Infrastructure as product

Commoditizing the cloud through platform engineering

/thoughtworks

Introduction

Not too long ago, cloud infrastructure was being universally praised for its incredible simplicity and manageability. For companies and teams burdened by complex on-premise infrastructure, it presented an intuitive, scalable, and cost-effective alternative that promised to transform how we approach IT infrastructure as a whole.

But, as offerings from the major public cloud providers have grown and become more sophisticated, that simplicity has fallen by the wayside. Today, cloud infrastructure service portfolios are incredibly large, increasingly complex, and harder than ever for engineering teams to keep pace with.

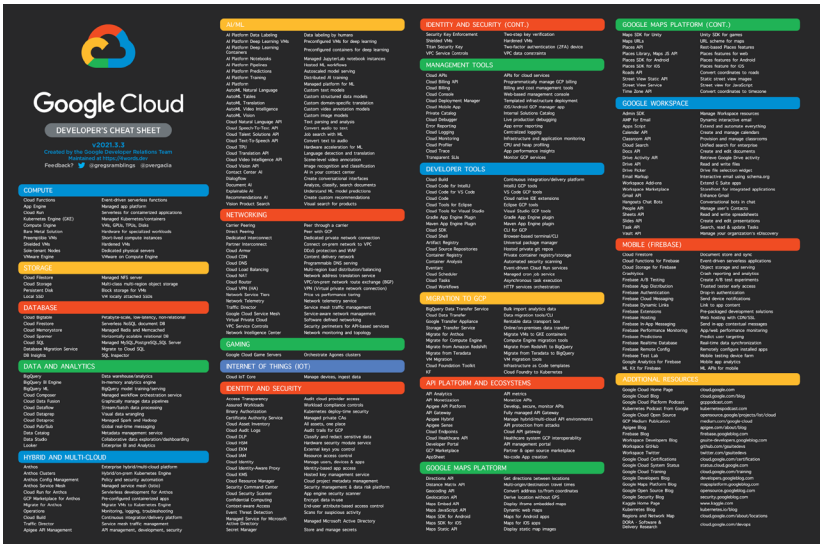

Figure 1. Snapshot of Google cloud capabilities

Figure 2. Snapshot of AWS cloud capabilities

Figures 1 and 2 show the sheer volume and variety of cloud capabilities offered by Google and AWS.

The market research firm 451 Research, which tracks cloud pricing on a quarterly basis, found that the number of line

items offered for purchase by the five largest cloud providers doubled in 2019, and now exceeds 2 million products[1].

As the number of these products has grown, the complexity of cloud infrastructure has surpassed that found in the physical data center. The design, architecture, and orchestration of these modern infrastructures requires a dedicated team with specific skills. Cloud infrastructure management has triggered an explosion of new disciplines – from telemetry to cloud database management – all of which demand their own specialist skills and expertise.

Cloud has changed infrastructure. We need to change how we think about infrastructure too.

Cloud and simplicity no longer go hand in hand. Today, cloud infrastructure management is often an extremely complex task. What was once a limited selection of basic compute, networking, and a few RDBMS database options has evolved into a rich smorgasbord of modern virtual data center wizardry. We're not dealing with simple components that can be trivially connected through a web interface anymore.

Even for those just getting started in their cloud journeys, you might be able to ship a product and start acquiring customers with only a small team of full-stack devs. But over time, as business needs grow, so will the complexity of the systems supporting it – both technical and operational.

To deliver and maintain a sustainable, production-ready infrastructure that is scalable, resilient, and adaptable, you must differentiate cloud infrastructure services as a set of internal domains, build expertise around them, and evolve

1. https://siliconangle.com/2020/07/09/rise-finops-liveramp-exec-reins-cloud-costs-melding-finance-developers/

them as internal products. Practically, that means delivering them through an intuitive platform approach that empowers the customers of those internal products – chiefly, your developers.

This internal platform approach isn't entirely new, but it's just beginning to become common practice across enterprises. It features significantly in Puppet's **2020 State of DevOps report**, where they highlight that 63 percent of organizations have at least one internal self service platform in place.

Defining a platform-based infrastructure approach

Typically, platform thinking encapsulates data, apps, infrastructure, and every part of your broader platform strategy. But for the purpose of this paper, when we refer to platform infrastructure and platform-based approaches, we're looking at using cloud to form the foundation of the infrastructure base. For a more comprehensive definition of a Platform, check out Evan Bottcher's[2].

