DevOps Benchmarking Study 2023

humanitec

Table of contents

Executive summary

Today's DevOps culture is accompanied by a growing interest in platform engineering. This reflects a widespread need for engineering organizations to finally address the original promise of DevOps; "you build it, you run it". As such, Gartner and many other industry experts named platform engineering a top strategic technology trend and are watching it closely. The discipline sets out to drastically improve developer experience in the cloud-native area, by letting teams build Internal Developer Platforms to enable developer self-service. These platforms are designed as compelling products by platform engineering teams to serve their customers, the developers. In short, platforms are being built that finally enable the true essence of DevOps.

Some past studies indicate that Internal Developer Platform build and usage correlates with higher DevOps evolution and performance (Puppet State of DevOps Report 2020 and 2021, Humanitec DevOps Benchmarking Study 2021, and the latest Puppet State of DevOps Report, Platform Engineering Special Edition). In this report we wanted to dig deeper, and detect more patterns that indicate the key differentiators between low and top performing engineering organizations.

For this reason we measured 1053 teams on their DORA metrics and how far they follow best practices, and scored them against these standards. We asked questions regarding the degree of developer self-service, developer experience, and platform tooling; as well as their methodologies for managing app configs, infrastructure management, and team setups.

Developers working in organizations with low or mediocre performance depend heavily on Ops teams for simple DevOps tasks. This results in ticket ops when developers need help deploying a new version of code, spinning up new environments, and provisioning infrastructure and resources—which then need wiring up to their dependent apps. This slows teams down and can make life miserable for developers and Ops. The same occurs for teams that take a more freestyle approach. When developers have more freedom with self-service capabilities that are not facilitated in a standardized way, too many are afraid of screwing things up. The result? A broken DevOps setup that causes employee burnout and toxic relationships between developers and Ops.

When it comes to top performing teams, we identified several key patterns that set them apart from low performing teams:

↪ Top performing teams built an Internal Developer Platform that drives DevOps success

93% of top performers use a platform built and maintained by a platform team. They follow a Platform as a Product approach and are based on a mix of open source and vendor software, glued together and built into the platform.

↪ Top performings teams use Internal Developer Platforms that boost developer experience and productivity

With a platform, top performing organizations enable developers to complete DevOps tasks like the creation of new feature or PR environments (83.6%), deploy to dev or staging environments (93%), and resource assignment to apps based on golden paths and a standardized approach (85%). Developers can also bootstrap a new app within less than two hours (53%) and complete all of these tasks entirely independently. What's more, they do so with a great developer experience and in a confident manner without fear of screwing things up.

↪ A new approach to config management

Top performing teams manage app configs in a standardized way across all apps (82%), and separate environment-specific from environment-agnostic configs (81%). They also manage to implement a clear separation of concerns, which lowers developers cognitive load without restricting their access to underlying technologies. This approach to app and infrastructure config management is a key differentiator between top and low performing teams, and is commonly referred to as Dynamic Configuration Management.

↪ Platform engineering and Internal Developer Platforms drive innovation in times of uncertainty

Internal Developer Platforms can improve developer experience, reduce developer cognitive load, and enable self-service capabilities that drive productivity. All of which can have a significant impact on overall business performance. This includes shortening time to market (thus accelerating new revenue stream creation) while minimizing security risks and optimizing cloud costs.

Key recommendations

01 Start building your platform NOW; "if you don't build your platform, it will build itself." Almost every organization has major parts of a platform already in place (think about CI/CD, IaC, cloud setup). The question is, whether you keep your setup free-floating, or facilitate it in a standardized way by building an Internal Developer Platform.

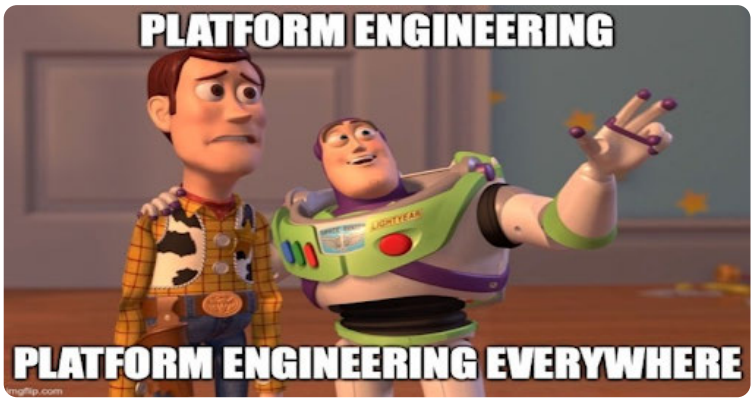
02 If you start building an Internal Developer Platform, your platform team should have a clear mission aligned with overall company goals. This could be the need to speed up innovation cycles, reduce time to market, get cloud costs under control, or mitigate security risks. So rather than simply replicating another DevOps or Ops team solving individual or team problems, the platform team should serve the entire organization.

03 Our study has shown that only low performing teams build Internal Developer Platforms entirely on their own. Top performing teams use open source tools, or a combination of proprietary software and open source tools, and treat their platform as a product.

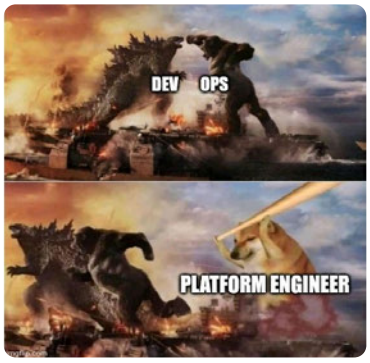
04 If you build an Internal Developer Platform make sure it supports Dynamic Configuration Management, and has a Platform Orchestrator that enables this. Avoid building a static platform, which has a high risk of creating new Ops bottlenecks, config drift, and infrastructure sprawl.

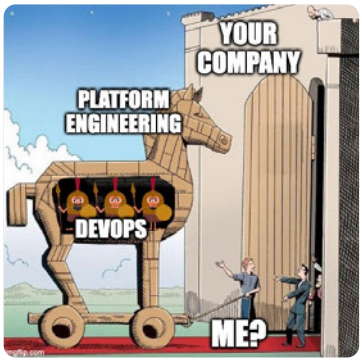
Getting platform engineering right and building a dynamic Internal Developer Platform enables teams to focus on what they're good at. It can help define the lines between developers and Ops, contribute to a well-oiled engineering setup, and subsequently heal the relationship between teams; without putting developers and Ops back into their silos.

Introduction



2022 has been an eventful year in DevOps to say the least. There was the launch of PlatformCon, a virtual conference which delivered 78 talks to 6000+ platform engineers from around the globe. platformengineering.org, the largest online community of DevOps experts, grew its Slack channel to 10000 active members. And platform engineering took the industry by storm, pitched by some as the DevOps killer[1]. Others like devops.com[2] disagreed, while DevOps Twitter grew white hot with debates. Then Gartner put platform engineering on their Hype Cycle[3] and even named it as one of the Top 10 Strategic Technology Trends for 2023[4]. This was a much needed conversation in the DevOps industry. And one that produced some great memes👇.





[1] "DevOps is dead, long live Platform Engineering!" (Platformengineering.org) https://platformengineering.org/talks-library/devops-is-dead-long-live-platform-engineering

[2] Alan Shimel: "Is DevOps Dead? I Don't Think So!" (DevOps.com, October 4, 2022) https://devops.com/is-devops-dead-i-dont-think-so/

[3] Luca Galante: "Why Gartner recommends Platform Engineering and building Internal Developer Platforms" (Humanitec, September 16, 2022) https://humanitec.com/blog/gartner-internal-developer-platforms-platform-engineering

[4] David Groombridge: "Top 10 Strategic Technology Trends for 2023" (Gartner, October 17, 2023) https://www.gartner.com/en/articles/gartner-top-10-strategic-technology-trends-for-2023

Today's DevOps culture is accompanied by an increasing interest in platform engineering, which shows a hunger amongst engineering organizations to address the original promise of DevOps: "You build it, you run it". The goal of DevOps was to break down the silos between developers and Ops, and enable faster software delivery. However, this quickly became a distant dream for most developer teams, who were weighed down by their increasingly complex cloud-native toolchains and home-grown delivery setups.

Aside from all the platform talk and our love of memes, it's also important to consider how platform engineering can translate to real business value. For example, how can Internal Developer Platforms (IDPs) drive efficiency across the entire organization, and help organizations reach their business goals? Later in this report we'll take a more in-depth look at how to accelerate innovation, improve team productivity, and attract and retain talented developers with the right approach to platform engineering.

> **Platform engineering is the discipline of designing and building toolchains and workflows that enable self-service capabilities for software engineering organizations in the cloud-native era. Platform engineers provide an integrated product most often referred to as an Internal Developer Platform, or IDP, covering the operational necessities of the entire lifecycle of an application[5].**
>
>  **Luca Galante** Product at Humanitec

Platform engineering guides organizations in the practice of delivering IDPs as products, finally enabling developer self-service, a better developer experience (DevEx), and the ability to innovate faster. As a result, we're seeing increasing DevOps engineers considering taking the platform engineering path. We're also seeing more organizations realize this potential and by adopting platform engineering, experience significant improvements that benefit the entire business. This is underpinned by Puppet in a special platform engineering edition of their State of DevOps report. According to the research, 93% of respondents say platform team adoption is a step in the right direction. 94% agree the concept is helping their organization better realize the benefits of DevOps, while 59% report greater productivity/efficiency as a result of platform engineering[6].

[5] Luca Galante: "What is platform engineering?" (Platformengineering.org, January 13, 2023) https://platformengineering.org/blog/what-is-platform-engineering

[6] "2023 State of Platform Engineering Report" (Puppet) https://www.puppet.com/resources/state-of-platform-engineering

About the survey

In 2022, we had hundreds of conversations with engineering teams. Our goal was to uncover DevOps best practices and areas where the industry can improve. We dug into organizations' degree of developer self-service, how teams manage configs and infrastructure, and their overall performance.

From here, we built a hypothesis: That top performing teams would be the ones managing apps and infrastructure config in a dynamic way. Proving this formed the basis of our study and as you'll see later, our assumption was correct.

For the 2023 report we asked 1053 teams from engineering organizations across NA, EMEA, LATAM and APAC about their technical setup and performance metrics.

Job title split:

Infra, Sys, Ops, DevOps, SRE, Platform Engineering, Architects	38%
Engineers, Developers (incl. Seniors, Principals)	29%
Management (Head of, Tech lead, Director, Manager), C-level (CTO, CEO, CXO)	33%

To present the results we first scored each teams' answer against well-established best practices and the four DORA metrics:

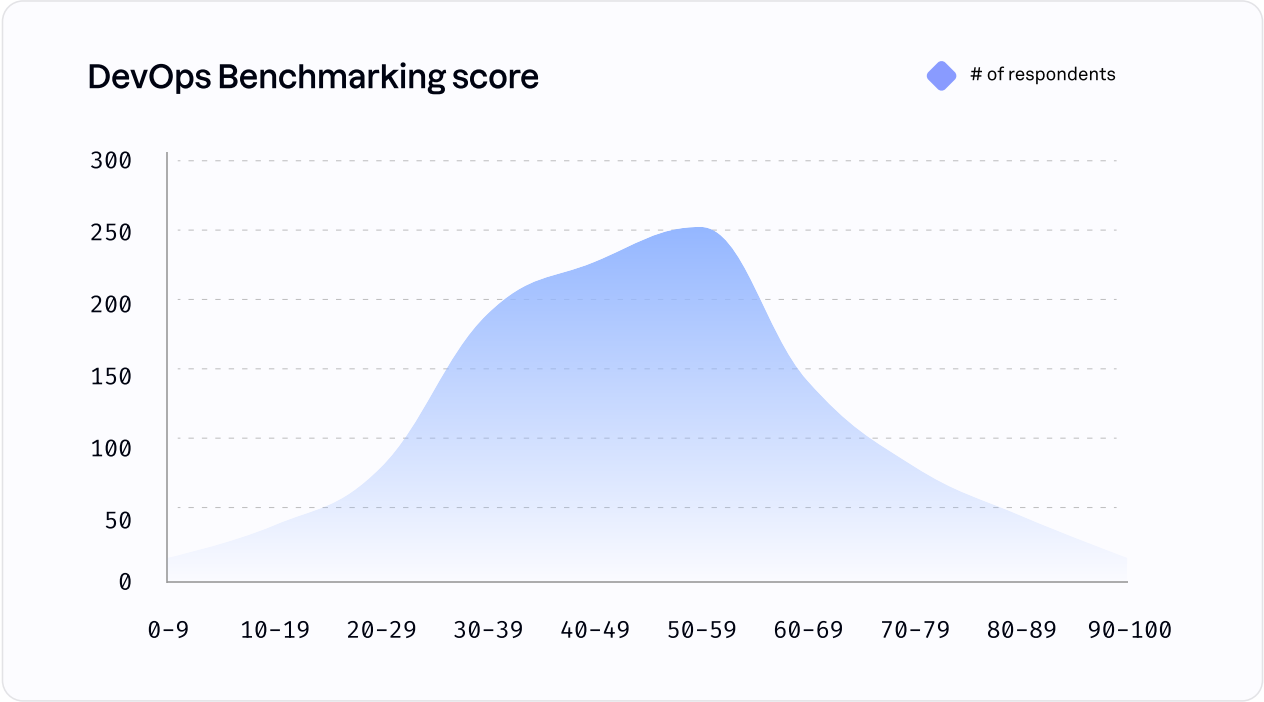
- Deployment frequency
- Change lead time
- Change failure rate
- Service production restoration time

Using this calculation we built four segments ranging from top to low performing teams, and used the results from each segment to explore our hypothesis.

Results

Scoring distribution

DevOps mountain of tears shows most teams are stuck with a mediocre setup



Let's take a quick look at how our respondents fared. We see a very clean, normal distribution, with a median value of 50 and an average DevOps maturity score of 49.88. Below are the exact splits across the four segments we'll analyze throughout the study:

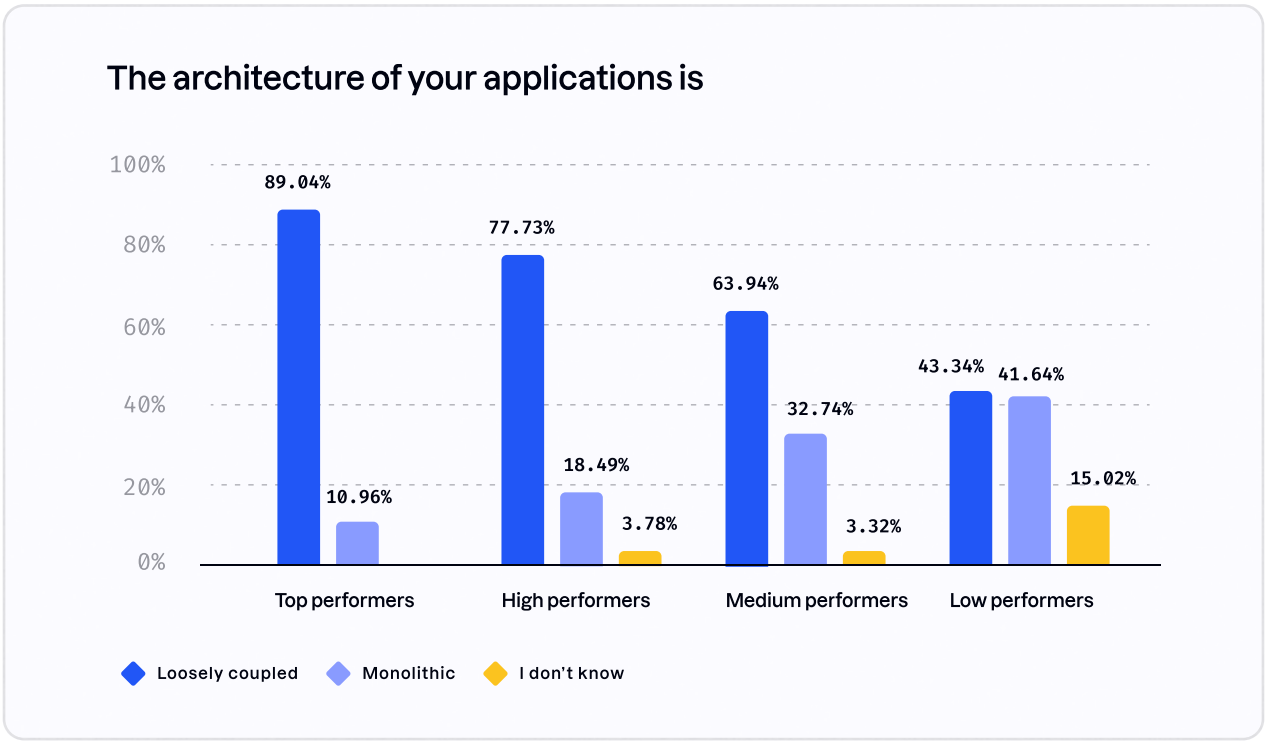
- Top performing teams → 80-100, 5.84%
- High performing teams → 60-79, 20.53%
- Medium performing teams → 40-59, 43.70%
- Low performing teams → 0-39, 29.93%

Technical setup

Architecture

Top performing teams have a microservice architecture

It's only the low performing teams that show monoliths are still on par with microservice architectures. The latter progressively dominates the rest of the field, with almost 90% of top performing teams using a microservice architecture.

