DZone

TREND REPORT

OCTOBER 2023

Kubernetes in the Enterprise

Redefining the Container Ecosystem

BROUGHT TO YOU IN PARTNERSHIP WITH

Teleport

Welcome Letter

By Ray Elenteny, Solution Architect at SOLTECH, Inc.

The adoption of Kubernetes continues its march forward. Is it a perfect technology stack? No, but what technology stack is? Does it have a somewhat steep learning curve? Yes, it does. With these and other questions, you might ask:

"Why does the adoption of Kubernetes continue to grow, and do I *want* to jump into the pool? Isn't it just (and yet) another deployment abstraction layer like how virtual machines eventually became ubiquitous?"

To some extent, those are legitimate questions.

As someone who has literally spent decades in this industry delivering products and applications, the drive to deliver applications more efficiently and quickly has, at the very least, remained a constant, and one could argue that the drive to deliver continues to increase in intensity. I see Kubernetes as an opportunity to take a significant step forward in improving application delivery.

Many of us love researching and adopting new technologies. Kubernetes and containerization, in general, are "cool stuff." However, many of us also make a living in this field, and businesses don't really care about cool stuff. Stakeholders want to be shown how they can leverage technology to their advantage.

When you look at the topics presented in this Trend Report, they focus on technical opportunities, managing costs, and streamlining processes — important topics to business stakeholders.

While Kubernetes is an evolution in technology, when applied with the right mindset, it encourages a cultural shift. To me, this is what makes Kubernetes exciting. I see it analogous to the Agile movement. Key to Agile methodologies is having representation from all aspects of an organization. Kubernetes, done well, is a collaborative effort across multiple organizational disciplines.

Kubernetes has allowed the replication of production-like environments to be available as early in the process as an engineer's workstation. This capability facilitates more communication between those creating an application and those who deploy and monitor it; all parties involved get to understand each other's requirements and challenges, and working as a team can resolve them. This requires cooperation, teamwork, empathy, and a host of other "soft skills." It can change the culture of an organization, and I would argue for the better.

So while you explore in the 2023 *Kubernetes in the Enterprise* Trend Report how Kubernetes can move your business forward, consider how Kubernetes can also impact the culture within your organization — for the better.

Seize the Opportunity,

Ray Elenteny

Ray Elenteny, Solution Architect at SOLTECH, Inc., DZone Core Member
@rbetae on DZone | @ray-elenteny on LinkedIn

With over 35 years of experience in the IT industry, Ray thoroughly enjoys sharing his experience by helping organizations deliver high-quality applications that drive business value. Ray has a passion for software engineering. Over the past ten years or so, Ray has taken a keen interest in the cultural and technical dynamics of efficiently delivering applications.

ORIGINAL RESEARCH

Key Research Findings

An Analysis of Results from DZone's 2023 Kubernetes Survey

By G. Ryan Spain, Freelance Software Engineer, former Engineer & Editor at DZone

Kubernetes: it is *everywhere*. To fully capture or articulate the prevalence and far-reaching impacts of this monumental platform is no small task — from its initial aims to manage and orchestrate containers to the more nuanced techniques to scale deployments, leverage data and AI/ML capabilities, and manage observability and performance — it's no wonder we, DZone, research and cover the Kubernetes ecosystem at great lengths each year.

In September 2023, DZone surveyed software developers, architects, and other IT professionals in order to understand the state of Kubernetes across enterprises.

Major research targets were:

1. The current state of containers and container orchestration
2. Kubernetes advantages, disadvantages, and pain points
3. Kubernetes practices and techniques

Methods: We created a survey and distributed it to a global audience of software professionals. Question formats included multiple choice, free response, and ranking. Survey links were distributed via email to an opt-in subscriber list, popups on DZone.com, the DZone Core Slack workspace, and various DZone social media channels. The survey was opened on August 31st and ended on September 19th; it recorded 103 complete and partial responses.

Demographics: Due to the limited response rate for our 2023 Kubernetes survey, we've noted certain key audience details below in order to establish a more solid impression of the sample from which results have been derived:

- Respondents described their primary role in their organization as "Technical Architect" (22%), "Developer/Engineer" (21%), "Developer Team Lead" (13%), "Consultant/Solutions Architect" (11%), and "DevOps Lead" (11%).
- 80% of respondents said they are currently developing "Web applications/Services (SaaS)," 55% said "Enterprise business applications," 22% said "Native mobile apps," and 21% said "High-risk software (bugs and failures can mean significant financial loss or loss of life)."
- "Java" (74%) was the most popular language ecosystem used at respondents' companies, followed by "JavaScript (client-side)" (56%), "Python" (52%), and "Node.js (server-side JavaScript)" (43%).
- Regarding responses on the primary language respondents use at work, the most popular by far was "Java" (43%), followed distantly by "Python" (15%) and "Go" (11%).
- On average, respondents said they have 17.28 years' experience as an IT professional, with a median of 18 years' experience.
- 33% of respondents work at organizations with < 100 employees, 18% work at organizations with 100-999 employees, and 47% work at organizations with 1,000+ employees.

In this report, we review some of our key research findings. Many secondary findings of interest are not included here.

Research Target One: The Current State of Containers and Container Orchestration

Motivations:

1. Kubernetes and containers are inextricably linked, and one of the main reasons behind container orchestration demand — and, by extension, the purpose of this very report — is the use of containers *seemingly everywhere* in contemporary software development. We wanted to know how often containers are being used today compared to our surveys from previous years, and whether organization/application size correlated with container usage.
2. Container usage can be so prevalent in modern software because of the availability and accessibility of robust container management tools. From free and open-source tools to paid, enterprise-level solutions, there are more than a few options for spinning up, managing, and orchestrating containers. We aimed to see which container tools were being used most often.

3. Finally, to get to the crux of this Trend Report, we wanted to find out how often Kubernetes is being used — both organizationally (how many companies are running Kubernetes clusters) and individually (how many software professionals have worked with Kubernetes themselves).

CONTAINER POPULARITY

In 2021's *Kubernetes in the Enterprise* "Key Research Findings," we speculated that containerization may have reached a point of saturation based on the results of our survey. Last year, our data indicated that container usage may have even begun declining, especially among smaller organizations.

To continue this analysis of the prevalence of containers, comparing with annual data from 2017 to the present, we asked:

> *Do you use application containers in either development or production environments?*

Results (*n*=91):

Figure 1

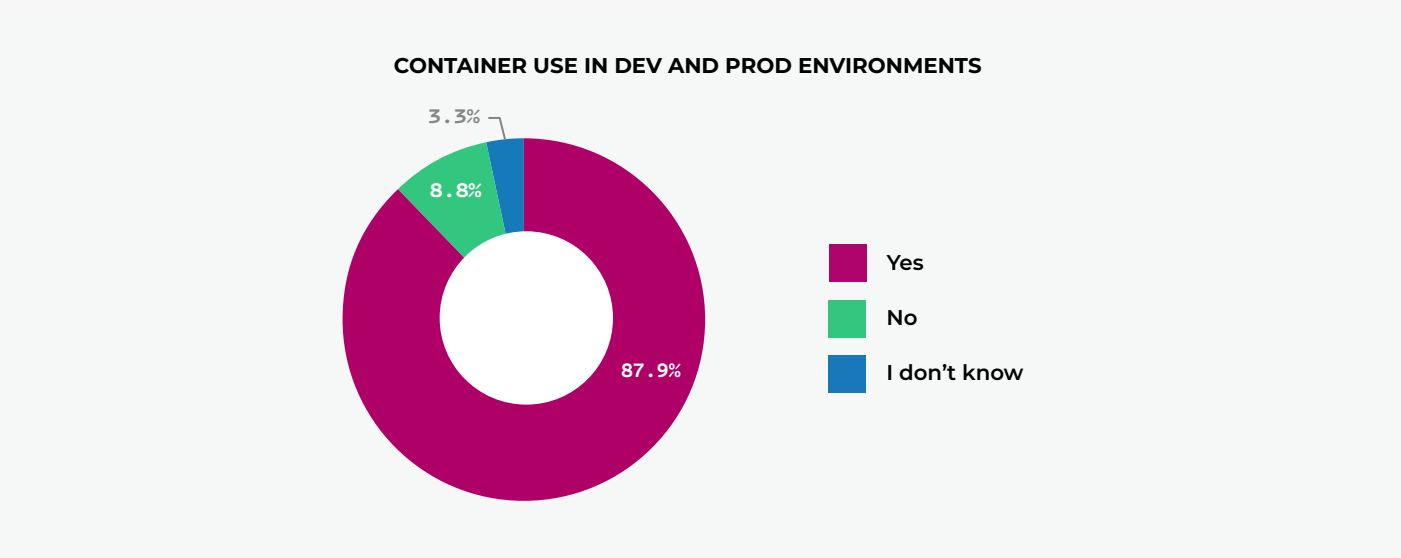
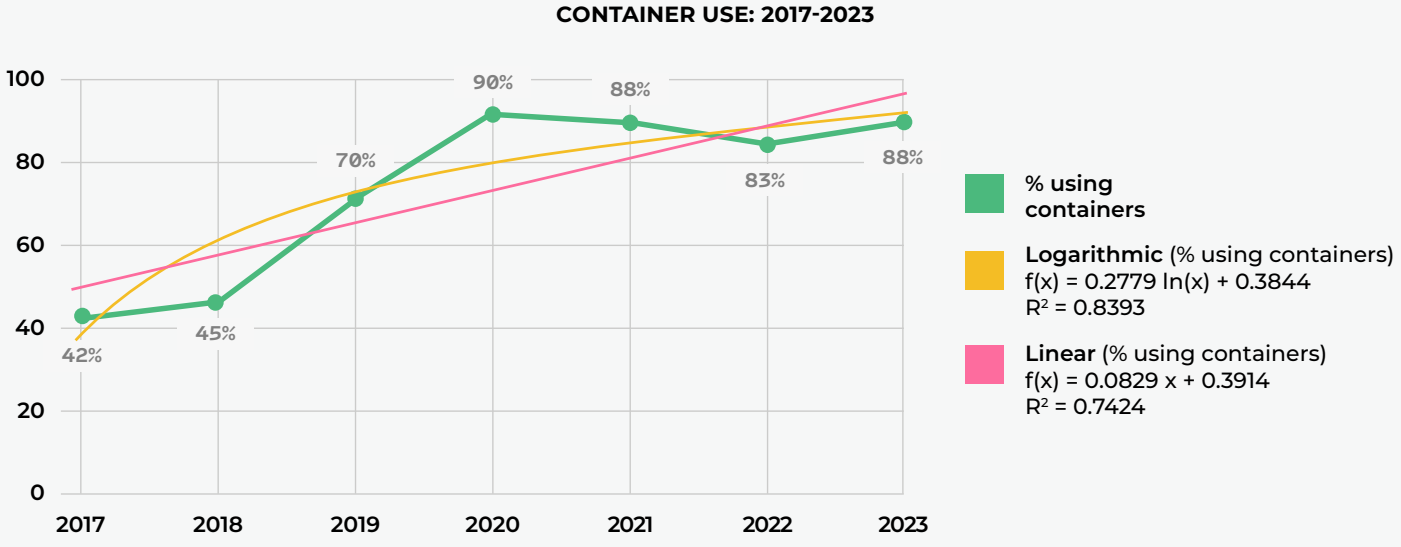


Figure 2



Observations:

1. Container usage has returned to the levels we saw in 2020 and 2021, with 88% of respondents saying that they use application containers in either development or production environments. This reaffirms some of the suspicions we had about the validity of a subset of responses to last year's survey and suggests that those results may have been lower than they should have been.

   The data also reinforces the hypothesis we stated in 2021 — that containerization has reached a saturation point. Further supporting the saturation hypothesis is the trendline for Figure 2, where a logarithmic model (R^2=0.839) fits much better than a linear one (R^2=0.742). We anticipate, based on the data, that next year's survey will show around a 90% container usage rate.

Table 1

CONTAINER USE BY ORGANIZATION SIZE

Response	Org Size		
	1-99	100-999	1,000+
Yes	71%	100%	98%
No	21%	0%	3%
I don't know	7%	0%	0%

2. We found again this year that respondents at the smallest organizations (1-99) were the least likely to use containers. In fact, nearly all respondents at organizations with > 100 employees said that they use application containers (99%), while only 71% of respondents at organizations with < 100 employees said they use containers.

   If we were to assume that larger organizations tend to have larger/more complex applications, this discrepancy may indicate that relatively small or simple applications just don't have the same need for containerization — even in development — as bigger applications. Alternatively, it could be that smaller organizations are less likely to be able to expend the resources necessary to properly manage and orchestrate containers.

CONTAINER MANAGEMENT TOOLS

Considering the apparent saturation of container usage, it seems that the tools being used for container management — and the demand for those tools — are more important than ever. To get a better sense of this demand and the tools being used to fill it, we asked:

What tools/platforms are you using to manage containers in development and production environments?

Results (*n*=75):

Table 2

TOOLS USED TO MANAGE CONTAINERS IN DEV AND PROD ENVIRONMENTS*

Tool	Development		Production	
	%	*n*=	%	*n*=
Docker	81%	61	48%	36
Kubernetes	71%	53	64%	48
Docker Compose	43%	32	16%	12
Terraform	32%	24	29%	22
AWS EKS	27%	20	28%	21
Azure AKS	24%	18	19%	14
OpenShift	19%	14	15%	11
GKE	15%	11	15%	11
AWS ECS	13%	10	17%	13
Ansible	11%	8	8%	6

**Note: This table only displays options selected by > 5% of respondents in either the Development or Production category.*

Observations:

1. Docker was the most commonly used container management tool in development environments, with 81% of respondents saying they use Docker on the dev side of things; Kubernetes was not far behind at 71%. Other commonly used container tools for development were Docker Compose (43%), Terraform (32%), AWS EKS (27%), and Azure AKS (24%).

2. In production environments, Kubernetes and Docker switched places, with 64% of respondents saying they use Kubernetes and 48% saying they use Docker in prod. Terraform (29%), AWS EKS (28%), and Azure AKS (19%) were again popular options here — though to a lesser extent than Kubernetes and Docker — followed by AWS ECS (17%).
3. 81% of respondents said they are using Docker in either development, production, or both types of environments (incidentally, this means that all respondents who said they were using Docker were using it in a development environment). 76% of respondents said they are using Kubernetes in either one or both of these types of environments.
4. Among the three biggest cloud providers — Amazon, Microsoft (Azure), and Google — AWS EKS (28%) was significantly more popular than Azure AKS (19%) in production environments, but in development, the 3% lead that AWS EKS (27%) had over Azure AKS (24%) is statistically insignificant (based on a 6% margin of error). GKE came behind both AWS EKS and Azure AKS in both production (15%) and development (15%) environments, though again, the Azure AKS lead in production environments here is statistically insignificant.

KUBERNETES USAGE

As we saw from the data presented in the "Container Management Tools" section, Kubernetes is being used by a lot of software professionals. We wanted to dive a little deeper into the frequency of its use, so we asked respondents:

> *Does your organization run any Kubernetes clusters?*
>
> and
>
> *Have you personally worked with Kubernetes?*

Results (*n*=88 and *n*=85, respectively):

Figure 3

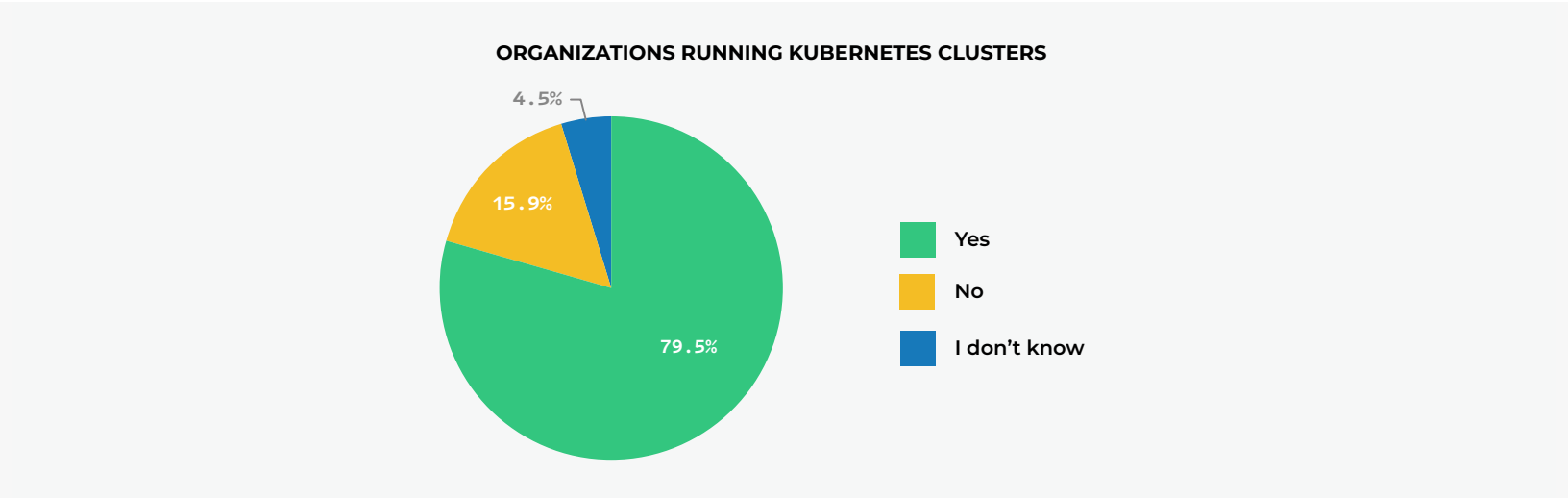
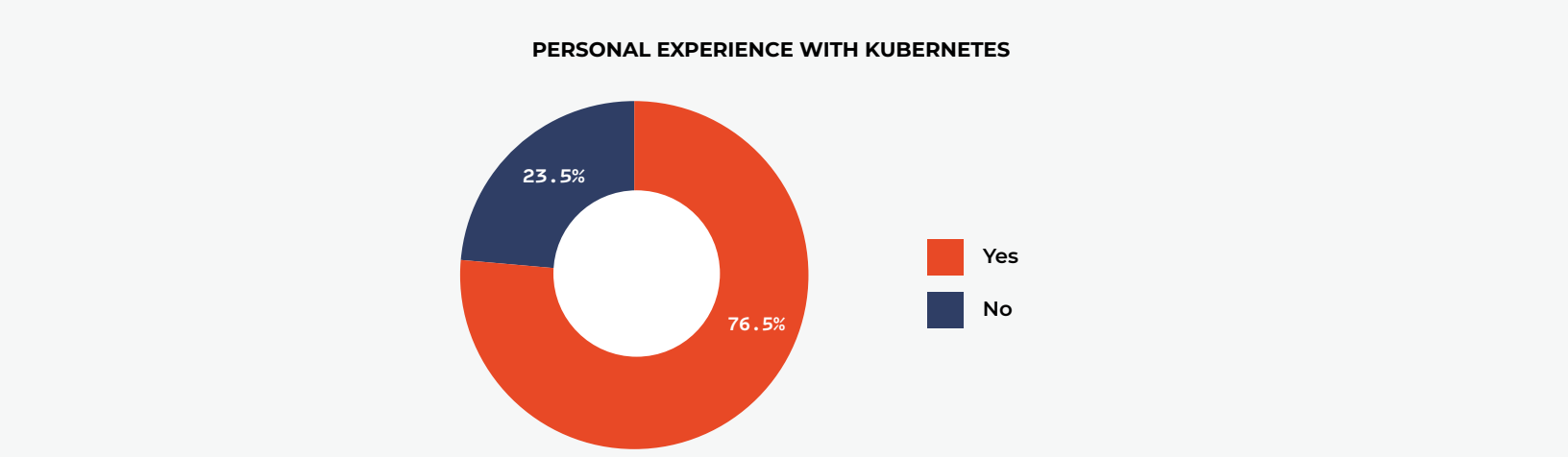


Figure 4



Observations:

1. 80% of respondents said that their organization is running Kubernetes clusters, 16% said that their organization does not, and 5% said that they did not know. Furthermore, 77% of respondents said that they have personally worked with Kubernetes, and 24% said they have not. The results of these two questions were, unsurprisingly, correlated: 90% of respondents at organizations using Kubernetes said that they have used it themselves, compared to only 29% of respondents at organizations not using Kubernetes.
2. Much like we saw with container use in general, smaller organizations are much less likely to run Kubernetes. 71% of respondents at smaller organizations (< 100 employees) said their organization has Kubernetes clusters running, compared to 91% of respondents at organizations with > 100 employees. Again, we presume this discrepancy may be related to small organizations/applications lacking either the need or the resources for running Kubernetes — or some mix of the two — compared to large, or even mid-sized, organizations.

