2022

Kubernetes

Benchmarking Study

humanitec

Kubernetes Benchmarking Study 2022

Executive summary 3
Introduction 5
Overall results 6
- Implementation state of Kubernetes 7
- Technical challenges & best practices 14
- Team setup and cultural aspects 19

Platform strategies to nail Kubernetes implementation 22
Key takeaways 24
About the Data 25
Literature 26

Executive summary

Kubernetes (or K8s) adoption keeps picking up pace. According to the State of Cloud Native Development Report in 2020 alone it grew by 67%. In December 2021, the CNCF reported the number of developers using Kubernetes at 6.8 million. The same applies to containerization, which is the underlying trend that creates the foundation for Kubernetes adoption.

K8s has long been the de facto industry standard for its scalability, portability, self-healing and automation capabilities, just to mention a few. This often leads engineering teams to the (wrong) impression that software delivery and operations gets easier when adopting Kubernetes. This is usually followed by an uncomfortable awakening. The complexity of K8s is underestimated time and again. 42.3% of the organisations that implemented it expected K8s to be easy to operate, but 53% admit they vastly misjudged its complexity.

Many teams find out the hard way how complicated it becomes to operate and maintain Kubernetes clusters efficiently. And not only that, onboarding new developers to the K8s setup becomes extremely tricky too. This report shows that 57% of low performing organisations need a minimum of one month to onboard a single developer to K8s, while 81% of the top performers take only one day up to one week. The results also illustrate how many teams get stuck midway through their Kubernetes adoption journey. More than 52% report their Kubernetes implementation is not fully completed yet, which leads to friction and parallel maintenance.

There's a whole set of technical implementation details you need to get right when implementing Kubernetes. It starts with having a proper documentation of the setup in place. The relationship to the CI/CD workflow and pipeline must be thoroughly designed. The way configurations are managed turns out to be one main differentiator between low performers, who apply direct changes via kubectl, and high performing teams, who mainly use Helm. Secrets management and security are another key determining factor.

A successful Kubernetes implementation also heavily depends on cultural change management and on shielding developers from its complexity. In top performing organizations, a high percentage of developers are exposed to Kubernetes (they interact with it), but they require only a few Kubernetes professionals (which are hard to hire) to support them. These organizations are able to abstract the complexity of the underlying setup and enable developer self-service. They understood that Kubernetes is not the solution to all problems, but can be a powerful foundational platform on top of which to build a self-service layer for the whole organization.

It really comes down to balancing cognitive load, without overwhelming the individual developers. In low performing teams, developers are afraid of messing things up when deploying to a dev or staging environment in 61% of the cases. As a result, they end up needing help from more experienced colleagues to get unstuck. In high performing teams, tasks like creating new namespaces for preview environments, provisioning infrastructure and deployments to dev and staging environments can be self-served by all developers, without expert support.

To reach this level of self-service, teams have to build Internal Developer Platforms (IDPs) on top of Kubernetes. This is what allows them to implement Kubernetes the right way, finding the right level of abstraction for different engineers to interact with it and consume their setup.

Aaron Erickson, who built the IDP at Salesforce, points out the obvious scalability and efficiency bottlenecks of not implementing such platforms:

> ”If I have to run a thousand services on Kubernetes, I shouldn’t have to have a thousand Kubernetes experts to do that.”

Many teams try to overcome the complexity of Kubernetes by using abstraction layers on top of it, that can have confusing category names from control planes, managed Kubernetes, Kubernetes developer tools, to more specialiced solutions for cluster management, orchestration, deployment or monitoring and observability. However, most of these tools are purely dedicated to Kubernetes and for this reason often not a sufficient solution, as Kubernetes cannot be thought of as separate from the rest of the infrastructure. K8s is one element, not the element. Most of these Kubernetes tools work for very specific use cases, but are of little help to establish golden paths are the right level of abstraction for developers’ workflows. IDPs on the other hand, provide teams with the opportunity of implementing a holistic approach to their setup. This creates a true end to end self-service experience that results in both operations and development teams making the most of their Kubernetes implementation.

Introduction

Throughout Q4 2021 and early 2022 we used the reach of our brand to offer teams a framework to benchmark their Kubernetes setups against other teams' in our databases. Our rationale for it: most studies tend to stay rather blurry when it comes to the details of teams' everyday workflows and practices. Research suggests teams have trouble making the most of their Kubernetes setup and tend to get overwhelmed by it, but what this actually means in practice is not clear.

The response to our study was much better than originally expected. We ended up with 1,164 organizations providing deep insights into their DevOps and Kubernetes setups. In the survey, we covered three key aspects:

- State of Kubernetes
- Technical approaches
- Team setup and culture

In this study we will look to answer three key questions:

- When we talk about a "high performing organization" that is operating Kubernetes successfully, what does that look like in terms of industry proven metrics, their KPIs, team setup, technical setup, and cultural approach? What are the key differences between low and top performers?
- What can low performing organizations do to make the most of Kubernetes and adopt it successfully?
- How do you platform Kubernetes the right way and make your setup future-proof?

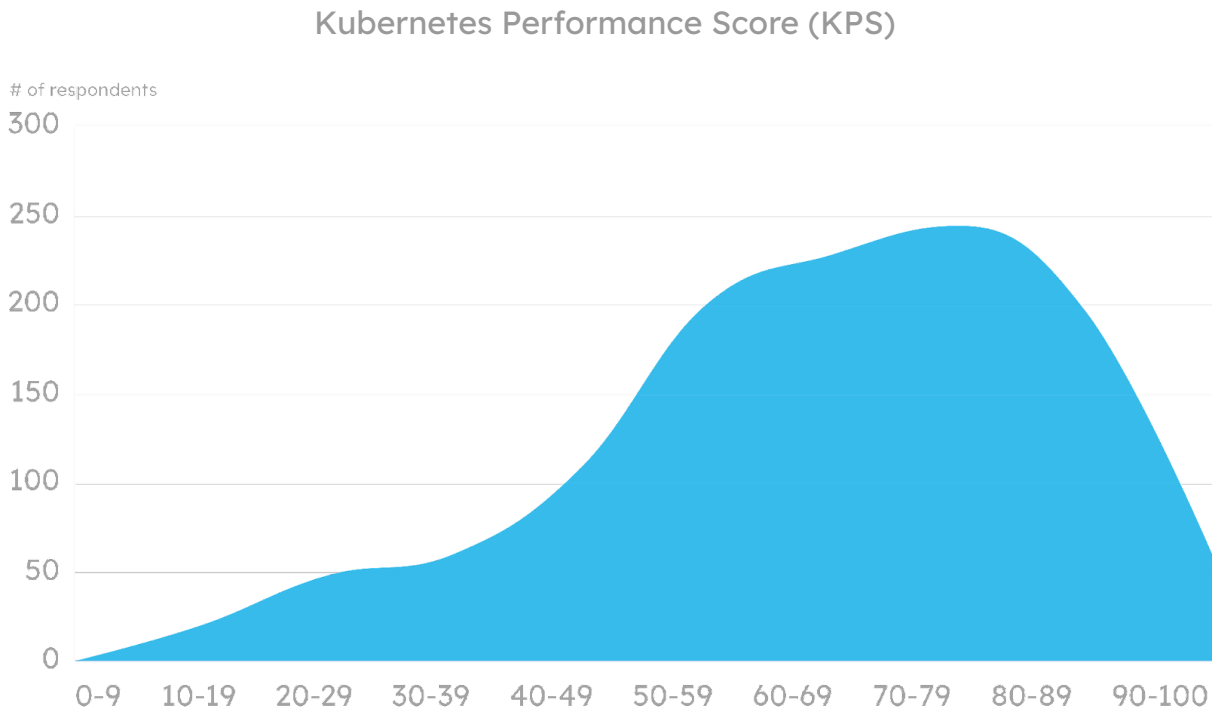
The dataset includes teams of all sizes, ranging from less than 30 to more than 250 developers, most of them based in the US or the EU.

Respondents primarily use the following titles to describe themselves on Linkedin: DevOps, CloudOps, Infrastructure, SRE, or Platform Engineer.

For a more thorough view of the data and a discussion on statistical significance, bias, etc. please refer to the appendix **About the data**.

Overall results

In order to benchmark teams' performance against one another, we developed a Kubernetes Performance Score (KPS) computed from all data we collected. Depending on the answers of the survey, a KPS between 0 – for indicating a low-performing Kubernetes setup – and 100 – for excellent performance, was calculated. Widely accepted best practices and industry standards served as the basis for these calculations. We also drew on findings from previous studies like our 2021 DevOps Setups Benchmarking Report, that we conducted ourselves. As we have already shown there, it is mainly the level of developer self-service that differentiates high performing teams from the average and low performers.



