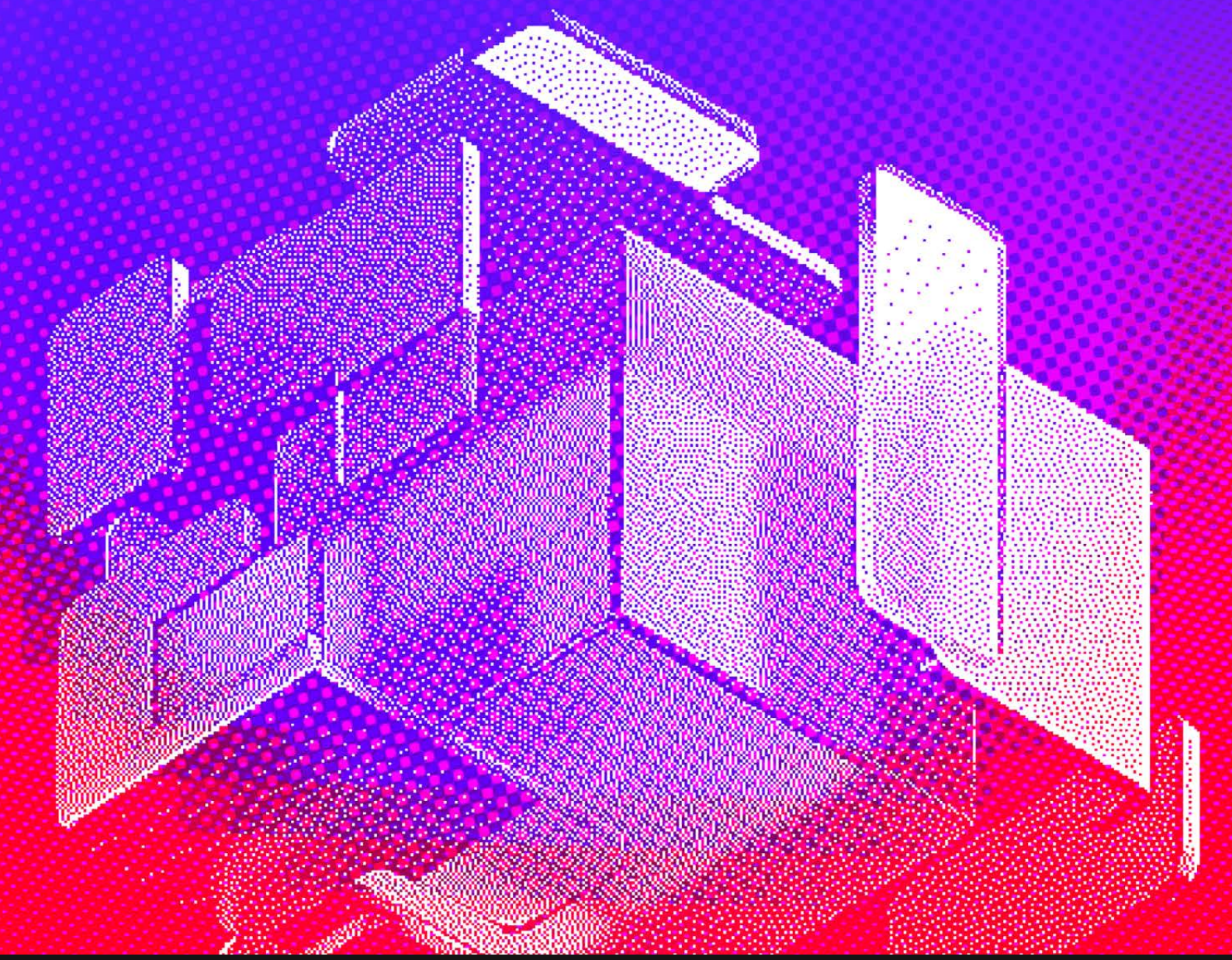
BROADCOM | Platform Engineering

Platform engineering 2.0: An evolution for the AI era

A REPORT COMMISSIONED BY BROADCOM



Weave Intelligence

PLATFORM ENGINEERING 2.0: AN EVOLUTION FOR THE AI ERA
PUBLISHED IN 2026

Platform engineering 2.0: An evolution for the AI era

Table of contents

Authors

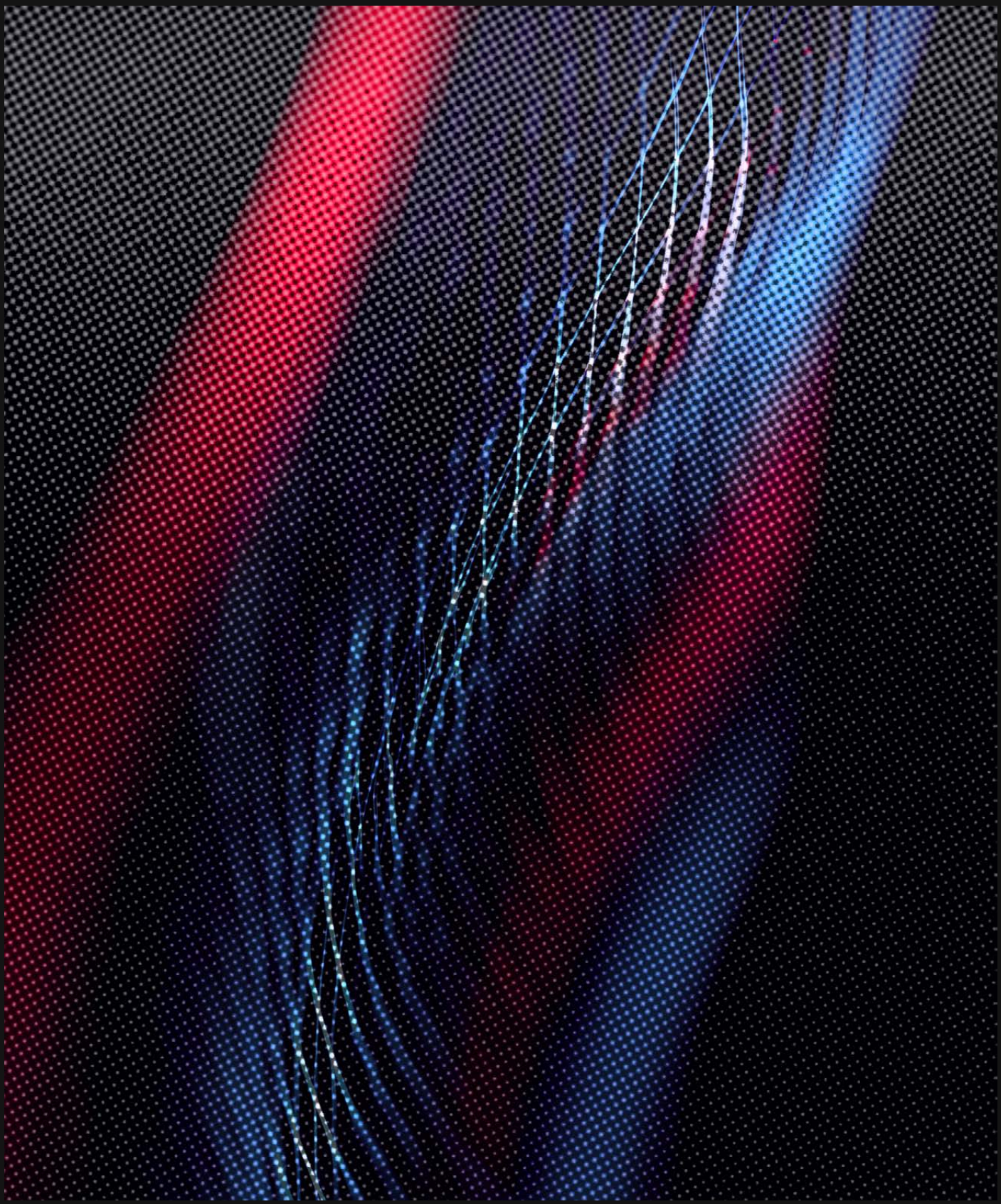
Pankaj Gupta

Pankaj is Senior Director of Private Cloud Solutions at VMware by Broadcom. In his role, Pankaj helps enterprise customers maximize the value of their private cloud investments. In prior roles, he spearheaded strategic initiatives for modern applications, networking, security, and cloud portfolios at Citrix, Cisco and others.

Sam Barlien

Sam is the Head of Ecosystem for the Platform Engineering Community. He is a tech nerd, and has been involved in tech communities for more than 10 years. He has been a platform engineering community leader from the beginning, and has helped drive key community initiatives like Platform Weekly, while co-hosting PlatformCon, and leading the community Ambassador program, blog and Youtube channel. He speaks to 100s of platform engineers a year and translates their experience into articles, webinars and reports for the wider community.

Introduction



The state of platform engineering today

Platform engineering has crossed the adoption threshold. According to the State of platform engineering report, and the 2025 DORA report, 90% of organizations have now adopted platform engineering practices, and 76% maintain a dedicated platform team, however maturity of platform team varies across organizations. At the same time, the evidence is becoming that the key to success with enterprise AI is platform engineering. The debate is no longer whether to build an internal platform, but how best to ensure success with the one you've got.

Platform engineering is reaching universal adoption, but over the past two years, five forces have converged to press hard on those platforms. AI-driven coding acceleration has shifted the bottleneck from writing code to delivering it and autonomous agents are emerging as a new class of platform user. Cloud waste meets expensive AI infrastructure, alongside a deluge of token spend on flat budgets. The question of sovereignty is accelerating everyday. And the platform must now serve security, data scientists, FinOps, and business personas alongside the developers you might have first had in mind. While at the same time, most developers use AI tools generating 2-10× more code, and your agents need GPU allocation, MCP gateways, and audit logging, while 35% of cloud spend evaporates as waste, it becomes clear your current platform was likely built for a different era. And it must either evolve or become the bottleneck it was designed to eliminate.

90%

of organizations have adopted platform engineering, making a high-quality internal platform the essential foundation for AI Success

76%

Have dedicated platform team

80%

See AI integration critical for IDP success

70%

Measure platform success

Sourced from State of Platform Engineering & 2025 DORA Report

This whitepaper offers an answer. Platform engineering 2.0 is an evolution rather than a reset. The foundations of platform engineering; Platform as Product, golden paths, security shift left and self-service Internal Developer Platforms (IDPs) remain essential. What changes is who the platform serves, what it must do, and how it must be built. This whitepaper extends those foundations across five fundamental pillars for the AI-era - AI-native platform, multi-persona experience, embedded FinOps, security shifting down, and more composable architecture. The AI-native pillar is anchored on the Agentic Development Platform, and the structural evolution of the IDP for the agentic era. The multi-persona experience explores how your platform must grow to serve far beyond the application developer. Embedded FinOps explores how your platform must embrace the challenges of rapidly growing cloud, AI infra, and token spend. While security shifting down and composable architecture demonstrate the flexibility to change building blocks with faster and confident reparability.

While at the same time, at the heart of every platform engineering initiative lies infrastructure, the foundational substrate of compute, storage, networking, and GPU resources that makes everything else possible. Platform Engineering 2.0 demands a fundamental rethinking of that substrate. Moving from static, developer-centric provisioning to a dynamic, AI-ready foundation that serves every persona, enforces policy at runtime, and scales on demand. Infrastructure is not the plumbing behind the platform; it is the platform's most strategic layer, and the primary determinant of how far any platform evolution can go.

This whitepaper sets a multi-year directional framework for platform teams and the executives who fund them, grounded in data from The State of Platform Engineering, Vol. 2. DORA 2025, and the State of platform engineering Vol. 4 alongside a host of expert interviews gathered from across our community of almost 280,000 practitioners and platform engineering leaders.

