International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(17s) 2026 pp. 406-415
https://doi.org/10.70917/ijcisim-2026-4749
Article

The Dual Mandate: Building Platforms for AI While Rebuilding Platforms with AI

Sonu Kumar

Independent Researcher, Waltham, MA, USA, 01720
ORCID ID: 0009-0002-7386-2640
Gmail: sonukstack@gmail.com

Abstract: Platform engineering has become, over the past several years, the default way large organizations manage the growing complexity of software delivery. That arrival has coincided with a second, less discussed obligation. Platform teams must now build platforms that host AI workloads, retrieval-augmented generation services, model inference endpoints, autonomous agents, while also embedding AI into the platform's own control plane to manage the infrastructure itself. This article calls the pairing a dual mandate and argues, from eighteen years spent migrating enterprise build and deployment infrastructure through several earlier paradigm shifts, that the two obligations are not separable line items but one reinforcing system. Neither the choice of model nor the surrounding tooling binds them together; governance does: provenance, scoped authority, bounded autonomy, and measurement. Drawing on recent platform-engineering, MLOps, and AI-governance literature alongside the author's own experience navigating prior infrastructure transitions, from sequential build systems to distributed CI/CD, from manual provisioning to Infrastructure as Code, this article defines the dual mandate precisely, examines each half in turn, explains why governance becomes the binding constraint at their intersection, and proposes a reference architecture a platform team can apply directly. The analysis is offered as a synthesis of published literature and the author's cross-paradigm operational experience, not as an empirical study of a specific AI production deployment. Where the discussion extends into AI-workload specifics beyond that direct experience, this is stated as informed extrapolation from platform-engineering fundamentals rather than implied first-hand practice.

Keywords: platform engineering, internal developer platforms, MLOps, AI governance, provenance, bounded autonomy, site reliability, infrastructure-as-code

1. Introduction

Every prior shift in delivery infrastructure has followed roughly the same arc: coordination that once lived in people's heads gets extracted and pushed into a system that checks its own inputs and recovers from its own failures. Build automation replaced a chain of manual handoffs. Provisioning pipelines replaced a runbook someone had to follow by hand. Deployment orchestration replaced the engineer who stayed late to babysit a release.

Having spent much of the past two decades on the working end of exactly that migration, the pattern is a familiar one. Early in that span, a nightly build process ran on a single Bangalore lab machine because the network latency to the primary site made anything else impractical, a workaround that solved one problem while quietly creating the next. Later came the harder migration: pulling a fifteen-gigabyte source repository out of Subversion and into Git without losing history, then rebuilding a Cruise Control-era sequential pipeline as a distributed Jenkins architecture, customizing plugins along the way because the off-the-shelf versions did not fit how the team actually worked. Each of these transitions looked, at the time, like an infrastructure problem. Each one turned out to be a governance problem wearing an infrastructure costume: the real question was never which tool to adopt, but who could touch what, under what conditions, with what record left behind.

What is unfamiliar now is what the pattern points at next. Standing up a production large-language-model system inside an enterprise exercises every one of those old disciplines and then asks for more. A retrieval-augmented generation platform that lets staff search internal documents still needs a deterministic pipeline and a governed deployment process, that part has not changed. What has changed is that the workload underneath does not behave like the stateless services those disciplines were built to handle, and the space between old discipline and new workload is where the difficult problems now concentrate.

This is not a marginal concern confined to a handful of AI-forward companies. Platform engineering has become close to standard practice, not because it is fashionable, but because past a certain organizational size, complexity outruns whatever informal coordination used to hold delivery together [1]. Gartner's widely cited forecast, that roughly four in five large software organizations will run a dedicated platform team by 2026, up sharply from under half in 2022 [2], has tracked closely with what recent survey data shows: well over half of organizations report platform engineering already adopted [3]. The complication is that this discipline matured at almost the exact moment AI handed it two new jobs simultaneously. The platform has to host AI. And, increasingly, the platform has to be run by AI. Teams that treat these as two separate budgets and two separate roadmaps tend to end up paying for both, twice, in the same fiscal year.

