A TECHSTRONG SPECIAL REPORT

The Great Unification

AI, Cloud Native, and the New Shape of Software Engineering

By Alan Shimel · CEO, Techstrong Group

Techstrong
a Futurum company

2026

Contents

How to use this report

This report is written to be read in one sitting or returned to in fragments. Three paths through the material:

Executive readers pressed for time can read Sections 1, 9, 11, and 13 in about twenty minutes and come away with the thesis, the operating model, the recommendations, and the close.

Practitioners who own delivery will get the most from Sections 4, 5, 6, and 7, which cover the stack changes, platform engineering's new remit, the developer role, and the day-to-day for DevOps, platform, QA, and security.

Researchers and skeptics should read Sections 2, 3, 8, 10, and 12 for the historical framing, the evidence that the cloud native stack IS the AI stack, the pluralist infrastructure reality, the objections and honest unknowns, and the falsifying signals.

Citations refer to Techstrong and Futurum research listed in the appendix; company-reported figures and vendor claims are identified in the prose.

SECTION 1 OF 13

Executive Summary

The disciplines are not disappearing. AI is the workload the stack must run and the worker operating inside it, and that dual position pulls infrastructure, platform, delivery, engineering, QA and security back into one system.

For the better part of twenty years, we kept making software infrastructure more capable and software engineering more specialized. Virtualization created virtualization teams. Cloud created cloud architects and eventually FinOps. DevOps set out to tear down a wall and somehow became a job title. Kubernetes brought portability, along with a new class of experts needed to make that portability work.

None of that was a mistake. Each layer solved a real problem. But each also handed somebody a new operational burden and a vocabulary the rest of the organization did not share.

AI is starting to reverse that pattern. It is doing so for a practical reason: AI is both a workload the stack must run and a worker operating inside that stack. It needs infrastructure, delivery, governance, evaluation, identity, and cost control at the same time. No team can provide all of that alone.

That is what we mean by the great unification. The disciplines are not disappearing, and the infrastructure is not collapsing onto one provider or one location. The common operating model is bringing the disciplines back into the same system.

What the research shows

The cloud native stack has become the AI stack. AI did not arrive with a separate enterprise infrastructure model. It moved into the one built for microservices. CNCF reports that 82% of container users run Kubernetes in production, while 66% of organizations hosting generative AI models use Kubernetes for at least some inference workloads. The Certified Kubernetes AI Conformance Program grew from 18 to 31 platforms between November 2025 and March 2026. Hyperscalers, on-premises distributions, a hypervisor vendor and GPU neoclouds all showed up on the same roster. When AI needed accelerator scheduling and model-aware routing, the ecosystem extended Kubernetes and its Gateway API. It did not build a second estate off to the side.

Platform engineering is becoming the delivery layer for AI capability. The platform team's product now includes inference endpoints, approved model catalogs, agent runtimes, GPU quota and governance built into the gateways people already use. The research supports the connection, with an important caveat about causation. Seventy-three percent of platform-mature organizations say platform maturity was critical or significant to AI success, compared with 44% of less mature organizations. On governance automation, the gap is 79% to 14%. DORA reaches a compatible conclusion: AI amplifies the organization it enters. Strong platforms and clear workflows get stronger. Dysfunction also scales.

> "AI amplifies the organization it enters.
> Strong platforms and clear workflows get stronger.
> Dysfunction also scales."
>
> §1 Executive Summary

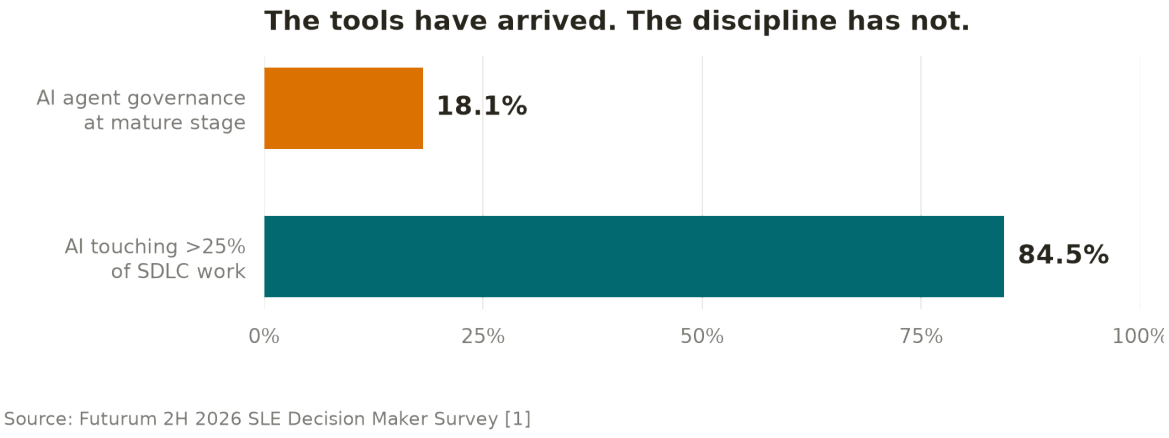
At the same time, infrastructure is becoming more plural, not less. Sovereignty, data gravity, GPU economics and edge latency push workloads in different directions. Fifty-seven percent of IT leaders already say they need to run infrastructure within a single country. A common operating model can now span hyperscalers, private clouds, bare metal, neoclouds and virtualized estates. Hybrid is no longer an apology for an incomplete cloud migration. For many organizations, it is the sensible destination.

The layered operating model

Infrastructure (pluralist) › platform (built and operated by platform engineering) › delivery process (CI/CD, owned by DevOps) › engineering work (AI-directed, human-verified) › trust layer (QA and security, across all of it), with AI at the center as both the workload the stack runs and the worker operating inside it.

What each reader should take from this

Engineering leaders: The binding constraint is platform and governance capacity, not access to another AI tool. AI agent governance sits at 18.1% maturity while more than 84.5% of organizations report AI touching over a quarter of their software development lifecycle work. The tools have already arrived. The operating discipline has not.



Source: Futurum 2H 2026 SLE Decision Maker Survey [1]

Figure 2 — The adoption–governance gap. AI has landed in most SDLCs; the operating discipline for AI agents has not.

DevOps engineers: the pipeline is the product. It gains evaluation gates, guardrail checks, and model deployment stages, while the underlying infrastructure becomes platform-provided. The role moves from building pipelines to deciding what the delivery process must prove.

Platform engineers: you are the control point for safe AI adoption. The alternative is shadow AI, and with 65% of executives calling AI policies clear against only 43% of knowledge workers, that gap is already open.

Software engineers: Code generation is getting cheaper. Specification, decomposition and verification are becoming more valuable. The scarce resource was never typing. It was knowing what to build and whether the result was right.

> **"The scarce resource was never typing.**
> **It was knowing what to build and whether the result was right."**
>
> §1 Executive Summary

QA: verification is now the binding constraint on the entire system. Your stock rises. The eval suite is unowned in most organizations and available to whoever claims it.

Security: you absorb the largest scope expansion of any role. 91% of organizations use AI agents; 10% have a developed non-human identity strategy; 68% cannot distinguish agent activity from human activity.

What this report does not claim

This report does not claim the transition is complete. Platform engineering sits at 36.7% maturity, 47.2% of organizations remain in individual-assistance mode and 15% report no structured AI use. Nor are these roles merging into some generic super-engineer. Specialization matters more when specialists share a platform and their work affects one another directly.

AI has not made engineering easy. Faros AI found pull requests grew 154% larger in 2025 and 51.3% larger in 2026, pushing more work into review and verification. The junior talent pipeline is unsettled. Governance has badly trailed adoption. The most valuable parts of the AI middleware layer may still harden into proprietary control points. Those are not footnotes to the thesis. They are the conditions under which it either holds or fails.

SECTION 2 OF 13

How We Got Here: Five Waves of Abstraction

Every wave since virtualization was described as unifying and produced new specialization anyway. Why this one may be structurally different.

Before declaring a great unification, it is worth remembering how we became so fragmented in the first place. Nobody woke up one morning and decided software engineering needed more silos. We built them one useful abstraction at a time.

The pattern

Virtualization decoupled the operating system from the machine. One physical server became many logical ones, utilization improved dramatically, and provisioning collapsed from weeks to minutes. It also created a new control plane sophisticated enough to require dedicated expertise, and so the virtualization team was born, sitting between the server people and the application people, owning a layer that had not existed.

Cloud and IaaS decoupled capacity from ownership. Infrastructure became an API call and a variable cost. It also introduced a new discipline's worth of concepts: regions, availability zones, managed service boundaries, identity and access models, and a billing surface complex enough to spawn its own profession. Cloud architects arrived, and shortly after, FinOps.

DevOps and CI/CD decoupled release from ceremony. This one was explicitly a unification movement: its founding argument was that the wall between development and operations was the problem, and that shared ownership and automated delivery would dissolve it. The culture was real and the results were real.

And then the industry hired DevOps engineers. A movement premised on removing a boundary produced a job title that formalized one, because the tooling required to do continuous delivery well turned out to need specialists. This is the clearest instance of the pattern in the whole sequence, and it should temper any claim, including this report's, that a unifying force cannot fragment.

Cloud native decoupled the application from its environment. Containers, Kubernetes, and microservices delivered genuine portability and independent deployability. They also produced the most complex operational surface the industry had ever asked application teams to absorb: an orchestrator with hundreds of resource types, a service mesh, a distributed data layer, and a supply chain with entirely new failure modes. Kubernetes specialists became a distinct labor market.

Platform engineering decoupled the developer from the substrate. It emerged as a direct response to the previous wave's cost: "you build it, you run it" was correct in principle and punishing in practice once running it meant absorbing all of the above. Golden paths, internal developer platforms, and self-service

infrastructure were the answer, and they worked.

They also produced platform teams: another specialization, another discipline, another set of conferences. The wave that existed to reduce cognitive load created a new group of people to hold it.

Why fragmentation kept happening

Each wave abstracted a layer without changing what the layers were for.

Every one of them made existing work easier: deploy faster, scale further, release more often, provision without a ticket. None of them asked the stack to do something it had not fundamentally been doing, which was running applications that receive requests, process data, and return responses.

So the abstraction was absorbed, and the complexity it introduced was managed the only way organizations know how to manage complexity: by assigning it to someone. Specialization outran abstraction because specialization is what happens when a new layer arrives and nobody's existing job description covers it.

There is a second reason, less often stated. Each wave's specialists had a rational interest in the boundary. A discipline with its own tooling, vocabulary, certification path, and conference circuit is a career, and careers are sticky. This is not cynicism: it is why "DevOps engineer" became a job title despite DevOps being a movement against exactly that kind of role definition.

Why this wave is different

AI applies pressure from two directions at once, and that is the structural difference.

AI is a workload the stack must run. It has requirements no previous workload had: accelerator-aware scheduling where the resource is scarce, non-fungible, and expensive enough that idle time shows up on the income statement. Model-aware routing where a request's cost varies by an order of magnitude. Versioned artifacts controlled by a vendor who can deprecate them. Behavior that drifts without any change to your code. Statistical evaluation instead of deterministic assertion. Identity for actors that are not human.

AI is also a worker inside the stack. It writes the code, reviews the pull request, triages the incident, generates the manifest and increasingly operates the platform. Between 65% and 70% of respondents in every role surveyed, including engineering management, QA, operations, product and platform, now use AI weekly.

No previous wave did both. Virtualization was infrastructure. Cloud was infrastructure. Containers were infrastructure. DevOps was practice. Platform engineering was practice. AI is simultaneously the thing being deployed and the thing doing the deploying, which means no discipline can hold itself apart from it and no discipline can own it alone.

