harness

EBOOK

Building Your Platform Engineering Team

harness

Contents

Introduction 3

What is Platform Engineering? 4

Build the Platform Engineering Team (PET) 5

1. Team make-up 6

2. Empathy is a Core Competency 8

Know your customer 10

Minimum Viable Platform (MVP) 12

Starting the Ignition 13

Building momentum 14

Scale the transformation 15

No Guard Rails = Chaos 16

How Harness can help your Platform Engineers 17

Introduction

Public cloud platforms have been changing the game since the mid-2000s. In a couple minutes (and a corporate credit card), developers could suddenly get the infrastructure they needed to get fast feedback on software changes, and unblock value delivery velocity. The “shift left” craze started around the same time as developers took on more and more responsibility to go faster and faster. But, like a kid gaining freedom to explore (and break things), most organizations quickly realized some control was needed to keep companies out of trouble once software was deployed beyond their fortified network walls. Insert manual security, compliance, cost, and other controls and we’re back to where we started with hand-offs and oversight handcuffing innovation. Literally stalling velocity that could help companies win more customers and beat the competition.

The idea of a Platform as a Service (PaaS) sounds great for a company going through a “shift left” transformation, but consider the cognitive load shift left creates for developers. Plus all the baggage on top of that on how to enable developer autonomy on the PaaS (opinions, security controls, regulatory requirements, etc.). It quickly becomes incredibly complex and shipping customer value grinds to a halt.

> **“I’m convinced the majority of people managing infrastructure just want a PaaS. The only requirement: it has to be built by them.” – Kelsey Hightower**

What is Platform Engineering?

Platform engineering is a discipline that focuses on building and maintaining the underlying infrastructure and systems that support software applications. It involves designing, developing, and managing the platform on which applications run, ensuring their stability, scalability, and reliability.

At its core, platform engineering aims to provide a solid foundation for software development teams to build and deploy their applications. It accelerates software delivery, ensures operational efficiency, and reduces the complexities of managing modern cloud-native ecosystems by providing developers with standardized tools and processes. Platform engineering empowers organizations to innovate faster, reduce costs, and maintain a competitive edge in today's dynamic digital landscape.

One of the key aspects of platform engineering is automation. By automating repetitive tasks and processes, platform engineers can streamline operations and improve efficiency. They use tools and technologies like configuration management, containerization, and orchestration frameworks to automate the deployment, scaling, and monitoring of applications.

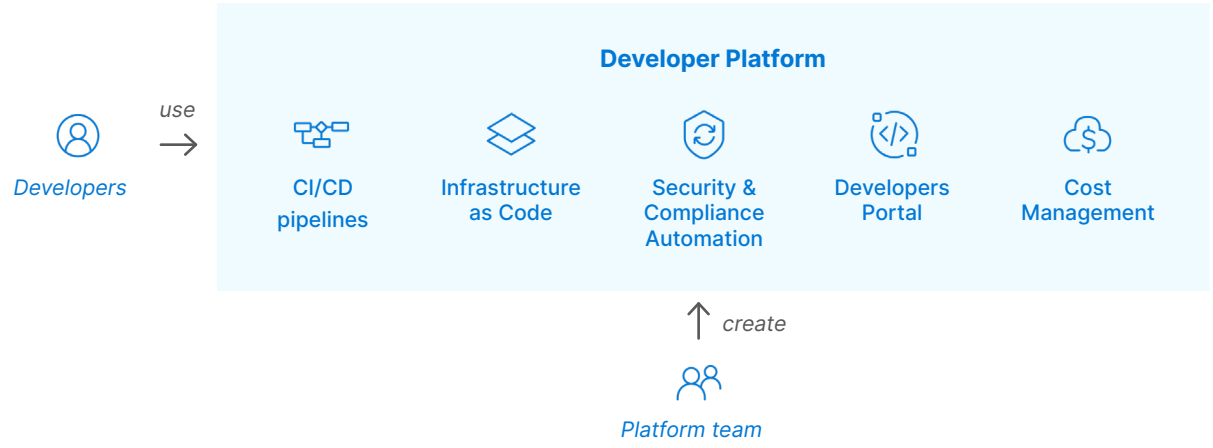


Figure 1: Developer platforms typically contain a portal, CI/CD pipelines, and infrastructure as code. Security and compliance automation is stretched across those capabilities, while easy access to cloud resources via Infra-as-code is governed by intelligent cloud cost management. These capabilities are built by Platform Engineering Teams for developers to consume.

