THE NEW STACK

KUBERNETES DEPLOYMENT & SECURITY PATTERNS

The New Stack
Kubernetes Deployment & Security Patterns
Alex Williams, Founder & Editor-in-Chief

Core Team:
Bailey Math, AV Engineer
Benjamin Ball, Marketing Director
Gabriel H. Dinh, Executive Producer
Judy Williams, Copy Editor
Kiran Oliver, Podcast Producer
Krishnan Subramanian, Technical Editor
Lawrence Hecht, Research Director
Libby Clark, Editorial Director
Norris Deajon, AV Engineer

20180622

TABLE OF CONTENTS

KUBERNETES DEPLOYMENT & SECURITY PATTERNS

INTRODUCTION

Kubernetes is one of the largest open source projects in the world, according to data from GitHub. It's so big that the tools to manage the development and deployment of Kubernetes are constantly catching up to the momentum behind the open source technology.

This continual evolution makes Kubernetes deployment a bit of an unsteady, fast-moving target. Still, the Kubernetes movement is the center of attention for organizations at the leading edge of technology innovation and adoption. Container technologies remain of great importance, but now the deepest issues are about scaling containers in orchestration environments. Containers are considered in context with Kubernetes. There is no other standard to speak of that can support the market scale that will be needed for containers to be used in production. The only standard is Kubernetes. Others are supporters of the technology, but only Kubernetes has enough wind behind it to steer the cloud-native technology market.

From this context, we present the second ebook in our series about Kubernetes. The market is now beyond the wonder of containers. It's beyond the early fascination with distributed architectures that may be used across multiple cloud platforms. Even the Kubernetes technology itself is getting boring, despite the fast pace of change. That's a welcome sign for an early market primed for its next big test. The big question is now about the technology's maturity: How well does Kubernetes work in production? We still don't know. It's a question that cannot be resolved quickly. And until it's resolved, we won't know how much of an impact Kubernetes will truly have.

In its infancy, Kubernetes grew more than most any open source project ever has. The project started at Google and was open-sourced in 2014 under this new vision of cloud-native infrastructure. Since then, numerous companies have joined the Cloud Native Computing Foundation, home of the Kubernetes open source project. They have contributed greatly to the platform, helping to define it and show the larger IT market that a multi-cloud infrastructure has considerable value compared to the alternative. There is no single provider, and hopefully there never will be. For Kubernetes, a lot depends on how the infrastructure is developed. It can't be built all at once. The work will take years.

The project has now passed its early development and is in its early adolescence. This transition has us thinking less about defining Kubernetes and more so about what needs to be developed in order for the technology to be viable in production. Success will be determined by the overall direction of the Kubernetes community. Of central importance is finding ways to make the community more inclusive of new voices and contributions. The community must gain more trust with users while patiently developing the orchestration project's core. It's a values question at its heart: How contributors are directed by the values, vision and objectives set by the most senior community leaders will play an increasingly important part in how well the multitude of projects and special-interest groups actually fare and participate in Kubernetes' overall development. The leaders have so far been outstanding in their work. It's time to build on the work they have already done.

How Kubernetes proves resilient to security threats will also serve as a test of the platform's longevity. Kubernetes deployment patterns that prioritize security will lead the way toward faster integration of container infrastructure and determine at what rate Kubernetes adoption will occur. Once customers have confidence in the security of Kubernetes deployments, it will manifest in the overall level of production across the market.

Security has to be baked into all deployments. But in a distributed, highly scalable environment, typical security patterns will not suffice. It requires an understanding of security in the context of Kubernetes. It is critical for operations teams to understand Kubernetes security in terms of containers, deployment and network security. Perimeters are now porous, making traditional security methodologies less effective. Containers must be secured at the node level, but also through the image and registry. This means a lot of new learning will be needed for operations teams developing and managing Kubernetes infrastructure. Security practices in the context of various deployment models will be a challenge for companies and will require particular attention.

Deployment pattern complexity decreases as the abstraction moves towards the development layer. Security requirements change depending upon the underlying infrastructure and the patterns used for deployment. Thus, understanding security responsibilities and the role of operations in various deployment patterns is of utmost importance for a successful roll out.

This book aims to provide explanation and analysis about container orchestration and security patterns for operations teams as they transition from a world of virtual machines to containers. How companies fare in the transition will depend on how effectively the Kubernetes community can work together to strengthen the technology's core.

Thanks, Alex.

Alex Williams
Founder and Editor-in-Chief
The New Stack

SPONSORS

We are grateful for the support of our ebook foundation sponsor:

KubeCon | CloudNativeCon

And our sponsors for this ebook:

alcide

aqua

Twistlock

WHAT THE DATA SAYS ABOUT KUBERNETES DEPLOYMENTS

by **LAWRENCE HECHT**

The considerable growth in the Kubernetes market is well documented. It is by far the most widely used orchestration platform, but it's not the only one, preventing it from receiving full default status. Kubernetes' acceptance has forced it to mature quite fast and has left the technology community to rapidly innovate. It has helped force a disruption in the market as new and more established vendors now compete in the cloud-native space.

Container technologies prompted the rise and development of the Kubernetes orchestration platform. Today, the largest users of containers are companies with more than 1,000 employees which run their own data centers. These companies are also the largest users of Kubernetes in production — a compelling reminder of the market forces driving the project's development and adoption. But these trends only tell part of the story.

The rest of the story is a bit more complex. The transition to an application-oriented architecture has just begun, and many forces in the market will affect how we perceive this shift. They encompass the various types of workloads that an organization deploys, the size of the organization and the breakdown of how users and vendors are each developing cloud-native architectures for larger market consumption.

Developers are finding containers transforming, adopting them at such a scale that it becomes a complex process to understand how usage is affecting the overall market. Data from our own research and a recent survey by the Cloud Native Computing Foundation (CNCF) offers some indication of the successes and challenges Kubernetes users encounter, which in turn can illuminate the broader ecosystem shifts we are seeing today. In the CNCF's fall 2017 survey, 764 respondents were recruited directly through outreach to CNCF participants, their social networks and a larger community of cloud-native-leaning companies. The early results of the survey, with 577 respondents, were published in a December 2017 blog post. Since then, CNCF received an additional 187 responses from a questionnaire that was translated into Mandarin. Almost all (97 percent) respondents were using containers in some way, while 61 percent were using containers in production. Overall, 69 percent of respondents said they were using Kubernetes to manage containers.

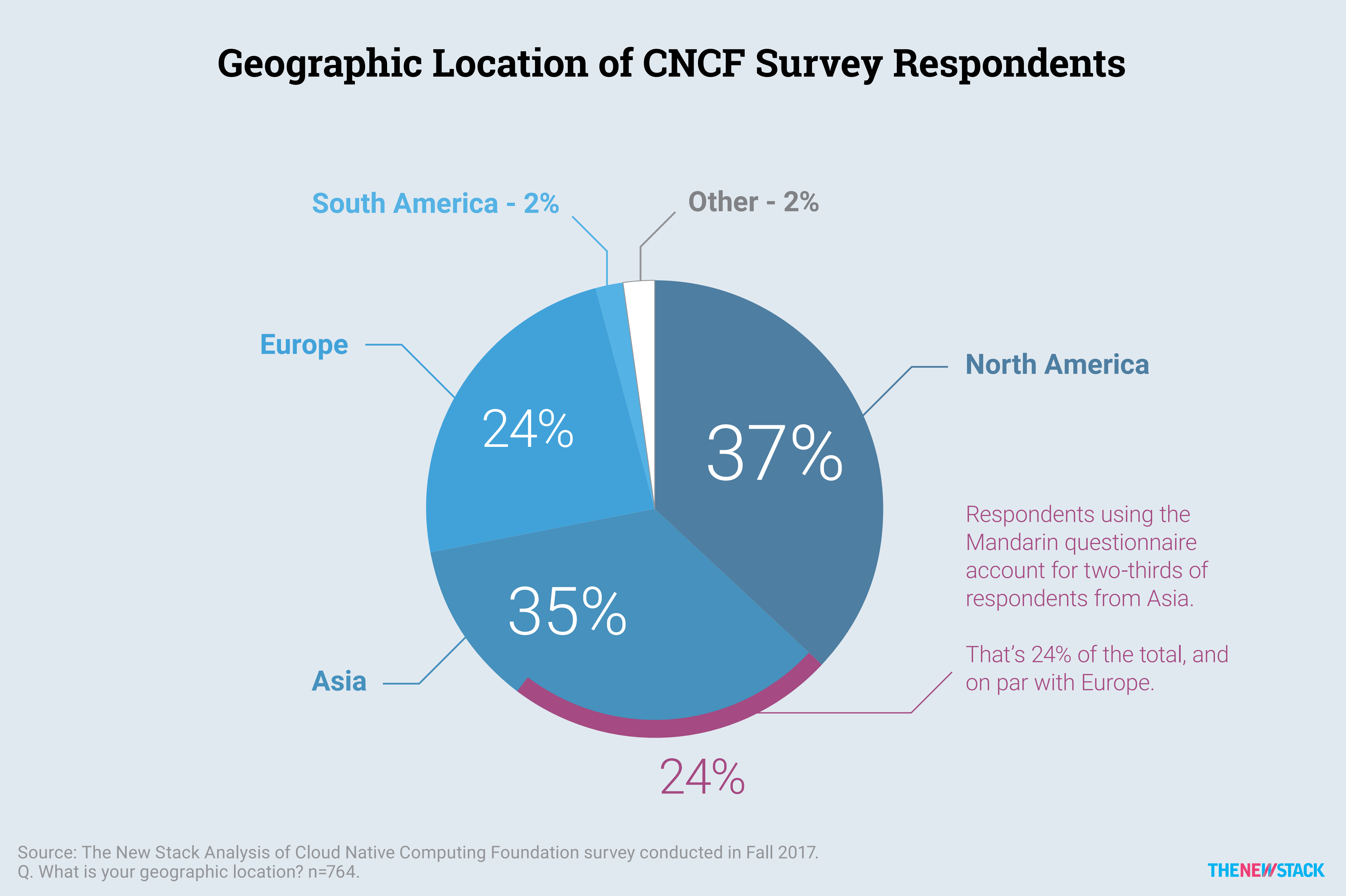
In addition to the CNCF survey, we also cite The New Stack's own study originally included in "The State of the Kubernetes Ecosystem." Based on responses collected in May 2017 from 470 individuals at organizations using containers, the findings focused on the 62 percent of respondents that were using Kubernetes in production.

Methodology and Container Adoption

Our analysis focuses primarily on an independent review of CNCF's survey data. Not only is it the most recent data available, but it also asked in-depth questions about topics The New Stack's May 2017 survey did not cover. Although participant recruitment was not based on a random sample, it represents a well-balanced cross-section of the IT community that would be interested in using Kubernetes. For example, 30 percent of respondents hold a DevOps or site reliability engineer (SRE) role and 42 percent have a developer or development management role. Technology companies, including those involved with container or cloud solutions, represent 53 percent of all respondents. Although this dwarves their position in the overall economy, it may be representative of Kubernetes-using companies. For most of the study's results, the size, rather than the industry of an organization, had a more significant impact. Only 22 percent of respondents work in organizations with less than 50 employees, while 27 percent are affiliated with those employing more than 5,000 employees. Throughout this chapter, we take these demographics into account when analyzing the data.

Administering the survey in Mandarin meant that, unlike other surveys, CNCF's was not dominated by respondents from North America. Respondents from Asia and Europe represented 59 percent of the sample. Due to the survey's translation into Mandarin, the Asian sample was tilted towards China as opposed to India or Japan. Although the survey questions were identical, the data had to be transformed because of slight variations in how the research instruments were programmed. In addition, the specific responses for "other, please specify" options were not translated from Mandarin to English. The data file used for this chapter is available here.

Respondents to the Mandarin-translated survey are, in general, less far along in their deployment of containers and Kubernetes. As mentioned earlier, 97 percent of the sample use containers to some degree, and 61 percent do so in production environments. That figure drops to 32 percent in production for the Mandarin language respondents.



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. What is your geographic location? n=764.

FIG 1.1: *Sixty-three percent of respondents came from outside of North America.*

The New Stack believes that although China's adoption may be several months behind compared to its Western counterparts, differences also arose for two other reasons. First, the Mandarin sample was much less weighted towards tech companies, with only 39 percent of respondents working in the tech sector compared to 58 percent for the rest of the sample. Second, the English questionnaire may have been completed more by early adopters that have been regularly attending CNCF and Kubernetes conferences. In this context, we are again reminded that KubeCon attendees are generally ahead of the curve compared to the rest of the world.

Key Kubernetes Deployment Data Points

- Sixty-nine percent of organizations surveyed by CNCF use Kubernetes to manage containers. However, Kubernetes is not the only orchestration method. Nearly two-thirds of Kubernetes users still utilize another method to manage containers.

- Most users are deploying Kubernetes to a public cloud. Eighty-three percent of Kubernetes-using organizations deploy it to at least one public cloud.
- Although vendor-provided Kubernetes is becoming more common, 91 percent of deployments are handled internally.
- Security is the top container-related challenge among organizations using Kubernetes. However, storage is the top challenge among organizations that only deploy Kubernetes to on-premises servers. Monitoring is the top challenge among those that only deploy Kubernetes to public clouds.
- The more containers an organization uses, the more likely they are to use Kubernetes. The number of containers being run changes the need for container orchestration. While only 12 percent of total respondents said the organizations they work for run more than 20 Kubernetes clusters, that number jumps to 35 percent for respondents whose organizations run more than 1,000 containers.
- While NGINX is the leading Kubernetes ingress provider, HAProxy rivals it among organizations with six or more clusters.

Kubernetes Overview

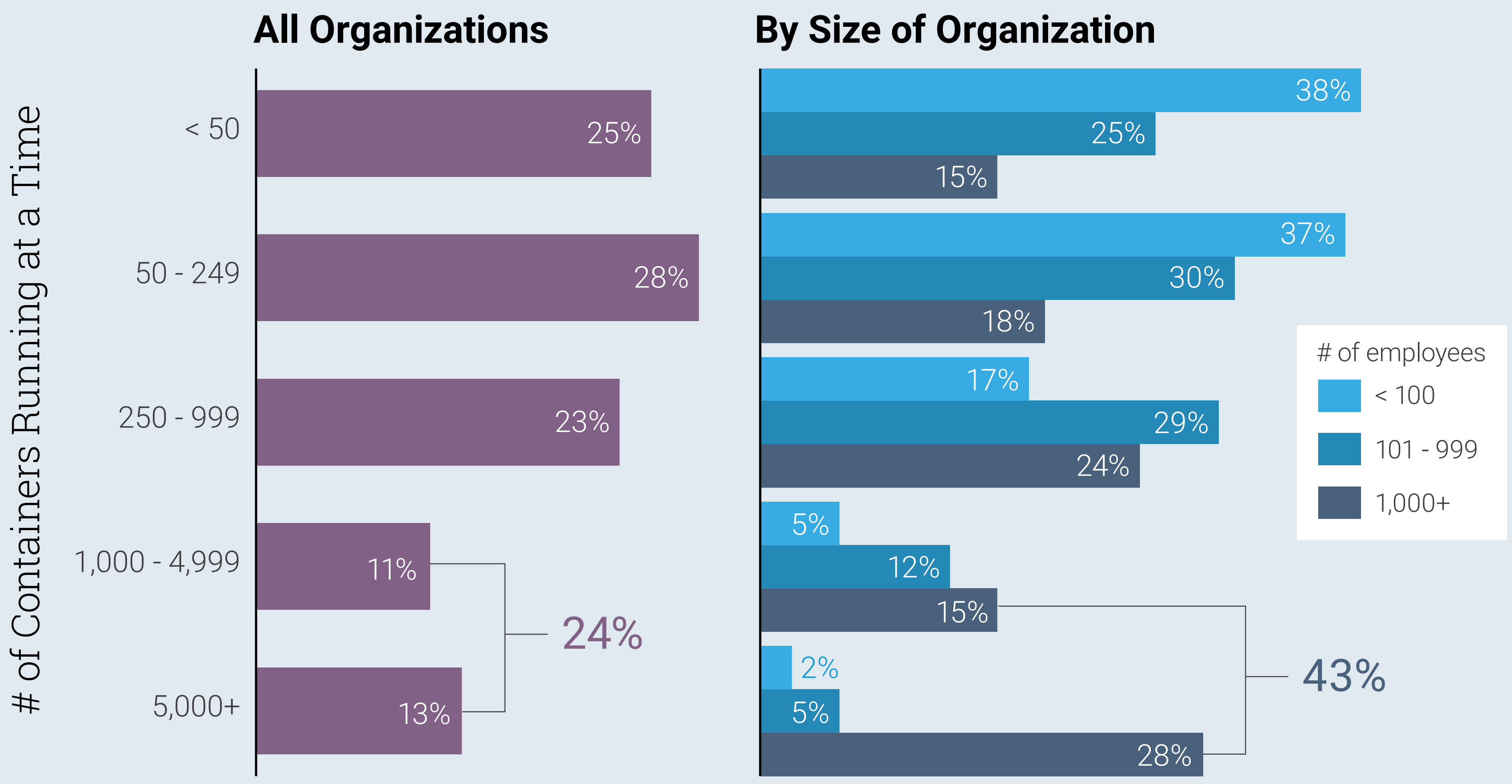
Over the last two years, surveys have shown that Kubernetes has a wide lead over competitive offerings. At a high level, Kubernetes won the first battle of the container orchestration wars. Companies with competitive offerings, such as Docker and Mesosphere, now promote how their products interoperate with Kubernetes. The major cloud providers have followed suit, with Alibaba Cloud, Amazon Web Services (AWS), Google Cloud Platform, Huawei Cloud and Microsoft Azure offering services to manage Kubernetes environments.

Today, Kubernetes is the leading choice for managing containers at scale, but that does not mean it will remain so. Kubernetes deployments have made a lot of progress over the last few years, moving from experiments to managing production workloads. Yet most Kubernetes deployments are still young and relatively small. Kubernetes' central spot in IT ecosystems is not guaranteed. Will Kubernetes become a niche technology, specialized in orchestrating the resources to deploy infrastructure at scale? Will developers move to platforms running on containers that are differentiated on factors beyond whether or not Kubernetes is inside?

This chapter does not predict the future. Nor does it pretend to report on the percentage of enterprises that have adopted Kubernetes worldwide. Instead, it describes the recent past, with a focus on organizations that use containers and have started adopting Kubernetes. Relying on two surveys of respondents who primarily work for container-using organizations, this analysis will help readers gain perspective on their own Kubernetes deployments.

Storage Matters for Large Organizations

Storage and networking technologies are pillars of data center infrastructure, but were designed originally for client/server and virtualized environments. Container technologies are leading companies to rethink how storage and networking technologies should be architected in a data center environment. We once thought about configuring the machine with storage and networking. Now it's a different way of thinking as architectures become more application-oriented and



24% of Organizations Run 1,000+ Containers at a Time. That Percentage Jumps to 43% at Orgs. With 1,000+ Employees

Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017. Q. How many containers does your organization typically run? n=748; < 100 Employees, n=251; 100-999 Employees, n=212; 1,000+ Employees, n=285.

THENEWSTACK

FIG 1.2: *Larger organizations have more containers running because they have more workloads.*

storage doesn't necessarily live on the same machine as the application or its services.

Larger companies tend to run more containers, and to do so in scaled-out production environments that may require a new approach to infrastructure. Twenty-eight percent of organizations with more than 1,000 employees are running more than 5,000 containers at a time, while only four percent of the other organizations are running at such volume. And 81 percent of large organizations with more than 1,000 containers say they are running containers in production. This speaks to the fact that large organizations by their very nature usually have a lot of workloads. On the flip side, 38 percent of small organizations (< 100 employees) are running fewer than 50 containers versus only 15 percent of the largest organizations.

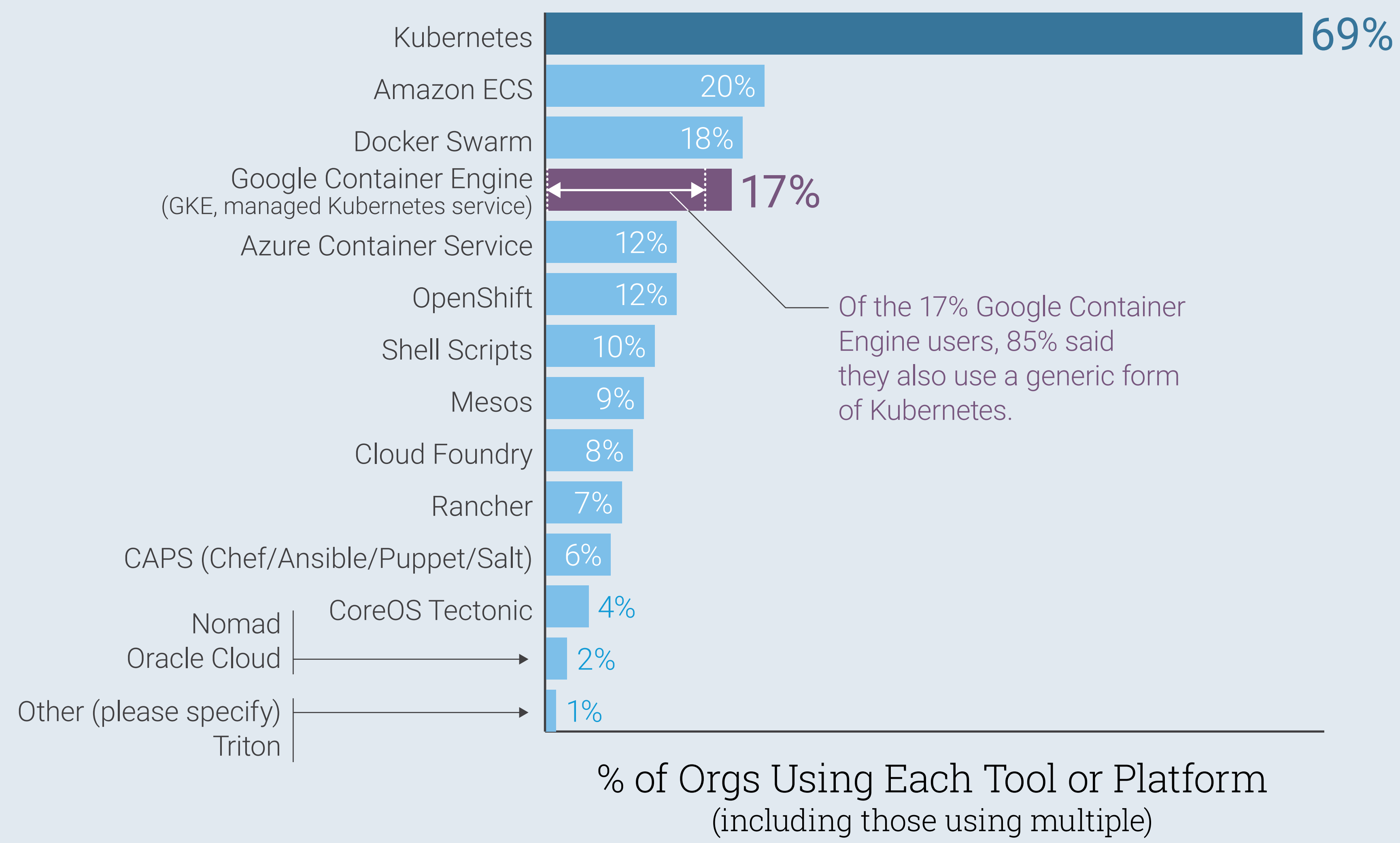
Kubernetes Adoption and Cloud Deployments

CNCF provided a partial list of its projects (e.g., gRPC, Kubernetes, OpenTracing, Prometheus) and asked in their survey if these cloud-native technologies were being used or evaluated. In those responses, overall, 74 percent said Kubernetes is a cloud-native project they are using.

