WHITE PAPER
August 2025

# The Upside-Down Economics of DIY PaaS

## Seven pitfalls of building your own platform

vmware® by Broadcom

## Table of contents

## Focus on What Sets You Apart

Platforms are only valuable because of the applications they run. Following sound strategic principles, this means you should buy a platform instead of building one. Buying a platform frees you to focus your time, talent, and budget on what makes your organization distinct—and delivers the most value.

When it comes to applications and platforms, applications are clearly the most valuable of the two. They're how your organization engages customers, serves citizens, and empowers employees. They're how your business functions. The platform exists to support those applications, not to compete for attention or resources. You can see this clearly when you imagine a platform with no applications running on it: it delivers no value to the business. The platform only matters because of what runs on top of it.

Choosing whether to build your own platform or not is a classic strategic question. The cheat code to answer it goes back to 1817, when David Ricardo introduced the principle of comparative advantage: even if you're capable of doing everything well, you'll achieve better results by focusing on what you do best and partnering or trading for the rest. Ricardo's case study was that Portugal could produce both wine and cloth, but it specialized in wine and traded for cloth with Britain, because that focus maximized its returns.

This strategic principle appears repeatedly, from Ricardo to Michael Porter, to management consulting frameworks, and, most recently, in Simon Wardley's maps. The guidance is consistent: Direct your effort toward what you're best at and what makes your organization stand out, i.e., your differentiation. For most organizations, that's your applications. Your apps are what deliver business value. Unless your product is a platform, building and maintaining one isn't a strategic advantage, it's just infrastructure. Necessary, but undifferentiating.

## Platforms Don't Differentiate, Apps Do

When you build your own platform, your developers must focus on container orchestration, stitching together infrastructure layers, writing YAML files, debugging service integrations, and numerous other tasks that occur below the application. They're not improving the business by focusing on applications, they're rebuilding the plumbing.

DIY efforts divert your best people to infrastructure tasks that don't generate direct business value, and that drag will compound for years. Once you see it this way, it's hard to unsee. And yet, many organizations still fall into the trap of building their own platform.

That's where the real risk begins.

### What About Government?

This principle applies just as strongly in the public sector. While government agencies aren't competing for market share, they still face real pressure: limited resources, high expectations, and the mandate to deliver services that work for everyone. In that context, building your own platform isn't just a distraction, it's a drain.

The goal isn't profit, but effective, reliable delivery. Every hour your team spends wiring together infrastructure is time not spent improving the citizen experience, ensuring mission effectiveness, modernizing core services, or addressing policy outcomes and compliance. Just as in the private sector, the platform is essential—but it's not where your focus should be.

## The 7 DIY Pitfalls

So far, we've looked at why building your own platform does not align with established business philosophy and strategic thinking. This can be a bit too academic for those who are getting their hands dirty with building and running platforms. Let's examine the practical reasons why building your own platform is a bad strategy by considering seven common pitfalls that organizations often encounter.

Teams building their own platform often underestimate the scope, focusing on the needs of one app and one team instead of hundreds or thousands.

### 1. Building a platform for one app, not hundreds of apps

Most DIY efforts start by solving a narrow problem: How do we get this one app deployed? This typically involves setting up containers, scripting a CI/CD pipeline, adding Kubernetes, and integrating basic logging and monitoring. With a control plane project and a bit of YAML, it can feel like you're 90% done.

Except you're not. You've solved "day one" for a single app. What comes next—day two and beyond—is where the real complexity lives. You now need backup and restore, patch management, observability, service discovery, RBAC, auditing, multi-tenancy, platform upgrades, vulnerability scanning, high availability, and eventually, things like multi-region deployment or sovereign cloud support. Everything needs to be consistent, secure, and usable across teams.

And this isn't just for one app; it's for hundreds, maybe thousands. A platform only delivers value when it supports many applications and teams. That means supporting diverse architectures, languages, services, and deployment models. It also involves integrating with infrastructure, production controls, compliance tooling, and more.

The CNCF's platform reference architecture, shown below, provides a sense of this full scope. Much of the discussion around "internal developer platforms" focuses on the top layer—interfaces. But building a real platform means delivering every box in that diagram. And each one adds time, effort, and headcount.

