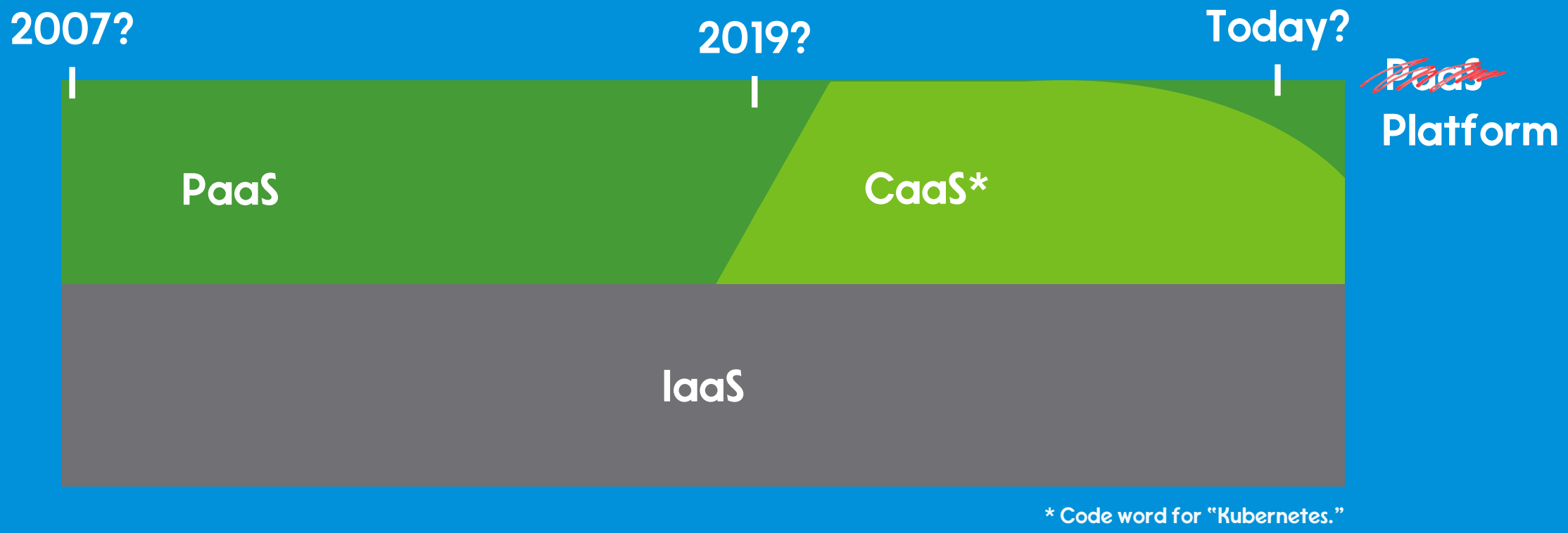
7 Ways to Fail at Building a Platform

Coté – PlatformCon 2026



* Code word for "Kubernetes."

- 350 apps / 7 ops
- 300 apps / 8 ops

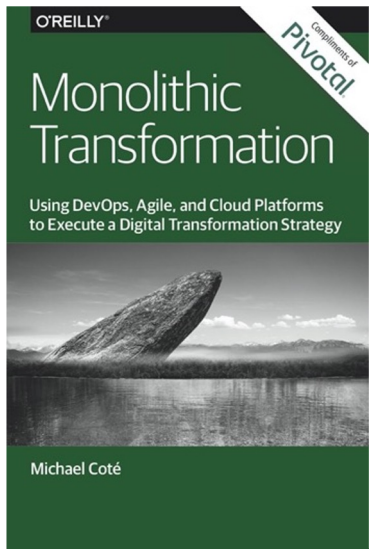
- 30,000 devs / 50 ops
- 6,500 devs/16 ops
- 2,500 devs / 5 ops
- 1,200 devs / 6 ops

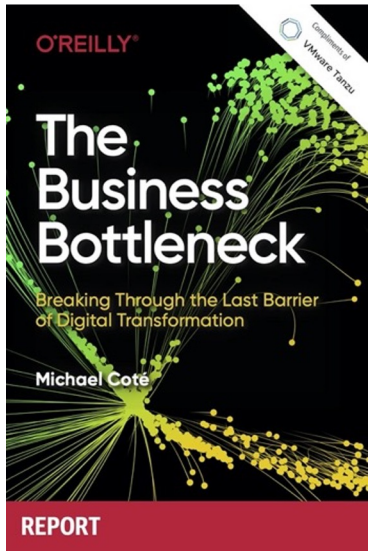
- 45 app teams / 5 ops
- 300 app teams/ 4 ops

