WHITE PAPER: **June 2026**

Why You Should Treat Your Platform as a Product

vmware® by Broadcom

Table of contents

Introduction: A platform is what you use to build and run your custom-made applications

A platform is the product you use to build, run, and manage the applications your organization depends on. In the "platform as a product" way of thinking, the goal is to use self-service and automation to accelerate deployment and to streamline the entire lifecycle of software. It means developers can ship code without worrying about infrastructure, and operators can enforce security and compliance without slowing everyone down. When you have a platform, instead of creating and configuring infrastructure, developers can spend their time designing and building applications.

The phrase "as a product" is used to highlight the practice and mindset you use for your platform. With this mindset, you think of the platform as an ongoing, evolving product. You think of developers as the customers for that product. And, so, you use the practices of product management and design to keep your platform up to date, useful, and valuable. Operations people typically take on this ongoing responsibility.

The Day 2 problem

Most platform discussions focus on the developer experience. Very little discussion focuses on what happens after the first day, when an application is deployed. "Day 2" is when most of the work and cost start. It's also when most of the risk kicks in and needs to be handled. This is especially true at larger organizations.

Day 2 describes running, monitoring, managing, remediating, scaling, securing, auditing, and paying for the application. A good platform delivers on more than just developer experience. The platform should also deliver on operator experience. Both are important, but the second is often neglected.

This paper covers what a platform is and does for your organization. It covers how to evolve the platform so that developers actually use it (lack of adoption is a surprisingly huge problem). It also covers the operations experience. We first published this paper in 2017 and have updated it several times since, including the 2026 version you're reading now. While technologies change, the care and feeding of them has changed very little. These practices have been road-tested by working with organizations like JPMorgan Chase, Charles Schwab, numerous militaries and government agencies, and other large organizations.

The new bottleneck that will slow down your AI adoption

Getting software from idea to production quickly has always been important, but the surge in AI-driven application deploys is about to overwhelm even the best-laid golden path to production. We can feel in our bones that developers are already reviving apps more frequently. The 2025 DORA Report study found that 95 percent of developers use AI and over 80 percent say it has made them more productive. And that's just developers.

Imagine every employee in a company—marketing, operations, finance, HR—using agentic AI apps that let them build and deploy their own applications. Not just once, but iterating multiple times a week, even daily. And the AI agents themselves will be creating additional agentic applications to carry out tasks, test code, build workflows.

"AI adoption now improves software delivery throughput.... However, it still increases delivery instability. This suggests that while teams are adapting for speed, their underlying systems have not yet evolved to safely manage AI-accelerated development."

-2025 DORA Report

The volume of software hitting your infrastructure is about to multiply in ways that will make the current CI/CD pipeline look like memories from a long-ago golden era. Without a platform to handle the builds, the deployments, the security scanning, the compliance checks, infrastructure will become the bottleneck that stymies enterprise dreams of AI productivity gains and ROI.

Without a real application platform and a platform engineering team to support it, the benefits you get from AI will be negligible. Everything will bottleneck at the deployment and runtime stage when your systems are overwhelmed by the velocity of AI apps. *With* a properly supported platform, that bottleneck can
be eliminated.

A good platform will help turn the flood of AI-generated applications into a manageable flow. An AI-minded platform will also add mandatory enterprise capabilities like security controls, governed data access, and the runtime management needed to keep those apps running and performant. Without a platform, getting a new app to production means filing a ticket to the infrastructure team, conducting a security review, and enduring weeks (if not months) of meetings. A platform replaces all of that with a self-service path to production.

When the number of people who can create software expands from your development team to your entire workforce, what a platform gives you goes from "nice to have" to "the thing that determines whether your AI investment pays off or just creates more enterprise sludge."

Platform engineering

Nowadays, platform operators are often called "platform engineers." This phrase is good, but as we'll see, it can lead to the ROI-wrecking assumption that platform engineers should be *building* platforms. One of the tricky parts of running a platform is to figure out when you are doing platform engineering versus platform operating.

Day to day, you're likely operating a pre-built platform, like VMware Tanzu Platform. Engineering is done more rarely to fill in the gaps, customize the platform, add in new functionality, and help people use the platform. In contrast, for a DIY platform, platform engineers are constantly engaged in the low-value, maintenance activities of building and integrating infrastructure components from scratch, which distracts from the core mission of helping application developers.