2. https://martinfowler.com/articles/talk-about-platforms.html

On-premise infrastructure isn't the problem

Our approach to infrastructure management is

On-premise infrastructure isn't the problem – our approach to infrastructure management is

Over the years, I've worked with many large enterprises, helping them move to the cloud and achieve the exciting benefits it promises; autonomous provisioning, "limitless" scale, access to the latest technologies, and immediate, on-demand resources.

In these larger organizations, I've seen IT infrastructure usually managed as a cost center, disconnected from the evolving business capabilities that it supports. The infrastructure team is either at the beck and call of those leading business initiatives, often scrambling to provision project-critical infrastructure at speed, or tucked away out of sight executing on a large plan constructed and signed off months ago. Both of these result in long lead times for important projects.

By the time one project goes live, others are left delayed, and the technology landscape shifts again – leaving you with a new infrastructure that's in need of an update from day one.

On the surface, it would be incredibly easy to fall into the trap of thinking that this is a problem caused by traditional on-premise infrastructure. It's slow, takes a long time to procure then deploy, and ultimately limits what businesses can achieve quickly.

However, the truth is that if a different approach to infrastructure engineering isn't considered, exactly the same thing will happen in a cloud-based data center. Infrastructure teams can become just as overburdened by requests, and processes can be just as slow, if not slower than they were in a physical, on-premise data center.

To get the results they really want to see from cloud infrastructure, organizations must define and implement an infrastructure engineering approach that supports the dynamic and fast-moving world of software defined infrastructure. And a big part of that is shifting from viewing IT infrastructure as an operational expenditure, to seeing it as the business enablement engine that it should be today.

Refocusing from cost to business enablement
When organizations begin planning their cloud infrastructure migration, the same question comes up time and time again: "How long will it be before we see ROI from the cloud?"

The answer to that question is often as unsatisfying as the question is reductive. If you're constantly focusing on managing infrastructure costs and ensuring that cloud delivers measurable ROI within the data center, you're missing the broader point of cloud transformation.

Figure 3 shows the operations of a modern infrastructure team, following a platform-based approach to create reusable, self-service solutions that enable and empower multiple teams across the business, and help them to work faster.

As the platform and the services it provides become more mature, it takes far less time to build and deploy the infrastructure that developers need. Products are reused and deployed instantly as needed, delivering superior time to value.

As the platform offering matures by creating more re-usable elements, these can be easily combined to create new product experiences for the product teams, and by extension the business.

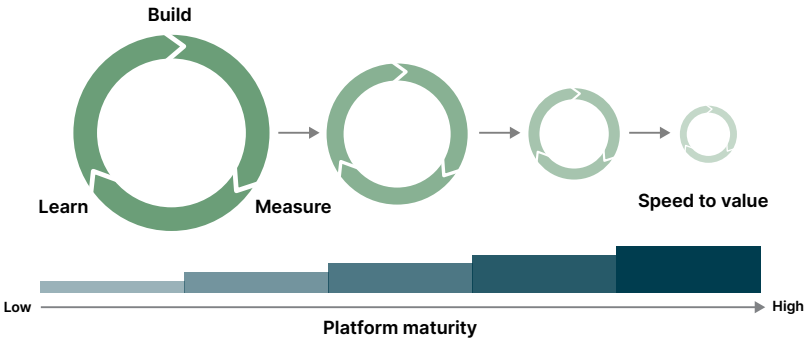

Figure 3. Platform speed experimentation and experience delivery

Within the data center and core IT team itself, the model doesn't necessarily reflect higher ROI. However, what it does reflect is the creation of ongoing value to the wider business, and faster enablement of new business initiatives.

It's this value that teams should be looking for from cloud infrastructure. That is where a platform-based approach has the ability to transform the wider organization and support business goals – not through the delivery of raw ROI within the four walls of the data center.

Do you have the right skills for the job?
Before you embark on a cloud modernization journey, it's important that you consider who's going to lead that journey and play a central role in defining its success.

Cloud infrastructure is entirely software-based, and brings with it a new set of challenges. While more traditional IT operations knowledge is still valuable – and in many cases critical – the modern landscape demands deep appreciation

for cloud API orchestration and continuous delivery software techniques such as automated testing and promotable programmatic changes.

One of the biggest pitfalls in this process is seeing the shift to cloud as a natural evolution for infrastructure teams, instead of the complete transformation that it really is. As cloud infrastructure is entirely software-based, cloud infrastructure management becomes a software delivery exercise – something entirely new to traditional infrastructure teams.

If transitioning from a more traditional data center team, it's likely this infrastructure team is the newest to the software engineering game of all your teams involved in the software development process – and they'll need to be supported.

