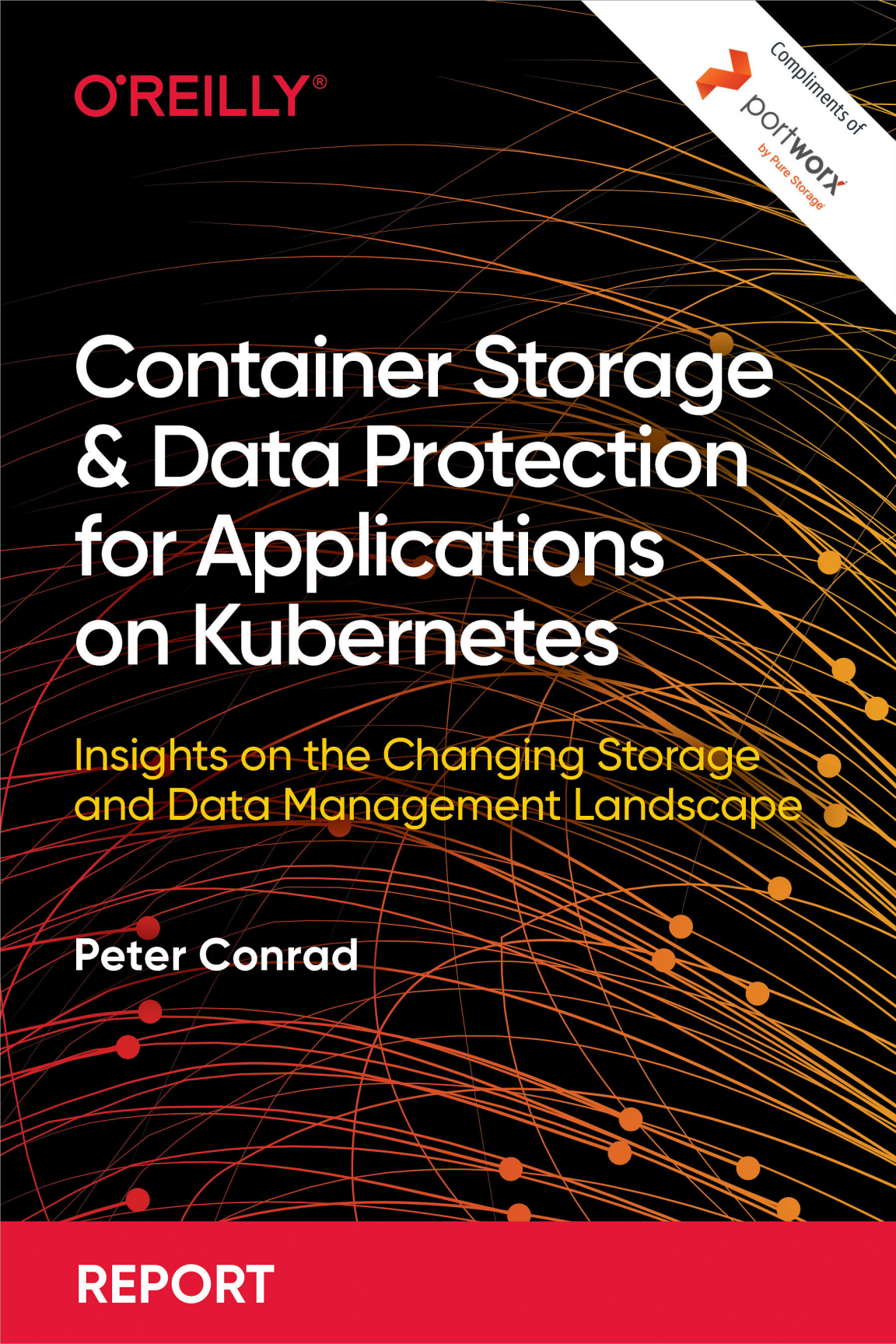
O'REILLY®
Compliments of
portworx
by Pure Storage
Container Storage & Data Protection for Applications on Kubernetes
Insights on the Changing Storage and Data Management Landscape
Peter Conrad
REPORT

portworx
by Pure Storage

Uncomplicate
Data on Kubernetes

Make your data services scalable, available and secure on Kubernetes

Get Started Today

Container Storage and Data Protection for Applications on Kubernetes

Insights on the Changing Storage and Data Management Landscape

Peter Conrad

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Container Storage and Data Protection for Applications on Kubernetes
by Peter Conrad

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: John Devins
Development Editor: Michele Cronin
Production Editor: Kate Galloway
Copyeditor: Audrey Doyle
Proofreader: Nidhi Ranjalkar
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Kate Dullea

March 2022: First Edition

Revision History for the First Edition
2022-04-05: First Release

This work is part of a collaboration between O'Reilly and Portworx. See our statement of editorial independence.