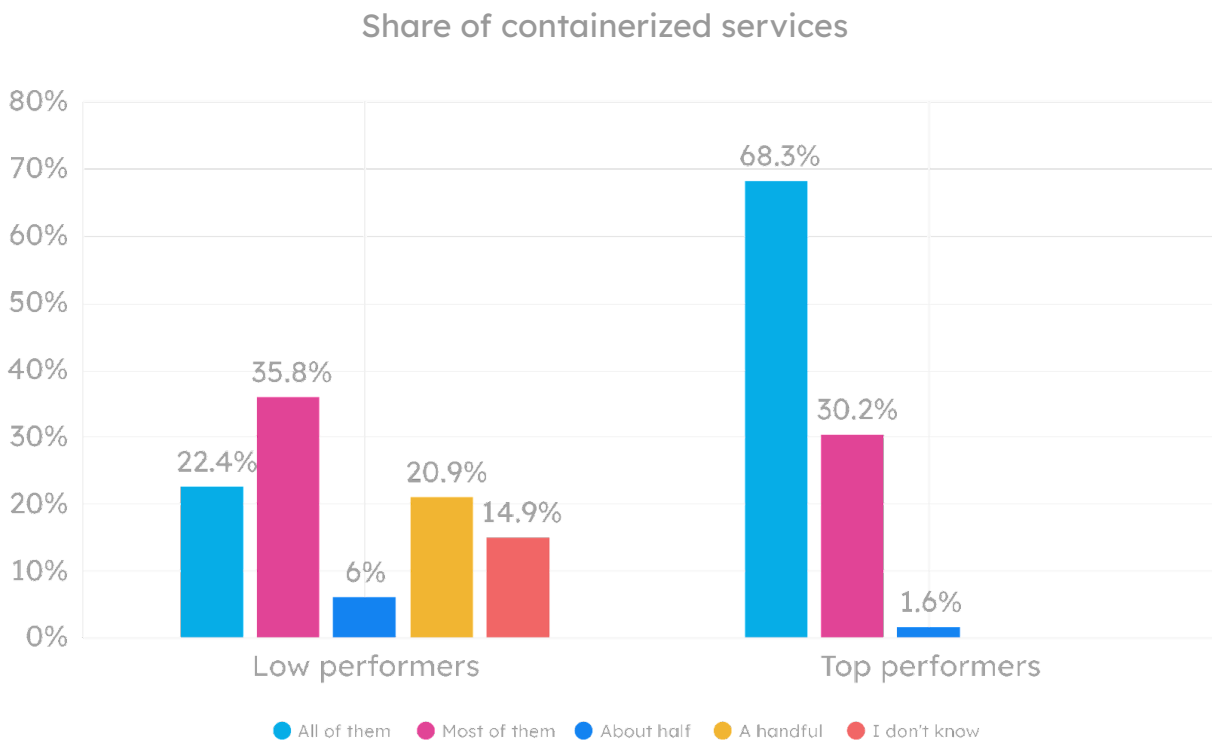
To gain a better understanding of what differentiates low-performing from high-performing teams, we clustered the data into three segments that we used for our analysis:

- Top performers with a score ≧ 80
- Mid performers with a score between 60 and 79
- Low performers with a score between 0 and 59

We mapped all scores against these segments and will analyze them in more detail.

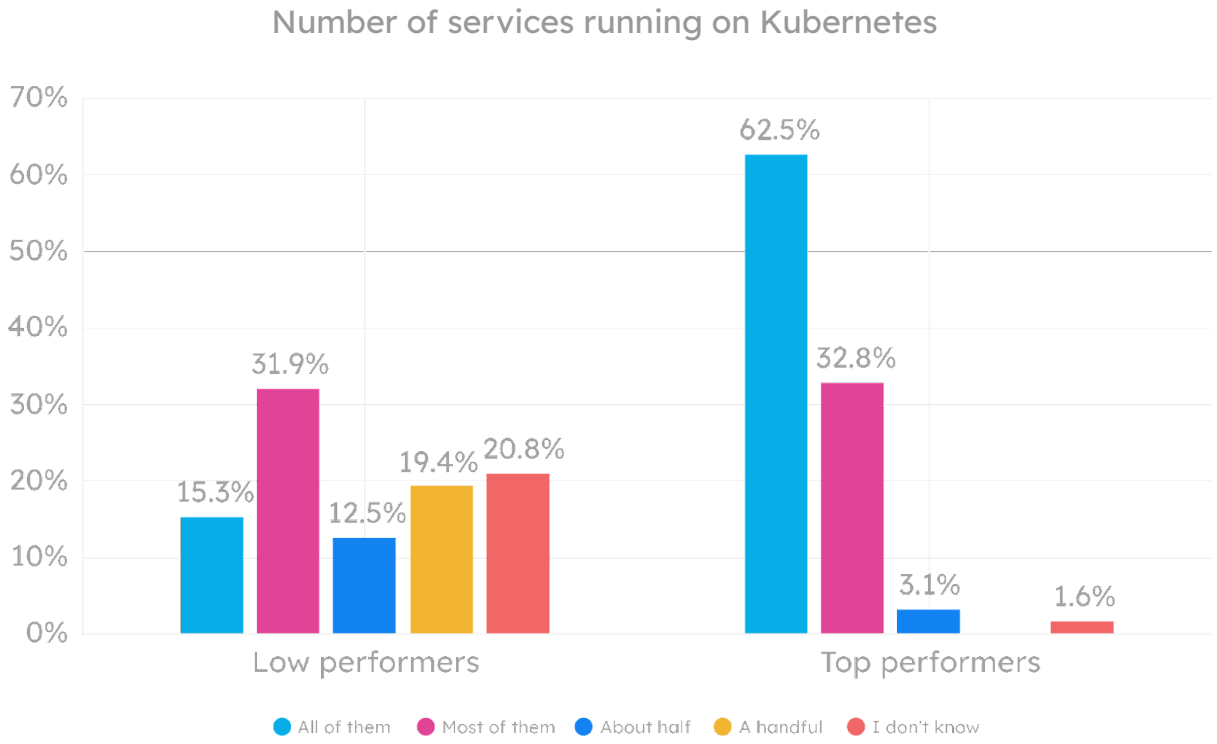
Implementation state of Kubernetes

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. In order to be able to migrate the application deployment setup to K8s, a few key requisites must be met. One in particular is essential for rolling out Kubernetes: containerization of services and applications. Below we show the degree of containerization of low vs top performers.

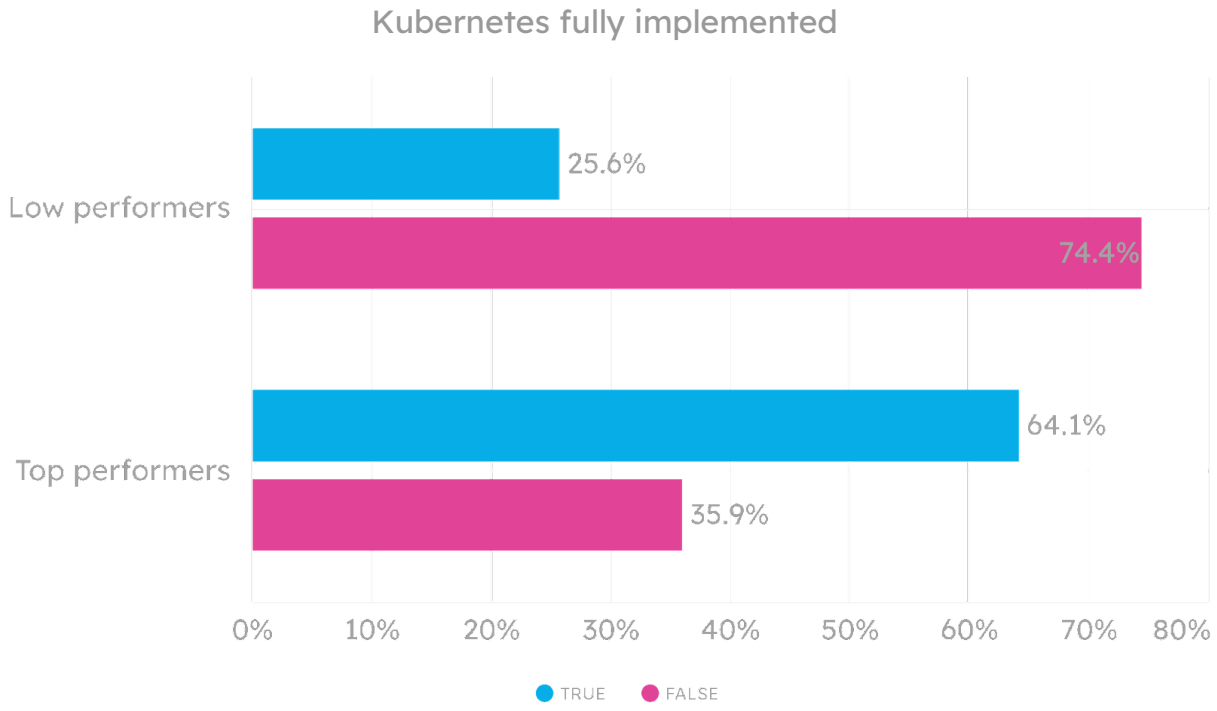


The difference between the two segments is quite stark. Over two thirds of top performing organizations have all of their services containerized and nearly all top performers have most of their apps containerized. This is compared to only 22.4% of low performers having all services containerized, with varying degrees of containerization for the rest of the segments.

Once services are containerized, they can be deployed with and orchestrated by Kubernetes. Below is a comparison of the percentages of services low performing vs top performing organizations run on top of their Kubernetes setups.



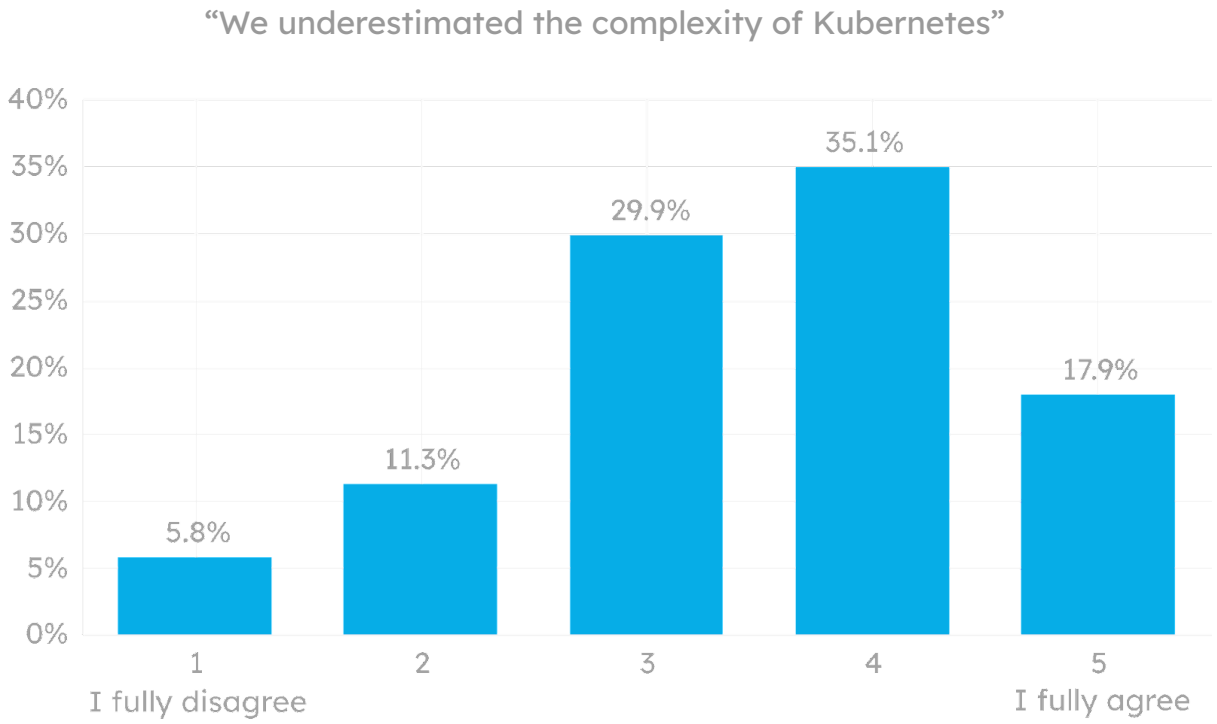
Unsurprisingly, the results mirror very closely the degree of containerization shown above, with top performers overwhelmingly running their workloads with K8s, and low performers seeing a much more varied distribution across the various percentages. What's important to notice here is that, in order to successfully roll out K8s, it is key to cross a certain proportion of services that are orchestrated with it.



