrets in Git and CI/CD: a problem of scale, not carelessness

GitGuardian's fifth annual State of Secrets Sprawl report, published in March 2026, found that 28,649,024 new hardcoded secrets — API keys, passwords, tokens, and certificates — were exposed in public GitHub commits during 2025 alone, a 34% year-over-year increase and the largest single-year jump the firm has recorded since it began tracking the metric. Secrets have grown roughly 1.6 times faster than the active developer population since 2021, meaning the problem is outpacing not just code volume but the growth of the workforce writing that code.

28.6M	81%	64%
new hardcoded secrets exposed on public GitHub in 2025 (+34% YoY)	year-over-year growth in leaked AI-service credentials specifically	of secrets confirmed leaked and valid in 2022 were still active in 2026

Source: GitGuardian, The State of Secrets Sprawl 2026.

Three findings from that report are especially relevant to the delivery-chain framing of this report. First, internal repositories — the ones teams consider "safe" because they are private — are six times more likely than public repositories to contain hardcoded secrets, because developers behave with less caution when they believe nobody outside the company can see their commits. Second, AI-assisted commits leak secrets at roughly double the baseline rate of ordinary commits, and AI coding tools can generate working code that includes hardcoded credentials without flagging them, because a code-completion model has no inherent understanding of what constitutes a secret. Third, and most consequential for long-term risk, 64% of secrets that were confirmed leaked and valid in 2022 were still exploitable when GitGuardian retested the same dataset in January 2026 — meaning the large majority of exposed credentials are never revoked at all, and a shortcut taken years ago can remain a live door into an organization's systems today.

Secrets sprawl also extends well beyond source code. Roughly 28% of secrets incidents now originate in collaboration and productivity tools — Slack, Jira, Confluence — rather than in a code repository at all, and these non-code leaks are more likely to be classified as critical, because credentials pasted into a chat message or a ticket are exposed to a broader, less-monitored audience than a line buried in commit history. New integration standards have reproduced the same pattern even faster than established ones: the Model Context Protocol, which emerged in early 2025 to connect AI models to external tools and data sources, already had 24,008 unique secrets found exposed in its configuration files within its first year, largely because official setup documentation demonstrated hardcoding credentials directly into configuration files as the default, fastest path to a working integration.

5.3 Leaked cloud and AI credentials: from exposure to exploitation

A leaked credential is not a theoretical risk that sits quietly until someone happens to notice it. Security researchers have documented that automated scanning by threat actors routinely harvests exposed cloud credentials from public repositories within minutes of exposure, while the median time for an enterprise security team to actually remediate a leaked secret once found runs into several months. That gap — measured in minutes for an attacker to find a credential, and months for a defender to revoke it — is where the large majority of credential-based breaches occur.

This is not an abstraction. In one incident documented in mid-2026, a contractor working with a cybersecurity firm committed roughly 844 megabytes of cloud infrastructure credentials, internal system passwords, SSH keys, and Kubernetes configuration files to a public repository. It remained publicly accessible for approximately six months before a researcher discovered it. In an unrelated 2025 incident, an AI company's misconfigured database was left publicly accessible with no authentication at all, exposing over a million records including user chat histories, API authentication tokens, and backend credentials — triggering platform-wide service interruptions and forcing the company to suspend new signups while it investigated. Neither incident required a sophisticated attack. Both required only that a credential or a database be left open, and that nobody was watching for it.

5.4 Excessive permissions, public storage, and open ports

Under deadline pressure, the fastest way to make an integration work is almost always the least secure one: granting a service account broad, unscoped permissions rather than the narrow set it actually needs; leaving a storage bucket or database publicly accessible during development because configuring proper access control takes longer than leaving it open; leaving a port open on a security group because narrowing it requires knowing exactly what traffic is legitimate, and that takes time nobody has budgeted. Each of these choices is individually rational in the moment — it unblocks the task in front of the engineer — and each is a form of security debt that, absent a deliberate cleanup step, has no natural expiration.

Cloud misconfiguration of exactly this kind is now recognized as one of the leading causes of cloud data breaches, serious enough that the U.S. Cybersecurity and Infrastructure Security Agency issued a binding directive in late 2024 requiring federal agencies to remediate cloud misconfigurations specifically because

they had become such a widespread and exploited source of exposure. The shared responsibility model that underlies every major cloud platform assigns configuration security to the customer, not the provider — which means every one of these shortcuts is, by design, the organization's own risk to own, not something a cloud vendor will catch or correct on its behalf.

5.5 Insecure staging and debug environments

Staging and debug environments are, almost by definition, configured with security controls loosened relative to production: authentication is often simplified or disabled outright to speed up testing, verbose debug logging and error messages that would never be acceptable in production are left on because they help engineers move faster, and the environment is frequently seeded with a full or partial copy of production data because realistic data makes for a more convincing demo. Each of those choices makes sense for the narrow purpose of getting a demo working quickly. None of them is reviewed against the same security bar as production, and because staging environments are also the ones most likely to be forgotten and left running — as documented in Section 3.8 — they combine the two worst properties an attacker could ask for: real or realistic data, and reduced scrutiny, running indefinitely with nobody watching.

5.6 Missing patching and forgotten servers

Every server, database, and container image that is actively used receives some baseline level of attention: someone notices when it behaves oddly, someone eventually gets around to applying security updates because the system is visibly part of daily operations. A forgotten staging server receives none of that attention by definition — it is not part of anyone's daily operational awareness, so it is not part of anyone's patching cycle either. Over months or years, a forgotten server accumulates unpatched vulnerabilities at exactly the same rate as a production server, with none of the monitoring that would catch exploitation of those vulnerabilities in progress.

5.7 The financial cost of getting this wrong

IBM's twentieth annual Cost of a Data Breach Report, based on analysis of 600 breached organizations across 17 industries, found the global average cost of a data breach was $4.44 million in 2025, a rare year-over-year decline driven primarily by faster detection and containment through AI-assisted security tooling. That global decline masks sharply divergent regional and structural trends that are directly relevant to the delivery-chain argument of this report.

Breach cost driver	Average cost impact	Why it matters to delivery
United States average breach cost	$10.22 million — an all-time high	Reflects higher regulatory fines and litigation exposure for exactly the kind of exposure documented in this chapter
Breach involving data spread across multiple environments	$5.05 million — the highest of any configuration studied	Directly describes duplicate and ungoverned environments as documented in Section 3.6

Breach cost driver	Average cost impact	Why it matters to delivery
Breach involving “shadow AI” (unauthorized AI tool use)	+$670,000 above the baseline average	Directly describes the AI governance gaps documented in Section 4
Mean time to identify and contain a breach	241 days overall; 292 days when stolen credentials were involved	The exposure window during which a leaked secret documented in Section 5.2 remains exploitable
AI-related security incidents with a documented access-control gap	97% lacked proper access controls, per IBM	Directly describes the excessive-permission pattern documented in Section 5.4

Source: IBM Cost of a Data Breach Report 2025 (Ponemon Institute, sponsored and published by IBM).

5.7b Breach cost by industry

The average figures cited above mask meaningful variation by sector, which matters directly for how an organization should prioritize the practices in Section 8 against its own risk profile — a point developed further in Section A's industry lens.

Industry	Average breach cost (2025)	Why
Healthcare	$7.42 million	Extensive patient records, strict compliance requirements, high resale value of protected health data
Financial services	$5.56 million	Regulatory fines, fraud liability, and sophisticated targeted attacks
Industrial	$5.00 million	Operational disruption cost, extended detection times in OT-adjacent environments
Energy	$4.83 million	Critical-infrastructure regulatory exposure and operational disruption
Technology	$4.79 million	High-value intellectual property and large-scale user data at stake
Pharmaceuticals	$4.61 million	Regulated data plus high-value research and development information

Source: IBM Cost of a Data Breach Report 2025.

5.8 Security as a process, not a final audit

The organizations that avoid the outcomes documented in this chapter are not, in general, the ones that run the most thorough pre-launch security audit. A single audit, however rigorous, is a snapshot of a system's security posture at one moment — and every pattern in this chapter describes drift that happens after that moment: a staging environment left running past its usefulness, a credential that outlives the project it was created for, a permission grant that was reasonable when the integration first shipped and excessive a year later as the system around it changed. Security treated as a gate before launch cannot see any of that drift, because by definition it stops looking the moment the gate is passed.

The alternative — examined in detail in Section 8 — is to treat security the same way this report treats cost governance: as a set of defaults that travel with a resource for its entire lifecycle rather than a checklist applied once at the beginning. Secrets that are never hardcoded because a secrets manager is the path of least resistance rather than a departure from it. Permissions that are scoped narrowly by default and expanded only on request rather than granted broadly and never revisited. Staging environments that inherit the same monitoring and patching cadence as production because they are provisioned from the same governed template. None of this requires slowing delivery down. It requires making the secure default the fast default — which is precisely the same principle this report applies to cost governance in the chapters that follow.

5.9 CI/CD pipelines and the software supply chain

Everything documented earlier in this chapter about individual developer behavior is compounded by a newer and less well-understood risk surface: the automated build and deployment infrastructure that sits between a developer's commit and a running production system. CI/CD runners routinely hold some of the most powerful credentials in an organization — the access needed to deploy to production — and GitGuardian's research into a 2026 supply-chain attack found that 59% of the machines compromised in that incident were CI/CD runners rather than personal developer laptops, a finding the researchers described as expanding the secrets problem well beyond the individual endpoint most security training is still focused on.

That same research, examining machines compromised during the incident, found that a typical exposed system held not one but eight separate copies of the same live credentials, scattered across environment files, shell history, IDE configuration, cached authentication tokens, and build artifacts — illustrating that a single credential, once created, tends to be copied and cached in far more places than the person who created it is aware of. A closely related supply-chain incident in the same period, involving a compromised open-source package used for connecting AI models to external tools, was found to harvest SSH keys and cloud credentials directly from the developer machines and CI environments that had installed it, demonstrating that the AI tooling supply chain — packages, plugins, and integrations installed to speed up AI-assisted development — is now a live target in exactly the way traditional open-source dependencies have been for years.

- Treat CI/CD runners as high-value targets, not neutral infrastructure: apply the same scoped-access and monitoring standard recommended for production systems in Section 8.4, rather than the looser standard many organizations still apply to “just the build system.”
- Use short-lived, dynamically issued credentials for pipeline access wherever the tooling supports it, rather than long-lived static credentials stored as pipeline secrets — a credential that expires in minutes is far less useful to an attacker than one that remains valid for years, directly addressing the 64% non-remediation rate documented in Section 5.2.
- Audit third-party packages, plugins, and MCP integrations installed into build and development environments with the same scrutiny applied to production dependencies — the convenience of a one-line install is, per the incidents above, an increasingly common initial access vector in its own right.

5.10 Identity and access management as a lifecycle, not a one-time grant

Section 5.4 described why excessive permissions are so often granted under deadline pressure. The less-discussed half of the problem is that permissions, once granted, are almost never revisited — an access grant made for a specific, time-bound project frequently outlives the project itself, because revoking access is nobody's assigned task in the same way granting it was. Over the life of a typical service account or long-tenured employee, this produces a slow, one-directional accumulation of privilege that security teams sometimes call privilege creep: permissions only ever added, never subtracted, until the account's actual access bears little resemblance to what its current work requires.

Treating access as a lifecycle rather than a one-time grant means pairing every permission grant with a review trigger — a project's end date, a role change, a fixed quarterly recertification — that forces an explicit decision to keep, narrow, or revoke access, rather than defaulting to silence. This is the same default-to-expiration principle recommended for infrastructure in Section 8.1, applied to identity instead of compute, and it is one of the highest-leverage, lowest-cost controls available to an organization that has not yet built the fuller governance program described in Section 8.