978-1-098-12699-5

[LSI]

Table of Contents

Preface

This report is for DevOps engineers, platform architects, site reliability teams, and application owners with a vested interest in managing storage for Kubernetes applications and services. The following pages provide a discussion of the challenges of data storage for distributed applications at scale, how to solve them, and considerations for moving to Kubernetes.

The first chapter outlines the reasons why adopting Kubernetes in an enterprise production environment is complex, focusing on how traditional storage fails to meet the needs of modern cloud native applications. A discussion of Kubernetes storage primitives follows, including the Container Storage Interface and a brief explanation of how software-defined storage can support Kubernetes workloads. Following this foundation, the remaining chapters enumerate enterprise data needs and strategies for meeting them with Kubernetes storage. Topics include topology strategies for scale and efficiency, security, and the challenges of data protection.

Read on to learn how Kubernetes changes the storage and data management landscape, the essential elements of a Kubernetes-native storage solution, and best practices for running stateful enterprise applications on Kubernetes.

Acknowledgments

I would like to thank the following people at Portworx for contributing their expertise: Sarvesh Jagannivas, Bhavin Shah, Rajiv Thakkar, Ryan Wallner, and Andy Gower.

Introduction

Cloud native application architecture relies on Kubernetes, the leading container management platform. The enterprise needs a storage solution that can keep up with the scale and volume of data processed by these applications. On-premises and cloud-based storage solutions, which were designed for traditional monolithic applications running on virtual machines (VMs), weren't designed to run in cloud native environments and aren't flexible enough to meet the requirements of the enterprise today.

Modern applications are implemented as discrete services that can be packaged and deployed independently in their own containers running on separate machines. Cloud native infrastructure is *elastic*, meaning it is controlled programmatically or declaratively to proactively meet changing compute needs. This design enables applications to quickly scale up and down based on demand, but requires nimble distributed data storage that scales along with the application. Traditional storage is simply not up to the task. To meet the needs of the enterprise, Kubernetes storage must be dynamic, automated, and application aware, and it must provide data protection in the form of intelligent backup and recovery at the container level.

As application architecture becomes distributed, ownership of application components becomes more dispersed. That's one reason why storage policy, traditionally the domain of dedicated administrators and managers, is becoming a shared responsibility among teams including the application owners themselves. As a wider group takes on these tasks, the burden of managing the complexity of provisioning and maintaining storage must shift to automation.

Traditional data platforms, which focus on the VM as the unit of compute and storage, can't effectively manage or protect storage for distributed applications. What's needed is a way to bring to data storage the dynamic, distributed approach that Kubernetes brought to compute. This report provides a foundation for understanding a storage paradigm that makes Kubernetes data highly available and secure, providing dynamic scale and data protection.

CHAPTER 1

Why Kubernetes in Production Is Hard

Kubernetes is the foundation for the modern cloud native application architecture. By running microservices in containers, the enterprise can move from a hardware-defined model, tied to physical or virtual machines, to a software-defined paradigm that enables horizontal scale across on-premises, cloud, and hybrid environments.

When moving to Kubernetes, day one is not as straightforward as simply deploying software. While the number of people who are well versed in Kubernetes operations is growing, there are still challenges. The following sections focus on three of the most significant challenges: automating storage, maintaining a cluster, and managing storage designed for a different computing era.

Automating Storage

Kubernetes was originally designed to handle *stateless* workloads, which don't store data or other information about previous operations. While Kubernetes has evolved to handle *stateful* applications, which store persistent data, it has not grown to automate storage the same way it handles compute resources. Instead, Kubernetes relies on storage that sits outside the orchestration system.

Automating storage is not easy, and storage administrators often have plenty of work to do. Provisioning storage as workloads appear and disappear, or as applications scale up and down, can be time-consuming operations even when assisted by automation. Configuring systems to add, change, and remove resources in response to thresholds and events is a complex job.

While Kubernetes load-balances services and requests within the cluster, it doesn't really know how to load-balance storage. Different workloads, applications, and use cases have different storage usage patterns. Storage administrators, responding dynamically to changing storage needs, must constantly consider when and where to scale out storage, how much and where to grow volumes, and how to load-balance storage requests across the cluster. Without integrated tools, it can be very difficult to automate these tasks sufficiently to keep applications running smoothly.

In a modern DevOps software development environment, automation is key. For the enterprise to adopt Kubernetes, it must be possible to automatically scale, balance, and protect data as demand changes. For the storage administrator, these are often high-touch efforts. Managing storage capacity can be time-consuming and disruptive to running applications. To keep efforts focused on application development means reducing this overhead through automation as much as possible. This requires storage platforms and tools that are aware of each application's API and storage requirements, and are capable of managing storage automatically in ways analogous to how Kubernetes handles compute resources.

