Platform Marketing in 2025

Coté, June, 2025.
*This is **a work in progress and incomplete**.*
Got feedback, your own stories, thoughts? I'd love to hear it (probably): cote@broadcom.com.

After spending the last decade studying enterprise platform initiatives, I keep encountering the same story: technically excellent platforms struggling with low developer adoption. The platform engineering teams are befuddled. The internal developer platform (IDP) they've built seems to match exactly what developers said they wanted. But as the months go by, those same developers aren't rushing to use the IDP. And now that we're starting to build platforms on top of Kubernetes installations, platform teams in large organizations are encountering this problem all over again. It's a recurring question I've heard several times over the past few months: how do we get developers to use this awesome platform we've spent so much time building? How do we get a return on our platform investment?

This repeated pattern points to three commonly missing elements in platform initiatives: product management, community building, and platform marketing. Let's focus on the last part—platform marketing. The first two are more familiar and trusted by technical teams, but marketing is often viewed with skepticism and misunderstanding by technologists. This is a missed opportunity because marketing can be understood and applied just like any other engineering discipline. Let's explore how to apply a systematic approach to marketing: driving awareness, understanding, and adoption of your internal developer platforms.

Side-note: There's a great paper from IT Revolution called "The Developer Platform" that discusses managing internal developer platforms in general. Part of that is a discussion on platform marketing. It's one of the few discussions of that topic that I've come across. I recommend reading it. Below, I'll mix insights from that paper with what the platform community has learned from IDP teams over the years.

Defining the customer

You can't have marketing without a customer. Good marketing strategies spend a lot of time defining the customer, or audience, for all the marketing activities. When it comes to internal developer platforms, your primary customer is right there in the title: developers.

Instead of just settling on "developers," it's good to narrow down as much as needed. First, these are likely *application* developers. They're not developers writing embedded systems, for example. They're probably not the developers creating the platforms in question, nor programming the custom development tools your application developers use.

Another developer aspect to narrow down to is which types of applications they work on and which parts of the business. Are we talking about developers in the trading desk part of a financial institution, or developers that work on the ecommerce front-end of a car manufacturer?

Are these Java developers that are adding AI functionality to existing applications, or developers exploring how to integrate Air Fryer interfaces with Apple Watches?

Many platform teams can cater to a wide range of developers - all of the ones uses as examples above. Early on your platform strategy, picking one or two types of developers can be handy to force you to focus on those developers and their needs. You learn what this platform stuff is all about by working with a handful of initial teams. Then, you take those skills and expand to other teams.

I'm focusing on platform *marketing* here, but a lot of this "who is the customer" thinking is also done by platform *product management*. That function and role is incredibly valuable, and, I think, what makes platform engineering different than more traditional IT service management and delivery. As you do more and more platform marketing, you'll find that it overlaps with product management a lot, and this is good!

Once you have your customer identified, you can then move on to the core parts of platform marketing.

Core marketing: messaging, positioning, and value props

There are three core parts for any marketing strategy: messaging, positioning, and value propositions. This is a good place to start for IDP marketing.

1. Platform messaging - what is it?

Messaging is how you communicate the value of your platform—the key points you want developers to understand and remember. Think of it as your elevator pitch, distilled into clear, memorable statements. Your messaging should connect platform capabilities to developer pain points and needs. Developers don't care about the platform itself—they care about how it helps them build their own software. When defining your platform, your messaging should start with how it benefits developers, not just a rundown of its features.

For example, lead with the benefit developers will gain, then specify which part of the platform delivers that benefit:

- **Deploy to production in under an hour** using the platform's automated CI/CD pipeline.
- **Zero environment setup time** thanks to the platform's infrastructure-as-code (IaC) templates and frameworks like Spring Boot.
- **Less waiting and fewer security review meetings** due to built-in security compliance and integrated vulnerability scanning.

Each message has two parts: the benefit and how the platform achieves it. When brevity is required, just state the benefit. Define the platform by what it does for developers, not just its technical specifications.

2. Platform positioning - what is it good for?

Positioning defines where your platform fits in your organization's technical landscape. It answers the crucial question: "when and why should developers choose this platform over other options?"

Oftentimes, platforms are positioned as the everything solution that solves all the problems and, thus, should be used for all applications. This might be technically right, but I think narrowing down to a set of smaller, specific positions is helpful at first.