Research Target Two: Kubernetes Use Cases, Advantages, and Disadvantages

Motivations:

1. As we observed in the previous section, Kubernetes is likely being used to some extent in most software-focused or software-adjacent organizations, and by most software professionals. As such, we wanted to know the types of use cases organizations are using Kubernetes for most commonly.
2. Choosing software tools always involves some amount of weighing the advantages and disadvantages, and even the best tools have their pros and cons. We asked respondents what Kubernetes has improved — and what it has worsened — at their organizations.
3. No technology is without its pain points, especially a technology as relatively new as Kubernetes. We aimed to find out which Kubernetes frustrations were causing the most agitation for software professionals.

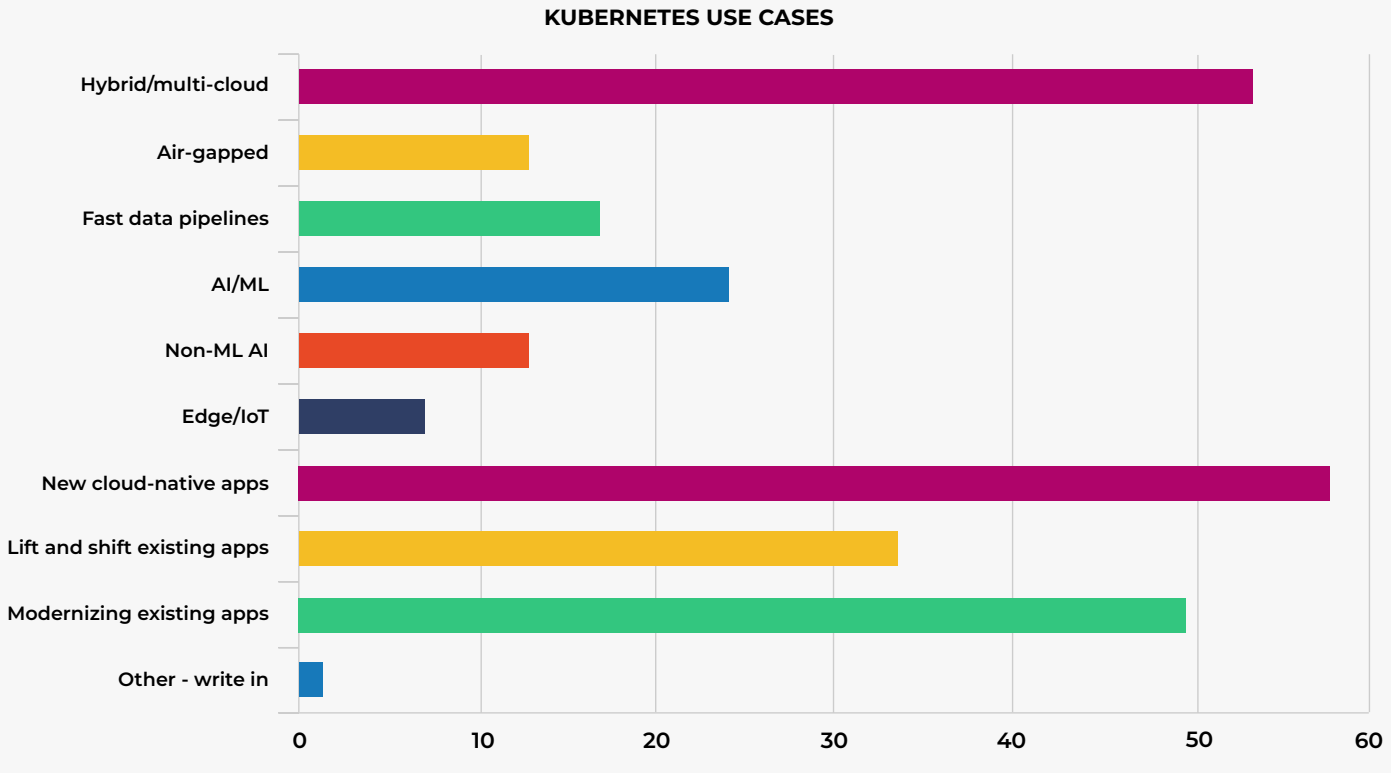
KUBERNETES USE CASES

Before diving into the Kubernetes pros and cons that respondents observed, we wanted to look at the use cases for which Kubernetes was being employed, so we asked respondents:

*What use cases do you deploy Kubernetes into?**

Results (*n*=69):

Figure 5



Observations:

1. The most common Kubernetes use cases were "New cloud-native apps" (58%), "Hybrid/multi-cloud" (52%), and "Modernizing existing apps" (49%). The least common use cases were "Air-gapped" (12%), "Non-ML AI" (12%), and "Edge/IoT" (7%).
2. Respondents at organizations with < 100 employees were considerably less likely than respondents at larger organizations to use Kubernetes for "Hybrid/multi-cloud" (24% vs. 63%), "Lift and shift [for] existing applications" (18% vs. 35%), and "Non-ML AI" (0% vs. 17%). On the other hand, there was no significant difference between small and large organization response rates for "Modernizing existing applications" (47% vs. 50%) and "Edge/IoT" (6% vs. 8%) use cases.
3. Respondents at smaller organizations, on average, selected fewer Kubernetes use cases (1.76) than those at larger organizations (2.94).

Table 3

KUBERNETES USE CASES BY ORGANIZATION SIZE

Use Cases	Org Size		Gap
	< 100 (*n*=17)	**> 100** (*n*=48)	
Hybrid/multi-cloud	24%	63%	39%
Air-gapped	6%	15%	9%
Fast data pipelines	6%	21%	15%
AI/ML	18%	25%	7%
Non-ML AI	0%	17%	17%
Edge/IoT	6%	8%	2%
New cloud-native apps	53%	60%	7%
Lift and shift existing apps	18%	35%	18%
Modernize existing apps	47%	50%	3%

**Note: This question was only asked to respondents who answered "Yes" to the question, "Does your organization run any Kubernetes clusters?"*

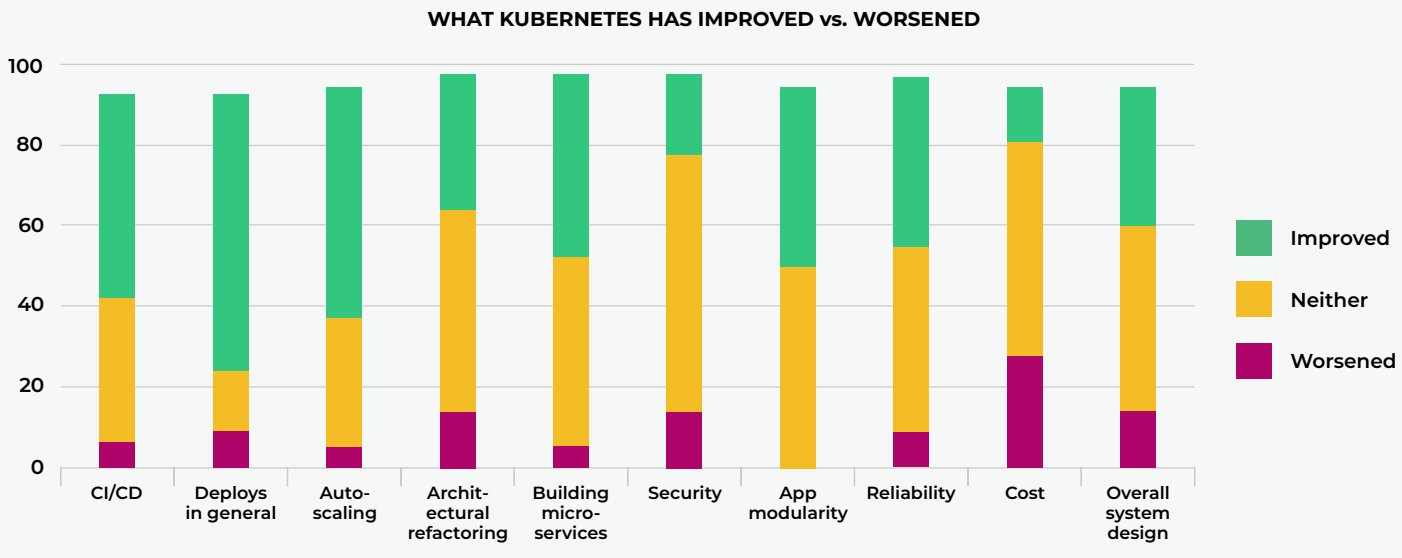
KUBERNETES ADVANTAGES AND DISADVANTAGES

Kubernetes, like any popular software tool, has only gained its tremendous popularity because of the improvements it is able to provide to software and software development. At the same time, using Kubernetes also has its share of potential drawbacks. While these advantages and disadvantages will be perceived differently from developer to developer and organization to organization, we wanted to get a sense of whether there is any consensus on what Kubernetes made better — or worse — at respondents' organizations. We asked:

> *What has Kubernetes improved at your organization?*
>
> and
>
> *What has Kubernetes worsened at your organization?**

Results (*n*=65):

Figure 6



**Note: These questions were only asked to respondents who answered "Yes" to the question, "Does your organization run any Kubernetes clusters?" Additionally, these results ignore any responses where both "Improved" and "Worsened" were selected; < 5% of responses were ignored this way per response option.*

Observations:

1. The most commonly reported improvements arising from Kubernetes use were "Deployment in general" (75%), "Autoscaling" (62%), and "CI/CD" (54%), which were all selected as improvements by more than half of respondents.
2. About half of respondents said that Kubernetes improved "App modularity" (49%), "Building microservices" (48%), and "Reliability" (48%). "Cost" was the most commonly reported disadvantage of Kubernetes (26%), which we discuss further below. Other than "Cost," options regarding what Kubernetes worsened were selected by only 12% of respondents or fewer.
3. Most respondents believed "Security" to be neither improved nor diminished by Kubernetes (62%), a quarter said Kubernetes improved security (25%), and only 12% said that Kubernetes worsened security at their organization. We examine these results further in the next section of this report, "Kubernetes Pain Points."
4. While most respondents also indicated that "Cost" was neither an advantage nor a disadvantage of Kubernetes (52%), "Cost" also had significantly more respondents claiming it as a disadvantage (26%) than any other option, and "Cost" was the only option that more respondents selected as a disadvantage than as an advantage.

    This data, on its own, helps to support the hypothesis that limited resources is a contributing factor to smaller organizations being less likely to use Kubernetes than larger organizations, as we discussed previously. Interestingly, however, respondents at organizations with < 100 employees were significantly less likely to say that Kubernetes worsened costs at their organization (17%) than those at larger companies (33%), and were significantly more likely to say that Kubernetes improved costs (33% at small organizations vs. 17% at large organizations).

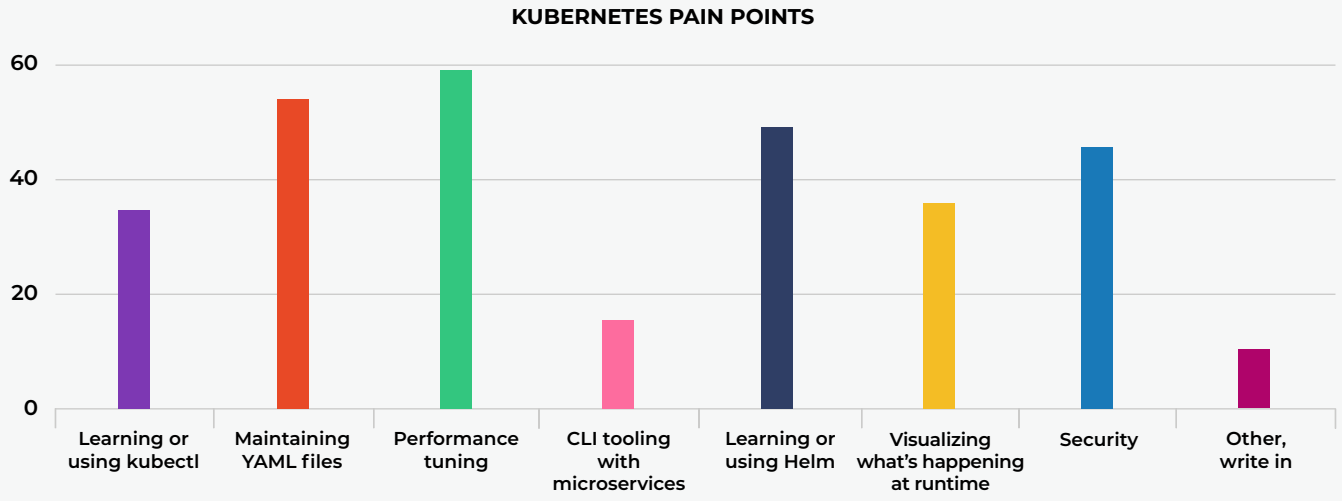
KUBERNETES PAIN POINTS

Beyond the disadvantages to software or development that might be observed, we wanted to know if Kubernetes presents any particularly glaring frustrations for the developers themselves. To find out what their biggest struggles are when working with Kubernetes, we asked respondents:

> *What pain points have you encountered while working with Kubernetes?*

Results (*n*=60):

Figure 7



Observations:

1. The most common Kubernetes pain points were "Performance tuning" (60%), "Maintaining YAML files" (55%), "Learning or using Helm" (45%), and "Security" (44%). On average, respondents selected 3.03 pain points — with a median value of 3 — out of the seven options listed.
2. As mentioned in the previous section, most respondents reported that security at their organization was neither improved nor worsened by Kubernetes, yet close to half of respondents selected "Security" as a pain point.

    Comparing these results, we found that a large majority of respondents who said that Kubernetes worsened security at their organization found security to be a pain point (89%) — which is to be expected. But over half of respondents who said that Kubernetes improved security found it to be a pain point as well (57%). 34% of respondents who believed security was neither improved nor diminished found security to be a pain point. We believe these results may imply

that developers are generally less concerned about the security capabilities of Kubernetes than they are about what it takes to appropriately implement security in Kubernetes clusters.

3. "CLI tooling with microservices" was the least common pain point selected by a significant margin, which is especially noteworthy when considering that 83% of respondents said that their organization runs microservices deployed on Kubernetes clusters.

Research Target Three: Kubernetes Practices and Techniques

Motivation: Kubernetes is a rather versatile tool. Earlier, we looked at a wide variety of use cases for which Kubernetes can be employed, but we wanted to know even more about how Kubernetes is being used. For this research target, we explored questions related to the following areas:

- Environments – How often is Kubernetes run in development and production environments?
- Locations – Is Kubernetes being run more often on bare metal or virtual machines (VMs)?
- Workloads – What types of workloads do organizations generally expect their Kubernetes clusters to handle?
- Autoscalers – Which Kubernetes autoscalers are most prevalent?