Maintaining a Kubernetes Cluster

Kubernetes abstracts compute infrastructure for distributed cloud native applications, decoupling them from the hardware where they run. Hardware health, failure, and other concerns no longer bear directly on the performance of the software. When hardware fails, the containers that contain the software move automatically and the application keeps running.

However, the underlying hardware that supports the cluster requires continual monitoring to ensure availability to applications and their users. While Kubernetes manages and moves containers in response to changing conditions and demands, any enterprise with an on-premises deployment in its portfolio must maintain physical

servers to keep the cluster running. The IT team must ensure that it can support the service level agreements (SLAs) that the business requires. This involves replacing hard drives and other components as they fail, which is a very common occurrence at scale.

This maintenance often requires migrating large pools of replicated data from one storage infrastructure to another, which is complicated by the different needs of the various applications that use the storage. It's nearly impossible to achieve a significant manual migration without downtime and without losing any application's operating state. An intelligent storage platform can help, easing the task of protecting and migrating data as hardware is provisioned and managed, or as applications move to new environments.

Storage for Another Era

Traditional applications, which run directly on physical or virtual machines, have simpler storage needs than distributed, containerized applications. Monolithic application state is often stored in a shared, mutable table that is straightforward to back up. In fact, some applications running on VMs rely only on local storage. In such cases, backing up the VM is sufficient to back up not only the application's data but its configuration and running state as well.

Containerized applications are different. Because containers are immutable and ephemeral, any workloads that require persistent data must connect to an external system. Backing up a container is not sufficient to capture persistent data, since the data doesn't reside in the container itself. Instead, Kubernetes provides a mechanism for associating a unit of persistent storage with a *pod*, one or more containers running on a single host node and sharing resources.

Kubernetes typically maintains a set number of replicas of every active pod to ensure application availability. Each pod can define one or more local or remote *volumes*, or cohesive units of persistent storage space. Volumes provide the capability to persist data beyond the lifetime of the containers or the pod itself. Multiple containers in a pod can mount and access the same volume, so applications can share data between containers that have different tasks. For example, an init container can run before the service starts, creating a custom configuration file for the environment where the application is running.

Initially, extending the capabilities of Kubernetes storage meant changing the Kubernetes codebase itself. Kubernetes has deprecated this "in-tree" approach, replacing it with an API called the Container Storage Interface (CSI), which enables the development of storage plug-ins without changing any Kubernetes code. The CSI provides a way to support multiple storage interfaces that allow containerized workloads to store persistent data on different types of externally managed storage pools (Figure 1-1).

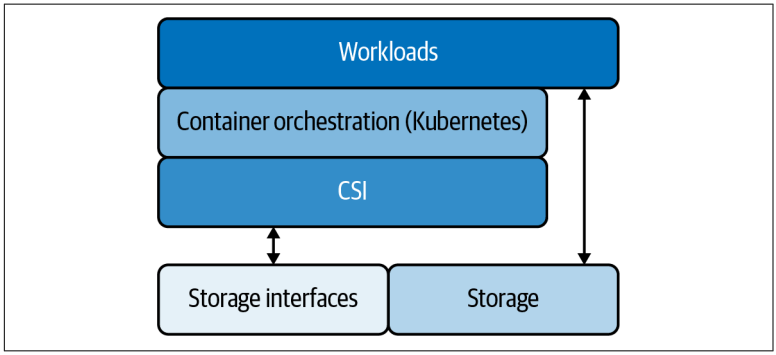

Figure 1-1. Kubernetes storage with the Container Storage Interface

Although this model solves some problems, it fails to bring storage into the virtualized, software-defined infrastructure where containerized applications run. While the compute resources used by modern enterprise applications run on application-aware infrastructure, provisioned and managed declaratively and driven by the end user, storage remains a physical concept tied to virtual or physical servers. Because cloud native applications are distributed, backing up any given VM is likely to capture partial data from multiple applications, while failing to store complete data from any single application.

For this reason, on-premises and cloud-based storage systems designed for physical or virtual machines are poorly suited to the scale and complexity of containerized applications. While containers are software defined, disposable, replaceable, and decoupled from hardware, traditional storage is concerned with managing pools of physical media. Containerized applications are highly dynamic, scaling up and down rapidly as demand changes by creating, destroying, and moving containers automatically. Traditional storage methods don't respond quickly enough to support these modern architectures.

For file, block, and object storage, cloud native storage solutions emerged to orchestrate a software-defined pool of persistent storage for access across a Kubernetes cluster, using a containerized architecture to provide dynamic storage at scale. These solutions work well, but they often target specific types of storage, a few filesystems, and a selection of database services.

