KASTEN BY VEEAM PRESENTS

THE GORILLA GUIDE TO...®

Getting Started with Kubernetes, Foundation Edition

Dan Sullivan

INSIDE THE GUIDE:

- Kubernetes and Cloud-Native Transformation
- Kubernetes Deployment Options and Top Use Cases
- Kubernetes and Multi-Cloud Management

HELPING YOU NAVIGATE THE TECHNOLOGY JUNGLE!

ActualTech MEDIA

In Partnership With

kasten by Veeam

Getting Started with Kubernetes

By Dan Sullivan

TABLE OF CONTENTS

Please register at Learning.kasten.io to get hands-on experience with Kubernetes.

ACTUALTECH MEDIA

6650 Rivers Ave Ste 105 #22489 | North Charleston, SC 29406-4829

www.actualtechmedia.com

Publisher's Acknowledgements

EDITORIAL DIRECTOR
Keith Ward

DIRECTOR OF CONTENT DELIVERY
Wendy Hernandez

CREATIVE DIRECTOR
Olivia Thomson

SENIOR DIRECTOR OF CONTENT
Katie Mohr

PARTNER AND VP OF CONTENT
James Green

WITH SPECIAL CONTRIBUTIONS FROM KASTEN BY VEEAM®
Thomas Keenan
Joey Lei
Tom Leyden
Lori Maupas

ABOUT THE AUTHOR

Dan Sullivan is a principal engineer and architect focused on cloud architecture, data science, machine learning, and data architecture. He's the author of six books, as well as several online courses, in addition to Google Cloud's official certification study guides out from Sybex. Follow Dan on LinkedIn at linkedin.com/in/dansullivanpdx/.

Introduction

Welcome to this Gorilla Guide® To... Getting Started with Kubernetes, Foundation Edition! If you're unsure of exactly what Kubernetes is, despite all the momentum it's gained over the last few years, you're not alone. Kubernetes adoption has grown rapidly. A report by the Cloud Native Computing Foundation (CNCF) found that more than 90% of organizations surveyed were using Kubernetes in 2020—an increase of 33% over the previous two years.

However, for many, Kubernetes is still somewhat of a mystery. While most people know Kubernetes has something to do with containers and is related to cloud-native development, that's the extent of their knowledge. That's why this book exists.

Software development is a constantly evolving practice, and like other areas of technology, it's changing at an accelerating pace.

By the time you finish reading this Guide, you'll have a strong base of information about Kubernetes, and it won't seem mysterious at all!

We'll cover a high-level summary of Kubernetes and why it's become an essential technology for companies that want to

achieve fast, efficient development that takes full advantage of what the cloud offers. This short book covers the fundamentals of the architecture, deployment, lifecycle management, and more—in short, it's a roadmap to get you started with Kubernetes, and get the most out of it.

So, if you're ready, let's get started. We'll begin with a brief overview of how software development has changed over the years, and why the old ways of development could be holding you back. For more introductory details about Kubernetes, see https://kubernetes.io/docs/home/ and https://kubernetes.io/docs/concepts/.

The Changing Development Landscape

Software development is a constantly evolving practice, and like other areas of technology, it's changing at an accelerating pace. Methodologies and frameworks that worked well enough in the past are no longer sufficient to support modern software development. Developers now depend on Agile development methodologies that support the continuous release of new features, use hyper-scalable cloud resources to adapt to the needs of our workloads, and employ new models for managing and utilizing those resources efficiently. Let's look at each of these key differentiators in the modern software development landscape.

To enable the frequent release of innovative features, software developers have shifted away from large, infrequent software updates in favor of small, incremental updates that are released frequently, sometimes multiple times per day. The microservices architecture creates applications as ensembles of many small, semi-independent services that together implement the functionality of an application. Each of these can be updated independently. This architecture is a key enabler of continuous integration (CI) and the continuous delivery and deployment (CD) of new features to production environments.

The days when we would deploy a single physical server for a single application are over.