The people writing software around these new cloud products need to have a continuous learning mindset. It's no longer sufficient to be great at one thing. To be successful today, we must build teams that are proficient in rapidly upskilling in new and emerging cloud technologies.

There's also increasing pressure to demonstrate the value of what are likely to be – initially at least – significant cloud infrastructure investments to the business at large. That requires a deep understanding of how a positive developer experience, ease of access, and overall development agility and speed deliver value – all measures that traditional infrastructure teams place limited focus on.

Platform engineering provides an opportunity to do just that, by measuring the impact of cloud engineering on business growth and customer satisfaction. But, once again, it represents major changes in how your team will operate.

What happens if your team doesn't change?

Or you have no team at all?

What happens if your team doesn't change? Or you have no team at all?

When it comes to managing a cloud migration and embracing a Platform Engineering approach, there are two main paths that companies with existing traditional infrastructure teams get pulled towards: Either they have their current infra team inherit and drive their cloud journey, or have development teams self-serve and run their own infrastructure.

Now, we'll look at the challenges, advantages, and drawbacks associated with each of those approaches in detail.

Scenario 1: Your infra team inherits your cloud journey
Most traditional data center teams have a wealth of experience racking and stacking hardware, writing scripts to keep machines running healthily and patched to the latest supported OS, and restoring service to business-critical applications.

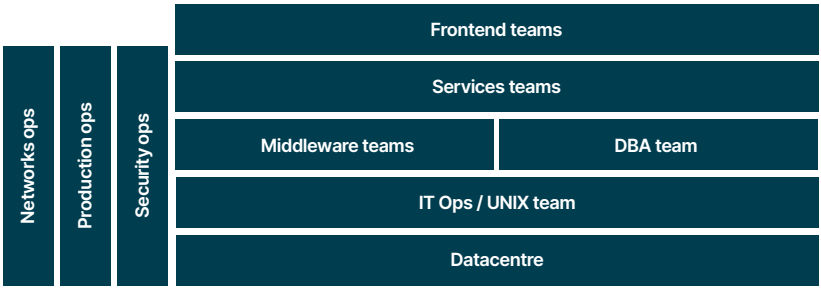

Figure 4. Areas of responsibility traditionally mapped around technology concern rather than business capability or outcome

However, transforming a team like this – one built on the principles and practices of traditional infrastructure – into one that is cloud-focused and software-driven carries some significant challenges:

1. It's a change of craft
Writing well factored, testable code is a different craft and, like others, takes years to learn and even more to master. Putting code freshers on your critical cloud path without adequate support or expertise will lead to a production infrastructure that is hard to change with any degree of confidence, even if it never goes down. Of course, equally impractical is waiting several years for them to graduate in **Infrastructure as Code**.

2. Adapting to new positioning and structure
In my experience, data center infrastructure teams rarely see developers as customers of their systems. At a push, they may brand developers as "tenants", which at least indicates their systems have users as occupants, and there is an unspoken acknowledgement of services being provided and consumed.

This simply doesn't align with the relationships and structures that need to be in place to get the most from Platform Engineering and cloud infrastructure. In an ideal Platform Engineering world, core infrastructure teams work primarily to serve developers and empower them – which is not the case in many traditional data center structures.

3. Changing measures and definitions of success
Traditional infrastructure departments measure success in availability and stability of infrastructure services. While important to maintaining operations, those goals naturally

lead them to limit change to reduce chances of service disruption – something traditionally seen as a positive. Of course, that also seriously limits the value their team will ever see from cloud infrastructure. The team in charge must embrace its speed and flexibility, and be change-oriented.

Getting traditional data center infrastructure teams to refocus on best serving their internal customers – your developers – and drive positive change proactively instead of shying away from it is unfortunately often a very difficult task[3].

3. This clash of working styles is fundamentally the original DevOps dichotomy

Scenario 1 in action:

15-year-old fintech enterprise

Scenario 1 in action: 15-year-old fintech enterprise

I recently worked with a 15-year-old fintech enterprise which had just this - a data center infrastructure team rebranded as “Platform Ops” who were eager to “learn cloud” and start migrating workloads and teams in order to “get out of the data center”.

Little support was provided for learning how to manage and execute this effectively, and the team were constantly getting pulled into production issues caused by the fragile heritage tech stack.