Here's some example of how to position your platform:

- Your platform is good for cloud native applications, not just *any* type of applications.
- Your platform is a good destination for modernized applications. Many modernized applications target cloud native architectures, moving apps to containers and microservices architectures.
- Your platform is the best place to run Java applications, especially ones that use the Spring Framework.
- Your platform is a great place to develop and run AI-enabled applications.
- You could say that your platform is good for classic three tiered, web applications: something with a UI, a middleware and business logic layer, and then a database.
- Another position could be that your platform is good for highly regulated apps that need to run in air gapped environments.

You don't need to pick just one positioning for your platform. After all, platforms are usually general and intended to be used for many different types of applications. However, coming up with multiple positions like the above allows you to speak to specific teams making it easier for them to sort through all the options and figure out if your platform is the right fit for them.

3. Platform value propositions - what's in it for me?

Value propositions, often shorthanded as "value props," are the concrete, measurable benefits your platform delivers. They answer the developer's question: "what's in it for me?" with specific, provable outcomes.

Good platform value props focus on specific benefits, not abstract capabilities or business outcomes. For example:

- **Time and Tedium Savings** - Reduce deployment time from 2 days to 30 minutes. Eliminate 80% of security review meetings.
- **Toil Reduction** - Automate and handle infrastructure configuration, simple network routing and load balancing, all so you can write apps, not program yaml.
- **Developer Experience** - Self-service environment provisioning. No more ticket queues for infrastructure.
- **Frictionless Onboarding** - New developers can start contributing to production-ready code in hours, not weeks.
- **Quick Access to Services** - self-service, pre-approved access to databases and AI models. No tickets needed.
- **Built in Observability** - Automatic logging, tracing, and monitoring help debug and optimize applications faster. No need to build these systems into your app.

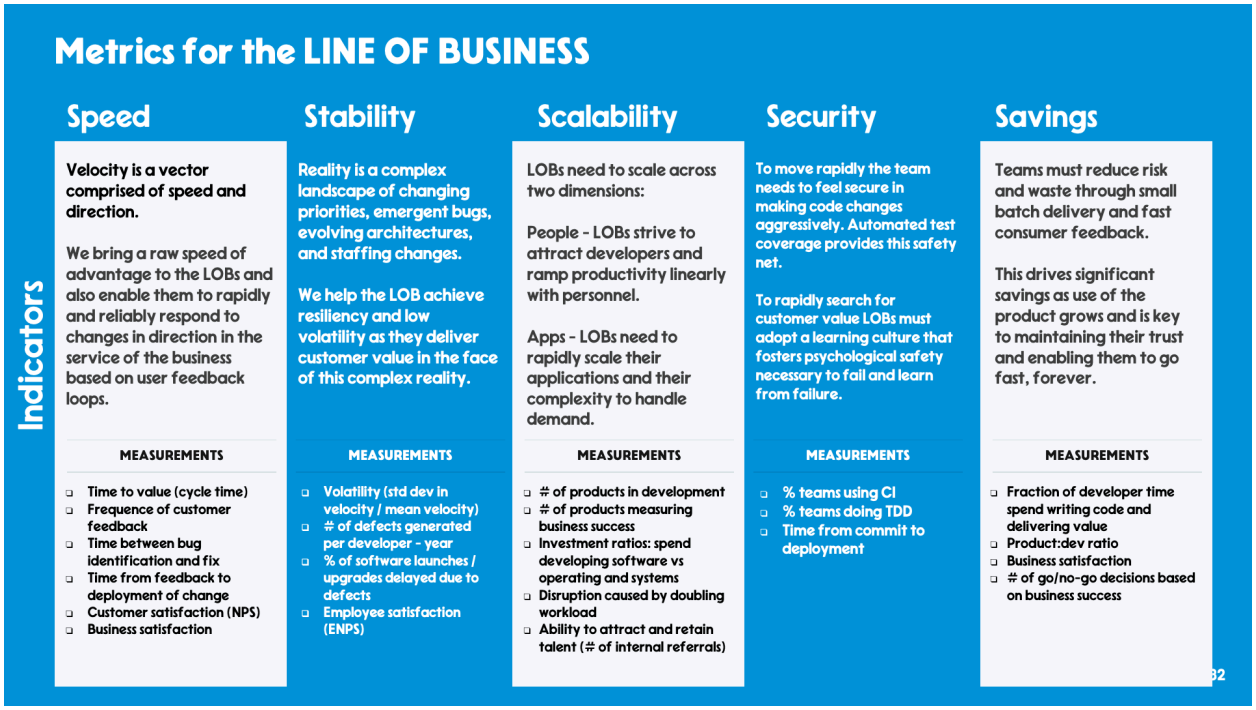
As you can see, developers like easy and speed. Even more so, they hate having to file tickets and waiting around for the ticket to be addressed.

