TREND REPORT

DZone

SEPTEMBER 2024

# Kubernetes in the Enterprise

## Once Decade-Defining, Now Forging a Future Beyond Containerization

BROUGHT TO YOU IN PARTNERSHIP WITH

Spot by NetApp

## Welcome Letter

**By Abhishek Gupta, Principal Developer Advocate at AWS**

A decade has passed since Kubernetes emerged in the tech world — a span that represents a significant portion of many IT careers. Amusingly, some of us may recall job postings demanding extensive Kubernetes experience long before such a feat was even possible. We can finally claim to have "10 years of Kubernetes experience"!

To be honest, when I first encountered Kubernetes many years ago, I was skeptical. The steep learning curve and challenges of running it in production were daunting. However, as I gradually peeled back the layers, I began to appreciate its design; I discovered a solid platform that had not only abstracted foundation components such as compute, storage, and networking, but also container runtimes, cloud providers, and a way to extend Kubernetes using CRDs.

I knew that I had just scratched the surface, but the realization of Kubernetes' extensibility was my true "aha" moment!

It's clear that Kubernetes isn't just a technology — it's a movement. It's reshaping how we think about application architecture, deployment, and scalability. Kubernetes has long outgrown its "buzzword" status, and it finds itself in a unique position: a mature technology with much left to offer.

If the past decade is any indication, the next 10 years of Kubernetes promise to be even more transformative.

As the world of application development continues to evolve, Kubernetes is poised to evolve right alongside it, with the community prioritizing changes that both improve user experiences and enhance the project's sustainability.

The 2024 *Kubernetes in the Enterprise* Trend Report will dive into the current and evolving state of Kubernetes. Articles written by DZone community experts explore how Kubernetes streamlines the SDLC, the challenges of managing AI/ML workloads in Kubernetes, mitigating container-specific threats, and considerations for Kubernetes observability. The "Key Research Findings" section supports these articles in addition to providing details on cost optimization, architecture, and design.

Drawing from real-world experiences and production environments, this report offers practical guidance to help you navigate the complexities of Kubernetes implementation and optimization in enterprise settings. Whether you're a seasoned Kubernetes professional or just starting out, we hope this report equips you with actionable strategies and best practices that you can apply directly to your own Kubernetes deployments.

Best,

Abhishek Gupta

CORE

**Abhishek Gupta**

@abhirockzz

@abhirockzz

@abhi_tweeter

Over the course of his career, Abhishek has worn multiple hats including engineering, product management, and developer advocacy. Most of his work has revolved around open-source technologies, including distributed data systems and cloud-native platforms. Abhishek is also an open source contributor and avid blogger.

ORIGINAL RESEARCH

# Key Research Findings

An Analysis of Results From DZone's 2024 Kubernetes Survey

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

In 2014, Kubernetes' first commit was pushed to production. Fast forward 10 years, and it is now one of the most prolific open-source systems in the software development space. So what is it about Kubernetes that has made it so deeply entrenched within organizations? Scale, speed, delivery — these make up the Kubernetes promise, and they're not going anywhere any time soon.

DZone's research for our fifth annual *Kubernetes in the Enterprise* Trend Report explored the nuances and evolving requirements for the now 10-year-old platform, including topics like architectural evolutions in Kubernetes, emerging cloud security threats, advancements in Kubernetes monitoring and observability, the impact and influence of AI. From May to September, 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 May 10, 2024, and September 10, 2024; we collected 139 complete and partial responses.

*Note: Time-based comparative analysis throughout these findings will compare results gathered from the same survey — focused on Kubernetes and cloud-native architectures — for DZone's Cloud Native Trend Report "Key Research Findings" and for these "Key Research Findings." Responses for the Cloud Native Trend Report were submitted between March 25, 2024, and April 29, 2024.*

## Demographics

We've noted certain key audience details below in order to establish a more solid impression of the sample from which results have been derived:

- 24% of respondents described their primary role in their organization as "Technical Architect," 22% described "Developer/Engineer," and 10% described "Developer Team Lead." No other role that we provided was selected by more than 10% of respondents.*
- 84% of respondents said they are currently developing "Web applications/Services (SaaS)," 44% said "Enterprise business applications," and 25% said "Native mobile apps."
- "Java" (72%) was the most popular language ecosystem used at respondents' companies, followed by "Python" (64%), "JavaScript (client-side)" (59%), "Node.js (server-side JavaScript)" (48%), "Go" (37%), and "TypeScript" (32%).
- Regarding responses on the primary language respondents use at work, the most popular was "Java" (42%), followed by "Python" (21%), "Go" (9%), "C#" (7%), and "Node.js" (6%). No other language was selected by more than 5% of respondents.
- On average, respondents said they have 18.08 years of experience as an IT professional, with a median of 18 years.
- 27% of respondents work at organizations with < 100 employees, 24% of respondents work at organizations with 100-999 employees, and 47% 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 2024 Kubernetes survey, we aimed to gather data regarding various topics related to the following major research targets:

1. Trends in Kubernetes and container management
2. Kubernetes' impact on other development trends

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

## Research Target One: Trends in Kubernetes and Container Management

Kubernetes use has exploded in the past few years, and now the container orchestration tool is a mainstay in contemporary software systems. We wanted to understand how organizations are currently using and managing this popular technology, and how developers feel about the impact of Kubernetes on development and software quality. In this section, we explore:

- Containers and container management
- Kubernetes use and use cases
- Developer opinions of Kubernetes
- Monitoring and observability

## Containers

Containers have become a staple of modern software architectures because they enable applications to run consistently across various environments, from local development to production, ensuring reliability and portability. By isolating applications and their dependencies, containers reduce conflicts between different systems and make it easier to test, deploy, and maintain software. They also offer a lightweight alternative to virtual machines, optimizing resource usage and allowing for easier scaling and management of applications. In order to determine the current state of container usage, we asked the following:

*Does your organization use application containers in either development or production environments?*

*What tools/platforms does your organization use to manage containers in development and production environments?*

Results:

**Figure 1.** Container use in dev and prod environments [*n*=126]

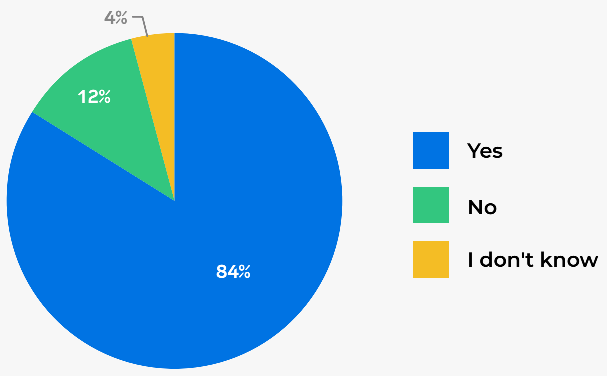


**Figure 2.** Container use: 2017–2024 [*n*=100]

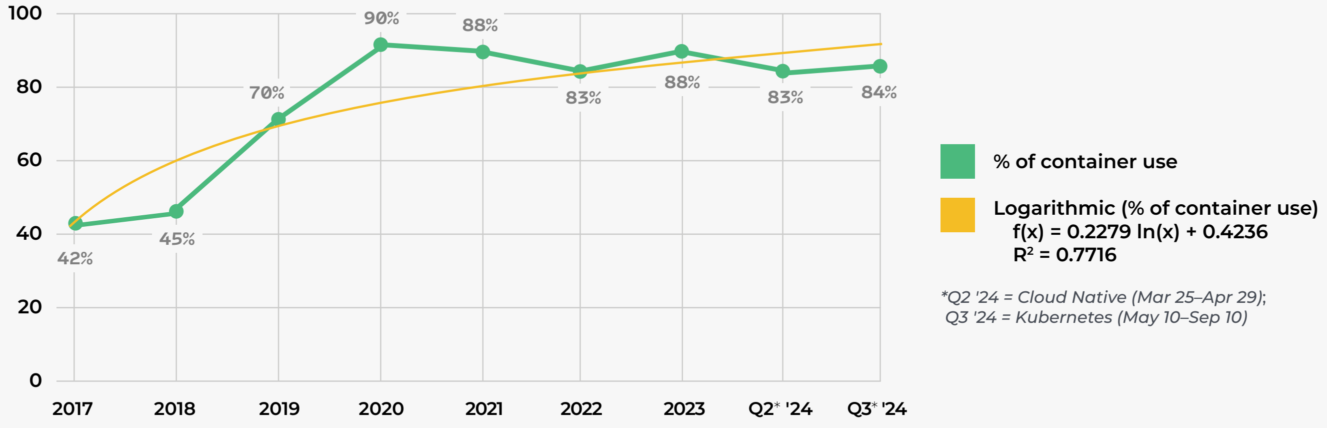


**Table 1.** Container management tools in dev and prod: *Cloud Native* vs. *Kubernetes* Trend Reports*

| Tool | Development | | | Production | | |
|---|---|---|---|---|---|---|
| | Cloud Native [*n*=179] | Kubernetes [*n*=100] | % Change | Cloud Native [*n*=179] | Kubernetes [*n*=100] | % Change |
| Docker | 71% | 83% | +12% | 50% | 64% | +14% |
| Kubernetes | 63% | 75% | +12% | 58% | 74% | +16% |
| Docker Compose | 42% | 56% | +14% | 24% | 29% | +5% |
| Terraform | 44% | 54% | +10% | 42% | 48% | +6% |
| AWS EKS | 37% | 40% | +3% | 38% | 44% | +6% |
| Ansible | 29% | 36% | +7% | 24% | 29% | +5% |
| Azure AKS | 34% | 31% | -3% | 33% | 29% | -4% |
| AWS ECS | 31% | 28% | -3% | 31% | 27% | -4% |
| GKE | 17% | 19% | +2% | 15% | 17% | +2% |
| OpenShift | 15% | 17% | +2% | 18% | 15% | -3% |
| QEMU | 7% | 6% | -1% | 3% | 1% | -2% |

**Only displays options selected by > 5% of respondents in either the Development or Production category*

## OBSERVATIONS

84% of respondents said that their organization uses containers in either development or production environments. Since 2020 — the second of two years of rapid growth in response rates of organizations' container use — between 83% and 90% of respondents have indicated that their organization uses containers in some capacity. Since 2021, we have maintained the hypothesis that container adoption has plateaued after reaching a natural saturation point, and our latest survey results support this hypothesis. Container use has been shown to be an effective pattern for web-based and cloud-native software systems, which comprise the majority of enterprise software today. As such, we believe that it will take a major shift in software architectures as a whole to cause a major deviation from this current saturation point.

As we have seen for the past few years, respondents at small organizations were much less likely than those at mid-sized and large organizations to say their organization uses containers. We mentioned in our 2024 *Cloud Native* "Key Research Findings" that the lower rate of container usage in small organizations may stem from lower complexity software systems in those companies, leading to lower benefit-cost ratios for containers, or smaller companies lacking resources for container adoption. Details on container usage segmented by organization size for this report can be found in Table 2.

**Table 2.** Container use by organization size*

| Response | Organization Size | | | Overall |
|---|---|---|---|---|
| | 1-99 | 100-999 | 1,000+ | |
| Yes | 71% | 100% | 85% | 75% |
| No | 26% | 0% | 13% | 19% |
| I don't know | 3% | 0% | 2% | 6% |
| *n*= | 31 | 28 | 55 | 126 |

**% of columns*

Once again, Docker was the most popular tool at respondents' organizations for managing containers in development, and Kubernetes was again the most popular tool for container management in production. Docker and Kubernetes saw significant increases in response rates for both development and production environments, and Docker Compose, Terraform, and Ansible saw significant increases in response rates for development environments only.

There was no significant change in most third-party container management tools, especially those linked to top cloud platforms like AWS and Azure, and Docker and Kubernetes use in development environments have returned to rates we observed in our 2023 *Kubernetes in the Enterprise* Trend Report (81% and 71%, respectively). Even so, we believe that the use of third-party container management tools will catch up to Docker and Kubernetes in the next few years as we hypothesized in our 2024 *Cloud Native* "Key Research Findings."

## Kubernetes Use and Use Cases

Kubernetes offers powerful container orchestration by automating the deployment, scaling, and management of containerized applications across clusters, ensuring high availability and fault tolerance. It simplifies operational tasks like load balancing, service discovery, and resource optimization, making it easier to manage complex microservices architectures. We saw in the previous section that Kubernetes is — and has been — the most used container management technology in production environments, second only to Docker in development environments.

To find out more about developers' experience with Kubernetes and how their organizations are using Kubernetes, we asked the following:

*Have you personally worked with Kubernetes?*

*Does your organization run any Kubernetes clusters?*

To respondents who said their organization runs Kubernetes clusters, we also asked:

*What use cases does your organization deploy Kubernetes into?*

Results:

**Figure 3.** Personal experience with Kubernetes [*n*=123]

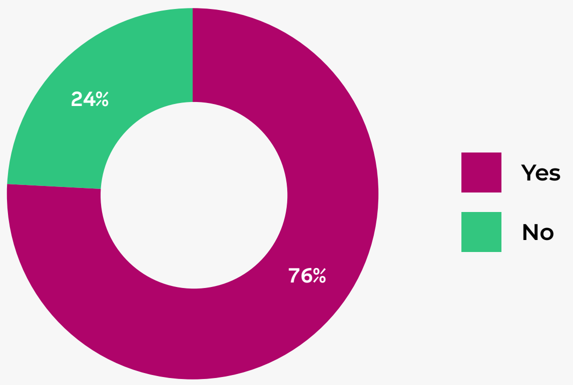
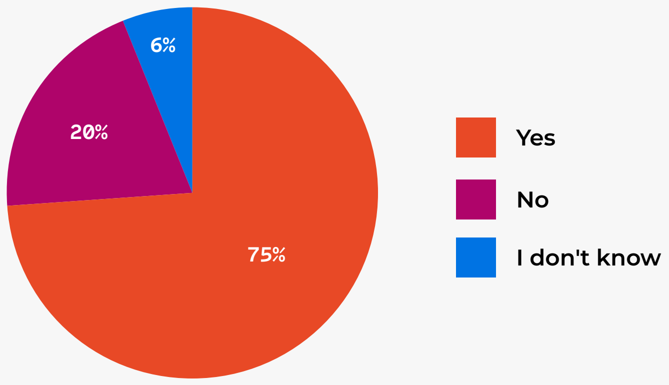
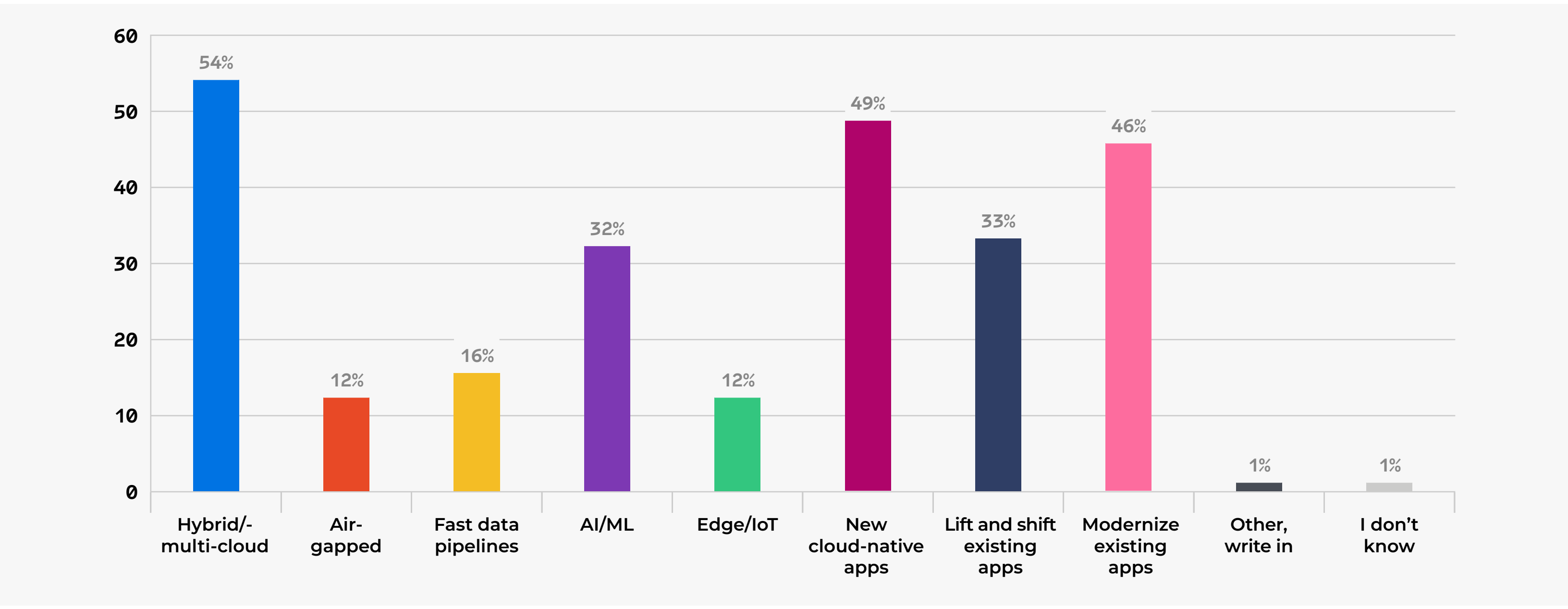


**Figure 4.** Organizations running Kubernetes clusters [*n*=126]



*SEE FIGURE 5 ON NEXT PAGE*

**Figure 5.** Kubernetes use cases [*n*=90]



## OBSERVATIONS

- 76% of respondents indicated that they have personally worked with Kubernetes, showing no significant change from our 2023 Kubernetes research findings (77%) or our 2024 cloud native research findings (79%). As we mentioned in our *Cloud Native* Trend Report, we do not anticipate seeing large changes in response rates regarding developers' personal Kubernetes use for at least two to three years.

**Table 3.** Organizational and personal Kubernetes use segmented by organization size*

| Response | Organization Size | | | Overall |
|---|---|---|---|---|
| | 1-99 | 100-999 | 1,000+ | |
| Organization | 53% | 85% | 85% | 75% |
| Personal | 65% | 68% | 89% | 76% |
| *n*= | 32 | 28 | 55 | 115 |

**% of columns*

- Three-quarters of respondents (75%) said their organization runs Kubernetes clusters, and 79% of respondents either said their organization runs Kubernetes clusters **or** said their organization uses Kubernetes in either development or production environments when asked about container management tools at their organization (which we examined in the previous section).

These response rates align more closely with those we observed in our 2023 *Kubernetes* Trend Report, where 80% of respondents said their organizations ran Kubernetes clusters, compared to those we saw in our 2024 *Cloud Native* Trend Report, where 68% of respondents said their organizations ran Kubernetes clusters.