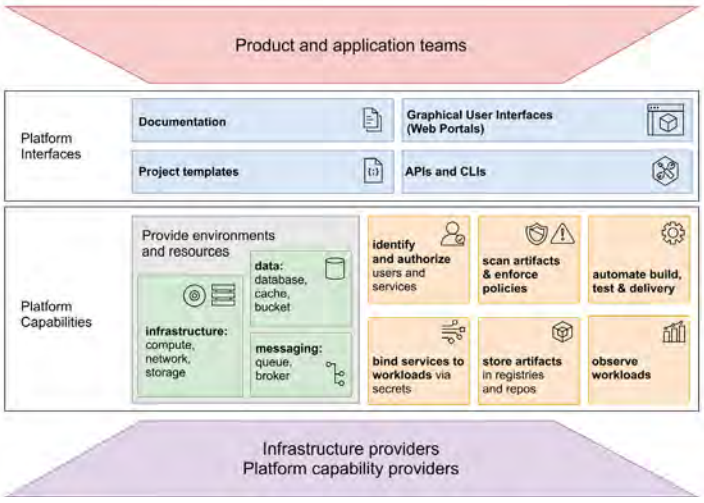


**Figure 1.** CNCF platform reference architecture

You're not just building tools. You're building a product—or more accurately, a suite of products.

To succeed, you'll need more than engineers. You'll need product managers, UX designers, back-end developers, front-end developers—and you'll need them continuously, not just for the initial build. And once it's built, you still have to run it, support it, and train developers to use it. The idea that the same team can do all of this while also delivering features for their "real" job is a dangerous assumption.

## Operating at scale

Most DIY platforms begin small, with one team and a few workloads. At that scale, it's easy to think your handcrafted stack is "good enough." But as adoption grows, so does the complexity—and the cracks start to show.

If you plan to scale later, build for scale now. Especially in large organizations, the real payoff of a platform comes from standardization and shared infrastructure across many teams. Even if you start with a handful of apps, it's smarter to begin with a platform that's designed to support real growth.

Your DIY platform doesn't just need code. It needs teams, processes, and years of full-time effort.

## 2. Underestimating the ongoing investment

The unspoken assumptions behind most DIY efforts are simple and seductive: we'll just download open source components, hook them together, and build our own platform in our spare time. How hard can it be?

Very hard. Very expensive.

As covered in the first pitfall, a real platform is not a toolchain or just an integrated stack of infrastructure residing behind a self-service portal. Building and maintaining an enterprise platform requires several teams responsible for reliability, usability, velocity, and roadmap. According to the CNCF maturity model, this means product managers, user research, documentation, onboarding, SLAs, and constant iteration.

It's not uncommon for these teams to add up to 50 or 60 people. It often takes them two years to build a DIY platform that comes close to matching the stability, security, and self-service of a commercial alternative. That's about $7 million a year in payroll alone. The cumulative expenses in headcount alone add up quickly over the years, as the table below shows:

| Seven Agile (Scrum) Product Teams (To Align With Reference Architecture Capability Domains) | Total of approximately 60 resources to build an enterprise cloud native platform team | Total Payroll Cost of DIY Platform Engineering Team |
|---|---|---|
| 1. Infrastructure<br>1. Operations<br>2. Deployment<br>3. Runtime & Middleware<br>4. Database<br>5. Security<br>6. Coaching & Developer Enablement (to train the developers on the platform) | • Each product team requires an average of 7-9 engineers per team (two-pizza teams)<br>• Scrum master shared across 2-3 teams<br>• Product owners shared across 2-3 teams | • 2 years<br>• 7 teams<br>• x8 people<br>• x $125k/year<br>• ~$7,000,000 per year in payroll |

**Figure 2.** DIY Platform Engineering Team Structure

| | Annual | Year 1 | Year 2 | Year 3 | Year 4 | Year 5 |
|---|---|---|---|---|---|---|
| **# of teams** | 7 | $7,000,000 | $7,000,000 | $7,000,000 | $7,000,000 | $7,000,000 |
| **Scrum Masters** | 2 | $250,000 | $250,000 | $250,000 | $250,000 | $250,000 |
| **Product Owners** | 2 | $250,000 | $250,000 | $250,000 | $250,000 | $250,000 |
| | **Total/year** | $7,500,000 | $7,500,000 | $7,500,000 | $7,500,000 | $7,500,000 |
| | **Cumulative** | $7,500,000 | $15,000,000 | $22,500,000 | $30,000,000 | $37,500,000 |
| **Headcount** | | 60 | 60 | 60 | 60 | 60 |

**Figure 3.** Cumulative spend over 5 year period for DIY platform

It's important to look at the cumulative spend over the years, as the above does. This is because you need to continually add new features to the platform, in addition to basic bug fixing, maintenance, and security patches. You can't reassign the platform builders to app development without the platform decaying underneath you. That team, dozens of engineers, plus scrum masters (or whatever similar role you might have), product managers, and developer enablement staff, have to stay in place.