Building a Platform Engineering Team (PET)

Martin Fowler's Strangler Pattern metaphor describes a pragmatic approach to modernizing an application architecture by refactoring edges of an application and over time, letting the refactored architecture overtake (or strangle) the old. Similar to how a strangler fig tree's branches gradually root themselves in the soil and overtake the tree itself. In a similar way, the tendency for most organizations building a platform engineering team is to transform an existing infrastructure engineering or maybe even DevOps team into a platform engineering team, but like modernizing application architecture, it's going to take time and sweat equity.

To accelerate the value delivered by a platform engineering team, start from the ground up by forming a new team that includes a combination of experiences, skill sets, and mindsets that will enable your company realize greater benefits sooner. This can be a hard sell to leadership and stakeholders, but change is never easy and if it is, then you're potentially misaligned on vision, mission, and possibly even the members you're picking for the team.

> **A good leader affects a team's ability to deliver code, architect good systems, and apply Lean principles to how the team manages its work products"**
> **- Accelerate, Building and Scaling High Performing Technology Organizations. pg 115-116. Forsgren, Humble and Kim.**

The success of the platform engineering team will depend on the leader and the culture he or she creates for the team. So getting the leadership part right is fundamental to the team's success. Like many people/culture challenges, choosing the right leader is more of an art than a science, but there are some factors to consider that can help increase the chances of success.

First, the leader should be politically savvy and have the ability to influence out and up. Things to look for include: good rapport with influential technology and business leaders, as well as open communication lines with high-performing engineers in the Dev, Sec, and Ops orgs. It's also important for leadership to have empathy for the responsibilities of the Dev, Sec, and Ops

orgs. Understanding and appreciating the challenges of the groups the platform engineering team is trying to bring together will position the team to collaboratively solve problems for the good of the organization.

Next, the leader should have a strong background in at least two technology domains between Dev, Sec, and Ops, combined with a passion for product mindset. This focus will help guide the team and organization in the right direction. The empathy and influence mentioned above will help get things done, while technology experience and product mindset will make sure the right things are getting done, the right way.

1. Team make-up

The PET leader should build a team that consists of the right mix of skill sets, mindsets, and personalities that will result in a high-performing engineering team. The “do not pass go” skillset/mindset required of all members of the team is “automate all the things.” Full stop. If they’re not striving to automate away operational work or streamline broken processes, then look elsewhere.

Your platform is a product, and your technology organization is your target market/customers. With that in mind, build out a product team. You will need engineers but also consider product, design and documentation skills.

Having a team that can anticipate roadblocks is important. Look to recruit members with experience in key bottleneck areas:

- **Security and Compliance**
- **Infrastructure**
- **Development**
- **Enterprise Architect**
- **Existing tools teams**
- **Employee Onboarding**

> **The strength of the team is each individual member. The strength of each member is the team."– Phil Jackson**