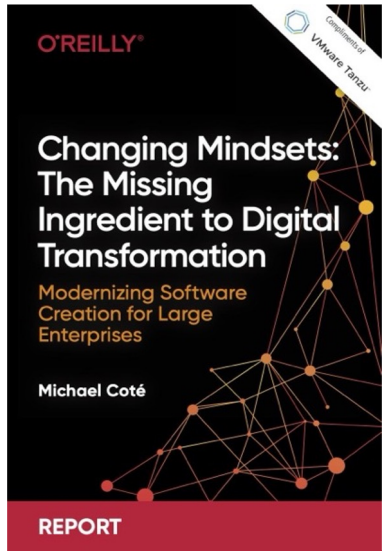
Sources: Kroger, GAIC, Mercedes-Benz, conversations with FSI platform engineers; "Enterprise Grade Platform Engineering at Charles Schwab," Coté, September, 2024, based on Schwab's Explore 2024 panel; Rabobank ops conversations, CF Day EU 2025, Oct 7th, 2025; "3 Cloud Foundry Stories," Coté, CF Day EU,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

Coté

https://www.cote.io/ | cote@broadcom.com







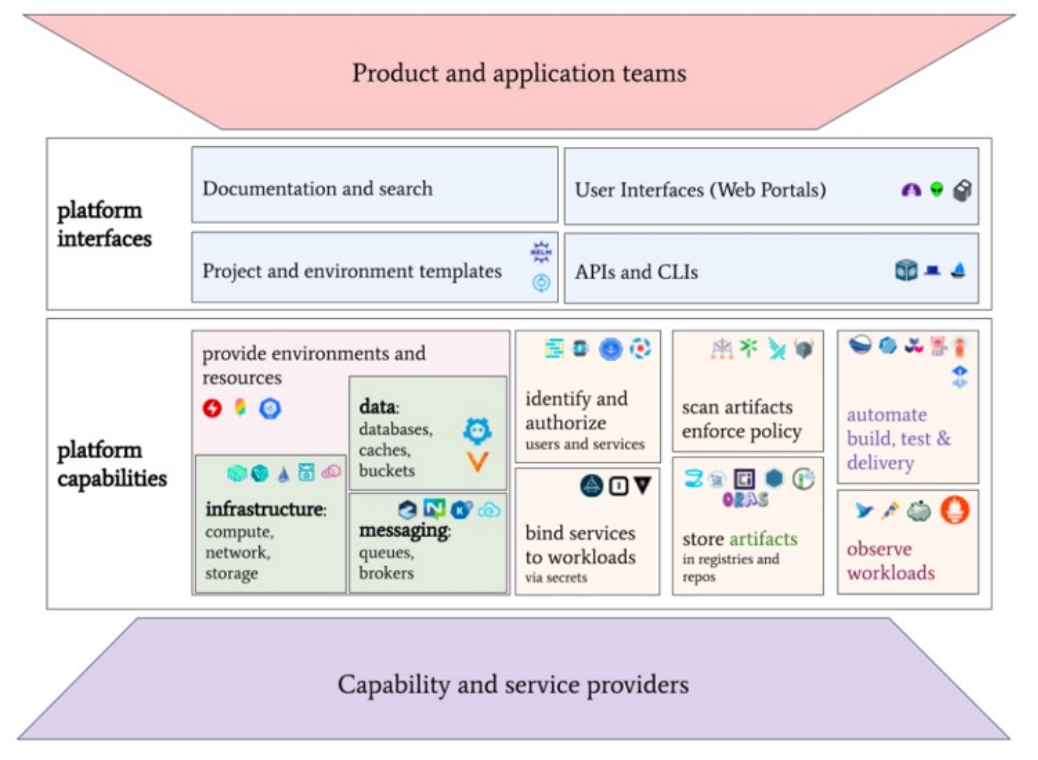
Tanzu Catsup

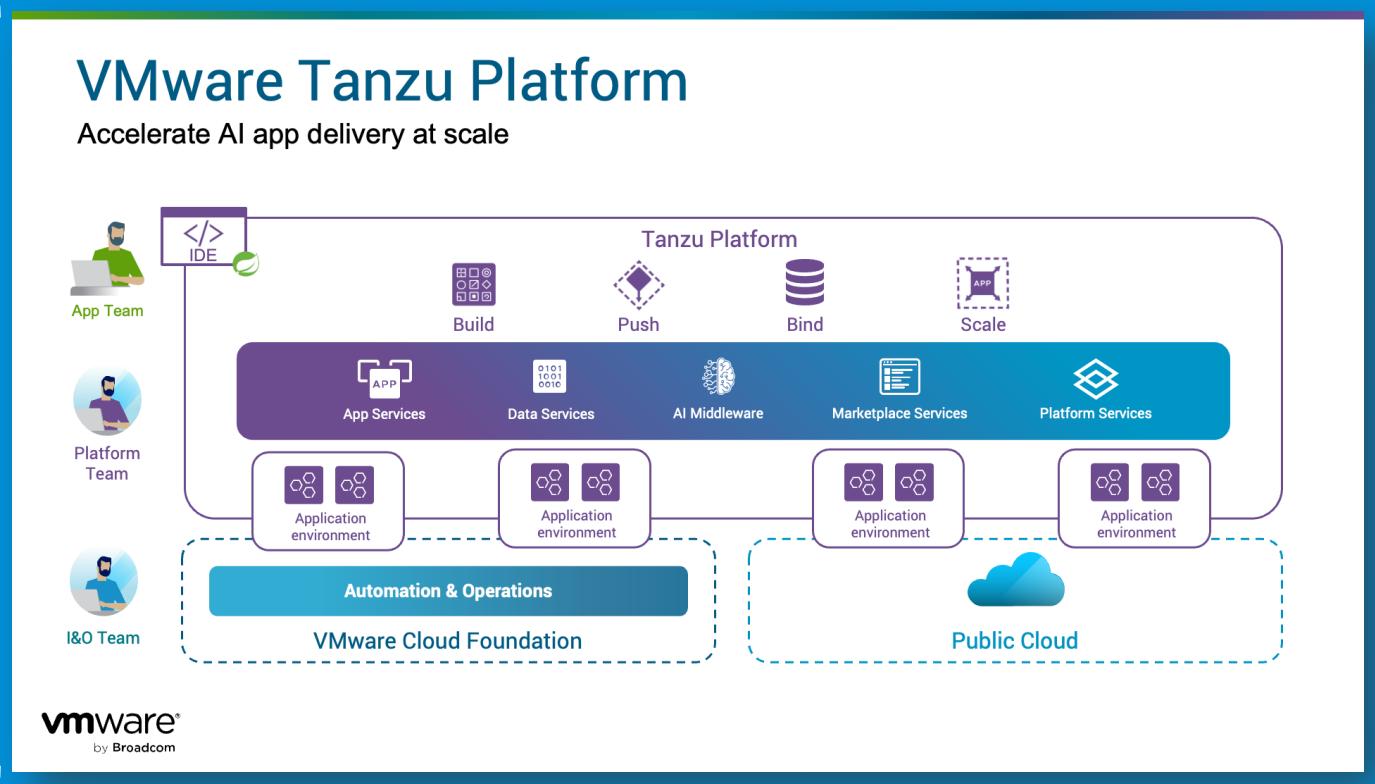
RedMonk DELL 451 Research S&P Global Market Intelligence Pivotal vmware Tanzu by Broadcom

#1
Unexpected scope creep.

What is a platform?

Centralized, standardized stack for building, running, and managing in-house apps.





Sources: "CNCF Platforms White Paper," March 2023; VMware Tanzu.

More than namespaces, yaml templates, & base container images

- App delivery.
- Backup and restore.
- Patch management.
- Observability, logs, monitoring.
- Service management & use.
- RBAC, etc.
- Vulnerability scanning.
- Dev framework integration.
- High availability & the other –ility's.
- Multi-region deployment.
- Sovereign cloud.
- Auditing and compliance.
- Multi-tenancy.
- Upgrading the platform.
- Gateways, brokers, load balancers, etc.
- CI/CD, itself or integration.