The enterprise needs software-defined, general-purpose storage that's capable of scaling up to meet the demands of big data: streaming, batch processing, transactional databases, high availability and disaster recovery, data locality, and data mobility. To support fast recovery time objectives, backups must incorporate all application state, including data and configuration.

Specific application types have additional, more stringent needs. For example, database applications need to guarantee that data transactions are ACID (atomic, consistent, isolated, and durable) and usually require secondary indexes. Because pods are ephemeral and movable, it is difficult to meet these guarantees in a containerized environment.

To make Kubernetes work for the enterprise, storage solutions must work at the container level rather than the VM level, must be aware of Kubernetes namespaces, and must be able to back up entire applications across VMs, including their configuration and state.

CHAPTER 2

Enterprise Storage for Kubernetes

To make Kubernetes storage ready for the enterprise, one must solve the problems of elasticity and scale to meet the velocity and volume of big data. Cloud native storage solutions have emerged for orchestrating pools of persistent storage that are *software defined*, meaning that they are abstracted away from the underlying hardware. Unlike software-defined storage (SDS) solutions for VM-based environments, cloud native software-defined storage runs natively as containers that can be managed by the same orchestration system that handles application containers. This paves the way for dynamic, elastic storage for containerized applications running on Kubernetes at scale. This chapter discusses Kubernetes storage concepts, how software-defined storage brings scale to Kubernetes, and how the CSI works with software-defined storage.

Kubernetes Storage Concepts

Kubernetes represents application entities as *primitives*, which are the basic Kubernetes building blocks. Primitives represent real or logical entities so that Kubernetes can manage them as if they were software objects. Storage is no exception. Kubernetes provides a number of storage primitives, including the following:

PersistentVolume (PV)
: A unit of persistent storage

PersistentVolumeClaim (PVC)
: A storage request, which becomes the binding between a PV and a pod

StatefulSet
: An object that manages the identity of a set of pods

StorageClass
: Describes the classes of storage the cluster offers

There are many other primitives, of course, but these four are interesting as we consider how to bring enterprise-ready scale to Kubernetes storage.

PersistentVolume

A PV is an object that represents storage at a specific location. PVs provide the ability to keep data longer than the lifetime of the workload or pod that uses the volume. PVs can store a workload's data on networked storage, on a cloud provider, or locally on the pod where the workload is running (Figure 2-1).

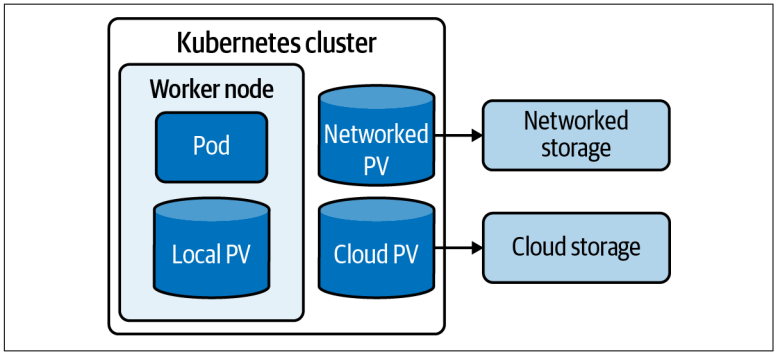

Figure 2-1. Local, cloud, and networked PersistentVolume locations

Kubernetes manages every PV's lifecycle, defining the following stages:

Provisioning
: A storage administrator creates the PV statically ahead of time, or dynamically using a *StorageClass*, which is a resource that provides a way for the administrator to describe the storage.

Binding
: A PVC binds the PV to a specific container.

Use
: Workloads running on the container use the PVC to access the PV.

Release
: The container removes its claim to the volume by removing the PVC.

Retention
: While the data is needed, the PV retains it, even across container and pod lifecycles.

Deletion and reclamation
: Kubernetes deletes the data when it is no longer needed, reclaiming the storage space for use by other volumes.

The storage administrator can provision PVs dynamically as needed, or ahead of time based on predicted storage needs. PVs define additional details about the data, including lifecycle policy, routes, IP addresses, and credentials.

PersistentVolumeClaim

A PVC is both a request for storage and an identifier that establishes a claim on the storage once it's granted. PVs on their own are not owned by specific applications or projects. A PVC requests access to a PV using one of the following access modes:

ReadWriteOnce (RWO)
: Read-write access by all pods on a single node

ReadOnlyMany (ROX)
: Read-only access by multiple nodes

ReadWriteMany (RWX)
: Read-write access by multiple nodes

ReadWriteOncePod (RWOP)
: Read-write access by a single pod only

PVs and PVCs work together as follows:

1. An application developer creates one or more PVCs describing the storage resources the application needs.
2. The storage administrator can either create PVs explicitly in response, or create a StorageClass that can dynamically provision new PVs as needed.
3. Kubernetes manages the binding of PVCs to PVs.

