

Platform Engineering for Software Developers, Architects, And the Rest of Us

Daniel Bryant

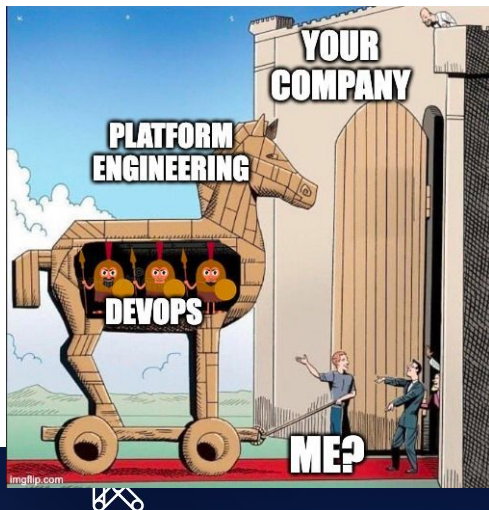
Developer, Platform Engineer, and Product Marketer



SYNTASSO

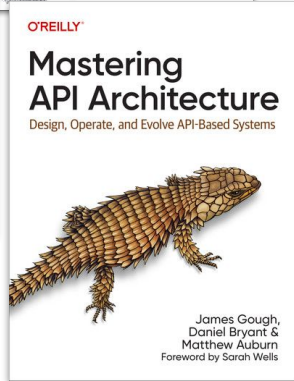
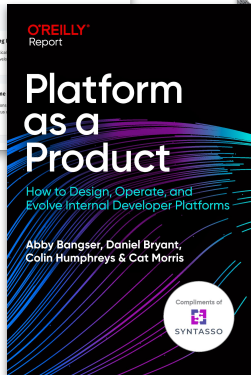
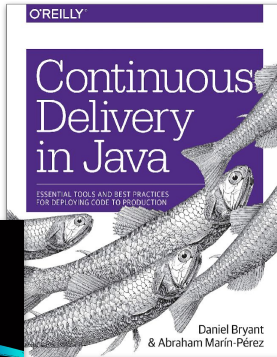
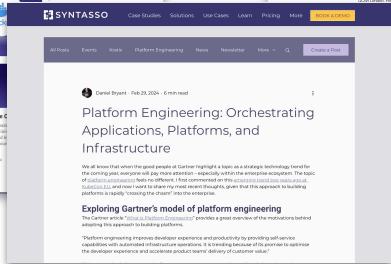
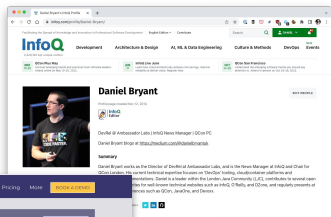
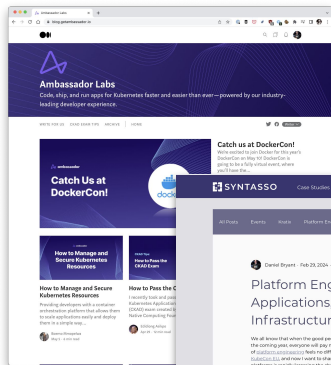
tl;dr:

- Platform engineering should have a product focus: devs are customers!
- Platform architecture and software architecture are symbiotic
- Good APIs, abstractions, and automation are the prize (for everything)



KRATIX.IO

@danielbryantuk



linktr.ee/danielbryantuk



EMISSARY
INGRESS



OpenJDK



KRATIX.IO

Previously, at...



KRATIX.IO

From Kubernetes to PaaS to... err, what's next?

Spoiler alert!

From Kubernetes to PaaS

My answer is **Golden Paths**, a.k.a.

The **real questions** are how much automation
and how should **you assemble** the pieces

Platform Engineer

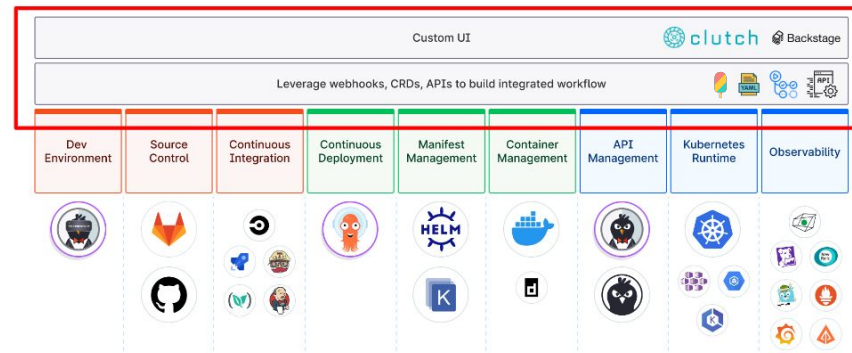
ambassador

@danielbryantuk

Looking back at my dev career

Year (Approx)	App Architecture	Infra	Tools & Platforms
2000	Monolith	On-prem	...
2005	Monolith / SaaS	Cloud / PaaS	...
2010	Microservices	Heroku / CF	...
2015	Microservices	Cloud	...
2020	Microservices++	K8s	...