The stack

When it comes to defining what a platform is, here is the best, vendor-neutral diagram of a platform that we currently have. It comes from the Cloud Native Computing Foundation platform working group.

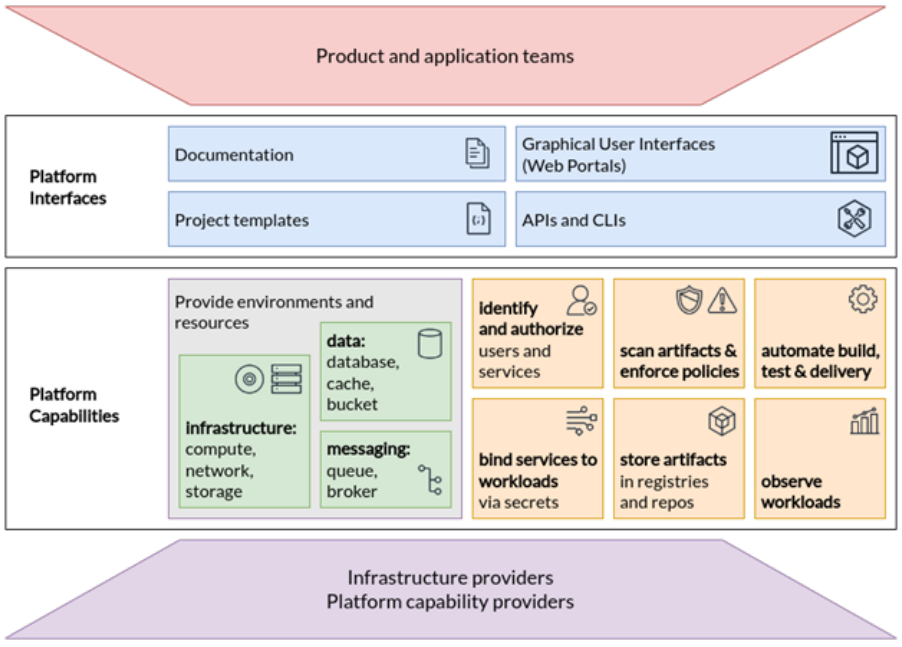

Figure 1: A diagram of a software development platform, showing the components and functionalities necessary for it to work. Credit: CNCF

What should jump out at you is the breadth of functionality. Each box represents a group of functionalities needed to build, run, or optimize applications. The scope is not just a build pipeline. It's not just providing base images, Helm charts, and Terraform templates. It's not just managing clusters of containers. It's not just self-service access to databases. It is all of the things needed to take an idea from code to test, deploy, run, and monitor in production.

It's this integration that makes the platform valuable to developers. Instead of building and integrating services together, platform engineers act as stewards of the quality and capabilities of platforms. They run the platform and help developers use it. Just as with developers, platform engineers generally don't have time to build platforms from scratch, or the patching and upgrade overhead that comes with a bespoke platform.

This DIY approach creates significant operational overhead, particularly in Day 2 activities like monitoring, patching, and securing applications running on top of unmanaged Kubernetes or bespoke platforms. Pre-built platforms like Tanzu Platform handle much of this underlying integration and lifecycle management, allowing platform engineers to focus on customization rather than maintenance.

For both developers and platform engineers, time spent building the actual platform is a low-value activity and a distraction from focusing on accomplishing whatever business goals you have. "Good job building a framework to build, configure, and deploy containers," said no CEO ever.

This sentiment is why many organizations decide to start with a pre-built platform, like Tanzu Platform, instead of building their own. The platform engineering team then focuses on customizing the platform for their organization's needs and building the truly unique services that help differentiate their business.

A platform is not infrastructure

You can think of a platform as your own "cloud" for running your applications. One key aspect of a platform is that it's infrastructure agnostic.

A platform needs some infrastructure to run on, but managing that infrastructure is not part of the platform, nor is it the responsibility of platform engineers. If your organization's platform engineers are spending time managing and building infrastructure, something is wrong. Platform engineers do their most valuable work when they are making the platform better. As we'll see, doing this is a full-time job.

Platform as a product

What does the platform-as-a-product approach look like? What are the activities?

Our view of the platform-as-a-product mindset has evolved through many stages. It was first developed around 2015, originating in our Tanzu Professional Services team (which evolved from the VMware Tanzu Labs consulting practice and formerly VMware Pivotal Labs).. What we'll go over here is the maximalist methodology that we've co-developed with customers. Based on our experience, we think the following is a tasty recipe. But we also have learned that perfection is not always possible. You can use this as a starter playbook, modifying to taste.