Together, PVs and PVCs provide a way for pods to define requests based on the storage requirements of their containers and applications. The storage administrator can either configure dynamic *provisioners* that allocate storage and a PV in response to these requests, or create PVs in anticipation of an application's storage needs. When the storage is granted, the cluster finds the PV associated with the PVC and mounts it for the pod. In other words, the pod uses the PVC as a volume. The PV is exclusively available to the pod as long as it's needed.

StatefulSet

A StatefulSet is the primitive that manages the deployment and scaling of a set of pods, maintaining a unique ID for each pod so that it can be identified for the purposes of persistent data or networking, or when the pod migrates to a different node. By providing unique, persistent IDs for pods, the StatefulSet API lets administrators manage the deployment and scaling of a set of pods. When individual pods fail, the persistent IDs help restore connections between their replacements and the existing volumes that serve them.

StorageClass

The StorageClass primitive lets the cluster administrator describe the different classes of storage the cluster offers. Storage classes can indicate different policies or levels of service. For example, the administrator might set up different StorageClass objects to represent different backup policies. Users can request a specific storage class by name.

Software-Defined Storage

Just as containerized application architecture decouples application logic from the hardware where the code runs, software-defined storage decouples persistent data and storage policies from storage media. SDS is distributed and hardware agnostic, and can run in a variety of environments including the cloud. By abstracting storage from its hardware, SDS enables the presentation of hardware capacity to applications, users, and other clients as a unified storage pool. SDS makes a storage administrator's job easier immediately by making it possible to manage a diverse assortment of different storage types consistently without worrying about the different characteristics of each type of storage (Figure 2-2).

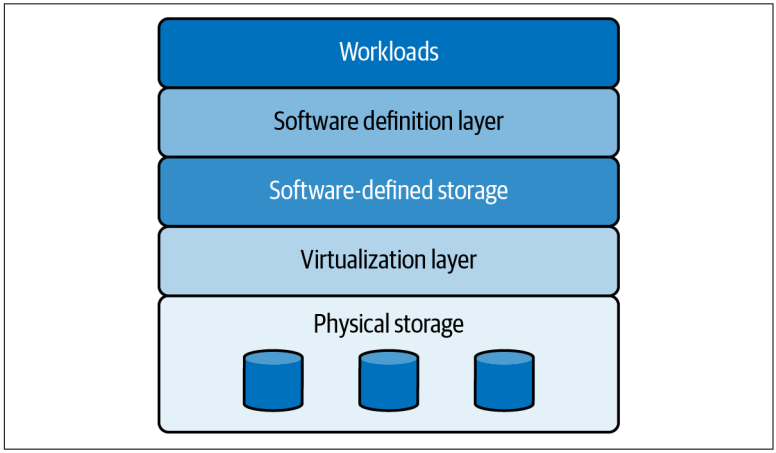

Figure 2-2. Software-defined storage

Just as Kubernetes brings elasticity, scale, and high availability to compute resources, SDS is the foundation for bringing these important characteristics to storage. Because the storage is abstracted, it's possible to build systems that distribute and scale storage in different environments, integrating them with Kubernetes or other orchestration systems and building in fault tolerance and high availability.

The advantages for the enterprise are clear:

- Developers and users don't need to think about storage hardware.
- Scale becomes a matter of on-demand provisioning.

- Data can be placed anywhere it is needed, regardless of the environment.

Because SDS presents all available physical storage as a shared pool, the resources can be allocated efficiently, reducing wasted storage space.

Connecting SDS to Kubernetes with the CSI

The CSI is an interface between containerized workloads and a third-party storage layer, making it possible for cloud native applications to create, manage, and use storage outside of Kubernetes. The CSI makes storage available in a pool that all instances of each application can access, keeping instances in sync and making it possible to back up applications consistently. The CSI provides abstracted access to Kubernetes over an interface, meaning that third parties can create plug-ins for accessing storage systems from Kubernetes without touching the core Kubernetes code. Using the CSI, containerized applications can use the normal Kubernetes storage primitives on top of these storage systems (Figure 2-3).