Cloud technologies enable greater scalability, reliability, and availability than we can provide using limited-resource, on-premises data centers. The cloud is designed to virtualize most of the infrastructure we use to deliver services, including compute, storage, and networking functionality. The way we use infrastructure in the cloud is fundamentally different from how we provision, deploy, and manage physical infrastructure on-premises. To take advantage of the cloud, we need to design cloud-native applications that are designed with cloud infrastructure requirements in mind.

The days when we would deploy a single physical server for a single application are over. That just wasn't efficient, especially if you had to size the server for the peak workload regardless of how often those peaks occurred. Virtual machines that run multiple, isolated operating systems on a single server improved efficiency. They depend on a layer of software called a hypervisor to isolate the different guest operating systems. A more efficient approach is to use containers, which build on advances in operating system capabilities for isolating processes. Containers can sufficiently isolate workloads within an operating system so that a hypervisor isn't needed. They form a key cornerstone of the cloud-native world. For more on how Kubernetes containers take on modern workloads, see https://kubernetes.io/docs/concepts/workloads/.

The Importance of Virtual Machines

The need to maximize efficiency in the way we utilize compute resources drove the development of containers, but virtual machines are still needed. With VMs, there is a strong isolation barrier provided by the hypervisor. Also, migrating an application from a physical server to a virtual machine is straightforward while migrating to a container (aka "containerizing") an application is a more involved process. Virtual machines may be a better choice in some cases because of compliance and security requirements, as well.

Kubernetes and Cloud-Native Transformation

As the popularity of containers increased, software and reliability engineers saw the need for tools and platforms to help coordinate and manage container operations. This led to the development of Kubernetes, an orchestration platform that can manage the full lifecycle of containers, from deploying containers to performing health checks, replacing failed containers, and efficiently allocating compute resources across a range of workloads. Kubernetes has become a key enterprise IT initiative and is definitely here to stay, so it's of paramount importance to understand how it works, particularly because the lack of Kubernetes training has been cited as one of the major challenges in the community.

Today's digital infrastructure is a combination of virtual machines and containers running on-premises and in the cloud. Containers are increasingly used because they provide significant advantages over virtual machines.

Containers provide a common level of abstraction that allows you to run applications and workloads in different environments with minimal, if any, changes to the container specification. For example, you can develop a service on a local machine and deploy it locally using a container. That container can then be deployed to a test environment running on-premises and eventually promoted to a production environment running in one or more clouds.

Software development pipelines, deployment services, and containers are a natural fit. Containers enable the automation of the build and deployment process, which reduces the risk of human error when manually performing those operations.

Another driver for the adoption of containers is that they provide an excellent mechanism for implementing modern software architecture. Containers are well suited to support microservices. Each microservice can be deployed in its own container. Also, services can be scaled by creating more instances of containers running the service or by reducing the number of containers depending on the workload.

Kubernetes has become a key enterprise initiative in many organizations because it's the foundation of how businesses can drive digital transformations.

There are disadvantages with containers, of course, and those stem from the same characteristics that bring its advantages. As containers typically run a single microservice, many containers are needed to support a service. Each of those containers must be monitored, scaled, and replaced if they're no longer functioning. This can create a significant burden, unless you have sufficient tooling in place to help manage the full lifecycle of large numbers of containers.

Kubernetes optimizes the use of infrastructure. This is crucial to realizing the efficiency benefits of containers at scale.

Running thousands of containers with manual processes or ad hoc automation would be far less efficient and error prone. For further information about containers, visit https://kubernetes.io/docs/concepts/containers/.

The benefits of Kubernetes are having an impact on organizations and how they operate. Kubernetes has become a key enterprise initiative in many organizations because it's the foundation of how businesses can drive digital transformations.

Kubernetes Architecture: The Core Concepts

Now it's time to turn our attention to some details of the Kubernetes architecture (check out https://kubernetes.io/docs/home/, a great resource for more in-depth information). With an understanding of basic Kubernetes concepts and design principles, we'll be able to see why Kubernetes orchestration is fundamentally different from the ways we've managed workloads and resources in the past.

This section will consider three core aspects of Kubernetes: infrastructure building blocks, the control plane, and workload building blocks.