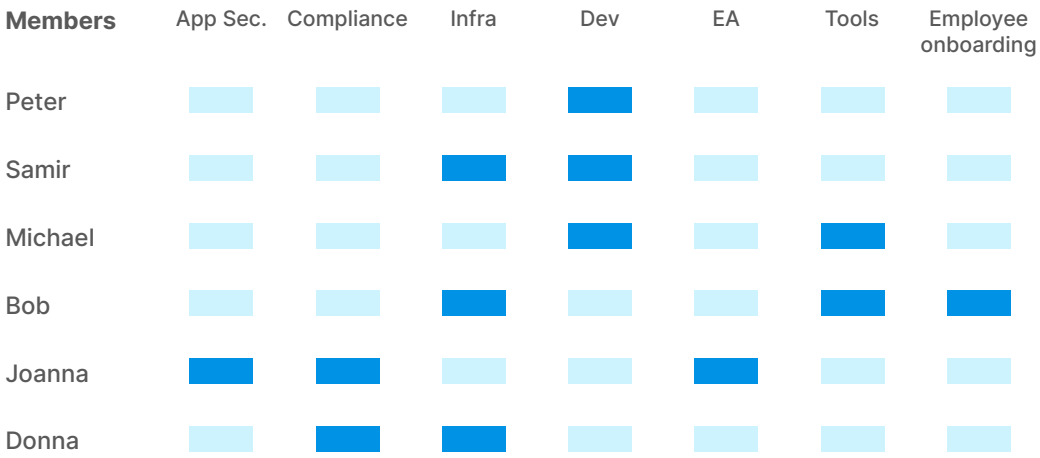


Figure 2: Consider the experience and skills you need on the team, and look across who you recruit to ensure that each experience is accounted for. A candidate like Joanna, who is the only one with App Security and Enterprise Architecture experience, is a priority to recruit to the team.

Ideally, the existing teams you're pulling the new PET members from will struggle temporarily due to the knowledge and skills leaving the team. The alternative of keeping the Brent around to continue shouldering the weight of the team's workload doesn't help the team grow or velocity to increase. For the good of everyone on the team and the organization, the "Brent band-aid" needs to be ripped off. Help the engineer grow by surrounding him or her with other strong engineers on the PET with an automation-first mindset. You also need to help the individuals on the team left behind grow by learning new technologies and taking on new responsibilities.

Sidebar: Meet Brent

Brent is a fictional character in Gene Kim's "The Phoenix Project" , a book about a fictional company going through a DevOps transformation. He is a talented and experienced engineer, known for his exceptional technical skills. However, Brent is more than just a brilliant engineer. He is the "Hero," often called upon to save the day when critical system issues arise. Brent's situation is a common one in many organizations. As a lone "Hero," he becomes a single point of failure. The Hero's Dilemma, as depicted in the book, highlights the challenges of relying on one individual to solve complex technical issues and maintain system stability. The Hero's Dilemma becomes an essential case study for understanding the importance of teamwork in platform engineering. If your organization is at risk if a key individual goes on vacation, it has a "Brent" problem to fix.

2. Empathy is a Core Competency

For both the leader and the members of the team, an essential quality required for success is empathy. All empathy is important, but in this case, we're looking for empathy specifically around the challenges developers face in delivering value for the business. We're also talking about empathy for the responsibilities of every other group or role involved in the building, delivering, and running software. Why is this important? Traditional responsibilities and motivations for "dev" and "ops" are in direct conflict with devs being pushed to make changes for the business, and ops being pushed to maintain stability. Depending where you sit in the org, change can look like an attack on the thing you were hired to do. Having an appreciation for these competing responsibilities and challenges will help the PET breakdown the wall of confusion described by Andrew Clay Shafer to collaboratively solve problems for the company.

Other characteristics needed to build a great platform engineering team include, having a growth mindset, a bias towards action, and strategic vision with pragmatic planning. Carol Dwek described growth mindset vs. fixed mindset, and since the platform engineering team will be solving some extremely complex problems, you need engineers who are hungry to learn and experiment with new ways of solving problems. They need to be curious and never satisfied with the status quo. They need a growth mindset.

But they can't just be researchers and information absorbers, they should be putting their learnings to the test with hypothesis-based, problem solving experimentation. Solution hardening with alignment to architecture principles, non-functional requirements, and enterprise implementation designs will come but first, the team needs to determine what to build. A good rule of thumb is if the team has spent more than two stand-ups or mobbing sessions talking about potential solutions, they've spent one too many meetings discussing it and they should test their hypotheses before discussing further.