Product management (the "what")

Most infrastructure teams operate as ticket-takers, i.e., a developer asks for a private LLM instance, a vector database, or an AI model governance service and the team provides it. Product management shifts this dynamic.

A platform product manager (PM) manages a backlog of capabilities, not a queue of tickets. They act as the buffer between the platform engineers and the endless stream of requests. Their job is to ensure the team focuses on high-leverage work prioritizing capabilities that deliver the highest return to the organization for engineering effort. This means saying "no" more than "yes," but this product lens is what heads off platform sprawl and paralysis.

The PM's work centers on a few concrete practices:

- **Constant backlog prioritization** – Sorting requests for new middleware, integrations, operations, security, compliance, etc. based on organizational value, not just who is shouting the loudest.
- **Customer research** – Getting out of the building (or virtually) for briefing sessions, brown bag lunches, and user interviews with app developers. You can't prioritize what you don't understand.
- **Hypothesis testing** – Using minimal viable products (MVPs) to validate assumptions before committing to comprehensive feature lists.
- **Evidence-based decisions** – Requiring data before action. Creating a new entry in the MCP server catalog requires usage data showing demand for that integration, not just a gut feeling.
- **Value stream mapping** – Proactively identifying waste in the path to production to find the next automation opportunity.
- **Stakeholder management** – Maintaining connections with business partners and IT leadership to communicate why priorities shifted.

The PM validates platform changes *before* implementation, balancing developer experience (speed) with operator experience (stability). Instead of just asking developers what they want, they validate utility and maintainability. For example, they might ask: "If we built this, would it be valuable enough to justify the long-term cost of running it? Will it help developers deliver what the business needs?" If the answer is no, the feature slides down the backlog in favor of other requests.

Focusing on user needs (UX)

Developers are your customers, but they aren't captive ones. If your platform is a pain to use, they'll find a way around it. That's how you acquire shadow IT.

User-centered design (UCD) in a platform context means building for what developers actually need, not what you think they need. It's about cutting down the cognitive load—making the platform intuitive enough that developers spend their mental horsepower on their applications, not wrestling with your infrastructure. You build the thing that fits their workflow. The CLI, the API, the error messages, and the documentation; that's all part of your product interface.

The core UCD practices in platform-as-a-product:

- **Talk to developers constantly** – Engage in regular dialogue through interviews and briefings. Ask them what's working, what's broken, where they're stuck. Then circle back three months later to refine what you learned because their needs will likely have changed in that time.
- **Watch them actually use the platform** – Sit with app developers (virtually or physically) and watch them try to deploy code. You'll see where they get confused, where they copy-paste commands they don't understand, and where they give up and Slack you for help.
- **Test everything with real users** – No guessing games. Not just surveys. Real usability testing with actual developers doing actual work. Demo early, demo often, and take the feedback to heart.
- **Design the path through the platform** – Create personas and map out the developer's journey from "I have an idea" to "it's running in production." Where does the journey bog down? Fix those parts first.
- **Make it delightful** – Design isn't just about looking good, though that helps. Make developers excited to use it. There are elegant ways to handle a command line interface and even configuration files.production to find the next automation opportunity.

The goal is simple: Self-service and fewer headaches. If a developer can't figure it out without DMing you, you haven't built a product; you've built a help desk. The platform should be so obvious, so frictionless, that using it is faster than hacking around it.[1]

Site reliability engineering (SRE)

SRE treats operations as an engineering problem by using software to manage and maintain systems. Agile software development practices are applied to add new features to your platform and automate away toil and waste. In SRE, toil is the manual, repetitive, and automatable work that is tied to running a production service.

SRE balances the traditionally conflicting goals of velocity and reliability. Your platform team and your application team will determine an appropriate level of reliability for the platform and for each application, while maximizing feature velocity. SRE regards 100 percent reliability as the wrong target for the platform or for applications running on a platform.

Instead, the platform team and app development team together define an error budget, which quantifies how reliable the product needs to be over a particular time period, as experienced by the end user. The app development team is responsible for planning how much change they can safely introduce—and on what schedule—to stay within their error budget.