As the AI era reshapes the world of work, many disciplines ask themselves about their relevancy in a new agentic world. Platform engineers, however, face an almost unparalleled challenge and opportunity, as data from almost every direction demonstrates that the foundational layer for the success of AI in the enterprise is platform engineering.

> Our research shows enterprises built platform engineering around a developer-centric model, but that model needs to evolve as AI-assisted development, autonomous agents, and cost volatility expose structural limitations. Organizations trying to advance their platform engineering practices should not abandon the core principles; instead, they need to enhance them.

Jim Mercer

PROGRAM VICE PRESIDENT, SOFTWARE DEVELOPMENT, DEVOPS, AND DEVSECOPS, IDC

Key takeaways

These statements capture the essential arguments and recommendations of this whitepaper. If you read nothing else, read these.

01	Platform engineering 2.0 is evolution, not revolution	It extends the foundation of Platform as Product, golden paths, shift left security and self-service IDPs rather than replacing them. Your existing platform investments are not obsolete; they are the substrate for what comes next.
02	AI is both the forcing function and the opportunity	It introduces a new workload class requiring GPU provisioning, model serving, guardrails and MCP gateways, plus a new class of platform user where agents need scoped permissions, non-human identity, guardrails and audit logging.
03	IDP evolves into an Agentic Development Platform	Same operating model, expanded substrate enabling collaborative work between humans and AI agents.
04	The platform must serve more than developers	Security teams, data scientists, FinOps, business, and AI agents are now first-class personas, each requiring the right tools, the right abstractions, and the right intelligence.
05	Cost becomes a first-class signal	Embedded FinOps moves cost intelligence from rear-view-mirror reporting to provisioning-time decisioning, with every developer and operator becoming a FinOps practitioner.

06 Security shifts down into the platform

This addresses AI's new attack surfaces, including shadow AI sprawl, prompt injection, model poisoning, and inference data leaks, that developer tooling alone cannot catch. It complements shift left security.

07 The future of platforms is not build vs. buy; it is compose

API-first, faster and confident repavable building blocks connected through well-defined contracts give you the architectural flexibility that AI workloads and rapid experimentation demand.

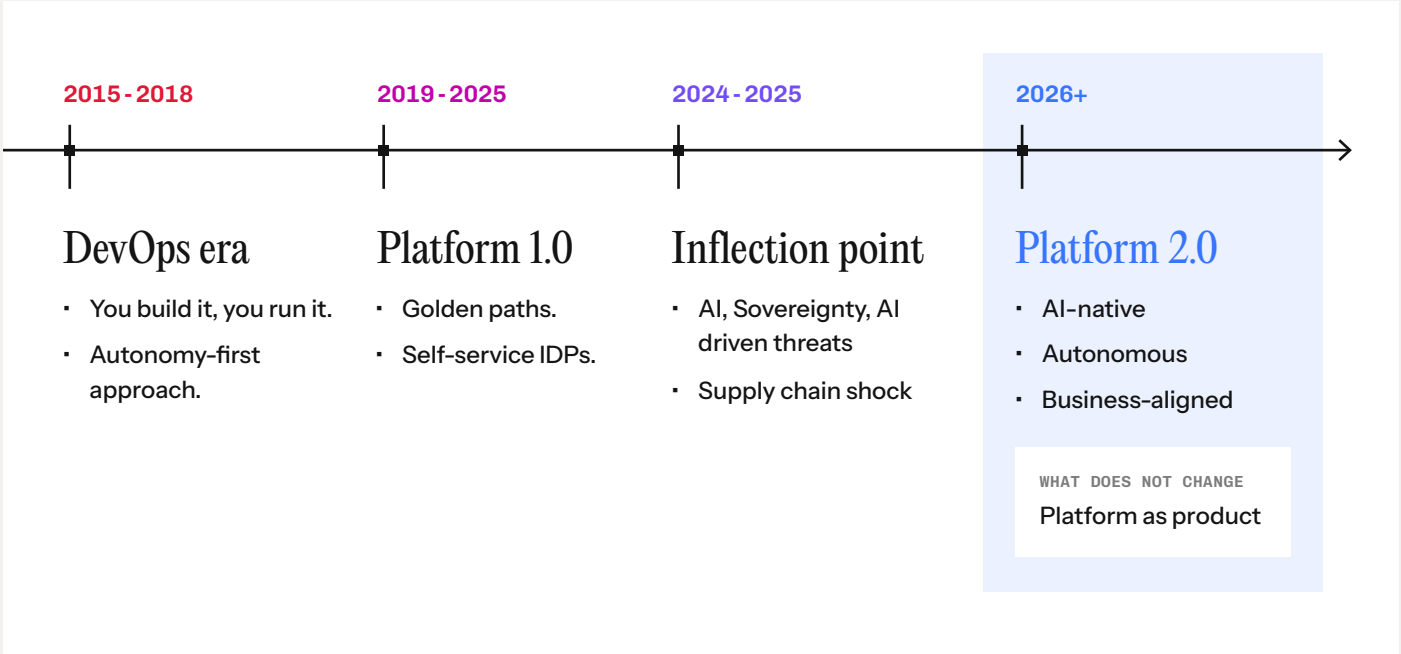
08 Infrastructure is what makes this evolution tangible

Every golden path, every self-service portal, every agentic workflow ultimately resolves to a provisioned resource, a container, a GPU instance, a network boundary, a storage volume. As platform engineering enters the AI era, the infrastructure layer must scale with it: provisioned dynamically, governed continuously, and optimized across every workload class and persona.

Journey of an evolution

Platform engineering's evolution has been continuous over the last decade. The discipline does not restart at each phase; it accumulates. Everything platform engineering built continues into the AI era. What changes is who the platform serves and what it must carry.

The developer's role is changing in parallel. Pre-vibe-coding, developers wrote most of the code by hand, one feature at a time, with the platform shouldering infrastructure complexity behind them. The developers job is moving from primary author to reviewer, designer, and operator of an AI-augmented production line, and the platform must support both modes at once.



2015 - 2018

DevOps era

"You build it, you run it." Teams owned their full stack, and the wall between development and operations came down. The era established the cultural foundation of shared ownership, fast feedback, and continuous delivery that platform engineering would later systematize.

2019 - 2025

Platform engineering matures

The discipline professionalized around three capabilities, all aimed at reducing cognitive load on application developers. Golden paths, popularized by Spotify, gave teams opinionated, pre-approved templates for common workloads and collapsed the decision surface across deployments. Internal Developer Platforms turned those paths into something developers could actually consume, a single standard , self-service way to provision environments, deploy services, and access infrastructure without context-switching across a dozen tools.

Platform as Product, the practice of treating the platform like a product and its users like customers with roadmaps, feedback loops, and adoption metrics, was the operating model that made all of this sustainable. The discipline also delivered on DevSecOps's shift-left ambition, embedding scanning, secrets management, and compliance into CI/CD pipelines.

2024-2025

The inflection point

AI-driven coding acceleration, sovereignty regulation, AI-driven security threats, and hardware supply chain shocks pressed hard on platforms designed for containerized, developer-centric, human-paced workflows. The architecture began to show structural strain under the new load.

2026 AND BEYOND

Platform engineering 2.0

AI-native, multi-persona, composable, with expanding agent autonomy under platform guardrails. The five pillars in this whitepaper define the capabilities that characterize this phase. The pace of change is itself a defining feature of the 2.0 era. New model capabilities, agent patterns, tooling, and attack surfaces arrive on much faster cycles rather than annual ones. A platform team that waits for these to stabilize before responding will spend the decade behind the curve. Proactive evolution, anticipating workloads and personas before they arrive at the platform door, becomes a core platform engineering competency rather than a nice-to-have.

What does not change is the governing operating model. Platform as Product remains the organizing principle, and developers, golden paths and self-service IDPs remain central. The customers have expanded; the operating model has not. The 2.0 era describes a direction, not a destination. It is a 3-4 year arc continuing through 2030. Organizations that begin the evolution now will have the foundation in place when the arc's later capabilities, including full Agent Infrastructure, autonomous platform operations, and vertical agentic platforms, become operational requirements.

> What started as a developer productivity function is now the centralised governance layer for the enterprise – enforcing cost discipline, security posture, and AI readiness across every team. The platforms that can absorb that scope without structural debt aren't the ones built around fixed architectures. They're the ones built to be composable from day one.

Atulpriya Sharma

CO-ORGANISER, CNCF PLATFORM ENGINEERING TECHNICAL COMMUNITY GROUP

Current platform's ceiling

The foundations platform engineering established remain stable But the platform must now carry loads it was not designed for. Five limitations - first two are strategic and three are operational - define the ceiling for most organizations.

AI-Blind Architecture

Need support for GPU/ TPU workloads, model serving, MCP servers; AI for platform.

Developer Only Focus

Serves one persona in a multi-persona org: Security, data eng, ML, business, FinOps.

Reactive, No Proactive

Always feels behind, No cost-aware provisioning and proactive drift detection. Lacks defense in depth.

Golden Paths Become Golden Cages

Standard predefined templates restrict experimentation, minor changes, unique needs.

Rigid and Static Platform

Compliance a snapshot, not a continuous guarantee, 50+ CNCF projects 2018 to 200+ now.

AI-Blind Architecture

Platform engineering was built for containerized workloads. It has no first-class support for GPU and TPU provisioning, model serving, model registries, MCP gateways, or AI-specific governance. When a data science team needs a self-service GPU environment, or when an agent needs a bounded scope, the platform has no native answer. The gap is structural, not configurational.

Developer-only focus

Current platform engineering serves one persona well. In the majority of platforms, security teams, data engineers, ML practitioners, FinOps analysts, and business users have been treated as second-class citizens or excluded entirely. Every organisation has multiple personas who can benefit from the platform. The benefits of platform engineering work just as well for security, FinOps or data teams as they do for application developers. A platform that serves one persona in a multi-persona organization is a platform that serves a shrinking fraction of its potential value.

Reactive, not proactive

Platform teams designed only for DevEx perpetually feel behind. There is no cost-aware provisioning, no proactive drift detection, and limited defense in depth. The platform responds to problems rather than preventing them. In an era where AI workloads generate cost spikes overnight and security threats evolve continuously, reactive is not a viable posture.

Golden paths becoming golden cages

The standardized templates that enabled 80% of deployments now restrict the experimentation and flexibility that newer workloads demand. Teams working on novel AI architectures find themselves blocked by the very guardrails designed to help them, but the same pattern shows up wherever the template hits its edges. Stateful workloads and VM-based systems get pushed off the path entirely. Acquired teams cannot migrate fast enough to adopt it. Regulated workloads need exception paths the template does not anticipate. Minor changes require platform team intervention. The path that was golden becomes a constraint. And the platform team gets bogged down in active maintenance of them.

Rigid and static

Compliance in your platform is likely a snapshot, not a continuous guarantee. The platform was built for a narrower tooling landscape. The CNCF ecosystem has grown from roughly 50 projects in 2018 to more than 200 today, dramatically widening the choice surface that rigid, monolithic platforms cannot accommodate. Swapping one component for another cascades changes across the stack.

These limitations are not failures of platform engineering. They are the natural edges of a discipline in the process of extending. Each of these limitations traces back, in part, to infrastructure architecture decisions made in a simpler era. Static provisioning models, CPU only-bound developer workloads, and infrastructure APIs designed solely for containerized services cannot carry the load that AI workloads, multiple personas, and autonomous agents now demand. Modernizing the infrastructure layer is the prerequisite for addressing every ceiling described here.

Forces demanding evolution

Many forces have converged on the platform over the past 2 years . Together they explain why evolution is no longer optional.