When asked in a separate question about how their organization manages containers, 69 percent mentioned Kubernetes. Using more containers most likely means the user will deploy with Kubernetes.The percent of respondents using Kubernetes increases especially when containers are deployed in higher volumes. For example, about 81 percent of respondents who run 1,000 or more containers say they use Kubernetes.

There are some findings that show uses for Kubernetes without containers. Interestingly, 15 percent of organizations that use the Kubernetes project in production do not manage containers with it. Some of these respondents, perhaps, use a platform or vendor-provided tools that incorporate Kubernetes technology in a bundled solution. This viewpoint is based on the fact that customers may be using any combination of container management platforms or infrastructure. It largely depends on their workloads and the infrastructure they use to run microservices and composed applications. Although the distinction is somewhat arbitrary, it appears that some people believe that using an open source project means that you are personally deploying the source code. Consequently, for the rest of this report, the term “Kubernetes user” will refer to those that use the orchestration platform to manage containers, rather than those that said they use the project itself.

Sixty-three percent of people who work in organizations that use



Kubernetes Manages Containers at 69% of Organizations Surveyed

Of the 17% Google Container Engine users, 85% said they also use a generic form of Kubernetes.

Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Your organization manages containers with... (check all that apply)? n=763.

FIG 1.3: *Kubernetes is the most common tool for container management.*

Kubernetes name at least one other tool or method they also use to manage containers.

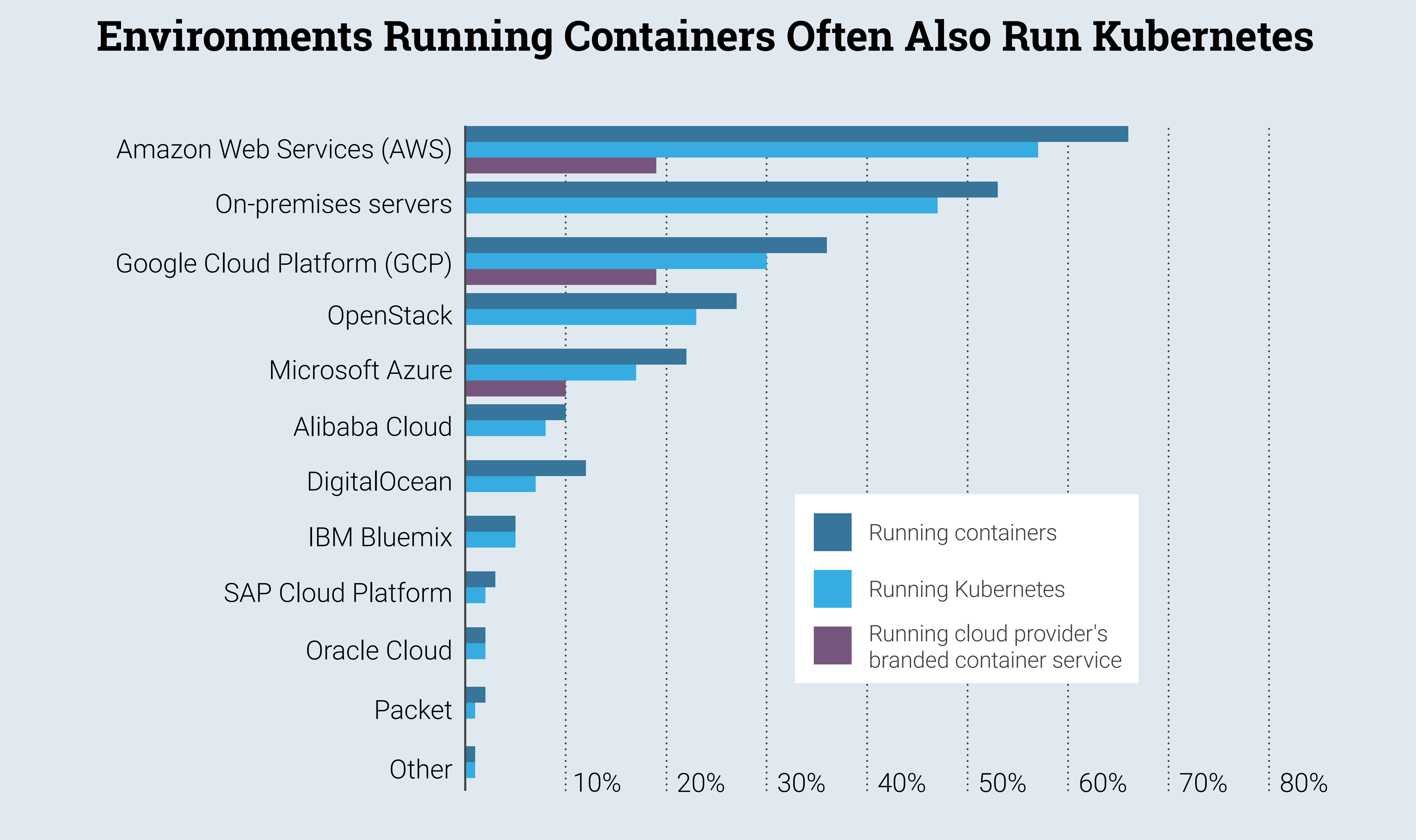
Using a particular cloud environment influenced users' Kubernetes deployments:

- Sixty-seven percent of companies that use Kubernetes say they deploy containers to AWS. The numbers drop to 57 percent for those on AWS who actually deploy Kubernetes. Nineteen percent said they were also using AWS Elastic Container Service (ECS) to manage containers.
- Microsoft Azure and Google Cloud Platform users are similar to AWS customers in their usage pattern.
- A relatively low percentage of customers have adopted their cloud provider's branded container services. Instead, many of these

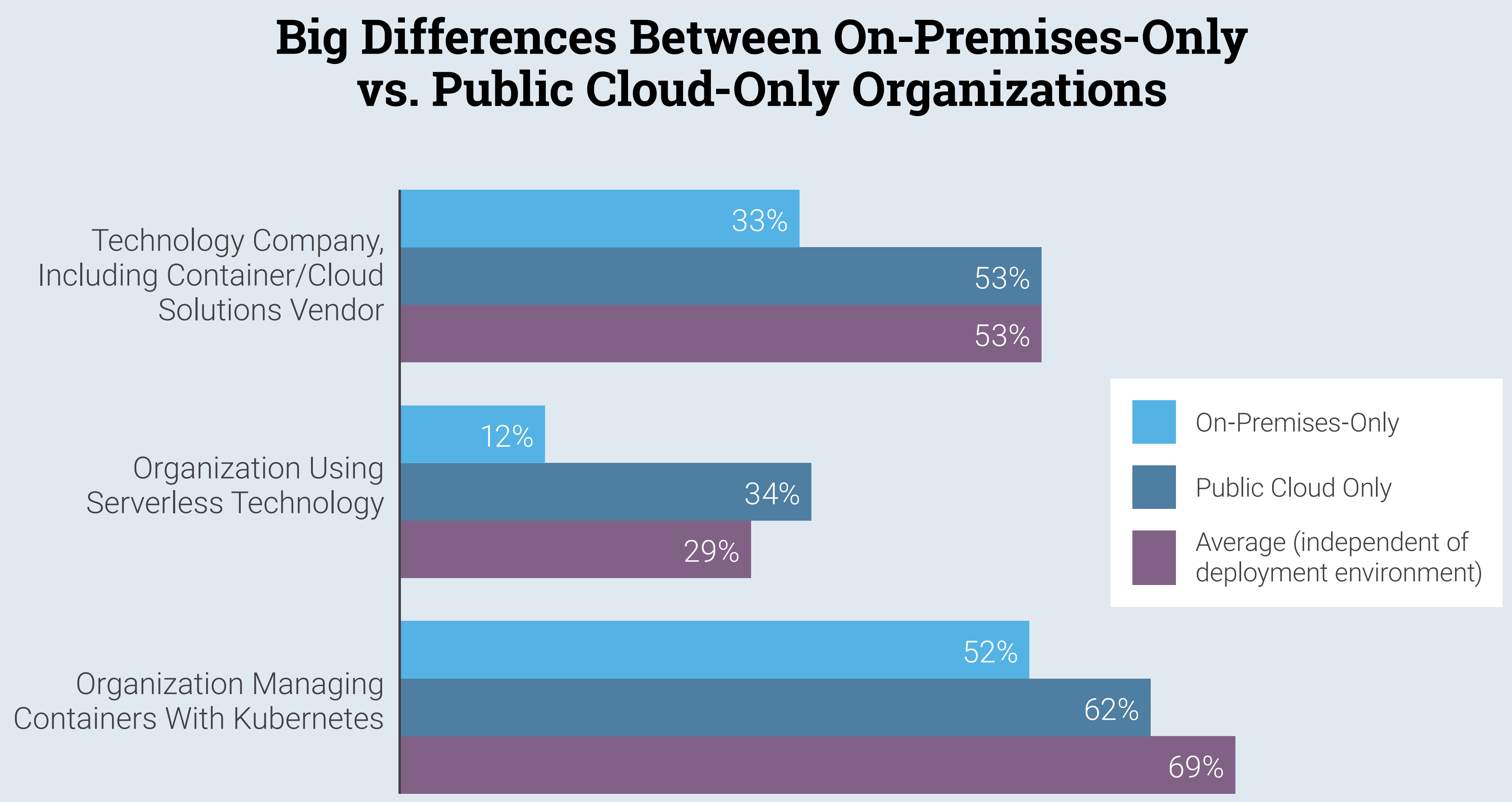
organizations were deploying a Kubernetes distribution directly onto the cloud provider's infrastructure.

The more employees in an organization or the more containers that are running, the higher the likelihood that Kubernetes is being deployed to on-premises servers. Many organizations are using multi-cloud environments. These customers are making a conscious decision to run workloads in different environments based on security, price and performance considerations. There is little evidence that these factors are instrumental in the decision regarding where Kubernetes is actually deployed. Simply, it's more a factor of workloads and the infrastructure chosen to run Kubernetes. Larger companies run lots of containers on-premises, but they may also use cloud services for managing containers.

FIG 1.4: *People will do their own Kubernetes deployments on cloud services, foregoing the branded offering from the cloud provider.*



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Your company/organization deploys containers to which of the following environments? (check all that apply). n=527.
Q. Your company/organization runs Kubernetes to which of the following environments? (check all that apply). n=527.



Big Differences Between On-Premises-Only vs. Public Cloud-Only Organizations

Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Your company/organization deploys containers to which of the following environments? (check all that apply).
Containers Deployed Only to On-premise Servers, n=90; Containers Deployed Only to Public Cloud, n=298.
Q. What industry does your company/organization belong to?
Q. Your organization manages containers with... (check all that apply)?
Containers Only On-premise Servers, n=90; Containers Deployed, but Only to Public Cloud, n=297.
Q. Is your organization using serverless technology?
Containers Deployed Only to On-premise Servers n=89; Containers Deployed Only to Public Cloud, n=293.

THENEWSTACK

FIG 1.5: *Organizations manage containers according to workloads and available infrastructure.*

Organizations use multi-cloud environments three-quarters of the time. The usage is a combination of public, private and on-premises services. Organizations exclusively using cloud services are most likely to be technology companies. Serverless technology adoption among cloud-only organizations is also about three times that of companies that only deploy containers on-premises. And Kubernetes use increases considerably among organizations that deploy containers to multiple types of clouds.

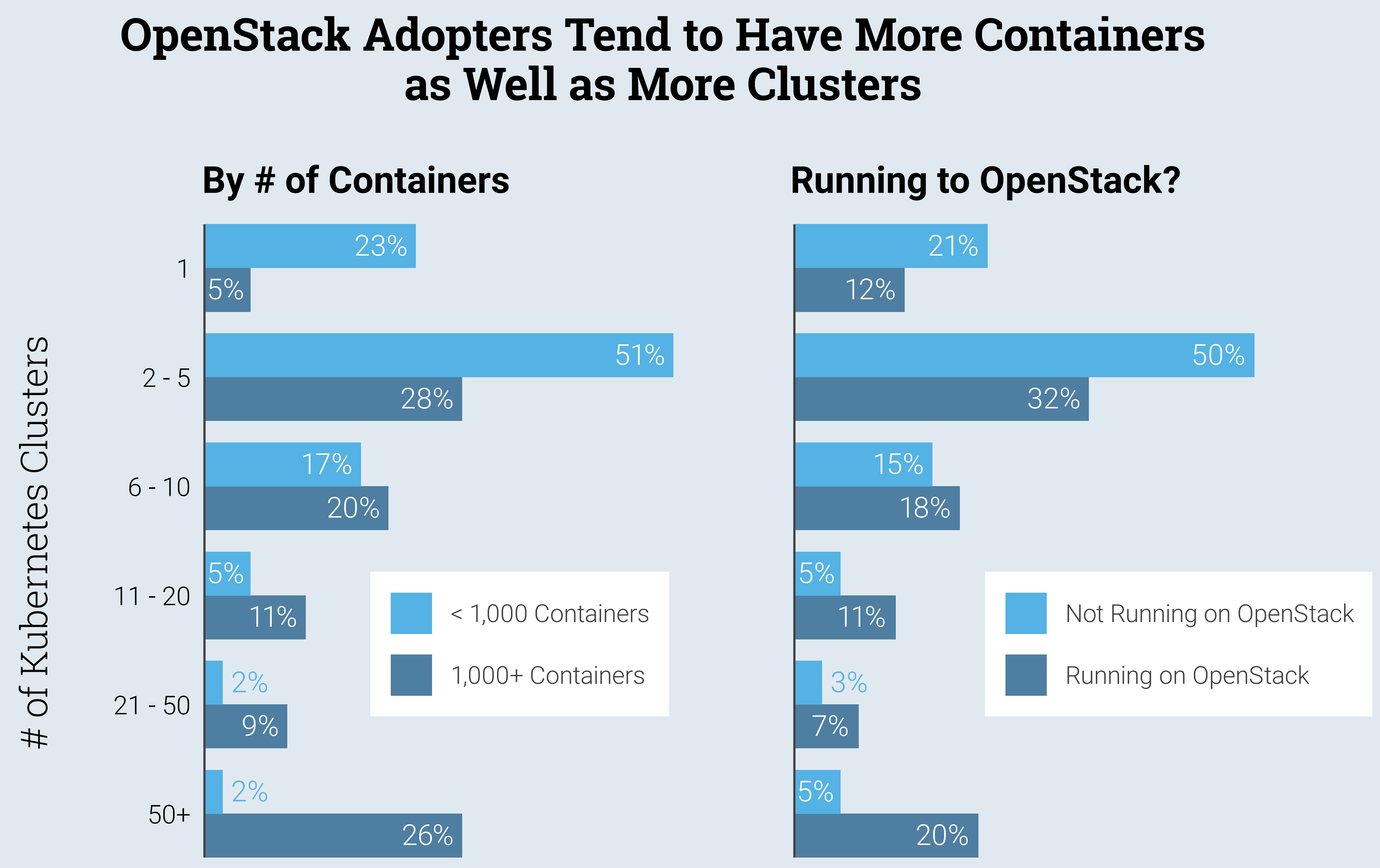
Size of Deployments — Clusters

Most organizations run far fewer than 20 clusters. Running containers at scale is largely limited to companies with on-premises deployments, cloud service providers and organizations using cloud services. In summary, the stark difference in container usage is most apparent when companies are running more than 1,000 containers.

It's a multi-faceted matter: Container usage is so widespread that understanding deployment can become quite nuanced. Analysis shows how deeply Kubernetes is being used across multiple types of workloads and infrastructure. Gaining an understanding of deployment becomes a matter of analyzing the workloads and the infrastructure where the services are running.

In one respect, container users may be deploying on cloud services and on their own infrastructure. Organizations using Kubernetes may also be using it in a limited manner on cloud services, but not their own infrastructure. Then again, they may also be running containers exclusively on their own infrastructure. Cloud services, arguably, stand at the center of the market, by hosting containers for customers while simultaneously building out their own container environments.

FIG 1.6: *Seventy-four percent of organizations with less than 1,000 containers running have five or fewer Kubernetes clusters.*



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. If you use Kubernetes, how many production clusters do you have?
< 1,000 Containers, n=336; 1,000+ Containers, n=130. Kubernetes Not Running on OpenStack, n=338; Kubernetes Running on OpenStack, n=111.

OpenStack users running Kubernetes are primarily large organizations that run 1,000 or more containers. It is noteworthy that the Mandarin language study participants were more likely than others to be both running large deployments within private data centers and running OpenStack.

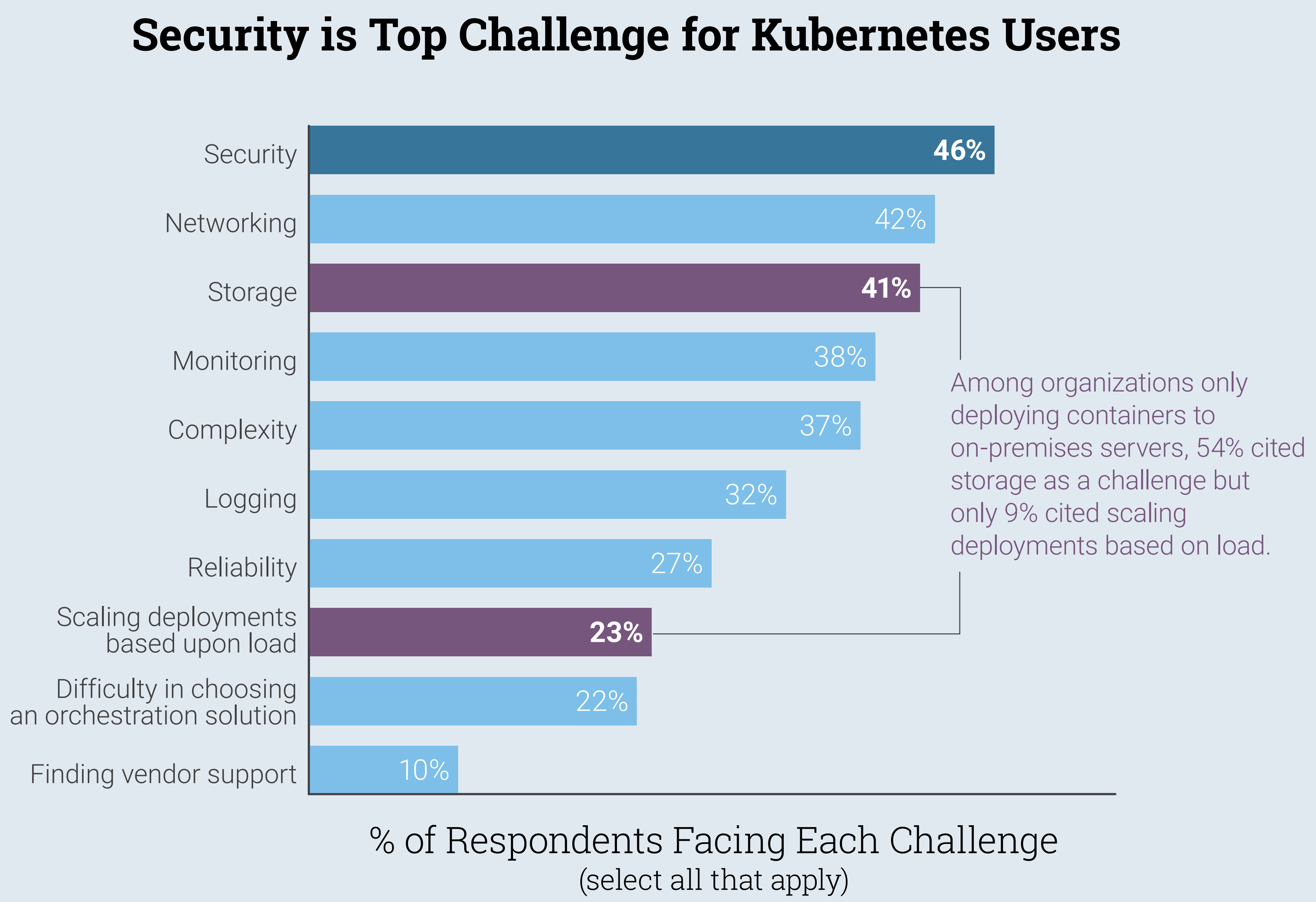
Challenges

People face a wide range of problems when using or deploying Kubernetes. While some challenges are unique to Kubernetes, many others are typical of the growing pains seen with the adoption of many technologies. "The State of the Kubernetes Ecosystem" reported on both the importance of different criteria in picking a container orchestration solution and the major factors inhibiting the adoption of Kubernetes. Scaling was more likely to be an essential requirement for an orchestration solution compared to criteria such as security or resource optimization. Among the biggest challenges mentioned was the fact that using Kubernetes often necessitated changes in the roles or responsibilities of several parts of the IT organization.

The CNCF survey asked about the challenges people face in using or deploying containers in general. We took those answers and narrowed the focus to just organizations using Kubernetes to manage containers. This provides a way to illustrate the issues facing Kubernetes users.

The results show that complexity — a common criticism of Kubernetes — is only the fifth most cited challenge. In the lead are infrastructure-related challenges. Security was cited by 46 percent of Kubernetes users, with networking and storage coming in second and third place.

Twenty-three percent said scaling deployments based on load is a challenge. This likely means that many requirements have been met, with Kubernetes actually helping with scaling as it is supposed to do. At the



Among organizations only deploying containers to on-premises servers, 54% cited storage as a challenge but only 9% cited scaling deployments based on load.

