LEARNING MADE EASY

VMware 2nd Special Edition

Kubernetes® on vSphere®

for dummies®
A Wiley Brand

Run Kubernetes on existing infrastructure

Integrate with vSphere components

Speed up developer velocity

Brought to you by

vmware®

Boskey Savla
Steve Hoenisch

About VMware

VMware software powers the world's complex digital infrastructure. The company's cloud, networking and security, and digital workspace offerings provide a dynamic and efficient digital foundation to over 500,000 customers globally, aided by an ecosystem of 75,000 partners. Headquartered in Palo Alto, California, VMware is committed to being a force for good, from its breakthrough innovations to its global impact. For more information, please visit **www.vmware.com/company.html.**

Kubernetes on vSphere

VMware 2nd Special Edition

by Boskey Savla and Steve Hoenisch

for dummies®
A Wiley Brand

Kubernetes on vSphere For Dummies®, VMware 2nd Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-85397-8 (pbk); ISBN 978-1-119-85374-9 (ebk)

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Elizabeth Kuball
Acquisitions Editor: Ashley Coffey
Editorial Manager: Rev Mengle
Business Development Representative: Cynthia Tweed
Production Editor: Mohammed Zafar Ali
Special Help: Eva Leong, Lawrence Miller

Introduction

In this always-connected mobile era, customers and consumers expect to engage with businesses anywhere and everywhere. Enterprises are looking to meet their customers' needs and drive business forward by

- Creating new applications that engage customers in innovative and captivating ways
- Improving operations to more efficiently deliver better products and services at a lower cost to the business
- Generating new revenue streams by rapidly adapting to changes in market conditions and consumer preferences

Containers provide the basis for a new cloud-native application architecture that lays the foundation for innovation. A container is a process that runs with its own isolated, self-described application, file system, and networking. A container packages an application in a reproducible way: It can be distributed and reused with minimal effort.

Kubernetes is an open-source orchestration system for containers that can work in your data center, across clouds, and in a hybrid data center. Kubernetes automatically places workloads, restarts applications, and adds resources to meet demand. Kubernetes helps enterprises successfully execute their business initiatives by

- Making it easier to run applications in public, private, or hybrid clouds
- Accelerating application development and deployment
- Increasing agility, flexibility, and the ability to adapt to change

In this book, you discover how building and running containerized applications with Kubernetes on vSphere drives business value by enhancing developer productivity, business agility, IT flexibility, and application scalability.

About This Book

Kubernetes on vSphere For Dummies consists of seven chapters that explore

- The evolution of virtualization technologies, the business need for speed and agility, and the basics of Kubernetes (Chapter 1)
- Similarities and differences between vSphere and Kubernetes (Chapter 2)
- How to map a Kubernetes cluster on vSphere (Chapter 3)
- Application deployment and maintenance with and without containers and Kubernetes (Chapter 4)
- Automating and optimizing key Kubernetes operations (Chapter 5)
- How to secure your cloud-native applications (Chapter 6)
- Key benefits of running Kubernetes on vSphere (Chapter 7)

Each chapter is written to stand on its own, so if you see a particular topic that piques your interest, feel free to jump ahead to that chapter. You can read this book in any order that suits you (though we don't recommend upside down or backward).

Foolish Assumptions

It's been said that most assumptions have outlived their uselessness, but we assume a few things nonetheless!

Mainly, we assume that you're a VMware vSphere administrator responsible for setting up and maintaining virtual environments — including operating systems, network resources, servers, storage, desktops, data, and applications — as well as traditional system administration duties. And now you need to add containers to your list! With so many responsibilities, you need intelligent automation and orchestration tools, such as Kubernetes, to deploy, scale, manage, monitor, optimize, and secure your containerized applications.

As such, this book is written primarily for technical readers, but we promise not to get too technical and we'll be sure to explain any acronyms and "techie" stuff.

If any of these assumptions describes you, then this is the book for you! If none of these assumptions describes you, keep reading anyway! It's a great book, and when you finish reading it, you'll know quite a bit about managing Kubernetes on vSphere.

Icons Used in This Book

Throughout this book, we use special icons to call attention to important information. Here's what to expect:


REMEMBER

This icon points out important information you should commit to your nonvolatile memory, your gray matter, or your noggin — along with anniversaries and birthdays!


TECHNICAL STUFF

You won't find a map of the human genome here, but if you seek to attain the seventh level of NERD-vana, perk up! Anything marked with this icon explains the jargon beneath the jargon!


TIP

Tips are appreciated, never expected — and we sure hope you'll appreciate these useful nuggets of information.


WARNING

These alerts point out the stuff your mother warned you about. Well, probably not, but they do offer practical advice to help you avoid potentially costly or frustrating mistakes.

Beyond the Book

There's only so much we can cover in 64 short pages, so if you find yourself at the end of this book, thinking, "Gosh, this was an amazing book — where can I learn more?," just go to `https://tanzu.vmware.com/blog`.

IN THIS CHAPTER

- Enabling innovation in the SDDC
- Addressing modern business challenges with container technology
- Driving efficiencies in the SDDC with containers and Kubernetes

Chapter 1
Data Center Abstractions and the Software Delivery Life Cycle

This chapter explores the journey of data center optimizations, brought by compute virtualization to the software-defined data center (SDDC), as well as how containers and Kubernetes take this journey further to help accelerate business strategies and the deployment of applications in private, public, and hybrid clouds.

Looking at the Evolution of the Data Center

Throughout its nearly 60-year history, virtualization technology has enabled innovation and disrupted entire industries. From its humble beginnings as a means to allow time-sharing on expensive mainframes to its modern reincarnation allowing compute

resources in x86 servers to be abstracted from their physical hardware as virtual machines (VMs), the hypervisor has come a long way.

Enterprises worldwide have used virtualization technology to significantly improve IT efficiency and performance. Operational efficiencies enabled by virtualization include the following:

- New servers can be provisioned in a matter of minutes (compared to procuring, installing, and configuring new physical server hardware, which can take weeks or longer).
- Workloads from different physical servers can be virtualized and consolidated on a single physical host to maximize resource utilization and reduce capital expenditures.
- Application workloads can be dynamically migrated across multiple VMs and load-balanced to ensure adequate resources are always available during periods of peak demand.
- VMs can be quickly "spun up" to support short-term needs, such as developer sandboxes, proofs-of-concept, and testing platforms.
- VM images can be quickly backed up and replicated to separate physical locations to provide robust high availability, business continuity, and disaster recovery.

Compute utilization was optimized with virtualization, but provisioning networking and storage for workloads was still done the old-fashioned way — manually, on physical hardware — which is replete with all the same challenges and limitations of physical infrastructure: long procurement cycles; data center rack space, power, and cooling requirements; disk and port capacity limits; and underutilized resources. To meet these challenges, storage and network were virtualized. This paved the way for a data center that was completely software defined, including compute, networking, and storage, such that all infrastructure services became as easy as provisioning and managing VMs.

A VMware SDDC abstracts traditional infrastructure silos, offering a cohesive platform that can respond to the dynamic needs of the business and be extended to the cloud. A VMware SDDC extends virtualization across the entire infrastructure (compute, storage, and networking) through common hardware that is managed with

existing tools and skill sets. This abstraction accelerates deployments, improves their scalability, and unifies and eases operations, monitoring, and IT management.

Network virtualization in a VMware SDDC abstracts Layer 2 through Layer 7 networking functions — such as switching, firewalling, load balancing, and routing — on top of existing physical networks. By moving network and security services into the data center virtualization layer, network virtualization enables the creation and deletion of networks, routers, load balancers, and firewalls to cater to specific application needs with the same simplicity and speed as when VMs were set up. Software-defined storage in a VMware SDDC gives granular nondisruptive scale-up or scale-out storage to expand capacity and performance by adding hosts to a cluster (scale-out) or expand only capacity by adding disks to a host (scale-up).

A VMware SDDC establishes the ideal architecture for private, public, and hybrid clouds. A VMware SDDC extends familiar virtualization concepts — abstraction, pooling, and automation — to all data center resources and services, including network virtualization and software-defined storage (see Figure 1-1).

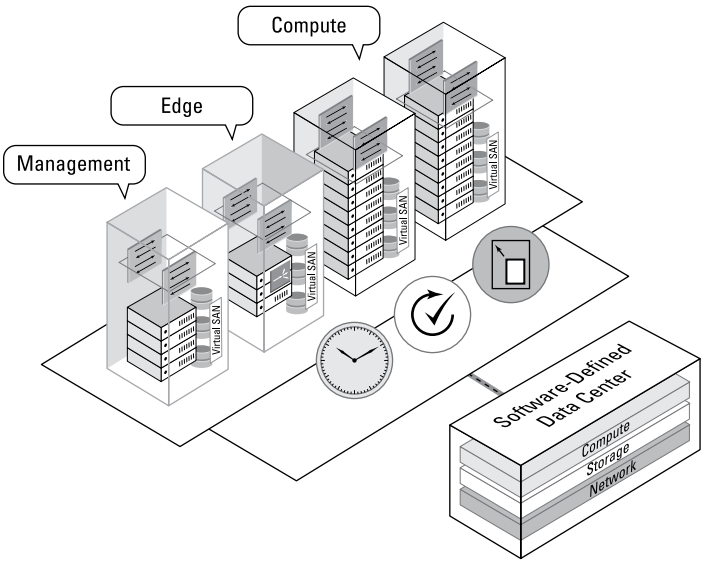

FIGURE 1-1: The software-defined data center.

The SDDC makes operations more efficient through:

- Automated, on-demand buildout of networks and datastores
- Faster turnaround times for infrastructure requests (no more waiting for handoffs between server, networking, and storage teams), reducing the software delivery cycle, in some cases from months to weeks
- Greater IT agility and efficiency with the flexibility to support a broad range of hardware and applications
- Increased infrastructure utilization and staff productivity, which reduces both capital expenditures and operating costs for the enterprise

The optimizations brought by the SDDC helped enterprises by decreasing the overall time to deliver and maintain application workloads. However, there are aspects of managing and scaling applications that lie beyond merely managing the infrastructure. With modern and always connected technology, application needs are changing. Today's always connected applications need coordination between an application's run-time state and the infrastructure. The infrastructure backing an application needs to move in lockstep with the run-time states of the application. The infrastructure, application end state, and configurations need to be looked at periodically and adjusted to ensure that business needs are met.