AI Driven Coding Acceleration

Bottleneck shifted from code gen to code delivery.

- 90% of developers now use AI tools
- Generating 2–10× more code, 2–10× pipeline throughput needed

Agentic Future

Exponential growth of AI agents, but lack of support

- Apps are embedding autonomous AI agents. Platforms will be next
- Need GPU/TPU allocation, MCP servers, guardrails, audits

FinOps Reckoning

Business ROI & proactive optimization is critical

- 35% cloud waste, Unpredictable cloud bill shock. Expensive AI Infrastructure
- Retrospective FinOps not sufficient, PE1.0 lacks focus and predictive FinOps

Sovereignty and Compliance

Fueled by regulatory, nationalist and AI workloads

- Regulatory requirements for sovereignty and continuous compliance
- Security and compliance is afterthought and add-on in Platform Engineering 1.0

Multi-Persona Enterprise

Value creation beyond Developer persona

- ML engineers, data scientists, FinOps, AI agents all need platform
- Brings VMs to platform engineering best practices

With 90% of developers using AI coding assistants and code volumes rising 2-10x, the bottleneck has shifted from writing code to delivering it. The developer's role is shifting in parallel. Where developers once wrote most code by hand, one feature at a time, they now specify, orchestrate, and validate work produced by AI assistants and agents across multiple streams in parallel. Platforms built for human-paced development cannot match this throughput, and they were not designed for the new shape of the developer's job.

At the same time, we are moving beyond humans into an agentic future. AI agents are moving from research projects to enterprise workloads at exponential pace. Applications are embedding autonomous agents today, and platforms themselves are next in line. Agents need GPU and TPU allocation, MCP gateways, bounded autonomy guardrails, audit logging, and policy enforcement for the decisions they make on the platform's behalf. Platforms that cannot support agents will not support the next generation of enterprise applications.

So, what does that mean? Well, the culture of retrospective FinOps, focused on a monthly bill review and quarterly optimization sprint, cannot respond without risk of massive overnight bill spikes.

And while the platform embraces these challenges. The EU AI Act, US executive orders on AI safety, sector-specific regulation in financial services and healthcare, and accelerating data residency requirements all converge to throw even more requirements in the platform mix. While at the same time, your ML engineers need self-service GPU provisioning, your data scientists need experiment tracking and pipeline orchestration, and FinOps teams need cost attribution, security teams need policy-as-code tooling, business leaders need ROI visibility, and AI agents have arrived as the first non-human platform user in over a decade.

WHAT DOES THIS MEAN FOR ORGANIZATIONS?

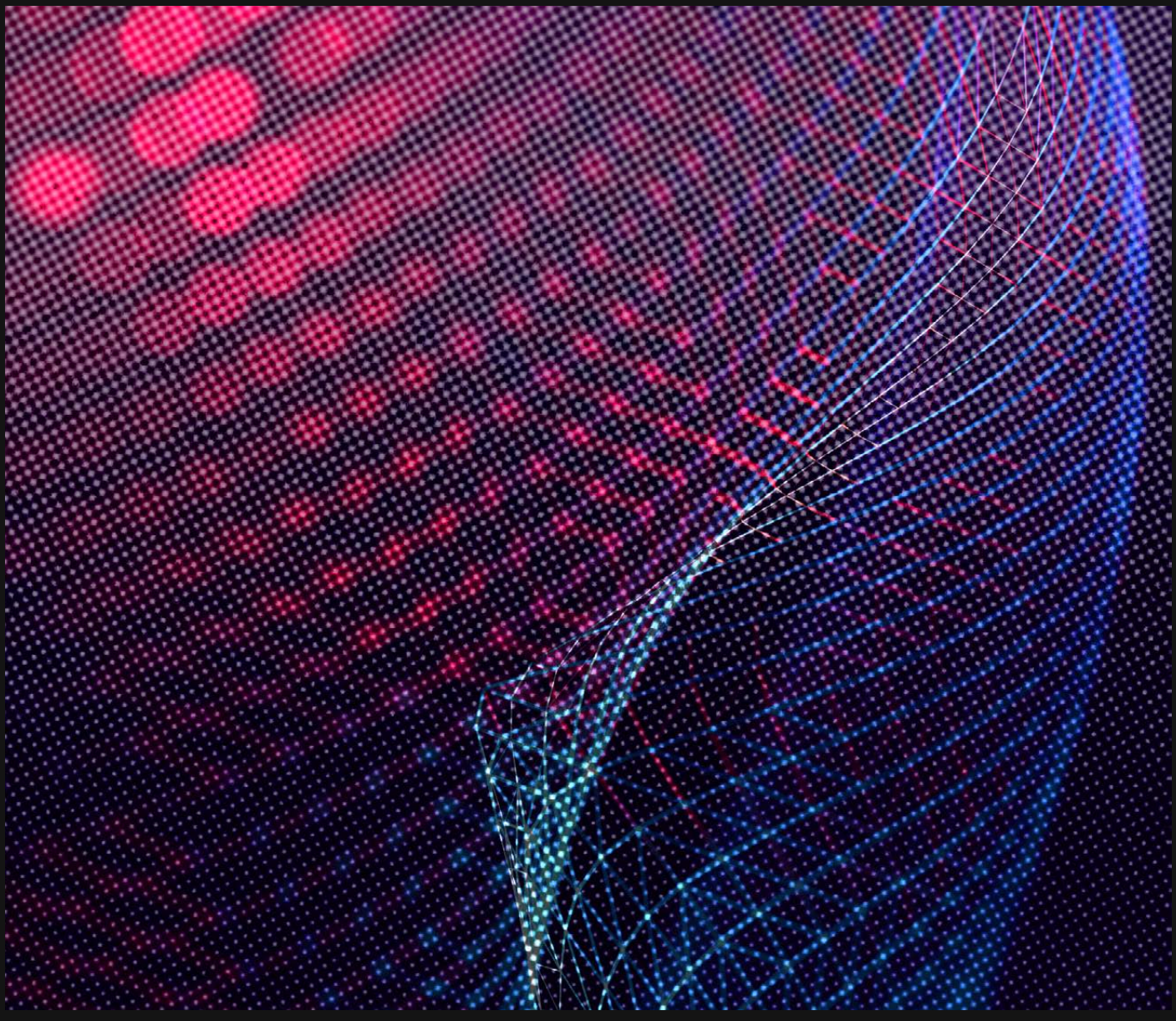
Cloud and AI cost discipline has to become a board-level concern. The industry baseline sits around 35% cloud waste, and AI infrastructure amplifies it far beyond what teams are used to handling. GPU instances, inference endpoints, and training jobs dwarf traditional cloud spend. On top of that sits the tokenomics of large language model usage: the per-token cost of every prompt, every retry, and every coding-assistant interaction, growing exponentially as agentic development spreads, opaque to most engineering teams, and invisible to most cost-reporting tools.

For platform engineers, there is no discipline better served for driving incredible value from these challenges. It is the platform team that must expand to serve these new personas, ensure the needed compliance and security, and grow to manage this new AI and agentic era in a secure and cost-effective way.

Each of these forces places direct, measurable pressure on infrastructure. Autonomous agents require GPU allocation, low-latency networking, and isolated execution environments. Cloud waste and AI infrastructure costs demand a substrate that self-optimizes. Sovereignty and compliance requirements demand infrastructure with data-residency controls and continuous auditability built in, not bolted on.

Introducing the five pillars

This framework is structured around five pillars that extend the foundation of platform engineering. Each pillar addresses a specific limitation from the ceiling described earlier and answers one or more of the forces just described. Together they define the capabilities that distinguish a modern platform from its predecessor. Understanding how the pillars interlock, and why the AI-native pillar anchors everything together, is the starting point for any evolution roadmap.



The five pillars

AI-Native Platform

Answers AI-driven coding acceleration and the agentic future. Addresses AI-blind architecture. Anchored on the Agentic Development Platform (ADP), the structural evolution of the IDP for the agentic era.

Multi-Persona Experience

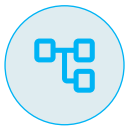
Answers the multi-persona enterprise. Addresses developer-only focus. Extends platform capabilities to six distinct personas and recognizes that the same Agent Infrastructure generalizes to other teams agentic platforms across the organization.

Embedded FinOps

Answers the FinOps reckoning. Addresses reactive, not proactive cost management. Moves cost intelligence from bolt-on reporting to provisioning-time decisioning.

Security shifts down

Answers sovereignty and compliance pressure and the new AI attack surfaces. Addresses reactive posture and AI security gaps. Embeds security into the platform and runtime layers.



Composable by design

Answers the pace of change across AI workloads, agentic systems, and a 200+ project CNCF ecosystem. Addresses rigid and static platforms. Enables API-first, repavable building blocks organized into five independently evolvable layers.

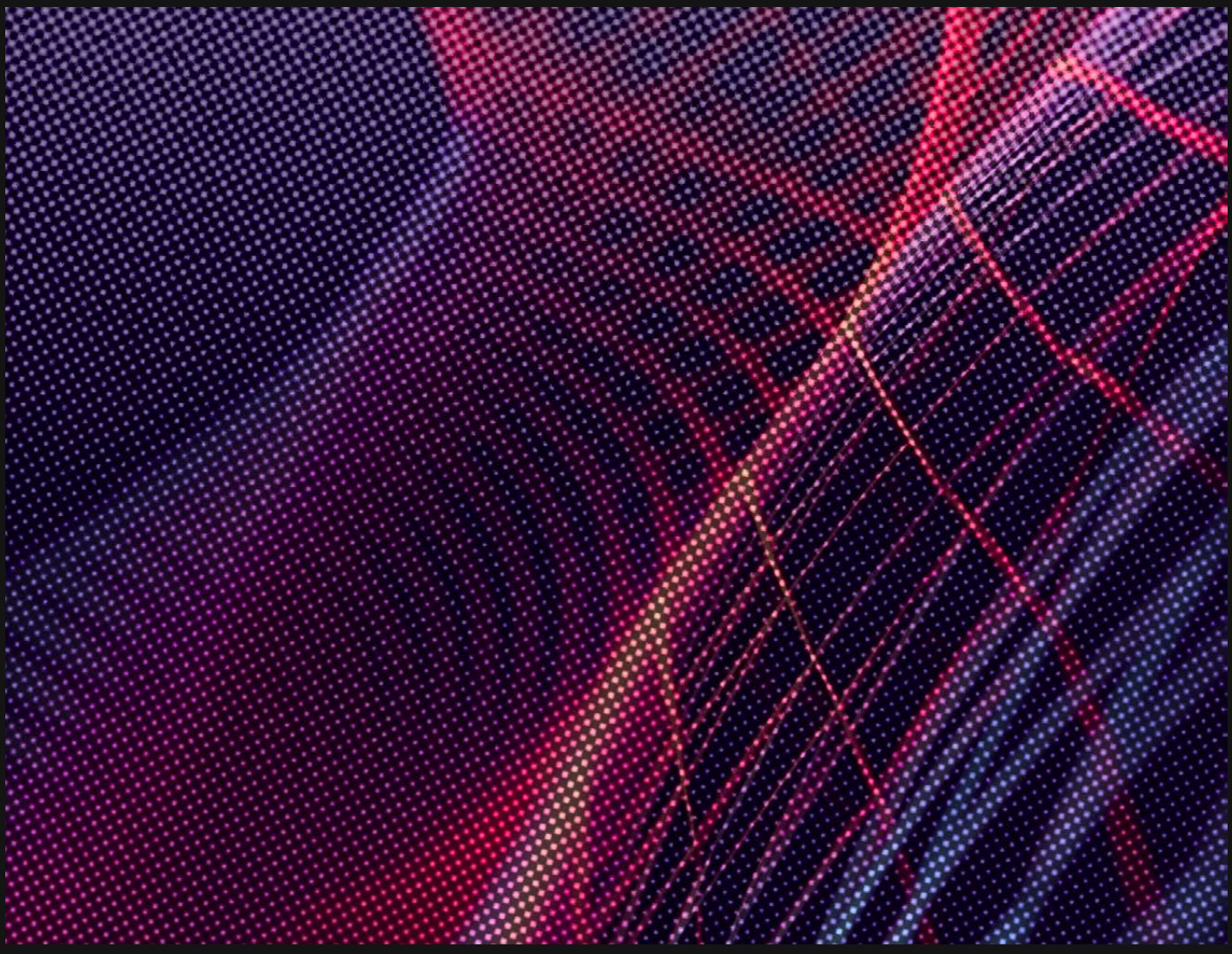
These pillars interlock in a number of crucial ways. Take AI workloads for example. They pull on all five simultaneously. They require GPU provisioning (Pillar 1), serve multiple personas (Pillar 2), demand cost and token usage visibility (Pillar 3), create new security surfaces (Pillar 4), and require architectural flexibility for new tools (Pillar 5). Composability and the multi-persona experience are the enabling conditions for the rest. Visualize the framework as Platform as Product at the foundation, with five pillars rising from it.