6. Root Cause: Mapping Waste and Risk to the Delivery Chain

6.1 One gap, three symptoms

Sections 3 through 5 documented three bodies of evidence that are usually researched, reported on, and budgeted for separately: cloud cost waste, AI cost waste, and security exposure during delivery. Read individually, each looks like a distinct problem requiring a distinct specialist — a FinOps analyst for cloud, an AI platform owner for model spend, a security engineer for exposure. Read together, against the delivery chain mapped in Section 2, the same handoffs recur across all three.

Delivery chain handoff	Cloud cost symptom	AI cost symptom	Security symptom
Developer creates a resource	Instance sized “to be safe,” no owner tag	API key generated with no spend cap	Credential hardcoded for speed
DevOps provisions environment	Staging built like production, never scheduled to expire	AI service wired in without usage attribution	Debug/staging config left in place
Handoff to “temporary” use	Demo environment left running indefinitely	Experimentation budget blends into production spend	Access controls loosened “for now,” never tightened
Nobody assigned to review	Orphaned disks, snapshots, unused IPs accumulate	Runaway or looping usage goes undetected	Leaked secret remains valid for years
Cost/risk surfaces later	Finance asks why the bill is high	Finance asks why the AI bill is high	Security or an outside party discovers the exposure

The pattern in every row is the same: a resource, a credential, or a configuration choice is created quickly to solve an immediate problem, and nothing in the process that created it also assigns an owner, a cost ceiling, an expiration date, or a security baseline. That absence is the velocity-governance gap defined in Section 1, and it is genuinely the same gap in all three domains — which is why the fix, developed in Section 8, is also genuinely the same fix applied in three places rather than three separate fixes.

6.2 Incentive misalignment: who provisions is not who pays

Underneath the structural gap is a simpler, more human explanation that is worth stating plainly: in most organizations, the person who decides to create a resource is not the person who is accountable for its ongoing cost or risk, and the two are rarely connected by any feedback loop. A developer choosing an instance size, an API key's permissions, or whether to add an expiration tag is optimizing for one thing — getting their immediate task done — because that is what they are measured on. Nobody in that moment is measuring the ongoing monthly cost or residual risk of the choice, so nothing about the decision reflects it.

This is not a call for developers to be more careful, and this report is explicit that it should not be read that way. Asking individuals to voluntarily absorb a concern that the organization's structure does not reward them for absorbing is a request that will not scale, no matter how well-intentioned the individuals are. The fix that works, examined in Section 8, changes the structure so that ownership, cost visibility, and expiration are

attached automatically at the moment of creation — not held in reserve as something a conscientious engineer might remember to do.

6.3 Composite case study: from demo to $40,000 surprise

The following composite scenario is constructed from patterns documented independently across Sections 3, 4, and 5, to show how they compound when they occur together — which, in organizations without a governance layer, they routinely do.

Week 1 — The demo

A team builds a prototype AI-powered support triage feature for a stakeholder review. A developer generates a personal API key against a frontier model to move quickly, with no spending cap, because setting one is an extra step and the deadline is two days away. DevOps stands up a staging environment — three servers, a database seeded from a production snapshot, full logging — to host the demo. The demo goes well.

Weeks 2–20 — The silence

The feature is deprioritized while the team focuses on the current quarter's roadmap. The staging environment keeps running. The API key, still active, is left wired into a scheduled job someone built to keep testing the triage logic in the background — nobody remembers to turn the job off either. The database snapshot used to seed staging includes real customer support tickets copied from production, and the staging environment's access controls, simplified for the original demo, are never tightened.

Month 6 — The discovery

A routine cost review flags an unexplained infrastructure line item and an unexplained AI usage line item on the same monthly statement, together totaling roughly $40,000 in cumulative, unbudgeted spend across compute, storage, and model calls. Reconstructing the source takes a platform engineer most of a week, because no tag or ticket connects the running resources back to the original demo request. In the course of that investigation, the team also discovers the staging database — containing real customer data — has been reachable with only the simplified demo-era access controls for the entire six months.

Nothing in this scenario required a single dramatic mistake. It required five small, individually reasonable shortcuts — skip the spend cap to save two minutes before a deadline, leave staging running because nobody owns turning it off, seed staging with real data because it makes for a better demo, leave the access controls loose because tightening them wasn't urgent, and go six months without a review because nothing forced one — compounding in the absence of any governance layer designed to catch them. Section 7 puts an illustrative number on how often this pattern repeats across a representative organization, and Section 8 details exactly which of those five shortcuts a governance framework is designed to close automatically.

6.4 The cultural dimension

It would be a mistake to read this chapter as purely a systems and incentives problem with no cultural component at all, because the two reinforce each other. In organizations where speed is celebrated publicly

— the fastest ship, the quickest demo turnaround — and where cleanup, decommissioning, and cost review are neither celebrated nor even visible, the incentive gap documented in Section 6.2 is amplified by a status gap on top of it. Turning off a staging environment does not appear in a sprint retrospective as an accomplishment; shipping a new feature does. Over time, this shapes what teams implicitly understand their job to be, independent of any formal incentive structure. The governance frameworks in Section 8 work best, and last longest, in organizations that also make a deliberate, visible effort to recognize the unglamorous work of ownership and cleanup with the same seriousness they recognize the glamorous work of shipping — not as a slogan, but as a genuine input to how teams and individuals are evaluated.

6.5 What this means for the org chart

A frequent question this diagnosis raises is whether closing the gap requires a new organizational function — a Chief AI Officer, a dedicated FinOps team, a platform engineering group — or whether it can be absorbed into existing roles. The honest answer is that it depends less on titles than on whether the four principles in Section 8.1 have a genuine, accountable home somewhere in the organization, with the authority described in Section 8.5 to act across team boundaries. In a smaller organization, this can credibly be a part of an existing platform or DevOps lead's mandate, provided it is an explicit, resourced part of that mandate rather than an unstated expectation layered on top of an already full plate. In a larger organization, the Cloud Center of Excellence and FinOps structures referenced in Section 8.5 exist precisely because the coordination problem across many independent teams becomes too large for any single embedded role to absorb informally. The specific structure matters far less than avoiding the most common failure mode observed across the organizations examined for this report: governance responsibility that exists on paper, assigned to a role or committee with no dedicated time, no budget, and no authority to enforce anything — which functions, in practice, identically to no governance function existing at all.

Section D: Three Real Incidents, in More Detail

The composite vignettes earlier in this report are illustrative. The three incidents below are drawn from public reporting, to show that every pattern this report describes has already happened, in full, at real organizations.

D.1 The uncapped enterprise account

In mid-2026, multiple outlets reported that a large enterprise had accumulated approximately $500 million in Claude API usage within a single month. According to the reporting, the proximate cause was straightforward: the organization had granted broad, agentic-workflow-enabled access across a large workforce without configuring an organization-level spending limit. Commentary following the story noted that an individual engineer running an AI coding assistant intensively can reasonably generate $500 to $2,000 in monthly usage simply by working effectively — a figure that, multiplied across a large, unmetered workforce running increasingly autonomous, multi-step agentic workflows, compounds into a very large number very quickly, with no single dramatic error required at any point. Independent commentary published shortly afterward from an unrelated small agency reported discovering they, too, lacked any hard spending cap on their own AI accounts, describing the same missing control as “mechanically achievable” and “reproducible” at any scale — and taking roughly twenty minutes to fix once someone finally checked.

The broader context reported alongside this incident is also instructive: within the same period, a major ride-hailing company was separately reported to have exhausted its entire annual AI coding budget by April, four months into its fiscal year, and a large technology company was reported to have quietly discontinued a coding-assistant subscription internally and directed engineers back to a lower-cost alternative. None of these three events, reported within days of each other, involved a security breach, an attacker, or any malicious actor. Each involved a spending ceiling that nobody had configured before granting access — precisely the mechanism this report identifies as the common root cause in Section 4.3.

D.2 The exposed database

In January 2025, a widely used AI company suffered a breach when a database was left publicly accessible with no authentication required. Reporting on the incident indicates it exposed more than a million sensitive records, including user chat histories, API authentication tokens, backend credentials, and internal operational logs — the kind of internal system information that, in the wrong hands, can be used to understand and potentially manipulate connected systems rather than simply read exposed data. The incident triggered service interruptions and led the company to suspend new user signups while it investigated, and drew regulatory attention in the affected markets.

What makes this incident particularly relevant to this report is what it did not require: no sophisticated exploit, no zero-day vulnerability, no insider threat. It required a database configured without authentication, left running, and not caught by any routine check before an outside party found it — the exact combination of a forgotten or insufficiently reviewed resource and a security control that was never applied described

throughout Section 5. It is a useful illustration of the point made in Section 5.4: cloud misconfiguration of this kind is now recognized as one of the leading causes of cloud data breaches industry-wide, serious enough to have prompted binding federal directives specifically targeting it.

D.3 The week-long side project

A fintech company with a reported valuation in the billions discovered that an employee had spent roughly $80,000 in AI model usage over the course of a single week, building a personal side project — by public accounts, a game — using company AI credits. Commentary on the incident noted that a week of active AI-assisted development can easily run to hundreds or thousands of API calls, and that at frontier-model pricing, a single call that loads a large amount of context can cost several dollars on its own; at high call volumes sustained over multi-hour sessions, the arithmetic adds up quickly. The commentary also noted that this pattern was not unique to this one company — concurrent reports around the same period cited other well-resourced organizations with sophisticated infrastructure and cost controls independently discovering and then implementing spending caps in response to unexpected AI token costs of their own.

The common thread the original reporting drew out is the same one this report has emphasized throughout Section 4: AI coding adoption moved faster than spend governance did. Individual engineers were given API keys with no visible ceiling, began experimenting — which is, in isolation, exactly the behavior an organization wants to encourage — and nobody had set a limit on how far that experimentation could go before the first billing cycle made the gap visible.

D.4 What these three incidents have in common

None of the three organizations involved were unsophisticated. Each operated at a scale where meaningful engineering and security resources existed. In each case, the failure was not a lack of capability — it was the absence of a specific, boring, entirely achievable control: a spending ceiling, an authentication requirement, a review process for access that outlives its original purpose. This is, in miniature, the argument of this entire report: the gap between what organizations are capable of building and what they have actually configured is where the cost and the risk both live, and closing it is a matter of discipline and default-setting rather than of acquiring new technical capability the organization does not already have.

D.5 Smaller in scale, identical in mechanism

Not every incident behind this report's evidence base ran to eight or nine figures, and the smaller ones are, in their own way, more useful for illustrating how ordinary the underlying mechanism is. In one widely discussed case, a developer's Google Cloud account — with a modest, everyday monthly budget in the single-digit dollars — was billed roughly $18,000 overnight after a single API key was left exposed and picked up by an automated scanner. In another, an AI agent set up to scan a personal home network ran up a $6,531 cloud bill within days, because no hard limit had been configured on the account it was running against. Neither incident involved a company at all — both involved an individual, and both are reproducible, per the mechanism

described throughout Section 4.3, at any scale from a single hobbyist account up to the largest enterprise, because the missing control is identical regardless of who is missing it.

This is the point worth taking from Section D as a whole: scale changes the number on the invoice. It does not change the fix. A hard spending ceiling, configured before access is granted, would have prevented every incident described in this section, at every scale, without exception.