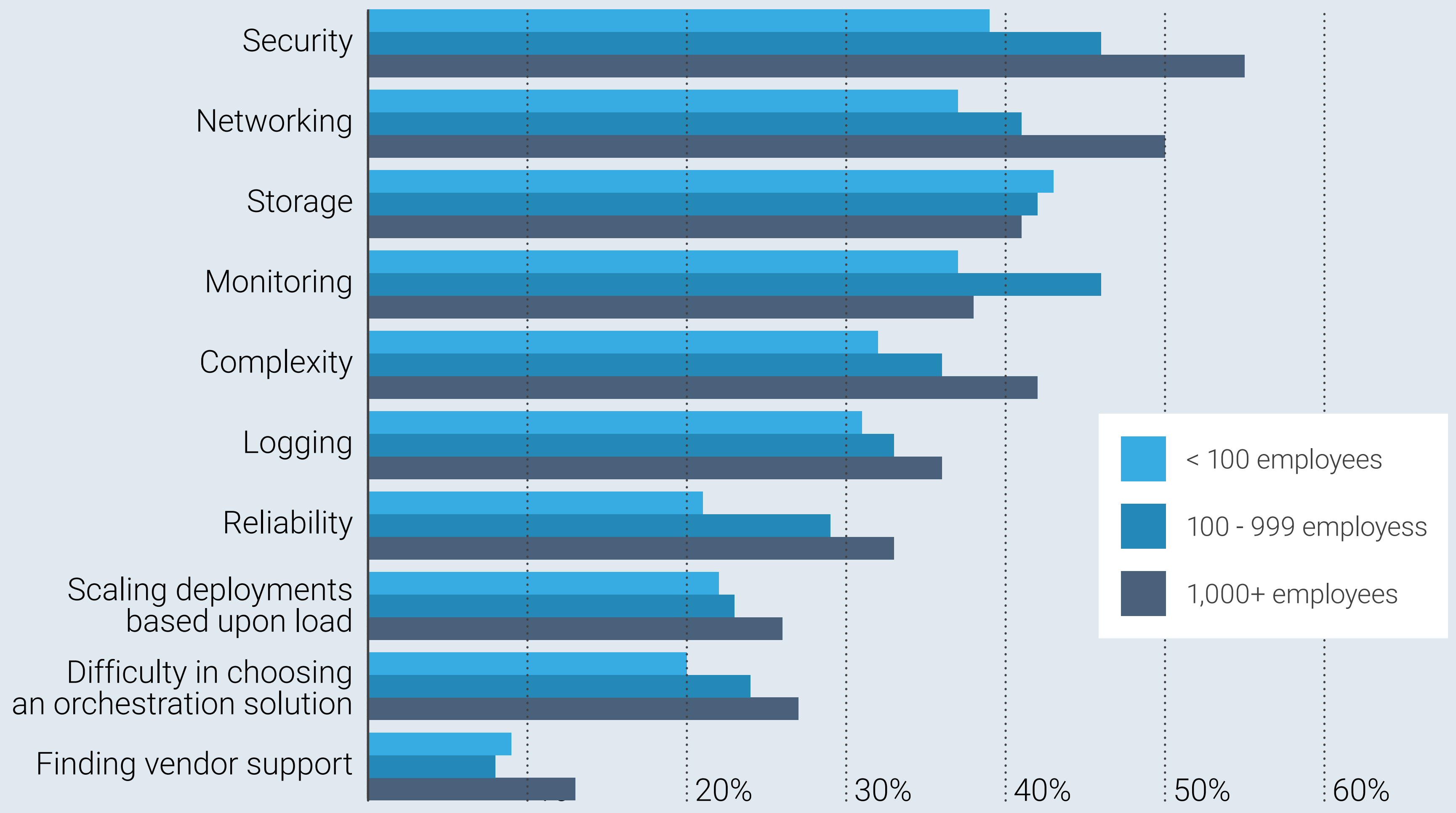
Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017. Q. What are your challenges in using/deploying containers? (check all that apply). n=527.Note, only respondents managing containers with Kubernetes were included in the chart.

FIG 1.7: *More than 40 percent say that security, networking and storage are container-related challenges.*

bottom of the list, 10 percent mentioned problems getting vendor support. One reason there are few complaints about vendor support for Kubernetes is that many deployments are not dependent on a vendor's distribution. Looking forward, there is a high likelihood that high-quality services will be available because the CNCF has recently introduced the Kubernetes Certified Service Provider program to guarantee that service providers meet a certain level of competence.

As in other studies, we found that larger organizations were more likely to cite many issues as challenges they care about. For example, 55 percent of organizations with 1,000 or more employees said security is a challenge, while only 39 percent of organizations with fewer than 100 employees said the same. In this case, as well as with other categories like reliability, it is likely that large enterprises' needs are different than those at smaller

The Larger the Company, the More Likely the Kubernetes User Is to Face Container Challenges



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017. Q. What are your challenges in using/deploying containers? (check all that apply). n=527; < 100 Employees, n=286; 100-999 Employees, n=140; 1,000+ Employees, n=203. Note, only respondents managing containers with Kubernetes were included in the chart

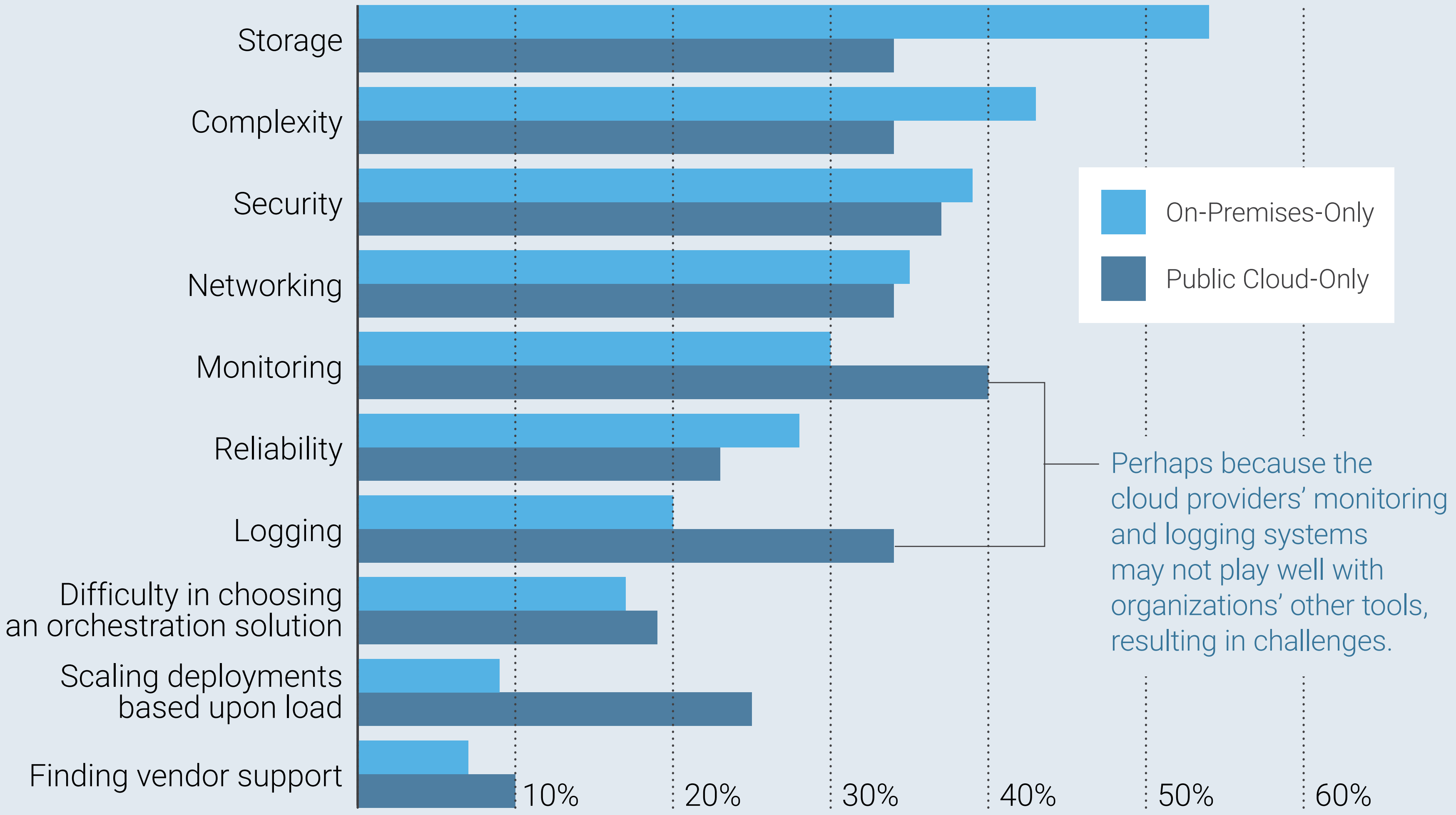
THENEWSTACK

FIG 1.8: *Security and networking are more likely to be cited as a container-related challenge at organizations with 1,000 or more employees.*

organizations. In other areas, such as networking, it is possible that the size and breadth of the IT infrastructure (bandwidth and number of sites) present Kubernetes with more unique challenges as compared to just the number of containers being used. In fact, among organizations with six or more clusters, the percentage citing networking as a challenge jumped from 42 to 53 percent.

A few challenges did not fit the aforementioned pattern. For storage, an explanation may be that the technology “issues” are not based on scalability. In the case of monitoring, midsize companies are more likely to face challenges. As we described previously in the article Rethinking Monitoring for Container Operations, smaller organizations generally have less need to create a formal monitoring process, while larger ones have the resources to create a more robust, customized monitoring

Storage and Complexity Are Bigger Challenges for On-Premises-Only Container Users



Perhaps because the cloud providers' monitoring and logging systems may not play well with organizations' other tools, resulting in challenges.

Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017. Q. What are your challenges in using/deploying containers? (check all that apply). Containers Deployed Only to On-premise Servers n=46; Containers Deployed Only to Public Cloud, n=183. Note, only respondents managing containers with Kubernetes were included in the chart.

THENEWSTACK

FIG 1.9: *Fifty-four percent of on-premises-only container users face storage challenges compared to 34 percent of public cloud-only organizations.*

system. Stuck in the middle are those organizations with 100 to 999 employees.

Another factor that affects an organization's container-related challenges is whether or not they are exclusively deploying containers to a public cloud or to on-premises servers. Among those that just use on-premises servers for containers, storage was the most common challenge. This may be because these organizations manage their own storage infrastructure, possibly even handled by a separate IT team. For organizations only using containers on a public cloud, monitoring and logging were more often cited as a challenge. Though cloud providers are supposed to enable scalability, organizations only using on-premises servers for containers were significantly less likely to say scaling deployments is a challenge.

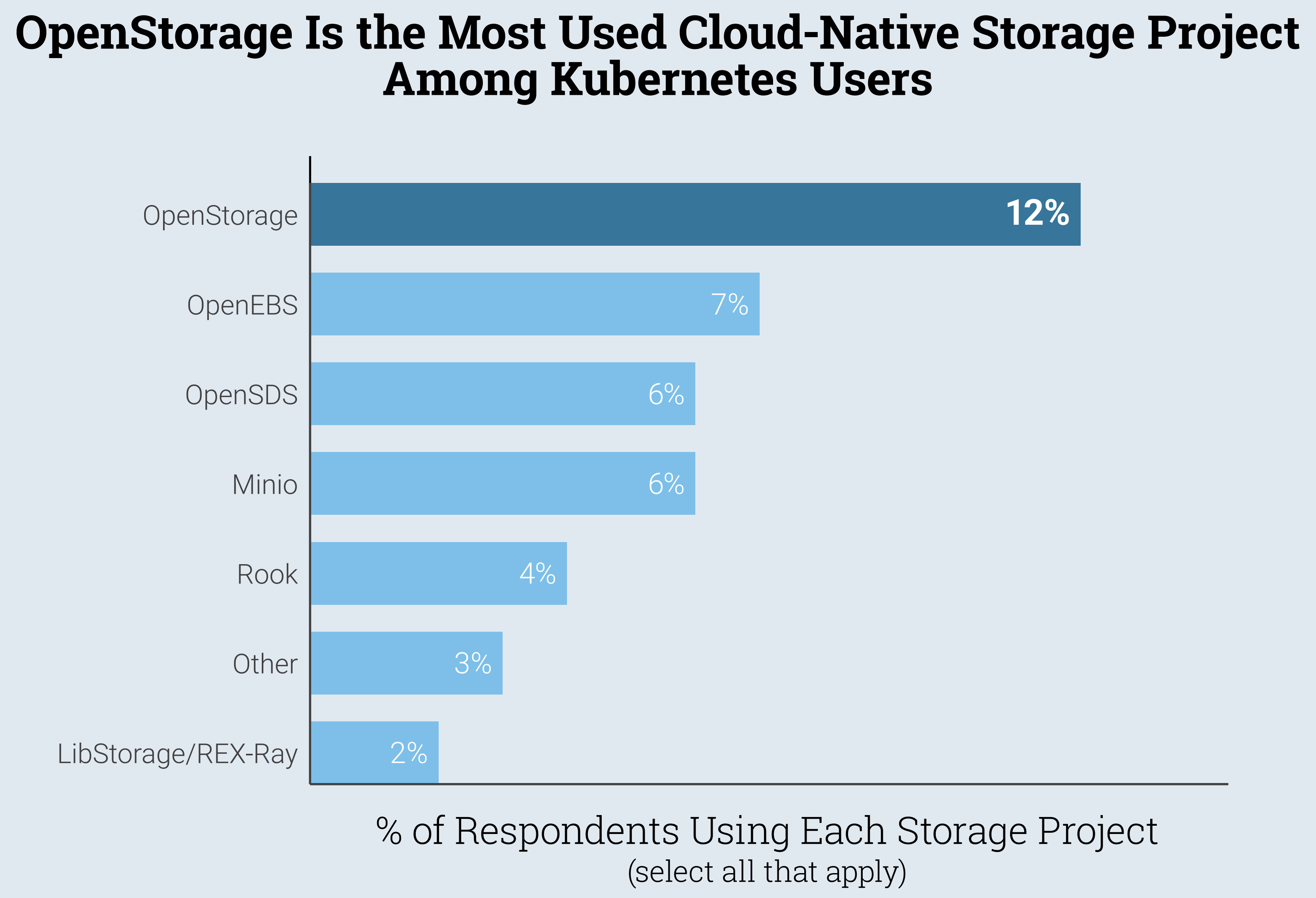
Tools and Infrastructure Surrounding Kubernetes

The CNCF survey also asked about several types of cloud-native infrastructure and tools, some of which are specifically marketed as working well with Kubernetes. The following section is based solely on the respondents who use Kubernetes to manage containers. Thus, even when the tools are not directly managing Kubernetes deployments, we do get a sense of the environments being used alongside Kubernetes.

Storage

The top cloud-native storage project among Kubernetes users is OpenStorage, followed by Minio, OpenEBS and OpenSDS. The questionnaire did not originally include OpenEBS, but it was added as

FIG 1.10: *Twelve percent of Kubernetes-using organizations have adopted technology from the OpenStorage project.*



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Which of these cloud native storage projects is your organization using? n=527.
Note, only respondents managing containers with Kubernetes were included in the chart.

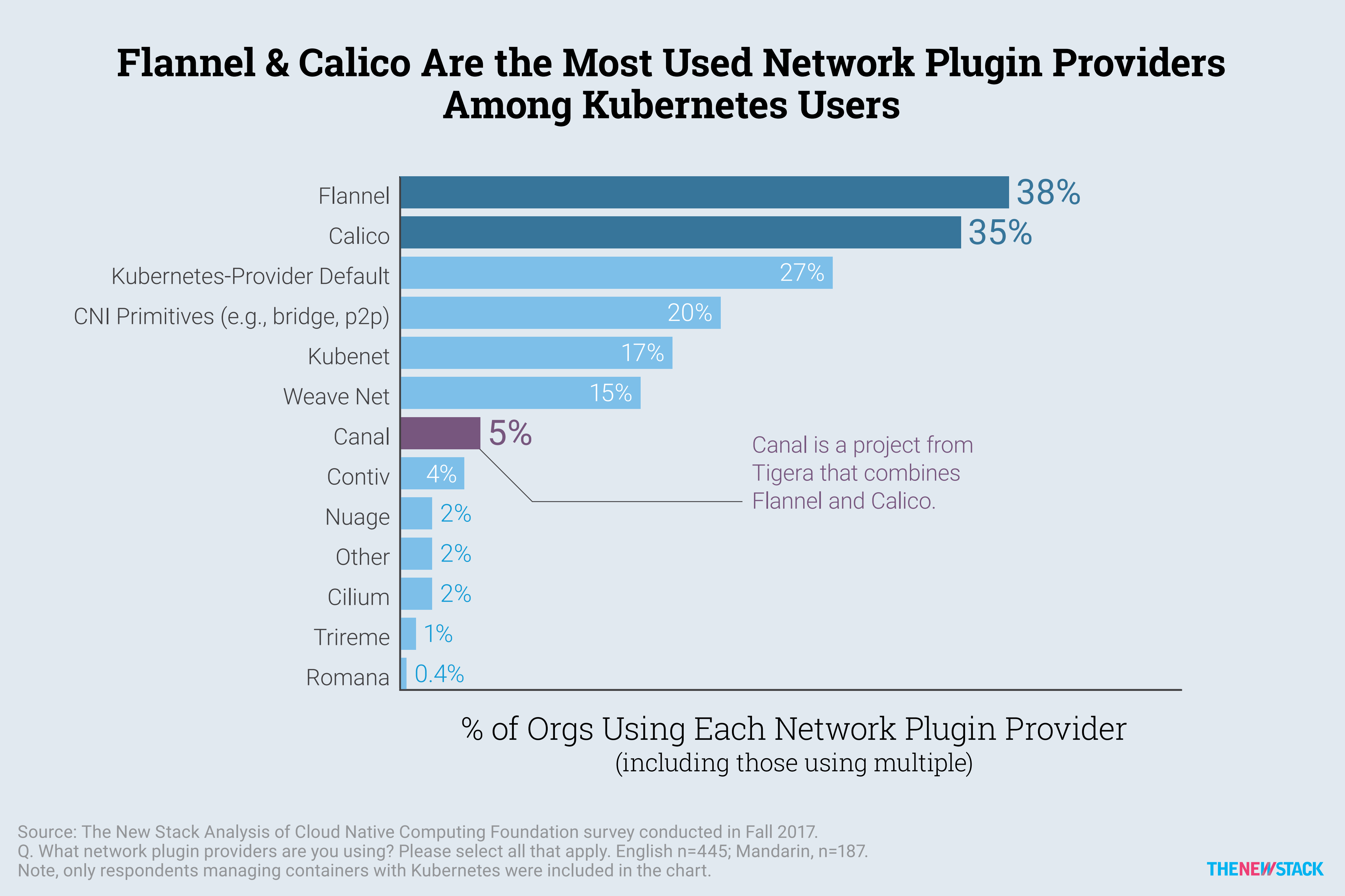
an option a few days after the survey launched. Excluding the first batch of respondents, OpenEBS' second place position increases slightly.

Networking

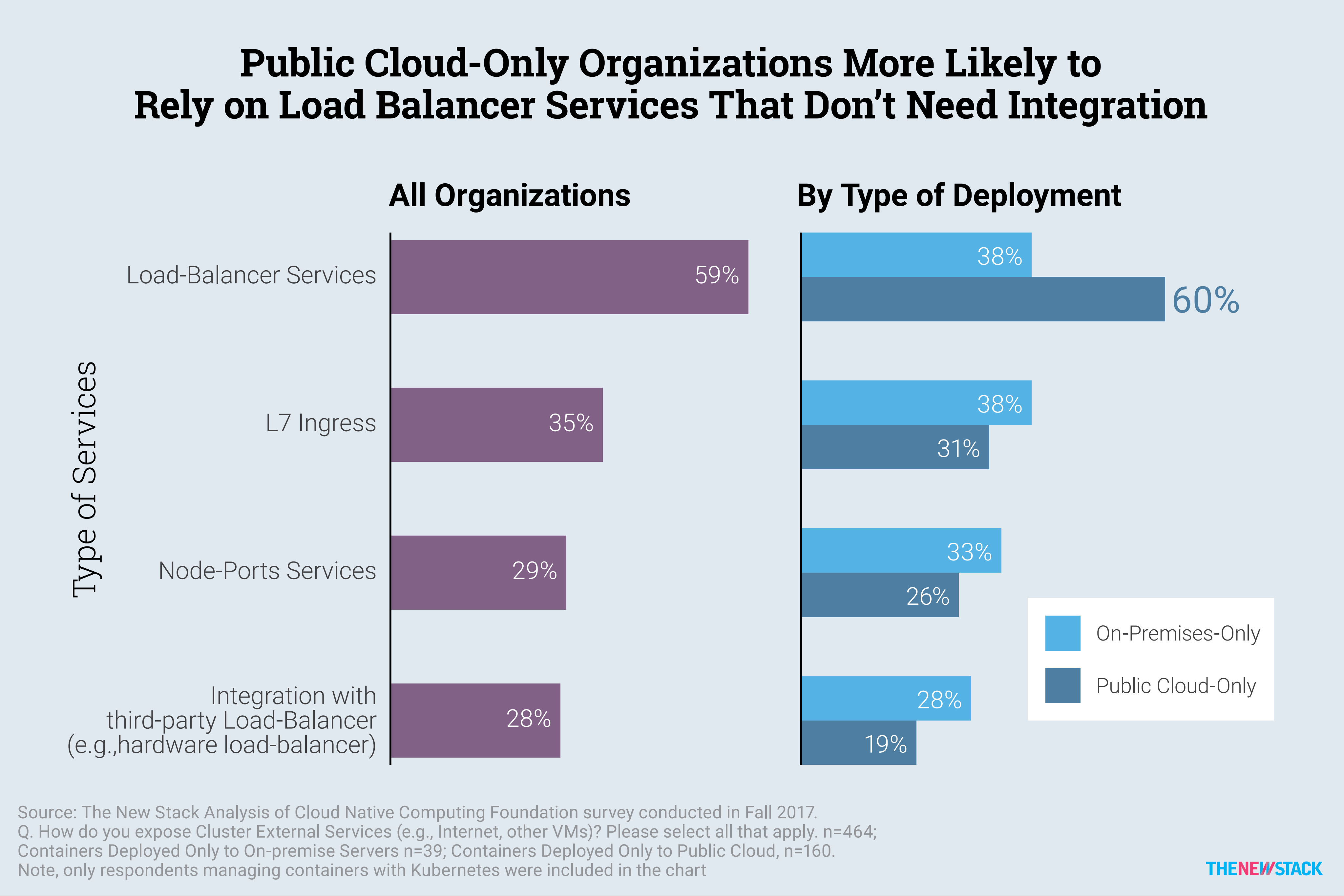
When asked about network plugin providers, Flannel came out on top, used by 38 percent of Kubernetes users, followed by Project Calico at 35 percent. The next most likely response was that a Kubernetes provider's default networking option was used. The results are similar to those from The New Stack's survey, which asked what software-defined networking solution was used in Kubernetes implementations.

The CNCF survey also asked how clusters are exposed to external services, such as from the internet or other virtual machines. At 59 percent, the most common response was load-balancer services. L7 ingress and node-port services were also used, but less often.

FIG 1.11: *Open source projects Flannel and Calico are the most widely used network plugins among organizations managing containers with Kubernetes.*



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. What network plugin providers are you using? Please select all that apply. English n=445; Mandarin, n=187.
Note, only respondents managing containers with Kubernetes were included in the chart.