KUBERNETES ENVIRONMENTS AND LOCATIONS

Firstly, we wanted to get an idea of "where" Kubernetes is being used, in a couple different senses of the word. For one, we tried to discern where in the dev cycle Kubernetes is being run (i.e., how often it runs in development environments vs. production environments). Furthermore, we attempted to find out where Kubernetes clusters are running in cloud environments (i.e., are clusters more frequently run on bare metal, virtual machines, or a combination of both?). So we asked respondents:

Where does your organization run Kubernetes clusters?

and

*Where do your Kubernetes clusters run?**

Results (*n*=70 and *n*=69, respectively):

Figure 8

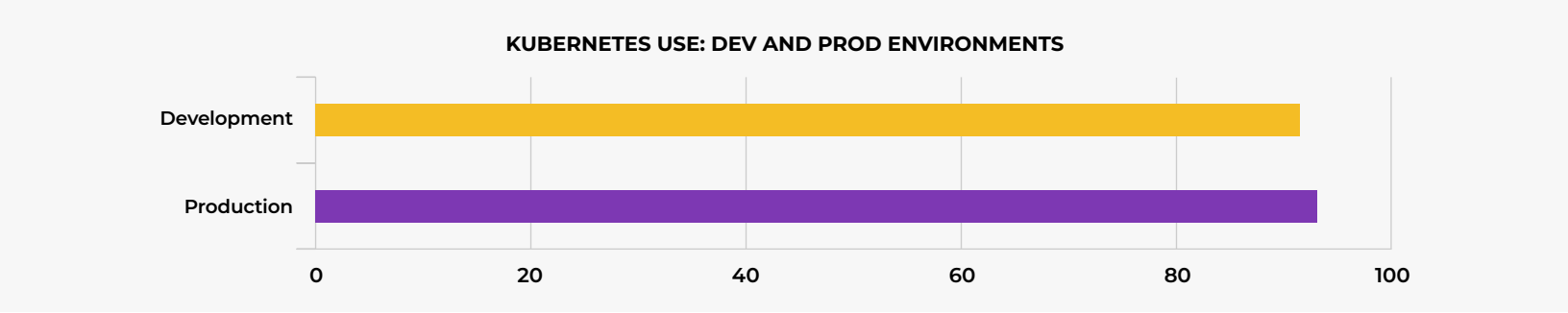
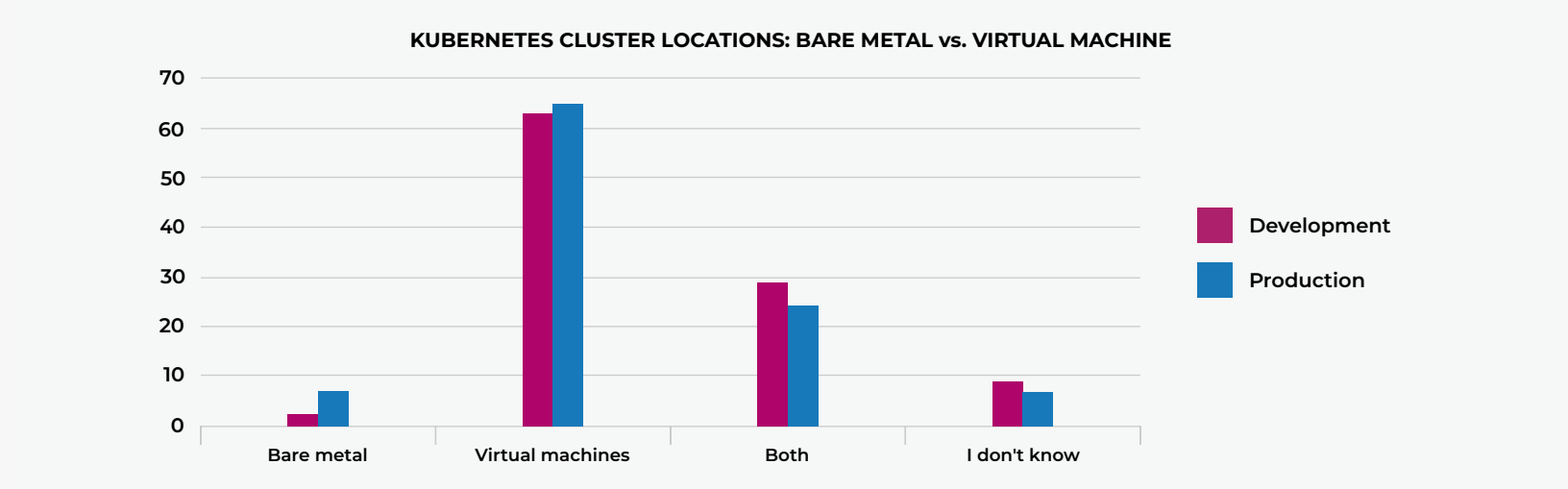


Figure 9



**Note: These questions were only asked to respondents who answered "Yes" to the question, "Does your organization run any Kubernetes clusters?"*

Observations:

1. A large majority of respondents (84%) said their organization runs Kubernetes in both development and production environments. Only 9% of respondents said their organization runs Kubernetes *solely* in production, and only 7% said their organization ran Kubernetes *solely* in development.

   The extremely low sample sizes of respondents reporting development-only and production-only Kubernetes use made the confidence levels for cross tabulation with other data too low for us to report on here. However, we believe this data indicates a general consensus across organizations and software professionals that Kubernetes is sufficiently mature for use throughout the SDLC.

2. Almost all respondents said their Kubernetes clusters run on "Virtual machines" in both development (91%) and production (88%) environments. In development environments, 62% said Kubernetes runs on virtual machines only, and 29% said they have clusters running on both bare metal and VMs. In production, 64% said Kubernetes runs on VMs only, and 23% said it runs on both bare metal and VMs.

   This seems to imply that, more often than not, the ease of use afforded by VMs supersedes any performance gains or orchestration layer simplification that bare metal provides, and that when bare metal is needed, a hybrid bare metal/VM approach will often make more sense than bare metal alone.

KUBERNETES WORKLOADS AND AUTOSCALERS

We observed earlier that performance tuning is a major pain point for developers working with Kubernetes. On the other hand, we saw that one of the biggest improvements developers noticed from Kubernetes use is autoscaling. We wanted to look further into the types of workloads that respondents expect Kubernetes to run performatively, how often different Kubernetes autoscalers are being used, and whether there is any correlation between the two. We asked:

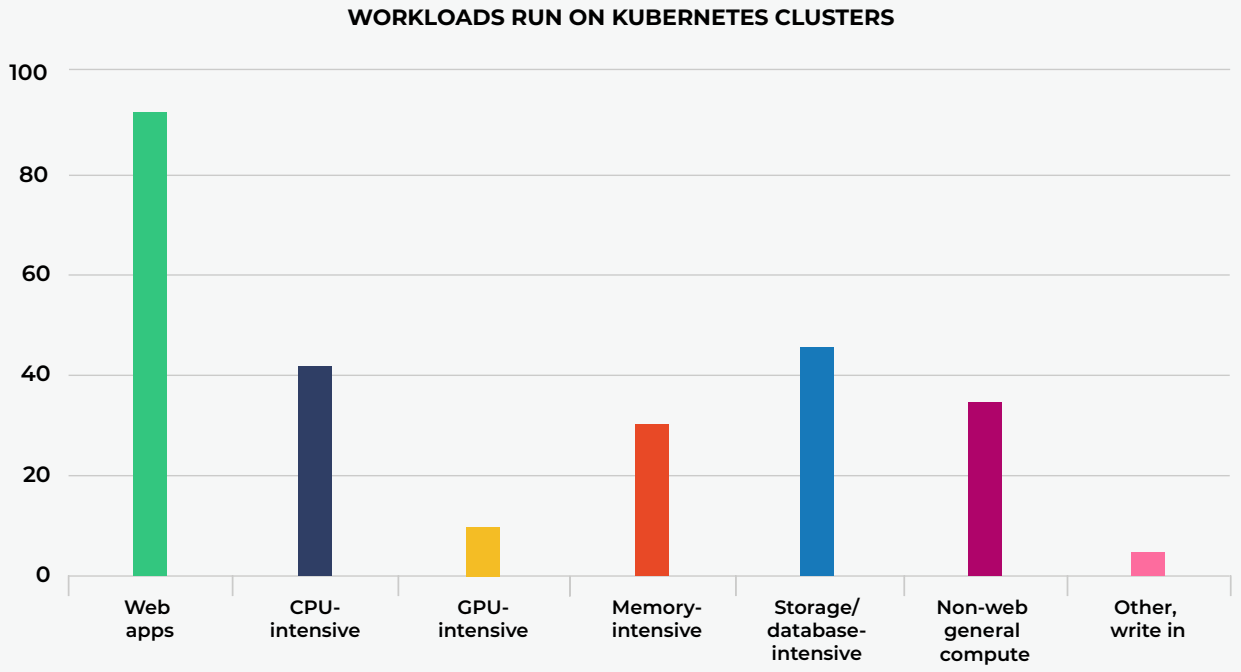
What types of workloads does your organization run on Kubernetes clusters?

and

*What autoscalers does your organization use in its Kubernetes clusters?**

Results (*n*=66):

Figure 10



SEE FIGURE 11 ON NEXT PAGE

Figure 11

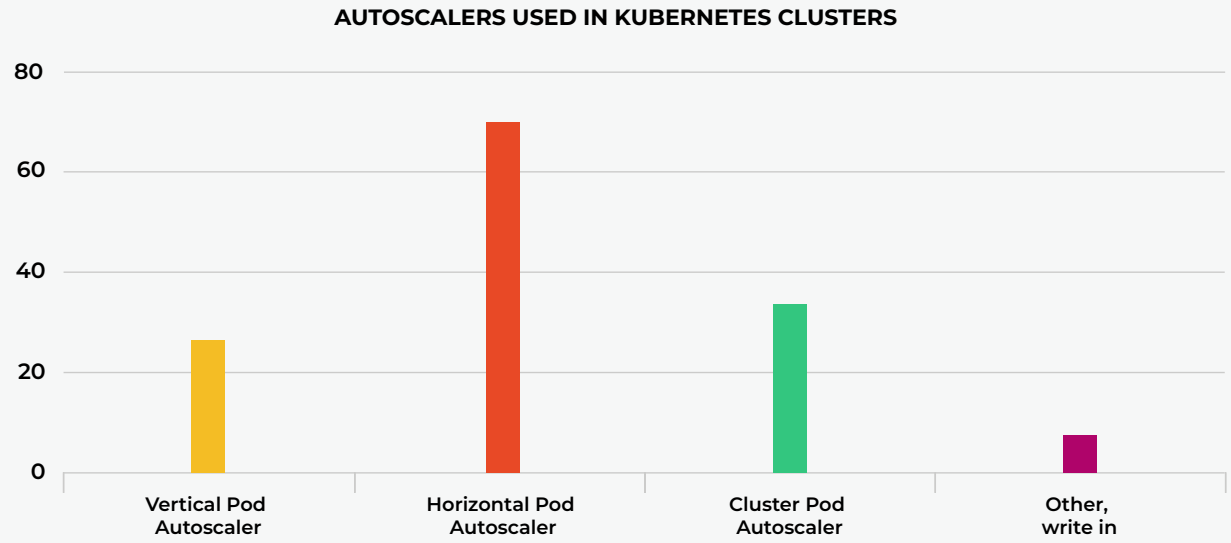


Table 4

WORKLOADS USED WITH AUTOSCALERS*

Workloads	Autoscalers			
	Vertical Pod	Horizontal Pod	Cluster Pod	Avg.
Web apps	86%	95%	95%	93%
CPU-intensive	21%	44%	50%	38%
GPU-intensive	7%	8%	15%	7%
Memory-intensive	29%	33%	50%	30%
Storage/database-intensive	36%	44%	50%	43%
Non-web general compute	36%	49%	50%	38%
n=	14	39	20	56

**Note: % of columns (table). The two aforementioned questions were only asked to respondents who answered "Yes" to the question, "Does your organization run any Kubernetes clusters?"*

Observations:

1. The vast majority of respondents said that their organizations run "Web applications" on Kubernetes (94%), which is unsurprising considering the sheer volume of web apps currently being created or maintained. "Storage/database-intensive" (42%), "CPU-intensive" (41%), and "Non-web general compute" (38%) workloads were moderately popular options. Very few respondents said their organization uses Kubernetes for "GPU-intensive" workloads (9%).
2. Respondents at small organizations (< 100 employees) were significantly less likely than respondents at larger organizations to report that their company uses Kubernetes for "Memory-intensive" workloads (6% vs. 38%) and "CPU-intensive" workloads (24% vs. 48%), though it seems likely that these correlations stem from a higher likelihood that larger organizations' applications deal with these types of workloads in the first place.
3. Horizontal autoscaling (67%) was a far more popular autoscaling method than cluster (35%) or vertical (24%) — a pattern we have seen for the past two years as well.

   This reinforces the hypothesis from our 2021 *Kubernetes in the Enterprise* "Key Research Findings" that, because horizontal autoscalers are "perhaps the most opinionated, least requiring of accurate guessing during cluster configuration," their overwhelming popularity "may suggest that Kubernetes is being used to pluck relatively low-hanging cluster management fruit."

4. Respondents who said that their organizations use vertical autoscalers were much less likely to report that their Kubernetes clusters run "CPU-intensive" (21%), "Storage/database-intensive" (36%), and "Non-web general compute" (36%) workloads compared to respondents at organizations using horizontal (44%, 44%, and 49%, respectively) and cluster (50% each) autoscalers.
5. Additionally, respondents at organizations using cluster autoscalers were much more likely to use Kubernetes for "Memory-intensive" workloads (50%) than respondents at organizations using horizontal (33%) and vertical (29%) autoscalers.

Future Research

Our analysis here only touched the surface of the available data, and we will look to refine and expand our Kubernetes survey as we produce further Trend Reports. Some of the topics we didn't get to in this report, but were incorporated in our survey, include:

- Kubernetes constructs for maintaining state
- Kubernetes security threats
- Kubernetes monitoring and monitoring tools
- Application vs. configuration code in Kubernetes
- Usage/traffic patterns served by Kubernetes clusters

Please contact publications@dzone.com if you would like to discuss any of our findings or supplementary data.

G. Ryan Spain, Freelance Software Engineer, former Engineer & Editor at DZone
@grspain on DZone, GitHub, and GitLab | gryanspain.com

G. Ryan Spain lives on a beautiful two-acre farm in McCalla, Alabama with his lovely wife and adorable dog. He is a polyglot software engineer with an MFA in poetry, a die-hard Emacs fan and Linux user, a lover of The Legend of Zelda, a journeyman data scientist, and a home cooking enthusiast. When he isn't programming, he can often be found playing Super Auto Pets with a glass of red wine or a cold beer.

PARTNER CASE STUDY

Case Study: Jasper.ai

Agile DevOps Strategy Relies on Engineers Having Instant Access to Resources

Challenge

Jasper.ai, a leader in AI content development, was grappling with the complexities of rapid growth and technological advancement. The company was under pressure to accelerate the delivery of enhancements to its AI content platform, which is crucial for staying ahead in a highly competitive industry.

Additionally, Jasper.ai aimed to empower its rapidly expanding engineering team, which grew from just five to more than 70 members within a year, with instant access to DevOps environments. This was essential for the team to build new products and features at a pace that matched the company's ambitious growth targets. These challenges were not just roadblocks but also significant obstacles that threatened to undermine Jasper.ai's competitive edge in the fast-paced, ever-evolving AI sector.

Solution

To address these challenges, Jasper.ai joined forces with Teleport, investing in their enterprise solution and seamlessly integrating it with existing cloud-native applications like Argo and Kubernetes clusters.

This collaboration involved the diligent efforts of Jasper.ai's DevOps and engineering teams over a three-month period, culminating in the successful implementation of Teleport's Kubernetes Access and identity-native proxy. Not only did this replace their conventional VPNs, but it also facilitated seamless authentication to internal web applications. Consequently, engineers can now access resources from any location without the need for a complex VPN client, enjoying secure and straightforward access to Jasper.ai's growing compute environment.

Results

While the exact metrics are confidential, Jasper.ai's implementation of Teleport's identity-native solution led to:

- Increased security by eliminating VPNs
- Faster development through automated credential provisioning
- Scalability through rapid onboarding and easy expansion to meet growing business needs

Key lessons underscore the critical role of automation for efficiency in high-speed tech landscapes and highlight the limitations of traditional VPNs, which can act as bottlenecks in agile, cloud-native settings. By adopting Teleport, Jasper.ai not only surmounted its initial challenges but also carved out a competitive edge, fortifying its leadership in the AI content development arena.

COMPANY
Jasper.ai

COMPANY SIZE
100+ employees

INDUSTRY
AI Content Development

PRODUCTS USED
Teleport Kubernetes Access, App Access

PRIMARY OUTCOME
Through its partnership with Teleport, Jasper.ai has accelerated development, enhanced security, and enabled rapid scalability, solidifying its industry lead.

"Teleport has automated our new user credential provisioning, saving us time and improving accuracy in a rapidly changing environment."

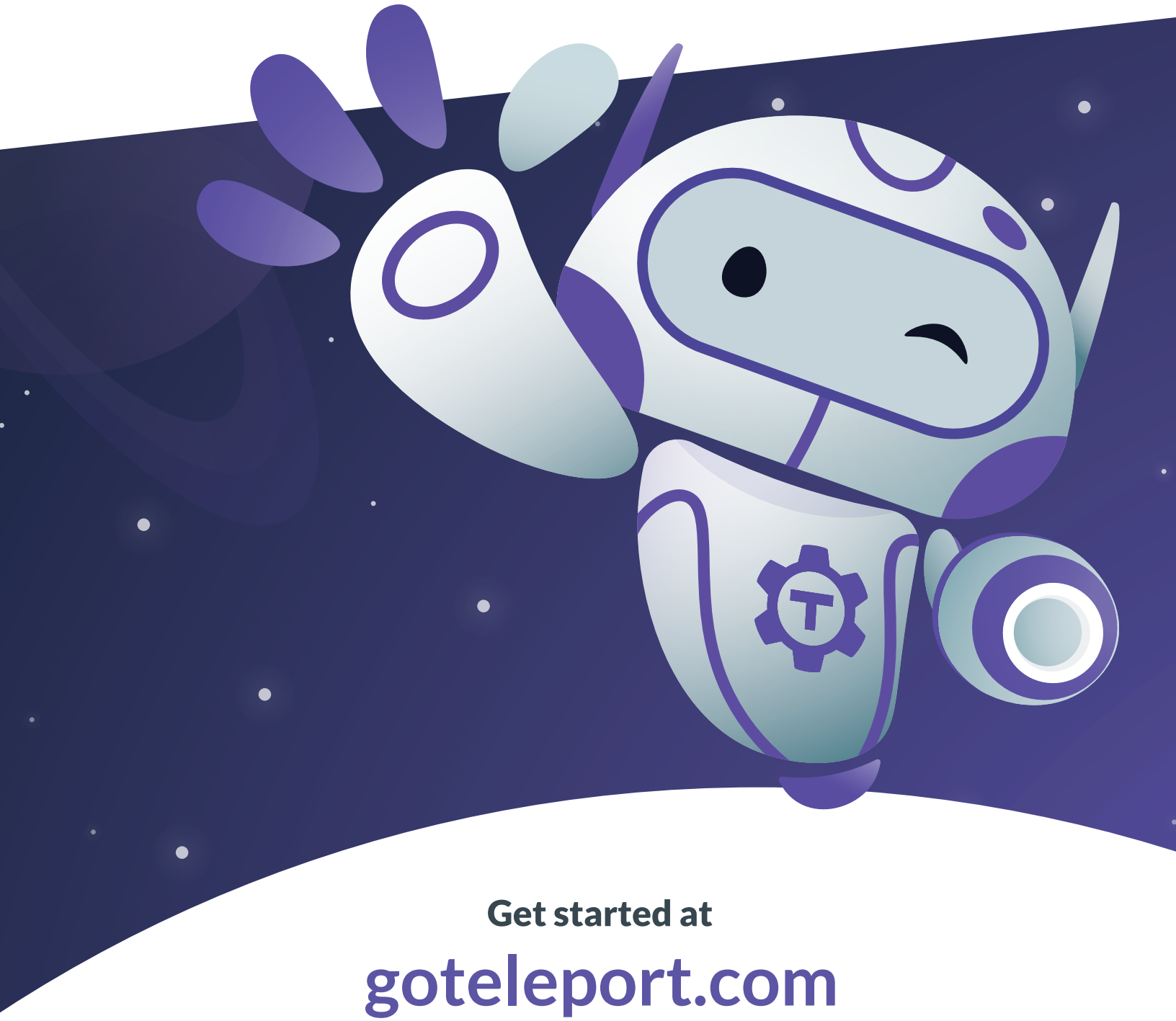
—**John Baird**,
Principal Infrastructure Engineer,
Jasper.ai

CREATED IN PARTNERSHIP WITH

Teleport

Teleport

The easiest, most secure way to access all your infrastructure.