To drive business needs, it's no longer sufficient to define, for example, a load balancer that points to a pool of servers that all share the same static application data. The front-end application itself needs to call multiple subcomponents, and the load balancer needs to decide which subcomponent to route traffic to based on certain parameters. At any given time, an application will have multiple versions running with multiple components. In a similar way to how a VMware SDDC abstracts away infrastructure dependencies, containers abstract away dependencies at the level of the operating system. Formally, containers provide operating system virtualization. They enable enterprises to accelerate the development of innovative software and rapidly adapt to changes in the marketplace with microservices, distributed systems, orchestration tools, and other virtualization technologies.

Recognizing the Need for Speed

Fundamental changes taking place in the world are ushering in a new era of computing. The proliferation of mobile devices, the growth of cloud computing, the ubiquity of data, and the acceleration of life mean that the only change that's guaranteed is change itself. Paperless, time-pressed consumers are demanding innovative products and services, and companies are responding by striving to rapidly deploy software that engages their customers.

Many successful companies today are defined by the applications and experiences they've created through the innovative use of technology. Uber and Airbnb, for example, are first and foremost technology companies — the ride-sharing and rental aspects of their businesses are important, but secondary to the technology. In this "uber"-competitive business environment, it's no longer acceptable to have a software delivery cycle that takes weeks to move to production. Modern businesses and their corresponding technologies must constantly move at an accelerated pace.

To address the need for speed, containers along with cloud-native architectures enable an application to be quickly packaged in an image, complete with all its dependencies, and deployed to production — all without the need for IT operations and network teams to manually configure and deploy infrastructure, such as servers and load balancers. Easy-peasy!

TIP

The fast track to cost-effectively adopting cloud-native architecture is to transform your existing virtualized infrastructure into a flexible, scalable, modernized SDDC capable of deploying cloud-native applications, as well as continuing to host traditional applications. Enterprises are increasingly moving toward container-based architectures to develop applications faster, increase automation, and improve resource utilization. By adopting containers, organizations can take a vital step toward transforming themselves into agile digital enterprises focused on accelerating the delivery of innovative products, services, and customer experiences. Enterprises can become the disrupters instead of the disrupted.

However, containerizing applications alone creates technology and management problems of their own, especially when containerized applications need to be deployed and managed at scale.

And that's where Kubernetes comes into play.

Understanding the Role of Kubernetes

Containers solve an age-old problem for application developers: how to port their applications from their laptops to production consistently. Inevitably, the developer's laptop is running a different operating system than the production server or it has different versions of libraries that the application depends on, or both. These differences slow down the entire development and testing cycle as developers and IT operations teams are constantly struggling to match their respective platform configurations and keep them synchronized.

Containers provide a fully self-contained virtual environment for an application to run in — regardless of the underlying platform, whether it's the developer's laptop or a production server.

Of course, no application developer ever said, "I want my application to run really, really slow." So, to address any potential performance issues in production, developers typically deploy as many resources as are available for their application. In the cloud that means *hyperscale* (scaling out horizontally by spreading the workload across multiple containers). But as you start deploying lots of containers, you need a control plane to manage the sprawl.

Containers are great for solving interoperability and packaging issues, but alone they don't scale particularly well. In addition, there is also a trend to make containers small and modular to ease application upgrades, which in turn requires an application to be broken up into modules. Each module and the containers supporting them need to pass configuration data between them. A control plane becomes essential to exchanging data consistently. Hence, container challenges that still need to be addressed include how to

- Scale on demand
- Handle failures and availability
- Pass environment parameters between application modules (for example, database passwords, keychains, and certificates)
- Expose a containerized application to accept and load-balance external traffic across multiple instances

TIP

In much the same way that it makes sense in the world of VMware administrators to have a VMware vCenter server if you have more than two or three ESXi hosts, if you're managing more than 10 or 15 container nodes, it makes sense to have an orchestration and management tool.

Kubernetes addresses these challenges. Kubernetes is an open-source platform that provides a simple application programming interface (API) that lets you define container infrastructure in a declarative fashion. Kubernetes enables containers to run and operate in a production-ready environment at enterprise scale.

REMEMBER

Kubernetes orchestrates containerized applications to manage and automate resource utilization, failure handling, availability, configuration, scalability, and desired state.

By combining container technology with the orchestration of Kubernetes and the modularity of microservices, cloud-native applications can be rapidly published, maintained, and updated through self-service automation and orchestration of application infrastructure in the modern SDDC. Using microservices and containers accelerates an application's time to market and delivers it in a highly modifiable, scalable state.

TECHNICAL STUFF

Kubernetes manages containers. Containers package applications and their dependencies into a distributable image that can run almost anywhere, streamlining the development and deployment of software.

As an application and its services run in containers on a distributed cluster, Kubernetes choreographs all the moving pieces, enabling them to operate in a synchronized way to optimize the use of computing resources and maintain the desired state. The results help deliver the promise of digital transformation:

- Kubernetes makes it easier to run applications in public, private, and hybrid clouds.
- Kubernetes accelerates application development and deployment.
- Kubernetes increases agility, flexibility, and the ability to adapt to change.

YOU TALKIN' TO ME? A TAXI COMPANY TRANSFORMS ITS BUSINESS WITH KUBERNETES

A well-known, long-established taxi company with a reputation for timely, courteous, safe drivers in a major metropolitan area is losing fares to ride-share services, imperiling its once-strong local market share. It needs to transform itself into a digital enterprise capable of competing with ride-share companies. To do so, the company wants to develop its own mobile application, cost-effectively run the application in its modest data center, and provide innovative services.

While recently hired developers worked on the mobile application, the taxi company modernized its data center with commodity hardware and virtualization. To maximize resource utilization of its small data center and minimize costs, the company plans to run its new application in Docker containers on VMs. Kubernetes will orchestrate the containerized application.

After being rolled out and advertised in and on its cars, the application is an instant success. To meet fluctuations in use of the application, the company uses Kubernetes to dynamically scale the number of containers running the application. For example, when metrics for the application hit a predefined threshold indicating high usage, which typically happens during rush hour, the company's DevOps team uses the horizontal pod autoscaling feature of Kubernetes to automatically maximize the number of containers so that the system can match demand. At the other end of the spectrum, the number of containers is reduced to elastically match the low demand in the early morning, predawn hours, thereby conserving resources.

The mobile application correlates ride requests with location. By mining the data and combining it with its intimate historic knowledge of the city's traffic patterns, the taxi company can station cabs in the perfect locations for hailing customers — preempting some car requests to the competition. What's more, because the company processes the application's logs as event streams, the company can do this dynamically throughout the day and night, shifting cars to hot spots as needed.

Because the company implemented the application by using containers, developers can roll out new changes daily. The data that the application collects helps the company pinpoint new features and quickly innovate to focus on its strengths, such as identifying recurring customers and rolling out a rewards program to retain them.

The business benefits of the company's technical agility, containerized application, and Kubernetes orchestration add up to a competitive advantage:

- The scheduling policies in Kubernetes give the company the elasticity it needs to dynamically match demand in a cost-effective way with its modest but now modernized data center.
- Faults and failures are handled automatically by Kubernetes, reducing troubleshooting demands on its small DevOps staff.
- Kubernetes services help keep multiple versions of the same application while service meshes help drive customers of specific profiles to different versions of the same application, giving each customer a personalized experience.
- Kubernetes services let teams roll out various versions of applications, making upgrades easier. The seamless modification of the application and its features helps the company beat its bigger, less local rivals by being more agile and better able to apply its knowledge of local traffic patterns.
- The ease with which the DevOps team can port containers from the test environment to production accelerates the development and deployment of new features.

IN THIS CHAPTER

- Comparing a cluster in vSphere to a Kubernetes cluster
- Looking at how Kubernetes abstracts away infrastructure

Chapter 2
Understanding Kubernetes Constructs

This chapter explains the similarities and differences between VMware vSphere and Kubernetes to help you understand how Kubernetes functions. It explains how Kubernetes abstracts compute, storage, and networking infrastructure along with vSphere.

vSphere and Kubernetes: The Similarities and Differences

Virtual machines (VMs) and containers both run application workloads. So, what exactly is Kubernetes and why do you need it? Simply put, Kubernetes is to containers what vSphere is to VMs (see Figure 2-1). In the early 2000s, VMware Workstation was a really cool technology, but it wasn't quite ready for production workloads in an enterprise data center — that is, until VMware vSphere came along. vSphere enabled centralized management and orchestration of VMs in a production environment, as well as important capabilities such as vMotion, Distributed Resource Scheduler (DRS), and high availability (HA).

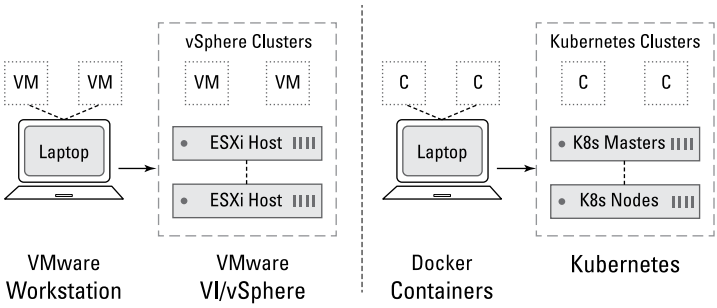

FIGURE 2-1: The VM evolution from Workstation to vSphere compared to the current evolution to Kubernetes.

Similar to a vCenter server and ESXi hosts in vSphere, Kubernetes uses the concept of control plane nodes and *worker* nodes to consume infrastructure resources. A vCenter server pools ESXi hosts to create a cluster across which VMs can be distributed. Kubernetes uses a cluster of nodes to distribute container instances. In this context, the Kubernetes control plane node is equivalent to vCenter in that it's the management plane (the "brains") of the distributed system. The control plane node contains the application programming interface (API) server and the scheduling capabilities of the Kubernetes cluster. The Kubernetes worker nodes act as compute resources, analogous to ESXi hosts. The worker nodes are where you run your workloads, which are known as *pods* in Kubernetes. A pod consists of one or more running container instances, similar to a vSphere virtual application (vApp). Kubernetes stores cluster data in a key-value store called etcd.

High availability and redundancy

A key difference between Kubernetes and vSphere is how they handle workload failures. A vCenter server doesn't have context into the state of applications running within a VM. vCenter can see and control behavior at the Guest OS layer, but not the application itself. Kubernetes, on the other hand, monitors the state of containers in a pod. Kubernetes uses a controller to constantly watch the state of a running pod. If a pod or a container has issues, Kubernetes tried to redeploy the pod. Likewise, if the underlying node in a Kubernetes cluster has issues, Kubernetes redeploys pods to another node. This is the default way Kubernetes operates. vSphere, in contrast, only uses vMotion to move a VM to another ESXi host if vSphere needs to maintain a uniform

load across ESXi hosts, with DRS turned on, or if an ESXi host fails and the cluster it belongs to has HA turned on. vCenter won't redeploy a VM if the application running in the VM has problems.