KEY TERMS IN THE KUBERNETES ARCHITECTURE

See lab 1 of the Learning.kasten.io series for a good review of Kubernetes terminology.

- **Control Plane:** The backbone of Kubernetes
- **K8s API/Desired State:** High-level overview
- **Pod:** The smallest deployable object in the Kubernetes object model
- **Replica Set:** Manages the number of running Pod replicas
- **Deployment:** Manages Pods and ReplicaSets
- **Services:** Abstract a set of Pods
- **Namespaces:** Divide your cluster
- **Volume:** A directory which is accessible to Pods
- **Job:** Creates one or more Pods and retries execution of the Pods until a specified number of them successfully terminate
- **DaemonSet:** Runs a Pod on all (or some) Nodes
- **StatefulSet:** Used to manage stateful applications

Check out https://kubernetes.io/docs/concepts/ for expanded explanations of key concepts.

INFRASTRUCTURE BUILDING BLOCKS

Kubernetes is deployed to a cluster, which is a set of virtual or physical servers that are managed by the Kubernetes platform (see **Figure 1**).

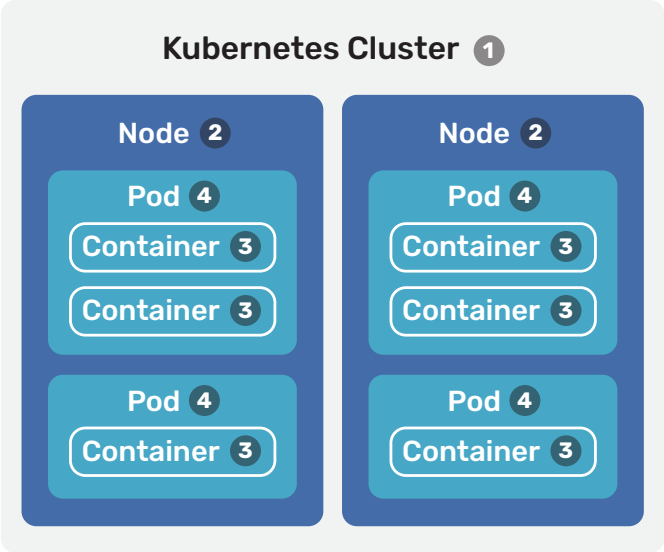


Figure 1: Kubernetes infrastructure components

We can think of clusters (1) as a pool of resources that are managed by Kubernetes. Typically, organizations run small numbers of clusters, preferring to increase the size of clusters as needed rather than adding more clusters to increase capacity. This is one of the ways that Kubernetes can efficiently use resources, as a single cluster can manage many different workloads with varying characteristics.

Within a cluster, nodes (2) are an abstraction of a server. Containers (3) run on nodes but just as Kubernetes has layers of abstraction about a server, it has a layer of abstraction around containers, too. That abstraction is the pod. Pods (4) are the smallest deployable compute object that Kubernetes manages. Pods typically house one container, or a primary container and ancillary containers to help with services

such as authentication and monitoring. The key thing to remember about multiple containers in a single pod is that the containers are tightly coupled and have the same lifecycle.

We can think of clusters as a pool of resources that are managed by Kubernetes.

Another key infrastructure building block is how we specify the characteristics of a cluster and services. Kubernetes uses a declarative model to describe the desired state of the cluster and services running in the cluster. This means we can specify that we would like to have a service deployed with at least four pods and not more than 10, and we'd like that number to scale according to CPU utilization. If CPU utilization surpasses a threshold, Kubernetes adds another node. Similarly, if CPU utilization drops, nodes are removed. Notice that we don't have to specify how Kubernetes should maintain the desired state; we only specify what state we want. Kubernetes handles the implementation details for us.

CONTROL PLANE

The control plane is the Kubernetes decision maker. It's a set of services that use policies we define to keep the cluster and services in the desired state. The control plane has four major components: the API server, etcd, a scheduler, and a controller manager (see **Figure 2**).