Get started at

gotеleport.com

CONTRIBUTOR INSIGHTS

The State of Kubernetes: Self-Managed vs. Managed Platforms

By Yitaek Hwang, Software Engineer at NYDIG

Kubernetes celebrates its ninth year since the initial release this year, a significant milestone for a project that has revolutionized the container orchestration space. During the time span, Kubernetes has become the de facto standard for managing containers at scale. Its influence can be found far and wide, evident from various architectural and infrastructure design patterns for many cloud-native applications.

As one of the most popular and successful open-source projects in the infrastructure space, Kubernetes offers a ton of choices for users to provision, deploy, and manage Kubernetes clusters and applications that run on them. Today, users can quickly spin up Kubernetes clusters from managed providers or go with an open-source solution to self-manage them. The sheer number of these options can be daunting for engineering teams deciding what makes the most sense for them.

In this Trend Report article, we will take a look at the current state of the managed Kubernetes offerings as well as options for self-managed clusters. With each option, we will discuss the pros and cons as well as recommendations for your team.

Overview of Managed Kubernetes Platforms

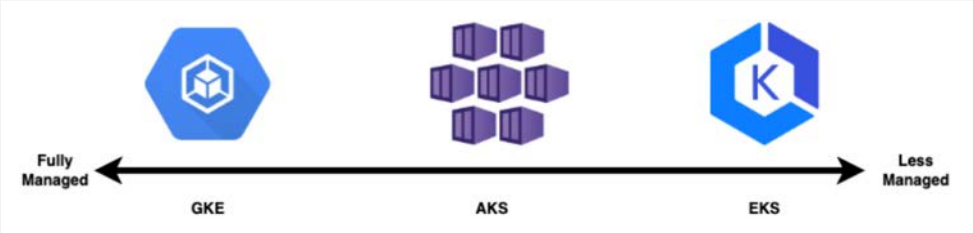
Managed Kubernetes offerings from the hyperscalers (e.g., Google Kubernetes Engine, Amazon Elastic Kubernetes Service, Azure Kubernetes Service) remain one of the most popular options for administering Kubernetes. The 2019 survey of the Kubernetes landscape from the Cloud Native Computing Foundation (CNCF) showed that these services from each of the cloud providers make up three of the top five options that enterprises use to manage containers. More recent findings from CloudZero illustrating increased cloud and Kubernetes adoption further solidifies the popularity of managed Kubernetes services.

All of the managed Kubernetes platforms take care of the control plane components such as `kube-apiserver`, `etcd`, `kube-scheduler`, and `kube-controller-manager`. However, the degree to which other aspects of operating and maintaining a Kubernetes cluster are managed differs for each cloud vendor.

For example, Google offers a more fully-managed service with GKE Autopilot, where Google manages the cluster's underlying compute, creating a serverless-like experience for the end user. They also provide the standard mode where Google takes care of patching and upgrading of the nodes along with bundling autoscaler, load balancer controller, and observability components, but the user has more control over the infrastructure components.

On the other end, Amazon's offering is more of a hands-off, opt-in approach where most of the operational burden is offloaded to the end user. Some critical components like CSI driver, CoreDNS, VPC CNI, and kube-proxy are offered as managed add-ons but not installed by default.

Figure 1: Managed Kubernetes platform comparison



By offloading much of the maintenance and operational tasks to the cloud provider, managed Kubernetes platforms can offer users a lower total cost of ownership (especially when using something like a per-Pod billing model with GKE Autopilot) and increased development velocity. Also, by leaning into cloud providers' expertise, teams can reduce the risk of incorrectly setting Kubernetes security settings or fault-tolerance that could lead to costly outages. Since Kubernetes is so complex and notorious for a steep learning curve, using a managed platform to start out can be a great option to fasttrack Kubernetes adoption.

On the other hand, if your team has specific requirements due to security, compliance, or even operating environment (e.g., bare metal, edge computing, military/medical applications), a managed Kubernetes platform may not fit your needs. Note that even though Google and Amazon have on-prem products (GKE on-prem and EKS anywhere), the former requires VMware's server virtualization software, and the latter is an open-source, self-managed option.

Finally, while Kubernetes lends itself to application portability, there is still some degree of vendor lock-in by going with a managed option that you should be aware of.

Overview of Self-Managed Kubernetes Options

Kubernetes also has a robust ecosystem of self-managing Kubernetes clusters. First, there's the manual route of installing "Kubernetes the Hard Way," which walks through all the steps needed for bootstrapping a cluster step by step. In practice, most teams use a tool that abstracts some of the setup such as `kops`, `kubeadm`, `kubespray`, or `kubicorn`. While each tool behaves slightly differently, they all automate the infrastructure provisioning, support maintenance functions like upgrades or scaling, as well as integrate with cloud providers and/or bare metal.

The biggest advantage of going the self-managed route is that you have complete control over how you want your Kubernetes cluster to work. You can opt to run a small cluster without a highly available control plane for less critical workloads and save on cost. You can customize the CNI, storage, node types, and even mix and match across multiple cloud providers if need be. Finally, self-managed options are more prevalent in non-cloud environments, namely edge or on-prem.

On the other hand, operating a self-managed cluster can be a huge burden for the infrastructure team. Even though open-source tools have come a long way to lower the burden, it still requires a non-negligible amount of time and expertise to justify the cost against going with a managed option.

Table 1

PROS AND CONS OF MANAGED vs. SELF-MANAGED KUBERNETES

Options	Pros	Cons
Managed	• Lower TCO • Increased development velocity • Lean on security best practices • Inherit cloud provider's expertise • Less maintenance burden	• Fully customizable to satisfy compliance requirements • Can use latest features • Flexible deployment schemes
Self-managed	• May not be available on-prem or on the edge • Not open to modification • Requires support from service provider in case of outage	• Requires significant Kubernetes knowledge and expertise • Maintenance burden can be high

Considerations for Managed vs. Self-Managed Kubernetes

For most organizations running predominantly on a single cloud, going with the managed offering makes the most sense. While there is a cost associated with opting for the managed service, it is a nominal fee ($0.10 per hour per cluster) compared to the engineer hours that may be required for maintaining those clusters. The rest of the cost is billed the same way as using VMs, so cost is usually a non-factor. Also, note that there will still be a non-negligible amount of work to do if you go with a vendor who provides a less managed offering.

There are few use cases where going with a **self-managed** Kubernetes option makes sense:

- If you need to run on-prem or on the edge, you may decide that the on-prem offerings from the cloud providers may not fit your needs. If you are running on-prem, likely this means that either cost was a huge factor or there is a tangible need to be on-prem (i.e., applications must run closer to where it's deployed). In these scenarios, you likely already have an infrastructure team with significant Kubernetes experience or the luxury of growing that team in house.
- Even if you are not running on-prem, you may consider going with a self-managed option if you are running on multiple clouds or a SaaS provider that must offer a flexible Kubernetes-as-a-Service type of product. While you can run different variants of Kubernetes across clouds, it may be desirable to use a solution like Cluster API to manage multiple Kubernetes clusters in a consistent manner. Likewise, if you are offering Kubernetes as a Service, then you may need to support more than the managed Kubernetes offerings.

- Also, as mentioned before, compliance may play a big role in the decision. You may need to support an application in regions where major US hyperscalers do not operate in (e.g., China) or where a more locked-down version is required (e.g., military, banking, medical).
- Finally, you may work in industries where there is a need for either cutting edge support or massive modifications to fit the application's needs. For example, for some financial institutions, there may be a need for confidential computing. While the major cloud providers have some level of support for them at the time of writing, it is still limited.

Conclusion

Managing and operating Kubernetes at scale is no easy task. Over the years, the community has continually innovated and produced numerous solutions to make that process easier. On one hand, we have massive support from major hyperscalers for production-ready, managed Kubernetes services. Also, we have more open-source tools to self-manage Kubernetes if need be.

In this article, we went through the pros and cons of each approach, breaking down the state of each option along the way. While most users will benefit from going with a managed Kubernetes offering, opting for a self-managed option is not only valid but sometimes necessary. Make sure your team either has the expertise or the resources required to build it in house before going with the self-managed option.

Additional reading:

- CNCF Survey 2019: Deployments Are Getting Larger as Cloud Native Adoption Becomes Mainstream
- "101+ Cloud Computing Statistics That Will Blow Your Mind (Updated 2023)" by Cody Slingerland, Cloud Zero

Yitaek Hwang, Software Engineer at NYDIG
@yitaek on DZone

Yitaek is a software engineer at NYDIG, applying new cryptographic protocols to improve the custody of Bitcoin. He formerly worked at Axoni and Leverege, mainly building internal development platforms and architecting cloud infrastructure. He writes about cloud, DevOps/SRE, and crypto topics on DZone.

CONTRIBUTOR INSIGHTS

Scaling Up With Kubernetes

Cloud-Native Architecture for Modern Applications

By Saurabh Dashora, Software Architect at Progressive Coder

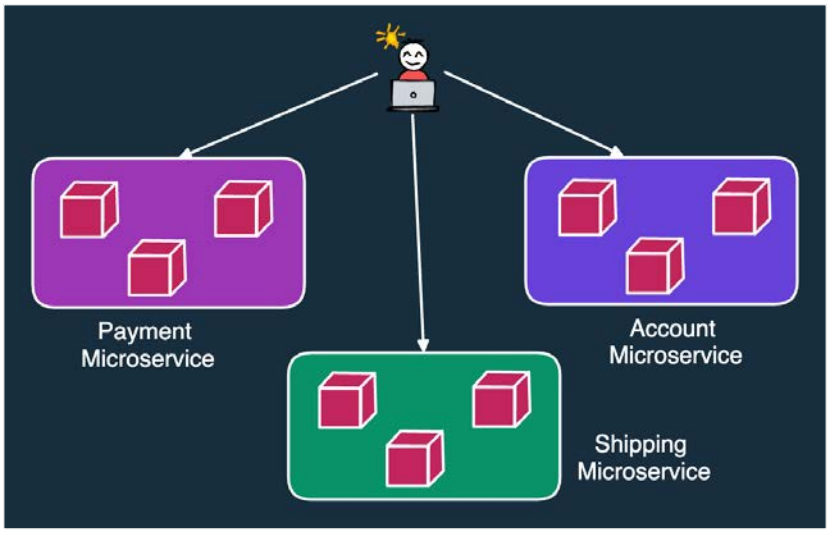
Cloud-native architecture is a transformative approach to designing and managing applications. This type of architecture embraces the concepts of modularity, scalability, and rapid deployment, making it highly suitable for modern software development. Though the cloud-native ecosystem is vast, Kubernetes stands out as its beating heart. It serves as a container orchestration platform that helps with automatic deployments and the scaling and management of microservices. Some of these features are crucial for building true cloud-native applications.

In this article, we explore the world of containers and microservices in Kubernetes-based systems and how these technologies come together to enable developers in building, deploying, and managing cloud-native applications at scale.

The Role of Containers and Microservices in Cloud-Native Environments

Containers and microservices play pivotal roles in making the principles of cloud-native architecture a reality.

Figure 1: A typical relationship between containers and microservices



Here are a few ways in which containers and microservices turn cloud-native architectures into a reality:

- Containers encapsulate applications and their dependencies. This encourages the principle of modularity and results in rapid development, testing, and deployment of application components.
- Containers also share the host OS, resulting in reduced overhead and a more efficient use of resources.
- Since containers provide isolation for applications, they are ideal for deploying microservices. Microservices help in breaking down large monolithic applications into smaller, manageable services.
- With microservices and containers, we can scale individual components separately. This improves the overall fault tolerance and resilience of the application as a whole.

Despite their usefulness, containers and microservices also come with their own set of challenges:

- Managing many containers and microservices can become overly complex and create a strain on operational resources.
- Monitoring and debugging numerous microservices can be daunting in the absence of a proper monitoring solution.
- Networking and communication between multiple services running on containers is challenging. It is imperative to ensure a secure and reliable network between the various containers.

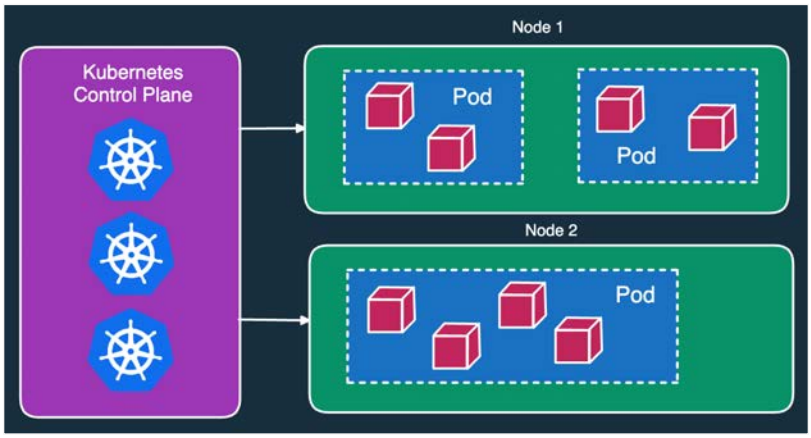
How Does Kubernetes Make Cloud Native Possible?

As per a survey by CNCF, more and more customers are leveraging Kubernetes as the core technology for building cloud-native solutions. Kubernetes provides several key features that utilize the core principles of cloud-native architecture: automatic scaling, self-healing, service discovery, and security.

AUTOMATIC SCALING

A standout feature of Kubernetes is its ability to automatically scale applications based on demand. This feature fits very well with the cloud-native goals of elasticity and scalability. As a user, we can define scaling policies for our applications in Kubernetes. Then, Kubernetes adjusts the number of containers and Pods to match any workload fluctuations that may arise over time, thereby ensuring effective resource utilization and cost savings.

Figure 2: Kubernetes managing multiple containers within the cluster



SELF-HEALING

Resilience and fault tolerance are key properties of a cloud-native setup. Kubernetes excels in this area by continuously monitoring the health of containers and Pods. In case of any Pod failures, Kubernetes takes remedial actions to ensure the desired state is maintained. It means that Kubernetes can automatically restart containers, reschedule them to healthy nodes, and even replace failed nodes when needed.

SERVICE DISCOVERY

Service discovery is an essential feature of a microservices-based cloud-native environment. Kubernetes offers a built-in service discovery mechanism. Using this mechanism, we can create services and assign labels to them, making it easier for other components to locate and communicate with them. This simplifies the complex task of managing communication between microservices running on containers.

SECURITY

Security is paramount in cloud-native systems and Kubernetes provides robust mechanisms to ensure the same. Kubernetes allows for fine-grained access control through role-based access control (RBAC). This certifies that only authorized users can access the cluster. In fact, Kubernetes also supports the integration of security scanning and monitoring tools to detect vulnerabilities at an early stage.

Advantages of Cloud-Native Architecture

Cloud-native architecture is extremely important for modern organizations due to the evolving demands of software development. In this era of digital transformation, cloud-native architecture acts as a critical enabler by addressing the key requirements of modern software development. The first major advantage is **high availability**. Today's world operates 24/7, and it is essential for cloud-native systems to be highly available by distributing components across multiple servers or regions in order to minimize downtime and ensure uninterrupted service delivery.

The second advantage is **scalability** to support fluctuating workloads based on user demand. Cloud-native applications deployed on Kubernetes are inherently elastic, thereby allowing organizations to scale resources up or down dynamically. Lastly, **low latency** is a must-have feature for delivering responsive user experiences. Otherwise, there can be a tremendous loss of revenue. Cloud-native design principles using microservices and containers deployed on Kubernetes enable the efficient use of resources to reduce latency.

ARCHITECTURE TRENDS IN CLOUD NATIVE AND KUBERNETES

Cloud-native architecture with Kubernetes is an ever-evolving area, and several key trends are shaping the way we build and deploy software. Let's review a few important trends to watch out for.

The use of Kubernetes operators is gaining prominence for stateful applications. Operators extend the capabilities of Kubernetes by automating complex application-specific tasks, effectively turning Kubernetes into an application platform. These operators are great for codifying operational knowledge, creating the path to automated deployment, scaling, and management of stateful applications such as databases. In other words, Kubernetes operators simplify the process of running applications on Kubernetes to a great extent.

Another significant trend is the rise of serverless computing on Kubernetes due to the growth of platforms like Knative. Over the years, Knative has become one of the most popular ways to build serverless applications on Kubernetes. With this approach, organizations can run event-driven and serverless workloads alongside containerized applications. This is great for optimizing resource utilization and cost efficiency. Knative's auto-scaling capabilities make it a powerful addition to Kubernetes.

Lastly, **GitOps** and **Infrastructure as Code** (IaC) have emerged as foundational practices for provisioning and managing cloud-native systems on Kubernetes. GitOps leverages version control and declarative configurations to automate infrastructure deployment and updates. IaC extends this approach by treating infrastructure *as* code.