**Table 4.** Kubernetes use cases: *Cloud Native* vs. *Kubernetes* reports

| Use Case | Trend Report | | % Change |
|---|---|---|---|
| | Cloud Native | Kubernetes | |
| Hybrid/multi-cloud | 44% | 54% | +11% |
| New cloud-native apps | 65% | 49% | -16% |
| Modernizing existing apps | 53% | 46% | -8% |
| Lift and shift existing apps | 34% | 33% | -1% |
| AI/ML | 24% | 32% | +8% |
| Fast data pipelines | 19% | 16% | -4% |
| Edge/IoT | 19% | 12% | -7% |
| Air-gapped | 12% | 12% | 0% |
| Non-ML AI | 13% | 8% | -5% |
| I don't know | 3% | 1% | -2% |
| Other, write in | 1% | 1% | 0% |
| *n*= | 144 | 90 | - |

- Respondents at small organizations were significantly less likely to say that their organization runs Kubernetes clusters than those at mid-sized or large organizations, with the latter having equal response rates. However, respondents at large organizations were much more likely to say they had personal experience with Kubernetes than others, whereas respondents at small and mid-sized organizations had no significant difference in response rates regarding personal Kubernetes experience (more details in Table 3).

"Hybrid/multi-cloud" was the most commonly selected use case for Kubernetes after a significant increase from our 2024 cloud native findings, returning to a rate aligned with results from our 2023 Kubernetes findings (52%). We also noticed a significant *decrease* in response rates for "New cloud-native apps" and "Modernizing existing apps" from our 2024 cloud native findings, with "New cloud-native apps" even falling significantly below the response rates we saw in our 2023 Kubernetes findings (58%). Further details can be found in Table 4 on the previous page.

## Developer Opinions of Kubernetes

Adopting Kubernetes can offer several advantages — automating the management of containerized applications, improving scalability, ensuring high availability and fault tolerance, and more. It simplifies complex tasks like load balancing, service discovery, and rolling updates, making it ideal for managing microservices architectures. On the other hand, Kubernetes' potentially steep learning curve and complexity can be a significant disadvantage, especially for smaller teams or those unfamiliar with container orchestration.

Developers may face challenges with initial setup, configuration, and ongoing management, as well as troubleshooting issues in large-scale deployments, which can become time consuming and resource intensive. To understand more about the advantages, disadvantages, and pain points developers have experienced when encountering Kubernetes in their organizations, we asked the following:

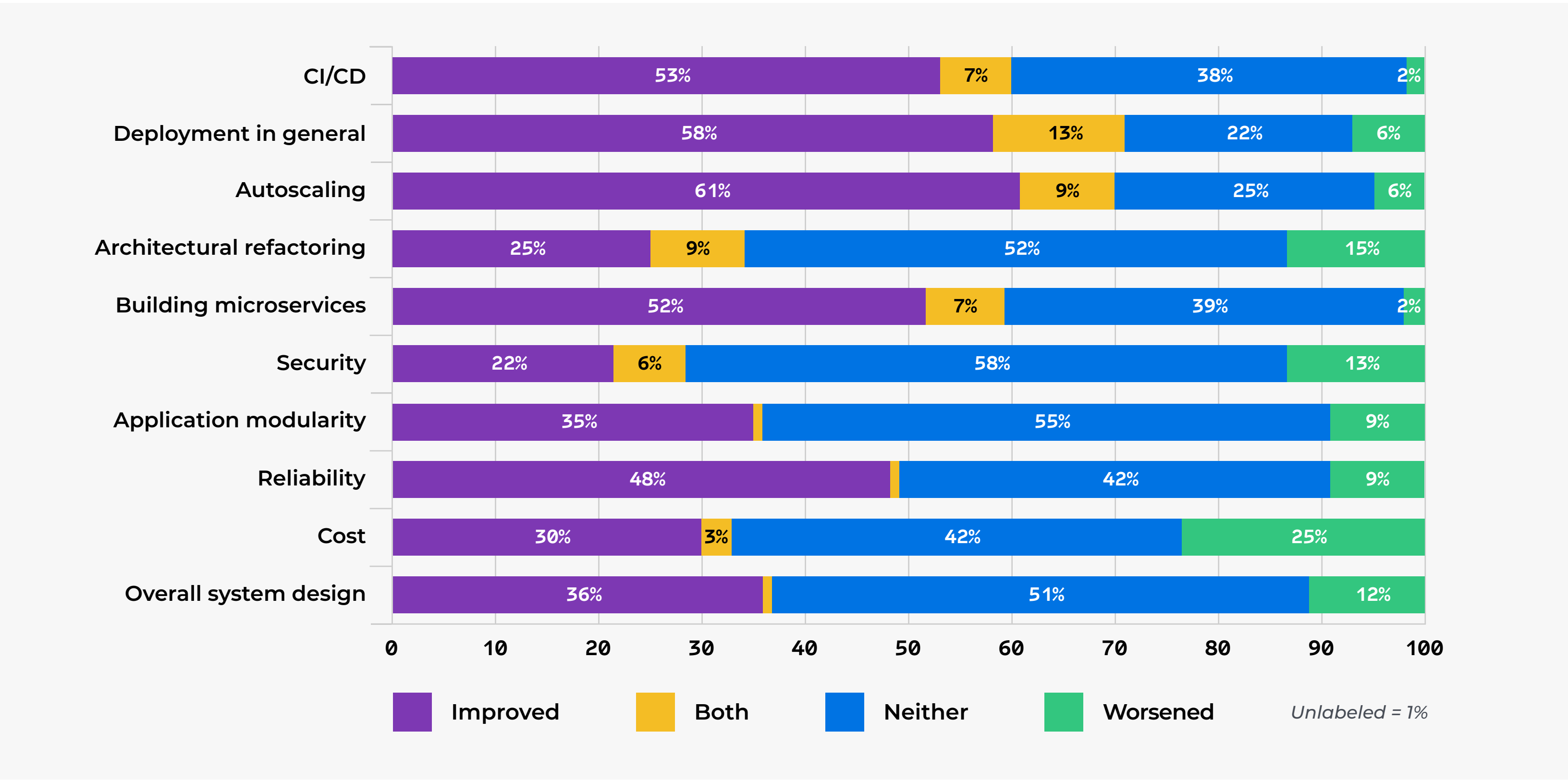
*What has Kubernetes improved at your organization?*

*What has Kubernetes worsened at your organization?*

*What pain points has your organization encountered while working with Kubernetes?*

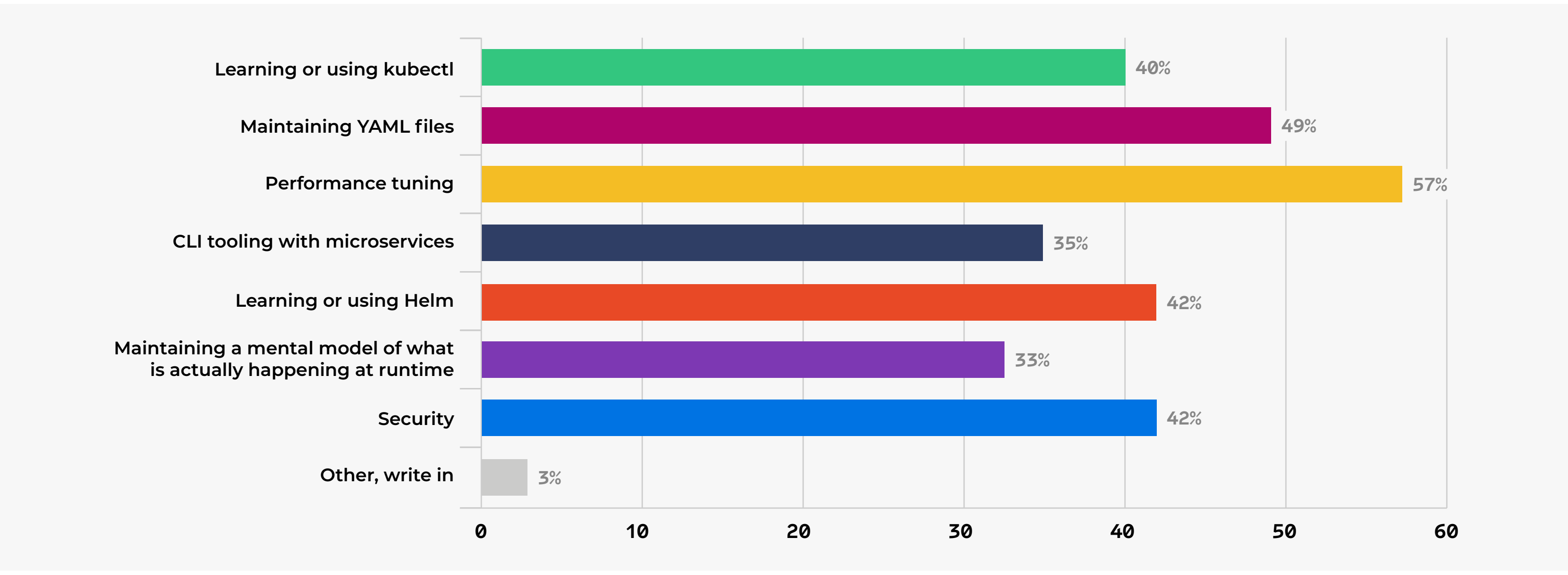
Results:

**Figure 6.** What Kubernetes has improved vs. worsened [*n*=89]



SEE FIGURE 7 ON NEXT PAGE

**Figure 7.** Kubernetes pain points [*n*=88]



## OBSERVATIONS

More than half of respondents reported that Kubernetes improved "Autoscaling," "Deployment in general," "CI/CD," and "Building microservices," and nearly half claimed that "Reliability" was improved. Furthermore, more than three-quarters of respondents (78%) selected three or more of the 10 improvement options provided, and more than half of respondents (54%) selected five or more.

No option received more respondents saying that it was worsened by Kubernetes than respondents saying it was improved, but "Cost" came closest, with only a 5% difference in response rates between the two. "Architectural refactoring" and "Security" had fewer respondents saying that these were improved by Kubernetes than respondents reporting "Cost," but they also had fewer respondents saying those aspects were worsened. Fewer than half of respondents (42%) selected more than one factor that was worsened by Kubernetes adoption at their organization, and fewer than one in six respondents (16%) selected three or more.

"Autoscaling," "CI/CD," and "Application modularity" all saw significantly fewer respondents indicating that these factors were improved by Kubernetes adoption than our 2024 cloud native research findings, and "Overall system design" saw significantly more. Details on the time-based data analysis can be found in Table 5:

**Table 5.** What Kubernetes has improved vs. worsened: *Cloud Native* vs. *Kubernetes* Trend Reports (*CN* vs. *K8s*)

| | Improved | | | Worsened | | | Both | | | Neither | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Factor | CN | K8s | % Change | CN | K8s | % Change | CN | K8s | % Change | CN | K8s | % Change |
| Autoscaling | 73% | 61% | -13% | 3% | 6% | +3% | 4% | 9% | +5% | 20% | 25% | +5% |
| Deployment in general | 56% | 58% | +2% | 6% | 6% | 0% | 15% | 13% | -2% | 23% | 22% | 0% |
| CI/CD | 61% | 53% | -8% | 1% | 2% | +1% | 10% | 7% | -3% | 28% | 38% | +10% |
| Building microservices | 51% | 52% | +1% | 3% | 2% | -1% | 8% | 7% | -2% | 38% | 39% | +1% |
| Reliability | 51% | 48% | -2% | 4% | 9% | +5% | 7% | 1% | -6% | 38% | 42% | +4% |
| Overall system design | 27% | 36% | +9% | 17% | 12% | -5% | 1% | 1% | 0% | 55% | 51% | -4% |
| App modularity | 46% | 35% | -12% | 4% | 9% | +5% | 3% | 1% | -2% | 46% | 55% | +9% |
| Cost | 24% | 30% | +6% | 31% | 25% | -6% | 8% | 3% | -5% | 37% | 42% | +5% |
| Architectural refactoring | 31% | 25% | -6% | 20% | 15% | -5% | 3% | 9% | +6% | 46% | 52% | +5% |
| Security | 27% | 22% | -4% | 17% | 13% | -3% | 7% | 6% | -1% | 49% | 58% | +9% |
| Other, write in | 0% | 0% | 0% | 11% | 9% | -2% | 1% | 0% | -1% | 87% | 91% | +4% |

"Performance tuning" and "Maintaining YAML files" were the most commonly selected Kubernetes pain points, though all listed pain points were selected by at least one-third of respondents. The response rate for "CLI tooling with microservices" increased significantly since the 2024 *Cloud Native* Trend Report, while "Maintaining a mental model of what is actually happening at runtime" saw a significant decrease (details in Table 6).

Overall, most pain points that developers encounter while working with Kubernetes are not diminishing quickly, though these pain points may be alleviated as more organizations begin to utilize third-party cloud platform tools that manage more of the "difficult" aspects of Kubernetes.

**Table 6.** Kubernetes pain points: *CN* vs. *K8s* Trend Reports

| Pain Point | Cloud Native | | Kubernetes | | % Change |
|---|---|---|---|---|---|
| | % | *n*= | % | *n*= | |
| Performance tuning | 53% | 73 | 57% | 50 | +4% |
| Maintaining YAML files | 52% | 72 | 49% | 43 | -3% |
| Learning or using Helm | 40% | 56 | 42% | 37 | +2% |
| Security | 47% | 66 | 42% | 37 | -5% |
| Learning or using kubectl | 39% | 54 | 40% | 35 | +1% |
| CLI tooling with microservices | 26% | 36 | 35% | 31 | +9% |
| Maintaining a mental model of what is actually happening at runtime | 41% | 57 | 33% | 29 | -8% |
| Other, write in | 4% | 6 | 3% | 3 | -1% |

## Monitoring and Observability

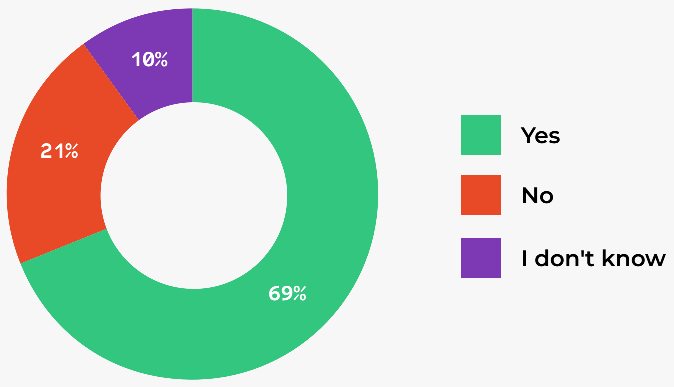
Monitoring and observability are critical for cloud-native and containerized applications, especially those running on Kubernetes, because they provide real-time insights into application performance, health, and resource utilization. With the dynamic and distributed nature of containerized environments, it's essential to track metrics, logs, and traces to detect issues, prevent downtime, and ensure scalability. While Kubernetes automatically manages and orchestrates multiple containers, without proper observability, identifying the root cause of performance bottlenecks, failures, or misconfigurations can be challenging.

Effective monitoring helps optimize resource use, improve reliability, and maintain seamless operations in complex, microservices-based architectures. To find out more about how developers and organizations are implementing monitoring and observability in their containerized applications, and the impact of Kubernetes use on monitoring decisions, we asked the following:

> *Do you use any tools to monitor Kubernetes?*
>
> *Which of the following tools does your organization use for monitoring and/or observability of cloud-native or containerized applications?*
>
> *In what ways has your organization adopted AI for monitoring and/or observability?*

Results:

**Figure 8.** Respondents using tools to monitor Kubernetes [*n*=86]



*SEE FIGURES 9-10 ON NEXT PAGE*

**Figure 9.** Monitoring/observability tools for cloud-native and containerized apps [*n*=114]

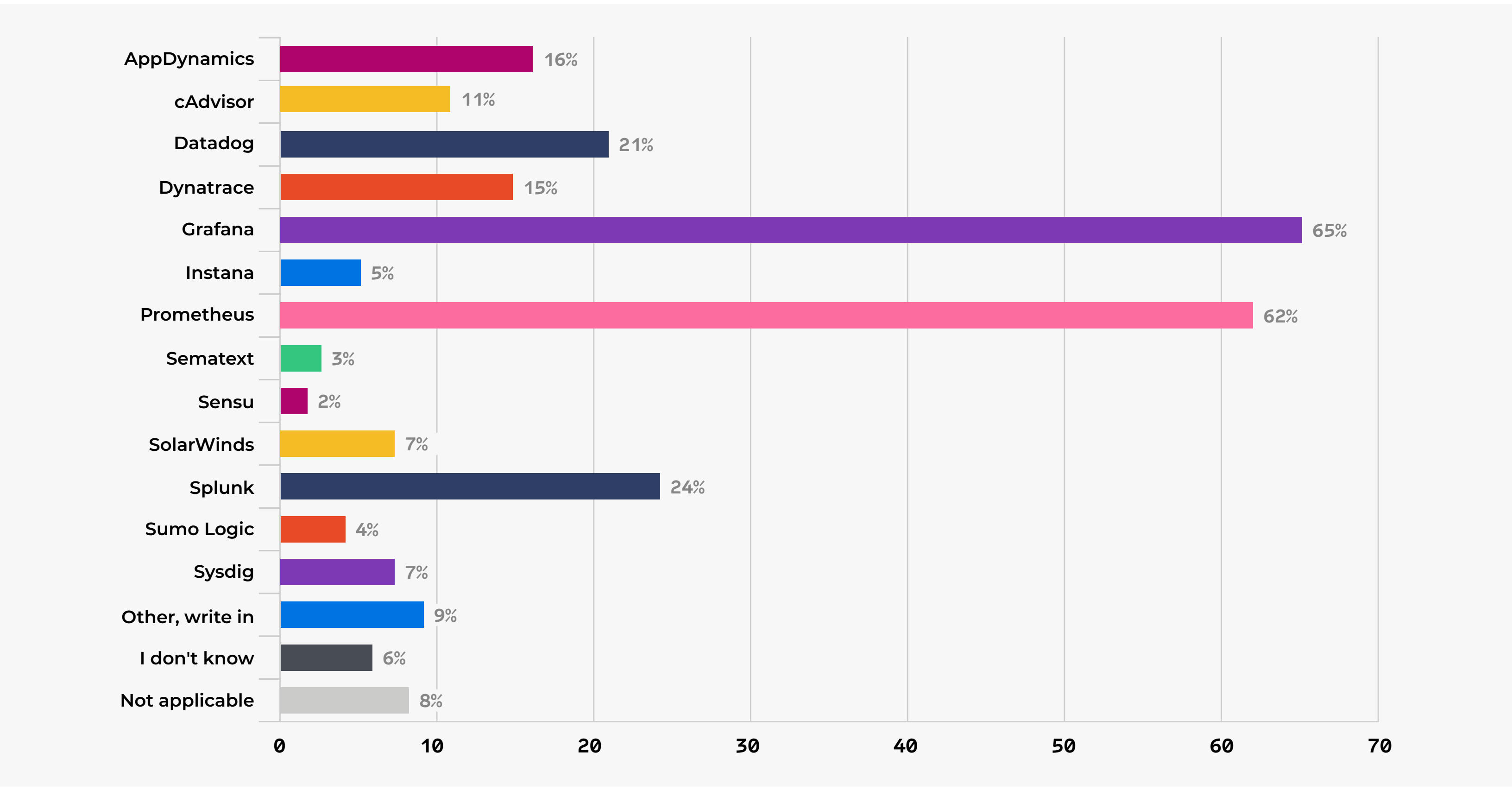
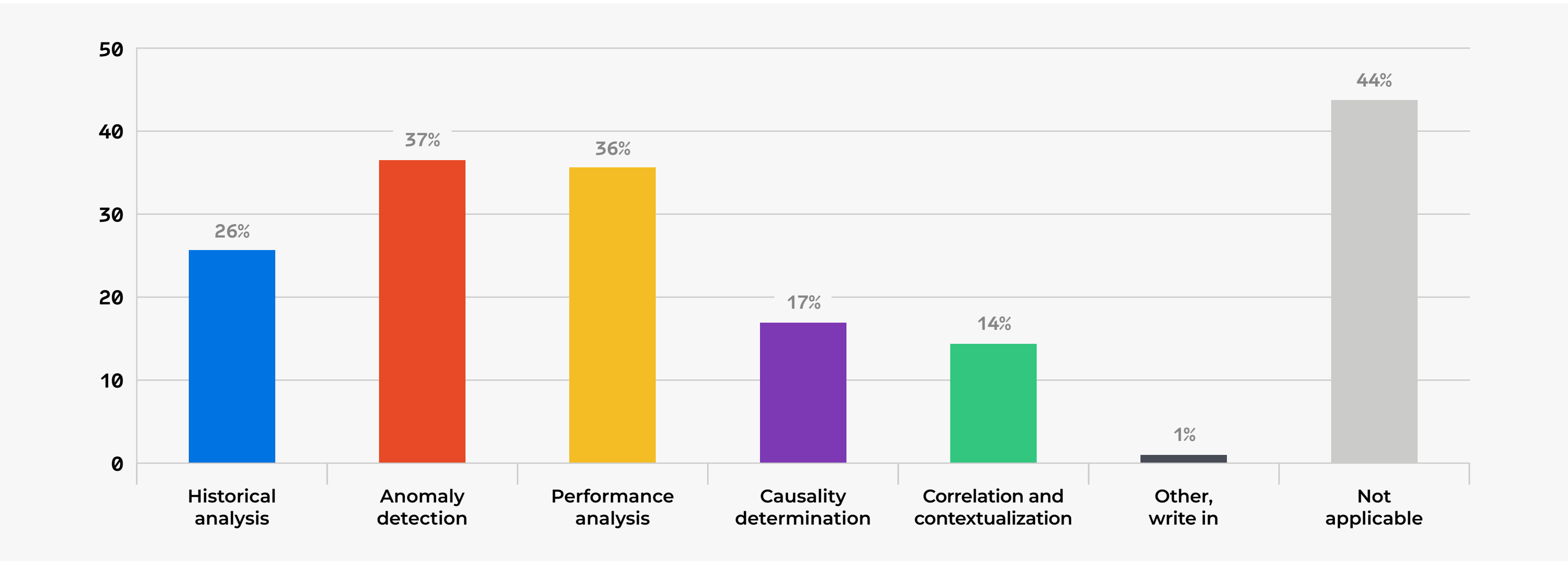


**Figure 10.** Adoption of AI for monitoring/observability [*n*=117]



## OBSERVATIONS

- More than two-thirds of respondents said they use tools for monitoring Kubernetes. When respondents were asked to write in the tools they used for monitoring Kubernetes, 40% said they use "Prometheus," 30% said they use "Grafana," and 10% said they use an Azure monitoring tool.

- "Grafana" and "Prometheus" were also the most frequently selected tools for monitoring and observability of cloud-native or containerized apps by far. "Splunk," "Datadog," "AppDynamics," "Dynatrace," and "cAdvisor" were all selected by more than 10% of respondents. Additionally, respondents at organizations using Kubernetes were significantly more likely to use several of the listed tools, including "Grafana," "Prometheus," "Datadog," and

"AppDynamics" than those at organizations not using Kubernetes (details in Table 7).

The majority of respondents (56%) 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." Respondents at organizations using Kubernetes were significantly more likely than those at organizations not using Kubernetes to use AI for "Anomaly detection," "Historical analysis," and "Causality determination" (details in Table 8).

**Table 8.** AI uses for monitoring/observability by organization Kubernetes use*

| Use Case | Kubernetes Use | | Overall |
|---|---|---|---|
| | Yes | No | |
| Anomaly detection | 41% | 19% | 37% |
| Performance analysis | 36% | 33% | 36% |
| Historical analysis | 28% | 19% | 26% |
| Causality determination | 21% | 5% | 17% |
| Correlation and contextualization | 15% | 10% | 14% |
| Other, write-in | 1% | 0% | 1% |
| Not applicable | 42% | 52% | 44% |
| *n=* | 86 | 21 | 114 |

**% of columns*

**Table 9.** AI uses for monitoring/observability by organization size*

| Use Case | Organization Size | | | Overall |
|---|---|---|---|---|
| | 1-99 | 100-999 | 1,000+ | |
| Historical analysis | 17% | 29% | 31% | 26% |
| Anomaly detection | 23% | 32% | 48% | 37% |
| Performance analysis | 30% | 25% | 46% | 36% |
| Causality determination | 3% | 18% | 25% | 17% |
| Correlation and contextualization | 7% | 14% | 19% | 15% |
| Other, write-in | 0% | 0% | 2% | 1% |
| Not applicable | 57% | 57% | 27% | 43% |
| *n=* | 30 | 28 | 52 | 110 |

**% of columns*

**Table 7.** Monitoring/observability tools for cloud-native and containerized apps by Kubernetes use*

| Tool | Kubernetes Use | | Overall |
|---|---|---|---|
| | Yes | No | |
| Grafana | 76% | 26% | 65% |
| Prometheus | 72% | 22% | 62% |
| Splunk | 25% | 22% | 24% |
| Datadog | 26% | 0% | 21% |
| AppDynamics | 18% | 9% | 16% |
| Dynatrace | 17% | 4% | 15% |
| cAdvisor | 14% | 4% | 11% |
| Other, write in | 6% | 17% | 9% |
| Not applicable | 2% | 26% | 8% |
| SolarWinds | 8% | 4% | 7% |
| Sysdig | 9% | 0% | 7% |
| I don't know | 2% | 13% | 6% |
| *n=* | 87 | 23 | 117 |

**% of columns; only displays options selected by > 5% of respondents*

Respondents at large organizations were significantly more likely than those at small or mid-sized organizations to indicate that their companies use AI for monitoring and observability, especially for "Anomaly detection" and "Performance analysis." Respondents at small organizations were significantly less likely than others to say that their organization uses AI for "Historical analysis," "Causality determination," and "Correlation and contextualization" (details in Table 9).

## Research Target Two: Kubernetes Impact on Other Development Trends

Software development involves countless interconnected systems, tools, and practices, and adding Kubernetes' technology stack has the potential to impact — and be impacted by — these systems, tools, and practices. We wanted to learn how organizations are using Kubernetes alongside other common trends and practices in the contemporary development landscape; in this section, we explore the following topics in relation to Kubernetes use:

- Cost optimization techniques
- Security
- Microservices

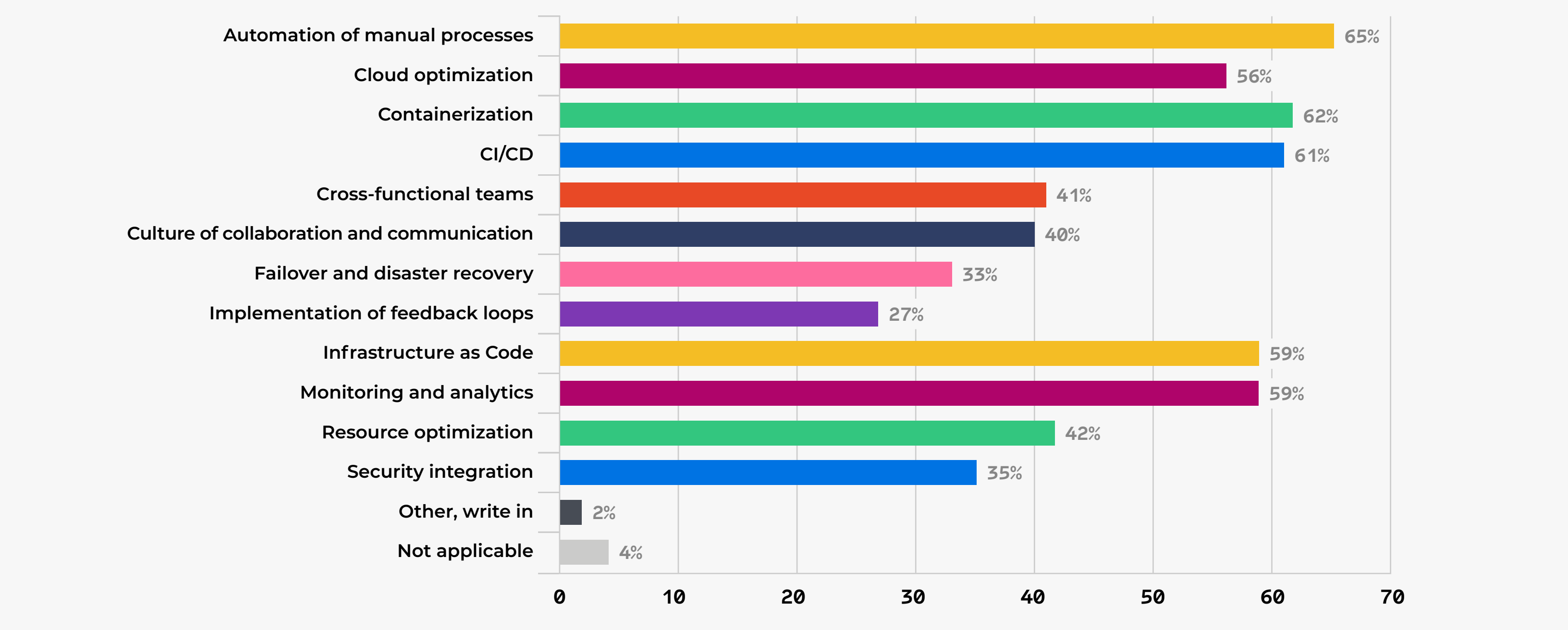
### Cost Optimization Techniques

Cost optimization is essential in software development to ensure efficient use of resources and to maximize return on investment, especially in cloud environments where expenses can scale quickly. Kubernetes can play a pivotal role in cost optimization by automating resource allocation, scaling applications based on demand, and efficiently distributing workloads across nodes. Its ability to dynamically scale resources up or down helps reduce overprovisioning, while features like auto-scaling and efficient bin-packing optimize infrastructure utilization. By managing resources more effectively, Kubernetes helps organizations control costs while maintaining performance and scalability. We wanted to know the types of cost optimization techniques organizations are using and how Kubernetes use relates to those techniques, so we asked:

*Which of the following practices and techniques does your organization use to manage and optimize costs?*

Results:

**Figure 11.** Techniques for managing and optimizing costs [*n*=117]



## OBSERVATIONS

- "Automation of manual processes," "Containerization," and "Continuous integration and continuous delivery (CI/CD)" were the most commonly selected cost optimization techniques. 92% of respondents selected at least two of the 12 listed techniques, and more than half of respondents (54%) selected five or more techniques. Since the 2024 *Cloud Native* Trend Report, the only significant changes in response rates were a decrease in the response rate for "Monitoring and analytics" (down 7% from 67%) and an increase in the response rate for "Cross-functional teams" (up 9% from 34%).

- Respondents at organizations running Kubernetes clusters were generally more likely to select each cost management technique. "Cloud optimization," "Infrastructure as Code (IaC)," "Containerization," and "Resource optimization" were all at least 30% more likely to be selected by respondents at organizations using Kubernetes than those at organizations not using Kubernetes. The only technique chosen more often by respondents at organizations not using Kubernetes was "Culture of collaboration and communication." Further details are available in Table 10.

**Table 10.** Techniques for managing and optimizing costs by organization Kubernetes use*

| Technique | Kubernetes Use | | Overall |
|---|---|---|---|
| | Yes | No | |
| Automation of manual processes | 69% | 52% | 65% |
| Containerization | 70% | 35% | 62% |
| CI/CD | 63% | 57% | 61% |
| Infrastructure as Code | 67% | 30% | 59% |
| Monitoring and analytics | 66% | 43% | 59% |
| Cloud optimization | 64% | 26% | 56% |
| Resource optimization | 48% | 17% | 42% |
| Cross-functional teams | 43% | 39% | 41% |
| Culture of collaboration and communication | 40% | 48% | 40% |
| Security integration | 38% | 26% | 35% |
| Failover and disaster recovery | 34% | 26% | 33% |
| Implementation of feedback loops | 31% | 17% | 27% |
| Not applicable | 2% | 4% | 4% |

**% of columns*

## Security

Kubernetes security is crucial as it protects containerized applications and infrastructure from vulnerabilities, unauthorized access, and potential attacks in a highly dynamic and interconnected environment. With Kubernetes managing critical workloads and sensitive data, securing the platform involves hardening cluster configurations, managing access controls, and ensuring secure communication between microservices. Proper security practices, such as implementing network policies, securing container images, and regularly updating Kubernetes components, help prevent breaches and data leaks. To determine how organizations are handling their Kubernetes security, we asked the following:

> *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 container and/or Kubernetes deployments to production due to security concerns?*

Results:

**Figure 12.** Kubernetes security measures [*n*=86]

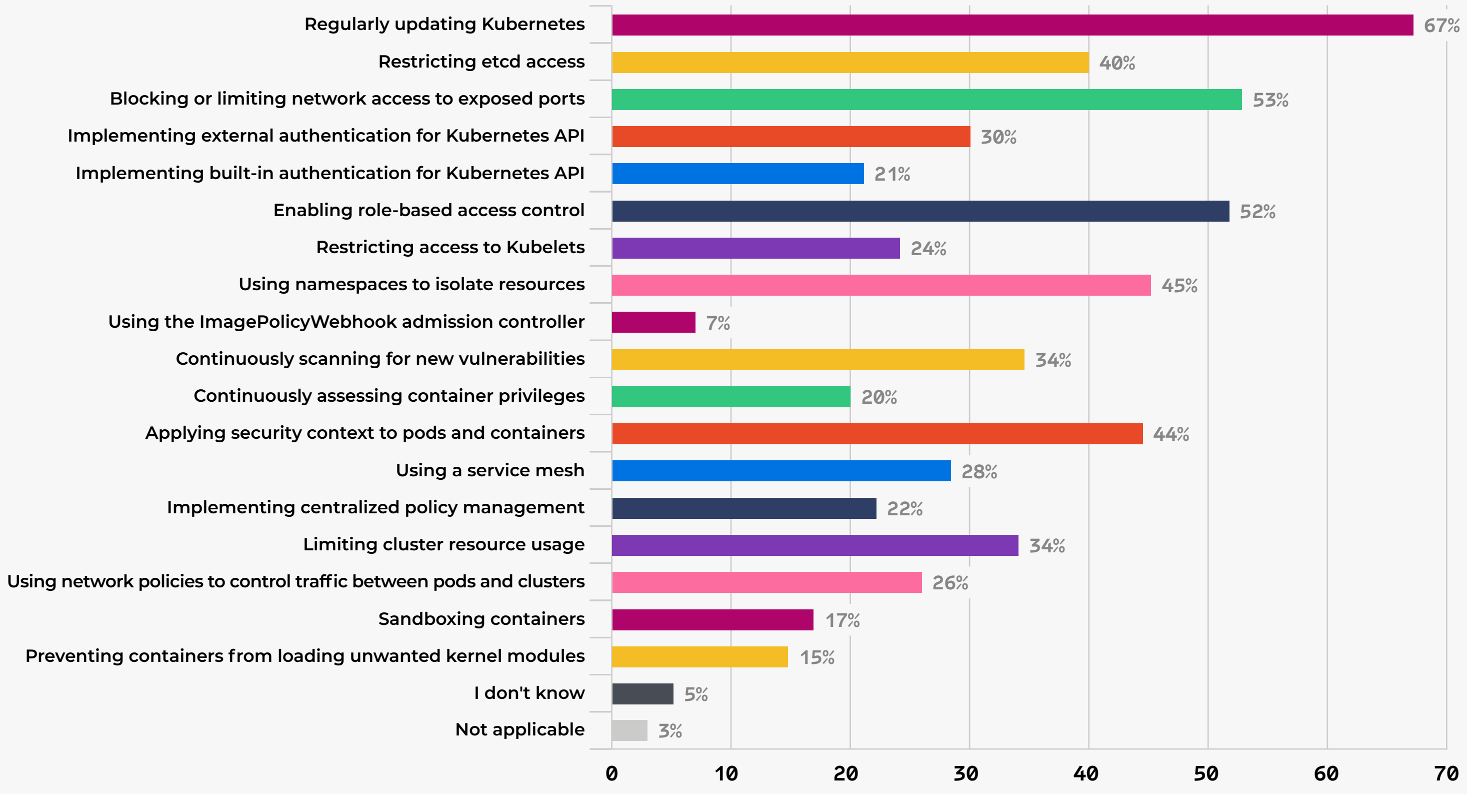
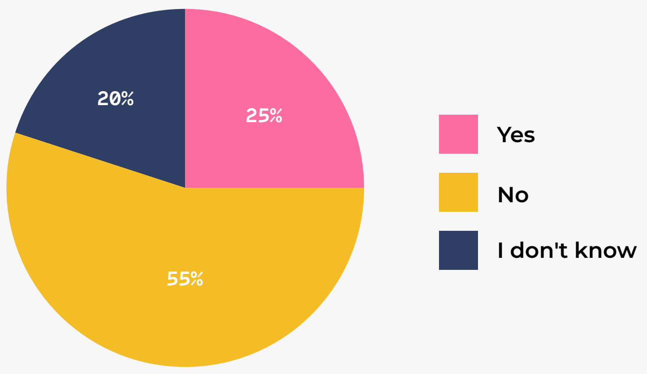


**Figure 13.** Kubernetes prod deployment reassessments due to security concerns [*n*=101]



## OBSERVATIONS

- More than half of respondents (62%) selected five or more security measures out of the 18 we provided, and 87% selected two or more. The most commonly selected measures were "Regularly updating Kubernetes," "Blocking or limiting network access to exposed ports," and "Enabling role-based access control (RBAC)." The only significant changes from the 2024 *Cloud Native* Trend Report were decreases in response rates for "Continuously scanning for new vulnerabilities" (down 9% from 44%), "Using network policies for controlling traffic between pods and clusters" (down 7% from 32%), and "Continuously assessing container privileges" (down 13% from 32%).