To solve this problem, a new team was created called Platform Engineering, seeded with some new cloud engineer hires, and a few of the existing team who had demonstrated an ability to code and an appetite to stretch out of their comfort zone. The new hires brought software engineering values and strong skills in the Infrastructure as Code space, and the infrastructure team knew the production support systems, traffic patterns, and various fragilities in the existing stack.

Their mission was set to provide a cloud platform that enabled continuous delivery for developer teams. Org positioning: check; craft: check. Looking good.

The team worked for 18 months to build a platform that enabled developers to provision their own cloud components by submitting Terraform[4] pull requests to the Platform Engineering team. The team would then merge and deploy the applications to a load balanced set of VMs, via a continuous delivery (CD) pipeline.

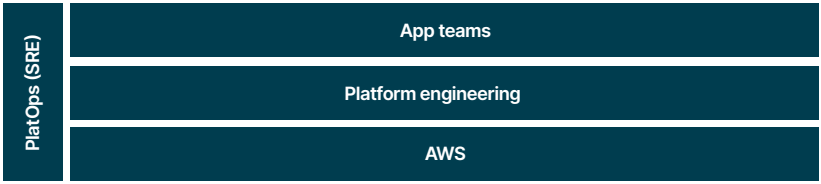

Figure 5. 15 year old fintech enterprise with the right idea of business value driven platform layers

After a year and a half of coding, however, only 2% of the company's production workloads were flowing through the new platform. Why?

Figure 6 shows the cross-backlog dependencies that this approach created. It wasn't a true self-service approach, because in many cases, other teams and individuals had to contribute to and edit a request before it was given the go-ahead.

Each dev team had to learn this new infrastructure approach, and write their own code before raising a request. Then, since that code had been written by a different team, it had to go to the platform team for review and feedback – introducing new inefficiencies, and creating dependencies that slowed deployment and time to market.

As new dev teams submitted more code, that inefficiency began to scale (shown in the bottom half of the diagram). There were more requests to review, but still only one platform team to review it all, creating a major bottleneck that slowed deployment speed further.

4. https://www.terraform.io/ - a popular infrastructure-as-code choice for orchestrating public cloud APIs

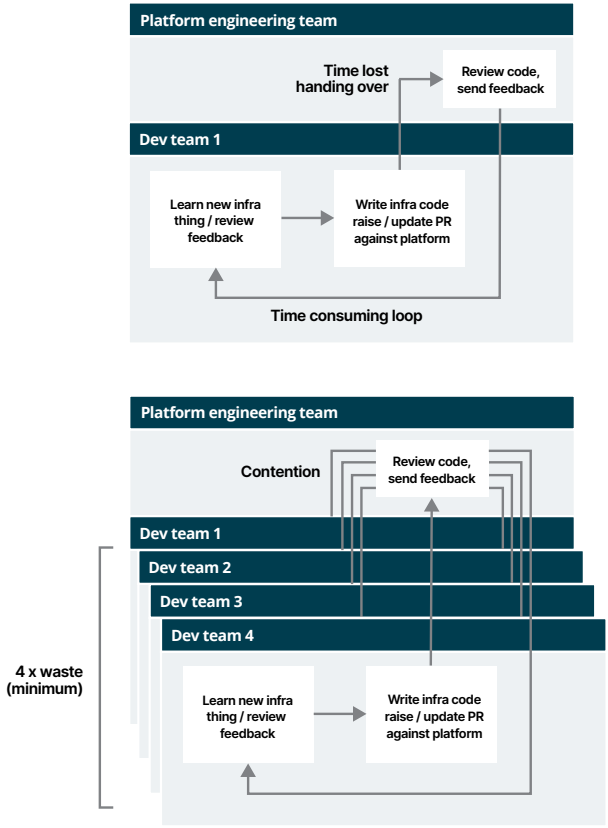

Figure 6. Cross-backlog dependencies on new platform

Finally, because each request was being created by an individual dev team to meet their specific needs, the approved outputs weren't always reusable or useful for other dev teams. They weren't creating reusable products that others could consume. So, every time a new team needed infrastructure, the whole process had to be repeated, ensuring that the review and feedback bottleneck grew with the number of teams and services.

How cross-backlog dependencies will slow you down up to 12x
One of the underpinning tenets of agile software delivery is the drive for autonomous teams. Having the required skills to design, develop, and deliver a product in the same team minimizes communication overheads and context switching, and ultimately reduces the potential for waste.