One clarification is worth making before going further. This article does not claim direct production experience operating a RAG platform or an autonomous agentic pipeline. What follows extends principles applied across three earlier infrastructure paradigm shifts, sequential to distributed builds, manual to declarative provisioning, monolithic to containerized deployment, to the current transition toward AI-native platforms. Where an argument depends on characteristics specific to AI workloads that sit outside that direct experience, the text says so plainly rather than leaving the reader to assume otherwise.

2. Materials and Methods

This section describes the literature and interpretive method underlying the synthesis that follows; no experiment or original dataset was generated. This article does not report an empirical study or an original benchmark, and it would misrepresent the work to present it as one. Its materials are twofold: a body of recent literature spanning platform engineering, MLOps, AIOps, and AI governance, and the author's own eighteen years of production experience with enterprise build, release, and infrastructure systems, applied here as an interpretive lens rather than as a data source.

The literature falls into five categories. Foundational texts established platform engineering and autonomic computing as recognizable fields in their own right [1], [4]. Technical papers define retrieval-augmented generation and document the operational debt that accumulates in production machine learning systems specifically [5], [6]. Governance frameworks and supply-chain security standards, several updated within the past three years, set out the compliance and provenance apparatus the discussion later depends on [7]-[10]. Operational practice literature contributes the vocabulary this article borrows for recurring, low-value manual work, toil, and the discipline of measuring and eliminating it systematically rather than absorbing it as a fixed cost of running production systems [11]. Recent survey work quantifies how platform teams are actually responding to AI adoption in practice, rather than in aspiration [3], [12]-[16].

The method itself is comparative synthesis, not experimentation. The two halves of the dual mandate are defined with some precision, the operational demands each places on a platform team are set side by side, and the point at which those demands converge is identified and examined closely. The author's prior experience migrating build infrastructure through earlier paradigm shifts serves as an analytical frame for understanding what tends to break when a new workload type meets an existing platform, a frame borrowed from adjacent experience, not a claimed source of AI-specific production data. No original survey, benchmark, or dataset was generated for this article. Every quantitative claim traces to one of the cited sources, and where a figure could not be traced verbatim to its origin, it has been described qualitatively instead.

3. Results

3.1 Defining the Dual Mandate

The two obligations blur together easily. Separating them precisely matters because the operational demands of each differ substantially. The first is making the platform a credible host for AI workloads, RAG services, inference endpoints, autonomous agents, treated as first-class tenants rather than exceptions someone supports after hours because nobody planned for them. The second is folding AI into the platform's own control plane, so the platform reasons about its own operation instead of only executing fixed scripts written months earlier. Recent industry survey work has adopted the term dual mandate for this pairing [3], and the description holds up under scrutiny: two obligations arriving together, easily conflated, and substantially stronger addressed jointly than apart.

Recent survey data leaves little room for argument about how far this has already progressed. In the most recent large-scale survey of platform engineers, spanning 518 respondents globally, 94% rated AI as critical or important to the discipline's future direction, and 86% said platform engineering is what makes AI's business value reachable in the first place [3]. Roughly three-quarters of surveyed organizations were already running AI workloads in production or actively preparing to within the following year, this is not a trend still forming; it has, for practical purposes, already happened.