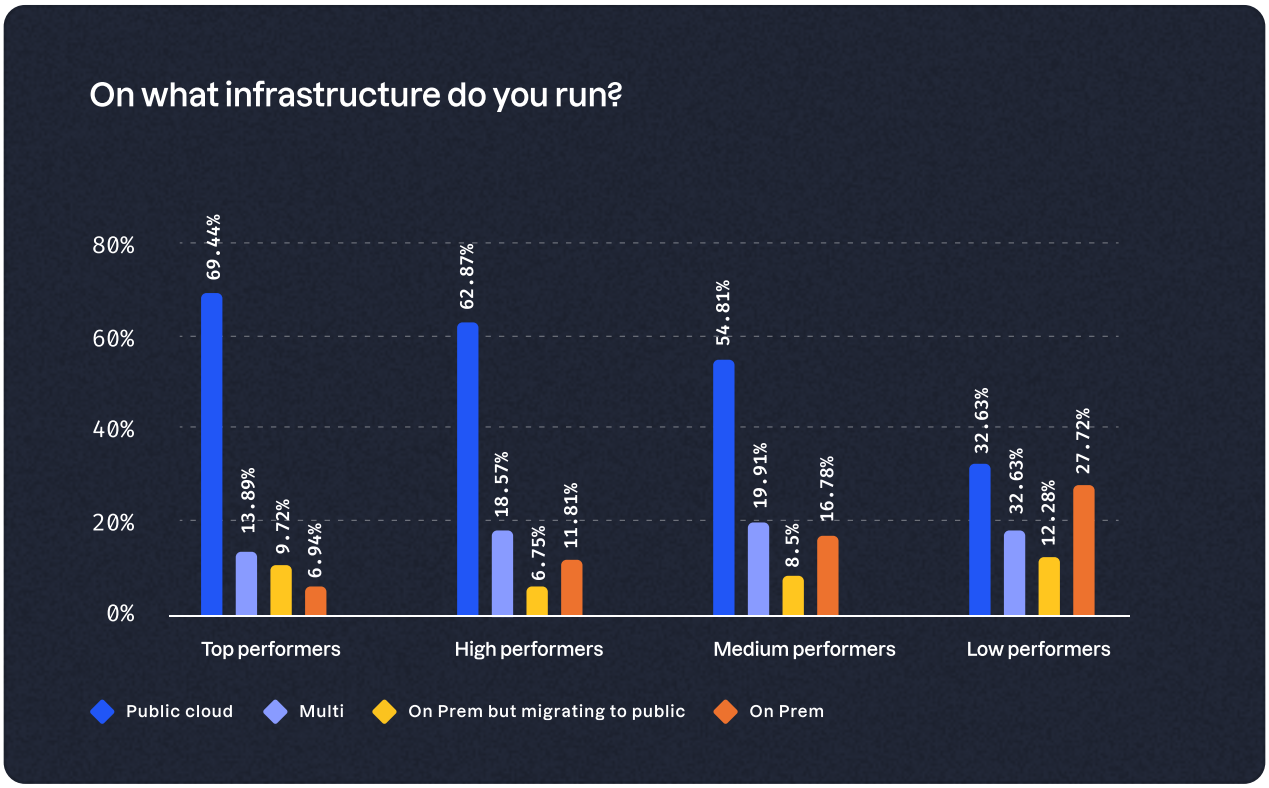


Infrastructure

More organizations are moving to multicloud, but the impact on performance might be negative

Public cloud use dominates with 50.8% of the total share, a relatively stable result compared to 49.6% in 2021. What's interesting to track is the double trend of falling on-prem (25.3% to 17.8%) and rising multicloud setups (17.2% to 21%)[7]. Another 9% of respondents state they are still on-prem but planning to migrate to cloud.

While the cloud migration trend may not come as a surprise, as most engineering organizations strive to modernize their infrastructure, the multicloud picture is more nuanced. There are many good reasons to go multicloud (e.g. avoiding vendor lock-in), but the data seems to indicate a potential negative impact on performance. Like most advanced implementations and rollouts, multicloud can have marginal benefits. However in most cases, it can risk doing more bad (Ops overhead) than good (diversification), especially for less experienced teams.

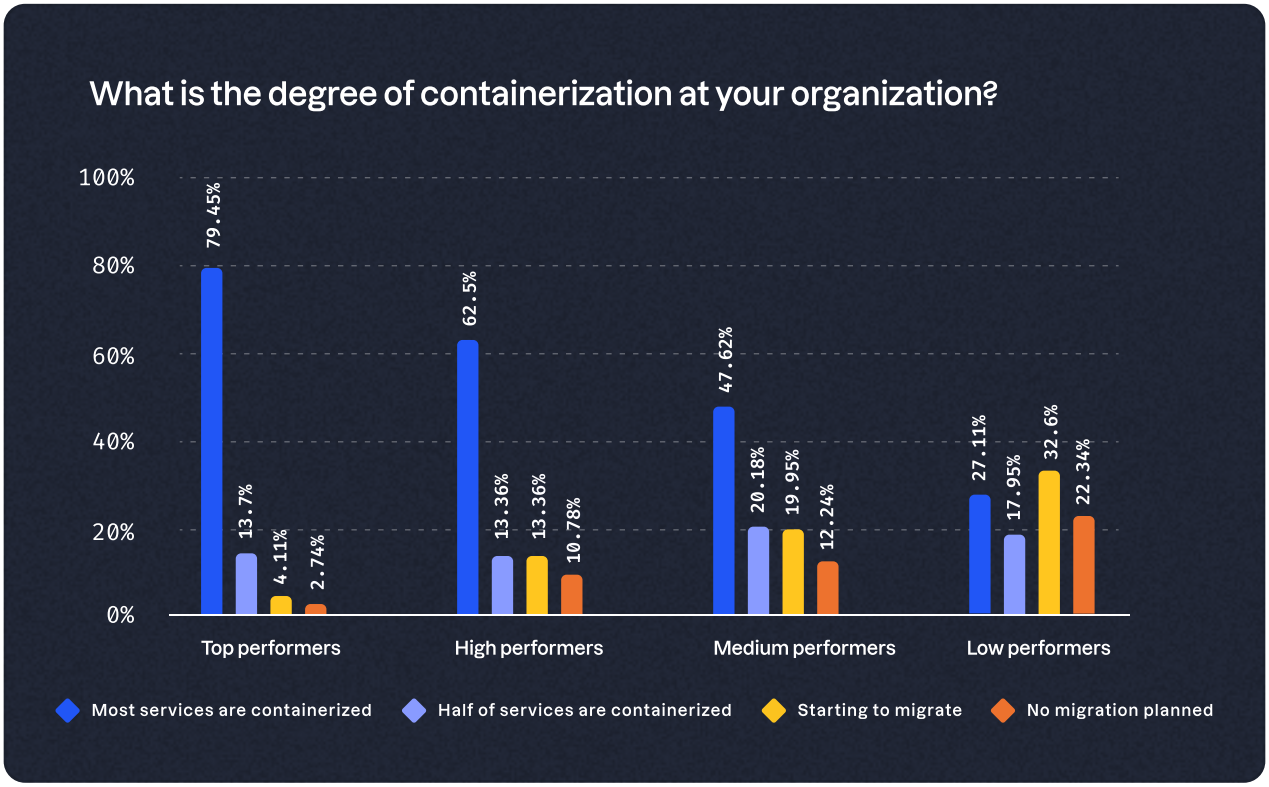


[7] Humanitec: "2021 DevOps Setups: Benchmarking Study"
https://humanitec.com/whitepapers/2021-devops-setups-benchmarking-report

Containerization

Low performing teams lag behind on migrating to a containerized setup

No surprises here; 93.15% of top performing teams have containerized at least half of their services (79.45% all services). While 54.95% of low performing teams have not even planned to migrate, or have only just started the migration process.

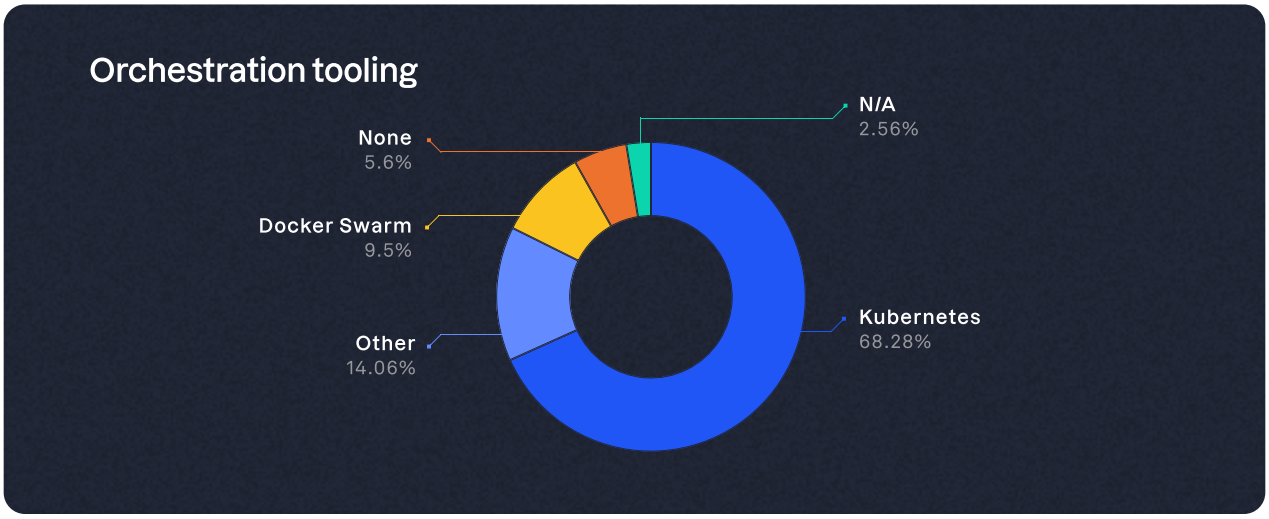


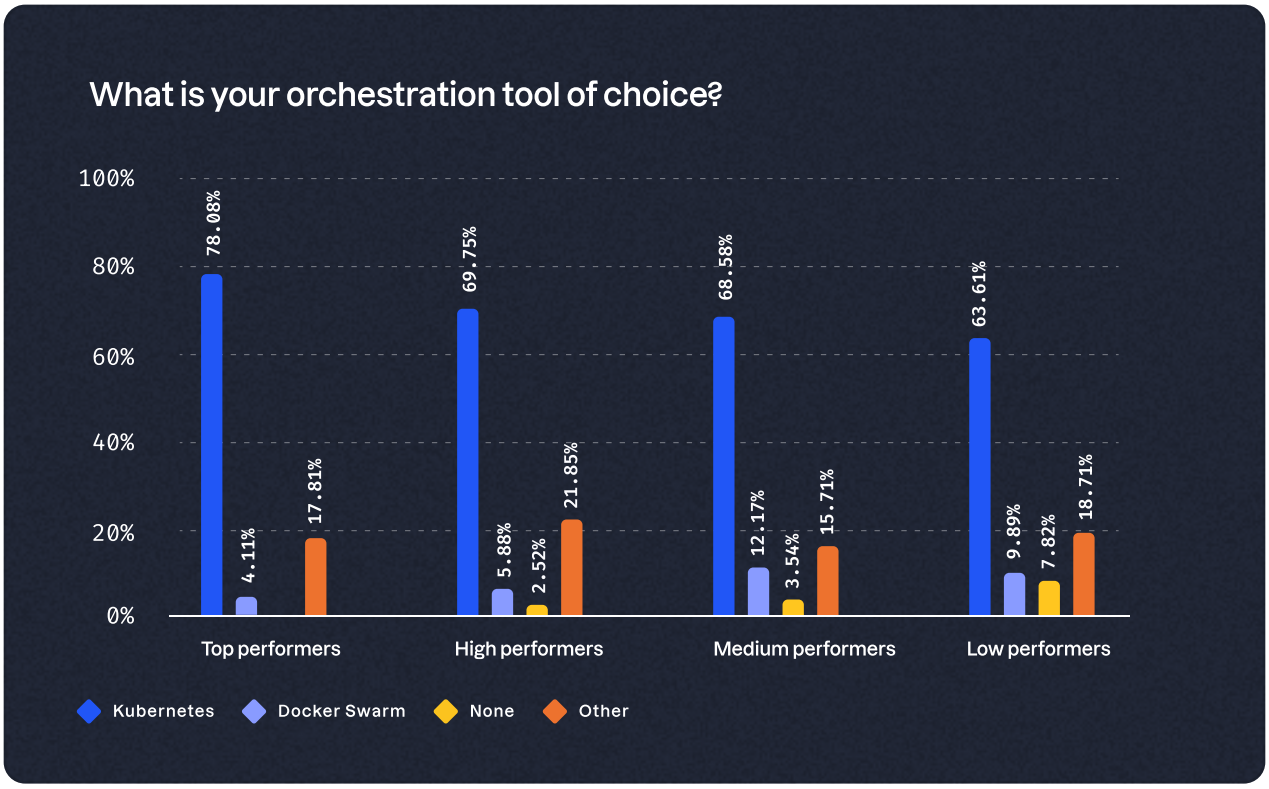
In this context, a containerized service refers to services that are ready to run on a container orchestration platform such as Kubernetes. In most cases this is a key element needed to run apps at scale, which explains the high correlation between containerization and performance.

Orchestration tooling

Kubernetes is our respondents' orchestration tool of choice

Speaking of orchestration tooling, Kubernetes continues to gain market share up from 62.4% to 68.2%. While Docker Swarm fell from 14.3% to 9.5%, smaller players remain approximately stable.



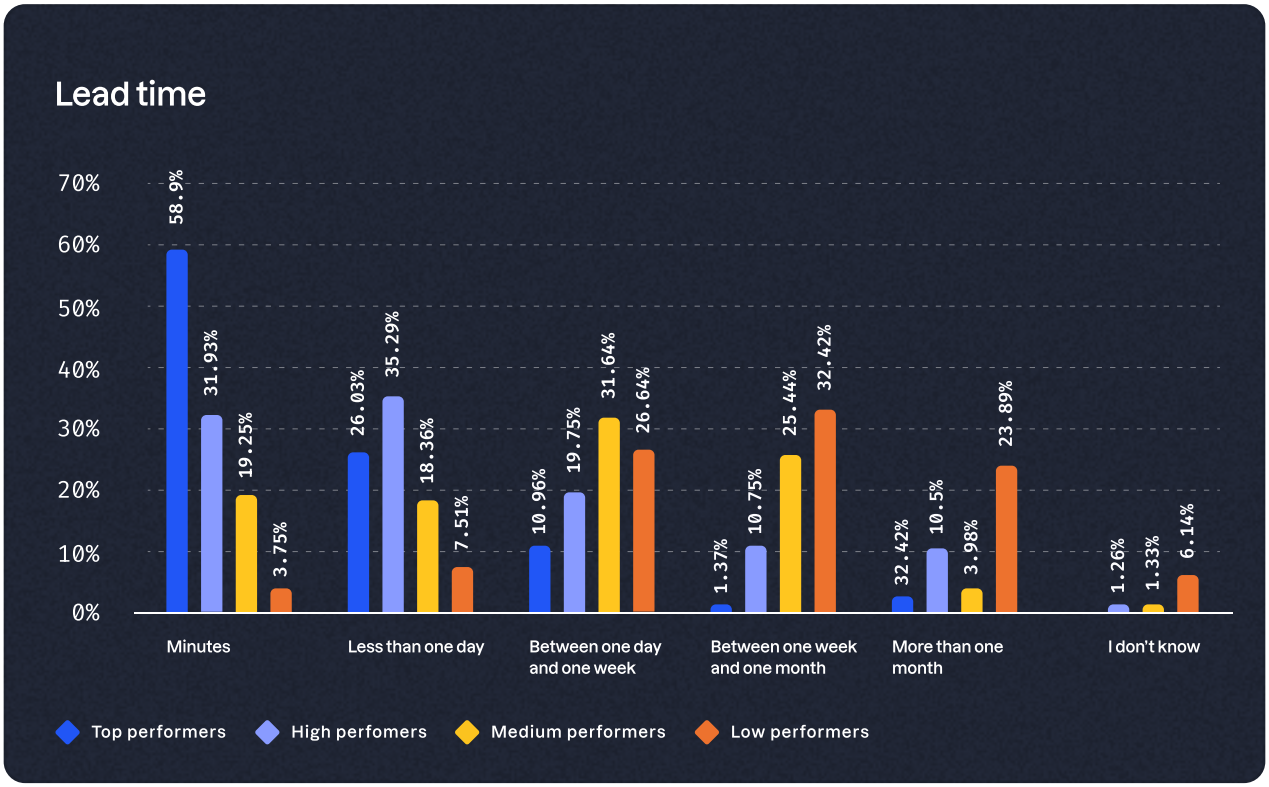


When it comes to the smaller players we discover a wide range of tools in use including serverless solutions like Lambda, Fargate, and ECS. Some teams are using PaaS-like solutions such as Heroku or OpenShift and legacy tools like Mesosphere (D2IQ). None of these solutions exceeded 2% across any of the four segments, and we were unable to measure any impact on performance.

Focusing on individual performance metrics, we looked at the classic four DORA metrics: Lead time, deployment frequency, MTTR, and change failure rate. For more information on the metrics as industry benchmarks, please check out Accelerate and DevOps Research.