Another key difference is how redundancy is managed. In vSphere, redundancy is achieved through VMware Fault Tolerance (FT), which maintains a VM with a running instance and a shadow VM running in lockstep. Instructions from the running instance are recorded and replayed in the shadow VM. If the running instance goes down, the shadow VM kicks in immediately. vSphere then tries to find another ESXi host to bring another shadow instance online to maintain the same level of redundancy.

In Kubernetes, you address availability by specifying multiple pod instances in a ReplicaSet. The VM in FT is in active-standby state, whereas Kubernetes replica sets run all the pods active-active. If a pod goes down, the other instances are available to service traffic. At the same time, Kubernetes tries to bring up a substitute for a failed pod on any available node to maintain the desired state configuration. For example, if you want to have four replicas of one pod, Kubernetes monitors those pod replicas. If one of the replicas dies or one of the nodes it's running has issues, Kubernetes self-heals and automatically creates the pod somewhere else. In vSphere, if an app in a VM stops functioning, vCenter won't try to automatically remediate the problem. The major difference between vSphere and Kubernetes is that Kubernetes pod instances are always live and available to service traffic, unlike vSphere FT, which uses shadowed workloads.

Workload form factors

In the same way that an ESXi host can run multiple VMs, a Kubernetes node can run multiple pods. Like VMs, each pod gets a routable Internet Protocol (IP) address to communicate with other pods. A vApp is a preconfigured VM that packages an application and the parameters that define the application's operational details. In much the same way that a vApp can have multiple Open Virtualization Format (OVF) files, a pod can have multiple containers. Typically, with VMs, application deployments are tiered (like a Linux, Apache, MySQL, PHP/Perl/Python [LAMP] stack or a three-tier application). To scale workloads easily and to provide dynamic capabilities, the cloud-native architectures adopt a distributed, more microscopic breakdown of an application. For example, a three-tier stack will have the front end as one of the

tiers, and every component of the front-end web app is packaged and deployed together in the VM. With cloud-native architectures, the front end is broken into multiple containers or pods. For example, the login/authentication part of the front end will be packaged and deployed in a container, the catalog will be packaged in another, and so on. This deconstruction of an app into multiple components helps with building dynamic applications, eases the deployment and upgrade process, and so on.

Resource and quota management

In vSphere, physical ESXi hosts can be logically grouped to form clusters, which can then be sliced into *resource pools.* Resource pools are primarily used for capping resources, such as CPU and memory. In Kubernetes, namespaces provide a similar function, ensuring that resource quotas are managed. However, Kubernetes namespaces are more commonly used to provide multitenancy across applications (or across users if you're using shared Kubernetes clusters).

Kubernetes namespaces segment resources to establish multitenancy. Typically, large teams working on a single Kubernetes cluster will operate in the scope of namespaces. A Kubernetes cluster can have multiple resources defined by the same name as long as they belong to different namespaces. We like to think of namespaces as subdomains and the Kubernetes cluster as the root domain that the namespace gets attached to.

Pod networks

Pods in Kubernetes are created on a network that's internal to the Kubernetes nodes. Typically, this network is an overlay network uplinking to the cluster network. By default, pods cannot talk to each other across the cluster of nodes unless a Service is created. A Service in Kubernetes uses either the cluster network, the node's network, or a load balancer to map an IP address and port of the cluster or node network to the pod's IP address and port. This way, pods that are distributed across nodes can talk to each other if needed (see Figure 2-2).

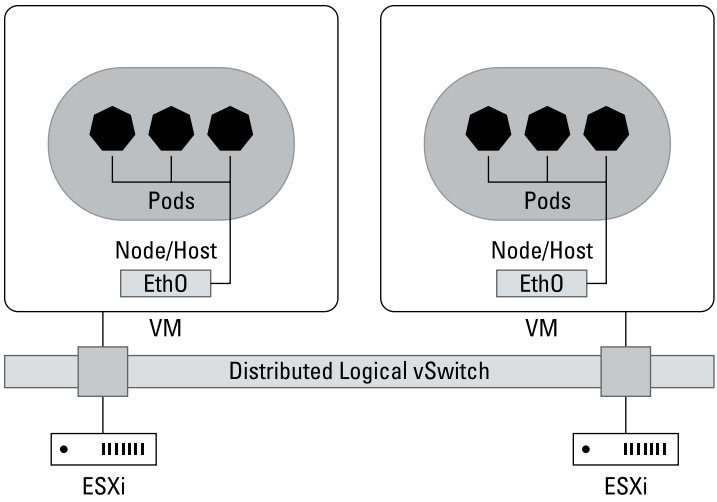

FIGURE 2-2: Networking in Kubernetes.

Pod storage

By default, a pod running in a Kubernetes cluster doesn't have access to persistent storage. The pod derives storage from the node it's running on for local storage needs. When the pod dies, the data associated with the pod is deleted. To have persistent storage, a persistent volume claim needs to be defined. A persistent volume claim defines what kind of storage is needed, how much is needed, and where in a pod the volume should be mounted. A controller running on the Kubernetes node ensures that the volumes are mounted and available to the pod, regardless of which node the pod is running on.

Management

Like the vSphere Web Client, Kubernetes can be accessed through a graphical user interface (GUI), known as the Kubernetes Dashboard. A Kubernetes cluster can also be accessed through its command-line tool, called kubectl. Both the Kubernetes Dashboard and kubectl query the Kubernetes API server to get or manage the state of various resources like pods, deployments, services, ReplicaSets, and so on.

Application configuration

Most VMs that consist of a tiered application will need to talk to each other. In most cases, two VMs need to exchange runtime configuration data, such as database passwords, certificates, and so on. With vSphere, these configurations can be passed using key-value pairs in OVF properties. However, these properties aren't global — they can only be read by the VMs of the same vApp. To effectively pass configuration data between VMs, you need a configuration management tool like Puppet, Chef, or Ansible. These tools need to be configured and scripted apart from the application code itself. Managing application configurations through automated configuration systems can consume a lot of time, although they automate the desired state configuration. Kubernetes, on the other hand, uses config maps and secrets to pass configuration data between containers. These are key-value pairs stored in etcd and accessible across the Kubernetes cluster.

Workload labels

In vSphere, tags are used to identify or group similar workloads so they can be indexed and searched. In Kubernetes, you can assign labels to pods. Labels can be used along with a selector to look up pods belonging to the same application or tier. Apart from inventory management, Kubernetes uses labels to define Services. A Service in Kubernetes allows a group of pods to be exposed by a common IP address. This common address helps you define network routing and load balancing policies without having to understand the IP addressing of individual pods.


TECHNICAL STUFF

The ability of Kubernetes to add labels and use them in routing network traffic opens the possibility of multiple use cases and is the basis of microservices.

Abstracting Away Infrastructure with Kubernetes

Kubernetes decouples the needs of an application from the infrastructure it runs on. Kubernetes serves the application's needs by transferring declarative states defined by users into corresponding infrastructure elements. The storage needs of an application, for example, can be defined using persistent volume claims in a

YAML file. Kubernetes then talks to the storage provider available in the infrastructure to carve out a volume that it can mount on a pod. Similarly, when an application needs load balancing, you can define a Service type in Kubernetes. Kubernetes, in turn, looks for a load balancer that it can map to a set of pods using the label selectors. Kubernetes itself can't do much unless the infrastructure it's running on can provide storage and networking capabilities to match the application's needs on demand.

Kubernetes includes standard plug-ins and interfaces that make it compatible with various infrastructure providers. For example, a standard way of interacting with storage is through the Container Storage Interface (CSI). To provision networks, Kubernetes uses the Container Network Interface (CNI).

Kubernetes provides a standard API to define an application's desired state. Kubernetes also converts the requirements for a desired state into infrastructure resources through standardized interfaces. This capability paves the way to building an application once and deploying it on any infrastructure or cloud, opening up the possibility of using multiple clouds.

Kubernetes and storage

Storage for containers on Kubernetes differs somewhat from how you typically provide storage for applications running on VMs on vSphere. There is, however, some common ground. Any vSphere datastore can be used to store data for containerized applications as well as traditional applications.

But first, let's look at how storage for containers differs from storage for VMs. Multiple factors need to be considered when handling persistent data using containers, such as the following:

- Containers are ephemeral by nature, so the data that needs to be persisted has to survive through the restarting or rescheduling of a container.
- When containers are rescheduled, they can die on one host and get scheduled on a different host. In such a case, the storage should also be shifted and made available on the new host for the container to start gracefully.
- You shouldn't have to worry about managing an application's volume and data. The underlying infrastructure should handle the complexity of unmounting and mounting.

Kubernetes provides seamless storage capabilities on nodes where workloads are scheduled. Its powerful plug-in framework, along with a standard API, allows different storage systems to be exposed as volume plug-ins. This framework standardizes the way persistent, ephemeral, or local storage is consumed by pods as file or block volumes.

Kubernetes has adopted the CSI, which is a community-driven effort to standardize how file and block storage is exposed to and accessed by containers and orchestrators like Kubernetes. The CSI defines a programmatic interface to perform standard volume operations on various kinds of back-end storage. The CSI enables third-party vendors to integrate their storage systems with Kubernetes, enabling support for a wide array of storage providers.

Kubernetes and network

Networking in Kubernetes is nested. Pods are connected to an overlay or software-only pod network. Kubernetes uses the CNI to create pod networks and connect them to the underlying node's network. These pods are exposed to external traffic through Services. The CNI plug-in is primarily responsible for allocating and assigning IP addresses to pods to facilitate pod-to-pod communications, to build Services, and to enable network policies.

Kubernetes has a huge ecosystem of network providers that integrate with the CNI to deliver various levels of network capabilities. For example, in the context of a software-defined data center (SDDC), the NSX-T Container Plug-in (NCP) integrates NSX-T with Kubernetes.

IN THIS CHAPTER

- Building a Kubernetes cluster in a VMware SDDC for application workloads
- Consuming network and storage for Kubernetes with VMware SDDC
- Managing workloads with Kubernetes in vSphere

Chapter 3
Kubernetes on vSphere Explained

In this chapter, you see how Kubernetes clusters can be built on top of vSphere and how a software-defined data center (SDDC) caters to the network and storage needs of Kubernetes. We also show you how Kubernetes embedded within vSphere can help with managing workloads on vSphere.

