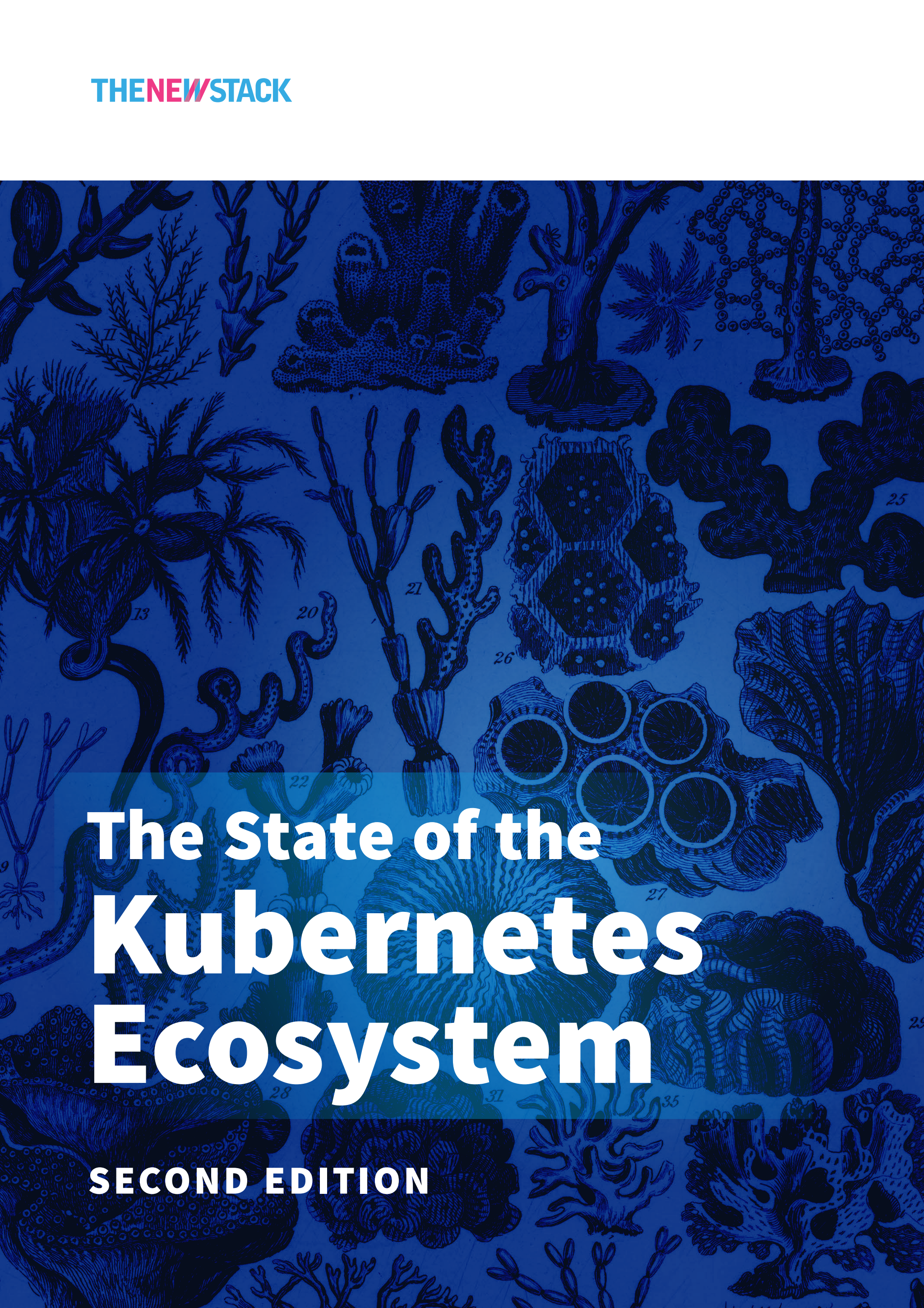


THE NEW/STACK



The State of the Kubernetes Ecosystem

SECOND EDITION

The New Stack

The State of the Kubernetes Ecosystem, Second Edition

Alex Williams, Founder & Publisher

Ebook Team:

Gabriel H. Dinh, Executive Producer

Janakiram MSV (Janakiram & Associates), Author

Jonathan Gold (Container Solutions), Technical Editor

Judy Williams, Copy Editor

Lawrence Hecht, Contributing Research Analyst

Libby Clark, Editorial & Marketing Director

Richard MacManus, Ebook Editor

Sebastien Goasguen (TriggerMesh), Technical Editor

Supporting Team:

B. Cameron Gain, EU Correspondent

Benjamin Ball, Director of Sales and Account Management

Colleen Coll, Manager of Digital Media Operations & Marketing

Eddie Rogers, Digital Media Producer

Jennifer Riggins, UK Correspondent

Joab Jackson, Managing Editor

Michelle Maher, Editorial Assistant

© 2020 The New Stack. All rights reserved.

20201005

Table of Contents

Sponsors & Partners	4
Credits.....	6
Introduction.....	7
Kubernetes, The Operating System for the Cloud.....	11
Deployment Size Matters	14
What Does Orchestration Mean?.....	17
Kubernetes Architecture.....	20
How Kubernetes Delivers the Promise of Web-Scale Computing.....	35
Adopting Kubernetes	36
Kubernetes as a Universal Control Plane	38
DataStax: Self-Service Architectures & the Kubernetes Operator for Cassandra.....	42
Dynatrace: AI Observability Cuts Through Kubernetes Complexity.....	43
Mapping the Kubernetes Ecosystem	44
The Rise of Cloud Native and CaaS.....	45
Key Attributes of a Container Management Platform	46
The Big Picture of Container Management Platforms.....	47
Enterprise Container Management Platforms.....	68
Managed Container Management Platforms	71
Uptake for Cloud Provider Kubernetes as a Service.....	74
Prisma by Palo Alto Networks: Kubernetes Has Evolved, So Should Your Security..	76
KC+CNC: Kelsey Hightower on His Very Personal Kubernetes Journey.....	77
Closing	78
Disclosure	81

Sponsors & Partners

We are grateful for the support of our ebook sponsors:



DataStax is the company behind the massively scalable, highly available, cloud-native NoSQL data platform built on Apache Cassandra. DataStax gives users and enterprises the freedom to run data in any cloud at global scale with zero downtime and zero lock-in.



Dynatrace is the leader in Software Intelligence, purpose built for the enterprise cloud. It's the only AI-assisted, full stack and completely automated intelligence platform that provides deep insight into dynamic, web-scale, hybrid cloud ecosystems. That's why the world's leading brands trust Dynatrace to deliver perfect user experiences.



KubeCon + CloudNativeCon conferences gather adopters and technologists to further the education and advancement of cloud native computing. The vendor-neutral events feature domain experts and key maintainers behind popular projects like Kubernetes, Prometheus, Envoy, CoreDNS, containerd and more. KubeCon + CloudNativeCon is run by the Cloud Native Computing Foundation (CNCF).



Governed access plus pervasive protection for data, apps, hosts, containers and serverless — this is the proper foundation for the journey to the cloud. Prisma, the industry's most comprehensive cloud security suite, helps customers accelerate their journey with risk visibility and continuous security.

And our partners:

Container Solutions is a professional services company that specializes in Cloud Native transformation. We bring together technology, culture, and strategy to make sure transformations are done right. Container Solutions has offices in Amsterdam, London, Berlin, Montreal, Prague, Warsaw, and Zurich.



TriggerMesh provides a real-time cloud native integration platform that allows you to connect services together to automate workflows and accelerate the flow of information across your organization. TriggerMesh enables enterprise developers to build applications that are event-driven and composed of services from multiple cloud providers and on-premises systems.

Credits



[Janakiram MSV](#) is the author of this ebook, Principal Analyst at Janakiram & Associates and an adjunct faculty member at the International Institute of Information Technology. He is also a Google Qualified Cloud Developer, an Amazon Certified Solution Architect, an Amazon Certified Developer, an Amazon Certified SysOps Administrator and a Microsoft Certified Azure Professional. Janakiram is an Ambassador for the Cloud Native Computing Foundation, and also one of the first Certified Kubernetes Administrators and Certified Kubernetes Application Developers. His previous experience includes Microsoft, AWS, Gigaom Research and Alcatel-Lucent.



[Lawrence Hecht](#) is the contributing research analyst for this ebook, and has been producing research reports about information technology markets for the last 17 years. Lawrence previously managed “voice of the customer” surveys for 451 Research and TheInfoPro about enterprise IT B2B markets such as Cloud Computing, Data Analytics and Information Security. In 1999, he created the Internet Public Policy Network (IPPN), a network of subject-matter experts that provided custom research, white papers and advice about technology-related public policy issues. Lawrence works with writers to identify data to develop actionable insights based on primary and secondary research. He currently uses these skills to create content for thought leadership and peer evangelism programs. Lawrence earned a Master of Public Policy from Georgetown University and Bachelor of Arts from Rutgers University.

Introduction

In June of 2015, just five years ago, people were struggling with the name and the concept behind Kubernetes, a project that had emerged out of Google earlier that year. “Kuber, Kuber ... what?”

Kubernetes was named after the ancient Greek term for the helm of a ship. It was a technology architecture that spoke to a shift that had been building for several years prior to 2015, with the rise of internet-scale businesses. Docker was still popular, but the community was starting to explore ideas around container orchestration. In June 2015, a new organization called the Cloud Native Computing Foundation (CNCF) was announced and Kubernetes had a new home.

Scale-out architectures had arrived. Docker was the first mega open source project. Kubernetes became the next project to watch, growing over the next few years into a movement that came to define the intersection of distributed computing, open source communities, and the advancement of application architectures. It quickly became one of the most significant open source projects of the past 20 years. Some even considered it the cloud’s version of Linux.

In 2017, to document the movement, The New Stack published our first ebook about the Kubernetes community and the accompanying cloud native ecosystem.

Since 2018, the community has changed so much and developed so fast that the first edition was starting to show its age. It needed updating and a new perspective to reflect the Kubernetes community of today and the direction it’s going.

Almost 80 percent of companies surveyed by the CNCF now use Kubernetes in production, according to a study conducted in 2019.

Kubernetes and cloud native technologies are reflected in the popularity of the KubeCon + CloudNativeCon event, which saw attendance increase from about 1,000 people in 2016 to 4,000 people at the 2017 North America show. Last fall, the

KubeCon event in San Diego surpassed 10,000 attendees. In 2020, KubeCon will be a virtual conference, with a potentially much larger attendance.

At the time of publishing, everyone is home and COVID-19 keeps open source participation a virtual exercise. But Kubernetes' growth continues ... as does the consolidation.

The CNCF landscape currently has a market cap of \$17.8 trillion, representing 570 member companies. It's an aggregated total of companies, reflecting how much investment is in the overall community. There have also been some notable acquisitions. In January of 2018, CoreOS, the software company behind container runtime Tectonic, sold to Red Hat for \$250 million. In late 2019, VMware bought Heptio, the company founded by Joe Beda and Craig McLuckie, who along with Brendan Burns led the original team at Google that developed Kubernetes. Burns had also left Google by then, to pursue work at Microsoft. The deal to purchase Heptio was worth a reported \$450 million. In May of 2019, Palo Alto Networks acquired Twistlock, a container security company, in a deal valued at \$410 million. Also last May, VMware acquired Bitnami. Earlier this year, VMware announced the purchase of Pivotal, which had led the commercialization of Cloud Foundry, the open source Platform as a Service. In July of 2020, just prior to the publishing of this ebook, SUSE acquired Rancher Labs for a reported \$600 million.

The growth of the community, along with the maturing of microservices as a development architecture model, has led to a deeper knowledge of container orchestration technologies and distributed computing in general.

It's now more understood that Kubernetes requires an underlying pool of resources for compute, storage and networking that can be run via on-premises infrastructure, or through a cloud service. Still, the underlying core architecture is helpful to explain for the beginner or more experienced user.

In this ebook, The New Stack provides a grounding in the basics of Kubernetes. The goal is to provide a foundation of understanding about what distinguishes the

concepts of control planes, worker nodes, service discovery, storage and networking. It explores how Kubernetes delivers the capabilities of web-scale computing, adoption patterns and its role as a universal aspect of a company's core infrastructure.

Container management platforms are the focus of the second chapter of the ebook. A variety of platforms are offered now by vendors and cloud services. But they all have a commonality: Kubernetes. This ebook explains how Kubernetes is the underlying architecture for platforms suited for enterprise data centers, cloud services and a hybrid approach — as well as at the edge.

The second chapter of the ebook also details the role of cloud native technologies in container platforms. We review how cloud native technologies are essentially container technologies, meant to work on containerized infrastructure, and in particular, Kubernetes.

Microservices use components that run across multiple containers. But what is the means for application management? It's one of the big questions still challenging developers. Applications consist of components, but the challenge is in gluing the components together in a manner that allows not only the building of the application, but also the management of the service throughout its life cycle.

The ebook's author, Janakiram MSV, argues that the Container as a Service (CaaS) model now defines the cloud native stack. But it's complex and difficult to manage. Think of it this way: a Kubernetes architecture in production is like a hive. It's chaos, but with engineering practices that allow for microservices to run in containers, orchestrated by Kubernetes. The ones that do get it have attained at-scale development and management. They are thriving. But really, almost everyone is still in the early days of their journey.

And that really sums up the challenge that the Kubernetes community faces as it moves from its hopeful beginnings, to its ongoing development as a platform that is relevant not just to large enterprise operations, but to mid-sized and small

operations, too.

It's the culture that is still the gap. Developers need a way to easily package applications and there needs to be a clearer understanding of who configures the services. Another challenge is that policy and security matters are still largely dependent on traditional practices.

There is great promise for Kubernetes, as well as parts of the ecosystem still in their infancy. Data on Kubernetes is still largely a space yet to be defined; if Kubernetes is the undisputed control plane now, questions remain about the data plane. And while service mesh capabilities allow for better traffic management, security and observability, the learning curve is still fairly steep.

But it's the concepts associated with observability that have us most excited. Monitoring is a practice that largely suited the on-premises, enterprise architecture. It was made to react. But we can't react anymore. The world requires us to observe what is happening, to see ahead, and to find the answers.

Thanks for downloading this ebook and please feel free to reach out any time.

Alex Williams

Founder and Publisher

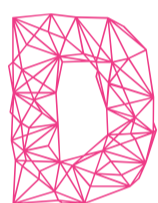
The New Stack

@alexwilliams

alex@thenewstack.io

CHAPTER 01

Kubernetes, The Operating System for the Cloud



During the last ten years, the IT infrastructure market has gone through a tectonic shift. First, it was Infrastructure as a Service (IaaS) that revolutionized the way compute, storage and network resources were provisioned and consumed by businesses. The second half of the last decade witnessed the rise of containers, container management platforms and Containers as a Service (CaaS) offerings in the cloud.

The cloud-based infrastructure services — such as Amazon EC2, Azure Virtual Machines and Google Compute Engine — delivered on-demand infrastructure based on a self-service and programmatic approach to provisioning resources. They enabled businesses to start small and scale fast.

In 2013, Docker, Inc. demonstrated [Docker](#), a lightweight operating system (OS)-level platform based on Linux containers. Docker relied on an inherent capability of Linux OS to run multiple isolated applications within the same operating system.

Traditionally, virtualization provided the ability to run multiple operating systems within the same OS. Hypervisors — software components that manage virtualization of compute resources — handle the partitioning of resources and strong isolation of virtual machines. Linux Containers offer OS-level virtualization

for running multiple isolated Linux processes (containers) on a host using a single Linux kernel. The Linux kernel leverages cgroups for resource control, linux namespaces for isolation, and prioritization of resources such as central processing unit (CPU), memory, block I/O and network — all without the need for launching virtual machines.

Unlike virtual machines, containers don't depend on a hypervisor. They share the underlying operating system services at the kernel level. Linux containers are smaller in size, start at regular process start-up speed, scale rapidly, and, most importantly, they are portable. A container built on Ubuntu can be quickly deployed on Red Hat with absolutely no changes. Administrators can start containerized applications in milliseconds and scale them to hundreds of instances in no time.

Though containers were a part of modern Unix-based operating systems — such as Linux (LXC), FreeBSD (Jails) and Solaris (Zones) — it was Docker, Inc. that made the technology accessible to developers. It revolutionized application development and deployment.

The combination of IaaS and containers in the public cloud promised unmatched scale and agility to organizations. The ability to launch tens — in some cases even hundreds — of containers in each virtual machine enabled maximum utilization of CPU and memory resources of each virtual machine (VM). Containers deployed in the public cloud made it possible to run web-scale workloads at an affordable cost. The potent combination of IaaS and containers became the secret sauce of web-scale startups.

While Docker, Inc. delivered the container runtime and tools to manage the life cycle of a container, the industry realized that it needed a platform to manage hundreds of containers running across hundreds of virtual machines. This led to the release of Docker Swarm by Docker, Inc. and DC/OS from D2iQ (formerly Mesosphere).

Well before containers became popular among developers, Google was running

some of its core web services in Linux containers. In a [presentation](#) at GlueCon 2014, [Joe Beda](#) (one of the creators of Kubernetes) claimed that Google was launching over two billion containers in a week. The secret to Google's ability to manage containers at that scale was its internal data center management tool, [Borg](#).

In June 2014, Google launched an open source software platform to manage containers at scale, called [Kubernetes](#). It was a flavor of Borg that was less opinionated. Google incorporated the best ideas from Borg into Kubernetes, while also addressing the pain points that users had identified with Borg over the years.

In 2015, Kubernetes 1.0 was contributed to [The Linux Foundation](#), which then formed the [Cloud Native Computing Foundation](#) (CNCF) to manage and govern the project. Today, CNCF is a custodian of multiple open source projects related to containers — including Containerd, Envoy, Prometheus and others.

Where did Docker fit into this picture? Docker's purpose was to simplify building modern applications. The developer installed the Docker Engine on her workstation and then used Docker application programming interfaces (APIs) and tools to manage the life cycle of containerized applications. Docker built Compose and Swarm as an extension to the core Docker Engine, making it possible to use the familiar workflow and toolchain to deploy and manage multicontainer workloads across multiple nodes/machines.

But it was Google that took the next step, by making it possible to run different types of containerized workloads at scale on Kubernetes. The extensibility, scale and choice of deployment environments soon made Kubernetes the favorite of developers and operators.

[Apache Mesos](#), an open source project developed at the University of California at Berkeley, was one of the original distributed computing architectures for managing application workloads on compute clusters. But ultimately it faded away, as the industry and open source community put their support behind the Kubernetes ecosystem. Mesos is a mature distributed computing environment, but

it is more suited for large-scale systems with hundreds of nodes. Mesos was originally designed for distributed applications based on Apache Hadoop, Apache Spark and Kafka. Container orchestration came much later in the form of the Marathon plugin. On the other hand, Kubernetes was designed with containerization in mind and an ability to run on small and large clusters with the same ease and flexibility.

Docker, Inc. has also embraced Kubernetes as the preferred container orchestration tool, by embedding it in Docker Desktop and Docker Enterprise. In November 2019, [Mirantis](#), an OpenStack-based infrastructure company, acquired Docker's enterprise business — including its Kubernetes as a Service offerings.

Given its simplicity, accessibility and ability to scale, Kubernetes went on to become the most preferred container management platform. Indeed, it is one of the fastest-growing open source projects in the history of computing. Modern applications and greenfield applications increasingly use Kubernetes, which has led to the rise of enterprise container management platforms like Google Anthos, Red Hat OpenShift and VMware Tanzu. There's also been an increase in managed Container as a Service offerings in the public cloud, such as Amazon Elastic Kubernetes Service, Azure Kubernetes Service and Google Kubernetes Engine.

Deployment Size Matters

While initial container challenges, such as managing storage, have been addressed, Kubernetes users continue to face security and cultural change challenges. They must also adapt to new developments, like the use of service mesh, which often uses sidecar proxies to control microservices within Kubernetes environments. Service mesh aims to improve application traffic control, observability and security for distributed systems. Let's take a look at the [CNCF's annual survey data](#).