Prerequisites for adopting a product mindset

Adopting a platform-as-a-product mindset is most successful when the adoption is recognized as transformational and is supported by more than the platform team. Part of the transformation is a shared desire to change for the better and challenge the status quo. As with many new technologies and ways of working, platform engineering is trying to address problems created by siloed organizations. The platform spans many teams, groups, and often lines of business. This means culture and politics become bottlenecks for platform adoption, developer productivity, and otherwise realizing business value with the platform. That's why minding the "people problems" is important.

Here are a few ways to make introducing the platform-as-a-product approach easier in your organization.

1. For a longer discussion of design in platform engineering, see "Designing for Success: UX Principles for Internal Developer Platforms," Kirsten Schwarzer, KubeCon EU, March 2024.

The current interest in AI is an example of a "why" for platforms.

Taking advantage of agentic and generative AI often means incorporating AI into your existing applications and building new applications.

You want to use AI in unique ways that help your business. In this case, your enterprise AI strategy is about building, running, and managing the applications, data and tools that agentic applications need to access (like MCP servers).

Building these agent applications *and* discovering how to use AI for your organization is a lot of work. Having a platform will help speed up that process and add in the centralized management, enterprise security, and governance controls you need.

We're seeing many organizations that have started with initial AI PoCs and projects outside of the usual way of doing things hit the AI Day 2 wall.

Once these projects are successful, integration with existing systems, scaling and replication of that success to other parts of the business is difficult without a platform.

Without a platform, each application team has to start from scratch, creating not only platform sprawl but AI sprawl as well.

The push to build a platform

Your organization should have strong motivation to start a platform-as-a-product strategy. In some cases, your IT organization knows that it wastes large amounts of effort building platform-like capabilities and wants to reduce costs. In other cases, a business unit in which multiple app development teams have already been forced to solve their own platform challenges wants to increase velocity.

Be clear on *why* you want a platform. This means knowing the problems you're solving and the outcomes you're hoping for. Usually, the problems you're solving are slow development cycles, too much fragmentation (sprawl) in how security and compliance are enforced, changing how and where you host your applications, adding new application types (like AI), and so on. The outcomes, of course, are improving each of those.

Knowing why you're putting a platform in place will help guide your decisions about platforms. You'll need to keep focus on the outcomes you want to achieve with your platform—the problems you want to solve, the capabilities you want to gain, and the outcomes you want to benefit from. Too often, platform teams focus on technology for itself, often as a result of chasing the technology fetish du jour.

Keeping track of why you have a platform in the first place will also help focus your organization on using the platform as a tool instead of an end to itself. Make sure you're frequently asking: Is the platform solving the problems we've identified? How well does it solve them? This will let you avoid gratuitous, fashion-driven decisions about replatforming and other boondoggle projects that turn out to be based on whim instead of wisdom.

There should be a lot of marketing that goes into introducing a platform as well. It's not too difficult, however. People will do and like things that benefit them. When talking with executives, find out what your corporate priorities are, and explain why having a platform will help achieve those metrics. When talking with staff, explain why using a platform will make their day-to-day life easier and more enjoyable.

Reliable local infrastructure

Try to avoid introducing changes in your infrastructure-as-a-service (IaaS) layer at the same time as embarking on the platform journey. You need a stable IaaS capability during the platform creation period. It's one thing to "change the engine while flying," and a far worse thing to change *both* engines while flying. Your platform relies on reliable, stable, automated infrastructure, so make sure you have that in place. You can jump-start that with integrated IaaS and platform stacks, such as with VMware Cloud Foundation (VCF) and Tanzu Platform.

Establish a production platform capability that's located close to—or ideally in the same physical location as—the existing applications and data where the platform needs to function. Otherwise, latency issues may be blamed on the platform. If you must have distance between the applications and data, establish a baseline with monitoring that you use to disambiguate changes to the platform or application.

Shared goals

Many stakeholders are involved in building a platform product, including app development teams, business application owners, change management, networking, production operations, release management, security, virtualization, and their end users. Defining shared goals ensures that stakeholders work together rather than against each other. Shared goals also change the conversation from, "Here's why this won't work" to "How can we make this work?"

We suggest using an inception exercise to align teams on the importance of the platform and how it fits into the bigger picture for your organization. Define quarterly objectives and key results (OKRs) for the platform (e.g., deploying the first app to production during the launch phase). Variable compensation schemes, rewards, and recognition can also be linked to OKRs.

