PLATFORM ENGINEERING IN THE AI ERA

BUILDING DEVELOPER PLATFORMS FOR CLOUD-NATIVE AND AI-NATIVE ORGANIZATIONS

- **ACCELERATE** DEVELOPER VELOCITY
- **EMBRACE** CLOUD-NATIVE ARCHITECTURES
- **LEVERAGE** AI INFRASTRUCTURE AND TOOLS
- **BUILD SECURE** AND RELIABLE PLATFORMS
- **AUTOMATE** EVERYTHING THAT SCALES
- **DELIVER INSIGHTS** THROUGH OBSERVABILITY

JASON RIDLEY

FOR ENGINEERS. FOR PLATFORMS. FOR THE FUTURE.

Platform Engineering in the AI Era

Building Developer Platforms for Cloud-Native and AI-Native Organizations

Steve Publications

This book is available at https://leanpub.com/platformengineeringintheaiera

This version was published on 2026-07-31

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

Building Developer Platforms for Cloud-Native and AI-Native Organizations

This book provides a comprehensive guide to designing, building, operating, and evolving modern internal developer platforms for organizations at scale. It covers the full spectrum from foundational DevOps practices through cloud-native architectures to AI infrastructure, offering technical depth, architectural patterns, implementation guidance, and strategic frameworks for senior engineers, platform teams, cloud architects, engineering managers, and technology leaders navigating the transition to platform-centric software delivery in an AI-driven world.

Introduction: The Platform Imperative

The Developer Productivity Crisis

Consider a mid-size fintech company in 2024. It has three hundred engineers, forty product teams, and a cloud bill approaching two million dollars per month. Every team wants to deploy faster, iterate quicker, and ship features that keep customers engaged. But the reality is something different. Developers spend more than thirty percent of their time waiting: waiting for access approvals, waiting for environments to be provisioned, waiting for infrastructure tickets to be resolved, waiting for compliance reviews. A new engineer joining the company needs three weeks before they can run their first service in production, because onboarding involves navigating a maze of internal wikis, shadow IT tools, tribal knowledge, and inconsistent processes that differ from team to team.

This is not an isolated case. It is the story of hundreds of engineering organizations struggling with what has become known as the developer productivity crisis. The problem is not that developers are slow. The problem is that complexity has grown faster than our ability to manage it. Cloud platforms now offer hundreds of services. Kubernetes clusters run thousands of containers across dozens of namespaces. Microservice architectures span multiple regions and availability zones. Security requirements demand continuous scanning, policy enforcement, and audit trails. Regulatory compliance requires immutable logs, data residency controls, and separation of duties. Add to this the explosion of AI workloads: training runs consuming GPU clusters for days, inference services requiring sub-millisecond latency, vector databases storing billions of embeddings, and autonomous agents calling internal APIs with unpredictable patterns.

The result is cognitive overload for every developer in the organization. They are expected to be experts in their application domain, plus Kubernetes, plus Terraform, plus CI/CD pipelines, plus observability stacks, plus security scanning, plus cost optimization, plus AI model serving, plus whatever else the platform team has bolted on. The original DevOps promise of "you build it, you

run it" has quietly morphed into something closer to "you build it, you burn out."

Platform engineering emerged as a response to this crisis. It is the discipline of productizing infrastructure to enable developer autonomy within guardrails. Instead of asking every team to assemble their own delivery pipeline from scratch, platform teams build curated, self-service capabilities that abstract away operational complexity. Developers get what they need through golden paths, templates, and portals rather than through tickets, tribal knowledge, and ad hoc arrangements. The platform becomes a product, developers become its customers, and the goal is to make the right way to do things the easiest way to do things.

Complexity as the Enemy

Complexity is not inherently bad. Software systems are complex because they solve complex problems. But unmanaged complexity is destructive. It slows delivery, increases defects, creates security gaps, and makes organizations brittle in the face of change. The challenge for modern engineering organizations is that complexity has grown in multiple dimensions simultaneously: architectural, organizational, operational, and now AI-related.

Architectural complexity comes from the shift to cloud-native patterns. Monolithic applications have given way to microservices, event-driven architectures, and distributed systems that span multiple clouds and regions. Each service needs its own deployment configuration, networking rules, scaling policies, and observability setup. The number of moving parts has multiplied, and so has the surface area for failure.

Organizational complexity comes from scale. As engineering organizations grow beyond a handful of teams, coordination becomes expensive. Shared services become bottlenecks. Different teams adopt different tools and practices, creating fragmentation. Security and compliance requirements add layers of review and approval that slow down delivery. The friction between the need for standardization and the need for team autonomy creates constant tension.