Lead time

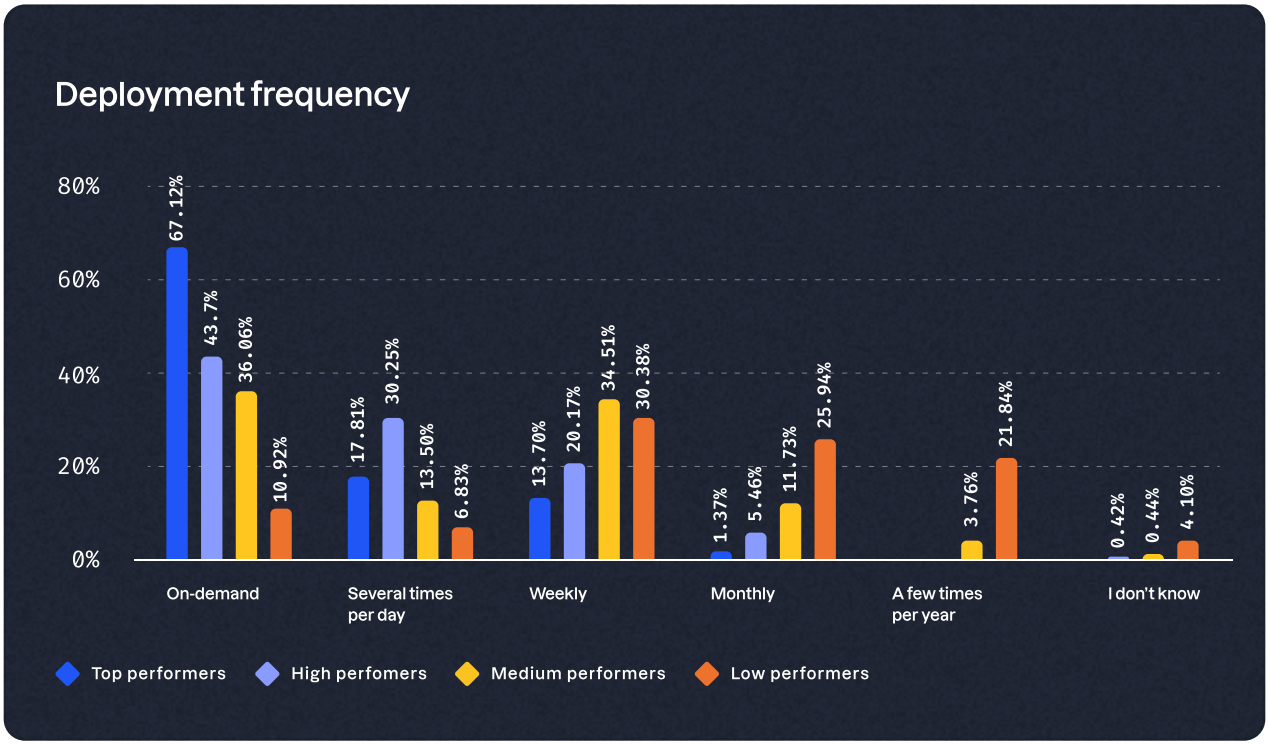
Lead time refers to the time between code commit and deployment to production



58.90% of top performing teams report a lead time of minutes, with 26.03% at less than a day. In comparison, 82.59% of low performing teams report a lead time of more than a day. 26.28% say this is between a day and a week, 32.42% between a week and a month, and a staggering 23.89% take longer than a month from commit to production.

Deployment frequency

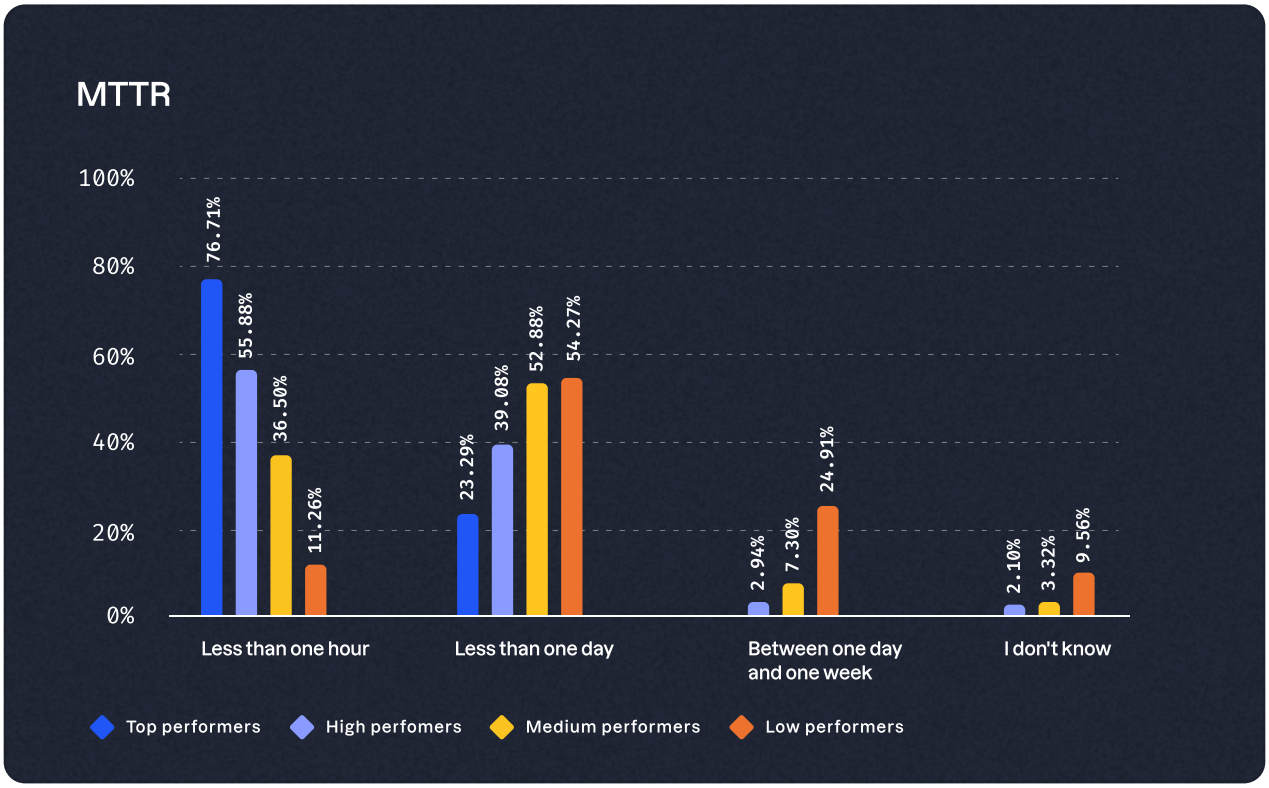
Deployment frequency tracks how often teams deploy to production, and shows how quickly engineering teams are able to fix bugs, update, or add new features.



67.12% of top performing teams deploy on demand. 17.81% of the same segment deploys several times per day, totalling 84.93%. Not too far behind are the high performing teams, with 73.95% deploying on demand or several times per day. Medium performing teams stand at 49.56%, and low performing teams only at 17.75% with the majority (30.38%) deploying weekly.

MTTR

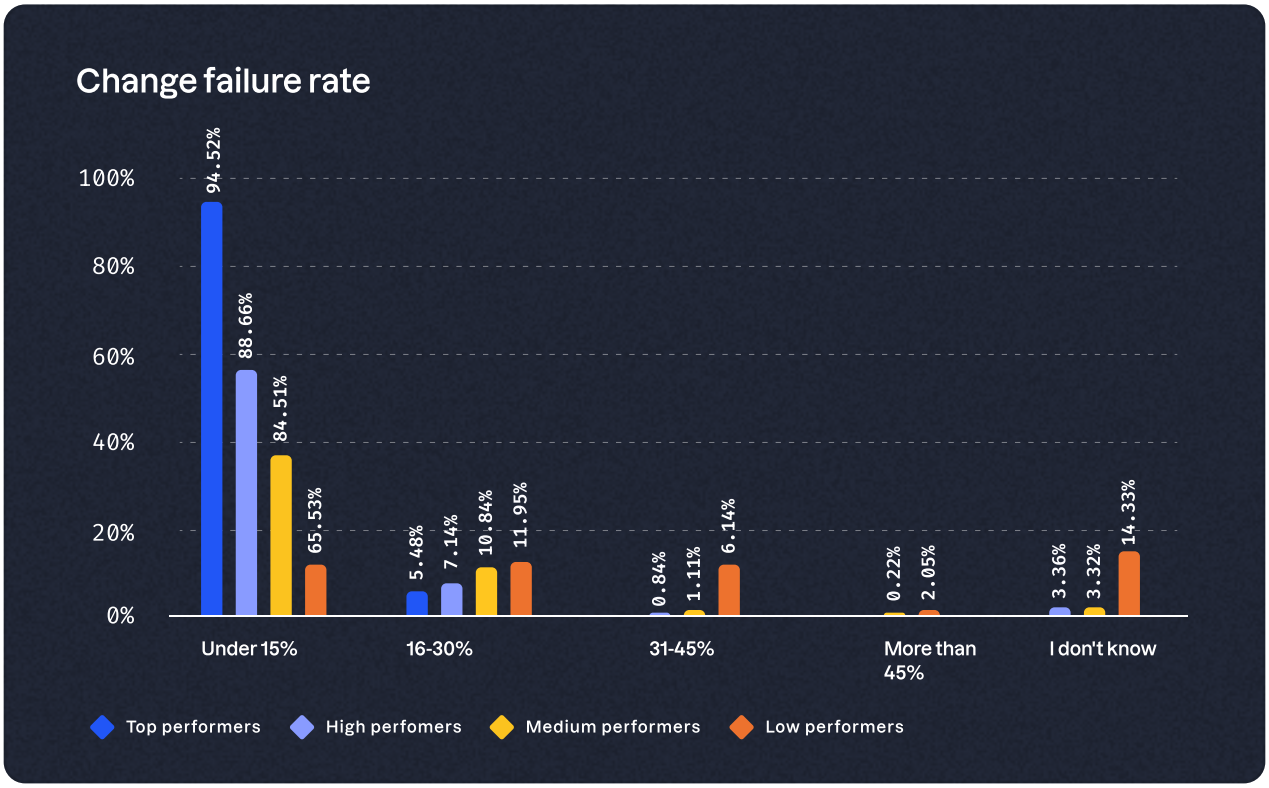
MTTR represents how long on average an engineering organization needs to recover from a system outage, and have all services up and running again.



Top performing teams report an MTTR of less than one hour (76.71%) or less than one day (23.29%). Only 11.26% of low performing teams say it takes less than an hour, and 24.91% need between a day and a week.

Change failure rate

Change failure rate indicates the percentage of failed deployments which require a complete rollback to the previous state, or immediate bug fixes.

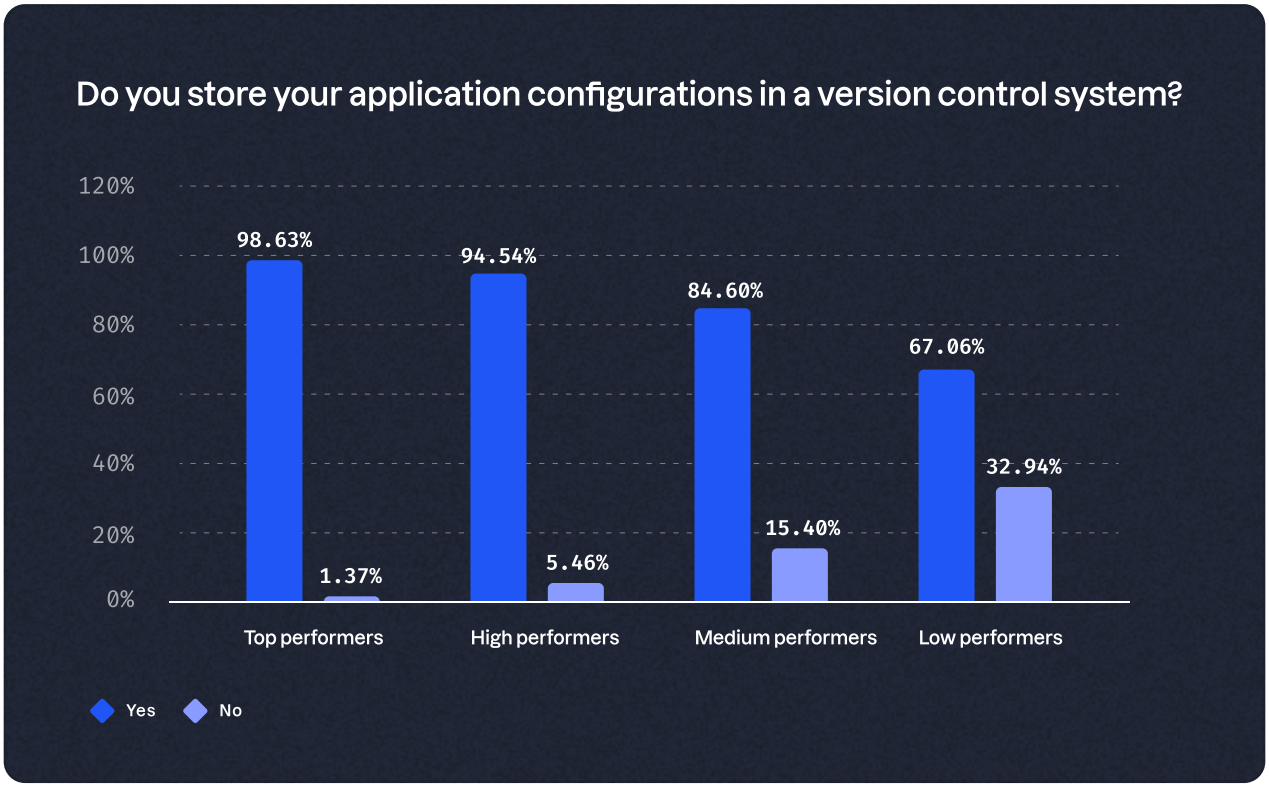


In this case we should have better framed the brackets, a lesson learned for next time when we'll take a more granular approach.

Configuration management

Application configuration

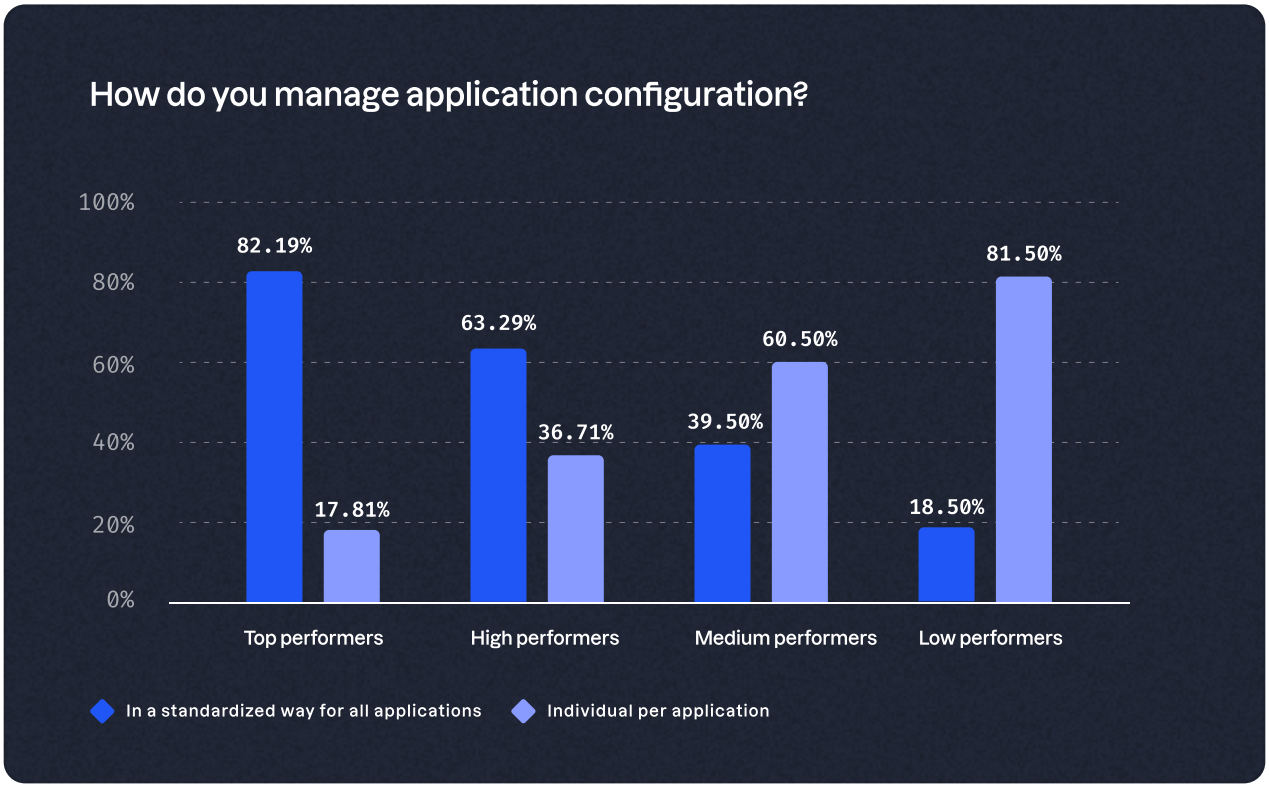
Storing app confs in a version control system (VCS) is a widely adopted best practice



Storing app configs in a VCS is a widely adopted best practice across all segments. Only 33% of low performing teams still don't do it.