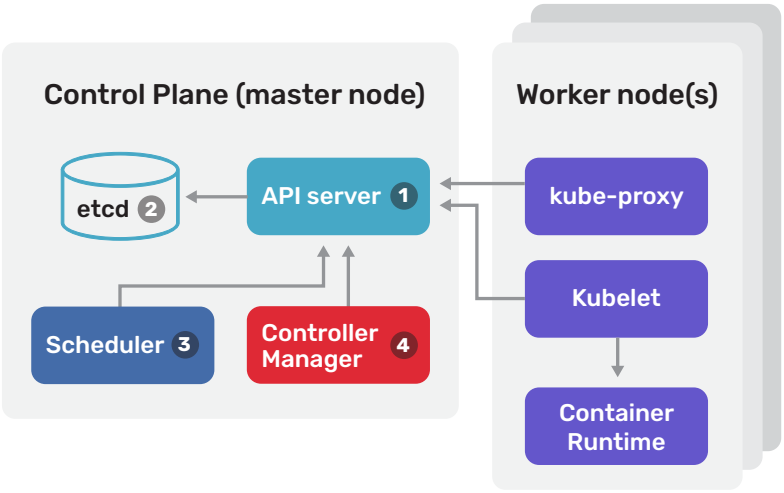


Figure 2: The Kubernetes control plane manages resources in the Kubernetes cluster

1. The API server provides access to the Kubernetes API, which is how we work with Kubernetes. It's essentially the interface to the control plane. The API service runs as one or more instances of the kube-apiserver service.

2. Etcd is a high-availability key-value store which is used by Kubernetes to store data about the cluster and its state.

3. The scheduler is the control plane service that assigns pods to nodes. This is particularly important because the efficiency and security of a service can depend on how a pod is assigned to a node. The scheduler considers multiple attributes of pods and nodes when making an assignment, including the resources required for a pod, the resources available on a node, and any policy constraints, such as restrictions that limit a pod to run on only nodes with certain characteristics.

4. The controller manager runs a set of controller processes. Controllers manage Kubernetes objects, such as nodes, jobs, endpoints, and service accounts. In a cloud, the cloud controller manager is used to implement cloud-specific functionality such as managing nodes, implementing some networking, and utilizing cloud-specific load balancers.

The control plane communicatees with the worker nodes to coordinate operations in the cluster. The control plane can communicate through the API server to the kubelet running on each node, but it can also communicate with any node, pod, or service.

WORKLOAD BUILDING BLOCKS

Workloads are applications running in Kubernetes. When we run workloads in Kubernetes we use an abstraction known as a deployment. This allows for the abstraction of the application away from the set of pods on which it runs and is a crucial feature of the Kubernetes architecture. Pods are ephemeral and may be shut down at any time. For example, if a pod fails a health check, the default behavior is to terminate that pod and replace it with another. Pods have IP addresses and replacement pods will have different IP addresses than the prior pods. Deployments are collections of pods governed by a policy that provide a long-lived abstraction for running a workload. Services are another Kubernetes abstraction, and they're responsible for providing stable IP addresses and network access to a set of pods.

Services often require multiple pods (3) (see **Figure 3**). ReplicaSets (2) are used to specify a set of identical, or replica,

Pods running in a cluster. ReplicaSets are described in policies and use the desired state model. For example, a ReplicaSet policy will specify how many Pods are in the ReplicaSet and the container image to run in the Pods. ReplicaSets work well when services are stateless, but when running applications, such as a database, you'll need to use StatefulSets. StatefulSets can maintain an identity across pods when one needs to be replaced by another.