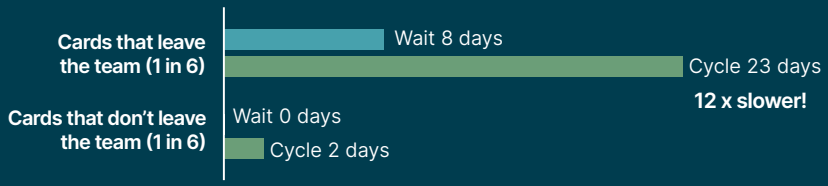

Figure 7. Cross-backlog dependencies at large Australian Telco
GOTO 2015 How I Finally Stopped Worrying and Learnt to Love Conway's Law - James Lewis

During a study at a large Australian telco, we measured stories with dependencies on other teams cycled 12 times slower than those that had no dependencies and were accomplished within the team. A self-service approach aims to reduce the likelihood of cross-backlog dependencies on a team like Platform Engineering.

The measures of success in place at the fintech were around how many services had been deployed to the cloud, regardless of whether they were receiving production traffic.

Rather than writing a simple self-service interface to accommodate developers and best serve them as customers, the Platform Engineering team were more likely to write the code themselves. The success of this pattern required all

developers to be curious polyglots, and even then the limit reusability would restrict the scalability of the pattern.

The initial organizational positioning looked good: a Platform Engineering team was put in place to harness the cloud. However, moving to the cloud was seen as a Platform Engineering concern instead of being communicated as a developer, product, or business concern.

The boundaries of ownership were confusing from the beginning, making it difficult for developers to drive the final steps of the transition. An absence of value-driven product thinking and delivery coordination meant that migration activities stalled as soon as tasks spanned multiple teams.

The initial outcomes didn't match anybody's definition of success. Which begs the simple question: What was missing?

Embedding key software delivery roles in an infrastructure team

If we want to help infrastructure teams function as software delivery teams, we must ensure that all of the roles traditionally held in a software delivery team are also held within the infrastructure team.

Based on the example we've just explored, there are two main software delivery skills missing from the picture: product thinking and delivery management.

Product

Putting your product and marketing hats on for a moment, you can see the issue above: customers had to change their behaviors in order to make the product (the cloud platform) successful. If behaviors didn't change, product value diminished.

When building a new product, a product manager will always be looking at the customer, whether they're current or prospective, to understand and map their needs to ensure that what's delivered meets those needs directly.

That's where our young fintech enterprise went wrong. The absence of any product thinking gave rise to a mismatch between purpose and need.

Delivery

Product thinking wasn't the only cause of such a small amount of product uptake. An absence of delivery management meant that a plan with someone to coordinate, publish, and drive it was a shared responsibility and therefore often a lower priority for the people involved.

Although only 2% had been migrated and was receiving production traffic, more than 40% was almost ready but missing crucial developer input for testing and transition. So, not only was the interface tricky to use for most developers, but the benefits of moving to the cloud were not understood, and the work wasn't being prioritized by the business.

In summary: The challenges of trying to fit square pegs into round holes

Hopefully, what you've taken away from this scenario so far is that, in practice, it's extremely challenging to turn a traditional data center infrastructure team into a cloud-ready infrastructure or Platform Engineering team.

The measures of success are different. The skills required are different. The outlook and need for a strong customer focus are different. If these challenges presented themselves

alone, perhaps retraining or re-education could do the job. But together, they make a very compelling case against repeating this scenario for yourself.

However, that's not to say it's impossible. It can be done effectively, but the business must be committed to the change – an essential characteristic for any **modern digital business**. And not just superficially either – teams must truly want and be ready to adapt what they do and who they do it for.

Scenario 2 :

Dev teams run their own infrastructure

Scenario 2: Dev teams run their own infrastructure

Of course, this pattern of pushing a square data center infrastructure team peg into a round cloud platform hole isn't representative of what every company will try to do when faced with this challenge.

An alternative infrastructure team pattern exists, more commonly seen in smaller organizations, occasionally referred to as "noops", whereby no team is dedicated to the company's infrastructure at all.

Here we see the simplicity of cloud return once more, enabling developers to provision and manage cloud infrastructure themselves. The range of products and services they access might not be straightforward, but their means of accessing them certainly are.

Scenario #2 in action: Small full-stack dev team at a high-street gym company
I recently had the pleasure of working with a 10-person full-stack team who built, deployed, and managed a new web-based membership platform for a popular high-street gym. In the beginning, everyone was happy to contribute to both infrastructure and website (Java) code, creating a very resilient team, arguably a DevOps utopia.

On first inspection, this may appear to represent a perfect pattern of shared ownership and low-latency change. The code wasn't monolithic and was intelligently organized into separate repos: one for each application or infrastructure component.