- 25% of respondents reported that their organization has needed to reassess a container/Kubernetes deployment because of security concerns in the last year, down significantly from 32% in the 2024 *Cloud Native* report. Respondents at small organizations were considerably more likely to indicate that their organization had not needed to perform such a reassessment in the last year than those at mid-sized organizations, who, in turn, were more likely than those at large organizations to say that a Kubernetes security-related deployment reassessment was not needed (details in Table 11).

**Table 11.** Kubernetes prod deployment reassessments due to security concerns by organization size*

| | Kubernetes Reassessment | | | |
|---|---|---|---|---|
| **Org Size** | **Yes** | **No** | **I don't know** | ***n*=** |
| 1-99 | 29% | 71% | 0% | 21 |
| 100-999 | 30% | 56% | 15% | 27 |
| 1,000+ | 22% | 46% | 32% | 50 |
| Overall | 26% | 54% | 20% | 98 |

**% of rows*

## Microservices

Running microservices on Kubernetes can offer significant advantages, including improved scalability, reliability, and simplified management of complex applications. Kubernetes automates the deployment, scaling, and monitoring of microservices, ensuring each service runs independently and can be updated, scaled, or restarted without affecting others. Its built-in service discovery, load balancing, and self-healing capabilities enhance the resilience and performance of microservices architectures. To discover more about how many organizations are running microservices on Kubernetes, we asked:

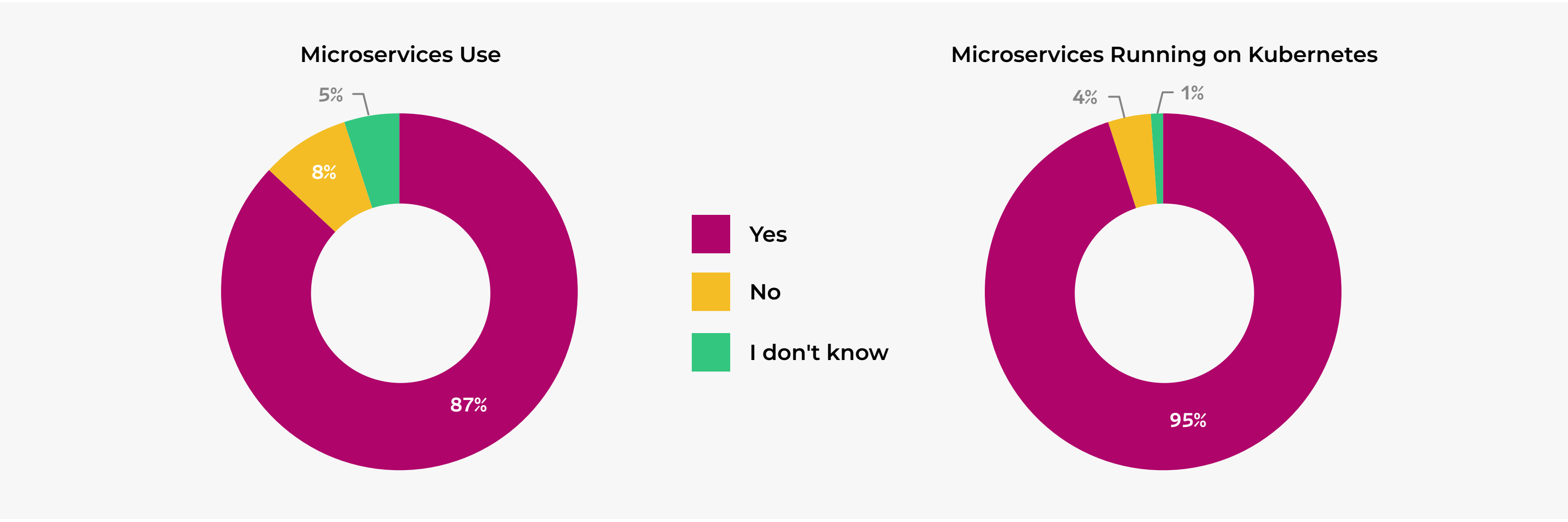
*Does your organization run any microservices?*

And to respondents who answered "Yes" to that question, we asked:

*Are any of these microservices deployed on Kubernetes clusters?*

Results:

**Figure 14.** Microservices use and microservices running on Kubernetes [*n*=121; *n*=80]



## OBSERVATIONS

- A large majority of respondents said that their organization uses microservices, and almost all respondents at organizations running microservices indicated that those microservices are deployed on Kubernetes clusters. There were no significant changes to the results of either question since our 2024 *Cloud Native* Trend Report, supporting the hypothesis we wrote in that report's "Key Research Findings" — that microservices adoption has reached a saturation point. As we said in those findings, "Unless a new paradigm emerges to disrupt microservices' popularity in modern software architectures, it seems likely that microservices' usage rate will remain roughly the same for years to come."

Respondents at small organizations were significantly less likely than others to say that their organization uses microservices, but organization size had no impact on whether respondents at organizations running microservices reported that those microservices are deployed on Kubernetes clusters. Details are available in Table 12:

**Table 12.** Microservices use and microservices running on Kubernetes by organization size*

| Organization Size | Microservices Use | | | | Microservices Running on Kubernetes | | | |
|---|---|---|---|---|---|---|---|---|
| | Yes | No | I don't know | *n=* | Yes | No | I don't know | *n=* |
| 1-99 | 81% | 16% | 3% | 31 | 93% | 0% | 7% | 15 |
| 100-999 | 93% | 7% | 0% | 28 | 95% | 0% | 5% | 20 |
| 1,000+ | 91% | 4% | 5% | 55 | 95% | 2% | 2% | 44 |
| Overall | 87% | 8% | 5% | 121 | 95% | 1% | 4% | 80 |

**% of rows*

## Future Research

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

- Kubernetes workloads
- Serverless tools
- Kubernetes autoscalers
- Technical debt and legacy code
- AI for release management
- Types of cloud infrastructure

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. 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 watching Make Some Noise on Dropout with a glass of red wine or a cold beer.

PARTNER CASE STUDY

## Porter

Leading Logistics Innovator Optimizes Containers and Costs

CREATED IN PARTNERSHIP WITH

Spot by NetApp

**Porter**
Logistics
2,600 employees
Bengaluru, India

**Solutions Used**
Amazon ECS,
Spot Ocean

**Primary Outcomes**
Porter was able to move 100% of its production ECS workloads to AWS Spot Instances, saving up to 20% on their cloud costs.

*"We never thought we'd move 100% of production to spot instances.... With Ocean, we saw that even if interruptions did happen, our uptime was intact. This qualified for migrating our production clusters as well."*

—**Jijo T. Joy**,
*Senior DevOps Engineer, Porter*

*Read the full case study here.*

Founded in 2014, Porter is a leading end-to-end logistics company providing a spectrum of intracity and intercity delivery services in India. Porter has serviced over 8 million customers across more than 20 cities in India. Porter has disrupted various domains of logistics by launching an on-demand marketplace and an online delivery app, which is built within a cloud-native framework on Amazon Web Services (AWS) ECS.

### Challenge

Porter faced a dual challenge: a very lean DevOps team and soaring cloud costs. Although the company's six-person cloud infrastructure team engaged in various cost management activities, its use of cloud-provider tools and custom scripts only addressed a portion of their consumption, and 70% of their compute usage consisted of expensive on-demand instances. They also lacked the correct tooling for forecasting usage, which led to wasteful overprovisioning to ensure optimal performance.

Alongside cost issues, the team was juggling multiple responsibilities maintaining all production systems and internal platforms, which prevented them from delivering desired platform improvements. Due to constraints in personnel, time, and resources, achieving innovation was challenging.

### Solution

Porter saw an opportunity to streamline their operations and reduce costs using Spot by NetApp's Ocean, a comprehensive solution for scale-up enterprises seeking to automate and optimize Kubernetes, ECS, and EKS infrastructures. Spot Ocean's automated, application-driven provisioning and scaling of compute resources, coupled with Virtual Node Groups that allow multiple node types to be managed on a single cluster, helped the Porter team eliminate the complexity of their infrastructure and drastically reduce manual maintenance.

The team was also able to improve efficiency by reducing infrastructure waste using Ocean's automated infrastructure optimization capabilities, such as bin-packing, scaledown, and rightsizing. And crucially, Spot Ocean empowered the team to run 100% of production workloads on low-cost Spot Instances, confident that its ML-driven automation would prevent downtime during market interruptions.

### Results

With continuous optimization and intelligent automation of their ECS infrastructure provided by Spot Ocean, Porter was able to:

- **Reduce cloud costs by up to 20%** with automated infrastructure optimization, including bin-packing and application-driven scaling
- Run 100% of its production environment on EC2 Spot Instances without risk of interruption
- Move from reactive infrastructure management to proactive cost and performance optimization

Spot
by NetApp

# Effortlessly deliver cost-optimized Kubernetes infrastructure at scale

Eliminate complexity, maximize efficiency and reduce costs with automated, continuous optimization for containerized applications across all clouds.

aws

Spot Ocean

## Maximize cost savings

Automatically provision the optimal mix of instance types and pricing options to precisely meet application needs

## See and understand usage

Get real-time detailed insights into containers costs, enabling data-driven decisions and better management of budget

## Automate container infrastructure management

Application-aware, AI/ML-driven controller continuously manages the scaling and sizing of compute resources

Get started today at spot.io/ocean

CONTRIBUTOR INSIGHTS

# A Decade of Excellence

The Journey, Impact, and Future of Kubernetes

**By Maryam Tavakkoli, Senior Cloud Engineer at Relex Solutions**

A decade ago, Google introduced Kubernetes to simplify the management of containerized applications. Since then, it has fundamentally transformed the software development and operations landscape. Today, Kubernetes has seen numerous enhancements and integrations, becoming the de facto standard for container orchestration. This article explores the journey of Kubernetes over the past 10 years, its impact on the software development lifecycle (SDLC) and developers, and the trends and innovations that will shape its next decade.

## The Evolution of Kubernetes

Kubernetes, often referred to as K8s, had its first commit pushed to GitHub on June 6, 2014. About a year later, on July 21, 2015, Kubernetes V1 was released, featuring 14,000 commits from 400 contributors. Simultaneously, the Linux Foundation announced the formation of the Cloud Native Computing Foundation (CNCF) to advance state-of-the-art technologies for building cloud-native applications and services. After that, Google donated Kubernetes to the CNCF, marking a significant milestone in its development.

Kubernetes addressed a critical need in the software industry: managing the lifecycle of containerized applications. Before Kubernetes, developers struggled with orchestrating containers, leading to inefficiencies and complexities in deployment processes. Kubernetes brought advanced container management functionality and quickly gained popularity due to its robust capabilities in automating the deployment, scaling, and operations of containers.

While early versions of Kubernetes introduced the foundation for container orchestration, the project has since undergone significant improvements. Major updates have introduced sophisticated features such as StatefulSets for managing stateful applications, advanced networking capabilities, and enhanced security measures. The introduction of Custom Resource Definitions (CRDs) and Operators has further extended its functionality, allowing users to manage complex applications and workflows with greater ease.

In addition, the community has grown significantly over the past decade. According to the 2023 *Project Journey Report*, Kubernetes now has over 74,680 contributors, making it the second-largest open-source project in the world after Linux. Over the years, Kubernetes has seen numerous enhancements and integrations, becoming the de facto standard for container orchestration. The active open source community and the extensive ecosystem of tools and projects have made Kubernetes an essential technology for modern software development. It is now the "primary container orchestration tool for 71% of Fortune 100 companies" (*Project Journey Report*).

## Kubernetes' Impact on the SDLC and Developers

Kubernetes abstracts away the complexities of container orchestration and allows developers to focus on development rather than worry about application deployment and orchestration. The benefits and key impacts on the SDLC and developer workflows include enhanced development and testing, efficient deployment, operational efficiency, improved security, and support for microservices architecture.

### Enhanced Development and Testing

Kubernetes ensures consistency for applications running across testing, development, and production environments, regardless of whether the infrastructure is on-premises, cloud based, or a hybrid setup. This level of consistency, along with the capability to quickly spin up and tear down environments, significantly accelerates development cycles. By promoting portability, Kubernetes also helps enterprises avoid vendor lock-in and refine their cloud strategies, leading to a more flexible and efficient development process.

### Efficient Deployment

Kubernetes automates numerous aspects of application deployment, such as service discovery, load balancing, scaling, and self-healing. This automation reduces manual effort, minimizes human error, and ensures reliable and repeatable deployments, reducing downtime and deployment failures.

### Operational Efficiency

Kubernetes efficiently manages resources by dynamically allocating them based on the application's needs. It ensures operations remain cost effective while maintaining optimal performance and use of computing resources by scheduling containers based on resource requirements and availability.

### Security

Kubernetes enhances security by providing container isolation and managing permissions. Its built-in security features allow developers to build secure applications without deep security expertise. Such built-in features include role-based access control, which ensures that only authorized users can access specific resources and perform certain actions. It also supports secrets management to securely store and manage sensitive information like passwords and API keys.

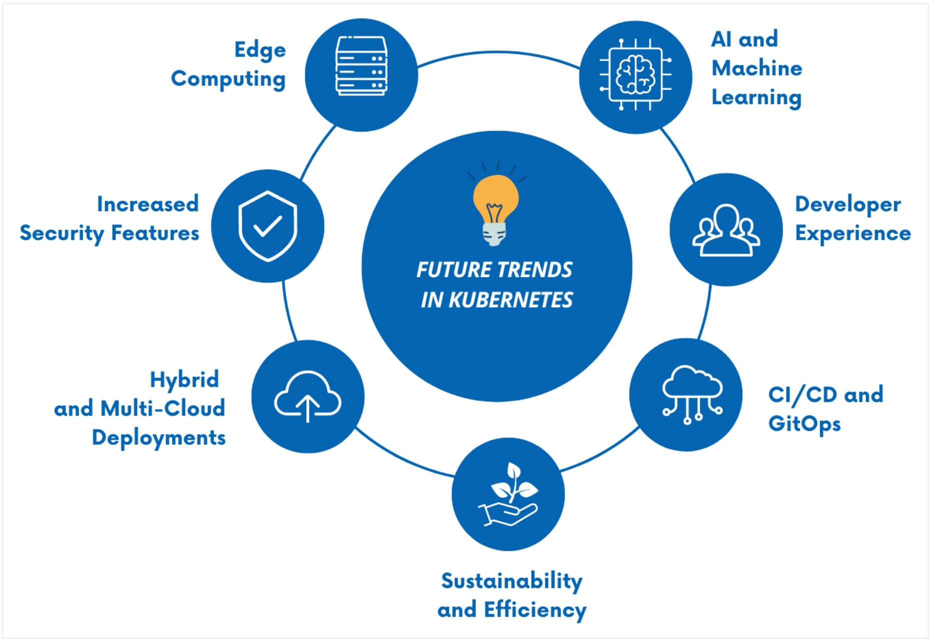
### Microservices Architecture

Kubernetes has facilitated the adoption of microservices architecture by enabling developers to deploy, manage, and scale individual microservices independently. Each microservice can be packaged into a separate container, providing isolation and ensuring that dependencies are managed within the container. Kubernetes' service discovery and load balancing features enable communication between microservices, while its support for automated scaling and self-healing ensures high availability and resilience.

## Predictions for the Next Decade

After a decade, it has become clear that Kubernetes is now the standard technology for container orchestration that's used by many enterprises. According to the *CNCF Annual Survey 2023*, the usage of Kubernetes continues to grow, with significant adoption across different industries and use cases. Its reliability and flexibility make it a preferred choice for mission-critical applications, including databases, CI/CD pipelines, and AI and machine learning (ML) workloads. As a result, there is a growing demand for new features and enhancements, as well as simplifying concepts for users. The community is now prioritizing improvements that not only enhance user experiences but also promote the sustainability of the project. Figure 1 illustrates the anticipated future trends in Kubernetes, and below are the trends and innovations expected to shape Kubernetes' future in more detail.

**Figure 1.** Future trends in Kubernetes


FUTURE TRENDS IN KUBERNETES
Edge Computing
AI and Machine Learning
Increased Security Features
Developer Experience
Hybrid and Multi-Cloud Deployments
CI/CD and GitOps
Sustainability and Efficiency

### AI and Machine Learning

Kubernetes is increasingly used to orchestrate AI and ML workloads, supporting the deployment and management of complex ML pipelines. This simplifies the integration and scaling of AI applications across various environments. Innovations such as Kubeflow — an open-source platform designed to optimize the deployment, orchestration, and management of ML workflows on Kubernetes — enable data scientists to focus more on model development and less on infrastructure concerns.

According to the recent CNCF open-source project velocity report, Kubeflow appeared on the top 30 CNCF project list for the first time in 2023, highlighting its growing importance in the ecosystem. Addressing the resource-intensive demands of AI introduces new challenges that contributors are focusing on, shaping the future of Kubernetes in the realm of AI and ML.

### The Developer Experience

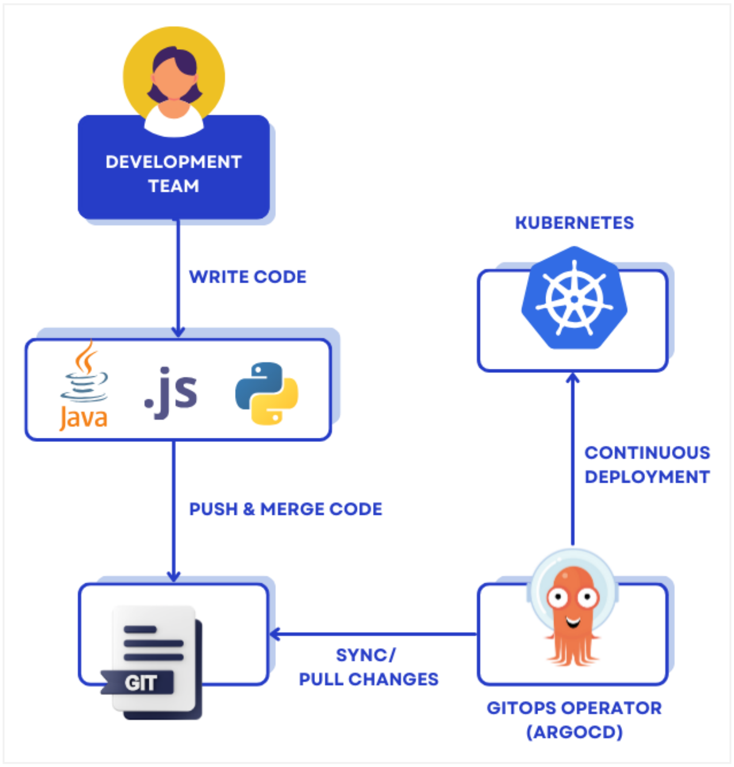
As Kubernetes evolves, its complexity can create challenges for new users. Hence, improving the user experience is crucial moving forward. Tools like Backstage are revolutionizing how developers work with Kubernetes and speeding up the development process. The CNCF's open-source project velocity report also states that "Backstage is addressing a significant pain point around developer experience."