Organizations that don't manage that, risk ending up in a situation where different types of services and parts of the infrastructure are managed through different kinds of solutions and orchestrators, which

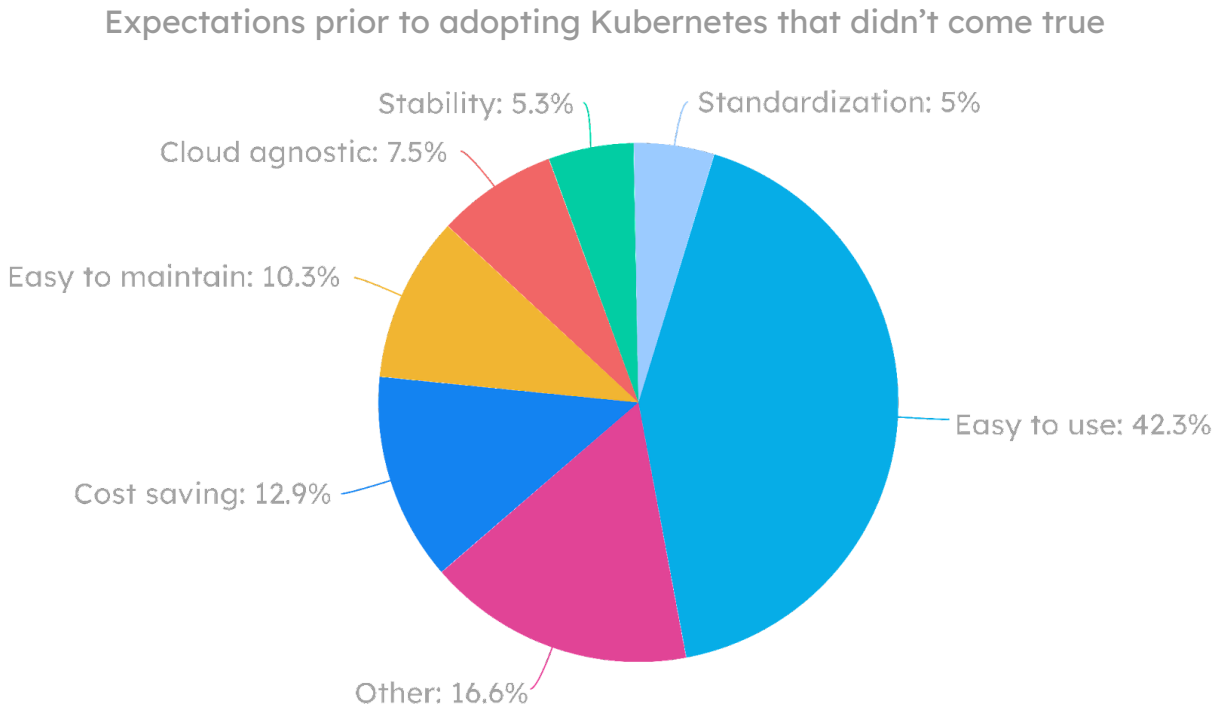
makes the entire application deployment setup a lot harder to handle. This only gets worse as the overall number of services grows. We can see how hard it is to complete a full migration to Kubernetes below.

In total, only 47.4% already completed their migration to Kubernetes, with 64.1% of top performers and only 25.6% of low performers crossing the finish line. It is apparent how easy it can be for organizations to get stuck midway through their migration. While the lack of containerization is certainly one of the key reasons, the fact that so many top performing teams also haven't completed their migration suggests something else might be involved.



Over 53% of respondents (>3 on a scale from "fully disagree" = 1 to "fully agree" = 5) state they underestimated the complexity of Kubernetes. What's even more interesting, there's only a small difference between low and top performers (~± 3%). The majority of high performers actually agree they too have underestimated the difficulty of rolling out K8s.

This raises the question of what organizations were expecting from adopting Kubernetes that didn't materialize, slowing down the implementation and preventing them from completing the roll out.



Formulated as an open question, we aggregated the answers and clustered them in the areas shown above. 42.3% thought K8s would be much easier to use and another 10.3% say K8s is harder to maintain than they originally expected. For 12.9% the promise of cost savings didn't come true. Although containerization should come with increased portability, 7.5% expected K8s to be cloud agnostic, but did not see this fulfilled.

In summary, prerequisites such as service containerization are naturally a key element for a successful adoption of Kubernetes. But reality is not so simple. All organizations, including top performers, incur the risk of getting stuck in their roll out if the latter is not carefully designed. Every team needs to be aware of the complexity they are introducing when adopting K8s and plan for it, so as to avoid halting implementation half way through, due to predictable and avoidable hurdles.

Technical challenges & best practices

Having looked at the prerequisites for a successful Kubernetes roll out, let's dive deeper into the practical technical challenges that can often emerge throughout the course of its implementation. We'll also discuss a handful of best practices teams can use to address these challenges.

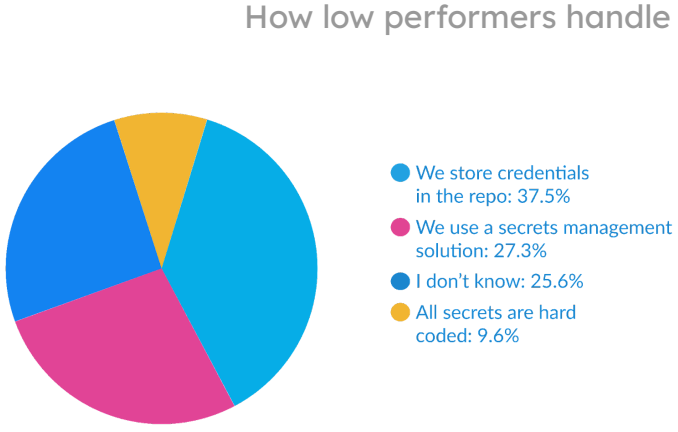
Security

Security is the first area that is often misunderstood when it comes to adopting K8s. Migrating to Kubernetes and operating applications there poses a whole new set of security challenges that organizations should be mindful of. It only takes one compromised pod, cluster or even container for a catastrophic network breach to take place. Thomas Fricke clearly showed examples of this in a recent webinar we hosted. Contrary to what some believe, Kubernetes is not secure per default.

According to the 2021 State of Kubernetes Security Report by Redhat, 94% of respondents experienced at least one security incident in their Kubernetes environments in the last 12 months. The main concern is misconfiguration that can lead to breaches:

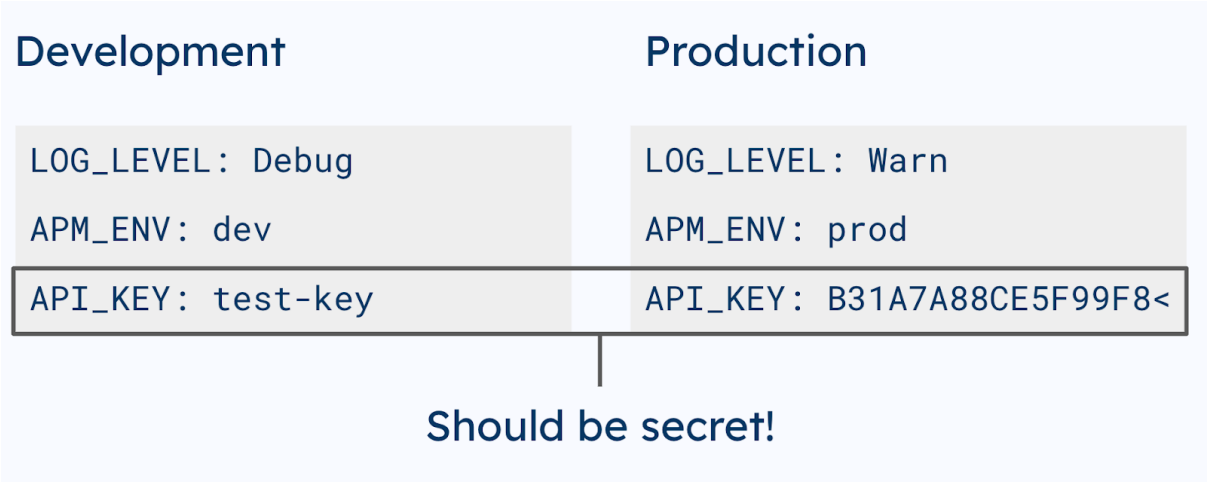
> "Survey respondents worry the most about exposures due to misconfigurations in their container and Kubernetes environments (47%)—almost four times the level of concern over attacks (13%), with vulnerabilities as the second leading cause of worry (31%)."