Executives like business outcomes

The above examples are value props developers care about, but there's another audience that's important to market to as well: executives. Executives will want to know why they're supporting and paying for the platform. This will come up in annual budget planning, when a new executive takes over, and especially when it's time for any license renewals for platform components you use.

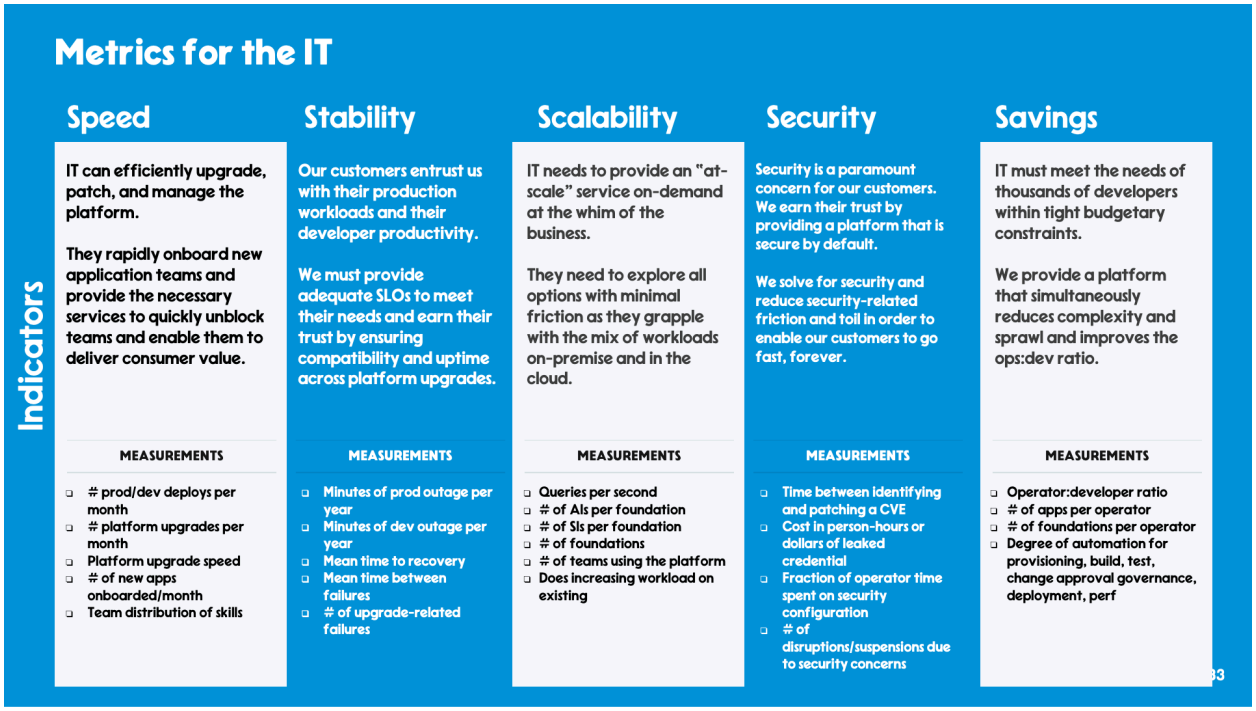
Most executives recognize the value of applications running on the platform based on business outcome: the business is either running well or not. However, drawing the connection between those outcomes and the platform itself isn't always obvious. It's similar to plumbing: you don't appreciate it when it works, but you sure notice when it doesn't.

To show the value of platforms to executives, over the years we've used the 5 S's set of metrics as value props with executives. Discussing these 5 S's is a whole topic on its own, but here's a quick overview of the clusters of metrics:



In these business-centric metrics, you can see how things the platform does are mapped to things the business needs to thrive. These metrics demonstrate the value of the platform to executives and help show them why the platform and the platform engineering team is contributing and worth supporting.