Building and fully owning your own platform demands significantly more staffing than simply running, maintaining, and evolving a platform that has been built and supported by a vendor, especially when that vendor is backed by a broad ecosystem of partners and open source communities. The burden of integration, support, evolution, and documentation all falls on your team.

In contrast, teams that purchase a platform, like VMware Tanzu Platform, can operate the platform with anywhere from 4 to 10 platform engineers on average. This applies to *all* layers of the platform, covering the entire platform. This is because they rely on the vendor to keep the platform up to date and to add new features. Considering ROI in terms of the minimal number of personnel required to support a large number of applications and developers is often more revealing than modeling costs as done above.

When you build your own platform, you're not saving money, you're just swapping software costs for people costs. And often, you end up spending more time and money to produce a platform that's less capable, less innovative, and slower to evolve than what you could have bought.

## Tanzu Platform Staffing

Here are some samples of the number of operators needed to run the Tanzu Platform:

- 350 apps supported by 7 platform engineers
- 300 apps supported by 8 platform engineers
- 1,200 developers supported by 6 platform engineers
- 2,500 developers supported by 5 platform engineers
- 45 app teams supported by 1 ops team

Sources: Broadcom internal analysis of customer data from retail and manufacturing enterprises based in North America and Europe, June 2025.

A real platform isn't a project - it's a product. And products evolve over a long lifetime.

### 3. Assuming the platform is ever "done"

Your company has invested in building a custom platform. It hits version 1.0, and the plan is to turn it over to a support team while reassigning the original engineers to other priorities. The assumption? It's "done."

If your platform is truly a product, then it's never done, only shipped.

Every part of your platform will need to change: APIs, integrations, compliance features, data layers, security patches, telemetry dashboards, and developer tooling. Each product team inside the platform organization owns a piece of this and is responsible for:

- Continuously developing and evolving their services to meet user needs
- Testing, integrating, and delivering updates using CI/CD practices
- Monitoring and reporting on performance, uptime, and business outcomes
- Adapting to support new runtime stacks, languages, and AI workloads
- Staying ahead of security, audit, and policy requirements
- Coordinating with developer experience teams to ensure usability and onboarding

You don't need a support desk. You need an ongoing product organization.

Very few companies can justify the long-term investment to maintain this pace internally. The innovation curve from commercial vendors outstrips most DIY efforts within 12–18 months. Meanwhile, the internal platform team gets bogged down in operational fire drills and incremental maintenance. Over time, the velocity gap becomes a strategic liability.

Looking at the CNCF Platform Engineering Maturity Model, you can see that the responsibility for the platform's functionality grows as more and more apps run on the platform. This drives more dedicated platform product managers to plan, prioritize, and evolve those capabilities based on real user needs.

Even if you buy a platform, you still need product management. But the job is smaller, clearer, and more focused on integration and developer experience. In a DIY scenario, that product manager is overwhelmed, forced to act as an orchestrator across multiple custom components, integrations, and internal infrastructure teams. And not just one product manager: most DIY efforts require several product managers just to keep the lights on. You've traded vendor complexity for internal coordination overhead, and that scales poorly.

DIY platforms don't eliminate lock-in, they internalize it.

### 4. Frozen in place by snowflakes

Concerns about lock-in often sound like this: "We must avoid lock-in to maintain leverage over our suppliers. Open source means we can control our fate, avoid vendor pressure, and switch infrastructure providers whenever we need to. No one will hold us hostage. And it'll probably be cheaper too.

Yes, open source can help avoid vendor lock-in. But stitching those open source components into a homegrown platform often creates something worse: internal lock-in.

Your organization ends up with a unique, one-off snowflake—a platform that only exists inside your company. It's built differently, behaves differently, and must be learned from scratch by every new platform engineer, operations person, developer, product manager, tester, and so on. Documentation is weak, tribal knowledge is strong, and onboarding becomes expensive and slow. Worse, if key individuals leave, you risk losing critical operational and architectural understanding.