Standardization

Top performing teams manage app config in a standardized way

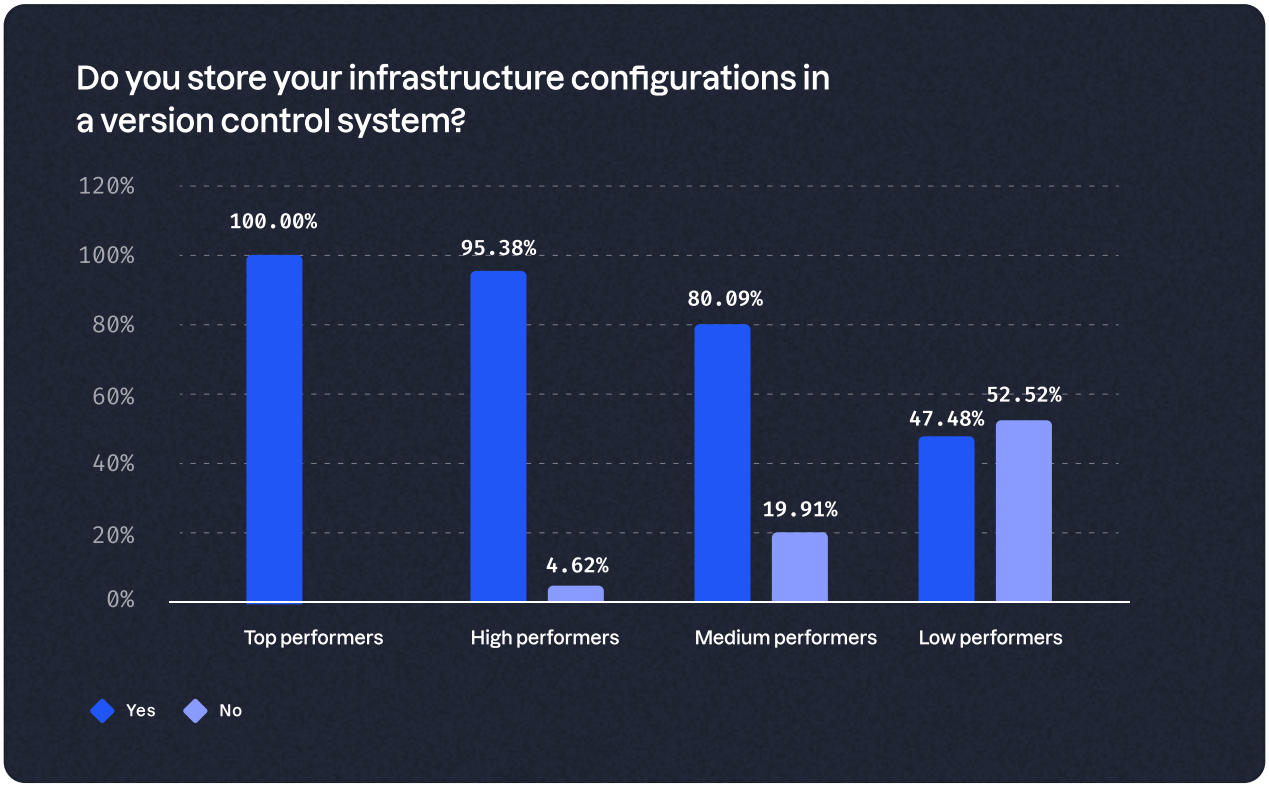


While 82.19% of top performing teams manage their app config in a standardized way for all apps, almost exactly the same number (81.50%) of low performing teams do not.

Organizations that try to standardardize config across their apps, such as by templating and automation, are hugely outperforming those leaving it to each team to decide. Lack of templating and standardization usually results in engineers inefficiently copying configs from existing services. This often leads to config drift, or redundant and incorrect settings applied to new services.

Infrastructure configuration management

Storing infrastructure configs in a version control system (VCS) is a widely adopted best practice

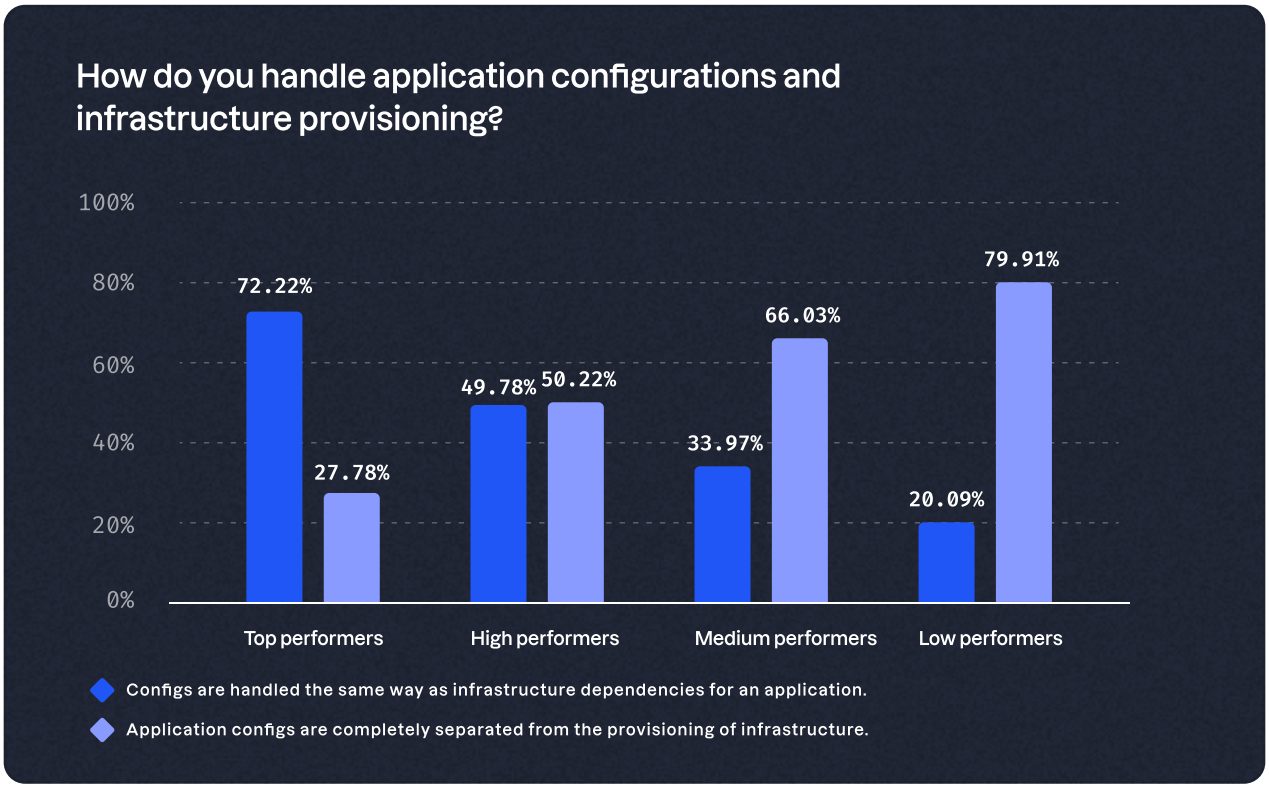


100% of top performing teams store their infrastructure config in a VCS, compared to 52.52% of low performing teams who don't.

It is an absolute must to have app and infrastructure config stored in a VCS. But this is only one cornerstone, which actually presents another challenge. Let's say an app has ten services, each with a certain amount of infrastructure dependencies. This leads to a few hundred files if your setup contains four different environments. With each deployment to one of those environments, new versions of these files are generated, which very soon can leave you with 10k+ files if you deploy several times daily.

Handling of application configuration management and infrastructure provisioning

For top performing teams, app config management (ACM) and infrastructure provisioning go hand-in-hand



72.2% of top performing teams handle app configs in the same way as infrastructure dependencies, while 50% of high performing teams report app configs are completely separate from infrastructure provisioning. This is also common practice among low performing teams, where close to 80% report app config is also isolated from infrastructure provisioning.

Handling ACM and infrastructure provisioning separately can be done by one or separate teams.

The latter case may result in a ticket ops workflow, where Ops teams have to fulfill developer requests manually. The big downside of this approach is its lack of scalability, and the subsequent delays caused by developers waiting for Ops to answer their requests.

In contrast, handling ACM and infrastructure provisioning together in a standardized way has several advantages:

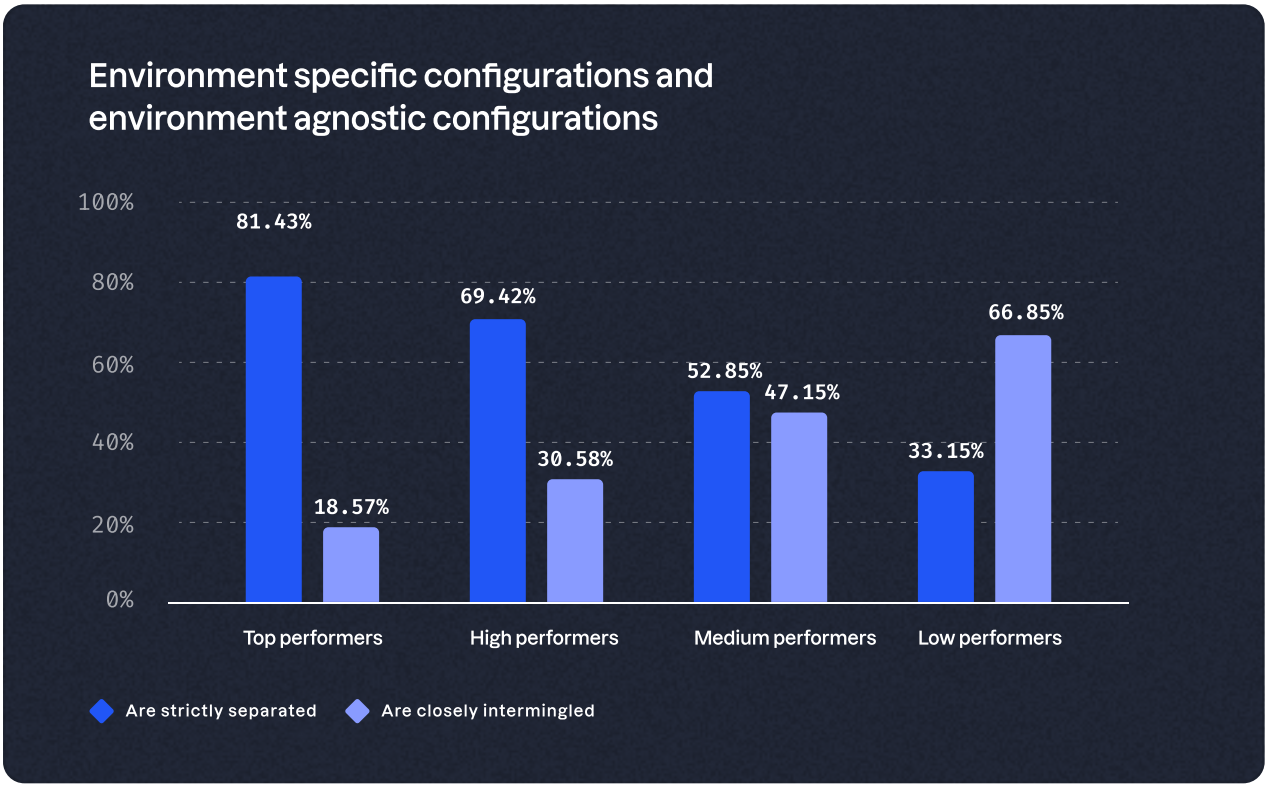
01 Elimination of orphan infrastructure:
It's easier to spot unused resources when most infrastructure provisioned is attached to a workload.

02 Better visibility:
Teams can more easily identify and resolve issues with a complete view of the entire application lifecycle, and its dependent infrastructure from development to deployment.

03 Improved compliance:
Organizations can better comply with regulations and standards, and keep infrastructure sprawl to a minimum.

Separation of environment-specific and environment-agnostic configuration

Separation of environment-specific and environment-agnostic configs distinguishes top and low performing teams



Among the top performing teams, 81.4% report environment-specific configurations are strictly separate from environment-agnostic configurations. Among low performing teams only 33.1% follow this best practice, which creates another key differentiator.

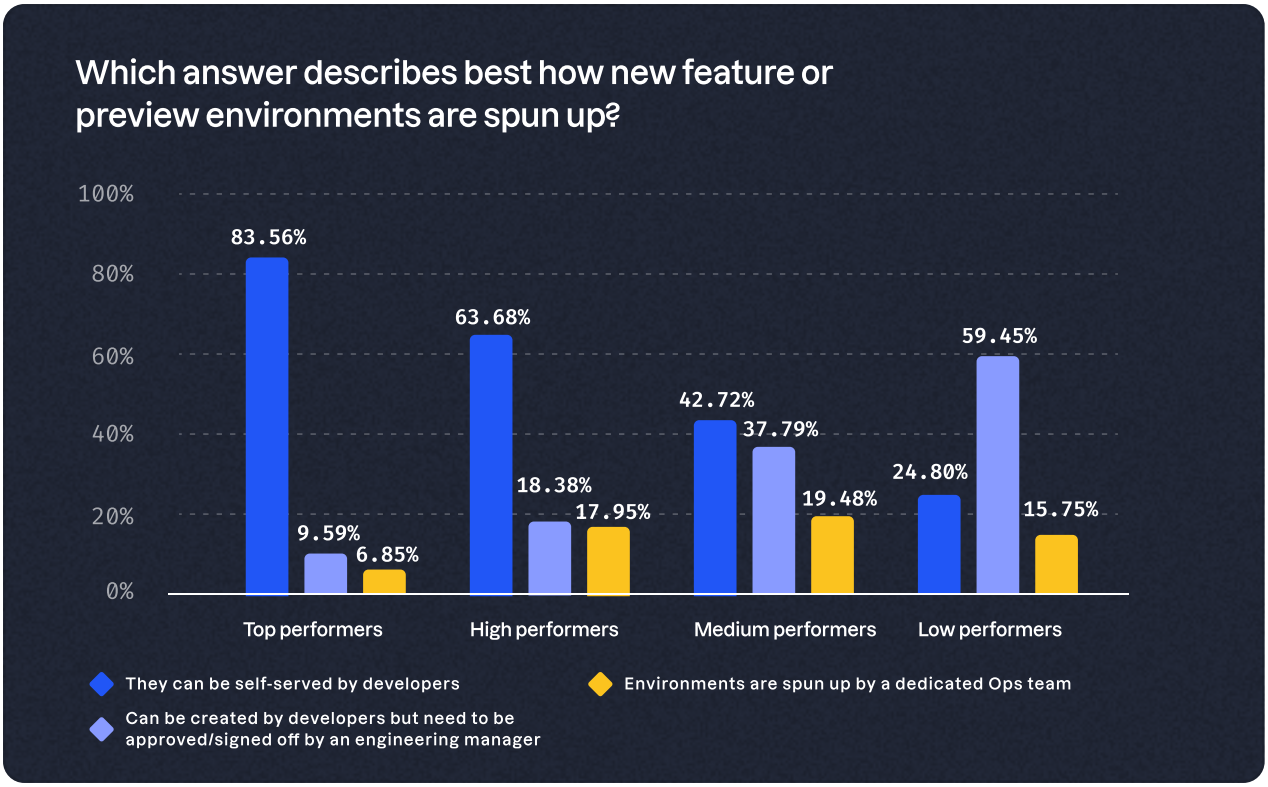
Degree of self-service

Application configuration

Developer self-service is a key indicator of high performance, good team culture, and employee satisfaction. This is because teams with a high degree of self-service can eliminate key person dependencies, waiting times, and hence frustration for both Ops (no more ticket ops) and developers (no bottlenecks). This is the essence of DevOps, as it accurately represents how close an organization is to true “you build it, you run it”.