The journey from experimentation to significant Kubernetes deployments is

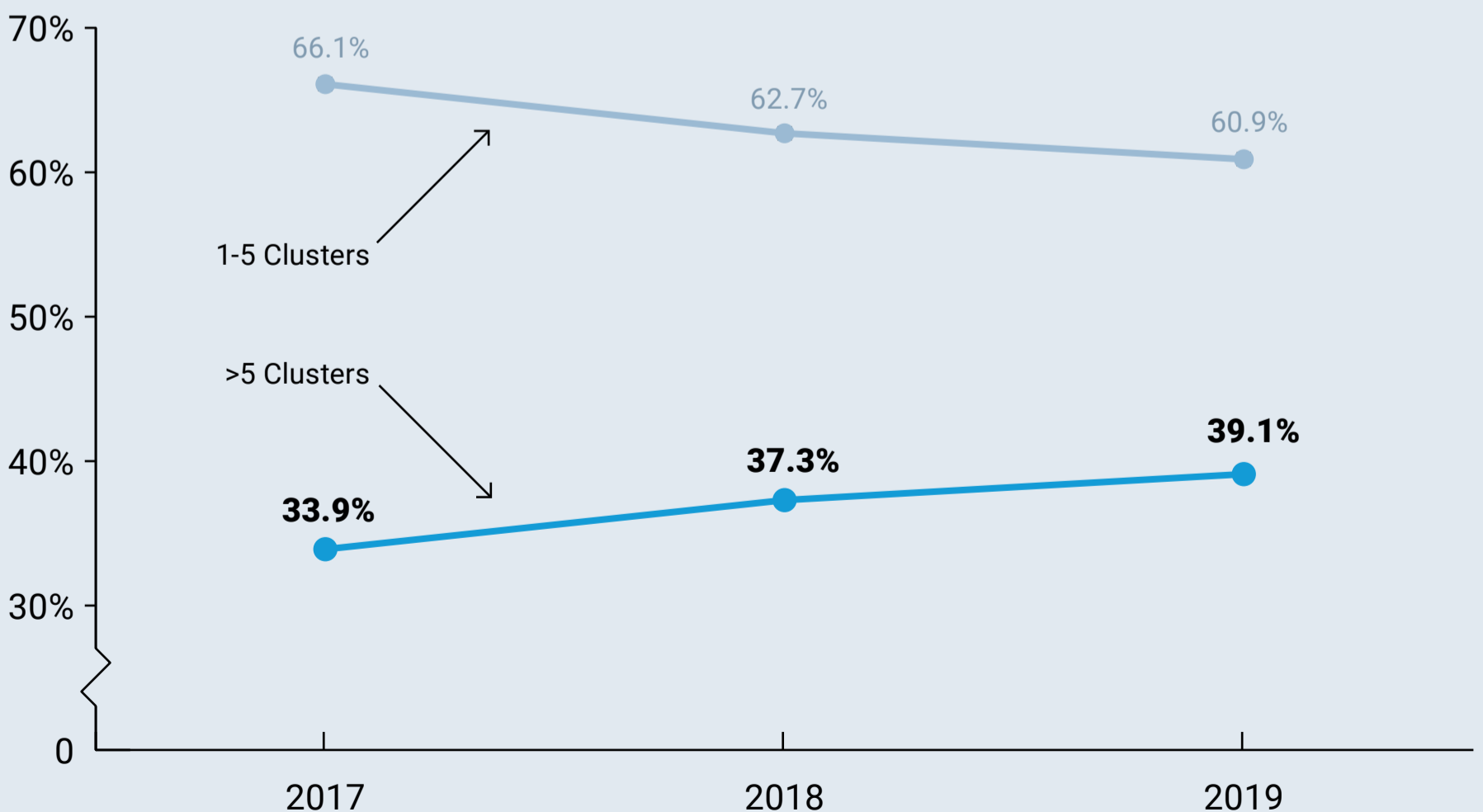
evident in the numbers. In 2017, 64% of the 550-plus respondents to CNCF’s annual survey used the Kubernetes project in production. By 2018, interest in Kubernetes had expanded to a wider group that were evaluating Kubernetes and manifested itself in more than 2,400 survey participants. In 2019, 78% of the 1,300-plus respondents were using Kubernetes in production, further confirming the centrality of Kubernetes in CNCF’s cloud native community.

What came first, the chicken or the egg? Increased container adoption originally created a demand for container orchestrators, but deployed Kubernetes infrastructure makes it easier to run more containers. Kubernetes users are typically running more containers, but their response to container proliferation has been uneven.

Kubernetes deployments with more than five clusters continue to rise, going from 34% in 2017 to 39% in 2019, as shown in figure 1.1 below. In fact, over half of this

FIG 1.1: Larger deployments are more prevalent. The percentage of Kubernetes users with more than five clusters rose from 34% in 2017 to 39% in 2019.

Deployments With >5 Clusters Continue to Rise



Source: The New Stack's analysis of CNCF's 2017, 2018 and 2019 surveys.
 Q. What are your challenges in using / deploying containers?
 Please select all that apply. Data only includes respondents that have at least one Kubernetes cluster.
 2017, n=454; 2018, n=1475; 2019, n=1197.

group is running a thousand or more containers, increasing from 55% in 2017 to 62% in 2019.

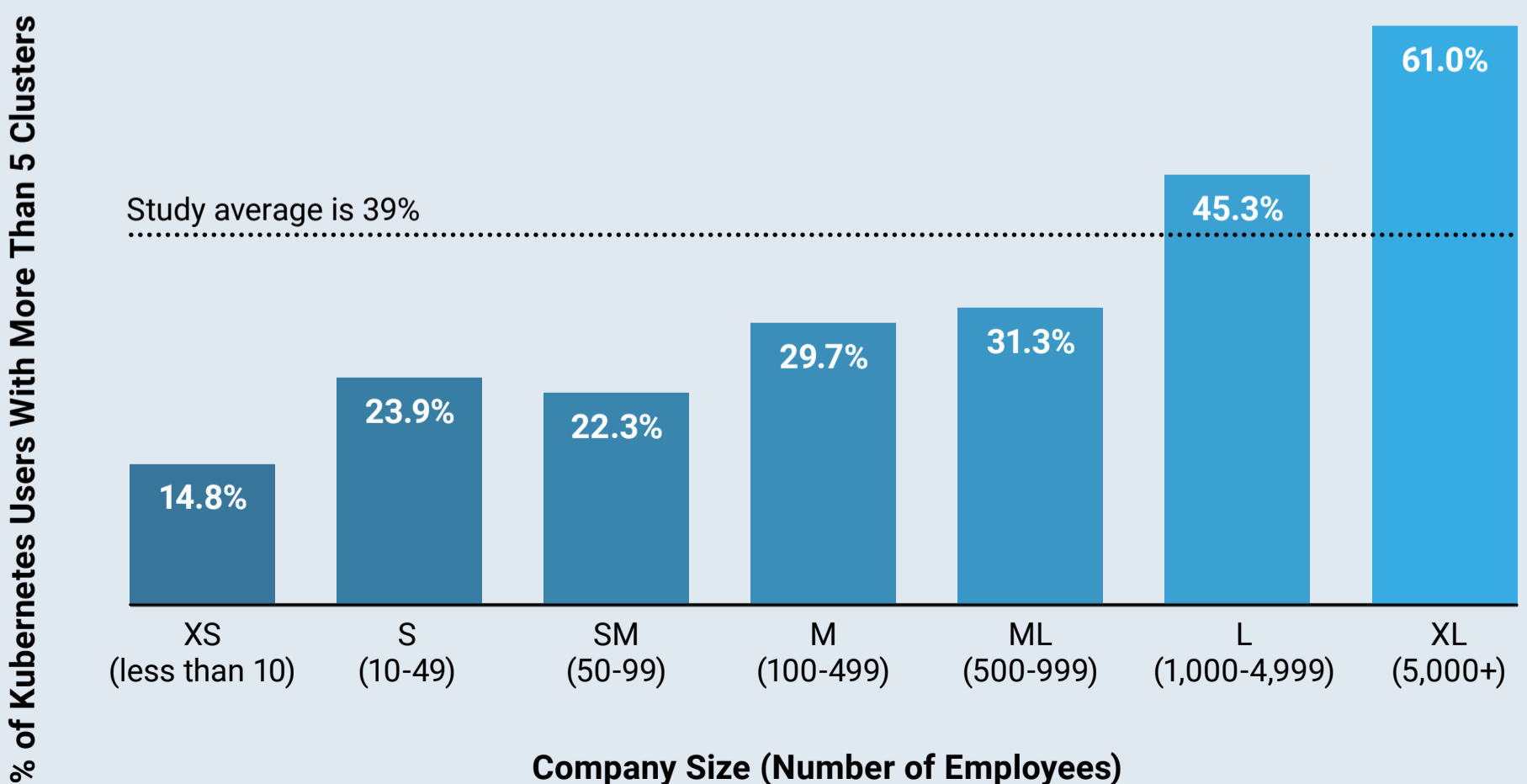
Takeaways:

- Large companies are the biggest adopters and they tend to go with their existing vendors for Kubernetes deployment.
- The scale of container usage has increased, but organizations with five or fewer Kubernetes clusters have more trouble adapting.
- A range of established and emerging storage, ingress and service mesh solutions are being used. The emerging tools are more likely to be evaluated to address challenges, but customers also tend to stick with their existing vendor relationships.

Figure 1.2 below shows that employee count is associated with large Kubernetes

FIG 1.2: Sixty-one percent of Kubernetes users at organizations with 5,000 or more employees have more than five clusters. The study average is 39%.

Big Companies Continue to Have Bigger Kubernetes Deployments



deployments, more so than the number of containers. Sixty-one percent of Kubernetes users at organizations with 5,000 or more employees have more than five clusters.

Casting the net a bit wider, organizations with at least a thousand employees are more likely to have more than five clusters, going from 51% in 2017 to 56% in 2019.

These figures are relevant because companies with large Kubernetes deployments have seen a much greater reduction in their container challenges.

Organizations with five or fewer clusters are facing challenges as they try to manage a larger number of containers.

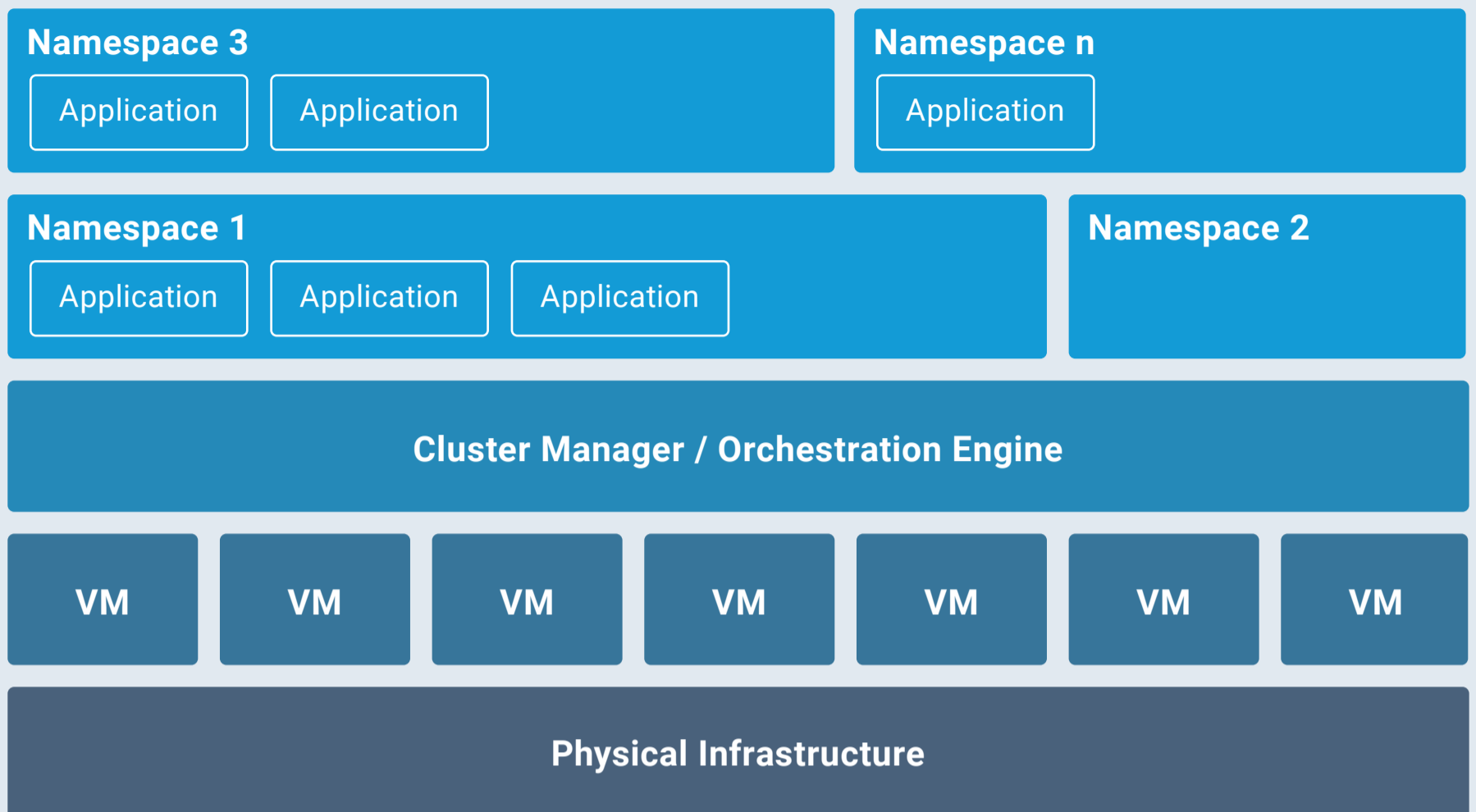
What Does Orchestration Mean?

Two trends that have influenced modern infrastructure are containers and DevOps. The DevOps ecosystem evolved to deliver continuous integration, continuous testing, continuous deployment and continuous monitoring, which increased the velocity of software development. The adoption of containers, combined with proven DevOps best practices, resulted in rapid deployments at scale.

While containers help increase developer productivity, orchestration tools offer many benefits to organizations seeking to optimize their DevOps and operations investments. Some of the benefits of container orchestration include:

- Efficient resource management.
- Seamless scaling of services.
- High availability.
- Low operational overhead at scale.
- A declarative model for most orchestration tools, reducing friction for more autonomous management.

Cluster Management and Orchestration Platform



Source: Janakiram MSV

© 2020 THE NEW STACK

FIG 1.3: *The resource layers of a system, from the perspective of the container orchestration engine.*

- Operations-style Infrastructure as a Service (IaaS), but manageable like Platform as a Service (PaaS).
- Consistent operating experience across on-premises and public cloud providers.

IaaS is chosen by operators for control and automation. Developers prefer PaaS for its flexibility, scale and productivity. Container orchestration tools bring the best of both worlds: automation and scale.

Containers solved the developer productivity problem, making the DevOps workflow seamless. Developers could create a container image, run a container, and develop code in that container to deploy it in local data centers or public cloud environments. Yet this introduction of seamlessness to developer productivity does not translate automatically into efficiencies in production environments.

The production environment is often quite different from the local environment of

a developer's laptop. Whether you're running traditional three tier applications at scale or microservices-based applications, managing a large number of containers and the cluster of nodes supporting them is no easy task. Orchestration is the component required to achieve scale, because scale requires automation.

The distributed nature of cloud computing brought with it a paradigm shift in how we perceive virtual machine infrastructure. The notion of “cattle vs. pets” — treating a container more like a unit of livestock than a favorite animal — helped reshape people's mindsets about the nature of containers and infrastructure.

Both containers and the infrastructure they run on are immutable — a paradigm in which containers or servers are never modified after they're deployed. If something needs to be updated, fixed or modified in any way, new containers or servers built from a common image with the appropriate changes are provisioned to replace the old ones. This approach is comparable to managing cattle at a dairy farm.

On the other hand, traditional servers and even virtual machines are not treated as immutable — they are more like pets and therefore not disposable. So their maintenance cost is high, because they constantly need the attention of the operations team.

Immutable infrastructure is programmable, which allows for automation.

Infrastructure as Code (IaC) is one of the key attributes of modern infrastructure, in which an application can programmatically provision, configure and utilize the infrastructure to run itself.

The combination of container orchestration, immutable infrastructure, and automation based on IaC delivers flexibility and scale.

Putting this notion into practice, containers at scale extended and refined the concepts of scaling and resource availability.

The baseline features of a typical container orchestration platform include:

- Scheduling.
- Resource management.
- Service discovery.
- Health checks.
- Autoscaling.
- Updates and upgrades.

The container orchestration market is currently dominated by Kubernetes. It has gained the acceptance of enterprises, platform vendors, cloud providers and infrastructure companies.

Container orchestration encourages the use of the microservices architecture pattern, in which an application is composed of smaller, atomic, independent services — each one designed for a single task. Each microservice is packaged as a container, and multiple microservices logically belonging to the same application are orchestrated by Kubernetes at runtime.

The rise of Kubernetes has resulted in the creation of new market segments based on the container orchestration and management platform. From storage to the network to monitoring to security, there is a new breed of companies and startups building container-native products and services. The next chapter highlights some of the building blocks of cloud native platforms and startups from this emerging ecosystem.

Kubernetes Architecture

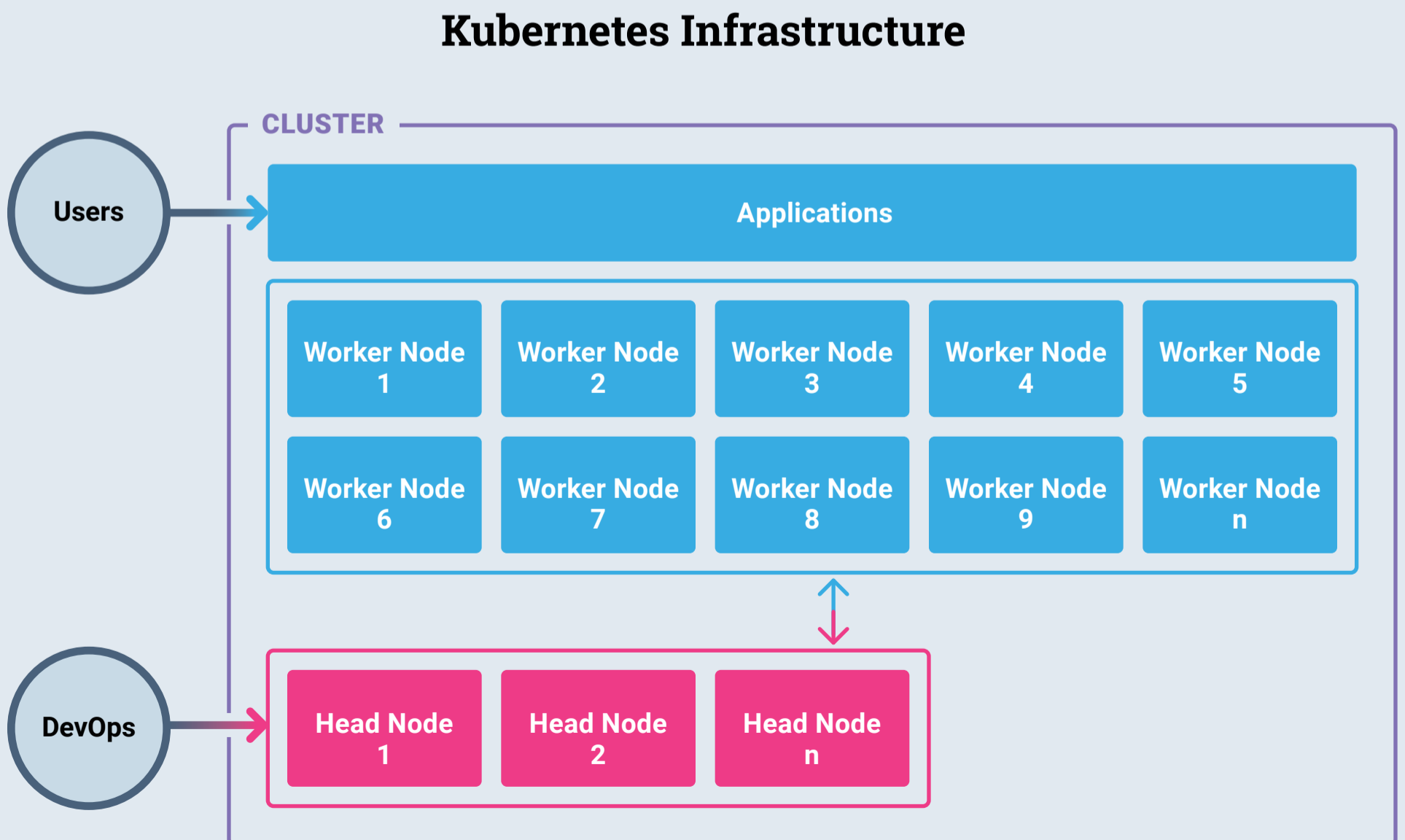
A contemporary application, packaged as a set of containers and deployed as microservices, needs an infrastructure robust enough to deal with the demands of clustering and the stress of dynamic orchestration. Such an infrastructure should provide primitives for scheduling, monitoring, upgrading and relocating containers across hosts. It must treat the underlying compute, storage and

network primitives as a pool of resources. Each containerized workload should be capable of taking advantage of the resources exposed to it, including CPU cores, storage units and networks.

