Building Composable Platforms: Patterns, Practices, and Pratfalls

Daniel Bryant
Developer, Platform Engineer & Product Marketer

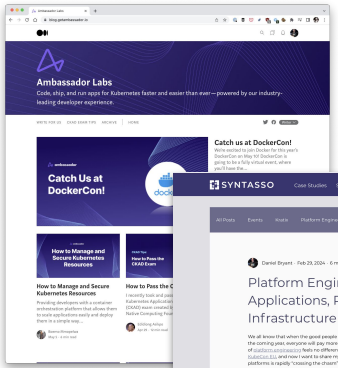


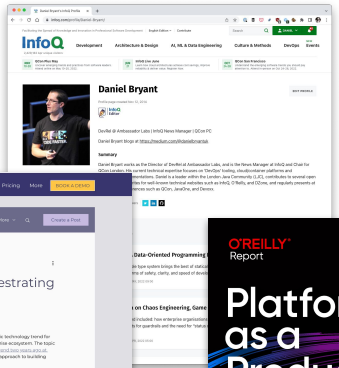
SYNTASSO

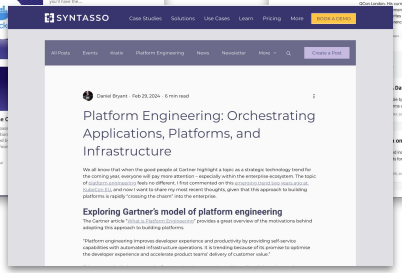
tl;dr

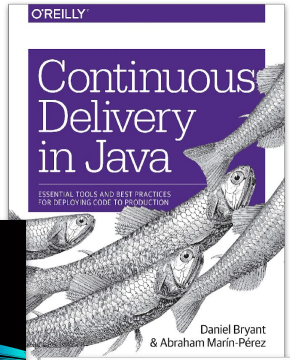
- Composition is valuable in three layers: app, platform, and infra
- Each layer (teams) must own their flow of value
- Think speed, safety, efficiency, and scalability
- Composability is all about APIs, abstraction, and automation
- Build your platform as a product