Creation of new feature or preview environments

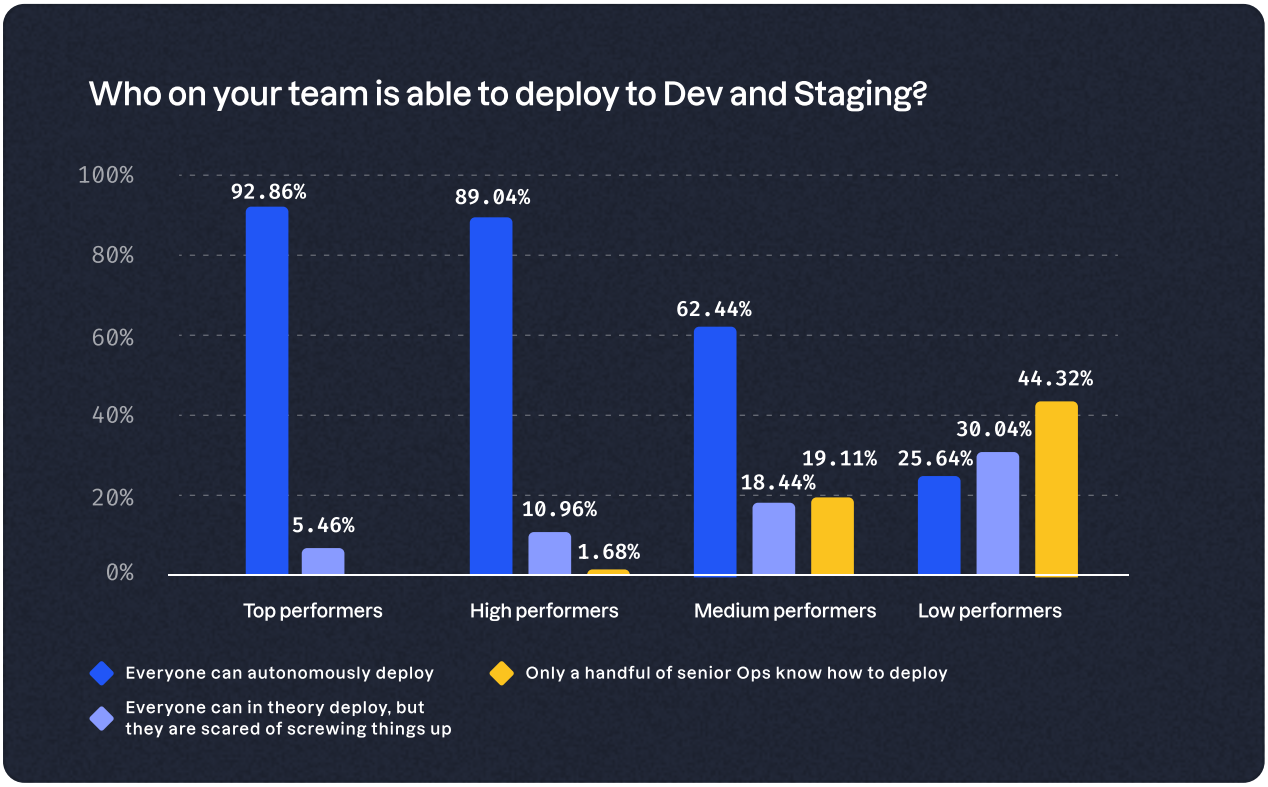
Developers in top performing setups can create new feature or preview environments on their own



The ability to spin up new features and preview environments on demand is a excellent proxy for developer self-service. In 83.6% of top performing teams, developers are able to create preview environments on the fly. For medium performing teams, already less than half (42.7%) of developers have this degree of autonomy. Close to 40% depend on approval from an engineering manager and 19.5% have to wait for Ops to provision it for them. Among low performing teams only 24.8% are able to create new environments on their own.

Deployment to development and staging

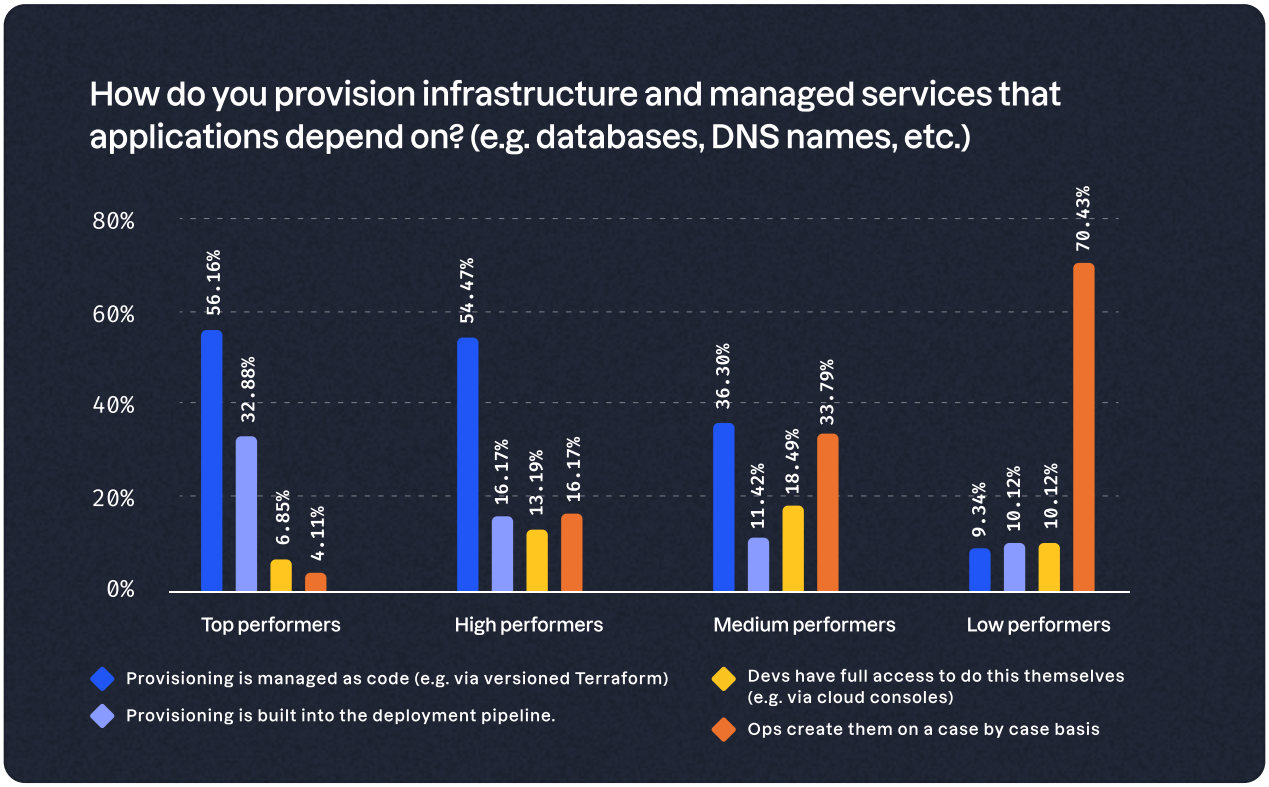
Reliance on Ops to deploy features might indicate lower performance



We see a similar picture when looking at a developer's ability to deploy to dev and staging environments on their own. Close to 90% of top performing teams feel confident deploying independently. Around 40% of medium performing teams struggle, either fearful of screwing things up or because they are dependent on Ops to deploy (19.1%). Among low performing teams, only 25.6% are able and confident enough to deploy. The rest are again too scared to do so, or are dependent on Ops.

Provisioning infrastructure and managed services

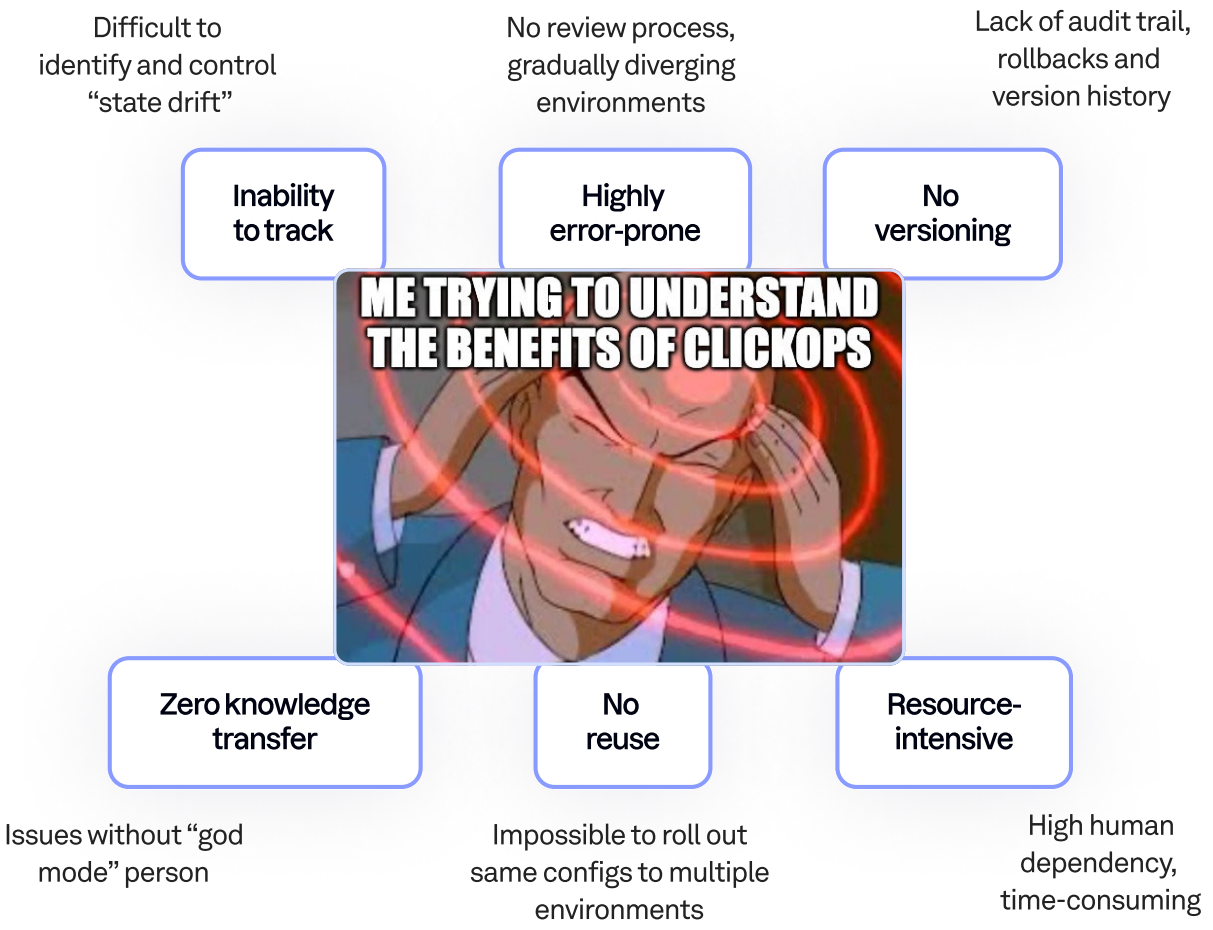
Low performing teams disproportionately rely on Ops to provision on a case-by-case basis



Key person dependencies on Ops are not only a big issue when it comes to environment creation, but for infrastructure provisioning too.

Among top performing teams there are two main solutions: The preferred option (56.2%) use an Infrastructure as Code (IaC) solution such as Terraform, while the main alternative (32.9%) is to have infrastructure provisioning baked into their deployment pipeline.

A less standardized and compliant approach is to allow developers to provision infrastructure themselves via cloud consoles. At first glance this might feel like a good indicator of developer self-service, but in reality is highly risky and error-prone.

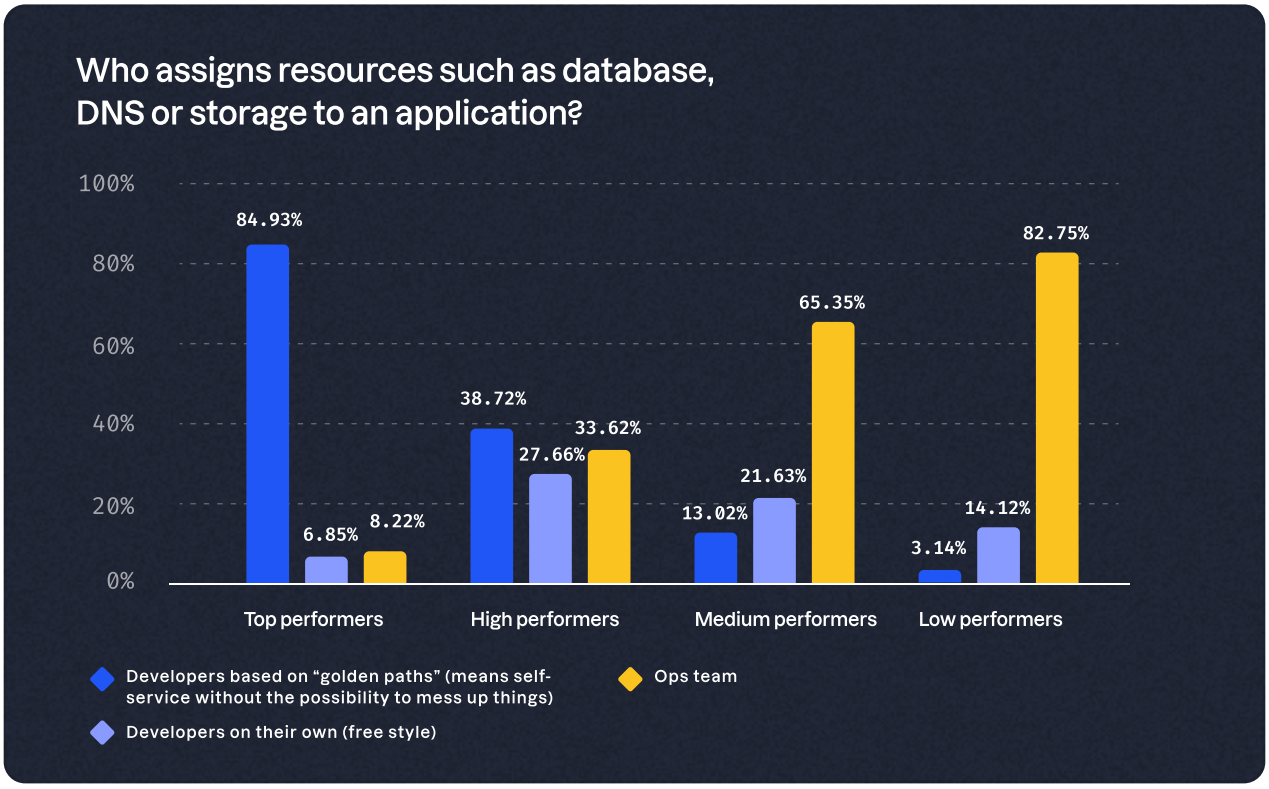


Cloud consoles, portals, and what's commonly defined as a click-ops approach all have several downsides. Teams working in a click-ops setup are slower and lack reusability, since they are unable to roll out the same configs to multiple environments. Audit and rollbacks become very difficult due to the lack of config versioning. And key person dependency risk is highest, with a "god mode" admin often the only one who can understand the setup and replicate it[8].

[8] Sören Martius "Lessons learned from 100s of Infrastructure as Code (IaC) setups" (Platformengineering.org) https://platformengineering.org/talks-library/infrastructure-as-code-setups

Assignment of resources

Low performing teams still overwhelmingly rely on Ops to assign resources to apps



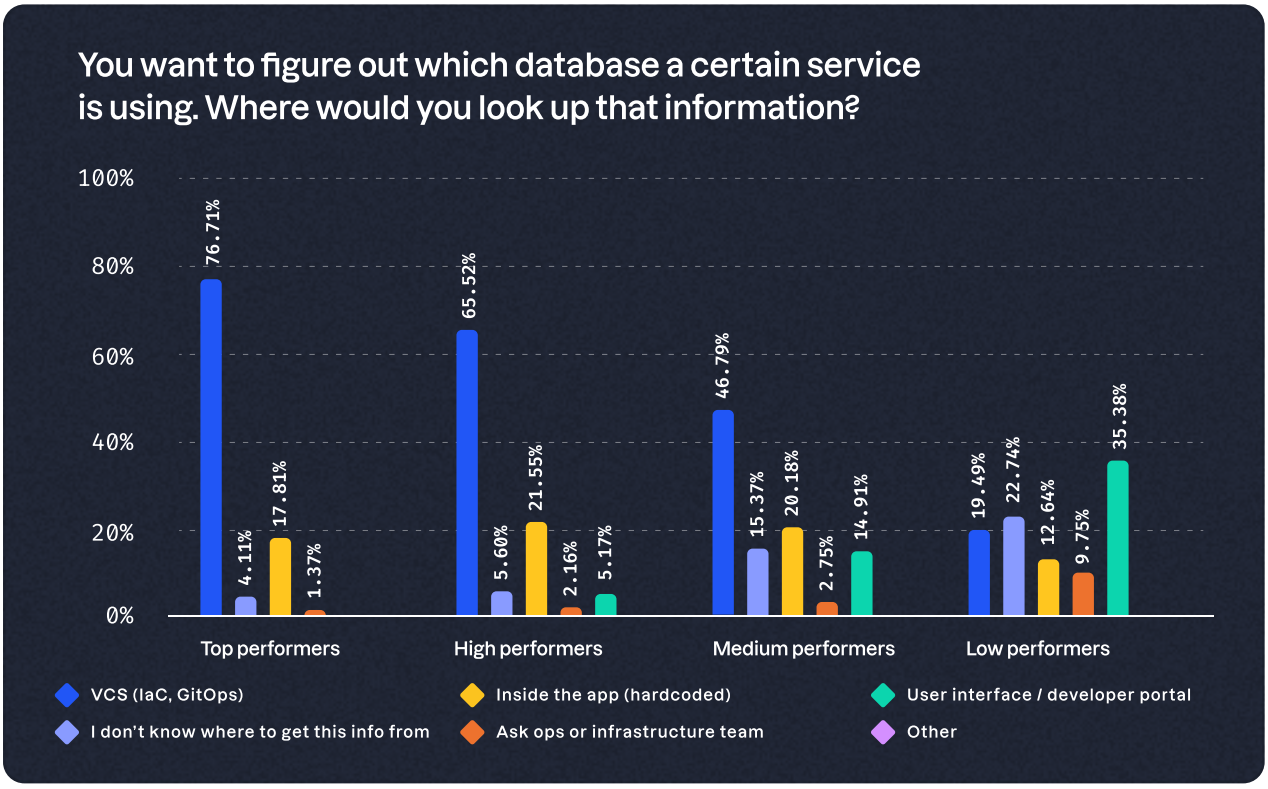
To be able to run, every service is dependent on a number of resources such as databases, DNS, storage, or similar. It's often the case that teams want to connect the same resource to different services across all environments (or vice versa). For example, you wouldn't want to run your eCommerce app in a testing environment with your real customer database, which is also used for production.