Kubernetes (see figure 1.4) is an open source distributed system that abstracts the underlying physical infrastructure, making it easier to run containerized applications at scale. An application, managed through the entirety of its life cycle by Kubernetes, is composed of containers gathered together as a set and coordinated into a single unit. An efficient cluster manager layer lets Kubernetes effectively decouple this application from its supporting infrastructure, as depicted in the figure below. Once the Kubernetes infrastructure is fully configured, DevOps teams can focus on managing the deployed workloads instead of dealing with the underlying resource pool — CPU and memory — which is handled by Kubernetes.

Kubernetes is an example of a well-architected distributed system. It treats all the

FIG 1.4: *The big picture of a Kubernetes cluster.*



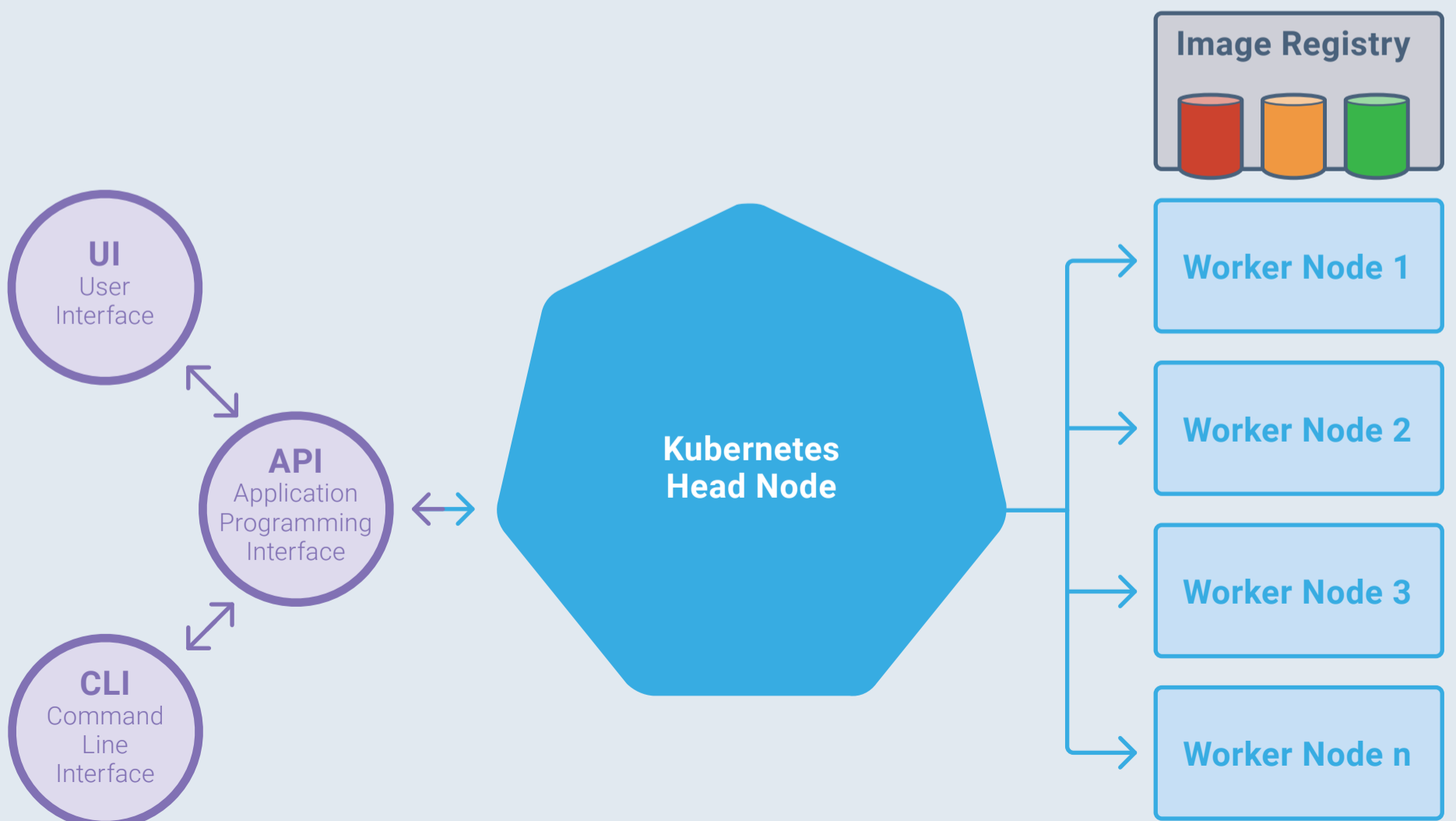
machines in a cluster as a single pool of resources. It takes up the role of a distributed operating system by effectively managing the scheduling, allocating the resources, monitoring the health of the infrastructure, and even maintaining the desired state of infrastructure and workloads. Kubernetes is an operating system capable of running modern applications across multiple clusters and infrastructures on cloud services and private data center environments.

Like any other mature distributed system, Kubernetes has two layers consisting of the head nodes and worker nodes (see figure 1.5). The head nodes typically run the control plane responsible for scheduling and managing the life cycle of workloads. The worker nodes act as the workhorses that run applications. The collection of head nodes and worker nodes becomes a cluster.

The DevOps teams managing the cluster talk to the control plane’s API via the command-line interface (CLI) or third-party tools. The users access the

FIG 1.5: *The role of head node in Kubernetes architecture.*

Kubernetes Architecture



applications running on the worker nodes. The applications are composed of one or more container images that are stored in an accessible image registry.

Control Plane

The control plane runs the Kubernetes components that provide the core functionalities: exposing the Kubernetes API, scheduling the deployments of workloads, managing the cluster, and directing communications across the entire system. As depicted in figure 1.5, the head node monitors the containers running in each node as well as the health of all the registered nodes. Container images, which act as the deployable artifacts, must be available to the Kubernetes cluster through a private or public image registry. The nodes that are responsible for scheduling and running the applications access the images from the registry via the container runtime.

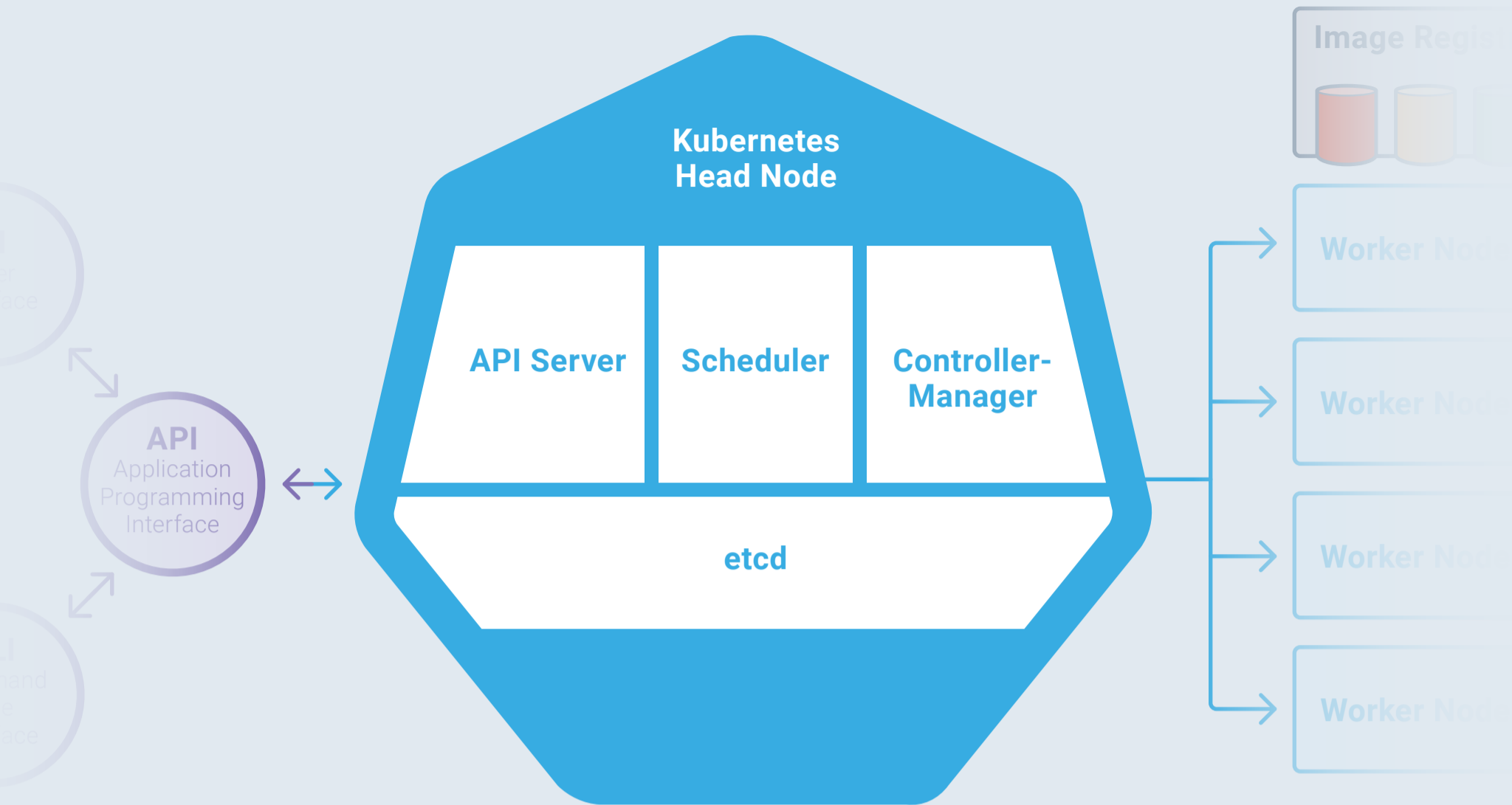
As figure 1.6 shows, the Kubernetes head node runs the following components that form the control plane:

etcd

Developed by CoreOS, which was later acquired by Red Hat, etcd is a persistent, lightweight, distributed, key-value data store that maintains the cluster's configuration data. It represents the overall state of the cluster at any given point of time, acting as the single source of truth. Various other components and services watch for changes to the etcd store to maintain the desired state of an application. That state is defined by a declarative policy; in effect, a document that states the optimum environment for that application, so the orchestrator can work to attain that environment. This policy defines how the orchestrator addresses the various properties of an application, such as the number of instances, storage requirements and resource allocation.

The etcd database is accessible only through the API server. Any component of the cluster which needs to read or write to etcd does it through the API server.

Kubernetes Head Node



Source: Janakiram MSV

© 2020 THE NEW STACK

FIG 1.6: Components of the head node.

API Server

The API server exposes the Kubernetes API by means of JSON over HTTP, providing the representational state transfer (REST) interface for the orchestrator's internal and external endpoints. The CLI, the web user interface (UI), or another tool may issue a request to the API server. The server processes and validates the request, and then updates the state of the API objects in etcd. This enables clients to configure workloads and containers across worker nodes.

Scheduler

The scheduler selects the node on which each workload should run based on its assessment of resource availability, and then tracks resource utilization to ensure the pod isn't exceeding its allocation. It maintains and tracks resource requirements, resource availability, and a variety of other user-provided constraints and policy directives; for example, quality of service (QoS), affinity/

anti-affinity requirements and data locality. An operations team may define the resource model declaratively. The scheduler interprets these declarations as instructions for provisioning and allocating the right set of resources to each workload.

Controller-Manager

The part of Kubernetes' architecture which gives it its versatility is the controller-manager, which is a part of the head node. The controller-manager's responsibility is to ensure that the cluster maintains the desired application state all the time through a well-defined controller. A controller is a control loop that watches the shared state of the cluster through the API server and makes changes attempting to move the current state towards the desired state.

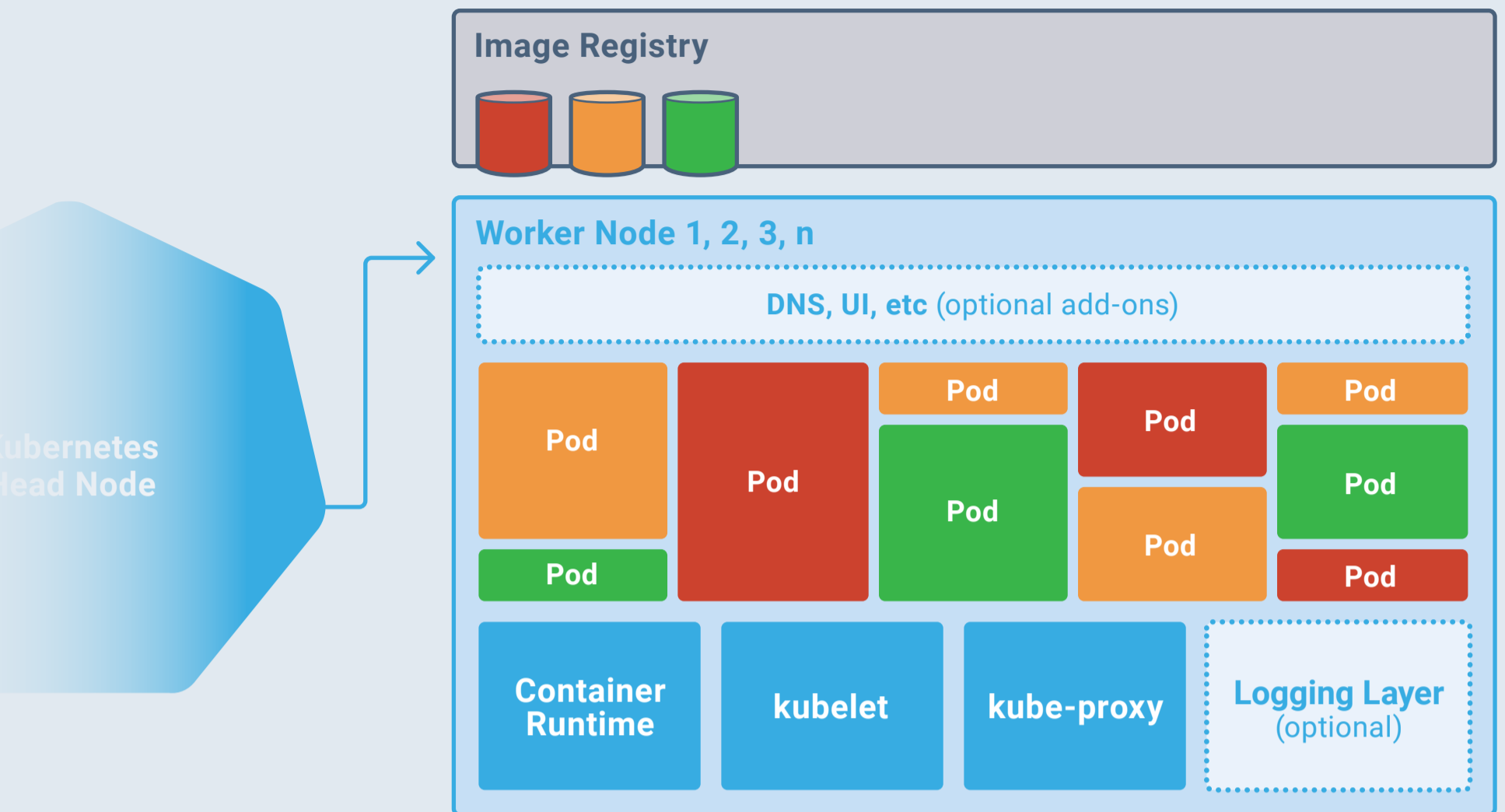
The controller maintains the stable state of nodes and pods by constantly monitoring the health of the cluster and the workloads deployed on that cluster. For example, when a node becomes unhealthy, the pods running on that node may become inaccessible. In such a case, it's the job of the controller to schedule the same number of new pods in a different node. This activity ensures that the cluster is maintaining the expected state at any given point of time.

Kubernetes comes with a set of built-in controllers that run inside the controller-manager. These controllers offer primitives that are aligned with a certain class of workloads, such as stateless, stateful, scheduled cron jobs and run-to-completion jobs. Developers and operators can take advantage of these primitives while packaging and deploying applications in Kubernetes.

Worker Nodes

The node is the workhorse of the Kubernetes cluster, responsible for running containerized workloads; additional components of logging, monitoring and service discovery; and optional add-ons. Its purpose is to expose compute, networking and storage resources to applications. Each node includes a container runtime, such as Docker, plus an agent (kubelet) that communicates with the head node. A node may

Kubernetes Node



Source: Janakiram MSV

© 2020 THE NEW STACK

FIG 1.7: Components of the worker nodes.

be a virtual machine (VM) running in a cloud or a bare-metal server inside the data center.

Each node, as shown in figure 1.7, contains the following:

Container Runtime

The container runtime is responsible for managing the life cycle of each container running in the node. After a pod is scheduled on the node, the runtime pulls the images specified by the pod from the registry. When a pod is terminated, the runtime kills the containers that belong to the pod. Kubernetes may communicate with any Open Container Initiative (OCI)-compliant container runtime, including Docker and CRI-O.

The OCI is a standard that defines the runtime specification and image specification, with a goal to drive standardization of container runtimes and image formats.

Kubelet

The kubelet is the Kubernetes agent whose responsibility is to interact with the container runtime to perform operations such as starting, stopping and maintaining containers.

Each kubelet also monitors the state of the pods. When a pod does not meet the desired state as defined by the deployment, it may be restarted on the same node. The node's status is transmitted to the head node every few seconds via heartbeat messages. If the head node detects a node failure, the replication controller observes this state change and schedules the pods on other healthy nodes.

Kube-Proxy

The `kube-proxy` component is implemented as a network proxy and a load balancer that orchestrates the network to route requests to the appropriate pods. It routes traffic to the appropriate pod based on the associated service name and the port number of an incoming request. It also takes advantage of OS-specific networking capabilities by manipulating the policies and rules defined through `iptables`. Each `kube-proxy` component may be integrated with network layers such as [Calico](#) and [Flannel](#).

Logging Layer

The orchestrator makes frequent use of logging as a means for gathering resource usage and performance metrics for containers on each node, such as CPU, memory, file, and network usage. The Cloud Native Computing Foundation hosts a software component that provides a unified logging layer for use with Kubernetes or other orchestrators, called [Fluentd](#). This component generates metrics that the Kubernetes head controller needs in order to keep track of available cluster resources, as well as the health of the overall infrastructure.