Platform champion

A platform champion has the motivation and the political capital to protect the platform team as they embark on the platform journey. The champion is often at—or close to—the CXO level and has a track record of internal entrepreneurship or organizational transformation. This person understands and can articulate the value of a platform product and evangelizes its use and growth within your company. Popular business lore calls this a single-threaded leader, which we think is helpful.[2]

2. Check out *The Octopus Organization* for much commentary on this type of role and leadership

Dedicated platform team

Your platform team builds and runs your platform as a product. The team must have the authority to change the production process as well as the platform itself. A platform team consists of at least three full-time employees, satisfying the following functions:

- A product manager focuses on building the thesis for the platform product, testing that thesis with the target customer (internal application developers), and iterating on the features provided to achieve the desired customer and business outcomes.
- Platform engineers work through a feature backlog to implement new platform features on a weekly basis, and they manage the delivery of those features. This involves balancing the rate of change introduced with the desired reliability objectives. There are usually several platform engineers.

There's more on these roles below.

How to build your platform product

Your platform team will start small and gradually extend the platform's capabilities. We see three successive phases during which your platform increases in maturity, taking you from creating a platform team to running thousands of apps in production on the new platform:

- **Launch the platform capability**
- **Extend the platform product**
- **Use the platform at scale**

Let's look at the details for each of these.

Launch the platform capability

The main objective in this phase is to launch a minimum viable product (MVP) version of the platform and deploy a single production app on that MVP. This MVP of the platform is intentionally just enough to get the first app to production, not an attempt to bring parity with existing platforms, nor deliver maximum ROI.

Choose your platform team

The platform team consists of a product manager and at least two platform engineers. Over time, the first members of the team will be responsible for staffing and training future team members, so choose them wisely.

Product manager

IT organizations often do not have product management experience, so you may need to recruit an external-facing product manager from a business unit. We've found that successful product managers fulfill several of the following personas:

- **Alchemist** - Takes disparate requirements and distills them down to a succinct product vision
- **Visionary** - Doesn't allow themselves to be constrained by legacy thinking and processes
- **Influencer** - Has strong relationships with business partners, application teams, and IT
- **Minimalist** - Understands the value of shipping early and often
- **Lean champion** - Relentlessly pursues the elimination of waste

For more details on what a product manager does day-to-day, see the "VMware Tanzu Labs Playbook."

Platform engineers

The platform engineering team requires a combination of infrastructure and software engineering skills. Building and then keeping your platform up to date in a sustainable way requires you to treat operations as a software problem, so platform engineers must think like software developers. In many cases, training will be needed to help platform engineers recruited from software teams deepen their infrastructure knowledge. The same is true for people recruited from operations, who often need to build their software engineering skills.

The launch platform team should include at least two of the following personas. As the team matures, it should contain all three personas:

- **Infrastructure architect who codes** - This person is very experienced in IaaS primitives (compute, storage, network), usually has experience in production operations, and is able to automate manual, repetitive activities.
- **Natural automator** - You'll often find this person doing continuous integration/continuous deployment work, automating your current release management processes, and otherwise living in the command line and sailing seas of yaml files.
- **Curious software engineer** - You'll find this person in an application product team that previously solved its own platform needs by automating infrastructure.

Pick a first production app

- Choose a single production application that all stakeholders agree will go into production on the new platform. The ideal launch app is not mission critical but has existing production users and is maintained by an app development team that can assist in moving the app to the platform product.
- Do not fall into the trap of choosing a mission-critical app to prove to detractors that "important" apps can immediately go into production on the new platform. Mission-critical apps can follow quickly once the platform team gains expertise by running the first real workloads in production.

Define a platform MVP with a brand

The platform product manager defines an MVP platform product with a self-service landing page and a product brand. An MVP is the simplest and most easily built version of the platform that can help validate or invalidate a product management hypothesis about user behavior or features.

VMware Tanzu customers who have successfully delivered platform products in large organizations define a recognizable, internal brand for the platform. A brand creates a sense of ownership and pride in the platform team and helps to build a customer base by raising awareness among app developers.

There is a lot more to say on this topic, which you can read in our three-part series on marketing platforms (one, two, and three). The AI Starter Kit for Tanzu Platform also contains a full paper on driving internal platform adoption, including how to market your platform to developers looking to deploy AI applications, or any type of application.

Define a new path to production

An effective platform allows application teams to release software faster, which stresses systems and processes that assume centralized control and throttle change in an attempt to achieve reliability. Do not shoehorn your platform into your existing path to production, or you'll make it easier for apps and features to pile up waiting to get to production on the platform.