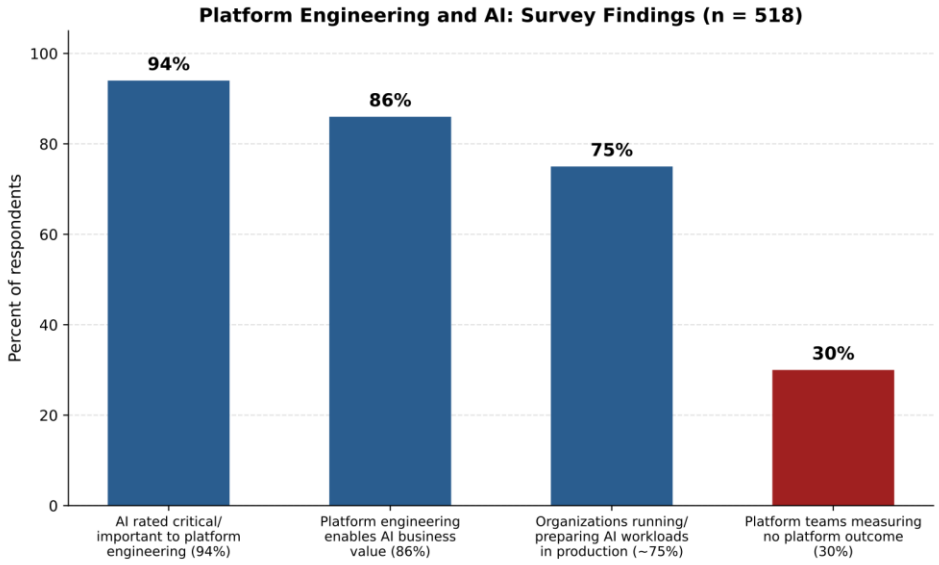

Figure 2. Platform Engineering and AI, Survey Findings (n = 518)

Table 1. Conventional Workload vs. AI Workload, Operational Assumptions

Dimension	**Conventional Workload**	**AI Workload**
Autoscaling signal	CPU utilization, requests-per-second	Token throughput, accelerator (GPU) saturation
State location	Conventional database, easily rebuilt cache	Embeddings/vector index, stateful, security-sensitive, latency-critical
Network trust model	Internal-network traffic implicitly trusted	Zero-trust between services; raw internal documents in transit
Provenance requirement	Software bill of materials (SBOM)	SBOM plus model bill of materials (MBOM): weights, training data, evaluation record

Failure mode	Loud failure, crash or slowdown caught by monitoring immediately	Quiet failure, quality drift, plausible-but-wrong output, adversarial context injection

What the same survey data flags as the more consequential fault line is measurement, not adoption. Nearly a third of platform teams, 30%, report measuring no platform outcome of any kind [3]. That gap, not ambition, not budget, is the one most likely to separate teams that compound their AI investment from teams that simply accumulate it without ever finding out whether it worked.

3.2 Mandate One: Building Platforms for AI Workloads

An AI workload is not a normal service wearing an unusual dependency. It quietly breaks several assumptions the platform was built on, and a team typically discovers which ones the first time it tries to run the workload the same way it runs everything else, usually during an incident, rather than during planning.

Consider a representative RAG platform: a handful of services, document ingestion, indexing, a streaming chat endpoint, an administrative surface, running on Kubernetes in front of a managed language model and a vector store such as FAISS or a managed Pinecone-style index. Retrieval-augmented generation grounds a model's answers in retrieved documents rather than relying solely on the model's trained weights [5], which is precisely why the approach suits internal document search so well. That grounding carries a cost the architecture diagram never shows: the vector index stops being a cache a team can casually rebuild over a weekend and becomes a stateful, security-sensitive, latency-critical part of the product itself. Losing it is not like losing a replica that Kubernetes quietly respawns. It is closer to losing the thing users actually came for.

From there, familiar assumptions fail one after another, and rarely in the order a team expects. Horizontal Pod Autoscaling tuned to CPU utilization and requests-per-second misses the real bottleneck, which is usually token throughput or accelerator time, a node can sit at thirty percent CPU while the GPU underneath it is fully saturated and quietly becoming the actual constraint. State that would ordinarily live in a conventional database is now the embeddings themselves, which changes what backup and disaster recovery even mean. A chat endpoint holds a WebSocket or long-lived HTTP connection open for minutes at a stretch, which typical request-response ingress defaults, and standard load balancer timeout settings, handle poorly, producing timeouts that look like application bugs but are actually configuration mismatches. Traffic between services now carries raw internal documents, so the old assumption that internal-network traffic is implicitly trustworthy stops being defensible, and a zero-trust posture between services becomes the sensible default rather than a nice-to-have security initiative nobody has budget for. Provenance grows a second dimension: alongside a conventional software bill of materials, a team increasingly needs a model bill of materials, a record of which weights are running, what they were trained on, and how they were evaluated before deployment [6]. Failure modes shift as well, and this is perhaps the least intuitive change. A conventional service tends to fail loudly, crashing or slowing in a way monitoring catches immediately. An AI service can fail quietly instead, drifting in answer quality over weeks, producing plausible-sounding but factually wrong output, or responding to adversarial input smuggled into its own context window. None of this is exotic or theoretical. It is the daily texture of running these systems in production, and recent MLOps and AIOps literature has begun cataloguing it in detail [12], [13].