How resources are assigned to an app varies a lot across the segments. Close to 85% of top performing teams enable their developers to self-serve such resources based on golden paths. This means they likely have an Internal Developer Platform (IDP) or similar, and have designed an easy way for engineers to self-serve resources while minimizing risk of screwing things up. This keeps cognitive load low and helps create a great developer experience (DevEX).

The alternatives are for developers to assign resources to apps in a freestyle way (this happens most prominently amongst the high performing teams at 27.7%). Or once again to depend on Ops which leads to Ops bottlenecks where developers write tickets and then simply wait. While 33.6% of the high performing teams are still in this situation, it's worryingly prevalent for both medium (65.3%) and low performing teams (82.7%).

Finding information

Low performing teams disproportionately rely on Ops to find information

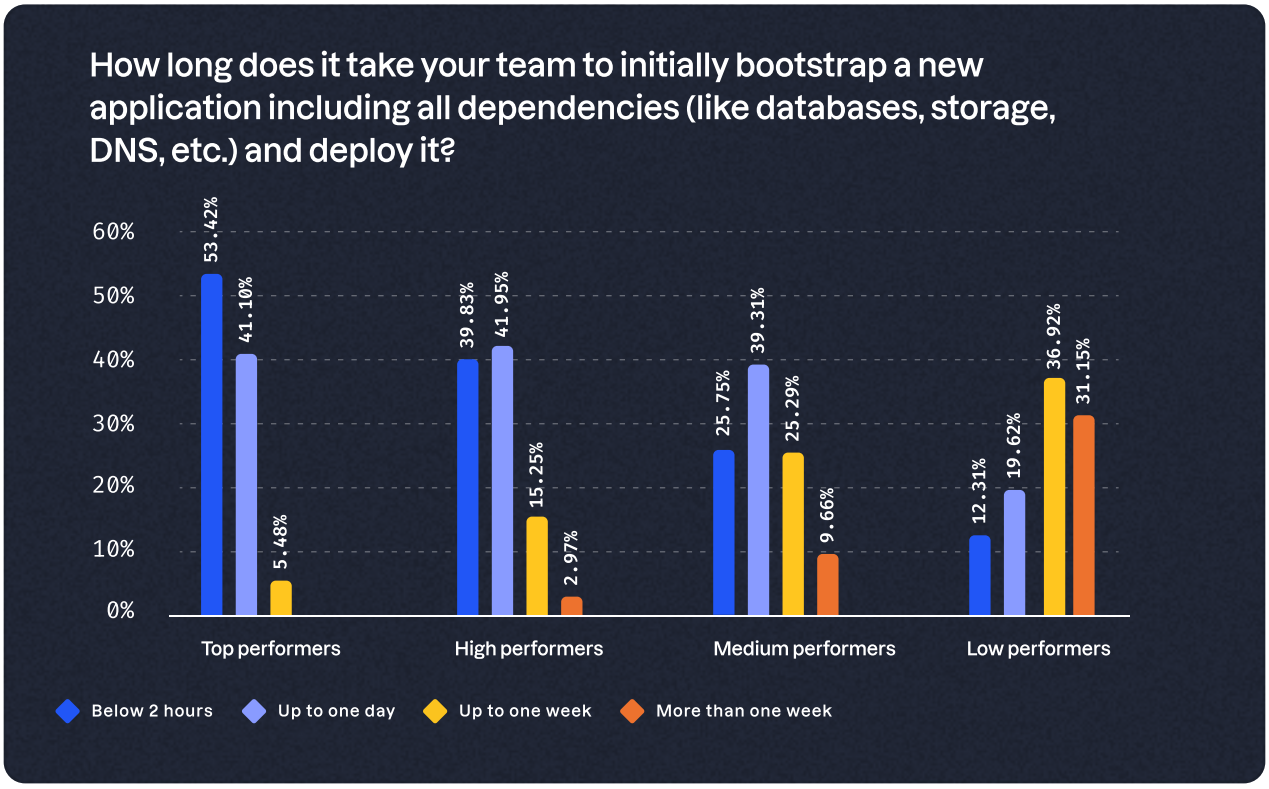


Overseeing all of these dependencies between apps and assigned resources can be challenging, especially in larger organizations with more complex setups. We wanted to know where engineers go to figure things out, for example, which database a particular service uses. The differences between segments are again telling.

Top performing teams go straight to their version control system (VCS) (76.7%), or use a service portal (17.81%). High performing teams are quite similar, with 65.52% using IaC or GitOps to figure out dependencies, and 21.5% going through a developer portal. On the other hand, low-performing teams need to ask their Ops or infrastructure team (35.4%) or look it up inside the app due to hardcoded connections (22.7%).

Bootstrapping an application

Low performing teams disproportionately rely on Ops to find information



A final indicator of the degree of self-service across organizations is how long it takes an engineering team to bootstrap a new app, including all dependencies such as databases, storage, DNS, etc—and deploy it.

The answers varied between less than two hours to more than one week. 53.4% of top performing teams can achieve this in less than two hours, while 39.8% can achieve this within one day. This is in contrast to low performing teams, who need up to one week (36.9%) or even longer (31.1%) simply to spin up a new app.

Key findings

Internal Developer Platforms correlate to DevOps success

The correlation between the degree of developer self-service, config management best practices, and how well teams perform based on DORA metrics becomes clear upon closer look. Reducing lead time and increasing deployment frequency require a setup free from bottlenecks caused by excessive ticket ops. And while MTTR largely remains an infrastructure and SRE topic, change failure rate is a good indication of the quality of the code that has been deployed, and how well app and infrastructure configs are managed.

> **Simple things should be simple, complex things should be possible.**
>
> Alan Kay

To find out how DevOps tasks are managed, we asked respondents to choose which of the following three answers best describes their current state:

01 Developer self-service is enabled by tooling like an Internal Developer Platform (IDP), built by a platform team for the rest of the engineering organization.

02 Developers are expected to perform most DevOps tasks on their own. Many of them are overwhelmed and senior engineers are constantly busy helping less experienced developers.

03 We have a dedicated Ops engineer/team that executes DevOps tasks. App developers are strictly separated and dependent on Ops.

04 We use a PaaS-like solution which is supposed to enable developer self-service, but is hard to scale and comes with a lot of restrictions (e.g. removes context and doesn't allow in-depth config).

A stunning 93.15% of top performing teams report using internal tooling. The same applies to 38.36% of high performing teams, in contrast to only 4.60% of medium and 1.88% of the low performing teams who did the same.

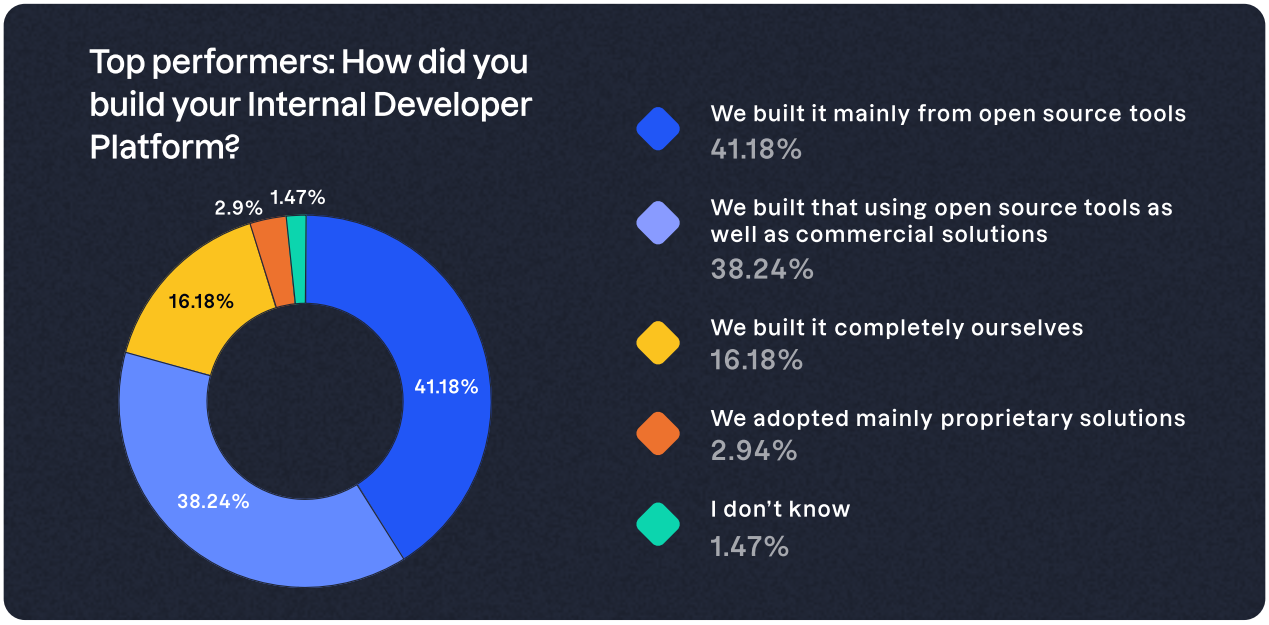
If we focus on the medium performing teams, two patterns emerge. The first is that while only a very low percentage of this segment have an IDP in place (4.60%), almost half (45.52%) operate in a broken DevOps setup. Developers are expected to manage DevOps tasks independently, but are completely overwhelmed and dependent on senior engineers.

The other pattern is that many medium performing teams (42.30%) still have a dedicated Ops team to execute DevOps tasks. This way, developers are not overwhelmed with DevOps tasks, but can't self-serve either. In reality this often means that developers are blocked by a ticket ops process that causes frustration, and results in a toxic relationship between developers and Ops.

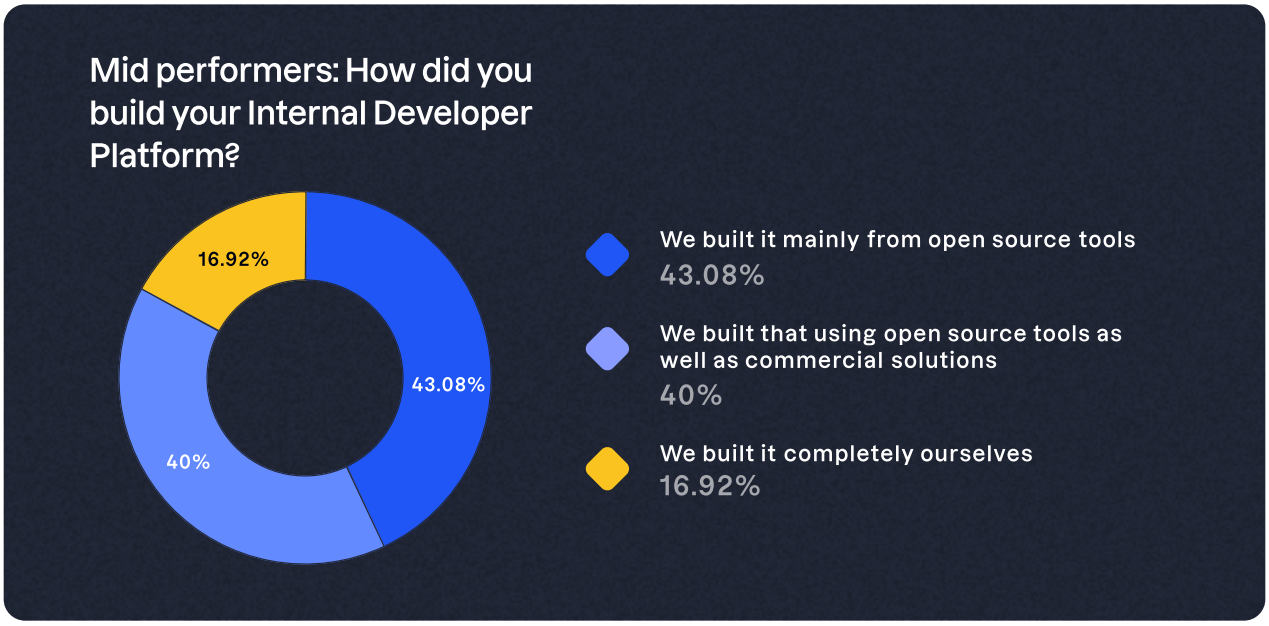
Low performing teams show a similar picture, but with a larger share relying on dedicated Ops teams for DevOps tasks (53.76%)

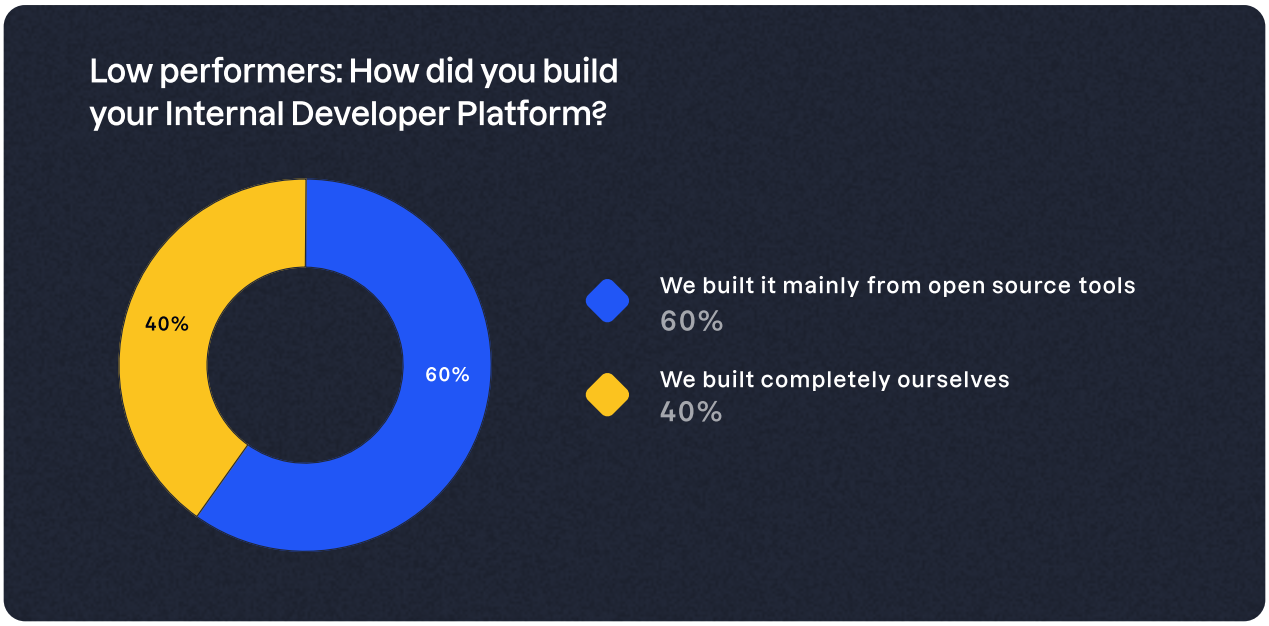
Also remarkable is that a “Heroku-like” experience (i.e. using a PaaS) is still a widely used synonym for successfully enabling developer self-service. However, top performing teams don’t use PaaS offerings at all and even among the remaining segments, only 6% to 8% do so.

So how do teams go about building IDPs?



For the top performing teams there is a near-even split between building IDPs with open-source tools (41.8%) and open-source and commercial tools (38.8%). The gap here is still very small, and it’s too early to reach a consensus.





What's interesting to see for medium and low performing teams, the lower the performance, the higher the share of teams who tried building an IDP entirely by themselves. The takeaway couldn't be more apparent; don't try to build everything from scratch and reinvent the wheel.

What is an Internal Developer Platform?

While there is industry consensus that top performing organizations use IDPs to build golden paths for their developers, there is still some confusion about what an IDP actually is.