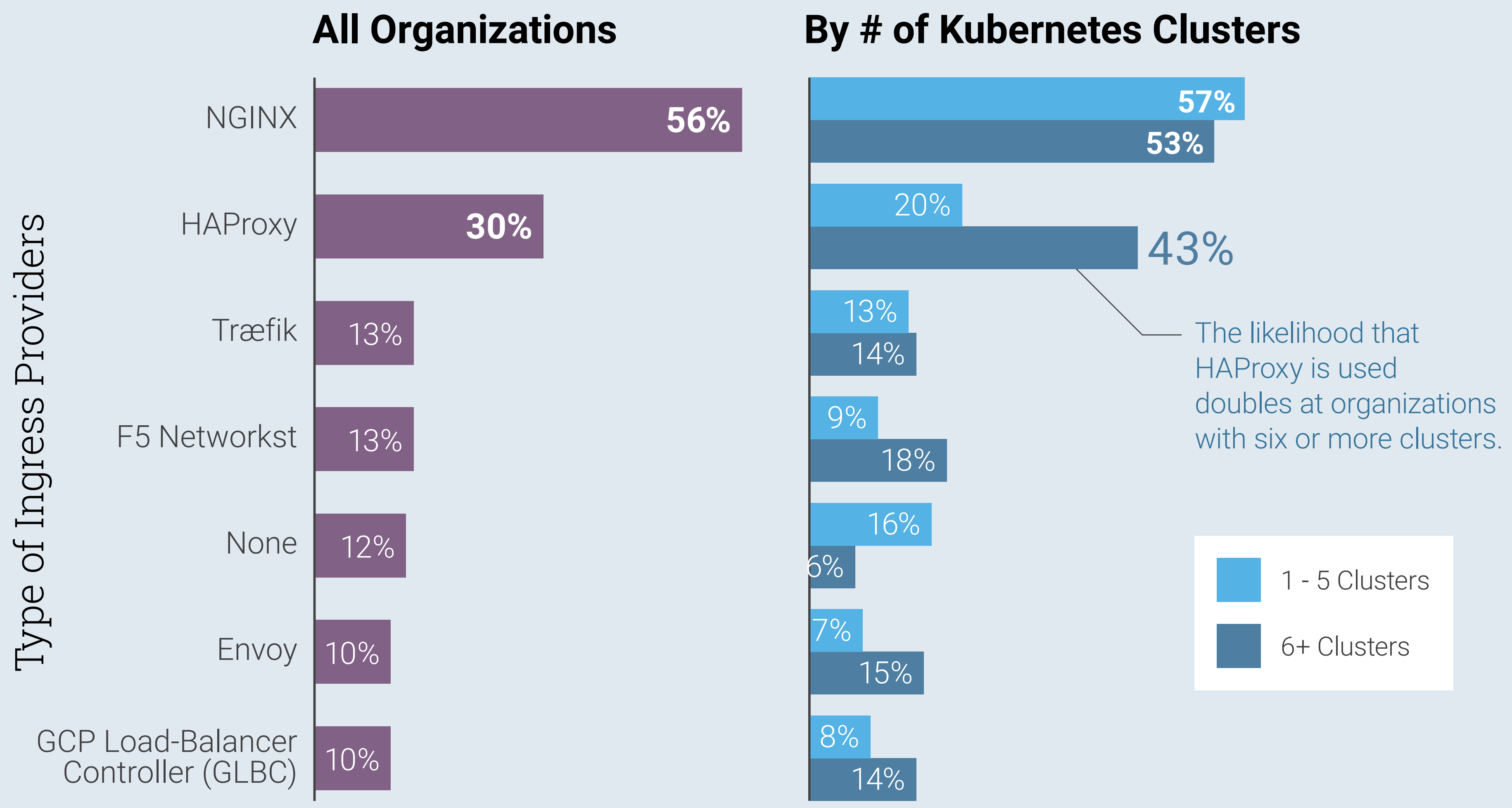


Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. How do you expose Cluster External Services (e.g., Internet, other VMs)? Please select all that apply. n=464;
Containers Deployed Only to On-premise Servers n=39; Containers Deployed Only to Public Cloud, n=160.
Note, only respondents managing containers with Kubernetes were included in the chart

FIG 1.12: *Fifty-nine percent of Kubernetes users expose external services with load-balancer services.*

Organizations that only deploy containers to on-premises servers were 37 percent less likely to use load-balancing services. Instead, they were almost 50 percent more likely to use an integrated approach that might include a hardware-based load balancer. These organizations may be using an integrated approach because their networking teams have already invested in a hardware solution. In these cases, organizations have one more moving part that they must manage instead of handle internally.

Respondents were asked specifically which ingress providers they used for Kubernetes. At 56 percent, NGINX is the most used, followed by HAProxy. Yet, usage patterns are different among organizations running six or more Kubernetes clusters. Among this group, HAProxy use doubles from 20 percent to 43 percent. The use of F5 Networks and Envoy also doubles among organizations with these increased needs.



NGINX and HAProxy Are Most Used Kubernetes Ingress Providers

Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. What Kubernetes ingress providers are you using? Please select all that apply. n=454; 1-5 Clusters, n=263; 6+ Clusters, n=160.

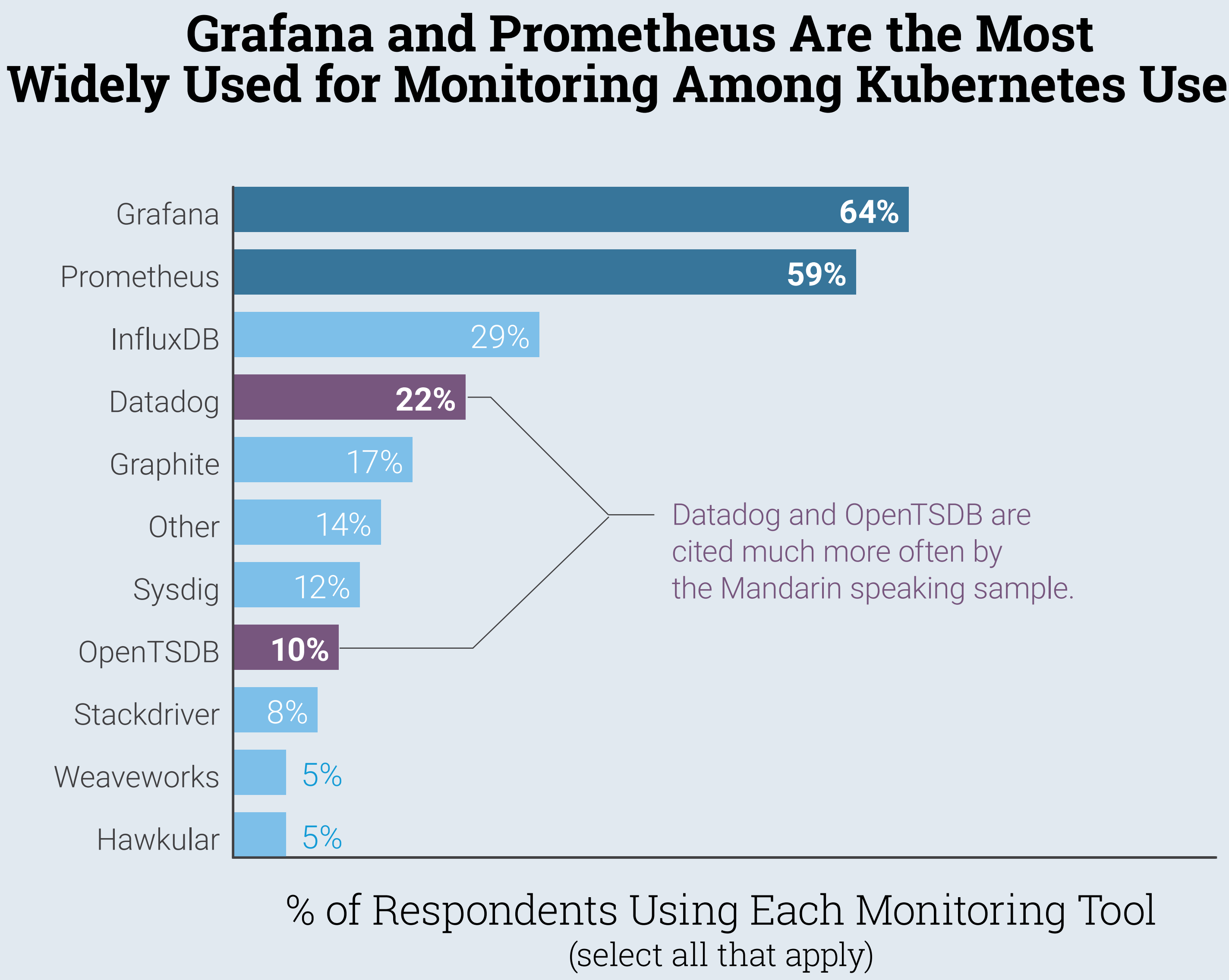
THENEWSTACK

FIG 1.13: *NGINX is the most used provider of Kubernetes ingress.*

Monitoring and Logging

When it comes to monitoring and logging, CNCF did not ask specifically about the tools used to track Kubernetes usage. That being said, the tools mentioned are commonly used for container management and will be familiar to the reader. For monitoring, Grafana is used by 64 percent of organizations that manage containers with Kubernetes, with CNCF's own Prometheus following closely behind at 59 percent.

As is the case with many reviews of monitoring tools, the responses differ significantly, with varying degrees of overlapping functionality. Grafana and Graphite are primarily visualization tools, but Kibana, Elastic's option, was not included in the questionnaire. In addition, CNCF did not ask about many monitoring vendors' offerings, possibly because their heritage is based on application instead of infrastructure monitoring.



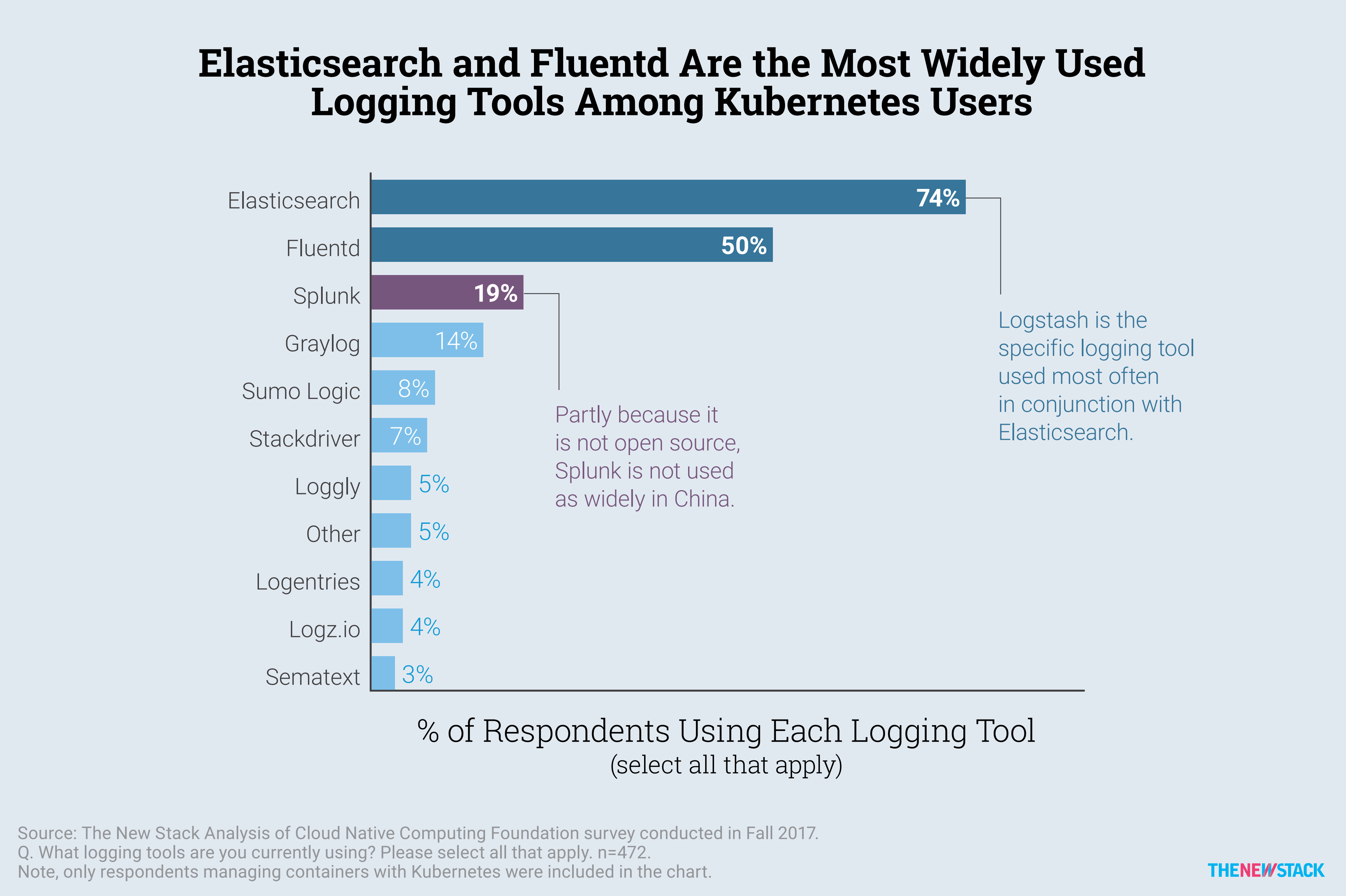
Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. What monitoring tools are you currently using? Please select all that apply. English n=489; Mandarin, n=187.
Note, only respondents managing containers with Kubernetes were included in the chart.

THENEWSTACK

FIG 1.14: *Grafana and Prometheus are the most commonly used monitoring tools, with InfluxDB coming in third.*

Time series database InfluxDB was used by 29 percent of respondents, and OpenTSDB was used by 10 percent. Although Prometheus can be set up to provide functionality similar to a time series database, it doesn't necessarily replace the need for one. Among Prometheus-using Kubernetes shops, InfluxDB's adoption rate increases slightly at the same time OpenTSDB use drops several percentage points.

Most monitoring stacks include a way to collect, process, store and visualize data. The previous chart dealt with ways data is processed and visualized. The next chart is about how it is stored. When asked what logging tools they use, 74 percent of respondents said Elasticsearch, which is part of the way in which the Elastic Stack (formerly known as ELK) collects data. The specific logging tool in the stack is called Logstash. Fluentd is used by half of respondents, often in place of Logstash. In fact,



Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. What logging tools are you currently using? Please select all that apply. n=472.
Note, only respondents managing containers with Kubernetes were included in the chart.

FIG 1.15: *Elasticsearch (which is part of the larger Elastic Stack) is the most widely used logging tool, but Fluentd is used by half of Kubernetes-deploying organizations.*

the EFK acronym is often used to describe an Elasticsearch, Fluentd, and Kibana stack. Splunk comes in third place, with its adoption inhibited by the fact that it is not an open source project.

It appears that organizations continue to build custom monitoring environments that simultaneously use multiple tools. Some respondents complained that Prometheus does not solve their logging problems. Below are direct quotes about what Kubernetes users want regarding monitoring and logging:

- “For monitoring, Prometheus could support authorization and authentication natively. When Prometheus is running inside Kubernetes, it should allow users to create rules within the Kubernetes API. Currently, we didn’t find a solution to easily deploy a production-ready logging solution for ELK stack, so we've ended up building our own."

- "Every vendor claims they interface with Kubernetes natively to pull logging information; none of them actually work. We keep having to write our own translator for whatever monitoring provider we go with every time."

- "The work on accelerating Prometheus is great. However, 'local storage only' does not seem to us to be fully production ready, in that we don't trust that architecture as much as we do even new containerized storage or similar."

- "It would be nice to be able to gather metrics from running services/pods in a unified way (pull). There is Prometheus, but we are using InfluxDB and right now we can't easily migrate to it since we already have alerts and monitoring setup using Influx's stack. Would be nice to be able to plug in some other solutions."

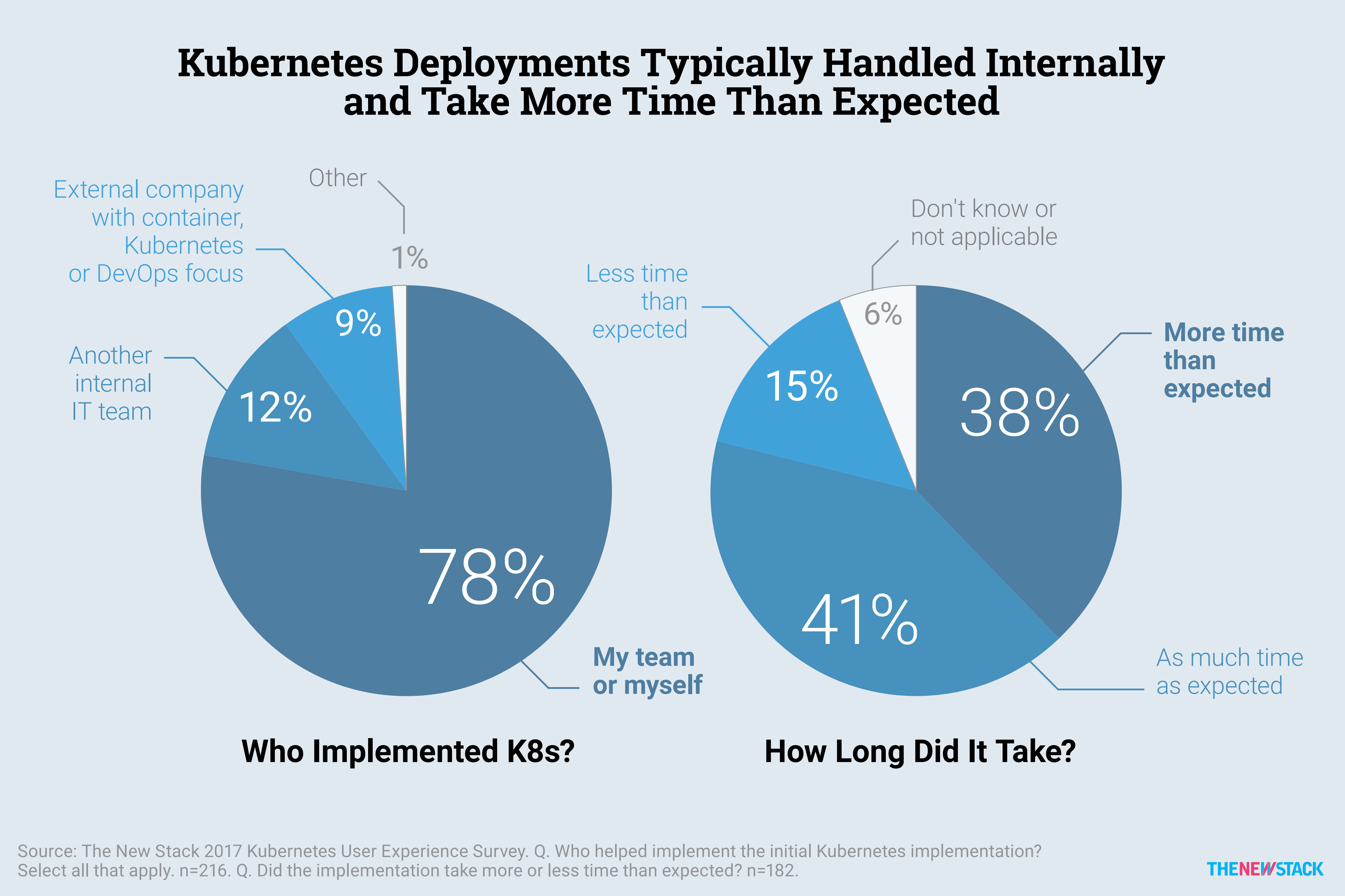
How Kubernetes Is Deployed

Chapter 2 will go into greater detail about the different options for you to deploy Kubernetes. In our May 2017 survey, 45 percent of people running Kubernetes in production were using a vendor-provided offering. Still, 74 percent were also using a community-supported distribution, meaning that organizations are likely using different implementations, depending on whether it is for test or production use cases. The CNCF questions were not as in-depth about the subject, but with respondents using multiple container management tools at the same time it is likely that their organizations are using more than one Kubernetes tool or platform at the same time.

The task of managing Kubernetes itself often falls to IT operations and SRE teams, with the DevOps role also being involved. In the May 2017 survey, only nine percent of respondents had actually used a third-party to help set up Kubernetes. Although people are deploying Kubernetes themselves, this did not impact the belief that the technology was meeting their goals. Nor did the fact that the hands-on deployments take longer than expected affect their level of satisfaction.

When asked about how how long it took to implement Kubernetes, twice as many respondents said it took more time than expected compared to those that said it took less. This points to room for improvement, which is expected to occur as experience with Kubernetes becomes more widespread in the workforce. CNCF's training and certification programs aim to help accelerate workforce development and curtail a potential skill shortage.

FIG 1.16: *Seventy-eight percent of respondents were directly involved with Kubernetes implementations.*



Source: The New Stack 2017 Kubernetes User Experience Survey. Q. Who helped implement the initial Kubernetes implementation? Select all that apply. n=216. Q. Did the implementation take more or less time than expected? n=182.

Final Considerations for Deployment

You have already started on your Kubernetes journey. It appears to be doing what you want it to do. The next decisions you face will be about how to expand Kubernetes' use in production environments. This chapter shows that although current Kubernetes implementations are still relatively small, many have moved beyond one-cluster experimentations.

Security, networking and storage are the top container challenges Kubernetes users face. As these organizations scale up their use of containers, they will face different challenges than those doing so in public, cloud-only environments. On-premises-only organizations, which are primarily challenged by storage, may want to pay attention to the top cloud-native storage projects in use: Minio, OpenEBS, OpenSDS and OpenStorage. For the public cloud-only Kubernetes deployments, monitoring and logging were more likely to be mentioned as concerns. These organizations should determine how they can integrate tools often used with Kubernetes with the software that is already being offered by their cloud provider.

When evaluating new services or solutions, consider how they will integrate with your existing and future stack. Container networking has started to standardize around Flannel and Project Calico, but there are still many options that are supported.

These and many other considerations for Kubernetes deployments will be covered in the next chapter. Rest assured that you can make these decisions informed by the latest data alongside your own organization's needs, processes and structure.

STRENGTHENING THE KUBERNETES CORE FOR IMPROVED OPERATIONS

The goal for the Kubernetes community in 2018 is to make Kubernetes rock solid. Over the past year, the community has focused on building out the Kubernetes core, such as networking, security and storage. For the new year, we shouldn't necessarily expect major changes or even Kubernetes 2.0. Instead, it's a year to focus on the basics, providing a base on which different distribution providers can build out their unique offerings.

In this context, The New Stack founder and Editor-in-Chief Alex Williams discusses existing and emerging deployment patterns with Ihor Dvoretskyi, developer advocate at the Cloud Native Computing Foundation. The Kubernetes community is working closely with the major cloud providers, all of which announced native Kubernetes integration in 2017, to build out their offerings in the coming year. As this work proceeds, Dvoretskyi says making the Kubernetes core rock solid means ensuring the same functionality of vanilla Kubernetes for any conformant distribution, regardless of the type of deployment. **Listen on SoundCloud.**

***Ihor Dvoretskyi** is a developer advocate at the Cloud Native Computing Foundation. He is a product manager for Kubernetes, co-leading the Product Management Special Interest Group, focused on enhancing Kubernetes as an open source product. In addition, he participates in the Kubernetes release process as a features lead.*

CONTAINER SECURITY IN MULTITENANT ENVIRONMENTS

In a typical containerized environment, it's still theoretically feasible for a container to exploit a host Linux kernel, and thereby impact any of the other containers sharing that host, says Aqua Security co-founder and CTO Amir Jerbi in this podcast. Projects like Google's gVisor or Kata Containers would eliminate that by trying "to add a layer that will deal with the shared kernel and multitenancy challenge." But the larger issue of application security presents challenges for anyone trying to minimize the application's attack service. "It doesn't need to be a kernel exploit ... It can be a wrong application logic that would allow someone to get access to your container and to your data. ... Aqua will take control and mitigate that risk."