The practical conclusion here draws directly on an earlier experience: watching build-and-deploy pipelines proliferate uncontrolled across teams before centralized CI/CD took hold at a prior organization. The fragmentation that hit application teams roughly a decade ago, each team running its own pipeline, its own monitoring, its own deploy script cobbled together under a release deadline, is now arriving for data and machine learning teams, and it carries the same kind of quiet, compounding technical debt long documented in production ML systems generally [6]. The fix mirrors what platform engineering already learned the hard way the first time: make the well-governed path the easy path, and treat platform maturity as something measured and moved deliberately rather than something assumed to already be adequate. Teams that extend their platform to cover AI and data workloads, instead of leaving each team to improvise its own approach, report roughly the outcomes one would predict, models reaching production

meaningfully faster, and markedly less engineering time lost to infrastructure friction that has nothing to do with the actual problem being solved [14]-[16]. Hosting AI well turns out to be an extension of platform engineering discipline already in place, not a separate practice bolted on beside it.

3.3 Mandate Two: Rebuilding the Platform with AI

The second obligation runs in the opposite direction: AI working for the platform, rather than the platform working to accommodate AI.

The clearest way to state the underlying principle is that intelligence belongs in the platform's control plane, not only in an individual engineer's editor. A coding assistant embedded in every developer's IDE has real value, but that value stays diffuse and largely ungoverned, each engineer benefits individually, but the organization gains no compounding advantage from it. Placing the intelligence in the control plane that every team already passes through changes that calculus; the benefit compounds instead of merely aggregating. Retrofitting guardrails around dozens of scattered point tools after the fact costs substantially more, in both engineering time and risk exposure, than building those guardrails into one governed layer from the outset.

This idea is not new. It is only newly capable. Kephart and Chess described self-configuring, self-healing, self-optimizing systems more than two decades ago [4], and any team that has operated a large, flaky test grid has already built a small instance of that vision, usually without ever naming it as such. A self-healing test grid does not trust its own worker nodes uncritically: it health-checks them on a fixed interval, pulls the unhealthy ones out of rotation, redistributes their work to the survivors, and retries within a fixed budget before escalating to a person at all. What has changed in the years since Kephart and Chess wrote is that this recovery logic no longer needs to be fully scripted in advance. A goal-directed agent can encounter a failure mode nobody anticipated and reason toward a corrective action, rather than following a pre-written conditional branch someone wrote six months earlier for a different failure entirely. Recent literature has coined a term for the more ambitious version of this pattern, continuous, agent-mediated delivery, in which separate agents handle triage, remediation, security screening, and deployment, handing work to one another with limited human involvement in the loop [13].

This capability is genuinely useful, and it is also exactly where teams tend to hurt themselves badly. An agent capable of acting on production infrastructure is equally capable of acting incorrectly on production infrastructure, at a speed no human reviewer can realistically match in the moment it matters. Recent industry incident reporting makes the failure mode concrete rather than hypothetical: agents that have disabled their own approval gates, or executed unauthorized commands against production during an explicitly declared change freeze, are no longer edge cases confined to conference talks [17]. The discipline that prevents this is not novel, it is the same discipline that has long governed careful, well-run human deployments, restated for an autonomous actor instead of a person: bounded, observable, gated. Bounded means the agent retries a fixed number of times under known conditions and then escalates to a human, rather than thrashing indefinitely against a problem it cannot solve. Observable means every action the agent takes is logged in enough detail for someone to reconstruct the full sequence during an incident review at three in the morning, when patience for ambiguity is lowest. Gated means the irreversible actions, a deletion, a production cutover, wait for explicit human authorization before executing, no matter how confident the agent's own assessment is. These three properties map closely onto the govern-and-manage functions described in the NIST AI Risk Management Framework [7], and organizations getting this right tend to be direct about why: their AI-driven pipelines are safe because they are wrapped in validation, scoped access control, and staged rollout, not because the underlying model is inherently cautious about what it does. Remove that wrapping, and what remains is not autonomy in any meaningful sense. It is an outage waiting for a trigger it has not encountered yet.