One key aspect that runs through all five pillars as the unifying substrate is again Infrastructure. It is what the AI-Native Platform provisions for GPU and model workloads, the foundation that Embedded FinOps must govern and optimize, the enforcement layer where Security Shifting Down becomes immutable, and the composable building blocks that give architecture the flexibility to adapt at the pace AI demands. No pillar delivers its full value without a modern, capable infrastructure layer beneath it.

PILLAR 1

AI-native platform

An AI-native platform supports two new realities simultaneously. AI workloads must run as first-class citizens, and AI agents must be supported as a new class of user. Meeting both is the evolution of the IDP into what is increasingly called the Agentic Development Platform, built and operated by the platform engineering team to enable collaborative software development between humans and AI agents. The pillar plays out across three horizons: AI workloads native to the platform now, AI agents as platform citizens in the near future with guardrails, and the AI-powered autonomous platform as the long-term destination.



AI workloads native on the platform (Now)

The most immediate and actionable dimension is the platform's ability to provision, manage, and govern AI workloads natively, alongside other applications. This requires GPU and TPU provisioning with dynamic allocation policies, model serving, versioned model registries, ML pipeline orchestration integrated with existing CI/CD tooling, and Model Context Protocol (MCP) server infrastructure for agent-tool integration. Every AI workload should flow through the same self-service platform interface as any other workload, with security policies, cost attribution, and observability instrumentation applied automatically.

A single platform for all applications, containers, VMs, and AI workloads on the same foundation, delivers consistent operations, policy, security, compliance, and observability. It also closes the utilization gap. Most environments today run at 5-20% utilization across CPU, memory, and GPU. AI-aware scheduling and agent-driven optimization are the mechanisms that close that gap, but they require the unified platform substrate to function.

AI agents as platform citizens (Near future with guardrails)

As AI agents become embedded in enterprise operations, the platform must provide the infrastructure these agents require. Agentic workloads need bounded autonomy. Agents can act within defined limits, must request approval beyond those limits, and every action is auditable. This is non-negotiable for enterprise agentic infrastructure.

The key infrastructure components include MCP-compatible APIs for agent discovery and invocation, policy guardrails constraining agent actions to pre-approved patterns, audit logging capturing the full decision chain, and escalation mechanisms routing uncertain decisions to human reviewers. Platform teams operationalising bounded autonomy typically organise these requirements into seven concerns: identity (who is acting), context (what they know), capability (what they are allowed to do, including the MCP gateway and tool registry), execution (where they run), evaluation (was the output acceptable), security (is the system protected), and observability (what happened). Defining all seven as Agent Infrastructure as Code (AIaC) brings the recoverability, auditability, and reproducibility of IaC to agent operations.

> AI tools are accelerating code generation, autonomous agents are arriving with no prior persona to inherit from, and AI infrastructure is introducing bill shock that retrospective FinOps cannot catch. The platform is either transformed to incorporate AI-native, multi-persona, cost-aware, secure by construction principles, or it can become the bottleneck it was built to eliminate.
>
> **Jim Mercer**
> PROGRAM VICE PRESIDENT, SOFTWARE DEVELOPMENT, DEVOPS, AND DEVSECOPS, IDC

AI-powered autonomous platform (Long term)

The longer-term vision is a platform that self-heals, self-optimizes, self-scales, and self-secures, with agents, control loops, policy engines, and telemetry working in concert to reduce operational burden. This includes agentic infrastructure operations for routine provisioning and scaling, predictive capacity management, automated incident triage with known-pattern remediation, and continuous policy enforcement that detects and remediates drift in real time. Organizations can begin building toward this today by instrumenting platforms with the telemetry and policy frameworks autonomous agents will require.

Infrastructure consequences

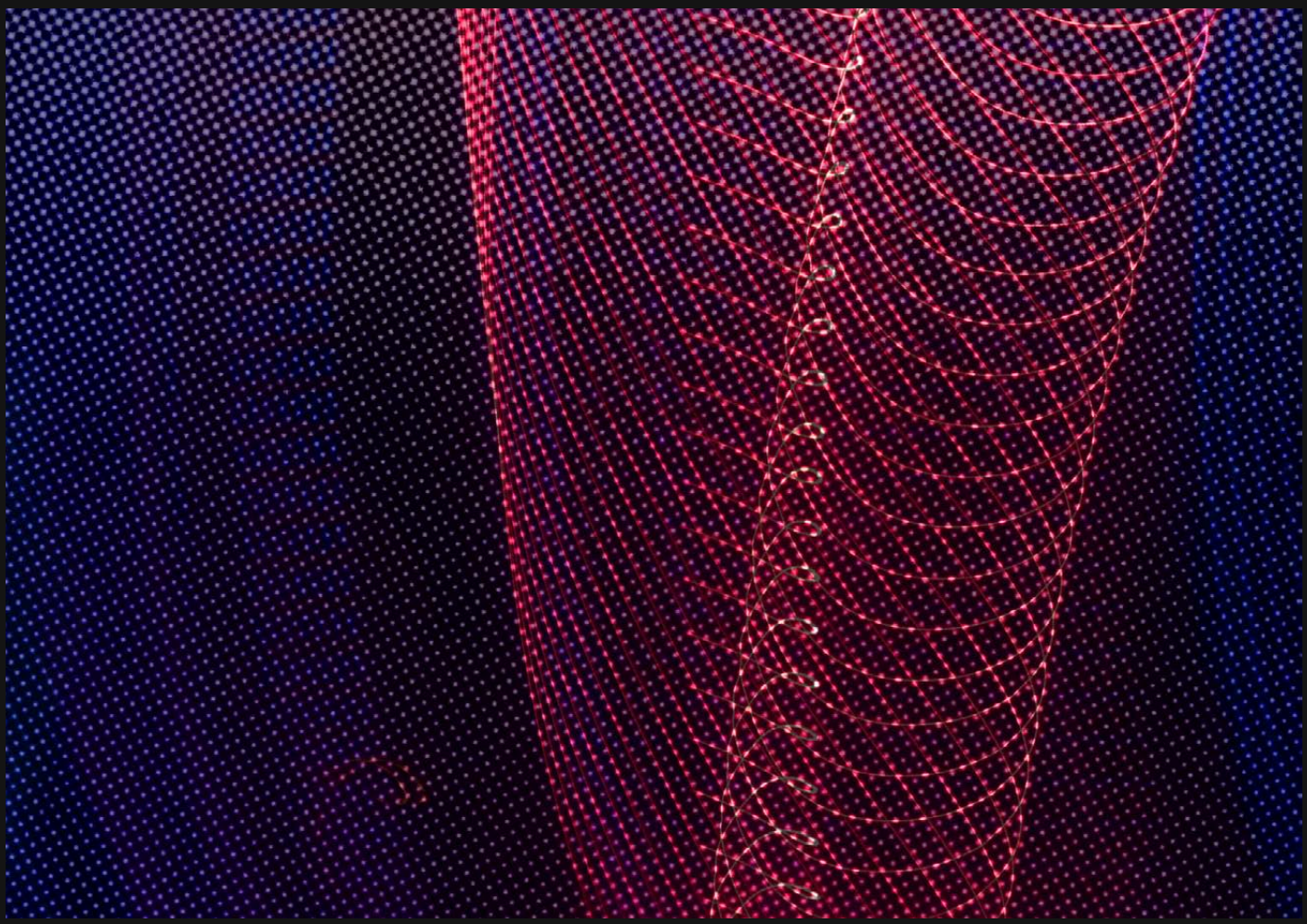
The infrastructure requirements of this pillar are precise and non-negotiable: GPU and TPU provisioning with dynamic allocation policies, model-serving infrastructure with versioned registries, MCP server infrastructure for agent-tool integration, and isolated execution environments with bounded autonomy guardrails.

These are not software configurations, they are infrastructure capabilities that must be provisioned, governed, and optimized at the platform layer before any agentic workload can run reliably at enterprise scale.

PILLAR 2

Multi-persona experience

The Multi-Persona Experience pillar addresses the developer-only focus of the past by recognizing that the platform engineering team now touches far more of the organization than it once did. We identify six distinct personas the platform must serve, some carried forward from the 1.0 era, others net new. Each has unique needs, and each requires the right tools, the right abstractions, and the right intelligence. A one-size-fits-all approach is not the answer, and is in fact a disaster in waiting.



> The most underrated limitation in PE 1.0 is persona scope. Most platforms left security teams, data scientists, AI agents, and FinOps analysts to build their own toolchains, producing shadow IT at scale. A platform that serves one persona in a multi-persona organization serves a shrinking fraction of its potential value. PE 2.0 addresses this structurally.
>
> Ron Westfall
>
> VICE PRESIDENT & PRACTICE LEAD, NETWORKING AND INFRASTRUCTURE AT HYPERFRAME

PERSONA 1

App Developer

EXISTING | EVOLVING

App developers remain the largest consumer of the platform and the team whose velocity most directly determines time to value for your organization. They need speed to ship, golden paths, CI/CD self-service, and now the pipeline throughput and validation loops to absorb AI-assisted work safely. At the same time, an emerging sub-class is the Agentic AI developer, who ships agents rather than just code and needs the full agentic substrate including evaluation frameworks, MCP gateways, and hybrid path tooling.

PERSONA 2

Platform Engineers

EXISTING | EVOLVING

Platform engineers have a dual role. They are builder and user of the platform. They build and operate the platform every other persona depends on. They need IaC, GitOps, observability, SLO management, and DORA telemetry, plus the architectural authority to make composability decisions that hold at scale. The role is expanding now to accumulate AI workloads, Agent Infrastructure, and the emerging AI Ops Engineer sub-discipline as well!

PERSONA 3

Engineering and Business Leaders

EXISTING | ACCELERATING

Leaders are accountable for engineering ROI, cost discipline, and AI readiness, and cannot make those calls from monthly PDF reports. They need live dashboards with cost attribution by team, DORA telemetry showing deployment trends, and AI readiness indicators showing how prepared the organization is for the next workload class.

PERSONA 4

Security and Compliance Teams

ACCELERATING NOW

Security and compliance teams operate in a regulatory environment that periodic audits cannot satisfy, with AI workloads creating attack surfaces developer tooling does not catch. They need direct access to policy enforcement surfaces, real-time compliance dashboards, proactive drift detection, policy-as-code with auto-remediation, and audit trails covering both human and agent actions.