D.6 The credential that sat exposed for six months

A separate and instructive incident, referenced briefly in Section 5.3, involved a contractor working with a cybersecurity firm who committed roughly 844 megabytes of sensitive material — cloud infrastructure credentials, internal system passwords, SSH keys, and Kubernetes configuration files — to a public code repository. The exposure was not caused by an attack of any kind; it was caused by a routine commit that included files that should have been excluded, made by someone with legitimate access, in the ordinary course of work. It remained publicly accessible for approximately six months before an outside security researcher discovered it, a window well within the pattern documented in Section 5.2: automated scanning finds exposed credentials within minutes of exposure, while the organizations that created them typically take vastly longer to notice on their own.

What makes this incident particularly worth including in a report aimed at cloud and AI governance, specifically, is what was exposed alongside conventional credentials: infrastructure-as-code and Kubernetes configuration files of exactly the kind this report recommends using to enforce governed defaults in Section 8.1 and 8.2. The same automation that makes governed, consistent provisioning possible also concentrates an organization's most sensitive access configuration into a small number of files — which makes protecting those specific files, through the secrets-management and CI/CD scanning practices in Section 8.4 and 5.9, a correspondingly higher-leverage priority as an organization adopts more infrastructure automation, not a lower one.

7. Quantifying the Hidden Cost

The figures below are an illustrative model, not a forecast for any specific organization — but the inputs are drawn directly from the sourced benchmarks in Sections 3 through 5, and the exercise is worth running with an organization's own numbers.

7.1 A simple model for estimating your own exposure

Every organization's actual waste and risk exposure depends on its scale, maturity, and industry, and this report makes no claim to predict a specific figure for a specific reader. What it can offer is a transparent, three-input model built entirely from the benchmarks cited earlier in this report, so that any finance or engineering leader can substitute their own numbers and get a directionally useful estimate in minutes rather than months.

8. Cloud waste estimate = monthly cloud spend × estimated waste rate. This report uses Flexera's 2026 industry benchmark of 29% as a starting midpoint; organizations with weak tagging, no chargeback, and no expiration policy should expect to sit at or above that figure, while organizations with mature FinOps practice can sit meaningfully below it.
9. AI waste estimate = monthly AI/model spend × estimated ungoverned-usage share. This report does not assert a single industry-wide percentage for AI waste, because the category is too new and too dependent on whether an organization has any spend controls at all — but the incident data in Section 4.3 makes clear the range runs from a modest single-digit percentage in well-governed organizations to effectively unbounded in organizations with no spending caps at all.
10. Annualized breach-risk exposure = estimated probability of a material incident in the next 12 months × average cost of a breach for the organization's region and industry, using IBM's benchmark of $4.44 million globally or $10.22 million for U.S. organizations as a starting reference point, adjusted upward for organizations carrying the specific risk factors documented in Section 5 — ungoverned staging environments, long-lived unrevoked credentials, and shadow AI usage.

7.2 Illustrative scenario: a mid-size SaaS company

To make the model concrete, consider an illustrative mid-size software company spending roughly $600,000 per month on cloud infrastructure and $80,000 per month on AI/model usage — a scale representative of a well-funded growth-stage company or a smaller enterprise business unit. The figures below apply the benchmarks above at their midpoints; an organization should replace them with its own numbers wherever it has better data.

Category	Monthly spend	Applied waste/risk rate	Estimated annual impact
Cloud infrastructure (IaaS/PaaS)	$600,000	29% estimated waste (Flexera 2026 midpoint)	~$2,088,000 in avoidable annual spend
AI / model usage	$80,000	20% illustrative ungoverned-usage share*	~$192,000 in avoidable annual spend

Category	Monthly spend	Applied waste/risk rate	Estimated annual impact
Breach-risk exposure (expected value)	n/a	Illustrative 8% annual probability × $4.44M avg. breach cost	~$355,000 in annualized expected risk

**AI waste share is illustrative, not a published industry benchmark; substitute an organization's own usage-review findings once a governance baseline (Section 8) is in place. Cloud waste rate: Flexera 2026 State of the Cloud Report. Breach cost: IBM Cost of a Data Breach Report 2025.*

Summed, this illustrative organization is carrying roughly $2.6 million a year in combined avoidable cloud and AI spend plus expected breach-risk exposure — against a starting cloud and AI budget of just over $8 million a year. That is not a claim that every organization at this spend level is losing exactly this amount; it is a demonstration of how quickly the benchmarks in this report compound into a figure large enough to justify a dedicated governance program, using nothing but publicly documented industry averages.

7.4 Two more reference points: startup and large enterprise

Because the model in Section 7.1 is meant to be applied at any scale, it is worth showing it applied at two other common reference points, so a reader can locate their own organization on the same spectrum rather than only against the mid-size example above.

Profile	Monthly cloud spend	Monthly AI spend	Estimated annual avoidable spend	Estimated annualized breach-risk exposure*
Early-stage startup	$25,000	$8,000	~$87,000 + ~$19,000 = ~$106,000	Lower absolute exposure, but often a larger share of total runway
Mid-size SaaS company (Section 7.2)	$600,000	$80,000	~$2,280,000	~$355,000
Large enterprise business unit	$5,000,000	$600,000	~$18,960,000	~$818,000 (using $10.22M U.S. average breach cost at a lower relative probability)

**Illustrative figures using the same methodology as Section 7.1–7.2; actual probability and cost figures vary substantially by industry, regulatory environment, and existing security maturity, and should be replaced with organization-specific estimates wherever available.*

The startup figure is the one most worth dwelling on, because it is the scenario most often assumed to be too small to matter. At $106,000 a year in illustrative avoidable spend and risk, this figure can represent a meaningful fraction of an early-stage company's total runway — the difference, in some cases, between an eighteen-month and a twenty-month runway. The evidence in Section 4.3 that the smallest, most resource-constrained organizations are, if anything, more exposed to a single catastrophic AI billing event — precisely because they are least likely to have a dedicated finance or platform function reviewing spend — makes this a governance problem worth addressing well before an organization reaches enterprise scale, not a luxury to defer until it does.

7.5 The compounding effect over time

Waste of this kind rarely appears as a step change. It accumulates the way the case studies in Sections 3.8 and 6.3 describe: one forgotten staging environment, one ungoverned API key, one loosened access control at a time, each individually small enough to escape notice. The practical consequence is that the cost of inaction grows every quarter a governance program is delayed, while the cost of remediation — examined in Section 8 — does not grow nearly as quickly, because the fixes are largely structural rather than proportional to the size of the mess that has accumulated. An organization that waits eighteen months to address this pays for eighteen months of compounding waste, and then still has to do the same remediation work it would have done at month one.

7.6 Total cost of ungoverned delivery

The single most important reframing this report offers finance and engineering leadership is this: cloud waste, AI waste, and security exposure are not three separate budget lines competing for three separate remediation projects. They are three financial expressions of one governance gap, and a program that closes the gap addresses all three simultaneously, because — as Section 6 demonstrated — they share the same root cause at the same points in the delivery chain. That is the single strongest argument for treating this as one initiative rather than three, and it is the argument the remainder of this report is built to support.

7.7 A five-minute self-assessment

Before moving to the framework in Section 8, it is worth running a rough, honest self-assessment rather than waiting for a formal audit to establish a starting baseline. Score one point for each statement below that is true today; the scoring guide beneath the table gives a rough read on where an organization currently sits against the maturity model developed fully in Section 8.6.

- We could not produce a complete list of every cloud account and AI provider account in active use within one business day.
- At least one non-production environment currently running has no identifiable owner who can confirm it is still needed.
- At least one AI API key in active use has no configured spending cap.
- We have not run a full-history secrets scan across our internal repositories in the past six months.
- Fewer than half of our cloud resources carry an owner tag, a cost-center tag, and an expiration date.
- Our most recent cost review was triggered by a surprising invoice rather than a scheduled process.

Score (statements marked true)	Approximate maturity stage (Section 8.6)	Suggested starting point
5–6	Stage 1: Ad hoc	Begin the Days 1–30 visibility phase in Section 9.1 immediately
3–4	Stage 2: Reactive	Prioritize closing the specific gaps identified, then build standing automation

Score (statements marked true)	Approximate maturity stage (Section 8.6)	Suggested starting point
1–2	Stage 3: Proactive	Focus on chargeback, hard AI spend ceilings, and CI/CD-integrated scanning
0	Stage 4–5: Governed / Optimized	Focus on the sustaining practices in Section 9.5 rather than new construction

7.8 A realistic return-on-investment timeline

Organizations building a business case for the program described in Section 9 reasonably want to know not just the size of the opportunity but when it materializes. Based on the sequencing of the 90-day roadmap, savings tend to arrive in three distinct waves rather than all at once, and setting that expectation up front avoids the common disappointment of a sponsor expecting month-one results from a program that is, by design, spending its first month purely on visibility.

Timeframe	What materializes	Typical share of total identified savings
Days 1–30	Visibility only — no savings realized yet, but the full scope of the opportunity is quantified	0%
Days 31–60	Quick wins: decommissioned environments, revoked unused keys, closed critical exposures	~40–50%
Days 61–90	Rightsizing and structural fixes that require more coordination to implement safely	~30–40%
Beyond day 90	Compounding savings from standing automation preventing new waste from accumulating	Ongoing, growing

This pattern — a slow first month followed by a sharp acceleration — is also why the visibility phase should never be compressed or skipped under pressure to show early results: the quick wins in month two depend entirely on knowing, with confidence, what is safe to turn off, which is precisely what month one's inventory establishes.

Section A: An Industry Lens

The velocity-governance gap documented in this report is universal, but the sharpest edges of its cost and risk differ meaningfully by industry.

Every pattern documented in Sections 3 through 5 shows up across every industry this report's research touched. What differs by sector is which consequence bites first and hardest — a distinction worth making explicit, because it changes where a given organization should start.

A.1 SaaS and software companies

Software companies typically have the most cloud-native infrastructure and, correspondingly, the widest surface area for the waste patterns documented in Section 3 — more environments, more microservices, more Kubernetes clusters, more independent teams each provisioning their own infrastructure. They are also, per Section 4, the heaviest and earliest adopters of AI coding tools and agentic workflows, which makes them disproportionately exposed to the runaway-usage incidents documented in Section 4.3. The practical starting point for most software companies is the cloud and Kubernetes waste documented in Sections 3.2 through 3.4, simply because the absolute dollar exposure there tends to be largest relative to their overall spend.

A.2 Financial services and fintech

Financial services organizations operate under some of the strictest regulatory scrutiny of any sector, which means the security findings in Section 5 — particularly around IAM discipline, secrets management, and breach cost — carry a regulatory dimension beyond the direct financial cost. IBM's data places the financial sector among the highest average breach costs of any industry studied, and the delivery pressure documented in Section 1 is often at its most acute here, where competitive pressure to ship new products collides directly with compliance requirements that were, in many organizations, designed for a much slower release cadence. Financial services organizations should weight Section 5's IAM lifecycle and secrets governance recommendations particularly heavily, alongside the compliance-automation practice noted in Section 10.1.

A.3 Healthcare

Healthcare organizations carry the highest average data breach cost of any industry IBM tracks, a distinction the report attributes to the combination of extensive patient records, strict compliance requirements, and the high resale value of protected health information. For healthcare organizations, the staging-environment risk documented in Section 5.5 — a forgotten environment seeded with realistic, and in this sector often genuinely regulated, patient data — deserves to be treated as a top-tier priority rather than a background cleanup task, because the cost asymmetry between a well-governed environment and a forgotten one is larger here than in almost any other sector this report examined.

A.4 Public sector and regulated industrial environments

Public sector organizations and regulated industrial operators — the kind of environment covered by OT and ICS security disciplines — face a version of the delivery chain in Section 2 that extends beyond conventional cloud and AI infrastructure into operational technology that was never designed with the same security assumptions as a modern cloud environment. The core lesson of this report still applies directly: security postponed until a final audit, rather than built into delivery as a continuous process, is exactly the pattern that has driven binding federal directives on cloud misconfiguration of the kind referenced in Section 5.4. For these organizations, the security-by-default practices in Section 8.4 are less a best practice than an increasingly explicit regulatory expectation.