Of course, for inward looking metrics, you mighty tweak these slightly to be IT-focused:



Brand - *Who* is your platform?

"Do you have a T-shirt yet?" my colleague DaShaun Carter asks platform teams. That seemingly frivolous question actually cuts to the heart of platform marketing: brand identity. While engineers often dismiss branding as marketing fluff, the reality is that technical teams form deep attachments to their tools - just like sneakerheads obsessing over the latest Air Jordans.

Consider how developers self-identify: ask them what type of developer they are, and they'll immediately name their primary language. "I'm a Java developer" or "I'm a Python dev." Operations folks do the same - they're "VMware admins" or "Linux admins." This tribal identification with technology tools drives both adoption and persistence through the inevitable technical challenges.

This brand affinity matters even more for internal platforms since you can't piggyback on existing open source or vendor communities. When developers weave your platform into their professional identity, they'll not only stick with it but evangelize it to colleagues. That word-of-mouth marketing is pure gold.

Little wonder, then, that many of the successful platform teams I've talked with over the years put a lot of effort into branding their platforms. Organizations like the US Air Force, JP Morgan Chase, and Mercedes-Benz each create brands for their internal development platforms.

Symbols: names, logos and color schemes

From here.

The basic building blocks of brand - logos, slogans, color schemes, and names - deserve serious attention from day one. While your platform's visual identity should align with your organization's brand, you have room to be more playful and personable. Avoid dry, bureaucratic names like "Enterprise Developer Services" or "Internal Cloud Platform Architecture Cluster." Instead, give your platform a name you'd use affectionately. If you find people referring to your platform by its initials (EDS or ICPAC, with the above), you probably need a better brand name.

Look at successful examples: the US Air Force's "Kessel Run" captures their platform's maverick spirit, while JP Morgan Chase's "Gaia" suggests global enterprise scale. Your platform's name should signal its essence to developers.

Here's some ways to start brain-storm brand names:

1. What superpower does your platform give developers? – “Turbo” for speed, “Nimbus” for seamless cloud integration, “Vault” for security and compliance.
2. How would a developer casually mention your platform in conversation? – “I’m pushing the app to Shipyard” or “Let’s test it in Vault.” Names that feel awkward in these contexts will struggle to gain organic adoption.
3. What metaphors or symbols from your industry, company culture, or technology space could represent your platform’s purpose? – Consider using terminology that aligns with your industry. The US Air Force's Kessel Run is a great example, it evokes flight and coolness.

Once you settle on a brand, you should put the logo, name, and colors on all of your platforms UIs, documentations, maybe even command line tools. You're also going to need to create physical manifestations of your brand: stickers, t-shirts, and banners. This last part is a serious recommendation, done by most organizations I've talked with. It's why my friend DaShaun always asks about the platform's t-shirt first thing.

The platform's brand signals what your platform does for developers and what your platform is. Speaking of, let's look at defining your platform's ethos as part of its brand.

Case: platform branding in twelve parsecs

The US Air Force's Kessel Run platform project is a great example of platform branding in action. As part of an ongoing effort to modernize how the US Air Force built and ran software, project Kessel Run was setup to introduce agile software development and platforms into service branch. This was an imposing task, and required a little irrelevant maverickness, done, of course, with permission and intentionally.

What better way of embodying that ethos than to evoke Han Solo, the maverick pilo from *Star Wars*. In the movie, Han Solo boasts that he "made the Kessel Run in less than twelve parsecs," something that eve his trust co-pilot balks at. Thus, when the US Air Force was looking for a name that would bring that spirit of doing the impossible - and maybe a little bit of bravado - they named the project Kessel Run, complete with a logo that evoked that same ethos, using a silhouette of Han Solo's ship:

This attention to branding, slogans, and other marketing continued with slogans like ""Code. Deploy. Win." and, of course, t-shirts such as the cleverly done "AgileAF" which could stand for "Agile Air Force" or the popular slang-expansion of "AF":

Kessel Run has been a great success. Obviously, clever t-shirts and logos are a small part of that. But, they're an example of what I see consistently at organizations who put in place and maintain platforms: marketing is a *necessary* component.