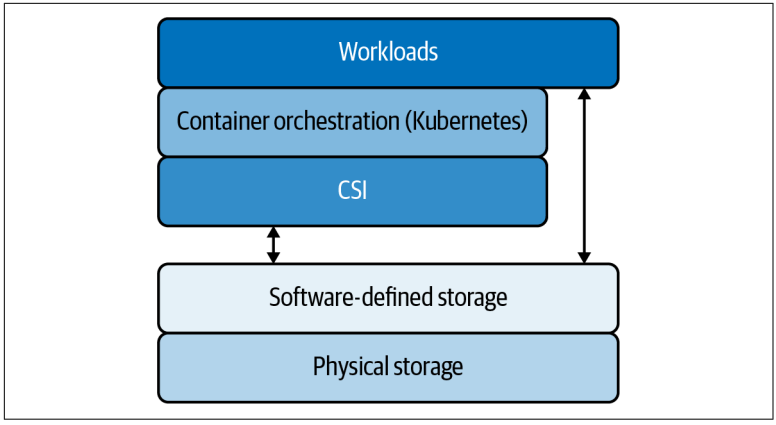

Figure 2-3. Connecting software-defined storage to Kubernetes with the container storage interface

The CSI gives storage vendors the freedom to design and manage storage as they see fit, providing Kubernetes and other orchestration platforms the freedom to provision and manage storage transparently using native abstractions. At the time of this writing, Kubernetes and the CSI don't know how to provide application-aware

backups, high availability, or other functions necessary for the enterprise, but the CSI makes it possible to add these capabilities at the storage layer itself. This is where SDS comes in.

Software-designed storage is not limited by hardware or standards, meaning that any required capabilities can be implemented and abstracted away from the underlying mechanics. For this reason, it doesn't matter that the CSI is a slow-moving standard, as standards should be. By connecting SDS to Kubernetes, the CSI makes it possible to build Kubernetes-ready storage that is granular at the container level, self-healing, and topology aware, without requiring Kubernetes itself to have these capabilities.

Cloud Native Storage: Bringing Scale to Kubernetes Storage

Adapting traditional SDS for containerized workloads is challenging for a number of reasons.

First, because containers are dynamic and ephemeral, they require storage that can be provisioned, attached, and deleted instantly, whenever and wherever it is needed. For high availability, it must be possible to create and move volumes as needed, with awareness of topology, and to back them up regularly and automatically. Manual storage provisioning can't keep up with these needs at scale.

Second, the number of volumes that physical and virtual servers can support is often insufficient for the number of pods or containers that need storage. A single host might run hundreds of small containers, requiring more volumes than the OS can provide.

Finally, because the point of containers is to be infrastructure agnostic, they should not care about the physical storage they're using. Different environments provide different types of storage, often multiple types within a single deployment. It must be possible to move data between storage pools without affecting the way containerized applications run.

Enter cloud native storage, a new model of SDS designed for distributed, containerized applications. Cloud native storage runs in containers on the cluster, meaning it can be provisioned and orchestrated, offering data locality and Kubernetes-integrated features like Stork (Storage Operator Runtime for Kubernetes) to provide

storage-aware scheduling. Figure 2-4 shows how cloud native storage becomes part of Kubernetes itself.

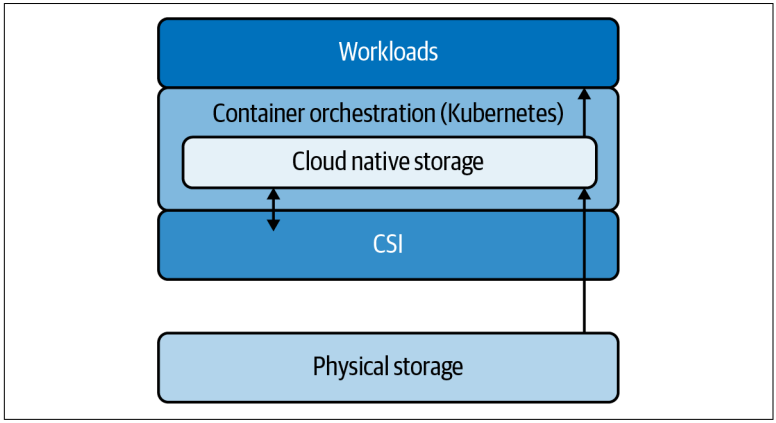

Figure 2-4. Cloud native storage

Cloud native storage must be container aware, and all operations must take place at the level of the application. Snapshots, backups, compression, encryption, and other operations are not relevant to the cluster or the storage pool as a whole, but to the container itself. This key aspect gives operational control over data to the application owner, relieving IT admins of the responsibility for provisioning storage and protecting application data.

Clearly, redefining storage for Kubernetes at scale requires rethinking the nature of storage itself. To meet the needs of containerized applications at scale, a storage paradigm must be elastic, nimble, able to serve multiple replicas of data to multiple instances of application services concurrently, and decoupled from the application logic itself. In other words, to bring scale to Kubernetes data, storage must be cloud native, which means it must be both software defined and containerized.