Operational complexity comes from running production systems at scale. Thousands of containers, hundreds of microservices, petabytes of data, millions of API calls per minute. Keeping all of this observable, reliable, secure,

and cost-efficient requires specialized expertise that most product teams do not have and should not need to develop.

And now AI complexity has arrived. AI workloads are fundamentally different from traditional application workloads. Training jobs consume massive GPU clusters for hours or days, with specific requirements for high-bandwidth networking and distributed storage. Inference services need sub-millisecond latency and autoscaling that responds to traffic spikes in seconds. Vector databases store and query billions of high-dimensional embeddings. Large language models require careful prompt management, evaluation frameworks, safety guardrails, and cost controls to prevent runaway token consumption. Autonomous agents call internal APIs with unpredictable patterns and need tool access, memory systems, and human-in-the-loop safeguards.

Every one of these complexity dimensions adds cognitive load to developers. Every additional tool, configuration file, approval process, or infrastructure decision they need to make is time they are not spending building product features. Platform engineering addresses this by creating abstractions that reduce cognitive load while preserving the flexibility teams need to innovate.

From DevOps to Platform Engineering

To understand where platform engineering came from, we need to trace the evolution of how software organizations have managed infrastructure over the past two decades.

Before DevOps, infrastructure was a siloed function. Operations teams owned the servers, networks, and deployment processes. Developers wrote code and threw it over the wall to ops, who would schedule deployments during maintenance windows, often weeks or months apart. This model worked for small organizations with simple applications, but it broke down as software became more central to business outcomes and the pace of change accelerated. The friction between development and operations created bottlenecks, blame games, and slow time to market.

The DevOps movement, which gained momentum in the late 2000s and early 2010s, sought to break down these silos through cultural change, automation, and shared responsibility. DevOps introduced practices like continuous integration and continuous delivery, infrastructure as code, and monitoring. The mantra “you build it, you run it” encouraged developers to take ownership

of their software through the entire lifecycle, from development to production operations. This model worked remarkably well for organizations that adopted it early, particularly tech companies with strong engineering cultures and relatively homogeneous workloads.

But DevOps hit a scaling wall in large, diverse organizations. The assumption that every developer could and should be an expert in infrastructure, networking, security, and operations proved unrealistic. In practice, DevOps created cognitive overload for application developers who now had to master dozens of tools and technologies just to deploy their services. It also led to fragmentation: each team built its own pipelines, adopted different tools, and developed inconsistent practices. Security and compliance became harder to enforce across a decentralized landscape. And the operational burden on product teams increased significantly, pulling focus away from building features and toward managing infrastructure.

Site Reliability Engineering, pioneered at Google and popularized through the 2016 SRE book, offered a different approach. SRE applied software engineering principles to operations problems, introduced concepts like service level objectives and error budgets, and emphasized automation to eliminate toil. SRE teams partnered with product teams to ensure reliability while enabling velocity. But even SRE, as originally conceived, assumed a certain level of infrastructure expertise within product teams and did not fully address the fragmentation problem in large organizations.

Platform engineering emerged in the early 2020s as the next evolution. It combined the cultural and automation principles of DevOps with the reliability focus of SRE, but added something new: a product mindset applied to internal infrastructure. Instead of expecting every team to assemble and maintain its own delivery pipeline, dedicated platform teams build curated, self-service capabilities that abstract away operational complexity. The platform becomes a product with developers as customers. It provides golden paths that make the right way to do things the easiest way. It enforces security, compliance, and reliability standards by default rather than through manual review. And it enables developer autonomy within guardrails, giving teams the flexibility to innovate while maintaining organizational consistency.

Gartner named platform engineering a Top 10 Strategic Technology Trend for 2024 and has since tracked its evolution through dedicated Hype Cycles and Strategic Trends reports. Gartner projects that by 2026, 80 percent of large software engineering organizations will establish platform teams as internal

providers of reusable services, up from 45 percent in 2022 [1]. The DORA Accelerate State of DevOps Report 2024, based on a survey of over 39,000 professionals globally, found that platform engineering improves delivery performance but can decrease stability if not carefully implemented alongside foundational practices like small batch sizes and robust testing [2]. These findings reflect both the mainstream adoption of platform engineering and the caution that infrastructure centralization alone does not guarantee success.

What This Book Will Teach You

This book is designed for senior software engineers, platform engineers, cloud architects, engineering managers, CTOs, and technology leaders at organizations with significant cloud adoption, multiple engineering teams, stringent security and compliance requirements, and growing AI workloads. If you are responsible for designing, building, or evolving a modern internal developer platform, this book will provide the knowledge and frameworks you need.

The book is organized as a progressive journey from foundations to advanced topics. You do not need to read it cover to cover in order, but each chapter builds on concepts introduced earlier, so reading sequentially will give you the most complete understanding.

We begin by tracing the historical evolution from traditional infrastructure teams through DevOps and SRE to modern platform engineering, establishing what platform engineering is and is not, and defining its core principles. We then explore how platform teams are organized, their relationships with product teams, and the organizational patterns that succeed or fail, drawing on the Team Topologies framework.

The book covers platform product management in depth, explaining how to treat the platform as a product with customers, roadmaps, metrics, and feedback loops. You will learn how to measure platform success beyond vanity metrics, prioritize features based on developer needs, and avoid common pitfalls that cause internal platforms to fail.

Technical chapters cover the core building blocks of modern platforms: Kubernetes and container orchestration, infrastructure as code and GitOps workflows, CI/CD pipelines optimized for scale, developer portals and self-service capabilities, golden paths and software templates, security and governance through policy as code, observability and reliability engineering, multi-cloud and hybrid cloud strategies, and FinOps practices for cost optimization.

The final section addresses the AI era specifically. We explore GPU infrastructure design, model training pipelines, vector databases, inference platforms, LLMOps practices for managing LLM workloads in production, and the emerging patterns for agentic AI systems. We examine how platforms must evolve to support AI-native development, including tool access patterns, safety guardrails, cost controls, and integration with AI-assisted coding tools.

Throughout the book, you will find architectural trade-offs explained, design decisions analyzed, implementation guidance provided, reference architectures described, decision frameworks offered, real-world scenarios discussed, code examples included, production best practices shared, common pitfalls flagged, anti-patterns identified, migration strategies outlined, operational considerations addressed, and lessons learned from industry experience distilled.

The goal is not to provide a tooling guide or a tutorial. The tools will change. The principles, patterns, and trade-offs will remain relevant for years to come. By the end of this book, you will understand how to design, build, operate, govern, and evolve a platform that enables developer productivity, reliability, security, scalability, compliance, cost efficiency, and AI-native software development at enterprise scale.

Chapter 1: Foundations: The Evolution of Platform Engineering

Before DevOps: The Infrastructure Silo

To understand platform engineering, we must first understand what came before it. In the early days of enterprise software, infrastructure was a distinct, siloed function. Operations teams owned the servers, networks, databases, and deployment processes. Developers wrote code in their environments and submitted requests to operations when they needed something deployed, scaled, or configured. This model had clear advantages: specialization allowed operations engineers to become deep experts in systems administration, and centralized control made it easier to enforce security policies and maintain stability.

The trade-offs were severe. Deployment cycles measured in weeks or months became the norm in many organizations. Developers had little visibility into how their code ran in production, leading to "it works on my machine" problems that only surfaced after deployment. The handoff between development and operations created friction, blame games, and misaligned incentives: developers were rewarded for shipping features quickly, while operations was measured on uptime and stability. When incidents occurred, each side pointed fingers at the other.

This siloed model worked adequately for organizations with simple applications running on a handful of servers, but it broke down as software became more central to business outcomes. The rise of the internet created competitive pressure to ship faster. Companies like Amazon, Netflix, and Etsy demonstrated that rapid deployment cycles could be a source of competitive advantage. The old model simply could not keep up.

The DevOps Movement and Its Limits

The DevOps movement emerged in response to these limitations. The term was coined by Patrick Debois in 2009, but the ideas had been brewing for years in organizations that had already started breaking down the wall between development and operations. DevOps was not a specific tool or technology. It was a cultural and organizational movement built on three pillars: culture, automation, and measurement.

Culturally, DevOps sought to replace silos with shared responsibility. The mantra "you build it, you run it" encouraged developers to take ownership of their software through the entire lifecycle, from development to production operations. This alignment of incentives reduced blame games and created a shared focus on delivering value to users.

Technically, DevOps introduced automation for everything that was repetitive, manual, or error-prone. Continuous integration ensured that code changes were automatically built and tested. Continuous delivery automated the path from code commit to production deployment. Infrastructure as code, pioneered by tools like Chef, Puppet, and later Terraform, allowed infrastructure to be version-controlled, reviewed, and deployed using the same practices as application code. Monitoring and logging provided visibility into production systems.

The results were dramatic for organizations that adopted DevOps early. Deployment frequency increased from monthly or quarterly cycles to multiple deployments per day. Lead times for changes dropped from weeks to hours. Mean time to recovery from failures improved significantly. These improvements were documented in the Accelerate State of DevOps Reports, which have tracked software delivery performance since 2014 and established four key metrics: deployment frequency, lead time for changes, change failure rate, and mean time to recovery.

But DevOps had limits that became apparent as organizations scaled. The assumption that every developer could and should be an expert in infrastructure, networking, security, and operations proved unrealistic. In practice, DevOps created cognitive overload for application developers who now had to master dozens of tools and technologies just to deploy their services. A developer joining a DevOps-mature organization faced a steep learning curve: Kubernetes for orchestration, Terraform for infrastructure, Prometheus for

monitoring, Grafana for dashboards, ArgoCD for GitOps, various security scanning tools, cloud provider consoles, and whatever else had been bolted on over time.

DevOps also led to fragmentation in large organizations. Each team built its own pipelines, adopted different tools, and developed inconsistent practices. Security and compliance became harder to enforce across a decentralized landscape where each team had autonomy over their tooling choices. The operational burden on product teams increased significantly, pulling focus away from building features and toward managing infrastructure. And the original promise of DevOps, that shared responsibility would create alignment, often devolved into developers doing ops work without having the expertise or desire to do it well.

Site Reliability Engineering and the SRE Model

Site Reliability Engineering emerged at Google in the late 1990s and was popularized through the 2016 book "Site Reliability Engineering" by Beyer, Jones, Petoff, and Murphy. SRE was not a replacement for DevOps. It was a specific implementation of DevOps principles, codified into a discipline with concrete practices and metrics.

SRE applied software engineering principles to operations problems. Instead of relying on manual procedures and tribal knowledge, SREs wrote code to automate operational tasks. The goal was to eliminate toil: work that was repetitive, manual, tactical, and offered no lasting value. Every operational task should be automated until the cost of automation exceeded the cost of doing it manually.

SRE introduced several concepts that became foundational to modern platform engineering. Service level objectives defined quantitative targets for reliability, latency, and throughput. Error budgets represented the acceptable amount of unreliability, creating a data-driven framework for balancing reliability and velocity. When error budgets were healthy, teams could ship features aggressively. When they were exhausted, teams focused on reliability improvements. This framework replaced subjective debates about "how reliable is reliable enough" with objective metrics that aligned engineering and product decisions.

SRE also emphasized observability: the ability to understand a system's internal state from its external outputs. Instead of relying on pre-defined alerts

and dashboards, SREs built systems that could answer arbitrary questions about production behavior. This required comprehensive logging, metrics, and distributed tracing, plus the cultural practice of using observability data to drive decisions rather than just to react to incidents.

The SRE model worked remarkably well at Google and other organizations that adopted it. But like DevOps, it had scaling challenges. The expertise required to be an effective SRE was deep and specialized. Not every organization could staff dedicated SRE teams, and product teams without SRE support often lacked the reliability practices needed to run production systems safely. The model also assumed a certain level of infrastructure maturity that many organizations did not have.

Platform Engineering as Product Thinking

Platform engineering emerged in the early 2020s as a response to the scaling challenges of DevOps and SRE. It combined the cultural and automation principles of both disciplines but added something new: product thinking applied to internal infrastructure.

The core insight was simple. In large organizations with many engineering teams, expecting every team to assemble and maintain their own delivery pipeline created duplication, inconsistency, and cognitive overload. A better approach was to treat infrastructure as a product: build it once, make it self-service, and let teams consume it without needing deep infrastructure expertise.

Platform engineering is the discipline of building and maintaining internal developer platforms (IDPs) that provide self-service capabilities for provisioning, testing, deploying, and operating software. The CNCF defines platform engineering as focused on "building and maintaining software development platforms that provide self-service for developer teams, offering the necessary infrastructure for provisioning, testing, deployment, and rollback" [3]. Microsoft describes it as combining "a product mindset with learnings from DevOps and DevSecOps to deliver a set of tools that provide sufficient automation, tracking, governance, and observability" [4].

The platform becomes a product with developers as customers. This is not metaphorical. Successful platform teams apply the same product management practices used for external products: customer discovery to understand

developer needs, roadmaps that communicate vision and priorities, iterative delivery based on feedback, and metrics that measure outcomes rather than outputs. Developers are treated as voluntary customers who choose to use the platform because it makes their lives easier, not because they are forced to.