The CNCF ecosystem is the foundation for a DCP



ambassador

@danielbryantuk | @ambassadorlabs

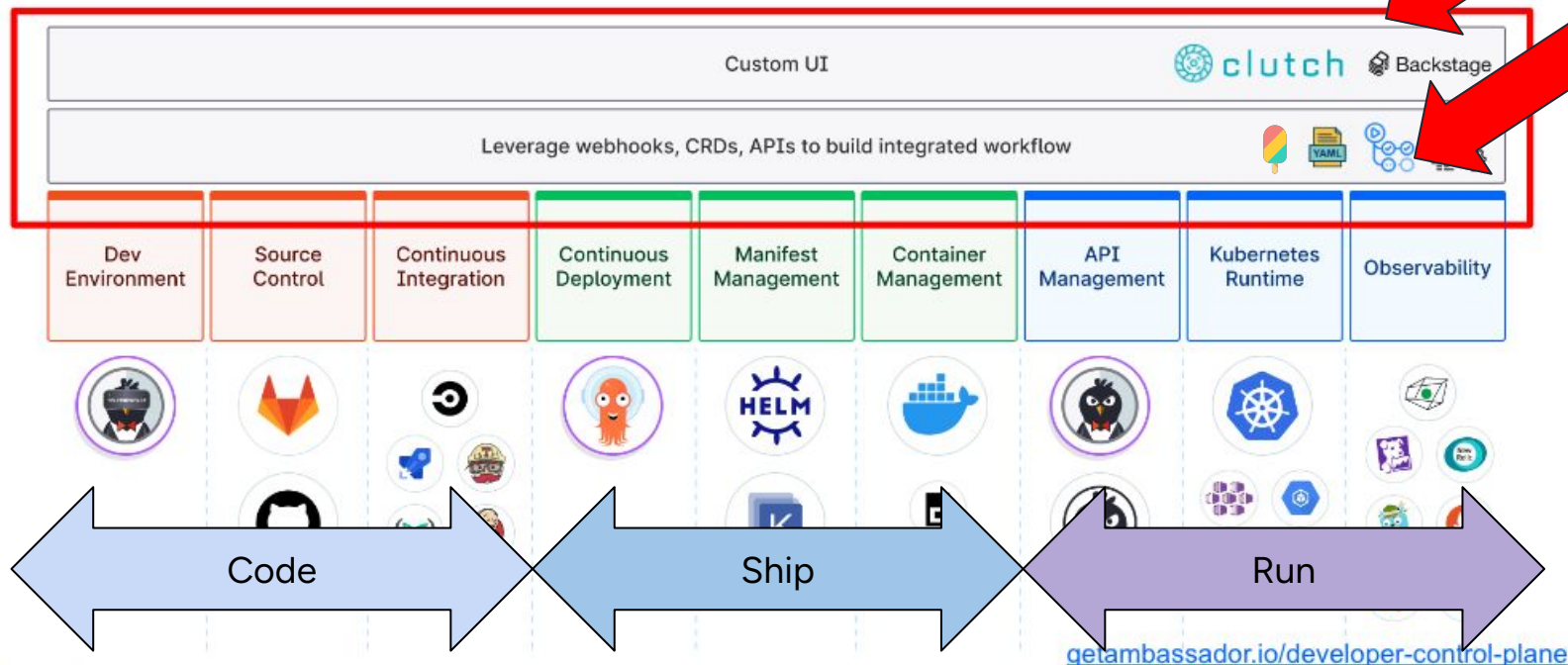
getambassador.io/developer-control-plane

<https://www.youtube.com/watch?v=zUpYEhaUJnM>



KRATIX.IO

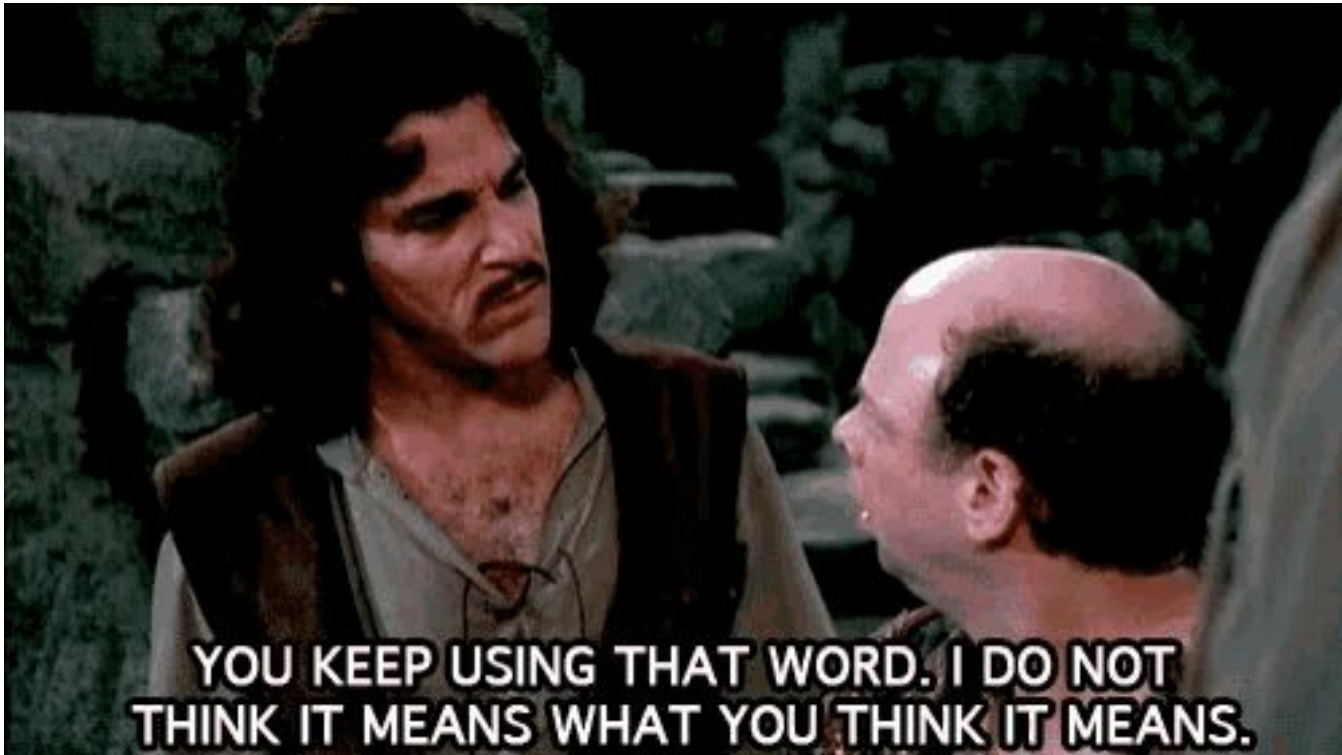
The CNCF ecosystem is the foundation for a DCP



The “what” of platforms



Platforms, platforms, platforms...