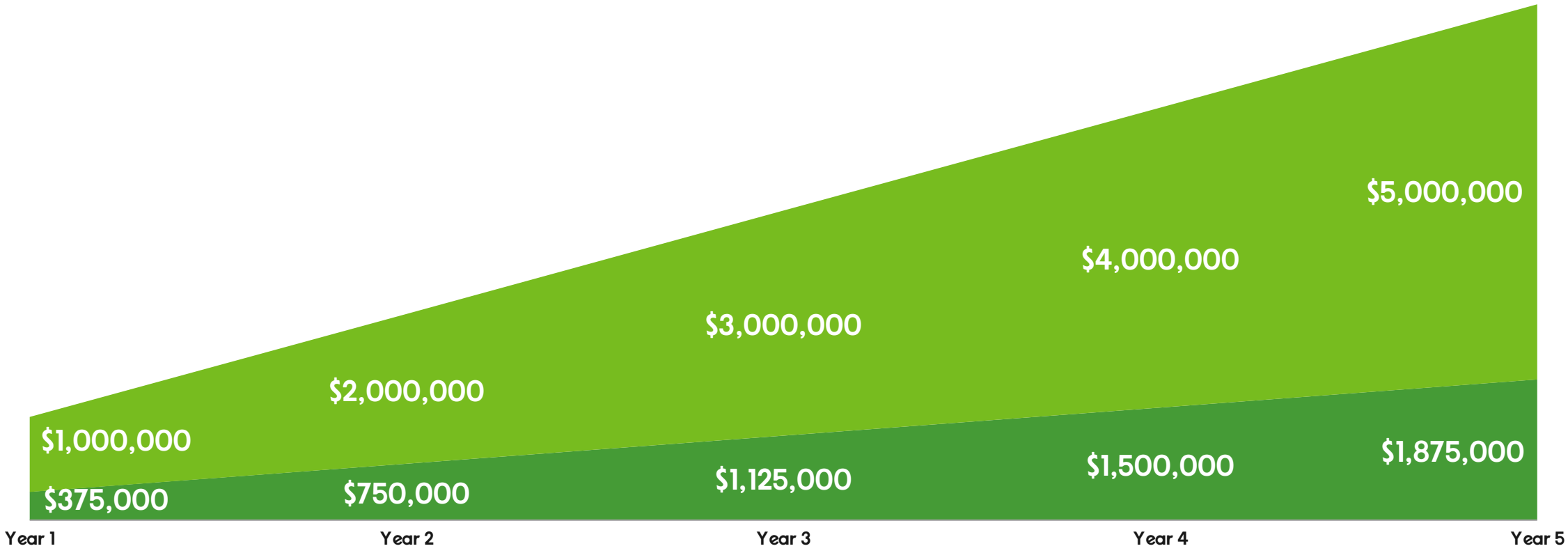
See also "Stop Renting Your Knowledge," Cur8s, Adib Saikali, Winter, 2025.

	Level 1 – Build	Level 2 – Operate	Level 3 – Scale	Level 4 – Improve	Level 5 – Adapt
Primary focus	First cloud native workloads	Production + standardization	Org-wide repeatability	Governance + security by default	Continuous optimization
Platform & infrastructure	Initial Kubernetes clusters Basic container registry Infrastructure as Code with Terraform	GitOps for apps via Argo CD or Flux Thin internal developer platform (templates, self-service) Centralized environments	Multiple standardized clusters (multi-region / multi-env) Formal Internal Developer Platform (portals, templates, APIs) GitOps extended to platform components	Drift detection + auto-remediation Zero-trust platform access Security posture management integrated with ops	Policy-driven workload placement (cloud/on-prem/edge) Platform continuously reshaped based on usage data
Application delivery	Helm or raw manifests via Helm Basic CI pipelines 1–2 pilot apps	Helm + Kustomize at scale Runtime config via ConfigMaps/Secrets Standard base images + scanning + SBOMs	Artifact signing + automated promotion pipelines Namespaces-as-a-Service Horizontal + event-driven autoscaling from app metrics	End-to-end supply chain security (signed images, enforced SBOMs) Admission controls everywhere	Progressive delivery (canaries, feature flags everywhere) Predictive scaling from production signals
Observability & operations	Minimal logging/metrics (often cloud defaults + early Prometheus)	Metrics + logs + early tracing using OpenTelemetry Central log aggregation	Distributed tracing as first-class signal Automated backup, DR, cluster lifecycle	Central audit pipelines (SCM, CI, clusters, apps) Runtime policy enforcement	Automated performance/reliability feedback loops Anomaly detection from live telemetry
Networking & security	Manual secrets Basic perimeter security	Admission controls Container/runtime scanning Early service mesh (often built on Envoy)	Operational service mesh (mTLS, retries, traffic shaping) Multi-tenancy with quotas	Workload identity + automated cert rotation Fine-grained authZ via mesh Zero-trust networking	AI-assisted remediation Continuous security optimization
Cost & optimization	Mostly manual cost awareness	Initial resource limits Early FinOps signals (namespace quotas, requests/limits)	Chargeback/showback per team or namespace	Integrated FinOps dashboards + budget controls	FinOps directly drives autoscaling and scheduling Continuous cost optimization
AI (where applicable)	Mostly experimental	First production models Basic observability + access controls			