Mapping a Kubernetes Cluster in a VMware SDDC

Moving to a cloud-native architecture is ideal for enterprises. However, most enterprises have applications that are critical to their businesses and are not containerized or cloud native. Although they move to cloud-native architectures, enterprises will have a mix of different workload types. For this reason, it's critical to have a common infrastructure and tool sets that cater to both legacy and modern architectures. The VMware SDDC has evolved to be the common engine that runs both legacy and modern applications side by side (see Figure 3-1).

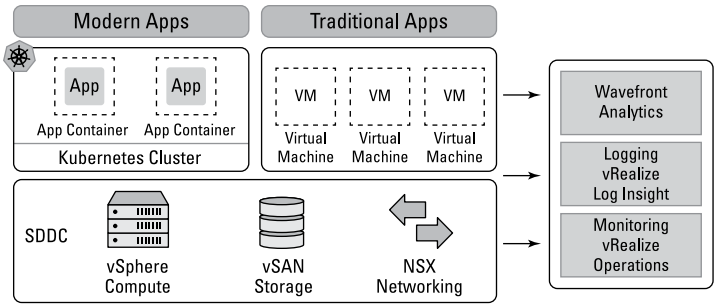

FIGURE 3-1: A VMware SDDC, along with Kubernetes, enables enterprises to run traditional and modern applications in the data center.

In Kubernetes, a control plane and worker nodes (see Chapter 2) make up a cluster. The Kubernetes cluster derives its resources by pooling resources from the nodes that are part of the cluster.

A Kubernetes cluster on vSphere uses a vCenter cluster or a resource pool to provision Kubernetes nodes as virtual machines (VMs). The elasticity of the vCenter cluster or resource pool helps scale a Kubernetes cluster by adding or deleting VMs for the Kubernetes nodes. You can use standardized control plane and worker node templates as Open Virtualization Format (OVF)/Open Virtualization Appliance (OVA) images to build Kubernetes node images. A vCenter server running existing workloads can also be used to create a Kubernetes cluster. Seen from vSphere, Kubernetes is just another workload consuming resources through VMs.

A highly available (HA) Kubernetes cluster can be created on vSphere with multiple master and worker nodes running as VMs (see Figure 3-2). In case of ESXi host failure, vSphere HA moves Kubernetes cluster nodes to another ESXi host. In case a Kubernetes cluster node fails, Kubernetes re-creates pods on the next cluster node VM.

After a Kubernetes cluster is deployed as VMs on vSphere, development teams can start accessing the Kubernetes API that gets deployed as part of the control plane VM. Developers can then use the Kubernetes API to deploy containerized applications that get deployed as pods within the cluster.

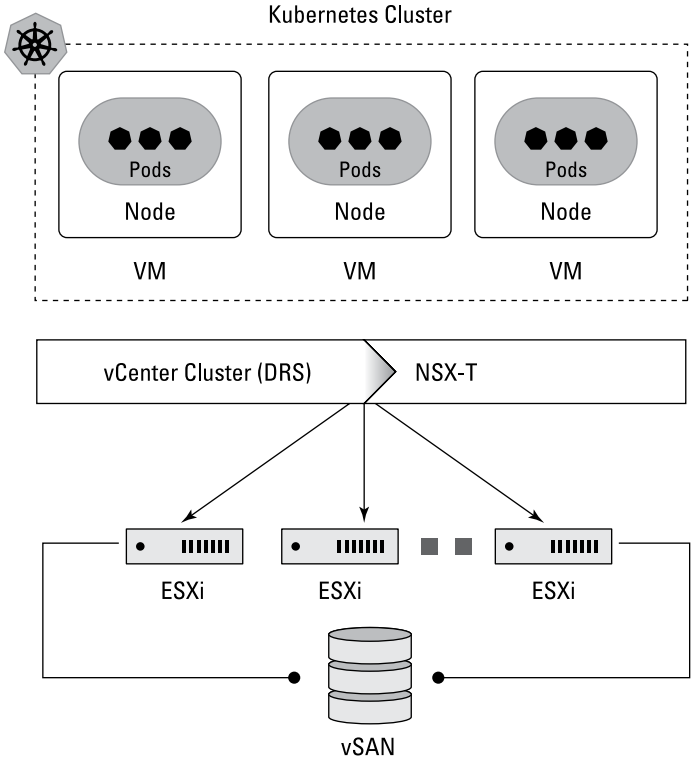

FIGURE 3-2: HA in Kubernetes clusters on an SDDC.

Delivering a Dynamic Network for Kubernetes on vSphere

An infrastructure that supports containerized applications must accommodate the dynamic and nested nature of container networking (see Chapter 2). The container network must share the life cycle of the applications deployed on Kubernetes — created dynamically and scaled on demand when the application is scaled. Ingress traffic must be routed to the right pods. Anything less results in incomplete automation and limited agility. For example, when a service of type load balancer needs to be created in order to allow external traffic to a pod, the Container Network Interface (CNI) must talk to a network provider that can automatically provision a load balancer (LB) and configure it to forward traffic to corresponding pods. Unless Kubernetes has a software-driven

network provider that can provision an LB on demand, Kubernetes can't expose a pod to receive external traffic dynamically.

Apart from the need to create logical network constructs on demand, there are three distinct network requirements within a Kubernetes cluster:

- **Node network:** The network that provides switching and routing for all the external network interfaces of the nodes of a cluster. This network also needs to run across ESXi hosts, because the Kubernetes node VMs can be distributed across ESXi hosts.
- **Pod network:** The network that provides switching and routing capabilities to the pods in a cluster. This is the east–west traffic within a cluster.
- **LB services and security policy:** Because pods are nested inside nodes, the cluster needs a mechanism to route external traffic to and from the cluster.

There are two methods that can implement the preceding network stack requirements within vSphere:

- Using VMware NSX-T to implement all three requirements
- Using a combination of vSphere Distributed Switch (vDS) to provide node networking, along with a CNI solution like Antrea to provide pod networking and NSX Advanced Load Balancer (see Figure 3-3)

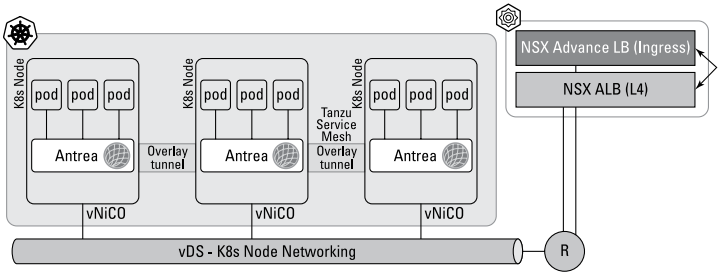

FIGURE 3-3: Kubernetes networking with vDS, Antrea, and NSX Advanced Load Balancer.

In the first solution, VMware NSX-T Data Center helps simplify networking and security for Kubernetes by providing on-demand software-driven services to build node networks, pod networks, and LBs, and by automating the implementation of network policies, network object creation, network isolation, and micro-segmentation. NSX-T has a Network Container Plug-in (NCP) that works with the Kubernetes CNI. The NCP runs as a container on each Kubernetes node.

NCPs play a prominent role in automation. They monitor the changes to containers and other resources and manage container networking requirements through application programming interface (API) calls to NSX Manager and the Kubernetes control plane. An NCP creates the pod networks as an overlay container network, and NSX can provision cluster networks as an overlay by using virtual switches.

Another way to implement networking for Kubernetes in vSphere would be to use a combination of vDS, a CNI implementation like Antrea, and the NSX Advanced Load Balancer. With this stack, the vDS takes care of the node networks and uplinks them to external networks. Antrea provides pod networking based on Open vSwitch. Antrea also provides implementing security policies and NSX Advanced Load Balancer takes care of providing LB services for ingress traffic to the cluster.

Creating Persistent Volumes for Kubernetes on vSphere

Kubernetes supports both stateful and stateless applications. Stateful applications need to persist data. Although Kubernetes supports persistent storage of data, it depends on the infrastructure to provision storage for persistent volumes. Kubernetes manages the creation, attachment, and deletion of persistent volumes by interacting with the infrastructure through the Container Storage Interface (CSI).

VMware has developed a vSphere cloud provider and volume plug-in to work with the CSI. vSphere datastores use them to carve out volume disks that Kubernetes can mount on its nodes. The plug-in works with Virtual Machine File System (VMFS) and virtual storage area network (vSAN) datastores.

An existing datastore in vCenter can also be used to support Kubernetes clusters. Nothing needs to be done within the datastore to enable it to interact with a Kubernetes cluster. The vSphere cloud provider running within the Kubernetes cluster handles the build-out of volumes using vSphere datastores.

A datastore can be used for VMs and as persistent volumes for containers (see Figure 3-4).

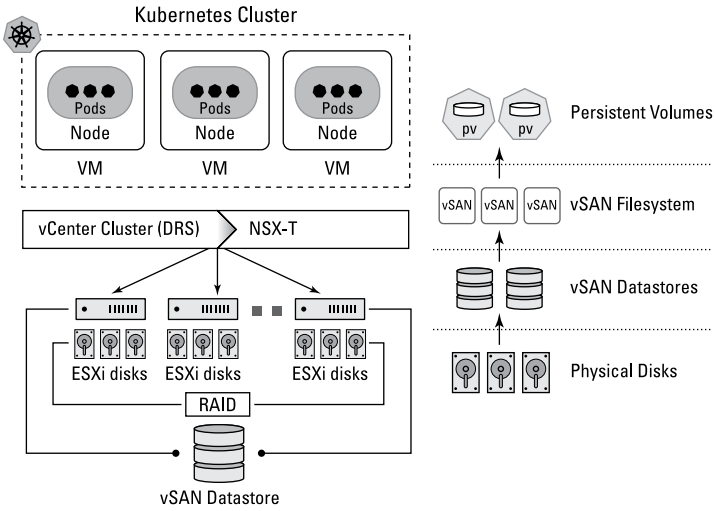

FIGURE 3-4: vSAN software-defined storage in the SDDC.

A VMware SDDC furnishes the API-driven infrastructure that Kubernetes expects to run efficiently.

Embedding Kubernetes within vSphere

So far, we've talked about enabling Kubernetes clusters on vSphere with Kubernetes cluster nodes being VMs leveraging SDDC. In Chapter 2, we outline the similarities and differences between Kubernetes and vSphere. In particular, the approaches to managing Infrastructure resources are very similar. vSphere is managing VM-based workloads and Kubernetes is managing container-based workloads. Table 3-1 lists all the similarities between the two platforms.