That dual position produces the mechanism this report describes. An AI workload needs scheduling, routing, delivery, verification, identity, and cost management simultaneously, and the moment a single

workload requires all of those at once, the disciplines that own them individually have to work as one system.

Why this time may still not be different

Every wave in this sequence was described at the time as unifying. Cloud was going to eliminate the distinction between infrastructure and application. DevOps was going to dissolve the wall between dev and ops. Kubernetes was going to make the substrate irrelevant. Each claim was partially true and each produced new specialization anyway.

A reader would be right to ask why this time is different, and "AI is bigger" is not an answer.

My answer is not that AI is bigger. We have heard that about every technology cycle worth selling. The difference is the position AI occupies. Earlier waves changed a layer of the stack. AI reaches across the layers because it is both the software being operated and an operator of the software around it.

That distinction does not guarantee unification. It does make the proposition testable. If AI produces a separate estate, if adoption remains trapped inside role-specific copilots or if proprietary control planes close off the shared operating model, the thesis fails. For now, the evidence is moving the other way.

SECTION 3 OF 13

The Cloud Native Stack IS the AI Stack

AI did not build a parallel infrastructure. It moved into the one already built for microservices, and the institutions have ratified the merger.

The clearest signal that AI is landing inside the existing stack is Kubernetes adoption itself. CNCF's 2025 Annual Survey puts 82% of container users running Kubernetes in production, up from 66% two years earlier. More usefully, 66% of organizations hosting generative AI models are using Kubernetes to manage at least some of their inference. CNCF has taken to calling Kubernetes the operating system for AI, which is the kind of thing a foundation says about its own project, except that here the survey data backs it up.

The stack we built for microservices turned out to be the stack AI needed, which was nobody's plan.

Why the patterns transferred

Strip away the novelty and AI workloads decompose into problems Kubernetes already solved.

An inference endpoint is a long-lived process holding expensive state in memory, sitting behind a load balancer, scaling against demand, versioned on deploy. That is stateful serving, and Kubernetes has handled it since StatefulSets. The weights are bigger and the warm-up is slower, but the shape is the same.

Agents are workloads. An agent runs, calls other services, holds state for the length of a task, and exits. It needs an identity, a network policy, a resource budget, somewhere to log. Every one of those has an existing controller.

Vector databases landed in the same category as every other stateful data service that arrived on Kubernetes over the past five years: operator-managed, backed by persistent volumes, with the usual backup and restore story.

GitOps governs model deployment the way it governs everything else. A model version referenced in a Git-tracked manifest, reconciled by a controller, rolled forward and back on the same machinery you already use for application releases. The artifact changed and the control loop didn't.

The shorthand I'd offer is that inference is the new web traffic. Requests arrive, they need routing and rate limiting and caching, they spike without warning, and somebody has to watch all of it. Platform teams have solved this before. What changed is the unit, tokens and GPU-hours instead of requests and CPU-seconds, and the price per unit, which is high enough now that inefficiency shows up in the quarterly numbers rather than in a capacity review nobody reads.

The industry has already ratified this

Arguments about convergence are cheap, so watch what the institutions did instead.

In November 2025 CNCF launched the Certified Kubernetes AI Conformance Program, applying the same community-defined conformance model that produced over 100 certified Kubernetes distributions. By KubeCon EU in March 2026 the certified list had gone from 18 platforms to 31, and the program had expanded to cover agentic workloads and require alignment with Kubernetes v1.35, including stable in-place pod resizing so an inference model can have its resources adjusted without a restart.

Look at who is on that list. Amazon EKS, Google GKE, Azure, Oracle Cloud Infrastructure, VMware vSphere Kubernetes Service, CoreWeave, Red Hat OpenShift, Akamai. Three hyperscalers, a hypervisor vendor, a GPU neocloud, an on-premises distribution, all certifying against one standard for one workload class.

Section 8 makes the full argument, but the short version is already visible here. The industry did not converge on a place to run things. It converged on a way to run them, and the place became negotiable.

What AI actually changed

There is real novelty here and the argument gets weaker if it pretends otherwise. Three things don't reduce to prior art.

Output is nondeterministic. The same input will not reliably produce the same result, which breaks the assumption sitting underneath most of the testing discipline. It is why evaluation pipelines are turning into a CI/CD stage rather than a research activity, and Section 7 covers what that does to QA.

GPUs are not fungible the way CPU cores are. They are scarce, expensive, allocated in awkward granularities, and idle time on one is margin you never get back. A distributed training job that gets seven of the eight GPUs it asked for does not run at 87% speed. It sits there and burns money.

And a model is a dependency you don't control. It drifts without any change to your code, and the vendor can deprecate it on their own schedule. Conventional dependency management has nothing useful to say about that.

How the ecosystem responded is the part that matters. It did not build a parallel stack. It extended the one that existed, in-tree, through normal project governance, using the same APIs.

Dynamic Resource Allocation, the mechanism for requesting and managing accelerators, reached GA in the core Kubernetes API, and Kueue now tracks DRA resources in ClusterQueue quotas the same way it tracks CPU and memory. The Gateway API Inference Extension added model-aware routing, per-request criticality, and metric-driven load balancing to the Gateway API that was already there. Your existing gateway learned what a model is, and nobody had to stand up a separate ingress path for AI traffic.

These extensions also perform. A peer-reviewed evaluation of a multi-stage inference pipeline found Kueue cut total makespan by up to 15%, the Dynamic Accelerator Slicer reduced mean job completion

time by 36%, and the Gateway API Inference Extension with llm-d improved tail time-to-first-token latency by up to 90% under load. Those are measurements of cloud native primitives doing AI work well, not vendor benchmarks.

The gap between ambition and infrastructure

One caution before the argument moves on, because it shapes everything that follows.

The same CNCF survey that produced the 82% figure names a gap between AI ambition and infrastructure reality as one of its three headline findings. Only 7% of organizations deploy AI models daily. More than half do not train models at all. They are running pre-trained models and mostly care about doing it reliably without spending a fortune.

That sharpens the point rather than undercutting it. If most enterprises are operating models somebody else trained, then enterprise AI is overwhelmingly an operations problem: serving, routing, scaling, securing, observing, and paying for inference. Not an ML research agenda, a platform engineering one. Which is why the disciplines in this report are being pulled together instead of further apart.

The stack didn't have to change identity to host AI. AI had to become an operations problem to fit the stack, and it did, faster than most people expected.

SECTION 4 OF 13

The Stack Is Changing to Meet AI

Scheduling, traffic, lifecycle, interop and observability are being remodeled inside the existing projects — not replaced by AI-only equivalents.

AI may have moved into the cloud native stack, but it did not move in quietly. The stack is being remodeled around the economics and operational demands of models and agents. Watch where that work is happening and you can see the industry's real priorities more clearly than in any product roadmap.

Scheduling: from placement to economics

Kubernetes scheduling was designed around a resource that is abundant, fungible, and cheap to waste. A CPU core is interchangeable with the core next to it. If a pod lands on a slightly suboptimal node, you lose a few percent of efficiency and nobody notices.

None of this is true of accelerators. GPUs are scarce, non-fungible across generations, allocated in awkward granularities, and expensive enough that idle time is visible in the quarterly numbers. Worse, distributed training needs many GPUs simultaneously or none usefully at all: a job that gets seven of the eight GPUs it needs does not run at 87% speed. It sits and burns money.

The stack has responded with a scheduling layer far more sophisticated than anything the microservices era required:

- Dynamic Resource Allocation replaced the device-plugin model with a proper API for requesting hardware, now integrated into Kueue's quota accounting, where DRA resources are tracked in ClusterQueue quotas exactly like CPU or memory.
- Queue and quota systems: Kueue for admission control and quota enforcement and Volcano for gang scheduling and fair-share frequently run together as layered controls rather than alternatives. Kueue's borrowing model lets teams draw on unused quota from sibling queues up to a configurable ceiling.
- Topology-aware placement matters again, because interconnect bandwidth between GPUs determines whether a distributed job is fast or merely expensive.
- Vendor-neutral abstraction is emerging: HAMi, a CNCF Sandbox project, surfaced at KubeCon EU 2026 as a reference implementation for GPU resource management spanning NVIDIA, AMD, Huawei, and Cambricon accelerators.

There is a bit of irony here. Batch scheduling, dismissed by much of the cloud native world as legacy HPC baggage, is back at the center of the platform. The reason is money. When the resource is this expensive, a queue is a financial control. Somebody has to own it.

Traffic: the gateway learns what a model is

An inference request is not a web request. It is long-lived, streamed and wildly variable in cost depending on context length. It is also cacheable in ways HTTP never anticipated, because two semantically similar prompts may warrant the same answer.

Round-robin load balancing across model replicas is actively wrong: it ignores KV-cache locality, so a request routed to a cold replica pays a latency penalty that a cache-aware router would have avoided entirely.

The Gateway API Inference Extension addresses this inside the existing Gateway API rather than beside it, adding model-aware routing, per-request criticality, and load balancing driven by real-time model metrics, with KV-cache-aware scheduling and LoRA adapter affinity at GA. Paired with llm-d, it improved tail time-to-first-token latency by up to 90% under high load.

Note what did not happen. There is no separate AI ingress path, no parallel service mesh for model traffic. The gateway you already run learned what a model is.

Lifecycle: evaluation becomes a pipeline stage

This is the layer where the remodeling is least finished and most consequential.

A model is a dependency you do not control, that drifts without any change to your code, and that a vendor can deprecate on their schedule. Conventional dependency management has no answer for this. What is emerging in response:

- Model registries as first-class artifacts alongside container registries.
- Prompt and agent versioning, because a prompt is production configuration with the blast radius of code and, until recently, none of the review discipline.
- Evaluation pipelines as the emerging test stage: the deepest change of the three, and the least settled. Deterministic pass/fail testing does not describe a system whose output varies run to run. What replaces it is statistical: a scored evaluation suite with thresholds.

In practice, the tooling has arrived well ahead of the discipline. LangChain's survey of more than 1,300 practitioners found 57% with agents in production and nearly 89% having implemented observability, while eval adoption trailed at 52%. Just over half were running offline evaluations against test sets, and 32% named quality as a top barrier to production.

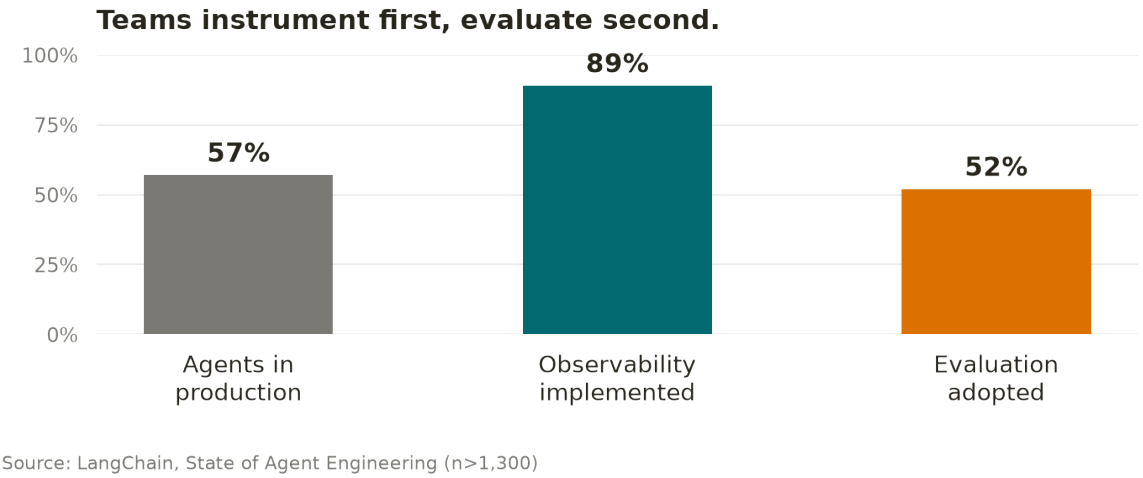


Figure 3 — The instrument-first pattern. Observability outran evaluation, reversing the order in which the discipline matured for conventional software.