What is a platform, anyway?

“A digital platform is a foundation of self-service APIs, tools, services, knowledge and support which are arranged as a compelling internal product. Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced coordination.”

Evan Bottcher

martinfowler.com/articles/talk-about-platforms.html



What is a platform, anyway?

“A digital platform is a foundation of **self-service APIs, tools, services, knowledge and support** which are arranged as a **compelling internal product** . Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced coordination.”

Evan Bottcher

martinfowler.com/articles/talk-about-platforms.html



What is a platform, anyway?

“A digital platform is a foundation of **self-service APIs, tools, services, knowledge and support** which are arranged as a **compelling internal product** . Autonomous delivery teams can make use of the platform to **deliver product features at a higher pace, with reduced coordination** .”

Evan Bottcher

martinfowler.com/articles/talk-about-platforms.html

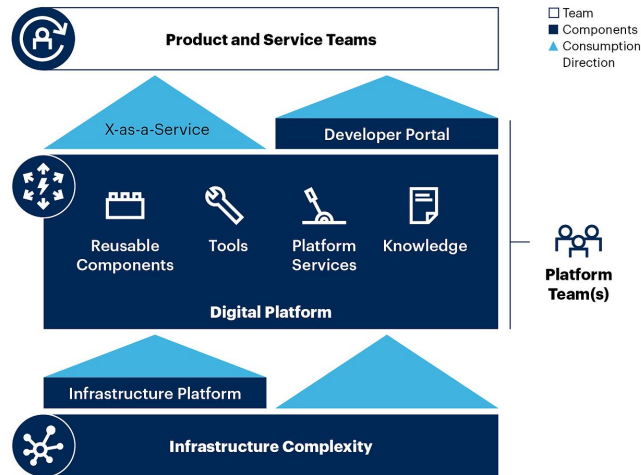


Gartner: What is platform engineering?

"Platform engineering **improves developer experience** and productivity by providing **self-service capabilities** with **automated infrastructure operations** .

It is trending because of its promise to optimise the developer experience and accelerate product teams' delivery of customer value."

Diagram of Platform Engineering



gartner.com

Source: Gartner
© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. CM_GTS_2479487

Gartner

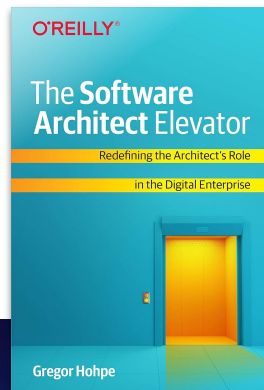
<https://www.gartner.com/en/articles/what-is-platform-engineering>



KRATIX.IO

What are the goals of your platform?

- **Go faster:** Platform teams need to provide “everything as a service” to help rapidly and sustainably deliver value to end-users
- **Decrease risk:** Teams need to automate manual processes in reusable components
- **Increase efficiency:** You need to manage and scale your digital platform and resources as a fleet



Platform architecture 🏢



Layer	Why and How?	Who?	What	Example tech

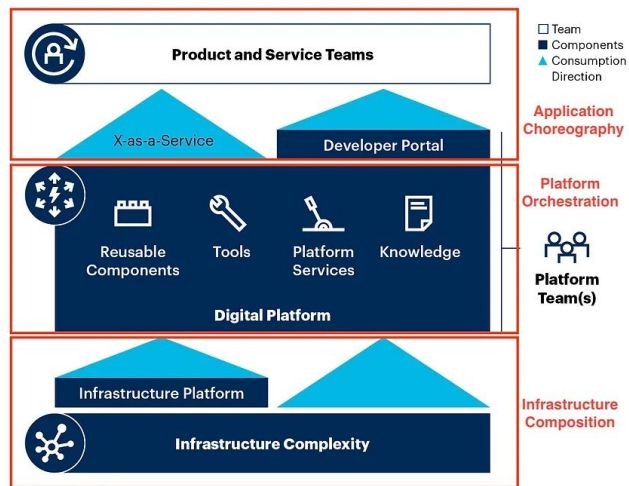
syntasso.io/post/platform-engineering-orchestrating-applications-platforms-and-infrastructure

Layer	Why and How?	Who?	What	Example tech
Application Choreography	“Code, ship, run” Sustainably deliver observable business value to customers (end users) Developer Control Plane	App developers, Full stack engineers, DevOps, SREs	UI (Portals), CLI, Declarative config <i>Software dev lifecycle</i>	Backstage, Heroku CLI & Netflix Newt, Score, Radius & KubeVela (OAM)
Platform Orchestration	“Design, enable, optimize” Provide x-as-a-service, process automation, and fleet management to developers Platform Orchestrator	Platform engineers, Engineering enablement, DevEx engineers, SREs	Platform API <i>Platform lifecycle</i>	Kratix Promise, Humanitec Resource Definition, Crossplane Compositions, Argo/Flux CRDs
Infrastructure Orchestration/ Composition	“Plan, build, maintain” Provide infrastructure building blocks for consumption and composition to platform team Infrastructure Control Plane	Platform engineers, DevOps, Operators, Sysadmins, Infrastructure engineers	IaC, CRDs, Bash scripts <i>Infrastructure lifecycle</i>	Terraform, Crossplane, Ansible, Bash

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

This three tier thing could catch on...