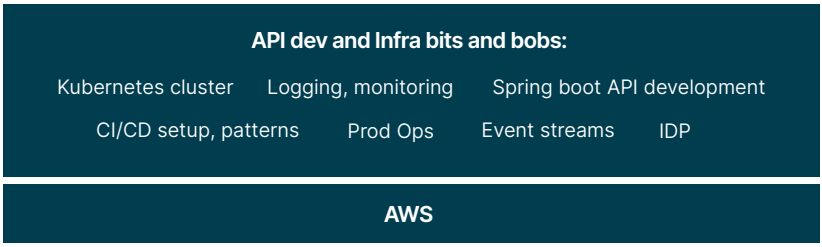

Figure 8. Scenario 2: An all-the-things platform layer may look efficient, but risks blurring important boundaries

However, as one service became two, and two became seven, it began to get cumbersome to add new services and adjust the infrastructure. As more people joined the team to support the increasing workload, it also became harder to maintain the proficiency and desire to build expertise across all areas of the platform.

Getting production-ready became the primary concern, which in turn led to people deepening their skills rather than broadening them. A decision was made to split the teams into two: one focusing on application and API development, and the other on everything else: the Kubernetes cluster, CI/CD setup, identity management, logging and monitoring infrastructure.

Who owns platform services when they don't belong in any existing domain?

Even if we had managed to refactor the codebase and upskill people to overcome these challenges, the question still remains of who should own these platform services that don't quite belong in an existing domain or team, but are depended on by multiple teams.

Take secrets management for example. It's pretty common these days to configure a secret store, such as HashiCorp Vault, first

setting up a trust system between application, secret store, and interactive user, and then maintaining this system over a long period of time.

But since this is a common resource that multiple teams will consume, it's not immediately clear who should own it as a service. Here, you've got a couple of options for how to manage that:

Option 1: developer team independence
Everyone has their own secret store. This leads to an obvious case of duplication, but there are a lot of benefits to be gained, including clear ownership and decoupled architecture. However, as the number of teams scales so does the cost of maintenance, linearly. n teams, n secret stores, n systems to maintain.

Option 2: Delegating one team as system owner
Here, one team is the delegated owner of the system and maintains it for the purposes of all other teams. The service becomes an internal product for internal customers, and with that should come service availability expectations, channels for feedback, and ideally a support and modernization roadmap.

Option 2 sees the service treated like any other, except the consumers are internal developer teams, just as if it were an internal core API. As the number of developer teams increases, the load on the service will certainly increase, and so might the complexity of use cases for the service.

This must be managed carefully to avoid the same linear scaling of complexity as seen in Option 1. Many organizations are attempting to solve these problems by folding such internal services into a Platform Engineering team.

In summary: Why direct self-service isn't the answer
It follows from Conway's Law that having a team (or collection of teams) responsible for a domain will help keep boundaries well defined and areas of ownership clear, making it easier to maintain and make changes over time.

This is what you sacrifice by enabling a dev team to entirely help itself to infrastructure services in the cloud, with no team taking direct control and responsibility for shared services.

What should be becoming clear now is that neither of the scenarios outlined offer a perfect approach to cloud-based infrastructure transformation and Platform Engineering. The teams in each of the examples explored had their own issues and challenges, but in hindsight clear indicators of diminishing value appeared early on.

In the case of the young fintech, boundaries of ownership were unclear, creating lots of back and forth "chatter" between developers and platform engineers, and slowing processes. Things were slowed further by a large number of dependencies that coupled developer team backlogs to Platform Engineering.

The organization in scenario 1 also lost sight of the business value cloud was intended to deliver. The Platform Engineering team in charge of migration was made up of engineers only, which meant that business value was often overlooked, and delivery commitments were repeatedly missed and had to be reset.

With the 10-person full-stack dev team, frustration built quickly as application developers struggled with infrastructure concepts. Motivation to work across all domains declined as complexity increased and demanded deeper levels of understanding.

Inefficiencies started to creep in quickly, and we saw domain bleeding and manually-intensive changes when adding new services. And as new teams were onboarded, we saw a need for shared services emerge – which no individual team took full responsibility for.