> **An Internal Developer Platform (IDP) is the sum of all the tech and tools that a platform engineering team binds together to pave golden paths for developers. IDPs lower cognitive load across the engineering organization and enable developer self-service, without abstracting away context from developers or making the underlying tech inaccessible. Well designed IDPs follow a Platform as a Product approach, where a platform team builds, maintains and continuously improves the IDP, following product management principles and best practices.[9]**
>
>  Kaspar von Grünberg CEO at Humanitec

[9] Kaspar von Grünberg: "What is an Internal Developer Platform" (Humanitec, July 29, 2021) https://humanitec.com/blog/what-is-an-internal-developer-platform

Another common misconception is that IDPs are the same as developer portals. In a nutshell:

> **Internal developer portals serve as the interface through which developers can discover and access internal developer platform capabilities."[10] - Manjunath Bhat, Research VP, Software Engineering Practice, Gartner**

Manjunath Bhat Research VP, Software Engineering Practice, Gartner

Platforms enable developer self-service and improve developer experience

Let's next look at how platforms enable developer self-service and a great developer experience (DevEx), and how this differentiates top and low performing teams

According to Gartner, software engineering leaders should "Improve developer experience by providing self-service, internal developer platforms — to reduce cognitive load and context switching, and to abstract away underlying complexity."[11] Improving developer satisfaction of course goes hand in hand with increasing productivity. Much like the broader employee experience, a great DevEx frees up time for developers which allows them to focus on solving complex problems. This in turn translates into higher performance and cost savings. And it's why accelerating development with the right approach to self-service is no longer a nice-to-have; it's a business imperative. In fact Gartner predicts that by 2025, 75% of organizations with platform engineering teams will provide self-service, internal developer platforms to improve developer experience and accelerate product innovation[12]. "And that by 2026, 80% of software engineering organizations will establish platform teams as internal providers of reusable services, components and tools for application delivery".[13] — **Gartner**

There's no doubt that developer self-service is the engineering team's dream scenario. When done right, it can help relieve frustration for both Ops and developers by enabling:

- Autonomous deployment with no reliance on Ops
- True "you build it, you run it" ability
- Strengthened security and improved compliance
- Shorter time from commit to deploy
- Ops to shift from tasks to strategy

[10] Manjunath Bhat: "A Software Engineering Leader's Guide to Improving Developer Experience" (Gartner)" https://www.gartner.com/document/4017457

[11] Manjunath Bhat, Arun Chandrasekaran, & Stephen White: "Cool Vendors in Platform Engineering for Improving Developer Experience" (Gartner, 6 October 2022) https://humanitec.com/whitepapers/gartner-cool-vendors-platform-engineering

[12] Manjunath Bhat, Arun Chandrasekaran, & Stephen White: "Cool Vendors in Platform Engineering for Improving Developer Experience" (Gartner, 6 October 2022) https://humanitec.com/whitepapers/gartner-cool-vendors-platform-engineering

[13] Gartner: "Top Strategic Technology Trends for 2023: Platform Engineering (17 October 2022) https://www.gartner.com/en/information-technology/insights/top-technology-trends

> **Self-service improves developer experience because it reduces process inefficiencies and, in many cases, eliminates unnecessary processes. For example, the need to raise a ticket to create a development database or stand up a test environment can impede developer productivity and disrupt their state of flow.** [14]
>
> Gartner

The issue is that in many organizations, developer self-service simply means taking a shift left approach in handing DevOps tasks to developers, which are usually executed quite poorly. As our research shows, a greater degree of developer self-service enabled by an IDP provides the capability to spin up new environments, deploy, roll back, and make changes in the architecture. All without the need to rely on Ops. While this sounds like developer heaven, it's important to focus on improving the Ops and developer relationship by ensuring a separation of concerns, which we'll discuss later in more detail. Essentially, both teams should be able to work closely together while having differentiated roles.

> **IDPs are about making it easier for developers to build and deliver software, while not abstracting away the useful and differentiated capabilities of the underlying core services.** [15]
>
> Gartner

McKinsey highlights that "a great DevEx can be enabled by a platform that serves developers' needs across several elements".[16] From our research, top performing teams can self-serve resources like databases, DNS, and storage based on golden paths. With platform standardization, you also have guard rails in place that ensure your entire setup still works and that deployment doesn't completely fail, should your developers make a mistake.

To sum up, IDPs enable the creation of golden paths which help balance developer cognitive load. They help ensure developers still have access to the underlying tech they need, and the freedom to to perform in-depth configs. Developers should be able to choose the right abstractions, and if you introduce an IDP, it shouldn't break their existing workflows. But by default, IDPs also shield away from this kind of complexity to help ensure developers are doing the right thing—and don't screw up. Enabling developer self-service allows you to implement a

[14] Manjunath Bhat: "A Software Engineering Leader's Guide to Improving Developer Experience" (Gartner) https://www.gartner.com/document/4017457

[15] Manjunath Bhat, Arun Chandrasekaran, & Stephen White: "Cool Vendors in Platform Engineering for Improving Developer Experience" (Gartner, 6 October 2022) https://humanitec.com/whitepapers/gartner-cool-vendors-platform-engineering

[16] Thomas Delaet, Arun Gundurao, Ling Lau, Stephan Schneider, and Lars Schor: "Why your IT organization should prioritize developer experience" (McKinsey, June 6, 2022) https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-forward/why-your-it-organization-should-prioritize-developer-experience

separation of concerns into the entire organization without putting developers and Ops back into their previous silos, thus helping you reach a higher stage in your DevOps evolution.

Dynamic Configuration Management is a best practice to prevent config drift

As we have seen, top performing teams manage their app config across the entire organization in a standardized way. Our study also shows that these teams handle app configs the same way as infrastructure dependencies, and manage to separate environment-specific from environment-agnostic configuration. Top performing teams are also faster in creating new environments. And they show a higher degree of self-service regarding deployments to ephemeral feature or staging environments, the provisioning of infrastructure, and how to assign this infrastructure based on golden paths.

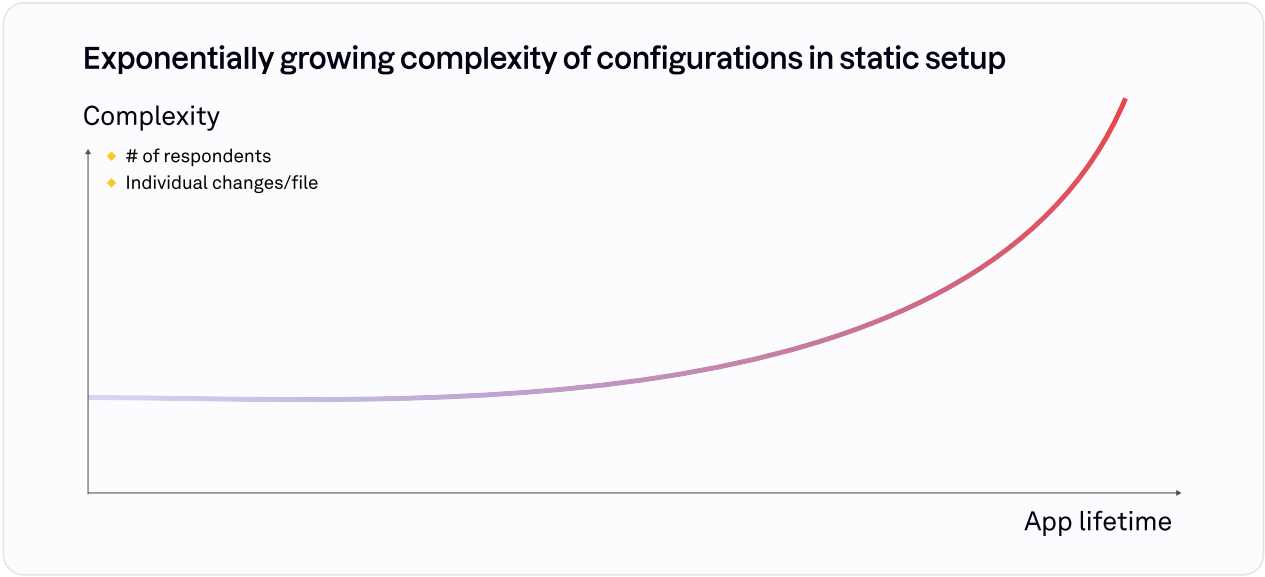
In summary, these teams all follow a methodology commonly known as Dynamic Configuration Management (DCM), which is "used to structure the configuration of compute workloads. Developers create workload specifications, describing everything their workloads need to run successfully."[18] In practice, this means that a workload (or service), and in sum applications, depend on certain external resources and infrastructure, such as databases, DNS, or storage. Workloads also run in different kinds of contexts (e.g. environments from dev to production) where they depend on other resources. Workload specifications are then used "to dynamically create the configuration, to deploy the workload in a specific environment. With DCM, developers do not need to define or maintain any environment-specific configuration for their workloads."[19]

Why static Internal Developer Platforms cause config drift

In contrast to a dynamic Internal Developer Platform (IDP) that enables DCM, most platforms we see in the market today enable developers to deploy an updated image from stage to stage. But only if the app infrastructure remains the same. This is because of the static config files that are manually scripted against a set of static environments and infrastructure. In this setup, static IDPs are prone to breaking or result in overhead, should teams need to perform tasks such as rollbacks, change configs, or refactoring. In addition, the majority of tooling that's often built into static IDPs can solve single pain points, such as environment as a service. They are also built into self-scripted workflows, which tends to cause shadow ops or a greater reliance on Ops. The issue is that most of these tools cannot address the core issue plagued by most delivery setups: A static way to manage both app and infrastructure configs.

[18] Chris Stephenson: "What is Dynamic Configuration Management?" (Humanitec, February 7, 2023) https://humanitec.com/blog/what-is-dynamic-configuration-management
[19] Chris Stephenson: "What is Dynamic Configuration Management?" (Humanitec, February 7, 2023) https://humanitec.com/blog/what-is-dynamic-configuration-management

Taking such a non-standardized approach can also result in an exponential increase in complexity and very often, config drift.



Config drift happens when new workloads are created by copying the structure of a pre-existing workload. Teams customize these structures to meet their requirements which over time, culminates in subtle configuration differences from workload to workload. The end result is added complexity when performing bulk updates across workloads.

Separation of concerns in a dynamic setup

In a dynamic IDP setup, there is a clear separation of concerns. Although DCM does not itself organize an engineering team, it offers a structure within which teams can work better together.

- **From a developer perspective,** a dynamic IDP setup enables workload-centric development.[20] With a single workload specification, workloads can be deployed simultaneously to all environments, with their dependent configuration and resources. And since only one specification needs to be created per workload no matter the number of environments, the risk of config drift is eliminated.

- **For platform engineers,** since all infrastructure is automatically generated from approved templates and modules, a dynamic IDP setup enables the delivery of a consistent and predictable infrastructure. It also eliminates the need for ticket ops and overburdening developers with infrastructure responsibilities, by enabling

[20] Susa Tünker "Why we advocate for workload-centric over infrastructure-centric development" (Score, November 22, 2023) https://score.dev/blog/workload-centric-over-infrastructure-centric-development

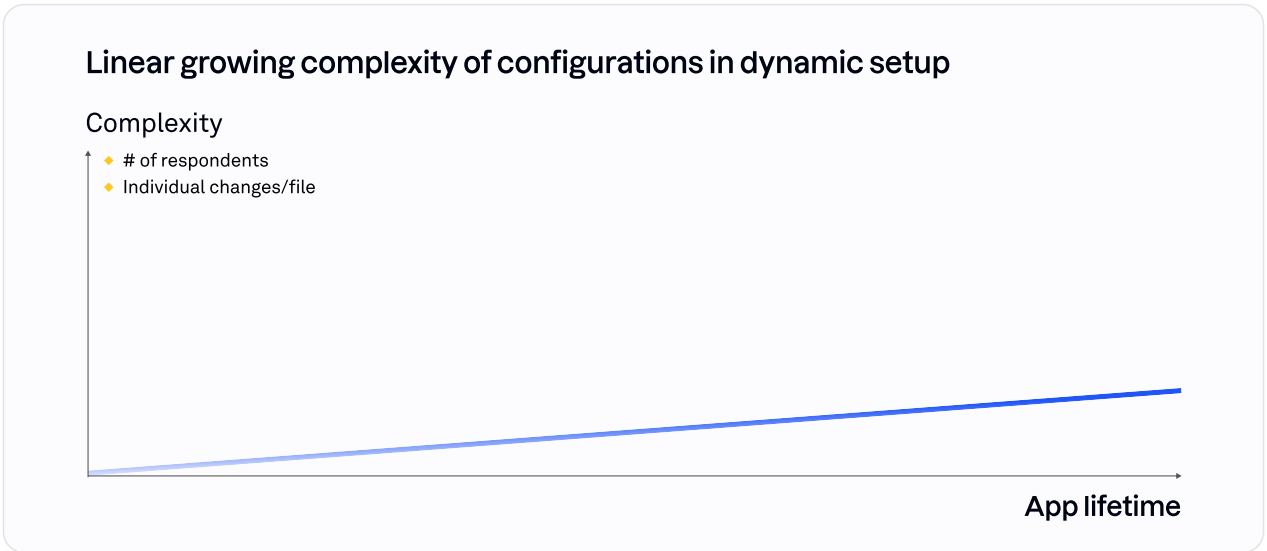
> engineers to build a self-service model with golden paths and sensible defaults. And when it comes to keeping everything current and updated, a dedicated platform team can maintain and update the templates.

In practice it would look like this: Let's say a workload needs a database. A static setup would require some interaction between the developer and the DBA team. The developer would manage connection string custody, which would likely contain credentials with secret information. Without this interaction the DBA team can't know if a developer needs a new database. Whereas in a DCM setup, when a new database is needed, automated tooling can notify the DBA team. Alternatively the entire process can be automated as and when required by workloads, to provision databases at deployment.

Dynamic Configuration Management drives standardization by design

Another huge advantage of implementing a dynamic IDP setup is that it enables platform engineering teams to enforce standardization by design. This approach aims to standardize and automate infrastructure provisioning and config management.

A dynamic IDP setup removes the need for developers to handle hundreds of config files, instead enabling them to use just one file per service. The final config files and manifests are then created based on the deployment context. Score (rated 7k+ stars on GitHub) is one example of an open source workload specification, which in a very short period of time has experienced massive uptake and interest amongst the platform engineering community. So much so that it is now amongst the top 1% of largest repositories on GitHub (as measured by stars).



The impact is stunning: compared to static setups, the number of config files is reduced by up to 95%. Through enforced standardization, the complexity grows linearly rather than exponentially.

More key benefits of Dynamic Configuration Management

In summary, DCM and standardization by design helps improve promotion between environments, drives a better DevEx, and ultimately, enables engineering organizations to foster a better way of working. Key benefits include:

01 Improved developer productivity: Through self-service enablement and separation of concerns, developers can focus on delivering their apps while the platform team manages infrastructure and configuration.

02 Less developer cognitive load: Thanks to the tightly scoped workload specification featuring simple syntax that's easy to read and learn.

03 Increased velocity: Developers have more time to prioritize app delivery, while the time to provision and configure infrastructure can be greatly reduced which improves time-to-market.

04 Reduced infrastructure sprawl and config drift: Standardizing and automating infrastructure provisioning and config management makes it easier to enforce standards, minimize risk of infrastructure sprawl, and reduce config drift.

05 Greater compliance with established standards: Thanks to consistency and predictability across all infrastructure and environments.

While there are many ways to implement DCM in your IDP setup, the most efficient route is by adopting a developer-centric and platform-agnostic workload specification like Score. When paired with a Platform Orchestrator like Humanitec, it's possible to build your own dynamic IDP in a very short amount of time.

Platforms drive innovation in times of uncertainty

This DevOps Benchmarking Study was published in early 2023, a time of much uncertainty caused by the macroeconomic situation. Whether your organization's priority is digital transformation, new revenue streams, or responding faster to change, never has there been a greater need to drive innovation and stay competitive.

To achieve this, enabling a great DevEx and increasing developer productivity will be a key competitive advantage. This means recognizing the pivotal role platform engineers can play in building IDPs that can help support this, and ultimately contribute to your organization's overarching goals.

> **The companies that invest in shorter innovation cycles, faster TTM, and shipping new features on time will outperform their competitors. It's up to you to demonstrate how an IDP will get your organization there: by reducing cognitive load on developers, enabling them to self-serve what they need to run their apps and services, driving standardization by design, and improving the developer experience.[21]**
>
>  **Lee Ditiangkin** Platform Product Manager

Due to frozen budgets, stagnation, high inflation rates, and impending recession in some countries, the right efforts are needed to build a business case for your platform engineering initiatives. This means understanding the top priorities that senior execs are concerned with, and proving what business value a platform can offer. According to Lee Ditiangkin, Product Manager at IBM,[22] several key strategic goals should be included with an overview of how platform engineering can support organizations to achieve them:

↪ Accelerate digital transformation

Moving to a cloud-native setup can be incredibly complex, especially if your organization has legacy systems to support. While Kubernetes adoption is a

[21] Lee Ditiangkin: "Step One to Successfully Building Your Platform: Building It Together" (InfoQ, March 7, 2023) https://www.infoq.com/minibooks/platform-engineering-guide

[22] Lee Ditiangkin: "Step One to Successfully Building Your Platform: Building It Together" (InfoQ, March 7, 2023) https://www.infoq.com/minibooks/platform-engineering-guide

great solution for smaller teams with new projects, at scale this is a different story. The additional developer cognitive load (caused by the need to adopt new technologies) can stifle productivity. And there is a common misconception that Kubernetes is a platform when in reality, it's just the foundation. In this instance, building an IDP can provide abstraction layers on top of Kubernetes, and help drive digital transformation success.

↪ Shorten time to market (TTM)

Including tangible business metrics such as TTM is integral to your IDP argument. For example, you could say building an IDP that enables developer self-service can shorten TTM by X%, by increasing developer productivity and reducing time waiting on Ops. Without an IDP developers may also struggle with executing architectural application changes which hinders productivity for them and Ops, rendering your organization unable to react fast to competitors.

↪ Minimize security risks

There's no doubt about the impact security and data breaches can have on brand reputation and loss of market share. An IDP can help mitigate those risks, by driving standardization that enforces security best practices. Further, the integration of smart security tools will better place your organization to detect security vulnerabilities in a fast, automated way.

↪ Increase operational efficiency

Without an IDP your organization could face unnecessary outages or downtime due to half-scripted manual workflows. Not only do fewer outages equate to less late-night calls and interruptions, they translate to better customer retention and high availability.

↪ Reduce cloud spend

An IDP can make it easier to manage your setup more efficiently, enabling small changes such as the ability to detect and pause unused environments which can help lower costs. A vendor-neutral cloud platform also makes it easier to switch between different vendor offers, and capitalize on different incentives.

↪ Attract and retain top talent

IDPs can help with talent acquisition and retention by improving DevEx. According to Gartner[23], investing heavily in better DevEx is the best way to safeguard developers' creative work and the key to boosting productivity. This not only results in happier engineers, it also helps your organization to innovate faster.

Regardless of which approach you take to building your business case, it's essential to keep DevEx at the forefront of your strategy. In doing so, developers will be better able to navigate siloed systems, improve delivery speed, and increase process agility—all of which contributes to greater business value.