Best Practices for Building Kubernetes Cloud-Native Architecture

When building a Kubernetes-based cloud-native system, it's great to follow some best practices:

- **Observability** is a key practice that must be followed. Implementing comprehensive monitoring, logging, and tracing solutions gives us real-time visibility into our cluster's performance and the applications running on it. This data is essential for troubleshooting, optimizing resource utilization, and ensuring high availability.
- **Resource management** is another critical practice that should be treated with importance. Setting resource limits for containers helps prevent resource contention and ensures a stable performance for all the applications deployed on a Kubernetes cluster. Failure to manage the resource properly can lead to downtime and cascading issues.
- **Configuring proper security policies** is equally vital as a best practice. Kubernetes offers robust security features like role-based access control (RBAC) and Pod Security Admission that should be tailored to your organization's needs. Implementing these policies helps protect against unauthorized access and potential vulnerabilities.
- **Integrating a CI/CD pipeline** into your Kubernetes cluster streamlines the development and deployment process. This promotes automation and consistency in deployments along with the ability to support rapid application updates.

Conclusion

This article has highlighted the significant role of Kubernetes in shaping modern cloud-native architecture. We've explored key elements such as observability, resource management, security policies, and CI/CD integration as essential building blocks for success in building a cloud-native system. With its vast array of features, Kubernetes acts as the catalyst, providing the orchestration and automation needed to meet the demands of dynamic, scalable, and resilient cloud-native applications.

As readers, it's crucial to recognize Kubernetes as the linchpin in achieving these objectives. Furthermore, the takeaway is to remain curious about exploring emerging trends within this space. The cloud-native landscape continues to evolve rapidly, and staying informed and adaptable will be key to harnessing the full potential of Kubernetes.

Additional reading:

- CNCF Annual Survey 2021
- CNCF Blog
- "Why Google Donated Knative to the CNCF" by Scott Carey
- *Getting Started With Kubernetes* Refcard by Alan Hohn
- "The Beginner's Guide to the CNCF Landscape" by Ayrat Khayretdinov

Saurabh Dashora, Software Architect at Progressive Coder
@saurabh.dashora on DZone | @saurabh-dashora on LinkedIn

I'm a full-stack architect, tech writer, and guest author in various publications. I have expertise building distributed systems across multiple business domains such as banking, autonomous driving, and retail. Throughout my career, I have worked at several large organizations. I also run a tech blog on cloud, microservices, and web development, where I have written hundreds of articles. Apart from work, I enjoy reading books and playing video games.

CONTRIBUTOR INSIGHTS

Kubernetes Today

The Growing Role of Serverless in Modern Kubernetes Clusters

By Gal Cohen, Backend Engineer at Firefly

Kubernetes, a true game-changer in the domain of modern application development, has revolutionized the way we manage containerized applications. Some people tend to think that Kubernetes is an opposing approach to serverless. This is probably because of the management bound in deploying applications to Kubernetes — the node management, service configuration, load management, etc. Serverless computing, celebrated for its autoscaling power and cost-efficiency, is known for its easy application development and operation. Yet, the complexities Kubernetes introduces have led to a quest for a more automated approach — this is precisely where serverless computing steps into Kubernetes.

In this exploration, we'll delve into the serverless trend advantages and highlight key open-source solutions that bridge the gap between serverless and Kubernetes, examining their place in the tech landscape.

Factors Driving Kubernetes' Popularity

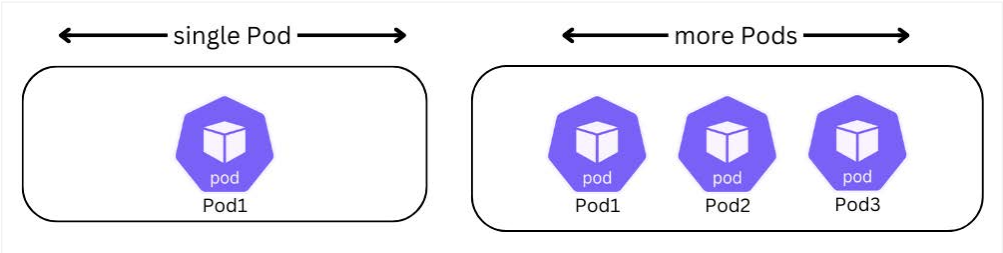
Kubernetes has experienced a meteoric rise in popularity among experienced developers, driven by several factors:

- **Extensibility** – Kubernetes offers custom resource definitions (CRDs) that empower developers to define and manage complex application architectures according to their requirements.
- **Ecosystem** – Kubernetes fosters a rich ecosystem of tools and services, enhancing its adaptability to various cloud environments.
- **Declarative configuration** – Kubernetes empowers developers through declarative configuration, which allows developers to define desired states and lets the system handle the rest.

Kubernetes Challenges: Understanding the Practical Complexities

That being said, experienced developers navigating the intricate landscape of Kubernetes are familiar with the complexities of setting up, configuring, and maintaining Kubernetes clusters. One of the common challenges is scaling. While manual scaling is becoming a thing of the past, autoscaling has become the de facto standard, with organizations who deploy in Kubernetes benefiting from native autoscaling capabilities such as horizontal pod autoscaling (HPA) and vertical pod autoscaling (VPA).

Figure 1: HorizontalPodAutoscaler



Nonetheless, these solutions are not without their constraints. HPA primarily relies on resource utilization metrics (e.g., CPU and memory) for scaling decisions. For applications with unique scaling requirements tied to specific business logic or external events, HPA may not provide the flexibility needed.

Furthermore, consider the challenge HPA faces in scaling down to zero Pods. Scaling down to zero Pods can introduce complexity and safety concerns. It requires careful handling of Pod termination to ensure that in-flight requests or processes are not disrupted, which can be challenging to implement safely in all scenarios.

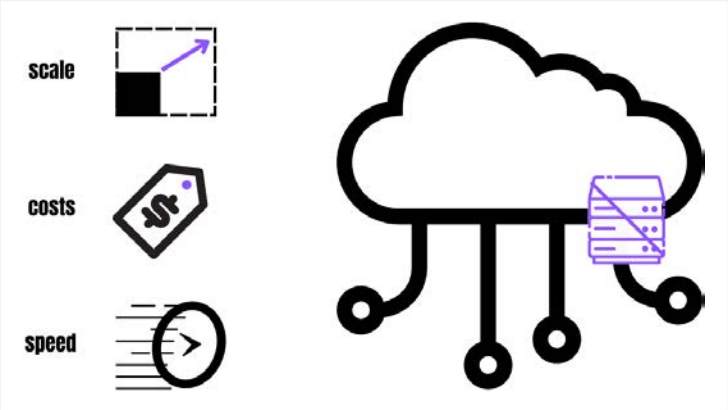
Understanding Serverless Computing

Taking a step back in time to 2014, AWS introduced serverless architectures, which fully exemplified the concept of billing accordingly and using resources only when, how much, and for as long as they're needed. This approach offers two significant

benefits: Firstly, it frees up teams from worrying about how applications run, enabling them to concentrate solely on business matters; secondly, it minimizes the hardware, environmental impact, and thus has a positive financial impact on running applications to the absolute minimum.

It's essential to understand that "serverless" doesn't imply that there is no server. Instead, it means you don't have to concern yourself with the server responsible for executing tasks; your focus remains solely on the tasks themselves.

Figure 2: Serverless advantages



Serverless Principles in Kubernetes

Serverless computing is on the rise, and in some ways, it's getting along with the growing popularity of event-driven architecture, which makes this pairing quite potent.

Event-driven designs are becoming the favored method for creating robust apps that can respond to real-world events in real time. In an event-driven pattern, the crucial requirement is the capability to respond to varying volumes of events at different rates and to dynamically scale your application accordingly. This is where serverless technology perfectly aligns and dynamically scales the application infrastructure accordingly.

When you combine the event-driven approach with serverless platforms, the benefits are twofold: You not only save on costs by paying only for what you need, but you also enhance your app's user experience and gain a competitive edge as it syncs with real-world happenings.

Who Needs Serverless in Kubernetes?

In practical terms, serverless integration in Kubernetes is beneficial for software development teams aiming to simplify resource management and reduce operational complexity. Additionally, it offers advantages to organizations looking to optimize infrastructure costs while maintaining agility in deploying and scaling applications.

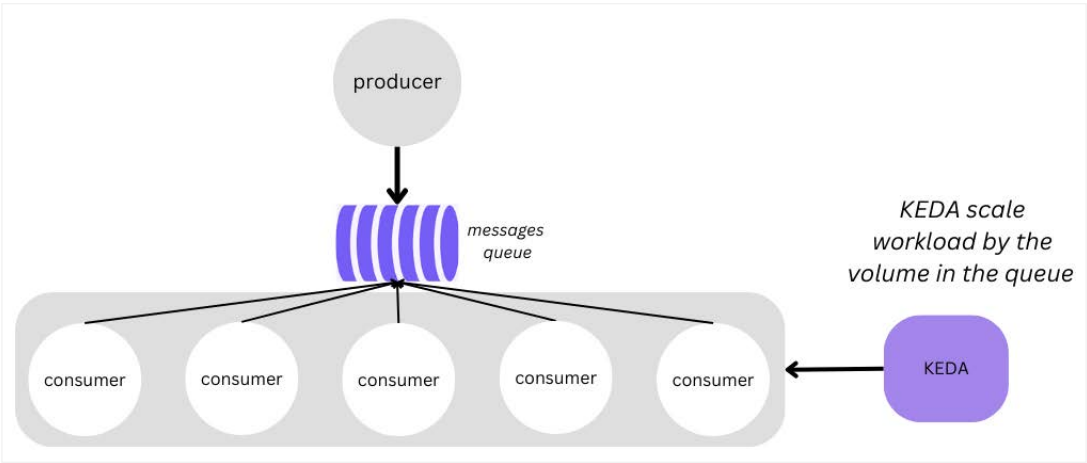
EXPLORE A REAL-WORLD SCENARIO

To illustrate its practicality, imagine a data processing pipeline designed around the producer-consumer pattern. The producer-consumer pattern allows independent operation of producers and consumers, efficient resource utilization, and scalable concurrency. By using a buffer and coordination mechanisms, it optimizes resource usage and ensures orderly data processing. In this architectural context, Kubernetes and KEDA demonstrate significant potential.

PRODUCER-CONSUMER SERVERLESS ARCHITECTURE WITH KEDA

The system works as the following — producers generate data that flows into a message queue, while consumers handle this data asynchronously. KEDA dynamically fine tunes the count of consumer instances in response to changes within the message queue's activity, ensuring optimal resource allocation and performance.

Figure 3: Producer-consumer architecture based on KEDA



This efficient serverless architecture includes:

- **Message queue** – Selected message queue system that is compatible with Kubernetes. Once chosen, it has to be configured to enable accessibility for both producers and consumers.
- **Producer** – The producer component is a simple service that is responsible for generating the tasks and pushing data into the message queue.
- **Consumer** – Consumer applications are capable of pulling data asynchronously from the message queue. These applications are designed for horizontal scalability to handle increased workloads effectively. The consumers are deployed as Pods in Kubernetes and utilized by KEDA for dynamic scaling based on queue activity.
  - It's essential to note that while the application operates under KEDA's management, it remains unaware of this fact. Other than that, in this kind of dynamic scale, it is important to highly prioritize robust error handling, retries, and graceful shutdown procedures within the consumer application to ensure reliability and fault tolerance.
- **KEDA** – The KEDA system contains scalers that are tailored to the message queue and scaling rules that cater to the system's unique requirements. KEDA offers multiple options to configure the events delivery to the consumers on various metrics such as queue length, message age, or other relevant indicators.
  - For example, if we choose setting the `queueLength` as target and if one Pod can effectively process 10 messages, you can set the `queueLength` target to 10. In practical terms, this means that if the actual number of messages in the queue exceeds this threshold, say it's 50 messages, the scaler will automatically scale up to five Pods to efficiently handle the increased workload. Other than that, an upper limit can be configured by the `maxReplicaCount` attribute to prevent excessive scaling.
  - The triggers are configured by the following format:

```
triggers:
- type: rabbitmq
  metadata:
    host: amqp://localhost:5672/vhost
    protocol: auto
    mode: QueueLength
    value: "100.50"
    activationValue: "10.5"
    queueName: testqueue
    unsafeSsl: true
```

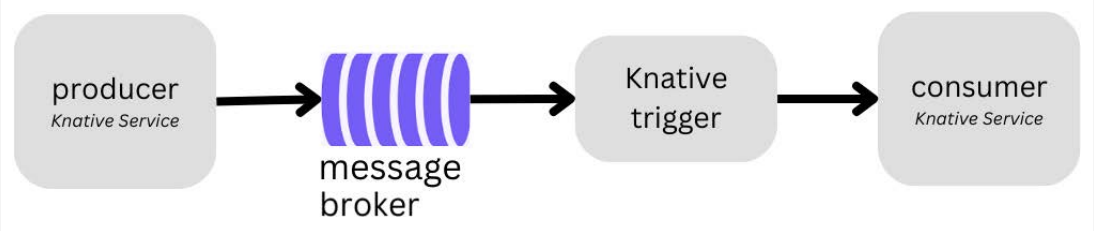
Let's go over this configuration: It sets up a trigger for RabbitMQ queue activity. This monitors the `testqueue` and activates when the queue length exceeds the specified threshold of `100.50`. When the queue length drops below `10.5`, the trigger deactivates. The configuration includes the RabbitMQ server's connection details, using the `auto` protocol detection and potentially unsafe SSL settings. This setup enables automated scale in response to queue length changes.

The architecture achieves an effortlessly deployable and intelligent solution, allowing the code to concentrate solely on essential business logic without the distraction of scalability concerns. This was just an example; the producer-consumer serverless architecture can be implemented through a variety of robust tools and platforms other than KEDA. Let's briefly explore another solution using Knative.

EXAMPLE: ARCHITECTURE BASED ON KNATIVE

The implementation of the Knative-based system distinguishes itself by assuming the responsibility for data delivery management, in contrast to KEDA, which does not handle data delivery and requires you to set up data retrieval. Prior to deployment of the Knative-based system, it is imperative to ensure its environment is equipped with Knative Serving and Eventing components.

Figure 4: Producer-consumer architecture based on Knative



The architecture can include:

- **Message broker** – Selected message queue that seamlessly integrates as a Knative Broker like Apache Kafka or RabbitMQ.
- **Producer** – The producer component is responsible for generating the tasks and dispatching them to a designated message queue within the message broker, implemented as Knative Service.
- **Trigger** – The Knative trigger establishes the linkage between the message queue and the consumer, ensuring a seamless flow of messages from the broker to the consumer service.
- **Consumer** – The consumer component is configured to efficiently capture these incoming messages from the queue through the Knative trigger, implemented as Knative Service.

All of this combined results in an event-driven data processing application that leverages Knative's scaling capabilities. The application automatically scales and adapts to the ever-evolving production requirements of the real world.

Indeed, we've explored solutions that empower us to design and construct serverless systems within Kubernetes. However, the question that naturally arises is: What's coming next for serverless within the Kubernetes ecosystem?

The Future of Serverless in Kubernetes

The future of serverless in Kubernetes is undeniably promising, marked by recent milestones such as KEDA's acceptance as a graduated project and Knative's incubating project status. This recognition highlights the widespread adoption of serverless concepts within the Kubernetes community. Furthermore, the robust support and backing from major industry players underscores the significance of serverless in Kubernetes. Large companies have shown their commitment to this technology by providing commercial support and tailored solutions.

It's worth highlighting that the open-source communities behind projects like KEDA and Knative are the driving force behind their success. These communities of contributors, developers, and users actively shape the projects' futures, fostering innovation and continuous improvement. Their collective effort ensures that serverless in Kubernetes remains dynamic, responsive, and aligned with the ever-evolving needs of modern application development.

In short, these open-source communities promise a bright and feature-rich future for serverless within Kubernetes, making it more efficient, cost-effective, and agile.

Gal Cohen, Backend Engineer at Firefly
@Galco on DZone | @galco5 on LinkedIn

Gal Cohen is a software engineer at Firefly, boasting years of experience in cloud and engineering. She's dedicated to disseminating her DevOps expertise through technical articles, videos, and social media. Her passion lies in DevOps practices and cloud-native technologies. Before joining Firefly, Gal served in the Elite Intelligence unit 8200.

CONTRIBUTOR INSIGHTS

Kubernetes Is Everywhere

By Daniel Stori, Software Development Manager at AWS

Daniel Stori, Software Development Manager at AWS
@Daniel Stori on DZone | @turnoff_us on X (Twitter) | turnoff.us

Passionate about computing since writing my first lines of code in Basic on Apple 2, I share my time raising my young daughter and working on AWS Cloud Quest and AWS Industry Quest, a fun learning experience based on 3D games. In my (little) spare time, I like to make comics related to programming, operating systems, and funny situations in the routine of an IT professional.

CONTRIBUTOR INSIGHTS

From Data to Insights: Kubernetes-Powered AI/ML in Action

Fine-Grained Control, Security, and Elasticity for AI/ML Workloads

By Boris Zaikin, Lead Solution Architect at CloudAstro GmbH

Kubernetes streamlines cloud operations by automating key tasks, specifically deploying, scaling, and managing containerized applications. With Kubernetes, you have the ability to group hosts running containers into clusters, simplifying cluster management across public, private, and hybrid cloud environments.

AI/ML and Kubernetes work together seamlessly, simplifying the deployment and management of AI/ML applications. Kubernetes offers automatic scaling based on demand and efficient resource allocation, and it ensures high availability and reliability through replication and failover features. As a result, AI/ML workloads can share cluster resources efficiently with fine-grained control. Kubernetes' elasticity adapts to varying workloads and integrates well with CI/CD pipelines for automated deployments. Monitoring and logging tools provide insights into AI/ML performance, while cost-efficient resource management optimizes infrastructure expenses. This partnership streamlines the AI/ML development process, making it agile and cost-effective.

Let's see how Kubernetes can join forces with AI/ML.