PERSONA 5

Data Scientists and ML Engineers

ACCELERATING NOW

This persona was most consistently underserved in the 1.0 era, despite being central to the AI roadmap leaders are now being asked to deliver. They need self-service GPU and TPU provisioning, versioned model registries, experiment tracking, ML pipeline orchestration integrated with CI/CD, and reproducibility guarantees that match what app developers take for granted.

PERSONA 6

AI Agents

ACCELERATING NOW

AI agents are the first net-new persona platform engineering has had to design for in over a decade, because the next generation of value creation runs through them. They consume APIs rather than interfaces, and need versioned well-documented APIs, scoped permissions, non-human identity, audit logging, budget controls, and egress controls.

That expansion is the strongest argument for why platform engineering's importance grows, not shrinks, in the agentic era. The discipline that built the platform for developer autonomy is now the discipline that builds the platform for enterprise-wide agentic autonomy. That is a significant expansion of mandate, and a significant expansion of value.

For platform teams, the implication of serving multiple personas instead of one is structural. Experience layers must be specialized, with different roadmaps, portals, CLIs, and APIs for different personas, while the underlying platform APIs remain shared. Fragmented backends create fragmented governance. A shared API surface with persona-specific frontends is the pattern that scales.

Simultaneously, serving multiple personas is, at its core, an infrastructure challenge. Data scientists need self-service GPU provisioning and reproducible compute environments. Security teams need direct access to policy-enforcement infrastructure. FinOps analysts need cost attribution tied to actual infrastructure consumption. AI agents need sandboxed execution with scoped resource quotas. Each persona's experience layer is only as capable as the infrastructure substrate beneath it, which is why infrastructure must be designed for multi-tenancy and multi-persona access from the outset.

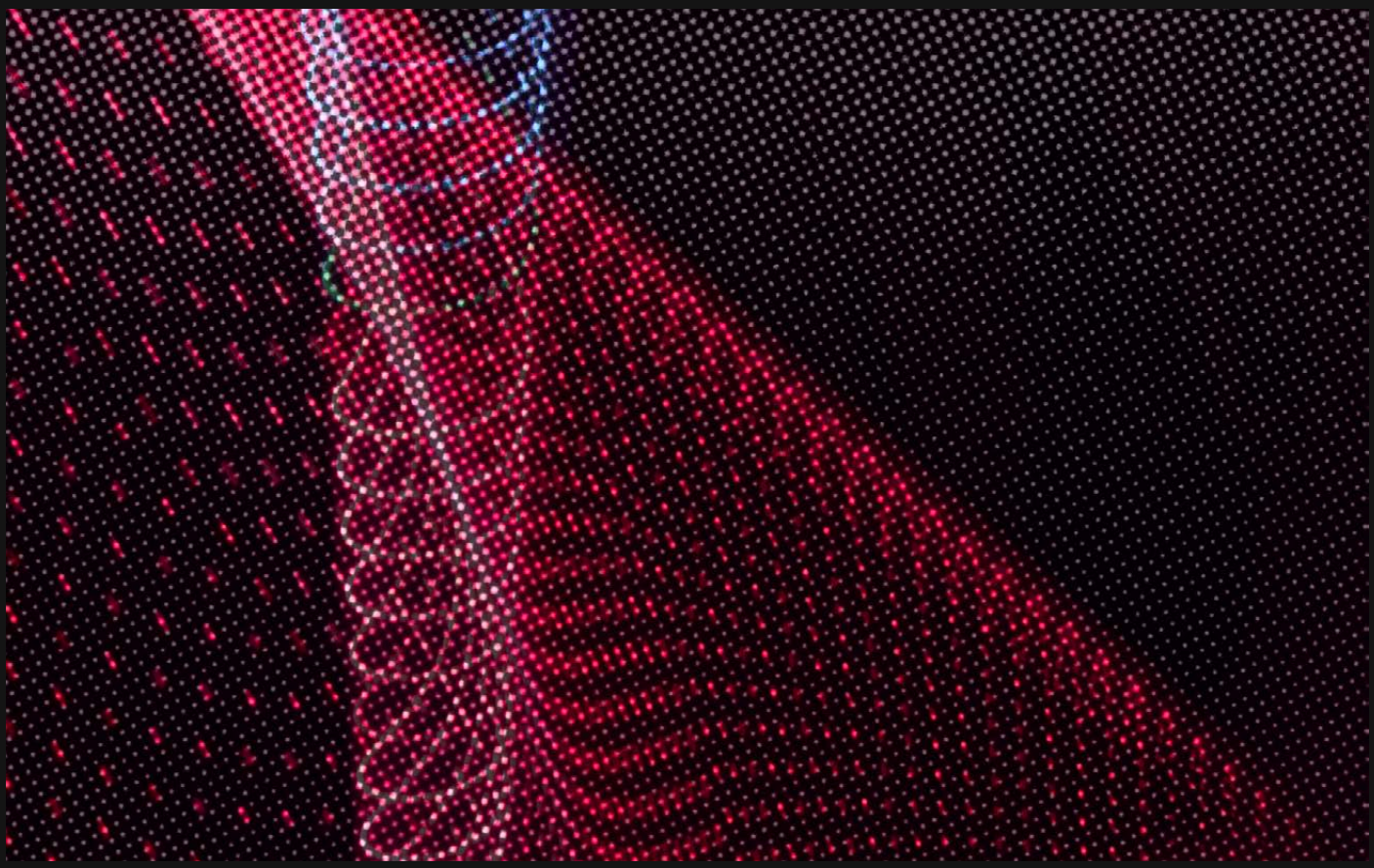
> Platform engineering teams that design exclusively for the application developer persona are leaving 60% of potential enterprise value on the table. High-performing platforms in 2026 serve ML engineers, data scientists, security teams, FinOps practitioners, and increasingly, AI agents as first-class consumers.

Roy Illsley
CHIEF ANALYST, OMDIA

PILLAR 3

Embedded FinOps

Every persona that provisions resources needs cost context at the moment of decision, not in a monthly report that arrives after the damage is done. The Embedded FinOps pillar moves cost intelligence from rear-view-mirror reporting to provisioning-time decisioning, making cost a first-class platform signal rather than an afterthought. There is a structural urgency to this. GPU and memory prices have risen 2-3× in the last few years while most engineering budgets have stayed flat, which means actual buying power is falling even when headcount holds steady. AI infrastructure spend is landing on top of existing cloud waste. While token costs from agentic software development add a new and largely invisible cost category that most organizations have no tooling to track. Retrospective FinOps - the monthly cost review and quarterly optimization sprint - cannot respond fast enough when a single misconfigured AI workload can generate bill shock overnight.



A fundamental source of cost but also a point of control is the infrastructure. Servers, GPU instances, storage tiers, network, utilization targets are infrastructure decisions, and it is at the moment of infrastructure provisioning that cost governance has the highest leverage. A FinOps capability that operates above the infrastructure layer can report on spend; only one embedded at the infrastructure layer can prevent it.

What makes embedded FinOps possible?

Three architectural preconditions make embedded FinOps possible. The first is infrastructure as code. Pre-deployment cost governance requires an intervention point between developer intent and provisioned resource, and IaC is what creates it. Platforms that wire portals directly to cloud provider APIs have no merge gate to attach policy to, and no shift-left FinOps capability can be retrofitted onto that architecture. The second is Cloud+ scope. IaaS billing is now a declining share of total OPEX as SaaS consumption (observability platforms, data warehouses, CDNs), third-party LLM APIs, and AI workload costs grow. A platform that allocates cloud bills but ignores the Datadog bill, the Snowflake bill, and the token cost of every coding assistant in the organization is governing a shrinking fraction of total spend. The third is allocation latency. Cost data that is 24 hours stale cannot route anomaly alerts to the right team, cannot inform provisioning decisions, and cannot support unit economics calculations. Real-time semantic allocation is what makes FinOps data operationally useful rather than historically interesting. Without these three preconditions, the capabilities below describe a destination the platform cannot reach.

Data from Harness FinOps in Focus 2025 and Cast.AI's 2026 State of Kubernetes Optimization makes the current state even more dire. 52% of engineering leaders cite a FinOps-developer disconnect as a driver of wasted spending. Kubernetes workloads in public cloud run at 8% CPU, 20% memory, and 5% GPU utilization on average, meaning the vast majority of provisioned capacity is idle. 55% of developers ignore cost management entirely, while 62% say they want more control over cloud costs but lack the tooling to act. 86% of developers take more than a week to identify and act on idle or orphaned resources. 68% lack fully automated cost-saving practices.

> *52%*
>
> **of engineering leaders cite a FinOps-developer disconnect as a driver of wasted spending.**
>
> Source: FinOps in Focus 2025

These are not individual failures; they are platform design failures.The platform does not give developers cost context at the moment they make provisioning decisions, so developers make decisions without it.

The evolved platform engineering initiative's answer is to embed cost intelligence directly into the platform so that every developer and operator becomes a FinOps practitioner by default - not through training, but through platform design that surfaces cost at the point of action, without additional cognitive load on platform users.

FinOps capabilities

Self-service FinOps embedded by default

Built-in cost transparency with showback and chargeback per tenant. Developers see all provisioning options and their team's budget impact at provision time, with real-time consumption visibility rather than end-of-month summaries.

Real-time cost attribution

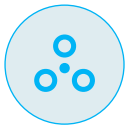
Every resource tagged and attributed to team or project automatically. Cost per deploy, cost per request, and peer comparison give developers the feedback loop they need to make cost-conscious decisions without requiring FinOps expertise.

Pre-deployment cost gates

Policy-as-code budget enforcement before production. When a service would exceed a threshold, the platform blocks the deployment and surfaces alternatives - spot pricing, right-sized instances, on-premises options - at the moment the decision can still be changed.

Tokenomics and cost attribution at scale

GPU time, token usage, and inference cost attributed per team and per model. A budget policy engine with team thresholds and alternative suggestions makes AI infrastructure costs as visible and governable as compute costs. This is the category most organizations currently have zero tooling for.