A.5 The common thread

Regardless of sector, the diagnosis in this report and the framework in Section 8 apply without modification — what changes is only the relative weighting of where an organization should start. A software company under cost pressure should likely start with Section 3's cloud waste findings. A financial services or healthcare organization under regulatory and breach-cost pressure should likely start with Section 5's security findings. Every organization, regardless of sector, is exposed to the AI governance gap documented in Section 4, simply because it is new enough that almost no industry has yet built mature practice around it.

A.6 Retail, media, and consumer platforms

Consumer-facing retail, media, and platform businesses tend to combine two of this report's risk factors at once: highly variable, seasonal demand that makes the “size up to be safe” instinct documented in Section 3.9 especially strong — nobody wants to be the engineer whose undersized infrastructure caused an outage during a peak sales event — and large volumes of customer data that make the staging-environment risk documented in Section 5.5 particularly costly if realized. These organizations are also, per IBM's data, among the sectors that bucked the overall 2025 trend toward declining breach costs, a pattern consistent with the operational disruption a breach causes to a business whose revenue depends on continuous customer-facing availability. For this sector, the rightsizing and autoscaling practices in Section 8.2, paired closely with the environment-parity recommendation in Section 8.4, are typically the highest-value starting point.

Section E: The Regulatory Backdrop

Governance is increasingly not just a financial and security discipline but a compliance obligation, and the direction of travel in every major regulatory environment points the same way: toward continuous accountability, not point-in-time attestation.

This report has deliberately kept its core argument grounded in financial and security evidence rather than legal advice, and nothing in this section should be read as a substitute for an organization's own legal or compliance counsel. It is included because the same governance gap documented throughout this report — unowned resources, undocumented access, and security treated as a final audit rather than a continuous process — increasingly intersects with binding regulatory obligation, not merely good practice.

E.1 Data protection regimes

Data protection frameworks in force across the European Union, the United Kingdom, and an increasing number of U.S. states share a common structural expectation relevant to this report: that an organization can demonstrate, not merely assert, where personal data resides, who can access it, and how long it is retained. A forgotten staging environment seeded with a production data snapshot, of the kind documented in Sections 3.8 and 5.5, is not only a cost and security problem — it is, in a large share of these frameworks, a data inventory an organization is legally obligated to know about and cannot currently produce on demand.

E.2 Sector-specific regimes

Healthcare, financial services, and payment-card handling each carry their own regulatory or contractual security frameworks — in the U.S. context, obligations tied to protected health information and to payment-card industry data security standards among them — that mandate specific, auditable controls around access management, encryption, and breach notification. Section A.2 and A.3 already noted that these are the sectors carrying the highest average breach costs; the regulatory dimension compounds the financial one, since a breach in a regulated sector typically triggers mandatory notification, potential fines, and a compliance review layered on top of the direct incident response cost documented in Section 5.7.

E.3 The emerging AI-specific regulatory layer

A newer and still-developing layer of obligation is emerging specifically around AI systems — covering areas including risk classification of AI use cases, transparency requirements, and, in several jurisdictions, specific obligations around high-risk automated decision-making. Organizations that cannot currently answer basic questions raised throughout Section 4 — which AI systems are in use, what data they process, who is accountable for their behavior — are not well positioned to meet obligations that are, across multiple jurisdictions, converging on exactly those questions as a starting compliance baseline. The AI provider inventory recommended in Section 8.3 is, in this light, not only a cost-governance practice but an increasingly load-bearing compliance one.

E.4 Why continuous governance is the more defensible position

Across every regime referenced above, the pattern is the same one this report has argued for on financial and security grounds throughout: a point-in-time audit, however thorough, demonstrates compliance on the day it was conducted and says nothing verifiable about the months that follow. Regulators, in the frameworks with the most developed enforcement track records, have increasingly signaled that they expect organizations to demonstrate ongoing, systematic control — the continuous inventory, ownership, and access review described throughout Section 8 — rather than a annual snapshot. An organization that has already built the governance practices this report recommends for cost and security reasons is, as a direct consequence and not a coincidence, substantially better positioned to meet this regulatory expectation than one that has not.

E.5 Contractual obligations, not only statutory ones

Alongside statutory and regulatory obligations, most organizations of any meaningful size now carry contractual security commitments to their own customers — data processing agreements, security addenda, and increasingly specific technical requirements written into enterprise sales contracts. These commitments are, in practice, enforced more immediately than most regulatory frameworks: a customer's security team conducting a vendor review will ask directly about the practices documented throughout Section 8 — access control, secrets management, environment governance — and an organization unable to answer those questions with confidence risks the deal itself, not only a future regulatory finding. Compliance Automation, referenced in Section 10.1, exists specifically to make answering these recurring questions a matter of pulling current evidence rather than reconstructing it under deadline pressure each time a prospective customer asks.

8. A Framework for Closing the Gap

The fix is not more approval gates. It is making ownership, cost visibility, expiration, and secure defaults automatic properties of every resource created — so speed and governance stop being a trade-off.

8.1 Four principles of continuous technology governance

Every specific practice recommended in this section is an implementation of one of four principles. Organizations that internalize the principles can adapt the specific practices to their own tooling; organizations that only adopt the practices without the principles tend to find the practices decay within a year as tooling and teams change.

11. Attach accountability at creation, not after the fact. An owner, a cost center, and an expected lifetime should be required inputs to create a resource, an API key, or an environment — not optional metadata added later by whoever happens to notice it is missing.
12. Make the governed path the fast path. Any control that is slower than the ungoverned alternative will be routed around under deadline pressure, every time. Self-service provisioning through a governed template, pre-approved instance sizes, and pre-configured secrets injection should be faster for a developer to use than the manual alternative, not merely more compliant.
13. Default to expiration, not permanence. Every environment, credential, and access grant should be created with an assumed end date. Extending it should be a deliberate, low-friction action; letting it lapse should be the default outcome of doing nothing, reversing the current default in most organizations.
14. Give cost and risk visibility to the people who create it, not only to the people who review it later. A developer who can see, at the moment of provisioning, what an instance size or a model choice will cost per month is far more likely to make an efficient choice than one who only discovers the cost on a retrospective report they cannot act on.

8.2 Cloud cost governance practices

Mandatory tagging and ownership

Enforce, at the infrastructure-as-code or provisioning-API level, that every resource carries an owner, a team, a project, and a cost center before it can be created — not as a policy document, but as a technical gate that blocks creation without it. This single control is what makes every other practice in this section possible, because without it, waste identified later cannot be traced back to anyone who can act on it.

Default expiration for non-production infrastructure

Every staging, demo, and development environment should be provisioned with a default time-to-live — commonly 7 to 30 days depending on the use case — enforced automatically, with a simple, low-friction renewal mechanism for environments that are still genuinely needed. This directly targets the single most

common pattern documented in Section 3: temporary infrastructure that becomes permanent by default rather than by decision.

Automated rightsizing and idle-resource detection

Continuous, automated detection of idle compute, unattached storage, unused IP addresses, and unreferenced snapshots — paired with an automated or semi-automated cleanup workflow — converts the manual, occasional cost review documented in Section 3.9 into a standing, low-effort discipline. Given that idle and overprovisioned resources account for an estimated 60% of cloud waste industry-wide, this is consistently the highest-return-on-effort intervention available to most organizations.

Chargeback or showback to the team level

Whether or not an organization implements full financial chargeback, every team should be able to see its own infrastructure cost broken down by resource, in the same tools it already uses daily. The CNCF's finding that only around 14% of organizations run full Kubernetes chargeback is, per Section 3.4, a strong predictor of which organizations carry the highest overprovisioning — visibility and accountability at the team level is the mechanism that makes rightsizing self-sustaining rather than a one-time cleanup project.

A single, governed cloud account/project inventory

Maintain one authoritative, centrally visible inventory of every cloud account, project, and subscription in active use, with a defined process — not a policy memo — for provisioning a new one. This closes the ungoverned-account gap documented in Section 3.6, which is otherwise invisible by definition.

8.3 AI governance practices

Hard spending ceilings, not just alerts

Every AI API key and every AI platform account should carry a pre-configured, enforced spending ceiling at the individual, team, and organizational level, set before access is granted rather than after a surprising invoice arrives. As Section 4.3 documented, an alert that fires after money has already been spent is not a control — it is a delayed notification. The control is a cap that blocks the next request before it happens.

Per-key identity and attribution

Route AI usage through a gateway or proxy layer that ties every request back to an individual user, team, or automated workflow, rather than issuing shared keys with no per-request identity. This is the single control that would have prevented the “nobody knows who used it” pattern documented across nearly every incident in Section 4.3, and it is a prerequisite for any meaningful chargeback or anomaly detection.

Model routing and cost-aware defaults

Provide developers with a default routing layer that sends a task to the smallest, cheapest model capable of handling it, reserving frontier-model access for tasks that genuinely require it. This does not need to restrict what a team can ultimately use — it needs to change the default from “most powerful available” to

"appropriate for the task," which, per Section 4.4, is currently the rational individual choice only when a cost signal is visible.

Separate, tighter budgets for development and experimentation

Development and evaluation usage should run against a meaningfully smaller budget than production, on a separate key or project, so that iterative experimentation — which is supposed to involve running the same task many times — does not silently blend into and mask production cost.

Loop and anomaly detection at the request layer

Deploy runtime guardrails — rate-of-spend monitoring, recursion-depth limits, and automatic session termination on anomalous usage patterns — in the request path between an application or agent and the model provider, not only in a downstream dashboard. Every runaway-agent incident documented in Section 4.3 and 4.5 was, in the language of one of this report's sources, a story about observability versus enforcement: a dashboard can tell you a runaway agent cost $40,000 last night, but only an enforcement layer sitting in the request path can stop the next one before it happens.

A single inventory of AI providers, keys, and subscriptions

Maintain the same kind of authoritative inventory recommended for cloud accounts in Section 8.2, extended to cover every AI provider account, API key, and AI-enabled SaaS subscription in use — directly closing the duplicate-subscription and shadow-AI gaps documented in Section 4.7.

8.4 Security-by-default practices

Centralized secrets management as the path of least resistance

Provide developers with a secrets manager that is easier to use than hardcoding a credential — injected automatically into local development, CI/CD, and runtime environments — so that the secure choice and the fast choice are the same choice. Given that GitGuardian found 64% of leaked secrets from 2022 were still valid four years later, prevention at the point of creation is dramatically more effective than any downstream detection and remediation program, however good that program is.

Automated secret scanning in CI/CD, with pre-commit prevention

Layer automated scanning at the commit stage, the CI pipeline stage, and continuously against the full history of every repository, public and internal — recalling that internal repositories were found to be six times more likely than public ones to contain hardcoded secrets precisely because they are assumed to be safe.

Scoped, time-limited access by default

Default every new IAM role, service account, and database grant to the narrowest permission set that unblocks the task at hand, with a fast, low-friction path to request broader access when genuinely needed — rather than the current default in most organizations, which grants broad access up front because narrowing it later requires effort nobody is assigned to spend.

Environment parity for staging and production

Staging and demo environments should inherit the same patching cadence, the same access-control baseline, and the same monitoring coverage as production, provisioned from the same governed templates recommended in Section 8.2 — removing the "reduced scrutiny plus real data" combination documented in Section 5.5 as one of the most dangerous and most common patterns in this report.