Evolution of platform services

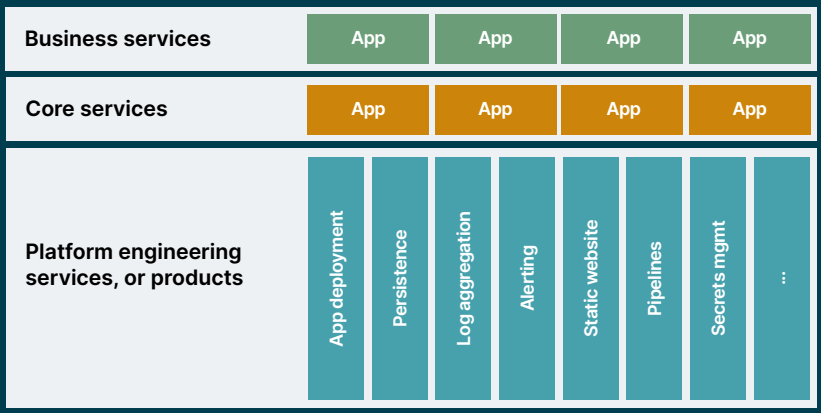

Figure 9. Platform services

This is an example of some platform services that evolved over 2 years of platform engineering at the 15 year old fintech. We didn't get it right the first time: in the beginning we established logging as an opt-in service before realising that every internal customer wanted logging for everything they did. So it became a baked-in component of every other platform service. These services became the composable building blocks developers would use to deliver new customer experiences.

The five pillars of product-oriented platform engineering

The five pillars of product-oriented platform engineering

With the lessons of the shared examples learned, it's clear that there is no single 'right' path forward for organizations looking to transform how they manage infrastructure services through a cloud-based Platform Engineering approach.

However, there are five factors that we know contribute to the long-term success of this approach:

1. Focus clearly on Developer Experience
Successful Platform Engineering teams must be able to think like product marketing teams, carefully considering their go-to-market – or rather go-to-developer – strategies.

Just as product and marketing teams maintain a laser focus on delivering seamless, intuitive customer experiences, cloud infrastructure teams must do the same for developers – putting them at the heart of every decision to ensure high uptake and long-term success.

If, for example, engaging with a platform requires a developer to learn a new language just to continue doing the same work they're already doing, that doesn't represent a good developer experience. And, just like a customer who is being asked to jump through too many hoops to buy a product on a new ecommerce platform, they'll simply walk away and revert to however they were doing their job previously.

2. Define and build in measures of success
There's currently a well-deserved buzz around the four key metrics as defined in Accelerate by Nicole Forsgren, et al. If your teams already have continuous delivery pipelines

in place, a good monitoring system, and well-bounded services relating to business capabilities, then it may not take too much effort to start measuring these high performance indicators across all of your developer teams.

But regardless of the stage your teams are at, it always helps to put some leading indicators in place to ensure your project is making a positive impact and having the desired effect on the organization at large.

Even if those indicators show you that adoption is low, for example, having them in place enables you to identify that and take the right actions at the right time to remedy any issues and get the project back on track.

3. Enable self-service and unlock scalability

The key to high platform scalability is to give developers the ability to serve themselves and access the infrastructure services they need without direct intervention from the infrastructure team.

In the example of the young fintech, we started this off small. By using simple existing features (labels in JIRA, in our case), we asked developer teams to flag any user stories where they felt Platform Engineering would need to be engaged to build or modify a supporting capability. We held weekly meetings with developer teams to review upcoming dependency candidates, pointing teams to existing tools, services, or documentation that would help them self-serve, removing Platform Engineering from the equation.

Figure 10 shows that as new teams were onboarded to the platform, we gradually saw the number of these flags fall. Teams learned quickly that much of what they needed already existed

within the platform, and by the time the fourth and fifth teams were onboarded, they were able to bootstrap and deliver a service to a production staging environment without a single request to the Platform Engineering team. Even without any financial data to back this up, the huge business benefits of that were clear to the organization.

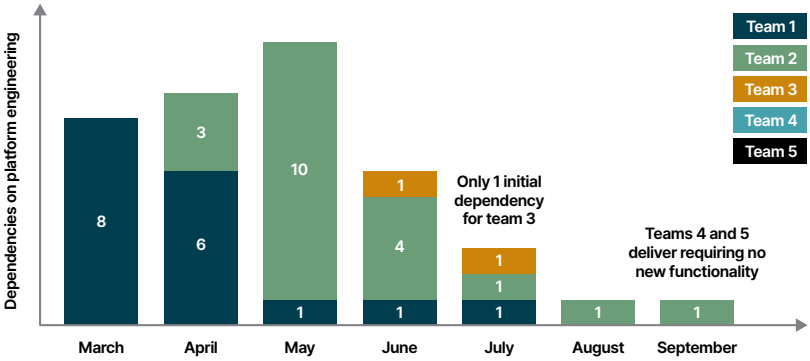

Figure 10. Indicators: cross-backlog dependencies

4. Agree on what's most valuable and important to the people you're serving

It's worth noting that while measuring impact (among other metrics) is important, what's more important is for everyone to understand what the customer of Platform Engineering – primarily developers[5] – want to get from it.