Container isolation separates the security issue into the infrastructure plane and the application plane. The issues of application behavior can now be addressed separately. It also means they may need to be addressed urgently, as the question of how such isolated multitenant services will behave in production is unresolved. **Listen on SoundCloud.**

Amir Jerbi *co-founded Aqua with the vision of creating a simpler and lighter security solution. Prior to Aqua, he was a chief architect at CA Technologies, and brings 17 years of security software experience in technical leadership positions. He holds 14 cloud and virtual security patents and enjoys backpacking in exotic places in his free time.*

KUBERNETES DEPLOYMENT PATTERNS

by **JANAKIRAM MSV**

The rapid growth of Kubernetes in the container ecosystem has led to multiple deployment models, ranging from do-it-yourself to completely automated and managed forms of clusters. Irrespective of how it is deployed, developers and operations teams follow a standardized, consistent workflow for managing the application life cycle of containerized applications. This is one of the key advantages of Kubernetes.

Customers considering Kubernetes have access to a wide spectrum of deployment models, available in the form of developer-friendly Platform as a Service (PaaS) environments to highly customized deployments running on bare metal servers. Each model has its own advantages and disadvantages. We learned in the previous chapter — What the Data Says About Kubernetes Deployments — for example, that storage was the biggest challenge for organizations that exclusively deploy containers to on-premises servers, while those that deploy solely to the cloud cite monitoring and logging as their biggest challenge.

This chapter attempts to highlight various deployment patterns employed by Kubernetes users. The objective is to help organizations understand the options for deployment, the challenges and considerations associated with each, as well as the management models for running production workloads in Kubernetes.

Keep in mind that security is an important aspect of any Kubernetes deployment and should be considered from the start when assessing various deployment patterns. Chapter 3 takes an in-depth look at security considerations from the perspective of containers, the Kubernetes deployment itself and network security. Such a holistic approach is needed to ensure that containers are deployed securely and that the attack surface is minimized. Although many security practices are still evolving, the next chapter reviews current best practices which apply broadly to any Kubernetes deployment, whether you're self-hosting a cluster or employing a managed service.

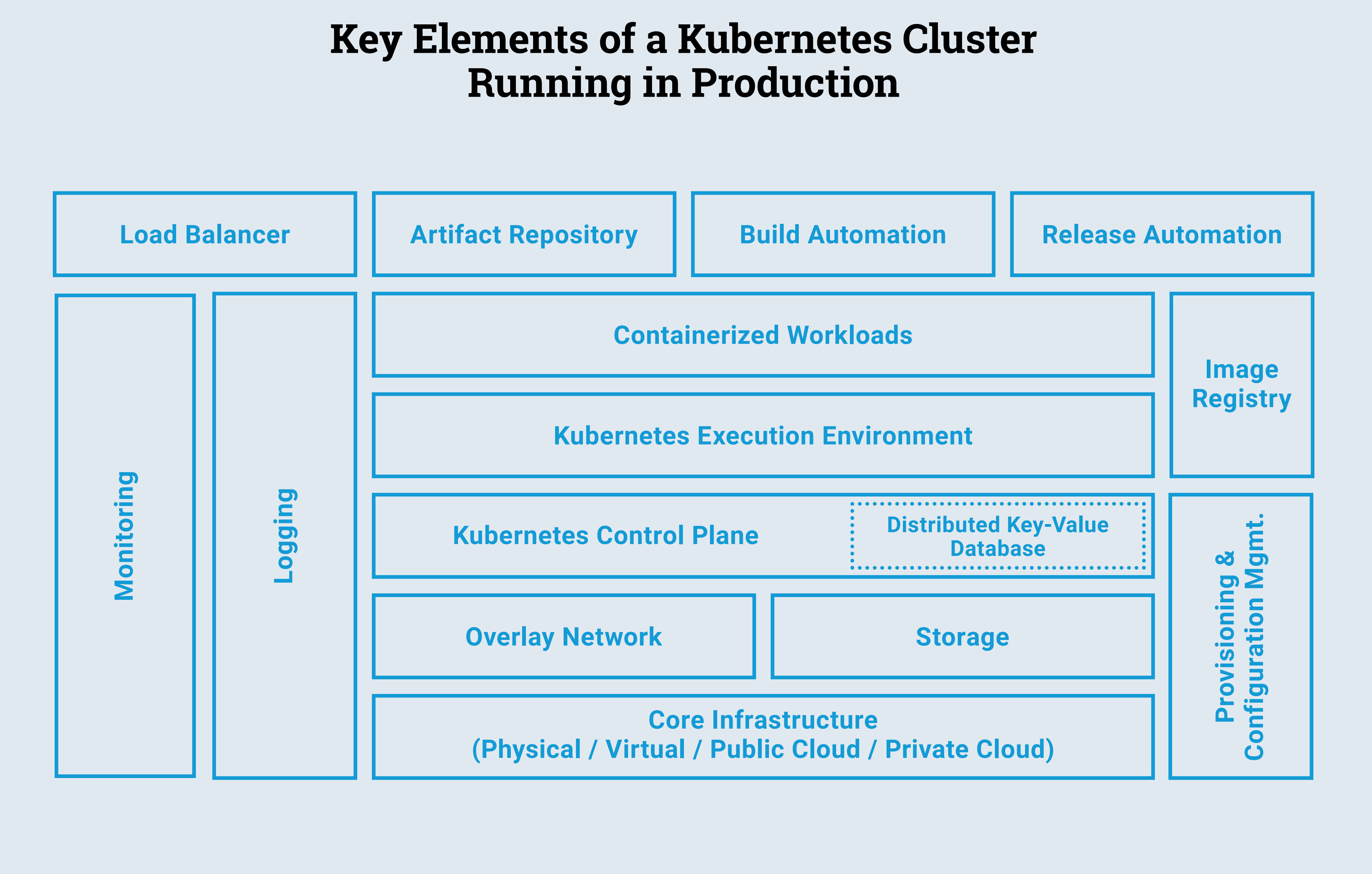
Key Elements of a Kubernetes Cluster Running in Production

Before exploring various options available for running containerized workloads in production, let's take a closer look at the stack.

Apart from Kubernetes, there are multiple components that are critical to a production cluster. An image registry and a robust monitoring and logging tool, for example, are components that ensure higher availability of the workloads.

This section introduces the core components of a production stack that runs mission-critical, containerized workloads.

Core Infrastructure: This acts as the foundation for the Kubernetes cluster and the containerized workload by exposing the compute, networking and storage infrastructure. The core infrastructure may be


Key Elements of a Kubernetes Cluster Running in Production
Load Balancer
Artifact Repository
Build Automation
Release Automation
Monitoring
Logging
Containerized Workloads
Image Registry
Kubernetes Execution Environment
Kubernetes Control Plane
Distributed Key-Value Database
Provisioning & Configuration Mgmt.
Overlay Network
Storage
Core Infrastructure (Physical / Virtual / Public Cloud / Private Cloud)
THENEWSTACK

Source: Janakiram MSV

FIG 2.1: *The production stack running containerized workloads in a Kubernetes environment contains multiple critical components.*

based on bare metal servers, a virtualized data center, private cloud or public cloud Infrastructure as a Service (IaaS).

Overlay Network: A Kubernetes cluster depends on a software-defined networking layer for internal communication. This overlay network enables all the components running within the cluster to talk to each other. Customers can choose from Calico, Flannel, Romana and Weave Net, among other networking options.

Storage: To run stateful workloads such as databases, a software-defined storage layer should be available to the Kubernetes cluster. This storage layer will be exposed to the containers as persistent volumes. Distributed storage software such as Gluster, Network File Systems (NFS) and block storage volumes are the preferred choices.

Kubernetes Control Plane: This layer runs the master nodes of Kubernetes that are responsible for the scheduling and orchestration of workloads. The master nodes that expose the control plane application programming interface (API) are configured for high availability to ensure maximum uptime of the cluster.

Distributed Key-Value Database: A Kubernetes cluster depends on a distributed database to maintain a single source of truth. This database maintains the current state of the cluster and deployed workloads. Since this database is critical for the health of the cluster, it is typically configured for redundancy and higher availability. The open source project from CoreOS, etcd, is used as the distributed key-value database.

Kubernetes Execution Environment: This layer consists of a set of worker nodes that act as the workhorses of the cluster. When a workload is deployed to Kubernetes, the master node makes scheduling decisions based on certain parameters such as node utilization. It allocates one of the available nodes to run the job. Since this layer is directly responsible for the availability and scalability of applications, it needs to be elastic. The worker nodes are configured to auto-scale in order to grow and shrink the cluster dynamically.

Containerized Workloads: These are the applications that are deployed within the Kubernetes cluster. A subset of the workload is exposed to the outside world to access the user interface and API layers of the application.

Provisioning and Configuration Management: Installing and configuring a Kubernetes cluster is not very different from deploying a highly available, mission-critical, distributed application. To ensure consistency and repeatability, customers often rely on toolchains such as Ansible, Chef, Puppet, Terraform and other automation tools. These tools make it easier to upgrade, patch and maintain Kubernetes infrastructure.

Image Registry: Before running the applications, Kubernetes nodes pull the corresponding container images from a registry. In environments where new images are automatically built each time the code is committed, the applications are upgraded to run the latest version of the image. To reduce latency and to increase security, images are stored in a registry that is co-located with the cluster. This architecture ensures that the later version of images are always available to the Kubernetes cluster.

Logging and Monitoring: Distributed applications generate a lot of logs, and Kubernetes is not an exception. Every component of the cluster, including the deployed application, emit logs that need to be captured and processed. The logs are useful for debugging problems and monitoring cluster activity. Logs, when combined with monitoring tools, provide rich insights into the state of a cluster. Tools such as those in the Elastic Stack (Elasticsearch, Logstash and Kibana), Grafana and Prometheus are used for logging and monitoring. This layer is an essential part of production deployments.

Load Balancer: The load balancer plays an important role in exposing two endpoints to the outside world: the control plane API and public-facing applications. Because the control plane is run across multiple master nodes, the API is accessed via a load balancer. Similarly, the API endpoints and web frontends of applications need a load balancer to become accessible to the users.

Artifact Repository: An artifact repository maintains the assets that belong to an application. As the complexity of distributed applications grows, there is a need to maintain various configuration settings, dependencies, packages, scripts and even binaries. In some cases, the artifact repository also doubles as a container registry.

Build and Release Management: With continuous integration and delivery becoming the preferred mechanism for application lifecycle management (ALM), build and release automation is becoming key. These tools connect the dots between source code management systems and production environments through an efficient pipeline. Atlassian Bamboo, CloudBees Jenkins and Shippable are some of the tools used for automated build and release management.

Depending on the deployment pattern, the ownership of these layers might shift to the platform provider or it may lie with the customer. We will explore the aspect of shared responsibility where the infrastructure is jointly managed by customers and the providers in the following sections.

Custom, Self-Hosted Kubernetes

Kubernetes is one of the most successful open source projects of the recent past. Under the supervision of the Cloud Native Computing Foundation (CNCF), the project enjoys contributions from skilled and passionate developers working at CoreOS, Google, Huawei, IBM, Red Hat and ZTE, among other companies. The source code is of high quality; it goes through a rigorous evaluation from the community. The upstream codebase available in the GitHub repo is used for deploying production Kubernetes clusters. The stock Kubernetes code is used by many users and third-party tools to run production-grade clusters. Still, complexity of implementation is among the the top reasons organizations cited for not using Kubernetes, according to the CNCF's fall 2017 survey and The New Stack's May 2017 Kubernetes User Experience Survey.

As Kubernetes matures, there is a great emphasis from the community on simplifying the installation. Though the initial versions of the software were complex to install, the addition of tools such as Kubeadm have made it easier for an average system administrator to deploy Kubernetes.

Kubeadm simplifies deploying a cluster by automating the configuration of control plane and the execution environments. It is available as native packages on CentOS, Ubuntu and other mainstream Linux distributions.

The availability of tools such as Cloud Foundry Container Runtime, Canonical conjure-up and kops have also made Kubernetes deployment simpler across data center and public cloud environments.

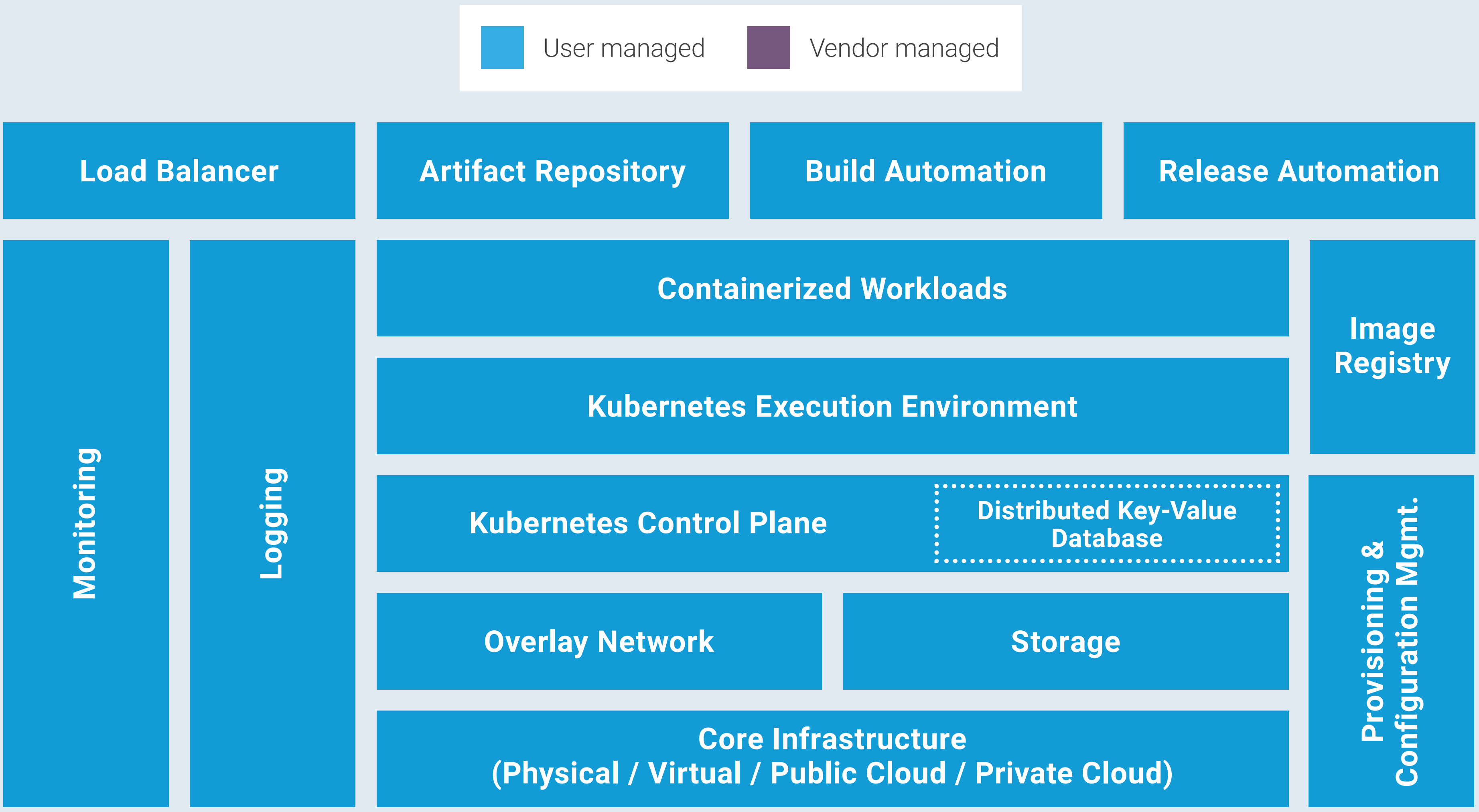
Access to high-quality code maintained by the community combined with the emerging set of installation tools is prompting customers to go for a custom deployment of Kubernetes clusters. Users can deploy Kubernetes in an enterprise data center running physical servers; on virtual machines; and in private cloud, public cloud and hybrid cloud environments. The setup, configuration and deployment experience can vary based on the choice of tools.

When to Opt for Custom Deployment?

A custom deployment offers ultimate choice to customers. They can choose from a broad range of target deployment environments, machine configurations, operating systems, storage backends, network plugins and high-availability configurations. While it delivers choice and control, the responsibility of maintaining the clusters lies solely with the customer.

In custom deployments, customers own the entire stack powering the production cluster. From underlying compute, network and storage resources to the image registry, the user is responsible for the installation, configuration and management of the entire stack.

In the case of public cloud, some of the resources, such as virtual machines (VMs) and block storage devices, are managed by the IaaS provider.



Custom Deployment of Kubernetes Cluster, Entirely Managed by Users

Source: Janakiram MSV

THENEWSTACK

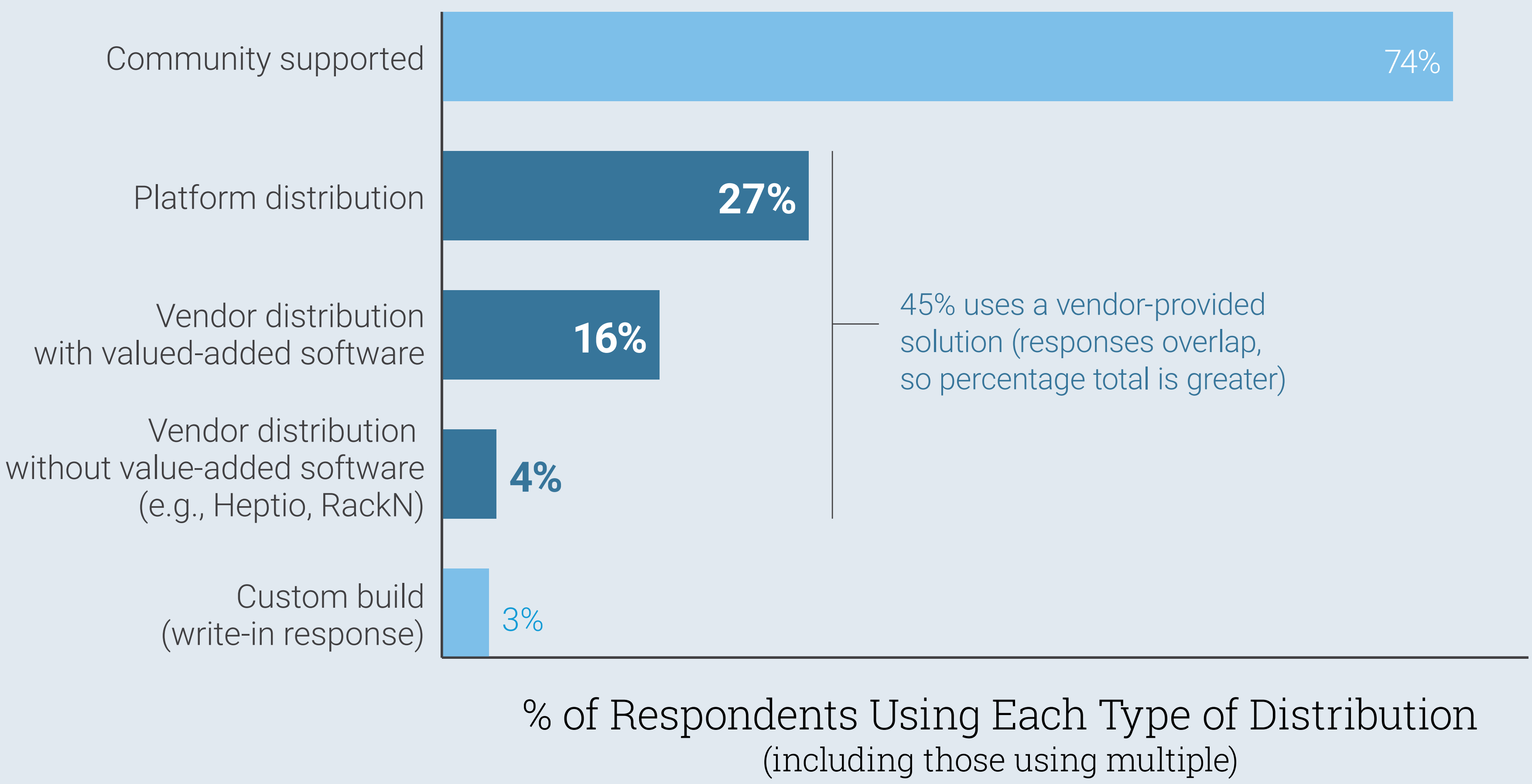
FIG 2.2: *All components running the production stack are maintained by users in a custom, self-hosted deployment.*

Organizations that need high customization in terms of operating systems, storage backends and overlay networks choose custom, self-hosted deployments. This approach also provides access to some of the cutting-edge features that may not be available in other deployment models.

Custom deployments are the cheapest option since the customers would only have to invest in the infrastructure. Almost all the tools are open source and freely available from the community. But the organization has to account for the staffing and support costs involved in maintaining the infrastructure.

The Kubernetes community frequently ships newer versions that improve the stability and security of the platform. Upgrading custom deployments

Types of Distributions Used



Source: The New Stack 2017 Kubernetes User Experience Survey.
Q. What types of distributions are being used? (multiple responses allowed) n=173.

FIG 2.3: *Nearly three-quarters of users deploy a community-supported Kubernetes distribution*

to the latest version with minimal downtime demands advanced Kubernetes administration skills.

Since the container images need to be kept close to the cluster, customers running self-hosted Kubernetes deployments will have to set up and manage their own private registry.

If control is the primary criterion, organizations may go for custom deployments.

Bare Metal Servers

Kubernetes can be deployed on bare metal servers running one of the following Linux distributions:

- CentOS 6.

- Fedora 25/26.
- RHEL 7.
- Ubuntu 16.04.

Kubeadm, the tool for setting up the cluster from the ground up, can be used to target bare metal installations. It's the preferred tool to quickly install production-grade Kubernetes clusters in both bare metal and virtualized environments. As of December 2017, kubeadm is still in beta but stable enough to deploy production clusters. The implementation of the tool may change slightly in the future to support easier upgrades and high availability of clusters.

For customers running Ubuntu 16.04 or above, Canonical's tool, conjure-up, can automate the deployment. The same tool can be used for targeting Amazon Web Services (AWS), Azure, Google Compute Engine (GCE), Joyent, OpenStack and VMware environments.

There are Ansible Playbooks built on top of kubeadm, which automate the deployment of a multi-node Kubernetes cluster.

CoreOS has a deployment tool based on Terraform to deploy Kubernetes on bare metal. It installs Tectonic, the commercial distribution of Kubernetes from CoreOS.

Virtual Machines and Private Cloud

Kubernetes can be set up on individual VMs running on a hypervisor or in private cloud environments based on CloudStack, OpenStack and vSphere.

Kubernetes-anywhere is a tool customized for OpenStack and VMware environments. The tool creates an open virtual machine format (OVF) template that is used to bootstrap the cluster. VMware customers can also use Photon Controller to deploy Kubernetes.

Kubernetes-anywhere can also be used to launch clusters in an OpenStack environment running Keystone v3, Neutron with Load Balancer as a Service (LBaaS) v2 and Nova.

Public Cloud

Kubernetes is often deployed in IaaS platforms such as Amazon EC2, Azure VMs and Google Compute Engine. There are custom tools that target specific public cloud environments.