For 70.67% of our respondents K8s security is a big topic. While 100% of top performers use a secrets management tool like Vault, among the low performers bad security practices are widespread.



37.5% store their credentials in their repo. This is a very bad idea as source code repositories are meant to be shared, with your teammates, your company, or possibly with the entire world (as is the case for open source software). Keeping secrets outside of Git is especially important for future-proofing. Git is designed to keep a persistent history of all your code changes, so once a secret is checked into source control, removing it requires rewriting history, and that can be really hard if not impossible. Because Git is distributed, other developers may preserve your secret in their own local copies of the repo.

25.6% don't know, which is also a bad sign of what their security policies might look like. Finally, close to 10% of low performers hardcode their secrets in their environment. This creates a big potential vulnerability that can allow an attacker to bypass the authentication. Below is a simple example to illustrate this.

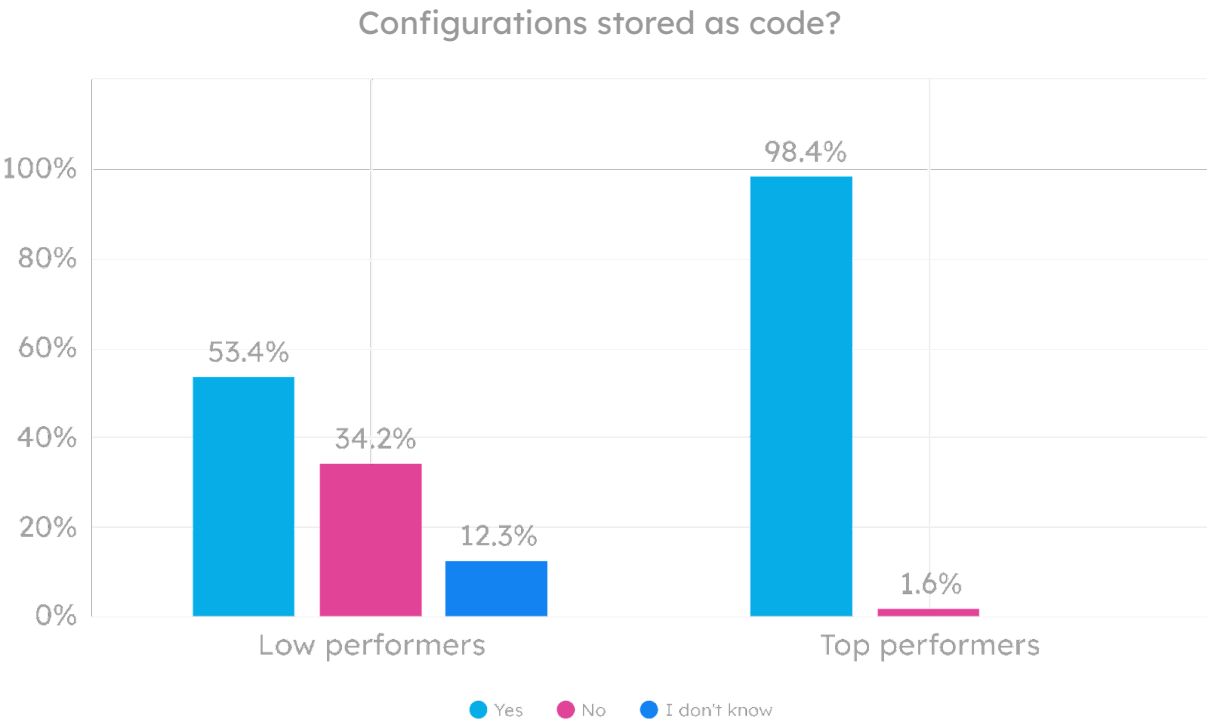


Application configuration management

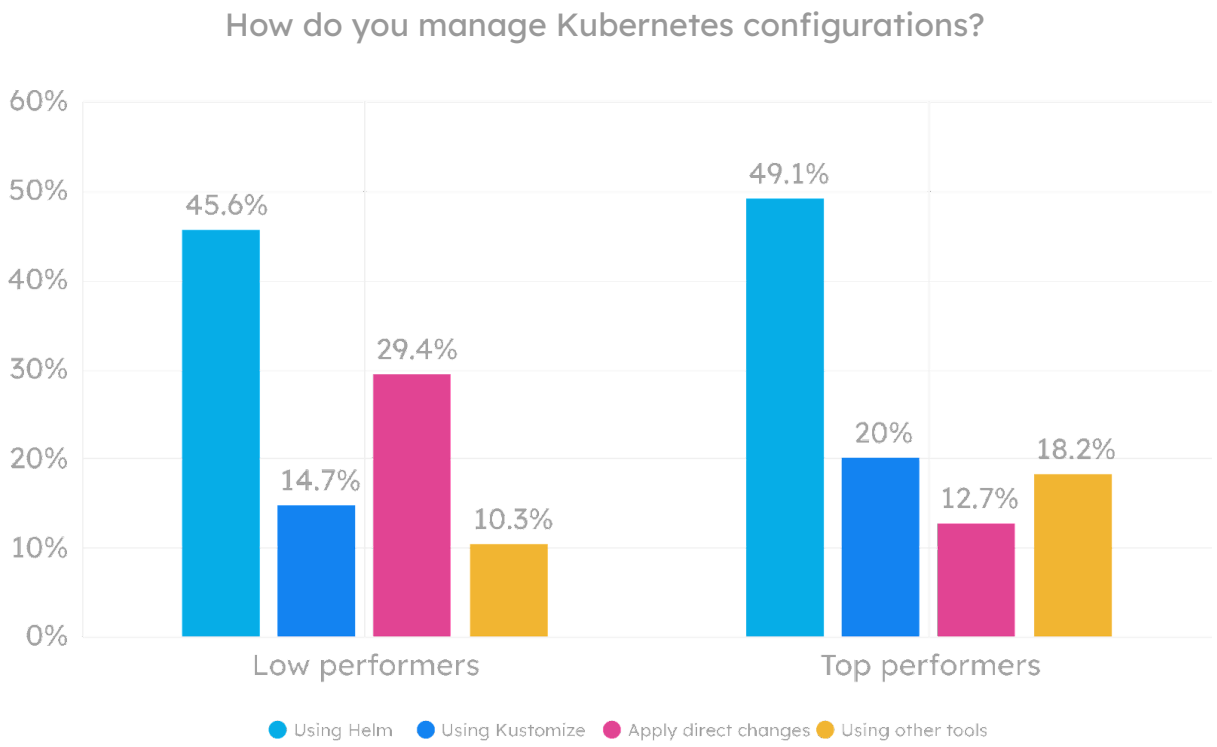
Best practices around application configuration management are some of the few things most DevOps experts and practitioners tend to agree on:

- Configs should be stored as code
- Environment agnostic configs need to be kept separated from environment specific configs
- Credentials, API keys, etc. should be secret

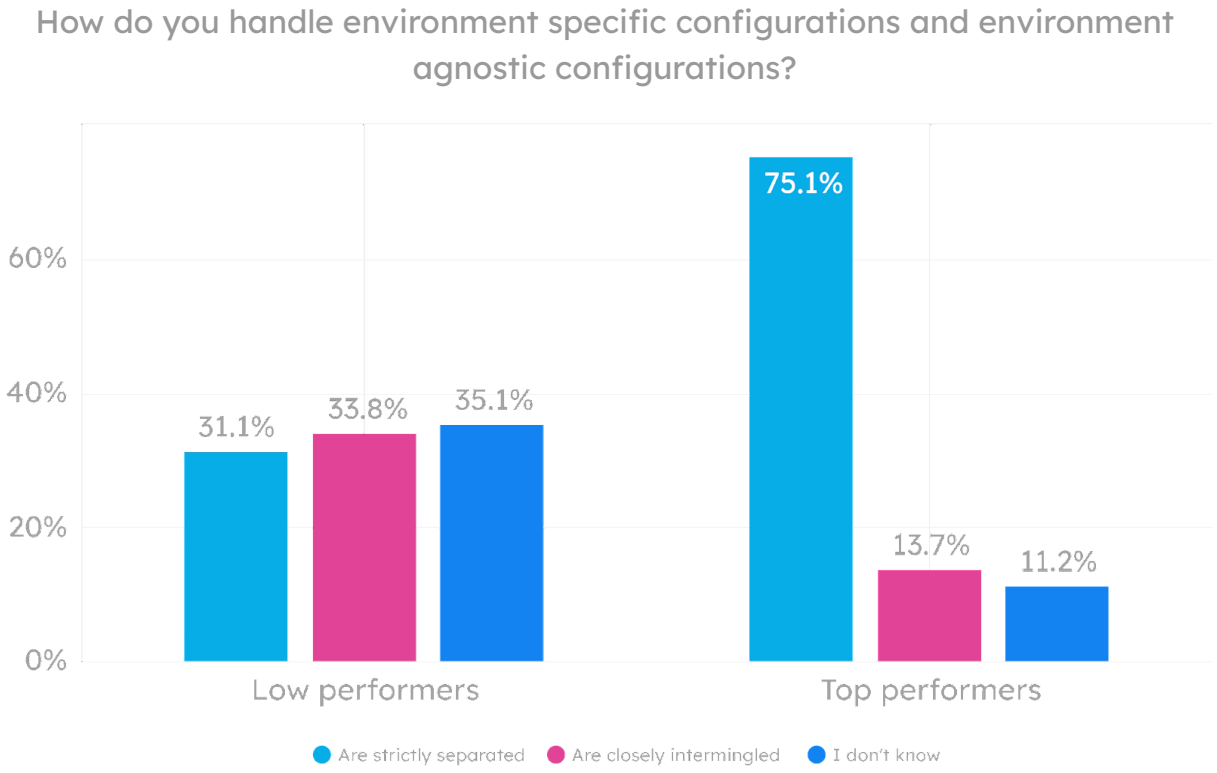
Below is a breakdown whether configurations are stored as code, across both low and top performers.



The difference here is quite impressive. Storing application configurations as code is a clear best practice and a determining factor in the success of your Kubernetes setup implementation. Next we look at what solutions teams use to manage K8s configurations.