Deploying the first production app on the platform will prompt important conversations among stakeholders about the processes in your current path to production and production operations (e.g., change management, compliance, release management, security, and testing). We recommend conducting sessions to create the customer journey maps and value stream maps that emerge as you introduce the platform.

The result should be a new and deliberately naive path to production that emphasizes velocity and is only suitable for a small proportion of your app portfolio. As you extend the platform, you'll discover new bottlenecks and challenges to tackle. Over time, the path to production is enhanced to meet the production needs of a larger proportion of your app portfolio, including things like artifact management, automated patching, and vulnerability scanning.

Create customer journey maps

A customer journey map is a compact visualization of an end-to-end customer experience. The platform product manager will generate journey maps of your app developers' current experience of going to production to understand where there are opportunities for improvement.

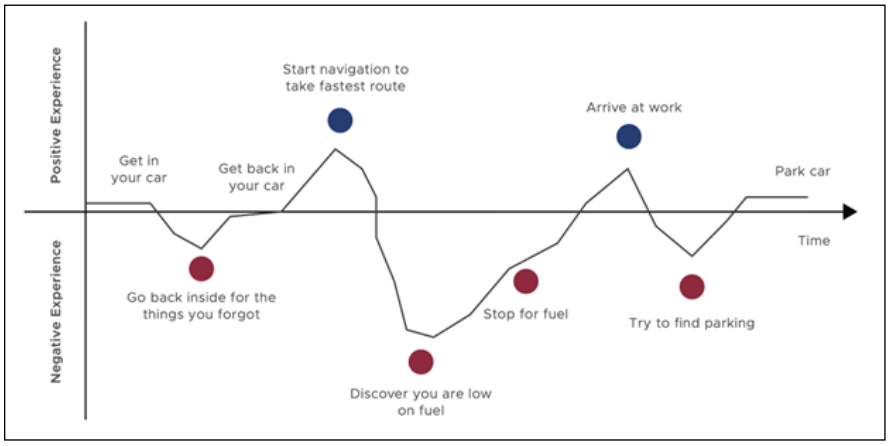

Figure 2: A drive-to-work journey map.

The product manager will also talk to teams currently running apps in production and generate journey maps of their experience of responding to incidents and performing root cause analyses. These maps will continue to help identify improvement opportunities for how teams run apps in production while increasing the velocity of change.

Once you know the current path to production, compare the path to production to the naive requirements of the first production app. Since the first app is not mission critical, it may not need to adhere to the same requirements for compliance or security that are demanded by other apps. Does it require all steps in the process? What can you change or leave out? This exercise helps define the essential steps in the new, naive path to production. It also starts to uncover steps in the path to production that you may no longer need. This set of essential steps will grow as you onboard new applications with additional requirements.

Define value stream maps

A value stream map is a visualization of the sequence of activities an organization undertakes to deliver on a customer request. In the case of the path to production, that request is getting a feature or app into production. The time between a particular trigger and the value delivery prompted by that trigger is the lead time. Shortening the lead time by removing steps and reducing waste in the activities used to deliver value is an objective of value stream mapping. Although customer journey maps are qualitative, value stream maps are quantitative.

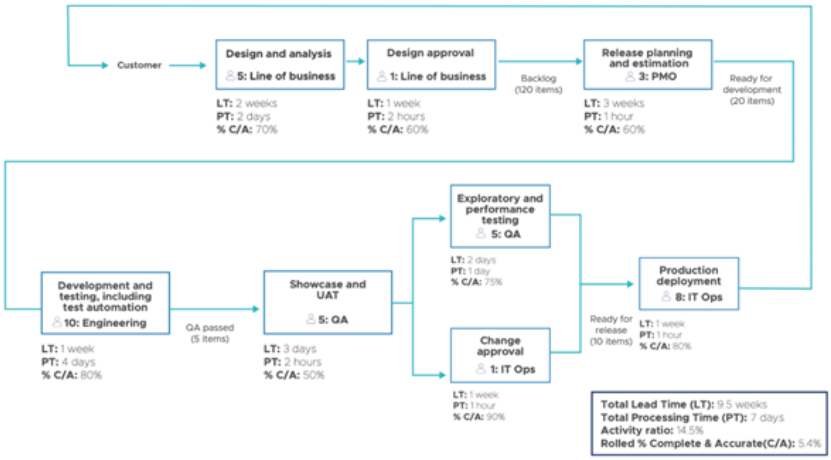

Figure 3: Software design and delivery value stream example.