Kops is a popular open source project that has become the official tool for deploying Kubernetes in AWS. It supports a variety of options including the ability to deploy a highly available cluster spread across multiple availability zones. Though AWS is the primary cloud platform, support for GCE and VMware vSphere is in alpha.

Conjure-up is a preferred tool for deploying Kubernetes Core of The Canonical Distribution of Kubernetes in AWS, Azure, CloudSigma, Google and Joyent. Canonical partnered with Google to optimize deployment on Ubuntu VMs running in GCE.

Kubespray can be used in the cloud as well as on bare metal. Based on Ansible, Kubespray is a collection of playbooks targeting a variety of operating systems and deployment environments. It supports mainstream Linux distributions and comes with a composable network architecture. Customers can choose from Calico, Canal, Flannel and Weave, among other network plugins.

Cloud Foundry Container Runtime is a recent addition to the supported tools to deploy Kubernetes using the popular BOSH toolchain. It is tightly integrated with Cloud Foundry PaaS where the Application Runtime delivers a consistent and familiar experience to existing Cloud Foundry developers. Cloud Foundry customers use this tool to deploy Kubernetes as the underlying container platform for the PaaS.

Key Takeaway: Choose a self-hosted, custom deployment when you want absolute control over the entire stack. This option demands skilled DevOps teams familiar with installation and maintenance of a Kubernetes cluster in production.

Managed Kubernetes Clusters

Customers who want to run Kubernetes within their data center or public cloud environments without installing or maintaining the clusters opt for managed Kubernetes offerings.

Vendors delivering managed Kubernetes services charge customers for the management and maintenance of clusters. Customers will have to spend on the core infrastructure along with the subscription or license fee, charged separately by the managed Kubernetes vendor.

In this environment, a third-party vendor maintains the cluster remotely to ensure the health of the environment. The cluster will be periodically upgraded to the most recent version of Kubernetes with minimal disruption to the workload.

In November 2017, the CNCF announced the Certified Kubernetes Conformance Program certification which recognizes the platforms that support the required APIs. This certification guarantees portability and interoperability across multiple Kubernetes offerings.

When to Opt for a Managed Kubernetes Deployment?

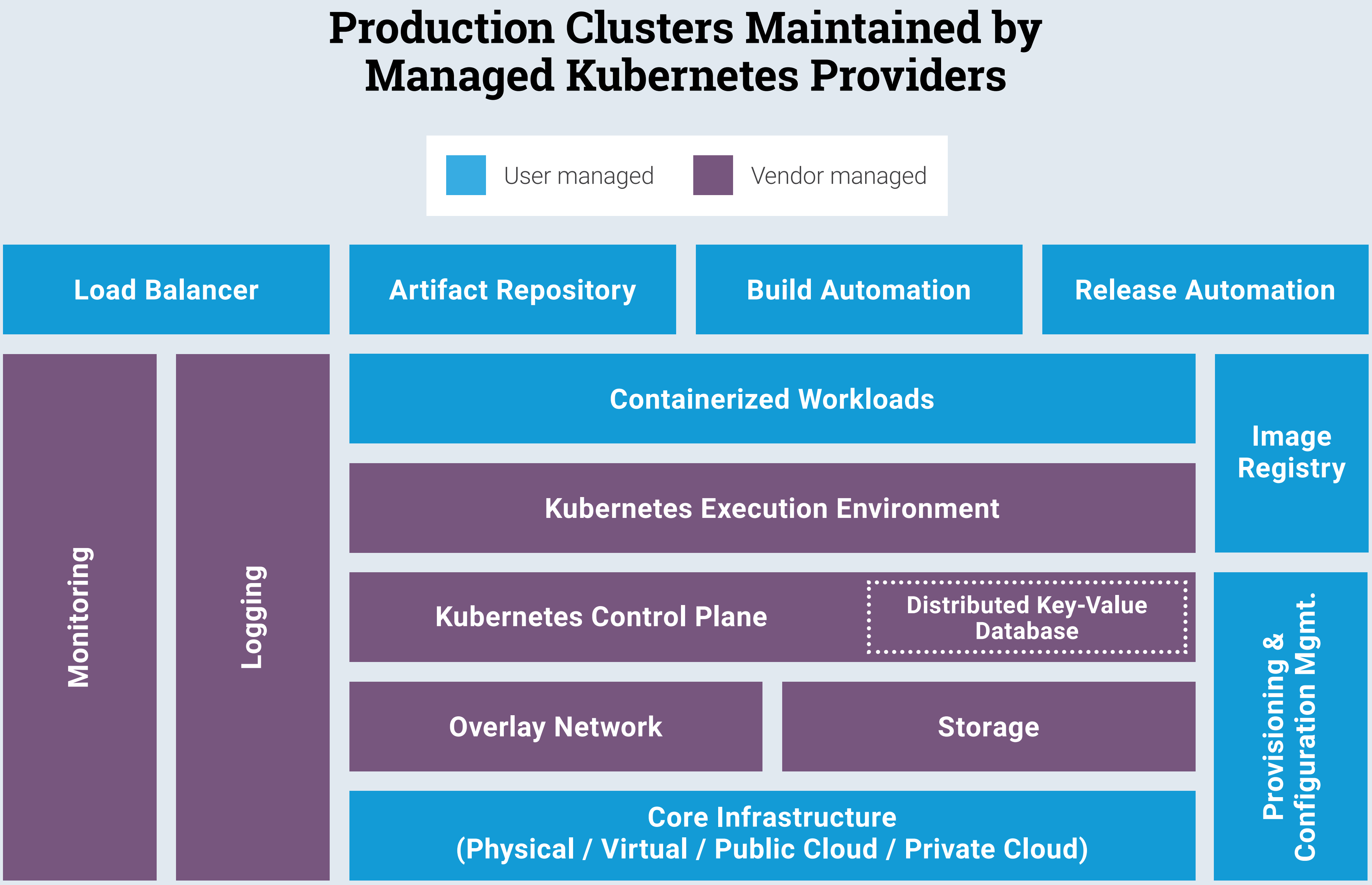
Managed Kubernetes platforms offer the best of both worlds — the choice of infrastructure combined with maintenance-free clusters. Organizations that do not have the required skills to set up, configure and manage large-scale deployments can opt for managed Kubernetes platforms.

Since the platform vendors remotely manage the clusters, users can focus on the application development instead of maintaining the container infrastructure.

Many managed Kubernetes providers, however, do not have an integrated container registry. It is the customer's responsibility to set up and manage a private container registry.

When compared to self-hosted deployments, managed Kubernetes offerings are more expensive. Customers will have to factor in the infrastructure cost along with the cluster management cost. But the advantage of periodic upgrades, patching, security and monitoring services of managed Kubernetes platforms can offset the cost in the long term.

FIG 2.4: *In managed Kubernetes clusters, only the core orchestration engine is managed by the vendors.*


Production Clusters Maintained by Managed Kubernetes Providers
User managed
Vendor managed
Load Balancer
Artifact Repository
Build Automation
Release Automation
Monitoring
Logging
Containerized Workloads
Image Registry
Kubernetes Execution Environment
Kubernetes Control Plane
Distributed Key-Value Database
Overlay Network
Storage
Provisioning & Configuration Mgmt.
Core Infrastructure (Physical / Virtual / Public Cloud / Private Cloud)

Source: Janakiram MSV

Customers can choose from some of the managed Kubernetes platforms, listed below.

Giant Swarm

Giant Swarm is a managed Kubernetes offering available on AWS and on-premises infrastructures. Unlike other implementations, Giant Swarm provides full administrator rights to the cluster through its APIs.

IBM Cloud Private

IBM offers a flavor of its cloud platform called IBM Cloud Private that can run within enterprise data centers. IBM Cloud Private is positioned as an application platform for developing and managing on-premises, containerized applications. It comes with an integrated environment for managing containers based on Kubernetes, a private image repository, a management console and monitoring frameworks. The stack has multiple other components that are originally available in IBM's public cloud.

Madcore

Madcore attempts to bring managed Kubernetes platforms and deep learning platforms together. The unique differentiator for this offering is the built-in support for Spark and associated deep learning tools within the Kubernetes cluster. Madcore is available only on AWS.

Platform9 Managed Kubernetes

Platform9 Managed Kubernetes is a Software as a Service (SaaS)-based managed solution which is infrastructure agnostic and can work across multiple public clouds and on-premises data centers. Based on its legacy of managing OpenStack, Platform9 delivers a single control plane to manage both environments. The service comes with features such as single-sign-on integration, support for block storage and a simple user interface.

StackPoint

StackPointCloud claims to deploy and manage a Kubernetes cluster to the cloud provider in three steps using a web-based interface. The service supports AWS, DigitalOcean, Google Cloud Platform and Microsoft Azure. The platform supports HA deployments with single-click upgrades. It has a marketplace of curated applications packaged to run on Kubernetes. Customers can also configure federated clusters to connect clusters running in different environments.

Tectonic

CoreOS — acquired by Red Hat in 2018 — offers a commercial, managed Kubernetes platform packaged as Tectonic. It can be installed in AWS, Azure, VMware and bare metal environments. Tectonic is tightly integrated with Container Linux, a lightweight, container-optimized Linux distribution, along with some of the extensions from CoreOS such as operators. CoreOS claims that the core components of the cluster — from Container Linux to etcd to Kubernetes — can be self-maintained without need for manual intervention. Tectonic comes with identity management that works with Lightweight Directory Access Protocol (LDAP) and Security Assertion Markup Language (SAML), along with enterprise governance features for fine-grained control.

Key Takeaway: Managed Kubernetes deployments offer the best of both worlds: Service Level Agreement (SLA)-driven managed clusters running within a controlled environment. Organizations that opt for this model own other essential components of a production stack.

Kubernetes as CaaS

In the early days of IaaS, virtual machines were the fundamental unit of deployment. The entire compute fabric, which is the foundation of IaaS, is based on hypervisors and VMs. Containers matured rapidly to become an alternative to VMs in the public cloud. Though they still run on VMs, customers wanted an easy mechanism to deploy, scale and manage containerized workloads. This led to a new cloud service delivery model where container management platforms are offered as a service, which is called Containers as a Service or CaaS.

Kubernetes evolved as the orchestration engine of choice for the public cloud providers. The distributed and scale-out architecture is well suited to run on IaaS. Similar to other managed services based on open source technologies, such as Hadoop and Spark, Kubernetes became available as a container management service in the public cloud.

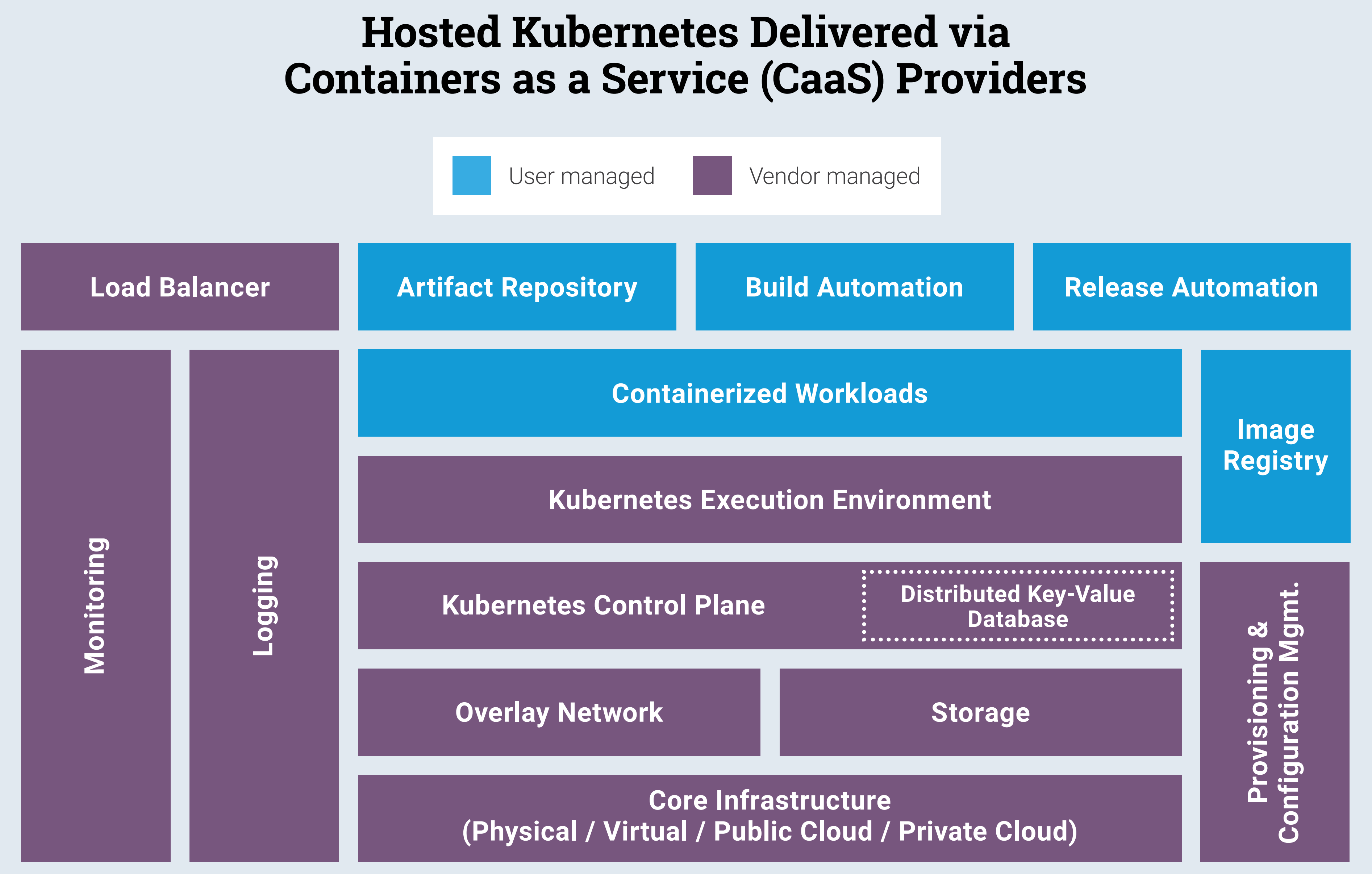
Most of the cloud providers expose Kubernetes worker nodes while managing the master servers themselves. This enables high availability and scale-out features to the clusters. The rapid provisioning of clusters with a few clicks is appealing to users.

Cloud providers offering CaaS may also include services such as secure private registry, ingress, container-optimized operating systems, auto-scaling of nodes and other features. This suite of services helps customers not only deploy containerized applications but also manage the life cycle.

When to Opt for Containers as a Service?

When compared to other deployment models, CaaS provides the shortest path to Kubernetes. Customers can have a fully configured, highly available, secure cluster in just a few minutes.

CaaS comes with features such as integrated monitoring, logging, auto-scaling, auto-upgradation and self healing. These out-of-the-box capabilities free the DevOps teams from managing the clusters, which results in long-term savings on infrastructure management.


Hosted Kubernetes Delivered via Containers as a Service (CaaS) Providers
🟦 User managed
🟪 Vendor managed
Load Balancer
Artifact Repository
Build Automation
Release Automation
Monitoring
Logging
Containerized Workloads
Image Registry
Kubernetes Execution Environment
Kubernetes Control Plane
Distributed Key-Value Database
Overlay Network
Storage
Core Infrastructure (Physical / Virtual / Public Cloud / Private Cloud)
Provisioning & Configuration Mgmt.

Source: Janakiram MSV

THENEWSTACK

FIG 2.5: *CaaS offerings manage the Kubernetes cluster along with the core infrastructure.*

With the heavy lifting moved to the cloud, users can focus on application deployment and management. The tight integration with the load balancers, firewalls, storage backends and container registry makes CaaS a compelling choice for public cloud users.

The DevOps toolchain, consisting of source code management, build automation, release management and artifact repositories, can be seamlessly integrated with CaaS. In most cases, the DevOps services are also exposed by the cloud provider.

The flipside of CaaS is lack of control. Cloud providers may take more time to upgrade Kubernetes versions. They may not support all the add-ons and features. Customers cannot choose the storage backend. Most of the CaaS providers charge for both the VMs running the cluster and the node management services, which makes CaaS more expensive than other models.

Customers can choose from some of the following CaaS offerings in the public cloud:

Alibaba Cloud Container Service

The Chinese cloud provider, Alibaba Cloud (formerly Aliyun) has a managed Kubernetes CaaS offering, Alibaba Cloud Container Service. Though the service has tight integration with other components of Alibaba Cloud, lack of a private registry comes across as a limitation.

Amazon Elastic Container Service for Kubernetes (EKS)

At AWS re:Invent 2017, Amazon announced Amazon Elastic Container Service for Kubernetes (EKS), a managed Kubernetes service that runs on top of Amazon EC2. The service complements its existing container management platform available as ECS. EKS is tightly integrated with Amazon Virtual Private Cloud (VPC) for networking; Identity and Access Management (IAM) for authentication and authorization; Amazon Elastic Block Store (EBS) for storage; and Elastic Load Balancing (ELB) and AWS CloudTrail for logging. Amazon EKS provides a scalable and highly-available control plane that runs across multiple availability zones. The service automatically manages the availability and scalability of the Kubernetes masters and the etcd persistence layer for each cluster. It runs three Kubernetes masters across three Availability Zones in order to ensure high availability, and it automatically detects and replaces unhealthy masters.

Apprenda

Apprenda started as an enterprise PaaS company with a heavy focus on .NET and Java-based line-of-business applications. Like most of its competitors, Apprenda has started to integrate its platform with Kubernetes. Its latest offering, Apprenda Cloud Platform (ACP), aims to bridge the gap between legacy applications written in .NET and Java with the contemporary cloud-native applications. Apart from the container management offering, Apprenda has commercial support packages for open source Kubernetes.

Azure Container Service (AKS)

Microsoft recently introduced a managed Kubernetes service as part of its container management platform, called Azure Container Service (AKCS), which includes other orchestration engines such as Docker Swarm and Mesosphere DC/OS. AKS is a flavor of ACS that delivers managed Kubernetes as a Service. The key differentiator of AKS when compared to other offerings is the pricing model. Microsoft only charges for the virtual machines running the Kubernetes nodes with no additional cost of running and managing the master nodes. Microsoft also has a managed container registry that complements AKS.

Google Kubernetes Engine (GKE)

As the original developer of Kubernetes, Google was one of the first to offer Kubernetes-based CaaS on its cloud platform. Branded as Google Kubernetes Engine (GKE), the service is built on top of existing Google Cloud Platform (GCP) components such as Compute Engine, Cloud Load Balancing, Persistent Disk and Virtual Private Cloud (VPC) . The service is backed by a service level agreement which ensures high availability. GKE is one of the first CaaS platforms to get upgraded to the latest version of Kubernetes. GCP comes with a private container registry in the form of Google Container Registry.

Huawei Cloud Container Engine

Another Chinese company, Huawei, also has a managed Kubernetes service known as Cloud Container Engine. This cloud platform doesn't include a managed container registry.

IBM Cloud Container Service

IBM Cloud Container Service is based on Docker and Kubernetes. It is tightly integrated with the DevOps toolchain available on IBM Cloud including the container registry. Apart from offering Kubernetes in the public cloud, IBM's private PaaS is built on top of Kubernetes.

Pivotal Container Service (PKS)

Pivotal partnered with VMware and Google to build Pivotal Container Service (PKS), a CaaS layer that runs on top of VMware vSphere and Google Cloud Platform. PKS deploys a highly-available Kubernetes cluster with built-in support for monitoring, analytics and automated health checks. It is tightly integrated with VMware tools like vRealize Operations Manager, vSAN network storage, and Wavefront monitoring and analytics for a full-featured, on-premises deployment. Customers using GKE can easily move workloads to and from PKS. The service is aimed at VMware customers considering running containerized workloads in hybrid environments.

Key Takeaway: CaaS delivers everything customers need to run Kubernetes in production; however, it is less flexible in terms of choosing custom network and storage backends.

Kubernetes as PaaS

Platform as a Service (PaaS) started as a polyglot, multi-tenant environment to run applications in isolated context. Early PaaS implementations were based on proprietary technologies for isolating the application context. When Docker became available as an open source containerization technology, PaaS providers replaced the proprietary execution environments with containers. Today, most of the PaaS offerings are layered on containers.

While code packaged as containers still runs in VMs or bare metal servers, Kubernetes has become the de facto orchestration engine that manages the containers. The PaaS layer is built on top of Kubernetes to deliver end-to-end application life cycle management services.

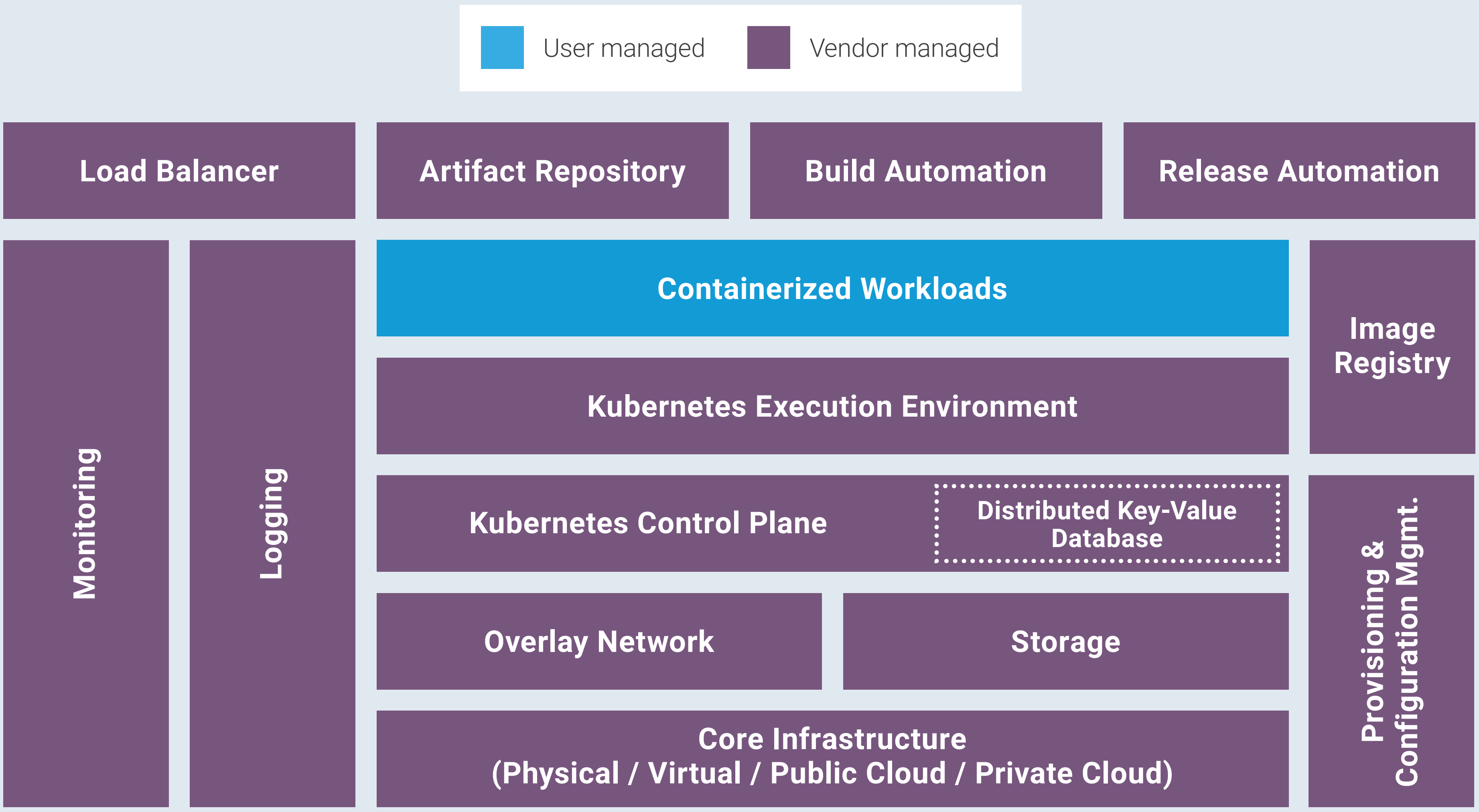
All the above discussed deployment models, such as self-hosted and managed Kubernetes clusters, target administrators and DevOps teams. Kubernetes-based PaaS is designed for developers so that they can bring the source code to the platform instead of packaged artifacts such as Docker images or Kubernetes pods. They do not need to deal with the operational aspects of the platform. Instead, they squarely focus on the code and the application life cycle.