Add-Ons

Kubernetes supports additional services in the form of add-ons. These optional services, such as the dashboard, are deployed like other applications, but are

integrated with other core components on the node, such as the logging layer and `kube-proxy`. For example, the dashboard add-on pulls the metrics from the kubelet to display rich visualizations of resource utilization. The DNS add-on, based on `kube-dns` or CoreDNS, augments `kube-proxy` through name resolution.

Workloads

While the control plane and the worker nodes form the core cluster infrastructure, the workloads are the containerized applications deployed in Kubernetes.

After developing and testing a microservice, the developers package it as a container, which is the smallest unit of deployment packaged as a pod. A set of containers belonging to the same application is grouped, packaged, deployed and managed within Kubernetes.

Kubernetes exposes primitives for deployment, while constantly scaling, discovering, and monitoring the health of these microservices. Namespaces are typically used to logically separate one application from the other. They act as a logical cluster by providing a well-defined boundary and scope for all resources and services belonging to an application.

Within a namespace, the following Kubernetes primitives are deployed:

Pods

A pod is the basic execution unit of a Kubernetes application. It is the smallest and simplest unit in the Kubernetes object model. A pod is also the smallest schedulable item in a Kubernetes application. If Kubernetes is an operating system, a pod represents a set of processes — where each process may be mapped to a container — running on the cluster.

The pod serves as the core unit of workload management for Kubernetes, acting as the logical boundary for containers sharing the same execution context and resources. Grouping related containers into pods makes up for the configurational challenges introduced when containerization replaced first-generation

virtualization, by making it possible to run multiple dependent processes together.

Each pod is a collection of one or more containers that use interprocess communication (IPC) for communication, and that may share the storage and networking stack. In scenarios where containers need to be coupled and co-located — for instance, a web server container and a cache container — they may easily be packaged in a single pod. A pod may be scaled out either manually, or through a policy defined by a feature called Horizontal Pod Autoscaling (HPA). Through this method, the number of pods that are a part of the deployment is increased proportionally based on available resources.

Pods enable a functional separation between development and deployment. While developers focus on their code, operators can concentrate on the broader picture of deciding which related containers may be stitched together into a functional unit. The result is the optimal amount of portability, since a pod is just a manifest of one or more container images managed together.

Controllers

In Kubernetes, controllers augment pods by adding additional capabilities, such as desired configuration state and runtime characteristics.

A deployment brings declarative updates to pods. It guarantees that the desired state is always maintained by tracking the health of the pods participating in the deployment. Each deployment manages a ReplicaSet, which maintains a stable set of replica pods running at any given time, as defined by the desired state.

Deployments bring PaaS-like capabilities to pods through scaling, deployment history and rollback features. When a deployment is configured with a minimum replica count of two, Kubernetes ensures that at least two pods are always running, which brings fault tolerance. Even when deploying the pod with just one replica, it is highly recommended to use a deployment controller instead of a plain vanilla pod specification.

A statefulset is similar to a deployment, but is meant for pods that need persistence and a well-defined identifier and guaranteed order of creation. For workloads such as database clusters, a statefulset controller will create a highly available set of pods in a given order that have a predictable naming convention. Stateful workloads that need to be highly available, such as Cassandra, Kafka, SQL Server and ZooKeeper, are deployed as statefulsets in Kubernetes.

To force a pod to run on every node of the cluster, a DaemonSet controller can be used. Since Kubernetes automatically schedules a DaemonSet in newly provisioned worker nodes, it becomes an ideal candidate to configure and prepare the nodes for the workload. For example, if an existing network file system (NFS) or Gluster file share has to be mounted on the nodes before deploying the workload, it is recommended to package and deploy the pod as a DaemonSet. Monitoring agents are good candidates to be used as a DaemonSet, to ensure that each node runs the monitoring agent.

For batch processing and scheduling jobs, pods can be packaged for a run-to-completion job or a cron job. A job creates one or more pods and ensures that a specified number of them successfully terminate. Pods configured for run to completion execute the job and exit, while a cron job will run a job based on the schedule defined in the crontab format.

Controllers define the life cycle of pods based on the workload characteristics and their execution context.

Services and Service Discovery

The services model in Kubernetes provides the most basic, but most important, aspect of microservices: discovery.

Any API object in Kubernetes, including a node or a pod, may have key-value pairs associated with it — additional metadata for identifying and grouping objects sharing a common attribute or property. Kubernetes refers to these key-value pairs

as labels and annotations. Service discovery takes advantage of the labels and selectors to associate a service with a set of pods.

A single pod or ReplicaSet may be exposed to internal or external clients via services, which associate a set of pods with a specific criterion. Any pod whose labels match the selector defined in the service manifest will automatically be discovered by the service. This architecture provides a flexible, loosely-coupled mechanism for service discovery.

When a pod is created, it is assigned an IP address accessible only within the cluster. But there is no guarantee that the pod's IP address will remain the same throughout its life cycle. Kubernetes may relocate or reinstantiate pods at runtime, resulting in a new IP address for the pod.

To compensate for this uncertainty, services ensure that traffic is always routed to the appropriate pod within the cluster, regardless of the node on which it is scheduled. Each service exposes an IP address, and may also expose a DNS endpoint — both of which will never change. Internal or external consumers that need to communicate with a set of pods will use the service's IP address, or its more generally known DNS endpoint. In this way, the service acts as the glue for connecting pods with other pods.

A deployment relies upon labels and selectors for determining which pods will participate in a scaling operation. Any pod whose label matches the selector defined by the service will be exposed at its endpoint. A service then provides basic load balancing by routing traffic across matching pods.

A selector is a kind of criterion used to query Kubernetes objects that match a label value. This powerful technique enables loose coupling of objects. New objects may be generated whose labels match the selectors' value. Labels and selectors form the primary grouping mechanism in Kubernetes for identifying components to which an operation applies.

At runtime, pods may be scaled by means of ReplicaSets, ensuring that every deployment always runs the desired number of pods. Each ReplicaSet maintains a predefined set of pods at all times. When a scaling operation is initiated by a deployment, new pods created by that operation will instantly begin receiving traffic.

Kubernetes provides three schemes to expose services:

1. **ClusterIP:** Meant for pods to communicate with each other within the cluster. For example, a database pod exposed through a ClusterIP-based service becomes available to the web server pods.
2. **NodePort:** Used to expose a service on the same port across all the nodes of a cluster. An internal routing mechanism ensures that the request is forwarded to the appropriate node on each pod. This is typically used for services with external consumers.
3. **LoadBalancer:** The type LoadBalancer extends the NodePort service by adding Layer 4 (L4) and Layer 7 (L7) load balancers. This scheme is often used with clusters running in public cloud environments that support automated provisioning of software-defined load balancers.

When multiple services need to share the same load balancer or an external endpoint, an ingress controller is recommended. It manages external access to the services in a cluster — typically HTTP — by providing load balancing, secure socket layer (SSL) termination and name-based virtual hosting.

Ingress is becoming increasingly popular for running production workloads in Kubernetes. It lets multiple microservices of the same application use the same endpoint, which is exposed by a load balancer, API gateway or an application delivery controller (ADC).

Network & Storage

Compute, network and storage are the foundations of any infrastructure service. In

Kubernetes, nodes represent the compute building block, which provides those foundational resources to pods running in the clusters. The network and storage services are delivered by software-defined, container-native plugins designed for Kubernetes.

The network component enables pod-to-pod, node-to-pod, pod-to-service, and external clients-to-service communication. Kubernetes follows a plugin model for implementing networking. Kubenet is the default network plugin and it is simple to configure. It is typically used together with a cloud provider that sets up routing rules for communication between nodes, or in single-node environments.

Kubernetes can support a host of plugins based on the [Container Network Interface](#) (CNI) specification, which defines the network connectivity of containers and deals with the network resources when the container is deleted. There are many implementations of CNI, including [Calico](#), [Cilium](#), [Contiv](#), [Weave Net](#) and more. Virtual networking available in public clouds is also supported by the CNI specification, which makes it possible to extend the network topology and subnets to Kubernetes clusters.

Some of the CNI-compliant network plugins, such as Calico, implement policies that enforce strict routing policies by isolating pods. They bring firewall-like rules to the pods and namespaces of a Kubernetes cluster.

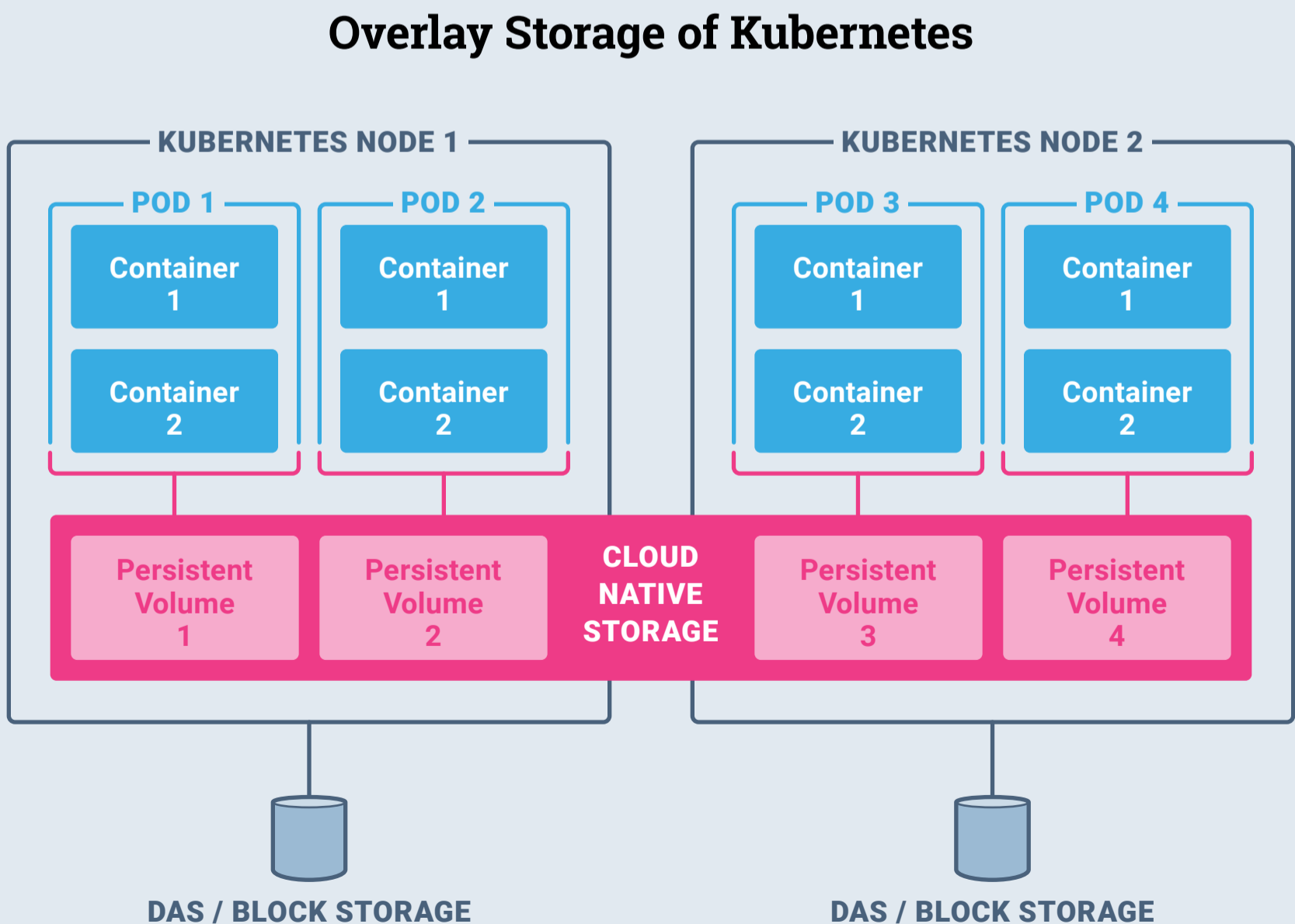
Persistent storage is exposed to Kubernetes via persistent volumes. Pods consume the volumes through persistent volume claims. Storage administrators provision storage by creating the persistent volumes from existing network attached storage (NAS), storage area network (SAN), direct attached storage (DAS), solid state drives (SSDs), non-volatile memory express (NVMe) or flash disk arrays. Developers and DevOps teams get a chunk of persistent volumes through the persistent volume claims associated with pods.

Kubernetes comes with storage primitives to expose storage from existing nodes. One such primitive is a volume type that makes the underlying storage accessible

to pods. Examples of volume types include `emptyDir` and `hostPath`. They are used for specific use cases: `emptyDir` is for scratch space and `hostPath` makes local volumes available to pods. But they don't have high availability and fault tolerance due to the tight coupling with the node. Overlay storage (see figure 1.8) layers pool storage volumes from block devices, NAS and SAN to expose external storage to Kubernetes objects.

To offer high availability and container-native storage capabilities, Kubernetes introduced plugins for storage vendors to expose their platforms to containerized workloads. Block storage from public cloud providers, distributed file systems based on NFS and GlusterFS, and a few commercial storage platforms have plugins included in the open source upstream distribution of Kubernetes. Storage administrators create storage classes for each type of storage engine based on their performance and speed. Persistent volumes and claims can be created from these storage classes for different types of workloads. For example, a relational database

FIG 1.8: *Exposing Storage to Pods and Containers*



management system (RDBMS) may be associated with a storage class with higher input/output operations per second (IOPS), while a content management system (CMS) may target a distributed storage engine through a different storage class.

Similar to CNI, the Kubernetes community has defined specifications for storage through [Container Storage Interface](#) (CSI), which encourages a standard, portable approach to implementing and consuming storage services by containerized workloads.

How Kubernetes Delivers the Promise of Web-Scale Computing

With its lineage in Borg, Kubernetes is designed for hyper-scale workloads. Its modern architecture ensures optimal utilization of infrastructure resources.

Additional worker nodes can be easily added to an existing cluster with almost no change to the configuration. Workloads will be able to immediately take advantage of the CPU, memory and storage resources of new nodes.

The idea of grouping a related set of containers together as a pod and treating it as a unit of deployment and scale results in better performance. For example, co-locating a web server and cache containers in the same pod reduces latency and improves performance. Containers within the pod share the same execution context, enabling them to use the interprocess communication, which reduces the overhead.

Pods that belong to the same ReplicaSet and deployment scale rapidly. It just takes a few seconds to scale a deployment to hundreds of pods. The pods are scheduled on the nodes based on the resource ability and the desired state of configuration. By configuring a Horizontal Pod Autoscaler (HPA), Kubernetes can automatically scale in and scale out a deployment.

When running in elastic infrastructure environments, Kubernetes can use Cluster Autoscaler to add and remove nodes to the cluster. Combined with HPA, this

technique can efficiently manage dynamic autoscaling of both the workload as well as the infrastructure.

The lightweight networking stack and service discovery of Kubernetes are designed for scale. They can handle tens of thousands of endpoints exposed by services for internal and external consumption.

The Kubernetes ecosystem and community continue to innovate to make the platform suitable for hyperscale workloads.

Adopting Kubernetes

Kubernetes is deployed in production environments as a container orchestration engine, as a PaaS, and as core infrastructure for managing cloud native applications. These use cases are not mutually exclusive. It is possible for operators to delegate complete application lifecycle management (ALM) to a PaaS layer based on Kubernetes. They may also use a standalone Kubernetes deployment to manage applications deployed using the existing CI/CD toolchain. Customers building greenfield applications can leverage Kubernetes for managing the new breed of microservices-based cloud native applications through advanced scenarios such as rolling upgrades and canary deployments.

Customers evaluating Kubernetes or using it for production deployments can choose from a variety of distributions and flavors.

Below is a list of mainstream Kubernetes distributions available in the market.

Upstream Distribution

The upstream, open source distribution of [Kubernetes available in GitHub](#) can be deployed in data centers, public clouds and private cloud environments. The codebase includes the core building blocks and the essential elements needed to run workloads on Kubernetes.

Kubernetes comes with a deployment tool called kubectl, which provides a simple

installation experience for configuring clusters based on [CentOS](#), [Red Hat Enterprise Linux](#), [SUSE](#), [Ubuntu](#) and other Linux distributions.

Customers may also use automated deployment tools such as [kops](#), [kubespray](#) and [Rancher Kubernetes Engine](#) to install and configure clusters. These tools offer basic to advanced levels of customization for configuring container runtimes, network and storage plugins.

The upstream distribution is free, transparent and a fully-configurable approach of installing Kubernetes. However, it demands a high level of expertise and skills from individuals and organizations planning to use Kubernetes.

Commercial Distributions

Some of the vendors offer a customized and optimized flavor of Kubernetes, bundled with professional services and support. They follow a proven model of commercializing open source software through services.

Commercial distributions not only accelerate the configuration and deployment of Kubernetes clusters, they also promise patches, hot fixes, and seamless upgrades to newer versions of the platform.

[Canonical](#), [D2iQ](#), [HPE](#), [Mirantis](#), [Rancher](#) and [VMware](#) are some of the vendors offering commercial Kubernetes distributions.

Containers as a Service

Kubernetes is also available in the form of a fully hosted, managed platform. Whether it is an enterprise data center, co-located infrastructure facility, or a public cloud environment, the vendors offering Containers as a Service (CaaS) promise an end-to-end Kubernetes environment backed by a commercial service-level agreement (SLA).

Almost all the major cloud providers now have a CaaS offering. [Alibaba Container Service for Kubernetes](#), [Amazon EKS](#), [Azure AKS](#), [DigitalOcean Kubernetes](#), [Google](#)

[Kubernetes Engine](#), [Huawei Cloud Container Engine](#) and [IBM Kubernetes Service](#) are examples of CaaS in the public cloud.

[Mirantis](#), [NetApp](#) and [Platform 9](#) offer CaaS for data center and private cloud environments.

Platform as a Service

Customers deploy PaaS primarily to standardize their development and deployment environments. By using a Kubernetes-based PaaS, they will be able to develop traditional line-of-business applications and greenfield applications. Kubernetes has been adopted by traditional PaaS vendors to deliver end-to-end platform capabilities to enterprise customers.

Built on top of core container orchestration capabilities, Kubernetes-based PaaS offerings deliver complete lifecycle management for containerized applications.

[Red Hat OpenShift](#) and [VMware Tanzu Kubernetes Grid](#) are examples of PaaS offerings based on Kubernetes.

Kubernetes as a Universal Control Plane

Kubernetes is emerging as one of the best control planes in the context of modern applications and infrastructure. The powerful scheduler, which was originally designed to deal with the placement of pods on appropriate nodes, is quite extensible. It can solve many of the problems that exist in traditional distributed systems.

Kubernetes is fast becoming the preferred control plane for scheduling and managing jobs in highly-distributed environments. These jobs may include deploying virtual machines on physical hosts, placing containers in edge devices, or even extending the control plane to other schedulers such as serverless environments.

From bare-metal servers to virtual machines to the internet of things (IoT) devices

to managed cloud services, Kubernetes has gone beyond containers and pods to tackle multiple provisioning and scheduling challenges.

Below are a few examples of this pattern:

Crossplane

[Crossplane](#) aims to standardize infrastructure and application management using the same API-centric, declarative configuration and automation approach pioneered by Kubernetes. It is a unified control plane that integrates seamlessly with existing tools and systems, and makes it easy to set policies, quotas and track reports.

Crossplane acts as a bridge between Kubernetes and traditional workloads such as databases, and even managed services in the public cloud. DevOps can declare external resources using the same YAML specification, along with the native Kubernetes applications. This approach encourages configuration as code by extending the versioning, continuous integration and deployment to non-Kubernetes resources.

K3s

[K3s](#) from Rancher is a certified Kubernetes distribution designed for production workloads running in highly-constrained environments such as IoT and edge computing deployments.

K3s can be deployed on the most virtual machine in the public cloud, or even on a Raspberry Pi device. Its architecture, while maintaining full compatibility and compliance with CNCF Kubernetes conformance tests, is highly optimized for unattended, remote deployments on resource-constrained devices.