That ordering is diagnostic. Teams instrument first and evaluate second, which is the reverse of how the discipline matured for conventional software, where testing long predated observability. It means most organizations can currently tell you that an agent behaved badly in production but cannot tell you, before merge, whether a prompt change made things worse.

So the claim to make here is a directional one, not an accomplished one: evals are becoming a CI/CD stage, and CI/CD is DevOps territory. Where that transition completes, the trust layer gets built into the delivery pipeline rather than bolted alongside it, and that is the mechanism by which QA and DevOps stop being separable disciplines. But roughly half the field has not made the move, and "offline evaluation against a test set" is not yet the same thing as a gate that blocks a release. Section 7 treats the organizational consequences; Section 10 returns to the gap itself as an open question.

Interop: MCP is the fastest-moving standard in the stack

The Model Context Protocol did not exist before late 2024. As of the first half of 2026, 20% of enterprises report actively running MCP in production, with a further 26.9% building pilot implementations [7]. Nearly half have moved beyond evaluation into implementation for a protocol roughly eighteen months old. Compared with service mesh adoption, OpenTelemetry's climb or the container transition itself, that is extraordinary velocity.

Why it matters structurally: MCP is the connective tissue that lets an agent reach across tool boundaries that map directly onto organizational boundaries. An agent that can query the issue tracker, read the observability backend, open a pull request, and check the deployment status is an agent operating across four tools historically owned by four different teams. The protocol does not respect the org chart. Futurum's own analyst guidance to vendors makes the interoperability bet explicit: use emerging open standards such as MCP and Agent-to-Agent to reduce lock-in and solve problems across multiple platforms [2].

This is where an architectural argument becomes operational. The connective tissue is not a manifesto. It is a wire protocol.

Observability: new signals, same discipline

The familiar golden signals of latency, traffic, errors and saturation still apply. They are no longer sufficient.

Four signals join them as first-class concerns: token consumption (the unit of cost), quality (did the output meet the bar, which is an eval question asked in production), cost per request (which now varies by an order of magnitude depending on routing decisions), and model attribution (which model, version, and prompt produced this).

Two things follow. First, this is where FinOps stops being a cloud-billing discipline and becomes an engineering one: inference cost optimization through quantization and distillation is now the single highest-ranked infrastructure priority among enterprise decision makers, at 56.7% [6]. Second, a purpose-built tooling category has formed: Futurum now tracks AI Agent Observability as a distinct submarket with named vendors competing for share [4], alongside a separate Agent Control Plane and Agentic Governance submarket [3]. Two years ago neither existed.

Where this is consolidating

A pattern runs through these changes. The ecosystem extended existing projects: DRA in the scheduler, inference routing in the Gateway API, agent identity in existing conformance and sandbox models. That is a meaningful choice. New workload classes usually attract new stacks. AI is bending this one without breaking away from it.

The counter-pressure is concentrated in AI's most valuable new layers, including gateways, control planes, evaluation and agent observability. Commercial vendors are moving fastest there, while the open standards are youngest. Whether these become portable primitives or proprietary control points remains unresolved. Section 10 returns to this as the most credible threat to the unification thesis.

SECTION 5 OF 13

Platform Engineering 2.0: The AI Platform Is the Platform

The platform team's product now includes inference endpoints, model catalogs, agent runtimes and governance embedded in the gateway developers already call.

What 1.0 was for

Platform engineering arrived when the industry admitted that "you build it, you run it" had become too literal. The idea was sound. The implementation asked every product team to solve Kubernetes, networking, secrets, observability, compliance and cost for itself. That was never going to scale.

The response was to treat internal infrastructure as a product. Golden paths offered paved, opinionated routes to production that were easier to follow than to circumvent. Internal developer platforms turned tickets into self-service APIs, with Backstage emerging as the canonical catalog. The measure of success was straightforward: how long did it take to move from a developer's intent to a running service, and how much of the underlying complexity did the developer have to understand?

That work is not finished. The mandate has simply grown.

The 2.0 shift: the platform's product is now AI capability

The question a platform team gets asked in 2026 is no longer only "how do I deploy a service." It is, which models are we allowed to use, how do I get an inference endpoint, who is paying for these tokens, what happens when the vendor deprecates the model I built on, can my agent have credentials to the production database, and who approved that.

Every one of those is a platform question. Answering them well produces a recognizable set of platform products:

- Self-service inference endpoints, so teams get model access without each one negotiating its own vendor contract and key management.
- Model catalogs: the approved set, with cost, latency, data-residency, and licensing characteristics attached, which is a golden path by another name.
- Agent runtimes with identity, scoped permissions, network policy, and audit built in.
- GPU quota and FinOps, because accelerator scarcity makes allocation a governance function rather than a billing report. Inference cost optimization through quantization and distillation is now the single highest-ranked infrastructure priority among enterprise decision makers, at 56.7% [6].
- AI gateways with governance baked in: the routing layer from Section 4, doing policy enforcement as a side effect of being the path traffic already takes.

The pattern that made platform engineering work in 1.0 applies unchanged: governance that lives on the paved road gets followed, and governance that lives in a policy document does not.

The governance gap

The number I keep coming back to is 18.1%. That is the share of enterprise software lifecycle decision makers who place AI agent governance at the standardizing or mastering stage, making it the least mature practice measured. DevSecOps is at 46.1%, FinOps at 48.3% and DevOps at 58.3% [1].

Set that against what the same population reports about AI in production. More than 84.5% say AI is involved in over a quarter of their SDLC work, 54.7% say more than half, and nearly 40% report that AI generates over half their production code [1]. The last figure is a decision-maker estimate, not repository telemetry. More than a third, 37.8%, are already operating agents in supervised, semi-autonomous or fully autonomous modes [1].

AI has been adopted far faster than it has been governed. That is not a warning about a future estate. It is a fair description of the one many companies are running today.

Identity Becomes the New Perimeter

The consequence is not hypothetical either. Okta's research found 91% of organizations already using AI agents while only 10% had a well-developed non-human identity strategy, and a Cloud Security Alliance survey found 68% of organizations cannot reliably distinguish AI agent activity from human activity in their environments. There is also a clear perception split by altitude: 65% of executives say their AI usage policies are very clear, while only 43% of knowledge workers agree.

That last figure captures the shadow AI problem. Leadership believes the guardrails exist; the people doing the work cannot find them. Developers respond to that gap the way they always have when the sanctioned path is unclear: they use whatever works, with whatever credentials they have.

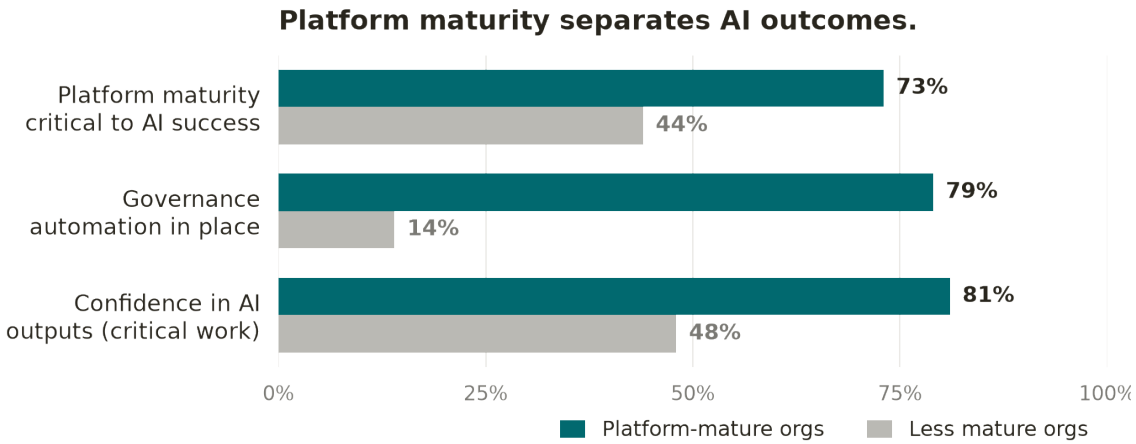
The platform team is in the best position to close that gap without bringing delivery to a halt. Policy attestations and extra review boards are easy to route around when the unsanctioned alternative is one browser tab away. Put the governance in the endpoint, gateway and runtime already on the paved road and it becomes part of the work instead of an interruption to it.

Does platform maturity actually predict AI success?

The connection between platform maturity and AI outcomes is central to this report, and there is direct evidence for it.

Perforce's 2026 State of DevOps research on platform engineering found that organizations with mature platform practices report dramatically different AI outcomes than those without: 73% of platform-mature organizations said platform maturity was a critical or significant factor in their AI success, against 44% of less mature ones. On governance automation maturity the gap is starker still: 79% versus 14%. Confidence in AI outputs within critical workflows runs at 81% for platform-mature organizations against 48% for others, rising to 92% among those with fully standardized internal

developer platforms.



Source: Perforce, State of DevOps: Platform Engineering 2026 (n=820) [5]

Figure 4 — Platform-mature organizations report better AI outcomes across three independent measures. The direction of causation is discussed in Section 10.

DORA's research arrives at the same conclusion from a different direction, and its framing is the most useful sentence available on the subject: AI functions as an amplifier, magnifying the existing strengths of high-performing organizations and the existing dysfunctions of struggling ones. DORA's finding is that the greatest return comes not from the AI tools themselves but from the quality of internal platforms, the clarity of workflows, and the alignment of teams.

Both findings are correlational. Organizations that are good at platform engineering may simply be good at adopting technology in general. Still, the direction is consistent across independent datasets, and the practical point survives the caveat. Leadership can improve the platform. It cannot buy its way to sound AI operations by choosing a more fashionable model.

Reconciling the maturity numbers

A reader tracking the figures will notice an apparent contradiction. Platform engineering sits at 36.7% maturity [1]: meaningfully behind DevOps at 58.3%, while adoption of platform practices is described elsewhere as near-universal, and while the Perforce data suggests platform maturity is decisive for AI success.

These are not in conflict. Adoption and maturity are different measures: most organizations have a platform, and roughly a third have a good one. The supporting evidence for this reading is unusually direct: the 2026 State of Platform Engineering survey found 55.9% of companies already operating more than one internal developer platform, segmented by team, with frontend, backend, data, and AI each getting their own. That is a field that has adopted the model broadly and is now dealing with its second-order problems.

Two further findings sharpen the picture. That same survey reports nearly 30% of platform teams do not measure success at all: a straightforward maturity deficit in a discipline whose founding premise was treating the platform as a product with users. And it published a reference architecture specifically for AI/ML internal developer platforms, which is the clearest possible signal that the AI platform is being understood as a platform engineering deliverable rather than a separate ML infrastructure concern.

Platforms built with AI, not only for AI

The second half of the 2.0 shift is that platform teams consume this technology as well as provide it. Two-thirds of organizations now use AI in infrastructure workflows, but only 31% report fully autonomous AI there. AIOps, oversold for a decade, is becoming real in a narrower and more useful form through agents that triage, correlate, propose remediations and generate golden-path scaffolding.

The 66%-versus-31% split is worth reading carefully. Platform teams are adopting AI enthusiastically for their own work while keeping humans in the loop on execution, which is a reasonable posture for the team whose blast radius is every other team.

When your users include agents

The founding metaphor of platform engineering was "platform as product," with developers as the customers. That metaphor now has to absorb a new customer class: the agents themselves.