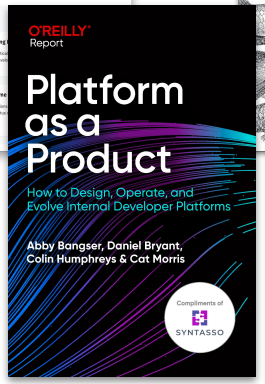
@danielbryantuk

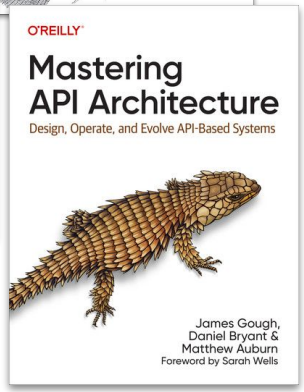


















EMISSARY INGRESS





linktr.ee/danielbryantuk

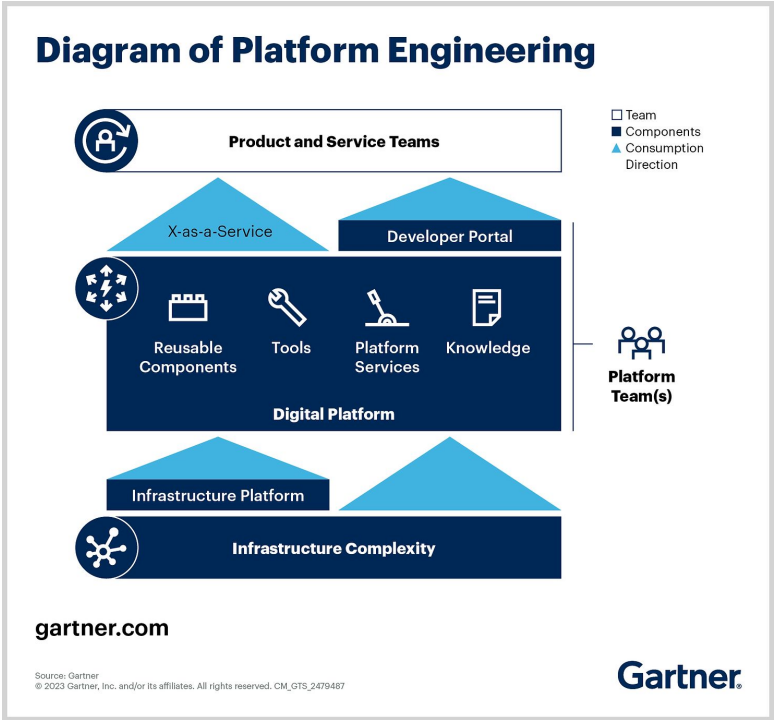
Setting the scene



The value of platform engineering

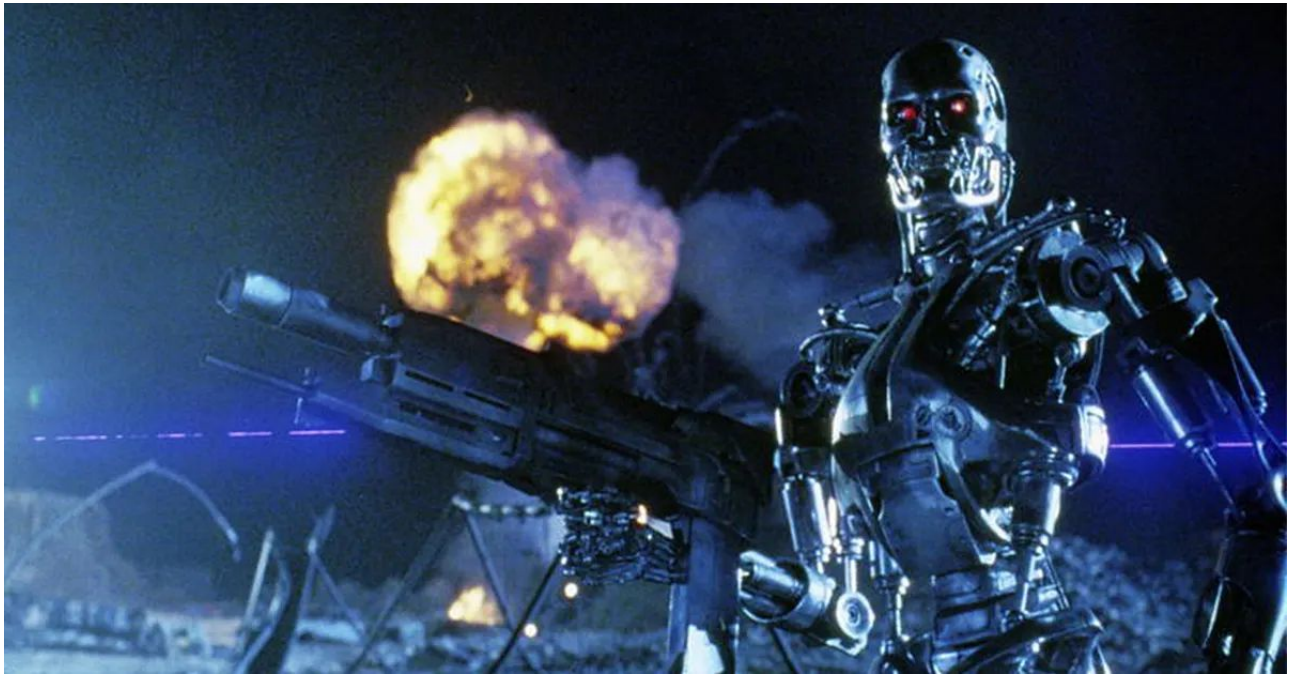
By 2026, approximately 80% of large software engineering organizations will establish dedicated platform engineering teams to create ***"Internal Developer Platforms"***

But something happened in 2024-2026...



gartner.com/en/articles/what-is-platform-engineering

(AI) Party like it's August 29th, 1997



Platform teams aren't feeling the love...



Platforms are still valuable...

Gartner Insights Tools Expert Guidance Conferences Webinars AI Hub What We Do

Get Started ↗

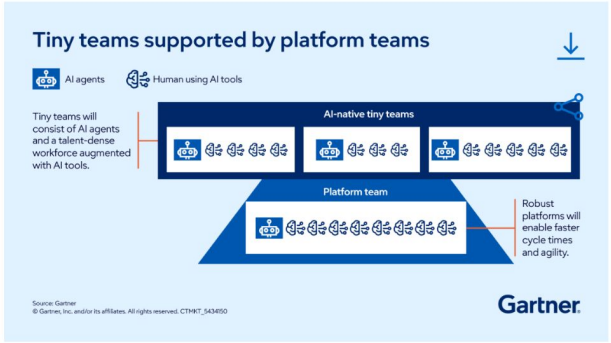
or call +44 1784 614 242

Software Engineering Overview Solutions Priorities Client Stories Events

The Future of Software Engineering: Smaller Teams, Bigger Ambitions

AI is not reducing the need for software engineers. It's reshaping how engineering teams are structured to deliver more innovation, agility and business value.

September 25, 2026

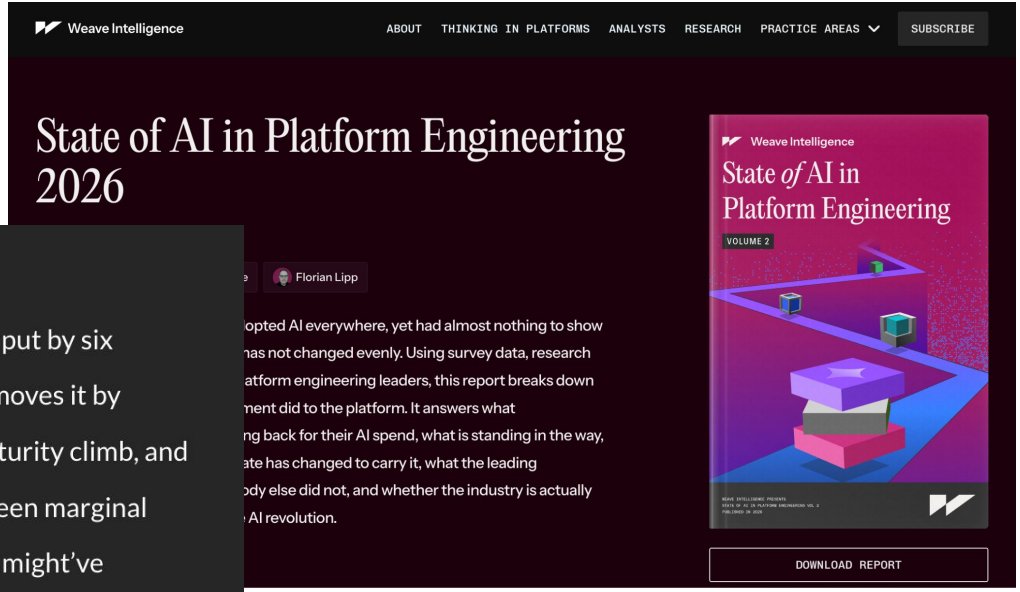


gartner.com/en/articles/software-engineering-tiny-teams

Maybe even more so...