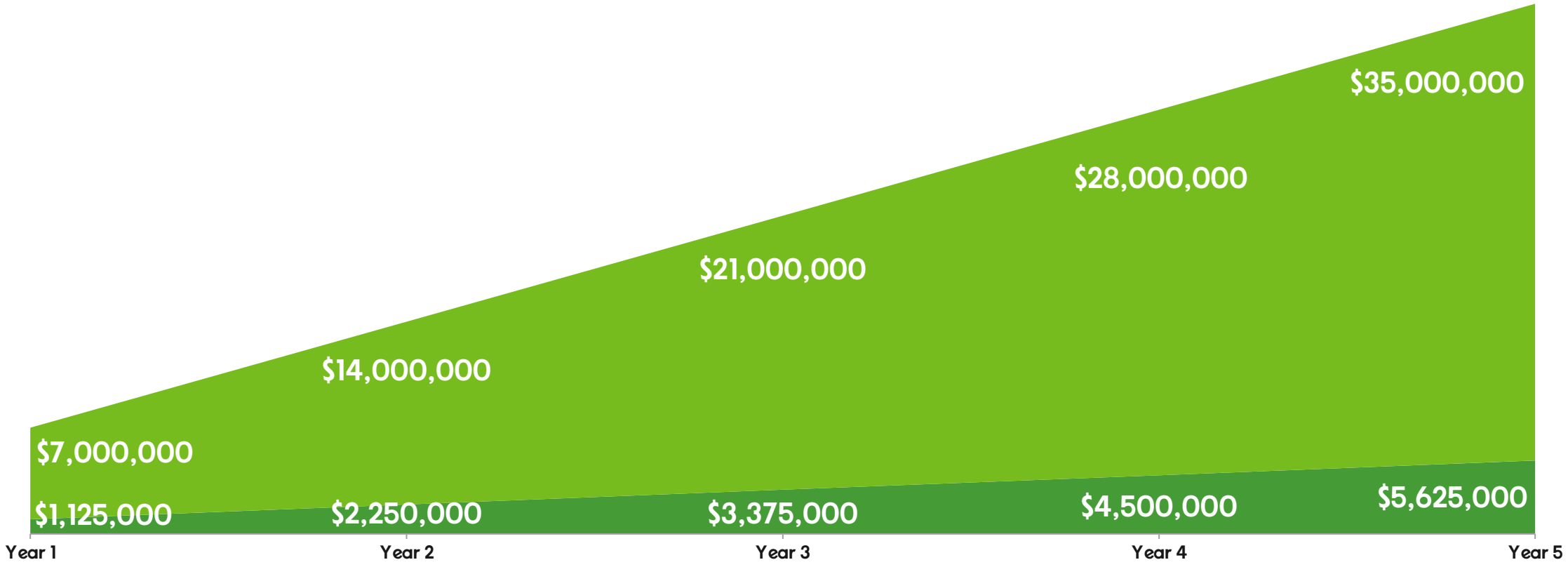
Source: beta version of Cloud Native Maturity Model: Core content updates for v4.0 (#84). Summarized by ChatGPT 5.2 on February 2nd, 2026. "Platform Engineering Maturity Model," CNCF Platforms Working Group, October, 2023.

#2
Underestimating the ongoing investment

Cumulative
Platform Salary Spend
One team of 3 to 8 people, annually



Cumulative
Platform Salary Spend
3 to 8 teams, annually



#3
Platform as a project

“We are building this platform not for us, we are building it for Mercedes-Benz developers.”