Agents consume platforms differently than humans do. They do not read documentation. They read schemas and tool definitions, which is what MCP standardizes and part of why its adoption moved so quickly. They do not exercise judgment about whether an action is reasonable; they need constraints enforced rather than advised. They operate at machine speed, so a permission that is merely broad becomes a permission that is exhaustively exercised. They also need identity, because an audit trail that ends at "the service account the agent used" answers nothing.

Designing the platform for agents makes it better for humans too: explicit schemas, enforced constraints, scoped identity, and auditable actions are things human developers have always deserved and rarely been given. But it is a genuine change in requirements, and platform teams that treat agents as just another workload will discover the difference in an incident review.

SECTION 6 OF 13

From Coder to Software Engineer: The Evolution of the Developer

Code generation is getting cheaper. Specification, decomposition and verification are becoming the scarce skills.

The direction of travel

For engineers working seriously with agents, the job has already changed. The rest of the field is behind that leading edge, not exempt from it.

The work is moving toward specifying intent, decomposing problems, directing generation and verifying results. Code writing is the part being compressed. Where that shift has taken hold, the job feels less like an enhanced version of coding and more like a different way of engineering software under the same title.

The distribution matters because it keeps the argument honest. Individual developer assistance, mostly IDE completion and chat, remains the dominant mode at 47.2% of organizations. Supervised agents account for 18.4%, semi-autonomous agents with checkpoint approval for 13.6%, and fully autonomous agents with outcome-only review for 5.8% [1].

Roughly 37.8% of organizations are operating agents in some mode. Fewer than one in sixteen have reached outcome-only review.

It would be easy to look at those numbers and dismiss agentic engineering as a minority practice. I think that would be a serious misread of the direction of travel.

Consider what these numbers describe. A capability that did not exist in production three years ago is now operating in agentic form at more than a third of enterprises. Nearly 40% of organizations report that more than half their production code is AI-generated [1]. Weekly use across engineering management, QA, operations, product and platform roles falls within a narrow 65% to 70% band [1]. AI has landed. Organizations are simply absorbing it at very different rates, consistent with DORA's finding that AI amplifies the environment it enters.

My read is that the 5.8% at outcome-only review are early, not irrelevant. The 47.2% still in individual-assistance mode have not yet built the platform, governance and verification capacity to go further. Tool access is not the constraint. Organizational readiness is, and readiness can be built.

Engineers who define their value by the volume of code they produce are optimizing for the part of the job that just became abundant. The median organization does not have to be fully agentic for that career warning to be valid.

The arc, without the ladder

The outline framed this as a progression: coder › developer › software engineer › engineering director of AI. I want to push back on the last step.

Every previous transition in that sequence traded keystrokes for judgment while keeping the practitioner inside the work. A software engineer who stops touching the material becomes a manager, and we already have a word for that. "Engineering director of AI" describes a role that mostly does not exist, and framing it as the destination invites the objection that this report is doing career-ladder fan fiction.

What the evidence actually shows is not a ladder but a bifurcation of the same job. The work is splitting into two activities that were previously fused:

- Specification and decomposition: deciding what should exist, breaking it into pieces an agent can execute, and establishing what "correct" means before anything is generated.
- Verification and judgment: determining whether what came back is right, safe, maintainable, and actually what was needed.

Generation, the part that used to consume most of the day, is compressing between them. That is the transformation. It does not require a new job title to be significant.

The verification tax

DORA describes AI as an amplifier of both strength and dysfunction. At the individual level, that amplification moves the bottleneck rather than eliminating it. Generation gets cheap. Verification does not.

The telemetry, covered in Section 10, is unambiguous. Faros AI's analysis across roughly 22,000 developers shows PRs growing sharply larger and developers juggling far more PR contexts daily than they did a year ago. The vendor calls the pattern Acceleration Whiplash: throughput gains at the point of generation, and compounding quality costs at every downstream stage.

Read that as a description of a workday. More code arrives, in larger chunks and across more simultaneous contexts, for the same human to understand. The scarce resource was never typing. It was the attention required to know whether something is right. AI has made everything upstream of that attention cheaper while leaving the attention itself just as expensive.

There is an encouraging counter-signal. AI adoption in code review runs at 37.7% of organizations, nearly level with code generation at 40.2% [1]. The verification side is not being ignored. Whether automated review actually scales human judgment or merely produces confident approval at higher volume remains one of the report's most important open questions.

What "managing the coding" actually requires

Treating AI direction as a discipline rather than a knack means naming what it consists of.

Specification quality. A prompt that produces good code is a specification with context, constraints and success criteria. It is requirements analysis applied at a granularity where that work was previously uneconomic.

Decomposition. Agents fail on scope. The judgment of how large a unit of work can be before an agent loses coherence is new, empirical, and currently transmitted by folklore.

Review at AI scale. When PRs are 50–150% larger and arrive faster, line-by-line review stops scaling. Property-based checks, invariant testing, architectural conformance and targeted review of high-risk diffs may form part of the answer. A complete methodology has not emerged.

Knowing when not to delegate. Work that is novel, or so load-bearing that a mistake would be expensive, still warrants human authorship. Recognizing that work requires judgment before any prompt is written.

What gets more valuable

The fundamentals appreciate, and they appreciate for a specific reason: they are the faculties used to evaluate output rather than produce it.

Architecture and systems thinking, because agents optimize locally and cannot see the shape of the whole. Debugging, which is the purest verification skill and gets harder when you did not write the code. Domain knowledge, because an agent cannot know that this field is nullable for a regulatory reason. And taste: the fast, largely tacit judgment that something is wrong before you can articulate why, which is the least teachable of these and, awkwardly for the profession, the one most reliably built by writing a great deal of code yourself.

The junior problem

Which leads directly to the thing this report should be honest about not knowing.

The traditional path to senior judgment ran through years of writing the code that AI now generates. If juniors no longer do that work, the mechanism by which they become seniors is unclear. The pipeline argument says this is a crisis in slow motion. The counter-argument says every abstraction wave provoked the same fear: compilers, garbage collection, Stack Overflow, frameworks, and each time the profession adapted its training rather than collapsing.

There is one datapoint worth putting on the table, and it is uncomfortable. Among decision makers, increasing investment in generative AI (40%) and AI/ML technologies (39%) now outrank increasing hiring of IT personnel (23%) as the most critical actions for accelerating software delivery [5]. Capacity is being bought as capability rather than headcount.

That is a budget-priority question, not a headcount measurement, and it does not prove juniors are being displaced. But it establishes that the substitution is being contemplated at the planning level, which is where displacement starts.

What we can say now is that the mechanism that produced senior engineers has been disrupted, and nothing has been designed to replace it. The historical optimists may eventually be proved right. But in every prior wave, adaptation required deliberate work. Curricula changed, apprenticeship models changed and the profession reconsidered its fundamentals. That work has barely started for this transition.

Organizations that assume the pipeline will regenerate itself are making an assumption no previous generation of engineering leadership got away with. The open question is not whether juniors can still become seniors. It is who is going to build the path, and right now the answer is nobody in particular.

What "10x" means now

The 10x engineer was always partly mythology, but the useful residue was that leverage came from judgment: knowing which thing to build, which abstraction would hold, which problem not to solve.

When everyone has agents, generation converges toward commodity. Leverage relocates to the parts agents cannot do: knowing what should be built, recognizing when output is subtly wrong, and holding the architectural line while many things are produced quickly. Those were always what separated strong engineers. AI made them the only thing that separates them.

The developer and the platform

There is a finding in the data that connects this section directly to Section 5, and it is easy to miss.

Between 65% and 70% of respondents in every role surveyed, from engineering managers and QA to operations, product and platform engineering, use AI weekly [1]. Yet 15.0% of organizations report no structured AI use [1].

Both figures can describe the same reality only if individual adoption is running well ahead of organizational structure. That is the shadow AI condition from Section 5 seen from the other end. Engineers are using these tools whether or not their organization has decided how.

This is why the platform matters to the individual engineer, not just to leadership. The internal developer platform is where AI-assisted engineering becomes governed, repeatable, and safe: approved models, scoped agent identity, cost visibility, eval and review stages in the pipeline they already use. Without it, every engineer improvises their own AI practice, and the organization inherits the aggregate risk of thousands of individual improvisations.

The evolution of the developer and the evolution of the platform are the same story told from two desks.

SECTION 7 OF 13

The Day-to-Day, Role by Role

What actually changes on Monday morning for DevOps, platform engineering, QA and security.

Architecture arguments eventually land on somebody's desk. Across DevOps, platform engineering, QA and security, the work is moving up a layer. People spend less time executing every step and more time deciding what the system may do, what it must prove and where human judgment stays in charge. That creates more leverage, but it also gives practitioners less direct control over the output. Anyone who has operated production systems knows that is not an uncomplicated trade.

Drawing the DevOps and platform engineering line

Platform engineering builds and operates the platform. DevOps owns the delivery process that runs on top of it.

Platform engineering's product is capability: the paved paths, the runtime, the inference endpoints, the quota systems, the golden-path scaffolding. Its customers are engineering teams, and increasingly agents. Its measure is whether teams can move without holding the whole substrate in their heads.

DevOps's product is flow: the pipeline, the release process, the feedback loops, and the culture that holds them together. Its measure is whether change reaches production quickly, safely, and reversibly.

The two get conflated because for a decade the same people did both, and because a pipeline is itself a piece of infrastructure. The distinction sharpens under AI. When the platform provides inference endpoints, model catalogs, agent runtimes, and GPU quota as products, the DevOps engineer stops building that infrastructure and starts operating a delivery process that consumes it, and gains new pipeline stages, evaluation and guardrails among them, that are process concerns rather than platform ones.

Where organizations get this wrong, they get it wrong in a predictable direction: the platform team ends up owning the pipeline, becomes a ticket queue for release engineering, and never builds the capability layer that would have made them valuable. The AI transition punishes that error faster than the pre-AI world did, because the capability layer is now where the hard problems are.

DevOps engineers: the pipeline becomes the product

DevOps was always culture and process before it was tooling, and the AI transition returns it to that. The pipeline gains stages that did not exist three years ago:

- Evaluation gates, replacing or supplementing deterministic tests for nondeterministic components: Section 4 established that this transition is directional rather than complete, with observability adoption at roughly 89% against evals at 52%.

- Guardrail checks: prompt-injection screening, output policy enforcement, data-egress controls.
- Model deployment stages, with their own promotion, rollback, and canary semantics. A model rollback is not a code rollback: the artifact is large, the warm-up is slow, and behavioral regression may only appear statistically.
- Provenance and attestation for AI-generated code, which matters more when nearly 40% of organizations report over half their production code is AI-generated [1].

AIOps also becomes real here, in a narrower form than it was sold. AI adoption in incident response sits at 16.2% and in observability at 21.0% [1]. Those figures are modest, but correlation and triage can reduce real toil in both phases.

Two numbers should worry this audience. AI adoption in CI/CD operations is 13.2%, and in deployment decisions just 6.2%, the lowest of any lifecycle phase measured [1]. Meanwhile only 9.5% of organizations release daily, despite 58.3% claiming DevOps maturity at standardizing or mastering [1].

The generous reading of the deployment number is that humans are correctly retained at the highest-consequence boundary. The uncomfortable reading is that generation has accelerated while the release process has not, which means AI has increased the pressure on a constraint that was already binding. Both readings can be true, and the delivery process is where they collide.

The role's center of gravity moves from building and maintaining pipeline infrastructure, increasingly provided by the platform, to operating and evolving the delivery process itself. The work is deciding what gates exist, what they block, what evidence a release must carry and how fast the loop closes. That is a judgment job, and it is a larger one than what it replaced.

Platform engineers: from infrastructure to capability brokerage

Section 5 covered the platform's new product line. What changes on the desk:

The skill set shifts toward economics. GPU quota, token cost attribution, and the quantization-versus-quality tradeoff are now platform concerns: inference cost optimization ranks as the highest-priority infrastructure item among enterprise decision makers at 56.7% [6]. Platform engineers are becoming the people who know what things cost, which is a materially different job from knowing how things run.