Security integrated into delivery, not gating it at the end

Move security review earlier and make it continuous — automated checks that run on every commit and every environment change — rather than concentrated into a single pre-launch audit that, as Section 5.8 argued, cannot see the drift that happens after it concludes.

8.5 Organizational structures that make this stick

Practices alone decay without an organizational home responsible for maintaining them. Flexera's 2026 data shows the direction the industry is already moving: 71% of organizations now operate some form of Cloud Center of Excellence, and 63% have a dedicated FinOps team, both up substantially as organizations formalize the oversight this report recommends. The specific structure matters less than three properties it needs to have.

- Cross-functional authority: the group responsible for governance needs representation from engineering, security, and finance, because — as Section 6 demonstrated — the problem this report documents is one that no single function can see or fix on its own.
- A mandate to build tooling, not only write policy: a governance function that produces documents without also producing the templates, gateways, and automated scanners described in Sections 8.2 through 8.4 will not change developer behavior, because policy alone does not compete with deadline pressure.
- Explicit executive sponsorship: given the incentive misalignment documented in Section 6.2, a governance function needs authority that outranks the individual project deadlines it will sometimes need to push back against — without that authority, governance recommendations become optional guidance that is the first thing dropped under schedule pressure.

8.6 A maturity model

Stage	What it looks like	Typical cloud waste	Typical governance state
1. Ad hoc	No tagging standard, no expiration policy, shared AI keys with no caps	30%+ of cloud spend, AI spend unbounded	Discovered reactively, usually via a surprise invoice
2. Reactive	Periodic manual cost reviews after the fact; security reviewed only pre-launch	~25–30% of cloud spend	Cleanup happens but does not prevent recurrence
3. Proactive	Tagging enforced, some automated idle-resource cleanup, basic AI spend alerts	~18–25% of cloud spend	Governance exists but relies on manual follow-through

Stage	What it looks like	Typical cloud waste	Typical governance state
4. Governed	Default expiration, chargeback to teams, hard AI spend ceilings, scoped IAM by default	~10–15% of cloud spend	Governance is largely automatic and self-sustaining
5. Optimized	Continuous automated rightsizing, model routing by task, security integrated into every pipeline stage	Approaching a practical efficiency floor	Governance is invisible — the fast path and the governed path are the same path

Waste-rate bands are illustrative, calibrated against the Flexera 2026 industry-wide average of 29% and the range of outcomes reported by organizations with mature FinOps practice; an organization's actual position should be established through the assessment recommended in Section 9.

Most organizations examined in the preparation of this report sit at stage 1 or 2 for AI governance specifically, even where they have made real progress on cloud governance over the past several years — a direct consequence of AI spend being new enough that the maturity most organizations built for cloud has not yet been extended to cover it. That gap, and how to close it within one financial quarter, is the subject of Section 9.

8.7 Build versus buy

Every practice in this section can, in principle, be built internally: a homegrown tagging enforcement script, an internally maintained secrets scanner, a custom AI usage gateway. Several well-resourced technology organizations have done exactly this. For most organizations, however, the honest calculation favors buying at least the foundational layer — secrets scanning, cloud cost visibility, and AI spend gateways are now mature, competitively priced categories with dedicated vendors who have solved the hard edge cases already, and the internal engineering time required to reach the same maturity is usually better spent on the organization's actual product. The practices worth building in-house are the ones specific to an organization's own delivery process — the governed provisioning templates described in Section 8.1, the specific ownership and expiration policy that fits how a particular organization's teams actually work — while the detection and enforcement layer underneath them is, for most organizations, a better buy than a build.

The decision is worth revisiting explicitly rather than defaulting either way. A useful rule of thumb: buy the layer that is undifferentiated across every company facing the same problem (scanning for a leaked AWS key looks the same at every company that uses AWS); build the layer that encodes an organization's own specific process and judgment calls (which categories of resource require which level of approval, and from whom).

8.8 Metrics that matter

A governance program that cannot demonstrate its own impact will not survive its first budget review under pressure. The metrics below are the ones that most directly connect the practices in this section back to the evidence in Sections 3 through 5, and are worth tracking from the first day of the 90-day roadmap in Section 9 rather than retrofitted later.

Metric	What it tracks	Where it connects in this report
Estimated cloud waste rate (% of spend)	Idle, orphaned, and overprovisioned resources as a share of total cloud spend	Section 3.1–column against the Flexera 29% industry benchmark
Share of resources with a named owner and expiration date	Coverage of the accountability-at-creation principle in Section 8.1	Section 3.6's ownerless-resource and no-expiration-date findings
Share of AI API keys with a configured hard spending ceiling	Coverage of the single highest-leverage AI control in this report	Section 4.3's incident data
Median time from secret exposure to revocation	How quickly a leaked credential is neutralized once created	Section 5.2's finding that 64% of 2022-era leaked secrets remained valid in 2026
Kubernetes chargeback / cost-attribution coverage	Share of cluster spend attributable to a specific owning team	Section 3.4's finding that only ~14% of organizations run full chargeback
Non-production infrastructure as a share of total cloud spend	Whether staging/dev environments are being actively managed or left to run indefinitely	Section 3.2's finding that ~44% of spend covers non-production resources

8.9 Vendor and third-party AI risk

The governance practices described throughout this section focus on an organization's own directly provisioned infrastructure and AI access, but a growing share of AI exposure now arrives indirectly, through third-party software that embeds AI features an organization did not directly evaluate or configure. A SaaS vendor that adds an AI-powered feature to its product may be sending an organization's data to a model provider the organization has never reviewed, under data-handling terms nobody on the security team has read. Extending the AI provider inventory recommended in Section 8.3 to explicitly include AI capabilities embedded in third-party vendor products — not only an organization's own direct API usage — closes a blind spot that is easy to miss precisely because no internal team consciously decided to create the exposure.

- Add AI-specific questions to vendor security review and procurement processes: does this product use AI features, what data is sent to which model provider, and can that be disabled or scoped if the answer is not acceptable.
- Treat a vendor's AI feature launch as a trigger for re-review, not a one-time approval — vendors add AI capabilities to existing, already-approved products on their own schedule, frequently without the kind of notice that would trigger a security team's attention.

8.10 Incident response readiness

Every practice in this section is aimed at prevention, and prevention will never be perfect. A governance program should pair its preventive controls with a rehearsed response plan for the two incident types most directly documented in this report — a leaked credential and a runaway AI spending event — both outlined in template form in Appendix F. The organizations in Section D that handled their incidents best were not the ones that avoided every mistake; they were the ones that had already decided, before the incident happened,

exactly who would revoke a credential, who would suspend a key, and who would need to be notified — removing the costly delay of figuring that out live, under pressure, which Section 5.7's data shows is precisely where breach cost and breach duration both compound fastest.

8.11 A note on legacy and brownfield environments

Everything in this section is easiest to apply to new resources created after a governance program begins — enforcing tagging and expiration at the point of creation is a straightforward technical control. It is harder, and unavoidably slower, to apply retroactively to an existing estate that has accumulated years of untagged, unowned, and undocumented infrastructure before the program started. Organizations in this position should resist the instinct to solve the entire backlog before enforcing new defaults going forward — the two should run in parallel, not sequentially. Stop the bleeding first (new-resource enforcement, which prevents the backlog from growing further) while working through the existing backlog on a slower, prioritized timeline driven by the risk and cost data the Section 9.1 inventory surfaces, rather than waiting for a complete retroactive cleanup before any forward-looking control is put in place.

9. A 90-Day Roadmap

Closing the governance gap does not require a multi-year transformation program before it produces results. The sequence below is built to surface real savings and real risk reduction inside a single fiscal quarter.

9.1 Days 1–30: Visibility

The first thirty days should produce no architectural change at all — only an honest, complete inventory. Every practice in Section 8 depends on knowing what currently exists, and in most organizations, nobody currently does.

15. Inventory every cloud account, project, and subscription across every provider in active use, including ones created outside central IT for a one-off project or trial.
16. Inventory every AI provider account, API key, and AI-enabled SaaS subscription, and identify which keys currently have no spending cap configured — in most organizations examined for this report, this is the majority of keys in existence.
17. Run automated idle-resource and orphaned-storage detection across every cloud account to produce a first-pass estimate of the waste documented in Section 3, without deleting anything yet.
18. Run a full-history secrets scan across every code repository, both public and internal, and every connected collaboration tool, to establish a baseline count of currently exposed and potentially still-valid credentials.
19. Identify every non-production environment currently running, and for each one, determine whether it has an identifiable owner who can confirm it is still needed.

> **End-of-month-1 deliverable**
>
> A single, shared inventory covering cloud resources, AI access, exposed secrets, and non-production environments — the first time most organizations will have seen all four in one place. This inventory alone typically surfaces enough quick wins to fund the next sixty days.

9.2 Days 31–60: Control

With visibility established, the second thirty days focus on closing the highest-severity gaps identified in month one — prioritized by a combination of dollar impact and risk, not attempted all at once.

20. Set hard spending ceilings on every AI API key and account identified as uncapped in month one, starting with the highest-usage keys first.
21. Revoke or rotate every exposed secret confirmed still valid from the month-one scan, prioritizing credentials with the broadest permissions and the longest exposure window.

22. Decommission or formally re-approve every non-production environment identified as unowned or unused in month one — the direct, actionable output of the case pattern documented in Sections 3.8 and 6.3.
23. Apply mandatory tagging and default expiration to the provisioning workflow for all new cloud resources, so that month one's inventory problem does not simply regrow.
24. Tighten the highest-risk excessive-permission grants identified during the inventory, starting with public storage, open database access, and overly broad service-account roles.

End-of-month-2 deliverable

A measurable reduction in both monthly cloud/AI spend and the count of unrevoked, exploitable credentials — typically the point at which the program's savings begin to be visible on the monthly invoice.

9.3 Days 61–90: Automation and culture

The final thirty days convert the manual cleanup of the first sixty into a standing, self-sustaining discipline — the difference between a one-time cost-cutting exercise and the continuous governance model described in Section 8.

25. Deploy continuous, automated idle-resource detection and rightsizing recommendations, rather than the one-time scan run in month one.
26. Stand up per-team cost visibility (chargeback or showback) for both cloud and AI spend, so that ongoing accountability does not depend on a periodic manual review.
27. Deploy pre-commit and CI/CD secrets scanning as a standing control, not a one-time audit, closing the loop on the leak vector documented in Section 5.2.
28. Establish or formalize the cross-functional governance structure recommended in Section 8.5, with clear, named ownership of the practices established over the previous sixty days.
29. Publish the governed, self-service provisioning templates — for cloud environments and for AI access alike — that make the governed path the fastest path, per the second principle in Section 8.1.

End-of-month-3 deliverable

A governance program that runs without requiring a large, dedicated team to keep it running — the point at which the organization moves from Section 8.6's Stage 2 or 3 toward Stage 4, with the majority of the remaining work being continuous tuning rather than new construction.

9.4 What to expect, and what not to

A 90-day program of this kind will not eliminate cloud, AI, or security waste entirely — Section 8.6's maturity model treats Stage 5 as an approach to an efficiency floor, not a final state anyone reaches and stops working at. What a well-run 90 days reliably delivers is the transition from Section 6's ad hoc, reactive condition to a governed one: a known inventory, closed high-severity exposures, and standing automation that prevents the same categories of waste and risk from silently regrowing the way they did the first time. That transition is also, not coincidentally, the point at which an organization is positioned to make an informed decision about

whether to continue building this capability internally or to bring in a specialist partner for the categories of work — security testing, cloud architecture, AI infrastructure — that benefit most from dedicated, outside expertise. Section 10 addresses that option directly.