CHAPTER 3

Kubernetes Performance and Security

As Kubernetes deployments become larger and more prevalent, the enterprise is turning its focus to performance and security, which are important for both business continuity and cost control. Performance helps save cost by maximizing infrastructure usage and preventing unnecessary resource scaling. Security helps control cost and ensure business continuity by protecting assets, including customer data and other proprietary information.

What Hasn't Changed: Performance, Availability, and Security Requirements

As applications have evolved from monolithic on-premises deployments, to virtual machines, to software as a service (SaaS) and platform as a service (PaaS) offerings running in the cloud, to modern containerized cloud native applications, business requirements haven't changed. Enterprise applications running on Kubernetes must meet nonnegotiable business requirements such as high availability, data protection, and strict performance metrics, all while operating across hybrid clouds or multiple data centers.

Performance requires using resources appropriately, ensuring that there is enough capability to serve the needs of the business without overprovisioning or overspending. Availability involves taking hardware failures and network latency in stride, and meeting

SLAs without blinking an eye. Security means protecting sensitive resources and data through encryption, access controls, and defense in depth while making them available to people and processes with a legitimate business case for access.

Traditional availability mostly meant uptime monitoring and alerts. Making sure the application frontend was running was not only sufficient to identify downtime, it was often the only available sign of a problem. Fortunately, monolithic software was comparatively simple, with predictable user interactions. End-to-end request monitoring was unimportant, and might have seemed absurd to IT teams at the time. Performance was less of a concern; as long as applications were up and behaving properly, IT and site reliability teams were happy.

Performance, availability, and security must now be managed at the container level.

The services that make up today's containerized applications are highly interdependent, meaning that the failure of one can take down part or all of a highly complex, mission-critical application. Keeping applications running properly involves maintaining multiple instances of critical services, load-balancing among them, and moving or restarting them when failures occur.

High Availability at a Global Scale

Global high availability is a nonnegotiable business requirement for the enterprise. This must go beyond keeping individual clusters running when a few control plane nodes or worker nodes fail. Application state, including configuration and distributed data, must be available without interruption everywhere it's expected. The bottom line is that cloud native high availability requires awareness of each application's persistent data, metadata, configuration, and running state.

High availability within a cluster can be as simple as replicating containers, volumes, and Kubernetes services so that at least a minimum number of instances of each is available continually. The goal is to ensure that there is no single point of failure. High availability across local or geographical areas involves replicating data and applications among separate data centers, either synchronously or asynchronously. These strategies permit the enterprise to choose

strategies that balance the cost of hardware or cloud resources against the goals of the business.

The key to high availability, whether local or global, is that the storage and data replication mechanisms are application consistent, meaning that they capture all the data and metadata necessary to keep services running or to restart them without interruption to the business flow. This includes not only persistent data but application configuration and running state as well. Simply backing up VMs or making copies of volumes is not enough, because they only capture part of the data any given service or application needs to continue running. Furthermore, because individual volumes and containers aren't guaranteed to last, there must be a way to identify applications and data so that the replacement instances can be associated with each other correctly.

Data Mobility

One key to high availability is *data mobility*, defined as the ability to move or replicate data quickly between clusters in a single data center or cloud, or between clouds. This capability supports not only high availability, but also upgrades, migrations, scaling, and disaster recovery.

As the enterprise embraces the cloud, the importance of data mobility becomes more and more apparent. It's clear that a single deployment environment or provider is no longer sufficient for all business needs. Organizations need the flexibility to make trade-offs between public and private clouds, on-premises servers, and edge data centers as needed to increase agility, meet regulatory requirements for data locality, deliver better service, and keep costs under control.

As Kubernetes comes into play, it's crucial to be able to move large volumes of data among diverse multicloud and hybrid cloud environments. While Kubernetes makes it easy to deploy applications anywhere, it has long been more difficult to move the underlying data. What is needed is an approach that makes it as easy to move persistent volumes as it is to migrate application containers.

Container-native storage is half the battle. By providing data and storage management across infrastructure types and providers, a container-native storage pool provides a home for data in any deployment environment. But traditional data migration and

replication approaches, which are time-consuming and operationally complex, aren't up to the task of migrating data when required. What's needed is a container-aware data orchestration layer that can provide the declarative automation for state that Kubernetes provides for applications themselves. In cases where an application moves, it's often not practical to move all application data on short notice. Data mobility must include the capability of moving the most immediately important data first, to allow the application to start working in its new location quickly by minimizing the amount of data that must be moved.