What's interesting to see here is that close to 30% of low performing organizations apply changes manually, which is again not best practice, while a higher percentage of top performers have a solution like Helm or Kustomize in place for that.



Another key differentiator is the way how environment specific configurations and environment agnostic configurations are handled. What is clear is that low performers don't often keep a strict separation, with 33.8% saying their configurations are closely intermingled and another 35.1% who don't even know.

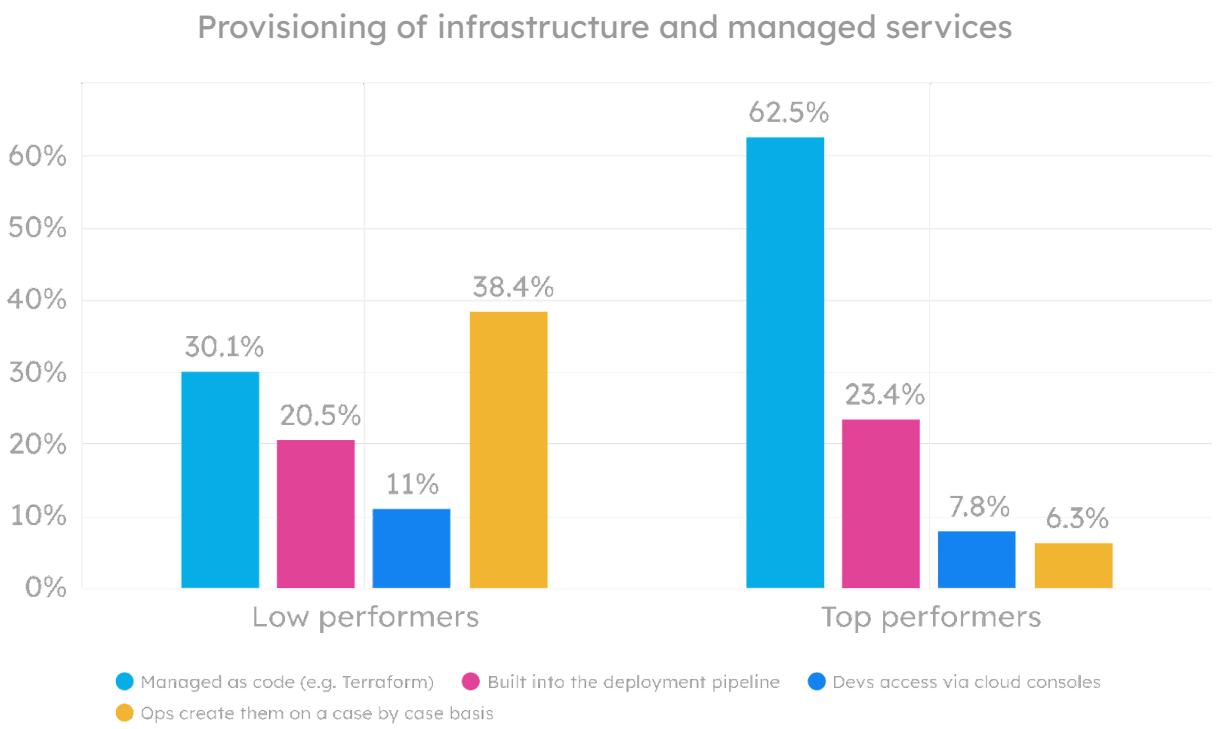
Top performing organizations on the other hand use Internal Developer Platforms to keep env specific and env agnostic configurations strictly separated, as we will show later.

Infrastructure provisioning

Having looked at how teams manage application configurations, we'll analyze how they go about provisioning the infrastructure their applications depend on.

It is particularly noteworthy that the provisioning and orchestration of infrastructure, such as databases, storage or DNS, goes far beyond Kubernetes. There are many different ways of provisioning a database and connecting it to an application running on Kubernetes, including intermediate steps like creating an Auth proxy in a sidecar container. All these different actions can soon lead to unstructured workflows that try to accommodate every developer's or dev team's preferences, while also requiring specialized experts, which in turn leads to key person dependencies.

So, how do teams deal with managed services like Google Cloud SQL or Amazon RDS, what about DNS and storage?



Low performers fiddle with infrastructure on a case-by-case basis, depending on what they need to run that specific application or set of services. Instead, top performers follow an IaC approach or build provisioning into their deployment pipeline.

Team setup and cultural aspects

As mentioned at the beginning of this study, introducing Kubernetes into your setup raises not only technical questions but also a number of cultural ones.

Given K8s' complexity, rolling it out and maintaining it, as well as the daily interactions developers have with it, require a well planned learning curve that goes along with its implementation.

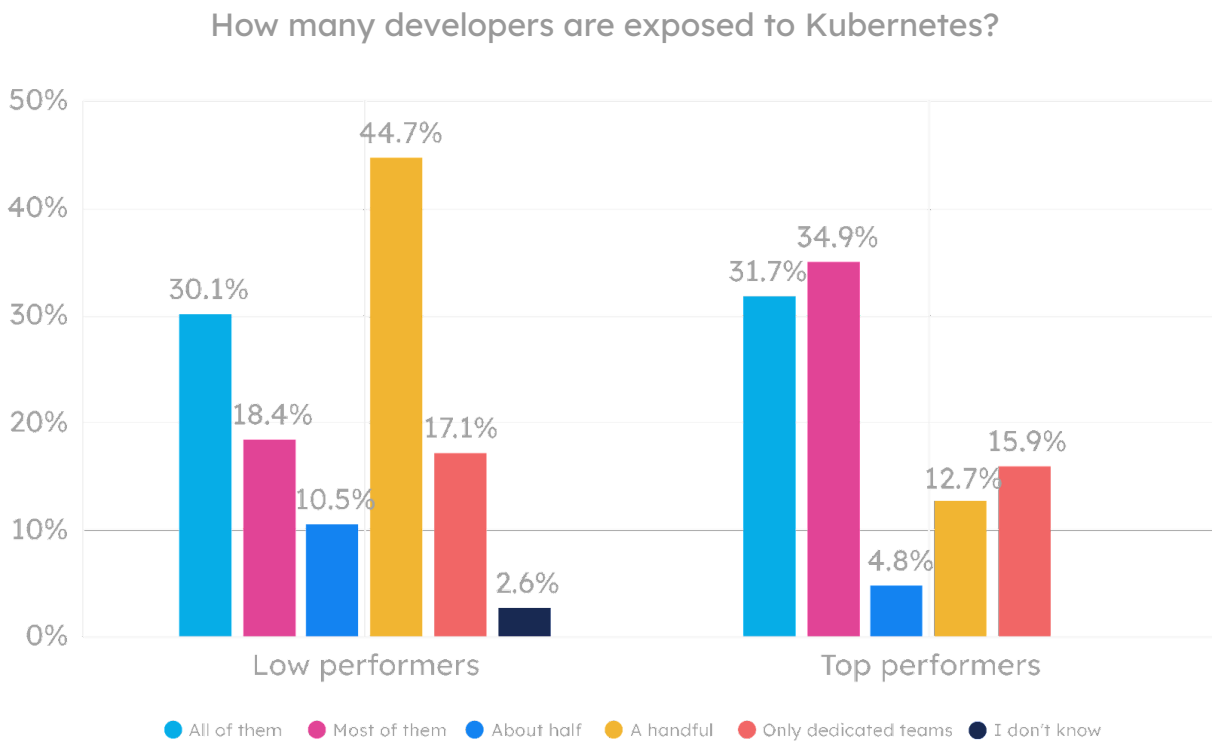
In this section, we look at three overall questions:

- How many developers from the team will be exposed to Kubernetes?
- Who needs to be an expert in it? Who of our existing staff can and is willing to learn it?
- Are we able to hire experts if we need to?

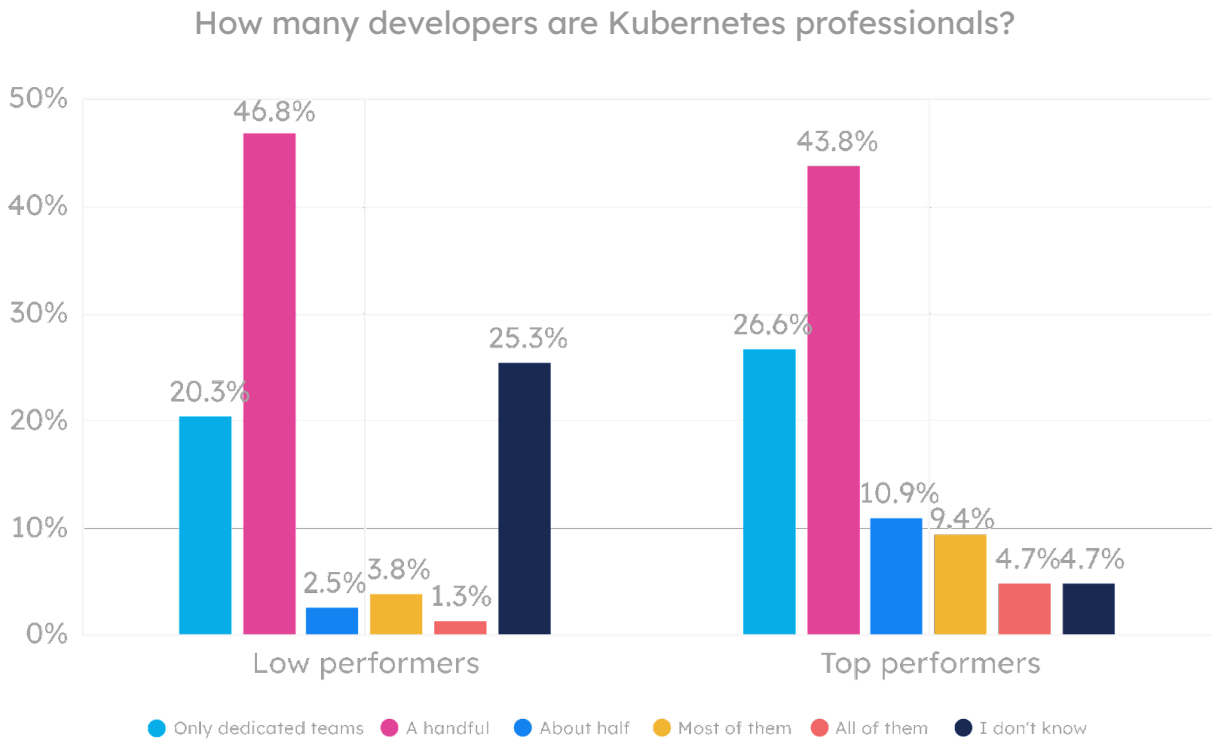
All these questions need to be thought through and answered before embarking on the Kubernetes journey. Once started then, many more will arise, e.g. around documentation, who should be able to perform which kind of tasks and, crucially, onboarding times. We recently published an article discussing the challenges of scaling your team alongside K8s, which can hopefully be a good starting point for internal conversations. Now let's look at the data.