This isn't just inconvenient, it's dangerous. You've locked yourself into a platform of one.

A commercial platform backed by an open source foundation offers a better balance. You still avoid hard vendor lock-in, but you gain shared understanding, documentation, and institutional knowledge. With a commercial product:

- Engineering is consistent and coherent; the parts are designed to work together.
- Knowledge is open and accessible through docs, wikis, and open source repos.
- Expertise is widely distributed, across customers, contributors, and partners.
- Value-added components, training, and enterprise support are optional but available.

You're still free to run on any major public or private cloud. That's because the best commercial platforms are designed for portability—not just across clouds, but across teams.

True portability means standardizing how apps are deployed and operated, regardless of the underlying infrastructure. Too much flexibility creates chaos. Too little creates rigidity. A well-designed platform lets you hit the right balance: enough consistency to scale, enough flexibility to adapt.

Instead of worrying about lock-in, focus on "the freedom to leave." This is another way of saying "application portability" i.e., how easily could you migrate off the platform if needed? When you reframe "lock-in avoidance" as a question of real-world portability, you can think about it more analytically.

The "freedom to leave" framing allows you to evaluate portability alongside other trade-offs: speed, cost, security, maintainability, and ecosystem maturity.

You may decide that full portability isn't worth the trade-offs it demands—like using a general-purpose framework instead of one optimized for your use case, or that some vendor alignment is a fair trade for a better developer experience that only they provide.

You want to make a deliberate choice, not just react out of un-analyzed fear. More importantly, you're evaluating *all* the options with clarity. There's no rule that says *only* vendor-backed platforms lack portability. Any platform—whether bought or built—deserves scrutiny. Analyze portability on its own terms, not just as a reflex against vendor involvement.

Retaining staff skilled at building platforms is difficult and costly.

### 5. Retaining skilled people

Staffing quickly becomes a challenge. You'll need a rare mix of skills: infrastructure expertise (including cloud, networking, and virtualization), security and compliance knowledge, application development experience, and the ability to write solid system code. Few people have all of these. That means investing significant time in training and increasing your budget to recruit the right people.

Even if you succeed in building this talent pool, the clock starts ticking. Most organizations have a two- to three-year window before those skilled engineers start leaving. Other companies—especially vendors and cloud providers—are dealing with the same challenges and will be eager to hire from you. And they often offer higher compensation, better career growth, and more attractive brands than non-tech organizations.

You can try to mitigate this risk. But in our experience, staff retention consistently becomes a major strategic liability.

Security and compliance are one of the major requirements that separate enterprise tech from consumer tech—and they never stop moving, especially with a platform."

### 6. Keeping up with security and compliance

Staying compliant with standards and regulations is a constant game of Whac-a-Mole, especially for global organizations that must comply with regulations in multiple regions and industry groups.

When you build your own platform, that burden is yours. Your team is responsible for understanding every requirement, implementing the necessary controls, and maintaining compliance as regulations evolve. Add new features? That's more compliance work. Enter a new region? Even more.

Security is no different. Every new CVE (Common Vulnerability and Exposure) becomes your problem. Your team must track vulnerabilities across every layer of the stack, trace how those vulnerabilities manifest in your unique platform, and ship fixes quickly.

One area that often gets overlooked is the application framework layer—the libraries, SDKs, and services your developers depend on to build apps. These are constantly evolving and frequently affected by vulnerabilities. Commercial platforms typically monitor, patch, and roll out updates for these components automatically. But in a DIY setup, that responsibility shifts to you. Your team has to track vulnerabilities, test patches, and roll them out across all running environments.

And unlike with commercial platforms, there's no vendor pushing patches behind the scenes. Worse, your platform may introduce entirely new CVEs that no one else is watching.

By building your own platform, you're opting out of the collective vigilance of the tech industry. That choice comes with serious risk.

Some platform builders are driven to create value to their resumes over value to their organization."

## 7. Resume-driven development

Many of the pitfalls we've discussed so far touch on strategic choices and project management challenges. Resume-driven development (RDD), however, is a particularly insidious pitfall because it stems from within the organization, often a consequence of misaligned incentives.

In its innocent form, RDD occurs when your staff, fueled by genuine curiosity and a desire to stay cutting-edge, underestimate the true costs and complexities of building and maintaining a new platform using the latest technologies. They see a fascinating new tool and believe it's the perfect fit, even if it adds unnecessary burden. In its more cynical manifestation, RDD is less about genuine interest and more about self-serving ambition: builders advocating for a platform built with trendy technologies specifically to pad their resumes with highly sought-after experience.