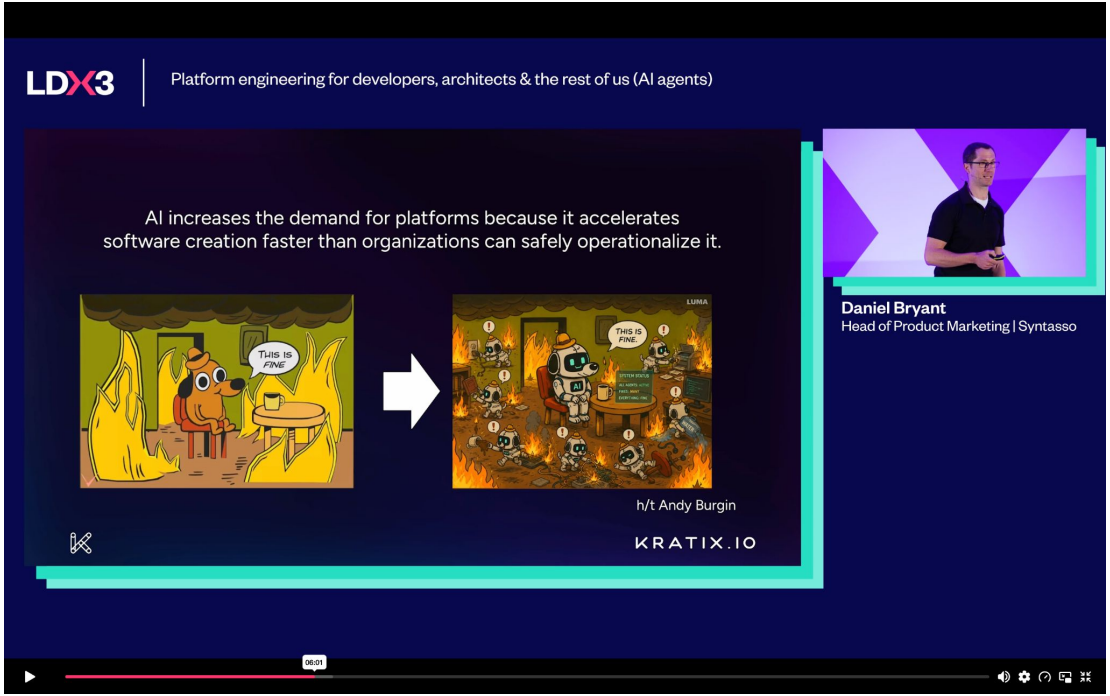
2. Maturity pays at a threshold, not as a gradient.

Deploying agents onto an immature platform moves throughput by six percentage points. Evolving your platform to enable agents moves it by twenty-four. The whole return sits on one step of agentic maturity climb, and most of the industry is standing below it. If your gains have been marginal despite real investment, this is very likely why. Your AI usage might've changed, but the platform underneath it did not.



weaveintelligence.io/research/state-of-ai-in-platform-engineering-2026

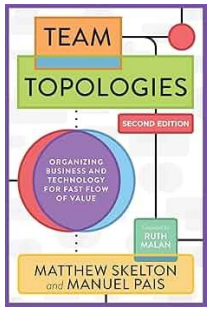
My 2c: AI is accelerating demand on your platform



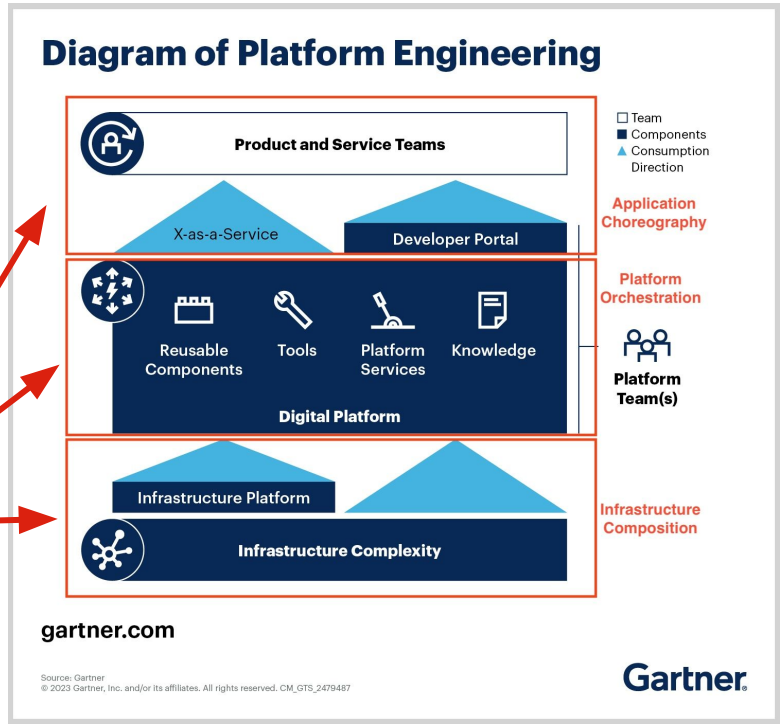
leaddev.com/culture/platform-engineering-for-developers-architects-the-rest-of-us-ai-agents

The value of platform engineering

By 2026, approximately 80% of large software engineering organizations will establish dedicated platform engineering teams to create ***"Internal Developer Platforms"***