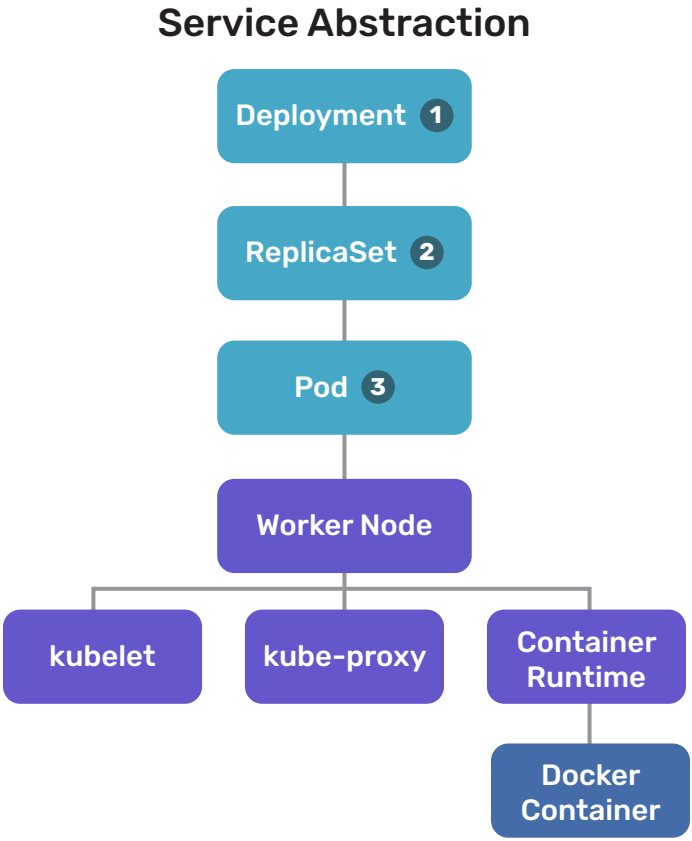


Figure 3: Workload abstractions enable a more scalable, reliable, and flexible environment for delivering services

Services are frequently updated with new features. In Kubernetes, there is the deployment (1) abstraction that's used to update pods and ReplicaSets. Deployments are used for rolling out ReplicaSets, changing the desired state of a Pod, or rolling back to a stable state.

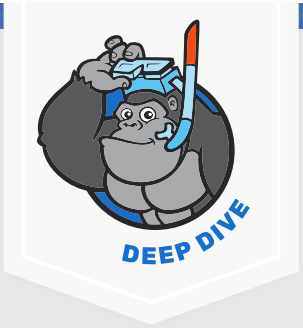
When we run workloads in Kubernetes we use an abstraction known as a deployment. This allows for the abstraction of the application away from the set of pods on which it runs and is a crucial feature of the Kubernetes architecture.

Sometimes it's desirable to segment resources within a cluster. For example, some nodes may be configured with GPUs that should be reserved for building machine learning models. In that case we can use namespaces, which is a Kubernetes mechanism for defining groups of resources within a cloud. When declaring services and pods, we can also specify what pool of resources, or namespace, they should be deployed to. This is particularly useful when a single cluster supports multiple teams and projects with varying priorities and resource requirements.

The Service abstraction works well for long-running services such as an API server, but some workloads just need to run once to completion. A Job is an abstraction for running an application such as a task, and once that task is complete, the pods are shut down.

When running a service that should have only one instance per node, you can use DaemonSets. These are particularly useful when you need to run a logging or monitoring agent on a node to collect data and forward it to a centralized log or observability platform.

With regard to storage, Kubernetes uses Volumes and VolumeClaims. Volumes are an abstraction used to enable persistent storage for pods when they crash and for sharing storage across pods. PersistentVolumes are an amount of storage provisioned by Kubernetes. A PersistentVolumeClaim is a request for storage and allows a user to access a Persistent Volume. Volumes can be configured to accommodate different requirements such as size and performance levels.



Namespaces, Affinity, and Taints: Allocating Resources in a Cluster

As mentioned, namespaces are subsets of resources in a cluster. Kubernetes has several different ways of assigning those resources. Node affinity is the property of Pods that attracts them to particular nodes. Taints are properties of nodes that can block the placement of a pod on a node. Tolerations are pod specifications that allow pods to schedule on a set of nodes with matching taints. We use taints and tolerations to ensure pods are scheduled on appropriate nodes. See https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/ for more on these terms.

Kubernetes Deployment Options

Kubernetes runs on-premises and in the cloud and you can even run a mini-version called minikube on your laptop.

If you're deploying on-premises from scratch, you'll need a set of servers to constitute the cluster infrastructure. Once you have that you can use kubeadm, a tool for creating minimum viable clusters that will pass Kubernetes conformance tests.