By clearly defining what developers, infrastructure teams, and the organization as a whole want to achieve through the platform approach, you'll have the best chance of actually delivering those outputs.

It's the same principle as customer-centric design. By building products around what you know customers want and want to

5. Other customers exist depending on the organization e.g Security, QA, SRE, Operations

achieve, you can deliver the best outputs and outcomes for them – as well as your organization.

5. Assign a product owner

All modern infrastructure is now predominantly configured by code. Yet we continue to see infrastructure-based teams lacking all the usual roles (and in many cases, disciplines) of a typical agile software delivery team.

Appointing a delivery lead and product owner can yield value in even the most modestly sized internal platform team. I also hope that quality and testing specialists will lean into this space, as there is a huge opportunity for contract testing, for example, at this level of the platform. And there's arguably a stronger need, since this is the foundation for everything else in the organization.

Having a product owner in particular though, reinforces the idea of the internal customer and the desire to understand their needs. It immediately inspires the concept of a business aligned roadmap, and hopefully encourages feedback loops with customers, with early success metrics.

There's enough work involved to warrant a person full time in the role, so don't give in to the temptation to share the responsibilities across the team. Dedicating a person to the role creates a stronger sense of accountability than simply assigning additional tasks to existing team members.

It will help draw attention, and at the same time send a message of intent to change, commitment to improve, and ultimately, your intention to start viewing and treating infrastructure as a product.

Key takeaways

Key takeaways

In summary, I'll leave you with the following key pieces of advice to use when reviewing your cloud infrastructure capabilities, the teams involved, and how well these are accelerating your internal developer teams:

- **Understand your skill gaps** – Put processes in place to help solve them. Set the right expectations. Hire in experts to help deliver, upskill, and increase the pace of the infrastructure team's transformation.

- **Be product-centric** – An infrastructure team is a product team. Introducing a dedicated technical product owner immediately refocuses purpose and helps map customer value back to the specific services the team provides.

- **Look for early, cheap measures** – If the four key metrics take time for the rest of the organization to prioritize, don't give up trying to measure your progress and set flags for when you're off track. Think small, cheap, and effective.

- **Domain boundaries matter** – Loosely coupling infrastructure is hard, but you can usually spot when it's not going well. Keep reviewing your domain boundaries, between platform products, and between teams and other parts of the architecture. You're building the foundation of your stack; mistakes will cost more, so architect with evolution in mind and optimize for change.

- **A single platform team may not be enough** – As your suite of internal products grows, your team will need to diversify and specialize appropriately. Just like with the small team at

the high-street gym company, let the domain boundaries form and note how each product and associated roadmap evolves. It's good practice to separate a large team into different sub-teams to support these changes.

- **Be customer-focused** – Developer experience and productivity impacts delivery efficiency in terms of quality and speed. Happy, productive developers are a key (if not the key) to any successful organization, and this is achieved by powerful, self-service platform capabilities that get the job done without introducing productivity friction. Remember: if developers (customers) don't want to use your stuff, you failed. It doesn't matter how neat your product or service may be.

About the author

Max Griffiths
Principal Infrastructure Consultant, Thoughtworks UK
Max is a leader of Infrastructure and Platform Engineering for Thoughtworks UK. In 16+ years he's worked across almost every industry sector which has provided both breadth and depth of experience in solving problems in very different contexts. The key constant has been a focus where technology meets the business, helping modernize and drive value through a combination of engineering, product leadership, and continuous delivery practices. Max started as a shell scripter in the early 2000s, fulfilling an urge to automate everything. Since then he's been a Java developer, infrastructure developer, business analyst, project manager, a Platform Engineering SME, and enjoys the people and technology challenges alike. Having lived and worked in Europe, Philippines, Australia, North America, Singapore, you might be forgiven for thinking he likes to move. He'd say it's all led by food. (And a passion for the world, people, and variety therein). More recently he's been working with our client CxOs to help build technology strategy, delivered via digital platforms, that helps accelerate growth and innovation across the organization.

Get in touch with us

contact-uk@thoughtworks.com

Thoughtworks Ltd.
First Floor, 76-78 Wardour Street
London, W1F 0UR, UK
+44 (0)20 3437 0990
thoughtworks.com

/thoughtworks