9.5 Beyond 90 days: sustaining the gains

The most common failure mode after a successful 90-day program is not a technical one — it is organizational drift back toward the ad hoc state the program was built to fix, as the initial focus and executive attention that funded the sprint moves on to the next priority. Three practices, established during the roadmap itself, are what determine whether the gains documented above are permanent or temporary.

30. Quarterly governance reviews, calendared in advance rather than triggered reactively, that re-run the Section 9.1 inventory exercise at a fraction of its original scale, since the standing automation from Section 9.3 should make each subsequent review far faster than the first.
31. A named, accountable owner — not a committee — for each of the standing controls established in the roadmap, so that when a tool needs to be upgraded, a policy needs to be updated, or a new cloud provider is added, there is a specific person whose job includes noticing.
32. Onboarding every new hire, and every newly acquired team following a merger or acquisition, directly into the governed provisioning path from day one, rather than treating governance as something applied retroactively to teams after they have already built their own habits.

Organizations that treat these three practices as seriously as the original 90-day build typically find that the maturity level reached at day 90 is durable. Organizations that treat the 90-day program as a one-time cleanup project typically find themselves, twelve to eighteen months later, looking at a smaller but familiar version of the same finance question this report opened with.

9.6 Who owns what: a starting RACI

A recurring reason governance programs stall is ambiguity about who is actually responsible for a given action versus who merely needs to be informed of it. The simplified responsibility assignment below is offered as a starting point for the roadmap in this section, to be adapted to an organization's actual team structure rather than adopted verbatim.

Activity	Responsible	Accountable	Consulted / Informed
Cloud/AI account and key inventory (9.1)	Platform or DevOps team	Head of Platform / CTO	Finance, security
Setting AI spend ceilings (9.2)	AI platform owner	CTO or Head of Engineering	Finance, individual team leads
Secret revocation and rotation (9.2)	Security engineering	CISO / security lead	Affected engineering teams
Environment decommissioning (9.2)	Owning engineering team	Engineering manager	Platform team, finance

Activity	Responsible	Accountable	Consulted / Informed
Standing automation and tooling (9.3)	Platform / DevOps team	Head of Platform	Security, finance, FinOps
Governance structure and cadence (9.3, 9.5)	FinOps / Cloud Center of Excellence	Executive sponsor	All engineering leadership

9.7 A note on tooling procurement timing

A common, avoidable delay in the roadmap above is treating tool procurement as a prerequisite for starting, rather than something that runs in parallel with the earliest manual steps. The Days 1–30 visibility phase in Section 9.1 does not require a fully selected and contracted toolset — native cloud-provider cost tools, open-source secrets scanners, and manual account review are sufficient to produce the first inventory, and starting with these while a more capable platform, of the kind surveyed in Section C, is evaluated in parallel avoids the common failure mode of a governance program stalling for a full procurement cycle before any visibility work begins at all. Procurement, where it is needed, should be scoped and initiated during month one specifically so a selected tool is ready to deploy at the start of month three's automation phase, not started only once month one's findings create urgency.

Section B: Common Objections, Answered

Every recommendation in this report will meet some version of the same handful of objections inside a real organization. They deserve honest answers rather than dismissal.

B.1 “This will slow us down.”

This is the most common objection, and it is worth taking seriously rather than waving away, because a poorly designed governance program genuinely can slow delivery down — that is precisely what happens when governance is implemented as a manual approval gate layered on top of an otherwise unchanged workflow. The framework in Section 8 is deliberately built around a different model: governance implemented as defaults built into the provisioning path itself, so that using the governed path is faster than the ungoverned alternative, not merely more compliant with it. A self-service template that provisions a correctly tagged, appropriately sized, automatically expiring environment in ninety seconds is faster for a developer than manually configuring an equivalent environment from scratch — which means the objection is answered not by arguing the trade-off is worth it, but by removing the trade-off.

B.2 “We are too small for this to matter yet.”

Section 7.4 addressed this directly with numbers: illustrative modeling suggests even an early-stage startup can be carrying a six-figure annual exposure across avoidable spend and breach risk, and the evidence in Section 4.3 that the largest individual runaway-AI-cost incidents have hit organizations of every size — including, by their own account, small agencies and individual developers — argues against waiting for scale to justify basic controls. The specific practices worth adopting early are cheap and do not require a dedicated team: a spending cap on every API key, a tagging convention enforced from day one, and a default expiration policy for non-production environments are each achievable in under a day at a ten-person company, and dramatically cheaper to establish before an organization's infrastructure footprint grows than to retrofit afterward.

B.3 “Our engineers are too senior to make these mistakes.”

The evidence throughout this report does not support the premise that this is a seniority or skill problem. The incidents documented in Section 4.3 include experienced engineers at well-resourced, technically sophisticated organizations; the secrets-sprawl data in Section 5.2 shows the pattern holding “across all experience levels and organization sizes,” in the words of one of this report's sources. The mechanism, described in Section 6.2, is an incentive and structural gap, not a competence gap — a senior engineer under the same deadline pressure and the same absence of visible cost or security feedback will make the same reasonable-in-the-moment shortcut as a junior one. Treating this as a training problem, rather than a structural one, is itself one of the more common reasons governance initiatives fail to produce lasting change.

B.4 “We already have a cost review process.”

A periodic, manual cost review — quarterly, or after a budget overrun triggers one — is real progress over having no review at all, and it is also, on its own, insufficient for the reasons documented in Section 3.7: waste accumulates in small increments between review cycles, and by the time a periodic review catches it, the specific context needed to safely and confidently remove it is often already lost. A periodic review is the Section 8.6 Stage 2 (“reactive”) condition; the goal of this report's framework is to move an organization to Stage 4 or 5, where waste is caught continuously and automatically rather than rediscovered from scratch at each review cycle.

B.5 “Security already reviews everything before launch.”

Section 5.8 addressed this directly: a pre-launch review is a snapshot, and every pattern documented in Section 5 describes drift that happens after that snapshot is taken — an environment left running past its useful life, a permission grant that outlives its original justification, a credential that was fine when issued and become a liability only once forgotten. A pre-launch review answers “was this secure on launch day” and cannot, by construction, answer “is this still secure eleven months later,” which is the question that actually determines most of the outcomes documented in this report.

B.6 “We don't have budget for new tooling.”

Section 7's financial modeling is offered specifically to reframe this objection: the evidence in this report suggests the cost of inaction — the accumulating waste and risk documented in Sections 3 through 5 — is, for most organizations at meaningful scale, substantially larger than the cost of the tooling and process changes recommended in Section 8. The 90-day roadmap in Section 9 is deliberately sequenced so that the visibility phase in the first thirty days, which requires minimal new tooling spend, typically surfaces enough immediate, actionable savings to fund the control and automation phases that follow — in practice, a well-run governance program is much more often self-funding within its first quarter than it is a net new cost the organization has to absorb up front.

B.7 “Our AI usage is too experimental to put guardrails on yet.”

This objection has real intuitive appeal — nobody wants to constrain genuine early-stage experimentation with process before it has found its shape. The distinction worth drawing, per Section 8.3's recommendation to give development and experimentation usage its own, separately tracked budget, is between guardrails that constrain what a team can try and guardrails that simply make what they are trying visible and bounded. A spending ceiling generous enough to permit ambitious experimentation, paired with per-key attribution so usage is traceable to a person, constrains nothing about the experimentation itself — it only removes the possibility that experimentation quietly becomes a six-figure surprise, which per Section 4.3's evidence is a real and recurring risk specifically in exactly this “we're still figuring it out” phase of AI adoption, not a risk that only appears once usage matures.

B.8 “We'll deal with this once we've raised our next round / hit our next milestone.”

Section 7.4's startup-scale illustration was included specifically to address this objection directly: deferring governance until after a funding milestone assumes the cost of deferring is small relative to the milestone, and the evidence in this report does not support that assumption. An organization's infrastructure footprint, AI usage, and accumulated technical and security debt all tend to grow between funding milestones, not shrink, which means the remediation task deferred until “after we raise” is reliably larger and more expensive by the time anyone returns to it — while the foundational practices in Section 8.1 and 8.2 cost measurably less to establish early, before scale multiplies the cleanup required. The right sequencing is not “governance after growth” but “enough governance, established early and cheaply, that growth does not have to be followed by an expensive cleanup at all.”

Section F: Culture and Change Management

The framework in Section 8 is a technical and organizational blueprint. Whether it survives contact with a real organization depends on factors this report would be incomplete without addressing directly.

F.1 Why technically sound programs still fail

It is common for a governance program to be technically well-designed — the right tooling, the right defaults, the right roadmap — and still fail to take root, for reasons that have little to do with the technology itself. The most common failure pattern observed across organizations examined for this report is a program introduced top-down, announced in a single all-hands meeting, with no ongoing mechanism for engineers to ask questions, push back on specific defaults that do not fit their workflow, or flag genuine edge cases the policy did not anticipate. A governance program that engineers experience as something imposed on them, rather than something built with their input, tends to generate exactly the kind of quiet workaround behavior — routing around a control rather than engaging with it — that defeats the program's purpose without ever triggering a visible conflict anyone has to resolve.

F.2 Sequencing: start where the pain is already felt

The most durable governance programs observed in the preparation of this report did not start with the category of waste or risk that was largest in dollar terms. They started with the category a specific, influential team was already frustrated by — a platform team tired of manually hunting for orphaned resources, a security team tired of chasing down leaked credentials by hand — and used that team's early, voluntary success as the proof point that made the next team's adoption easier. This sequencing matters because a governance program's credibility compounds the same way the waste it addresses does: an early, visible win, championed by the team that experienced it firsthand, does more to drive adoption across a wider organization than any executive mandate delivered without that grounding.

F.3 Making the case in language each audience already speaks

Section 7's financial modeling, Section 5's breach-cost data, and Section 8's practical framework are, deliberately, three different arguments for the same program, aimed at three different audiences who evaluate proposals in different terms. A CFO is persuaded by the dollar figures in Section 7. A CISO is persuaded by the breach-cost and exposure-window data in Section 5.7. An engineering leader is persuaded by the observation, made throughout Section 8, that the recommended defaults make delivery faster, not slower, once they are in place. A program pitched using only one of these three arguments will struggle to build the cross-functional support that Section 8.5 identifies as a structural requirement; a program that can fluently make all three cases, tailored to whoever is in the room, moves markedly faster.

F.4 Handling resistance without becoming the friction this report warns against

Some resistance to a new governance control is a legitimate signal that the control, as designed, genuinely does not fit a real workflow — and should be treated as design feedback rather than something to be overridden. Some resistance is simply discomfort with any change to an established habit, and should be worked through with clear communication about why the change matters, using the evidence in this report directly rather than an unsupported directive. The practical test that distinguishes the two, worth applying deliberately: does the specific objection point to a real task the new control makes meaningfully harder to accomplish, or does it only point to unfamiliarity with a new process? The first deserves a design change. The second deserves patience, clear communication, and time — not a reversal of the policy.

F.5 Celebrating the boring work

Section 6.4 named the underlying cultural gap directly: organizations celebrate shipping and rarely celebrate cleanup, ownership, or decommissioning, even though the evidence throughout this report suggests the latter category is just as consequential to the organization's financial and security health as the former. Practical, low-cost interventions that close this gap over time include naming waste-reduction and risk-reduction wins in the same forums — sprint reviews, all-hands updates, performance conversations — that already celebrate feature launches; tracking and publishing the metrics recommended in Section 8.8 with the same visibility given to product metrics; and explicitly crediting the engineers and teams whose environment decommissioning, credential rotation, or rightsizing work produced a measurable result. None of this requires a formal recognition program. It requires treating the work as worth mentioning out loud, consistently, until it stops feeling like an exception to what the organization values and starts feeling like an ordinary part of doing the job well.