Exposure of developers to Kubernetes

In general we see two approaches emerge when it comes to exposing developers to the Kubernetes setup: as many as possible (everyone should be able to do everything) vs. as few as possible (only specialists should be exposed to Kubernetes).



In low performing organizations, only a minority of developers are exposed to Kubernetes. The issue with this is that when only a few specialists or dedicated teams are exposed to K8s, there is a pretty high risk of introducing key person dependencies. On the other hand, the vast majority of developers at top performers are exposed to Kubernetes. You might argue that is also bad, as everyone would have to be an expert in K8s. But if we look at the next data point, we can see how not all developers from top performing teams that are exposed to Kubernetes are specialists in this domain:

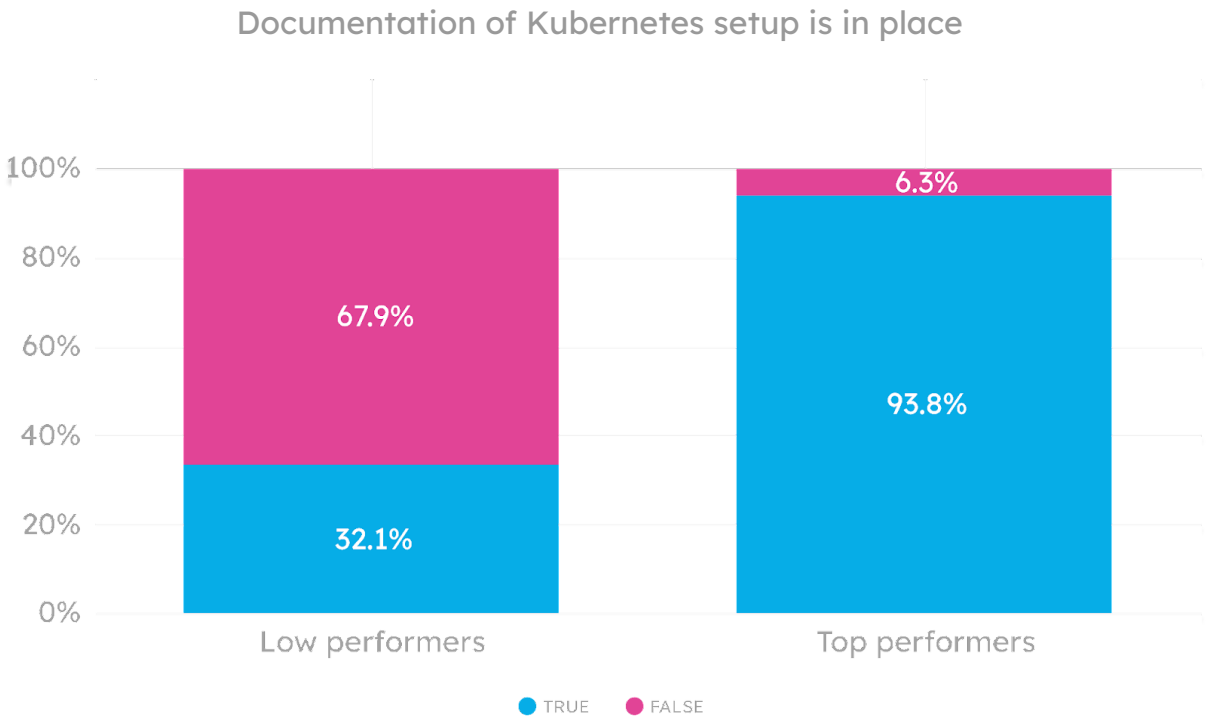


Here the data are quite similar across the two segments, it doesn't seem to be the case that top performing teams just have more Kubernetes specialists. This is good news, as they don't need more budget and resources to hire high paid specialists, but rather leverage only a handful of dedicated teams and Kubernetes professionals.

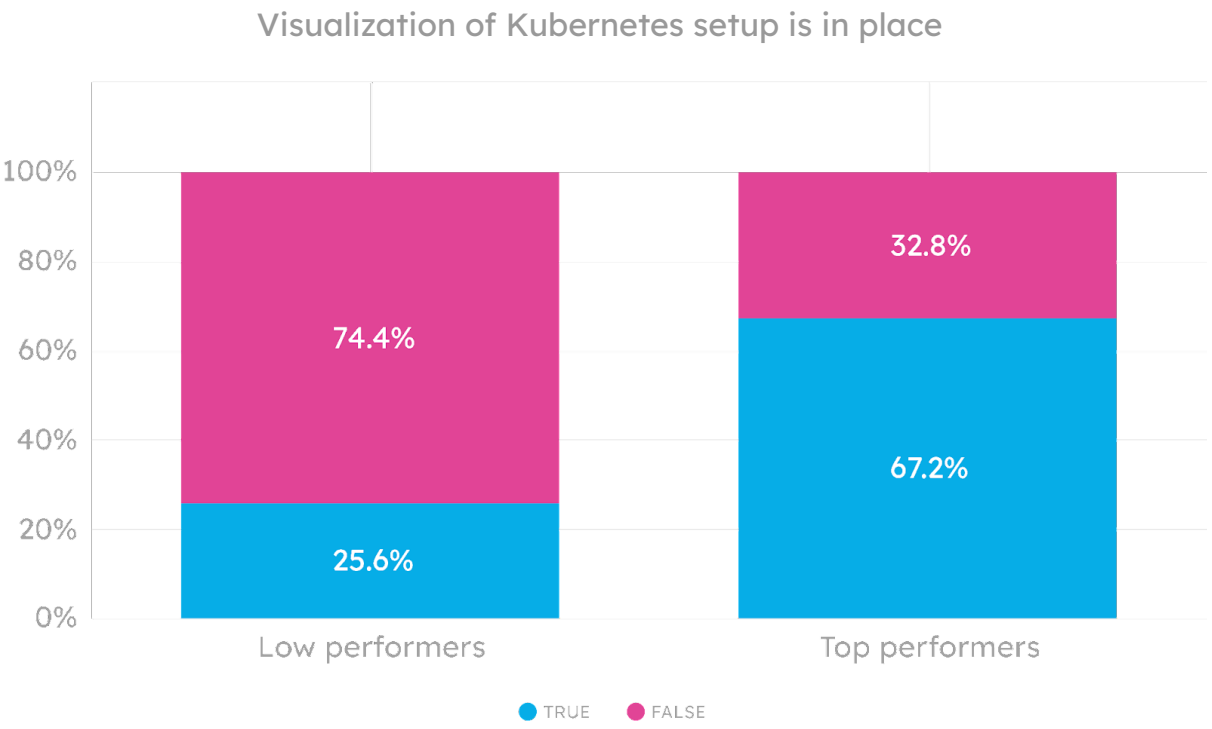
So how do top performers get to a place where they can manage their K8s setup efficiently and expose most developers to it, though the majority of them are not experts in Kubernetes?

Documentation

Let's start with looking at documentation. The more complex your setup, the more important your internal documentation is.

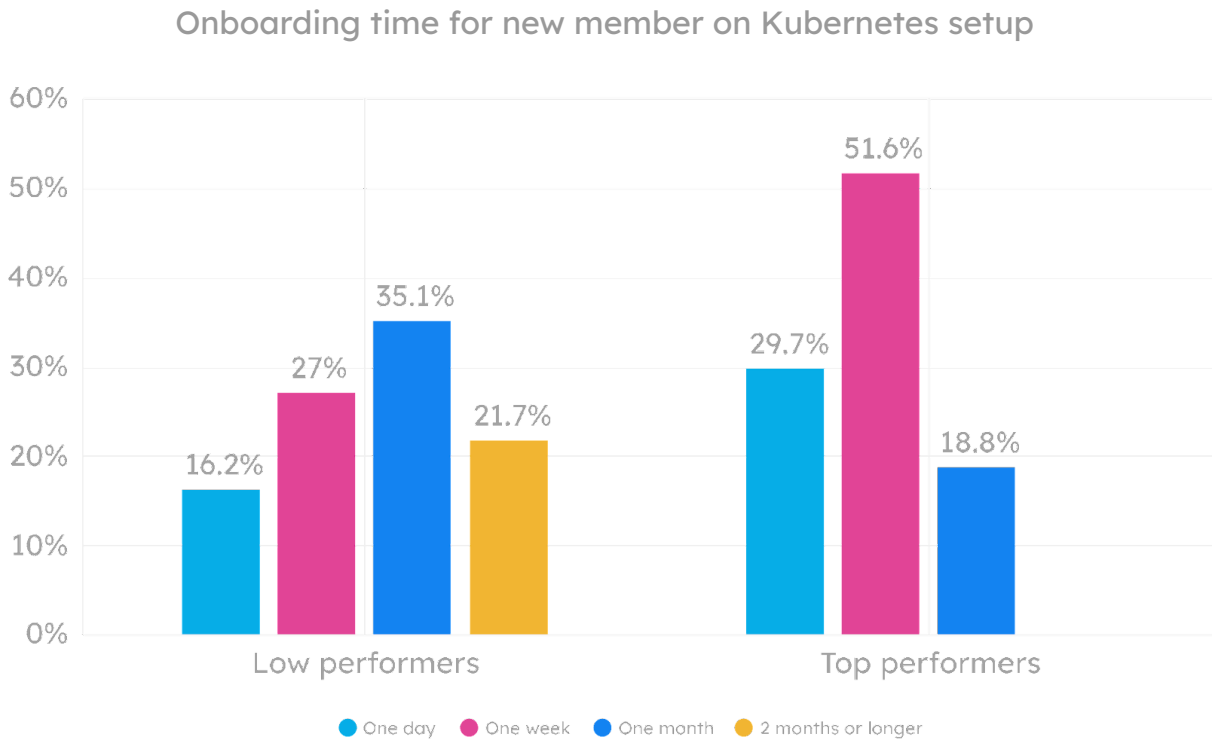


Top performers document their Kubernetes setup overwhelmingly more accurately than low performers. Same goes for visualizing its state.