This product mindset addresses several problems that DevOps and SRE left unresolved. It reduces cognitive load by abstracting away infrastructure complexity behind well-designed interfaces. It eliminates duplication by providing shared capabilities that all teams can use. It enforces consistency through standardized pipelines, templates, and guardrails. And it enables specialization: platform engineers focus on building and maintaining the platform, while product engineers focus on building features for end users.

The shift from DevOps to platform engineering is not a rejection of DevOps principles. It is an evolution that recognizes what works at scale. DevOps culture (collaboration, shared ownership, automation, continuous improvement) remains essential. But the implementation model changes: instead of every team doing DevOps, dedicated platform teams build the infrastructure and tooling that enables DevOps practices across the organization.

Core Principles of Modern Platforms

Modern internal developer platforms are built on a set of core principles that distinguish them from traditional infrastructure teams and ad hoc tooling arrangements. Understanding these principles is essential for designing and operating effective platforms.

Self-service over ticket-based operations. The defining characteristic of a platform is self-service. Developers should be able to provision resources, deploy services, and access tools without submitting tickets or waiting for approval from a centralized team. This does not mean no governance. It means governance is automated and embedded into the platform through policy as code, guardrails, and approved templates. When developers request a new service, the platform provisions it automatically within defined boundaries, rather than requiring manual review and provisioning.

Golden paths over unlimited choice. Platforms provide golden paths: recommended, standardized ways to accomplish common tasks like spinning up services, configuring CI/CD, or deploying to production. These paths

bundle together templates, tooling, configurations, and best practices into opinionated workflows that make the right way to do things the easiest way. Golden paths are not mandatory, but they are designed to be so convenient and well-integrated that most teams choose them voluntarily. This balances standardization with flexibility: teams can go off-path when they have legitimate needs, but the default is consistency.

Developer experience as a first-class concern. The platform is designed for its users, not for the convenience of the platform team. Every interface, workflow, and tool is evaluated through the lens of developer experience. Is it discoverable? Is it intuitive? Does it reduce cognitive load or add to it? Does it get developers what they need quickly, or does it create friction? Platform teams measure developer satisfaction and use that feedback to drive improvements, just as product teams use customer feedback to improve external products.

Automation over manual processes. Manual processes do not scale. Every operational task that is repetitive, predictable, and rule-based should be automated. This includes provisioning infrastructure, deploying applications, running security scans, enforcing policies, collecting metrics, and responding to common incidents. Automation reduces errors, speeds up delivery, and frees engineers to focus on work that requires human judgment and creativity.

Security and compliance by default. Security and compliance are not bolted on after development. They are built into the platform from the start. Golden paths include security scanning, policy enforcement, secrets management, and audit logging. Developers do not need to be security experts to ship secure code, because the platform enforces security standards automatically. This is often called "secure by default" or "security as a service."

Observability and reliability baked in. Every service deployed through the platform includes observability: metrics, logs, and traces that make it visible and debuggable. Reliability patterns like health checks, retries, circuit breakers, and graceful degradation are included in templates. Service level objectives are defined and monitored. Developers do not need to be reliability experts to ship reliable services, because the platform provides the tools and patterns to do so.

Cost transparency and accountability. The platform makes cost visible. Teams can see how much their services cost to run, broken down by resource type, environment, and team. This enables cost optimization and creates

accountability for infrastructure spending. Cost information is integrated into developer workflows, so engineers can make cost-aware decisions without needing separate tools or processes.

Evolution over perfection. Platforms are never finished. They evolve continuously based on feedback, changing needs, and emerging technologies. Platform teams start with a minimal viable platform that solves the most pressing problems, then iterate based on usage data and developer feedback. This avoids the trap of building a comprehensive platform in isolation that no one uses when it is finally released.

These principles form the foundation for everything that follows in this book. Every architectural decision, tooling choice, and organizational pattern should be evaluated against them. A platform that violates these principles may look impressive technically, but it will fail to deliver on the core promise of platform engineering: enabling developer productivity through well-designed, self-service infrastructure.

Chapter 2: Platform Team Topologies and Organization

The Four Team Topology Patterns

How you organize platform teams matters as much as the technology choices you make. A brilliant platform will fail if it is built by a team that is structurally misaligned with the rest of the organization. Conversely, a modest platform can succeed if the team topology supports effective collaboration, fast feedback, and continuous improvement.

The Team Topologies framework, developed by Matthew Skelton and Manuel Pais, provides a vocabulary and set of patterns for organizing teams in software-intensive organizations. It identifies four fundamental team types: stream-aligned teams, platform teams, enabling teams, and complicated subsystem teams. Understanding these types and how they interact is essential for designing effective platform organizations.