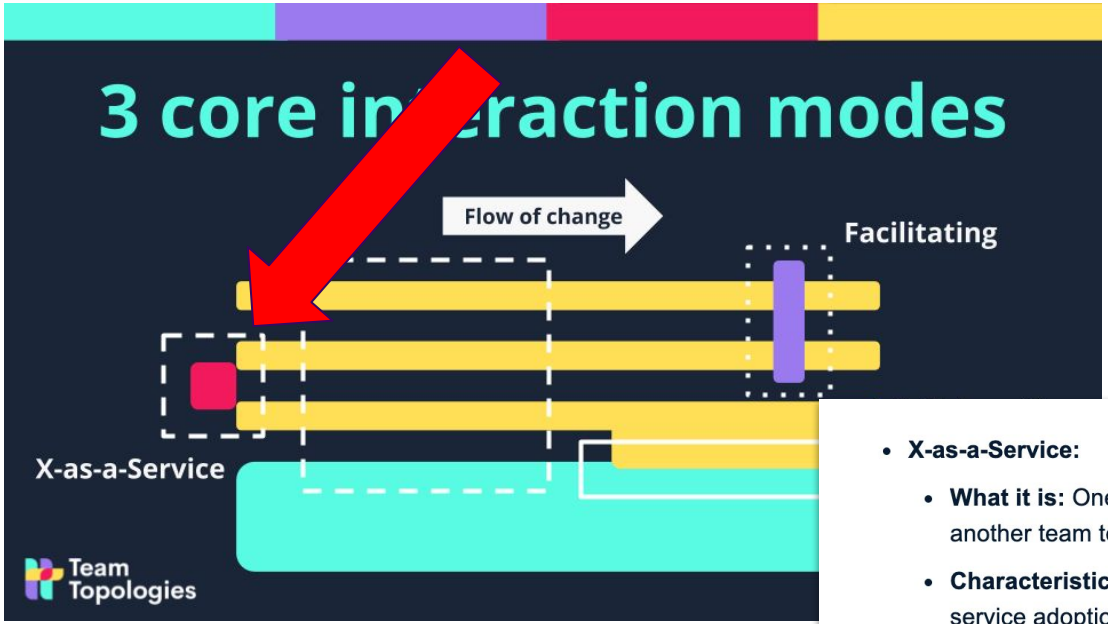


"Teams must own their own flow of value"



https://www.gartner.com/en/articles/what-is-platform-engineering

Team Topologies recap: XaaS FTW



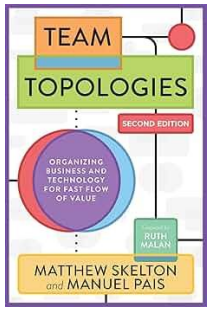


- **X-as-a-Service:**
  - **What it is:** One team prov[illegible]esource (like an API, a tool, or a platform) for another team to consume[illegible]inimal fuss or interaction.
  - **Characteristics:** Low cost, clear boundaries, and focuses on the provider's service adoption and the consumer's satisfaction.
  - **Goal:** To enable predictable delivery and speed up development by allowing teams to consume ready-made capabilities without complex conversations or setup.