Diagram of Platform Engineering



gartner.com

Source: Gartner
© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. CM_GTS_2479487

Gartner

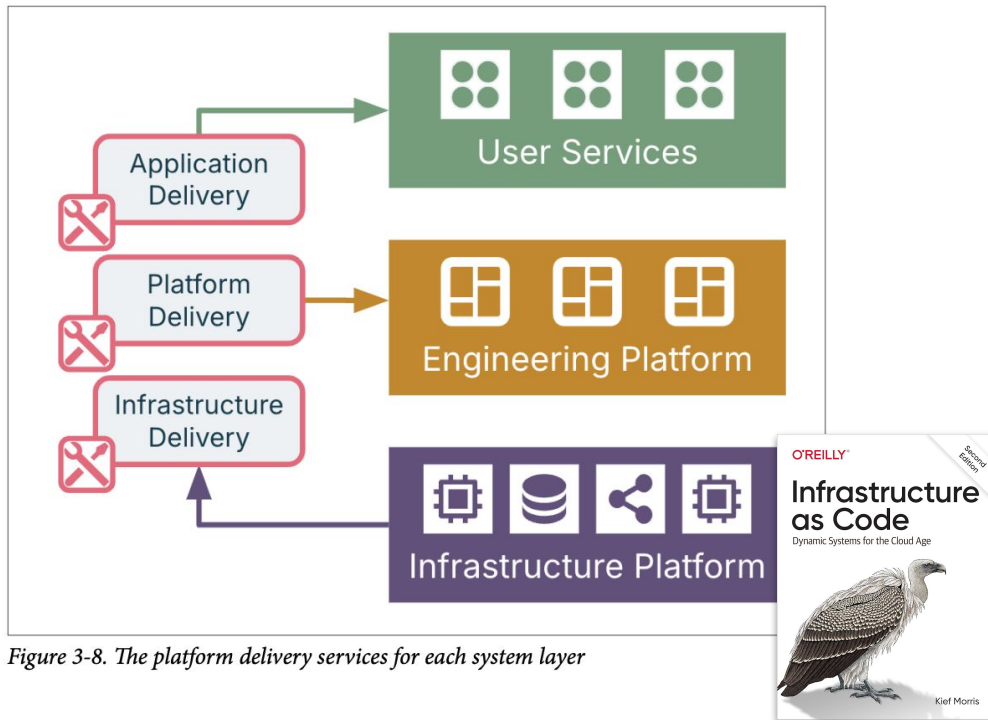


Figure 3-8. The platform delivery services for each system layer



KRATIX.IO

This three tier thing could catch on...

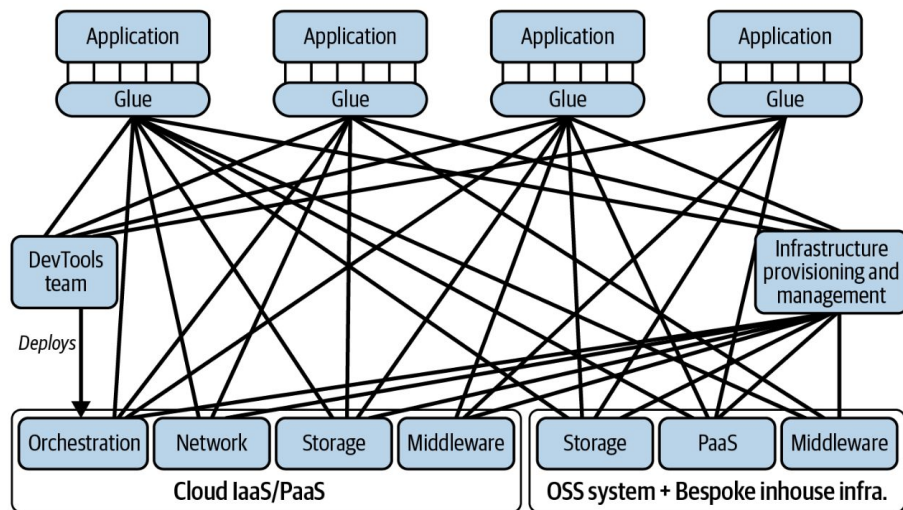


Figure 1-1. The over-general swamp, held together by glue

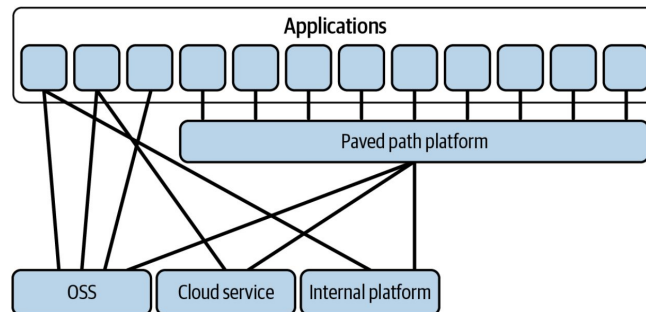
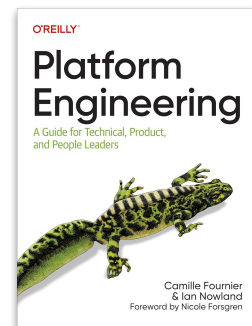