TABLE 3-1 Similarities between vSphere and Kubernetes Functions

Function	vSphere	Kubernetes
Compute resources	Collection of ESXi hosts as a vSphere cluster	Collection of nodes as a Kubernetes cluster
Resource distribution	vCenter via Distributed Resource Scheduler (DRS)	Kubernetes Controller
Application distribution	vApp	Pod
High availability	vSphere Fault Tolerance (FT)	ReplicaSets
Resource policies	Resource pools quotas and limits	Namespace quotas

Running a Kubernetes cluster in VMs on vSphere is a great way to give developers access to a declarative API for containerized workload management, but the overall stack architecture becomes complex with separate entities (vSphere and Kubernetes) solving similar problems (VM and container orchestration, allocation, and so on). At the same time, managing Kubernetes clusters as vSphere VMs also means there is more handoff involved between various teams that operate the stack. For example, a vSphere administrator will manage the infrastructure and allocation via vSphere resource pools/clusters. A DevOps engineer will be responsible for creating and managing admin-level functions of a cluster before it's handed over to a developer. Also, a developer using the platform will have to talk to different sets of APIs (vSphere and Kubernetes) if they're working with both VMs and containers.

As such, with vSphere 7.0, VMware decided to bring the two platforms together by embedding Kubernetes constructs within vSphere. This is called vSphere with Tanzu. vSphere with Tanzu combines the overall architecture needed to run Kubernetes and vSphere together. It does so by running processes that implement Kubernetes control plane elements like the Kube-proxy, Kubernetes API server, container runtime, and so on, within the ESXi host. As such, a cluster of ESXi hosts in vSphere is also a Kubernetes cluster (see Figure 3-5).

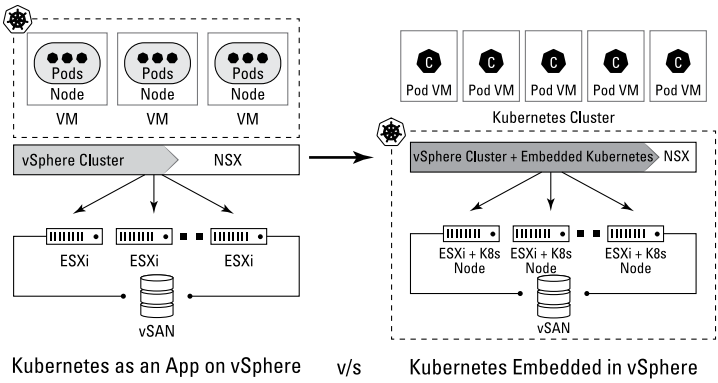

FIGURE 3-5: A Kubernetes cluster implemented with VMs as nodes versus ESXi hosts as nodes.

The Kubernetes cluster that's also a vSphere cluster is called a *supervisor cluster.* Similar to creating a resource pool in vSphere, when vSphere with Tanzu is enabled, a VI admin can also set a namespace with resource allocation and quotas within vSphere called as the vSphere Namespace. When a developer deploys a container image via the Kubernetes API in the supervisor cluster, vSphere will deploy the container image in a lightweight VM called the vSphere Pod, similar to a Kubernetes Pod, except a vSphere Pod is a very lightweight VM and highly secure.

Apart from embedding Kubernetes within the ESXi hosts, another set of functionalities that goes within vSphere with Tanzu (shown in Figure 3-6) is the ability to create more Kubernetes Clusters with VM-based nodes, this is called the vSphere with Tanzu Kubernetes Service. This service gives teams flexibility to choose which Kubernetes cluster they would like to use depending on the use case, team size, and workload. These clusters can be created in a vSphere Namespace.

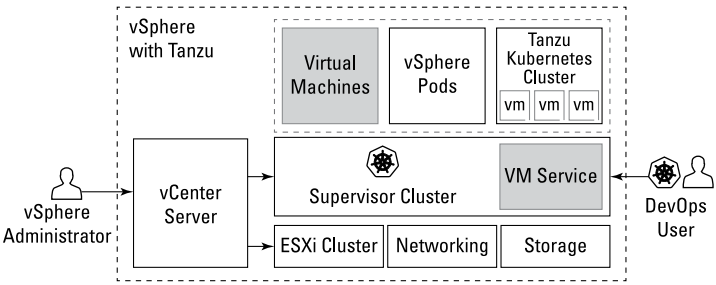

FIGURE 3-6: vSphere with Tanzu.

NSX-T and vSphere Storage provide networking and storage capabilities to the supervisor cluster.

VM service

So far, with vSphere with Tanzu, development teams get the ability to provision containerized workloads or even create new Kubernetes clusters. But what if the development team working with containerized workloads is also working on workloads that operate on VMs? They'll be dealing with two different APIs — one is the vSphere API to manage VMs, and the other is the Kubernetes API to manage containers.

We have Kubernetes API embedded in the supervisor cluster, the same cluster that's also a vSphere cluster, and this architecture can be leveraged to use a single declarative API to manage both containers and VMs. vSphere with Tanzu enables this feature and is called the VM Service. A Kubernetes style declarative YAML or Kubectl (CLI) can be used to define a VM service. The development teams would have to provide what the VM class is (which is essentially explaining what the sizing for the VM would be), a VM image (which is what VM template to replicate the image from), and a storage class (which tells the API which datastore to use to provide storage for the VM).

VI administrators can predefine and allocate VM classes, VM images, and storage classes to vSphere Namespaces based on team requirements.

IN THIS CHAPTER

- Empowering DevOps teams with self-service infrastructure
- Deploying applications rapidly with Kubernetes
- Updating applications safely and seamlessly with Kubernetes

Chapter 4
Business-Ready Infrastructure

In this chapter, you explore the possibilities of a self-service platform for DevOps teams enabled by Kubernetes on vSphere, and how Kubernetes accelerates application deployment and maintenance, ultimately giving businesses the power to be more agile and make faster decisions.

Enabling DevOps with a Self-Service Platform

Deploying infrastructure to support application development is often cumbersome, error-prone, and time-consuming, even for DevOps teams that follow automated code build and deployment practices. Often, heavy scripting is required to work with infrastructure platforms in order to orchestrate application deployments, configurations, and so on. Passing config data between apps, certificates, passwords, and so on is necessary for most applications. Typically, DevOps teams use tools like Chef, Puppet, and Ansible to manage applications beyond just deployments. Building such systems to deploy applications has its own challenges. For example, the scripts used to automate deployments may not be idempotent — that is, they may not produce consistent

results each time they're run. A failure or a bug in these scripts can take valuable time away from development teams building applications that matter to the business and force them to, instead, spend time debugging issues with their build and deployment systems. Or, consider a scenario in which the application programming interfaces (APIs) to the underlying infrastructure have changed or the business wants to adopt a new cloud provider. In each of these scenarios, the deployment automation will have to be rewritten to accommodate new environments.


TIP

To increase business agility and improve productivity, DevOps teams need to streamline the way they roll out applications and manage the infrastructure to support their applications, as well as the ability to roll out updates after applications have been deployed. Kubernetes and containerized applications provide a common API-driven framework to streamline these important tasks.

Containers package applications and their dependencies into a distributable image that can run almost anywhere, thereby eliminating the need for separate infrastructure and streamlining the development and deployment of software. Developers like containers because they simplify application development, distribution, and testing. Operations teams like containers because they eliminate the manual configuration, provisioning, and scripting typically required for traditional infrastructure deployments.


TIP

Business-ready infrastructure radically reduces manual infrastructure processes and the requirement for developers to handle nondevelopment tasks, resulting in significantly greater developer productivity. Kubernetes on vSphere provides secure, software-based compute, storage, networking, and operational tooling optimized for application workloads running in containers. With Kubernetes on vSphere, IT operations can let development teams run their own applications. The IT operations team provides development teams with a self-service platform to address their applications' needs.

Deploying an Application: Before Kubernetes and with Kubernetes

Now, let's look at life for a vSphere administrator before Kubernetes, and how much better life is after Kubernetes on vSphere.

A typical application deployment workflow before Kubernetes and containers might look something like this:

1. The development team writes code for version 1.0 of an application.
2. The development team tests the code on a virtual machine (VM) with a specific operating system (OS) and libraries running on their laptops or in a sandbox environment.
3. After the code is tested, the development team hands the application binaries over to the IT operations team.
4. The IT operations team creates a VM for the application with the same OS and libraries that were installed on the developers' laptops in the sandbox environment.
5. The IT operations team deploys the application.
6. The IT operations team replicates copies of the application for high availability and load balancing.
7. The IT operations team handles configuration management for the application to talk to other tiers, such as the database and application back ends.
8. The IT team also creates a security profile for the app based on the kind of traffic the application is expected to interact with.
9. After the app is deployed, the application needs to be prepared to handle workload and external traffic. To do this, the IT operations team captures the IP addresses for all the VMs replicated for the application.
10. The IT operations team then gives the IP addresses of the VMs and the security profile to the network team.
11. The network team then creates a load balancer, firewall rules, and so forth.
12. The application is now ready to receive external traffic.

Keep in mind that any time the IP addressing scheme of the VMs backing the app changes, IT and network teams need to reconfigure their load balancers and security profiles.

In case the app needs to scale, coordinated efforts between IT and network teams need to happen to incorporate additional application instances.


TIP

Some of the preceding steps can be automated using infrastructure management tools like Terraform and configuration management tools like Chef and Puppet.

Now, let's look at this same workflow with Kubernetes in a software-defined data center (SDDC):

1. The development team writes code for version 1.0 of an application.
2. The development team tests the code in a container image on laptops or in sandbox environments.
3. After the code is tested, the development team builds a container image for the application.
4. The development team defines the desired state of the application in a YAML file, which includes how many instances of the container to run, which ports to expose the applications on, and what other configuration information needs to be passed between container instances.
5. The development team deploys the application using the Kubernetes API to process the YAML files defining the desired state.
6. The application is now ready to receive external traffic.

That's it! Kubernetes will deploy the necessary number of instances defined in the application across its nodes. It will talk to VMware NSX to provide a load balancer service, configure the load balancer to route traffic to the deployed instances using their labels, and talk to vSphere to provision a persistent volume and attach it to the nodes. Kubernetes will also scale the app by deploying additional pods using its pod auto-scaler.