Regardless of its flavor, RDD carries significant organizational risk. As one study on this phenomenon notes:

> Extensive RDD-based technology selection may therefore lead to complex or even unmaintainable software consisting of technologies which are not suitable for the requirements, which are unfamiliar to current or future employees, or which did not deliver on their promise and were discontinued.

Part of the blame for this incentives problem can be found in the hiring process itself. Knowing that technical talent is often drawn to novel challenges, those crafting job descriptions might, perhaps unknowingly, make roles sound more cutting-edge and trendy than they truly are. This inadvertently fuels a vicious cycle: prospective employees expect to use new technologies, they actively seek roles that promise this exposure, and once hired, they then push to build with these new technologies.

"Extensive RDD-based technology selection may therefore lead to complex or even unmaintainable software consisting of technologies which are not suitable for the requirements, which are unfamiliar to current or future employees, or which did not deliver on their promise and were discontinued."

"Résumé-Driven Development: A Definition and Empirical Characterization," Jonas Fritzsch, Marvin Wyrich, Justus Bogner, Stefan Wagner, January 2021. Survey conducted May to July, 2020 with 591 respondents, ~90% in Germany.

It's a natural inclination, of course. When you hire people to build systems, it should come as no surprise that they will, indeed, build systems. This tendency becomes especially salient when it comes to platforms. As new building blocks and architectural patterns emerge, these talented builders will be eager to learn their intricacies and integrate them into new platforms.

However, while there are perfectly valid reasons to adopt and even build with new technologies, the danger with RDD is that fundamental architectural and strategic decisions get made based on the individual career development of your staff, not on the long-term, strategic benefit to the organization. This inherent conflict of interest can be a costly blind spot.

When contemplating building your own platform, resume-driven development is a common pitfall that demands careful scrutiny. Be sure to look beyond the allure of the new and evaluate the true benefits of each option against the comprehensive costs and risks, many of which are outlined in the preceding sections.

For a deeper dive into the pitfalls of the DIY route, see Camille Crowell-Lee's article "Why Build GenAI Apps the Hard Way? Get an App Platform Instead!"

## Avoiding the AI Platform Treadmill

As we enter several years of building out generative AI-driven applications, it's important to consider AI platforms as well. The pitfalls are largely the same: building, maintaining, and constantly updating AI infrastructure takes time and money—resources that could be spent delivering actual applications.

For example, the kind of AI services and infrastructure needed to use AI in enterprise applications includes model gateways, prompt templates, embedding stores, vector databases, retrieval pipelines, fine-tuning workflows, policy enforcement, safety checks, output monitoring, and runtime observability. And those are just the AI components. You'll also need SDKs, APIs, and tools your developers can actually use when building their applications.

Then there's governance. GenAI systems introduce new risks around privacy, security, compliance, and unpredictable output. Monitoring for accuracy, bias, and harmful content isn't optional—and bolting those capabilities onto a bespoke system is a major ongoing investment.

All of this "AI middleware" must be integrated, scaled, secured, and maintained before a single line of app code gets written. As with platform engineering in general, building this yourself is rarely a sound strategic choice.

Additionally, because the GenAI landscape is evolving rapidly, the APIs and behaviors your team builds against today may be obsolete within months. New patterns and protocols, such as the Model Context Protocol, will continue to emerge and evolve. A DIY AI platform team has to keep pace with these new AI innovations—more work for your IT staff to take on instead of working on the applications.

This is the same story we've seen with DIY platforms.

Tanzu Platform treats AI services the same way it treats all platform services: They're built in, maintained, and kept up to date for you. You get the benefits of new innovations without the overhead of building or integrating them yourself. As new capabilities arrive, they're tightly connected to the platform your developers already use. In practice, this means teams can start building AI-powered features right away, without delays or learning yet another stack. The AI services and AI middleware your developers need is already there—in the platform you've chosen.

"What is the best use of my limited resources to help meet business goals?"