K3s is bringing Kubernetes to the edge computing layer by making it accessible and lightweight.

KubeEdge

At KubeCon+CloudNativeCon 2018 in Seattle, Huawei presented [KubeEdge](#), the official project to bring the power of Kubernetes to the edge.

KubeEdge is based on Huawei's [Intelligent Edge Fabric](#) (IEF) — a commercial IoT edge platform based on Huawei IoT PaaS. A large part of IEF has been modified and open sourced for KubeEdge. Available in version 1.3, KubeEdge is stable and addresses the key use cases related to IoT and edge. It can be installed on a supported Linux distribution and on an ARM device like a Raspberry Pi.

The KubeEdge project is part of the CNCF Sandbox.

KubeVirt

[KubeVirt](#), a virtual machine management add-on for Kubernetes, is aimed at allowing users to run VMs right alongside containers in their Kubernetes or OpenShift clusters. It extends Kubernetes by adding resource types for VMs and sets of VMs through Kubernetes' CustomResourceDefinitions (CRD) API. KubeVirt VMs run within regular Kubernetes pods, where they have access to standard pod networking and storage, and can be managed using standard Kubernetes tools such as `kubectl`.

KubeVirt is also a part of the CNCF Sandbox.

Virtual Kubelet

Microsoft's [Virtual Kubelet](#) project is the most interesting extension of the Kubelet agent and Kubernetes API. The Virtual Kubelet is an agent that runs in an external environment which is registered as a node within the Kubernetes cluster. The agent creates a node resource through the Kubernetes API. By leveraging the concepts of taints and tolerations, it schedules pods in an external environment by calling its native API.

Virtual Kubelet works with [Azure Container Instances](#), [Azure IoT Edge](#), and [AWS](#)

[Fargate](#) control plane.

Though Kubernetes had a humble beginning in container orchestration, it quickly evolved to become the operating system of the cloud and the edge. Kubernetes is the foundation of modern infrastructure across the data center, hybrid cloud, public cloud and multicloud environments.

The next chapter explores the growing ecosystem of Kubernetes and the cloud native industry.

Self-Service Architectures and the Kubernetes Operator for Cassandra



Over the past decade we've seen the rise of distributed databases, like the open source Apache Cassandra, to complement Kubernetes and its scaled-out architecture. This trend has ushered in the era of fully self-service architectures, which we discuss in this podcast featuring

Kathryn Erickson and Patrick McFadin from the cloud native, NoSQL data platform company DataStax.

DataStax provides a guide for Cassandra through a new Kubernetes operator, which abstracts the database layers so developers can focus on doing what they do best: code.

Self-service architectures free up developers to use data in fresh and unique ways. Or as McFadin puts it, "Databases [have] become less of a cognitive burden for developers."

[Listen on SoundCloud](#)

[Watch on YouTube](#)



Kathryn Erickson leads self service business strategy for DataStax.



Patrick McFadin is chief evangelist for Apache Cassandra and vice president of developer relations at DataStax.

AI Observability Cuts Through Kubernetes Complexity



Kubernetes has made scaled-out applications on multiple cloud environments a reality. But it has also introduced a tremendous amount of complexity into IT departments.

Andreas Grabner, a DevOps activist from software intelligence platform Dynatrace, recently noted that in enterprise Kubernetes environments, “there are billions of interdependencies to account for.” Yes, billions.

In this podcast, Grabner explains how AI technology can tame this otherwise overwhelming Kubernetes complexity. We also discuss AI observability use cases for operators and developers, what value telemetry data brings to operations teams managing Kubernetes, and the cultural changes in development teams during the Kubernetes era.

[Listen on SoundCloud](#)

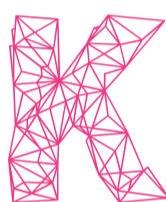
[Watch on YouTube](#)



Andreas Grabner is a DevOps activist at Dynatrace. He helps developers, testers, operations and xOps professionals become more efficient in their job by using Dynatrace.

CHAPTER 02

Mapping the Kubernetes Ecosystem

 Kubernetes has won the battle of orchestration tools to become the preferred container management platform. It has emerged as the consistent, unified infrastructure software that runs within enterprise data centers, public cloud platforms, hybrid cloud platforms, hyperconverged infrastructure appliances, and even at the edge.

Kubernetes is the foundation of modern infrastructure that runs contemporary workloads, often referred to as cloud native applications. The core infrastructure layer is augmented and complemented by open source projects and commercial software that form the cloud native stack.

This chapter provides a big picture view of cloud native technologies, followed by a detailed discussion of the building blocks of the cloud native stack.

From the operating system to the developer experience, we will dissect the stack to take a closer look at the leading open source projects and commercial offerings delivering cloud native computing to businesses and users.

Note that this chapter does not cover anything outside of the Kubernetes core. For instance, database systems are a part of the broader Kubernetes ecosystem. In particular, Databases as a Service (DBaaS) have grown in influence over the past

several years, to include products such as [Amazon Aurora](#), [Azure SQL Database](#), [MongoDB Atlas](#) and [Redis Cloud Essentials](#). However, the database is not a core building block of the cloud native stack; rather, it is a workload. The key component enabling databases and stateful workloads is storage, a core part of the cloud native stack which we cover below.

The Rise of Cloud Native and CaaS

According to The Linux Foundation, cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container and dynamically orchestrating those containers to optimize resource utilization.

Cloud native is a term used to describe container-based environments. Cloud native technologies are used to develop applications built with services packaged in containers, deployed as microservices, and managed on elastic infrastructure through agile DevOps processes and continuous delivery workflows.

One of the key attributes of cloud native is portability, which is possible only when the infrastructure is consistent across environments. Thus, Kubernetes becomes the lowest common denominator of infrastructure and also the foundation of the cloud native stack.

While Kubernetes is an important element of the cloud native stack, developers and DevOps engineers need additional software to deploy, scale and manage modern applications. Platform vendors such as [Red Hat](#) and [VMware](#) offer end-to-end platforms based on Kubernetes. Public cloud vendors — including [Amazon Web Services](#) (AWS), [Google Cloud Platform](#) (GCP) and [Microsoft Azure](#) — offer Kubernetes-based managed services running on existing compute, storage and network infrastructure.

Integrated container management platforms based on Kubernetes created a new category of application delivery model: Containers as a Service (CaaS). Similar to a

Platform as a Service (PaaS), the container management platform can be deployed behind the firewall running in an enterprise data center, or consumed as a managed cloud service offering.

With CaaS as the common fabric across the data center and public cloud, organizations can build hybrid applications that securely connect internal assets to the public cloud. CaaS is fast becoming an enabler of the hybrid cloud and multicloud deployments. Developers and operators can easily move applications across disparate environments.

Key Attributes of a Container Management Platform

Irrespective of where it is deployed, the container management platform has to meet the below requirements:

- **Consistent platform:** Developers and operators expect a consistent experience in public cloud and on-premises environments.
- **DevOps processes:** Integration with proven DevOps practices that ensure the rapid delivery of software.
- **Security:** Container management platforms must ensure the security of infrastructure and applications. They should detect vulnerabilities before deploying applications, while constantly monitoring the infrastructure for potential violations.
- **Infrastructure reliability:** The platform should support a service-level agreement (SLA)-driven service delivery model that delivers maximum uptime of both the infrastructure and platform.
- **High availability of workloads:** Apart from infrastructure, business applications deployed in the platform need to be highly available.
- **Observability:** The platform should provide insights into the infrastructure, resources and applications by capturing metrics, events, logs and traces from

the entire stack and storing them in a centrally accessible location.

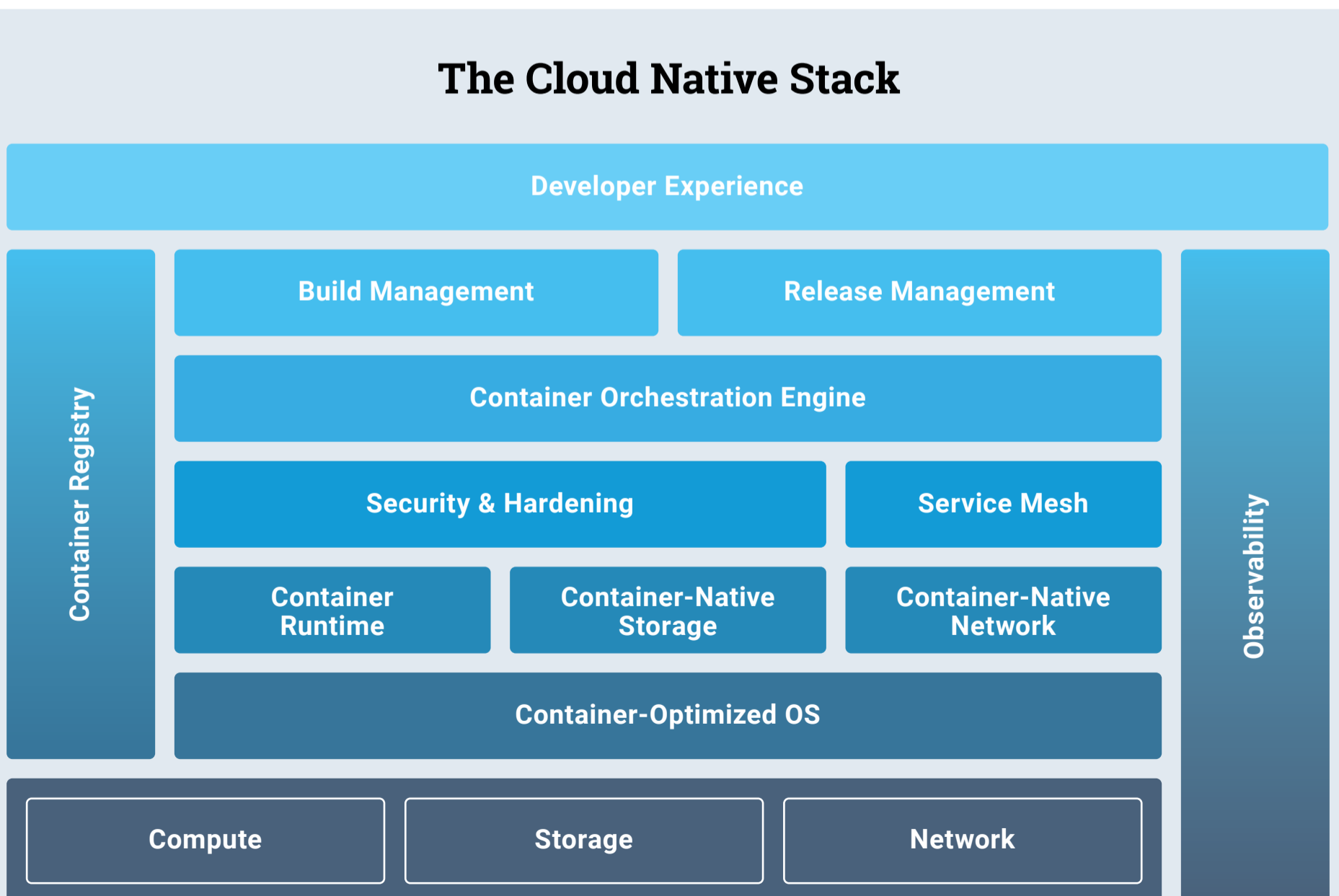
- Multitenancy and policy-driven management:** Optionally, the container management platform has to provide strong isolation among the tenants utilizing the platform. Application deployment and management should be governed by well-defined policies.

The Big Picture of Container Management Platforms

The modern cloud native stack delivered as an integrated container management platform is composed of multiple building blocks. Some of these building blocks are available as open source projects, while others are commercial offerings sold by independent software vendors.

The bottom-most layer represents the physical infrastructure of the cluster in the form of compute, storage and network components. The platform adds various

FIG 2.1: *The Cloud Native Stack.*



layers of abstraction to optimally utilize the underlying physical infrastructure.

Let's take a closer look at each of these layers.

Container-Optimized Operating System

Commercial Products	Vendor	Open Source Projects	CNCF Status
Fedora CoreOS	Red Hat	Flatcar Container Linux	Not Submitted
Talos	Talos Systems	RancherOS	Not Submitted

Containers redefined the role of an operating system (OS). With much of the heavy lifting moving to container runtimes, an OS has become a thin layer that provides access to physical resources. This shift has resulted in a new breed of operating systems called container-optimized OS (COS).

When compared to a traditional OS, COS is a lightweight OS with a much smaller footprint. It contains the most essential components that are required to run the container runtime. Choosing the right COS goes a long way in maintaining the CaaS deployment.

Customers can choose between [Fedora CoreOS](#) from Red Hat, [Talos](#) from Talos Systems, [Flatcar Container Linux](#) from Kinvolk GmbH., or [RancherOS](#) from Rancher Labs (in the process of being acquired by SUSE as of July 2020) to deploy the COS.

Most of the vendors offer an optional commercial subscription plan that includes regular updates, patches, and professional support.

Container Runtime

The container runtime is responsible for managing the life cycle of a container, providing the execution environment, and acting as an interface between the workload and the host operating system.

In 2015, the [Open Container Initiative](#) (OCI) was launched by The Linux Foundation

Commercial Products	Vendor	Open Source Projects	CNCF Status
Docker Engine Enterprise	Mirantis	containerd	Graduated
--	--	CRI-O	Incubation
--	--	Docker-CE	Not Submitted
--	--	Kata Containers	Not Submitted
--	--	runC	Not Submitted

to bring parity among the container runtime implementations. The OCI currently defines two specifications: the [Runtime Specification](#) (runtime-spec) and the [Image Format Specification](#) (image-spec).

According to the OCI website, the Runtime Specification outlines how to run a “filesystem bundle” that is unpacked onto a disk. At a high level, an OCI implementation would download an OCI Image and then unpack that image into an OCI Runtime file system bundle.

The Image Format Specification defines how to create an OCI Image — which will generally be done by a build system — and how to output an image manifest, filesystem (layer) serialization, and image configuration.

After the acquisition of Docker Enterprise by Mirantis, the commercial edition of Docker Engine ([Docker Engine Enterprise](#)) is sold by Mirantis; this offers enterprise-class support and professional services.

The [containerd project](#) has evolved as an industry standard for the container runtime. It’s a CNCF graduated project which is used in many production environments. [CRI-O](#) is currently a [CNCF incubation project](#) with active participation from the community.

[Docker Engine](#) (now Docker-CE) is one of the most popular container runtimes used by container management platforms. [Frakti](#) (an older approach) is a hypervisor-based container runtime for Kubernetes which provides a stronger

isolation by running pods in dedicated VMs. Apart from these, other choices include [Kata Containers](#) and [runC](#).

Container-Native Storage

Commercial Products	Vendor	Open Source Projects	CNCF Status
Red Hat OpenShift Container Storage	Red Hat	Ceph	Not Submitted
Kubera	MayaData	Longhorn	Sandbox
Portworx	Portworx	OpenEBS	Sandbox
Robin	Robin Systems	Rook	Incubating
StorageOS	StorageOS	--	--
Trident	NetApp	--	--

Storage is one of the most critical components of a CaaS platform. Container-native storage exposes the underlying storage services to containers and microservices. Like software-defined storage, it aggregates and pools storage resources from disparate mediums.

Container-native storage enables stateful workloads to run within containers by providing persistent volumes. Combined with Kubernetes primitives such as [StatefulSets](#), it delivers the reliability and stability to run mission-critical workloads in production environments.

Even though Kubernetes can use traditional, distributed file systems such as network file system (NFS) and GlusterFS, we recommend using a container-aware storage fabric which is designed to address the requirements of stateful workloads running in production. Customers can choose from a variety of open source projects and commercial implementations.

The cloud native ecosystem has defined specifications for storage through the [Container Storage Interface](#) (CSI), which encourages a standard, portable approach to implementing and consuming storage services by containerized workloads.

[Ceph](#), [Longhorn](#), [OpenEBS](#) and [Rook](#) are some container-native storage open source projects, while [Kubera](#) by MayaData, [Trident](#) by NetApp, [Portworx](#), [Red Hat OpenShift Container Storage](#), [Robin](#) by Robin System and [StorageOS](#) are commercial offerings combined with support.

Traditional vendors such as [NetApp](#), [Pure Storage](#) and [VMware](#) also provide storage plugins for Kubernetes.

Addressing Infrastructure Challenges

Managed Kubernetes offerings can reduce the complexity and skills needed to manage large container deployments. Streamlining the infrastructure supporting Kubernetes workloads is one of the most significant criteria when IT professionals evaluate their technology roadmaps.

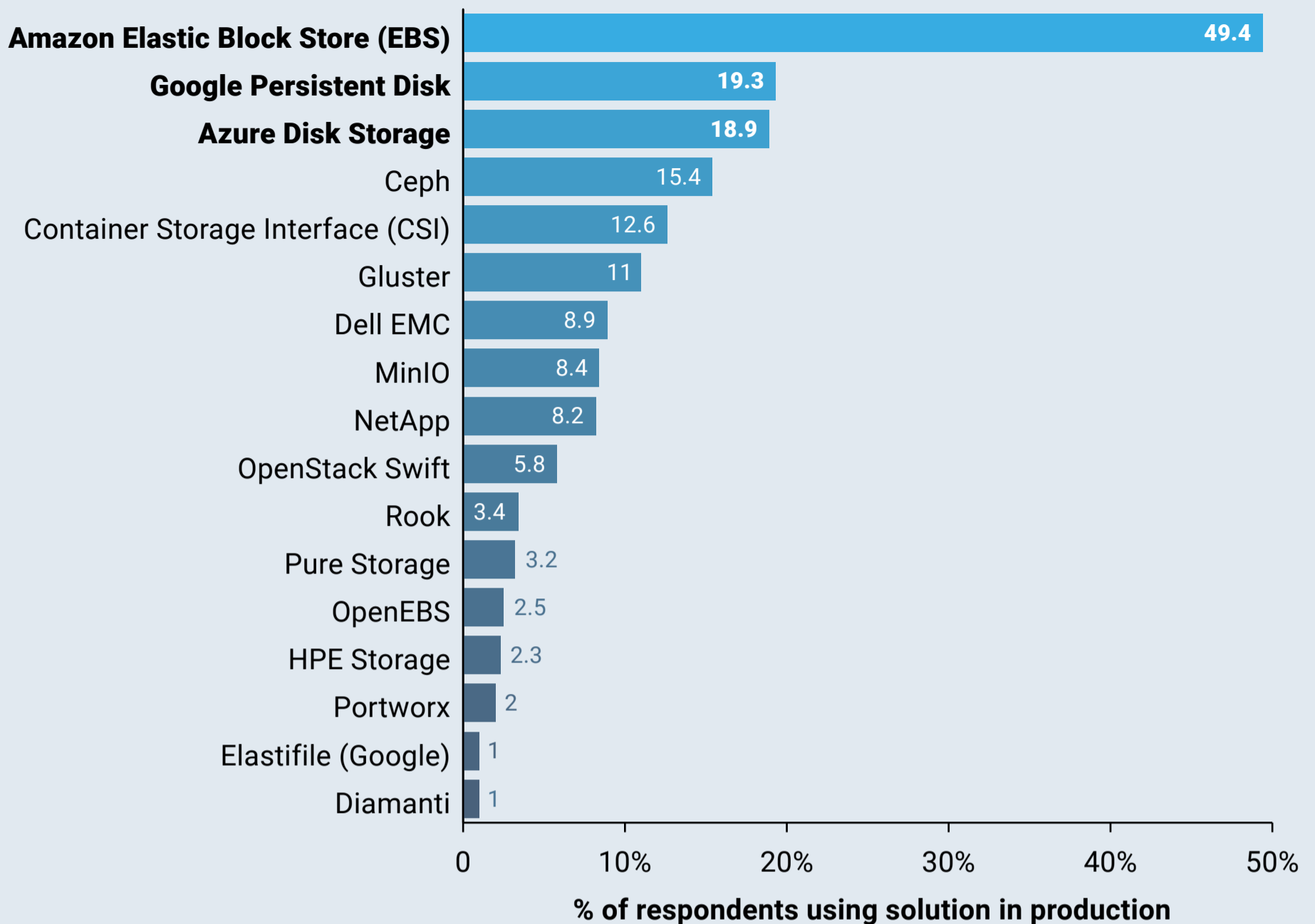
We'll be looking at data from the [2019 CNCF survey](#) about current and future Kubernetes adoption plans, along with the container challenges facing Kubernetes users. The results showed that early technology adoption was influenced by then-current vendor relationships. However, satisfaction was mixed.