Governance becomes a build target. The 18.1% AI agent governance maturity figure [1] is a platform engineering backlog item. Every control that exists as a policy document rather than an enforced default is a control that will be bypassed.

They are heavy AI users themselves. 65.7% of platform engineers use AI weekly [1], and 66% of organizations use AI in infrastructure workflows: though only 31% report fully autonomous AI there. Keeping humans on execution is a defensible posture for the team whose blast radius is everyone else.

The customer list now includes agents. Covered in Section 5; the operational consequence is that platform interfaces need machine-readable contracts and enforced constraints rather than

documentation and conventions.

QA: verification becomes the scarce skill

QA has spent two decades being told it was going to be automated away. The AI transition inverts that argument completely, and this is the role whose stock rises most.

The evidence is already visible in adoption: QA engineers report 68.8% weekly AI use, second-highest of any role measured and effectively level with engineering managers at 69.6% [1]. AI in testing runs at 28.0%, third among lifecycle phases behind code generation and code review [1]. This is not a function being displaced. It is a function reaching for the tooling first.

The reason is structural. Section 6 described the verification tax: generation got cheap and verification did not, and the Faros telemetry shows the review load rising steeply while the humans doing the reviewing did not multiply. When output volume rises and confidence in any individual output falls, the discipline that determines whether things are correct becomes the constraint on the entire system.

What the work becomes:

- Testing nondeterministic systems, where pass/fail gives way to scored thresholds, statistical confidence and acceptable-variance bands. This methodology is closer to experimental design than conventional test scripting.
- Owning the eval suite as a first-class engineering artifact: versioned, reviewed, and maintained with the same discipline as the code it gates. In most organizations nobody currently owns this, which is an opportunity for whoever claims it.
- Review methodology at AI scale, the open problem from Section 6. Line-by-line review does not survive the volume increase. Property-based checks, invariant testing, architectural conformance, and risk-targeted deep review are the plausible replacements, and QA is where that methodology should be developed.

The tooling for evaluating nondeterministic systems is immature, and the practice is uneven. "QA owns evals" is still a recommendation rather than an observation. But the demand is clear, and developers grading their own agents' homework at volume is not a system anyone should want.

Security: from perimeter to identity

Security absorbs the largest scope expansion of any role, in two directions at once: securing AI, and using AI to secure everything else.

The threat assessment is not subtle. A 2026 Dark Reading poll found 48% of security professionals ranking agentic AI as the top attack vector for the year. The exposure is structural rather than incidental: an agent is a workload with credentials, network access, and the ability to be talked into things by data it reads.

The governance data is worse than the threat data. Okta found 91% of organizations already using AI agents while only 10% had a well-developed non-human identity strategy; 81% of CISOs worry about

excessive AI access, and just 31% feel fully aligned with their board on acceptable AI risk. The Cloud Security Alliance found 68% of organizations cannot reliably distinguish AI agent activity from human activity in their environments.

That last figure is the whole problem compressed into one number. An audit trail that cannot separate agent from human is not an audit trail.

> **"An audit trail that cannot separate agent from human is not an audit trail."**
>
> §7 The Day-to-Day, Role by Role

Meanwhile AI adoption for security scanning sits at just 12.4% [1], among the lowest of any lifecycle phase and lower than documentation. Security is being asked to secure a rapidly expanding AI surface while being one of the slowest functions to adopt AI for its own work. That is not a criticism of security teams; it reflects appropriate caution about automated tooling in a domain where false confidence is expensive. But the asymmetry is real and widening.

The structural change is that DevSecOps absorbs MLSecOps. Model supply chain, prompt injection, agent permissions and data-egress control do not require a separate organizational silo. They extend the existing DevSecOps mandate to new artifact types. DevSecOps maturity at 46.1% [1] provides a stronger foundation than AI agent governance at 18.1%.

The perimeter concept changes with it. Agent identity and least privilege replace network boundaries as the primary control, because an agent's blast radius is defined by its credentials rather than its location. The market has noticed: Cisco announced its intent to acquire Astrix in May 2026, and SailPoint acquired Entro Security in June 2026. Both specialize in non-human identity.

What the four have in common

Each role moves from doing to governing the doing. The DevOps engineer designs which gates exist rather than building the pipeline that runs them. The platform engineer brokers capability rather than provisioning infrastructure. QA designs what correctness means rather than executing checks. Security governs identity rather than defending a boundary.

Seen from these four desks, unification does not look like a merger of roles. The specialties remain real and deep. What converges is the kind of work each role does, on the same substrate and increasingly through the same platform. Each role spends more time defining intent, applying judgment and verifying automated work.

SECTION 8 OF 13

Where It All Runs: The Pluralist Infrastructure Reality

Infrastructure is becoming more plural, not less. Hybrid is the destination, not the transition — and one operating model now spans every substrate.

Unification does not mean everything runs in one place. If it did, the argument would collapse under the weight of sovereignty rules, GPU shortages, data gravity and cloud pricing.

Mainframes unified by centralizing. Cloud promised a different kind of centralization in somebody else's data center. Both models ran into the same reality: workloads diverge, and no location remains the best answer for all of them.

The operating model is now converging while the underlying infrastructure becomes more varied. That is what makes this version of unification durable. The common layer can survive a change in provider, location or economics because it does not depend on any one of them.

The evidence that one model spans every substrate

Section 3 introduced the Certified Kubernetes AI Conformance Program. Its participant roster is the single best piece of evidence in this report, and it belongs here as much as there.

Among the initial certified participants: Amazon EKS, Google GKE, Azure, Oracle Cloud Infrastructure, VMware vSphere Kubernetes Service, CoreWeave, Red Hat OpenShift, and Akamai. By KubeCon EU in March 2026 the certified roster had grown from 18 platforms to 31.

Read what that list contains. Three hyperscalers. A hypervisor vendor's Kubernetes service. A GPU neocloud. An on-premises distribution. An edge and CDN provider. All certifying the same workload class against the same community-defined standard, on the same release cadence.

There is no historical precedent for this. Prior portability claims were vendor marketing or standards-body aspiration. This is a conformance program with a test suite, a version requirement, and a public roster: the same mechanism that produced more than 100 certified Kubernetes distributions and made "runs on Kubernetes" a statement with content.

Hyperscalers: still dominant, no longer default

The public cloud remains the center of gravity and is not losing it. The market continues to grow at roughly 21.5% annually, because new workloads are created faster than existing ones return. Any argument that reads as "the cloud is over" is wrong and will date this report badly.

What changed is that the hyperscalers stopped being the automatic answer. The reasons are specific rather than ideological:

Data gravity. Training and fine-tuning are data-adjacent activities. When the data is large, regulated, or already sitting somewhere, moving compute to it beats moving it to compute.

Sovereignty. Nutanix's 2026 Enterprise Cloud Index found 57% of IT leaders feel the need to run infrastructure within a single country. That is not a preference, it is a constraint arriving through regulation, and it is the fastest-growing driver of substrate decisions.

GPU economics. Accelerators break the elasticity argument that justified cloud for everything else. Steady-state inference at scale on rented GPUs, running near-continuously, does not exhibit the bursty profile that makes rental cheaper than ownership. It exhibits the opposite profile: the one that has always favored owning.

Repatriation without the mythology

This is the topic where practitioner papers routinely overclaim, so the framing matters more than the number.

The claim that some overwhelming majority of CIOs plan to repatriate workloads traces to a Barclays CIO survey from late 2024 and has been recycled through low-quality secondary coverage ever since. It is stale and usually misread.

The better reading is that organizations are moving workloads selectively, and only a small single-digit share plan a full exit from public cloud. Repatriation is portfolio management, not exodus. Workloads with known steady-state profiles, predictable capacity and unfavorable cloud unit economics come back. Bursty, experimental and globally distributed workloads stay.

Broadcom claims modern private cloud delivers 40–50% lower TCO for steady-state workloads and says it saved more than $10 million by moving critical workloads off public cloud DBaaS. That is a self-interested vendor claim and should be read as one. The underlying arithmetic is less controversial: predictable load tends to favor ownership, and AI inference at scale can be highly predictable.

Bare metal and the neoclouds

The GPU cloud category barely existed at the start of this decade and is now a structural feature of the market. Neoclouds generated over $25 billion in revenue in 2025, with forecasts approaching $400 billion by 2031. CoreWeave reached roughly $5 billion in revenue, reportedly faster than any cloud provider before it, with backlog growing from $66.8 billion at the end of 2025 to $99.4 billion by March 2026.

Neoclouds matter to the unification argument because they compete on being closer to the metal, not further from it. They win on GPU density, interconnect topology and price per accelerator-hour. Architecturally, they are a return to bare metal with a modern Kubernetes control plane on top. CoreWeave appearing on the AI Conformance roster alongside the hyperscalers makes the point.

One shift worth flagging for the outlook section: the binding constraint in this market is moving from GPU supply to power. The race increasingly looks less like a scramble for accelerators and more like a fight over electricity and the sites that can deliver it.

Hypervisors: three strategies, one of which is the thesis

The Broadcom acquisition of VMware forced tens of thousands of organizations to make an infrastructure decision they had been deferring for a decade. The responses sort into three strategies, and the taxonomy is more useful than any single market-share number:

Consolidators stay on VMware, negotiate, and standardize on VMware Cloud Foundation. This remains the majority behavior for regulated core production. ING and Barclays have both publicly committed to VCF 9.0 as their strategic private cloud platform.

Replacers swap the hypervisor for Nutanix, Proxmox, or a KVM-based alternative while keeping the virtualization operating model intact. Proxmox VE has grown to roughly 1.5 million hosts. Churn concentrates in SMB, edge, and discrete workload tiers rather than regulated core estates, and most estates that evaluated alternatives retained 60–80% of workloads on VMware.

Absorbers move virtual machines into Kubernetes itself, via KubeVirt and its commercial derivatives, and stop running a separate virtualization platform at all.

The third option is the one that matters for this report, and the argument does not depend on how many organizations have taken it. What matters is that it exists and works. KubeVirt is production-viable, Red Hat ships it as OpenShift Virtualization with a supported migration toolkit for vSphere estates, and enterprises have gone on the record describing the move. At Red Hat Summit in 2026, VMware customers publicly recounted migrating to OpenShift Virtualization following Broadcom's licensing changes.

An organization that absorbs VMs into Kubernetes has not chosen a hypervisor. It has decided that the container control plane is the control plane, and that virtual machines are one more workload type it schedules alongside everything else. Five years ago that was not a credible option for production estates. Now it is, which means the operating model can absorb the substrate rather than being determined by it.

Whether absorbers end up as 3% of the market or 30% is a forecast question, and Section 12 treats it as one. For the argument here, the availability of the option is the finding.

Kubernetes as the portability layer

The reason all four of these substrate models can coexist is that the layer above them is the same.

This is a stronger claim than "Kubernetes runs everywhere," which has been true and largely uninteresting for years. The interesting version is that the entire operating model now runs everywhere: the same scheduling primitives, the same GPU allocation mechanisms via DRA, the same inference routing via the Gateway API Inference Extension, the same GitOps reconciliation, the same observability

instrumentation, the same policy enforcement, the same agent runtimes and identity model.

An engineer who learned to operate AI workloads on GKE can operate them on OpenShift on bare metal, vSphere Kubernetes Service or CoreWeave. The skills transfer, the manifests mostly transfer and the platform team's golden paths transfer. Substrate choice becomes more of a procurement and economics decision and less of an architectural commitment.

Hybrid is the destination, not the transition

For fifteen years "hybrid" was a euphemism for an unfinished migration: a state organizations were passing through on the way to somewhere coherent.