PaaS can be deployed within public cloud environments or inside the enterprise data center. Large organizations are adopting PaaS to run internal applications as well as customer-facing applications. They want the development teams to have a consistent experience irrespective of the deployment target: private or public cloud. Kubernetes-based PaaS offerings deliver this promise to enterprises through consistent workflow and deployment patterns. In many cases, developers need not even know that their code would be running inside a Kubernetes cluster. The PaaS layer abstracts the underlying details of Kubernetes and exposes only the API endpoints that developers understand.

Kubernetes has become the foundation of contemporary PaaS implementations. Below is a partial list of PaaS offerings that are integrated with Kubernetes.

When to Opt for Managed PaaS?

PaaS is developer-oriented, while other deployment models are operations-oriented. If an organization is considering a uniform, consistent layer that hides the complexity of container orchestration and



Kubernetes-Based PaaS Designed to Deliver End-to-End ALM Capabilities

Source: Janakiram MSV

THENEWSTACK

FIG 2.6: *PaaS vendors deliver the entire application platform that manages the life cycle of a workload.*

infrastructure management, they choose PaaS.

When DevOps and ALM tools are layered on top of Kubernetes orchestration, it turns into a container PaaS. Developers will not have to deal with packaging the code for containers as the platform includes tools to convert source code into images. Developers need not understand how to configure the service discovery for exposing internal and external services since the platform handles the connectivity between stateless and stateful services. Scaling, healing, monitoring and logging are built into the platform, which are configurable through APIs or web-based consoles.

Organizations that are looking for end-to-end platforms that deliver consistent developer experience and advanced ALM capabilities opt for PaaS. These offerings reduce the cost of owning and managing certain DevOps tools.

Though PaaS reduces the complexity, it lacks flexibility. Lack of customization is one of the limitations of these platforms. When compared to other delivery models, the PaaS licensing model makes it slightly expensive for users.

Hasura

Hasura.io is a Backend as a Service (BaaS) offering targeting modern web and mobile developers. It has authentication, data and file services APIs built on top of Kubernetes. Developers can use a command-line interface (CLI) to push their source code hosted in GitHub to Hasura's PaaS. The platform is built from the ground up on Kubernetes hosted on DigitalOcean. This PaaS/BaaS is ideal for startups and developers looking for rapid development and deployment.

Mesosphere DC/OS

Mesosphere's DC/OS is an open source distributed computing platform for running highly scalable workloads. To attract developers, Mesosphere initially added a PaaS layer to DC/OS called Marathon. Recently, Kubernetes is integrated with DC/OS to become an orchestration engine for containerized applications. It complements Marathon in delivering a fully-fledged platform experience to developers. Mesosphere promises to bridge the gap between stateful services, such as Hadoop and Spark, and stateless applications modeled around 12-factor apps.

Red Hat OpenShift

Red Hat OpenShift is one of the first platforms to fully adopt Kubernetes. This PaaS offering is available as an open source project (OpenShift Origin), hosted platform in the public cloud (OpenShift Online) and as an enterprise private PaaS (OpenShift Container Platform). Red Hat built OpenShift on top of its proven open source technology stack.

Types of Kubernetes Deployments

Features	Self-Hosted, Custom Deployment	Managed Kubernetes Cluster	Containers-as-a-Service (CaaS)	Platform-as-a-Service (PaaS)
Ability to customize	🔧🔧🔧 High	🔧🔧 Medium/High	🔧🔧 Medium/Low	🔧 Low
Overall costs (software and infrastructure)	💵💵💵 High	💵💵 Medium	💵💵💵 High	💵💵💵 High
Staffing and support costs	💵💵💵 High	💵💵 Medium	💵 Low	💵 Low
Administration skills	🔧🔧🔧 High	🔧🔧 Medium	🔧 Low	🔧 Low
Level of control	🔧🔧🔧 High	🔧🔧 Medium	🔧 Low	🔧 Low
Container image registry	✗ Not included	✗ ✓ Varies by provider	✓ Included	✓ Included
Portability and interoperability across clouds	🔧 Low	🔧🔧🔧 High (on cloud provider certified distributions)	🔧🔧🔧 Varies by provider	🔧🔧🔧 Varies by provider
Built-in patching, security and monitoring services	✗ Not included	✓ Included	✓ Included	✓ Included
Built-in high-availability features	✗ Not included	✗ Not included	✓ Included	✓ Included
Built-in infrastructure autoscaling (container autoscaling always included)	✗ Not included	✗ Not included	✓ Included	✓ Included
Rapid provisioning features	✗ Not included	✗ Not included	✓ Included	✓ Included
Complete application lifecycle management	✗ Not included	✗ Not included	✗ Not included	✓ Included

TABLE 2.1: *This table compares the varying levels of control, costs and features to expect from each deployment pattern.*

OpenShift comes with an application lifecycle management (ALM) tool that automates the entire deployment pipeline. Developers bring their source code to the platform which will be packaged as a container image and deployed within Kubernetes. OpenShift is integrated with DevOps tools such as Ansible, Git and Jenkins.

Key Takeaway: PaaS completely hides the complexity involved in setting up and managing the entire stack in production environments. The ALM tools available in PaaS make the platform developer-friendly.

Emerging Deployment Patterns

As Kubernetes matures, the use cases go beyond mundane container orchestration services. It is now being used in a variety of niche scenarios.

Here is a list of emerging patterns in which Kubernetes is utilized in purpose-built platforms.

Edge Computing

Edge computing is an evolving technology where compute, storage and networking are deployed closer to the data sources. The platform mimics the cloud by exposing endpoints for ingesting data, storing and processing and a dynamic rules engine. AWS Greengrass and Azure IoT Edge are examples of edge computing platforms that act as an extension of the public cloud. Containers are becoming an obvious choice to package and deploy various elements at the edge. Since the edge computing infrastructure consists of multiple resources that are collectively treated as a cluster, Kubernetes can be used as the cluster manager and orchestration engine. Most of the edge computing platforms support both x64 and Advanced RISC Machine (ARM). Kubernetes support for these architectures makes it an ideal choice for managing the cluster.

Public cloud vendors may eventually utilize Kubernetes to orchestrate the containers deployed in the edge. The portal or management console may provide the user experience to remotely manage the Kubernetes cluster.

Machine Learning

Machine learning (ML) is becoming one of the key components of public cloud computing. To train ML models at scale, customers upload large datasets to a centralized storage location and use a distributed computing cluster based on graphics processing units (GPUs). Companies, such as NVIDIA, are investing in container images that come with Compute Unified Device Architecture (CUDA) libraries to take advantage of the GPU resources. Multiple container images are launched in parallel to train machine learning models. Data scientists create and test models with smaller data sets in the local environment, package the algorithm into a Docker image and use that to launch a few thousand containers to perform large-scale training operations in the cloud. Since this scenario demands orchestrating containers at scale, Kubernetes is becoming a preferred platform for machine learning. Kubeflow is an open source project from Google to run ML workloads on Kubernetes. It adds a Tensorflow Custom Resource (CRD) that can be configured to use central processing units (CPUs) or GPUs, and adjusted to the size of a cluster with a single setting. CaaS environments, such as Google Kubernetes Engine and Azure Container Service, are used in conjunction with hosted machine learning platforms running on GPU-based compute resources.

Serverless Computing

Serverless computing is gaining popularity among developers. Along with VMs and containers, serverless has become an essential compute service offered by public cloud providers. Unlike other compute services, serverless is event -driven, which means that the code will only be executed when an external event is triggered. Developers upload code snippets to the serverless platform in the form of functions. AWS Lambda, Azure Functions, and Google Cloud Functions are some of the popular serverless computing choices in the public cloud. Given the polyglot and isolated nature of serverless, Docker can be used for serverless computing. Kubernetes becomes the obvious choice for managing these Docker containers at runtime. Apache OpenWhisk, Fission, Kubeless, nuclio and OpenFaaS are some of the serverless environments based on Kubernetes.

Stream Analytics

The rise of Internet of Things (IoT), multiplayer gaming, ad tech and social media analysis has resulted in the need for analyzing the data in real time. Technologies such as Apache Kafka and Apache Spark are used for ingesting and analyzing the streaming data. The layers responsible for the ingestion and real-time analytics have to scale rapidly and dynamically to meet the demand. Kubernetes is used for scaling these workloads that deal with stream analytics. The workloads take advantage of Kubernetes primitives such as StatefulSets and the Horizontal Pod Autoscaler to deliver the required scale. Iguazio is an example of a data platform built on top of Kubernetes to run big data workloads, including stream analytics.

WHY CLOUD-NATIVE ARCHITECTURES ARE INHERENTLY MORE SECURE

One of the most touted virtues of cloud-native application deployment is that it aims to free the software developer from having to worry about the state of their infrastructure. Cloud-native security is a service provided on behalf of the maintainer of the cloud-native development space who, in more and more enterprises, is someone in development. So how can cloud-native architectures be easier to use while also placing more responsibility on developers for delivering secure applications?

The New Stack correspondent Scott Fulton and Twistlock CTO John Morello discuss why cloud native is the future of application development and what makes it inherently more secure.

“One of the characteristics of containers is that they’re very predictable, or they should be,” Morello said. “This allows you to do security in a more predictable, automated way.” **Listen on SoundCloud.**

***John Morello** is the chief technology officer at Twistlock. As CTO, John leads the work with strategic customers and partners and drives the product roadmap. Prior to Twistlock, John was the chief information security officer of Albemarle, a Fortune 500 global chemical company, and spent 14 years at Microsoft.*

KUBERNETES SECURITY PATTERNS

by **CHENXI WANG**

As an orchestration platform, Kubernetes impacts many runtime security functions. These include authentication, authorization, logging and resource isolation, as well as more advanced implications such as workload placement and network segmentation. Because of the orchestrator's comprehensive reach in the container runtime environment, Kubernetes security is a critical aspect to the security posture of the container infrastructure. In this chapter, we will talk about threat models and various security considerations for a Kubernetes deployment and discuss best practices that organizations can follow.

Kubernetes Threat Models

To properly address security, we must first discuss the threat models. In the Kubernetes environment, there are generally four types of threats that apply regardless of the deployment model (with some edge case exceptions):

1. **External attacks aiming to compromise Kubernetes controls:** This threat exists for all connected systems. In the context of a

Kubernetes cluster, this represents attackers gaining access to the system or compromising certain controls that will impact the security of the system.

2. **Compromised containers/nodes:** If there are malicious containers within the Kubernetes-controlled environment, or malicious nodes within a cluster, what is the impact to the rest of the nodes or the cluster? Can you effectively contain the “blast radius” both on the node and within a cluster?
3. **Compromised credentials:** What happens when a Kubernetes administrator’s credential is compromised? How much does that affect the cluster?
4. **Misuse of legitimate privileges:** This could happen when systems are misconfigured, controls are lacking or operations are not closely monitored.

These different threats may result in a multitude of compromises and undesirable scenarios, including elevation of privileges, exfiltration of sensitive data, compromise of operations or breach of compliance policies.

One of the attack scenarios that has garnered a fair amount of attention is the concern of “blast radius” — how much damage can a compromised container do to other containers on the same node, or how much damage can a compromised node do to the rest of the cluster? The discussions henceforth in this chapter are motivated by these threat models as well as the need to minimize “blast radius.”

External Attacks

In a Kubernetes environment, the components that are accessible externally will be exposed to external attacks. Those components can include the application programming interface (API) server, kubelet and etcd. (The first ebook in this series, "The State of the Kubernetes Ecosystem," reviews the components in a Kubernetes node.)

To alleviate the threat of external attacks, ensure that only the necessary Kubernetes services are exposed and no more. Always enforce authentication and configure the right network security policies for any exposed services.

Compromised Containers

Ultimately, Kubernetes manages workloads running in containers. If a container is compromised, the concern is whether the container can escalate privileges to take control of other containers or the cluster.

Kubernetes' isolation mechanisms, such as namespaces or network segmentation, as well as some of the built-in, OS-level controls regulating what the container can access, can help limit the impact of compromised containers. One other control users should pay attention to is the ability to limit the number of containers that can run in a privileged mode — if a privileged container is compromised, it will be able to do much more harm than a normal container.

Compromised Credentials

When legitimate user credentials have been compromised, you may have a malicious user in the system. It is critical, therefore, to properly regulate and limit a user's access to cluster resources and to closely monitor user actions.

To reduce the impact of malicious users, you need to enforce least-privilege access, role-based access control or other fine-grained access controls.

Misuse of Privileges

If the system is not configured correctly, it could lead to misuse of user privileges. For instance, if network policies do not restrict access between

Kubernetes Threat Models at a Glance

Threat	Cause	Actions
External attacks	API server, kubelet or etcd components are compromised.	Only expose the necessary Kubernetes services. Always enforce authentication. Configure the right network security policies for any exposed services.
Compromised containers	A container escalates privilege to take control of other containers or the cluster.	Utilize Kubernetes' isolation mechanisms, such as namespaces or network segmentation. Utilize built-in OS-level controls. Limit the number of containers that can run in a privileged mode.
Compromised credentials	A malicious user gains access to the system.	Enforce least-privilege access, role-based access control or other fine-grained access controls.
Misuse of legitimate privileges	The system is not configured correctly and user privileges are misused.	Ensure all system components are properly hardened. Design and implement authorization correctly. Utilize Kubernetes' plugin architecture, which allows sophisticated authorization logic.

TABLE 3.1: *In the Kubernetes environment, there are generally four types of threats that apply regardless of the deployment method.*

pods or namespaces, a user might be able to read traffic for other namespaces that he or she should not have access to in the first place.

A major defense against misuse of privileges is proper hardening. Ensure that all the system components, such as containers, pods, namespaces and kubelets, are hardened to significantly reduce the possibility of privilege misuse.

Another important defense is authorization control. Kubernetes supports a plugin architecture, which allows sophisticated authorization logic to be included. Design and implement authorization correctly; it's one of the most effective weapons against privilege misuses.

Security Considerations for Kubernetes

With the threat model in mind, we explore Kubernetes security along four major tenets: authentication and authorization; resource isolation; hardening and network security; and logging and auditing. We look at security from the perspective of containers, Kubernetes deployment itself and network security. Such a holistic approach is needed to ensure that containers are deployed securely and that the attack surface is minimized. The best practices that arise from each of the above tenets apply to any Kubernetes deployment, whether you're self-hosting a cluster or employing a managed service.

We should note that there are related security controls outside of Kubernetes, such as the Secure Software Development Life Cycle (S-SDLC) or security monitoring, that can help reduce the likelihood of attacks and increase the defense posture. We strongly urge you to consider security across the entire application life cycle rather than take a narrow focus on the deployment of containers with Kubernetes. However, for the sake of brevity, in this chapter we will only cover security controls within the immediate Kubernetes environment.

Authentication and Authorization

The Kubernetes APIs are the central interfaces for administrators, users and applications to operate and communicate in the Kubernetes environment. Both users and service accounts can access Kubernetes APIs to initiate operations. As such, controlling API access is the main task of authentication and authorization within Kubernetes.

The Kubernetes platform has built-in authentication and authorization controls, as well as admission controls, which intercept and regulate requests to the Kubernetes APIs after authentication and authorization. Admission controls include built-in constructs as well as webhook-enabled methods that can be used to invoke external logic.

Authentication and authorization are at the core of Kubernetes security. Securing access to the Kubernetes API server is therefore one of the first priorities of a secure Kubernetes environment. Later in this chapter, we will explore in-depth authentication and authorization options and policies.

Resource Isolation

Isolation of resources is another major security lever within Kubernetes. Isolation not only prevents denial-of-service attacks, but also provides privacy and data protection. The Kubernetes platform provides isolation mechanisms for a number of resource types, including pods and namespaces. The resource limits you can place on pods and namespaces include central processing unit (CPU) cycles, memory requests and persistent storage space.

Hardening and Network Security

Environment hardening, including hardening of containers and the underlying Kubernetes infrastructure, is essential to the security of a Kubernetes environment. It helps defend against threats brought by compromised containers, misuse and mis-configurations.

Hardening operations include restrictions on running privileged containers, limiting privilege escalations and whether a container can access the host networking interface and file system.

Network security handles segmentation, secures API access with Transport Layer Security (TLS) client authentication and manages service network Access Control Lists (ACLs).

Logging and Auditing

In addition to native application and system logging, it may be beneficial to have Kubernetes-specific logs to understand Kubernetes operations, such as who accessed which Kubernetes API.

Kubernetes 1.9 provides a beta feature — Audit Logging — to perform separate logging and auditing functions. Audit Logging records actions taken by the API. The records can then be archived for later analysis. An administrator can specify which events should be logged by specifying an audit-policy YAML file.

Container Security

Before we dive into the security aspects of Kubernetes, we must first understand the relevant security issues with the container infrastructure.

Kubernetes supports Docker containers and, experimentally, the rkt container format. For the rest of this chapter, all discussions on containers center on Docker containers.

Node Security

To run containers in a secure fashion, each Linux node must be properly configured and hardened. The Center for Internet Security (CIS) benchmarks for Docker and a corresponding CIS benchmark for Kubernetes contain many hardening guidelines that operational teams should follow.

For instance, one of the recommended practices is to enable built-in Linux security measures, such as SELinux and Seccomp profiles. SELinux is a kernel-level capability that regulates access to files and network resources, while Seccomp profiles restrict the set of system calls an application can make. Together, these capabilities allow a level of fine-grained control over the workloads that run on the node.

In general, major considerations of node security include:

- **Securing node communications** with a TLS client certificate, to ensure all critical API access points are secured with end-to-end TLS.
- **Enabling relevant kernel-level security controls** like SELinux or Seccomp. These capabilities help to limit the attack surface on the node, thereby giving greater control over security of the entire system.
- **Limiting direct access**, e.g., Secure Shell (SSH) access, to Kubernetes nodes: Forcing all access to nodes via Kubernetes ensures proper access control and logging. This helps to reduce risk for unauthorized access to host resources.
- **Follow industry best practices**, such as CIS Docker Benchmark, to properly configure and harden the Linux nodes that run containers.

Container Image Security

The most important aspect of container image security is managing vulnerabilities. Because running containers with vulnerabilities exposes your system to attacks and compromises, you must actively manage the images used in your system to discover and remove known vulnerabilities.

A number of commercial and open source packages can perform container image scanning to discover known Common Vulnerabilities and Exposures (CVE) identifiers. The trick is not to stop at scanning. Rather, the scanning function should be integrated with runtime enforcement and remediation capabilities.

For runtime enforcement, consider a process that deploys only those images that pass vulnerability scanning and those that adhere to the organization’s hardening policies. Kubernetes provides mechanisms to exercise such enforcement policies.

For remediation, ensure that the vulnerability scanning and security assessment function is integrated with the organization's continuous integration/continuous deployment (CI/CD) pipeline. This way, the scanning results are fed directly to the pipeline, thereby kicking off remediation workflows before deployment. This should, in turn, be integrated with Kubernetes' rolling updates feature, ensuring vulnerable containers are taken offline and replaced with freshly built images without the known flaws.

Container Registry Management

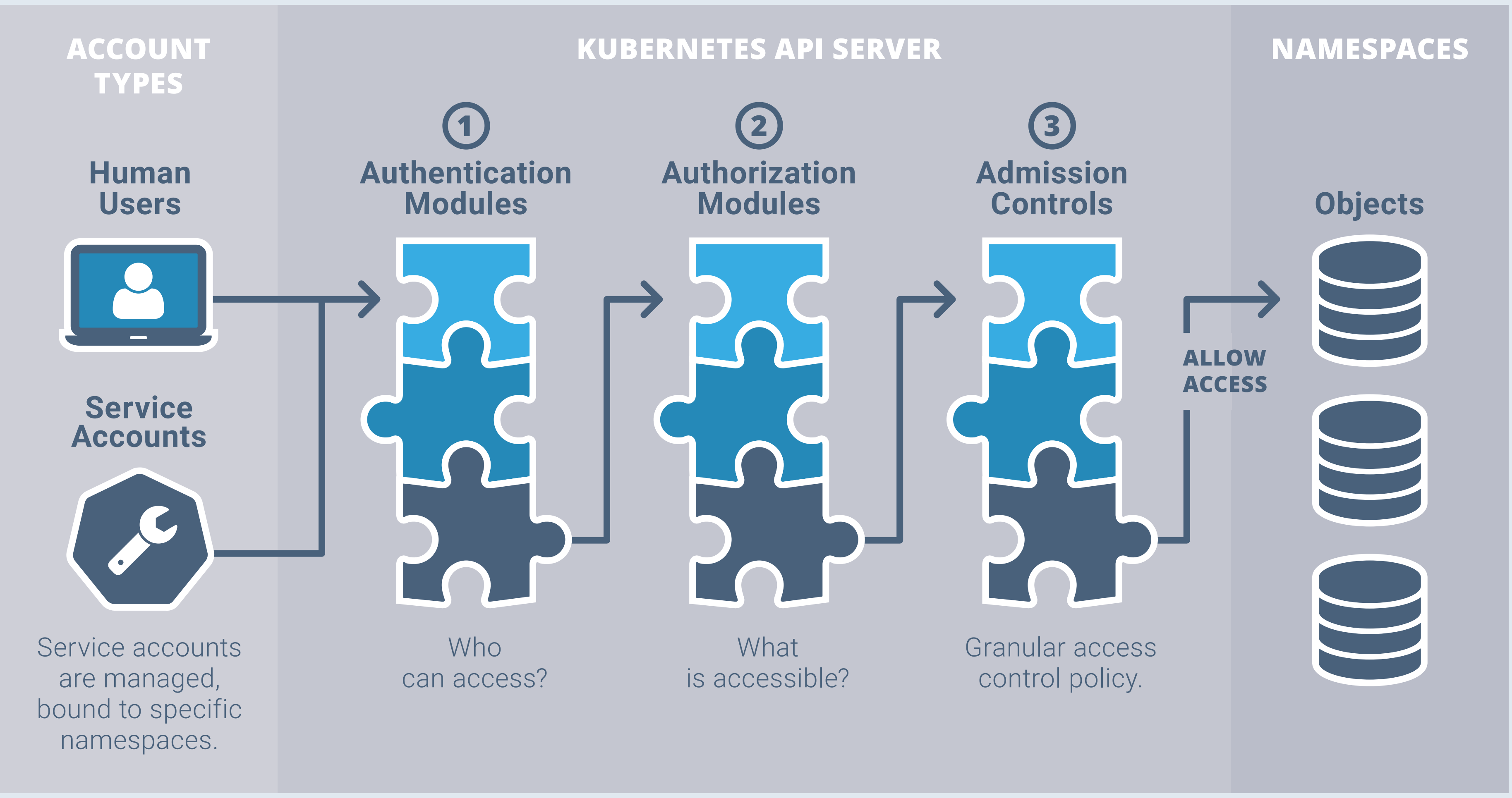
Container registries are the source from which images come. You must manage which registries can be used by your organization to pull images because downloading images from unknown registries can lead to the proliferation of vulnerable and dangerous software.