Moreover, the importance of platform engineering is increasingly recognized by companies. This emerging trend is expected to grow, with the goal of reducing the learning curve and making it easier for developers to adopt Kubernetes, thereby accelerating the development process and improving productivity.

### CI/CD and GitOps

Kubernetes is revolutionizing continuous integration and continuous deployment (CI/CD) pipelines through the adoption of GitOps practices. GitOps uses Git repositories as the source of truth for declarative infrastructure and applications, enabling automated deployments. Tools like ArgoCD and Flux are being widely adopted to simplify the deployment process, reduce human error, and ensure consistency across environments. Figure 2 shows the integration between a GitOps operator, such as ArgoCD, and Kubernetes for managing deployments. This trend is expected to grow, making CI/CD pipelines more robust and efficient.

**Figure 2.** Kubernetes GitOps



### Sustainability and Efficiency

Cloud computing's carbon footprint now exceeds the airline industry, making sustainability and operational efficiency crucial in Kubernetes deployments. The Kubernetes community is actively developing features to optimize resource usage, reduce energy consumption, and enhance the overall efficiency of Kubernetes clusters.

CNCF projects like KEDA (Kubernetes event-driven autoscaling) and Karpenter (just-in-time nodes for any Kubernetes cluster) are at the forefront of this effort. These tools not only contribute to cost savings but also align with global sustainability goals.

## Hybrid and Multi-Cloud Deployments

According to the *CNCF Annual Survey 2023*, multi-cloud solutions are now the norm: Multi-cloud solutions (hybrid and other cloud combinations) are used by 56% of organizations. Deploying applications across hybrid and multi-cloud environments is one of Kubernetes' most significant advantages. This flexibility enables organizations to avoid vendor lock-in, optimize costs, and enhance resilience by distributing workloads across multiple cloud providers.

Future developments in Kubernetes will focus on improving and simplifying management across different cloud platforms, making hybrid and multi-cloud deployments even more efficient.

## Increased Security Features

Security continues to be a top priority for Kubernetes deployments. The community is actively enhancing security features to address vulnerabilities and emerging threats. These efforts include improvements to network policies, stronger identity and access management (IAM), and more advanced encryption mechanisms. For instance, the 2024 CNCF open-source project velocity report highlighted that Keycloak, which joined CNCF last year as an incubating project, is playing a vital role in advancing open-source IAM, backed by a large and active community.

## Edge Computing

Kubernetes is playing a crucial role in the evolution of edge computing. By enabling consistent deployment, monitoring, and management of applications at the edge, Kubernetes significantly reduces latency, enhances real-time processing capabilities, and supports emerging use cases like IoT and 5G. Projects like KubeEdge and K3s are at the forefront of this movement. We can expect further optimizations for lightweight and resource-constrained environments, making Kubernetes even more suitable for edge computing scenarios.

## Conclusion

Kubernetes has revolutionized cloud-native computing, transforming how we develop, deploy, and manage applications. As Kelsey Hightower noted in Google's Kubernetes Podcast, "We are only halfway through its journey, with the next decade expected to see Kubernetes mature to the point where it 'gets out of the way' by doing its job so well that it becomes naturally integrated into the background of our infrastructure." Kubernetes' influence will only grow, shaping the future of technology and empowering organizations to innovate and thrive in an increasingly complex landscape.

References:

- "10 Years of Kubernetes" by Bob Killen et al, 2024
- *CNCF Annual Survey 2023* by CNCF, 2023
- "As we reach mid-year 2024, a look at CNCF, Linux Foundation, and top 30 open source project velocity" by Chris Aniszczyk, CNCF, 2024
- "Orchestration Celebration: 10 Years of Kubernetes" by Adrian Bridgwater, 2024
- "Kubernetes: Beyond Container Orchestration" by Pratik Prakash, 2022
- "The Staggering Ecological Impacts of Computation and the Cloud" by Steven Gonzalez Monserrate, 2022

**Maryam Tavakkoli**

@maryamtavakkoli

@maryam-tavakoli

@maryam.tavakoli.3

I am a senior cloud engineer at RELEX Solutions, specializing in designing and implementing advanced cloud and Kubernetes infrastructure. I am also a CNCF Ambassador and Microsoft MVP. With a strong passion for open-source cloud-native solutions, I actively participate in collaborative projects and share innovative ideas within the community.

CONTRIBUTOR INSIGHTS

# Building a CI/CD Pipeline With Kubernetes

A Development Guide With Deployment Considerations for Practitioners

**By Naga Santhosh Reddy Vootukuri, Senior Software Engineering Manager at Microsoft**

In the past, before CI/CD and Kubernetes came along, deploying software to Kubernetes was a real headache. Developers would build stuff on their own machines, then package it and pass it to the operations team to deploy it on production. This approach would frequently lead to delays, miscommunications, and inconsistencies between environments. Operations teams had to set up the deployments themselves, which increased the risk of human errors and configuration issues. When things went wrong, rollbacks were time consuming and disruptive. Also, without automated feedback and central monitoring, it was tough to keep an eye on how builds and deployments were progressing or to identify production issues.

With the advent of CI/CD pipelines combined with Kubernetes, deploying software is smoother. Developers can simply push their code, which triggers builds, tests, and deployments. This enables organizations to ship new features and updates more frequently and reduce the risk of errors in production.

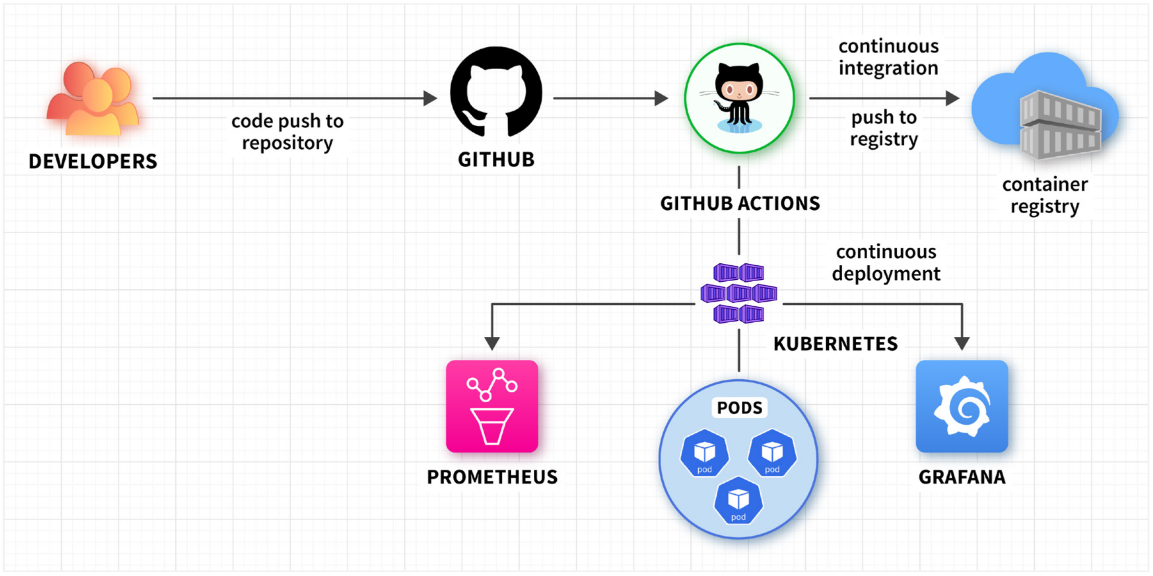
This article explains the CI/CD transformation with Kubernetes and provides a step-by-step guide to building a pipeline.

## Why CI/CD Should Be Combined With Kubernetes

CI/CD paired with Kubernetes is a powerful combination that makes the whole software development process smoother. Kubernetes, also known as K8s, is an open-source system for automating the deployment, scaling, and management of containerized applications. CI/CD pipelines, on the other hand, automate how we build, test, and roll out software. When you put them together, you can deploy more often and faster, boost software quality with automatic tests and checks, cut down on the chance of pushing out buggy code, and get more done by automating tasks that used to be done by hand.

CI/CD with Kubernetes helps developers and operations teams work better together by giving them a shared space to do their jobs. This teamwork lets companies deliver high-quality applications rapidly and reliably, gaining an edge in today's fast-paced world. Figure 1 lays out the various steps:

**Figure 1.** Push-based CI/CD pipeline with Kubernetes and monitoring tools



There are several benefits in using CI/CD with Kubernetes, including:

- **Faster and more frequent application deployments**, which help in rolling out new features or critical bug fixes to the users
- **Improved quality** by automating testing and incorporating quality checks, which helps in reducing the number of bugs in your applications
- **Reduced risk** of deploying broken code to production since CI/CD pipelines can conduct automated tests and roll-back deployments if any problems exist
- **Increased productivity** by automating manual tasks, which can free developers' time to focus on important projects
- **Improved collaboration** between development and operations teams since CI/CD pipelines provide a shared platform for both teams to work

## Tech Stack Options

There are different options available if you are considering building a CI/CD pipeline with Kubernetes. Some of the popular ones include:

- Open-source tools such as Jenkins, Argo CD, Tekton, Spinnaker, or GitHub Actions
- Enterprise tools, including but not limited to, Azure DevOps, GitLab CI/CD, or AWS CodePipeline

Deciding whether to choose an open-source or enterprise platform to build efficient and reliable CI/CD pipelines with Kubernetes will depend on your project requirements, team capabilities, and budget.

## Impact of Platform Engineering on CI/CD With Kubernetes

Platform engineering builds and maintains the underlying infrastructure and tools (the "platform") that development teams use to create and deploy applications. When it comes to CI/CD with Kubernetes, platform engineering has a big impact on making the development process better. It does so by hiding the complex parts of the underlying infrastructure and giving developers self-service options.

Platform engineers manage and curate tools and technologies that work well with Kubernetes to create a smooth development workflow. They create and maintain CI/CD templates that developers can reuse, allowing them to set up pipelines without thinking about the details of the infrastructure. They also set up rules and best practices for containerization, deployment strategies, and security measures, which help maintain consistency and reliability across different applications.

What's more, platform engineers provide ways to observe and monitor applications running in Kubernetes, which let developers find and fix problems and make improvements based on data.

By building a strong platform, platform engineering helps dev teams zero in on creating and rolling out features more without getting bogged down by the complexities of the underlying tech. It brings together developers, operations, and security teams, which leads to better teamwork and faster progress in how things are built.

## How to Build a CI/CD Pipeline With Kubernetes

Regardless of the tech stack you select, you will often find similar workflow patterns and steps. In this section, I will focus on building a CI/CD pipeline with Kubernetes using GitHub Actions.

**Step 1: Setup and prerequisites**

- GitHub account – needed to host your code and manage the CI/CD pipeline using GitHub Actions
- Kubernetes cluster – create one locally (e.g., MiniKube) or use a managed service from Amazon or Azure
- kubectl – Kubernetes command line tool to connect to your cluster
- Container registry – needed for storing Docker images; you can either use a cloud provider's registry (e.g., Amazon ECR, Azure Container Registry, Google Artifact Registry) or set up your own private registry
- Node.js and npm – install Node.js and npm to run the sample Node.js web application
- Visual Studio/Visual Studio Code – IDE platform for making code changes and submitting them to a GitHub repository

**Step 2: Create a Node.js web application**

Using Visual Studio, create a simple Node.js application with a default template. If you look inside, the `server.js` in-built generated file will look like this:

```
// server.js
'use strict';
var http = require('http');
var port = process.env.PORT || 1337;

http.createServer(function (req, res) {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello from kubernetes\n');
}).listen(port);
```

**Step 3: Create a `package.json` file to manage dependencies**

Inside the project, add a new file `Package.json` to manage dependencies:

```
// Package.Json
{
  "name": "nodejs-web-app1",
  "version": "0.0.0",
  "description": "NodejsWebApp",
  "main": "server.js",
  "author": {
    "name": "Sunny"
  },
  "scripts": {
    "start": "node server.js",
    "test": "echo \"Running tests...\" && exit 0"
  },
  "devDependencies": {
    "eslint": "^8.21.0"
  },
  "eslintConfig": {
  }
}
```

**Step 4: Build a container image**

Create a Dockerfile to define how to build your application's Docker image:

```
// Dockerfile
# Use the official Node.js image from the Docker Hub
FROM node:14

# Create and change to the app directory
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code
COPY . .
```

*CODE CONTINUES ON NEXT PAGE*

```
# Expose the port the app runs on
EXPOSE 3000

# Command to run the application
CMD ["node", "app.js"]
```

**Step 5: Create a Kubernetes Deployment manifest**

Create a `deployment.yaml` file to define how your application will be deployed in Kubernetes:

```
// deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
    spec:
      containers:
      - name: nodejs-container
        image: nodejswebapp
        ports:
        - containerPort: 3000
        env:
        - name: NODE_ENV
          value: "production"
---
apiVersion: v1
kind: Service
metadata:
  name: nodejs-service
spec:
  selector:
    app: nodejs-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

**Step 6: Push code to GitHub**

Create a new code repository on GitHub, initialize the repository, commit your code changes, and push it to your GitHub repository:

```
git init

git add .
```

*CODE CONTINUES ON NEXT PAGE*

```
git commit -m "Initial commit"

git remote add origin "<remote git repo url>"

git push -u origin main
```

**Step 7: Create a GitHub Actions workflow**

Inside your GitHub repository, go to the *Actions* tab. Create a new workflow (e.g., `main.yml`) in the **.github/ workflows directory**. Inside the GitHub repository settings, create Secrets under actions related to Docker and Kubernetes cluster — these are used in your workflow to authenticate:

```
//main.yml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

    - name: Build Docker image
      run: docker build -t <your-docker-image> .

    - name: Log in to Docker Hub
      uses: docker/login-action@v1
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Build and push Docker image
      uses: docker/build-push-action@v2
      with:
         context: .
         push: true
         tags: <your-docker-image-tag>

  deploy:
```

*CODE CONTINUES ON NEXT PAGE*

```
    needs: build
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up kubectl
      uses: azure/setup-kubectl@v1
      with:
        version: 'latest'

    - name: Set up Kubeconfig
      run: echo "${{ secrets.KUBECONFIG }}" > $HOME/.kube/config