That framing is now wrong, and the evidence is that the reasons for pluralism are structural rather than transitional. Sovereignty requirements are not going away. Data gravity is increasing as datasets grow. GPU economics favor ownership for steady-state inference and rental for burst, which is a permanent argument for both. Edge inference is growing because latency and bandwidth are physics rather than preferences.

None of these resolve into a single substrate under any plausible future. An organization running training on a neocloud, steady-state inference on owned GPUs in colocation, regulated workloads in a sovereign region, burst capacity on a hyperscaler, and inference at the edge is not mid-migration. It is correctly configured.

The unification thesis does not require any of that to collapse into one place. It requires one operating model to span all of it, and that model now exists, is certified against a common standard, and is running in production across every substrate class simultaneously.

SECTION 9 OF 13

Putting the Pieces Together

The layered model that emerges when infrastructure, platform, delivery, engineering and the trust layer share one substrate.

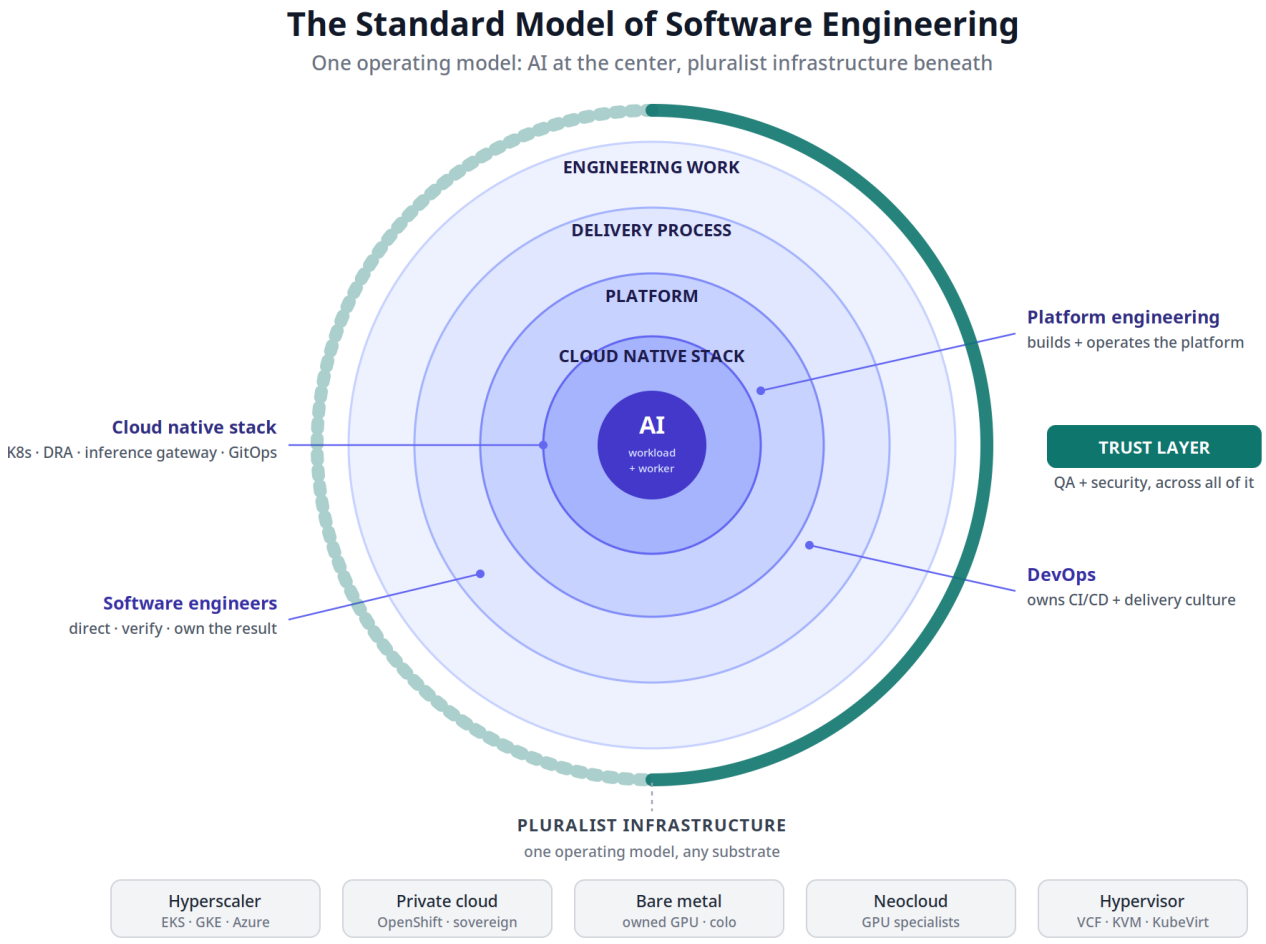


Figure 1 — The Standard Model of the AI-era software stack: infrastructure, platform, delivery, engineering and the trust layer, unified by AI as both workload and worker.

The pieces now fit together, although not quite in the way most vendor diagrams suggest.

The operating model

AI sits at the center because it occupies two positions at once. It is a workload the stack must run, and it is increasingly a worker operating inside that stack. Virtualization, cloud, DevOps and containers each pressured a part of the system. AI crosses it.

Cloud native provides the common substrate. Platform engineering packages that substrate into usable AI capability. DevOps owns the delivery process running on top of it. Software engineers specify and verify the work. QA and security provide the trust layer across the whole system. Underneath, a mix of public cloud, private cloud, bare metal, neocloud and virtualized infrastructure can change without forcing the operating model to change with it.

This is not tidy because real systems are not tidy. Ownership overlaps. The same people may wear multiple hats. A platform team can still become a release-engineering queue, and a DevOps team can still spend most of its time maintaining infrastructure. The model describes where the work is going and where organizations should draw the lines if they want AI adoption to scale safely.

Why the boundaries blur

The layers above describe structure. The more consequential claim is about what happens to the people inside them.

Each role in Section 7 moved the same direction: from doing to governing the doing. The DevOps engineer designs which gates exist rather than building the pipeline that runs them. The platform engineer brokers capability rather than provisioning infrastructure. QA designs what correctness means rather than executing checks. Security governs identity rather than defending a boundary. The software engineer specifies and verifies rather than types.

These are not five unrelated transformations. They are one shift observed from five desks.

The specializations remain real. A platform engineer's knowledge of GPU topology is not interchangeable with a security engineer's knowledge of credential scoping, and this report is not arguing that job titles collapse. What converges is the kind of work. Every role now consumes the same platform and uses AI tooling at strikingly similar weekly rates, between 65% and 70% across those surveyed. The common output is sound human judgment applied to increasingly automated work.

The disciplines are not merging. They are converging on a common substrate, a common toolset and a shared responsibility for governing automated work.

> **"The disciplines are not merging. They are converging on a common substrate, a common toolset and a shared responsibility for governing automated work."**
>
> §9 Putting the Pieces Together

The strongest evidence is not any single percentage. It is that several independent movements point in the same direction. AI moved into Kubernetes instead of building beside it. Weekly adoption clustered

tightly across roles that used to inhabit different toolchains. MCP gave agents a practical way to cross those tool boundaries. Platform-mature organizations reported better AI outcomes. The operating model spread across infrastructure classes that are otherwise moving further apart.

None of that means the work is finished. Platform engineering sits at 36.7% maturity, AI agent governance at 18.1%, and 15% of organizations report no structured AI use at all. The claim is about direction and operating shape, not universal arrival.

The shape of it

For twenty years, abstraction produced specialization because each wave solved one layer while leaving the purpose of the system intact. AI requires scheduling, routing, delivery, evaluation, identity, and cost control together. The disciplines can remain specialized, but they can no longer behave as independent shops.

This is the practical shape of the unification: shared infrastructure, clearer ownership and a common move toward setting intent and governing automated work. It is less dramatic than eliminating departments, and far more consequential than adding copilots to the tools those departments already use.

SECTION 10 OF 13

Tensions, Objections, and Honest Unknowns

The seven strongest counter-arguments to this thesis, and the signals that would falsify it.

I believe the unification thesis is supported by the evidence. I do not believe it gets a free pass. Several findings can be read another way, and three problems remain without a convincing answer: the verification bottleneck, the junior talent pipeline and the risk that proprietary middleware recreates lock-in one layer above Kubernetes.

1. Does AI-generated code break the DevOps feedback loop?

DevOps is built on small batches and fast feedback. AI does the opposite: it produces more code, in larger units, faster than review can absorb. If the constraint moves from writing to verifying, and verification is human and unscaled, then AI does not accelerate delivery: it accelerates the accumulation of unverified work in front of a fixed-capacity gate.

Faros AI's telemetry across roughly 22,000 developers found pull requests grew 154% larger in 2025 and 51.3% in 2026, with developers handling 67.4% more PR contexts daily. They call it Acceleration Whiplash: genuine gains at generation, compounding costs at every downstream stage. DORA's framing of AI as an amplifier says the same thing: it magnifies whatever the organization already is, including its dysfunctions.

Two Futurum figures make this concrete. Only 9.5% of organizations release daily, despite 58.3% claiming DevOps maturity at standardizing or mastering [1]. AI adoption in CI/CD operations sits at 13.2% [1], making delivery one of the least AI-augmented parts of the lifecycle while the code entering it is increasingly AI-generated.

Batch size is a process choice, not a property of AI. A large PR is a decision to submit one, and organizations that hold batch size constant while raising generation rate get more frequent small changes rather than fewer large ones. Review tooling is also being adopted at close to generation's rate: 37.7% against 40.2% [1], which suggests the field is not ignoring the problem.

We still do not know whether AI-assisted review scales judgment or merely scales approval. An LLM reviewing LLM-generated code may catch real defects. It may also produce confident sign-off at higher volume with correlated blind spots. No available dataset settles this, and anyone claiming otherwise is probably selling review tooling.

This objection is not defeated. Organizations with weak delivery discipline may be made worse by AI, not better. That is exactly what we should expect if AI acts as an amplifier.

2. GPU economics could re-fragment the stack

Unification assumes a common substrate. Accelerators, however, are scarce, expensive and allocated in ways that reward specialization. If GPU access requires bespoke scheduling, specific interconnect topologies, particular vendor toolchains and dedicated capacity contracts, AI infrastructure could become a separate estate with a separate team. That would recreate the fragmentation this report says is ending.

Inference cost optimization is the top infrastructure priority for enterprise decision makers at 56.7% [6], which indicates the economics are pressing enough to drive architectural decisions. Vendor toolchain lock-in at the accelerator layer is real. And the binding constraint in AI infrastructure is shifting from GPU supply toward power availability, which pulls decisions toward specific sites and specific contracts rather than toward portable abstractions.

The stack absorbed this rather than fragmenting around it. Dynamic Resource Allocation put accelerator requests in the core Kubernetes API. Kueue tracks DRA resources in the same quota system as CPU and memory. HAMi is building vendor-neutral GPU management spanning NVIDIA, AMD, Huawei, and Cambricon. The scheduling sophistication that GPUs demanded arrived inside the existing scheduler.

Cost pressure could still re-centralize AI workloads into a small number of specialized environments for purely economic reasons, leaving the rest of the estate on a different operating model in practice even where it is the same in principle. This is a real risk and the paper should not pretend the extensions settle it.

3. Platform engineering's own immaturity undercuts the argument

This report makes platform engineering the load-bearing layer: the thing that makes AI adoption safe, and the variable that determines outcomes. Yet platform engineering sits at 36.7% combined standardizing-and-mastering maturity, nearly 22 points behind DevOps at 58.3% [1]. If the unifying layer is itself immature, the unification is aspirational.

Worse: nearly 30% of platform teams do not measure success at all, in a discipline whose founding premise was treating the platform as a product with users.