Figure 2-1. Architecture of a paved path platform

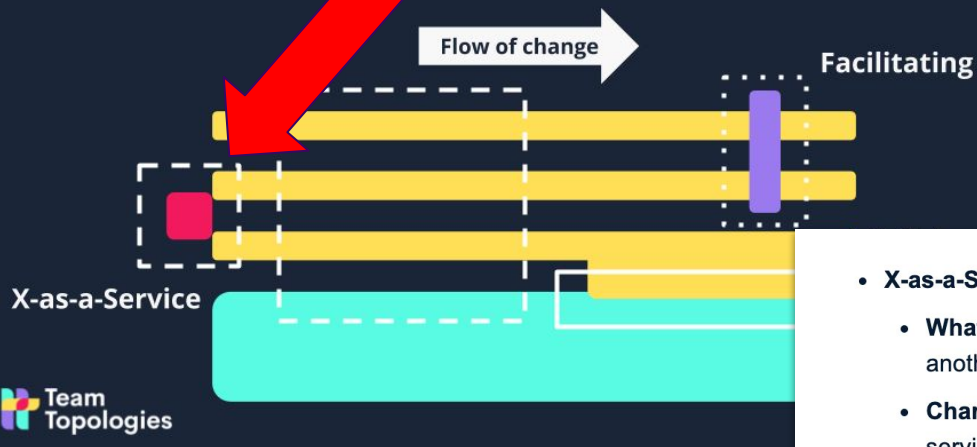


Layer	Why and How?	Who?	What	
Application Choreography	“Code, ship, run” Sustainably deliver observable business value to customers (end users)	App developers, Full stack engineers, DevOps, SREs	UI (Portals), CLI, Declarative code Software dev lifecycle	Each team must own their flow of value Kubernetes, Argo, Radius & KubeVela (OAM)
Platform Orchestration	“Design, enable, optimize” Provide x-as-a-service, process automation, and fleet management to developers	Here is the “platform as a product”		Form API Form lifecycle Kratix Promise, Humanitec Resource Definition, Crossplane Compositions, Argo/Flux CRs
Infrastructure Orchestration/ Composition	“Plan, build, maintain” Provide infrastructure building blocks for consumption and composition to platform team	Platform engineers, DevOps, Operators, Sysadmins, Infrastructure engineers	IaC, CRDs, Bash scripts Infrastructure lifecycle	Terraform, Crossplane, Ansible, Bash

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

Team Topologies

3 core interaction modes



teamentopologies.com/key-concepts

4 fundamental topologies

- Stream-aligned team
- Enabling team
- Complicated Subsystem team
- Platform team



• X-as-a-Service:

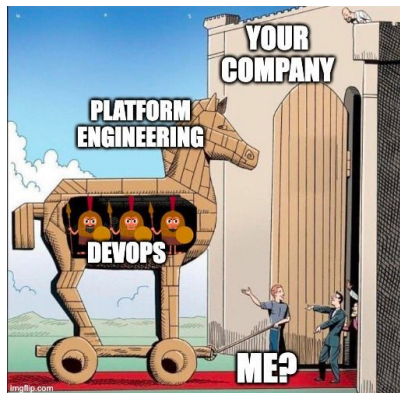
- **What it is:** One team provides a resource (like an API, a tool, or a platform) for another team to consume with minimal fuss or interaction.
- **Characteristics:** Low cost, clear boundaries, and focuses on the provider's service adoption and the consumer's satisfaction.
- **Goal:** To enable predictable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





tl;dr:

- Platform engineering should have a product focus: devs are customers!
- Platform architecture and software architecture are symbiotic
- Good APIs, abstractions, and automation are the prize (for everything)



(Platform) Architecture case studies 🧐



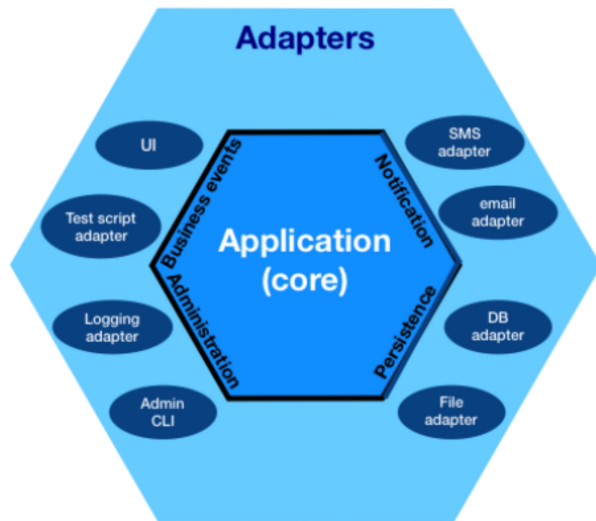
Hexagonal architecture: Ports, adapters, platforms

- **The before times**

- Layered/tiered apps were often highly coupled
- Old SOA platforms contained too much “magic”
 - EJBs, XA/JTA, ESBs

- **The evolution**

- Create cohesive business code
- Loose coupling facilitates testing
- Encapsulated platform components (e.g. DBs, queues)



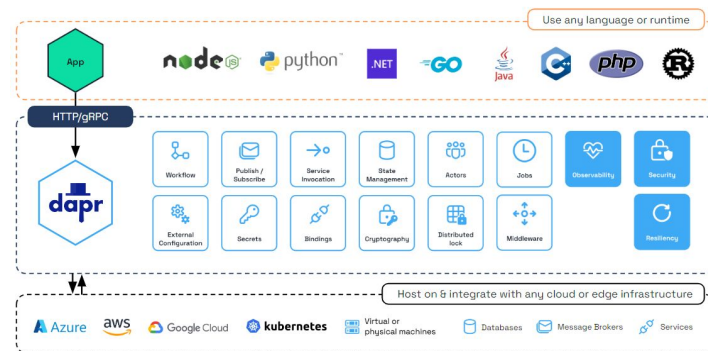
Hexagonal architecture: Platform evolution