Many Kubernetes users included their current storage and cloud vendors on their cloud native shortlists. Users appeared to have trouble shortening that list. All 38 of the survey's choices were evaluated by at least 5% of Kubernetes users.

With the rise of managed Kubernetes, cloud providers exposed block storage through storage classes and dynamic provisioning. Customers could attach Amazon Elastic Block Store (EBS) volumes on AWS, Azure managed disks, Google Persistent Disks and to Kubernetes worker nodes running in AWS, GCP and Microsoft Azure. This gave cloud providers an edge.

When asked about the cloud native storage they utilize (see figure 2.2 below), Amazon EBS, Google Persistent Disk and Azure Disk Storage were cited as the most used by Kubernetes users. In many cases, StatefulSets allowed cluster workloads to

Most-Used Cloud Native Storage Solutions are by Cloud Providers



Source: The New Stack's analysis of CNCF's 2019 survey. Q. Please indicate if your company/organization is evaluating, or currently using in production, any of these cloud native storage projects. What are your challenges in using / deploying containers? Please select all that apply. The chart only shows data for offerings used by at least 1% of respondents with at least one Kubernetes cluster. n=1149.

© 2020 THE NEW STACK

FIG 2.2: File systems like Ceph are often seen as competitive to cloud storage, versus offerings from more traditional storage companies.

access the block storage offered by the cloud provider. While widely adopted, block storage from big cloud providers was not designed specifically for Kubernetes workloads.

Next up on the list were Ceph, CSI and Gluster, with 37% of Gluster users also using Ceph. Ceph and Gluster are distributed file systems that add a persistence layer across multiple nodes. However, they are not well integrated into Kubernetes tools and workflow, so storage administrators may find them more difficult to maintain and configure.

Lower on the list were offerings from established storage-focused companies like

Dell EMC, NetApp and Pure Storage. Initially Kubernetes had integrated volume plugins to connect to these company's storage backends. Unfortunately, the upstream Kubernetes distribution became bloated, meaning that any minor update or change to a plugin meant rebuilding and compiling the entire code.

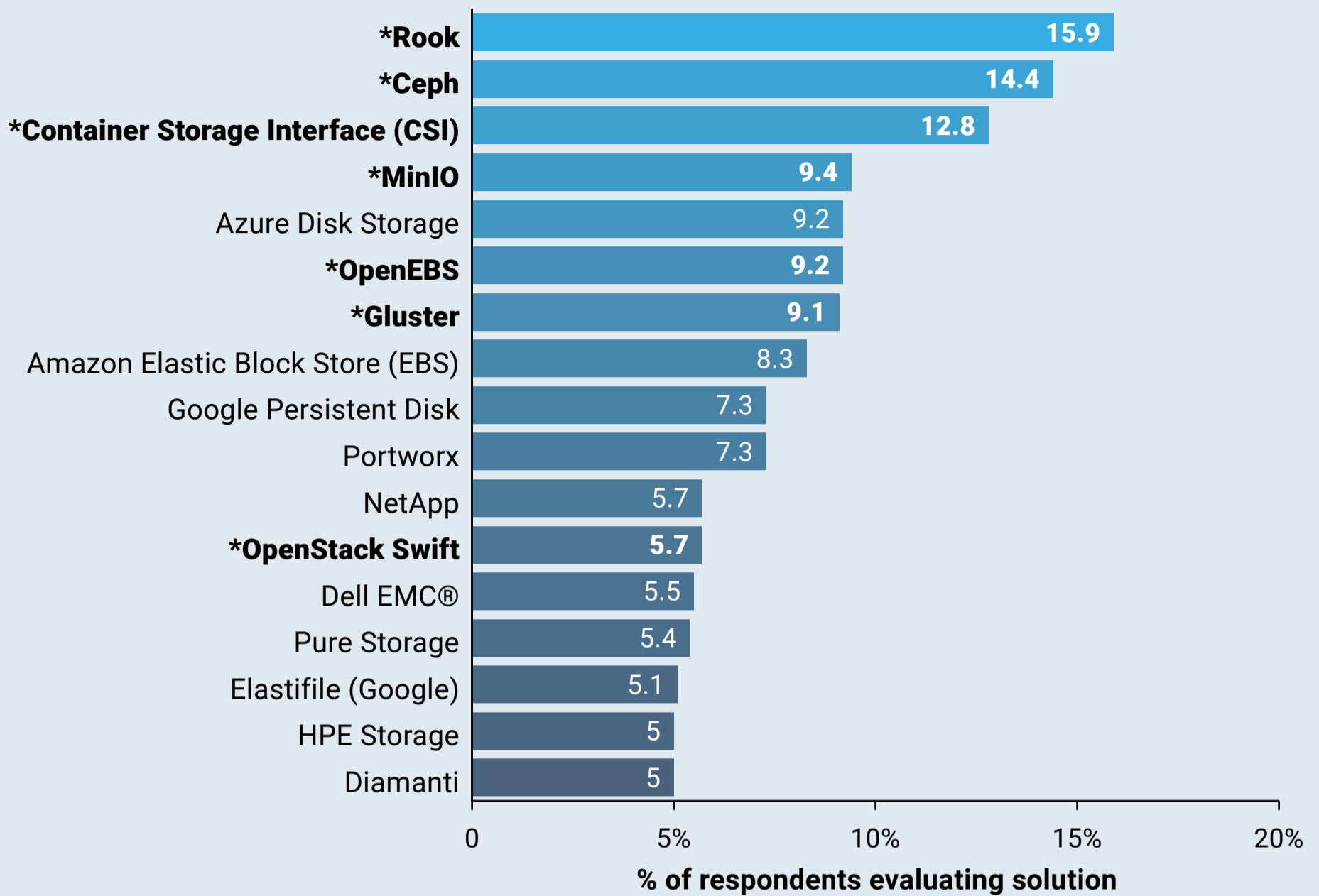
Customers of traditional storage companies were significantly more likely to complain of storage challenges. For example, 46% of Pure Storage customers had challenges handling container-related storage, compared to just 27% for the average Kubernetes user. There was hope on the horizon though, as 13% utilized the Container Storage Interface (CSI). CSI became generally available for Kubernetes in 2019 and eliminates the challenge of constantly needing to integrate upstream. Traditional storage vendors, cloud providers and pure-play container storage companies like Portworx are switching to CSI.

CSI was on the minds of people evaluating new options to address their current container storage challenges. While overall only 13% of Kubernetes users were considering CSI, that jumped to 22% among those that had storage challenges.

While established companies were being considered by some, open source projects were top of mind among those looking for new storage options (see figure 2.3). Compared to the average respondent, the 26.6% of Kubernetes users who were storage-challenged were more likely to evaluate Rook (25.9% vs 15.9%), Ceph (23.4% vs 14.4%), Gluster (14.6% vs 9.1%), OpenEBS (14.6% vs 9.2%) and MinIO (12.8% vs 9.4%). These open source efforts were notably not driven by a need to sell hardware.

For both traditional storage companies and the newer breed of uniquely cloud native storage offerings, users were more likely to cite storage challenges. However, by implementing new approaches like CSI, traditional storage companies were addressing the concerns of their customers. While many users of newer offerings — like MayaData's OpenEBS, Minio and Portworx — indicated they had storage challenges, they were likely referring to problems they had connecting legacy datastores.

Open Source* Options Targeted by the Storage Challenged



Source: The New Stack's analysis of CNCF's 2019 survey. Q. Please indicate if your company/organization is evaluating, or currently using in production, any of these cloud native storage projects. What are your challenges in using / deploying containers? Please select all that apply. The chart only shows data for offerings used by at least 1% of respondents with at least one Kubernetes cluster. n=1149.

© 2020 THE NEW STACK

FIG 2.3: Further analysis of Kubernetes users who cited storage challenges (data not shown) reveals that consideration of Rook, Ceph, and OpenEBS is 50% higher among those with storage challenges. All three have CSI drivers.

Implementation challenges were common for early adopters of third-party, best-of-breed solutions. As time goes on, it will be interesting to evaluate how effective the new players will be. It could impact the ability of cloud and traditional storage companies to retain their customers in this segment.

Container-Native Networking

Similar to container-native storage, the container-native network abstracts the physical network infrastructure to expose a flat network to containers. It is tightly integrated with Kubernetes to tackle the challenges involved in pod-to-pod, node-

Commercial Products	Vendor	Open Source Projects	CNCF Status
Calico Enterprise	Tigera	Cilium	Not Submitted
Weave Net (contact vendor for Enterprise subscription)	Weaveworks	Contiv	Not Submitted
--	--	Flannel	Not Submitted
--	--	Project Calico	Not Submitted
--	--	Tungsten Fabric	Not Submitted
--	--	Weave Net	Not Submitted

to-node, pod-to-service and external communication.

Kubernetes can support a host of plugins based on the [Container Network Interface](#) (CNI) specification, which defines the network connectivity of containers and deals with the network resources when the container is deleted. The CNI project is a part of CNCF incubating projects.

Container-native networks go beyond basic connectivity. They provide dynamic enforcement of network security rules. Through a predefined policy, it is possible to configure fine-grained control over communications between containers, pods and nodes.

Choosing the right networking stack is critical to maintain and secure the CaaS platform. Customers can select the stack from open source projects including [Cilium](#), [Contiv](#), [Flannel](#), [Project Calico](#), [Tungsten Fabric](#) and [Weave Net](#). On the commercial side, Tigera offers [Calico Enterprise](#) and an enterprise subscription of [Weave Net](#) can be purchased by contacting Weaveworks.

Managed CaaS offerings from public cloud vendors come with tight integration of the existing virtual networking stack. For example, AWS has a [CNI plugin](#) for Amazon Elastic Kubernetes Service (EKS) based on Amazon Virtual Private Cloud (VPC), while Microsoft has built [Azure Virtual Network Container Networking Interface](#) (CNI) for Azure Kubernetes Service (AKS).

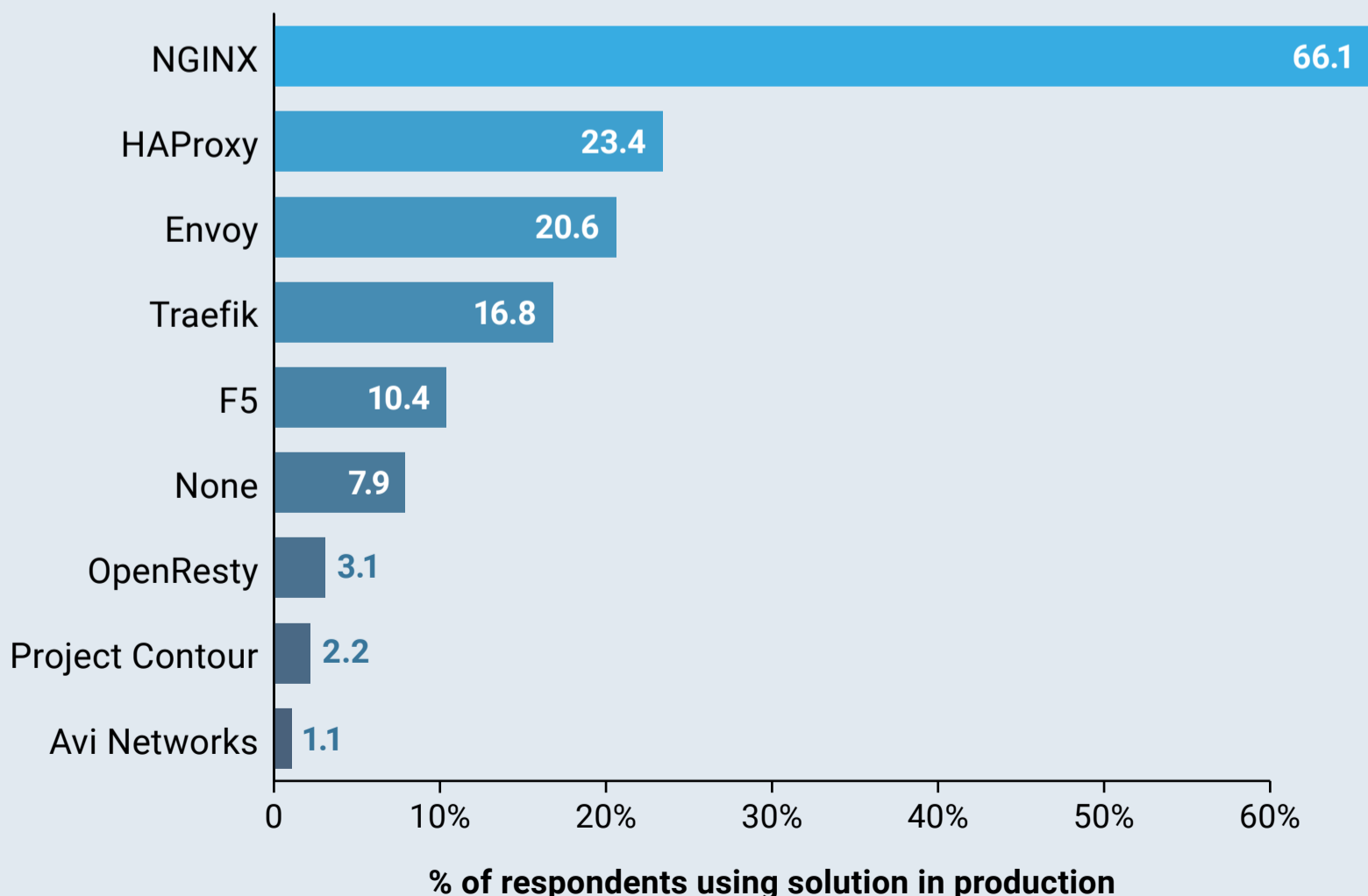
Let's return now to data from the [2019 CNCF survey](#), this time in reference to networking.

Networking is a challenge that has declined over the years, although Kubernetes users continued to assess ingress providers. In fact, while a Kubernetes user had an average of 1.5 ingress providers, the 28% of respondents who cited networking as a challenge had on average three ingress providers. NGINX was in 66% of Kubernetes stacks — but by itself, it wasn't able to address the needs of all users. Adoption of HAProxy and Envoy was lower, but rose among those with networking challenges.

This suggests that newer market entrants were adopted to address problems. Looking forward, expect offerings to differentiate themselves based on the protocols they support and whether or not they include an API gateway.

FIG 2.4: *Although involved with ingress, using NGINX with Kubernetes has little impact on networking challenges.*

HAProxy, Envoy and Traefik Have Yet to Reach NGINX's Adoption Level



Source: The New Stack's analysis of CNCF's 2019 survey. Source: Q. What Kubernetes ingress provider(s) (i.e., service proxy) are you using? Please select all that apply. What are your challenges in using / deploying containers? Please select all that apply. The chart only shows data for offerings used by at least 1% of respondents with at least one Kubernetes cluster. n=1197.

Security & Hardening

Commercial Products	Vendor	Open Source Projects	CNCF Status
Aqua	Aqua Security Software	Clair	Not Submitted
Calico Enterprise	Tigera	Falco	Incubating
Prisma Cloud	Palo Alto Networks	Open Policy Agent (OPA)	Incubating
Snyk	Snyk	Snyk (Open Source)	Not Submitted
StackRox	StackRox	--	--

Container management platforms are secured by hardening the network, scanning container images, and performing regular risk profiling and threat detection.

Network security is an important part of hardening the platform. Enterprise IT should invest in a zero-trust network layer to meet security and compliance requirements. These networks enforce fine-grained policy control at multiple points in the infrastructure, analyze for anomalies, and encrypt and authorize traffic flowing between microservices using secure protocols like mutual Transport Layer Security (mTLS). Calico Enterprise by [Tigera](#) is one of the leading zero-trust network providers for Kubernetes and CaaS platforms.

A secure container platform implements the below techniques to ensure that the infrastructure and applications are secure:

- A secure container platform uses **trusted images** that are signed and verified. This feature is tightly integrated with the container registry component that stores container images.
- It implements an **encrypted store** to securely store and retrieve the secrets, including usernames, passwords and other sensitive data.
- The platform is integrated with existing lightweight directory access protocol (LDAP) and Active Directory (AD) deployment to configure integrated **role-based access control** (RBAC).

[Aqua](#), [Prisma Cloud](#), and [StackRox](#) provide commercial solutions for securing containers and Kubernetes infrastructure. [Snyk](#) provides both commercial and open source solutions.

Open source projects such as [Clair](#) deliver static analysis of vulnerabilities in application containers.

[Falco](#), a CNCF incubating project, is a behavioral activity monitor designed to detect anomalous activity in cloud native applications. Falco audits a system at the most fundamental level: the kernel. Falco then enriches this data with other input streams, such as container runtime metrics and Kubernetes metrics, to alert users when one of the predefined rules are met.

[Open Policy Agent](#) (OPA), another CNCF incubating project, provides a unified toolset and framework for policies across the stack. OPA provides a high-level declarative language that lets developers and operators specify policy as code and simple APIs to offload policy decision-making from the software. OPA can enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways and more.

Service Mesh

Commercial Products	Vendor	Open Source Projects	CNCF Status
Aspen Mesh	F5	Consul	Not Submitted
Consul Enterprise	HashiCorp	Contour	Incubating
Grey Matter	Decipher Technology Studios	Envoy	Graduated
Kong Enterprise	Kong	Istio	Not Submitted
Linkerd Commercial Support	Buoyant	Kuma	Sandbox
Maesh	Containous	Linkerd	Incubating
Service Mesh Hub	Solo.io	Service Mesh Interface (SMI)	Sandbox
Tetrate	Tetrate	Zuul	Not Submitted

Service mesh is becoming a key component of the cloud native stack. It enables fine-grained control of traffic among various microservices, while providing insights into the health of each microservice.

Service mesh complements container-native networking. It focuses on east-west traffic, while the north-south traffic is handled by the core networking layer.

Service mesh adds a proxy to each microservice, which intercepts the inbound and outbound traffic. Since it is placed close to the microservice, it can track the health of that service.

[Consul](#), [Contour](#), [Envoy](#), [Istio](#), [Kuma](#), [Linkerd](#), [Service Mesh Interface](#) (SMI), and [Zuul](#) are some of the popular open source service mesh projects.

Customers can choose from commercial implementations such as [Aspen Mesh](#) by F5, [Consul Enterprise](#) by HashiCorp, [Grey Matter](#) by Decipher Technology Studios, [Kong Enterprise](#) by Kong, [Linkerd Commercial Support](#) by Buoyant, [Maesh](#) by Containous, [Service Mesh Hub](#) by Solo.io and [Tetrate](#).