The platform team will continuously evaluate the path to production and perform value stream mapping to find opportunities to remove waste and increase the rate of automation.

Deploy to production

Launch the platform MVP and deploy the first app to production using the new, naive path to production. This is success! It's also not the end of the process, but another step in learning what works by doing the work. You take feedback from this initial process and fold it into the next iteration, perhaps taking on two or three more apps.

Extend the platform product

In this phase, you'll extend the features of the platform so that more of your app portfolio can go to production while meeting security, compliance, auditing, financial reporting, and other enterprise requirements. You've discovered what it takes to get one or two apps to production, and now you can focus on automating many of the requirements.

The platform-as-a-product team usually has full responsibility for the actual platform.

In some cases, the platform team has full responsibility for the path to production: build pipelines, repositories, etc.

In other cases, that ownership is shared with software development lifecycle (SDLC) and tools teams.

Either way, we've found that the platform team has a very active role in the path to production.

Keeping those teams too separate can cause a lot of waste, friction, and silo-driven animosity.

Instead of standardizing on one build process, the tools team often writes a completely separate layer (or "wrapper") that layers on top of the platform's build process.

Iterate on features and reliability

The platform product manager will gather information from application and IT teams to determine what platform features are required to run additional apps from your portfolio in production on the new platform. Since the platform should create value for your business, the exact features will vary, but here are a few typical extensions based on what Tanzu Platform teams often implement:

- Additional service brokers to permit self-service access to application dependencies
- Additional buildpacks to allow new applications to run on the platform or to reduce the time required to modify an app to run on the platform
- Modifications to the onboarding process to add more self-service features
- Additional backing services such as Tanzu for Postgres, Tanzu for MySQL, Tanzu AI services, Tanzu RabbitMQ, or application modernization services like Application Advisor for VMware Tanzu.
- Solutions for cross-cutting concerns like security, logging, metrics, and load balancing
- Integrations with other enterprise systems
- Multiple variants of the platform that have different service-level objectives (SLOs) and characteristics
- Additional deployments of the platform in new regions and/or on new infrastructure

Iterate on the path to production

The platform team will iterate on the path to production to meet nonfunctional and compliance objectives for security, auditing, financial reporting, or other enterprise requirements, and make it possible to deploy a larger proportion of your app portfolio on the platform.

The platform team can use the 5 Whys or the Infinite Hows technique to determine the process objectives in your original path to production to achieve the same outcomes with less waste and more automation. The exercise should result in journey maps with improved sentiment (i.e., the emotional state of your app developers at each stage of the journey) and updated value stream maps with less waste.

Use the platform at scale

During this phase, you'll scale the platform and the platform team to serve thousands of applications while meeting all mandatory security, compliance, auditing, financial reporting, or other enterprise requirements.

Increase self-service

Self-service helps the platform team to scale sublinearly with the workloads running on the platform. At scale, app development teams are able to self-service their way to production, while individual developers are able to use the platform to learn and experiment with preproduction.

Self-service changes the communication pattern between the platform team and the app developers they serve. Instead of waiting for developers to send a request to IT, the platform team tracks self-service activities. The platform team is also proactive about reaching out to new and existing platform tenants whenever they see opportunities to educate, plan for the tenant's needs, or identify missing features in the platform's capability.

Scale the platform team

Two platform engineers are only sufficient while you launch the platform and have a small number of platform tenants. If your platform engineers spend more than 50% of their time on toil, or if you want to provide enhanced support for the platform such as 24x7 on-call support, then it's time to add more engineers.

When adopting a pre-engineered platform like Tanzu Platform, the platform team will scale sublinearly with platform workloads and generally max out at 9–10 people. A team of this size is capable of building and running a platform that can service more than 3,000 apps (See sidebar on team sizes we've seen over the years.). This ratio may seem ridiculous compared to what you're used to. But that is the point and power of using a pre-engineered platform, and those numbers have been proven out by many large organizations over the past decade.

We recommend creating a sustainable work environment where each engineer works a maximum of 40 hours a week and spends most of their day working from a prioritized backlog. Each engineer should only be on call for a maximum of one week per month, ideally with a primary and secondary engineer on call during each rotation. Providing 24x7 support coverage in that environment requires 6 to 8 engineers per site.