Thomas Müller, Mercedes-Benz

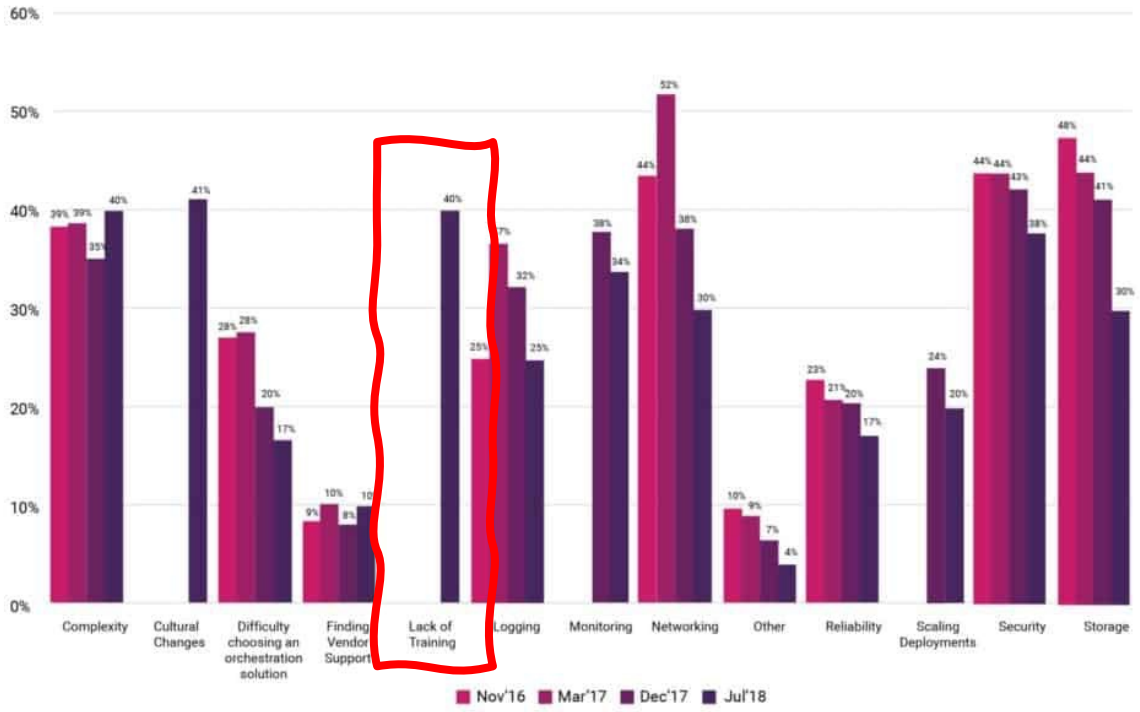
#4
Homegrown Lock-in

- The Freedom to Leave.
- Portability.
- Switching costs.

Sources: "Freedom To Leave," Simon Phipps, June, 2006; "Switching Costs and Lock-In," Mark Schwartz, December, 2018; "Thinking About VMware Alternatives?" Keith Townsend, August, 2025. See also "Don't get locked up into avoiding lock-in," Gregor Hohpe, September, 2019.