Stream-aligned teams are the backbone of any software organization. They are aligned to a value stream that delivers direct customer value, with autonomy to execute without constant handoffs. A stream-aligned team might own a specific product, a business domain, or a user journey. They are cross-functional, containing the skills needed to deliver end-to-end: developers, testers, designers, and sometimes product management. Stream-aligned teams focus on shipping features that matter to users.

Platform teams exist primarily to support stream-aligned teams. They build internal products (tools, services, infrastructure, and capabilities) that accelerate delivery by reducing the cognitive load and operational burden on stream-aligned teams. A platform team might own the Kubernetes clusters, the CI/CD pipelines, the developer portal, the observability stack, or any combination of shared capabilities. The key characteristic is that platform teams provide self-service interfaces that stream-aligned teams consume without needing deep infrastructure expertise.

Enabling teams help stream-aligned teams overcome obstacles and develop new capabilities. They are temporary by design: once a capability has been transferred, the enabling team dissolves or moves to a different problem. An enabling team might help product teams adopt a new technology, migrate from a legacy system, or implement security practices. They are mentors and coaches, not long-term dependencies.

Complicated subsystem teams own domains that require significant specialized expertise. These are areas where deep domain knowledge is concentrated in a small number of people, and it would be impractical to distribute that knowledge across many teams. Examples include search infrastructure, database engines, or real-time bidding systems. Complicated subsystem teams provide well-defined interfaces that other teams consume, similar to platform teams, but they focus on technical complexity rather than broad enablement.

For most organizations building internal developer platforms, the relevant distinction is between stream-aligned teams and platform teams. Stream-aligned teams build product features for external customers. Platform teams build infrastructure products for internal customers: the stream-aligned teams. The relationship between these two team types determines whether a platform succeeds or fails.

Centralized Platform Teams and Their Limits

The most common pattern for platform engineering is a centralized platform team: a dedicated group of engineers who build and maintain shared infrastructure for the entire organization. This pattern has clear advantages. It concentrates expertise, eliminates duplication, enables standardization, and creates economies of scale. A single team can invest in deep Kubernetes expertise, build sophisticated CI/CD pipelines, implement comprehensive security controls, and maintain observability tooling that serves hundreds of developers across dozens of product teams.

But centralized platform teams also have well-documented failure modes that organizations must guard against.

The first is the bottleneck problem. When every infrastructure request flows through a single team, that team becomes a constraint on delivery velocity. Stream-aligned teams wait for the platform team to provision resources, fix

bugs, add features, or approve changes. The platform team becomes a ticket-processing operation rather than a product-building team, and delivery slows down across the organization. This is particularly acute when the platform team is under-resourced relative to demand, which is common because infrastructure work is often undervalued compared to feature development.

The second is the ivory tower problem. Platform teams that operate in isolation, building what they think developers need rather than what developers actually need, create platforms that are technically impressive but practically useless. Teams end up with sophisticated tooling nobody uses, golden paths that do not match real workflows, and templates that solve hypothetical problems instead of actual pain points. The result is low adoption, shadow IT, and frustration on both sides.

The third is the one-size-fits-all problem. A centralized platform team naturally gravitates toward standardization, which is good up to a point. But different product teams have different needs. Some work with high-latency batch processing; others need sub-millisecond real-time responses. Some are heavily regulated; others operate in fast-moving markets where speed matters more than compliance. A platform that optimizes for one type of workload may be inadequate or even harmful for another.

These problems are not inevitable, but they are common enough that organizations adopting a centralized platform team pattern should actively mitigate them. The antidote to bottlenecks is self-service: the platform should enable teams to get what they need without waiting for human intervention. The antidote to ivory tower thinking is customer discovery: platform teams must understand developer needs through direct observation and feedback, not assumptions. The antidote to one-size-fits-all is flexibility within guardrails: golden paths that cover common cases, but escape hatches for legitimate edge cases.

Embedded Platform Engineers

An alternative to centralized platform teams is embedding platform engineers within product teams. Instead of a separate team building shared infrastructure, individual platform specialists work directly with stream-aligned teams, helping them design, build, and operate their services while contributing to shared platform capabilities.

This pattern has advantages. Embedded engineers understand the specific needs of their product teams deeply, because they work alongside them daily. They can provide real-time guidance, unblock issues quickly, and ensure that platform practices are applied correctly in context. They also serve as a two-way communication channel, bringing platform concerns into product teams and product team needs back to the platform organization.

But embedding has trade-offs. It fragments platform expertise across many teams, making it harder to maintain consistency and share knowledge. Embedded engineers may become too focused on their local team's needs and lose sight of broader platform strategy. And it can be expensive: if you have twenty product teams and each needs a platform engineer, that is twenty specialized roles rather than a single platform team serving all of them.