The Intersection of AI/ML and Kubernetes

The partnership between AI/ML and Kubernetes empowers organizations to deploy, manage, and scale AI/ML workloads effectively. However, running AI/ML workloads presents several challenges, and Kubernetes addresses those challenges effectively through:

- **Resource management** – This allocates and scales CPU and memory resources for AI/ML Pods, preventing contention and ensuring fair distribution.
- **Scalability** – Kubernetes adapts to changing AI/ML demands with auto-scaling, dynamically expanding or contracting clusters.
- **Portability** – AI/ML models deploy consistently across various environments using Kubernetes' containerization and orchestration.
- **Isolation** – Kubernetes isolates AI/ML workloads within namespaces and enforces resource quotas to avoid interference.
- **Data management** – Kubernetes simplifies data storage and sharing for AI/ML with persistent volumes.
- **High availability** – This guarantees continuous availability through replication, failover, and load balancing.
- **Security** – Kubernetes enhances security with features like RBAC and network policies.
- **Monitoring and logging** – Kubernetes integrates with monitoring tools like Prometheus and Grafana for real-time AI/ML performance insights.
- **Deployment automation** – AI/ML models often require frequent updates. Kubernetes integrates with CI/CD pipelines, automating deployment and ensuring that the latest models are pushed into production seamlessly.

Let's look into the real-world use cases to better understand how companies and products can benefit from Kubernetes and AI/ML.

Table 1

REAL-WORLD USE CASES

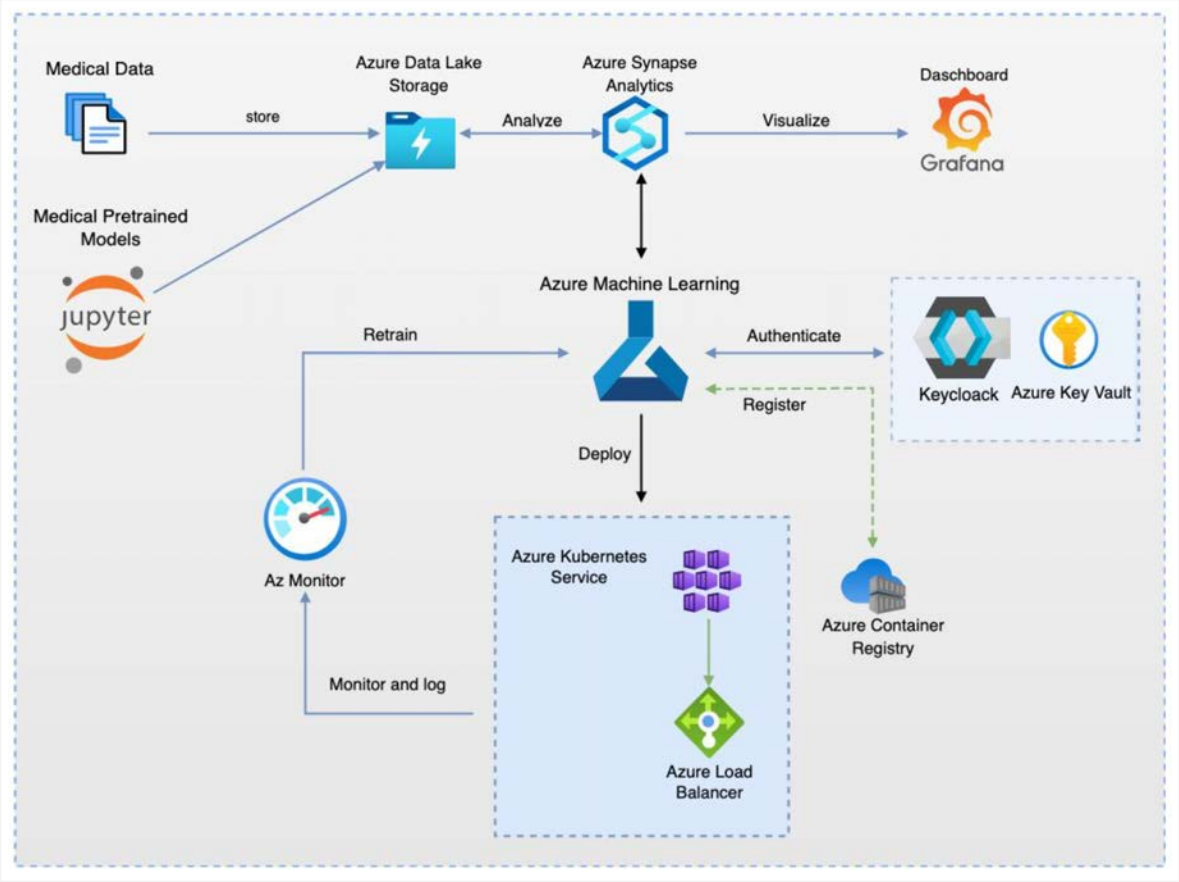
Use Case	Examples
Recommendation systems	Personalized content recommendations in streaming services, e-commerce, social media, and news apps
Image and video analysis	Automated image and video tagging, object detection, facial recognition, content moderation, and video summarization
Natural language processing (NLP)	Sentiment analysis, chatbots, language translation, text generation, voice recognition, and content summarization

REAL-WORLD USE CASES	
Use Case	**Examples**
Anomaly detection	Identifying unusual patterns in network traffic for cybersecurity, fraud detection, and quality control in manufacturing
Healthcare diagnostics	Disease detection through medical image analysis, patient data analysis, drug discovery, and personalized treatment plans
Autonomous vehicles	Self-driving cars use AI/ML for perception, decision-making, route optimization, and collision avoidance
Financial fraud detection	Detecting fraudulent transactions in real-time to prevent financial losses and protect customer data
Energy management	Optimizing energy consumption in buildings and industrial facilities for cost savings and environmental sustainability
Customer support	AI-powered chatbots, virtual assistants, and sentiment analysis for automated customer support, inquiries, and feedback analysis
Supply chain optimization	Inventory management, demand forecasting, and route optimization for efficient logistics and supply chain operations
Agriculture and farming	Crop monitoring, precision agriculture, pest detection, and yield prediction for sustainable farming practices
Language understanding	Advanced language models for understanding and generating human-like text, enabling content generation and context-aware applications
Medical research	Drug discovery, genomics analysis, disease modeling, and clinical trial optimization to accelerate medical advancements

Example: Implementing Kubernetes and AI/ML

As an example, let's introduce a real-world scenario: a medical research system. The main purpose is to investigate and find the cause of Parkinson's disease. The system analyzes graphics (tomography data and images) and personal patient data (which allows the use of the data). The following is a simplified, high-level example:

Figure 1: Parkinson's disease medical research architecture



The architecture contains the following steps and components:

1. **Data collection** – gathering various data types, including structured, unstructured, and semi-structured data like logs, files, and media, in Azure Data Lake Storage Gen2
2. **Data processing and analysis** – utilizing Azure Synapse Analytics, powered by Apache Spark, to clean, transform, and analyze the collected datasets
3. **Machine learning model creation and training** – employing Azure Machine Learning, integrated with Jupyter notebooks, for creating and training ML models
4. **Security and authentication** – ensuring data and ML workload security and authentication through the Key Cloak framework and Azure Key Vault
5. **Container management** – managing containers using Azure Container Registry
6. **Deployment and management** – using Azure Kubernetes Services to handle ML model deployment, with management facilitated through Azure VNets and Azure Load Balancer
7. **Model performance evaluation** – assessing model performance using log metrics and monitoring provided by Azure Monitor
8. **Model retraining** – retraining models as required with Azure Machine Learning

Now, let's examine security and how it lives in Kubernetes and AI/ML.

Data Analysis and Security in Kubernetes

In Kubernetes, data analysis involves processing and extracting insights from large datasets using containerized applications. Kubernetes simplifies data orchestration, ensuring data is available where and when needed. This is essential for machine learning, batch processing, and real-time analytics tasks.

Kubernetes ML analyses require a strong security foundation, and robust security practices are essential to safeguard data in AI/ML and Kubernetes environments. This includes data encryption at rest and in transit, access control mechanisms, regular security audits, and monitoring for anomalies. Additionally, Kubernetes offers features like role-based access control (RBAC) and network policies to restrict unauthorized access.

To summarize, here is a AL/ML for Kubernetes security checklist:

- ☐ **Access control**
  - ☐ Set RBAC for user permissions
  - ☐ Create dedicated service accounts for ML workloads
  - ☐ Apply network policies to control communication
- ☐ **Image security**
  - ☐ Only allow trusted container images
  - ☐ Keep container images regularly updated and patched
- ☐ **Secrets management**
  - ☐ Securely store and manage sensitive data (Secrets)
  - ☐ Implement regular Secret rotation
- ☐ **Network security**
  - ☐ Segment your network for isolation
  - ☐ Enforce network policies for Ingress and egress traffic
- ☐ **Vulnerability scanning**
  - ☐ Regularly scan container images for vulnerabilities

Last but not least, let's look into distributed ML in Kubernetes.

Distributed Machine Learning in Kubernetes

Security is an important topic; however, selecting the proper distributed ML framework allows us to solve many problems. Distributed ML frameworks and Kubernetes provide scalability, security, resource management, and orchestration capabilities essential for efficiently handling the computational demands of training complex ML models on large datasets.

Here are a few popular open-source distributed ML frameworks and libraries compatible with Kubernetes:

- TensorFlow – An open-source ML framework that provides `tf.distribute.Strategy` for distributed training. Kubernetes can manage TensorFlow tasks across a cluster of containers, enabling distributed training on extensive datasets.
- PyTorch – Another widely used ML framework that can be employed in a distributed manner within Kubernetes clusters. It facilitates distributed training through tools like PyTorch Lightning and Horovod.
- Horovod – A distributed training framework, compatible with TensorFlow, PyTorch, and MXNet, that seamlessly integrates with Kubernetes. It allows for the parallelization of training tasks across multiple containers.

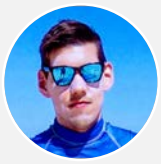
These are just a few of the many great platforms available. Finally, let's summarize how we can benefit from using AI and Kubernetes in the future.

Conclusion

In this article, we reviewed real-world use cases spanning various domains, including healthcare, recommendation systems, *and* medical research. We also went into a practical example that illustrates the application of AI/ML and Kubernetes in a medical research use case.

Kubernetes and AI/ML are essential together because Kubernetes provides a robust and flexible platform for deploying, managing, and scaling AI/ML workloads. Kubernetes enables efficient resource utilization, automatic scaling, and fault tolerance, which are critical for handling the resource-intensive and dynamic nature of AI/ML applications. It also promotes containerization, simplifying the packaging and deployment of AI/ML models and ensuring consistent environments across all stages of the development pipeline.

Overall, Kubernetes enhances the agility, scalability, and reliability of AI/ML deployments, making it a fundamental tool in modern software infrastructure.



Boris Zaikin, Lead Solution Architect at CloudAstro GmbH
@borisza on DZone | @boris-zaikin on LinkedIn

I'm a certified senior software and cloud architect with solid experience designing and developing complex solutions based on the Azure, Google, and AWS clouds. I have expertise in building distributed systems and frameworks based on Kubernetes and Azure Service Fabric. My areas of interest include enterprise cloud solutions, edge computing, high-load applications, multi-tenant distributed systems, and IoT solutions.

CONTRIBUTOR INSIGHTS

Secure the Cluster

A Blazing Kubernetes Developer's Guide

By Akanksha Pathak, Senior Cybersecurity Consultant at Visa, Inc.

Kubernetes security is essential in today's digital landscape. With the increasing adoption of containerization and microservices, Kubernetes has become the go-to solution for orchestrating and managing containers. However, this also means that it has become a target for attackers, making Kubernetes security a top priority. The dynamic and complex nature of Kubernetes requires a proactive and comprehensive approach to security. This involves securing the Kubernetes cluster itself, the workloads running on it, and the entire CI/CD pipeline. It's important to ensure secure configurations, enforce least privilege access, isolate workloads, scan for vulnerabilities regularly, and encrypt sensitive data.

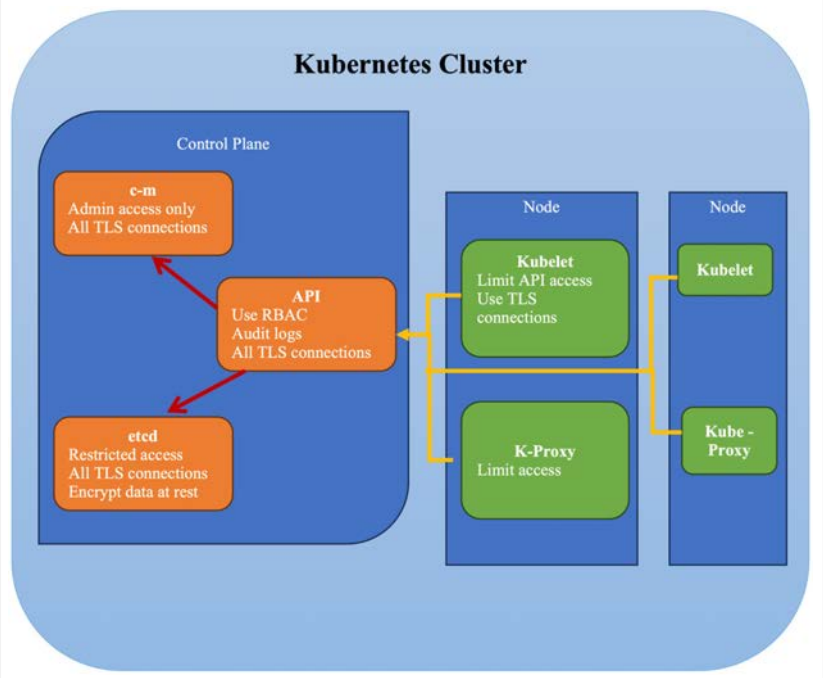
This article will serve as a comprehensive guide to Kubernetes security, aimed at helping developers protect their applications and data.

Important Kubernetes Security Considerations

Before diving into key security considerations, it's crucial to understand the architecture. In Kubernetes, the control plane communicates with nodes via the Kubernetes API, which the API server exposes. Nodes use the kubelet to report back to the control plane and communicate with etcd to read configuration details or write new values.

Kubernetes follows a client-server architecture with two main types of servers: the control plane and the nodes.

Figure 1: Kubernetes cluster with security controls



CONTROL PLANE

The control plane (formerly known as the master node) is responsible for managing the Kubernetes cluster. It is the entry point for all administrative tasks. Components of the control plane include the API server, controller manager and scheduler, and etcd.

Important security controls for each are as follows:

- **API server** (`kube-apiserver`) – Use role-based access control (RBAC) to limit who can access the API server and what actions they can perform. Enable audit logs to track and analyze every request made to the API server. Use transport layer security (TLS) for all API server traffic.
- **Controller manager** (`kube-controller-manager`) and **scheduler** (`kube-scheduler`) – These components should only be accessible by administrators. Use TLS for connections and ensure they are only accessible over the local network.
- **etcd** – This is one of the most critical components from a security perspective, as it stores all cluster data. It should be accessible only by the API server. Protect it with strong access controls and encryption, both in transit and at rest.

NODES

Nodes (formerly known as worker nodes) run the actual workloads. Each node contains the necessary services to manage networking between containers, communicate with the control plane, and assign resources to containers. Components of a node include the kubelet, kube-proxy, and container runtime.

Below are security controls to consider:

- **kubelet** – The kubelet can be a potential attack surface. Limit API access to the kubelet and use TLS for connections.
- **kube-proxy** – This component should be secured by ensuring it can only be accessed by the control plane components.
- **Container runtime** – Ensure you're using secure, up-to-date container images. Regularly scan images for vulnerabilities. Use Pod Security Admission to limit a container's access to resources.

PODS

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod encapsulates an application's container (or multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.

The following are security controls to consider:

- Use namespaces to isolate your Pods from each other.
- Implement network policies to control which Pods can communicate with each other.
- Limit the privileges of a Pod to only what it needs to function.
- Use Kubernetes Secrets to manage sensitive information that Pods need to access.
- Ensure your application code is secure. Even the most secure Kubernetes configuration can't protect against application-level vulnerabilities.

General Guidelines for Kubernetes Security

Let's review some general guidelines for Kubernetes security.

KUBERNETES HARDENING

Kubernetes hardening involves implementing robust security measures — including access control, network policies, audit logging, and regular updates — to enhance the resilience and protection of Kubernetes clusters against potential threats and vulnerabilities.

- **RBAC** – Implement RBAC to regulate access to your Kubernetes API. Assign the least privilege necessary to users, groups, and service accounts. Kubernetes itself provides RBAC as a built-in mechanism for access control.
- **Network policies** – Define network policies to dictate which Pods can communicate with each other. This acts as a basic firewall for your Pods. You can use Project Calico or Cilium for implementing network policies.
- **etcd security** – Configure etcd peer-to-peer communication and client-to-server communication with mutual TLS. Enable etcd's built-in authentication and RBAC support.
- **Audit logging** – Enable audit logging in the API server using the `--audit-log-path` flag. Define your audit policy to record the necessary level of detail. Fluentd or Fluent Bit are often used for processing Kubernetes audit logs.
- **Update and patch** – Regularly apply patches and updates to your Kubernetes components to protect against known vulnerabilities using the Kubernetes built-in mechanism.
- **Admission controllers** – Admission controllers are built-in plugins that help govern how the cluster is used. Enable specific admission controllers like `AlwaysPullImages` to ensure images are always pulled from the registry, and `DenyEscalatingExec` to prevent granting a Pod more privileges than its parent.

DEVSECOPS AND KUBERNETES SECURITY

DevSecOps aims to integrate security practices into the DevOps process. It involves introducing security earlier in the lifecycle of application development, rather than relying on end-stage security measures. In a Kubernetes environment, DevSecOps could involve:

- **Secure CI/CD pipelines** – CI/CD pipelines are commonly used in Kubernetes deployments. A DevSecOps approach ensures that these pipelines are secure and free of vulnerabilities by integrating security checks and tests at every step of the pipeline. Use practices like static code analysis, dynamic analysis, and dependency checks at the coding and building stages.
- **Configuration management** – Kubernetes configurations can be complex, and misconfigurations can lead to security vulnerabilities. DevSecOps practices involve managing and reviewing these configurations continuously to ensure security. Use automated configuration management tools, like Ansible or Terraform, to ensure consistent and secure configurations. Regularly audit and update configurations as necessary.

- **Image scanning** – Container images used in Kubernetes should be scanned for vulnerabilities as part of the CI/CD pipeline. This is a key DevSecOps practice. Use open-source tools like Clair or Grype to regularly scan your container images for known vulnerabilities.
- **Runtime security** – DevSecOps also involves monitoring and securing the application when it's running in the Kubernetes environment. Implement runtime security through tools that can detect anomalous behavior.