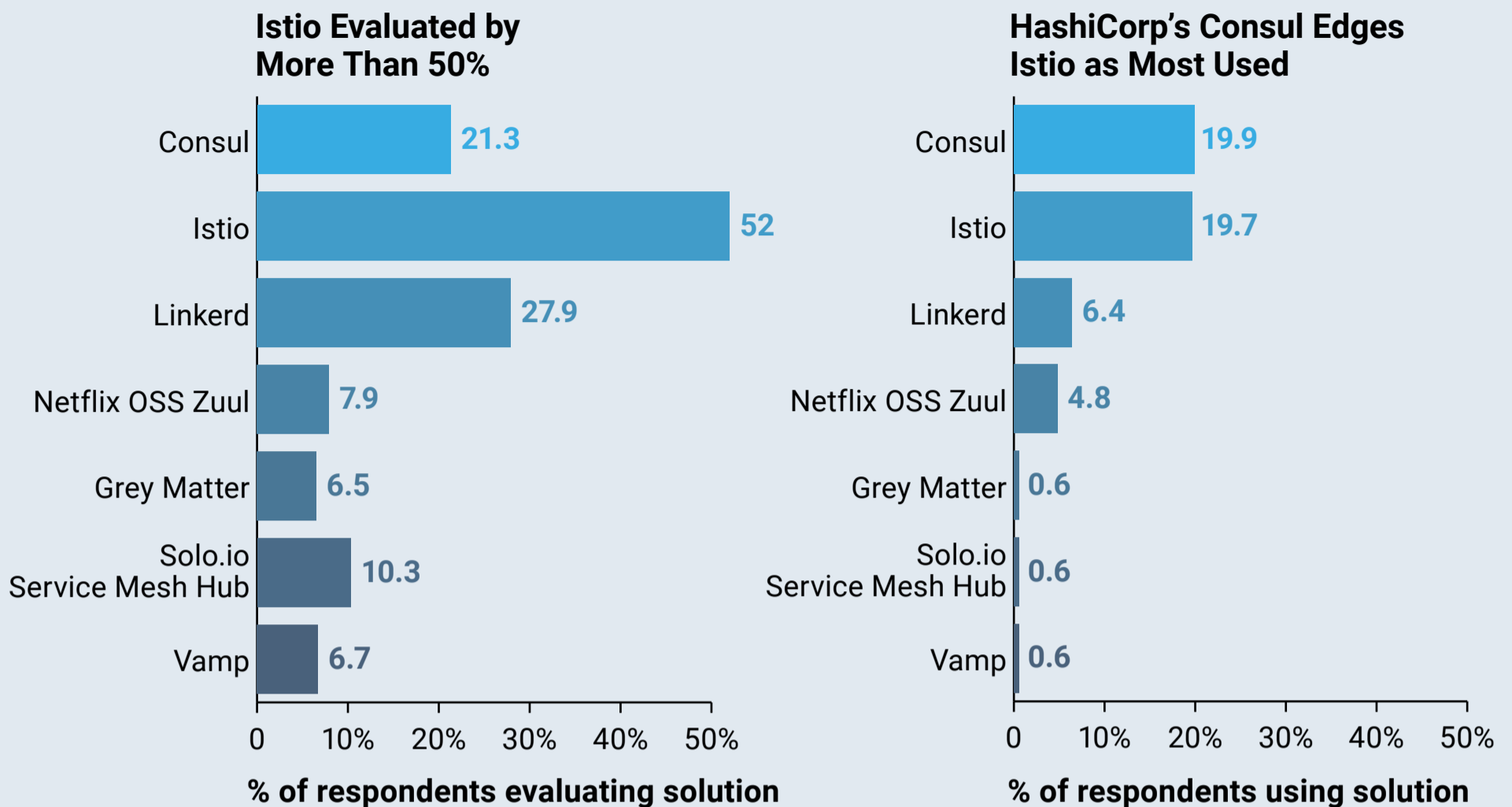
Returning once more to data from the [2019 CNCF survey](#), now we're going to look at service mesh.

According to the survey results, service meshes were challenging for 27% of Kubernetes users — both in regards to picking the right tool and in day-to-day operations.

HashiCorp's [Consul](#) and [Istio](#) were in a virtual tie as the service mesh that's used most often by Kubernetes users. However, many of the people who mentioned Consul may have already been using it for service discovery and were just experimenting with its service mesh use case.

That wasn't the case for Istio and [Linkerd](#), and both got significantly more evaluation among those who found service mesh challenging.

Users Looking For Service Mesh Solution Are Highly Likely to Evaluate Istio or Linkerd



Source: The New Stack's analysis of CNCF's 2019 survey. Q. Please indicate if your company/organization is evaluating, or currently using in production, any of these service mesh projects / products. Please select all that apply. What are your challenges in using / deploying containers? Please select all that apply. Data only includes respondents that have at least one Kubernetes cluster. n=1082.

© 2020 THE NEW STACK

FIG 2.5: An existing base of developers using Consul for service discovery likely means that people are using it for related service mesh functionality as well. At the same time, over half of all Kubernetes users are evaluating Istio for the first time.

Netflix's [Zuul](#) got an honorable mention as the fourth most likely to be on the radar of users. Notably, only 23% of Zuul users complained about service mesh challenges, as compared to 27% for all respondents. Since it had less name recognition than Istio, the respondents that were using it were more likely to have learned about Zuul because of its capability to address a specific need.

Grey Matter, Service Mesh Hub and Vamp had few, if any, users, but decent name recognition gave them a place on shortlists.

Looking forward, we expect that users of all service meshes will be challenged by how well they integrate with the rest of their Kubernetes stack. In the meantime, expect to see Istio, Linkerd, and other services try to improve their integration with Envoy.

Container Orchestration Engine

Commercial Products	Vendor	Open Source Projects	CNCF Status
Canonical Distribution of Kubernetes (CDK)	Canonical	Kubernetes	Graduated
Cisco Container Platform	Cisco	OKD	Not Submitted
Docker Enterprise	Mirantis	Rancher Kubernetes Engine (RKE)	Not Submitted
HPE Ezmeral Container Platform	HPE	--	--
Mesosphere Kubernetes Engine (MKE)	D2IQ	--	--
Mirantis Cloud Platform	Mirantis	--	--
Rancher Kubernetes Engine (RKE)	Rancher Labs	--	--
Red Hat OpenShift	Red Hat	--	--
SUSE CaaS Platform	SUSE	--	--
Tanzu Kubernetes Grid	VMware	--	--

As discussed in chapter one, the most common container orchestration engine for CaaS platforms is Kubernetes.

Even though [Kubernetes](#) may be deployed from the upstream open source distribution available on GitHub, customers may want to invest in a commercial distribution that comes with maintenance and support packages. [Canonical Distribution of Kubernetes](#), [Cisco Container Platform](#), [Docker Enterprise](#), [HPE Ezmeral Container Platform](#), [Mesosphere Kubernetes Engine](#), [Mirantis Cloud Platform](#), [Rancher Kubernetes Engine](#), [Red Hat OpenShift](#) (which can also be used as a PaaS), [SUSE CaaS Platform](#) and [VMware Tanzu Kubernetes Grid](#) are some of the commercial distributions in the market.

There are open source projects available too, such as [OKD](#) and [Rancher Kubernetes](#).

[Engine](#) (RKE).

Customers with investment in public cloud CaaS platforms based on Google Kubernetes Engine or IBM Kubernetes Service can choose [Anthos](#) or [IBM Cloud Paks](#), which are tightly integrated with the control plane running in the cloud.

The previous chapter had a detailed discussion of Kubernetes delivery models.

Container Registry

Commercial Products	Vendor	Open Source Projects	CNCF Status
Docker Hub	Docker	Docker Registry	Not Submitted
Docker Trusted Registry	Mirantis	Harbor	Graduated
GitLab Container Registry	GitLab	Project Quay	Not Submitted
JFrog Container Registry / Artifactory	JFrog	--	--
Red Hat Quay	Red Hat	--	--

Microservices are packaged as container images before they are deployed and scaled by the orchestration engine. A container registry acts as the central repository that stores container images. It is integrated with build management tools for automatically generating images each time a new version of the service is built. A release management tool picks the latest version of the image from the registry and deploys it.

[Docker Trusted Registry](#) from Mirantis, [GitLab Container Registry](#), [JFrog Container Registry](#) and [Red Hat Quay](#) are some of the commercially available container registries in addition to those offered by the cloud providers. Docker also offers a hosted registry called [Docker Hub](#) and its open source counterpart, [Docker Registry](#).

Apart from storing the container images, these products typically perform vulnerability scanning on images, authentication and authorization of users. Some

enterprises require that repositories exist on-premises — they disallow access to internet-based registries. In those cases, some of the commercial offerings include internal registries. Otherwise, it is possible to build an internal registry with many of the same features.

Cloud providers offering a managed Kubernetes service provide an option to host a private container registry within the customer account. Since the cluster and registry are co-located in the same region and account, they deliver low-latency and security.

[Harbor](#), originally developed by [VMware](#), is an open source container image registry that secures images with role-based access control, scans images for vulnerabilities, and signs images as trusted images.. [Project Quay](#) is the open source distribution of Red Hat’s Quay. It offers a consumer-grade web UI, image vulnerability scanning, and enterprise-grade data storage and protection.

Build Management

Commercial Products	Vendor	Open Source Projects	CNCF Status
CircleCI	Circle Internet Services	GitLab	Not Submitted
CloudBees CI	CloudBees	Jenkins / Jenkins X	Not Submitted
Digital.ai	Digital.ai Software	--	--
GitHub Actions	GitHub	--	--
GitLab CI	GitLab	--	--
JFrog Pipelines	JFrog	--	--
SemaphoreCI	Rendered Text	--	--
Travis CI	Travis CI	--	--

For rapid delivery of microservices, each time code is committed to the repository, a new container image is built and pushed into the container registry. These images may then be deployed to staging or test environments within CaaS for automated and manual testing.

Build management involves converting the latest version of code into various artifacts, including container images, libraries and executables. It forms an important stage of the continuous integration and continuous deployment (CI/CD) pipeline.

The source code management system, based on internal or external Git repositories, triggers an automated and secure build process, which will result in a deployment artifact. The outcome from this stage may include container images, Helm charts and Kubernetes deployments.

[GitLab](#) and [Jenkins](#) are popular build management software products, available in both open source and commercial versions. [CloudBees CI](#) is a commercial version of Jenkins build management server.

[CircleCI](#), [Digital.ai](#), [GitHub Actions](#), [JFrog Pipelines](#), [SemaphoreCI](#) and [Travis CI](#) are commercial CI/CD solutions for microservices.

Commercial Products	Vendor	Open Source Projects	CNCF Status
Armory Enterprise Spinnaker	Armory	Argo CD	Incubating
Open Enterprise Spinnaker	OpsMx	Flux	Sandbox

Release Management

Release management represents the final stage of a CI/CD pipeline. It involves deploying a fully-tested version of microservices in the production environment. Advanced deployment strategies, such as canary releases, blue/green deployments, rollbacks and roll forwards, are a part of release management.

[Spinnaker](#) is one of the most popular release management tools for microservices. It complements Jenkins and other build management tools by automating the management and deployment of artifacts. [Armory](#) and [OpsMx](#) have commercial

implementations of Spinnaker.

GitOps is an emerging technique to implement Configuration as Code (CaC). The YAML artifacts and Helm Charts are version controlled and maintained in a Git repository. Using a pull or push strategy, the changes made to the configuration are applied to the cluster. [Argo CD](#) (CNCF incubating project) and [Flux](#) (CNCF sandbox project) are two popular open source projects for configuring GitOps.

Commercial Products	Vendor	Open Source Projects	CNCF Status
AppDynamics	Cisco	Fluentd	Graduated
Datadog	Datadog	Grafana	Not Submitted
Dynatrace	Dynatrace	Jaeger	Graduated
Honeycomb	Honeycomb	OpenTelemetry	Sandbox
Lightstep	Lightstep	OpenTracing	Incubating
New Relic One	New Relic	Prometheus	Graduated
Sysdig Monitor	Sysdig	--	--

Observability

Observability involves capturing the metrics, logs, events and traces from the entire stack.

Monitoring collects metrics from the infrastructure and application stack of the platform. A robust monitoring platform monitors the state of the Kubernetes cluster, as well as the deployed applications. The agents installed in each node collect detailed information about the resource utilization; cluster health; node health; storage and memory availability; number of containers running on each node; and detailed metrics related to Pods. Customers can deploy open source monitoring tools such as [Grafana](#) and [Prometheus](#), or invest in commercial platforms such as [DataDog](#), [Dynatrace](#), [New Relic One](#) and [Sysdig Monitor](#).

Logging collects and aggregates the information, warnings and errors raised by

various components of the cloud native platform. Almost every component of Kubernetes generates logs that provide detailed insight into the current state of the cluster. As a best practice, developers are encouraged to integrate logging into microservices. Various agents are deployed within the cluster to collect and stream the logs to a central repository. [Fluentd](#) is a popular open source data collection tool deployed in Kubernetes, which can be used in combination with Elastic and Kibana to visualize and search logs. Service mesh software such as Istio and Linkerd are tightly integrated with the logging platforms.

Prometheus and Fluentd are both CNCF graduated projects.

Since cloud native applications are assembled from disparate and autonomous services, it is important to track the chain of communication and the time it takes for each service to respond. This mechanism is critical to monitoring and analyzing application performance.

Open source tools like [Jaeger](#) and [OpenTracing](#) can deliver application performance monitoring (APM) capabilities to microservices. Commercial offerings include [AppDynamics](#), [Honeycomb](#) and [Lightstep](#), both of which provide end-to-end tracing and monitoring features for modern applications.

[OpenTelemetry](#) provides a single set of APIs, libraries, agents and collector services to capture distributed traces and metrics from cloud native applications.

OpenTracing is a CNCF incubating project, while OpenTelemetry is a sandbox project.

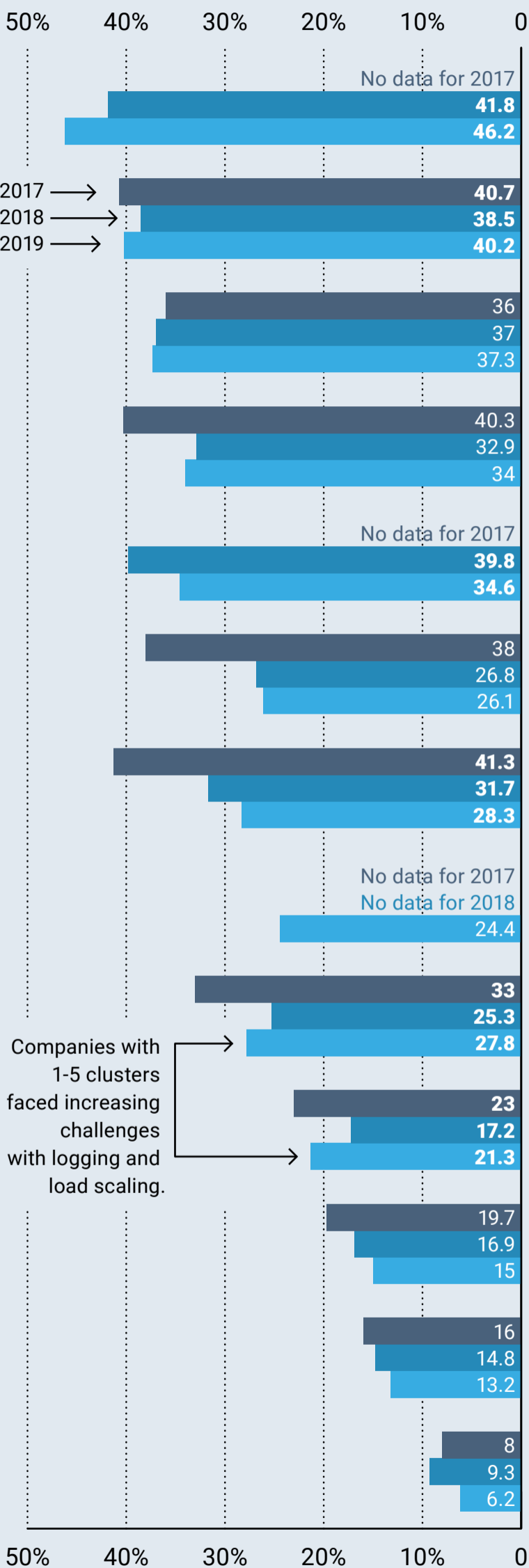
The Container Challenges of Kubernetes Users

Returning once more to data from the [CNCF 2017 – 2019 surveys](#), we're going to look at container challenges for Kubernetes users.

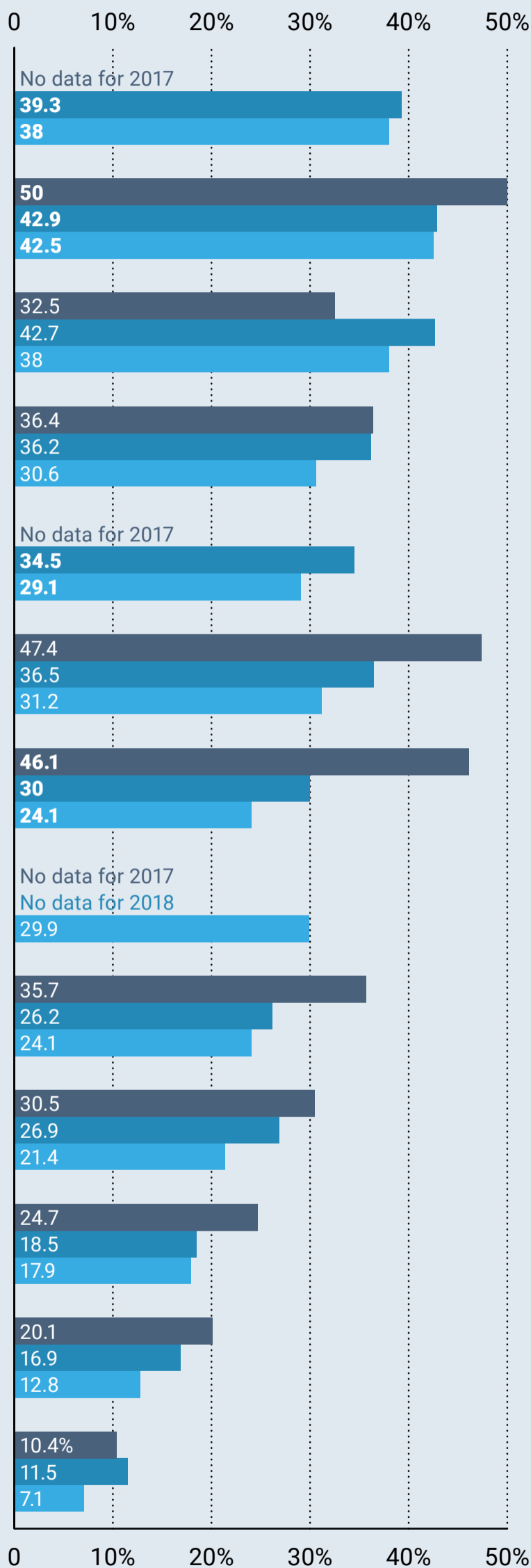
For companies with more than five clusters, container challenges eased markedly in 2018 and continued to improve over 2019. For Kubernetes users with one to five

Training and Storage Are Easier, but Culture Change and Security Continue to be Challenging

Kubernetes users with 1-5 clusters, who face challenges using / deploying containers



Kubernetes users with >5 clusters, who face challenges using / deploying containers



Companies with 1-5 clusters faced increasing challenges with logging and load scaling.

Source: The New Stack's analysis of CNCF's 2017, 2018 and 2019 surveys. Q. What are your challenges in using / deploying containers? Please select all that apply. Q. What are your challenges in using / deploying containers? Please select all that apply.

1-5 Kubernetes clusters: 2017, n=300; 2018, n=925; 2019, n=729. > 5 Kubernetes clusters: 2017, n=154; 2018, n=550; 2019, n=468.

FIG 2.6: Kubernetes users with 1-5 clusters face challenges with development team culture. Those with >5 clusters saw modest declines in 2019 for all challenges.

clusters, security, complexity and the cultural change it creates within the development team were the most cited challenges in using or deploying containers.

No matter how many clusters were in use, according to the CNCF, data networking and storage issues were being addressed. The decline in storage pain was maybe due to the maturity of the CSI spec and the availability of a CSI interface in most storage solutions. Networking challenges may have been less severe because Container Networking Interface (CNI)-based technologies like Flannel became the assumed standard for container communication within clusters.

The success of CNCF, KubeCon + CloudNativeCon and the rest of the open source ecosystem is demonstrated by the continued drop in users complaining about a lack of training or vendor support.