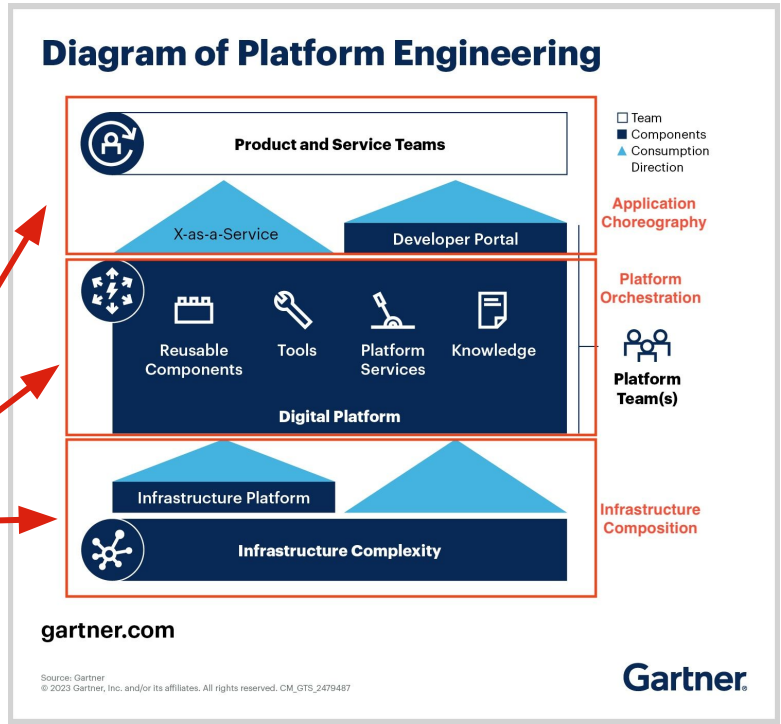
teamtopologies.com/key-concepts

The value of platform engineering

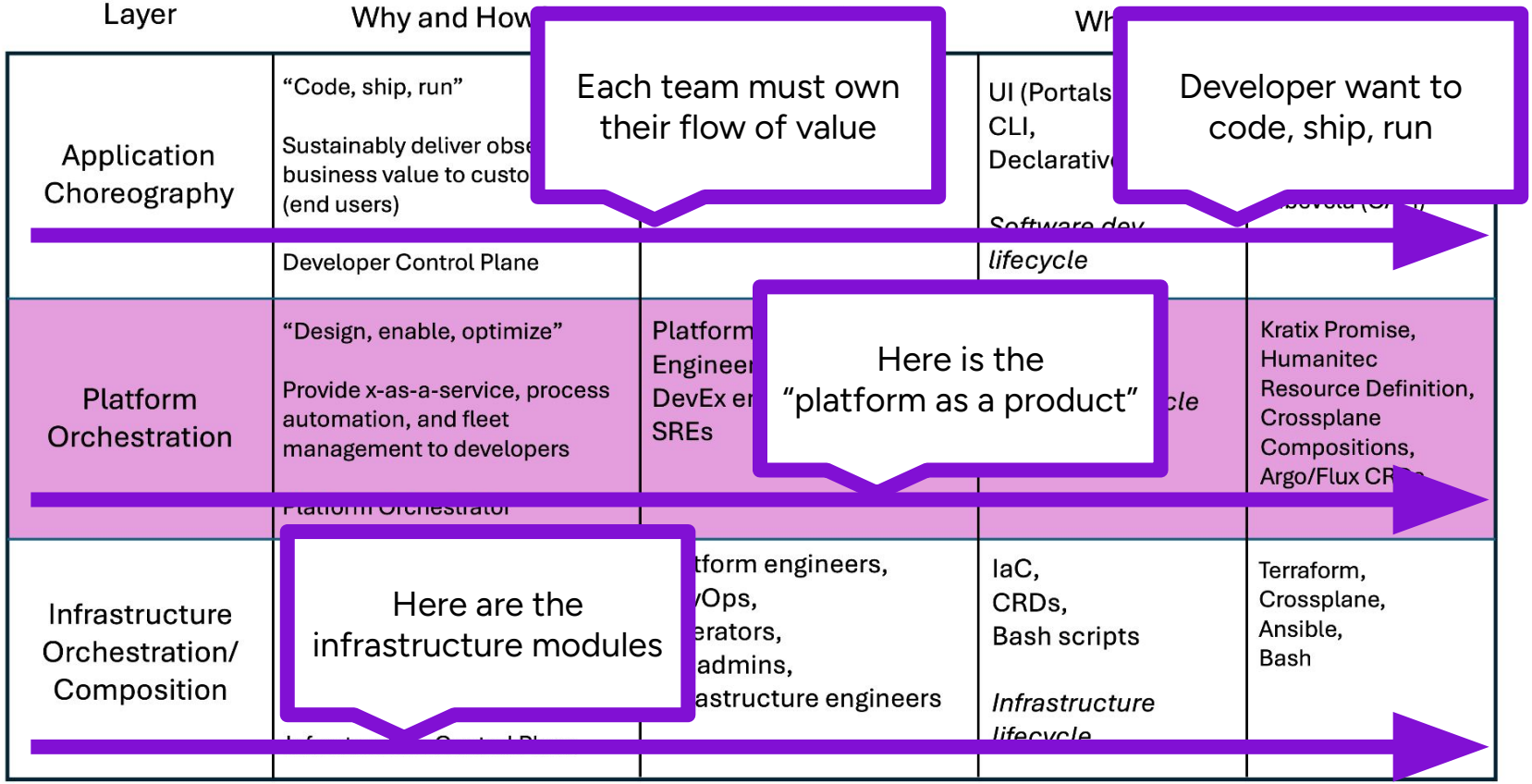
By 2026, approximately 80% of large software engineering organizations will establish dedicated platform engineering teams to create ***"Internal Developer Platforms"***



"Teams must own their own flow of value"



https://www.gartner.com/en/articles/what-is-platform-engineering



syntasso.io/post/platform-engineering-orchestrating-applications-platforms-and-infrastructure

Why Composable Platforms?



KRATIX.IO

The benefits of composable platform architecture

- **Fast time to market**
  - Quick assembly and deployment
- **Increased resilience**
  - Easier to understand, chain, and maintain
- **Cost-efficiency**
  - Easier to upgrade
- **Supports evolution**
  - Modular growth



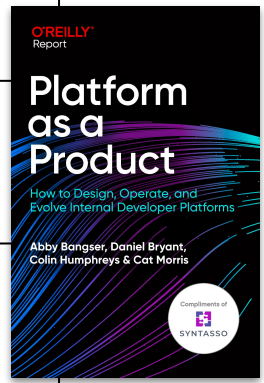
The benefits of composable platform architecture

- **Fast time to market (Speed)**
  - Quick assembly and deployment
- **Increased resilience (Safety)**
  - Easier to understand, chain, and maintain
- **Cost-efficiency (Efficiency)**
  - Easier to upgrade
- **Supports evolution (Scalability)**
  - Modular growth



Internal Platforms Scorecard

Attribute	Definition	One Metric That Matters
Speed	Fast, on-demand provisioning of services and infrastructure	**Mean Time to First Deploy**
Safety	Customisation with control, delivering business-specific services within guardrails	**% of Services Running that are Policy-Compliant**
Efficiency	Seamless fleet-wide operations with reduced operational toil	**Mean Time to Upgrade Service Instances**
Scalability	Ability for teams to extend and evolve the platform independently	**Mean Time to Add a Service to the Platform**



Composability impacts all of the properties

- **Speed** 🏎️
  - API-first, highly automated, with consistent lifecycle management
- **Safety** 👷
  - Re-usable, organisation-specific, and approved compliance components
- **Efficiency** 🎯
  - Upgradable as a fleet (easy to upgrade 100 as it is 1)
- **Scalability** 🚀
  - Supporting inner sourcing & “platform democracy” (easy and effective contributions)

Platform Patterns



KRATIX.IO

#1: Build-Your-Own Kit vs Composable Blocks

😡 Fails: DevOps Tool Dump

- “Here’s Terraform, Helm, ArgoCD, Vault…”
- Every team builds its own mini-platform
- No shared standards, no upgrade path

🏎️✅✅▪(only for the right teams)

👷❌❌▪

🎯❌❌❌

🚀❌❌❌

😃 Works: Small set of reliable primitives

- Few org-specific building blocks
- Blocks versioned, upgradable, combinable
- Good discoverability of primitives