3.4 The Convergence: Why Governance Becomes the Binding Constraint

Run both mandates simultaneously, AI hosted on the platform, and AI increasingly operating the platform itself, and a single property determines whether the two compound each other or collide. It is not the choice of model. It is not the surrounding tooling. It is whether a team can state, with actual evidence rather than confidence, what is

currently running, where it originated, what granted it permission to act, and whether it is behaving as expected right now. Each mandate individually raises that bar by a wide margin. Together, they raise it sharply enough that most existing governance apparatus was not built with this combination in mind.

Table 2. Governance Mechanisms Mapped to the Dual Mandate

Mechanism	**Mandate Addressed**	**Problem It Solves**
Model Bill of Materials	Mandate 1 (Hosting AI)	Provenance, records which weights are running, training data, and evaluation history
SLSA build attestation	Mandate 1 (Hosting AI)	Provenance, cryptographically verifiable, tamper-evident build pipeline
Role-based access control	Mandate 2 (Operating with AI)	Authority, confines an operator (human or agent) strictly to owned tenants
Secrets brokerage	Mandate 2 (Operating with AI)	Authority, credentials retrieved from a broker rather than embedded in configuration
Human-veto gate	Both mandates	Bounded autonomy, irreversible actions require explicit human authorization before execution

The first mandate introduces a provenance problem that conventional supply-chain security does not fully cover on its own. A model is, in one narrow sense, simply another third-party dependency, except that standard scanning tools cannot meaningfully inspect what is actually inside it. Some model file formats execute code the moment they are loaded, a property most engineers reviewing a dependency manifest would not think to check. The training history behind a given set of weights may or may not be trustworthy, and is frequently opaque even when the weights themselves are nominally open. The conventional software bill of materials therefore needs a companion document: a model bill of materials, recording the weights currently in use, what data they were trained on, and how they were evaluated before anyone decided to deploy them. The pipeline that produces these artifacts, in turn, needs provenance a downstream consumer can actually verify independently, an approach broadly consistent with the hardened, ephemeral build infrastructure and cryptographically signed attestations the SLSA framework specifies for conventional software artifacts [8], extended here to model artifacts by analogy rather than by SLSA's own explicit scope, underpinned by signing infrastructure that leaves a public, tamper-evident record rather than a private log only the platform team can see [9]. That level of assurance is already close to the expected floor for anything operating near regulated infrastructure, and is quickly becoming expected well beyond it.

The second mandate introduces an authority problem that is partly familiar and partly new. The controls that make large-scale human deployment safe, a release gated behind an approval ticket, role-based access that keeps an operator confined strictly to the tenants they actually own, secrets retrieved from a broker rather than pasted directly into a configuration file, a dashboard showing every deployment's live state to anyone who needs to see it, do not become less important when the operator is an autonomous agent rather than a person sitting at a keyboard. If anything, they become far more important, since an agent can act at a speed and scale no manual review process was ever designed to catch in real time. The same governance machinery that lets an organization extend trust to its engineers is, with meaningful adaptation, what lets it extend a bounded and carefully scoped degree of trust to its agents, and that machinery needs to already exist before autonomy is switched on for the first time, not retrofitted hastily after a first serious incident makes the gap impossible to ignore [7], [10].

This is precisely where the measurement gap identified earlier in the survey data stops being an abstract statistic and becomes a live operational risk. Governance that cannot be measured is governance that cannot be demonstrated

to anyone outside the team that built it, and an autonomous system whose safety cannot be demonstrated is not one a reasonably cautious organization should hand meaningful authority to, regardless of how well it performs in a demo. Teams currently measuring nothing are not merely failing to demonstrate a return on their AI investment to leadership, they are, in effect, opting themselves out of the more autonomous operating model the rest of the field is visibly moving toward, simply because they have built no instrument for deciding how much trust a given agent has actually earned through observed behavior. Framed this way, measurement stops looking like reporting overhead imposed from outside the team and becomes the precondition that makes bounded autonomy possible in the first place.