Cloud providers have managed Kubernetes services that make deployment even easier. For example, Google Cloud provides Google Kubernetes Engine (GKE), Amazon Web Services (AWS) has the managed service Amazon EKS, and Microsoft Azure offers Azure Kubernetes Service (AKS).

It's becoming more common to run Kubernetes in multiple environments. You may have some development workloads running in a small on-premises development environment that are tested in a small Kubernetes cluster created in your primary cloud provider, and then deployed to a production environment in a multi-region cluster running in the cloud.

In practice, it's also common to run multiple workloads and services across multiple clusters. This can lead to a range of management challenges (learn more about workloads at https://kubernetes.io/docs/concepts/workloads/).

Consider networking, for example. You'll have to configure pools of IP addresses for nodes, as well as for pods. Pod-to-pod communication can be done using internal addressing while connections to services outside the cluster will require some way to use an external IP address to expose those services outside the cluster.

There are also lifecycle management considerations when working with Kubernetes. As services are updated, you'll release new deployments. A common practice is to route a small amount of traffic to a new deployment to test for any problems before releasing the new versions to all users (so-called "canary deployment"). Kubernetes makes it easy to configure deployments and specify how traffic should be routed to those deployments.

There will be times when pods fail to function as expected. Fortunately, Kubernetes has built-in health check and auto-healing features. This reduces the burden on DevOps engineers who can instead focus on more challenging issues that can't be resolved by something as straightforward as terminating a pod and creating another.

Top Use Cases for Kubernetes

Kubernetes is widely adopted because it addresses some common and otherwise time-consuming challenges of creating and maintaining cloud-native applications. Kubernetes is especially well suited to a handful of services including stateless and stateful applications and CI/CD pipelines.

Kubernetes is often used to run stateless applications, such as Web servers or API-based services. Stateless applications don't keep track of information between requests, so there's no need to ensure a client or user always makes requests to the same pod. Stateless applications are easy to scale horizontally because requests can be routed easily to newly added pods without worrying about maintaining state across requests.

It's becoming more common to run Kubernetes in multiple environments.

Stateful applications are more challenging. Sometimes there's the need to maintain state information. For example, if you're running a database that caches query results, you would want to be able to access that cache across a series of database queries. You would also want the database to persistently store data and have access to that data, even after a pod has been replaced. Kubernetes manages state using a combination of StatefulSets and PersistentVolumes.

Another type of application common in Kubernetes clusters is CI/CD pipelines. These pipelines automate the process of building images, running tests on applications, running security scans to check for vulnerabilities before releasing a new image, and of course rolling back deployments that don't function as expected.

Kubernetes and Multi-Cloud Management

No cloud or on-premises data center is a digital island. Platforms and environments are connected, and this leads to hybrid cloud and multi-cloud infrastructures. Fortunately, Kubernetes can radically improve the manageability of these increasingly complex environments.

Consider some of the challenges with multi-cloud infrastructures (see **Figure 4**). One that becomes apparent quickly to most cloud engineers is the need to co-locate compute and storage resources. It's great to have access to a large cluster of servers with GPUs to train your machine learning models, but

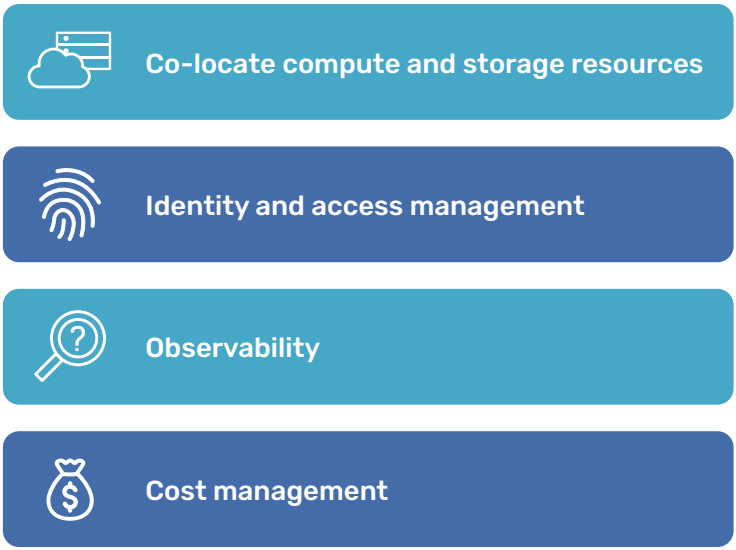


Figure 4: Challenges of multi-cloud infrastructure

if your training data is in another cloud, you could be facing some significant egress charges to copy that data to the cloud where your compute resources are located.