This may be the strongest objection to the report's argument.

The answer is that maturity and adoption are different measures, and the objection conflates them. Most organizations have a platform; roughly a third have a good one. The 2026 State of Platform Engineering survey found 55.9% of companies already operating more than one internal developer platform, segmented by team: frontend, backend, data, and AI. That is not a field that has failed to adopt the model. It is a field dealing with the second-order problems of having adopted it broadly and unevenly.

The maturity gap is also directionally informative rather than merely embarrassing. Platform engineering at 36.7% sits above AI-assisted development at 32.9% and far above AI agent governance at 18.1% [1]: organizations are building the platform layer before formalizing the governance that will run on it, which

is the correct order.

Where the causal story could break

The causal direction in the Perforce and DORA findings is ambiguous. Platform-mature organizations report better AI outcomes, but they may simply be better at adopting technology in general. The correlation is consistent across independent datasets, and the practical implication survives: leadership can act on the platform. Still, the findings do not prove causation.

4. Is unification actually happening, or is AI just landing inside existing silos?

The dominant mode of AI use remains individual developer assistance at 47.2% of organizations, with only 5.8% at outcome-only autonomy [1]. If each role adopts its own copilot inside its own tool, AI could reinforce silos rather than dissolve them. Everyone gets faster at the existing job, and the boundaries remain intact.

Supporting this: only 6.2% use AI for deployment decisions [1], the lowest of any lifecycle phase. If AI were genuinely unifying Dev and Ops, the actual Dev/Ops boundary is where you would expect to see it, and it is the place you see it least.

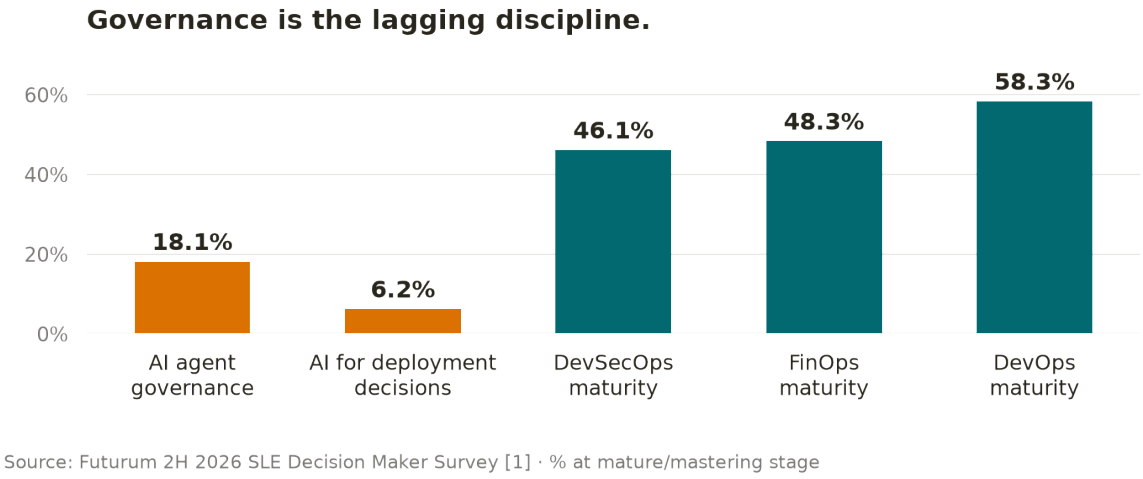


Figure 5 — Governance is the lagging discipline. AI agent governance and AI use in deployment decisions trail every other measured practice.

Section 6 argued the distribution point and it holds here: 37.8% operating agents in some mode, for a capability that did not exist in production three years ago, is not a stalled transition. But the copilot-in-a-silo concern is legitimate for the 47.2%, and the paper should not claim those organizations are experiencing unification.

In Section 7, I offered the generous reading of the deployment figure: humans are being retained at the highest-consequence boundary. The skeptical reading is that the Dev/Ops divide persists. Both fit the data. The fact that 66% of organizations use AI in infrastructure workflows while only 31% report full autonomy there tips me slightly toward deliberate human retention, but it does not settle the question.

The unresolved question is whether individual assistance is a stage on the way to agentic operation or a stable end state for most organizations. The unification thesis requires the former. Nothing in the current data proves it.

5. The skills gap and the junior engineer problem

The mechanism that produced senior engineers has been disrupted, and nothing has been designed to replace it. That belongs here as an objection to the thesis, not merely as a workforce concern.

The optimistic argument is historical: compilers, garbage collection, Stack Overflow, and frameworks each provoked the same fear and each time the profession adapted. The pessimistic argument is that in every one of those cases the adaptation was built deliberately, and none of that work has begun here.

Add the budget signal: increasing investment in generative AI (40%) and AI/ML technologies (39%) now outrank increasing hiring of IT personnel (23%) as the most critical actions for accelerating delivery [5]. Capacity is being bought as capability rather than headcount.

This objection is not answered in this report. No data exists on time-to-senior, junior hiring rates, or review depth under AI conditions. It is a genuine unknown with a plausible bad outcome.

6. The security surface of agentic systems

Unification means agents crossing tool and team boundaries. That is also the description of an attack surface. An agent with credentials, network access, and susceptibility to instructions embedded in data it reads is a novel and poorly understood risk, and the governance to manage it does not exist.

AI agent governance is the least mature practice measured at 18.1% [1], and Section 5 detailed the 91-vs-10 gap between agent adoption and non-human identity strategy plus the two-thirds of organizations that cannot separate agent activity from human activity. Nearly half of security professionals now rank agentic AI as the top attack vector for 2026.

DevSecOps has a real foundation to extend from at 46.1% maturity [1], the non-human identity market is consolidating rapidly, and the platform layer is the right place to enforce scoped agent identity. But adoption has outrun governance by a wide margin. Asking platform teams to close that gap is a prescription, not a description of current practice.

7. Is this unification, or Kubernetes lock-in by another name?

"One operating model, any substrate" is what every platform vendor has claimed. Kubernetes is not neutral ground; it is a specific, complex, opinionated system with a large operational tax. Declaring it the universal substrate is a lock-in argument wearing a unification costume.

The distinction is governance and conformance. Kubernetes is vendor-neutral in a specific, verifiable sense: CNCF-governed, with a public conformance program that produced more than 100 certified distributions and now certifies AI platforms across hyperscalers, hypervisor vendors, neoclouds, and on-premises distributions. Lock-in to a foundation-governed open standard with multiple certified

implementations is a materially different proposition from lock-in to a vendor.

The newest and most valuable layers include AI gateways, agent control planes, evaluation platforms and agent observability. Commercial vendors are moving fastest there, while open standards remain young. Futurum now tracks AI Agent Observability [4] and Agent Control Plane and Agentic Governance [3] as distinct submarkets. If those layers consolidate as proprietary control points rather than open primitives, portability may survive at the substrate and disappear where it matters most.

MCP's trajectory is the counter-signal worth watching: 20% production adoption in eighteen months for an open protocol suggests the ecosystem is defaulting toward openness at the interoperability layer. Whether that extends to the governance and evaluation layers is undetermined.

What would falsify this thesis

Five signals, stated so the paper can be judged against them:

- A separate AI stack emerges. If AI infrastructure standardizes on a purpose-built scheduler and runtime that displaces rather than extends Kubernetes, the central claim fails.
- Role adoption diverges. The current four-point spread across roles is core evidence. If that widens substantially, AI is landing in silos rather than across them.
- Agentic mode stalls. If the 37.8% operating agents plateaus while individual-assistance mode stays dominant through 2028, the developer transformation is a leading-edge phenomenon rather than a trajectory.
- Proprietary control planes win the AI middleware layer. Portability at the substrate is worth little if gateways, evals, and agent governance become vendor-locked.
- Verification fails to scale. If change failure rates rise durably alongside AI adoption and no methodology emerges to review at volume, the feedback-loop objection wins and organizations will rationally slow adoption.

SECTION 11 OF 13

What To Do Now

Concrete moves for engineering leaders, practitioners and the ecosystem — with the sequencing that actually works.

The useful dividing line is no longer between organizations that have AI and organizations that do not. Adoption is already too broad for that. The line is between organizations that can absorb AI safely and those that are still confusing tool access with readiness.

For engineering leaders

Close the governance gap before it closes on you. AI agent governance sits at 18.1% maturity while over 84.5% of organizations report AI involved in more than a quarter of their SDLC work [1]. That gap is not a future risk; it describes your estate today. The specific things to have in place: an approved model catalog, scoped identity for every agent, cost attribution by team, and audit that can distinguish agent activity from human activity, which 68% of organizations currently cannot do.

Governance that lives in a policy document will be routed around. Governance embedded in the endpoint developers already call is governance that does not require anyone's compliance. Build it into the paved road or do not bother.

> **"Governance embedded in the endpoint developers already call is governance that does not require anyone's compliance."**
>
> §11 What To Do Now

Fund the platform, not the tools. This is the clearest evidence-backed recommendation in the report. Platform-mature organizations report 73% saying platform maturity was critical to AI success versus 44% of less mature ones, 79% versus 14% on governance automation, and 81% versus 48% confidence in AI outputs in critical workflows. Confidence rises to 92% with fully standardized IDPs. DORA reaches a compatible conclusion: returns depend more on the quality of internal platforms and workflows than on tool selection.

The causal direction is not proven, and Section 10 says so. But the platform is the variable you can actually move, and no amount of tool procurement substitutes for it.

Make the PE/DevOps boundary explicit. Platform engineering builds and operates the platform; DevOps owns the delivery process running on it. Organizations that blur this reliably fail in the same direction: the platform team becomes a ticket queue for release engineering and never builds the capability layer. The

AI transition punishes that error faster than the pre-AI world did.

Treat verification capacity as a first-class investment. Generation got cheap and verification did not. If you are funding AI coding tools without funding review methodology, eval infrastructure, and QA capability, you are buying throughput into a fixed-capacity gate. Pull requests grew 154% larger in 2025 and 51.3% in 2026; the humans reading them did not change.

Build an infrastructure portfolio, not an infrastructure strategy. Sovereignty, data gravity, and GPU economics are permanent forces pushing in different directions: 57% of IT leaders already report needing to run infrastructure within a single country. The correct posture is a deliberate mix, held together by one operating model, with substrate treated as a procurement decision reviewed on economics rather than an architectural commitment.

Decide who is building your junior pipeline. The mechanism that produced senior engineers has been disrupted and nothing has replaced it. In every prior abstraction wave, adaptation was built deliberately. If your organization is buying capability instead of headcount, as the 40% priority for GenAI investment versus 23% for hiring suggests [5], someone needs to own how judgment gets developed. Right now, in most organizations, nobody does.

For practitioners: one capability to build this year

Developers: Build skill in specification and decomposition. State intent clearly enough that generated output can be evaluated against it, and size work so an agent does not lose coherence. This is requirements analysis at a granularity where it was previously uneconomic, and much of the technique is still transmitted as folklore. Practice it deliberately.

DevOps engineers: evaluation gates in the pipeline. Observability adoption runs near 89% against evals at 52%, and offline evaluation is not the same as a gate that blocks a release. Whoever builds the first working eval stage in your delivery process defines how your organization ships AI-dependent software.

Platform engineers: cost and quota as a product surface. Inference cost optimization is the top infrastructure priority at 56.7% [6]. The platform engineer who can attribute token spend by team, enforce GPU quota fairly, and explain the quantization-versus-quality tradeoff becomes indispensable quickly.

QA: Own the eval suite. Nobody currently owns it in most organizations, and it is the natural extension of the discipline into nondeterministic systems. This is a recommendation, not an observation. No data shows QA teams are claiming this ground, which is exactly why it remains available.