Respondents with more than five clusters continued to see a decline in challenges related to scaling of deployment. Scaling up workloads, monitoring and logging them in larger clusters, was a use case the Kubernetes ecosystem was addressing.

Companies with five or fewer clusters were faring worse, with challenges notably increasing in 2019 for scaling deployments, development culture and logging.

Changes in the development team's culture were particularly challenging, because smaller companies were not as far along in realigning their organizational structure. Although many organizations formed DevOps or SRE teams, the developer's role in these organizations continues to be pulled towards handling aspects of the infrastructure layer.

Enterprise Container Management Platforms

Platform vendors and operating system vendors have built commercial distributions of Kubernetes that can be installed in an enterprise data center. The infrastructure choices include physical, bare-metal servers, virtual machines

running on a hypervisor, or public cloud Infrastructure as a Service (IaaS).

Below is a list of the mainstream vendors offering a commercial distribution of Kubernetes:

Canonical

Canonical's Ubuntu is the most popular Linux OS in the public cloud. Ubuntu also runs on private clouds and Internet of Things (IoT) devices, and as a desktop PC operating system. Enterprise organizations prefer Ubuntu for its compatibility, performance and versatility.

Canonical has two flavors of Kubernetes:

- [MicroK8s](#): A lightweight distribution that can be installed on IoT and edge devices with a single command.
- [Charmed Distribution of Kubernetes](#): An upstream version of Kubernetes and fully conformant with the CNCF. Canonical offers a managed service for enterprises, which includes training, installation, configuration, orchestration tooling and optimization of production Kubernetes clusters.

Mirantis

Mirantis has pivoted from being a pure-play OpenStack company to a Kubernetes player. The company now has its own distribution of Kubernetes. It is positioned as an open cloud infrastructure company, with a platform that can seamlessly manage virtual machines and containers.

In 2019, Mirantis acquired [Docker Enterprise](#) from Docker, Inc. That includes Docker Engine Enterprise edition, Docker Desktop Enterprise, Docker Trusted Registry and Docker Enterprise CaaS.

Apart from Docker Enterprise, Mirantis also has a managed Kubernetes offering branded as [KaaS](#) — Kubernetes as a Service — which can be deployed on OpenStack or AWS.

Rancher

Rancher Labs (in the process of being acquired by SUSE as of July 2020) provides a platform for deploying and managing Kubernetes as an enterprise service.

[Rancher Kubernetes Engine](#) targets enterprise IT that is considering delivering Kubernetes as a Service to their employees. Apart from maintaining its own distribution of Kubernetes, Rancher can manage other hosted Container as a Service offerings — such as Amazon Elastic Container Service for Kubernetes (Amazon EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE) — while enforcing consistent security policies. The platform integrates with Active Directory, LDAP, and other IT services to deliver authentication, role-based access control, and security policies across multiple Kubernetes clusters.

Red Hat

[Red Hat OpenShift Container Platform](#) is an application platform built on Kubernetes. It is based on proven open source technologies, such as Red Hat Enterprise Linux, CRI-O, and Kubernetes, for end-to-end application lifecycle management.

Red Hat OpenShift Container Platform may be deployed in the public cloud or on-premises. The platform automates container and application builds, deployments, scaling, and health management. As a CNCF certified Kubernetes distribution, OpenShift enables portability and interoperability of containerized workloads.

OpenShift is available for deployment in private cloud and hybrid cloud environments, and also as a managed service in the public cloud. Red Hat has partnered with public vendors such as Amazon, Microsoft and Google to offer a managed OpenShift PaaS to customers.

SUSE

SUSE, the well-known Linux distribution company, has ventured into the CaaS

market with a Kubernetes distribution.

[SUSE CaaS Platform](#) supports deploying Kubernetes with multiple head nodes in public and private cloud environments. It includes enterprise security features like role-based access control, image scanning, and transport layer security (TLS) support. The platform also includes a purpose-built container host operating system, container runtime, and container image registry.

SUSE is leveraging its enterprise presence to push the CaaS platform. Like its competitors, SUSE is moving towards delivering an integrated stack based on the best of open source technologies.

VMware

After VMware acquired Pivotal Labs, it rebranded Pivotal Kubernetes Service (PKS) as VMware Tanzu Kubernetes Grid.

VMware's Kubernetes distribution is available in two flavors: Tanzu Kubernetes Grid and Tanzu Kubernetes Grid Integrated Edition (TKGI).

[Tanzu Kubernetes Grid](#) is a CNCF-certified, enterprise-ready Kubernetes runtime that streamlines operations across a multcloud infrastructure. It can be deployed on vSphere (on-premises), IaaS (public cloud), or resource-constrained devices (edge/IoT).

[Tanzu Kubernetes Grid Integrated Edition](#) (TKGI) is a production-grade, Kubernetes-based container solution equipped with advanced networking, a private container registry, and full life cycle management. It's tightly integrated with vSphere, vCenter, vSAN and NSX. It can be deployed either in a data center running a VMware stack, or in the public cloud running VMware Cloud (VMC) Foundation.

Managed Container Management Platforms

Below is a list of the mainstream vendors offering Kubernetes as a managed service:

Amazon Elastic Kubernetes Service

[Amazon Elastic Kubernetes Service](#) (EKS) is a managed Kubernetes offering from AWS. It is based on the key building blocks of AWS such as Amazon Elastic Compute Cloud (EC2), Amazon EBS, Amazon Virtual Private Cloud (VPC) and Identity Access Management (IAM). AWS also has an integrated container registry in the form of Amazon Elastic Container Registry (ECR), which provides secure, low latency access to container images.

Amazon EKS is tightly integrated with CloudWatch for monitoring and IAM for security. AWS App Mesh provides native service mesh capabilities to the workloads deployed in AWS. Through AWS Fargate, Amazon has exposed a serverless mechanism to run containers in EKS.

For accelerating machine learning training and inference, workloads can be scheduled on nodes that are attached to NVIDIA graphics processing units (GPUs).

AWS Outposts, Amazon's hybrid cloud platform, runs EKS on-premises.

Azure Kubernetes Service

[Azure Kubernetes Service](#) (AKS) is a managed container management platform available in Microsoft Azure. AKS is built on top of Azure VMs, Azure Storage, Virtual Networking and Azure Monitoring. Azure Container Registry (ACR) may be provisioned in the same resource group as the AKS cluster for private access to container images.

AKS takes advantage of virtual machine scale sets and Availability Sets to dynamically scale in and scale out the number of nodes in a cluster.

AKS is available on Azure Stack Hub, Microsoft's hybrid cloud offering for running Azure services in data centers.

Google Kubernetes Engine

[Google Kubernetes Engine](#) (GKE) was one of the first managed services to offer

Kubernetes in the public cloud. Like other public cloud-based CaaS, GKE takes advantage of Google Cloud Platform's core services, such as Compute Engine, Persistent Disks, VPC and Stackdriver.

Google has made Kubernetes available in on-premises environments and other public cloud platforms through the Anthos service. Anthos is a control plane that runs in GCP, but manages the life cycle of clusters launched in hybrid and multicloud environments.

Google extended GKE with a managed Istio service for service mesh capabilities. It also offers Cloud Run, a serverless platform — based on the Knative open source project — to run containers without launching clusters.

IBM Cloud Kubernetes Service

[IBM Cloud Kubernetes Service](#) (IKS) is a managed offering to create Kubernetes clusters of compute hosts to deploy and manage containerized applications on IBM Cloud. As a certified provider, IKS provides intelligent scheduling; self-healing; horizontal scaling; service discovery and load balancing; automated rollouts and rollbacks; and secret and configuration management for modern applications.

IBM is one of the few cloud providers to offer a managed Kubernetes service on bare metal.

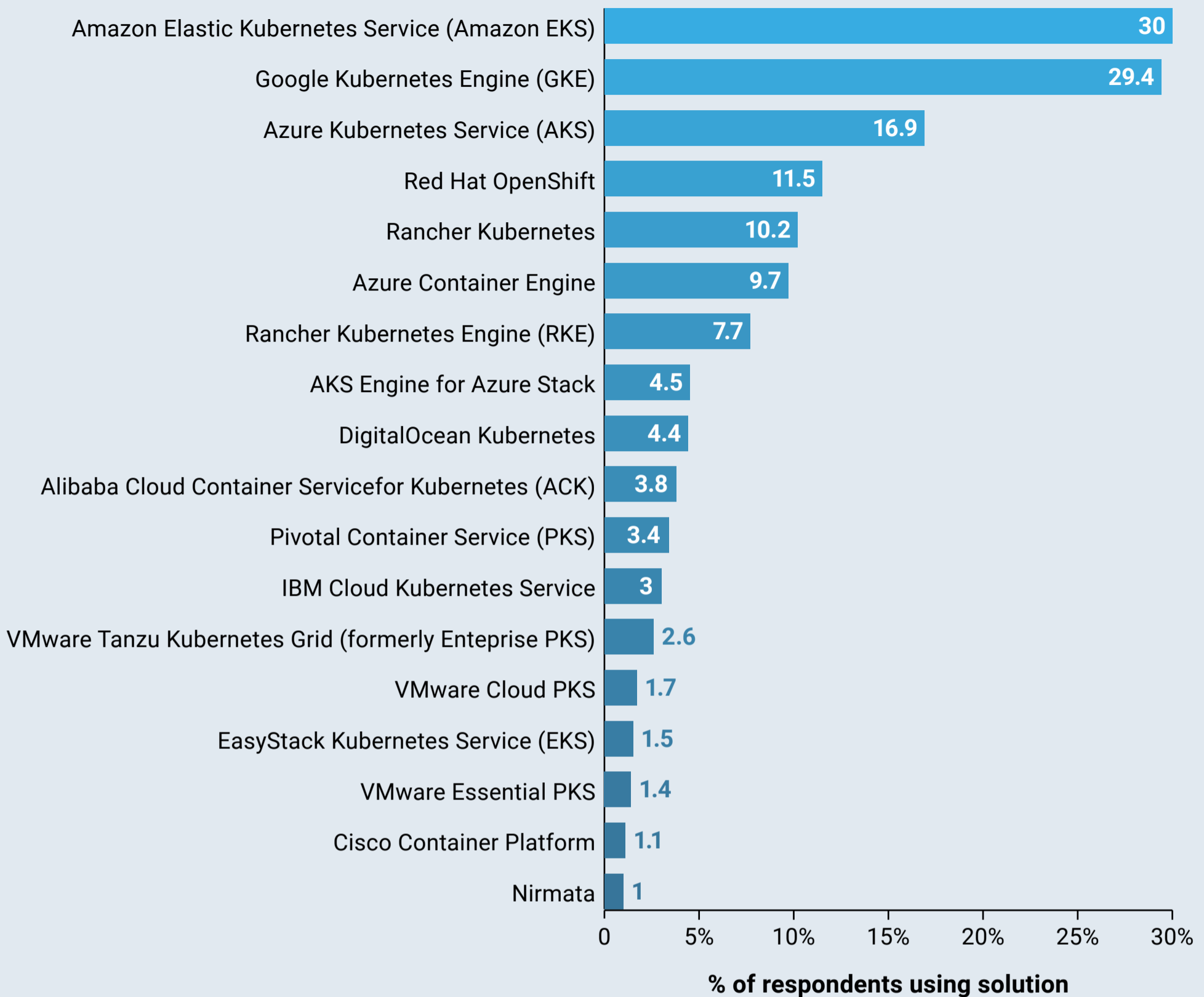
Via its acquisition of Red Hat, IBM offers a choice of community Kubernetes or OpenShift clusters available through IKS.

IBM also built Cloud Paks, a set of hyperconverged appliances based on Kubernetes and OpenShift, that can run in enterprise data centers.

Other Managed Kubernetes Services

Apart from the services listed above, customers can choose managed Kubernetes from [Alibaba Cloud Container Service](#), [DigitalOcean Kubernetes](#), [Huawei Cloud Container Engine](#) and [Platform9](#).

Amazon's EKS Edges Out Google's GKE Among Container Management Offerings From Cloud Providers



Source: The New Stack's analysis of CNCF's 2019 survey. Q. Your company/organization manages containers with: Please select all that apply. n=1197. The chart only shows offerings from cloud providers that are used by at least 1% of respondents with at least one Kubernetes cluster.

© 2020 THE NEW STACK

FIG 2.7: The names of many offerings have changed since the question was originally asked. Combined with both the Pivotal and VMware offerings, Tanzu may see an improved position when 2020's survey results are released.

Uptake for Cloud Provider Kubernetes as a Service

When the [CNCF survey was conducted in 2017](#), 25% of Kubernetes users surveyed ran containers with Google Kubernetes Engine (GKE). AWS was still touting its Elastic Container Service (ECS) and Microsoft's Azure Container Service had just

been launched. But even then it was clear Kubernetes would become the container orchestrator of choice. Despite having only a nascent Kubernetes offering, 52% of Kubernetes users ran it in an AWS environment, 34% on Google Cloud Platform (GCP), with Azure and VMware 21% and 15% respectively. Half of all Kubernetes workloads were running on AWS, even if they were being run on top of VMs and being managed by AWS customers.

At that year's AWS re:Invent, Amazon Elastic Container Service for Kubernetes (EKS) was announced — [with general availability coming in mid-2018](#) and AWS became a CNCF member soon after. Kubernetes had won the container orchestration war, but companies tended to run containers where they ran other workloads. For this reason, Amazon EKS and Azure Kubernetes Service (AKS) saw significant adoption in 2019.

The [latest CNCF survey](#) in 2019 included over 100 choices in its question about how containers are managed, with Amazon EKS (30%) used a bit more often than GKE (29%) among Kubernetes users. In addition to Microsoft's offering, both Red Hat OpenShift and Rancher Kubernetes also had sizable adoption figures.

Kubernetes Has Evolved, So Should Your Security



Kubernetes has many built-in security features, but that doesn't mean it's secure right out of the box. Security for dependency management is still lacking and new attack vectors, such as malicious containers, are emerging.

Despite advances in security, the API remains

Kubernetes' main entry point for attackers.

The good news is that security teams have learned a lot about how to protect Kubernetes deployments and applications running on containers over the past few years. Such threats can be more easily addressed through a combination of workflows and tooling that span developers, security teams and IT operations (DevSecOps). For example, malicious containers and other attack vectors are easy to spot through anomaly detection and scanning tools.

Robert Haynes, cloud security evangelist at Prisma Cloud by Palo Alto Networks, discusses Kubernetes security above and beyond the native features. He also talks about the evolution of the Kubernetes vulnerability landscape since the first API attacks took place a few years ago. The first critical attacks “taught us something we already knew ... that Kubernetes is software and all software has vulnerabilities,” Haynes said.

[Listen on SoundCloud](#)



Robert Haynes' career has spanned from a humble helpdesk operator to the lofty heights of a Unix sysadmin and then back again to marketing. He's currently with the Palo Alto Networks Prisma Cloud team, where he's thoroughly enjoying telling technical tales about security, technology and people.

Kelsey Hightower on His Very Personal Kubernetes Journey



In this podcast, we talk with a prominent figure in the Kubernetes ecosystem, Kelsey Hightower, principal developer advocate at Google Cloud and a former member of the Cloud Native Computing Foundation's (CNCF) Technical Oversight Committee which governs all CNCF projects, including Kubernetes. Hightower speaks about his role in Kubernetes and the challenges that lie ahead.

“I remember going through the codebase and there wasn't really a good review on how to even install it,” Hightower said about his first encounter with Kubernetes in 2015. “The first thing that I did was put together what some would consider to be one of the very first installation guides.”

Nowadays, Hightower wants the learning curve to decrease as the platform continues to mature. “Developers should not have to become Kubernetes network experts to do their work as software engineers,” he said.

[Listen on SoundCloud](#)



[Kelsey Hightower](#) is a principal developer advocate at Google Cloud. He helped develop and refine many Google Cloud products, including Google's Kubernetes Engine, Google Cloud Functions, and Apigee's API Gateway.

Closing

It's been just over five years since Kubernetes was introduced and it's now pretty much built. The concepts of pods, nodes and clusters are a part of the language that describe scale-out architectures. It's no longer about the marvels of the plumbing, now it's more about what is built on Kubernetes.

Kubernetes is getting boring, simply because in 2020 it's resilient and scalable. The plumbing just works.

It's easy to forget that Kubernetes' plumbing was anything but boring a few years ago. But since the publishing of The New Stack's first ebook about Kubernetes in 2017, a continual release schedule for the container orchestrator has established an iterative process for solidifying the underlying architecture for scale-out, distributed applications.

The challenge now is how to build on top of Kubernetes. In the world of scale-out architectures, it is code that gets distributed in containers. The code is the component in the container. The container is a process that runs on top of an operating system. As a process, the container is portable. It is designed to be immutable, meaning the containers can be replaced with new containers to update the code for the application component. Immutable infrastructure is based upon the concept that an application can be deployed to a consistent environment and thus is continually updated by seamlessly replacing containers that have the updated code.

The challenge with building on top of Kubernetes comes with the nature of containers as immutable processes. A container is stateless. To make Kubernetes useful to enterprises, engineers have had to develop interfaces to connect stateless containers through interfaces.

The Container Network Interface (CNI) connects containers and associated plugins

with the network that the organization is using to run an application. In this environment, the value is in the wide support that allows for plugins to connect into networks. The companies developing a common CNI are a pointer to why the overall Kubernetes community is so successful. It's because the spirit of cooperation in open community development is on display in Kubernetes projects. Kubernetes' plumbing may be boring now, but the technologists developing the underlying architecture are an energetic and passionate group, building out an architecture for one of the first operating systems for software and services on sophisticated distributed infrastructure.

But Kubernetes is still relatively new. Client-based operating systems are built for personal computers or servers, primarily for monolithic applications. Linux dominates the server arena. Containers are Linux-based — an extension of the Linux kernel. Kubernetes makes the container a base process for orchestrating code that runs as components. It's a control plane independent of the cloud or client architectures. It allows APIs to manage the control plane and the correlating compute, network and storage.

The youthful state of Kubernetes is reflected in the developer experience. The Kubernetes architecture is still not exactly developer ready. The Platform as a Service (PaaS) age is passing by; it's far less relevant than three years ago. PaaS is a way to run applications on cloud services and enterprise environments. A PaaS is built and assembled as an application structure. But even the best of them can't handle all the corner cases, primarily older monolithic applications.

Containers are changing the game. Code can run in a container and be managed far more easily, in far less time. But there are still few ways for developers to build services on Kubernetes. Enter Containers as a Service (CaaS), also known as container management platforms, that integrate various cloud native services to run both on monolithic enterprises and modern microservices architectures. A management platform runs multiple services on top of Kubernetes. It starts with the CPU and the physical infrastructure. The management platform includes the

container optimized operating system, container runtime, container-native networking, storage, security, the registry, observability, the container orchestration engine itself and more.

CaaS platforms will take some time to mature. Managed services will become more predominant, taking the burden off developers.

Until then, the roles people play — the personas so to speak — will evolve for developers, the people responsible for configuring the services, and those who define the policies for Kubernetes. The container-based universe has arrived with a new view of what an operating system means when running on infrastructure, no matter where it may be running.

Kubernetes may not be perfectly boring, after all. But it's getting there. As Kelsey Hightower likes to say, the next step is to make it disappear.

Disclosure

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

Accurics, AppDynamics, Aspen Mesh, Amazon Web Services, Bridgecrew, CircleCI, Citrix, CloudBees, Cloud Foundry Foundation, Cockroach Labs, Dell Technologies, Diamanti, Futurewei, GitLab, Gravitational, HAProxy, HashiCorp, Honeycomb, InfluxData, Lightbend, Lightstep, LogDNA, MayaData, MongoDB, New Relic, NS1, Packet, PagerDuty, Portworx, Red Hat, Redis Labs, Rezilion, SaltStack, SAP, Sentry, Snyk, Sonatype, The Linux Foundation, TriggerMesh and VMware.