- **Software solutions: Adapters**

- Wrap platform components in custom code
- Custom libraries & SDKs (& Netflix OSS)
- Potential for high-coupling at the platform interface

- **Platform APIs & abstractions**

- Sidecars to the rescue!
- Distributed Application Runtime (dapr)
- Still need platform components (X)-as-a-Service





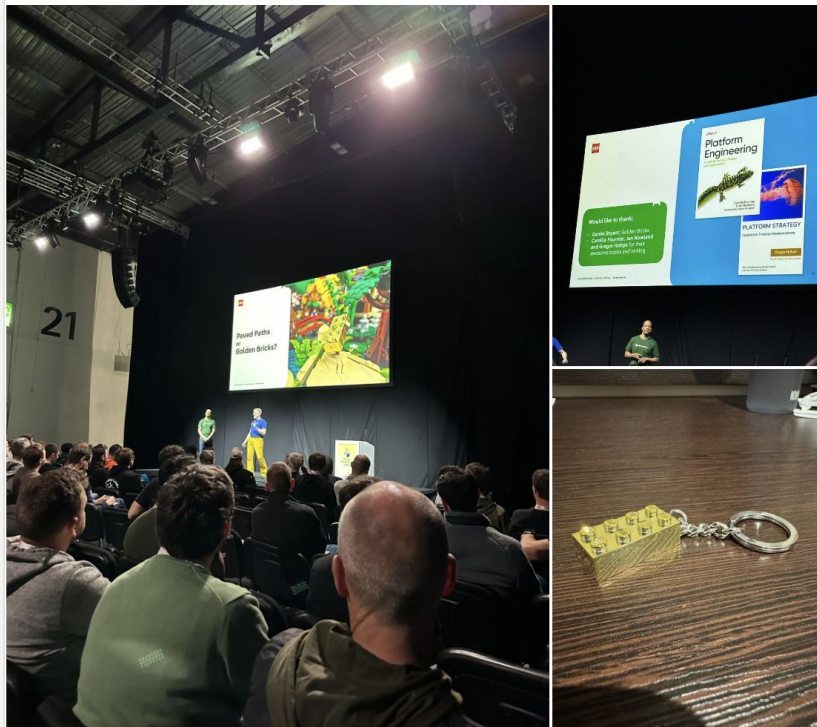
Daniel Bryant · You

Helping you build better platforms | InfoQ & QCon | Java Champion | Co-...

[View my newsletter](#)

11h · 🌐

A big thanks to [Mads Høgstedt Danquah](#) and [Christian Trolle Mikkelsen](#) from The LEGO Group for a shout-out in their awesome [#PlatEngDay](#) talk today to the concept of "golden bricks vs golden paths" that I mentioned in my previous [#KubeCon](#) talk. And an even bigger thanks for giving me my very own "golden brick" 🙌



Paula Kennedy and 11 others

2 comments · 1 repost



KRATIX.IO

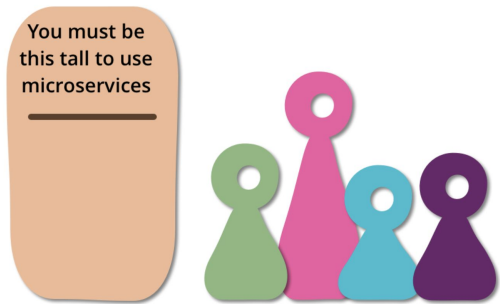
Microservices: Mo' services, mo' problems

- **Evolution**

- Create cohesive and loosely coupled services
- But, you need to be this tall...

- **Challenges**

- Ops teams slow to provision infrastructure
- DevOps: You build it, you run it
 - But, this doesn't scale across an enterprise
 - How do teams own their flow of value?



Rapid provisioning: you should be able to fire up a new server in a matter of hours. Naturally this fits in with [CloudComputing](#), but it's also something that can be done without a full cloud service. To be able to do such rapid provisioning, you'll need a lot of automation - it may not have to be fully automated to start with, but to do serious microservices later it will need to get that way.

Basic Monitoring: with many loosely-coupled services collaborating in production, things are bound to go wrong in ways that are difficult to detect in test environments. As a result it's essential that a monitoring regime is in place to detect serious problems quickly. The baseline here is detecting technical issues (counting errors, service availability, etc) but it's also worth monitoring business issues (such as detecting a drop in orders). If a sudden problem appears then you need to ensure you can quickly rollback, hence...

Rapid application deployment: with many services to manage, you need to be able to quickly deploy them, both to test environments and to production. Usually this will involve a [DeploymentPipeline](#) that can execute in no more than a couple of hours. Some manual intervention is alright in the early stages, but you'll be looking to fully automate it soon.



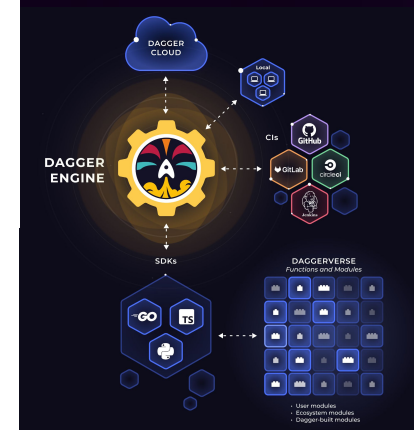
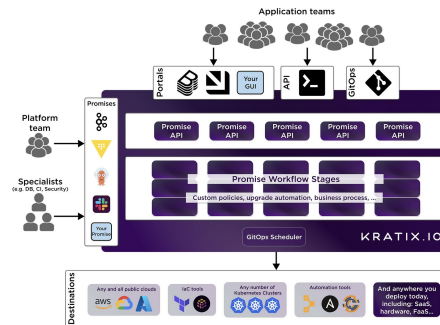
Microservices & platforms

- **Infrastructure solutions**

- PaaS! Heroku, Cloud Foundry (+ service broker/marketplace)
- App scaffolding focus (a.k.a. "Puppy for Christmas")
- Developers learn Terraform or Crossplane or Pulumi...