Finally, the team needs to be aligned on where the team and the organization is heading, i.e. the mission of the platform engineering team. Yes, this includes the vision for the company, but also a culture that appreciates the people, process, and technology hurdles getting in the way of achieving that vision. This will position the team to start laying down the stepping stones needed to get the platform engineering team, engineering organization and company as a whole moving in the right direction.

Now that we have our team, let's talk about ways the team can become the change agents the organization needs to ignite the modernization transformation.

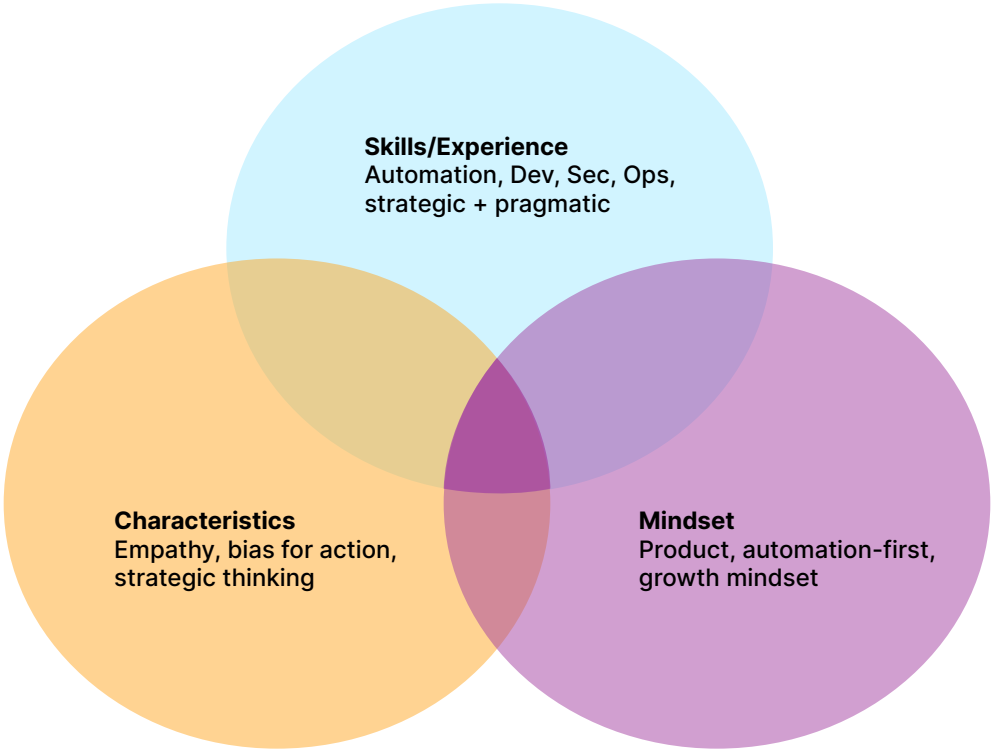
Know your customer

An influential DevOps engineer coined the term “immersion” to describe an effective engagement method for the platform engineering team members to develop the empathy needed to bring Dev, Sec, and Ops together. An immersion is conceptually similar to the lean concept of Gemba Walk, where leaders observe work activities of their employees to better understand the difference between what they think is happening to complete a unit of work and what’s really happening. A Gemba Walk for the PET is an immersion that’s treated as both a partnership and a project, where the PET engineer is considered a member of the development team and there is a mutually agreed-upon start and end (date, outcome, objectives, etc.).

Look for a team that’s innovating with support from the business, ideally on a greenfield project, e.g. a team with a business initiative to build a new cloud-native solution, or a team rapidly innovating with new customer experience solutions. If needed, lean on the rapport of the product manager or team leader to help identify a team that will be a good partner on an initiative that has some visibility with key stakeholders. In the immersion, the PET engineer(s) literally sit next to software engineers from the time they clock in to the time they leave for the day to write and deliver software collaboratively. Look for repeat challenges, especially fighting tools and processes, as well as context switching. Help them solve problems with solutions they may not know already are available, but more importantly, be sure to bring unsolved problems back to the team for awareness. This funnel of software delivery challenges is the start of the team backlog.

In addition, conduct some focus group-style interviews with high-performing and influential engineering teams and leaders across the org. If you're in a federated environment, include teams from multiple lines of business, and include teams from Dev, Sec, and Ops domains. The goal here is to understand the challenges each team faces in their day-to-day, as well as their strategic goals, to bridge the gaps in Dev, Sec, and Ops.

The PET team needs to understand these day-to-day delivery and long-term goal attainment challenges across the business and across technology domains to make informed decisions on the top priorities to solve with the platform they're building. While the backlog is being built, and the high value/low effort prioritization method is surfacing the problems for the team to solve, the product manager/team leader should be establishing their "demo group." This demo group should consist of your executive sponsor, leadership team members, and engineers. To start, 6-10 members is a large enough group and you can treat this like a traditional scrum demo where you're demonstrating what's been built and seeking feedback on your next iteration.



Minimum Viable Platform (MVP)

In most organizations, the traditional developer tools team (e.g. the original CI/CD pipeline owners) is a centralized team that was originally created to help streamline the process of building and testing software. Over time, new challenges surfaced that created bottlenecks (e.g. handoffs between Dev and Ops, maturing CI practices to CI/CD practices), especially in organizations with security and compliance obligations to fulfill.

The platform engineering team is not looking to replace the duties of the CI/CD team, but instead rewrite the internal, unofficial contract of who's fulfilling them. This happens by decentralizing Dev-Sec-Ops responsibilities and moving towards the AWS "you build it, you run it" model that eliminates hand-offs. But with so much cognitive load already being shifted left to the developers, Sec and Ops can't just throw these things over the wall to the Devs and say "this is yours now."

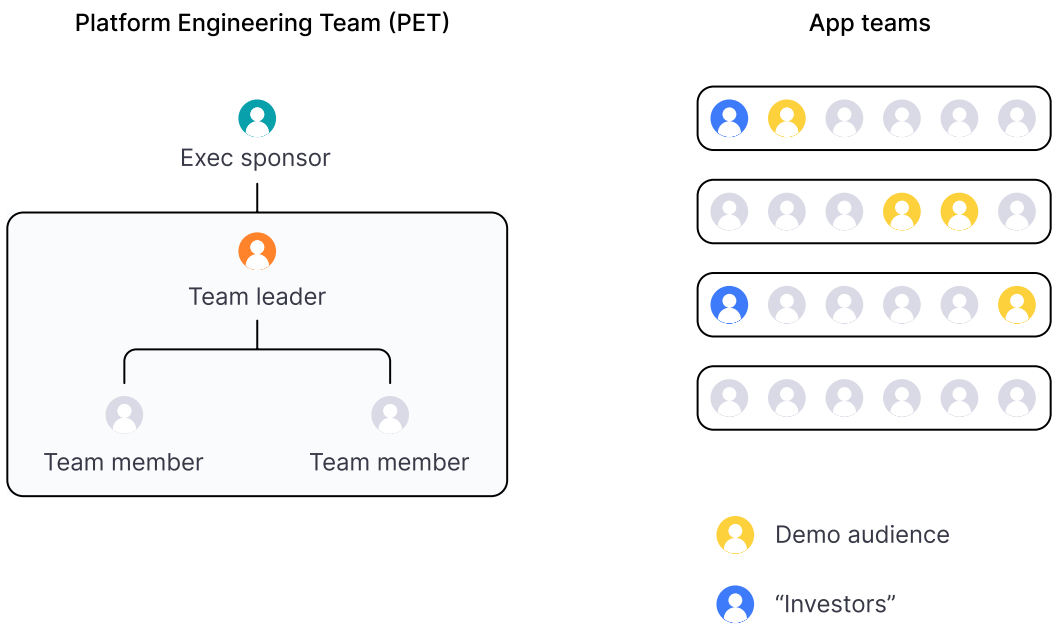
Instead, platform engineering teams should build automation, the right abstractions, and foster a great developer experience that helps IT organizations incrementally move towards the "run what you build" model. Ideally, Platform Engineers work to build standard templates that encapsulate best practices for the organization, making those available to application teams. While the application teams stay in control, the easiest thing to do is also the most standard.