The most effective organizations often use a hybrid approach. A central platform team defines standards, builds core capabilities, and maintains shared infrastructure. Embedded platform engineers or "platform champions" within product teams act as liaisons, helping their teams adopt platform practices while providing feedback to the central team. This combines the consistency and economies of scale of centralization with the contextual understanding and responsiveness of embedding.

Platform Communities of Practice

Communities of practice are voluntary groups of people who share a common interest or domain and come together regularly to learn, share knowledge, and solve problems. In the context of platform engineering, communities of practice bring together engineers across teams who are interested in infrastructure, DevOps, SRE, security, observability, or related topics.

Communities of practice serve several important functions. They disseminate knowledge: when one team discovers a best practice or solves a hard problem, the community ensures that learning spreads to other teams. They build relationships: engineers who participate in communities develop trust and communication channels that make cross-team collaboration easier. They influence culture: active communities create enthusiasm for platform practices and help normalize behaviors like automation, observability, and security-first development.

Communities of practice are particularly valuable in organizations without dedicated platform teams, where they can serve as an informal mechanism

for sharing infrastructure knowledge. They are also valuable alongside formal platform teams, providing a channel for feedback, ideas, and collaboration that complements the product-oriented relationship between platform teams and their customers.

The key to successful communities is voluntary participation and genuine value. Communities should not be mandated or forced. They should exist because people find them useful, not because leadership says they should. Meeting formats should be practical: short, focused sessions with actionable content, not abstract discussions or status updates. And communities should be facilitated, not managed: facilitators help keep things productive and inclusive, but they do not control the agenda or membership.

Scaling Platform Organizations

As organizations grow, platform teams must scale too. But scaling is not simply adding more people. It requires deliberate design of team structures, interfaces, and processes that support growth without creating bottlenecks, fragmentation, or bureaucracy.

A common pattern for scaling is to break a monolithic platform team into smaller, focused teams organized around capabilities or domains. For example:

- A Kubernetes team owns cluster operations, networking, storage, and multi-tenancy.
- A CI/CD team owns build pipelines, deployment automation, and progressive delivery.
- A developer experience team owns the developer portal, golden paths, templates, and tooling integrations.
- A security team owns policy as code, secrets management, vulnerability scanning, and compliance automation.
- An observability team owns metrics, logging, tracing, alerting, and incident response tooling.
- An AI infrastructure team owns GPU clusters, model training pipelines, vector databases, and inference platforms.

Each team operates semi-autonomously, with its own roadmap and priorities, but coordinates with other platform teams to ensure consistency and integration. A lightweight platform leadership group (perhaps a platform

engineering manager, an architect, and representatives from each capability team) aligns strategy, resolves dependencies, and ensures that the platform evolves coherently rather than fragmenting into disconnected tools.

The number of platform engineers relative to product engineers is an important scaling consideration. There is no universal ratio that works for every organization, but research and industry experience suggest some guidelines. Organizations with mature platforms typically have between five and fifteen percent of their engineering headcount in platform roles. Below five percent, the platform team is likely under-resourced and becomes a bottleneck. Above fifteen percent, there is risk of over-engineering and building capabilities that product teams do not need or use.

The right ratio depends on several factors. Organizations with complex regulatory requirements, heterogeneous workloads, or legacy systems often need more platform engineering relative to product engineering, because the platform must handle more complexity and provide more abstractions. Organizations with homogeneous workloads, simple compliance needs, and greenfield architectures can often get by with fewer platform engineers, because standardization is easier and less abstraction is required.

Scaling also requires investment in automation and self-service. As the number of teams using the platform grows, manual processes become unsustainable. Every capability must be automatable and consumable without requiring human intervention from the platform team. This means investing in robust APIs, comprehensive documentation, intuitive interfaces, and reliable tooling that product teams can use independently.

Finally, scaling requires measuring what matters. Platform teams should track adoption metrics (how many teams are using the platform, how deeply), satisfaction metrics (developer surveys, NPS scores, qualitative feedback), and outcome metrics (impact on delivery performance, reliability, security, and cost). These measurements inform roadmap decisions, resource allocation, and organizational adjustments as the platform grows.

The goal of scaling is not to build the largest platform team possible. It is to build an organization structure that enables product teams to deliver value efficiently while maintaining standards for reliability, security, and compliance. The platform exists to serve stream-aligned teams, not the other way around. Every structural decision should be evaluated against that principle.