    - name: Deploy to Kubernetes
      run: kubectl apply -f deployment.yaml
```

**Step 8: Trigger the pipeline and monitor**

Modify `server.js` and push it to the `main` branch; this triggers the GitHub Actions workflow. Monitor the workflow progress. It installs the dependencies, sets up npm, builds the Docker image and pushes it to the container registry, and deploys the application to Kubernetes.

Once the workflow is completed successfully, you can access your application that is running inside the Kubernetes cluster. You can leverage open-source monitoring tools like Prometheus and Grafana for metrics.

## Deployment Considerations

There are a few deployment considerations to keep in mind when developing CI/CD pipelines with Kubernetes to maintain security and make the best use of resources:

- Scalability
  - **Use horizontal pod autoscaling** to scale your application's Pods based on how much CPU, memory, or custom metrics are needed. This helps your application work well under varying loads.
  - When using a cloud-based Kubernetes cluster, use the **cluster autoscaler** to change the number of worker nodes as needed to ensure enough resources are available and no money is wasted on idle resources.
  - Ensure your CI/CD pipeline incorporates **pipeline scalability**, allowing it to handle varying workloads as per your project needs.
- Security
  - **Scan container images regularly** to find security issues. Add tools for image scanning into your CI/CD pipeline to stop deploying insecure code.
  - Implement **network policies** to limit how Pods and services talk to each other inside a Kubernetes cluster. This cuts down on ways attackers could get in.
  - Set up secrets management using Kubernetes Secrets or external key vaults to secure and manage sensitive info such as API keys and passwords.
  - Use **role-based access control** to control access to Kubernetes resources and CI/CD pipelines.
- High availability
  - Through **multi-AZ or multi-region deployments**, you can set up your Kubernetes cluster in different availability zones or regions to keep it running during outages.
  - **Pod disruption budgets** help you control how many Pods can be down during planned disruptions (like fixing nodes) or unplanned ones (like when nodes fail).
  - Implement **health checks** to monitor the health of your pods and automatically restart if any fail to maintain availability.

- Secrets management
  - Store API keys, certificates, and passwords as **Kubernetes Secrets**, which are encrypted and added to Pods.
  - You can also consider **external secrets management** tools like HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault if you need dynamic secret generation and auditing.

## Conclusion

Leveraging CI/CD pipelines with Kubernetes has become a must-have approach in today's software development. It revolutionizes the way teams build, test, and deploy apps, leading to more efficiency and reliability. By using automation, teamwork, and the strength of container management, CI/CD with Kubernetes empowers organizations to deliver high-quality software at speed. The growing role of AI and ML will likely have an impact on CI/CD pipelines — such as smarter testing, automated code reviews, and predictive analysis to further enhance the development process.

When teams adopt best practices, keep improving their pipelines, and are attentive to new trends, they can get the most out of CI/CD with Kubernetes, thus driving innovation and success.

CORE

**Naga Santhosh Reddy Vootukuri**

@sunnynagavo

@naga-santhosh-reddy-vootukur

As a seasoned professional with 17+ years working at Microsoft and specialized skills in cloud computing and AI, I lead a team of SDEs focused on initiatives in the Azure SQL deployment space, where we emphasize high availability for SQL customers during critical feature rollouts. Aside from work, I am a technical book reviewer for Apress, Packt, Pearson, and Manning publications; judge hackathons; mentor junior engineers; and write for DZone. I am also a senior IEEE member working on multiple technical committees.

CONTRIBUTOR INSIGHTS

# Guarding Kubernetes From the Threat Landscape

Effective Practices for Container Security

**By Gal Cohen, Backend Engineer at Mind**

Kubernetes is driving the future of cloud computing, but its security challenges require us to adopt a full-scale approach to ensure the safety of our environments. Security is not a one-size-fits-all solution; security is a spectrum, influenced by the specific context in which it is applied. Security professionals in the field rarely declare anything as *entirely* secure, but always as more or less secure than alternatives.

In this article, we are going to present various methods to brace the security of your containers.

## Understanding and Mitigating Container Security Threats

To keep your containerized systems secure, it's important to understand the threats they face. Just like a small leak can sink a ship, even a tiny vulnerability can cause big issues. This section will help you gain a deeper understanding of container security and will provide guidance on how to mitigate the threats that come with it.

### Core Principles of Container Security

Attackers often target containers to hijack their compute power — a common example is to gain access for unauthorized cryptocurrency mining. Beyond this, a compromised container can expose sensitive data, including customer information and workload details. In more advanced attacks, the goal is to escape the container and infiltrate the underlying node. If the attacker succeeds, they can move laterally across the cluster, gaining ongoing access to critical resources such as user code, processing power, and valuable data across other nodes.

One particularly dangerous attack method is **container escape**, where an attacker leverages the fact that containers share the host's kernel. If they gain elevated privileges within a compromised container, they could potentially access data or processes in other containers on the same host. Additionally, the Kubernetes control plane is a prime target. If an attacker compromises one of the control plane components, they can manipulate the entire environment, potentially taking it offline or causing significant disruption.

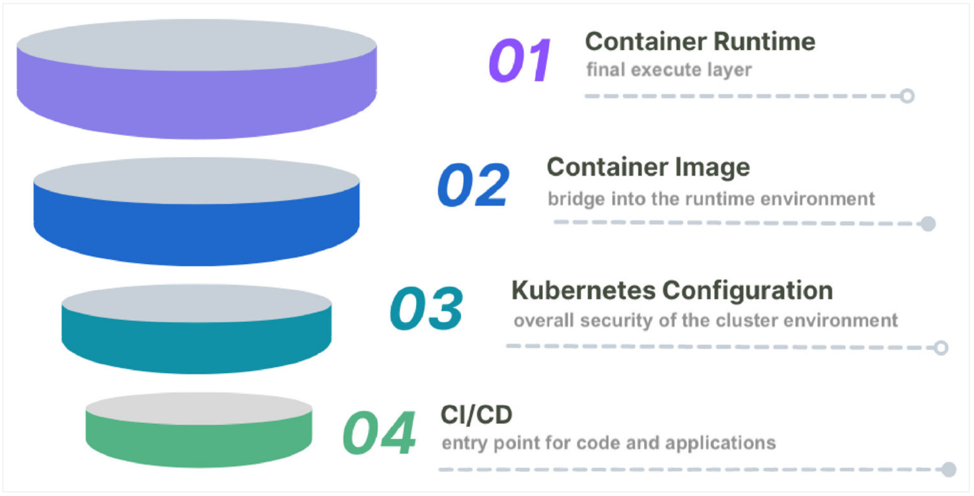
Furthermore, if the etcd database is compromised, attackers could alter or destroy the cluster, steal secrets and credentials, or gather enough information to replicate the application elsewhere.

#### DEFENSE IN DEPTH

Maintaining a secure container environment requires a layered strategy that underscores the principle of defense in depth. This approach involves implementing multiple security controls at various levels.

By deploying overlapping security measures, you create a system where each layer of defense reinforces the others. This way, even if one security measure is breached, the others continue to protect the environment.

**Figure 1.** Defense-in-depth strategy


01
Container Runtime
final execute layer
02
Container Image
bridge into the runtime environment
03
Kubernetes Configuration
overall security of the cluster environment
04
CI/CD
entry point for code and applications

### UNDERSTANDING THE ATTACK SURFACE

Part of the security strategy is understanding and managing the attack surface, which encompasses all potential points of exploitation, including container images, runtime, orchestration tools, the host, and network interfaces. Reducing the attack surface means simplifying the system and minimizing unnecessary components, services, and code.

By limiting what is running and enforcing strict access controls, you decrease the opportunities for vulnerabilities to exist or be exploited, making the system more secure and harder for attackers to penetrate.

## Common Threats and Mitigation Strategies

Let's shift our focus to the everyday threats in container security and discover the tools you can immediately put to work to safeguard your systems.

### VULNERABLE CONTAINER IMAGES

Relying on container images with security vulnerabilities poses significant risks as these vulnerable images often include outdated software or components with publicly known vulnerabilities. A **vulnerability**, in this context, is essentially a *flaw in the code* that malicious actors can leverage to trigger harmful outcomes. An example of this is the infamous Heartbleed flaw in the OpenSSL library, which allowed attackers to access sensitive data by exploiting a coding error. When such flaws are present in container images, they create opportunities for attackers to *breach systems*, leading to potential data theft or service interruptions.

Best practices to secure container images include the following:

- To effectively reduce the attack surface, start by **using minimal base images** that include only the essential components required for your application. This approach minimizes potential vulnerabilities and limits what an attacker can exploit.
  - Tools like Docker's FROM scratch or distroless images can help create these minimal environments.
- **Understanding and managing container image layers** is crucial as each layer can introduce vulnerabilities. By keeping layers minimal and only including what is necessary, you reduce potential attack vectors.
  - Use multi-stage builds to keep the final image lean and regularly review and update your Dockerfiles to remove unnecessary layers.

It's important to **avoid using unverified or outdated images**. Unverified images from public repositories may contain malware, backdoors, or other malicious components. Outdated images often have unpatched vulnerabilities that attackers can exploit. To mitigate these risks, always source images from trusted repositories and regularly update them to the latest versions.

### INSECURE CONTAINER RUNTIME

An insecure container runtime is a critical threat as it can lead to privilege escalation, allowing attackers to gain elevated access within the system. With elevated access, attackers can disrupt services by modifying or terminating critical processes, causing *downtime* and impacting the *availability* of essential applications. They can gain full control over the container environment, manipulating configurations to deploy malicious containers or introduce malware, which can be used as a launchpad for further attacks.

Best practices for hardening the container runtime include the following:

- Implementing strict **security boundaries** and adhering to the **principle of least privilege** are essential for protecting the container runtime.
  - Containers should be configured to run with only the permissions they need to function, minimizing the potential impact of a security breach. This involves setting up role-based access controls.
- **Admission control** is a critical aspect of runtime security that involves validating and regulating requests to create or update containers in the cluster. By employing admission controllers, you can enforce security policies and ensure that only compliant and secure container configurations are deployed.
  - This can include checking for the use of approved base images, ensuring that security policies are applied, and verifying that containers are not running as root.
  - Tools like Open Policy Agent (OPA) can be integrated into your Kubernetes environment to provide flexible and powerful admission control capabilities. On the following page is an example for OPA policy that acts as a

gatekeeper, ensuring no container runs with root privileges:

```
package kubernetes.admission

deny[msg] {
    input.request.kind.kind == "Pod"
    input.request.object.spec.containers[_].securityContext.runAsUser == 0
    msg = "Containers must not run as root."
}
```

There are a few practices to avoid when securing container runtime:

- If a **container running as root** is compromised, an attacker can gain root-level access to the host system, potentially leading to a full system takeover.
- When **containers have unrestricted access to host resources**, like the file system, network, or devices, a compromised container could exploit this access to then tamper with the host system, steal sensitive data, or disrupt other services.
  - To prevent such scenarios, use tools like seccomp and AppArmor. These tools can restrict the system calls that containers make and enforce specific security policies.
  - By applying these controls, you can confine containers to their intended operations, protecting the host system from potential breaches or unauthorized activities.

#### MISCONFIGURED KUBERNETES SETTINGS

Misconfigured Kubernetes settings are a significant threat as they expose the cluster to attacks through overly permissive network policies, weak access controls, and poor secrets management:

- Overly permissive network policies enable attackers to *intercept and tamper with data*.
- Weak access controls *allow unauthorized users* to perform administrative tasks, disrupt services, and alter configurations.
- Poor secrets management *exposes sensitive information* like API keys and passwords, enabling attackers to escalate privileges.

Best practices for secure Kubernetes configuration are as follows:

- The risk of transmitting sensitive information without protection is that it can be intercepted or tampered with by malicious actors during transit. To mitigate this risk, **secure all communication channels with transport layer security** (TLS).
  - Kubernetes offers tools like cert-manager to automate the management and renewal of TLS certificates. This ensures that communication between services remains encrypted and secure, thereby protecting your data from interception or manipulation.
- **Network policies** control the traffic flow between Pods and services in a Kubernetes cluster. By defining network policies, you can isolate sensitive workloads and reduce the risk of lateral movement in case of a compromise.
  - Use Kubernetes' native NetworkPolicy resource to create rules that enforce your desired network security posture.

On the other hand, it's important to **avoid exposing unnecessary application ports**. Exposure of ports provides multiple entry points for attackers, making the cluster more vulnerable to exploits.

### CI/CD Security

CI/CD pipelines are granted extensive permissions, ensuring they can interact closely with production systems and manage updates. However, this extensive access also makes CI/CD pipelines a significant security risk. If compromised, attackers can exploit these broad permissions to manipulate deployments, introduce malicious code, gain unauthorized access to critical systems, steal sensitive data, or create backdoors for ongoing access.

There are several best practices to implement when securing CI/CD. The first best practice is ensuring that once a container image is built and deployed, it is **immutable**. We always want to make sure the Pod is running on exactly what we intended. It also helps in quickly identifying and rolling back to previous stable versions if a security issue arises, maintaining a reliable and predictable deployment process.

Implementing immutable deployments involves several key steps to ensure consistency and security:

1. Assign unique version tags to each container image build, avoiding mutable tags like "latest," and use Infrastructure-as-Code tools like Terraform or Ansible to maintain consistent setups.
2. Configure containers with read-only file systems to prevent changes post-deployment.
3. Implement continuous monitoring with tools like Prometheus and runtime security with Falco to help detect and alert to unauthorized changes, maintaining the security and reliability of your deployments.

Another best practice is implementing **image vulnerability scanning in CI/CD**. Vulnerability scanners meticulously analyze the components of container images, identifying known security flaws that could be exploited. Beyond just examining packages managed by tools like DNF or apt, advanced scanners also inspect additional files added during the build process, such as those introduced through Dockerfile commands like `ADD`, `COPY`, or `RUN`.

It's important to include both third-party and internally created images in these scans as new vulnerabilities are constantly emerging. To guarantee that images are thoroughly scanned for vulnerabilities before deployment, scanning tools like Clair or Trivy can be directly embedded into your CI/CD pipeline.

**Do not store sensitive information directly in the source code** (e.g., API keys, passwords) as this increases the risk of unauthorized access and data breaches. Use secrets management tools like SOPS, AWS Secrets Manager, or Google Cloud Secret Manager to securely handle and encrypt sensitive information.

## Conclusion

Regularly assessing and improving Kubernetes security measures is not just important — it's essential. By implementing the strategies we introduced above, organizations can protect their Kubernetes environments, ensuring that containerized applications are more secure and resilient against challenges. In the future, we anticipate that attackers will develop more sophisticated methods to specifically bypass Kubernetes' built-in security features. As organizations increasingly rely on Kubernetes for critical workloads, attackers will likely invest time in uncovering new vulnerabilities or weaknesses in Kubernetes' security architecture, potentially leading to breaches that are more difficult to detect and mitigate.

The path to a secure Kubernetes environment is clear, and the time to act is now. Prioritize security to safeguard your future.

**Gal Cohen**

@Galco

@galco5

Gal Cohen is a software engineer with years of experience in the cloud and engineering. She has earned a reputation as an expert in her field. Gal is also a passionate tech blogger with thousands of frequent readers.

CONTRIBUTOR INSIGHTS

# Kubernetes Observability

Lessons Learned From Running Kubernetes in Production

**By Yitaek Hwang, Software Engineer at NYDIG**

In recent years, observability has re-emerged as a critical aspect of DevOps and software engineering in general, driven by the growing complexity and scale of modern, cloud-native applications. The transition toward microservices architecture as well as complex cloud deployments — ranging from multi-region to multi-cloud, or even hybrid-cloud, environments — has highlighted the shortcomings of traditional methods of monitoring.

In response, the industry has standardized utilizing logs, metrics, and traces as the three pillars of observability to provide a more comprehensive sense of how the application and entire stack is performing. We now have a plethora of tools to collect, store, and analyze various signals to diagnose issues, optimize performance, and respond to issues.

Yet anyone working with Kubernetes will still say that observability in Kubernetes remains challenging. Part of it comes from the inherent complexity of working with Kubernetes, but the fact of the matter is that logs, metrics, and traces alone don't make up *observability*. Also, the vast ecosystem of observability tooling does not necessarily equate to ease of use or high ROI, especially given today's renewed focus on cost. In this article, we'll dive into some considerations for Kubernetes observability, challenges of and some potential solutions for implementing it, and the oft forgotten aspect of developer experience in observability.

## Considerations for Kubernetes Observability

When considering observability for Kubernetes, most have a tendency to dive straight into tool choices, but it's advisable to take a hard look at what falls under the scope of things to "observe" for your use case. Within Kubernetes alone, we already need to consider:

- **Cluster components** – API server, etcd, controller manager, scheduler
- **Node components** – kublect, kube-proxy, container runtime
- **Other resources** – CoreDNS, storage plugins, Ingress controllers
- **Network** – CNI, service mesh
- **Security and access** – audit logs, security policies
- **Application** – both internal and third-party applications

And most often, we inevitably have components that run outside of Kubernetes but interface with many applications running inside. Most notably, we have databases ranging from managed cloud offerings to external data lakes. We also have things like serverless functions, queues, or other Kubernetes clusters that we need to think about.

Next, we need to identify the users of Kubernetes as well as the consumers of these observability tools. It's important to consider these personas as building for an internal-only cluster vs. a multi-tenant SaaS cluster may have different requirements (e.g., privacy, compliance). Also, depending on the team composition, the primary consumers of these tools may be developers or dedicated DevOps/SRE teams who will have different levels of expertise with not only these tools but with Kubernetes itself.

Only after considering the above factors can we start to talk about what tools to use. For example, if most applications are already on Kubernetes, using a Kubernetes-focused tool may suffice, whereas organizations with lots of legacy components may elect to reuse an existing observability stack. Also, a large organization with various teams mostly operating as independent verticals may opt to use their own tooling stack, whereas a smaller startup may opt to pay for an enterprise offering to simplify the setup across teams.

## Challenges and Recommendations for Observability Implementation

After considering the scope and the intended audience of our observability stack, we're ready to narrow down the tool choices. Largely speaking, there are two options for implementing an observability stack: open source and commercial/SaaS.

## Open-Source Observability Stack

The primary challenge with implementing a fully open-source observability solution is that there is no single tool that covers all aspects. Instead, what we have are ecosystems or stacks of tools that cover different aspects of observability. One of the more popular tech stacks from Prometheus and Grafana Lab's suite of products include:

- Prometheus for scraping metrics and alerting
- Loki for collecting logs
- Tempo for distributed tracing
- Grafana for visualization

While the above setup does cover a vast majority of observability requirements, they still operate as individual microservices and do not provide the same level of uniformity as a commercial or SaaS product. But in recent years, there has been a strong push to at least standardize on OpenTelemetry conventions to unify how to collect metrics, logs, and traces. Since OpenTelemetry is a framework that is tool agnostic, it can be used with many popular open-source tools like Prometheus and Jaeger.

Ideally, architecting with OpenTelemetry in mind will make standardization of how to generate, collect, and manage telemetry data easier with the growing list of compliant open-source tools. However, in practice, most organizations will already have established tools or in-house versions of them — whether that is the EFK (Elasticsearch, Fluentd, Kibana) stack or Prometheus/Grafana. Instead of forcing a new framework or tool, apply the ethos of standardization and improve what and how telemetry data is collected and stored.

Finally, one of the common challenges with open-source tooling is dealing with storage. Some tools like Prometheus cannot scale without offloading storage with another solution like Thanos or Mimir. But in general, it's easy to forget to monitor the observability tooling health itself and scale the back end accordingly. More telemetry data does not necessarily equal more signals, so keep a close eye on the volume and optimize as needed.

## Commercial Observability Stack

On the commercial offering side, we usually have agent-based solutions where telemetry data is collected from agents running as DaemonSets on Kubernetes. Nowadays, almost all commercial offerings have a comprehensive suite of tools that combine into a seamless experience to connect logs to metrics to traces in a single user interface.

The primary challenge with commercial tools is controlling cost. This usually comes in the form of exposing cardinality from tags and metadata. In the context of Kubernetes, every Pod has tons of metadata related to not only Kubernetes state but the state of the associated tooling as well (e.g., annotations used by Helm or ArgoCD). These metadata then get ingested as additional tags and date fields by the agents.

Since commercial tools have to index all the data to make telemetry queryable and sortable, increased cardinality from additional dimensions (usually in the form of tags) causes issues with performance and storage. This directly results in higher cost to the end user. Fortunately, most tools now allow the user to control which tags to index and even downsample data to avoid getting charged for repetitive data points that are not useful. Be aggressive with filters and pipeline logic to only index what is needed; otherwise, don't be surprised by the ballooning bill.

## Remembering the Developer Experience

Regardless of the tool choice, one common pitfall that many teams face is over-optimizing for ops usage and neglecting the developer experience when it comes to observability. Despite the promise of DevOps, observability often falls under the realm of ops teams, whether that be platform, SRE, or DevOps engineering. This makes it easy for teams to build for what they know and what they need, over-indexing on infrastructure and not investing as much on application-level telemetry. This ends up alienating developers to invest less time or become too reliant on their ops counterparts for setup or debugging.

To make observability truly useful for everyone involved, don't forget about these points:

- **Access.** It's usually more of a problem with open-source tools, but make sure access to logs, dashboards, and alerts are not gated by unnecessary approvals. Ideally, having quick links from existing mediums like IDEs or Slack can make tooling more accessible.
- **Onboarding.** It's rare for developers to go through the same level of onboarding in learning how to use any of these tools. Invest some time to get them up to speed.

- **Standardization vs. flexibility.** While a standard format like JSON is great for indexing, it may not be as human readable and is filled with extra information. Think of ways to present information in a usable format.

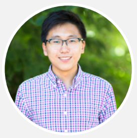
At the end of the day, the goals of developers and ops teams should be aligned. We want tools that are easy to integrate, with minimal overhead, that produce intuitive dashboards and actionable, contextual information without too much noise. Even with the best tools, you still need to work with developers who are responsible for generating telemetry and also acting on it, so don't neglect the developer experience entirely.

## Final Thoughts

Observability has been a hot topic in recent years due to several key factors, including the rise of complex, modern software coupled with DevOps and SRE practices to deal with that complexity. The community has moved past the simple notion of monitoring to defining the three pillars of observability as well as creating new frameworks to help with generation, collection, and management of these telemetry data.

Observability in a Kubernetes context has remained challenging so far given the large scope of things to "observe" as well as the complexity of each component. With the open source ecosystem, we have seen a large fragmentation of specialized tools that is just now integrating into a standard framework. On the commercial side, we have great support for Kubernetes, but cost control has been a huge issue. And to top it off, lost in all of this complexity is the developer experience in helping feed data into and using the insights from the observability stack.

But as the community has done before, tools and experience will continue to improve. We already see significant research and advances in how AI technology can improve observability tooling and experience. Not only do we see better data-driven decision making, but generative AI technology can also help surface information better in context to make tools more useful without too much overhead.



CORE

**Yitaek Hwang**

@yitaek

yitaekhwang.com

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

CONTRIBUTOR INSIGHTS

# Key Considerations for Effective AI/ML Deployments in Kubernetes

**By Rajesh Vishnupant Gheware, Director at Gheware UniGPS Solutions LLP**

Kubernetes has become a cornerstone in modern infrastructure, particularly for deploying, scaling, and managing artificial intelligence and machine learning (AI/ML) workloads. As organizations increasingly rely on machine learning models for critical tasks like data processing, model training, and inference, Kubernetes offers the flexibility and scalability needed to manage these complex workloads efficiently.

By leveraging Kubernetes' robust ecosystem, AI/ML workloads can be dynamically orchestrated, ensuring optimal resource utilization and high availability across cloud environments. This synergy between Kubernetes and AI/ML empowers organizations to deploy and scale their ML workloads with greater agility and reliability.

This article delves into the key aspects of managing AI/ML workloads within Kubernetes, focusing on strategies for resource allocation, scaling, and automation specific to this platform. By addressing the unique demands of AI/ML tasks in a Kubernetes environment, it provides practical insights to help organizations optimize their ML operations. Whether handling resource-intensive computations or automating deployments, this guide offers actionable advice for leveraging Kubernetes to enhance the performance, efficiency, and reliability of AI/ML workflows, making it an indispensable tool for modern enterprises.

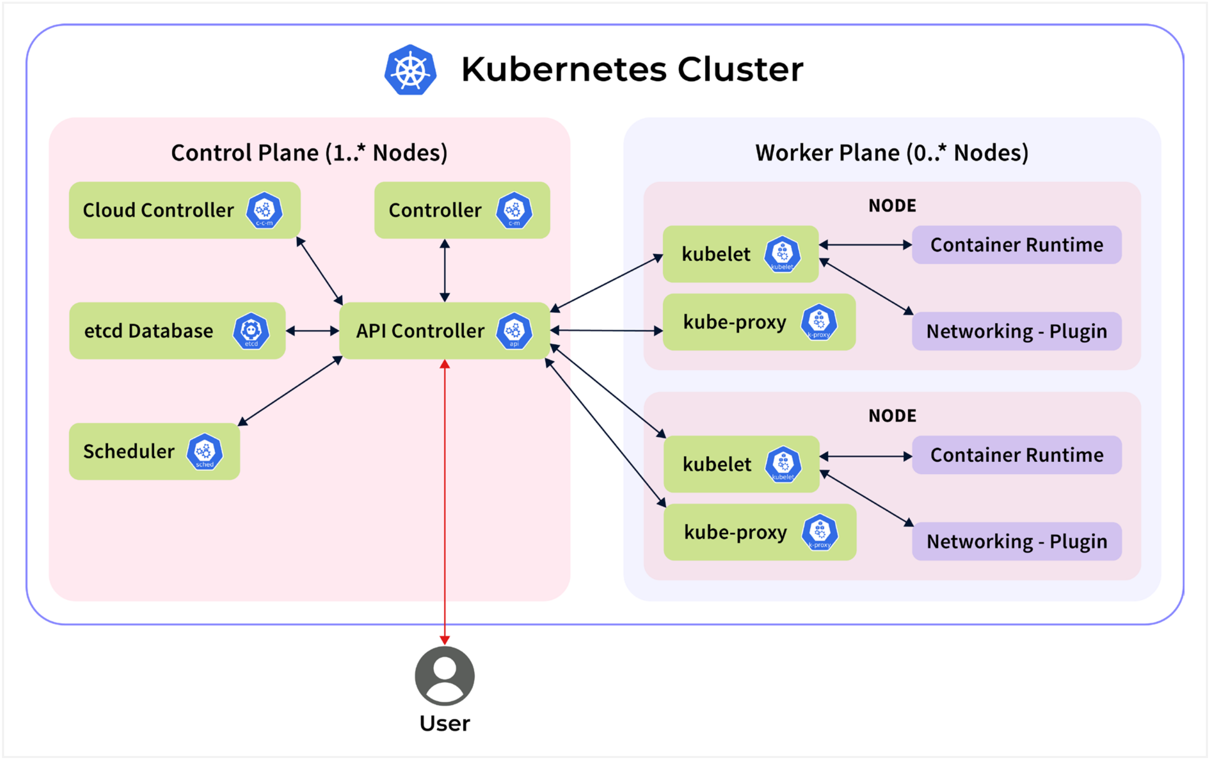
## Understanding Kubernetes and AI/ML Workloads

In order to effectively manage AI/ML workloads in Kubernetes, it is important to first understand the architecture and components of the platform.

### Overview of Kubernetes Architecture

Kubernetes architecture is designed to manage containerized applications at scale. The architecture is built around two main components: the **control plane** (coordinator nodes) and the **worker nodes**.

**Figure 1.** Kubernetes architecture



For more information, or to review the individual components of the architecture in Figure 1, check out the Kubernetes Documentation.

### AI/ML Workloads: Model Training, Inference, and Data Processing

AI/ML workloads are computational tasks that involve training machine learning models, making predictions (inference) based on those models, and processing large datasets to derive insights. AI/ML workloads are essential for driving innovation and making data-driven decisions in modern enterprises:

- **Model training** enables systems to learn from vast datasets, uncovering patterns that power intelligent applications.
- **Inference** allows these models to generate real-time predictions, enhancing user experiences and automating decision-making processes.
- Efficient **data processing** is crucial for transforming raw data into actionable insights, fueling the entire AI/ML pipeline.

However, managing these computationally intensive tasks requires a robust infrastructure. This is where Kubernetes comes into play, providing the scalability, automation, and resource management needed to handle AI/ML workloads effectively, ensuring they run seamlessly in production environments.

## Key Considerations for Managing AI/ML Workloads in Kubernetes

Successfully managing AI/ML workloads in Kubernetes requires careful attention to several critical factors. This section outlines the key considerations for ensuring that your AI/ML workloads are optimized for performance and reliability within a Kubernetes environment.

### Resource Management

Effective resource management is crucial when deploying AI/ML workloads on Kubernetes. AI/ML tasks, particularly model training and inference, are resource intensive and often require specialized hardware such as GPUs or TPUs. Kubernetes allows for the **efficient allocation of CPU, memory, and GPUs** through resource requests and limits. These configurations ensure that containers have the necessary resources while preventing them from monopolizing node capacity.

Additionally, Kubernetes supports the use of node selectors and taints/tolerations to assign workloads to nodes with the required hardware (e.g., GPU nodes). Managing resources efficiently helps optimize cluster performance, ensuring that AI/ML tasks run smoothly without over-provisioning or under-utilizing the infrastructure. Handling **resource-intensive tasks** requires careful planning, particularly when managing distributed training jobs that need to run across multiple nodes.

These workloads benefit from Kubernetes' ability to distribute resources while ensuring that high-priority tasks receive adequate computational power.

### Scalability

Scalability is another critical factor in managing AI/ML workloads in Kubernetes. **Horizontal scaling**, where additional Pods are added to handle increased demand, is particularly useful for stateless workloads like inference tasks that can be easily distributed across multiple Pods. **Vertical scaling**, which involves increasing the resources available to a single Pod (e.g., more CPU or memory), can be beneficial for resource-intensive processes like model training that require more power to handle large datasets.

In addition to Pod autoscaling, Kubernetes clusters benefit from **cluster autoscaling** to dynamically adjust the number of worker nodes based on demand. Karpenter is particularly suited for AI/ML workloads due to its ability to quickly provision and scale nodes based on real-time resource needs. Karpenter optimizes node placement by selecting the most appropriate instance types and regions, taking into account workload requirements like GPU or memory needs.

By leveraging Karpenter, Kubernetes clusters can efficiently scale up during resource-intensive AI/ML tasks, ensuring that workloads have sufficient capacity without over-provisioning resources during idle times. This leads to improved cost efficiency and resource utilization, especially for complex AI/ML operations that require on-demand scalability.

These autoscaling mechanisms enable Kubernetes to dynamically adjust to workload demands, optimizing both cost and performance.

## Data Management

AI/ML workloads often require access to large datasets and persistent storage for model checkpoints and logs. Kubernetes offers several **persistent storage options** to accommodate these needs, including PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs). These options allow workloads to access durable storage across various cloud and on-premises environments. Additionally, Kubernetes integrates with cloud storage solutions like AWS EBS, Google Cloud Storage, and Azure Disk Storage, making it easier to manage storage in hybrid or multi-cloud setups.

Handling **large volumes of training data** requires efficient data pipelines that can stream or batch process data into models running within the cluster. This can involve integrating with external systems, such as distributed file systems or databases, and using tools like Apache Kafka for real-time data ingestion. Properly managing data is essential for maintaining high-performance AI/ML pipelines, ensuring that models have quick and reliable access to the data they need for both training and inference.

## Deployment Automation

Automation is key to managing the complexity of AI/ML workflows, particularly when deploying models into production. **CI/CD pipelines** can automate the build, test, and deployment processes, ensuring that models are continuously integrated and deployed with minimal manual intervention. Kubernetes integrates well with CI/CD tools like Jenkins, GitLab CI/CD, and Argo CD, enabling seamless automation of model deployments. **Tools and best practices** for automating AI/ML deployments include using Helm for managing Kubernetes manifests, Kustomize for configuration management, and Kubeflow for orchestrating ML workflows.

These tools help standardize the deployment process, reduce errors, and ensure consistency across environments. By automating deployment, organizations can rapidly iterate on AI/ML models, respond to new data, and scale their operations efficiently, all while maintaining the agility needed in fast-paced AI/ML projects.

## Scheduling and Orchestration

Scheduling and orchestration for AI/ML workloads require more nuanced approaches compared to traditional applications. Kubernetes excels at managing these different scheduling needs through its flexible and powerful scheduling mechanisms. **Batch scheduling** is typically used for tasks like model training, where large datasets are processed in chunks. Kubernetes supports batch scheduling by allowing these jobs to be queued and executed when resources are available, making them ideal for non-critical workloads that are not time sensitive. Kubernetes Job and CronJob resources are particularly useful for automating the execution of batch jobs based on specific conditions or schedules.

On the other hand, **real-time processing** is used for tasks like model inference, where latency is critical. Kubernetes ensures low latency by providing mechanisms such as Pod priority and preemption, ensuring that real-time workloads have immediate access to the necessary resources. Additionally, Kubernetes' HorizontalPodAutoscaler can dynamically adjust the number of pods to meet demand, further supporting the needs of real-time processing tasks. By leveraging these Kubernetes features, organizations can ensure that both batch and real-time AI/ML workloads are executed efficiently and effectively.

**Gang scheduling** is another important concept for distributed training in AI/ML workloads. Distributed training involves breaking down model training tasks across multiple nodes to reduce training time, and gang scheduling ensures that all the required resources across nodes are scheduled simultaneously. This is crucial for distributed training, where all parts of the job must start together to function correctly. Without gang scheduling, some tasks might start while others are still waiting for resources, leading to inefficiencies and extended training times. Kubernetes supports gang scheduling through custom schedulers like Volcano, which is designed for high-performance computing and ML workloads.

## Latency and Throughput

Performance considerations for AI/ML workloads go beyond just resource allocation; they also involve optimizing for latency and throughput. **Latency** refers to the time it takes for a task to be processed, which is critical for real-time AI/ML workloads such as model inference. Ensuring low latency is essential for applications like online recommendations, fraud detection, or any use case where real-time decision making is required. Kubernetes can manage latency by prioritizing real-time workloads, using features like node affinity to ensure that inference tasks are placed on nodes with the least network hops or proximity to data sources.

**Throughput**, on the other hand, refers to the number of tasks that can be processed within a given time frame. For AI/ML workloads, especially in scenarios like batch processing or distributed training, high throughput is crucial. Optimizing throughput often involves scaling out workloads horizontally across multiple Pods and nodes. Kubernetes' autoscaling capabilities, combined with optimized scheduling, ensure that AI/ML workloads maintain high throughput — even as demand increases. Achieving the right balance between latency and throughput is vital for the efficiency of AI/ML pipelines, ensuring that models perform at their best while meeting real-world application demands.

## A Step-by-Step Guide: Deploying TensorFlow Sentiment Analysis Model on AWS EKS

In this example, we demonstrate how to deploy a TensorFlow-based sentiment analysis model using AWS Elastic Kubernetes Service (EKS). This hands-on guide will walk you through setting up a Flask-based Python application, containerizing it with Docker, and deploying it on AWS EKS using Kubernetes. Although many tools are suitable, TensorFlow was chosen for this example due to its popularity and robustness in developing AI/ML models, while AWS EKS provides a scalable and managed Kubernetes environment that simplifies the deployment process.

By following this guide, readers will gain practical insights into deploying AI/ML models in a cloud-native environment, leveraging Kubernetes for efficient resource management and scalability.

### Step 1: Create a Flask-based Python app setup

Create a Flask app (`app.py`) using the Hugging Face transformers pipeline for sentiment analysis:

```
from flask import Flask, request, jsonify
from transformers import pipeline

app = Flask(__name__)
sentiment_model = pipeline("sentiment-analysis")

@app.route('/analyze', methods=['POST'])
def analyze():
    data = request.get_json()
    result = sentiment_model(data['text'])
    return jsonify(result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

### Step 2: Create requirements.txt

```
transformers==4.24.0
torch==1.12.1
flask
jinja2
markupsafe==2.0.1
```

### Step 3: Build Docker image

Create a Dockerfile to containerize the app:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Build and push the Docker image:

```
docker build -t brainupgrade/aiml-sentiment:20240825 .
docker push brainupgrade/aiml-sentiment:20240825
```

**Step 4: Deploy to AWS EKS with Karpenter**

Create a Kubernetes Deployment manifest (`deployment.yaml`):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sentiment-analysis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sentiment-analysis
  template:
    metadata:
      labels:
        app: sentiment-analysis
    spec:
      containers:
      - name: sentiment-analysis
        image: brainupgrade/aiml-sentiment:20240825
        ports:
        - containerPort: 5000
        resources:
          requests:
            aws.amazon.com/neuron: 1
          limits:
            aws.amazon.com/neuron: 1
      tolerations:
      - key: "aiml"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
```

Apply the Deployment to the EKS cluster:

```
kubectl apply -f deployment.yaml
```

Karpenter will automatically scale the cluster and launch an `inf1.xlarge` **EC2** instance based on the resource specification (`aws.amazon.com/neuron: 1`). Karpenter also installs appropriate device drivers for this special AWS EC2 instance of `inf1.xlarge`, which is optimized for deep learning inference, featuring four vCPUs, 16 GiB RAM, and one Inferentia chip.

Reference Karpenter spec as follows:

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  limits:
    resources:
      cpu: "16"
  provider:
    instanceProfile: eksctl-KarpenterNodeInstanceProfile-<cluster-name>
```

*CODE CONTINUES ON NEXT PAGE*

```
    securityGroupSelector:
      karpenter.sh/discovery: <cluster-name>
    subnetSelector:
      karpenter.sh/discovery: <cluster-name>
  requirements:
  - key: karpenter.sh/capacity-type
    operator: In
    values:
    - spot
  - key: node.kubernetes.io/instance-type
    operator: In
    values:
    - inf1.xlarge
  - key: kubernetes.io/os
    operator: In
    values:
    - linux
  - key: kubernetes.io/arch
    operator: In
    values:
    - amd64
  ttlSecondsAfterEmpty: 30
```

**Step 5: Test the application**

Once deployed and exposed via an AWS Load Balancer or Ingress, test the app with the following cURL command:

```
curl -X POST -H "Content-Type: application/json" -d '{"text":"I love using this product!"}'
https://<app-url>/analyze
```

This command sends a sentiment analysis request to the deployed model endpoint: `https://<app-url>/analyze`.

## Challenges and Solutions

Managing AI/ML workloads in Kubernetes comes with its own set of challenges, from handling ephemeral containers to ensuring security and maintaining observability. In this section, we will explore these challenges in detail and provide practical solutions to help you effectively manage AI/ML workloads in a Kubernetes environment.

### Maintaining State in Ephemeral Containers

One of the main challenges in managing AI/ML workloads in Kubernetes is **handling ephemeral containers while maintaining state**. Containers are designed to be stateless, which can complicate AI/ML workflows that require persistent storage for datasets, model checkpoints, or intermediate outputs. For maintaining state in ephemeral containers, Kubernetes offers PVs and PVCs, which enable long-term storage for AI/ML workloads, even if the containers themselves are short-lived.

### Ensuring Security and Compliance

Another significant challenge is **ensuring security and compliance**. AI/ML workloads often involve sensitive data, and maintaining security at multiple levels — network, access control, and data integrity — is crucial for meeting compliance standards. To address security challenges, Kubernetes provides role-based access control (RBAC) and NetworkPolicies. **RBAC** ensures that users and services have only the necessary permissions, minimizing security risks. **NetworkPolicies** allow for fine-grained control over network traffic, ensuring that sensitive data remains protected within the cluster.

### Observability in Kubernetes Environments

Additionally, **observability** is a key challenge in Kubernetes environments. AI/ML workloads can be complex, with numerous microservices and components, making it difficult to monitor performance, track resource usage, and detect potential issues in real time. **Monitoring and logging** are essential for observability in Kubernetes. Tools like

Prometheus and Grafana provide robust solutions for monitoring system health, resource usage, and performance metrics. Prometheus can collect real-time metrics from AI/ML workloads, while Grafana visualizes this data, offering actionable insights for administrators. Together, they enable proactive monitoring, allowing teams to identify and address potential issues before they impact operations.

## Conclusion

In this article, we explored the key considerations for managing AI/ML workloads in Kubernetes, focusing on resource management, scalability, data handling, and deployment automation. We covered essential concepts like efficient CPU, GPU, and TPU allocation, scaling mechanisms, and the use of persistent storage to support AI/ML workflows. Additionally, we examined how Kubernetes uses features like RBAC and NetworkPolicies and tools like Prometheus and Grafana to ensure security, observability, and monitoring for AI/ML workloads.

Looking ahead, AI/ML workload management in Kubernetes is expected to evolve with advancements in hardware accelerators and more intelligent autoscaling solutions like Karpenter. Integration of AI-driven orchestration tools and the emergence of Kubernetes-native ML frameworks will further streamline and optimize AI/ML operations, making it easier to scale complex models and handle ever-growing data demands.

For practitioners, staying informed about the latest Kubernetes tools and best practices is crucial. Continuous learning and adaptation to new technologies will empower you to manage AI/ML workloads efficiently, ensuring robust, scalable, and high-performance applications in production environments.

**Rajesh Vishnupant Gheware**

@rajeshg007

in @rajesh-gheware

Rajesh Gheware is an accomplished chief architect with more than 24 years of experience in cloud computing, Kubernetes, DevOps, and security. A published author and technical mentor, he drives innovation in complex IT projects. Connect with him on LinkedIn for insights into advanced Kubernetes operations and cutting-edge technology strategies.