Start with a minimum viable platform (MVP) that addresses the high-value/low-effort challenges identified through immersions, interviews, and PET insights. What constitutes an MVP will look different in most organizations, but it likely starts with infrastructure as code (IaC) that includes appropriate security controls. Follow that with an automated pipeline to provision that infrastructure, and shortly after, an automated pipeline that gets software built, tested, and deployed on that ephemeral infrastructure.

Looking ahead, we'll discuss how the platform engineering team iterates their way to delivering more process and technology solutions. Things like an internal developer portal (IDP), reusable pipeline templates, security orchestration, feature toggling, and governance guard rails that work together to create autonomy, reduce burnout, increase velocity, and mitigate risks to the business.

Starting the ignition

Once the team has solved a few meaningful problems for their customers, it's time to start marketing the team's success. Create developer-friendly communication channels, like a platform engineering Slack channel, where engineers can share feedback, ask questions, and opt-in to helping solve others' problems. Use this communication channel to identify the next members of your "demo group" who will engage in the PET demos, provide constructive feedback, and participate in solutions. Within that group, identify your "investors," i.e. the customers or stakeholders who are willing to put something on the line (time, political capital, etc.) to help your team succeed. You'll need these investors to break down roadblocks that come up throughout the transformation journey.



Establishing a center of excellence (CoE) is an effective model for bridging the Dev-Sec-Ops divide and building momentum for the transformation. Use those active members you're seeing in your demo group and Slack channel to establish the CoE and help shield the PET from the developers who are struggling to take on new responsibilities. The CoE members should sit closer to the developers, organizationally, so they can also help with context the centralized PET may not have. This CoE group eventually becomes an extension of the PET to transform the culture, processes, and technologies needed to achieve modernization goals.

Building momentum

With the PET leading the charge, along with investments from the CoE, now it's time to light the transformation on fire. Start with opening up cross-organizational access for your learning, information sharing, collaboration, and source code management tools. Obviously, abide by security requirements that must be met, but if there is not a security or regulatory requirement for keeping Confluence as View Only, then open up write access to everyone in the org. If someone makes a change they shouldn't have, roll it back!

Same with your source code management repos. Establish an "innersource culture" where information and code are freely shared within the organization, and contributions are welcome. Better yet, incentivize those community contributions and celebrate successes at tech-sharing events like lunch & learns.

Communities of practice (CoP) can be a useful engagement method for cultivating learning across the org. For example, architects can teach devs about non-functional requirements, devs can learn from other devs about XP practices, security engineers can help devs and ops understand the threats the security team blocks every day.

As the CoE group is maturing, and CoP groups are forming, don't forget about that original demo group. It might make sense at this point to spawn another demo group where the entire community is invited to see what the PET developed or experimented with in the last sprint, what the CoE and CoP groups have been enabling in their communities, and what's coming up next. It won't take long for the CoE, CoP, and innersource groups to build momentum for changing the way Dev, Sec, and Ops deliver software and for your stakeholder demo group to appreciate the value delivered to your end customers and want more, before the PET runs into a scaling problem. Trust when you've hit a scaling problem, you're on the right track to transforming how value is delivered at your organization.