If you're thinking, "Big deal — I saved six steps," you're missing the real magic! Notice that with Kubernetes on vSphere, there is no longer the need to hand over different steps in the workflow

between the development, IT operations, and network teams — which can take even longer when key team members aren't available or need more information. Not only does Kubernetes on vSphere eliminate time-consuming steps and tickets, but it also automates previously manual tasks, reduces the opportunity for errors, and greatly accelerates the entire software-deployment process. Another great benefit is the ability to move the application across different cloud and infrastructure without changing any aspect of the application code. The same YAML file can be applied against a Kubernetes cluster on any other cloud, and the application will be deployed with no changes or only minute changes.

Updating an Application: Before Kubernetes and with Kubernetes

Of course, deploying an application isn't the only way in which life for a vSphere administrator gets better with Kubernetes on vSphere. Updating your applications gets better, too. In fact, they should name an ice cream flavor for Kubernetes — maybe, "Peanut Butter and Chocolate Kubernetes Clusters on vSphere"!

Let's look at typical workflow for updating an application before Kubernetes:

1. The development team writes code for version 1.x of an application.
2. The development team tests the code on a VM with a specific operating system and libraries running on their laptops or in a sandbox environment.
3. The development team hands the tested version 1.x application binaries to the IT operations team.
4. The IT operations team identifies a maintenance window to deploy the application update.
5. The IT operations team updates the individual VM instances of version 1.0 with any necessary OS patches and updated libraries.
6. The IT operations team deploys version 1.x on each VM instance.

7. The IT operations team runs some basic tests to ensure the updated application works correctly (that is, it's reachable).
8. The application is updated and the maintenance window is completed.

And now, with Kubernetes:

1. The development team writes code for version 1.x of an application.
2. The development team tests the code in a container running on laptops or in sandbox environments.
3. The development team builds the container images for version 1.x of the application.
4. The development team deploys version 1.x of the application in production with a version 1.x label.
5. The development team changes the configuration for service (load balancing service in Kubernetes) for version 1.0 to point to all instances running with the version 1.x label.
6. The application is updated.

Once again, with Kubernetes on vSphere, there is no need for a handover between the development and IT operations teams. No manual tasks need to be performed. And, there's no need for a maintenance window — the application can be updated without a service outage!

IN THIS CHAPTER

- Aligning development and IT operations teams
- Simplifying Kubernetes log management
- Gathering key Kubernetes metrics
- Ensuring backup and recovery capabilities

Chapter 5
Automating and Optimizing Operations

In this chapter, you find out about the evolving and converging roles of developers and IT operations teams in a DevOps environment and how tools such as Fluentd, VMware vRealize Log Insight, VMware vRealize Operations, Tanzu Observability by VMware, and Velero help DevOps teams automate and optimize key operations in a Kubernetes environment.

From Managing vSphere to Managing Kubernetes

While developers and IT personnel collaborate on operations to release software early and often — even daily or hourly — Kubernetes is at work automating the building, scaling, and publishing of applications.

Kubernetes breaks down the organizational barriers between developers and IT operations to align both roles behind the common goal of quickly turning ideas and innovations into releasable, maintainable software. A culture of collaboration is key, not only

between these two roles — who often come together to form a single DevOps team — but also with other teams, such as security.

Unlike applications in a client–server model or a tiered application stack, applications running on Kubernetes are broken down into smaller modules. The reasons for doing this are simple:

- Individual application development teams can be further organized into smaller teams mapped to specific functionality, and the modules developed by these teams can exchange data through an application programming interface (API) via Kubernetes services.
- Overall development of applications can be multi-threaded.
- Individual application components can be updated asynchronously.

With so many pieces of an application exchanging data with each other, debugging and tracing the application becomes more complex if a problem occurs. Monitoring and observing the system across the entire stack — all the way from the application to the infrastructure — becomes increasingly important. The tools used to monitor a Kubernetes stack must be able to see what's going on within the application, as well as the underlying infrastructure supporting the application. There is also a need to monitor various kinds of data, such as metrics that show time-based events.


TIP

In a Kubernetes cluster running in a VMware software-defined data center (SDDC), existing monitoring and observability tools are extended to look across the Kubernetes stack.

Processing Logs with Fluentd and vRealize Log Insight

Centralized logging is an essential part of any enterprise Kubernetes deployment. Configuring and maintaining a real-time high-performance central repository for log collection can ease the day-to-day operations of tracking what went wrong and its impact. Effective central logging also helps DevOps teams quickly observe application logs to analyze application performance. Compliance and auditing requirements often require a company to maintain digital trails of who did what when. In most cases, a

robust logging solution is the most efficient way to satisfy these requirements.

Kubernetes on vSphere creates a powerful logging layer on top of Kubernetes using a combination of Fluentd and VMware vRealize Log Insight.

Fluentd

Fluentd is an open-source log processor and forwarder that collects logs from different sources, unifies them, and sends them to various monitoring destinations. With Fluentd, you start by defining directories where log files are stored (the source configuration), applying transform or filter rules based on the type of the message (the filter configuration), and deciding how to route the transformed message to a set of destinations by using output rules (the output configuration), as follows:

- **Source configuration:** The source configuration tells Fluentd where to look for logs, such as Kubernetes worker and master nodes and container log directories.
- **Filter configuration:** Filtering is about transforming the data stream, appending additional information to simplify queries, or extracting information to provide global context. For container logs, the kubernetes_metadata filter is an open-source plug-in that uses basic container information, such as the container name, to query the Kubernetes API server to obtain and append Kubernetes metadata to raw container logs.
- **Output configuration:** The output plug-in determines the routing treatment of formatted log outputs. Fluentd offers three types of output plug-ins: non-buffered, buffered, and time-sliced.

Fluentd itself runs in the Kubernetes cluster and forwards logs from across the cluster to an aggregator such as vRealize Log Insight.

vRealize Log Insight

An effective log aggregator must support the processing of events from thousands of endpoints, the ability to accommodate real-time queries, and an analytics engine to provide intelligent metrics to solve complex technical and business problems. vRealize

Log Insight delivers heterogeneous and highly scalable log management with intuitive, actionable dashboards, insightful analytics, and broad third-party extensibility. Log Insight also provides deep operational visibility and faster troubleshooting across physical, virtual, and cloud environments.


TIP

You can build your interactive queries as complex as you need to match various conditions. Wildcards can help your queries fulfill your requirements.

Kubernetes operators can also convert interactive analysis filters into custom dashboards. Custom dashboards let you monitor Kubernetes pods, namespaces, or types of events based on the events that come in. You can share dashboards to ensure that everyone responsible for the infrastructure or application is looking at the same set of metrics. You can also convert interactive analysis filters into alerts, which can be forwarded to your email and vRealize Operations.

Collecting Metrics with vRealize Operations and Tanzu Observability

Monitoring is a cornerstone of profiling, analyzing, and optimizing applications. As an increasing number of applications either move to or are built on container technologies, the challenges of monitoring these ephemeral workloads become a greater part of the operational burden for application developers and vSphere administrators.

Kubernetes environments on vSphere present some unique monitoring challenges that take you beyond your usual day-to-day work of monitoring applications on virtual machines because of the following factors:

- The ephemeral nature of containers
- The increasing density of objects, services, and metrics within a given Kubernetes node
- A focus on services rather than virtual machines
- More diverse consumption of monitoring data
- Changes in the software development life cycle

To address these factors, you must gather metrics from two sets of Kubernetes sources: clusters and pods. Information from clusters lets you monitor the system's arrangement and utilization of resources. Metrics from pods provide insight into how the services running within and between containers are functioning. Both sets of data are required to get a complete operational picture of a Kubernetes environment. Together, vRealize Operations and Tanzu Observability enable real-time health monitoring and analytics for cloud-native applications and platforms.

vRealize Operations

vRealize Operations is an infrastructure operations and monitoring tool for a software-defined data center (SDDC). It's primarily used by infrastructure administrators, cloud administrators, and platform reliability engineers.


TIP

The Kubernetes management pack for vRealize Operations provides complete topology and health status information for namespaces, clusters, ReplicaSets, nodes, pods, containers, and services. The Kubernetes dashboard in vRealize Operations highlights the performance metrics and alerts for the following:

- Kubernetes clusters and their objects, total metrics, alerts, and health scores
- Node and pod performance, health, properties, and trend metrics (CPU and memory usage, disk input/output read and write)
- Topology view of pods associated with other components in a cluster
- Topology view of health status of all cluster members with alerts

Tanzu Observability

Tanzu Observability monitors Kubernetes clusters and the applications deployed on them. Tanzu Observability ingests, stores, and displays metrics, and it can alert you to address system and application issues. It's primarily used by DevOps teams, developers, and platform reliability engineers.

Tanzu Observability monitors containers at scale to deliver the following capabilities and benefits:

- Finding problem indicators of containerized applications — at any scale — using real-time analytics
- Improving developers' and DevOps teams' productivity with self-serve, customizable dashboards and metrics for Kubernetes, containers, and applications
- Proactively detecting cloud service resource bottlenecks with query-driven alerts
- Correlating top-level application metrics with Kubernetes orchestration metrics down to the container and resource level

Tanzu Observability integrates with Kubernetes to ingest platform-level metrics at various levels, such as the cluster, node, pod, and container. To begin gathering metrics, you complete the following steps:

1. **Deploy a Tanzu Observability proxy in Kubernetes.**

   This proxy is deployed as a Kubernetes ReplicationController, which is a construct that ensures a specified number of pod replicas are running at a given time.

2. **Deploy the kube-state-metrics service.**

   This service enables the proxy container to communicate with the Tanzu Observability platform.

3. **Deploy an instance of Heapster.**

   Heapster aggregates monitoring and event data. It supports Kubernetes natively and works on all Kubernetes deployments. Heapster runs as a pod in the cluster similar to any other Kubernetes application. Heapster forwards metrics from the Kubernetes cluster to the Tanzu Observability proxy.


TIP

You can find instructions for integrating Tanzu Observability with Kubernetes at `https://docs.vmware.com/en/VMware-Tanzu-Observability/index.html`.

Performing Backup and Recovery Operations

As a vSphere administrator, one of your chief concerns is data protection. When you're managing Kubernetes on vSphere, you must ensure that you can restore a cluster to a known good state, recover from a crashed cluster, or migrate to a new environment if disaster strikes.

Velero is an open-source tool from VMware to safely back up, recover, and migrate Kubernetes clusters and persistent volumes. It works both on premises and in a public cloud. Velero consists of a server process running as a deployment in your Kubernetes cluster and a command-line interface (CLI), which DevOps teams and platform operators use to schedule backups, trigger ad-hoc backups, and restore data.