Section C: The Tooling Landscape, at a Category Level

Organizations rarely fail to close the governance gap because no tool exists. They fail because nobody has mapped which category of tool solves which specific finding in this report.

This section is deliberately vendor-neutral: it describes categories of tooling relevant to the framework in Section 8, not specific products, so that an organization can use it to scope a build-versus-buy decision (Section 8.7) without this report reading as an advertisement for any single platform. Bithost's own services, described in Section 10, span several of these categories; the mapping below applies equally whether an organization builds a capability internally, buys a point solution, or engages a partner.

C.1 Cloud cost visibility and FinOps platforms

This category ingests billing and usage data across cloud providers and surfaces waste, anomalies, and rightsizing opportunities — directly addressing the idle-compute, orphaned-storage, and Kubernetes-overprovisioning findings in Section 3. Maturity varies significantly: entry-level tools provide visibility and reporting only, while more capable platforms provide automated or semi-automated remediation — rightsizing recommendations that can be applied with one click, or fully autonomous continuous optimization. Section 8.7's build-versus-buy framing applies directly here: the underlying billing-data ingestion is highly commoditized across vendors, which is one of the stronger arguments for buying this layer rather than building it.

C.2 Kubernetes-specific cost and rightsizing tools

A distinct sub-category has emerged specifically for the container-orchestration waste documented in Section 3.4, because general-purpose cloud cost tools frequently lack the pod-, namespace-, and label-level granularity needed to attribute Kubernetes spend accurately. Tools in this category range from open-source, CNCF-backed cost-allocation projects that provide visibility only, to commercial platforms that continuously and autonomously rightsize resource requests against live usage — the latter category is the one most directly responsible for the 30–75% savings figures reported by mature adopters cited throughout Section 3.4.

C.3 AI spend gateways and usage governance

This is the newest and least mature category referenced in this report, and it maps directly onto the AI governance gap documented throughout Section 4. Tools here sit as a proxy or gateway between an application and an AI model provider, and their value lies specifically in enforcement rather than observation: a genuinely useful tool in this category blocks a request that would exceed a budget before the request is sent, rather than only reporting after the fact that a budget was exceeded — the precise distinction Section 4.3 identified as the difference between an alert and a control. Capability varies from simple per-key spend caps up to full per-user attribution, model-routing logic, and loop/anomaly detection of the kind recommended in Section 8.3.

C.4 Secrets management and detection

This category splits into two complementary halves that address different points in the secrets lifecycle documented in Section 5.2. Secrets management platforms provide the centralized vault and dynamic-credential-issuance capability that makes the secure default the fast default, per Section 8.4's first recommendation. Secrets detection and scanning platforms find credentials that were hardcoded despite that default — at commit time, in CI/CD, and retroactively across full repository history — closing the gap for the credentials that inevitably slip through prevention alone. Organizations get the most value from running both halves together, since detection without a good remediation and prevention story simply produces a growing backlog of findings nobody has bandwidth to act on.

C.5 Cloud security posture management (CSPM) and vulnerability scanning

This category continuously scans cloud environments for the misconfiguration patterns documented in Section 5.4 — public storage, overly permissive IAM roles, open ports, and missing patches — and is the most direct technical answer to the “security as a continuous process, not a final audit” argument made in Section 5.8. The strongest implementations integrate directly into the CI/CD pipeline discussed in Section 5.9, flagging a misconfiguration before it is ever deployed rather than only after a scan discovers it running in production.

C.6 Governed provisioning and infrastructure-as-code templates

This is less a single product category than a practice, usually built on top of existing infrastructure-as-code tooling, that encodes the mandatory tagging, default expiration, and appropriate-sizing defaults recommended throughout Section 8.1 and 8.2 directly into the templates developers use to provision resources. This is consistently the category organizations are best served building internally rather than buying, per Section 8.7, because it is the layer that most needs to reflect an organization's own specific process rather than a generic industry pattern.

C.7 A practical selection principle

Across every category above, the same evaluation question recurs, and is worth applying deliberately rather than defaulting to whichever tool a vendor's sales process makes easiest to buy: does this tool enforce a control before the waste or exposure happens, or does it only report on the waste or exposure after it has already occurred? Both observability and enforcement have a place — observability is what makes the Days 1–30 visibility phase of the roadmap in Section 9.1 possible at all — but the tools that ultimately close the gap documented throughout this report are, without exception, the ones capable of enforcement, not only reporting.

C.8 Evaluating a vendor against this report's framework

For organizations evaluating specific products in any category above, the questions below — drawn directly from the findings throughout this report — tend to separate genuinely capable tooling from dashboards that describe a problem without helping to fix it.

- Does the tool block or prevent the waste/exposure before it happens, or only report on it afterward? (Sections 4.3, C.7)
- Can findings be attributed to a specific owner automatically, or does someone still need to manually investigate who is responsible? (Section 3.6)
- Does the tool integrate into the provisioning workflow developers already use, or does it require a separate, additional step? (Section 8.1's second principle)
- For AI-specific tools: does it provide per-identity attribution and pre-request enforcement, or only aggregate, after-the-fact reporting? (Section 4.8)
- How quickly, in practice, does the vendor's own customer base see measurable results — and is that evidence based on customers with a comparable starting maturity level to the organization's own (Section 8.6)?

10. How Bithost Can Help

Everything in this section is optional context for organizations that want an experienced partner to help close the gap described in Sections 1 through 9. Nothing in the diagnosis or the framework depends on it.

The preceding sections deliberately avoided naming a vendor, because the governance gap they describe is real regardless of who helps close it, and the framework in Section 8 is designed to be actionable with internal resources alone. This final section is for organizations that have read the diagnosis, recognize their own environment in it, and want to move faster than a purely internal build allows — or that lack a specific capability, such as offensive security testing or sovereign AI infrastructure, that is more efficient to bring in than to build from scratch.

Bithost is a security, cloud, and AI engineering firm operating from India, built specifically around the problem this report describes: technology delivery that has outrun technology governance. Its service lines map directly onto the framework in Section 8, and the table below lays out that mapping explicitly, so an organization can see exactly which part of its own gap each service is built to close.

10.1 Where Bithost's services map to this report's framework

This report's finding	Relevant Bithost service	What it addresses
Section 3: cloud cost waste, idle and orphaned resources	Cloud Audit and Cloud Cost Optimization	Finds what is running, why, and what it costs — then eliminates the waste and rightsizes what remains, closing the visibility gap documented in Section 3.7
Section 3–9: DevOps discipline, environment sprawl, deployment reliability	DevOps & SRE Managed Services	Builds and operates the governed, self-service provisioning patterns recommended in Section 8.2, so deployments do not depend on any one engineer remembering the right steps
Section 2, 8.2: architecture that bakes in governance from the start	Infrastructure Architecture	Designs cloud environments with ownership, tagging, and expiration built in at the architecture stage, rather than retrofitted after the waste has already accumulated
Section 4: unrestricted AI API keys, model sprawl, agentic cost growth	AI Integration	Integrates AI into specific workflows with usage attribution and cost awareness designed in, rather than a bare API key handed to a team with no guardrails
Section 4.6–4.7: data exposure risk from public model APIs, GPU cost control	Sovereign AI	Deploys private, self-hosted models on an organization's own infrastructure — removing both the public-API data-leak risk documented in Section 5 and the unbounded usage-based billing risk documented in Section 4

This report's finding	Relevant Bithost service	What it addresses
Section 4.5, 4.8: agentic AI operating with excessive autonomy or access	Agentic AI Security	Secures the AI systems an organization is already running, so agentic workflows operate within defined, monitored limits rather than the ungoverned autonomy documented in Section 4.5
Section 5.2–5.3: leaked secrets, credentials, and API keys	Cybersecurity VAPT and API Security Audit	Finds exploitable exposures — including the secrets-sprawl and excessive-permission patterns documented in Section 5 — before an attacker does
Section 5.7: breach response and containment cost	Incident Response	Provides the rapid containment that IBM's data shows is the single largest lever for reducing the cost and duration of a breach once one occurs
Section 4.5, 5.2: AI-generated code shipped without review	Vibe Code Cleaning	Reviews and hardens AI-generated ("vibe coded") applications for production — directly addressing the elevated secret-leak rate in AI-assisted commits documented in Section 5.2
Section 5.6, 6: legacy systems accumulating unpatched, forgotten risk	System Modernization	Migrates aging, under-maintained systems off infrastructure that has stopped receiving the attention documented in Section 5.6 as necessary to stay secure
Section 8.4–8.6: compliance evidence and continuous governance overhead	Compliance Automation	Replaces manual, spreadsheet-based compliance tracking with automated evidence collection, supporting the continuous-governance model described in Section 8

10.2 Products built for the same gap

Alongside its consulting services, Bithost maintains a set of purpose-built products that operationalize specific parts of the framework in Section 8: Zscanner, a vulnerability scanning platform; EWSAF for Windows environment security; Legac-o-Meter, which assesses legacy-system risk of exactly the kind documented in Section 5.6; Contract Pilot; and PromptShield, aimed at the AI-specific exposure surface documented throughout Section 4 and Section 5.2's discussion of AI-tooling secrets leaks.

10.3 A practical starting point

For organizations that recognize the patterns in this report but are not yet sure where their own biggest exposure sits, the natural starting point is the same one recommended in Section 9: a combined cloud and AI audit — the Days 1–30 visibility phase of the roadmap — conducted by a team that has done this diagnostic work across many organizations rather than once, internally, for the first time. Bithost's Cloud Audit and Cybersecurity VAPT services are built to deliver exactly that kind of baseline inventory, typically within weeks rather than the quarter a fully internal effort often requires, giving an organization the same Section 9 starting point with outside pattern-recognition behind it.

Get in touch

Bithost operates from offices in Patna, Bangalore, Kolkata, Bhubaneswar, and Delhi. For a cloud, AI, or security audit scoped against the framework in this report, reach the team at sales@bithost.in or +91 911-336-6525, or visit www.bithost.in.

10.4 Engagement models

Organizations engage Bithost differently depending on where they sit against the maturity model in Section 8.6. Three patterns are most common, and are offered here to help a reader identify which is the right starting conversation rather than defaulting to the largest possible scope.

- Point audit: a scoped Cloud Audit, API Security Audit, or Cybersecurity VAPT engagement, suited to an organization that wants an independent baseline — the Section 9.1 visibility phase — without committing to an ongoing program up front.
- Managed ongoing service: DevOps & SRE Managed Services or Compliance Automation engaged on a continuing basis, suited to an organization that has identified the gap through its own review (or through a point audit) and wants the standing automation described in Section 9.3 operated by a specialist team rather than built and maintained internally.
- Build partnership: Infrastructure Architecture, AI Integration, or Sovereign AI engaged for a specific build — a new platform, a new AI-powered product, a legacy modernization — where governance is designed in from the architecture stage per Section 8.1's first principle, rather than retrofitted after launch.

None of these require an organization to hand over its entire technology function. Each is scoped to close a specific part of the gap this report documents, consistent with the build-versus-buy framing in Section 8.7: bring in specialist help for the categories of work that benefit most from outside pattern-recognition and dedicated focus, while keeping the organization-specific process decisions in-house.

11. Conclusion

This report opened with a story every reader will recognize: a demo, a staging environment nobody remembers to turn off, and a question from finance nobody can answer with confidence. It closes with the same story told from the other direction — not as an inevitable cost of moving fast, but as the predictable, well-documented, and fixable consequence of one specific, structural gap between how quickly organizations can create technology and how slowly they notice, own, and retire it.