Autonomous janitor and resizing agents

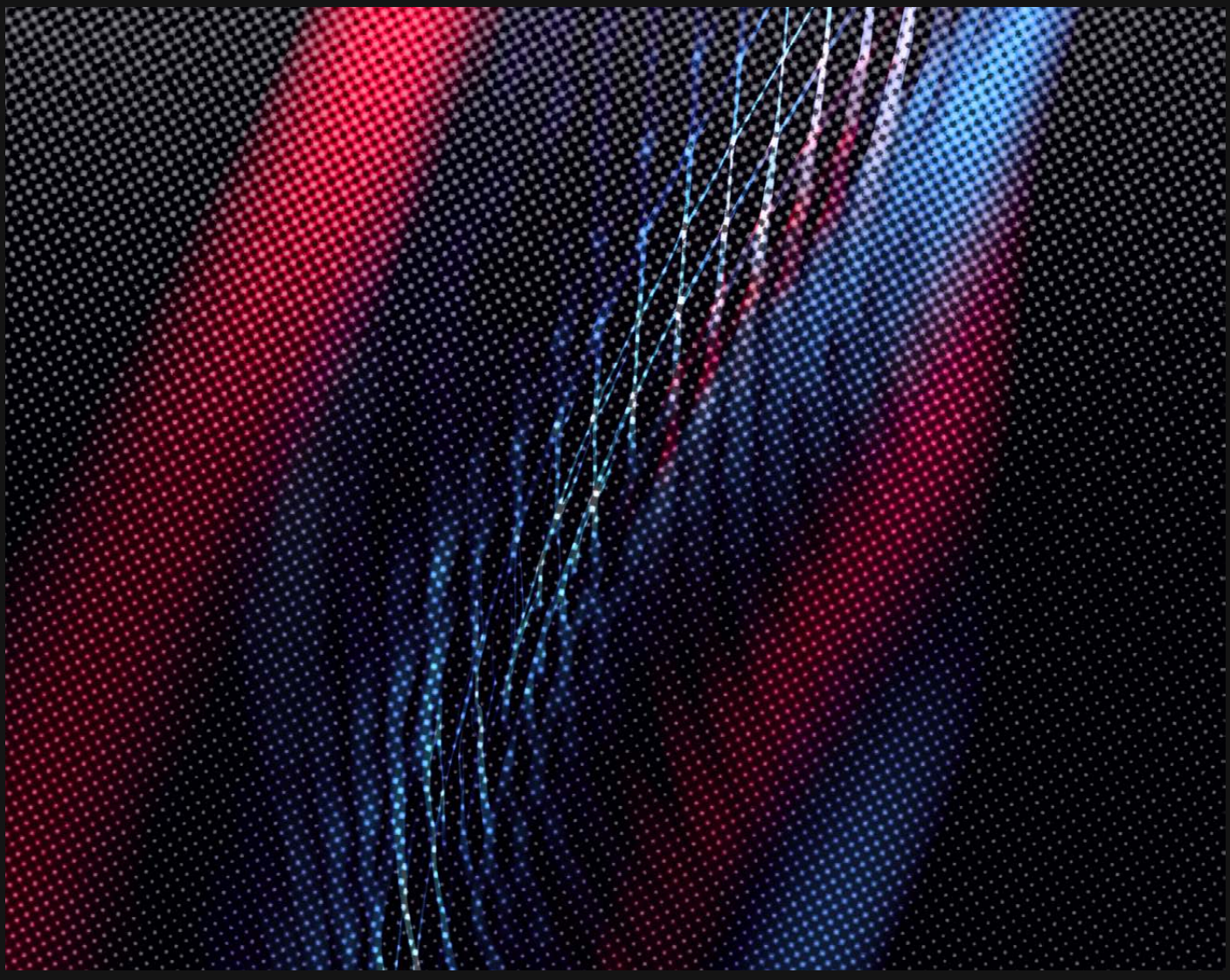
Agents that discover and decommission zombie infrastructure, right-size over-provisioned resources, and shift optimization from reactive to predictive. The CNCF ecosystem has active projects moving in this direction, and the open-source foundation for cost-aware scheduling is maturing rapidly.

A single through-line runs across all five capabilities. Move cost from a reporting function to a platform function. When cost is embedded at the platform layer, it becomes as automatic as security scanning in a CI/CD pipeline - present at every decision point, invisible in its friction, fundamental in its effect.

PILLAR 4

Security shifts down

Shift-left tried to fix security by moving it earlier. It didn't work fully, at least not the way it promised. Developers got more tools, more findings, and more responsibility, but the platform itself stayed largely unchanged, and most vulnerabilities still get caught in production. This evolution corrects the architectural mistake. Shift-left moves security across the timeline using the platform; shift-down moves it into the substrate. The platform becomes the thing that's secure, rather than the thing developers have to remember to secure.



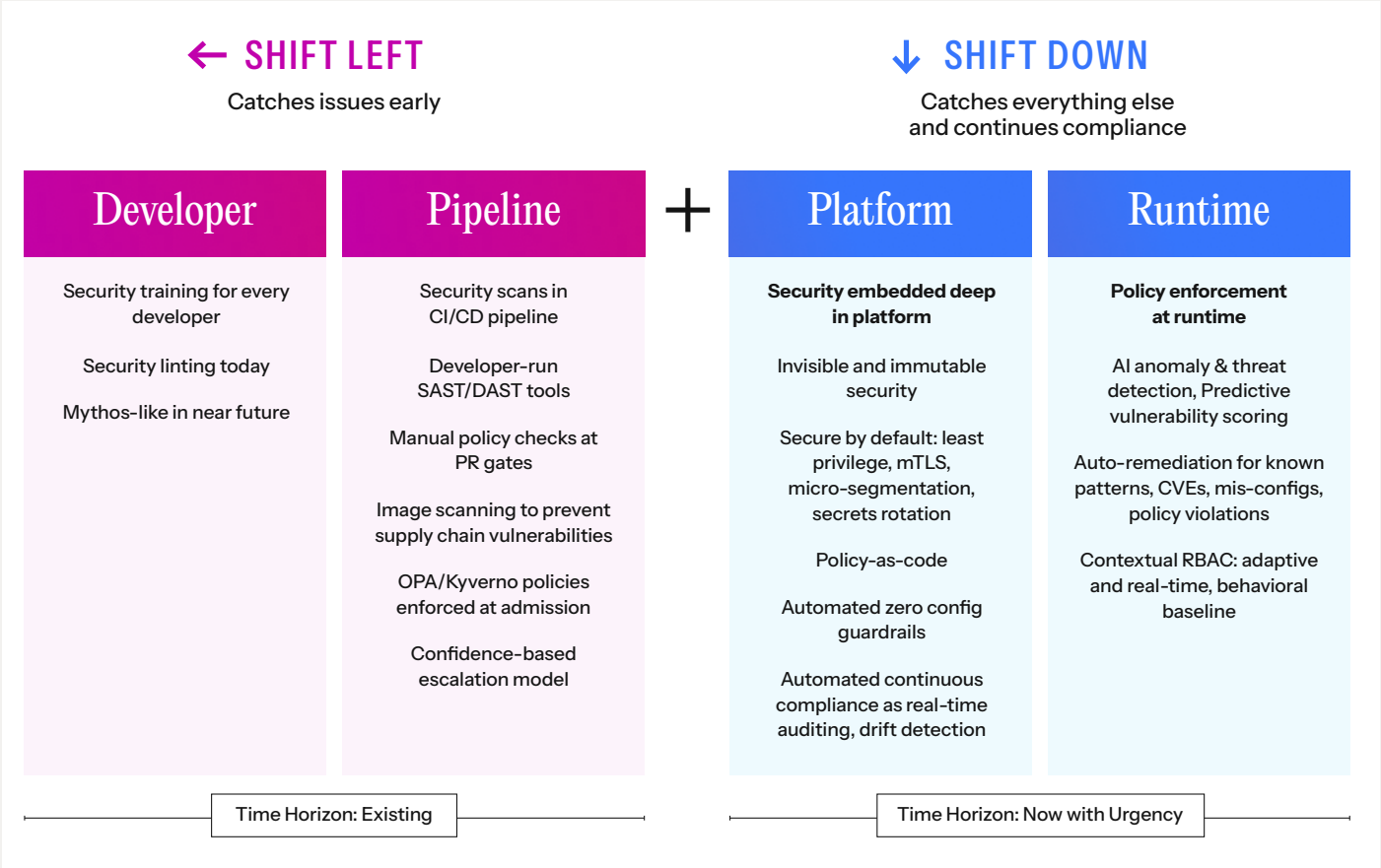
The shift-left promise and its limits

Shift-left delivered real value. Security training for developers, security linting in IDEs, SAST/DAST/SCA scans in CI/CD pipelines, and manual policy checks at PR gates all moved vulnerability detection earlier in the development lifecycle. Earlier detection means lower fix cost. That logic is sound and the practice remains valuable.

In practice, however, shift-left pushed responsibility to developers often with dire consequences. Developer cognitive overload increased as security tooling multiplied. And at the end of the day, most vulnerabilities are still found in production, because pipeline scans miss runtime context and runtime threats evolve continuously after code ships. Organisations then suffer from long remediation cycles as bolted-on security creates additional friction rather than comfortably resolving it. It was necessary to shift left, as a world in which security is only thought of by security teams isn't realistic. However, the simple approach of shifting that load and responsibility onto developers was never going to work alone.

Platform and runtime responsibilities

Shift-down succeeds where shift left fails because it ensures that security is embedded into the platform and runtime layers themselves., making it invisible to developers and immutable by design. The platform layer handles what developers should never have to configure manually.



Platform layer responsibilities

Security sits deep in the platform, invisible and immutable by default. Configurations are secure by default, with least privilege, mTLS, micro-segmentation, and automated secrets rotation built in. Policy-as-code provides automated zero-config guardrails, and continuous compliance runs as real-time auditing and drift detection rather than point-in-time snapshots.

Runtime layer responsibilities

At runtime, policy enforcement continues rather than ending at deploy time. AI anomaly and threat detection runs with predictive vulnerability scoring. Auto-remediation handles known patterns, CVEs, misconfigurations, and policy violations as they occur. Contextual RBAC adapts in real time, with behavioral baselines that adjust to changing context rather than relying on static rules.

Fundamentally, shifting security down means embedding it into infrastructure itself, not as an overlay, but as an immutable property. Secure-by-default infrastructure delivers least-privilege networking, encrypted storage at rest, mTLS between services, and policy-as-code enforced at the infrastructure admission layer. When infrastructure is built with security as a first-class property, developers inherit compliance rather than configure it, and the attack surfaces that AI workloads introduce are addressed at the layer best positioned to contain them.

The new AI security frontier

Agentic development adds a specific challenge to this layer. AI has created an entirely new category of security challenges that neither shift-left tools nor Platform Engineering 1.0 architectures were designed to address.

Shadow AI Sprawl

Unsanctioned model use bypasses DLP and data governance controls.

Prompt Injection

New attack surface: No SAST/DAST tool detects injection in live inference streams.

Model Poisoning

Models pulled from registries without signing or poisoning checks.

Inference Data Leaks

Unseen by current tools: PII and IP exposed via responses with no audit trail, no controls

The platform must become the trust boundary. Four control surfaces define what that means in practice:

Model Registry Governance

Versioned model registry with provenance tracking and approval gates, Drift monitoring.

Time Horizon: Existing

Data Isolation

Tenant-level data boundaries with encryption at rest and in transit, Access Control.

DLP policies embedded in inference pipelines with PII detection and masking.

Time Horizon: Now

Prompt Security

Input/output guardrails against injection, data and hallucination risks.

Platform-level input sanitization, output filtering, and context boundary enforcement.

Time Horizon: Near Future with Urgency

Inference Audit

Full audit trail of every AI inference with explainability, Compliance reporting.

Time Horizon: Near Future with Urgency

When security shifts down into the platform, every path is a secure path, every deployment is a compliant deployment, and every AI inference is a governed inference. This is the architectural victory of embedding security at the platform layer rather than bolting it onto the developer layer.

> Platform engineering is no longer a software delivery discipline. It is becoming the operational foundation for the enterprise's agentic future. The teams that built golden paths and self-service IDPs now hold the substrate for AI-native infrastructure and autonomous agent governance. This is the largest mandate the discipline has ever held, and a defining opportunity for IT leaders willing to act now.