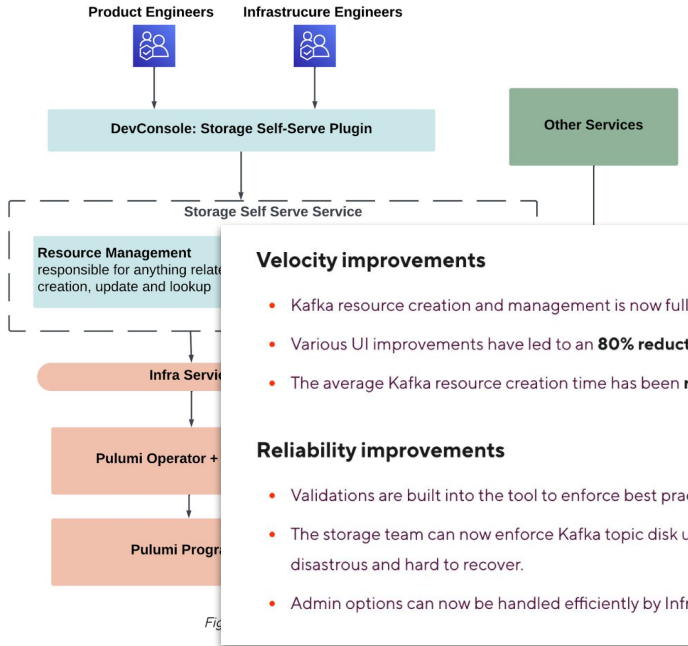
🏎️✅✅▪ (with abstraction + automation)

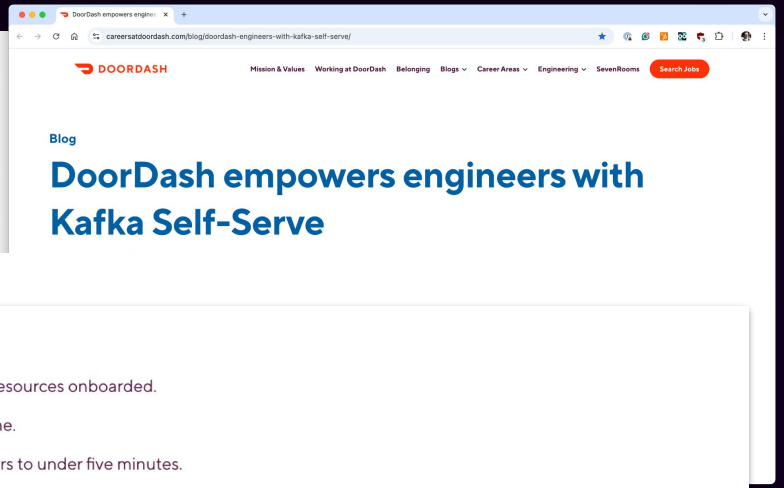
👷✅✅▪ (needs fleet management)

🎯✅▪▪

🚀✅✅▪

#1: Build-Your-Own Kit vs Composable Blocks





Velocity improvements

- Kafka resource creation and management is now fully self-serve, with 23,335 resources onboarded.
- Various UI improvements have led to an **80% reduction** in average loading time.
- The average Kafka resource creation time has been **reduced 99%**, from 12 hours to under five minutes.

Reliability improvements

- Validations are built into the tool to enforce best practices at creation time.
- The storage team can now enforce Kafka topic disk usage to properly size our Kafka clusters, preventing us from ever running out of disk space on a Kafka cluster — which is disastrous and hard to recover.
- Admin options can now be handled efficiently by Infra engineers to quickly mitigate incidents.

careersatdoordash.com/blog/doordash-engineers-with-kafka-self-serve/

#2: Policy in Confluence vs Policy in the Runtime

😡 Fails: Docs-based Governance

- Golden-path wiki nobody reads
- Platform team becomes gatekeeper
- Drift, exceptions, and manual reviews

🏎️❌❌▪️

👷✅▪️▪️**(yes, at the start)**

🎯❌❌❌

🚀❌❌❌

😃 Works: Reusable Governance as Code

- Compliance baked into delivery path
- Evidence is machine-readable
- Reusable components (CALMS, OPA, etc)

🏎️✅✅▪️

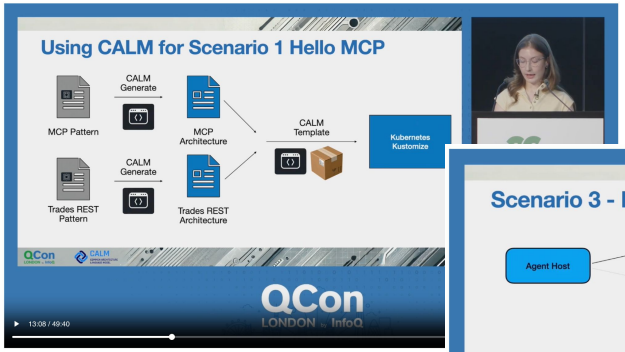
👷✅✅✅

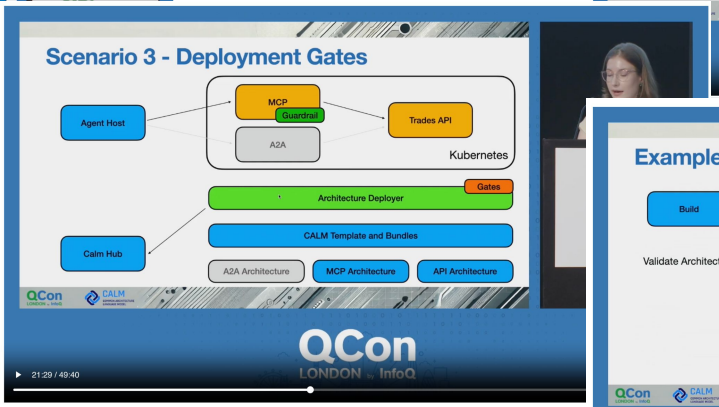
🎯✅✅✅

🚀✅✅ ▪️(need all teams involved)