Over the years, we've collected several benchmarks for the size of platform teams. Here are some of them:[3]

- 350 apps supported by 7 platform engineers
- 300 apps supported by 8 platform engineers
- 30,000 developers supported by 50 platform engineers
- 6,500 developers supported by 16 platform engineers
- 2,500 developers supported by 5 platform engineers
- 1,200 developers supported by 6 platform engineers
- 45 application teams supported by 5 platform engineers
- 300 application teams supported by 4 platform engineers

3. Sources for platform team sizes: Kroger, GAIC, Mercedes-Benz, conversations with FSI platform teams; "Enterprise-Grade Platform Engineering at Charles Schwab," Coté, September 2024, based on Schwab's Explore 2024 panel; Rabobank conversations at Cloud Foundry Day EU 2025, Oct 7, 2025; "3 Cloud Foundry Stories," Coté, CF Day EU, Oct 7, 2025

Spreading those 6 to 8 platform engineers across multiple time zones in an effort to reduce "out of hours" work is the wrong approach. Maximizing time zone overlap between engineers is more important to ensure that context about platform changes is shared. Since platform incidents are almost always the result of planned changes, you can schedule when most of your incidents are likely to occur.

Add a developer enablement team

Adopting a new set of platform capabilities and transforming applications to run on the platform can be challenging for app development teams. A dedicated developer enablement team helps app developers improve their craft and establishes bidirectional communication between the platform team and lines of business. A developer enablement team can help transform your existing app portfolio and launch new products faster by advising other teams on how to meet business requirements using the features of the platform product.

Putting on quarterly internal conferences highlighting recent apps that have moved to your platform is vital as well. These events are for training, driving awareness, and sharing knowledge within your organization. Plus, they're fun!

For an additional view of what the platform journey looks like, check out the CNCF's platform engineering maturity model.

Tracking your success

Tanzu Platform customers often ask how they can tell if they're building an effective platform product. The following signals show that you're on the right path:

- Features are delivered to end-users faster
- Developers ask for more platform capabilities
- Deploying to production requires no special reviews, checks, balances, or ceremony
- Simply clicking "accept" triggers automated deployment, audit trails, and approvals
- Applications and their dependencies can be patched immediately in response to new common vulnerabilities and exposures (CVEs)
- Application outages are quickly traced to a root cause without a fire drill
- Events that previously generated downtime are autohealed without paging anybody
- Alerting "noise" is reduced, alerts become more actionable, and humans are only paged when there's a need for immediate action
- The platform team reaches a level of platform mastery where they provide new insights to their peers or even the IT industry

Using regular developer toil sentiment surveys is also very helpful to track your efforts over time.

If you're looking for concrete metrics to start with, use the five DORA metrics. These are widely known and understood. They, oddly, don't track actual business value delivered, but that is something you'll likely need to figure out yourself. There are numerous other metrics if you're looking for more.

Which platform?

The most strategic decision you can make about your platform is to buy it instead of building it. Reading between the lines, as the snarky CEO said above, there is no business value in building a platform. That does not help you sell more widgets or process more citizen requests.

From a business standpoint, a platform is non-differentiating. You don't want to spend your time and money hand-holding the development of a platform over one year, then three, then five, and then ten. As you can see, the practice of platform engineering is a lot of work on its own, and it's very valuable work. Your platform engineers do not have time to build a platform, and it is far from the most valuable use of their time.

As our customers have found over the years, Tanzu Platform is a great option for your platform stack. It will save you the time, money, and risk of building your own. For more on the build vs. buy discussion, see "The Upside-Down Economics of DIY PaaS."

Over the years we've learned a lot from seeing how our customers use Tanzu Platform as the center of their platform-as-a-product strategy. The day-to-day work of people in those organizations informed much of the advice given in this white paper. More importantly for those considering and planning out their platform engineering strategy, there's a rich history of lessons learned and proof that the platform-as-a-product approach works.

If you have questions or insights about starting up, or even maintaining your platform product strategy, contact us at tanzu.vmware.com/contact.

This paper was originally written by Joe Fitzgerald (now the CEO of MomentumAI, a platform engineering consultancy that is a Broadcom Expert Advantage Partner for Tanzu and VCF) and Colin Humphreys in 2017. Paula Kennedy's discussions over the years are always very valuable for understanding the platform as a product approach. This paper was revised in 2018 and 2021. The current version is from Michael Coté's tinkering in early 2026.

vmware® by Broadcom

Item No: Platform as a Product 6/26