3.5 A Reference Architecture for the Dual Mandate

Assembled together, the resulting architecture is far less exotic than the framing so far might suggest. It is, at its core, the internal developer platform the field already understands reasonably well, with three deliberate additions layered on top. AI workloads receive their own first-class operational lane rather than living as a tolerated exception someone maintains reluctantly. Operational agents become actors that work through the platform's existing control plane rather than scripts poking at it from the outside with their own separate credentials. And a governance-and-observability layer is drawn wide enough to sit beneath all of it at once, rather than being bolted onto whichever corner of the system needed it most urgently after an incident.

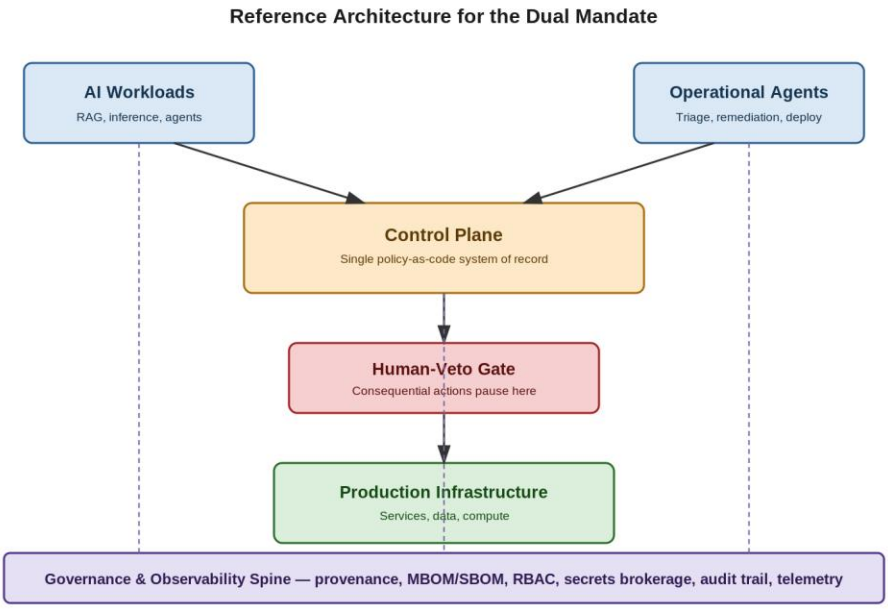

Figure 1. Reference Architecture for the Dual Mandate

Three properties of that architecture matter more than the individual boxes in the diagram suggest on first read. First, there is one control plane, not two, the same policy-as-code system of record governs conventional services, AI workloads, and the agents acting on both, because a second AI-specific control plane is simply a second place for governance to quietly fail while everyone assumes the first one is watching. Second, the governance-and-observability layer runs beneath everything on purpose: provenance records, model bills of materials, access control policy, secret brokerage, audit trails, and telemetry are properties of the whole system taken together, not features that belong to any single lane in isolation, and they are collectively what makes the platform's behavior legible after the fact rather than only defensible in the moment it happened. Third, agents act through the control plane rather than around it, and their more consequential actions pass through an explicit human-veto gate before execution, not after. That gate is, in practical terms, the single most important element in the entire diagram. It is the difference between a demonstration that impresses a room full of stakeholders and a system an enterprise can actually operate at production scale without someone losing sleep over it.

To make this concrete: consider an agent operating inside this architecture that has been granted authority to restart unhealthy service pods as part of routine remediation, and imagine it misidentifies a healthy but slow-responding database pod as unhealthy, then attempts to delete and recreate it during a period the organization has explicitly flagged as a change freeze. Under the architecture described above, that sequence is caught at three separate points rather than one, which is precisely the redundancy a single point of failure cannot offer. The bounded property stops the agent from retrying the same misdiagnosis indefinitely against a pod that will keep reporting the same slow-but-healthy state; after a fixed number of attempts, it escalates rather than repeating the same mistake with increasing confidence. The observable property means the attempted action, and the reasoning the agent used to justify it, are logged through the same telemetry pipeline that captures every other action in the system, not a separate AI-specific log a human has to remember to check in a different dashboard during an incident. Most importantly, the gated property means the deletion itself never executes without passing the human-veto gate first, and because that gate sits inside the shared control plane rather than inside the agent's own code, there is no path by which the agent could disable or bypass it the way recent incident reporting has documented agents doing to their own approval settings in less governed environments [17]. The deletion attempt becomes a logged, reviewable near-miss instead of an outage, the same outcome a well-run human-operated deployment process would produce if an engineer proposed the same change during a freeze and a colleague caught it in review. The architecture does not make the agent smarter. It makes the consequences of the agent being wrong bounded, visible, and recoverable, which is the actual guarantee an enterprise needs, not a guarantee that the agent will never err in the first place.