Security: non-human identity. An agent's blast radius is defined by its credentials, not its location. With 91% of organizations using agents and 10% having a developed NHI strategy, this is the widest gap between exposure and capability in the field.

For the ecosystem

Vendors: the interoperability bet is the winning one. Futurum's guidance is direct: customers will reward vendors who reduce the burden of adopting agentic AI across multiple platforms, using open standards such as MCP and A2A [2]. The AI middleware layer is where portability is most at risk, and Section 10 named proprietary consolidation there as the most credible threat to everything this report describes. Vendors who build open primitives at the gateway, eval, and agent-governance layers will be trusted with the control points; vendors who build closed ones will be routed around by the same instinct that produced Kubernetes.

Open source communities: the gaps are specific. Evaluation frameworks that work as CI/CD gates rather than offline test harnesses. Agent identity and permission models that are portable across runtimes. Vendor-neutral accelerator management: HAMi is the right shape and needs company. Reference implementations for AI/ML internal developer platforms, which the field is currently reinventing per organization.

Standards bodies and foundations: The conformance model worked. More than 100 certified Kubernetes distributions, and 31 certified AI platforms as of KubeCon EU 2026, span hyperscalers, hypervisor vendors, neoclouds and on-premises distributions. Apply the same mechanism to agent protocols, evaluation interfaces and identity models before those layers ossify commercially.

The sequencing that actually works

Three moves, in order, because the order is what most organizations get wrong:

- Platform first. Not because it is the most urgent, but because everything else depends on it. Organizations building governance before the platform end up with policy nobody follows; organizations adopting AI tools before the platform get shadow AI at scale.
- Governance embedded, not layered. Into the endpoint, the gateway, the runtime, the pipeline. The 18.1% governance maturity figure is not a failure of intent: it is a failure of placement.
- Verification funded proportionally to generation. Whatever you spend on making code appear faster, spend a comparable amount on determining whether it is right. This is the discipline that separates the organizations AI amplifies upward from the ones it amplifies downward.

None of this requires predicting which model wins, which vendor consolidates, or which substrate proves cheapest in 2028. It requires building the layer that makes those questions answerable later without re-architecting.

That is what the platform has always been for. AI just made it urgent.

SECTION 12 OF 13

Outlook: 2026–2028

Six directional predictions the evidence supports, and the falsification tests for each.

Predictions in technology reports are usually cheap. They are written vaguely enough to survive and forgotten before anyone checks. These six are specific enough to be wrong. By 2028, we should know whether this report described a real operating shift or mistook a noisy transition for one.

1. Agentic operations become mainstream, with humans on execution

Confident. The 37.8% of organizations currently operating agents in supervised, semi-autonomous, or autonomous modes [1] becomes a clear majority by 2028. Individual-assistance-only mode, at 47.2% today, drops below a third.

What will not happen at the same rate is full autonomy. Today 66% of organizations use AI in infrastructure workflows while only 31% report fully autonomous AI there, and 5.8% operate at outcome-only review [1]. That ratio moves slowly. The pattern to expect is broad agentic adoption with deliberate human retention at high-consequence boundaries: deployment decisions, currently at 6.2% AI adoption [1], will remain the last phase to automate and should.

The realistic 2028 picture is agents doing most of the work and humans owning most of the decisions, which is a larger change than it sounds and a smaller one than the vendor pitch.

2. Evals become a standard CI/CD stage

Moderately confident, and this is the prediction I would most like to be right about.

Today observability adoption runs near 89% against evals at 52%, and most of that 52% is offline evaluation against test sets rather than a gate that blocks a release. The direction is clear: nondeterministic components cannot be validated by deterministic tests, and organizations shipping AI-dependent software will not tolerate having no pre-merge signal indefinitely.

By 2028, I expect evaluation stages to be a default feature of managed CI/CD platforms rather than something each team assembles. The capability will be platform-provided, following the same path as golden paths before it.

The uncertainty is honest: no adoption data exists on evals as blocking gates, the tooling is immature, and this could plausibly stall as an offline practice that never enters the pipeline. If it does stall, Section 10's verification objection strengthens considerably.

3. Platform teams own AI governance

Confident. AI agent governance is the least mature practice measured at 18.1% [1], and the gap between adoption and governance is too large to persist. The question is who closes it.

It will be platform teams, for a structural reason rather than an organizational preference: they own the endpoints, gateways, runtimes, and quota systems where governance can be enforced rather than requested. Security will define policy; platform will implement it as defaults. Governance that lives anywhere else gets routed around.

Expect AI governance capability to become a standard component of internal developer platforms by 2027, and expect platform team charters to name it explicitly.

4. Hybrid and sovereign infrastructure keep growing

Confident. With 57% of IT leaders already reporting a need to run infrastructure within a single country, and sovereignty requirements arriving through regulation rather than preference, the pluralist substrate mix intensifies rather than resolving.

Public cloud continues growing at roughly 21.5% annually, so this is not an exodus story. Other categories may grow faster in percentage terms, including sovereign regions, owned GPU capacity in colocation, neocloud consumption and edge inference. The neocloud category alone went from over $25 billion in 2025 toward forecasts approaching $400 billion by 2031.

The constraint to watch is power. The AI infrastructure race is already shifting from a scramble for accelerators toward a fight over electricity and the sites that can deliver it, and that pushes decisions toward specific geographies in ways that make substrate choice less portable in practice than in principle.

5. Hypervisor equilibrium, with quiet absorption underneath

Moderately confident. The VMware disruption settles into a stable three-way split rather than a mass migration. Consolidators remain the majority for regulated core production: most estates that evaluated alternatives retained 60–80% of workloads on VMware, and ING and Barclays have publicly committed to VCF 9.0. Replacers continue taking SMB, edge, and discrete workload tiers, with Proxmox VE now at roughly 1.5 million hosts.

The interesting movement is the absorbers: organizations folding VMs into Kubernetes via KubeVirt rather than running a separate virtualization platform. This is the smallest category today and the one most aligned with everything else in this report. I expect it to grow fastest in percentage terms through 2028 without becoming dominant.

This prediction is the least well-evidenced in the section, because no clean data exists on VMware migration destinations broken out by Kubernetes-based virtualization versus another hypervisor. Treat it as a directional read.

6. AI middleware consolidates: openness undetermined

Genuinely uncertain, and the most consequential item here.

AI gateways, agent control planes, evaluation platforms, and agent observability are now tracked as distinct submarkets with named vendors competing for share [3][4]. That market will consolidate: new categories always do. Whether it consolidates around open primitives or proprietary control points is undetermined and will be substantially decided within this window.

MCP is the encouraging signal: 20% production adoption in roughly eighteen months [7] suggests the ecosystem defaults toward open standards at the interoperability layer when good ones exist. Whether that instinct extends to governance and evaluation, where the commercial incentive to close is stronger, is the open question.

If these layers close, portability holds at the substrate and disappears where it matters. That is the outcome that would do most damage to the argument in this report.

Signals to watch

Section 10 listed five conditions under which this report's thesis fails. Each has a specific, observable signal you can bookmark and check.

A separate AI stack emerges. How you'd know: a new orchestration platform appears in the CNCF landscape or a hyperscaler roadmap that is explicitly not Kubernetes-based, with vendor commitments to run AI workloads there instead of extending Kubernetes.

Role adoption diverges. How you'd know: the LangChain, Perforce, and Futurum surveys next year show weekly-use spreads across DevOps, platform engineering, software engineering, QA, and security widening past ten points, not the current four.

Agentic mode stalls. How you'd know: the 2028 lifecycle survey still has individual-assistance mode above 40% and supervised/semi-autonomous below 30%.

Proprietary control planes win the middleware layer. How you'd know: no open-source competitor for the leading AI gateway, evaluation platform, or agent control plane crosses 10% enterprise adoption by end of 2027, and CNCF projects in these categories fail to reach Sandbox or Incubation.

Verification fails to scale. How you'd know: DORA's next report shows change failure rates rising alongside AI adoption rather than holding steady, and the Faros Acceleration Whiplash pattern deepens rather than resolves.

The first is unlikely. The second would surprise me. The third and fifth are live risks. The fourth is the one I would watch most closely, because it is the only one where the outcome is still being decided by choices vendors and foundations are making right now.

SECTION 13 OF 13

Conclusion: The Question the Stack Was Asking

For twenty years the stack asked whether the disciplines could be run as one system. AI is what finally forced the answer.

We did not spend twenty years building cloud native infrastructure because we knew an AI workload was coming. We solved the problems in front of us. Virtualization separated the operating system from the machine. Cloud separated capacity from ownership. DevOps attacked the ceremony around release. Containers separated applications from their environments. Platform engineering tried to give developers the benefits without requiring every developer to become an infrastructure specialist.

Each step delivered value. Each also produced another specialty, another toolchain and, inevitably, another conference. By the time AI arrived, software engineering had become a collection of expert communities working on adjacent parts of the same system.

AI does not fit neatly inside any one of those communities. It requires accelerator-aware scheduling, model-aware routing, evaluation for nondeterministic output, identity for non-human actors and cost controls that reach down to individual requests. Then the agents cross the very tool and team boundaries we spent years drawing. The platform engineer, DevOps practitioner, developer, QA engineer and security professional may still own different parts of the problem. They can no longer solve them separately.

The test of this argument will not be a keynote or a survey. It will be what the KubeCon 2027 program actually contains: whether agentic workloads, evaluation gates, and non-human identity have moved from experimental tracks into the certified conformance surface, and whether the operating model this report describes is being ratified in the same way containers and their orchestrator once were. If it is, the great unification has moved from thesis to infrastructure.

There is plenty here that can still go wrong. Governance maturity is nowhere close to adoption. Verification has not caught up with generation. We have no settled way to turn today's junior engineers into tomorrow's senior ones if the entry-level work that built judgment is increasingly automated. Open infrastructure at the bottom will not save us if gateways, evaluation platforms and agent control planes become proprietary tollbooths at the top.

Those risks are reasons to shape the transition, not deny it. Organizations should build the platform before buying their way into more fragmentation. Governance belongs in the paths people and agents already use. Verification capacity must grow with generation capacity. The profession also has to rebuild apprenticeship for a world in which producing code is no longer the same thing as learning to engineer software.

The great unification is not a finished architecture. It is the new shape of the work. The cloud native stack gave AI somewhere to run. AI is now giving the rest of software engineering a reason to operate as one system.

SOURCES & RESEARCH NOTES

Sources and Research Notes

The numbered citations in the body refer to the following Techstrong and Futurum research. Additional public sources are attributed in the prose where used.

[1] Futurum 2H 2026 Software Lifecycle Engineering Decision Maker Survey (n=839).

[2] Mitch Ashley, Futurum research on multi-platform agentic AI interoperability, October 2025.

[3] Futurum Agent Control Plane and Agentic Governance submarket research.

[4] Futurum AI Agent Observability submarket research.

[5] Futurum 1H 2026 Software Lifecycle Engineering Decision Maker Survey.

[6] Futurum 1H 2026 Data Intelligence Decision Maker Survey.

[7] Futurum 1H 2026 decision-maker research on Model Context Protocol adoption (n=416).

Additional research cited throughout the report includes CNCF and Kubernetes project documentation; DORA research on AI as an organizational amplifier; Perforce's State of DevOps: Platform Engineering 2026 (n=820); the State of Platform Engineering, Volume 4 (n=518); LangChain's State of Agent Engineering (n>1,300); Nutanix's 2026 Enterprise Cloud Index; Okta's AI Agents at Work 2026; Cloud Security Alliance research on agent identity and visibility; Faros AI telemetry covering roughly 22,000 developers; and Synergy Research market data.

Company-reported figures and vendor claims are identified in the prose where used. Survey findings on the share of AI-generated production code reflect decision-maker estimates, not repository telemetry, and should be read as directional.