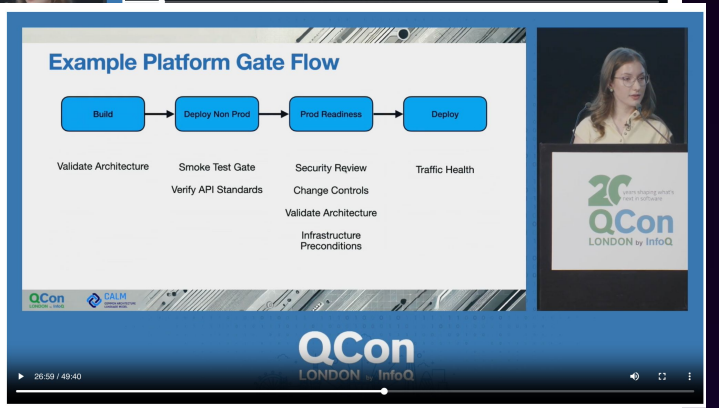
#2: Policy in Confluence vs Policy in the Runtime

APIs for Agents: Rethinking API Programs in the MCP Era









infoq.com/presentations/mcp-calm-api-architecture/

#3: Puppy for Christmas vs Platform as a Product

😡 Fails: Puppy for Christmas

- Templates for Terraform, Helm, etc
- No owner, no upgrade, no guarantees
- Day-2 chaos: drift, patches, audit pain

🏎️✅▪▪ (yes, initially)

👷❌❌❌

🎯✅▪▪ (yes, initially)

🚀✅▪▪ (yes, if everyone is an expert)

😃 Works: Platform as a Product

- Capabilities offered as versioned products
- Each has owner, SLA, lifecycle,
- Automatable, reconcilable (idempotent)

🏎️✅✅✅(minimise approvals)

👷✅✅✅(engage InfoSec)

🎯✅✅✅(GitOps++)

🚀✅✅✅ (embracing Innersourcing, etc)

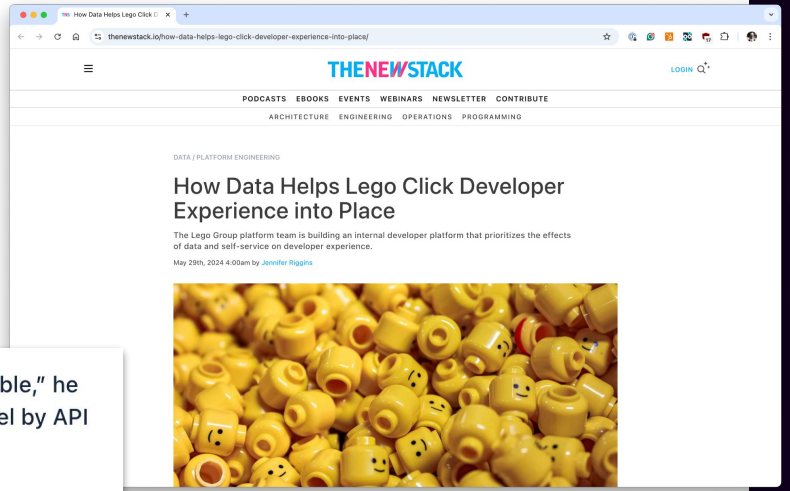
#3: Puppy for Christmas vs Platform as a Product

Why so long? Druedhahl Rask grouped the onboarding challenges into:

- **Discoverability.** As he said, "You need to know someone who knows someone who knows where to go in order to get things done."
- **Automation.** Too many humans in the loop and not enough automation, increasing lead time.
- **Ownership.** When individuals are tied to IT assets, it's harder to determine ownership.

"If you want to use it for something real, you also need to make it available," he said. "So we are democratizing access to all of the data within the model by API enabling it."

The API is the standard that has emerged for how developers prefer to access and extend an internal developer platform.



thenewstack.io/how-data-helps-lego-click-developer-experience-into-place/

Putting it all together



For composability to thrive...

- **Clear API (contract) and lifecycle management (versioning, reconciliation, etc)**
  - CNCF Whitepaper soon published "Platform producer factors"
- **Correct level of abstraction (relevant to your organisation)**
- **Building blocks appropriate for the layer/department/specialisation**
- **Components must be discoverable**
- **Productise your services for the win!**

Abstractions all the way down

Layer	Too Low	Useful Abstraction	Too High
Application			
Platform			
Infrastructure			