## Build Your Platform Strategy Around Business Value

By this point, hopefully you can see that buying a platform is a better option than building and maintaining on your own. The above can seem confrontational, maybe even insulting if you're thinking you're capable of doing it. Perhaps you are! However, start with the question "What is the best use of my limited resources to help meet business goals?" Even if you are incredibly capable and competent, that brilliance is likely best applied directly to those business goals. Remember the founding economic principle of comparative advantage: even if you're good at something, if you can make more money focusing your efforts on another line of business, do that if you want more profit. And, if you decide to build your own platform, enter it with a clear understanding of the pitfalls and risks above: Be sure to plan for how you will handle them.

Let's briefly look at the inverse, the benefits of buying a platform:

- **Faster time to value —** Developers start writing and shipping apps immediately, not waiting months (or years) for infrastructure to stabilize.
- **Lower ongoing cost —** You avoid building and retaining a large platform engineering organization. Your spending is predictable and efficient.
- **Predictable cost —** Your platform's costs are known: the license cost and the number of platform engineers needed. You avoid difficult and often wrong platform development cost estimates.
- **Security and compliance controls are built in —** You inherit a security posture that's actively maintained, tested, and patched by full-time teams, not your own.
- **Proven scalability —** Commercial platforms are already running thousands of workloads across enterprises and public sector organizations.
- **Portability and multi-cloud support —** Run your apps across clouds and regions without rebuilding pipelines and abstractions from scratch.
- **Reliable support and roadmap —** You gain a clear upgrade path, roadmap alignment, and a partner that's accountable when things go wrong.
- **Ecosystem leverage —** You benefit from the shared lessons, tools, and expertise of a global user community—not just your own internal team. For example, you can get advice from others who've been in a similar situation as yours.

"The most outstanding KPI we achieve is time to market. We can innovate faster, going from an idea to an application in production, with all the necessary tests and security checks, within one or two business days. VMware Tanzu Platform and Tanzu Spring solutions reduce the complexity and that is what gives confidence to our developers."

Jürgen Sußner, Principal Cloud Platform Engineer, DATEV EG

## Tanzu Platform: Deliver Apps Faster with a Trusted, End-to-End PaaS

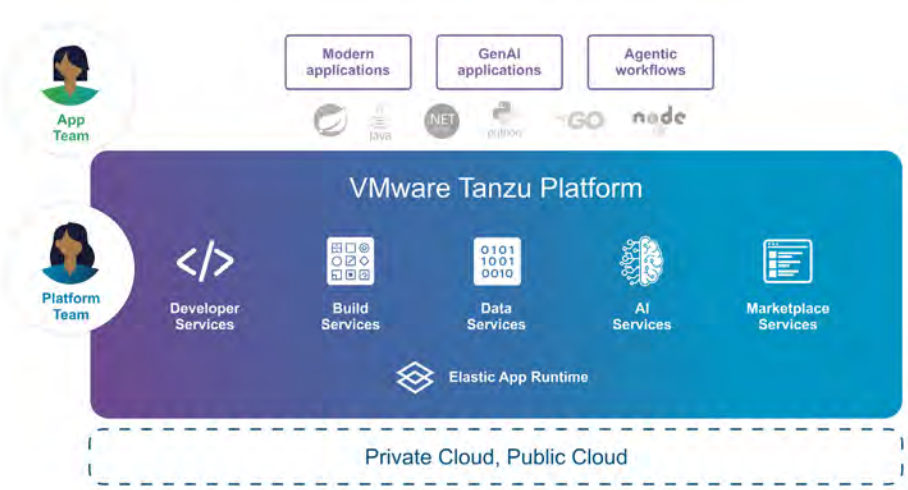


**Figure 4.** Tanzu Platform provides necessary services to deliver apps faster using a trusted PaaS

Tanzu Platform offers all of this. It's a proven platform, built on open technologies and hardened for enterprise and public sector use. For the past ten years, it's been used by many Global 2000 companies and government agencies, running thousands of applications and services around the world.

Developers love it. They can deploy and manage apps on day one, without worrying about YAML, networking, or orchestration. They get a secure, compliant, production-ready platform that just works, so they can focus on building the things that matter.

Even better, if you're working in a large organization, chances are you already have it at your disposal, installed, running, and paid for.

Want to learn more? You can check out Tanzu on the website, read this brief overview of the Tanzu Platform, and contact us: We're always happy to discuss Tanzu Platform.

Colophon

*This paper was originally written by Jared Ruckle, Bryan Friedman, and Matt Walburn. Michael Coté updated this paper in 2025.*

vmware® by Broadcom


Item No: The Upside Down Economics of DIY PaaS_v2 8/25