[23] Manjunath Bhat: "A Software Engineering Leader's Guide to Improving Developer Experience" (Gartner) https://www.gartner.com/document/4017457

Conclusion

When it comes to the adoption of platform engineering practices, advanced engineering organizations have been leading the way. However, as the discipline goes mainstream, we're now seeing this trickle down across the rest of the market.

Platform engineering finally enables true DevOps "you build it, you run it", removes cognitive load on developers, and kills ticket ops and waiting times. In a world where tools and setup complexity are ever-evolving, new stresses and strains are created daily for those building and deploying software. This is where platform engineering comes into its own. Its real meaning stands for a separation of concerns, and specialization that simply allows teams to focus on what they're good at. It can help define the lines between developers and Ops, drive productivity, and subsequently heal the relationship between teams; without putting developers and Ops back into their silos.

Through our research we found that:

01 IDPs boost DevOps success.

02 Platforms enable developer self-service and improve DevEx.

03 Dynamic Configuration Management (DCM) is the new hot thing.

04 IDPs drive innovation and contribute to overarching business goals.

Because of these potential capabilities, more and more organizations are building IDPs. For those considering this route, we highly recommend choosing DCM over a static approach to managing configurations. You should avoid building your platform from scratch, and instead aim to use a combination of open source and proprietary tools to save you from reinventing the wheel.

In the past, DevOps tended to focus on individual or team problems. It's now essential that platform engineering takes the broader company context into account. This presents an excellent opportunity to reduce time to market and deliver high-speed innovation cycles; ; ultimately, this discipline can help drive customer satisfaction, create real business value, and boost the bottom line.

There's no doubt 2022 was a huge year for platform engineering. But we predict even bigger things for 2023 and beyond. This includes more comprehensive case studies, with platform engineering initiatives already in high demand. We expect an increase in knowledge sharing for

DevOps and platform tooling, as well as blueprints and reference architectures within the community. And as awareness of the discipline expands, DevOps and platform roles will continue growing in popularity. We also expect DevOps titles to change and more accurately reflect evolving responsibilities; driven by a renewed industry focus to embrace platform engineering, and create the best DevEx possible.

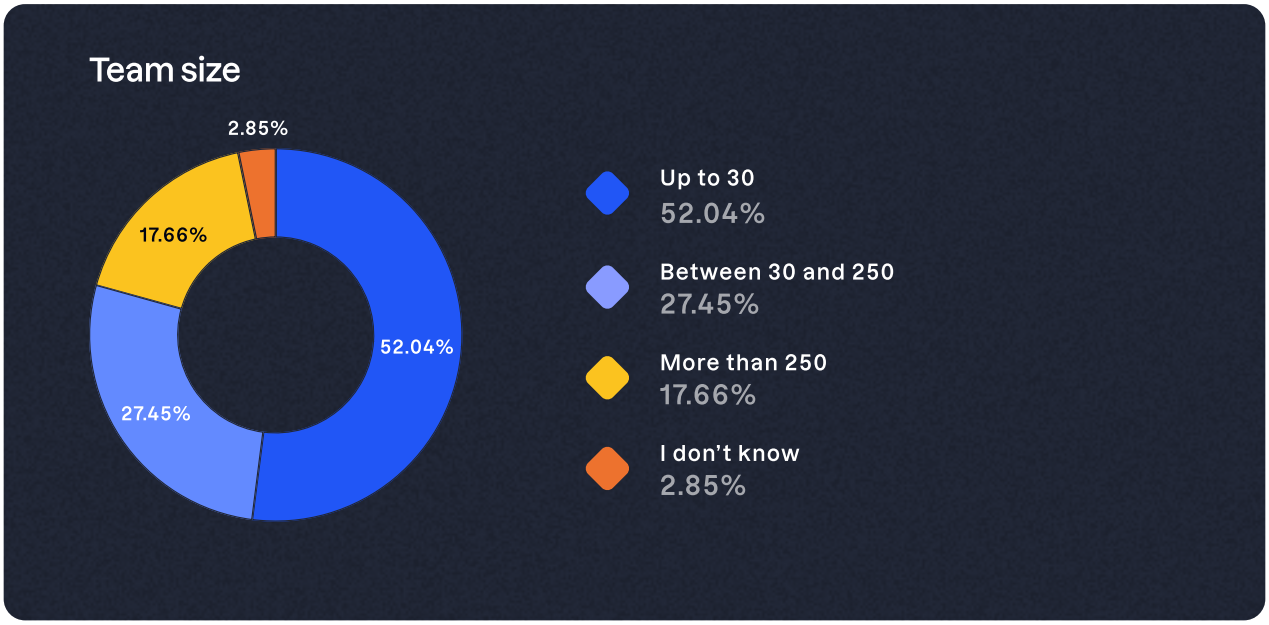
Stay tuned!

Appendix

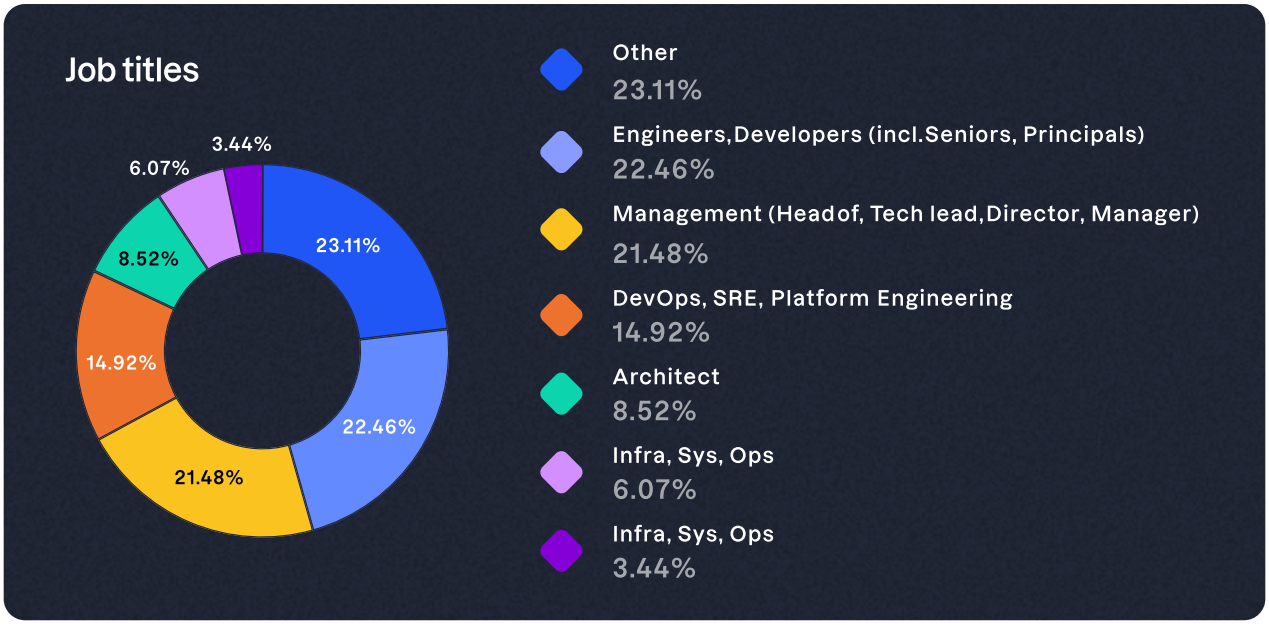
Number of respondents

n=1053

Team size

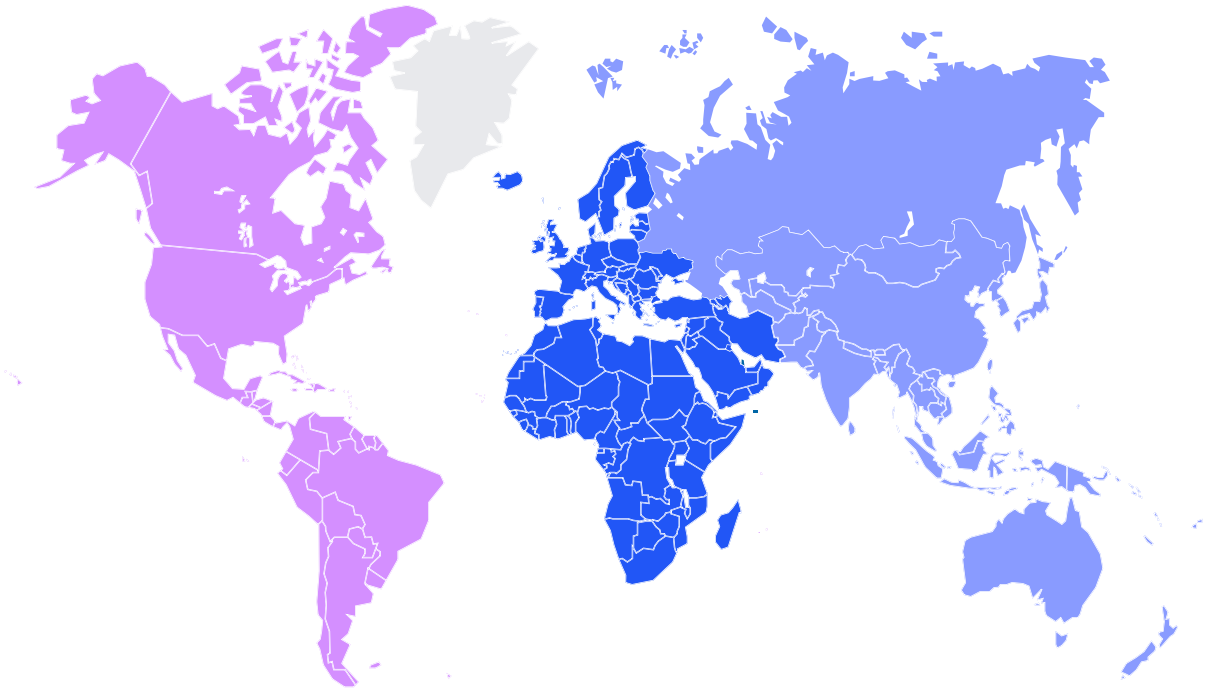


Job titles



Infra, Sys, Ops	7.89%
DevOps, SRE, Platform Engineering	19.40%
Architect	11.09%
Engineers, Developers (incl. Seniors, Principals)	29.21%
Management (Head of, Tech lead, Director, Manager)	27.93%
C-level (CTO, CEO, CXO)	4.48%

Regions



AMAERICAS 29.5%

EMEA 48.7%

APAC 21.9%

How DevOps tasks are managed	Deployment frequency						Lead time						MTTR				Change failure rate				
	On-demand	Several times per day	Weekly	Monthly	Few times per year	?	Mins	<1 day	1 day - 1 week	1 week - 1 month	>1 month	?	<1 hour	<1 day	1 day - 1 week	?	<15%	16-30%	31-45%	>45%	?
With an IDP	**47%**	**24%**	**18%**	**8%**	**3%**	**1%**	**35%**	**24%**	**29%**	**17%**	**14%**		**52%**	**41%**	**5%**	**2%**	**89%**	**7%**	**1%**		**4%**
Without an IDP	**29%**	**14%**	**31%**	**12%**	**9%**	**5%**	**19%**	**14%**	**29%**	**22%**	**10%**	**7%**	**32%**	**49%**	**10%**	**9%**	**76%**	**9%**	**3%**	**1%**	**12%**
Devs handle most DevOps tasks	34%	14%	28%	15%	8%	1%	20%	19%	28%	23%	9%	1%	31%	53%	13%	4%	80%	11%	3%	1%	5%
I don't know	28%	6%	26%	8%	16%	16%	16%	12%	26%	14%	12%	20%	18%	54%	10%	18%	70%	2%	2%	2%	24%
Dedicate Ops handles all tasks	28%	13%	33%	16%	9%	1%	14%	18%	28%	26%	10%	3%	36%	48%	12%	4%	81%	11%	2%	1%	5%
With a PaaS	28%	23%	35%	9%	2%	3%	25%	8%	34%	23%	8%	3%	42%	42%	6%	11%	72%	11%	5%		12%
Overall	**33%**	**16%**	**29%**	**13%**	**8%**	**1%**	**21%**	**20%**	**26%**	**22%**	**9%**	**3%**	**37%**	**48%**	**11%**	**4%**	**81%**	**10%**	**2%**	**1%**	**6%**

Contact details:
Humanitec GmbH
Wöhlertstraße 12-13, 10115 Berlin, Germany
Phone: +49 30 6293-8516

Humanitec Inc
228 East 45th Street, Suite 9E, New York, NY 10017

E-mail: info@humanitec.com
Website: humanitec.com

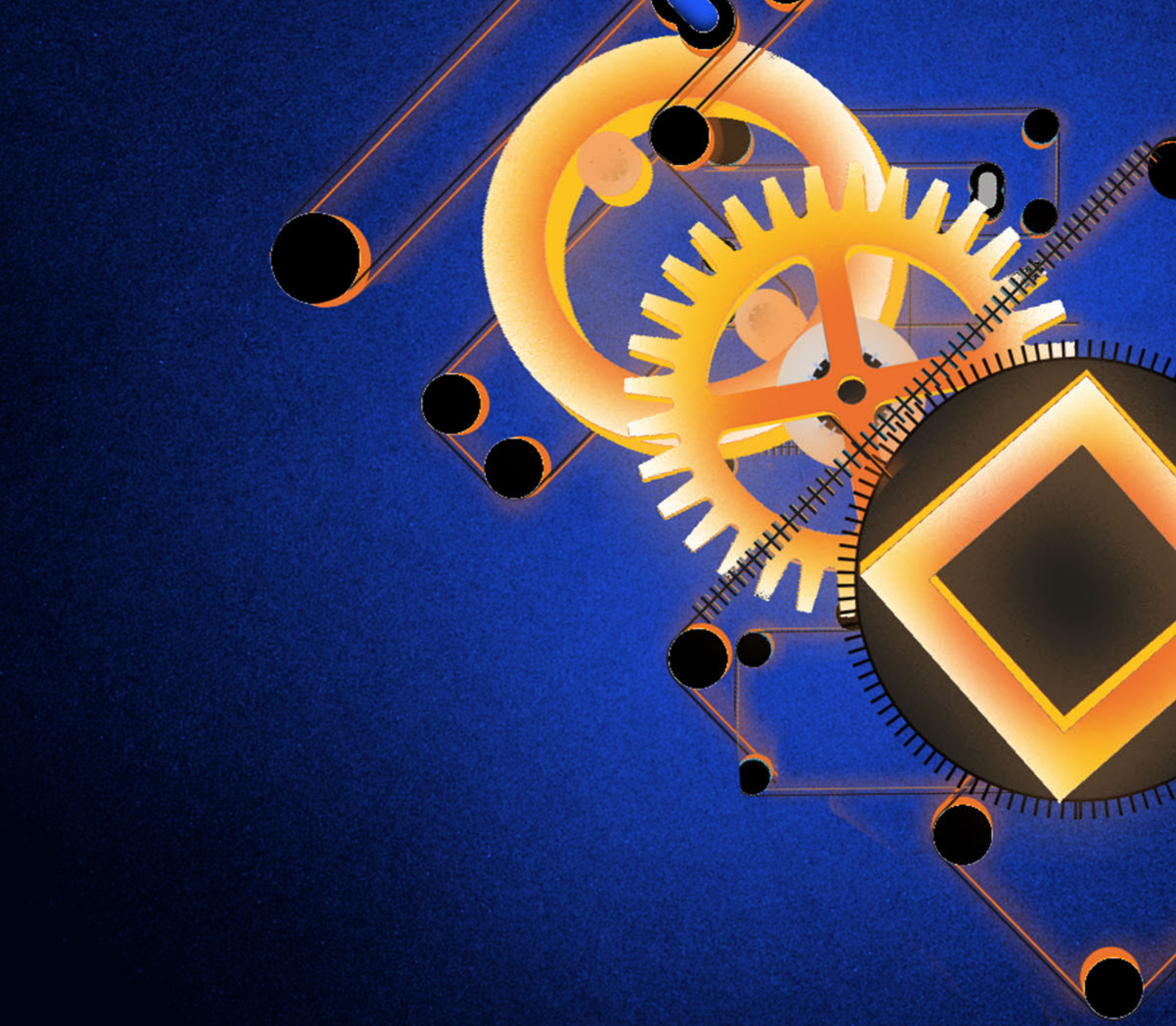
CEO: Kaspar von Grünberg
Registered at Amtsgericht Charlottenburg, Berlin: HRB 196818 B
VAT-ID according to §27a UStG: DE318212407
Responsible for the content of humanitec.com ref. § 55 II RStV: Kaspar von Grünberg

humanitec

Platform engineering is revolutionizing how enterprises build and run their cloud-native setups. Humanitec is at the core of this revolution, enabling teams to build Internal Developer Platforms (IDPs) and reach true developer self-service.

Humanitec's Platform Orchestrator is the engine at the heart of a dynamic IDP. It lets platform teams, from growing startups to enterprises, remove bottlenecks by letting them build golden paths for developers. With a dynamic Internal Developer Platform, developers self-serve the tech they need to deploy and operate their apps, driving productivity and velocity.

Before Humanitec, building dynamic IDPs was hard and very expensive. It required significant budgets, talent and time. Humanitec makes this process much easier and efficient, while keeping platform teams and developers flexible along the way.



DevOps Benchmarking Study 2023

humanitec

humanitec.com