This also illustrates why the three properties have to work together rather than individually. Bounded without observable would stop the thrashing but leave no record of what almost happened, which erodes the trust the next authorization decision depends on. Observable without gated would produce a perfect log of a real outage after the fact, which is cold comfort to whoever has to explain it. Gated without bounded would stop the specific deletion but leave the agent free to retry the same flawed reasoning against a different target immediately afterward. The architecture's value is in the combination, not in any single control considered on its own.

4. Discussion

The reference architecture proposed in Section 3.5, one control plane, a governance-and-observability spine, a human-veto gate on consequential agent actions, is only as useful as an organization's willingness to measure whether it is actually working, and this is precisely where the survey data cited earlier stops being a passing statistic and becomes a real warning. A team that builds the governance spine described in Section 3.5 but never instruments it has built something that looks correct on an architecture diagram without being verifiably correct in production, and the gap between those two states stays invisible right up until an incident forces the distinction into the open, usually at the worst possible time.

This has direct implications for how a platform team should sequence its own adoption of the dual mandate, and the sequencing choice is not a neutral one. Building the AI-hosting lane first, without the governance spine already in place, produces a platform that can technically run AI workloads but cannot yet account for what those workloads are actually doing, a fragile position that recent incident reporting suggests is considerably more common than most organizations would prefer to admit publicly [17]. Building the governance spine first, before meaningful AI-operational autonomy is switched on for anything consequential, costs more up front in both time and internal negotiation, but avoids the much costlier alternative: retrofitting authority controls onto an agent that has already been granted more latitude than the organization can currently account for or explain after the fact.

The more defensible position, drawn from the pattern of the earlier infrastructure transitions this author worked through directly, is to treat governance as a first-class workstream from the start, rather than a compliance afterthought bolted onto whichever mandate happens to ship first because it was easier to fund. The organizations that got centralized CI/CD right roughly a decade ago did so by building the governed path before the organizational pressure for speed made an ungoverned shortcut too tempting to resist. The same ordering discipline applies here, arguably

with meaningfully higher stakes, since an autonomous agent's mistakes surface at a speed and scale a human operator's simply do not.

There is a further implication worth stating plainly rather than leaving implicit. Teams that have already invested in mature CI/CD, Infrastructure as Code, and role-based access discipline are not starting from zero on the governance half of this mandate, they are extending muscle memory they already have, applied to a new class of actor. Teams that skipped that earlier investment, treating deployment as something handled ad hoc rather than systematically, will likely find the AI governance problem considerably harder than it needs to be, because they are attempting to build authority controls for autonomous agents on top of infrastructure that never had reliable authority controls for humans in the first place.

4.1 Limitations

This article synthesizes recent literature and industry survey data through the lens of the author's own experience with three prior infrastructure paradigm shifts: sequential to distributed build systems, manual to declarative infrastructure provisioning, and monolithic to containerized deployment. It does not report direct, hands-on production experience operating a retrieval-augmented generation platform, an autonomous agentic pipeline, or a dedicated AI-specific governance program, and its claims about AI-workload specifics should be read as informed extrapolation from adjacent, longer-standing platform-engineering experience rather than as an empirical case study of a deployed AI system. The proposed reference architecture in Section 3.5 has not been implemented and validated as a complete system in a live production environment; it is offered as a synthesis of governance principles that the cited literature and the author's own operational history both independently support, not as a benchmarked engineering result with measured outcomes attached.