Another challenge is identity and access management. Ideally, your organization has a single identity provider that works with your authentication and authorization mechanisms. It can be difficult, if not impossible, to keep multiple access controls and identity systems synchronized with up-to-date policies without some kind of automation to support federation across the systems.

Cost management is another challenge, even when working with a single cloud. And if workloads, and therefore costs, are spread over multiple clouds, problems are compounded.

Although you may run workloads on-premises and in the cloud, your DevOps engineers and systems administrators would be better off with observability tools that allow them to see monitoring and logging data and alerts from a centralized tool. Trying to diagnose incidents in real time is difficult enough without having to try to collect data from multiple monitoring, logging, and tracing services.

Cost management is another challenge, even when working with a single cloud. And if workloads, and therefore costs, are spread over multiple clouds, problems are compounded.

Fortunately, Kubernetes supports multi-cloud infrastructures with tools to help localize compute and storage. It also provides the foundation for centralized policy management and consolidated logging and monitoring. Because it's a unified platform, it's also easier to predict and track costs.

Kubernetes has much to offer and now's the time to start learning it. Register at Learning.kasten.io to get hands-on experience with Kubernetes, beginning with lab 1, which guides you through deploying your first Kubernetes cluster.

Out with the Old, in with the New

At the beginning of this book, we promised you a quick education on Kubernetes—a way to get up to speed on what it is, how it works, and how it can transform your software development operations.

If you've gotten this far, that's exactly what you now have. Kubernetes isn't a simple technology, for sure, but it's getting easier to manage, especially with the available help from third-party vendors who really understand it.

One thing is certain—Kubernetes isn't going anywhere. It will continue to grow in importance as companies ditch the traditional methods of slow, monolithic software development for the cloud-native DevOps model that emphasizes speed and efficiency, leveraging the new technology for all it's worth.

There is no better way to learn Kubernetes than by trying it hands-on. The Learning.kasten.io learning series provides just such an opportunity. It offers instructor-led labs that allow you to learn Kubernetes at your own pace, and earn badges and other awards. Join the learning community today and get your questions answered by experts. Go to Learning.kasten.io and register to begin your journey!

We hope you found this e-book valuable, and that it helps you do more for your business!

About Kasten by Veeam

Kasten by Veeam® is the leader in Kubernetes backup. Kasten K10 is a Cloud Native data management platform for Day 2 operations. It provides enterprise DevOps teams with backup/restore, disaster recovery and application mobility for Kubernetes applications. Kasten K10 features operational simplicity and integrates with relational and NoSQL databases, all major Kubernetes distributions, and runs in any cloud to maximize freedom of choice. Our customers are confident that their Kubernetes applications and data are protected and always available with the most easy-to-use, reliable and powerful Cloud Native data management platform in the industry. For more information, visit www.kasten.io or follow @kastenhq on Twitter.

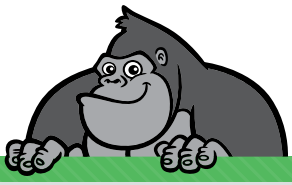
About ActualTech Media

ActualTech MEDIA

ActualTech Media is a B2B tech marketing company that connects enterprise IT vendors with IT buyers through innovative lead generation programs and compelling custom content services.

ActualTech Media's team speaks to the enterprise IT audience because we've been the enterprise IT audience.

Our leadership team is stacked with former CIOs, IT managers, architects, subject matter experts and marketing professionals that help our clients spend less time explaining what their technology does and more time creating strategies that drive results.



If you're an IT marketer and you'd like your own custom Gorilla Guide® title for your company, please visit https://www.gorilla.guide/custom-solutions/