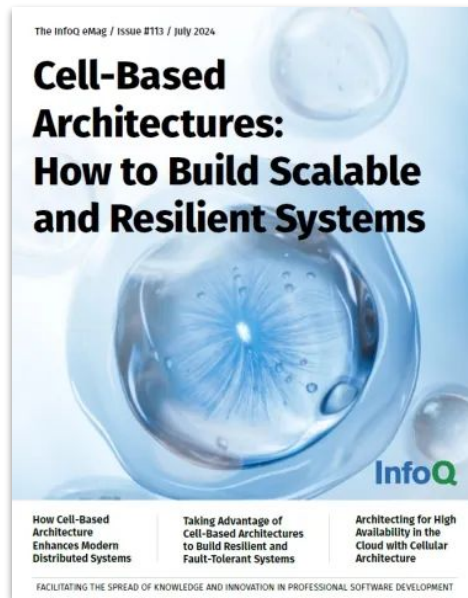
- **Platform solution**

- Workload definitions: OAM, Score, Radius (Dagger)
- Platform Orchestrator: Kratix, KusionStack, Humanitec, etc.
- Dev-focused IaC: Pulumi, AWS CDK, etc.



Cell-based Architecture

- **Context:**
 - Self-sufficient cells
 - Replication for scalability
 - Isolation and fault containment:
- **Challenges?**
 - Microservices++
- **Infrastructure solution**
 - Cluster API + Automation ([HPA, VPA, KEDA](#))
 - Advanced routing (K8s GW, OSM)
- **Platform solution**
 - Fleet management?



infoq.com/minibooks/cell-based-architecture-2024/



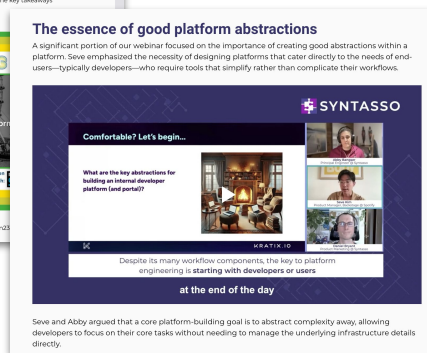
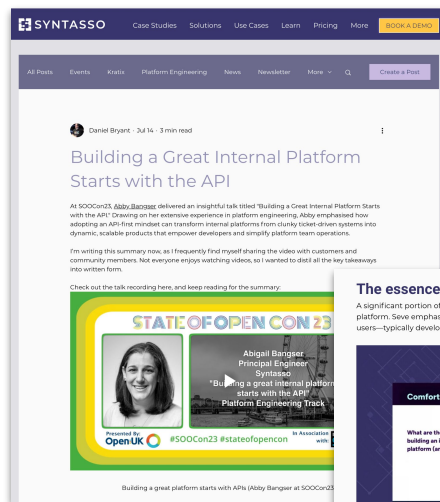
Key insights



Platforms: You can't have good DX without good UX

- **Design control planes for API, CLI, & UI**
 - Optimise for automation (API-first)
 - Minimize cognitive load
 - Build for progressive disclosure

syntasso.io/post/building-a-great-internal-platform-starts-with-the-api



syntasso.io/post/when-backstage-met-terraform-and-platform-orchestrators-webinar-recap



Platforms: Abstract(ion) thinking

“Devs don’t want magic; they want to be magicians”

(h/t Ellen Chisa)

- **CUPID, SRP, principle of least surprise, etc**

- Just as important for platforms
- Watch for leaky abstractions
 - The size of the spell matters!



Platforms: Don't forget the customer (product) focus

“Sarah particularly stressed viewing **“platforms as products”** that serve internal customers—the development teams—ensuring they are user-friendly and genuinely solve the teams' problems.”

The role of platforms (as a product)

The session started with Daniel emphasising the increasing expectations for executives to “do more with less”. He laid out the premise that platform strategies, if led and managed correctly, could provide the needed leverage.

The rest of the discussion covered the vital role of platforms, with the consensus being that they allow development teams to focus more on delivering value and less on managing infrastructure complexities.

Sarah particularly stressed viewing “**platforms as products**” that serve internal customers—the development teams—ensuring they are user-friendly and genuinely solve the teams' problems.

Build platforms for your customers (developers). Syntasso Au. Watch how > Share

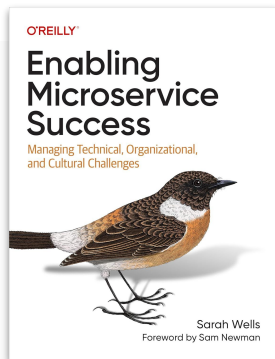
Platform Engineering for Execs: Team Topologies, Tools, and Tradeoffs

Comfortable? Let's begin...

Why and what to communicate about platform usage across the org?
(e.g. getting buy-in for platform and tooling work vs feature delivery)

PLATFORMS SHOULD BE TREATED AS PRODUCTS THAT ARE THE SOURCE OF REVENUE, NOT JUST A MEANS TO AN END

Watch on **usually means whatever you've built**



thoughtworks

Looking Glass 2024 coming in January

Download the 2022 report

Platforms as products
Delivering value through customer focus

As more organisations migrate to the cloud and seek the opportunity to build their own customised platforms on top of cloud infrastructure, different stages of platform performance or maturity are becoming clearer.