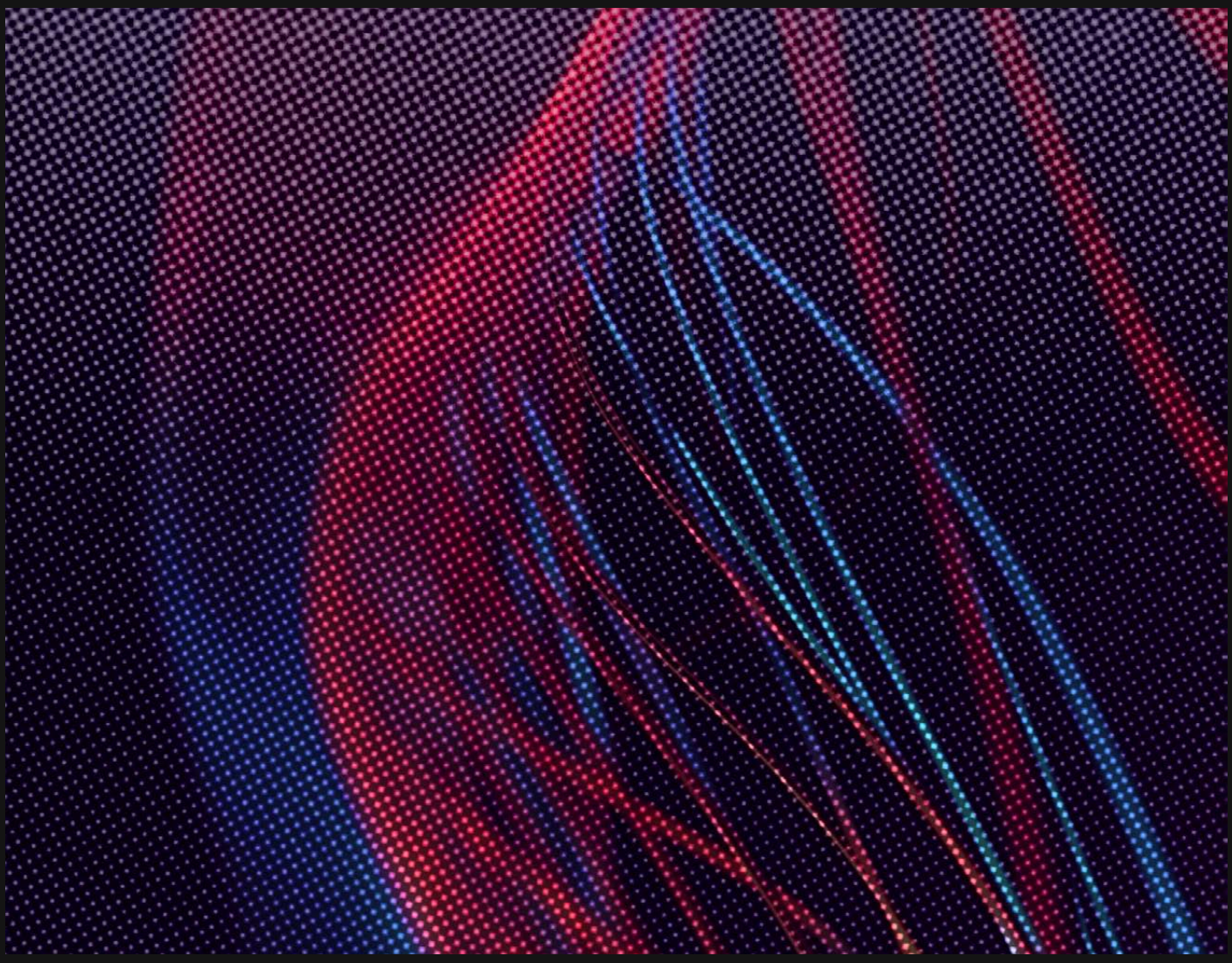
Steven Dickens

CEO & PRINCIPAL ANALYST, HYPERFRAME RESEARCH

PILLAR 5

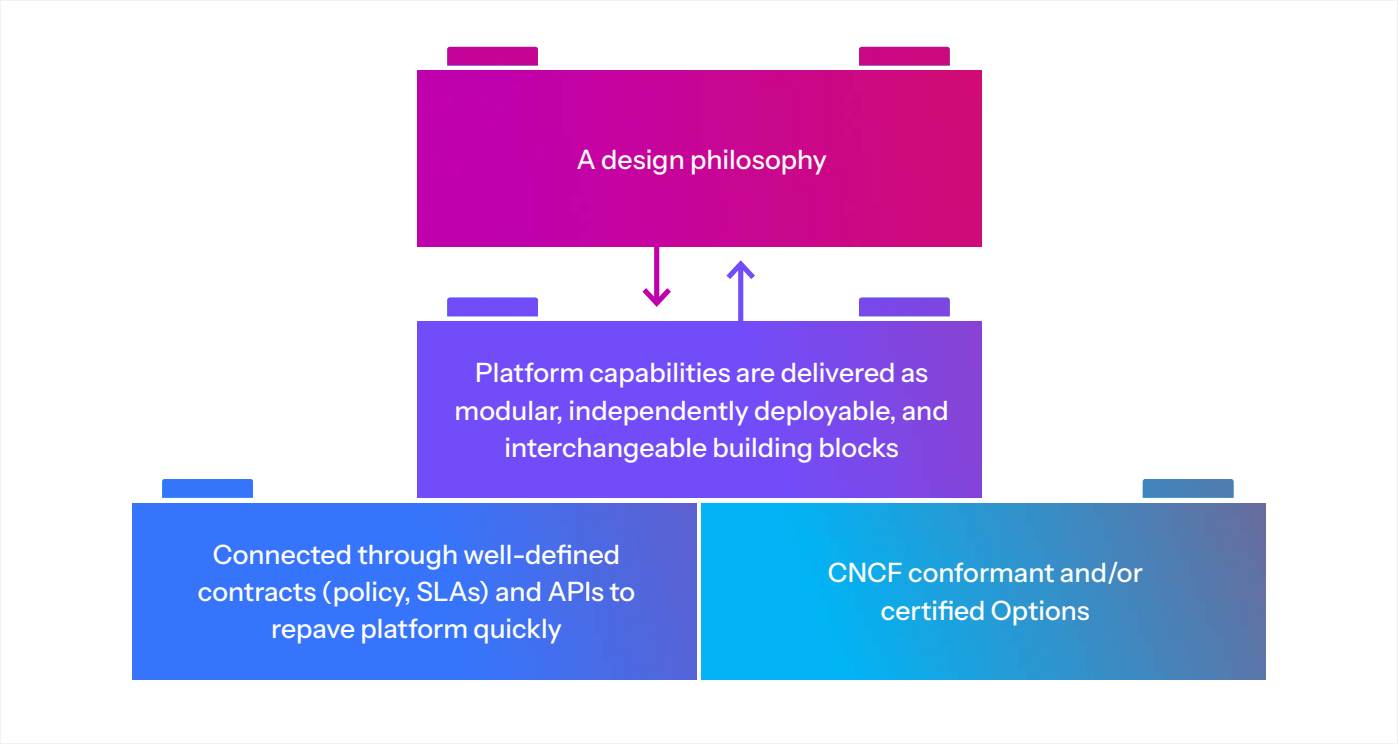
Composable by design

The future of platform engineering is not build versus buy. It is compose. Composable architecture delivers the agility of best-of-breed selection with the consistency of a unified platform through modular, independently deployable building blocks connected by well-defined APIs and contracts.



The five composable layers

A composable platform architecture treats each capability as a discrete, independently deployable, versioned module with a well-defined API contract. Modules can be swapped, Jenkins replaced by Harness, one observability stack replaced by another, without cascading changes to dependent systems. This is made possible by API-first contracts that decouple producers from consumers, allowing each layer of the platform to evolve independently while maintaining interoperability.



A composable platform organizes into five independently evolvable layers:

01 **Experience layer**

Developer and other persona portals, CLI tools, self-service UIs and more. Each persona's interface evolves on its own roadmap without blocking others.

02 **Orchestration layer**

Workflow engines, policy enforcement, GitOps pipelines and more. The logic layer that connects experience to capability.

03 **Capabilities layer**

Modular services for CI/CD, observability, security, LLMs and more. This layer maps directly onto the canonical Tooling-layer of platform Developer Control, Integration & Delivery, Resource, Security, Observability

04	Integration layer	APIs, event buses, service mesh. The contracts that decouple producers from consumers.
05	Infrastructure layer	Compute, GPUs, storage, networking. The substrate that all layers depend on.

It's crucial to understand that composability starts at the infrastructure layer. An infrastructure foundation built from independently deployable, API-first building blocks, compute, GPU, storage, and networking, is what makes the five composable layers above it genuinely swappable.

Four design principles

Four design principles turn composability from an aspiration into an operating discipline. Modular by design means every capability is a discrete, versioned building block. API-first contracts give those blocks standard interfaces that decouple producers from consumers. Plug and play lets teams swap implementations without disrupting dependents. Progressive migration means adoption happens at each team's own pace. Together they let the platform evolve continuously while staying interoperable.

Modular by Design

- Each capability is a discrete, versioned module
- Select best of breed, Vendor-agnostic layers e.g. O11Y
- Platform built from independent, self-contained, reusable building blocks
- State is kept at control plane level for easier reparability

Time Horizon: Existing

API First Contracts

- Standard interfaces decouple producers from consumers
- Humans and AI agents consume the same APIs; Built MCP servers around standard APIs
- Orchestration: Low-code workflows link modules in days, not months. Declarative config replaces custom integration code
- Support Open source API

Time Horizon: Now

Plug and Play

- Swap implementations without disrupting dependents
- Faster M&A integration. Plug into API contracts via adapters.
- Custom Differentiation: Build only what's unique: your golden paths, governance, DevEx layer. Outsource commodity
- Support Open source API

Time Horizon: Now

Progressive Migration

- Deploy GitHub Actions alongside Jenkins as an independent service
- Teams opt in at own pace
- Guided by comparative DORA telemetry

Time Horizon: Near Future

01 Modular by design

Each capability is a discrete, versioned module. State is kept at the control plane level for easier repavability. Vendor-agnostic layers prevent lock-in at any single component. Platform teams build only what is genuinely differentiating - golden paths, governance, the DevEx layer - and outsource the commodity.

02 API-first contracts

Standard interfaces decouple producers from consumers. Humans and AI agents consume the same APIs, which means MCP gateways built around standard APIs give agents the same access surface that human tooling uses. Declarative configuration replaces custom integration code. Orchestration links modules in days, not months.

03 Plug and play

Swap implementations without disrupting dependents. Deploy GitHub Actions alongside Jenkins as an independent service and let teams opt in at their own pace, guided by comparative DORA telemetry. Faster M&A integration becomes a side effect of good composable design - new teams plug into API contracts via adapters rather than requiring platform rebuilds.

04 Progressive migration

No composable architecture requires a big-bang cutover. Deploy new capabilities alongside existing ones. Let adoption be intrinsic - teams move when the new capability demonstrably outperforms the old one, and the telemetry makes that comparison visible.

Distributed architecture and guardrails

Composability without governance produces fragmentation. A distributed architecture pattern prevents the platform team from becoming the bottleneck while keeping thousands of developers moving safely. Specialized portals share a common foundation, so security, data, ML, FinOps, and development teams each get purpose-built interfaces, all consuming the same underlying platform APIs. No platform roadmap bottleneck means teams stop competing for the platform team's attention.

Guardrails prevent decentralization from becoming chaos. Policy-as-code and admission controllers enforce consistency at the source. Governance becomes an immutable contract - codified and enforced at build and deploy time, tamper-evident rather than bolted on.

Immutable infrastructure completes the picture. Environments are rebuilt from versioned images rather than patched in place, eliminating configuration drift and snowflake systems. Declarative state in Git means every component - infrastructure, policies, pipelines, AI configurations - is versioned, auditable, and rollback-ready. For AI workloads specifically, model versions, prompts, and RAG indices become immutable artifacts with full lineage tracking, which is essential for private AI auditability and regulatory compliance.

Composable

Modular Design

API-First Contracts
Standardized interfaces (MCP) decouple producers from consumers.

Layered Independence
Swap implementations (CI/CD, O11y) without cascading changes.

Plug-and-Play
Use CNCF-conformant blocks for fast M&A and customization.

Immutable

Disposable Execution

Replace, Never Patch
Eliminate configuration drift by rebuilding from versioned images.

GitOps Declarative State
Every component (Infra, Policy, AI) versioned in Git.

AI/ML Reproducibility
Model versions and RAG indices treated as immutable artifacts.

Strategic Value

Repavable Foundations

Impact
Transition from "TicketOps" to Intent-Driven Delivery.