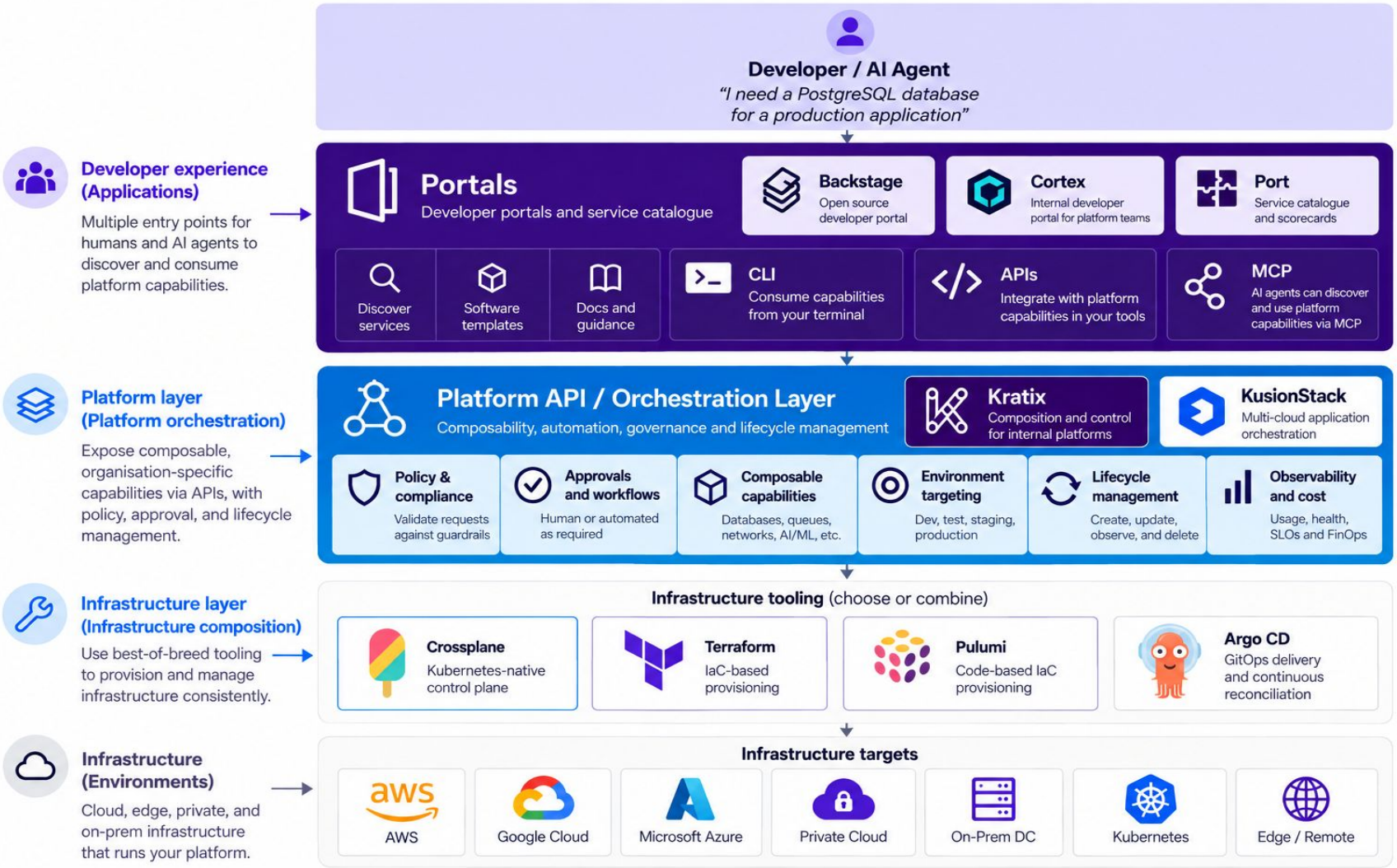
Each layer should expose abstractions meaningful to its consumers, without leaking the implementation details of the layer below.



Abstractions all the way down

Layer	Too Low	Useful Abstraction	Too High
Application	`Terraform + Helm + Vault + Argo CD` Consumer understands implementation details	**`WebApplication`** / **`Worker`** Developer expresses app requirements, deps and intent	`deployMyApp()` Easy initially, but rigid and hard to extend
Platform	Raw K8s resources, Terraform modules, cloud services Consumer assembles infrastructure into a capability	**`PostgresDatabase`** / **`KafkaTopic`** / **`Environment [FinOpsReporter / PiiLeakDetector / ProdDeployApprover]`** Organisation-specific capabilities with policy and lifecycle built in	`createProductionApp()` Bundles too many capabilities and removes choices teams legitimately need to make
Infrastructure	Individual API calls, scripts, cloud resources Operators manually assemble primitives	**`Network`** / **`DatabaseInstance`** / **`KubernetesCluster [FinOpsReporter / OpenPortCheck]`** Reusable infrastructure components with clear contracts	`createProductionEnvironment()` Hides topology, resilience, networking and capacity decisions

Each layer should expose abstractions meaningful to its consumers, without leaking the implementation details of the layer below.


Developer / AI Agent
"I need a PostgreSQL database for a production application"
Developer experience (Applications)
Multiple entry points for humans and AI agents to discover and consume platform capabilities.
Portals
Developer portals and service catalogue
Backstage
Open source developer portal
Cortex
Internal developer portal for platform teams
Port
Service catalogue and scorecards
Discover services
Software templates
Docs and guidance
CLI
Consume capabilities from your terminal
APIs
Integrate with platform capabilities in your tools
MCP
AI agents can discover and use platform capabilities via MCP
Platform layer (Platform orchestration)
Expose composable, organisation-specific capabilities via APIs, with policy, approval, and lifecycle management.
Platform API / Orchestration Layer
Composability, automation, governance and lifecycle management
Kratix
Composition and control for internal platforms
KusionStack
Multi-cloud application orchestration
Policy & compliance
Validate requests against guardrails
Approvals and workflows
Human or automated as required
Composable capabilities
Databases, queues, networks, AI/ML, etc.
Environment targeting
Dev, test, staging, production
Lifecycle management
Create, update, observe, and delete
Observability and cost
Usage, health, SLOs and FinOps
Infrastructure layer (Infrastructure composition)
Use best-of-breed tooling to provision and manage infrastructure consistently.
Infrastructure tooling (choose or combine)
Crossplane
Kubernetes-native control plane
Terraform
IaC-based provisioning
Pulumi
Code-based IaC provisioning
Argo CD
GitOps delivery and continuous reconciliation
Infrastructure (Environments)
Cloud, edge, private, and on-prem infrastructure that runs your platform.
Infrastructure targets
AWS
Google Cloud
Microsoft Azure
Private Cloud
On-Prem DC
Kubernetes
Edge / Remote

Wrapping up 🎉



KRATIX.IO

Conclusion

- Think about your platform architecture in three layers:
  - Application, platform and infrastructure
- Each team must own their flow of value
- Think speed, safety, efficiency, and scalability
- Composability is all about APIs, abstraction, and automation
- *Build your platform as a product*

Thank you!

@danielbryantuk | daniel@syntasso.io

syntasso.io/platform-as-a-product-oreilly-report

docs.kratix.io/ske/installing-ske/intro