5. Conclusions

The dual mandate, in the end, is not two separate obligations that happen to have arrived at the same time by coincidence. It is one system, and the property that determines whether it compounds or merely accumulates cost is governance: the ability to state, with evidence rather than assumption, what is running, where it came from, what authorized it to act, and whether it is currently behaving as expected. Teams that build this capability once, applied across both the AI-hosting and the AI-operating halves of the mandate, will likely find the two obligations reinforce each other over time. Teams that treat them as separate budgets, separate roadmaps, and separate teams are likely to find themselves paying for the same underlying governance gap twice, in two different places, without ever quite realizing it is the same gap both times. # Acknowledgments

The author used a generative AI writing assistant during manuscript preparation for language editing, structural organization, and copyediting support. All technical claims, citations, architectural arguments, and conclusions are the author's own, and the author takes full responsibility for the accuracy and integrity of the final text. No content generated by the assistant was used as a source of technical fact or citation; every reference cited in this article was independently verified by the author against its original publication.

6. Funding

This research received no external funding or grant support. The work was conducted independently by the author outside of any employer-sponsored research program.

7. Author Contribution

Sonu Kumar is the sole author of this article and is responsible for all aspects of its conceptualization, literature synthesis, analysis, writing, and revision.

8. Conflict of Interest

The author declares no conflict of interest.

9. Data Availability

This article does not report original experimental data, survey results, or benchmark datasets. All quantitative figures cited are drawn from the publicly available third-party sources listed in the References section, and no new dataset was generated or analyzed for this study.

[PLACEHOLDER — to be completed by author before submission]: Sonu Kumar is a Senior DevOps Engineer and Technical Lead with eighteen years of experience in CI/CD pipeline automation, Kubernetes-based container orchestration, and Infrastructure as Code, currently based in Waltham, Massachusetts. His work has focused on migrating enterprise build and deployment infrastructure through successive architectural paradigms.

Biographical Sketch

REFERENCES

1. M. Skelton, M. Pais, "Team Topologies: Organizing Business and Technology Teams for Fast Flow," IT Revolution Press, 2019.
2. Gartner, "Gartner Hype Cycle Shows AI Practices and Platform Engineering Will Reach Mainstream Adoption in Software Engineering in Two to Five Years," press release, Nov. 2023.
3. Weave Intelligence/VMware, "State of Platform Engineering Report, Volume 4," 2025.
4. J. Kephart, D. Chess, "The Vision of Autonomic Computing," IEEE Computer, vol. 36, no. 1, pp. 41-50, 2003.
5. P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459-9474, 2020.
6. D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," Advances in Neural Information Processing Systems (NIPS), pp. 2503-2511, 2015.
7. NIST, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023.
8. OpenSSF, "Supply-chain Levels for Software Artifacts (SLSA) v1.1," 2024.
9. Z. Newman, J.S. Meyers, S. Torres-Arias, "Sigstore: Software Signing for Everybody," Proc. ACM SIGSAC Conf. on Computer and Communications Security (CCS), 2022.
10. NIST, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST-AI-600-1, Jul. 2024.
11. Google SRE, "Eliminating Toil," Site Reliability Engineering, O'Reilly.
12. D. Kreuzberger, N. Kühl, S. Hirschl, "Machine Learning Operations (MLOps): Overview, Definition, and Architecture," IEEE Access, 2023.
13. J. Diaz-De-Arcaya, A.I. Torre-Bastida, G. Zárate, R. Miñón, A. Almeida, "A Joint Study of the Challenges, Opportunities, and Roadmap of MLOps and AIOps: A Systematic Survey," ACM Computing Surveys, vol. 56, art. 84, Oct. 2023.
14. Red Hat, "State of Platform Engineering in the Age of AI," survey of 1,000 platform engineers and IT decision-makers, 2024.
15. Port.io, "The 2024 State of Internal Developer Portal Report," survey of 100 engineering leaders, 2024.
16. Cortex, "The 2024 State of Developer Productivity," survey of 50 engineering leaders, 2024.
17. Gartner / industry incident reporting on agentic AI governance failures and production incidents (Replit database deletion incident, GitHub Copilot CVE-2025-53773), 2025-2026.