SUPPLY CHAIN SECURITY AND KUBERNETES SECURITY

Supply chain security involves securing the software supply chain — from the components used to build your software to the infrastructure and processes used to build and deploy it.

In a Kubernetes environment, supply chain security could involve:

- **Image assurance** – Ensuring that the container images you're using in your Kubernetes deployments are from trusted sources, not tampered with, and free of known vulnerabilities is critical. Use Docker Content Trust or Notary to sign your images and verify signatures before deployment. Use private registries like Harbor or Quay, and secure them using TLS and access controls.
- **Dependency management** – Kubernetes applications will likely depend on external libraries and components. It's important to ensure these dependencies are secure and up to date. Regularly audit your dependencies for vulnerabilities using tools like OWASP Dependency-Check.
- **Secure build processes** – The tools and processes used to build your application and create your container images need to be secure. This could involve securing your CI/CD pipelines and using signed images. Use DevOps tools like Jenkins or CircleCI to ensure they are properly secured and updated.
- **Secrets management** – Safely manage sensitive information such as API keys, passwords, and certificates. Use Kubernetes Secrets or external Secret management tools to store and distribute Secrets securely.

GOVERNANCE

Governance in Kubernetes security ensures the implementation of policies, access controls, and best practices, fostering a secure ecosystem for managing containerized applications and safeguarding sensitive data within Kubernetes clusters.

- **Policy review** – Regularly review and update your security policies to keep them aligned with the latest security best practices and compliance requirements. Tools like `kube-score` or `kube-bench` (Go application that checks whether Kubernetes is deployed securely) can be used to assess how well security policies are being followed.
- **Documentation** – Document all security procedures and ensure your team is aware of them. Use a centralized, version-controlled repository like GitHub for your documentation.
- **Compliance audit** – Regularly audit your cluster for compliance with your security policies. Use tools like `kube-bench` or `kube-score` for automated checks.
- **Namespaces** – Use Kubernetes' built-in namespaces to segregate different projects or teams. Apply RBAC and network policies at the namespace level to enforce access and communication restrictions.
- **Collaborative vendor security** – For third-party services or vendors within your Kubernetes ecosystem, ensure they adhere to robust security practices. Regularly review and validate security protocols to maintain a secure supply chain.

OTHER CONSIDERATIONS

In addition to fundamental security practices, several advanced considerations are vital for a robust Kubernetes security strategy:

- **Monitoring** – Use a comprehensive monitoring solution like Prometheus or Grafana to monitor your cluster. Set up alerts for any signs of suspicious activity.
- **Incident response** – Have an incident response plan in place. This should include steps for identifying, isolating, and mitigating security incidents. The ELK (Elasticsearch, Logstash, Kibana) or EFK (Elasticsearch, Fluentd, Kibana) stacks can be used for log management and analysis during incident response.
- **Backup** – Regularly back up your etcd data using etcd's built-in snapshot feature.
- **Resource quotas** – Use resource quotas and limit ranges to prevent any one application from consuming too many cluster resources.
- **Service mesh** – Consider using a service mesh for additional security and observability features. This can provide mutual TLS, fine-grained traffic control, and detailed telemetry data. Istio and Linkerd are popular open-source service mesh implementations.

Conclusion

Securing Kubernetes is not a one-time effort. As Kubernetes and its ecosystem continue to evolve, so do its security threats. Because of this, it's important to continuously monitor and adapt your security practices. Conducting regular security audits, staying updated with the latest Kubernetes version, and training your team on Kubernetes security are all crucial. Moreover, a strong security culture is key. Everyone involved in the Kubernetes lifecycle — from developers to operators — should be aware of the security best practices and their responsibilities. Security should be a shared responsibility across the organization.

To summarize, Kubernetes security is essential and requires a continuous, proactive approach. By combining robust security practices with a strong security culture, organizations can leverage Kubernetes' benefits while minimizing security risks.

Akanksha Pathak, Senior Cybersecurity Consultant at Visa, Inc.
@pathakakanksha on DZone | @akankshapathak1991 on LinkedIn | akankshapathak.com

Akanksha specializes in cloud and application security, TDR, and vulnerability management. As a senior member of the corporate governance team, she oversees the third-party cybersecurity. Her expertise lies in managing relationships while also architecting and analyzing application designs. Additionally, she is an active participant in cybersecurity communities like GIAC Advisory Board and IEEE. View her professional website to learn more.

CONTRIBUTOR INSIGHTS

Optimizing Kubernetes Costs With FinOps Best Practices

By Sudip Sengupta, Technical Writer at Javelynn

The financial intricacies of Kubernetes deployments demand more than reactive measures alone. Organizations have a choice: react to costs as they arise or employ FinOps (financial operations) practices to anticipate and manage expenditures proactively. Yet the road to efficient Kubernetes FinOps is far from one-dimensional. It's an ever-evolving practice that must be fine-tuned according to operational realities and architectural demands. If a certain cost model continues to yield returns without overwhelming resources, perhaps it's due for scaling. Conversely, a recurring budgetary shortfall may signal the need for an extensive financial overhaul.

In this article, we delve into the multifaceted complexities of a distributed Kubernetes ecosystem and cost implications. We also discuss the recommended FinOps practices for Kubernetes that offer guidance on their seamless integration into overarching financial and operational frameworks.

Inherent Complexities of Kubernetes Costs

Venturing into the multi-faceted, distributed landscape of Kubernetes, one soon realizes that its cost dynamics are far from straightforward. These nuanced cost elements give rise to specific challenges in fiscal oversight that demand targeted exploration. Some cost management challenges in a Kubernetes ecosystem include cluster distribution, microservices architecture, resource heterogeneity, multi-tenancy, and compliance and security.

CLUSTER DISTRIBUTION

Traditional, centralized-data-center models are now less relevant. Instead, deploying Kubernetes clusters across multiple regions and cloud providers is the standard approach. While this aids in high availability and fault tolerance, it introduces financial nuances as regional variations in resource pricing can skew budget forecasts. The crux of the challenge lies in regional resource pricing variances and the costs associated with data egress — often hidden fees that only surface when closely scrutinized. Additionally, latency between clusters can result in performance issues, necessitating more robust — and costly — solutions to maintain service levels.

MICROSERVICES ARCHITECTURE

Besides being an architectural pivot, microservices can often result in a considerable shift in your expense structure. Disaggregating a monolith into microservices requires each service needing its own set of resources and policies for autoscaling, resiliency, and network Ingress/egress. This disintegration amplifies the volume of Pods and containers, each becoming its own line item on your budget. Service meshes, such as Istio or Linkerd, which are used to facilitate inter-service communication, add an extra layer of complexity and ultimately lead to higher costs.

RESOURCE HETEROGENEITY

Kubernetes helps you orchestrate a variety of resource types, including VM-based workloads, serverless functions, or managed databases. The diversity is considered great for performance; however, since each resource type comes with its own pricing model, the heterogeneity complicates the precise correlation of resource usage and cost allocation. In addition, not all resources are billed the same way — some might incur costs per request, others per minute or per GB of data transferred. This fragmentation calls for advanced tagging and granular monitoring tools to demystify your operational expenses.

MULTI-TENANCY

As enterprises scale, the practice of sharing Kubernetes cluster resources among multiple teams or projects — known as multi-tenancy — becomes more prevalent. While this strategy can be cost-efficient, it raises concerns around security and isolation. Resource quotas and limits must be set to prevent a noisy neighbor problem, where one team's activities are limited to consume resources of others. Isolated namespaces can help, but what about shared costs like cluster-level logging or monitoring? This balancing act ultimately has its own cost implications, making it vital to monitor usage carefully to ensure equitable distribution of costs among tenants.

COMPLIANCE AND SECURITY

Operating in a regulated environment adds a recurring financial burden to your Kubernetes setup. Regulations like GDPR and HIPAA mandate not just encryption but end-to-end data protection measures that extend beyond basic compliance checklists. These requirements necessitate the adoption of specialized tools, mostly third-party services, designed for secure data handling, auditing, and logging. Each of these tools or services adds its own cost layer, complicating your FinOps strategy.

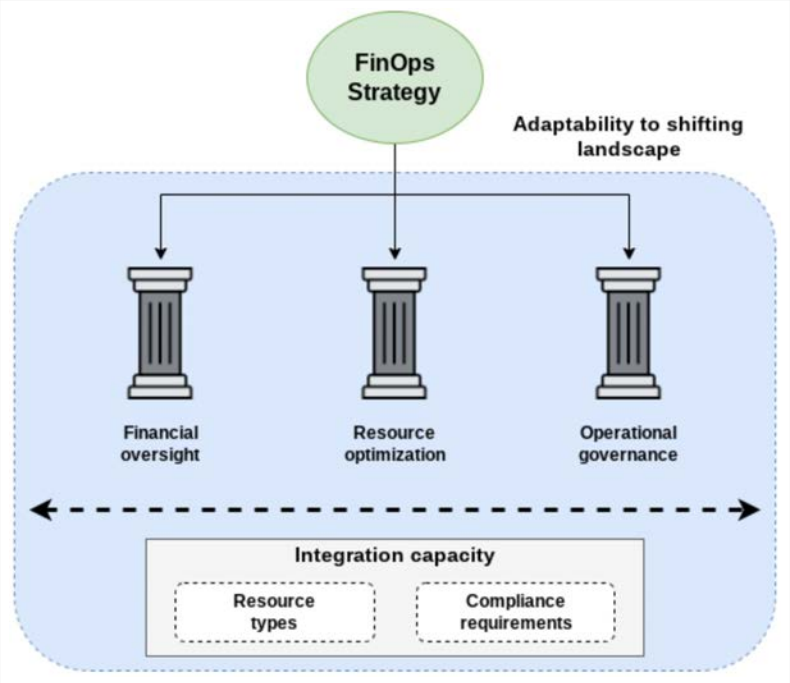
Implementing FinOps for Efficient Finance Governance

When resource heterogeneity and regional pricing variations complicate the cost equation, visibility becomes paramount. FinOps bridges the gap between IT and finance, empowering teams to derive more value from their cloud spend.

Although the foundational pillars of FinOps center around financial oversight, resource optimization, and operational governance, there are several other factors that strengthen its effectiveness.

The real efficacy of a FinOps strategy lies in its adaptability to shifting operational landscapes and its capacity to integrate disparate elements — be it the heterogeneity of resource types or the intricacies of compliance requirements.

Figure 1: Foundational pillars of FinOps



MONITORING AND VISIBILITY

FinOps advocates for transparent, real-time reporting that enables you to monitor not just your total cloud expenditure, but the granular costs associated with each cluster, node, or even Pod. Real-time monitoring ensures that resource utilization aligns with allocated budgets and allows for proactive scaling decisions. If a service is consuming resources inefficiently, real-time tracking provides the intelligence needed to rectify the issue before it escalates into a financial burden.

Adopt advanced tagging and cost allocation methods for attributing costs to specific projects, departments, or teams. Once metrics are scoped, the next logical step is to delve into the tools built to track them effectively.

The following table lists some open-source FinOps tools. Each tool brings its own set of capabilities and focuses on distinct metrics that are essential to measure both financial and operational benchmarks. A typical approach is to integrate them together to form a robust, open-source stack for FinOps in Kubernetes environments.

Table 1

OPEN-SOURCE TOOLS FOR COST TRACKING IN KUBERNETES

Tool	Key Features	Critical Metrics
Kubecost	• Offers visibility into Kubernetes spending • Tracks CPU, memory, and storage usage • Compatible with multiple cloud providers	• Cost per namespace • Cost per Pod • Cluster efficiency
Grafana	• Visualization tool for monitoring data • Includes cost-dashboards for Kubernetes • Requires manual setup for financial metrics	• CPU usage • Memory utilization • Disk I/O • Network throughput
Prometheus	• Monitoring tool native to Kubernetes • Captures performance metrics • Customizable to include financial metrics through exporters	• Query performance • Request rate • Response latency • Resource consumption

TABLE CONTINUES ON NEXT PAGE

OPEN-SOURCE TOOLS FOR COST TRACKING IN KUBERNETES

Tool	Key Features	Critical Metrics
Kubernetes Operational View	• Provides a read-only system dashboard for Kubernetes • Useful for tracking resource usage • Lacks in-depth financial analytics	• Node status • Pod distribution
Kubernetes Resource Report	• Reports resource usage • Helps in identifying over-provisioned resources • Not as comprehensive for cost calculations	• CPU allocation • Memory allocation by service
kube-state-metrics	• Exposes raw metrics for Kubernetes objects • Useful for granular cost allocation • Requires other tools for visualization and analysis	• Object counts • Resource quota
Helm plugins	• Simplifies allocating costs tags to deployments and packages • Estimates the cost implications of rolling back to a previous deployment	• Deployment history • Rollbacks • Release tracking • Plugin dependencies

RESOURCE OPTIMIZATION

Resource optimization in a FinOps parlance goes beyond simple cost cutting while helping you extract maximum value from your deployments. Through predictive analytics and continuous performance monitoring, FinOps tools can identify underutilized resources and suggest consolidation. Achieving optimal financial governance in Kubernetes demands a three-pronged approach:

- Over-provisioning a container wastes compute cycles, just as under-provisioning can result in sluggish performance. Consider right-sizing containers and Pods to strike the right balance between cost control and operational efficacy.
- Accumulated idle resources drain budgets without contributing to productivity. Effective management of these dormant assets recaptures value, streamlining your financial operations.
- Scaling of resources should align with demand curves, ultimately ensuring that you pay only for what you actually use. Utilize solutions such as Kubernetes' native HorizontalPodAutoscaler or third-party offerings to dynamically adjust resource allocation.

The following table shows various recommended strategies to optimize resources:

RESOURCE OPTIMIZATION STRATEGIES IN KUBERNETES

Resource Type	Cost Driver	Unit of Measurement	FinOps Optimization Strategies
Nodes	Compute power	CPU cores, RAM	Right-sizing, spot instances
Pods	Compute and storage	CPU, memory, disk	Resource limit/quota settings
Services	Network traffic	Data transfer	Load balancing, caching
Storage	Data retention	GB, IOPS	Dynamic provisioning
Ingress	Data traffic	Requests/sec	Rate limiting, geo-fencing

BUDGET FORECASTING

When it comes to budget forecasting of a Kubernetes setup, the best approach aligns overarching key performance indicators (KPIs) with granular system metrics. This multi-layered approach enriches your financial strategy, adding depth and detail to fiscal planning. Kubernetes namespaces serve as effective categorization tools, categorizing your costs to project-level granularity. Metrics from tools like Prometheus or Grafana can further refine your budget models by providing insights into resource utilization. This facilitates agile budgeting practices, enabling dynamic allocation of funds to projects based on their real-time resource consumption.

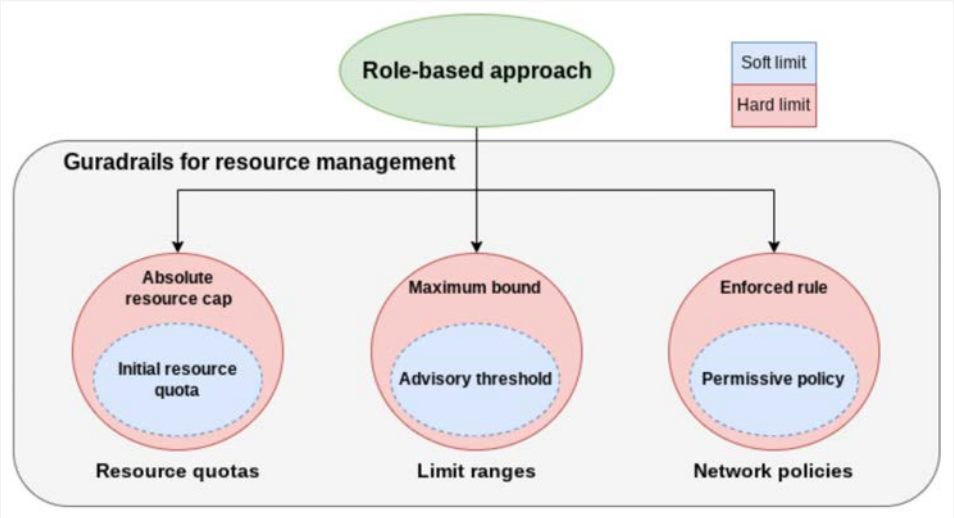
Perhaps the most pivotal aspect of budget forecasting is the integration of system metrics with business KPIs. Metrics such as CPU usage, memory allocation, and I/O operations not only indicate system performance but also translate into quantifiable costs. This integration yields a multi-dimensional financial strategy that accommodates both operational realities and business objectives. For instance, a KPI focused on maximizing application uptime would directly influence budget allocations toward fault tolerance and high availability solutions.

GOVERNANCE AND CONTROL

A clear framework of roles and permissions stands at the core of effective financial governance in Kubernetes. Assigning distinct roles — like developers overseeing deployments within budget confines and financial teams supervising expenditures — enriches your cost visibility. This built-in structure mitigates the risk of uncontrolled spending.

A role-based approach is further strengthened by implementing resource constraints in Kubernetes, using features like resource quotas, limit ranges, and network policies. These guardrails help implement "soft" and "hard" limits to prevent resource overutilization.

Figure 2: Implementing resource constraints



Following the definition of roles and limits, FinOps policies are the pillars upon which everything is built. These hard-coded guidelines act as the governance playbook, aligning both financial planning and operational strategy. From outlining minimum security standards to delineating the resource scaling approval process, these policies act as your rulebook for fiscal control.

Conclusion

The success of a FinOps practice in Kubernetes is shaped by various factors, from distributed services and multi-tenancy to compliance and security. While these complexities bring challenges, they also offer opportunities for refined cost control and performance optimization.

However, mastering these variables requires a continuous process of calibration and readjustment. This doesn't undermine the significance of FinOps practices, though. On the contrary, it emphasizes the need to augment them with specialized tools, granular analytics, and team collaboration. Such a comprehensive stance fosters a culture that prioritizes fiscal prudence, maximizes efficiency, and innovates in the face of Kubernetes's financial complexities.

Resources:

- The FinOps and MLOps platform by OptScale, GitHub
- "Adopting FinOps Tool for Pod-Level Kubernetes Cost Management" by Asaf Liveanu, CNCF

Sudip Sengupta, Technical Writer at Javelynn
@ssengupta3 on DZone and LinkedIn

Sudip Sengupta is a TOGAF Certified Solutions Architect with more than 18 years of experience working for global majors such as CSC, Hewlett Packard Enterprise, and DXC Technology. Sudip now works as a full-time tech writer, focusing on Cloud, DevOps, SaaS, and cybersecurity. When not writing or reading, he's likely on the squash court or playing chess.

ADDITIONAL RESOURCES

Diving Deeper Into Kubernetes

TREND REPORTS

Containers: Modernization and Advancements in Cloud-Native Development

In DZone's 2023 *Containers* Trend Report, we explore the current state of containers, key trends and advancements in global containerization strategies, and constructive content for modernizing your software architecture. This is examined through DZone-led research, expert community articles, and other helpful resources for designing and building containerized applications.

Kubernetes in the Enterprise: Container Management Reimagined

DZone's 2022 *Kubernetes in the Enterprise* Trend Report provides insights into how developers are leveraging Kubernetes in their organizations. It focuses on the evolution of Kubernetes beyond container orchestration, advancements in Kubernetes observability, Kubernetes in AI/ML, and more. Our goal is to help inspire developers to leverage Kubernetes in their own organizations.

DZONE EVENT STREAMS

Tame the Chaos: Make Multi-Cloud Management Secure and Cost Effective [Fireside Chat]

As the number of Kubernetes cloud and cluster deployments proliferates, organizations are experiencing the pain of managing disparate environments. When cluster sprawl is left unchecked, it can introduce all kinds of complexity and excessive cost. This Fireside Chat discusses how to achieve standardization, security, and performance through declarative APIs and GitOps, and more.

Containers: Modernization and Advancements in Cloud-Native Development [Virtual Roundtable]

The need to efficiently manage and monitor containerized environments remains a crucial task for teams. This Virtual Roundtable discusses core strategies and principles for securing container environments, how to tackle the biggest changes and challenges facing containers and cloud-native development today, and the IaC-containers relationship.

REFCARDS

Kubernetes Monitoring Essentials: Exploring Approaches for Monitoring Distributed Kubernetes Clusters

This Refcard presents the primary benefits and challenges of monitoring Kubernetes. You'll also learn the fundamentals of building a Kubernetes monitoring framework, such as how to capture monitoring data insights, leverage core Kubernetes components for monitoring, identify key metrics, and the critical Kubernetes components and services to monitor.

Kubernetes Multi-Cluster Management and Governance

Due to the performance nature of modern cloud-native apps, Kubernetes environments must be highly distributed. Proper multi-cluster management and governance ensure consistent, secure operations across all environments. In this Refcard, we further explore Kubernetes multi-cluster management and governance, why it's important, and core practices for success.

MULTIMEDIA

Kubernetes Podcast from Google [podcast]

Podcast hosts Fields and Sghiouar invite guests to discuss anything Kubernetes-related, from news and updates to history and future predictions. Guests include industry experts who have experience with Kubernetes, and special episodes include KubeCon, Cloud Native Security Con, and more!

Kubernetes Bytes [podcast]

Ryan Wallner's and Bhavin Shah's podcast focuses on new and hot topics in Kubernetes. Featuring developers and engineers from Speedscale, InfluxDB, Redis, Datastax, and more, you will learn from their challenges and experiences in the cloud-native ecosystem.

@kubesimplify [YouTube channel]

Saiyam Pathak, a CNCF ambassador, uses his channel to discuss cloud-native technologies, covering everything from Secrets and security to open-source tools, Kubernetes deep dives, and more. Pathak simplifies cloud native for you so that you don't have to.

Kubernetes Community [YouTube channel and community]

Want to get more involved in Kubernetes? Check out the official Kubernetes-hosted YouTube channel and community page for users and contributors. Watch recordings of meetings for releases, community syncs, testing, APIs, and more. You can also join the community to keep up to date with the latest news and attend meet ups to engage with fellow devs.

ADDITIONAL RESOURCES

Solutions Directory

This directory contains Kubernetes and cloud-native tools to assist with deployment, management, monitoring, and cost optimization. It provides pricing data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY

2023 PARTNERS

Company	Product	Purpose	Availability	Website
Ambassador Labs	Edge Stack API Gateway	Kubernetes-native API gateway	Free tier	getambassador.io/products/edge-stack/api-gateway
Kasten by Veeam	Kasten K10	Kubernetes backup and cloud-native data management	Trial period	kasten.io/product
	Kubestr	Discover, validate, and evaluate Kubernetes storage options	Open source	kubestr.io
Platform9	Platform9 Managed Kubernetes (PMK)	Kubernetes for AI/ML projects for on-prem and colocation infrastructure	By request	platform9.com/managed-kubernetes
Teleport	Teleport Access Platform	Secure infrastructure access	Open source	goteleport.com/kubernetes-access
Company	**Product**	**Purpose**	**Availability**	**Website**
Amazon Web Services	Amazon EKS Distro	Kubernetes distribution built and maintained by AWS	Open source	aws.amazon.com/eks/eks-distro
	Amazon Elastic Container Service	Containers as a Service	By request	aws.amazon.com/ecs
	Amazon Elastic Kubernetes Service (EKS)	Managed service to run Kubernetes on AWS and on-prem		aws.amazon.com/eks
	AWS Fargate	Serverless compute for containers		aws.amazon.com/fargate
Anchore	Anchore Enterprise	Software supply chain management	Trial period	anchore.com
	Grype	Vulnerability scanning for container images and filesystems	Open source	github.com/anchore/grype
Aqua Security	Aqua CNAPP	Cloud-native security platform	By request	aquasec.com/aqua-cloud-native-security-platform
	kube-bench	Kubernetes compliance check with CIS benchmark	Open source	github.com/aquasecurity/kube-bench
Argo	Argo CD	Declarative, GitOps continuous delivery tool	Open source	argoproj.github.io/cd
	Argo Workflows	Kubernetes-native workflow engine		argoproj.github.io/workflows
Backstage	Backstage	Build developer portals	Open source	backstage.io
Buildpacks	Cloud Native Buildpacks	Code transformation to create OCI-compliant containers	Open source	buildpacks.io
Canonical	Juju	Orchestration engine for software operators	Open source	juju.is
CAST AI	CAST AI	Kubernetes automation, optimization, security, and cost management	Free tier	cast.ai
Chaos Mesh	Chaos Mesh	Cloud-native chaos engineering platform	Open source	chaos-mesh.org
Chronosphere	Chronosphere	Cloud-native observability	By request	chronosphere.io

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY				
Company	**Product**	**Purpose**	**Availability**	**Website**
Cilium	Cilium	eBPF-based networking, observability, security	Open source	cilium.io
	Hubble	Network, service, and security observability for Kubernetes		github.com/cilium/hubble
Circle Internet Services	CircleCI	CI/CD platform	Free tier	circleci.com
CloudBees	CloudBees CI	Cloud-native solution for CI delivery at scale	By request	docs.cloudbees.com/docs/cloudbees-ci
CloudEvents	CloudEvents	Specification for describing event data in a common way	Open source	cloudevents.io
CNI	CNI	Networking for Linux containers	Open source	cni.dev
Cockroach Labs	CockroachDB	Distributed SQL database	Free tier	cockroachlabs.com
containerd	containerd	Industry-standard container runtime	Open source	containerd.io
Contour	Contour	Kubernetes Ingress controller	Open source	projectcontour.io
Couchbase	Autonomous Operator	Containerized Couchbase	Free	couchbase.com/products/cloud/kubernetes
CRI-O	CRI-O	Lightweight container runtime	Open source	cri-o.io
Crossplane	Crossplane	Cloud-native control plane framework	Open source	crossplane.io
Crowdstrike	Falcon LogScale Humio Operator	Automate provisioning, management, and operations	Open source	github.com/humio/humio-operator
CubeFS	CubeFS	Cloud-native unstructured data storage	Open source	cubefs.io
D2iQ	D2iQ Kubernetes Platform	Kubernetes platform	Trial period	d2iq.com/kubernetes-platform
	DKP Edge/IoT	Kubernetes platform		d2iq.com/products/edge_iot
	DKP Enterprise	Kubernetes platform		d2iq.com/products/enterprise
	DKP Essential	Kubernetes platform		d2iq.com/products/essential
	Kaptain AI/ML	Kubernetes platform		d2iq.com/products/kaptain
Datadog	Container Monitoring	Monitor and secure containerized environments	Free tier	datadoghq.com/product/container-monitoring
Diamanti	Ultima Accelerator	Kubernetes distribution, storage, security, and I/O acceleration	By request	diamanti.com/products/ultima-accelerator
	Ultima Enterprise	Deploy, operate, and scale Kubernetes	Trial period	diamanti.com/products/ultima-enterprise
DigitalOcean	DigitalOcean Kubernetes	Managed Kubernetes clusters	By request	digitalocean.com/products/kubernetes
Docker	Docker	Container platform	Free tier	docker.com
	Docker Scout	Container development security		docker.com/products/docker-scout
Envoy	Envoy Proxy	Edge and service proxy for cloud-native apps	Open source	envoyproxy.io
etcd	etcd	Distributed key-value store	Open source	etcd.io
F5	Aspen Service Mesh	Istio-based service mesh	By request	f5.com/products/aspen-service-mesh
	NGINX Ingress Controller	Kubernetes-native API gatways, load balancers, and Ingress controllers	Trial period	nginx.com/products/nginx-ingress-controller

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY

Company	Product	Purpose	Availability	Website
Fairwinds	Fairwinds Insights	Kubernetes security and governance	Free tier	fairwinds.com/insights
Falco	Falco	Cloud-native security tool	Open source	falco.org
Fluent Bit	Fluent Bit	End-to-end observability pipeline	Open source	fluentbit.io
Flux	Flagger	Kubernetes progressive delivery operator	Open source	flagger.app
	Flux	Kubernetes continuous delivery		fluxcd.io
Google Cloud	Google Kubernetes Engine (GKE)	Scalable and fully automated Kubernetes service	Trial period	cloud.google.com/kubernetes-engine
Grafana Labs	Grafana Cloud	Analytics and monitoring tool	Free tier	grafana.com/products/cloud
Harbor	Harbor	Manage artifacts across cloud-native compute platforms	Open source	goharbor.io
harvesterhci.io	Harvester	Cloud-native hyperconverged infrastructure	Open source	harvesterhci.io
HashiCorp	Terraform	Automated configuration management tool	Open source	terraform.io
Helm	Helm	Package manager for Kubernetes	Open source	helm.sh
Horovod	Horovod	Distributed deep learning training framework	Open source	horovod.ai
IBM	Cloud Kubernetes Service	Managed Kubernetes platform	Trial period	ibm.com/cloud/kubernetes-service
Istio	Istio	Service mesh for Kubernetes	Open source	istio.io
Jenkins X	Jenkins X	Cloud-native CI/CD built on Kubernetes	Open source	jenkins-x.io
K3s	K3s	Kubernetes distribution for IoT and edge computing	Sandbox	k3s.io
Kanister	Kanister	Framework for app-level data management on Kubernetes	Open source	kanister.io
KEDA	KEDA	Kubernetes-based, event-driven autoscaler	Open source	keda.sh
Keptn	Keptn	Cloud-native application lifecycle orchestration	Open source	lifecycle.keptn.sh
Knative	Knative	Serverless and event-driven app development	Open source	knative.dev
Kong	Ingress Controller	Kubernetes-native API management	Trial period	konghq.com/products/kong-ingress-controller
	Kong Mesh	Service mesh that runs on Kubernetes and VMs	By request	konghq.com/products/kong-mesh
KubeEdge	KubeEdge	Kubernetes-native edge computing framework	Open source	kubeedge.io
Kubernetes	Kubernetes	Container orchestration	Open source	kubernetes.io
KubeVirt	KubeVirt	Build virtualization APIs for Kubernetes	Open source	kubevirt.io
Kuma	Kuma	Envoy service mesh for distributed service connectivity	Open source	kuma.io
Kyverno	Kyverno	Kubernetes-native policy management	Open source	kyverno.io
Lacework	Polygraph® Data Platform	Data-driven CNAPP	By request	lacework.com/platform

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY

Company	Product	Purpose	Availability	Website
Linkerd	Linkerd	Service mesh for Kubernetes	Open source	linkerd.io
Longhorn	Longhorn	Cloud-native distributed block storage for Kubernetes	Open source	longhorn.io
Microsoft Azure	Azure Kubernetes Service (AKS)	Managed Kubernetes platform	Trial period	azure.microsoft.com/en-us/products/kubernetes-service
Mirantis	Mirantis Container Cloud	Container infrastructure management	Trial period	mirantis.com/software/mirantis-container-cloud
	Mirantis Container Runtime	Industry-standard container runtime	By request	mirantis.com/software/container-runtime
	Mirantis Kubernetes Engine	Enterprise container platform	Trial period	mirantis.com/software/mirantis-kubernetes-engine
	Mirantis Secure Registry	Enterprise container registry	By request	mirantis.com/software/mirantis-secure-registry
New Relic	Pixie	Kubernetes observability	Free tier	newrelic.com/platform/kubernetes-pixie
Notary	Notary	Specs and tools for software supply chain security	Open source	notaryproject.dev
Nutanix	Kubernetes Engine	Kubernetes management platform	By request	nutanix.com/products/kubernetes-engine
Okteto	Cloud Development Environments	Developer experience platform	Free tier	okteto.com/development-environments
Ondat	Ondat	Data mesh for block storage	Open source	docs.ondat.io
Open Policy Agent	Open Policy Agent	Policy engine	Open source	openpolicyagent.org
Operator Framework	Operator Framework	Toolkit to manage Kubernetes-native apps	Open source	operatorframework.io
Oracle	Cloud Infrastructure (OCI)	Cloud-native services and software	Free tier	oracle.com/cloud
	OCI Container Engine for Kubernetes	Managed Kubernetes service		oracle.com/cloud/cloud-native/container-engine-kubernetes
	OCI Service Mesh	Managed service mesh for cloud-native apps	Free	oracle.com/cloud/cloud-native/service-mesh
Palo Alto	Prisma Cloud	CNAPP	By request	paloaltonetworks.com/prisma/cloud
PlanetScale	Vitess Operator	Kubernetes operator for Vitess	Open source	github.com/planetscale/vitess-operator
Portainer	Portainer	Container management platform	Trial period	portainer.io
Portworx	Backup	Kubernetes backup and data protection	Free	portworx.com/products/px-backup
	Data Services	Kubernetes DBaaS platform	By request	portworx.com/products/portworx-data-services
	Enterprise	Kubernetes storage platform		portworx.com/products/portworx-enterprise
Prometheus	Prometheus	Cloud-native monitoring and alerting	Open source	prometheus.io
Rafay	Kubernetes Operations Platform	Kubernetes automation for platform teams	Trial period	rafay.co
Rancher	Rancher	Kubernetes management	Free tier	rancher.com/products/rancher

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY

Company	Product	Purpose	Availability	Website
Red Hat	Ansible	Automated configuration management tool	Free tier	ansible.com
	OpenShift	Build, modernize, and deploy apps at scale	By request	redhat.com/en/technologies/cloud-computing/openshift
	OpenShift Container Platform	Build and scale containerized applications	Trial period	redhat.com/en/technologies/cloud-computing/openshift/container-platform
	Quay	Container registry		quay.io
	Quay Clair	Parse image contents and report vulnerabilities	Open source	github.com/quay
Release Technologies	Release Delivery	Container-based EaaS platform	By request	prod.releasehub.com/product/release-delivery
Replicated	Replicated Platform	Kubernetes app delivery and management	Trial period	replicated.com
Rook	Rook	Cloud-native storage orchestration	Open source	rook.io
Snyk	Container	Container and Kubernetes security	Free tier	snyk.io/product/container-vulnerability-management
Solo.io	Gloo Platform	Unified application networking for APIs	Trial period	solo.io/products/gloo-platform
SPIFFE	SPIFFE	Identify and secure communications between app services	Open source	spiffe.io
	SPIRE	API toolchain for trust-building between software systems		github.com/spiffe/spire
Spot by NetApp	CloudCheckr	Cloud cost management	By request	cloudcheckr.com
	Ocean	Serverless infrastructure container engine		spot.io/product/ocean
Stackwatch	Kubecost	Kubernetes cost optimization	Free tier	kubecost.com
Sumo Logic	Sumo Logic	Cloud-native SaaS analytics	Free tier	sumologic.com
SUSE	NeuVector	Container security platform	By request	suse.com/neuvector
Sysdig	Sysdig Platform	CNAPP	By request	sysdig.com/products/platform
TensorFlow	TensorFlow	Create production-grade ML models	Open source	tensorflow.org
The Linux Foundation	PyTorch	Optimized tensor library for deep learning	Open source	pytorch.org
Tigera	Calico Cloud	Security for containers and Kubernetes	Trial period	tigera.io/tigera-products/calico-cloud
	Calico Enterprise	Zero trust security for Kubernetes	By request	tigera.io/tigera-products/calico-enterprise
	Calico Open Source	Networking and security for containers and Kubernetes	Open source	tigera.io/tigera-products/calico
Traefik Labs	Traefik Enterprise	Unified API gateway and Ingress	Trial period	traefik.io/traefik-enterprise
Trilio	Kubernetes Backup	Kubernetes backup and recovery for on-prem or cloud	By request	trilio.io/products/kubernetes-backup-and-recovery
Vitess	Vitess	Database clustering system for horizontal scaling of MySQL	Open source	vitess.io

DZONE'S 2023 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY				
Company	**Product**	**Purpose**	**Availability**	**Website**
VMWare Tanzu	Build Service	Automate container creation, management, governance	By request	tanzu.vmware.com/build-service
	Kubernetes Grid	Kubernetes runtime		tanzu.vmware.com/kubernetes-grid
	Kubernetes Operators	Automated Kubernetes platform operations		tanzu.vmware.com/kubernetes-operations
	Service Mesh	E2E connectivity, security, and insights for modern apps		tanzu.vmware.com/service-mesh
Volcano	Volcano	Cloud-native batch scheduling system for Kubernetes	Open source	volcano.sh
Weaveworks	Weave GitOps Enterprise	Docker, Oracle, SQL Server containers	By request	weave.works/product/gitops-enterprise
Windocks	Windocks	SQL Server containers	Free tier	windocks.com

Visit Daniel's DZone profile for more comics!

DZone

3343 Perimeter Hill Dr, Suite 100
Nashville, TN 37211
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.