Ethos: how your platform's beliefs drive what you get

More than just a name, the brand often comes with a set of principles and values, an ethos. Organizations often make little booklets of their platform ethos. At the very least, they write them up and include them in the documentation.

Here are some examples of technology brand and ethos:

- **Java:** Stability, portability, and backward compatibility. “Write Once, Run Anywhere” embodies the philosophy that code should be able to run across platforms without modification, prioritizing reliability and enterprise-grade robustness over cutting-edge innovation. This clear brand and ethos signals that Java is a perfect pick for programming enterprise applications.
- **Apple:** User experience, aesthetics, and tightly controlled integration. Apple optimizes for a seamless, intuitive experience, often at the expense of user-level customizability. Here, we have signaling that Apple is perfect for consumers, maybe not so much for "enterprise grade" needs.
- **Cloud Foundry**: Developer productivity, simplicity, and enterprise requirements for security and reliability. Cloud Foundry assumes that developers should be able to push code with minimal configuration, while the platform handles everything else—networking, scaling, load balancing, and service bindings. Developers and platform engineers using Cloud Foundry should not have to spend much time assembling or maintaining the platform; instead, they can focus on delivering applications efficiently.
- Kubernetes: infinite customizability with a toolbox mentality for building platforms. Kubernetes is built on the philosophy that developers and operators should have the ability to customize every detail of the platform and control over their applications and infrastructure, even if that means greater complexity and more low-level platform building.

For your platform, let’s look at some example principles that can form your platform ethos. These are five principles that I think map to most platform teams:

1. **Developer Experience First** - A fast, delightful way to get apps to production. Remove friction through self-service and automation, integration with developer tools and workflows
2. **Ship Now, Customize When Needed** - Production-ready from day one. Begin with enterprise-grade defaults, configure and extend as your needs evolve.
3. **Security and Compliance as a Feature** - Make the right path the easy path. Embed compliance and security into platform primitives. Replace approval meetings with automated policy enforcement.
4. **Observable By Default** - Fixable on day one. Built in logging, monitoring, and debugging from day one. When things break, provide clear paths to resolution.
5. **Reliable by Default** - Enterprise-grade from the start. Built-in stability at every layer and ready to scale.

Brand and ethos are two faces of the same thing: they define what your platform "stands for" and they give the people who use it (developers and operators) some shared identity. For example, developers who use the platform are known for getting ideas to production quickly but keeping their apps compliant and secure. Operations people who work on the platform are known for reliability in production while still catering to developer needs.

You should also think about how brand and ethos are part of the platform itself, a tool that helps guide and reinforce how it's used. For example, most platform branding I've encountered drives the idea that the platform is used for fast evolving applications, shipping apps frequently. The self-service (no tickets needed!) aspects of a platform embody this ethos. Providing default project templates and automating security and compliance checks also embody this ethos. You want the ethos you have to match up to thing your platform does for developers.

Case Study: Kessel Run

Let's look at a quick case study that brings a lot of this together.

Community management

Most platform teams I've talked with put a lot of time and thinking into the basics of marketing as discussed above. Well, most of the *successful* platform teams. The other thing these successful platform teams do is create and tend to the communities around their platform. Communities are a key part of your platform and of the broader goal: corporate success. New ways of working and the technologies that enable that way of working succeed in part through the strength of community. We've seen this over the past few decades, first in the web development developer

world, open source, DevOps, and now platform engineering itself. And I've seen this inside large organizations several times.

There's three parts to this:

(1) **Your community of support** - where do platform users congregate to ask questions, share knowledge, look up information and docs, troubleshoot, etc. Often, these communities are based on internal chat groups (like Slack, Google Chat, Teams, Matter, etc.) and forums for questions. Here, you also have good, searchable, frequently updated documentation and educational material like how to's and tutorials.

(2) **Events** - iInternal conferences and a speaker series keep developers engaged, informed, and connected to the platform. They provide structured opportunities to learn best practices, share experiences, and stay ahead of industry trends—all while strengthening the platform community. If you want people to actually use and advocate for the platform, you need more than good docs; you need a thriving, engaged developer ecosystem

(3) **Platform advocacy** - there's one aspect that is often overlooked, underbudgeted, and simply a huge unknown unknown for platform teams: the role of platform advocacy. As we'll discuss, this at least one full time person who focuses on just community management by working with, talking with, and otherwise "running around" with the developers using your platform.

Gardening the platform community

To get developers using your platform and get ongoing value out of it, you need to think about building and managing the community around your platform. The first community management you'll need to do is around how you provide support.

For most operations people, "support" is not a source of daily joy. "Let me greet the sun by logging into my helpdesk and finding gratefulness in the stack of tickets" is not a common refrain of people who work in IT. But, for as wonderfully automated as platform are, support is still required.

A platform can also surface new types of support that your comfortably small team cannot address. When you take care of infrastructure need and provide sensible defaults for how to package and manage applications, your customers - developers - will have more time to spend getting befuddled about problems higher up the stack. They may have more application development and dev tool questions. They might have questions about how to use and integrate services instead of how to configure them. If you're lucky, they'll start asking "how can we make this application more secure from the start?"

Your platform team may not be equipped to answer all of these questions. And while there's some early stories of generative AI helping, what I've seen is that platform teams rely on the internal developer community to help with this support.

One of the platform teams at Mercedes-Benz has used a familiar community-based support method. They used an internal chat application - MatterMost, but Slack, etc. could be used too - as one of their primary support channels. I've talked with other people about this approach which seems scary at first: instead of gating all those requests behind a helpdesk, now they're all there, all at once, every day. But, as more and more developers started using these platforms, fellow developer start pitching in. And, the sense I get from talking with people using this support by community-chat method is that it just feels better because it's more human.

In addition to support, there are some standard community management practices that platform teams find useful, if not necessary. Bryan Ross has a great overview of these practices, showing that community management is a "cornerstone of successful platform engineering teams". While I've distilled the essence here, you should read his full article:

In addition to thinking about support as part of community management, Bryan Ross goes over a comprehensive internal community management and communications in a recent article. I've summarized his recommendations below, but you should really read Bryan's whole piece:

1. Your product website is very important. Most internal web sites are...not good. Your internal platform will be competing with vendors and cloud services who are pouring resources into their web sites and community. As Bryan points out, getting a good platform web site isn't too difficult or costly.
2. A regularly updated team blog helps your community keep up with what's happening and get to know the platform and the people who run it. This is also a way to scale marketing in a classic one to many approach.
3. Reporting on platform metrics is another "face" of your platform. That "health dashboard" might seem dull and painful to operations people who stare at it all day, but developers using it will find it helpful. And, as Bryan points out: "Users are generally much more forgiving if they know what the problem is, that someone is looking at it, and they know when they can expect a further update."
4. It's easy to neglect documentation, but this is likely the place developers will spend most of their time. Keeping the docs about your platform up-to-date and useful makes a huge difference.
5. Email updates keep people informed but also keep your platform top-of-mind. A lot of community management is just fostering and gardening the relationship you have with your community. As with the blog, a monthly email can help you inform the developers and others about what's happening. And, as Bryan points out, whether people read it or not, even seeing just the subject heading of an email is "simply to remind people that you exist and you're moving forward."

Start thinking about the community around your platform as part of the value of the platform. Rather: the community is part of your platform, so the better you manage the community, the better you'll be managing the platform.

Running internal conferences and a speaker series for your platform team

Why do this at all?

If you want people to use your platform, they need to (1) know it exists, (2) understand what it does, and (3) actually want to use it. Internal conferences and a regular speaker series help with all three. They provide structured, recurring opportunities to showcase what the platform does, train developers and ops folks, and build a sense of community.

The goal isn't to just throw events, that's just a fun side effect. The goal is to make sure developers understand what's available, learn best practices, and, ideally, share what they've built.

Done right, this makes adoption easier, reduces onboarding, retains existing developers and grows their use of the platform, helps attract new developer teams to the platform, and helps platform teams stay connected to their users. Remember, that in a platform as a product approach, developers are your customers. If they don't know what's available, how to use it, or what's coming next, they'll find workarounds. These conferences and speaker series are a way to keep developers engaged, improve adoption, and ensure the platform stays relevant.

There's a human side to this, too often left out of focusing on "the business value" and outcomes in corporate-land: just having a friendly community of humans who like to spend time with each other and learn. That's why we call it a "community," not something like "a phalanx of continuously improving enterprise effective entities."