Enable scale for 1,000+ developers with built-in guardrails.

Disposable Execution

A resilient, automated infrastructure that treats every component as a versioned building block, enabling rapid repavability and developer velocity.

The CNCF ecosystem provides the open-source foundation for composability across all five layers. CNCF-conformant and certified options exist at every layer, and the ecosystem's growth from roughly 50 projects in 2018 to more than 200 today is itself evidence that composability is the direction the industry has chosen.

How the pillars compound

The five pillars are not five independent initiatives. They compound, and the compounding is where the 2.0 evolution becomes more than the sum of its parts.

Consider what happens when an AI workload lands on a modern platform. The AI-Native Platform pillar provides GPU provisioning, model registries, and MCP gateways through the Tooling layer. The Multi-Persona Experience pillar gives the data scientist a self-service interface that matches their workflow, while giving the security team direct access to the inference audit trail. The Embedded FinOps pillar surfaces GPU and token cost at provision time, with cost gates that prevent runaway spending. The Security Shifts Down pillar embeds model governance, prompt security, data isolation, and inference governance at the platform layer. The Composable Architecture pillar means the team can swap model serving implementations without rebuilding the experience layer.

The compounding effect is what justifies the investment. Each pillar individually delivers value. Together, they change what the platform makes possible. Organizations that treat the five pillars as five separate projects miss the compounding. Organizations that treat them as five faces of a single platform evolution capture it.

> The most successful platform teams in our community treat Platform Engineering 2.0 as a single strategic evolution, not a portfolio of five projects. The pillars compound, but only when they are designed and funded together. Treat them as five separate initiatives and you get a fraction of the value. Treat them as one and you get the leverage that will define the next decade of the discipline.

Luca Galante

CORE CONTRIBUTOR, PLATFORMENGINEERING.ORG

Next steps

Platform engineering 2.0 is a direction, not a destination. Use these prompts to identify where to focus first.

For platform teams

Audit your platform against the five pillars. Which platform engineering ceiling is creating the most friction today: AI-blind architecture, developer-only focus, a reactive rather than proactive posture, golden paths that have become golden cages, or a rigid and static platform? Start with the pillar that addresses your most urgent pressure, because that is where the platform team will have exec sponsorship and clear ROI. You don't need to evolve all five simultaneously. Begin where the pressure is highest.

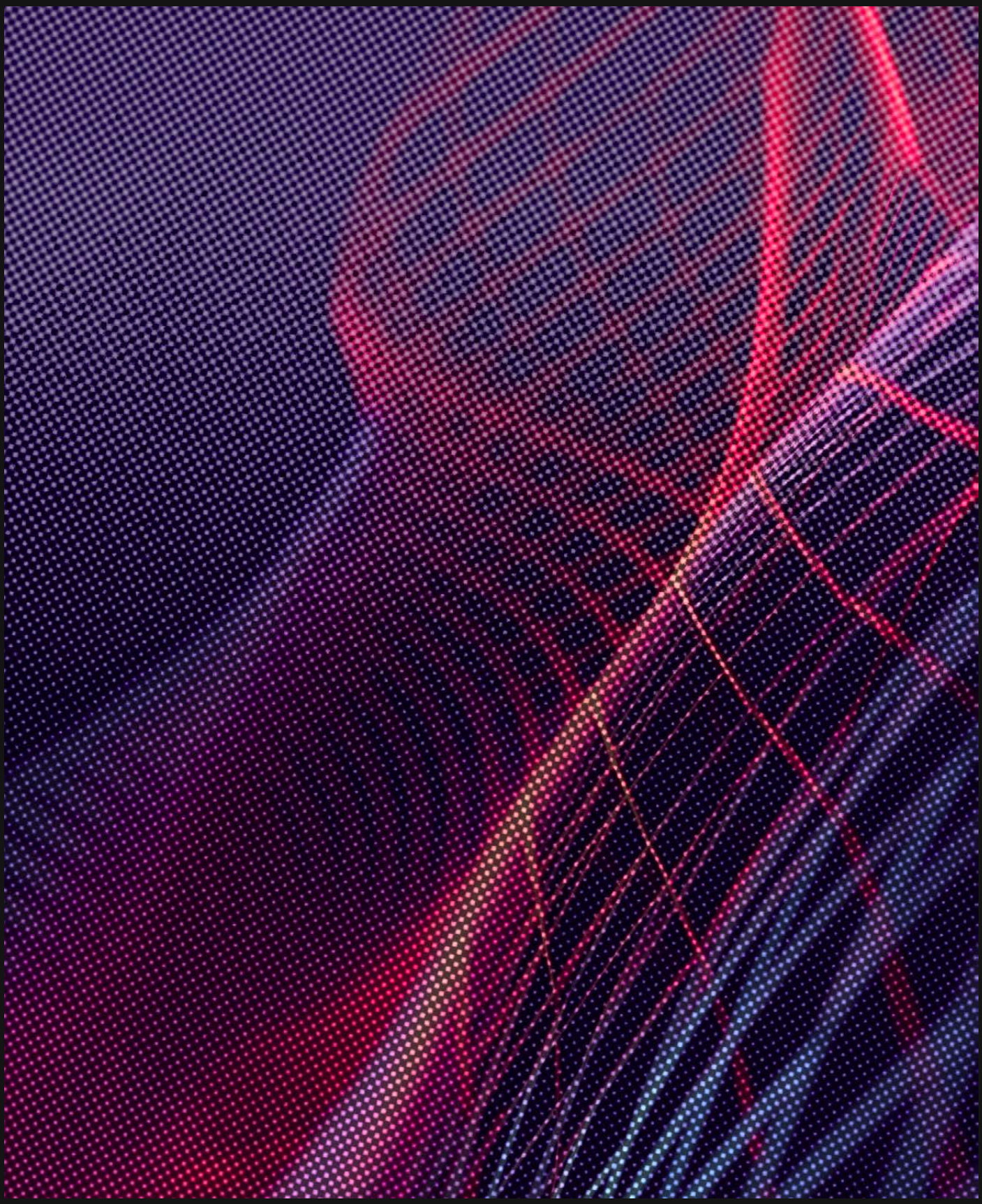
For executives

The 12-month milestone that matters most is AI-native platform readiness, including GPU workload support, non-human identity, and MCP gateway exposure. Mandate a platform engineering 2.0 business case tied to developer velocity, cloud cost reduction, and AI readiness metrics. Establish a Platform P&L with FinOps-grade cost attribution.

> Platform Engineering 1.0 solved the developer productivity problem. Platform Engineering 2.0 must solve the enterprise intelligence problem, supporting AI workloads, multi-persona teams, and autonomous agents as first-class infrastructure consumers. Organizations that do not integrate agentic AI directly into the workflows and control plane will fall significantly behind.

Roy Illsley
CHIEF ANALYST, OMDIA

Conclusion



Platform engineering 2.0 as the direction for the AI era

Every discipline reaches a moment where the foundations it has built are asked to carry more than they were designed for. Platform engineering is at that moment now. The foundations of the discipline established between 2019 and 2025 - Platform as Product, golden paths, self-service IDPs, are not finished work. They are the substrate that everything in this whitepaper builds on. The Agentic Development Platform inherits the IDP. The five new pillars all rise from Platform as Product. The discipline does not reset at this inflection point; it accumulates.

What is changing is the load. AI workloads need substrate that containerised IDPs were never designed to provide. Autonomous agents are arriving as users with no prior persona to inherit from. Cost is moving to a defining platform signal. Security is becoming structurally unanswerable at the developer layer. And the platform engineering team is being asked to serve not just one persona but the whole organisation. None of these forces will retreat. And, the platforms that extend their foundations to answer them will succeed, while the platforms that do not will be routed around by the teams whose work they were meant to enable.

Platform engineering 2.0 is the shape of that extension. AI-native, multi-persona, cost-aware, secure by construction, composable by design. Not five projects, but five faces of a single evolution in what the platform is for. Platform engineering is becoming the discipline that builds the operational foundation for the entire enterprise's agentic future, not just its software delivery pipeline. That is a larger remit than the discipline has ever held, and it is built on the work platform engineering teams have already done.

Platform engineers have always been the masters of infrastructure. It was the first sub-discipline of platform engineering, and today that is where immense opportunity sits for your platform team. Infrastructure is the foundation on which Platform Engineering 2.0 stands, and the determinant of how far that evolution can reach. Organizations that invest in modern, composable, AI-ready infrastructure give their platform teams the substrate to deliver across every pillar: AI-native workloads that provision on demand, FinOps controls that act at the moment of provisioning, security that is immutable by construction, and composable architectures that evolve as fast as the AI landscape demands. The platform team that masters infrastructure is the platform team that masters the AI era, and that mastery begins not with tooling choices or operating models, but with a deliberate, strategic commitment to infrastructure as a first-class platform concern.

The teams that move now will define what that remit looks like. They will be the reason their organisations succeed with AI. The window is open and the foundations are already in place. What remains is to extend them.

The discipline is evolving. And teams that evolve with it will define what platform engineering means for the next decade.

POWERED BY

BROADCOM

Broadcom's VMware Cloud Foundation (VCF) is the platform for the modern private cloud, delivering a single unified platform for VMs and containers that supports all applications, traditional, modern, or AI, with a consistent operating model, governance, and controls spanning data centers, edge, and managed cloud infrastructure. VCF combines the agility and scalability of public cloud with the security, performance, architectural control and total cost of ownership (TCO) benefits of an on-premises environment. VCF frees development teams to focus on applications instead of infrastructure. Through the native VMware vSphere Kubernetes Service (VKS), an integrated CNCF-certified Kubernetes runtime, platform engineering teams can support agile modern app development directly from the private cloud. GitOps-driven, self-service infrastructure with guardrails balances developer autonomy with IT control and reduced risk, while multi-tenant, policy-driven delivery and built-in observability enable more secure and visible applications that are always in their desired state.

Platform Engineering

Platform Engineering is the world's largest platform engineering community and is the global home for platform engineers. What began as a handful of Platform Engineering Meetup groups across Europe has grown into a worldwide movement spanning 50+ local chapters, a Slack community of 30,000+ practitioners, the Platform Weekly newsletter, and PlatformCon, the largest platform engineering gathering of the year, where 50,000+ practitioners join for free from around the globe. The community created and maintains the shared frameworks, blueprints, and best practices that turned platform engineering from an unnamed pattern into the operating model for the modern enterprise, now reinforced by Platform Engineering University and its thousands of certified practitioners. Weave Intelligence is the community's independent research arm, combining senior analysts with industry experts and enterprise leaders to produce leading platform engineering research, including the annual State of Platform Engineering Report, reference architectures for Internal Developer Platforms on AWS, Azure, and GCP, and ongoing market guides across observability, security, and FinOps. Together they give practitioners and enterprise leaders the trusted insight and shared standards needed to navigate this shift with confidence.

About Weave Intelligence

Weave Intelligence

Weave Intelligence is a leading analyst firm specializing in platform engineering. By uniting a team of senior analysts with industry experts and enterprise leaders, we deliver the rigorous research that defines the field.

We enable organizations to leverage the #1 trend in IT as the modern framework for operational excellence and innovation.

Weave Intelligence GmbH
Wöhlertstraße 12-13
10115 Berlin

Disclaimer

Weave Intelligence does not endorse any specific vendor, product, or service. The information contained in this report has been obtained from sources believed to be reliable. Weave Intelligence disclaims all warranties as to the accuracy, completeness, or adequacy of such information. This publication is provided on an “as-is” basis without warranty of any kind, either express or implied. Weave Intelligence shall have no liability for errors, omissions, or inadequacies in the information contained herein or for interpretations thereof.