Onboarding time

Better documentation and visualization of your K8s setup not only means it's easier and safer to expose more developers to it. You can also observe a much faster onboarding in high performing teams.



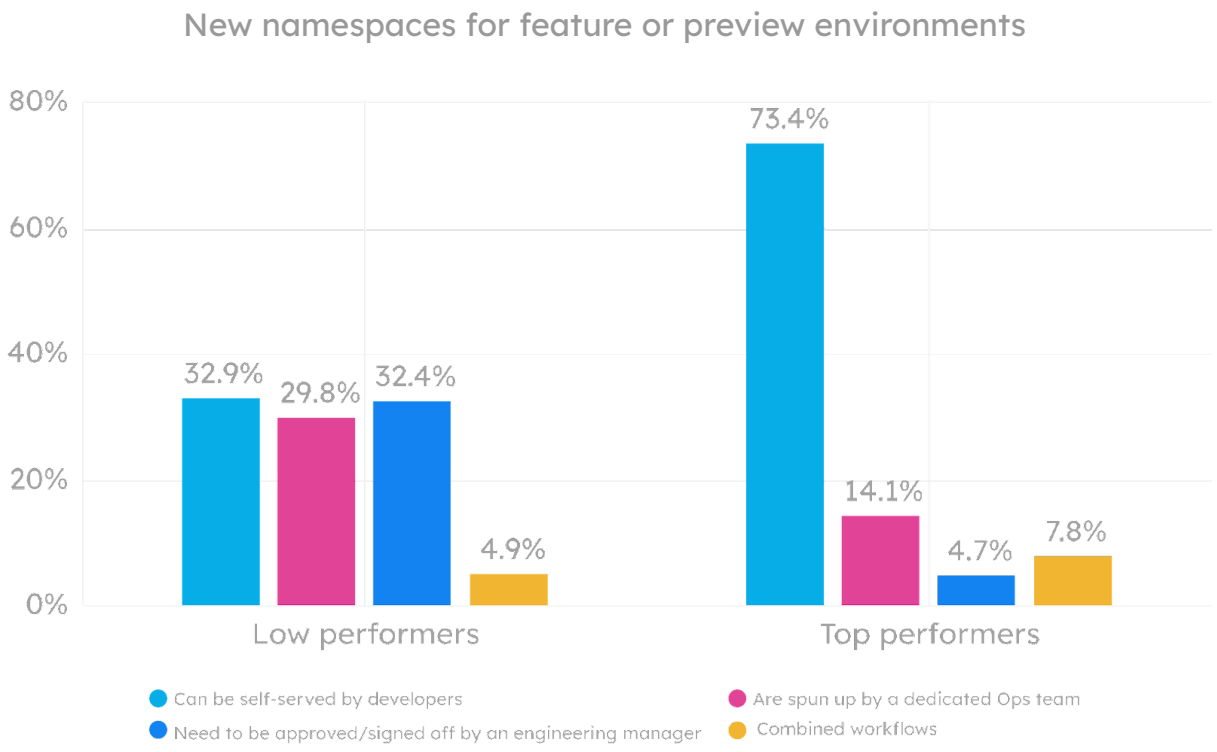
Degree of developer self-service

Next, we look at the degree of self-service organizations have enabled for their developers to interact independently with their Kubernetes setup and, in general, their infrastructure.

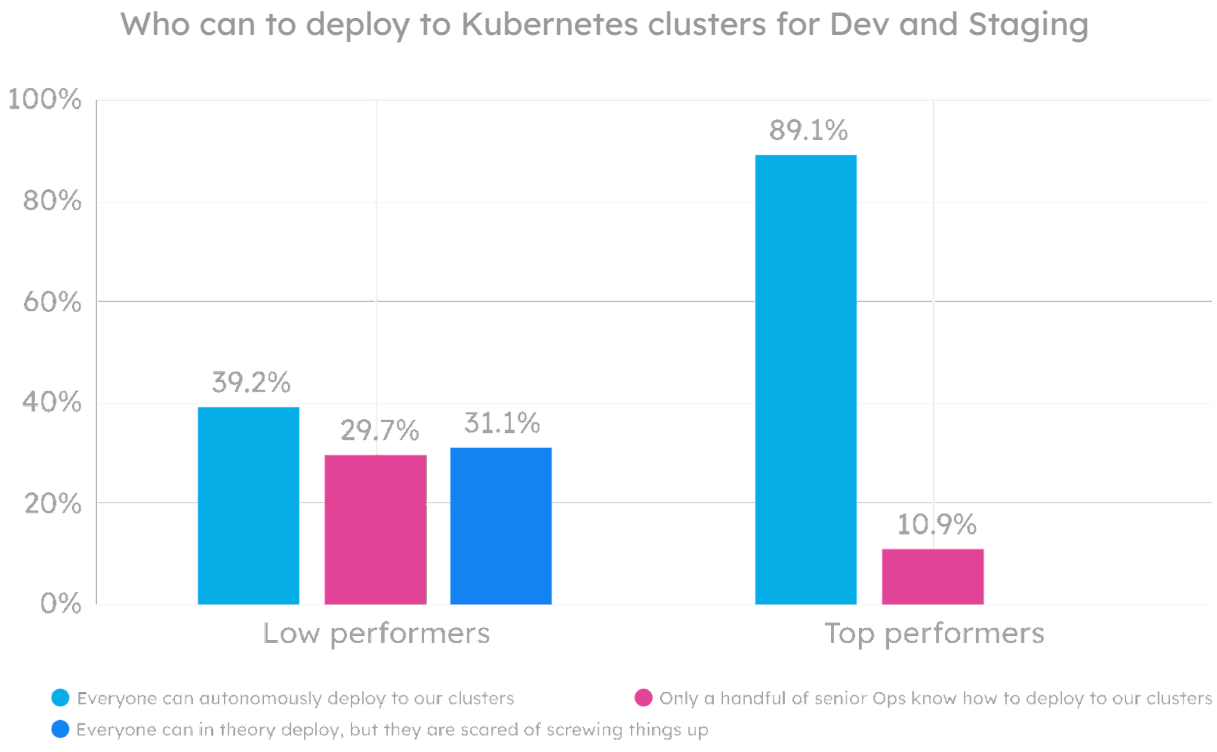
First however, what does developer self-service mean? It means engineers can self-serve the tech they need to run their apps and services, without having to wait on operations to provision resources or spinning up environments for them. That, however, doesn't necessarily mean developers need to be experts in everything and master their entire toolchain, including Kubernetes. It also doesn't mean everyone should be able to deploy to production.

But if we think about repetitive tasks, like spinning up new feature or preview environments or provision resources for apps running in those environments, that's where enabling developer self-service can save teams a huge amount of time and resources.

The higher degree of self-service in top performers is clear looking at the graphics below. In these teams, developers can self-serve K8s namespaces for feature and preview environments in 73.4% of cases. That is true for only 33.3% of low performers.



The data showing who is able to deploy to Kubernetes clusters for Dev and Staging paints a similar picture.



For low performers, 29.7% say only a handful of senior Ops engineers know how to deploy to their Kubernetes clusters. This leads to a bottleneck situation, where Ops end up doing TicketOps all day and can't focus on more important (and less menial) tasks. As mentioned above, this also creates key person dependencies every company would like to avoid, if possible.

Even more revealing though is the 31.1% claiming everyone can in theory deploy to their K8s clusters, but they usually don't because they are scared of screwing things up. This indicates a lack of understanding of their setup and shows distrust in how the tooling is configured and documented. As a result, lead time increases and deployment frequency decreases, because the team is not able to deploy with confidence.

Compared to this, 89.1% of the top performers are able to deploy to dev or staging on their own and on demand. Once again, a stark difference.