Formats that work

You don't need giant, multi-day vendor-style conferences. Those are great for different purposes, but expensive, over-kill, and not effective for what we're talking about here. The right format for an internal platform community is more like:

- **Quarterly Conferences** – Half-day or full-day events (virtual or hybrid) where platform teams, developers, and management come together. Keynotes, roadmap talks, deep dives, and developer stories. Speakers here are mostly your own community members, but often have at least one external speaker to bring an outside perspective and new information into the community
- **Monthly Speaker Series** – Shorter, focused talks (40 min talk, 20 min Q&A) from internal or external experts. Topics should include a mix of platform-specific updates,

programming techniques (with relevant examples, like Spring Framework), and industry trends. This is also a chance to discuss ongoing and important internal strategy and initiatives related to the platform. For example, what open banking standards are and how the platform could support them.

Everything should be recorded and stored somewhere accessible. These sessions aren't just for the live audience—they build a growing library of training material.

Comfort through branding

For these events, you should establish and have consistent graphic assets. Presentations should use the same formats, in-person events should have branded banners and schwag (like water bottles, stickers, t-shirts, even higher cost items like jackets as prizes for any contests). Remember that part of the power of brand is consistently using the brand assets. Since you've done your branding work as discussed above (right?), you have the graphics, logos, and even color schemes to start using. Just make sure you use them.

The archive is as important as the live show

Don't think of these events as just one point in time. Just as with YouTube, you should expect a lot of viewers to come to the sessions afterwards. One goal is to have people share these sessions broadly, relying on the viral nature of sharing to help extend out and grow your community. Thus, you need to put some thinking and gardening into the internal archives of your recorded sessions.

First, finding and watching the videos should be easy. For example, people often distribute recordings as Zoom meetings that require a password to watch. It's much better to have an internal web page that allows viewers to just click on a video and watch. Intranet systems like Google Apps and Teams make this easy: just download the video to a corporate drive and link to it.

Having good titles and brief descriptions of the recordings is required. Transcripts are also handy to have, especially since most services like Zoom can auto-generate them: it's fine if the transcripts aren't perfect. Just think about your own experience using YouTube and what works well for you when you're researching a topic…and do those things.

Case: it's not impossible

In large organizations, the difficulty of changing old, "legacy" systems can be daunting and demoralizing. One of the benefits of sharing success stories in events like this is to show change is possible.

with these types of objects in mind, you should pick some of those difficult applications early

on. When you successfully change that project, you can hold it up as an example in your internal marketing and conferences. For example, in telling the story of modernizing their payment system, AirFrance-KLM's Oya Ünlü Duygulu says, "[f]rom the organization side, there is no more fear of big changes. If such an old application as EPASS can transform, then it's possible for any application."

Getting the right topics

A good event needs a good agenda. If people aren't interested in the topics, they won't show up. Here's what works:

- **Platform-Specific How-Tos** – Feature deep dives, best practices, common pitfalls, and real-world examples.
- **General Development Practices** – Bring in experts to talk about topics like testing strategies, security, or performance tuning (with practical examples relevant to your platform).
- **Internal Use Cases** – Developers love seeing how other teams solved problems. Get teams to present their experiences—what worked, what didn't, and what they learned. This also serves as marketing for your platform and helps your community get to know each other.
- **Industry Trends** – Give people insight into things like "Agentic AI" (or whatever the buzzword of the month is) and how it might impact their work. Technical people, especially developers, love learning about the latest and greatest things.

If you can, it's good to source speakers from other companies and organizations. For example, you might have developers come in from a different industry than yours to talk about their experience using the platform. Knowing "how other people do it" is valuable and often has more credibility than "how we do it." This also helps establish new relationships across companies, which is valuable for individuals' ongoing careers.

Not the party you're thinking

One of the longer running platform teams I know of is at JPMorgan Chase. For many years, they've run several internal platforms based on Cloud Foundry and Kubernetes. They've learned a lot. They know the power of regular events to both educate, but also further community. As Nadi Awad says:

> We have a great team that runs what we call "cloud party" - not the party you are thinking about. But, the goal here really is to bring everyone together in one room these are developers who want to move their applications to the cloud, cyber, network engineers, vendors, etc.

Regular "parties" like these will help educate, raise platform awareness, and establish new connections in your community. That whole human side of things I mentioned above also benefits from events like these.

When people from different groups and "silos" regularly attend the same events, they tend to soften up prickly relationships. These relationships are often based on talking through helpdesk tickets and policy. For the most part, that doesn't work out great. Instead, once the pricklies are lessened, even a bit, people start talking without a ticket between them. They might even start to help each other rather than just helping the all too often rigid bureaucracy.

Example talks

Here are some sample talks for events:

“Escaping Tech Debter’s Jail: Modernizing Legacy Apps with Spring Boot”

- What it’s about: Breaking down how to refactor old, slow-moving apps into Spring Boot-based microservices.
- Why it’s useful: Helps teams move away from legacy monoliths without completely rewriting everything.
- How it helps the platform community: Encourages people to modernize while staying within the supported platform.

“That Time We Could Process Payments for 2 Hours: Lessons Learned from a Major Incident”

- What it’s about: A detailed post-mortem from a real production outage—what went wrong, how it was fixed, and what was learned.
- Why it’s useful: Gives teams insight into real-world troubleshooting and how to avoid the same issues.
- How it helps the platform community: Builds transparency and trust while reinforcing operational best practices.

“Getting Secure With Zero Hassle: Building Apps with Zero Trust Principles”

- What it’s about: An overview of zero-trust security models and how to apply them when building on the platform.
- Why it’s useful: Security that’s baked in rather than bolted on means fewer surprises and compliance headaches.
- How it helps the platform community: Ensures platform users follow best practices, keeping the ecosystem secure.

“Agentic AI and the Future of Development”

- What it’s about: Exploring how AI-driven development tools can improve coding efficiency and decision-making.
- Why it’s useful: Gives developers insight into how AI can assist rather than replace them.

- How it helps the platform community: Encourages adoption of new tools while keeping them aligned with platform best practices.

Making it a sustainable habit

To keep these events going, they need to be:

1. **Predictable** – Set a recurring cadence on the same day each month, e.g., for the monthly series, every third Thursday. People should know when the next event is happening without having to check.
2. **Well-Promoted** – Use internal newsletters, chat channels, and direct outreach to get the word out.
3. **Easy to Attend** – Hybrid or virtual-first works best for accessibility.
4. **Engaging** – Keep talks interactive with Q&A, live demos, and discussion.
5. **Measured** – Track attendance, gather feedback, and adjust based on what's working.

Platform Advocacy

The successful platform teams I talk with have very active platform advocacy. This means having at least one person working full time to just talk with, work with and listen to the people who use your platforms, usually developers. The role of "developer advocate" is pretty well understood by us vendors and cloud providers. Developer advocates love talking to people, and we also love talking about our craft. This means you can find out how it's done easily by just asking us.

You'll probably start with just one platform advocate who visits with developer teams throughout your organization listening to what these teams do, teaching them how to use the platform and associated methodologies and listening to their feedback. The advocate acts as a spreader of your platform, a booster and an explainer. Also, often overlooked, the advocate takes feedback from developers and others back to the platform team. They advocate for both the platform team and for the platform users.

As your platform and overall software transformation scale, you'll add more advocates. At large organizations, you might even have a whole team of platform advocates. The Cloud Foundry platform team at Mercedes-Benz provides training, systematic feedback collection, quarterly community updates and numerous other community management functions that you'd expect an advocate to help with.

An final insight

TK{ in this section, we go over what we discussed above. Then I give one insightful aside, and transition to the next section. }

Appendix: Resources

While the concept of "platform engineering" and "internal developer platforms" is relatively new (in 2025), platforms and platform teams have existed for a decade or more in large organizations. Much has been written about them, though not much about marketing. Here are some other resources on the topic, many mentioned and linked to above:

1. **"Improving JPMorgan Chase's Developer Experience on the Cloud," Nadi Awad, July, 2022.** An overview of JPMorgan Chase's internal platform with an emphasis on community management platform advocacy.
2. **"The Developer Platform," Rosalind Radcliffe, Charles Betz, Betty Junod, Ravi Maduposu, Luke Rettig, Mark Imbriaco, Levi Geinert, IT Revolution, September, 2022.** Discussion of the platform concept in general, including platform as a product/product managing the platform, plus a section focusing on platform marketing.
3. For more on the **US Air Force's Kessel Run**, see the wikipedia page and their website.