Velero uses the Kubernetes API to capture the state of cluster resources and to restore them when necessary. This API-driven approach provides key benefits, including the following:

- Backups can capture subsets of the cluster's resources, filtering by namespace, resource type, and label selector, providing a high degree of flexibility around what's backed up and restored.
- Users of managed Kubernetes offerings often don't have access to the underlying etcd database, so directly backing up and restoring it isn't possible.
- Resources exposed through aggregated API servers can easily be backed up and restored even if they're stored in a separate etcd database.

Velero also enables you to back up and restore your applications' persistent data alongside their configurations by using either your storage platform's native snapshot capability or an integrated file-level backup tool called restic.

REMEMBER

Velero is a production-grade backup solution for Kubernetes cluster resources and applications.

IN THIS CHAPTER

- Protecting the full cloud-native stack
- Managing identity and access

Chapter 6
Locking Down Cloud-Native Applications on vSphere

This chapter describes inherent security risks and prudent countermeasures at the container application, management, orchestration, and infrastructure layers of a cloud-native environment. It also looks at the identity and access control measures that are necessary to secure Kubernetes on vSphere.

Locking Down the Environment

Containerized applications require full-stack security. Threats and security risks abound throughout an unsecured cloud-native stack, and containers, like any other computer technology, are subject to various attack vectors.

A full cloud-native stack can be divided into the following layers (see Figure 6-1), each of which must be secured:

- Applications
- Container management

- Orchestration platform
- Infrastructure

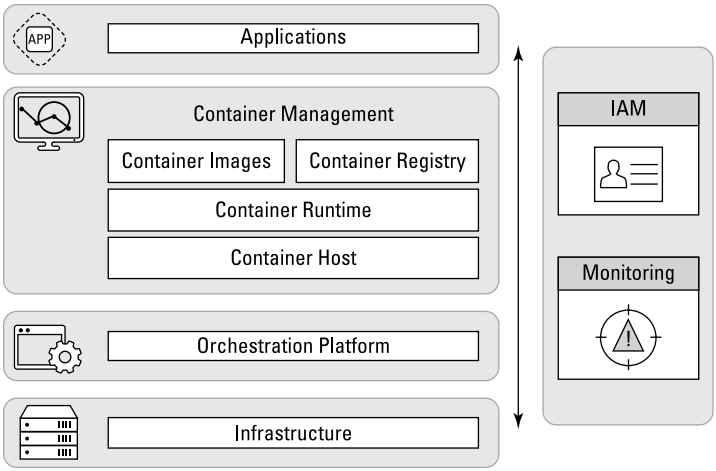

FIGURE 6-1: The layers in the cloud-native stack.

Within these layers, container images are created, deployed, managed, orchestrated, and replaced. As part of their life cycle, containers rely on a collection of systems and constructs embedded in these layers — systems and constructs such as image registries, networking, persistent storage disks, application programming interfaces (APIs), multitenancy, identity management, and access control.

WARNING

Without the ability to integrate containers and an orchestration system with your existing security systems and your data center, the security requirements of containerized applications can lead you to build custom components or integrations at great risk and expense.

The following sections identify the risks, threats, attack vectors, and countermeasures associated with each component in the cloud-native stack.

Containerized applications

Just like traditional applications, containerized applications are vulnerable to software flaws, which an attacker can exploit to gain unauthorized access to sensitive data, attack other containers, or

attack the host operating system. Without isolating containers on virtual machines (VMs), for instance, containers running on the same host can potentially connect to one another, creating a path through which an intrusion can spread.

Countermeasures should not only seek to eliminate vulnerabilities and reduce the attack service but also isolate containerized applications on VMs to impose strong security boundaries.

It's critical to point out that containers are not miniature VMs, and containers do not establish security boundaries as VMs do. An important implication of the National Institute of Standards and Technology (NIST) *Application Container Security Guide* (Special Publication 800-190) is to run containerized applications on VMs.

Deploying containers with VMs encases an application with two layers of isolation, an approach that is well suited to cloud environments with multitenancy and multiple workloads. You can — and should — take isolation a step further by also segmenting containerized workloads by cluster. This approach is particularly powerful when a segmented cluster is a unit of tenancy in a multitenant context and when the isolation can be automatically and logically enforced in the network.

Packaging an application in a container also gives you a prescribed way to apply security principles with more depth and breadth. You can, for example, adopt core protection strategies to prevent exploits and protect an application's memory, configuration, and interfaces.

Key countermeasures for containerized applications include the following:

- Profile an application and then monitor it for changes to its memory, endpoints, and configuration to ensure a known good state at runtime.
- Eliminate vulnerabilities and reduce the attack surface.
- Isolate containers on VMs to impose strong security boundaries.
- Segment each containerized application in its own cluster.

Container management

Components in the container management layer include the following:

- **Container images:** Vulnerabilities, defects, malware, and plain-text passwords can mar container images — and such risks shift from possible to probable when the origin of a container is dubious or unknown. The portability of containers makes it easy to obtain an image and run it to solve an immediate problem, but using untrusted images in haste can introduce malware into an environment, lead to a data breach, or expose a vulnerability. An image of unknown origin or with unknown base layers can contain malicious files. The components in an image may be missing security updates or patches, exposing vulnerabilities that an attacker can exploit. Meanwhile, defects can take root in configurations, such as being run with more privileges than required, or in unnecessary components, such as the Secure Shell (SSH) daemon. There is also the risk of embedded secrets that you don't know about. An image can contain plain-text passwords or private keys that let one component of an application connect to another, but anyone with access to the image can find the secret and use it to do harm. Key countermeasures for container images include
  - Keeping images patched and up to date
  - Configuring images to limit privileges and exclude unnecessary components, such as the SSH daemon
  - Securely storing secrets, encrypted, in Kubernetes, not in the image
- **Container registry:** A container image registry without curation and security steeps your cloud-native environment in risk. The risk starts with insecure connectivity. Allowing connections to a registry over an insecure channel gives attackers a vector that can be exploited to expose proprietary services or embedded secrets, steal credentials, or send spoofed images to an orchestrator. Another risk is that without vigilance, vulnerable versions of images can linger in the registry, leading to accidental use. Curation is key to keeping images scanned, patched, up to date, and signed as trusted. The most important risk centers on inadequate authentication and access control. A compromised registry

can lead to contaminated containers, intellectual property theft, and many other risks. Clinical countermeasures are authentication with a standard directory service and role-based access control (RBAC) that applies the principle of least privilege and separation of duties. Key countermeasures for the container registry include

- Scanning images for vulnerabilities by using the Common Vulnerabilities and Exploits (CVE) database
- Signing images as known and trusted by using a notary
- Setting up secure, encrypted channels for connecting to the registry
- Authenticating users and controlling access by using existing enterprise accounts managed in a standard directory
- Tightly controlling access to the registry by using the principles of least privilege and separation of duties
- Enacting policies that let users consume only those images that meet your organization's thresholds for vulnerabilities

» **Container runtime:** A containerized application with unrestricted outbound network access can let an attacker who has gained control of a container scan for other weaknesses to exploit. And one of those weaknesses can reside in an insecure runtime configuration, especially if a container is run in privileged mode or allowed to mount sensitive directories on the host. In testing an application, developers can unleash unscanned images or misconfigured containers that turn into rogue containers, extending an open invitation to attackers, especially when such containers persist beyond a short test cycle. Finally, there is the possibility of an attacker escaping from one container and jumping over to another. It's important to keep the container runtime itself (typically, Docker) patched and up to date to avoid the possibility of a container escape. Key countermeasures for the container runtime include

- Restricting outbound network access of containers
- Running images as nonprivileged, immutable containers without SSH
- Keeping the container runtime patched and up to date

- **Container host:** The container host is typically a Linux operating system that runs the Docker daemon. Without carefully selecting the right container host operating system and then taking pains to reduce its attack surface, you could give attackers a foothold into your environment. Because containers share the Linux kernel, a hacker who escapes from one container can get into another container and continue hacking away at your environment. One point of attack might be package vulnerabilities in the operating system. Another attack vector materializes when you manage a container through access rights granted to the underlying container host, creating unnecessary risk. A best practice is to manage containers through the orchestration system, not the container host. To add defense in depth, another best practice is to only group containers on the same host if they have the same sensitivity level and threat posture. Grouping containers by sensitivity level and threat posture limits the attack surface and increases your likelihood of detecting malfeasance. Containers should also not require changes to basic aspects of the host operating system, such as the `/etc` or `/boot` directories, because such changes could potentially let an attacker elevate privileges, access other containers running on the host, or attack the host itself. Key countermeasures for the container host include
  - Using a container-optimized operating system to minimize the attack surface, reduce vulnerabilities, and add kernel-level security
  - Managing containers through the orchestration engine
  - Limiting, logging, and auditing host operating system access to detect anomalies and privileged ops
  - Adding defense in depth by only running containerized applications with the same sensitivity level, purpose, and threat posture on the same host
  - Avoiding mounting sensitive directories on the host
  - Setting the root file system to read-only
  - Keeping the operating system fully patched and up to date

Orchestration platform

The multitenant context of orchestration systems like Kubernetes spells trouble if administrators and users have unbounded access — a malicious or just plain clueless user could sabotage container operations. In a worst-case scenario, a compromised privileged account could expose the entire system. Unauthorized access to the orchestrator can undermine not only containers but also their storage volumes.

Risk also underlies internode traffic, especially when it isn't monitored or when different applications share the same virtual network. A shared virtual network heightens risk because an attacker can use the network to attack the different applications using it and possibly gain access to the most sensitive among them.

And that's why you shouldn't mix the workloads of applications with different sensitivity levels and threat postures. Placing them on the same host, for example, to use readily available resources increases the sensitive application's risk of compromise.

Kubernetes includes powerful core data types for specifying network access controls between pods. Network policy in Kubernetes can limit inbound traffic to a pod based on the source pod's namespace and labels, plus the IP address for traffic that originates outside your cluster. Network policy can also limit outbound traffic with the same set of selectors. A good starting point is to restrict ingress to your application namespace by default, as described in the Kubernetes Network Policy documentation. The enforcement of network policy relies on your container network interface (CNI) provider. Not all CNI providers implement these controls — check your CNI provider's documentation.

Unsecured access to a Kubernetes Dashboard is a surefire way to create your own security nightmare. A recent analysis by a security consulting firm found more than 20,000 unsecured administrative dashboards for orchestration systems on the Internet. At a few big companies, hackers were stealing cloud compute resources through the dashboards to mine cryptocurrency at utilization levels so low that the hijacking was difficult to detect.