#5
Retaining skilled people

Lack of training, 2017 to 2025

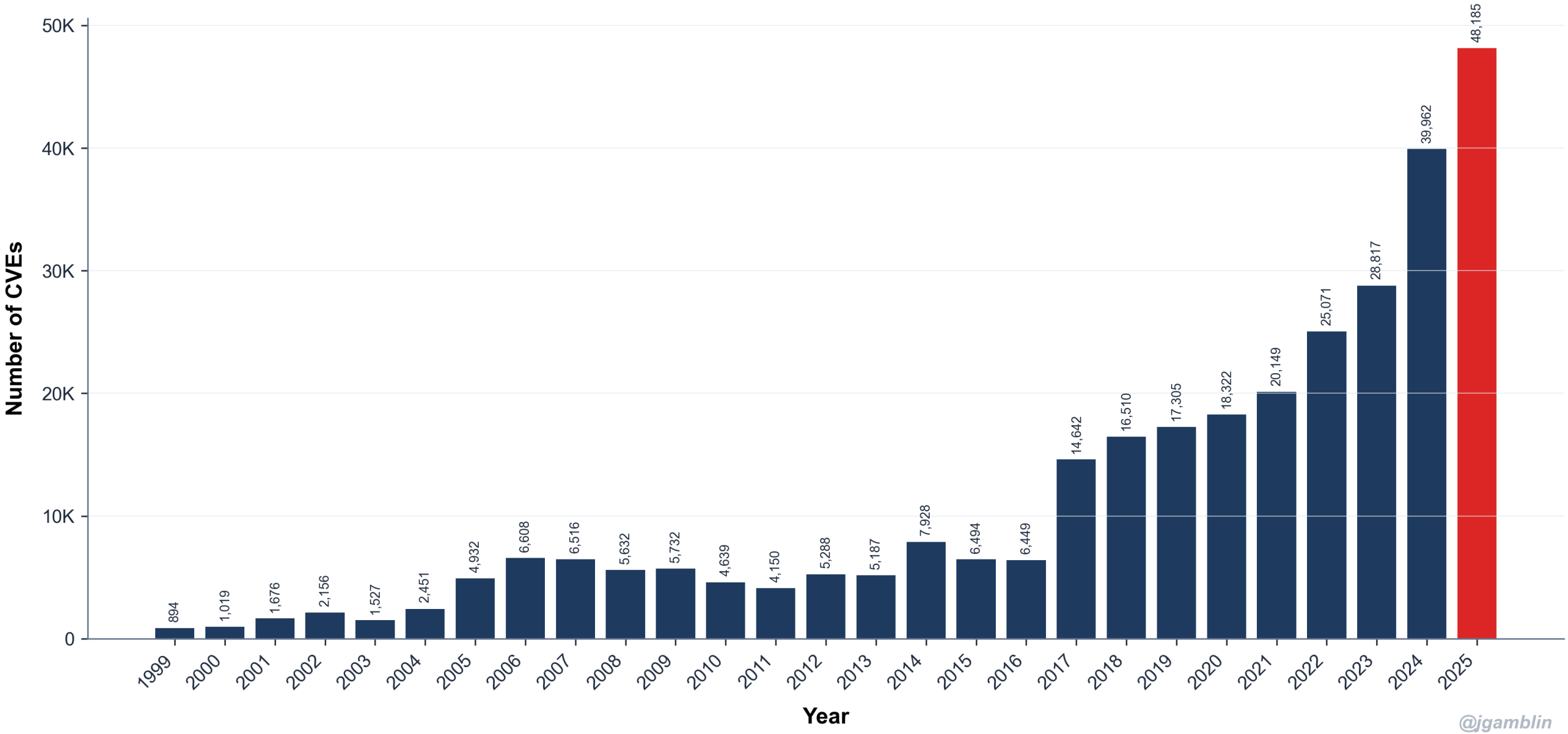


40% in 2017

36% in 2017

Sources: CNCF Survey, August, 2018; CNCF Survey, January, 2026 (for 2025).

#6
Keeping up with security & compliance



Source: "2025 CVE Data Review," Jerry Gamblin, January 1st, 2026.

Source: AI generated Trump Cloud; "The Cloud Man," found on Whoopah.

#7
Resume-driven development

Résumé-Driven Development: A Definition and Empirical Characterization

Jonas Fritzsch, Marvin Wyrich, Justus Bogner, Stefan Wagner
University of Stuttgart, Germany, Institute of Software Engineering
{firstname.lastname}@iste.uni-stuttgart.de

***Abstract*—Technologies play an important role in the hiring process for software professionals. Within this process, several studies revealed misconceptions and bad practices which lead to suboptimal recruitment experiences. In the same context, grey literature anecdotally coined the term *Résumé-Driven Development* (RDD), a phenomenon describing the overemphasis of trending technologies in both job offerings and resumes as an interaction between employers and applicants. While RDD has been sporadically mentioned in books and online discussions, there are so far no scientific studies on the topic, despite its potential negative consequences. We therefore empirically investigated this phenomenon by surveying 591 software professionals in both hiring (130) and technical (558) roles and identified RDD facets in substantial parts of our sample: 60% of our hiring professionals agreed that trends influence their job offerings, while 82% of our software professionals believed that using trending technologies in their daily work makes them more attractive for prospective employers. Grounded in the survey results, we conceptualize a theory to frame and explain Résumé-Driven Development. Finally, we discuss influencing factors and consequences and propose a definition of the term. Our contribution provides a foundation for future research and raises awareness for a potentially systemic trend that may broadly affect the software industry.**

***Index Terms*—software development, technology, hiring, career**

dents [...] learn a new technology at least every few months or once a year" [4]. The survey also revealed that technologies are regarded as the most important job factor by software professionals. This focus may come at the expense of other skills, leading to *"insufficient development competences"* [2] as Ebert and Counsell state it.