Chapter 3: Platform Product Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Treating Developers as Customers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Platform Roadmaps and Prioritization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Measuring Platform Success

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Feedback Loops and Continuous Improvement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Avoiding the Internal Tool Trap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 4: Organizational Change Management for Platform Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Why Technical Platforms Fail Without Organizational Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Hidden Costs of Resistance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Understanding the Roots of Resistance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Stakeholder Alignment Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Building a Change Coalition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Communication Strategies That Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Pilot Program Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Handling Shadow IT and Existing Tooling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Aligning Incentives and Performance Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Measuring Cultural Adoption Beyond Vanity Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Long-Term View: Continuous Cultural Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 5: Cloud-Native Foundations: Kubernetes and Containers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Why Kubernetes Became the Standard

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Cluster Architecture and Control Plane Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Workload Management: Pods, Deployments, and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Networking and Service Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Storage Abstractions and Data Persistence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Multi-Tenancy and Resource Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Service Mesh: Istio, Linkerd, and Traffic Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 6: Infrastructure as Code and GitOps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Declarative Infrastructure Paradigm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Terraform Ecosystem and Module Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Kubernetes-Native IaC: Crossplane and Helm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

GitOps Workflows with ArgoCD and Flux

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Policy Enforcement in the Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Drift Detection and Remediation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 7: CI/CD Pipelines and Developer Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Pipeline Architectures for Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Ephemeral Environments for Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Progressive Delivery: Canaries, Blue-Green, and Feature Flags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Test Automation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Optimizing Build Speeds at Enterprise Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

CI/CD for AI Workloads and Model Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Developer Experience as a First-Class Concern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 8: Developer Portals and Self-Service Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Role of the Developer Portal

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Software Catalogs and Service Inventory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Golden Paths and Software Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Self-Service Infrastructure Provisioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Balancing Standardization with Flexibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Platform APIs and Extensibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 9: Security, Governance, and Compliance by Default

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Security as a Platform Service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Policy as Code with OPA and Kyverno

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Identity, Access, and Secrets Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Software Supply Chain Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Compliance Automation and Audit Trails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Zero Trust Architecture Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Security for AI Workloads and Agentic Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 10: Observability, Reliability, and Resilience Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Three Pillars of Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Service Level Objectives and Error Budgets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Distributed Tracing at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Incident Response and Postmortem Culture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chaos Engineering and Resilience Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Resilience Patterns: Circuit Breakers, Bulkheads, and Retries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Event-Driven Architecture and Stream Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 11: Multi-Cloud, Hybrid Cloud, and Platform Portability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

When Multi-Cloud Makes Sense

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Hybrid Cloud Architecture Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Abstracting Provider-Specific Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Networking Across Cloud Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Residency and Sovereignty

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Migration Strategies Between Clouds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

AI Workload Portability Across Clouds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 12: FinOps and Platform Economics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The FinOps Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Cost Allocation and Showback Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Right-Sizing and Resource Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Spot Instances and Cost-Aware Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Platform ROI and Value Measurement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Building Cost Awareness into Developer Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

FinOps for AI and GPU Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 13: Data Platform Architecture for AI-Native Organizations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Why Data Platforms Are the Foundation of AI Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Platform Reference Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Ingestion Patterns: Batch, Streaming, and Hybrid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Lakehouse Table Formats: Iceberg, Delta Lake, and Hudi

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Feature Stores: Architecture and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Transformation: Batch ETL, Streaming, and dbt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Quality: Enforcing Trust at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Governance: Access Control, Lineage, and Compliance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Medallion Architecture Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Platforms Supporting AI Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 14: AI Infrastructure: GPUs, Vector Databases, and Model Serving

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

GPU Infrastructure and Cluster Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Training Pipelines and Distributed Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Vector Databases and Embedding Infrastructures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Model Serving and Inference Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

LLMOps: Managing LLM Workloads in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Cost Management for AI Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Data Platforms: Feature Stores, Lakehouses, and Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 15: Agentic AI Systems and AI-Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Architecting for Agentic Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Tool Use and API Access Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Safety, Guardrails, and Human-in-the-Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

AI-Assisted Software Development at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Platform Integration with AI Coding Assistants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Emerging Patterns and Unknowns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Chapter 16: Platform Automation, Orchestration, and Future Trends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Advanced Automation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Platform Orchestration and Control Planes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

WebAssembly and the Next Abstraction Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Edge Computing and Distributed Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Serverless Evolution and FaaS at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Future of Platform Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Conclusion: Building Your Platform Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Assessing Your Current State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

A Phased Approach to Platform Maturity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Building the Business Case

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Key Decisions and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

The Path Forward

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.

Index

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/platformengineeringintheaiera.