The evidence assembled in Sections 3 through 5 is not a warning about a hypothetical future risk. It describes the present condition of the median technology organization in 2026: 29% of cloud spend delivering no business value, AI adoption outpacing the governance built to control its cost, and credentials leaking onto the public internet at the fastest rate ever recorded. None of it is unique to any single company, and none of it reflects a failure of any individual team's competence or effort. It reflects the absence of a governance layer that most organizations have simply not yet built — not because the fix is unknown, but because it has not yet been prioritized against the next deadline.

The organizations that close this gap successfully, examined throughout Section 8, do not do so by adding friction. They do it by making ownership, cost visibility, expiration, and secure defaults automatic properties of the systems developers already use — so that the fast way to do something and the governed way to do it become the same action. That is a solvable engineering and organizational problem, not a permanent trade-off between speed and control, and the 90-day roadmap in Section 9 is built to prove that inside a single fiscal quarter.

The next time someone asks why the cloud bill — or the AI bill — is higher than expected, the goal is not to have a better excuse. It is to have already closed the gap that made the question necessary in the first place.

Appendix A: Governance Quick-Reference Checklist

A condensed version of the practices detailed in Section 8, organized for use as a working checklist rather than a narrative.

Cloud

- Every resource requires an owner, team, and cost center before it can be created.
- Every non-production environment carries a default expiration date, enforced automatically.
- Idle, orphaned, and unattached resources are detected continuously, not reviewed periodically.
- Every team can see its own infrastructure cost, broken down by resource, without requesting a report.
- A single, centrally visible inventory exists for every cloud account, project, and subscription.

AI

- Every API key and AI platform account has a hard, pre-configured spending ceiling — not an after-the-fact alert.
- Every request is attributable to an individual, team, or workflow, not a shared, anonymous key.
- A cost-aware default routes routine tasks to smaller, cheaper models, reserving frontier models for tasks that need them.
- Development and experimentation usage runs against a separate, tighter budget than production.
- Runtime guardrails detect and stop cost anomalies and loops before they complete, not after a dashboard reports them.
- A single inventory covers every AI provider account, API key, and AI-enabled subscription in use.

Security

- A secrets manager is the fastest way to use a credential — faster than hardcoding one.
- Automated secret scanning runs at commit, at CI/CD, and continuously against full repository history, public and internal.
- New IAM roles and access grants default to the narrowest permission set that unblocks the task.
- Staging and demo environments inherit production's patching cadence, access controls, and monitoring coverage.
- Security checks run continuously through the delivery pipeline, not only as a pre-launch audit.

Appendix D: Sample Resource Tagging Schema

A starting tagging schema, referenced in Section 8.2, that most organizations can adapt directly. The specific keys matter less than enforcing that all of them are required, not optional, at the point of resource creation.

owner: The individual directly responsible for the resource — a person, not only a team distribution list.

team: The team or business unit whose budget the resource's cost should be attributed to.

project: The initiative or product the resource supports, used to group related resources for cost and lifecycle review.

environment: One of a fixed, small set of values — e.g., production, staging, development — used to apply the correct default lifecycle and security policy automatically.

expiry: The date after which the resource should be automatically flagged for review or deletion, required on every non-production resource.

cost-center: The finance cost center the resource's spend should roll up to, enabling the chargeback and showback practices recommended in Section 8.2.

data-classification: Whether the resource may contain sensitive, regulated, or customer data, used to apply the appropriate security baseline from Section 8.4 automatically.

Enforcement matters more than the specific schema: a tagging policy documented but not technically enforced at the point of creation will, per the incentive analysis in Section 6.2, be followed inconsistently under deadline pressure. The schema above is designed to be enforced through infrastructure-as-code validation or a cloud provider's native policy engine, rejecting a resource creation request that omits a required tag rather than relying on a human to remember to add it.

Appendix E: Sample AI Usage Policy (Excerpt)

A starting excerpt an organization can adapt for its own AI usage policy, addressing the specific gaps documented in Section 4. This is offered as a drafting aid, not a complete or legally reviewed policy — have any adopted version reviewed by legal and security stakeholders before publishing it.

1. Access

- All AI provider accounts and API keys must be requested through [designated owner/team] and are issued to a named individual, team, or automated workflow — never as an anonymous, shared credential.
- Every key is issued with a spending ceiling appropriate to its use case, configured before the key is activated, not after.

2. Usage

- Development and experimentation usage must run against a designated development-tier budget, separate from any production key.
- Model selection should default to the smallest, lowest-cost model appropriate to the task; use of frontier-tier models for routine tasks should be a deliberate, documented exception rather than the default.
- Agentic workflows that call AI models autonomously, in a loop, or on a recurring schedule must implement a per-run or per-day cost ceiling and a recursion or loop-depth limit before deployment.

3. Data

- Sensitive or regulated data must not be sent to a third-party AI provider unless that provider and use case have been explicitly reviewed and approved by [security/legal].
- Use of AI tools not on the approved provider list — “shadow AI” as documented in Section 4.7 — is prohibited without prior review.

4. Review

- AI provider accounts, keys, and subscriptions are reviewed on a [quarterly] cadence for continued need, correct budget configuration, and appropriate access scope.
- Any single anomalous usage event above [threshold] triggers automatic notification to [designated owner] and, where the anomaly is consistent with a leaked credential or runaway process, automatic suspension of the key pending review.

Appendix F: Sample Incident Response Runbook Outline

A condensed outline for responding to the two incident types most directly documented in this report — a leaked credential and a runaway AI spending event — offered as a starting structure rather than a complete runbook.

Leaked credential / exposed secret

33. Revoke the credential immediately — revocation, not merely rotation, is the first action, per Section 5.2's finding that the gap between exposure and remediation is where most credential-based breaches occur.
34. Identify everywhere the credential was used or cached (per Section 5.9's finding that a single leaked credential often exists in multiple locations on a compromised system) and confirm each has transitioned to the new credential.
35. Review access logs for the exposure window to determine whether the credential was used by anyone other than its legitimate owner.
36. If unauthorized use is confirmed or suspected, escalate to a full incident response process and follow applicable breach-notification obligations (see Section E).
37. Conduct a brief retrospective: how long was the credential live before exposure, how long between exposure and discovery, and what specific control from Section 8.4 would have prevented or shortened either window.

Runaway or anomalous AI spending event

38. Suspend or rate-limit the implicated key immediately, even before root cause is fully understood — per Section 4.3, minutes matter.
39. Determine whether the cause is a leaked credential (follow the runbook above in parallel), a misconfigured retry or recursion loop, or legitimate but unexpectedly expensive usage.
40. If the cause is a technical bug (a retry storm, a recursion loop), fix and deploy the correction before restoring the key's access.
41. Document the incident's cost impact and root cause for the quarterly governance review referenced in Section 9.5, and confirm a spending ceiling is now in place on the affected key and any similarly configured ones.

Appendix G: Sample Executive Memo Template

A short template for framing a governance program to leadership, drawing directly on the evidence in this report. Replace the bracketed figures with an organization's own findings from the Section 9.1 visibility phase.

> **Subject: Closing our cloud, AI, and security governance gap**
>
> Summary: A review of our cloud and AI infrastructure has identified an estimated [$X] in annual avoidable spend and [N] high-severity, unresolved security exposures, consistent with industry-wide benchmarks showing organizations waste roughly 29% of cloud spend and carry, on average, a $4.44M breach-cost exposure when governance gaps of this kind go unaddressed. Root cause: our provisioning process does not currently require an owner, expiration date, or spending cap at the point a resource or AI credential is created, which is the same gap documented as the leading driver of both waste and risk across the industry. Recommendation: a 90-day program, starting with a full inventory (no architectural change required in month one), followed by closing the highest-severity gaps identified, followed by standing automation so the gains are durable. Ask: [budget/headcount/sponsorship required]. Expected outcome: [estimated savings] recovered within two quarters, and a measurable reduction in unresolved security exposure, tracked against the metrics in Appendix A.

This memo is deliberately short. The supporting detail — the specific findings, the financial model, and the full roadmap — belongs in this report or in an organization's own audit findings, referenced rather than reproduced in the memo itself, so the ask stays legible to an executive audience with limited time.

Appendix B: Glossary

Agentic AI: AI systems that plan and take multiple sequential actions — calling tools, retrieving information, and self-correcting — with limited human review of each individual step.

Chargeback / Showback: Practices that attribute cloud or AI cost back to the specific team or workload that generated it; chargeback bills the team directly, showback simply makes the cost visible to them.

CCOE (Cloud Center of Excellence): A cross-functional group responsible for cloud strategy, standards, and governance across an organization.

CSPM (Cloud Security Posture Management): Tooling that continuously scans cloud environments for misconfigurations — public storage, excessive permissions, open ports — against a defined security baseline.

FinOps: The discipline of bringing financial accountability to variable, usage-based cloud (and increasingly AI) spend through collaboration between engineering, finance, and business teams.

Governed template: A pre-approved, pre-configured infrastructure-as-code pattern that automatically applies an organization's tagging, sizing, and security defaults when a developer provisions a new resource.

IAM (Identity and Access Management): The systems and policies that control which users and services can access which resources, and what they are permitted to do with them.

Idle resource: Infrastructure that is provisioned and billing but receiving little or no active use, commonly measured by CPU utilization sustained below a low threshold over a rolling window.

MCP (Model Context Protocol): An open standard, introduced in 2025, for connecting AI models to external tools, data sources, and services.

Overprovisioning: Allocating more compute, memory, or capacity to a workload than it actually uses, commonly measured as the ratio of requested to actually-used resources.

Privilege creep: The gradual, one-directional accumulation of access permissions over an account's lifetime, as grants are added but rarely revoked.

Rightsizing: Adjusting the size or type of a provisioned resource to match its actual, observed usage rather than its originally estimated or requested usage.

Secrets sprawl: The uncontrolled proliferation of hardcoded credentials — API keys, passwords, tokens, certificates — across code repositories, configuration files, CI/CD pipelines, and collaboration tools.

Shadow AI / Shadow IT: AI tools or technology systems adopted and used within an organization without formal review, approval, or oversight from IT or security.

Sovereign AI: AI models and infrastructure deployed and operated on an organization's own infrastructure, so that data does not leave the organization's control via a third-party public API.

Time-to-live (TTL): A predefined lifespan attached to a resource at creation, after which it is automatically flagged for review or deletion unless explicitly renewed.

Token: The basic unit of text — roughly a word fragment — that large language model providers meter and price usage by.

VAPT (Vulnerability Assessment and Penetration Testing): A structured security testing practice combining automated vulnerability scanning with manual, adversarial testing to find exploitable weaknesses before an attacker does.

Appendix C: Selected Sources

This report draws on publicly available industry research current as of August 2026. Primary sources referenced throughout include:

- Flexera, 2026 State of the Cloud Report (survey of 753 cloud decision-makers, published March 2026).
- Cast AI, 2026 State of Kubernetes Optimization Report.
- Datadog, State of Cloud Costs Report.
- CNCF (Cloud Native Computing Foundation), 2024 Annual Survey and cloud-native FinOps microsurvey.
- GitGuardian, The State of Secrets Sprawl 2026 (fifth annual edition, published March 2026).
- IBM (Ponemon Institute), Cost of a Data Breach Report 2025.
- SpendArk, The State of Cloud Waste 2026.
- Verizon, 2025 Data Breach Investigations Report (secrets remediation timelines).
- Publicly reported AI usage-cost incidents aggregated from Axios, HackerNoon, DEV Community, KuCoin Learn, and independent engineering and FinOps blogs, 2026.
- Bithost (www.bithost.in), corporate service and product descriptions, accessed August 2026.

End of report.