ADDITIONAL RESOURCES

# Diving Deeper Into Kubernetes

## DZONE TREND REPORTS

**Kubernetes in the Enterprise:** Redefining the Container Ecosystem

In DZone's 2023 report, we dive into Kubernetes over the last year, its core usages as well as emerging trends (and challenges), and what these all mean for our developer community. Featured in this report are actionable observations from our original research, expert content written by members of the DZone Community, and other helpful resources to help you go forth in your organizations, projects, and repos.

**Cloud Native:** Championing Cloud Development Across the SDLC

DZone's 2024 Trend Report explores how evolving technology and methodologies are driving the vision for what cloud native means today. Articles from DZone Community experts bring the cloud native "pillars" into conversation via topics like automating the cloud via orchestration and AI, using shift left to improve delivery and strengthen security, surviving observability challenges, and strategizing cost optimizations.

## DZONE REFCARDS

**Kubernetes Monitoring Essentials:** Exploring Approaches for Monitoring Distributed Kubernetes [...]

This Refcard presents the benefits and challenges of monitoring Kubernetes, followed by the fundamentals of building a Kubernetes monitoring framework: how to capture monitoring data insights, leverage core Kubernetes components for monitoring, and identify key metrics — plus the critical Kubernetes components and services you should be monitoring.

**Cloud-Native Application Security:** Patterns and Anti-Patterns

This Refcard walks through the critical challenges of cloud-native application security, how to build security into the CI/CD pipeline, and the core patterns and anti-patterns for securing your cloud-native apps.

## CORE CREATORS

**Abhishek Gupta,** *Principal Developer Advocate,* AWS

Over the course of his career, Abhishek has worn multiple hats including engineering, product management, and developer advocacy. Most of his work has revolved around open-source technologies, including distributed data systems and cloud-native platforms. Abhishek is also an open source contributor and avid blogger.

**Marija Naumovska,** *Product Manager,* Microtica

As a co-founder of Microtica, Marija helps developers get their applications deployed on the cloud in minutes. She's a software engineer with 10+ years of experience, who now works as head of growth and technical content producer full time. She writes about cloud, DevOps, GitOps, and containerization topics.

**Saurabh Dashora,** *Founder,* ProgressiveCoder

Saurabh is a full-stack architect, technical writer, and guest author for various publications. He has expertise in building distributed systems across multiple business domains, such as banking, autonomous driving, and retail. He runs a tech blog on cloud, microservices, and web development, where he has written hundreds of articles. Apart from work, he enjoys reading and playing video games.

ADDITIONAL RESOURCES

# Solutions Directory

This directory contains Kubernetes and cloud-native tools for end-to-end container management. 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 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

2024 PARTNERS

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Fairwinds | Fairwinds Managed Kubernetes-as-a-Service | Production-ready Kubernetes: architected, built, managed | By request | fairwinds.com/managed-kubernetes |
| Spot by NetApp | Ocean | Serverless infrastructure container engine | By request | spot.io/product/ocean |
| Sysdig | Sysdig Secure | Cloud and container security | By request | sysdig.com/products/platform |

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Akamai | Linode Kubernetes Engine (LKE) | Fully managed Kubernetes container orchestration engine | By request | linode.com/products/kubernetes |
| Amazon Web Services | Amazon Elastic Container Service (ECS) | Fully managed containers | Free tier | aws.amazon.com/ecs |
|  | Amazon Elastic Kubernetes Service (EKS) | Managed Kubernetes service | By request | aws.amazon.com/eks |
|  | AWS Fargate | Serverless compute for containers |  | aws.amazon.com/fargate |
| Ambassador Labs | Edge Stack API Gateway | Kubernetes-native API gateway | Free tier | getambassador.io/products/edge-stack/api-gateway |
| Anchore | Anchore Enterprise | Software composition analysis for cloud-native apps | Trial period | anchore.com |
|  | Grype | Vulnerability scanning for container images and filesystems | Open source | github.com/anchore/grype |
| Apache Software Foundation | Ozone | Scalable, distributed object storage | Open source | ozone.apache.org/docs/1.0.0/index.html |
|  | SkyWalking | App performance monitoring tool for distributed systems |  | skywalking.apache.org |
| Aqua Security | Aqua CNAPP | Cloud-native security platform | By request | aquasec.com/aqua-cloud-native-security-platform |
|  | kube-bench | Kubernetes compliance check with CIS benchmark | Open source | github.com/aquasecurity/kube-bench |
| Argo | Argo CD | Declarative continuous delivery tool | Open source | argoproj.github.io/cd |
|  | Argo Workflows | Kubernetes-native workflow engine |  | argoproj.github.io/workflows |
| Backstage | Backstage | Open-source framework for building developer portals | Open source | backstage.io |
| Buildpacks | Cloud Native Buildpacks | Automate build and runtime environments | Open source | buildpacks.io |
| Canonical | Juju | Orchestration engine for software operators | Open source | juju.is |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| CAST AI | CAST AI | Kubenrnetes automation | Free tier | cast.ai |
| Chaos Mesh | Chaos Mesh | Cloud-native chaos engineering platform | Open source | chaos-mesh.org |
| Chronosphere | Chronosphere | Observability with metrics, logs, traces, and telemetry pipelines | By request | chronosphere.io |
| Cilium | Hubble | Network, service, and security observability for Kubernetes | Open source | github.com/cilium/hubble |
| CircleCI | CircleCI | CI/CD automation | Free tier | circleci.com |
| CloudBees | CloudBees CI | CI for Jenkins in the enterprise | By request | cloudbees.com/capabilities/continuous-integration |
| Cloudsmith | Cloudsmith | Secure, cloud-native artifact management | Free tier | cloudsmith.com |
| CNI | CNI | Networking for Linux containers | Open source | cni.dev |
| Cockroach Labs | CockroachDB | Distributed SQL database | Free tier | cockroachlabs.com |
| containerd | containerd | Industry-standard container runtime | Open source | containerd.io |
| Contour | Contour | Kubernetes ingress controller | Open source | projectcontour.io |
| Cortex | Cortex | Internal developer portal | By request | cortex.io |
| Couchbase | Autonomous Operator | Self-managed Couchbase for Kubernetes | Free | couchbase.com/products/operator |
| CRI-O | CRI-O | Lightweight container runtime | Open source | cri-o.io |
| Crossplane | Crossplane | Cloud-native control plane framework | Open source | crossplane.io |
| DaoCloud | DaoCloud 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 |
| Diamanti | Ultima Accelerator | Optimized storage and networking to improve Kubernetes performance | By request | diamanti.com/products/ultima-accelerator |
|  | Ultima Enterprise | High-performance Kubernetes storage and data management | Trial period | diamanti.com/products/ultima-enterprise |
| DigitalOcean | DigitalOcean Kubernetes | Managed Kubernetes clusters | By request | digitalocean.com/products/kubernetes |
| Docker | Docker Desktop | Container software | Free tier | docker.com/products/docker-desktop |
|  | Docker Hub | Container registry | Trial period | docker.com/products/docker-hub |
| Dynatrace | Dynatrace Platform | End-to-end observability with AIOps and app security | Trial period | dynatrace.com |
| Envoy | Envoy Proxy | Edge and service proxy for cloud-native apps | Open source | envoyproxy.io |
| etcd | etcd | Distributed key-value store | Open source | etcd.io |
| F5 | Aspen Mesh | Deploy, manage, and monitor microservices at scale | By request | f5.com/products/aspen-mesh |
|  | NGINX Ingress Controller | Kubernetes-native API gateways, load balancers, Ingress controllers | Trial period | nginx.com/products/nginx-ingress-controller |
| Falco | Falco | Container runtime security | Open source | falco.org |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Fluent Bit | Fluent Bit | End-to-end observability pipeline for containerized environments | Open source | fluentbit.io |
| Flux | Flagger | Progressive delivery operator for Kubernetes | Open source | flagger.app |
|  | Flux | Continuous delivery solutions for Kubernetes |  | fluxcd.io |
| Google Cloud | Google Kubernetes Engine | Scalable and fully automated Kubernetes service | Trial period | cloud.google.com/kubernetes-engine |
| Grafana Labs | Grafana Cloud | Analytics and monitoring tool | Free tier | grafana.com/products/cloud |
| Harbor | Harbor | Cloud-native registry for managing and securing container images | Open source | goharbor.io |
| Harness | Harness | Software delivery pipeline automation | Free tier | harness.io |
| harvesterhci.io | Harvester | Cloud-native hyperconverged infrastructure | Open source | harvesterhci.io |
| HashiCorp | Terraform | Infrastructure automation | Open source | terraform.io |
| Helm | Helm | Package manager for Kubernetes | Open source | helm.sh |
| Huawei Cloud | 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 |
| Isovalent | Isovalent | eBPF-based networking, security, and observability | By request | isovalent.com |
| Istio | Istio | Service mesh | Open source | istio.io |
| Jenkins X | Jenkins X | Cloud-native CI/CD | Open source | jenkins-x.io |
| JFrog | JFrog Software Supply Chain Platform | DevOps, software supply chain security, pipeline automation | Free tier | jfrog.com/platform |
| K3s | K3s | Kubernetes distribution built for IoT and Edge computing | Open source | k3s.io |
| Kanister | Kanister | Framework for app-level data management on Kubernetes | Open source | kanister.io |
| KEDA | KEDA | Kubernetes-based, event-driven autoscaler | Open source | keda.sh |
| Keptn | Keptn | Cloud-native app delivery automation | Open source | keptn.sh |
| Knative | Knative | Serverless and event-driven app development and deployment | Open source | knative.dev |
| Komodor | 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 |
| KubeEdge | KubeEdge | Kubernetes-native edge computing framework | Open source | kubeedge.io |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Kubermatic | KubeOne | Automation of Kubernetes cluster management | By request | kubermatic.com/products/kubermatic-kubeone |
| | Kubernetes Platform | Auomated hybrid and multi-cloud Kubernetes | | kubermatic.com/products/kubermatic-kubernetes-platform |
| Kubernetes | Kubernetes | Container orchestration | Open source | kubernetes.io |
| KubeVirt | KubeVirt | Build a virtualization API for Kubernetes | Open source | kubevirt.io |
| Kuma | Kuma | Service mesh for controlling, securing, and monitoring microservices | Open source | kuma.io |
| Kyverno | Kyverno | Kubernetes-native policy management | Open source | kyverno.io |
| Lacework | Polygraph® Data Platform | Security for DevOps, containers, and cloud environments | By request | lacework.com |
| Linkerd | Linkerd | Service mesh for observability, security, and reliability in Kubernetes | Open source | linkerd.io |
| Loft | vCluster | Lightweight, virtual Kubernetes clusters | Free tier | vcluster.com |
| Logz.io | Open 360 Platform | Full infrastructure and app observability | Trial period | logz.io |
| Longhorn | Longhorn | Cloud-native distributed block storage for Kubernetes | Open source | longhorn.io |
| Mend.io | Mend Platform | App security | By request | mend.io/mend-platform |
| Metalstack.cloud | Metalstack.cloud | Automate bare-metal infrastructure for Kubernetes clusters | By request | metalstack.cloud |
| Mia-Platform | Mia-Platform | Build IDPs, self-serve developers, and ship applications faster | By request | mia-platform.eu |
| Microsoft Azure | Azure Kubernetes Service | Managed Kubernetes platform | Trial period | azure.microsoft.com/en-us/products/kubernetes-service |
| MinIO | MinIO | Object storage solution | By request | min.io |
| Mirantis | Mirantis Container Cloud | Container infrastructure management | By request | mirantis.com/software/mirantis-container-cloud |
| | Mirantis Container Runtime | Secure, industry-standard container runtime | | mirantis.com/software/container-runtime |
| | Mirantis Kubernetes Engine | Container orchestration | Trial period | mirantis.com/software/mirantis-kubernetes-engine |
| | Mirantis Secure Registry | Enterprise container registry | | mirantis.com/software/mirantis-secure-registry |
| NetApp | Astra | Protect, move, and store Kubernetes data | Free tier | netapp.com/cloud-services/astra |
| | Cloud Insights | Continuously monitor and troubleshoot Kubernetes storage infrastructure | Trial period | netapp.com/cloud-services/kubernetes |
| | Spot CloudCheckr | Cloud cost management platform | By request | spot.io/product/cloudcheckr |
| New Relic | Pixie | Kubernetes observability | Free tier | newrelic.com/platform/kubernetes-pixie |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Ngrok | Ngrok | Globally distributed reverse proxy | Free tier | ngrok.com |
| Nirmata | Nirmata Enterprise for Kyverno | Policy-based security, compliance, and governance | Free tier | nirmata.com/nirmata-enterprise-for-kyverno |
| | Nirmata Policy Manager | Security, automation, and governance for Kubernetes | | nirmata.com/policy-manager |
| Notary | Notary | Specs and tools for software supply chain security | Open source | notaryproject.dev |
| Nutanix | Kubernetes Engine | Kubernetes management | By request | nutanix.com/products/kubernetes-engine |
| | Nutanix Kubernetes Platform | Kubernetes management for platform engineering | Trial period | nutanix.com/products/kubernetes-management-platform |
| Octopus Deploy | Octopus Deploy | DevOps automation | Trial period | octopus.com |
| Okteto | Okteto | Developer environment automation | Free tier | okteto.com |
| Open Policy Agent | Open Policy Agent | Policy engine | Open source | openpolicyagent.org |
| Operator Framework | Operator Framework | Toolkit to manage Kubernetes native apps | Open source | operatorframework.io |
| Oracle | Cloud Infrastructure (OCI) | Scalable cloud services for enterprise apps | Free tier | oracle.com/cloud |
| OVHcloud | Managed Kubernetes | Container orchestration | Free | us.ovhcloud.com/public-cloud/kubernetes |
| Palo Alto | Prisma Cloud | CNAPP | By request | paloaltonetworks.com/prisma/cloud |
| Platform9 | Platform9 Managed Kubernetes (PMK) | Kubernetes for AI/ML projects for on-prem and colocation infrastructure | By request | platform9.com/managed-kubernetes |
| Portainer | Portainer | Container management | Free tier | portainer.io |
| Portworx | Portworx Platform | Container data management | Free tier | portworx.com/platform |
| Prometheus | Prometheus | Cloud-native monitoring and alerting platform | Open source | prometheus.io |
| Rafay | Cloud-Native Platform | Automation for Kubernetes operations, security, and app delivery | Trial period | rafay.co/platform/accelerate-cloud-native-adoption |
| Rancher | Rancher | Kubernetes management platform | Free tier | rancher.com |
| Red Hat | Ansible Automation Platform | IT automation | By request | ansible.com |
| | OpenShift | Build, modernize, and deploy apps at scale | Trial period | redhat.com/en/technologies/cloud-computing/openshift |
| | Quay | Container registry | | quay.io |
| | Quay Clair | Vulnerability static analysis for containers | Open source | github.com/quay/clair |
| Redapt | Redapt | End-to-end integration | By request | redapt.com |
| Release Technologies | Release Delivery | Container-based EaaS platform | By request | prod.releasehub.com/product/release-delivery |
| Replicated | Replicated Platform | Kubernetes app delivery and management | Trial period | replicated.com |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Rook | Rook | Cloud-native storage for Kubernetes | Open source | rook.io |
| ScaleOps | ScaleOps | Auomated Kubernetes resource optimization | Trial period | scaleops.com |
| Scaleway | Container Registry | Docker repository | Free tier | scaleway.com/en/container-registry |
| | Kubernetes Kapsule | Managed Kubernetes | | scaleway.com/en/kubernetes-kapsule |
| | Kubernetes Kosmos | Pod launch and Kubernetes workload management | By request | scaleway.com/en/kubernetes-kosmos |
| Snyk | 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/products/gloo-platform |
| SPIFFE | SPIFFE | Universal identity control plane for distributed systems | Open source | spiffe.io |
| | SPIRE | API toolchain for building trust between software systems | | github.com/spiffe/spire |
| Splunk | Cloud Platform | Cloud-based data analytics for real-time analysis | Trial period | splunk.com/en_us/products/splunk-cloud-platform.html |
| Stackwatch | Kubecost | Kubernetes cost management and monitoring | Free tier | kubecost.com |
| Sumo Logic | Sumo Logic | SaaS log analytics platform | Free tier | sumologic.com |
| SUSE | NeuVector Prime | Full lifecycle container security | Free tier | suse.com/products/neuvector |
| Sysdig | Sysdig Monitor | Kubernetes and cloud monitoring with a managed Prometheus service | By request | sysdig.com/products/monitor |
| Teleport | Teleport Access Platform | Scalable Kubernetes RBAC | Trial period | goteleport.com/kubernetes-access |
| TensorFlow | TensorFlow | Create production-grade ML models | Open source | tensorflow.org |
| The Linux Foundation | PyTorch | Optimized tensor library for deep learning | Open source | pytorch.org |
| Tigera | Calico Cloud | Security for containers and Kubernetes | Trial period | tigera.io/tigera-products/calico-cloud |
| | Calico Enterprise | Kubernetes network security and observability | | tigera.io/tigera-products/calico-enterprise |
| | Calico Open Source | Networking and security for containers and Kubernetes | Open source | tigera.io/tigera-products/calico |
| Traefik Labs | Traefik Enterprise | Unified API gateway and Ingress | Trial period | traefik.io/traefik-enterprise |
| | Traefik Hub | Kubernetes-native API management | | traefik.io/traefik-hub |
| | Traefik Proxy | Cloud-native application proxy | Open source | traefik.io/traefik |
| Trilio | Kubernetes Backup | Kubernetes backup and recovery for on-prem or cloud | By request | trilio.io/products/kubernetes-backup-and-recovery |
| Upwind | Upwind Cloud Security Platform | CNAPP | By request | upwind.io |

**DZONE'S 2024 KUBERNETES IN THE ENTERPRISE SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Veeam | Kubestr | Discover, validate, and evaluate Kubernetes storage options | Open source | kubestr.io |
| | Veeam Kasten | Kubernetes data protection and mobility | Sandbox | veeam.com/products/cloud/kubernetes-data-protection.html |
| Vitess | Vitess | Deploy, scale, and manage large clusters of MySQL database instances | Open source | vitess.io |
| VMWare Tanzu | Kubernetes Grid Integrated | Kubernetes management | By request | tanzu.vmware.com/kubernetes-grid |
| | Tanzu Platform | Accelerate delivery of apps | | tanzu.vmware.com/platform |
| Volcano | Volcano | Cloud-native batch scheduling system | Open source | volcano.sh |
| Windocks | Windocks | Protect and provision structured and unstructured data | Free tier | windocks.com |
| Wiz | Wiz CNAPP | Unified cloud security from prevention to detection | By request | wiz.io/product |
| Zesty | Commitment Manager | Automated AWS discount management | By request | zesty.co/commitment-manager |
| | Zesty Disk | Autoscale EBS volumes | | zesty.co/zesty-disk |

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