As the initial hype fades, determining what kind of platform is fit for purpose and defining the value you expect it to deliver are critical initial steps. But then comes the time to get the most out of your investments and maximise the value platform provides—in other words, understanding the platform as a product to manage.

Many of the disappointments that organisations experience with platforms stem from the failure to treat them as products. For example, many platforms intended for developer infrastructure lack the initial research into user needs and contextual insights into context to support other types of products. This leads to problems with user experience and adoption, and can ultimately mean developers to leave their sought after work elsewhere.

The challenge can be addressed by viewing platforms as products for their customers who must be defined, with a value proposition, a clear roadmap and dedicated resources to help those customers create value for the organisation. In other words, a platform needs ongoing attention, and to evolve and adapt in response to developer feedback and the changing business landscape.

Signals include:

The list of feedback and other **platforms enabling developer experience**, which can play a powerful role in enabling an existing platform.

Industry customers and vendors **working on engineering-focused platforms**, which shows the view of developers as platform customers is entering the mainstream.

There is a move beyond **data lakes** towards **federated data stores and data products**, making a distinction of responsibility and smooth flow of value across an organisation. The equivalent of **CI/CD** must encompass the value of this approach.

Next, **thoughtworks** is an increasingly visible approach to structuring business and technology teams for peak efficiency, **highlighting the need for platform teams to offer platforms as internal products** to create and accelerate other teams.

thoughtworks.com/en-gb/insights/looking-glass/platforms-as-products

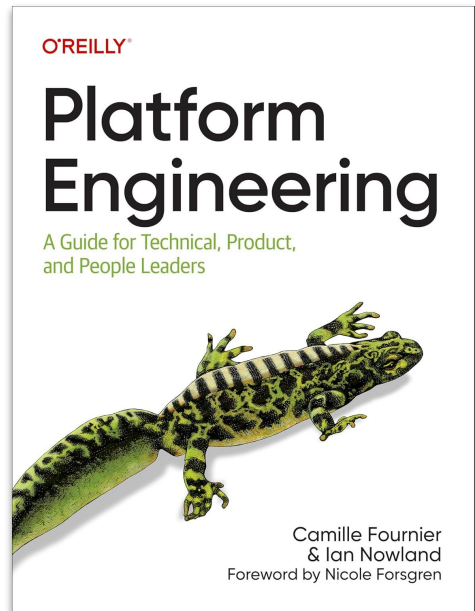
syntasso.io/post/platform-engineering-for-execs-team-topologies-tools-and-trade-offs-webinar-recap



KRATIX.IO

Platforms: What gets measured, gets managed

- **Establish goals and measurements**
 - Impact metrics
 - Guardrail metrics
 - Product health metrics



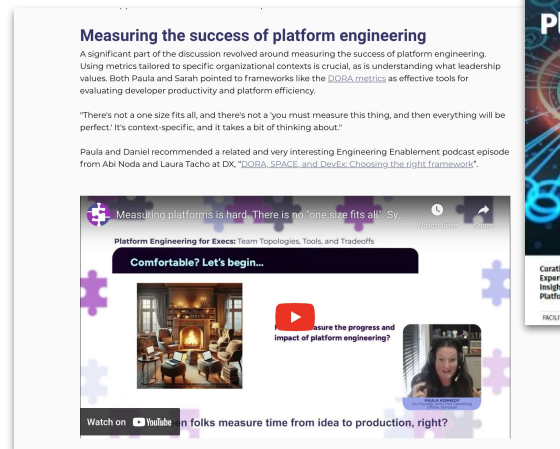
Platforms: What gets measured, gets managed

- **Leading indicators**

- Adoption rates (PH)
- Onboarding times (PH)
- Time to nth PR (PH)

- **Lagging indicators**

- App retention rate (PH)
- Upgrade/patch cycle resourcing (I)
- Speed of app/product delivery (I)
- Change fail percentage (G)
- # of Sev 1 incidents (G)



The (Goldilocks) DevEx Framework

- **DORA**

- Well understood
- Delivery-focused

- **SPACE**

- Covers all bases
- Potentially complicated

- **DevEx**

- Nicely balanced

infoq.com/articles/devex-metrics-framework/

TABLE 1: **EXAMPLE DEVEX METRICS**

	FEEDBACK LOOPS	COGNITIVE LOAD	FLOW STATE
PERCEPTIONS <i>Human attitudes and opinions</i>	• Satisfaction with automated test speed and output • Satisfaction with time it takes to validate a local change • Satisfaction with time it takes to deploy a change to production	• Perceived complexity of codebase • Ease of debugging production systems • Ease of understanding documentation	• Perceived ability to focus and avoid interruptions • Satisfaction with clarity of task or project goals • Perceived disruptiveness of being on-call
WORKFLOWS <i>System and process behaviors</i>	• Time it takes to generate CI results • Code review turnaround time • Deployment lead time [time it takes to get a change released to production]	• Time it takes to get answers to technical questions • Manual steps required to deploy a change • Frequency of documentation improvements	• Number of blocks of time without meetings or interruptions • Frequency of unplanned tasks or requests • Frequency of incidents requiring team attention
KPIS <i>North star metrics</i>	• Overall perceived ease of delivering software • Employee engagement or satisfaction • Perceived productivity		



Wrapping up 🎉



- KRATIX.IO



Thank you!

@danielbryantuk | daniel@syntasso.io

docs.kratix.io/main/quick-start

speakerdeck.com/syntasso

Kratix docs



KRATIX.IO