Use private registries and approved images; that is, only scanned and vetted images can be pushed into your private registries. If you must use public registries, scan the images before deployment. If the security scanning process fails to clear the image, you must fail deployment as a result. That is the only way to ensure the hygiene of your container environment.

Deployment Security

In this section, we focus on the implementation of security functions for each of the four major tenets of Kubernetes security: authentication and authorization, resource isolation, network security and logging and auditing. These may be implemented with built-in Kubernetes controls and configuration options as well as the Kubernetes plugin infrastructure that allows external policies to be instantiated for greater levels of control.

API Requests Are Authenticated and Authorized With Three Kubernetes Modules



Source: https://kubernetes.io/docs/admin/accessing-the-api/

FIG 3.1: *A high-level depiction of how Authentication, Authorization, and Admission Control modules are used to control API access in Kubernetes.*

User Authentication and Authorization

In Kubernetes, both users and service accounts can access Kubernetes APIs. Each API request should be authenticated and authorized prior to gaining access.

Unlike a normal user, service accounts are managed, bound to specific namespaces and associated with credentials known as Secrets. As such, the authentication and authorization processes for service accounts are radically different than those for human users.

The Kubernetes authentication and authorization architecture is a layered process. Depicted in Figure 3.1, this process invokes authenticator modules first, before authorization modules, and finally the request is passed through admission controls (if any), before it attains access to any objects.

To enable authentication, the cluster administrator must configure the API server to run one or more authentication modules at cluster creation time.

User Authentication

The authentication operation in Kubernetes is accomplished via the use of Authentication Modules, also known as authenticators (see Figure 3.1). Kubernetes supports a variety of built-in authenticator modules to authenticate an API request from a user.

Kubernetes authenticators allow the use of Client Certificates, Passwords, Static Tokens, Bootstrap Tokens and JSON Web Tokens (JWT) for authentication purposes. You can program additional authentication logic by using an authentication proxy or the webhook method of authentication.

If the administrator specifies multiple authenticators, they will be invoked in sequence. The user is considered authenticated when one of the modules returns a success result. If none of the modules are successful, the server rejects the request with a 401 error code.

- **Client Certificates:** Passing the `--client-ca-file=FILENAME` to the API server enables client certificate-based authentication. The file referenced must contain information about the certification authority to validate the certificate. If a client certificate is presented and verified, use the common name of the subject as the user name for the request.

- **Static Tokens:** Pass the `--token-auth-file=FILENAME` option on the command line to tell the API server to use the static bearer token authentication method. The file referenced must contain the actual tokens.

- **Passwords:** Pass the `--basic-auth-file=FILENAME` option to the API server to indicate the password authentication method. Note that the use of static passwords is not considered a secure mode of authentication. Also, at present, the password credentials cannot be changed without restarting the API server.

- **Bootstrap Tokens:** Currently an alpha feature, Bootstrap Tokens are stored as Secrets in the system namespace. The difference between a Bootstrap Token and a static token is that the former is dynamically created and managed. You can enable Bootstrap Token Authenticator with the `--experimental-bootstrap-token-auth` flag.

One more authentication category worth noting is service account tokens, which are JSON Web Tokens. A service account is usually attached to a pod in the cluster. A service account token is mounted into the pod and its presence allows the pod to access the API server. Note that users with read access to pod secrets can read the token and authenticate to the API server as the service account.

User Authorization

For user authorization, Kubernetes supports multiple authorization modules such as Node, Attribute-Based Access Control (ABAC), Role-Based Access Control (RBAC) and Webhook.

- **Node:** Node is a special-purpose authorization module that specifically deals with API requests by kubelets. A Node authorizer handles read, write and auth-related API operations.

- **ABAC:** An ABAC module assesses attributes of an access request against an access control policy to grant or deny access.

- **RBAC:** RBAC controls access to Kubernetes API resources based on the assigned role of the user.

- **Webhook:** In this context, Kubernetes uses Webhook to query an external service on whether to grant or deny the current API request. The webhook method provides a means for Kubernetes to incorporate a third-party policy and potentially implement arbitrary logic for authorization decisions.

For this discussion, we will focus on the authorizers that deal with user actions. Readers who are interested in learning more about Node Authorization, please refer to the Kubernetes documentation.

To use the authorization modules, an administrator must configure them at the time of cluster creation. If multiple modules are configured, they are linked together in a logical "OR" fashion for granting access — if any module authorizes the request, the request can proceed. But only if all of the modules deny the request, can the access request be denied.

Attribute-Based Access Control (ABAC)

For ABAC, attributes used for authorization may include user identity, user group, namespace, resource name and API request verb (the action being requested), among others. The ABAC policy is a JSON file, which must also be specified at the time of cluster creation.

Below is a sample authorization policy for user Alice. The policy gives Alice read permission to the namespace AccountInfo.

```
{
   “apiVersion”: “abac.authorization.kubernetes.io/
v1beta1”,
   “kind”: “Policy”,
   “spec”: {
      “user”: “Alice”,
      “namespace”: “AccountInfo”
      “resource”: “pods”,
      “readonly”: true
   }
}
```

If Alice makes a request to read an object within the AccountInfo namespace, the request will be granted as Alice has read access to the namespace. In contrast, if Alice makes a write request to any object inside the AccountInfo namespace, the request will be denied.

To enable ABAC authorization control, an administrator can start the API server with the flag: `--authorization-mode=ABAC`. In addition, you need to specify the authorization policy file by including this flag:

```
--authorization-policy-file=MyPolicyFile.json
```

Role-Based Access Control (RBAC)

A Kubernetes user role is similar to roles used in classic RBAC in that it encapsulates a set of logically-grouped permissions. For instance, you can create a role that has read access to a particular resource by combining the request verbs “get,” “watch” and “list,” in a single role.

A role can be cluster wide, denoted as `ClusterRole`, which can be used to grant access rights to resources across different namespaces in the same cluster. Or, it can be applied to a single namespace, in which case it is simply denoted as `Role`.

RBAC provides a mechanism for administrators to edit roles to efficiently prevent unwanted privilege escalations. Below is an example role definition:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
```

```
rules:
- apiGroups: [“”]
  resources: [“pods/log”]
  verbs: [“get”, “list”, “watch”]
- apiGroups: [“batch”]
  resources: [“jobs”]
  verbs: [“get”, “list”, “watch”, “create”, “update”,
“patch”, “delete”]
```

This cluster role can read resources pod logs in the core API group and can read/write “jobs” in the batch API group. When a user with this cluster role issues a request to read pod logs of the core API group, the request will be granted.

To enable RBAC authorization control, an administrator can start the API server with the flag:

```
--authorization-mode=RBAC
```

Webhook

As previously mentioned, the webhook authorization method allows Kubernetes to query an external representational state transfer (REST) service for authorization decisions. This method comes in handy when there are specific policies for determining user privileges that cannot be easily expressed with existing authorization controls. For instance, if you wish to incorporate authorization logic that takes into account dynamic context of the access, such as location of the machine that initiated the request or the time of day, it is best to use a separate authorization (AuthZ) function to implement this logic, which can be invoked via Webhook.

To use Webhook, one needs to specify a configuration file, which defines the remote AuthZ service, the certificate of the service and other information necessary to access via webhook.

When a request subjected to a webhook authorizer comes in, the API server will package up pertinent information about the request in a JSON object and POST to the remote service, as specified in the config file. The JSON file might include information such as the user identity, the group in which the user belongs and the resource that the user is trying to access (namespace, operation, etc.). The remote service will return a “true” or “false” for the “allowed” field.

To enable the webhook method of authorization, an administrator can start the API server with the flag:

```
--authorization-webhook-config-file=MyAuthZConfigurFile
```

Admission Control

After a request to the Kubernetes API server is authenticated and authorized, but before the request is fully accepted, you can utilize admission controllers to enable advanced features of access control.

Admission control is used to specify granular access control policies that go beyond user identities, resources, operations and namespaces. Rather, it is best used if you wish to define policies based on the specific operational context of the container or the resources.

Admission control plugins must be compiled into the API server library to take effect. If multiple plugins are defined, they will be run in sequence. A request can only proceed when all the admission controllers pass it. If any of the plugins reject the request, the access request is rejected immediately.

To use admission controls, you must start the Kubernetes API server with the flag of `admission-control`, which takes a comma-delimited, ordered list of controller names. For example, you might see a command that looks like this:

```
--admission-control=NamespaceLifecycle,PersistentVolumeLa
bel,PodSecurityPolicy, DenyEscalatingExec
```

Kubernetes 1.8 and 1.9 come with a set of built-in admission controls. For this discussion, we will focus on two controllers: PodSecurityPolicy and DenyEscalationExec.

PodSecurityPolicy

PodSecurityPolicy is a key admission control in the Kubernetes environment. It allows the administrator to specify hardening constraints, such as restricting privileged containers or privilege-escalating operations. Once PodSecurityPolicy is specified, the target pod must run with the conditions specified within the PodSecurity object in order to be admitted into the cluster.

More specifically, PodSecurityPolicy allows an administrator to control these security aspects of the pod:

- Whether to allow running of privileged containers.
- Whether to allow the pod access to the root namespaces, host networking and ports, and the host filesystem.
- Whether the root filesystem should be read-only.
- Whether to allow privilege escalation.
- When applicable, the AppArmor and Seccomp profiles.

An example PodSecurityPolicy in YAML may look like this:

```
  apiVersion: extensions/v1beta1
```

```
kind: PodSecurityPolicy
metadata:
  name: Example
spec:
  privileged: false
  allowPrivilegeEscalation: false
  runAsUser:
    rule: ‘MustRunAsNonRoot’
  hostNetwork: false
  seLinux:
    rule: ‘RunAsAny’
  fsGroup:
    rule: ‘RunAsAny’
```

With this policy, the target pods must satisfy these conditions:

- Containers cannot run in privileged mode.
- Containers that require root privileges are not allowed.
- Containers cannot access the host’s network directly.
- Privilege escalation is not allowed.

It is important to note that pod security policies can only be used if they are enabled in the Kubernetes cluster’s admission controller.

Restricting containers from running as root, and in general creating a more secure cluster environment are the most common uses for pod security policies.

DenyEscalatingExec

If your pod runs privileged containers, it is a good practice to consider the use of the DenyEscalationExec plugin. It denies exec and attach commands to pods that run with escalated privileges.

If a user performs an exec or an attach command on a privileged container, the user may attain privileges that he or she cannot attain otherwise. This is why DenyEscalatingExec was created — if enabled, the minute a user issues an exec command on the privileged container, the admission controller will block that specific request to prevent privilege escalation.

If you run containers with escalated privileges, you should strongly consider enabling the DenyEscalatingExec plugin to mitigate your risk against accidental privilege escalation. For those reasons, DenyEscalatingExec is a key hardening control in Kubernetes.

Container Runtime Resource Isolation

Within the Kubernetes platform, there are two separate mechanisms via which resources are isolated: Namespaces and Resource Quota.

Namespaces

Kubernetes Namespaces are a method of virtual partitioning to allow multiple groups of users to share the same cluster, yet still retain isolation. Resources within one namespace are not visible from other namespaces.

Namespaces is a useful construct in environments with different users across many teams. You can create specific namespaces and assign users and resources to them, and have different policies associated with different namespaces.

Kubernetes starts with three initial namespaces:

- **Default:** The default namespace for objects with no other namespace.
- **kube-system:** The namespace for objects created by the Kubernetes system.

- **kube-public:** The namespace that is readable by all users. It’s reserved for resources that should be publicly readable throughout the whole cluster.

An administrator can define additional namespaces. Whenever there is a business case to partition a group of users while maintaining enterprise central control you can use namespaces. For example, a common practice is to have separate namespaces for development, testing/QA, staging and production.

One of the biggest benefits of using namespaces is that you can use the same name for services or endpoints across the different namespaces. This means you don’t have to change the name of a service in testing to a different name in staging or production. For instance, you can have account_balance in testing and use exactly the same account_balance in staging or production without fear of name collision. But, you can also use full names like account_balance.dev.myapp to uniquely refer to the account_balance service in the development namespace.

API access requests and authorization policies can all target a specific namespace. For instance, you might see a request like this:

```
$ kubectl --namespace=AccountInfo run nginx --image=nginx
```

This request asks to run the NGINX image within the namespace called AccountInfo. The ABAC policy below allows the user Alice to read pods within the AccountInfo namespace.

```
{
   “apiVersion”: “abac.authorization.kubernetes.io/
v1beta1”,
   “kind”: “Policy”,
   “spec”: {
      “user”: “Alice”,
      “namespace”: “AccountInfo”
      “resource”: “pods”,
      “readonly”: true
   }
}
```

Resource Quotas

A recommended practice is that you do not run resource-unbound containers because it puts your system at risk of resource starvation or a denial-of-service attack. To minimize these risks, Kubernetes provides resource quotas for administrators to define resource constraints that apply to different namespaces.

A resource quota, defined by a `ResourceQuota` object, puts limits on resource consumption for each namespace. For example, it can limit how many pods (or other objects) can run in a namespace and the amount of memory requests and CPU cycles that may be consumed by resources in that namespace, as well as limit storage consumption.

Kubernetes enforces resource consumption for a particular namespace when a `ResourceQuota` object is defined for that namespace in a `compute-resources.yaml` file. Below is an example of such a YAML file for the AccountInfo namespace.

```
$ kubectl create namespace AccountInfo

$ cat <<EOF > compute-resources.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
   name: compute-resources
spec:
   hard:
      pods: “7”
      requests.cpu: “1”
      requests.memory: 2Gi
      limits.cpu: “3”
      limits.memory: 3Gi
EOF
```

This file specifies that there can be at most seven pods running in the namespace. The memory request total for all containers must not exceed 2 gibibytes (GiB). The memory limit total for all containers must not exceed 3 GiB. The CPU request total for all containers must not exceed 1 CPU. The CPU limit total for all containers must not exceed 3 CPU.

You can enable Resource Quotas by including `ResourceQuota` as one of the `--admission-control` flag arguments.

Similarly, you can limit the storage resources, object count and local ephemeral storage for a particular namespace.

Logging and Auditing

Logging and auditing is an important security function. Not only can you use the logs for monitoring, debugging and forensics, you can also use them to satisfy compliance requirements and generate compliance reports.

In Kubernetes, it is useful to maintain cluster-level logs that are independent from the logs generated by the container application, the container engine or the host system. To that end, Kubernetes 1.9 provides a beta function — Audit Logger — to perform separate logging and auditing functions.

Audit Logger records actions taken on the APIs by users, administrators or other system components. More specifically, it records pertinent information about an action, such as the action that happened, the time when the action happened, the user that initiated the action, the resources that the action affected and more.

An administrator can specify which events to be logged and how they are logged by defining an audit-policy YAML file. For example, this audit-policy file specifies metadata-level logging for access to the "secrets," "Configmaps" and "Tokenreviews" resources. Because these resources may contain sensitive data, only log at the metadata level.

```
{
   apiVersion: audit.k8s.io/v1beta
   kind: Policy
   Rules:
      - Level: Metadata
      - Resources
         - group: ""
         Resources: ["secrets", "Configmaps"]
         - group: "authentication.k8s.io"
         Resources: ["Tokenreviews" ]
}
```

Network Policies

To minimize the risk of a compromised application attacking another application in the same namespace, it is important to consider network segmentation to ensure that pods only communicate with approved pods and network sources.

By default, Kubernetes pods accept traffic from anyone. To isolate a pod and restrict who can talk to it, you can use the NetworkPolicy resource. Once enabled, it determines how pods can communicate with each other and with other network endpoints.

Each NetworkPolicy must specify the group of pods to which the policy applies. A NetworkPolicy that does not specify pods applies to all pods in the namespace. In addition, the NetworkPolicy must define whether its policies are for egress or ingress traffic.

Below is an example NetworkPolicy that denies all ingress traffic from pods that are labeled "db" to all pods in the namespace.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
   name: default-deny
spec:
   podSelector: {}
   policyTypes:
   - Ingress

   - from:
   - podSelector:
   matchLabels: "db"
```

The NetworkPolicy construct can be used to configure automatic and dynamic firewall rules, which are deployed when certain conditions are met. However, to achieve true microsegmentation, the policies must go beyond ingress, egress, IP and ports.

Suggested Best Practices

It is important to understand that many open source distributions of Kubernetes may not have every security feature enabled or even included by default. Therefore, it requires intimate knowledge of the Kubernetes platform to configure the system properly and ensure secure operations.

As an alternative to managing your own Kubernetes clusters, you could consider using commercial management platforms such as Red Hat OpenShift and CoreOS Tectonic. These uber-management platforms typically include already configured security plugins and policies, which can be a good starting point.

For those who prefer to configure and manage Kubernetes directly, some emerging commercial products can make your task a little easier. Alcide, Aqua Security, Cavirin, HyTrust and Twistlock offer products that let you specify a high-level policy, which can be translated automatically to Kubernetes-native policies and controls.

Of course, Kubernetes itself provides many options to craft your specific security policies and enforcement. We recommend that you start with these simple actions:

Configure PodSecurityPolicy to Properly Control Runtime Privileges

Make sure that you configure PodSecurityPolicy properly to set proper privileges and rights for your namespaces, pods, containers and volumes. Some of the important questions to ask are:

- Does the pod need to run in a privileged mode?
- Does privilege escalation need to be allowed?
- Does the pod need to be allowed access to the host network, process

ID and inter-process communication (IPC) namespace?

- Do containers need to be run as root?

For each of the questions, choose “no” by default unless you absolutely have to say “yes.” In this case, the conservative approach is usually the safer choice, because even the simple option of permitting pods access to host network (i.e., “hostNetwork=true”) can result in unwanted privilege escalations.

Enable Authorization Plugins and Appropriate Admission Controls

Authentication and authorization plugins can help you define fine-grained access control policies for your Kubernetes resources. Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC) are critical tools to contain cluster-wide compromises. Spend time to determine which are the right controls for your environment. For example, enabling certain admission controllers such as DenyEscalatingExec is a good hygiene practice, but may impede special operations or corner use cases.

Use a NetworkPolicy Object to Restrict Access to Services

Service-level security is critical to a microservices environment. In addition to enabling client certificates and ensuring proper authentication and authorization, a Kubernetes administrator should utilize NetworkPolicy as an important construct to provide network isolation. By imposing restrictions on who can talk to whom, an administrator can reduce the system attack surface and improve the cluster’s security posture.

Manage Cluster Vulnerabilities Proactively

Vulnerabilities within a cluster present opportunities for exploits and attacks. These may either arise from a container image or from the environment, such as the operating system or even at the hardware level (e.g., the recent Spectre/Meltdown zero days). As such, an administrator must manage cluster vulnerabilities proactively. That means utilizing vulnerability assessment and management tools throughout the life cycle of the application, the environment and the deployment process to ensure that only robust code is deployed and that the environment is free of known defects and configuration vulnerabilities.

Control Security From a Life Cycle Standpoint

Making Kubernetes secure is not just a runtime proposition. Rather, security needs to be treated from an end-to-end life cycle perspective. This means the administrator must:

- Ensure only approved code is built into the images and only approved images make it into production.
- Configure and harden the node environment.
- Ensure the right set of controls are enabled for the runtime environment.
- Log everything and log always.

SECURING A KUBERNETES DEPLOYMENT

What is the context for Kubernetes if the data center is Mount Olympus? Alcide co-founder and CTO Gadi Naor makes this analogy between the mythic Mount Olympus, home of the gods, and the modern data center in this podcast interview with The New Stack founder and Editor-in-Chief Alex Williams.

The comparison brings into play a concept about how the overall organization views infrastructure. The data center, now empowered by Kubernetes, is a core asset that must be valued and protected the way Hercules guarded Olympus.

Cloud-native technologies are considerably complex. Existing network security does not meet the new demands that come with containers. It may provide a top-view policy, but there needs to be more than that. What's needed is what Naor calls "policy fusion," that allows for multiple policies to be unified in one cohesive manner so security may run at scale and organizations may take true advantage of Kubernetes' scalability. **Listen on SoundCloud.**

Gadi Naor, *co-founder and CTO of Alcide, brings 15 years of experience in leading the development of cyber security products. Previously, he worked at CheckPoint leading the development of Firewall core security engine and VPN software. Naor also served as a senior software engineer at Altor Networks, a pioneer in virtualized data center security, later acquired by Juniper Networks.*

CLOSING

Kubernetes is like a hive. It's constantly managing microservices that traverse across its connected infrastructure. There is no single, correct way to assess it. No tidy command or single pane of glass that will give you visibility and clarity. To make sense of this complex infrastructure, it helps to evaluate Kubernetes by focusing on the value your organization puts on factors such as security, storage, logging and load balancing. These factors help define how companies manage workloads, whether on cloud services, on-premises or through a multi-cloud approach.

Kubernetes has fast emerged as the standard for container orchestration and plays a critical role in enterprise IT modernization. In this book, our focus has been on informing the operations teams that deploy and manage the clusters needed for containerized applications. As more and more workloads move to Kubernetes-based container deployments, IT operations teams need a lens for understanding various deployment and security patterns.

With this second book in The New Stack's Kubernetes series, we've aimed to help cluster operators deploy Kubernetes to manage containerized workloads. This book has covered various deployment patterns and security considerations that will help them in the process. The next, and final, part in our Kubernetes ebook series will cover application patterns for Kubernetes and delivery best practices for developers and DevOps professionals. Taken as a whole, this three-part series will give you a firm foundation to prepare your own Kubernetes deployments and understand this complex project and ecosystem. But reading and planning can only get you so far — the next, and best, way to learn what works for your organization is to go out there and try it. Best of luck with your Kubernetes deployments and godspeed!

DISCLOSURE

In addition to our ebook sponsors, the following companies mentioned in this ebook are sponsors of The New Stack:

AppDynamics, Blue Medora, Buoyant, CA Technologies, Chef, CircleCI, Cloud Foundry Foundation, Cloud Native Computing Foundation, {code}, Codeship, Containership, CoreOS, Dell Technologies, DigitalOcean, Google, HPE, InfluxData, Mesosphere, Microsoft, OpenStack Foundation, PagerDuty, Portworx, Puppet, Red Hat, SaltStack, StackRox, The Linux Foundation, Univa, VMware, Wercker and WSO2.

thenewstack.io