Key countermeasures for Kubernetes include

- Authenticating user access by using your standard enterprise directory service, not an ad-hoc directory service dedicated to the orchestrator
- Allowing only authorized hosts to join a cluster
- Imposing RBAC to limit user and administrator access to clusters, containers, and hosts
- Using RBAC to apply the principle of least privilege and separation of duties
- Isolating containers on separate hosts based on the sensitivity level of the applications running in them
- Monitoring health
- Logging and monitoring user access and their actions
- Logging and monitoring resource consumption of containers to ensure availability of critical resources
- Segmenting clusters by tenant
- Isolating orchestrator traffic from workload traffic
- Using logical switches and routers to isolate namespaces
- Segmenting pods by applying dynamic security groups and network policies
- Limiting the interaction of pods by using micro-segmentation
- Managing the life cycle of the orchestration system and its components to keep nodes patched
- Automatically repairing and repaving nodes when problems occur
- Encrypting stored data

Infrastructure

Finally, there is the infrastructure underlying the orchestration system. The infrastructure needs to be locked down, managed, controlled, and monitored just like infrastructure for anything else. Unprotected data in transit and in storage (at rest) heightens the risk of a breach. Lack of logging, monitoring, and visibility — not only for infrastructure itself but also across the entire stack — can make it hard to identify intrusions and respond

quickly. Adequate authentication and access control is key to controlling access to not only Kubernetes itself but also the underlying infrastructure.

Controlling Access

In today's cloud and mobile world, it's often said that identity is the new security perimeter because traditional network perimeters have all but disappeared. So, it makes sense to begin securing your Kubernetes containers by controlling access to them.

Porting identities and access control policies to applications depends on how easily you can integrate your corporate directory service with your container platform. For enterprise environments, that typically means integrating Kubernetes with Active Directory or Lightweight Directory Access Protocol (LDAP) to authenticate users and groups and control access to the platform.

Controlling access to the Kubernetes Dashboard is also critical to the security of your entire container environment.

TIP

Here are some best practices for securing the Kubernetes platform and the Kubernetes Dashboard:

- Secure access to the Dashboard with strong authentication.
- Implement RBAC and limit permissions using the principle of least privilege.
- Require the use of Hypertext Transfer Protocol Secure (HTTPS) and Transport Layer Security (TLS) for all connections to the Dashboard.
- Control access to the Kubernetes API by, for example, using an admission controller.

You should secure access to the Kubernetes Dashboard by requiring authentication. Make sure users need to use their kubectl credentials to access the Kubernetes Dashboard. This requirement prevents unauthorized access to the Kubernetes cluster through a browser.

Finally, controlling access is critical because Kubernetes can direct the orchestration of infrastructure based on configuration information in kubeconfig. This capability is part of what makes Kubernetes so awesome, but at the same time it means that an error can create havoc in the infrastructure. For example, a persistent volume claim can create a 100GB volume by mistake, using the vSphere datastore. The operations team can mitigate this risk in multiple ways, including RBAC, admission controllers, and namespace quota limits.

Monitoring Access

Even with strong identity and access control measures in place, you should proactively monitor the system to track access and remediate vulnerabilities. Here are a few final things to add to your Kubernetes security to-do list:

- Monitor cluster and network utilization.
- Monitor for suspicious activity and analyze failed logging and RBAC events.
- Monitor components and configurations, such as Dashboard access, for risks and vulnerabilities.
- Routinely test for vulnerabilities and attack vectors by using standard tools.
- Keep your system patched and use recent versions of Kubernetes. Recent versions typically have stronger, more mature security controls than older versions do.

IN THIS CHAPTER

- Driving business value
- Hosting cloud-native and traditional applications
- Empowering developers with self-service infrastructure
- Optimizing deployment, operations, and security processes
- Consolidating servers and increasing resource utilization
- Eliminating outages and reducing downtime
- Enabling portability across cloud environments
- Simplifying management and operations

Chapter 7
Ten (or So) Benefits of Kubernetes on vSphere

Here are some great business benefits of running Kubernetes on vSphere.

Evolving Faster to Accommodate Business Needs

Building and deploying containerized applications on VMware infrastructure drives business value. VMware solutions enhance developer productivity, business agility, IT flexibility, and application scalability.

The result helps your business adapt to changes in the marketplace and shorten the time it takes to bring applications to market.

Extending vSphere Environments to Host Traditional and Modern Applications

Running Kubernetes on vSphere prepares your data center for the digital era by letting you provision and manage containers. The solution moves you one step closer to a modernized software-defined data center (SDDC) that delivers infrastructure, services, data, and applications on demand. For a traditional application, however, a common first step toward modernization is repackaging part or all of it in a container.

However, most infrastructure platforms are not designed to run traditional and modern applications side by side while working with existing hardware and software, making it difficult to repackage a traditional application with containers.

Beyond the infrastructure, modern applications pose their own challenges. Modern applications change frequently, are developed in short release cycles, and may be built with microservices. In addition, you may need to connect applications across clouds and devices with security, compliance, and availability.

With Kubernetes on vSphere, you can establish a consistent operational model for infrastructure and application delivery that works with both traditional and containerized applications. The model creates a powerful bridge to move from traditional software development practices to new, more flexible forms geared toward innovation, speed of execution, resource consolidation, and easier maintenance.

Presenting Instant On-Demand Infrastructure to Developers

Docker furnishes a platform for developers to rapidly build applications and then port them to a production environment. A developer working on a traditional Java application running on Apache

Tomcat, for example, can containerize the application and then, because of its inherent portability, shift it to a Kubernetes cluster running on vSphere. Kubernetes makes it easy for a developer to define what an application needs (for example, how many container instances to run and which ports need to be exposed) and then works with vSphere, vSAN, and NSX to implement the corresponding changes for the application in the infrastructure.

Reducing Time Spent Building, Operating, and Securing Infrastructure

Without being able to integrate containers with your existing security systems and your data center, trying to fulfill the security requirements of containerized applications and orchestration systems would lead you to build custom security components or integrations at great risk and expense.

Kubernetes on vSphere helps secure containers throughout the stack by providing a unified policy layer for virtual machines (VMs) and Kubernetes pods. Kubernetes on vSphere:

- Integrates with monitoring and logging tools like Wavefront from VMware
- Works with VMware NSX so you can apply micro-segmentation to secure containerized applications
- Works with Harbor, an open-source container registry that secures images with role-based access control (RBAC), scans them for vulnerabilities, and signs them as trusted

Increasing Server Consolidation and Resource Utilization

Increasing server consolidation and resource utilization is a proven use case for virtualization. With Kubernetes on vSphere, you can further extend this benefit to the thousands of containers and microservices that modern enterprises are building and deploying in the SDDC and cloud today.

Improving Uptime and Mean Time to Repair

If your application is down for ten seconds a day over a period of ten years, there is a chance no one will ever notice. If, however, your application is down for a whole business day, people will almost certainly notice — even if it only happens once every ten years. Yet, over a ten-year period, both of these scenarios achieve the same service-level agreement (SLA) availability metric of 99.99 percent.

A relentless focus on driving down mean time to failure (MTTF) can introduce complexities that may increase mean time to repair (MTTR). Although MTTF can be difficult to quantify, MTTR can be tested and understood, which makes trying to reduce MTTR a smart move.

One of the great things about Kubernetes is that, in most cases, it significantly reduces MTTR for most application-level outages from potentially hours (if operators are involved in recovery operations) to seconds. You can move a container to a new node in a matter of seconds. If a node fails, Kubernetes on vSphere automatically moves it to a new node to maintain the desired state of the application.

Many Kubernetes adopters get excited when they realize that many formerly pageable (that is, a phone call or alert in the middle of the night) events become a nonissue — the cluster detects a failure, restarts the container, and no one is the wiser. You can significantly boost availability for a traditional application by containerizing it, making sure the health-checking model is correct, and deploying it on Kubernetes on vSphere. Moreover, application-level outages usually dominate MTTF. Kubernetes can smooth over outages and isolate your application from the effects of node failures.

Increasing Flexibility with Cloud Portability

Containers and Kubernetes are technologies primed for portability. The packaging of containers decouples applications from machines to let developers decide where and how to deploy an

application. The portability of containers combined with the power of Kubernetes on vSphere gives you the independence to pursue a multi-cloud strategy.

Kubernetes decouples the needs of an application from specific infrastructure via its resource application programming interface (API). For example, a pod's load-balancing needs are defined with a service definition. Kubernetes will implement the service the same way, no matter what underlying infrastructure it's operating on, as long as the infrastructure supports container network interface (CNI).

The Kubernetes ecosystem is always growing. Kubernetes supports all major infrastructure providers. This makes it easy to define an application when you're using Kubernetes constructs and use them to deploy and manage an application consistently on any provider.

Kubernetes on vSphere further supports portability by exposing Kubernetes in its native form without proprietary extensions and by implementing a common operating model. Developers use the native Kubernetes command-line interface (CLI) and API.

Supporting Any Application While Simplifying Management and Operations

Simplicity is at the heart of running Kubernetes on vSphere. The Kubernetes control plane gives you a self-service API interface to deploy Kubernetes clusters on demand on virtualized infrastructure. The API of the control plane lets you automate the creation and deletion of Kubernetes clusters.

In addition, the simplicity of Kubernetes on vSphere combines with the inherent simplicity of containers to ease the maintenance and updating of containerized applications and the Kubernetes clusters in which they run.

Run Kubernetes on your existing infrastructure

Enterprises are increasingly defined by the way they develop software. It's how they engage customers and compete in the marketplace. To speed up development, organizations are adopting Kubernetes. The good news is that you can start with Kubernetes right now by using familiar tools and your existing infrastructure to manage traditional applications and cloud-native applications side by side. That's the power of running Kubernetes on vSphere. This book will help you realize that potential.

Inside...

- Optimize your data center
- Compare Kubernetes and vSphere constructs
- Understand the world before and after Kubernetes
- Automate and optimize container operations
- Ensure security, compliance, and consistent operations
- Build flexibility and portability into your foundation

vmware

Boskey Savla is Technical Marketing Manager at VMware focusing on Cloud Native Applications. She has 15 years of experience in Linux System Engineering, Infrastructure Automation, and Dev-ops. **Steve Hoenisch** is a writer and editor at VMware who has written numerous influential white papers on emerging technology, Kubernetes, and security.

ISBN: 978-1-119-85397-8
Not For Resale

for dummies® A Wiley Brand

Also available as an e-book

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.