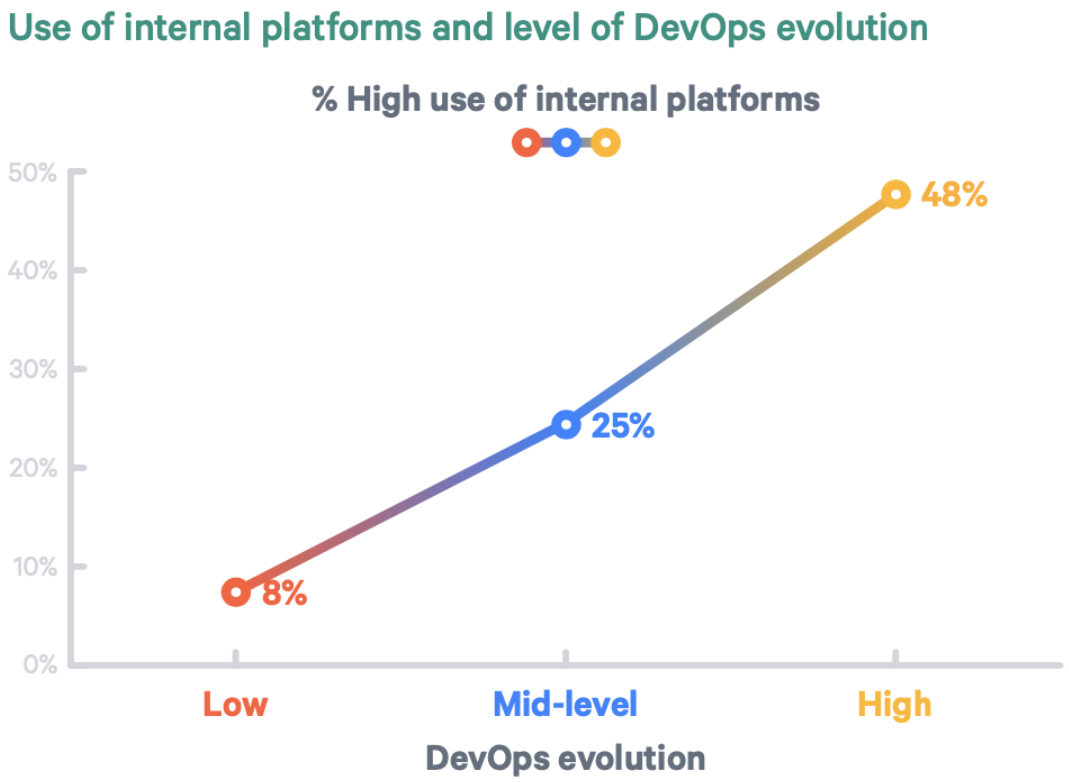
Platform strategies to nail Kubernetes implementation

There is a common misconception that adopting Kubernetes is equivalent to building your internal platform. Although the Kubernetes documentation describes it as “a portable, extensible, open-source platform for managing containerized workloads and services”, the reality is that K8s can only be one building block of a much more comprehensive setup.

As Manuel Pais, co-author of Team Topologies explains it in a recent blog post:

> “Kubernetes is not your platform, it is just the foundation.”

Summarizing all the data points we analyzed throughout this study, it becomes clear that successful Kubernetes roll outs, as well as its day to day operations and interactions across different teams, are heavily dependent on it being part of a larger platform. Top performing engineering organizations build internal platforms to enable developer self-service, to allow their developers to interact independently with the underlying infrastructure and Kubernetes setup, while at the same time shielding them from its complexity.



Source: Puppet State of DevOps Report 2020

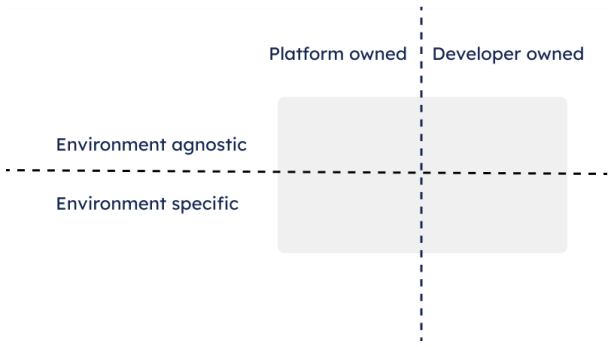
This result also corresponds with further studies. The 2020 State of DevOps Report showed a high correlation between a high DevOps evolution (low lead time, high deployment frequency, low change failure rate, quick MTTR) and the usage of an internal platform and self-service capabilities.

Compared to Kubernetes native tooling (managed K8s, control planes, abstraction layers, etc.), Internal Developer Platforms provide a holistic approach. They do not only abstract Kubernetes complexity away from developers' daily workflows, but also enforce app config management best practices by default.

What does this mean in detail? The real complexity of Kubernetes comes from applications depending on resources like file storage, databases or DNS, especially when these resources need to be handled in different ways in different environments. As we have seen, deploying such applications to different environments can lead to key person dependencies and waiting times, especially if developers are not able to self-serve. This is where IDPs can help.

Let's take one more example: if you deploy an application that is running in a dev environment to the next, a testing environment, you'd expect some parts of the configuration to change, others not. Additionally you might have requirements from different teams involved, for example infrastructure teams that want to have cloud resources managed in a certain way and database admins that worry about database credentials.

To have to know all this about application configuration is beyond the cognitive load developers can handle, especially when it is not their actual job. Rather than having dozens of devs fiddling around in unstructured YAML files, IDPs nudge developers to follow standardization by design. IDPs auto-generate app configs dynamically for every deployment, based on the context of every environment.



As we learned, top performing teams enforce a strict separation between environment specific and environment agnostic configurations. IDPs allow for these policies and best practices by default. They also provide additional layers that separate configs that are owned by the platform team and those developers are responsible for.

How certain parts of the infrastructure are provided for different environments is preconfigured by the platform team, for example which static database to use for production and which dynamic ones for dev and testing. Same applies for file storage or DNS.

Well designed IDPs do not only help developers to manage K8s itself, but also orchestrate dependent resources and manage their relationships to workloads running inside Kubernetes. Modern setups orchestrate IaC modules, inject credentials as secrets into containers at runtime and manage the setup on an environment by environment basis.

Such a standardization approach leaves way less possibilities that developers have to think about, which eliminates not only key person dependencies and waiting times, but also reduces developers' cognitive load. This ensures flawless deployments with a stable platform, developers can do their work more effectively with much more confidence.

When you reached this stage, then you really nailed building an IDP to bring all the benefits of Kubernetes to scale. With that you also laid the foundation to grow your engineering organization without having to slow down as new developers onboard. Top performing teams don't have to hire more Kubernetes experts, they just put them to work where they can lay the groundwork for the other developers.

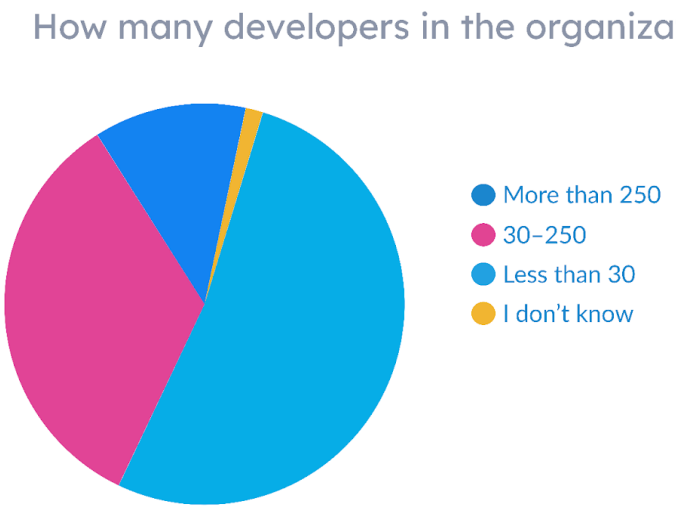
Kubernetes was always meant to be a platform, but as pointed out by Kelsey Hightower, K8s is a platform to build platforms, not the endgame in itself. To make the most of Kubernetes, as well as any other new technology you want to add to your setup, it is key to implement them as part of a larger framework, an Internal Developer Platform. That is why top performing organizations create internal platform teams to build and maintain IDPs.

Key takeaways

1. If your organization's setup is not running on Kubernetes yet, make sure to come up with a migration and implementation plan that reflects both prerequisites and best practices (technical incl. e.g. security, as well as cultural).
2. Enable developer self-service for your Kubernetes setup, it is the key to developer productivity. Without it, you risk slowing down your whole development lifecycle, increasing change failure rate, bottlenecks, key person dependencies.
3. To enable self-service, work with an IDP that not only enforces Kubernetes best practices (e.g. by separating environment-specific and environment-agnostic configs, managing secrets, etc.) but also provides self-service capabilities for infrastructure provisioning beyond Kubernetes.
4. Choose an IDP that provides different levels of abstraction, depending on the preferences of your individual engineers and specific development teams.

About the Data

Respondents in total: 1,164



Trying to be fair statisticians, we always ask the question of significance and bias. Of course, this data is not a random set of engineering teams selected without any bias. These are teams that signed up on Humanitec.com, interested in building an Internal Developer Platform. This intention is already an indication of a certain sophistication and openness to innovate.

We decided not to do N/A replacements to approximate results. Instead, we deleted data with vital entries missing, such as the number of developers. We correlated other attributes of N/A data points to ensure deleting these data points wouldn't skew the final analysis. We ended up with a total of 1,164 engineering organizations.

So yes, the data is biased to a certain extent. But it's still a good representation of the roughly 10,000 teams that match our above-mentioned criteria.

Literature

Galante, Luca: Internal Platform Teams: What Are They and Do You Need One?
https://humanitec.com/blog/internal-platform-teams-what-are-they-and-do-you-need-one

Grünberg, Kaspar von: What Is an Internal Developer Platform,
https://humanitec.com/blog/what-is-an-internal-developer-platform

Humanitec DevOps Setups: A Benchmarking Study 2021,
https://humanitec.com/whitepapers/2021-devops-setups-benchmarking-report

Kubernetes by the Numbers,
https://enterprisersproject.com/article/2020/6/kubernetes-statistics-2020

New SlashData Report: 5.6 Million Developers Use Kubernetes, an Increase of 67% over One Year,
https://www.cncf.io/blog/2021/12/20/new-slashdata-report-5-6-million-developers-use-kubernetes-an-increase-of-67-over-one-year/

Manuel Pais, "Kubernetes Is Not Your Platform, It's Just the Foundation",
https://www.infoq.com/articles/kubernetes-successful-adoption-foundation/

Puppet State of DevOps Report 2020
https://puppet.com/resources/report/2020-state-of-devops-report

Puppet State of DevOps Report 2021
https://puppet.com/resources/report/2021-state-of-devops-report

Q1 2021 State of Cloud Native Development,
https://www.cncf.io/wp-content/uploads/2021/12/Q1-2021-State-of-Cloud-Native-development-FINAL.pdf

2021 State of Kubernetes Security Report
https://www.redhat.com/rhdc/managed-files/cl-state-kubernetes-security-report-ebook-f29117-202106-en_0.pdf

Stephenson, Chris: Scaling Your Team Alongside Kubernetes,
https://humanitec.com/blog/scaling-your-team-alongside-kubernetes

Skelton, Matthew, and Manuel Pais. 2019. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution.