DZone

TREND REPORT

SEP 2025

KUBERNETES IN THE ENTERPRISE

Optimizing the Scale, Speed, and Intelligence of Cloud Operations

BROUGHT TO YOU IN PARTNERSHIP WITH

Table of Contents

Meet the Team

DZone's Content and Community team can be found reviewing contributor pieces, working with authors and sponsors, and coordinating with our researcher and designer to deliver valuable, high-quality content to global DZone-ians.

Carisse Dumaua	*Content Editor*
Melissa Habit	*Senior Manager, Production Ops*
Lucy Marcum	*Acquisitions Editor*
Dominique Roller	*Community Specialist*
Ging Villena	*Content Editor*

Welcome Letter

By Lucy Marcum, Acquisitions Editor at DZone

The debate around Kubernetes today seems not to be how useful it is but whether the struggle of implementing the orchestrator is worth it. As we will see in the "Key Research Findings" section, large companies identify Kubernetes as useful, but the experience of smaller companies indicates that the infrastructure concerns far outweigh the benefits.

We all know the decade-old tale of the Kubernetes learning curve; however, as we enter this next decade, the technical complexity is becoming more of a focal point, especially as AI continues to take center stage.

In fact, Kubernetes can be so complicated that when I set a reminder for myself to write this Welcome Letter, Siri recorded the reminder as "Cooper Netties welcome letter." So if Apple Intelligence can't even figure out how to spell it — although let's be honest, Apple Intelligence is a bit behind other AI models — how are companies expected to implement Kubernetes in a seamless manner that makes everybody's lives easier?

This question is coming at a time when data is scaling exponentially by the day thanks to universal adoption of AI across industries.

On the tough, mind-boggling days when it seems like both Kubernetes and AI are increasing complexity, the key thing to focus on is that we don't have to be part of the problem. There are solutions to it all.

In the 2025 *Kubernetes in the Enterprise* Trend Report, the research and Contributor Insights will highlight different implementation tactics to ensure the best Kubernetes experience.

With eye-opening data and an actionable checklist, you'll be able to form your own opinion on whether Kubernetes is worth the heavy lift. You'll also read key insights into how platform engineering, IDPs, and/or self-service platforms just might be the keys to the future of this orchestrator; and you'll discover how to integrate AI Kubernetes environments and address Kubernetes sprawl.

As you peruse this Trend Report and identify useful strategies, we hope you will find a path to implementing Kubernetes at its full potential. Or at the very least, we hope you find an AI model that can understand the word "Kubernetes" when spoken aloud.

Keep on sailing ,

Lucy Marcum

Lucy Marcum

@lucymarcum

Lucy manages the acquisition process and strategy for DZone Publications, from sourcing new contributors and community members to leading them through the editorial review. She also leads content strategy for dzone.com and works with the Community Specialist to create a streamlined contributor experience. Outside of work, Lucy spends her time reading, writing, running, and trying to keep her cats, Olive and Tiger Lily, out of trouble.

ORIGINAL RESEARCH

Key Research Findings

An Analysis of Results From DZone's 2025 Kubernetes Survey

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

Over a decade in, Kubernetes is the central force in modern application delivery. However, as its adoption has matured, so have its challenges: sprawling toolchains, complex cluster architectures, escalating costs, and the balancing act between developer agility and operational control. In addition to navigating the technical demands of running Kubernetes at scale, organizations are also met with the not-so-simple task of realizing the cultural and strategic shifts required to make Kubernetes truly work for their teams.

As the industry pushes toward more intelligent and integrated operations, platform engineering and internal developer platforms are helping teams address issues like Kubernetes tool sprawl, while AI continues cementing its usefulness for optimizing cluster management, observability, and release pipelines.

In August, DZone surveyed software developers, architects, and other IT professionals in order to gain insight on the current state of Kubernetes in the software development space.

Methods

We created a survey and distributed it to a global audience of software professionals. Question formats included mainly single and multiple choice, with options for write-in responses in some instances. The survey was disseminated via email to DZone and TechnologyAdvice opt-in subscriber lists as well as promoted on DZone.com, in the DZone Core Slack workspace, and across various DZone social media channels.

The data for this report were gathered from responses submitted between August 5, 2025 to August 30, 2025; we collected 173 complete and partial responses.

Demographics

Before diving in, we noted certain key audience details in order to establish a more solid impression of the sample from which results are derived:

- 26% of respondents described their primary role in their organization as "Technical architect," 18% described "Developer/engineer," and 13% described "Developer team lead." No other role that we provided was selected by more than 10% of respondents.*
- 83% of respondents said they are currently developing "Web applications/Services (SaaS)," 40% said "Enterprise business applications," and 24% said "Native mobile apps."
- "Python" (65%) was the most popular language ecosystem used at respondents' companies, followed by "Java" (62%), "JavaScript (client-side)" (45%), "Node.js (server-side JavaScript)" (45%), and "TypeScript" (39%).
- Regarding responses on the primary language respondents use at work, the most popular was "Java" (40%), followed by "Python" (22%). No other language was selected by more than 10% of respondents.
- On average, respondents said they have 16.78 years of experience as an IT professional, with a median of 15 years.
- 32% of respondents work at organizations with < 100 employees or reported being self-employed, 23% of respondents work at organizations with 100-999 employees, and 43% of respondents work at organizations with 1,000+ employees.*

Note: For brevity, throughout the rest of these findings we will use the term "developer" or "dev" to refer to **anyone actively involved in the creation and release of software, regardless of role or title. Additionally, we will define "small" organizations as having < 100 employees, "mid-sized" organizations as having 100-999 employees, and "large" organizations as having 1,000+ employees.*

Major Research Targets

In our 2025 Kubernetes survey, we aimed to gather data regarding various topics related to the following major research targets:

1. Kubernetes usage, workloads, and operations
2. Developer experience and tooling
3. Security, compliance, and observability

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

Research Target One: Kubernetes Usage, Workloads, and Operations

Our research related to **Kubernetes usage**, **workloads**, and **operations** two key topic areas:

1. Kubernetes usage and use cases
2. Workloads and application architectures

Kubernetes Usage and Use Cases

We asked the following questions:

- *Have you personally worked with Kubernetes?*
- *Does your organization run any Kubernetes clusters?*
- *What use cases does your organization deploy Kubernetes into?**

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

Results:

Figure 1. Personal experience with Kubernetes [*n*=149]

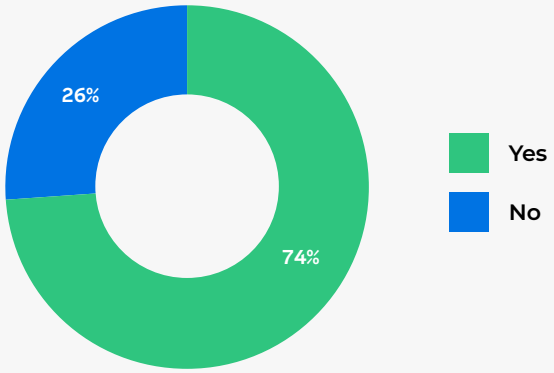


Figure 2. Organizations running Kubernetes clusters [*n*=155]

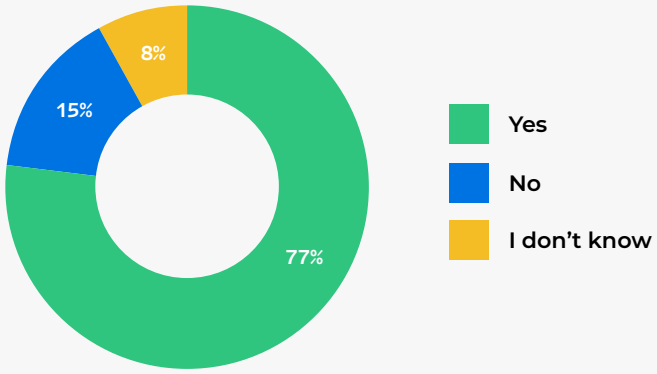
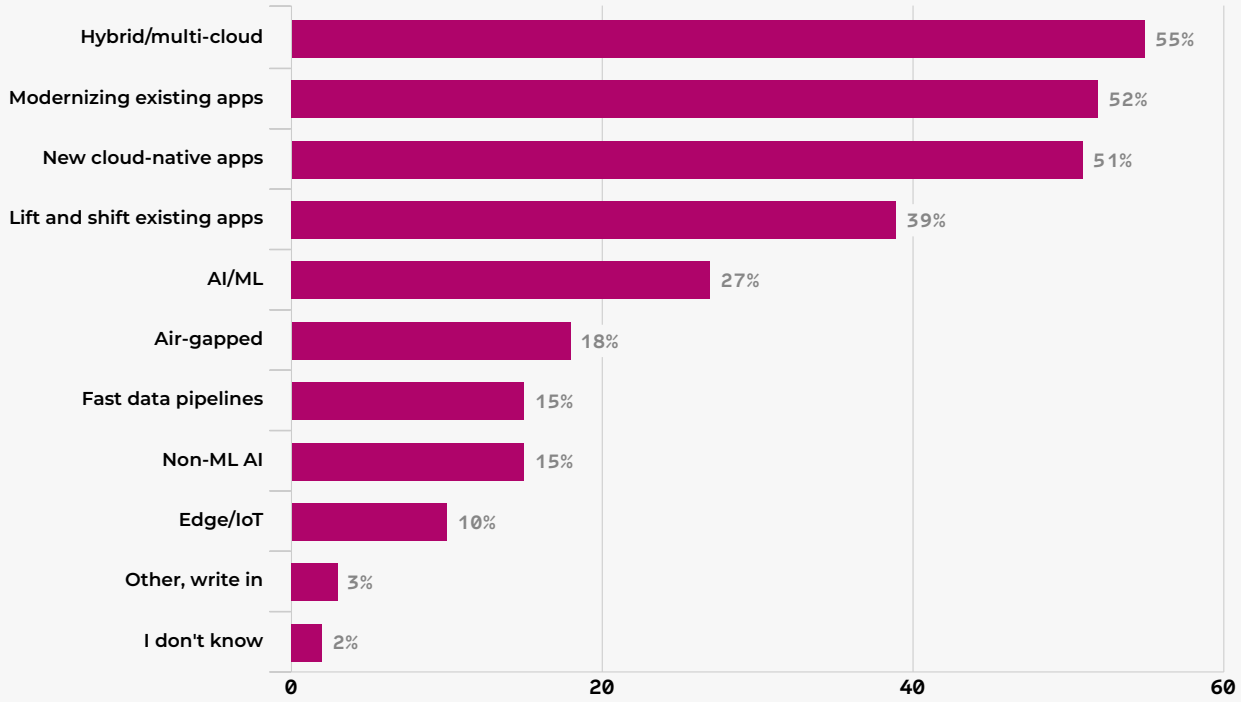


Figure 3. Kubernetes use cases [*n*=114]



OBSERVATIONS

74% of respondents indicated that they have personally worked with Kubernetes, showing no significant change from our 2024 Kubernetes research findings (76%) or our 2024 Cloud Native research findings (79%). And over three-quarters of respondents (77%) said their organization runs Kubernetes clusters. These response rates align closely with those we observed in our 2024 *Kubernetes in the Enterprise* Trend Report, where 75% of respondents said that their organizations ran Kubernetes clusters.

Respondents at **large organizations** were more likely than others to say their organization runs Kubernetes clusters, while respondents at **small organizations** were less likely than others to say they had personal experience with Kubernetes, and less likely to say their organization uses Kubernetes (see Table 1).

"Hybrid/multi-cloud" was the most commonly selected use case for Kubernetes after the significant increase we saw between our 2024 Cloud Native findings (44%) and our 2024 Kubernetes findings (54%). No use case was selected significantly more or less this year than last year's Kubernetes Trend Report findings (see Table 2).

We found that respondents working at **large organizations** were more likely than others to say their organization uses Kubernetes to "Modernize existing apps." Respondents at **small organizations** were less likely than others to say their organization uses Kubernetes for "Hybrid/multi-cloud," "Modernizing existing apps," "Air-gapped," and "Non-ML AI" (see Table 3).

CONCLUSIONS

Kubernetes adoption has likely reached a level of maturity and stability across the industry, with both individual and organizational usage holding steady compared to previous years. As we mentioned in our last Kubernetes Trend Report, we do not anticipate seeing large changes in response rates regarding developers' personal Kubernetes use for at least two to three years, possibly more, pointing toward Kubernetes having become a core part of the modern infrastructure stack rather than a rapidly emerging technology.

Larger organizations continue to lead in Kubernetes deployment, possibly because they have more resources, complex architectures, and modernization initiatives that benefit from container orchestration. Conversely, smaller organizations may be slower to adopt or invest in Kubernetes expertise, perhaps due to cost, operational complexity, or a preference for managed services.

The lack of significant shifts in Kubernetes use cases compared to last year may signal that the technology has reached a stable phase of implementation, with hybrid/multi-cloud and application modernization remaining key drivers. These patterns together indicate that the zeitgeist is no longer experimenting broadly with Kubernetes but is instead optimizing and scaling established strategies.

ADDITIONAL TABLES

Table 1. Personal and organizational Kubernetes use by organization size*

Use	Organization Size			Overall
	1-99	100-999	1,000+	
Personal	63%	82%	79%	72%
Organizational	60%	85%	87%	77%
n=	48	34	63	155

**% of columns*

Table 2. Kubernetes use cases: 2024–2025

Use Case	2024	2025	Δ*
Hybrid/multi-cloud	54%	55%	+1%
Modernizing existing apps	46%	52%	+6%
New cloud-native apps	49%	51%	+2%
Lift and shift existing apps	33%	39%	+6%
AI/ML	32%	27%	-5%
Air-gapped	12%	18%	+5%
Fast data pipelines	16%	15%	-1%
Non-ML AI	8%	15%	+7%
Edge/IoT	12%	10%	-3%
Other, write in	1%	3%	+2%
I don't know	1%	2%	+1%
n=	90	114	-

**Change shown as percentage points*

Table 3. Kubernetes use cases by organization size*

Use Case	Organization Size			Overall
	1-99	100-999	1,000+	
Hybrid/multi-cloud	43%	55%	61%	55%
Modernizing existing apps	29%	52%	65%	52%
New cloud-native apps	50%	52%	52%	51%
Lift and shift existing apps	32%	38%	44%	39%
AI/ML	25%	31%	26%	27%
Air-gapped	4%	28%	19%	18%
Fast data pipelines	18%	14%	15%	15%
Non-ML AI	0%	21%	19%	15%
Edge/IoT	4%	7%	13%	10%
Other, write in	4%	0%	4%	3%
I don't know	4%	0%	0%	2%
n=	28	29	54	114

**% of columns*

Workloads and Application Architectures

We asked the following questions:

- *What types of workloads does your organization run on Kubernetes clusters?**
- *What constructs does your organization use to maintain state within a Kubernetes cluster?**
- *Does your organization run any microservices?*
- *Are any of these microservices deployed on Kubernetes clusters?***

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

***This question was only asked to respondents who selected "Yes" to **both** of the following questions: "Does your organization run any Kubernetes clusters?" and "Does your organization run any microservices?"*

Results:

Figure 4. Workload types run on Kubernetes [*n*=116]

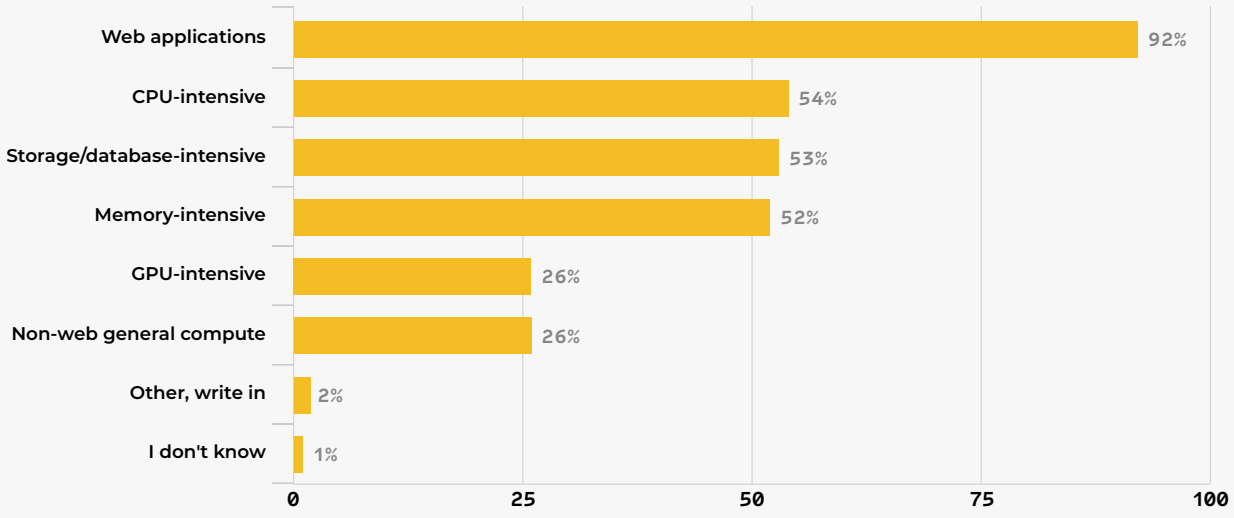


Figure 5. Constructs to maintain Kubernetes state [*n*=116]

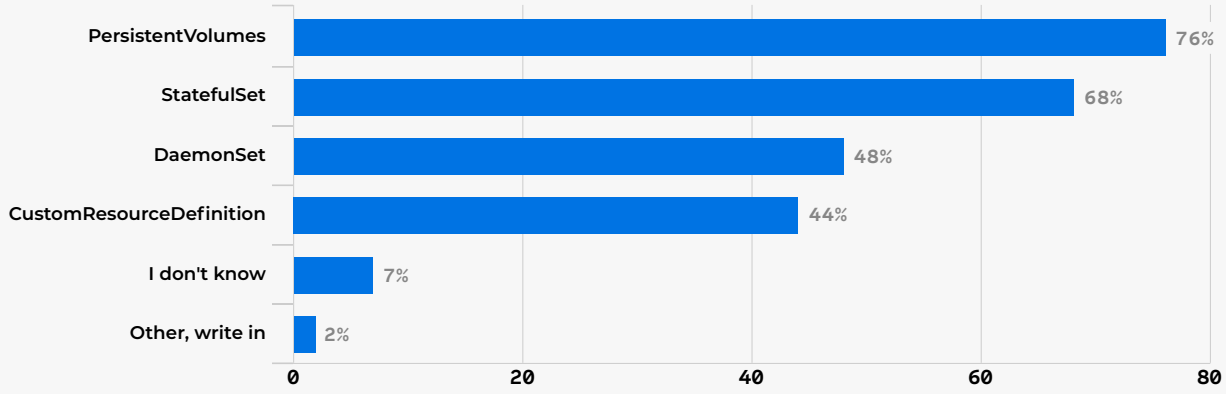


Figure 6. Use of microservices [*n*=154]

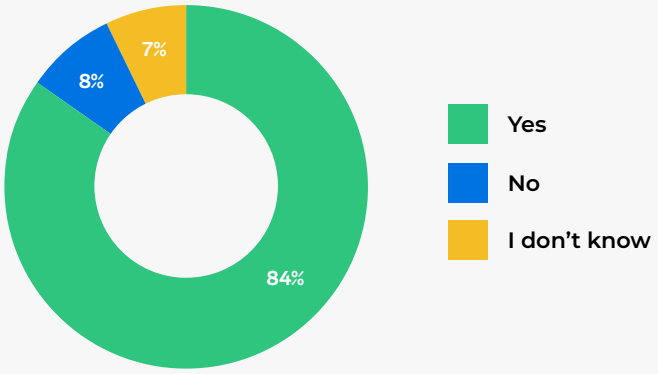
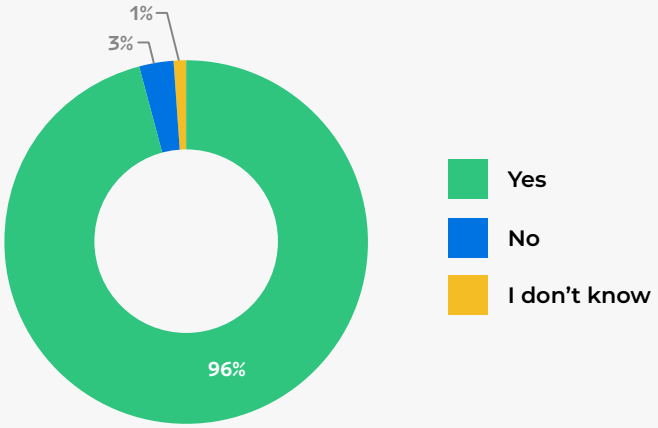


Figure 7. Use of Kubernetes for microservices [*n*=112]



OBSERVATIONS

"Web applications" was by far the most common type of workload respondents said their organization runs on Kubernetes, up 10% from last year's Trend Report. "CPU-intensive" workloads were the next most commonly selected type, though response rates had fallen significantly from 2024 (see Table 4). 85% of respondents said their organization uses Kubernetes for at least two different types of workloads, and 62% selected three types or more.

Segmenting the responses by respondents' organization size, we found that respondents at **mid-sized organizations** were more likely than others to use Kubernetes for "Non-web general compute," and respondents at **small organizations** were less likely than others to use Kubernetes for "CPU-intensive" and "Non-web general compute" workloads (see Table 5).

"PersistentVolumes" was the most commonly selected construct for maintaining state in Kubernetes clusters, followed by "StatefulSet." Response rates for both "PersistentVolumes" and "CustomResourceDefinition" increased significantly since 2024 (see Table 6). About three-quarters of respondents (74%) said their organization uses at least two of the four constructs to maintain state.

Respondents at **mid-sized organizations** were more likely than others to say their organization uses "CustomResourceDefinition," while respondents at **small organizations** were less likely than others to say they use "PersistentVolumes" and "CustomResourceDefinition" (see Table 7).

A large majority of respondents said their organization runs microservices, remaining consistent with our survey results from the last few years. Respondents at **large organizations** were more likely than others to say their organization uses microservices, while respondents at **small organizations** were less likely (see Table 8). Almost all respondents who said that their organization runs microservices *and* uses Kubernetes reported that their organization deploys microservices on Kubernetes clusters. Once again, this was consistent with our results from the past few years.

CONCLUSIONS

Kubernetes continues to serve as a versatile platform for a wide variety of workloads, with web applications dominating more now than ever — a likely reflection of both the ubiquity of web-based services and Kubernetes' maturity in handling them. The decline in CPU-intensive workload usage may stem from organizations offloading certain compute-heavy tasks to specialized platforms or cloud-native services. The majority of organizations leveraging Kubernetes for multiple workload types suggests that once Kubernetes is adopted, it tends to expand across use cases rather than remain isolated to a single purpose.

State management appears to be evolving, with increased usage of constructs like PersistentVolumes and CustomResourceDefinitions potentially pointing to more sophisticated, production-grade workloads and a greater need for flexibility. The variation by organization size suggests that mid-sized companies may be leading innovation in extending Kubernetes' native capabilities, while smaller organizations could be adopting simpler patterns due to resource constraints.

Finally, the consistency of microservices adoption — and their near-universal deployment on Kubernetes among organizations already using both — reinforces Kubernetes' role as the de facto orchestration layer for modern application architectures, particularly in large and scaling enterprises.

ADDITIONAL TABLES

Table 4. Workload types run on Kubernetes: 2024–2025

Workload Type	2024	2025	Δ*
Web applications	82%	92%	+10%
CPU-intensive	67%	54%	-13%
Storage/database-intensive	51%	53%	+2%
Memory-intensive	60%	52%	-8%
GPU-intensive	29%	26%	-3%
Non-web general compute	20%	26%	+6%
Other, write in	0%	2%	+2%
I don't know	0%	1%	+1%
n=	87	116	-

**Change shown as percentage points*

Table 5. Workload types run on Kubernetes by organization size*

Workload Type	Organization Size			Overall
	1-99	100-999	1,000+	
Web applications	93%	93%	96%	92%
CPU-intensive	39%	55%	62%	54%
Storage/database-intensive	57%	55%	51%	53%
Memory-intensive	43%	52%	56%	52%
GPU-intensive	25%	21%	27%	26%
Non-web general compute	11%	41%	24%	26%
Other, write in	0%	0%	2%	2%
I don't know	0%	0%	0%	1%
n=	28	29	55	116

**% of columns*

Table 6. Constructs to maintain Kubernetes state: 2024–2025

Construct	2024	2025	Δ*
PersistentVolumes	58%	76%	+18%
StatefulSet	61%	68%	+7%
DaemonSet	43%	48%	+5%
CustomResourceDefinition	33%	44%	+11%
Other, write in	1%	2%	+1%
I don't know	-	7%	-
n=	88	116	-

**Change shown as percentage points*

Table 7. Constructs to maintain Kubernetes state by organization size*

Construct	Organization Size			Overall
	1-99	100-999	1,000+	
PersistentVolumes	68%	83%	78%	76%
StatefulSet	61%	66%	75%	68%
DaemonSet	39%	55%	49%	48%
CustomResourceDefinition	32%	55%	44%	44%
Other, write in	4%	0%	2%	2%
I don't know	7%	3%	5%	7%
n=	28	29	55	116

**% of columns*

Table 8. Use of microservices by organization size*

Microservices Use	Organization Size			Overall
	1-99	100-999	1,000+	
Yes	84%	75%	85%	97%
No	8%	17%	97%	0%
I don't know	7%	8%	9%	3%
n=	154	48	0%	63

**% of columns*

Research Target Two: Developer Experience and Tooling

Our research related to **developer experience** and **tooling** centered around two key topic areas:

1. Developer platforms and productivity
2. Tooling and cost efficiency

Developer Platforms and Productivity

We asked the following questions:

- *In what ways has Kubernetes affected developer productivity at your organization?*
- *Does your organization provide a self-service platform or internal developer platform (IDP) for Kubernetes or container-based development?*

Results:

Figure 8. Kubernetes' effect on developer productivity [*n*=150]

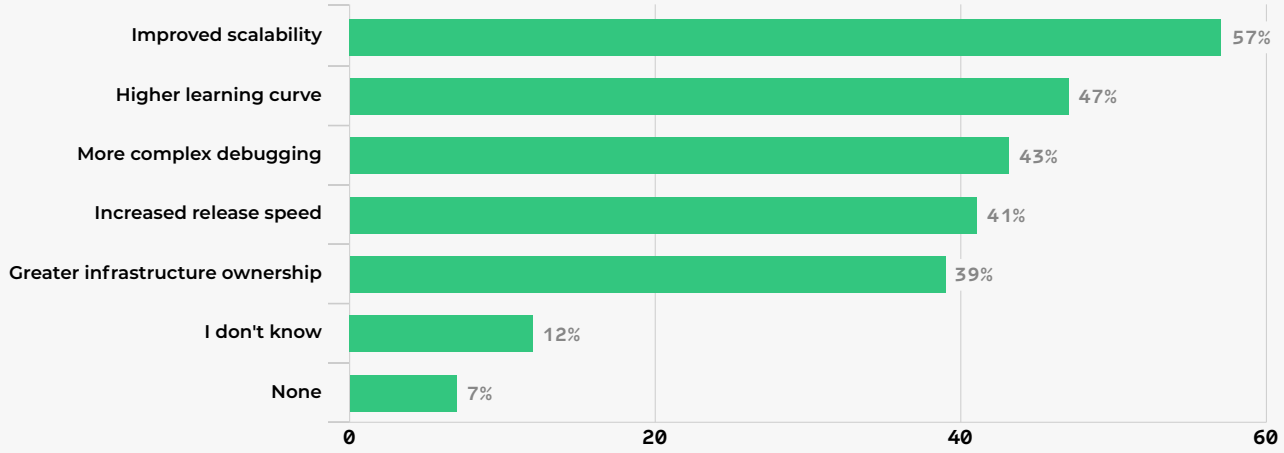
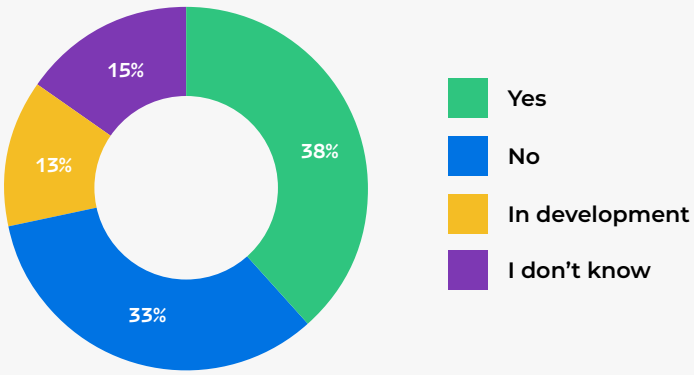


Figure 9. Provision of self-service platform or IDP [*n*=149]



OBSERVATIONS

The most commonly selected effect of Kubernetes on developer productivity was "Improved scalability," followed by "Higher learning curve" and "More complex debugging."

Segmented by organization size, we observed the following (see Table 9 for more):

- Respondents at **large organizations** were more likely than others to say Kubernetes "Increased release speed" at their organization.
- Respondents at **mid-sized organizations** were more likely than others to say Kubernetes caused "More complex debugging."
- Respondents at **small organizations** were less likely than others to say Kubernetes "Improved scalability" at their organization. They were more likely than others to say they didn't know how Kubernetes has affected developer productivity.

About half of respondents said that their organization does provide a self-service platform or IDP for Kubernetes/container-based development — either already implemented (38%) or in development (13%). Those at **large organizations** were much more likely than others to say their organization offered such a platform (i.e., answered "Yes"), and much less likely to say their organization did not offer a self-service platform or IDP (see Table 10).

Respondents who said their organizations offered a self-service platforms or IDP were more likely than others to say Kubernetes has affected developers' productivity in their organizations in a positive way — saying Kubernetes "Improved scalability," "Increased release speed," and allowed for "Greater infrastructure ownership."

Respondents who said their organizations did not offer any self-service platforms or IDPs were more likely than others to say their organization saw no effects on developer productivity — positive or negative (see Table 11).

CONCLUSIONS

Our findings suggest that Kubernetes' impact on developer productivity is nuanced and dependent on both organizational size and supporting platform practices. The prevalence of "Improved scalability" as the top reported Kubernetes benefit to productivity likely reflects Kubernetes' strength in enabling teams to handle larger and more dynamic workloads, though the accompanying "Higher learning curve" and "More complex debugging" underscore the technical complexity it introduces.

Larger organizations reporting increased release speed may be because they have the resources and platform engineering maturity to fully integrate Kubernetes into their delivery pipelines, whereas smaller organizations' uncertainty may stem from less operational experience or fewer supporting tools.

The correlation between offering a self-service platform or IDP and reporting positive developer productivity impacts suggests that these platforms play a key role in unlocking Kubernetes' potential benefits. This may be because self-service and developer-focused abstractions reduce friction, streamline deployments, and give teams more autonomy, all while masking underlying complexity.

Conversely, organizations without such platforms may struggle to see productivity changes at all, possibly because Kubernetes remains an infrastructure concern rather than a developer-enabling capability in those environments.

ADDITIONAL TABLES

Table 9. Kubernetes' effect on developer productivity by organization size*

Effect	Organization Size			Overall
	1-99	100-999	1,000+	
Improved scalability	47%	59%	68%	57%
Higher learning curve	45%	53%	52%	47%
More complex debugging	34%	68%	40%	43%
Increased release speed	38%	32%	48%	41%
Greater infrastructure ownership	34%	38%	45%	39%
I don't know	17%	6%	6%	12%
None	9%	12%	2%	7%
n=	47	34	62	150

**% of columns*

Table 10. Provision of self-service platform or IDP by organization size*

Platform Provision	Organization Size			Overall
	1-99	100-999	1,000+	
Yes	38%	28%	26%	55%
In development	13%	15%	12%	13%
No	33%	41%	44%	21%
I don't know	15%	15%	18%	11%
n=	149	46	34	62

**% of columns*

Table 11. Kubernetes' effect on developer productivity by IDP availability*

Effect	IDP Availability				Overall
	Yes	In dev	No	I don't know	
Improved scalability	77%	65%	45%	30%	57%
Higher learning curve	54%	45%	51%	26%	47%
More complex debugging	47%	50%	41%	30%	43%
Increased release speed	61%	30%	31%	22%	41%
Greater infrastructure ownership	51%	40%	37%	13%	39%
None	2%	5%	16%	0%	7%
I don't know	5%	10%	2%	48%	12%
n=	57	20	49	23	150

**% of columns*

Tooling and Cost Efficiency

We asked the following questions:

- *What tools/platforms does your organization use to manage containers in either development or production environments?*
- *Has your organization experienced tool sprawl in its container/Kubernetes ecosystem?*
- *What are your organization's biggest challenges related to managing Kubernetes tooling?*
- *Which of the following practices and techniques does your organization use to manage and optimize costs?*

Results:

Figure 10. Tools/platforms for managing containers [*n*=152]

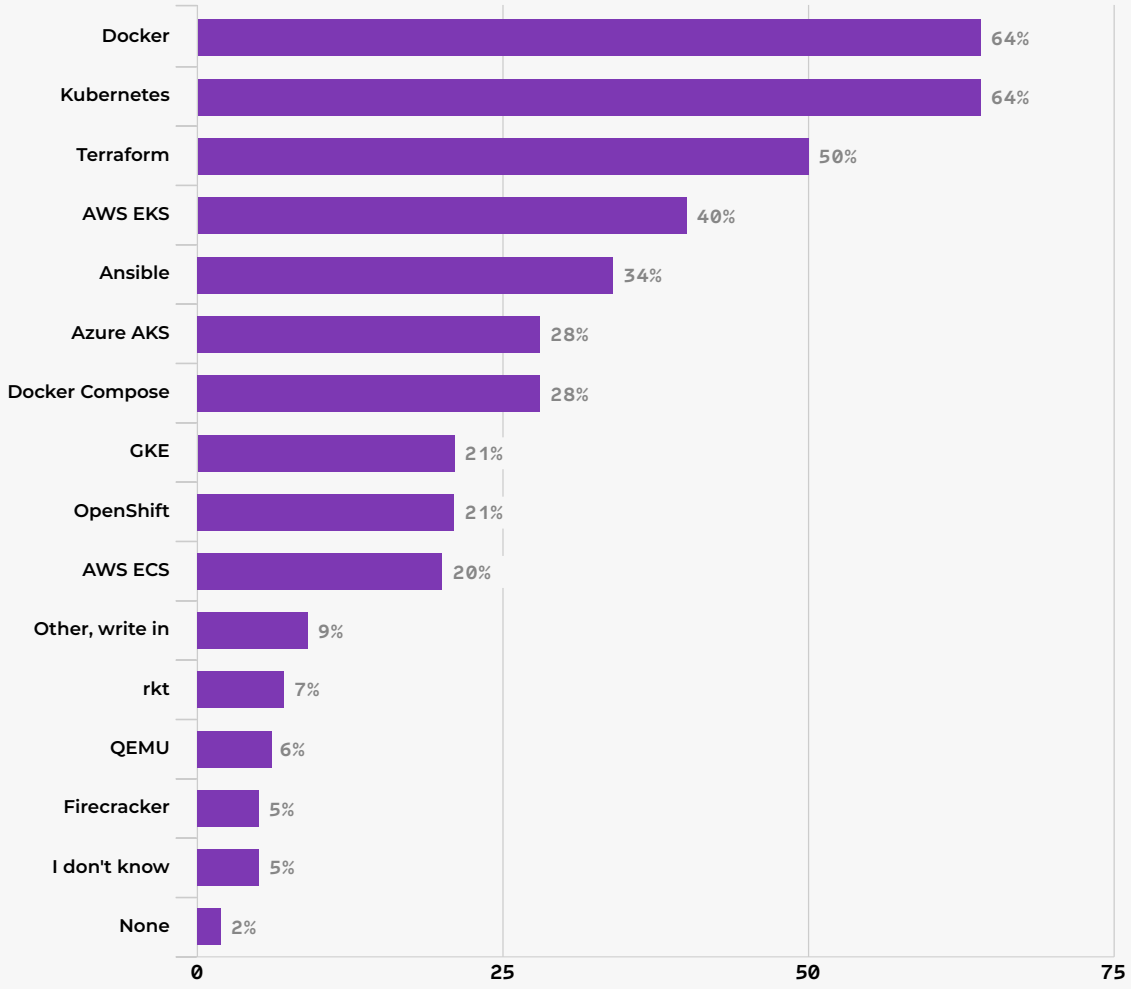


Figure 11. Tool sprawl in container/Kubernetes ecosystems [*n*=150]

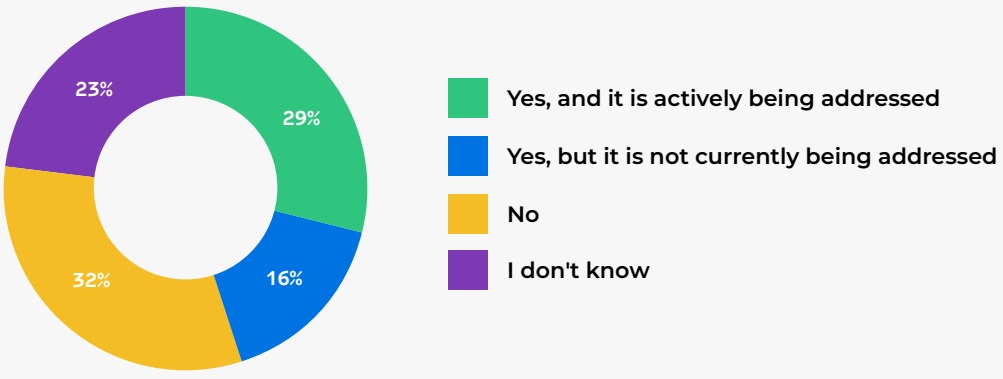


Figure 12. Biggest challenges regarding Kubernetes tooling [*n*=151]

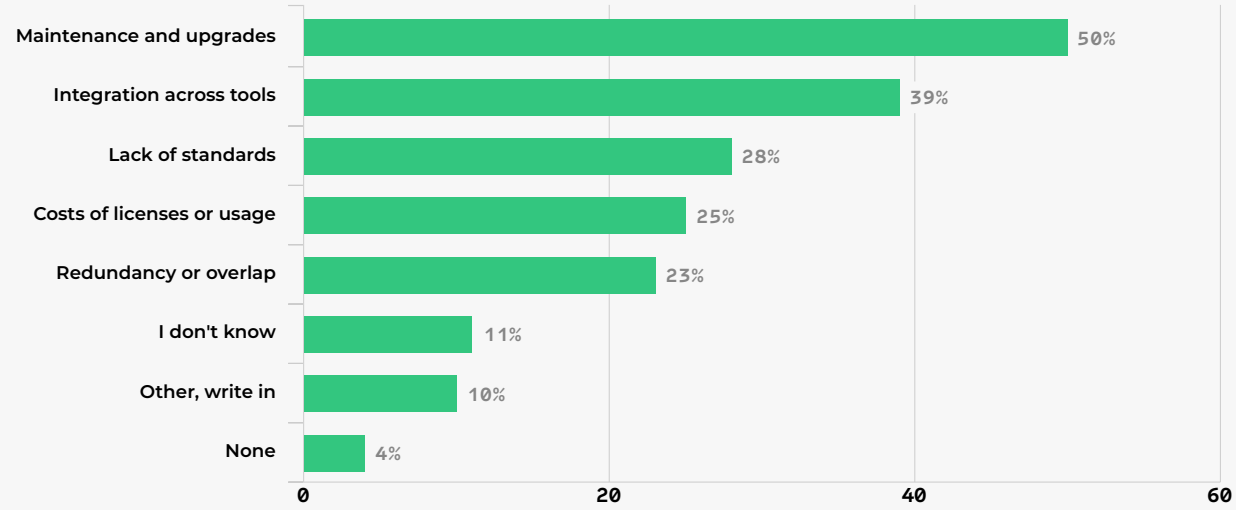
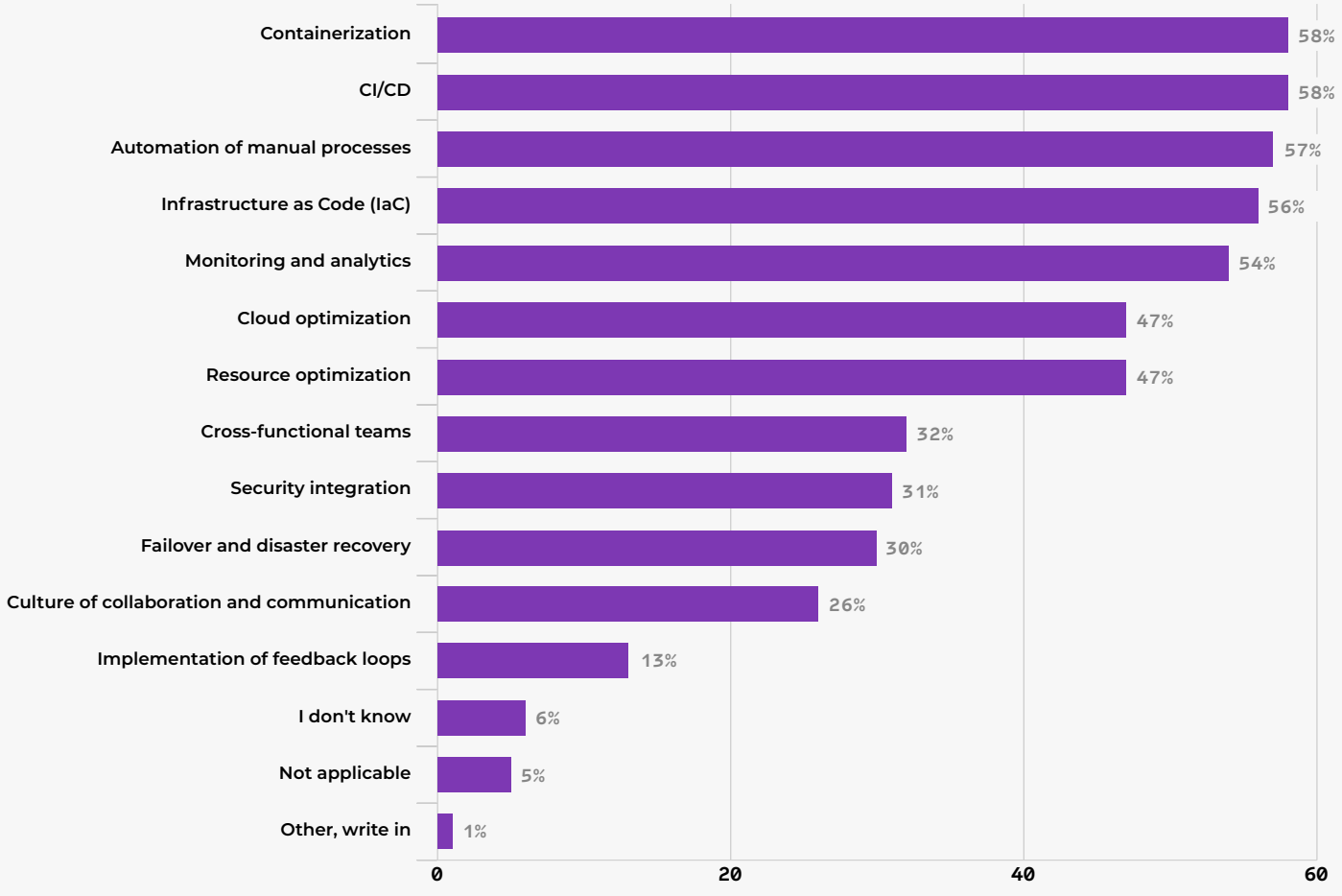


Figure 13. Practices/techniques for managing/optimizing costs [*n*=151]



OBSERVATIONS

About three-quarters of respondents (73%) said their organization uses at least three of the 13 provided tools/platforms to manage containers, and over half of respondents (55%) selected four or more. "Docker" and "Kubernetes" were the most commonly selected container management tools, with each garnering a response rate of nearly two-thirds of respondents. Since last year's Kubernetes survey, several tools saw significant decreases in response rates, including "Docker" and "Kubernetes" (see Table 12).*

Segmenting responses by organization size, we noted the following (see Table 13):

- Respondents at **large organizations** were more likely than others to say their organization uses "Terraform" and "Ansible."
- Respondents at **mid-sized organizations** were more likely than others to say their organization uses "AWS ECS."
- Respondents at **small organizations** were less likely than others to say their organization uses "Kubernetes," "Terraform," and "AWS EKS."

**Note: Year-over-year (YOY) results regarding container management tools may have been affected by a change in question response type from last year's survey.*

Nearly half of respondents (45%) said their organization has experienced tool sprawl in their container/Kubernetes ecosystem, and roughly two-thirds of those respondents said the tool sprawl is actively being addressed. Respondents at **large organizations** were less likely than others to say their organization has not experienced tool sprawl (see Table 14).

Over three-quarters of respondents (78%) selected at least one of the five Kubernetes tooling challenges we provided, and about half of respondents (51%) selected two or more. "Maintenance and upgrades" was the most commonly selected challenge, followed by "Integration across tools." Respondents at **large organizations** were less likely than others to say that a "Lack of standards" is a challenge at their organization, while respondents at **mid-sized organizations** were less likely than others to select "Integration across tools" as a challenge (see Table 15).

A large majority of respondents (90%) said their organization uses at least one of the 12 practices/techniques to optimize costs that we provided, and over half of respondents (53%) said their organization uses five or more. The most commonly selected cost-optimization practices were "Containerization," "CI/CD," and "Automation of manual processes." Since 2024, response rates for "Culture of collaboration and communication" and "Implementation of feedback loops" fell significantly (see Table 16).

Segmenting responses by respondents' organization size, we observed the following (see Table 17):

- Respondents at **large organizations** were more likely than others to select several cost optimization techniques, including "Containerization," "CI/CD," "Infrastructure as Code (IaC)," and "Cloud optimization."
- Respondents at **mid-sized organizations** were more likely than others to say their organization uses "Resource optimization" as a cost-optimizing technique.
- Respondents at **small organizations** were less likely than others to say their organization uses "Containerization," "Infrastructure as Code (IaC)," and "Monitoring and analytics" to optimize costs. They were more likely than others to say they did not know what cost-optimization practices their organization uses.

CONCLUSIONS

The results point to a Kubernetes and container tooling ecosystem that is both broad and in flux. The fact that most organizations are using several tools simultaneously likely reflects the complexity of container orchestration at scale, but the YOY declines in usage of core tools like Docker and Kubernetes may stem from changes in how organizations categorize tools — though these declines may also be related to shifts in our survey methodology. Large organizations leaning into Terraform and Ansible suggests a continued trend toward Infrastructure as Code and automated configuration management, while smaller organizations' lower usage rates may be because of resource or skill limitations.

Tool sprawl remains a persistent challenge, with nearly half of respondents experiencing it, though most of those are actively working to address it — likely indicating growing platform engineering maturity. Maintenance, upgrades, and tool integration rising as top challenges could be due to the increasing diversity and specialization of the Kubernetes tooling market. Cost optimization appears widespread, but differences by organization size suggest that larger enterprises are adopting a broader portfolio of techniques, whereas smaller companies may focus on

selective practices or lack visibility altogether. Together, these findings underscore a Kubernetes ecosystem in which standardization and consolidation pressures coexist with innovation and experimentation.

ADDITIONAL TABLES

Table 12. Tools/platforms for managing containers: 2024–2025

Tool/Platform	2024	2025	Δ*
Docker	87%	64%	-23%
Kubernetes	84%	64%	-20%
Terraform	57%	50%	-7%
AWS EKS	47%	40%	-7%
Ansible	37%	34%	-3%
Docker Compose	56%	28%	-28%
Azure AKS	38%	28%	-10%
GKE	22%	21%	-1%
OpenShift	19%	21%	+2%
AWS ECS	33%	20%	-13%
rkt	6%	7%	+1%
QEMU	6%	6%	0%
Firecracker	3%	5%	+2%
n=	100	152	-

**Change shown as percentage points*

Table 13. Tools/platforms for managing containers by organization size*

	Organization Size			
Tool/Platform	1-99	100-999	1,000+	Overall
Docker	61%	71%	70%	64%
Kubernetes	43%	76%	81%	64%
Terraform	33%	53%	67%	50%
AWS EKS	17%	50%	54%	40%
Ansible	28%	21%	44%	34%
Azure AKS	22%	38%	29%	28%
Docker Compose	33%	24%	27%	28%
GKE	17%	21%	22%	21%
OpenShift	17%	21%	21%	21%
AWS ECS	13%	32%	19%	20%
rkt	7%	3%	6%	7%
QEMU	9%	3%	5%	6%
Firecracker	9%	0%	5%	5%
Other, write in	4%	6%	13%	9%
None	2%	0%	2%	2%
I don't know	7%	6%	2%	5%
n=	46	34	63	152

**% of columns*

Table 14. Tool sprawl in container/Kubernetes ecosystems by organization size*

Tool Sprawl	Organization Size			Overall
	1-99	100-999	1,000+	
Yes, and it is actively being addressed	28%	24%	34%	29%
Yes, but it is not currently being addressed	20%	18%	13%	16%
No	37%	38%	26%	32%
I don't know	15%	21%	27%	23%
n=	46	34	62	150

**% of columns*

Table 15. Biggest challenges regarding Kubernetes tooling by organization size*

Challenge	Organization Size			Overall
	1-99	100-999	1,000+	
Maintenance and upgrades	46%	56%	54%	50%
Integration across tools	41%	29%	46%	39%
Lack of standards	33%	35%	22%	28%
Costs of licenses or usage	28%	21%	25%	25%
Redundancy or overlap	22%	24%	25%	23%
Other, write in	2%	6%	3%	4%
None	13%	9%	5%	10%
I don't know	15%	3%	8%	11%
n=	46	34	63	151

**% of columns*

Table 16. Practices/techniques for managing/optimizing costs: 2024–2025

Practice	2024	2025	Δ*
Containerization	62%	58%	-5%
CI/CD	61%	58%	-3%
Automation of manual processes	65%	57%	-8%
Infrastructure as Code (IaC)	59%	56%	-3%
Monitoring and analytics	59%	54%	-5%
Cloud optimization	56%	47%	-9%
Resource optimization	42%	47%	+5%
Cross-functional teams	41%	32%	-9%
Security integration	35%	31%	-4%
Failover and disaster recovery	33%	30%	-4%
Culture of collaboration and communication	40%	26%	-14%
Implementation of feedback loops	27%	13%	-15%
Other, write in	2%	1%	0%
Not applicable	4%	5%	0%
I don't know	-	6%	-
n=	117	151	-

**Change shown as percentage points*

Table 17. Practices/techniques for managing/optimizing costs by organization size*

Practice	Organization Size			Overall
	1-99	100-999	1,000+	
Containerization	45%	59%	71%	58%
CI/CD	45%	53%	74%	58%
Automation of manual processes	53%	59%	61%	57%
Infrastructure as Code (IaC)	34%	62%	73%	56%
Monitoring and analytics	43%	59%	65%	54%
Cloud optimization	34%	38%	65%	47%
Resource optimization	43%	59%	48%	47%
Cross-functional teams	26%	29%	42%	32%
Security integration	30%	21%	40%	31%
Failover and disaster recovery	26%	26%	39%	30%
Culture of collaboration and communication	26%	26%	31%	26%
Implementation of feedback loops	15%	9%	15%	13%
Other, write in	2%	0%	2%	1%
Not applicable	6%	6%	0%	5%
I don't know	13%	0%	0%	6%
n=	47	34	62	151

**% of columns*

Research Target Three: Security, Compliance, and Observability

Our research related to **Kubernetes security**, **compliance**, and **observability** focused on two key topic areas:

1. Observability and intelligent operations
2. Kubernetes security posture

Observability and Intelligent Operations

We asked the following questions:

- *Which of the following tools does your organization use for monitoring and/or observability of Kubernetes-based applications?**
- *In what ways has your organization adopted AI for monitoring and/or observability?*
- *Does your organization run AI/ML workloads on Kubernetes?*

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

SEE RESULTS ON NEXT PAGE

Results:

Figure 14. Tools for monitoring/observability of Kubernetes apps [*n*=112]

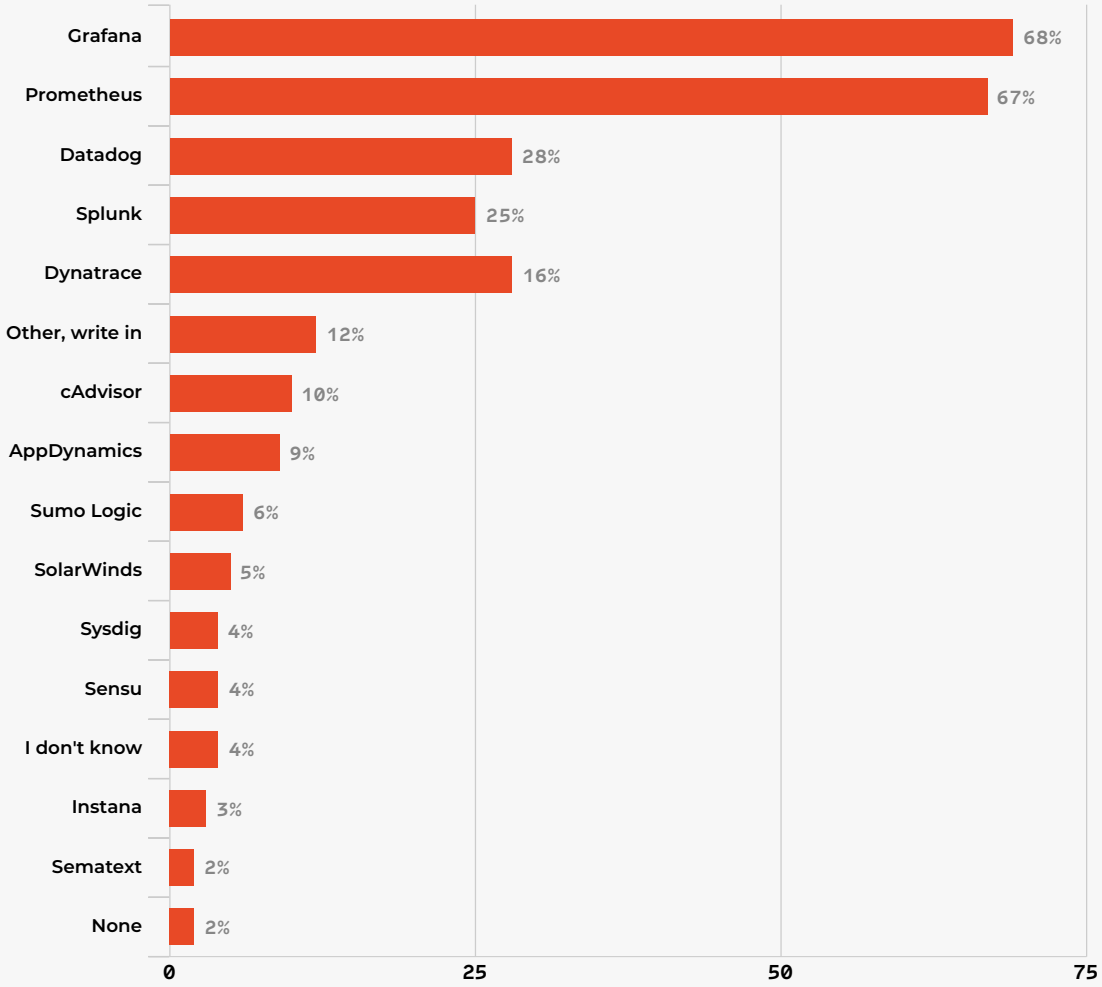


Figure 15. AI adoption for monitoring/observability [*n*=150]

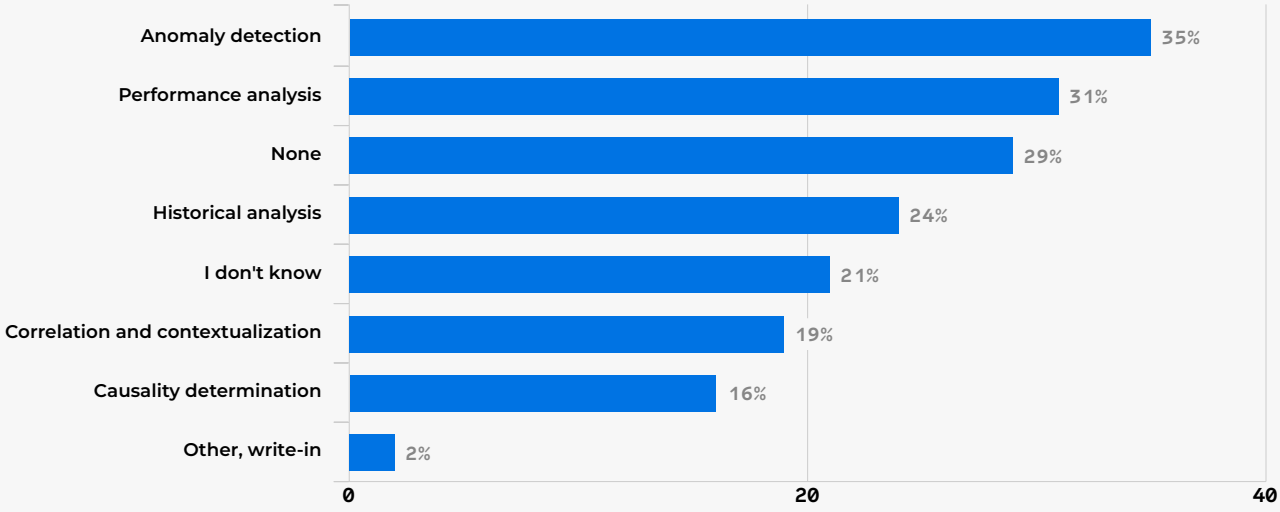
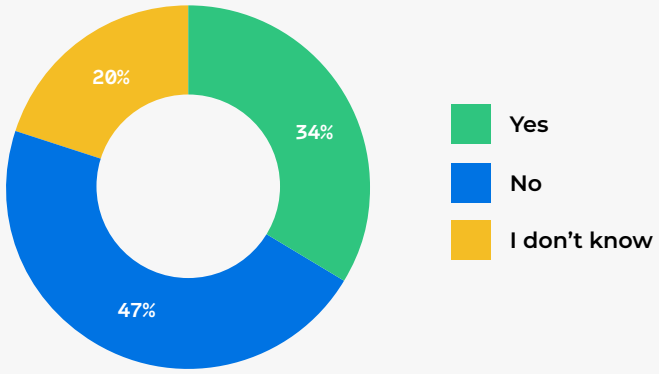


Figure 16. AI/ML workloads on Kubernetes [*n*=145]



OBSERVATIONS

The vast majority of respondents (92%) said their organization uses at least one tool for monitoring/observability of Kubernetes-based applications, and 82% said their organization uses two or more. "Grafana" and "Prometheus" were by far the most commonly selected observability tools, with 75% of respondents saying their organization uses one of the two and 60% of respondents saying their organization uses both. There have been no significant changes in response rates since last year's survey (details available on request).

Segmenting responses by organization size, we noted the following (see Table 18):

- Respondents at **large organizations** were more likely than others to say their organization uses "Datadog" for Kubernetes observability.
- Respondents at **mid-sized organizations** were more likely than others to say their organization uses "Prometheus" and "Grafana." They were less likely than others to say their organization uses "Splunk."
- Respondents at **small organizations** were more likely than others to say their organization uses "SolarWinds" for Kubernetes observability, and less likely than others to say their organization uses "Prometheus."

About half of respondents (49%) selected one or more ways in which their organization uses AI for monitoring and observability, the most common ways being "Anomaly detection" and "Performance analysis." We saw no significant YOY changes from last year's survey.

We found that respondents at **mid-sized organizations** were more likely than others to say that their organization has not adopted AI for monitoring/observability, and that respondents at **small organizations** were more likely than others to say their organization has adopted AI for "Performance analysis" (see Table 19).

Only about a third of respondents said that their organization runs AI/ML workloads on Kubernetes, and nearly half of respondents said their organization does not. We found that respondents at organizations that do run AI/ML workloads on Kubernetes were more likely than others to say their organization has adopted each of the five AI monitoring/observability use cases we discussed in the previous observation (see Table 20).

CONCLUSIONS

Kubernetes observability is nearly universal, with most organizations using more than one tool, indicating a strong emphasis on visibility and operational insight. Grafana and Prometheus remain the dominant choices, likely because of their open-source roots, wide community support, and ease of integration, while enterprise tools like Datadog and Splunk appear more often in larger organizations, possibly reflecting budget and complexity differences.

AI for monitoring and observability remains in a growth phase, with only about half of respondents reporting adoption and no meaningful changes year over year, which may mean organizations are still experimenting with or evaluating its value. The fact that AI/ML workloads on Kubernetes are reported by only about one-third of respondents — and that those who run them are more likely to use AI in monitoring — suggests a reinforcing loop: organizations already investing in AI/ML on Kubernetes are also *building operational intelligence around it*.

This possibly signals a slow but steady convergence of Kubernetes as both the platform for modern workloads and the engine powering intelligent automation of its own operations.

ADDITIONAL TABLES

Table 18. Tools for monitoring/observability of Kubernetes apps by organization size*

Tool	Organization Size			Overall
	1-99	100-999	1,000+	
Grafana	57%	86%	62%	68%
Prometheus	50%	83%	66%	67%
Datadog	18%	21%	38%	28%
Splunk	25%	10%	34%	25%
Dynatrace	18%	14%	21%	18%
cAdvisor	11%	10%	9%	10%
AppDynamics	7%	3%	13%	9%
Sumo Logic	7%	3%	8%	6%
SolarWinds	14%	0%	4%	5%
Sensu	11%	0%	2%	4%
Sysdig	4%	0%	8%	4%
Instana	7%	0%	2%	3%
Sematext	4%	0%	2%	2%
Other, write in	11%	10%	11%	12%
None	0%	3%	2%	2%
I don't know	11%	0%	2%	4%
n=	28	29	53	112

**% of columns*

Table 19. AI adoption for monitoring/observability by organization size*

Use Case	Organization Size			Overall
	1-99	100-999	1,000+	
Anomaly detection	32%	38%	37%	35%
Performance analysis	40%	29%	26%	31%
Historical analysis	26%	21%	26%	24%
Correlation and contextualization	26%	15%	16%	19%
Causality determination	19%	12%	16%	16%
Other, write-in	4%	0%	2%	2%
None	26%	38%	26%	29%
I don't know	19%	12%	26%	21%
n=	47	34	62	150

**% of columns*

Table 20. AI adoption for monitoring/observability by AI/ML workloads on Kubernetes*

Use Case	AI/ML Workloads			Overall
	Yes	No	I don't know	
Anomaly detection	50%	29%	21%	35%
Performance analysis	42%	25%	24%	31%
Historical analysis	42%	17%	10%	24%
Correlation and contextualization	32%	14%	10%	19%
Causality determination	28%	10%	3%	16%
Other, write-in	4%	1%	0%	2%
None	18%	46%	7%	29%
I don't know	10%	14%	59%	21%
n=	50	69	29	150

**% of columns*

Kubernetes Security Posture

We asked the following questions:

- *Which of the following security-related measures does your organization take with regard to Kubernetes?**
- *In the past year, has your organization needed to reassess any planned Kubernetes deployments to production due to security concerns?**

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

Results:

SEE FIGURES ON NEXT PAGE

Figure 17. Security measures for Kubernetes [*n*=112]

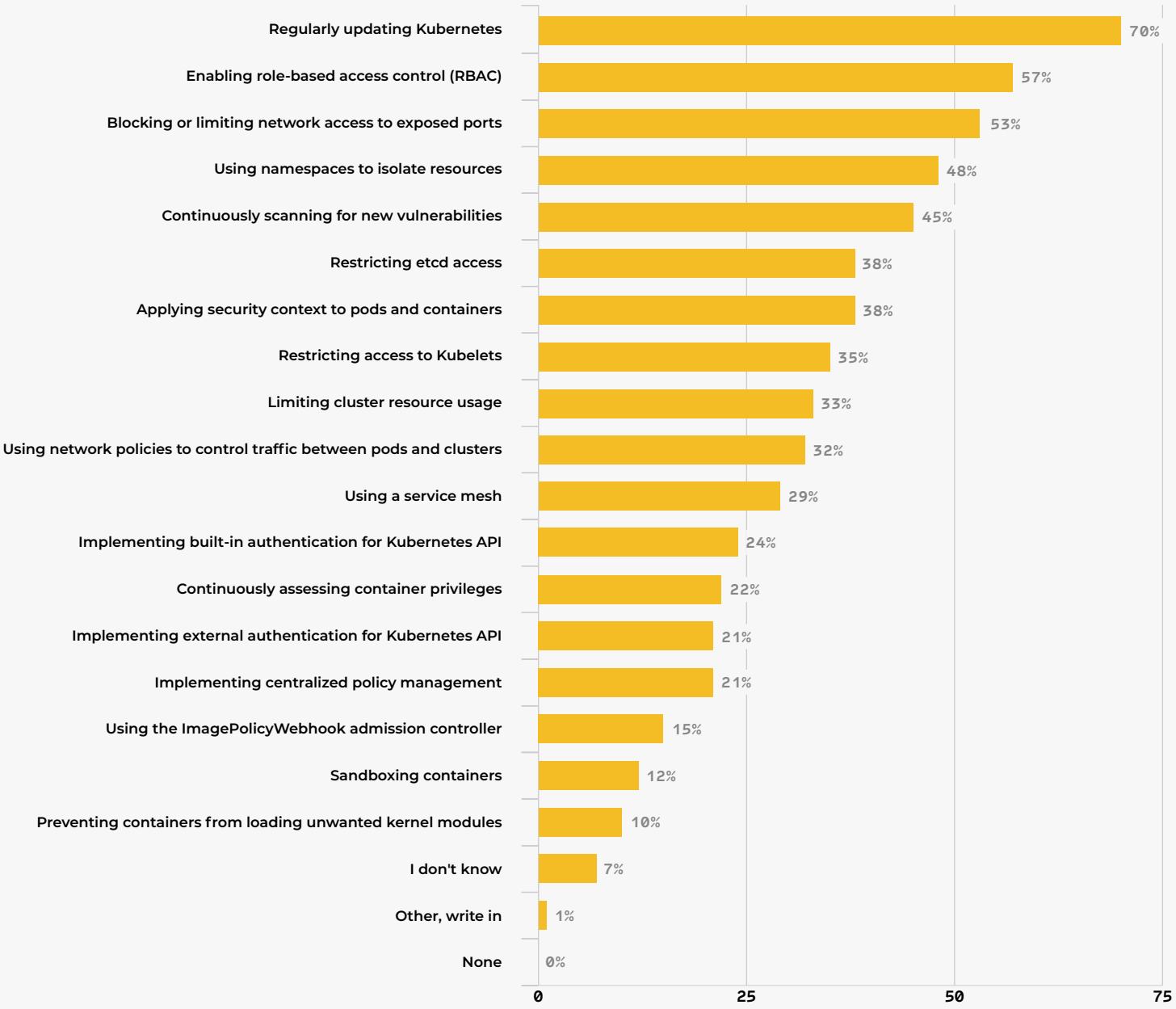
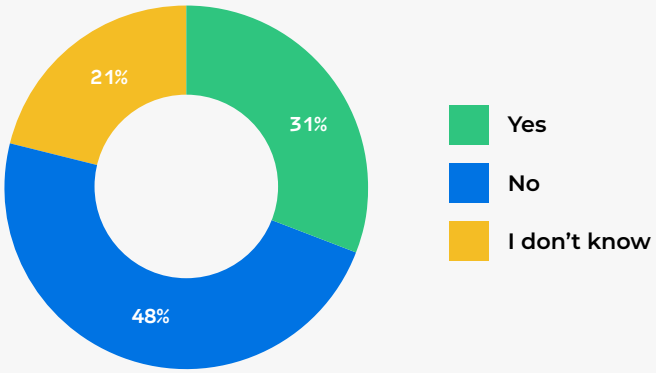


Figure 18. Prod deployment reassessment due to Kubernetes security concerns [*n*=108]



OBSERVATIONS

A large majority of respondents (88%) selected two or more security measures out of the 18 we provided, and over half (56%) selected five or more. The most commonly selected measures were "Regularly updating Kubernetes," "Enabling role-based access control (RBAC)," and "Blocking or limiting network access to exposed ports." The only significant changes from the 2024 *Kubernetes in the Enterprise* Trend Report were increases in response rates for "Continuously scanning for new vulnerabilities" and "Restricting access to Kubelets" (details available on request).

Segmented by organization size, we found the following (see Table 21):

- Respondents at **large organizations** were more likely than others to select "Using a service mesh."
- Respondents at **mid-sized organizations** were more likely than others to select "Regularly updating Kubernetes," "Restricting etcd access," "Restricting access to Kubelets," and "Continuously assessing container privileges." They were less likely than others to select "Applying security context to pods and containers" and "Using a service mesh."
- Respondents at **small organizations** were less likely than others to select "Continuously scanning for new vulnerabilities" and "Continuously assessing container privileges."

About a third of respondents said their organization has needed to reassess Kubernetes deployments to production due to security concerns in the past year, which did not differ significantly from our 2024 Kubernetes survey results. We found that respondents at **mid-sized organizations** were less likely than others to say their organization has had to reassess a prod deployment because of concerns over Kubernetes security.

CONCLUSIONS

Our findings suggest that Kubernetes security practices remain robust and widely implemented, with most organizations layering multiple measures to reduce risk. The dominance of practices like regularly updating Kubernetes, RBAC, and network access restrictions highlights a continued focus on foundational security.

Differences across organization sizes may point to maturity levels and operational priorities; large organizations adopting service meshes likely reflects efforts to standardize communication security at scale, while mid-sized organizations' emphasis on updates and privilege controls suggests a focus on hardening core components. Smaller organizations adopting fewer advanced practices may be due to resource limitations or lower perceived risk exposure.

The consistent rate of production deployment reassessments due to security concerns indicates that while security practices are in place, Kubernetes environments still encounter situations where planned rollouts must be reevaluated, likely as new threats or compliance requirements emerge.

ADDITIONAL TABLES

Table 21. Security measures for Kubernetes by organization size*

Measure	Organization Size			Overall
	1-99	100-999	1,000+	
Regularly updating Kubernetes	64%	79%	68%	70%
Enabling role-based access control (RBAC)	50%	66%	58%	57%
Blocking or limiting network access to exposed ports	46%	59%	55%	53%
Using namespaces to isolate resources	50%	55%	45%	48%
Continuously scanning for new vulnerabilities	29%	52%	51%	45%
Restricting etcd access	29%	48%	38%	38%
Applying security context to pods and containers	39%	28%	42%	38%
Restricting access to Kubelets	29%	55%	28%	35%
Limiting cluster resource usage	36%	34%	30%	33%
Using network policies to control traffic between pods and clusters	25%	41%	32%	32%
Using a service mesh	25%	14%	42%	29%

TABLE 21 CONTINUED

Measure	Organization Size			Overall
	1-99	100-999	1,000+	
Implementing built-in authentication for Kubernetes API	29%	28%	21%	24%
Continuously assessing container privileges	11%	34%	23%	22%
Implementing external authentication for Kubernetes API	18%	21%	25%	21%
Implementing centralized policy management	29%	24%	17%	21%
Using the ImagePolicyWebhook admission controller	14%	10%	19%	15%
Sandboxing containers	18%	10%	9%	12%
Preventing containers from loading unwanted kernel modules	11%	10%	9%	10%
Other, write in	0%	0%	2%	1%
None	0%	0%	0%	0%
I don't know	7%	7%	6%	7%
n=	28	29	53	112

**% of columns*

Table 22. Prod deployment reassessments due to Kubernetes security concerns by organization size*

Reassessment	Organization Size			Overall
	1-99	100-999	1,000+	
Yes	33%	18%	37%	31%
No	44%	64%	42%	48%
I don't know	22%	18%	21%	21%
n=	27	28	52	108

**% of columns*

Future Research

Our analysis here only touched the surface of the available data, and we will look to refine and expand our Kubernetes research as we produce future Trend Reports. Please contact publications@dzone.com if you would like to discuss any of our findings or supplementary data.

G. Ryan Spain

@grspain

@grspain

@grspain

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 Hollow Knight with a glass of red wine or a cold beer.

CONTRIBUTOR INSIGHTS

Death by a Thousand YAMLs

Surviving Kubernetes Tool Sprawl

By Yitaek Hwang, Software Engineer at NYDIG

Kubernetes is eating the world.

More than 10 years after Google open-sourced its container orchestration technology, Kubernetes is now everywhere. What started as a tool to primarily manage containers in the cloud has since bled into every facet of infrastructure. We now see companies using Kubernetes to manage not just their applications running in containers but also virtual machines and databases to edge deployments and IoT devices.

The numbers are staggering. Taking a look at this *State of Production Kubernetes 2025* report, we can see that over a third of the organizations run more than 50 clusters in production. Those clusters aren't small either: More than half run 1,000+ nodes, and one in 10 run 10,000+ nodes. In addition, organizations are now running clusters in more than five different clouds (e.g., AWS, Azure, GCP) and other environments (e.g., on-prem, edge, airgap, GPU clouds).

But that growth has come with a serious operational burden. Running a production-ready Kubernetes cluster is not simple. More than 40% of companies say they have more than 20 software elements in their Kubernetes stack. When you consider the fact that things like ingress, storage, secrets management, monitoring, and GPU operators are all "add-ons," it's not hard to see why that number is so high.

The result? Teams are drowning in YAML files, each managing yet another tool to keep the Kubernetes cluster humming. Developers are increasingly confused about how to deal with all these files, the security team is growing worried about new attack vectors, and DevOps teams are struggling to rein in this chaos.

The pace of innovation in the Kubernetes space has so far pushed tremendous growth, but it has also brought meaningful challenges to teams dealing with Kubernetes sprawl. Let's break down what this looks like in practice, the pain points it creates, and some emerging solutions to help us survive.

Anatomy of Kubernetes Sprawl

When we talk about "Kubernetes sprawl," we often point to two related but distinct issues: **cluster sprawl** and **tool sprawl**.

Cluster Sprawl

When Kubernetes was first released, multi-tenancy was not natively baked in. Besides using namespaces as soft isolation mechanisms, tooling was very immature to provide a true, multi-tenant solution. So at least initially, multiple clusters had to be created by necessity. Ten years later, the story is a bit more complicated. Cluster sprawl is now more of a function of organic growth.

The obvious reasons for cluster sprawl stem from environment separation. Whether to limit the blast radius or comply with organizational structure, it's common to see at least prod and nonprod separation. But if you look deeper, we now see Kubernetes clusters both locally and in CI that may be running a different distribution or topology than either prod or non-prod environments. Then we have multi-region and even multi-cloud clusters for resiliency or business-driven reasons (e.g., cost, compliance mandate). Finally, as Kubernetes bleeds into managing VMs, edge, and even other cloud workloads, separate clusters are spun up to keep separation of concerns or for convenience.

Tool Sprawl

Another side-effect of cluster sprawl is tool sprawl. Just take a look at the CNCF landscape grid. In order to manage and operate a production-ready Kubernetes cluster, we need related tooling for networking, storage, CI/CD, observability, security, and cost management.

Even though some managed Kubernetes providers now package them in add-ons or extensions, teams are still having to make choices on ingress controllers, service meshes, and secrets management, just to name a few. There are hundreds of tools that have overlapping capabilities. And while each tool solves a problem, together they create confusion and duplication of work.

Pain Points From Sprawl

Kubernetes sprawl is not simply a nuisance. It manifests in a serious operational burden and causes business risks.

Toil and Complexity

Starting with the most obvious, it causes immense toil work on teams maintaining all of those clusters and tools. Even with some automation in place, it takes time and effort to keep up with the speed of innovation in this space. Toil work ranges from simply upgrading Kubernetes versions to making sure that said upgrade does not break the numerous tools that run on top of Kubernetes, not to mention the applications that it supports.

It also places a huge mental burden on the developers interacting with the platform. If you are living and breathing Kubernetes every day, it's easy to become lost in the YAML hell and not realize how exactly one goes from pushing code to main to deploying it into Kubernetes with all the bells and whistles in place. Sure, some of it may be all black-box magic, but when things break, how much visibility do the developers have to self-service the issues themselves?

Security and Observability Gaps

Fragmented tooling and cluster sprawl lead to observability blind spots, and worse, security gaps stemming from inconsistent RBAC, uneven enforcement of policies, and a patchwork of agents and controllers scanning and alerting on security vulnerabilities. While there are efforts like OpenTelemetry and CNCF security projects to standardize how to tackle observability and security, most teams currently struggle to deal with a plethora of tools that address one of these concerns.

Cost Management

Finally, we have growing cost concerns. Over 42% of the *State of Production Kubernetes 2025* report respondents cited cost as their top challenge, and 88% said their Kubernetes total cost of ownership had increased in the past year. As the number of clusters and tools grows, cost can easily balloon. It's not simply about build vs. buy either. Given the complexity of the ecosystem and growing sprawl, there isn't a single solution where a managed Kubernetes option can simply outweigh the cost of operational management. Everything becomes a tradeoff with the rate of innovation outpacing cost control.

Emerging Solutions

While the pain from tool sprawl persists, the Kubernetes community has made some progress in combatting these issues.

Platform Engineering

Platform engineering might have been the hottest keyword in the DevOps space in recent years. New teams focused on developing tools and workflows to unlock self-service capabilities in the cloud-native world are standardizing and defining "paved paths" to reduce drift across clusters and environments. Platform engineering teams aim to curb Kubernetes sprawl by publishing:

- Reusable pipelines
- Centralized observability
- Security guardrails

We can see such growth in the 2025 report: Over 80% of organizations say that they have a mature platform engineering team, and 90% provide an internal developer platform (IDP) to allow self-service capabilities. But not all "platforms" are built the same and usage doesn't always equal effectiveness.

AI-Driven Solutions

Another promising path involves using AI to drive operational efficiency. The vision here is using a "Kubernetes Copilot" to tune resources, troubleshoot faster, or generate YAML manifests from natural-language prompts. The advantage of platform engineering discipline is that large language models (LLMs) can potentially index and produce these materials more than resource-constrained teams manually curating internal solutions.

While skepticism remains, early experiments with Kubernetes MCPs and LLM-powered resource generators are helping reduce the cognitive load of sprawl.

Conclusion

In 2025, there's no doubt that Kubernetes is a mature container orchestration technology of choice for many. It is powering tens of thousands of nodes across environments, regions, clouds, and at the edge. But with its meteoric rise in popularity has come at a cost: too many clusters, too many tools, and too much complexity. Kubernetes sprawl is impacting various teams in more ways than one.

There's no silver bullet, but several solutions are emerging. We now have a clearer picture of what platform engineering means as a discipline along with mature IDPs and policy frameworks to enforce consistency. And whatever your thoughts on the current AI hype are, it is undoubtedly affecting this ecosystem, from workload optimization and bootstrapping various YAML files to exposing more resources via natural language.

At the end of the day, sprawl is inevitable given its growth. But innovation will continue, which will bring new tools and paradigms to conquer the chaos, just like Kubernetes did to manage the container ecosystem.

CORE

Yitaek Hwang

@yitaek

yitaekhwang.com

Yitaek is a software engineer at NYDIG, where he works with Bitcoin technology. He has experience building infrastructure and developer tooling for various industries, ranging from IoT to blockchain. He often writes about cloud, DevOps/SRE, data engineering, and AI topics.

PARTNER OPINION

More Than a Checkbox

Why Kubernetes Security, Compliance, and Observability Need a New Mindset

By Megan Gay, Director of Global Campaigns, Kubernetes at Veeam

Kubernetes is everywhere; it's the de facto orchestration layer for modern infrastructure. Thanks to its scalability, declarative management, and ability to dynamically allocate resources across heterogeneous environments, Kubernetes powers everything from cloud-native apps to VMs and AI. As we rush to modernize our stacks and containerize everything, are we built to withstand the next generation of attacks?

Kubernetes workloads are ephemeral, scalable, and constantly shifting. This flexibility, while powerful, introduces data protection complexity. Your security and compliance strategy must keep up with continuous changes. Between misconfigurations, ransomware, and compliance gaps, the stakes are higher than ever before.

Is resilience something we only think about after an incident, or should it be engineered into every layer of our stack from day one? Is compliance just another audit to pass, or should it be an automated process that evolves? For observability, are we satisfied with dashboards that track uptime, or do we demand real-time insight into security posture, early threat detection, and anomalous insider activity?

Implement these key practices to ensure you're protected at all angles.

Automate Compliance

Relying on manual compliance processes and static checklists is a recipe for disaster in Kubernetes environments. Enforcement of data protection policies that use Kubernetes-native primitives must be automated. Leveraging CRDs, RBACs, and IaC enables consistent, scalable, and auditable workflows. Integrated audit logging ensures the traceability of all operations, which supports alignment with regulatory frameworks. This approach reduces human error, accelerates remediation, and strengthens your security posture.

Engineer for Resilience

Downtime equals lost trust and revenue, so resilience is essential. Engineer for data resilience by designing systems that can recover ***and*** maintain operational continuity under failure conditions, with automated workflows that support zero-trust principles and disaster recovery objectives. As a last line of defense against ransomware, insider threats, and misconfiguration, true resilience aligns with NIST's guidance on incident response and recovery.

Approach Observability With a Security Lens

Monitoring system health is necessary, but insufficient. Security teams need insight into who did what, when, and why across every layer, at all times. Get better visibility into backup operations, policy compliance, and anomalies by integrating with Prometheus, Grafana, or SIEM platforms. This way, you can have real-time alerting, audit trails, and forensic readiness, all of which are key components to both ISO/IEC 27011 and NIST CSF compliance frameworks.

Foster Shared Responsibility

Security and compliance are not siloed functions; they're shared responsibilities. Enhance DevSecOps workflows by embedding protection policies into CI/CD pipelines and enabling fine-grained RBACs and multi-tenancy that meet your needs. When your dev lifecycle aligns with secure software practices, your business will meet compliance from design to deployment, and your teams will be equipped for success from the get-go.

A New Mindset

There are new rules when it comes to Kubernetes data protection, and this means a shift in how you approach security, compliance, and observability. The days of treating these concerns as afterthoughts or relying on checklists are over. Success at scale is proactive. By automating compliance, threading in resilience from the ground up, prioritizing observability with a security-first mindset, and sharing responsibilities across teams, organizations can truly harness the benefits of Kubernetes. Those who are successful will be best poised to innovate without compromise.

Veeam is More

It's the Future of Data Resilience.
Your Data. Secure and Accessible.
Anytime, Anywhere.

LEARN MORE

CONTRIBUTOR INSIGHTS

Boosting Developer Productivity in Kubernetes-Driven Workflows: A Practical Checklist

By Louis-Guillaume MORAND, Senior Cloud Solutions Architect at Microsoft

Kubernetes has become the backbone of application deployment. Its flexibility and scalability are long-time proven, but its adoption by developers can still be a challenge. The misuse of Kubernetes configuration, through the thousands of options, can make applications less performant or less resilient in that they would be a single old-school server.

To fully take advantage of Kubernetes, organizations must prioritize the developer experience by embracing platform engineering practices that abstract complexity and provide self-service capabilities, enabling teams to deploy applications with confidence.

Simplify YAML Templates and Configuration Management

In Kubernetes, if you want to work properly (declarative way), you must use YAML manifests, which are powerful but verbose. If the basic configurations are easy to understand, then following organizational rules while implementing a production-ready workload with good practices regarding scalability, resiliency, and performance can be much harder. You must rationalize and implement governance but make it as transparent as possible for your developers.

To start this process:

- ☐ Define base templates for workloads (e.g., web services, batch jobs, resilient autoscalable workloads)
- ☐ Use Helm or Kustomize, which allow templating and reuse of configuration files
- ☐ Parameterize environments to avoid duplication of files
- ☐ Lint and validate by integrating tools (e.g., kubeval, kubelinter) to catch errors early
- ☐ Store all manifests in Git for traceability and rollback

Streamline CI/CD Pipelines

CI/CD pipelines are critical for reliable software deployment, yet Kubernetes adds complexity. Your developers must now handle image building, managing artifacts in a container registry, manifest generation, roll-out strategies, and some DevSecOps (e.g., image scanner).

To streamline CI/CD pipelines:

- ☐ Automate builds and push images to container registries
- ☐ Automate creation of deployment and service YAMLs
- ☐ Implement staging to production workflows with approval/automated gates
- ☐ Leverage rollout deployments (or blue-green/canary deployments) and ensure PodDisruptionBudgets are set to maintain service continuity
- ☐ Use Kubernetes features (e.g., ReplicaSets, Helm) to roll back and recover quickly
- ☐ Integrate metrics and logs into the pipeline for post-deployment analysis
- ☐ Leverage GitOps tools (e.g., ArgoCD, Flux) to automate deployments and maintain consistency across clusters

Provide Intuitive Tooling for Developers

Developers shouldn't need to be Kubernetes experts to deploy code, so provide tools that are adapted to developers, not the other way around.

To support developers, it is important to:

- ☐ Provide custom CLIs or wrappers around kubectl for common tasks
- ☐ Use dashboards (e.g., Lens, HeadLamp) for visual cluster management
- ☐ Integrate with vault, sealed secrets, or external providers for secrets management
- ☐ Allow containers to run locally with Docker Desktop or Podman
- ☐ Improve feedback loops with tools (e.g., Skaffold, Tilt) to sync code changes with clusters in real time
- ☐ Provide IDE extensions to help developers debug and interact with clusters directly from their editor

Enhance Observability and Feedback Loops

Visibility into application behavior is critical for debugging and optimization. Developers need access to logs, metrics, and traces from production.

To do this:

- ☐ Use Fluentd or Loki to aggregate logs
- ☐ Integrate Prometheus and Grafana for real-time insights
- ☐ Implement OpenTelemetry or Jaeger for distributed tracing
- ☐ Set up alerts for failed deployments, high latency, or resource exhaustion, and add SLO-aligned alerts with owners and runbooks
- ☐ Publish ready-made Grafana dashboards per golden path (HTTP latency, errors, saturation, HPA)
- ☐ Provide actionable insights via feedback loops (e.g., performance regressions, error rates)
- ☐ Implement KPIs between releases to gather data on quality of deployments

Enforce Policy as Code (in CI and at Admission)

Prevent bad configs before they reach the cluster, and at the door if they do. Use Gatekeeper or Kyverno to enforce policies inside the cluster, while linters such as kubeval and kubelinter are great for pipelines.

Other ways to enforce Policy as Code include:

- ☐ Establish baseline rules — e.g., require labels/owners, set resource requests/limits, disallow :latest for image version, implement Pod Security Standards, and define allowed registries
- ☐ Continuously integrate, run policy tests in pipelines, and fail early with actionable messages
- ☐ Enforce the same policies cluster-side, start in warn mode, and graduate to enforce
- ☐ Implement exception workflow via documented, time-boxed waivers with audit trails

CHECKLIST CONTINUES ON NEXT PAGE

Establish an Internal Developer Platform

Your maturity level will be reached when you provide an internal developer platform (IDP) as a starting point. An IDP acts as a self-service layer between developers and infrastructure. It abstracts complexity and provides standardized actions/triggers. Through forms or simple clicks, developers can quickly create a new environment, trigger a deployment, request access requests, etc.

Start your IDP using the following steps:

- ☐ Create centralized onboarding so that developers can spin up environments with minimal friction
- ☐ Offer reusable components (e.g., databases, queues, APIs) via a service catalog
- ☐ Enable one-click creation of dev/staging/prod environments
- ☐ Ensure secure access to resources with role-based access controls
- ☐ Track changes and enforce policies automatically

Provide Ephemeral, PR-Scoped Environments

The feedback loop is critical. Ideally, each developer/code reviewer should be able to validate changes in an ephemeral environment, which can be triggered by a Git branch or a pull request and then automatically cleaned up when the branch is merged/deleted.

Here are a few key considerations:

- ☐ Add namespace/cluster per PR; GitOps creates previews, wires DNS/certs, and tears down on merge/close
- ☐ Use quotas and TTL to prevent budget leaks and resource contention
- ☐ Implement secrets management with least privilege (and no sharing)
- ☐ Identify golden path overlays for lightweight configuration to preview differences (small replicas, reduced data)

Bring the Platform Into the IDE

Reduce context switching by embedding platform actions directly in developers' tools via:

- ☐ IDE packs – VS Code extensions (K8s, YAML, Helm), snippets, tasks for common flows
- ☐ Create service command – launches a Backstage template from the IDE
- ☐ Debug in cluster – bridge to Kubernetes actions baked into the editor; use tools such as Mirrord or Telepresence
- ☐ Diagnostics – schema validation and linting surfaced in the IDE

Foster a Culture of Documentation and Enablement

Upskilling is key to keep your talent and ensure that everyone knows what they are doing. Providing the right tools or the right platform to an unqualified person will only result in bad outcomes. It is important to:

- ☐ Maintain living documentation on a central wiki
- ☐ Provide runbooks for common tasks (e.g., deploying a service, rotating secrets)
- ☐ Offer code samples and boilerplate projects
- ☐ Run regular training sessions or build community (e.g., forum, discord, Teams)
- ☐ Encourage feedback channels for workflows and tooling

Conclusion

Kubernetes is a critical tool for operations, but due to its complexity, developers may be reluctant to use it or may use it improperly. Improving developer productivity with Kubernetes will require effort such as implementing templated configurations, automating pipelines, and offering intuitive tooling. This way, organizations can reduce friction, accelerate delivery, and ensure that their applications are running successfully, therefore allowing them to benefit from what Kubernetes can bring in terms of scalability, performance, and resiliency.

Louis-Guillaume MORAND

@lgmorand

@lgmorand

Louis-Guillaume MORAND has 20 years of experience in development, architecture, and DevOps. He helps clients with cloud modernization projects and is a technical referent on DevOps and Kubernetes. He has vast experience with various DevOps tools and now spends all his time deploying cloud-native solutions, often leveraging container solutions. He loves to learn and share his knowledge with others (and he loves cats too).

PARTNER OPINION

Kubernetes Won; Developer Workflows Didn't. Here's the Fix.

By Utobong Frankson, Product Owner for Klutch at anynines

anynines is the primary sponsor and core maintainer of the open-source Klutch project

Kubernetes has transformed how organizations build and run applications, but the developer experience hasn't kept pace. The ecosystem now offers countless tools, operators, and services, each solving a piece of the puzzle. This diversity accelerates innovation but also creates friction: Developers face inconsistent workflows, while platform teams struggle to tame growing sprawl. Kubernetes promises agility, but without the right abstractions — self-service, policy-backed APIs that standardize how teams request, operate, and observe data services — many end up managing endless configs instead of building software.

The Challenge

Three out of four organizations are no longer just running a cluster — they're running many. Each comes with its own operators, manifests, and policies, stacking dozens of tools. The result? Tool sprawl that fragments the developer experience and overwhelms platform teams. It's a challenge felt across the industry; according to DZone's 2024 Kubernetes in the Enterprise Trend Report, 75% of organizations now run Kubernetes clusters, and nearly 80% use Kubernetes in development or production.

For developers, the pain is tangible. Instead of focusing on building features, they're pulled into infrastructure work. For platform and operations teams, the challenge is multiplied when managing databases and data services across clusters inherently comes with handling a patchwork of operators, hyperscaler services, and VM-based solutions. The result is operational toil, rising costs, and gaps in security and observability.

Emerging Approach: Platform Engineering for Data Services

To address these challenges, organizations are adopting platform engineering as an approach that abstracts away complexity and provides developers with a consistent, self-service experience. When applied to data services, this shift creates clarity where there was fragmentation. Common practices include:

- **Abstraction over infrastructure:** Developers declare what they need (e.g., PostgreSQL) without worrying about whether it comes from an operator, a hyperscaler, or a VM back end.
- **Lifecycle automation:** Backups, restores, and upgrades are automated, reducing repetitive operational work.
- **Ephemeral environment:** Ephemeral environments let developers spin up databases for pull requests or tests, accelerating feedback.
- **Policy as code:** Security, compliance, and cost controls are baked into the platform, enforced consistently across clusters.
- **Multi-cluster orchestration:** Emerging approaches like open-source frameworks (e.g., Klutch) provide a control-plane perspective, not to orchestrate every cluster resource but to solve the specific challenge of consistent data service orchestration across many environments.

Developer Experiences

The impact is immediate: Developers onboard faster and get shorter feedback loops, while platform teams maintain governance at scale. Organizations that centralize data service management cut onboarding time 5x versus manual setup, while ops teams see fewer manual interventions for updates and backups.

Final Thoughts

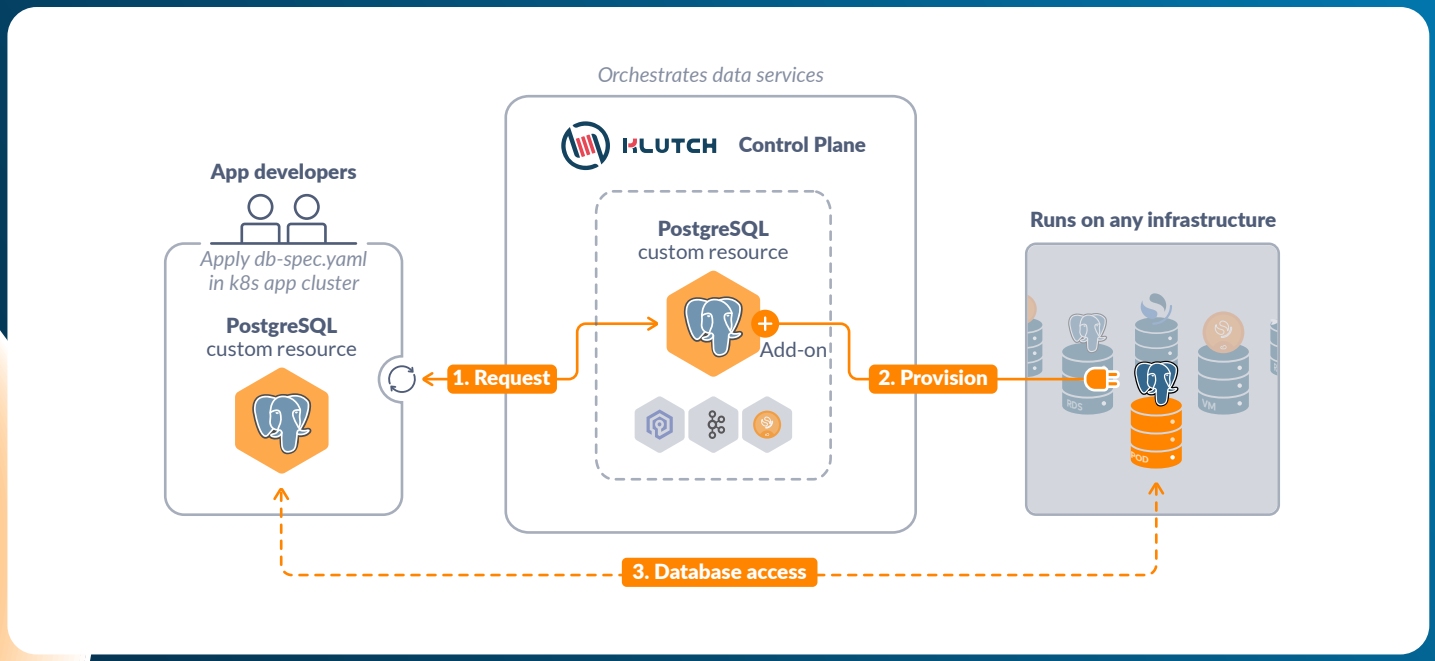
Kubernetes excels at orchestrating stateless workloads, but without a platform approach, stateful services like databases remain fragmented and inefficient. As adoption grows, the cost of sprawl in time, money, and risk will only increase. Smart platform engineering provides the antidote for organizations to move to a future where Kubernetes enables both developer velocity and operational clarity. To learn more about abstracting, automating, and standardizing how databases and data services are managed in Kubernetes, visit anynines.com/klutch.

anynines sponsors KLUTCH

Easy cloud-native data service orchestration

You've built a platform but now your developers are stuck waiting on databases, cache, logs, and more. With Klutch as your centralized control plane, teams get secure, self-service data services across Kubernetes, VMs, and clouds without the bottlenecks.

RUNS ON | On-prem | aws | vmware by Broadcom | Microsoft Azure | + more



Accelerate delivery

Give devs self-service access to databases with built-in lifecycle automation directly from Kubernetes.

Simplify operations

Centrally manage data services while keeping apps distributed across clusters, clouds, and environments.

Stay open & flexible

Support Kubernetes operators, VM-based services, and hyperscaler databases, without vendor lock-in.

Learn more at anynines.com/klutch

CONTRIBUTOR INSIGHTS

Running AI/ML on Kubernetes: From Prototype to Production

Use MLflow, KServe, and vLLM on Kubernetes to Ship Models With Confidence

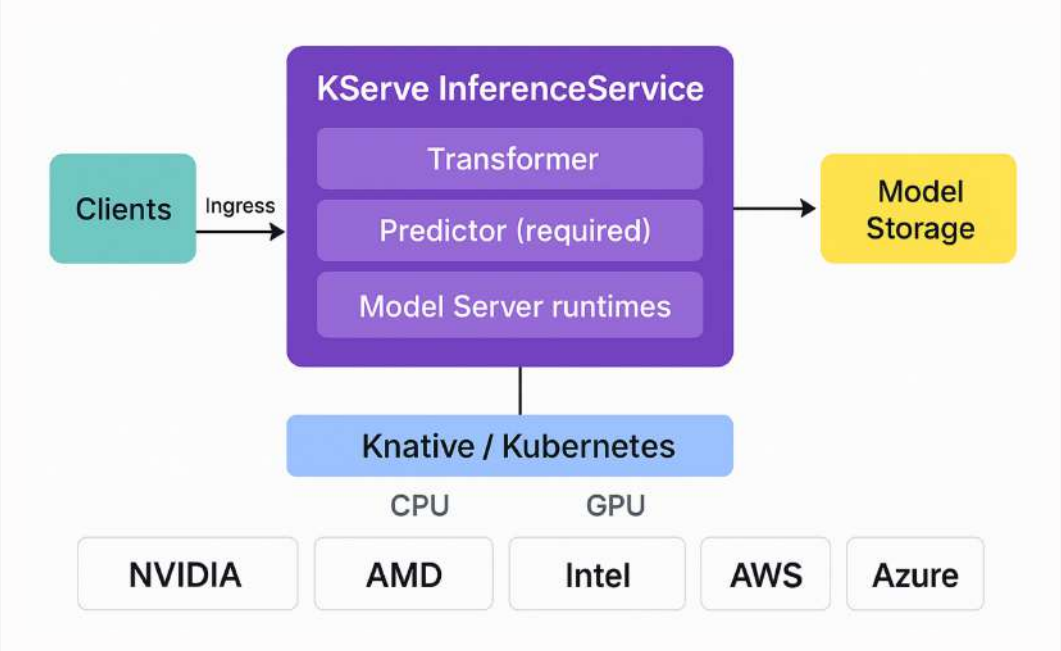
By Boris Zaikin, Leading Architect at CloudAstro

After training a machine learning model, the inference phase must be fast, reliable, and cost efficient in production. Serving inference at scale, however, brings difficult problems: GPU/resource management, latency and batching, model/version rollout, observability, and orchestration of ancillary services (preprocessors, feature stores, and vector databases). Running artificial intelligence and machine learning (AI/ML) on Kubernetes gives us a scalable, portable platform for training and serving models. Kubernetes schedules GPUs and other resources so that we can pack workloads efficiently and autoscale to match traffic for both batch jobs and real-time inference. It also coordinates multi-component stacks — like model servers, preprocessors, vector DBs, and feature stores — so that complex pipelines and low-latency endpoints run reliably.

Containerization enforces reproducible environments and makes CI/CD for models practical. Built-in capabilities like rolling updates, traffic splitting, and metrics/tracing help us run safe production rollouts and meet SLOs for real-time endpoints. For teams that want fewer operations, managed endpoints exist, but Kubernetes is the go-to option when control, portability, advanced orchestration, and real-time serving matter.

Let's look into typical ML inferencing setup using KServer on Kubernetes below:

Figure 1. ML inference setup with KServe on Kubernetes



Clients (e.g., data scientists, apps, batch jobs) send requests through ingress to a KServe InferenceService. Inside, an optional **Transformer** pre-processes inputs, the **Predictor** (required) loads the model and serves predictions, and an optional explainer returns insights. Model artifacts are pulled from **model storage** (as seen in the diagram) and served via the chosen runtime (e.g., TensorFlow, PyTorch, scikit-learn, ONNX, Triton).

Everything runs on Knative/Kubernetes with autoscaling and routing, using the **CPU/GPU** compute layer from providers such as NVIDIA/AMD/Intel on AWS, Azure, Google Cloud, or on-prem.

Part 1: MLFlow and KServe With Kubernetes

Let's dive into the practical implementation of an AI/ML scenario. We will use a combination of MLFlow to orchestrate ML processes, scikit-learn to train ML models, and KServe to inference our model in Kubernetes clusters.

Introduction to MLFlow

MLflow is an open-source ML framework, and we use it to bring order to the chaos that happens when models move from experiments to production. It helps us track runs (parameters, metrics, and files), save the exact environment and code that produced a result, and manage model versions so that we know which one is ready for production.

In plain terms, MLflow fixes three common problems:

1. Lost experiment data
2. Missing environment or code needed to reproduce results
3. Confusion about which model is "the" production model; its main pieces — Tracking, Projects, Models, and the Model Registry — map directly to those needs

We can also use MLflow to package and serve models (locally as a Docker image or via a registry), which makes it easy to hand off models to a serving platform like Kubernetes.

Using MLflow and KServe to Kubernetes

MLflow offers a straightforward way to serve models via a FastAPI-based inference server, and `mlflow models build-docker` lets you containerize that server for Kubernetes deployment. However, this approach can be unsuitable for production at scale; FastAPI is lightweight and not built for extreme concurrency or complex autoscaling patterns, and manual management of numerous inference replicas creates significant operational overhead.

KServe (formerly KFServing) delivers a production-grade, Kubernetes-native inference platform with high-performance, scalable, and framework-agnostic serving abstractions for popular ML libraries such as TensorFlow, XGBoost, scikit-learn, and PyTorch.

We've created a short step-by-step guide on how to train an ML model with MLflow and scikit-learn, and how to deploy to Kubernetes using KServe. This guide walks you through a complete MLflow workflow to train a linear-regression model with MLflow tracking and perform hyperparameter tuning to determine the best model:

1. **Prerequisites** – Install Docker, kubectl, and a local cluster (Kind or Minikube) or use a cloud Kubernetes cluster. See Kind/Minikube quickstarts.
2. **Install MLflow + MLServer support** – Install MLflow with the MLServer extras (`pip install mlflow[mlserver]`) and review MLServer examples for MLflow.
3. **Train and log a model** – Train and save the model with `mlflow.log_model()` (or `mlflow.sklearn.autolog()`), following the MLflow tutorial.
4. **Smoke-test locally** – Serve with MLflow/MLServer to validate invocations before Kubernetes: `mlflow models serve -m models:/<name> -p 1234 --enable-mlserver`. See MLflow models/MLServer examples.
5. **Package or publish**
   - **Option A** – Build a Docker image: `mlflow models build-docker -m runs:/<run_id>/model -n <your/image> --enable-mlserver` → push to a registry.
   - **Option B** – Push artifacts to remote storage (S3/GCS) and use the `storageUri` in KServe. Documents and examples can be found here.
6. **Deploy to KServe** – Create a namespace and apply an `InferenceService` pointing to your image or `storageUri`. See KServe's InferenceService quickstart and repo examples. On the following page is an example (Docker image method + Kubernetes) InferenceService snippet.

```yaml
apiVersion: "serving.kserve.io/v1beta1"
kind: InferenceService
metadata:
  name: mlflow-wine-classifier
  namespace: mlflow-kserve-test
spec:
  predictor:
    containers:
      - name: mlflow-wine-classifier
        image: "<your_docker_user>/mlflow-wine-classifier"
        ports:
          - containerPort: 8080
            protocol: TCP
        env:
          - name: PROTOCOL
            value: "v2"
```

7. **Verify and productionize** – Check Pods (`kubectl get pods -n <ns>`), call the endpoint, then add autoscaling, metrics, canary rollouts, and explainability as needed (KServe supports these features).

The official MLflow documentation also has a good step-by-step guide that covers how to package the model artifacts and dependency environment as an MLflow model, validate local serving with mlserver using `mlflow models serve`, and deploy the packaged model to a Kubernetes cluster with KServe.

Part 2: Managed AutoML: Azure ML to AKS

For this example, we selected Azure. However, Azure is just one of many tool providers that can work in this scenario. Azure Machine Learning is a managed platform for the full ML lifecycle — experiment tracking, model registry, training, deployment, and MLOps — that helps teams productionize models quickly.

Defining a reliable ML process can be difficult, and Automated ML (AutoML) can simplify that work by automating algorithm selection, feature engineering, and hyperparameter tuning. For low-latency, real-time inference at scale you can run containers on Kubernetes, the de facto orchestration layer for production workloads.

We pick Azure Kubernetes Service (AKS) when we need custom runtimes, strict performance tuning (GPU clusters, custom drivers), integration with existing Kubernetes infrastructure (service mesh, VNETs), or advanced autoscaling rules. If we prefer a managed, low-ops path and don't need deep cluster control, Azure ML's managed online endpoints are usually faster to adopt.

We run AutoML in Azure ML to find the best model, register it, and publish it as a low-latency real-time endpoint on AKS so that we keep full control over runtime, scaling, and networking:

1. **Prerequisites** – Acquire an Azure subscription, an Azure ML workspace, the Azure CLI/ML CLI or SDK, and an AKS Cluster (create one or attach an existing cluster).
2. **Run AutoML and pick the winner** – Submit an AutoML job (classification/regression/forecast) from the Azure ML studio or SDK and register the top model in the Model Registry.
3. **Prepare scoring + environment** – Add a minimal `score.py` (load model, handle request) and an environment spec (Conda/requirements); you can reuse examples from the azureml-examples repo.
4. **Attach AKS and deploy** – Attach your AKS compute to the workspace (or create AKS), then deploy the registered model as an online/real-time endpoint using the Azure ML CLI or Python SDK.
5. **Test and monitor** – Call the endpoint, add logging/metrics and autoscaling rules, and use rolling/canary swaps for safe updates.

The figure on the following page shows a typical AI/ML pipeline as an example of how AutoML works.

This ML pipeline contains steps to select, clean up, and transform data from datasets; to split data for training, selecting the ML algorithm, and testing the model; and finally, to score and evaluate the model.

These steps can be automated with AutoML, including several options to deploy models to the AKS/Kubernetes Real-Time API endpoint.

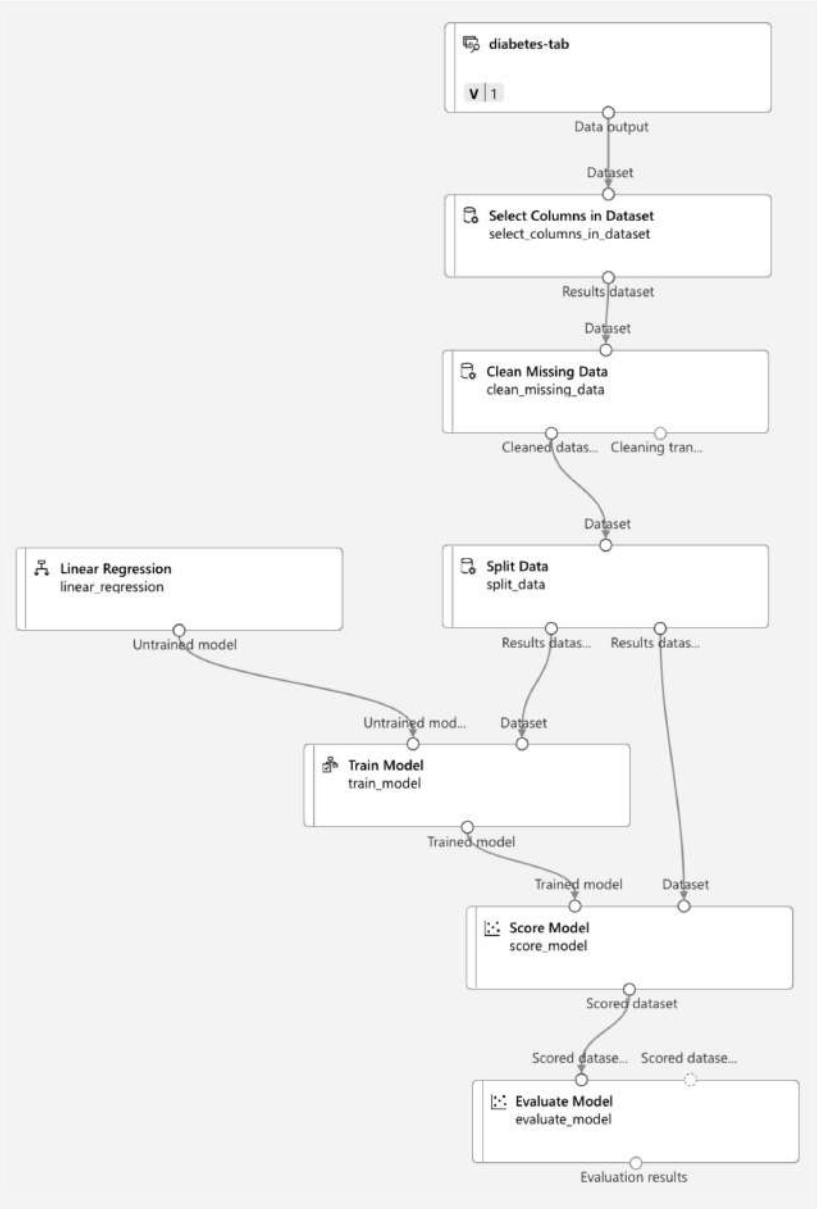
Part 3: Serving LLMs on Kubernetes

Let's have a look into the combination of LLMs and Kubernetes. We run LLMs on Kubernetes to get reliable, scalable, and reproducible inference: Kubernetes gives us GPU scheduling, autoscaling, and the orchestration primitives to manage large models, batching, and multi-instance serving.

By combining optimized runtimes, request batching, and observability (metrics, logging, and health checks), we can deliver low-latency APIs while keeping costs and operational risks under control. To do so, we can use the open-source framework vLLM, which is used when we need high-throughput, memory-efficient LLM inference.

On Kubernetes, we run vLLM inside containers and couple it with a serving control plane (like KServe) so that we get autoscaling, routing, canary rollouts, and the standard InferenceService CRD without re-implementing ops logic. This combination gives us both the low-level performance of vLLM and the operational features of a Kubernetes-native inference platform.

Figure 2. Example AI/ML pipeline



Let's see how we can deploy LLM to Kubernetes using vLLM and KServe:

1. **Prepare cluster and KServe** – Provision a Kubernetes cluster (AKS/GKE/EKS or on-prem) and install KServe, following the quickstart.
2. **Get vLLM** – Clone the vLLM repo or follow the docs to install vLLM and test `vllm serve` locally to confirm that your model loads and the API works.
3. **Create a vLLM ServingRuntime/container** – Build a container image or use the vLLM ServingRuntime configuration that KServe supports (the runtime wraps `vllm serve` with the correct arguments and environment variables).
4. **Deploy an InferenceService** – Apply a KServe InferenceService that references the vLLM serving runtime (or your image) and model storage (S3/HF cache). KServe will create pods, handle routing, and expose the endpoint.
5. **Validate and tune** – Hit the endpoint (through ingress/port-forward), measure latency/throughput, and tune vLLM batching/token-cache settings and KServe autoscaling to balance latency and GPU utilization.

Last but not least, we can run vLLM, KServe, and BentoML together to get high-performance LLM inference and production-grade ops. Here is a short breakdown:

- **vLLM** – the high-throughput, GPU-efficient inference engine (token generation, KV-cache, and batching) — the runtime that actually executes the LLM
- **BentoML** – the developer packaging layer that wraps model loading, custom pre-/post-processing, and a stable REST/gRPC API, then builds a reproducible Docker image or artifact
- **KServe** – the Kubernetes control plane that deploys the container (Bento image or a vLLM serving image) and handles autoscaling, routing/ingress, canaries, health checks, and lifecycle management

How do they fit together? We package our model and request logic with BentoML (image or Bento bundle), which runs the vLLM server for inference. KServe then runs that container on Kubernetes as an InferenceService (or ServingRuntime), giving autoscale, traffic controls, and observability.

Pros and Cons of Kubernetes Inference Frameworks for ML

We already had a look at the KServe library. However, there are other powerful alternatives. Let's look at the table below:

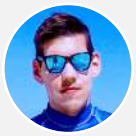
Table 1. KServe alternative tools and libraries

Library	Overview	Pros	Cons
Seldon Core	Kubernetes-native ML serving and orchestration framework offering CRDs for deployments, routing, and advanced traffic control	Kubernetes-first (CRDs, Istio/Envoy integrations); rich routing (canary, A/B); built-in telemetry and explainer integrations; supports multiple runtimes	Steeper learning curve; more operational surface to manage; heavier cluster footprint
BentoML (with Yatai)	Python-centric model packaging and serving; Yatai/Helm lets you run Bento services on Kubernetes as deployments/CRDs	Excellent developer ergonomics and reproducible images; fast local dev loop; simple CI/CD image artifacts	Less cluster-native controls out of the box (needs Yatai/Helm); autoscaling and advanced Kubernetes ops require extra setup
NVIDIA Triton Inference Server	High-performance GPU-optimized inference engine supporting TensorRT, TensorFlow, PyTorch, ONNX, and custom back ends	Exceptional GPU throughput and mixed-framework support; batch and model ensemble optimizations; production-grade performance tuning	Less cluster-native controls out of the box (needs Yatai/Helm); autoscaling and advanced Kubernetes ops require extra setup

Conclusion

Our goal is to run reliable, low-latency AI/ML in production while keeping control of cost, performance, and repeatability. Kubernetes gives us the orchestration primitives we need — GPU scheduling, autoscaling, traffic control, and multi-service coordination — so that models and their supporting services can run predictably at scale. Paired with optimized runtimes, serving layers, and inference engines, we get both high-inference performance and production-grade operational controls. The result is portable, reproducible deployments with built-in observability, safe rollout patterns, and better resource efficiency.

Start small, validate with a single model and clear SLOs, pick the serving stack that matches your performance and ops needs, then iterate. Kubernetes lets you grow from prototype to resilient, scalable serving.



CORE

Boris Zaikin

@borisza

boriszaikin.com

Leading 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, multitenant distributed systems, and IoT solutions.

ADDITIONAL RESOURCES

Solutions Directory

This directory contains managed Kubernetes platforms, service mesh and networking, security, policy, observability, and cost-management tools to help you evaluate, deploy, secure, and operate Kubernetes at scale. 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 2025 KUBERNETES SOLUTIONS DIRECTORY

2025 PARTNERS

Product	Purpose	Availability	Website
anynines	Cloud-native platforms for data service automations providers	By request	anynines.com
a9s Data Services	Managed database and data service automations on Kubernetes and VMs	Trial period	anynines.com/products/data-services
anynines' Klutch	Commercial distribution of Klutch with multi-tenancy and more	By request	anynines.com/klutch
Klutch	Control plane for data service provisioning and lifecycle management	Open source	klutch.io
Veeam Kasten	Kubernetes data protection and mobility	Trial period	veeam.com/products/cloud/kubernetes-data-protection.html
Company	**Purpose**	**Availability**	**Website**
Akamai Linode Kubernetes Engine (LKE)	Fully managed Kubernetes container orchestration engine	By request	linode.com/products/kubernetes
Amazon Elastic Kubernetes Service (EKS)	Managed Kubernetes service	By request	aws.amazon.com/eks
AWS Fargate	Serverless compute for containers; integrates with Amazon EKS and ECS	By request	aws.amazon.com/fargate
Ambassador Labs Edge Stack API Gateway	Kubernetes-native API gateway	Free tier	getambassador.io/products/edge-stack/api-gateway
Anchore Enterprise	Software composition analysis for cloud-native applications	Trial period	anchore.com
Apptio IBM Kubecost	Kubernetes cost monitoring	Free tier	apptio.com/products/kubecost
Aqua CNAPP	Cloud-native security platform	By request	aquasec.com/aqua-cloud-native-security-platform
Aqua kube-hunter	Kubernetes penetration testing tool to identify cluster security weaknesses	Open source	github.com/aquasecurity/kube-hunter
Aqua Trivy	Scanner for container images, file systems, Git repos, and Kubernetes	Open source	aquasecurity.github.io/trivy
Argo CD	GitOps continuous delivery for Kubernetes	Open source	argo-cd.readthedocs.io
Argo Workflows	Container-native workflow engine for Kubernetes	Open source	argo-workflows.readthedocs.io
CAST AI	Kubernetes automation	Free tier	cast.ai
Chronosphere	Observability with metrics, logs, traces, and telemetry pipelines	By request	chronosphere.io
Cilium (OSS)	eBPF-based networking, security, and observability for Kubernetes	Open source	cilium.io

DZONE'S 2025 KUBERNETES SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
CloudBees CI	CI for Jenkins in the enterprise	By request	cloudbees.com/capabilities/continuous-integration
CNCF Kubespray	Ansible-based production-ready Kubernetes cluster installation and lifecycle tool	Open source	kubespray.io
Cortex	Internal developer portal	By request	cortex.io
DaoCloud Enterprise 5.0	Cloud-native operating system	Free tier	daocloud.io
Datadog Container Monitoring	Monitor and secure containerized environments	Free tier	datadoghq.com/product/container-monitoring
derailed K9s	Terminal-based UI to interact with Kubernetes	Open source	k9scli.io
Popeye	Kubernetes cluster sanitizer that lints live clusters for best-practice issues	Open source	github.com/derailed/popeye
Diamanti Ultima Enterprise	High-performance Kubernetes storage and data management	Trial period	diamanti.com/products/ultima-enterprise
DigitalOcean Kubernetes	Managed Kubernetes clusters	By request	digitalocean.com/products/kubernetes
Docker Desktop	Container software	Free tier	docker.com/products/docker-desktop
Docker Hub	Container registry and image distribution	Free tier	docker.com/products/docker-hub
Dynatrace Platform	End-to-end observability with AIOps and application security	Trial period	dynatrace.com
F5 Aspen Mesh	Deploy, manage, and monitor microservices	By request	f5.com/products/aspen-mesh
F5 NGINX Ingress Controller	Kubernetes-native API gateways, load balancers, ingress controllers	Trial period	f5.com/products/nginx/nginx-ingress-controller
Fairwinds Managed Kubernetes-as-a-Service	Production-ready Kubernetes: architected, built, managed	By request	fairwinds.com/managed-kubernetes
Flexera Ocean	Kubernetes infrastructure automation and right-sizing on-demand instances	Trial period	flexera.com/spot/ocean
Flux CD	GitOps-based continuous delivery for Kubernetes	Open source	fluxcd.io
Google Kubernetes Engine	Scalable, fully automated Kubernetes service	Free tier	cloud.google.com/kubernetes-engine
Grafana Cloud	Fully managed observability with Kubernetes monitoring and dashboards	Free tier	grafana.com/products/cloud
Grafana Labs k6	Load testing tool with cloud service option; integrates with Kubernetes CI/CD	Open source	k6.io
Grafana Tempo	Distributed tracing back end	Open source	grafana.com/oss/tempo
Harness	Software delivery pipeline automation	Free tier	harness.io
Helm	Kubernetes package manager (charts)	Open source	helm.sh
Huawei Cloud Container Engine	Fully hosted Kubernetes service	By request	huaweicloud.com/intl/en-us/product/cce.html
IBM Cloud Kubernetes Service	Managed Kubernetes platform	Trial period	ibm.com/products/kubernetes-service
Istio	Service mesh for traffic management, security, and observability	Open source	istio.io
Jenkins	CI server for building and deploying applications	Open source	jenkins.io

DZONE'S 2025 KUBERNETES SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
KEDA	Event-driven autoscaling for Kubernetes (including scale-to-zero)	Open source	keda.sh
Komodor	Cluster management and troubleshooting	Trial period	komodor.com
Kong Ingress Controller	Kubernetes-native API management	Trial period	konghq.com/products/kong-ingress-controller
Kong Mesh	Enterprise service for Kubernetes	By request	konghq.com/products/kong-mesh
Kubeflow	ML toolkit on Kubernetes: pipelines, training, serving, and notebooks	Open source	kubeflow.org
Kubermatic KubeOne	Automation of Kubernetes cluster management	By request	kubermatic.com/products/kubermatic-kubeone
Kubermatic Kubernetes Platform	Automated hybrid and multi-cloud Kubernetes	By request	kubermatic.com/products/kubermatic-kubernetes-platform
Kustomize (SIG CLI)	Kubernetes-native configuration management tool	Open source	kustomize.io
kube-state-metrics (SIG Instrumentation)	Expose cluster object states as Prometheus metrics	Open source	github.com/kubernetes/kube-state-metrics
Minikube (SIGs)	Run a single-node Kubernetes cluster locally for development and testing	Open source	minikube.sigs.k8s.io
Kyverno (OSS)	Kubernetes-native policy engine (validate/mutate/generate via CRDs)	Open source	kyverno.io
Linkerd	Lightweight service mesh for Kubernetes	Open source	linkerd.io
Litmus	Chaos engineering for Kubernetes and resilience testing	Open source	litmuschaos.io
Loft vCluster	Lightweight, virtual Kubernetes clusters	Free tier	vcluster.com
Logz.io Open 360 Platform	Full infrastructure and application observability	Trial period	logz.io
Loki	Log aggregation optimized for labels and Prometheus-style queries	Open source	grafana.com/oss/loki
Metalstack.cloud	Automate bare-metal infrastructure for Kubernetes clusters	By request	metalstack.cloud/en
Mia-Platform	Build IDPs, self-serve developers, and ship applications faster	By request	mia-platform.eu
Microsoft Azure Kubernetes Service	Managed Kubernetes platform	Free tier	azure.microsoft.com/en-us/products/kubernetes-service
MinIO	Object storage solution	Open source	min.io
Mirantis Openstack for Kubernetes	Containerized OpenStack on Kubernetes to automate ops and lifecycle management	Free trial	mirantis.com/software/mirantis-openstack-for-kubernetes
Mirantis Container Runtime	Secure, industry-standard container runtime	By request	mirantis.com/software/container-runtime
Mirantis Kubernetes Engine	Container orchestration	Trial period	mirantis.com/software/mirantis-kubernetes-engine
Mirantis Secure Registry	Enterprise container registry	Trial period	mirantis.com/software/mirantis-secure-registry
NetApp Trident	Provision and protect stateful Kubernetes applications	Open source	netapp.com/trident
New Relic Kubernetes Pixie	Codeless eBPF-based observability for Kubernetes	Free tier	newrelic.com/platform/kubernetes-pixie

DZONE'S 2025 KUBERNETES SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
ngrok Kubernetes Operator	Globally distributed reverse proxy	Free tier	ngrok.com
Nirmata Policy Manager	Kubernetes security, automation, and governance	Free tier	nirmata.com/policy-manager
Nutanix Kubernetes Platform	Kubernetes management	Trial period	nutanix.com/products/kubernetes-management-platform
Okteto	Developer environment automation	Free tier	okteto.com
OCI Container Engine for Kubernetes (OKE)	Managed Kubernetes service	Free tier	oracle.com/cloud/cloud-native/container-engine-kubernetes
Palo Alto Prisma Cloud	CNAPP	By request	paloaltonetworks.com/prisma/cloud
Portainer	Container management software	Free tier	portainer.io
Portworx Platform	Container data management	Free tier	portworx.com/platform
Prometheus	Metrics collection, time series storage, and alerting	Open source	prometheus.io
Rafay Cloud-Native Platform	Automation for Kubernetes operations, security, and application delivery	Trial period	rafay.co/platform/accelerate-cloud-native-adoption
Rancher	Kubernetes management platform	Free tier	rancher.com
Red Hat Ansible Automation Platform	IT automation	By request	docs.ansible.com
Red Hat OpenShift	Build, modernize, and deploy applications	Trial period	redhat.com/en/technologies/cloud-computing/openshift
Red Hat Quay	Container registry	Trial period	quay.io
Release Delivery	Container-based EaaS platform	By request	release.com/product/release-delivery
Replicated Platform	Kubernetes application delivery and management platform	Trial period	replicated.com
ScaleOps	Automated Kubernetes resource optimization	Trial period	scaleops.com
Scaleway Container Registry	Docker repository	Free tier	scaleway.com/en/container-registry
Scaleway Kubernetes Kapsule	Managed Kubernetes	Free tier	scaleway.com/en/kubernetes-kapsule
Scaleway Kubernetes Kosmos	Pod launch and Kubernetes workload management	By request	scaleway.com/en/kubernetes-kosmos
Snyk Container	Container and Kubernetes security	Free tier	snyk.io/product/container-vulnerability-management
Solo.io Gloo Platform	Unified application networking for APIs	By request	solo.io
Soluto Kamus	Encrypts/decrypts secrets for Kubernetes applications	Open source	github.com/Soluto/kamus
Spinnaker	Multi-cloud continuous delivery platform with Kubernetes support	Open source	spinnaker.io
Sumo Logic	SaaS log analytics platform	Free tier	sumologic.com
SUSE Security	Full lifecycle container security	Free tier	suse.com/products/rancher/security
Sysdig Monitor	Kubernetes and cloud monitoring with a managed Prometheus service	By request	sysdig.com/products/monitor
Sysdig Secure	Cloud and container security platform	By request	sysdig.com/products/platform

DZONE'S 2025 KUBERNETES SOLUTIONS DIRECTORY

Product	Purpose	Availability	Website
Teleport Zero Trust Access	Scalable secure access for Kubernetes	By request	goteleport.com/platform/protected-identities/kubernetes
Tigera Calico Open Source	Networking, network security, and observability for Kubernetes	Open source	tigera.io/tigera-products/calico-cloud
Tigera Calico Commercial Editions	Unified network security and observability for Kubernetes	Trial period	tigera.io/tigera-products/calico-commercial-editions
Traefik Enterprise	Unified API gateway and Ingress	Trial period	traefik.io/traefik-enterprise
Traefik Hub	Kubernetes-native API management	Trial period	traefik.io/traefik-hub
Trilio Kubernetes Backup	Kubernetes backup and recovery for cloud or on-prem	By request	trilio.io/products/kubernetes-backup-and-recovery
Upwind Cloud Security Platform	CNAPP	By request	upwind.io
Velero (backup)	Backup and restore for clusters and persistent volumes	Open source	velero.io
VMWare Tanzu (Broadcom) Platform	Accelerate application delivery	By request	vmware.com/products/app-platform/tanzu
Wiz CNAPP	Unified cloud security from prevention to detection	By request	wiz.io/platform

DZone

SOFTWARE DESIGN AND ARCHITECTURE

Cloud Architecture | Containers | Security Integration | Microservices | Performance

Software design and architecture focus on the development decisions made to improve a system's overall structure and behavior in order to achieve essential qualities such as modifiability, availability, and security.

The Zones in this category are available to help developers stay up to date on the latest software design and architecture trends and techniques.

VISIT THE ZONE

TUTORIALS

TECHNIQUES

CODE SNIPPETS

RESEARCH

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.