In the context of the software professional recruiting process, lively discussions have come up in developer forums, blogs, or social media where occasionally the terms *Résumé-Driven Development* (RDD) [5], [6], [7] and similarly *CV-Driven Development* [8] were used. Classon associates this notion with selecting *"tech stacks, architecture, methodologies, and protocols based on what looks good on the resume"* [9], while Ford et al. describe it as a pitfall by architects becoming enamored in latest technologies [10]: *"utilizing every framework and library possible to tout that knowledge on a resume"*. The topic tends to provoke heated and even polemic debates, as e.g. visible in [11] or [12].

While the term RDD has been sporadically used in books and online discussions, we have not found any empirical investigation of the phenomenon nor a definition and theory

65.000 software developers found that *"around 75% of respon-*

are commonly associated exclusively with one perspective.

additional language or framework. Moreover, introducing new technologies implies a learning curve and may come with maturity issues that impact reliability. 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.

may impact maintainability: technologies need to be regularly updated, dependencies have to be managed, and knowledge sharing efforts among team members increase with every additional language or framework. Moreover, introducing new technologies implies a learning curve and may come with maturity issues that impact reliability. 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.

Second, RDD can lead to false expectations and disappointment in the recruiting process on both sides. In a recent HackerRank survey among 71,281 developers [32], the top answer for *"What turns developers off from employers?"* with an affirmation of more than two-thirds was *"not enough clarity on role or where I'll be placed"*. Similarly, Behroozi at al. [21] identified a number of suboptimal practices which may sabotage recruitment processes in a study of over 10,000 Glassdoor reviews. A prevalent theme among them were inadequately communicated criteria by HR. RDD-based hiring can lead to similar frustrations. A strong focus on technologies during hiring may also lead to the neglection of other important skills for creating high-quality software products in a team. This is reinforced by nine *hiring* professionals who provided free-text comments about applicant traits they value much more than experience with trending technology, e.g. soft skills like communication, self-motivation, the willingness to learn, being a cultural fit for the team, or an understanding of the fundamental principles behind technologies. Since high employee

minimal in the software engineering field. Lastly, we did not use the term RDD to avoid bias in participants.

A threat to conclusion validity could arise from the exploratory nature of our survey. As such, we did not have formal hypotheses, e.g. for testing if RDD exists, but relied on our interpretation of distributions. To minimize researcher bias, we discussed all important results between the first three authors until consensus was reached. A confounding factor for answers to the *applicant* perspective could be prescribed technology choices by employers, as reported by 43% of our participants. We may have missed other such factors for the phenomenon, which is also supported by the low degree of variance our regression models are able to explain. Moreover, while using linear regression instead of simple correlation analysis improved our understanding of relationships in our data, we still cannot make statements about causality.

For scientific SE surveys, we had a high number of 558 responses for the *applicant* and 130 for the *hiring* perspective, with diversity in work experience, company size, and domain. A threat to external validity, i.e. generalizability, may be that the majority of participants were located in Germany (~90%). Regional and cultural factors could influence the phenomenon and results could partially differ in a sample dominated by participants from, e.g., the US. Furthermore, our *applicant* sample contained 102 students (18%). While we think that the views of students are also relevant for the *applicant* perspective, they may differ from the opinions of professionals. Nonetheless, we still believe the fundamentals of our theory to be applicable to a broad range of software engineering contexts.

phenomenon by surveying 591 software professionals in both hiring (130) and technical (558) roles and identified RDD facets in substantial parts of our sample: 60% of our hiring professionals agreed that trends influence their job offerings, while 82% of our software professionals believed that using trending technologies in their daily work makes them more attractive for prospective employers. Grounded in the survey results, we conceptualize a theory to frame and explain Résumé-Driven Development.

Sources: "Resume-Driven Development: A Definition and Empirical Characterization," Jonas Fritzsch, Marvin Wyrich, Justus Bogner, Stefan Wagner, January, 2021.

“Don’t build something if you can buy it.”

Sarah Wells, formally at FT, August, 2024.

“Do not blindly start with Kubernetes. Seriously. If your application can get by with a simple PaaS or Serverless offering I’d consider that first. Even VMs make sense for most situations.”

Kelsey Hightower, Autum, 2025

“It’s not about rebuilding what we can purchase that is available on the market. It’s about making sure we spend our time building the things that are bespoke and important for our organization.

Abby Bangser, Syntaso, November 2025.

Stop building platforms.

Start building apps.

Thanks!

💾 https://cote.io/diy/

🏢 cote@broadcom.com