Scale the transformation

Like any technology problem, you can always throw more headcount at the situation and hope more hands on keyboards will result in more problems solved. If there is an appetite for headcount investment, it should be a consideration for tackling the scaling problem, but if the cooks hired for the kitchen don't have the right skills and mindsets, then you'll end up with too many (wrong) cooks in the kitchen scenario. As you mature the platform and step into new problem domains, don't hesitate to bring in new skillsets to help attack those problems. For example, at some point you're probably going to need a developer advocate and you'll probably need a strong technical writer. These investments could produce a positive ROI for the team and the company, so hiring additional engineers in these situations makes a lot of sense.

Speaking of ROI, continue operating with the product mindset and treat the platform you're building as your start-up business. Think outside the box and experiment with different ways you can grow "revenues" (e.g. user adoption) and increase customer satisfaction (e.g. developer experience). Look at what's popular in your innersource repos and make appropriate investments to harden, aka operationalize, those solutions in your platform so the rest of your customers can benefit. Make sure your employees are as efficient and productive as possible, investing in the right tools and fixing broken processes to protect your business' most valuable assets, your platform engineering team engineers in this context.

The platform engineering team should already be investing in self-service for everything built in the platform. If something you're building requires a developer to submit a ticket and someone to manually approve it, you should revisit the solution. That said, in some situations, you need to get a repeatable process established before it can be automated so manual processes can be an appropriate MVP. Regardless, doubling down on self-service investments will produce the highest returns for the platform engineering team somewhere along the journey from igniting to scaling the transformation.

No Guard Rails = Chaos

Throughout this transformation journey, the laggards and doubters will look for failures and create friction that will steer decision makers to sticking with the old way of doing things. The only way the self-service model and the platform engineering Team as whole can succeed is if there's alignment and control to prevent chaos. With appropriate guard rails preventing chaos, autonomy and velocity can flourish. Trade-offs between control and velocity or security and velocity or quality and velocity no longer exist, provided self-service and the IDP is built with enforceable (and auditable) guard rails in place.

Most of this article has focused on building the team and culture for a great platform engineering team and transforming how value is delivered to customers but the endgame that Kelsey Hightower talks about will be here before it starts if the platform engineering team hasn't applied enforceable (and auditable) guard rails to keep the journey on track. The 1st publicly accessible S3 bucket provisioned via an automated CI/CD pipeline or the 1st unapproved software deployment will take you back to the start of the journey and require political capital you may not have in the tank. Do it right from the start with guard rails in place to keep engineers out of trouble. Make it easy for them to do the right thing, and hard to do the wrong thing.

How Harness can help your Platform Engineers

Harness Internal Developer Portal (IDP) enables you to create new software components quickly while adhering to your company's best practices. It enables you to manage the software you own by presenting a developer-centric view of all relevant information such as service health, deployments, and alerts. It also enables you to explore the internal software ecosystem of the company, discover technical documentation, APIs, and services, all of which enable better collaboration. Fully integrated into the Harness platform, IDP reduces the maintenance overhead and investment required from platform engineering teams and is enterprise ready.

As a platform engineer, you can orchestrate the onboarding of services by creating pipelines in the Harness Pipeline Studio. On the other hand, as a developer, you can create a new backend service, API, or website by submitting a few details as configured by your platform engineering. Developers focus on what they do best, which is writing features, while platform engineers focus on creating software templates, automating processes, and enforcing standards.

Harness IDP includes curated plugins. You can choose the plugins you need based on the tools that you use. The plugins enable you to customize a software component on the basis of its type, and to present all information relevant to developers in a single view. The Harness IDP plugin library is based on the hundreds of open-source Backstage plugins available in the marketplace.

If you're interested in getting a demo of the IDP solution, you can click **here** and sign up. Eliminate cognitive overload by letting developers manage their software effortlessly and access everything at their fingertips.

The Modern Software Delivery Platform™

Follow us on

/harnessio

/harnessinc

Contact us on

www.harness.io