Data mobility enables the enterprise to improve operations in a number of ways. By moving low-priority applications to auxiliary clusters, the enterprise can free up capacity on critical clusters, making room for additional data or replication. Automating the movement of data makes it easier to test new versions, promoting workloads from development to staging clusters or maintaining two or more live environments. When moving applications and data among environments or taking a cluster offline to perform hardware maintenance and upgrades, data mobility is key to continuous availability.

Tuning Kubernetes Data for Enterprise-Scale Performance

As the enterprise drives cloud native applications to greater and greater scale, tuning Kubernetes data is increasingly important to balance performance, cost, and availability. Although there are many factors that affect the performance of a Kubernetes cluster, one of the most important is to make the right choices about data. There are two significant considerations: overall storage configuration and data placement.

Storage configuration means determining what kind of storage hardware you need and how you set it up. You can set up a single storage pool or multiple pools that each define their own StorageClass, for example. You'll need to think about cost versus capacity and speed, determine how much memory and CPU to allow for use by the storage layer, and how many different types of storage you will need. These questions are especially important in on-premises data center environments, where the decisions have ramifications

that can last for years. In cloud or hybrid environments, there is more flexibility to start small and grow or change as needed.

Data placement, or *data topology*, refers to strategies for managing where data is stored, usually to meet either performance or availability goals. For maximum performance, one strategy is *hyperconvergence*, which keeps a workload and its data together in a single node (Figure 3-1). If the node goes down, the data is lost, but that might be acceptable.

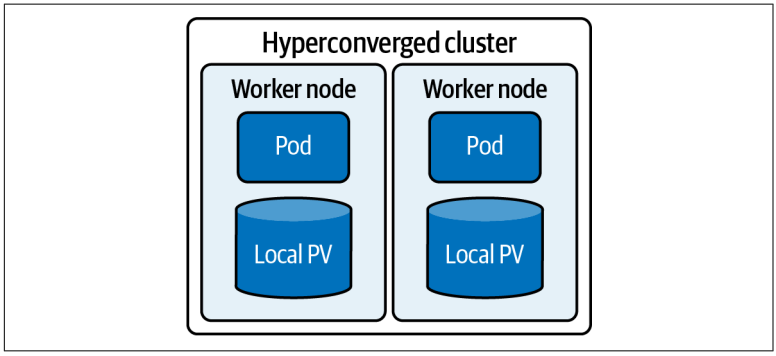

Figure 3-1. Hyperconverged topology

In hyperconverged topologies, storage is usually a software-defined layer, often on commodity servers shared by the compute nodes. Hyperconvergence is very flexible, allowing storage to scale in step with the compute workload. However, because the failure of a single node affects not only the workload on that server but the storage system as well, hyperconverged topologies can be more difficult to manage from the point of view of both operations and security. Storing additional replicas of the data on other nodes allows a new instance of the workload to recover to a true hyperconverged topology after a failure, starting from the most recent data written to the replicas.

For higher availability, data can be separated from workloads. Keeping compute and storage nodes separate (called *disaggregation*) provides slower performance but prevents loss in the event of a node failure. For a very dynamic environment, where the number of compute nodes increases and decreases in response to workload demand, one strategy is to separate compute and storage into their own clusters (Figure 3-2). This way, scaling and management

operations in the compute cluster don't interfere with the storage cluster, and vice versa.

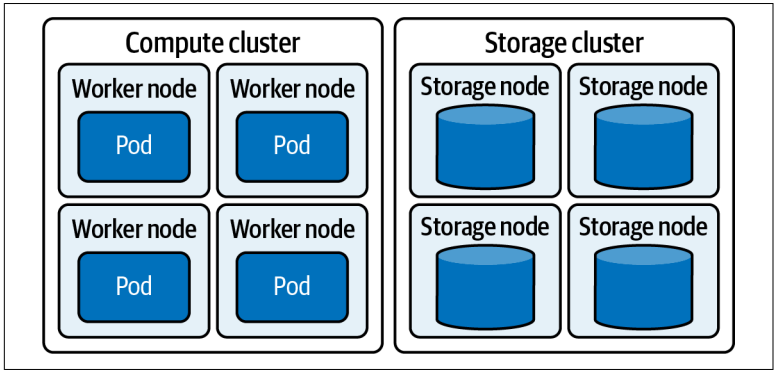

Figure 3-2. Disaggregated topology

Keeping the Cluster Secure

Modern security practices use a *defense in depth* strategy, which means that multiple layers of security controls work together, providing redundancy so that a breach in one layer doesn't grant access to a critical system. Cloud native security is no different: protections at the levels of the cloud, the cluster, the container, and the code itself build upon each other to protect applications and data from unauthorized access.

The environment is the first layer of defense. If the data center or cloud is secure, it is easier to protect the cluster and its services. The cluster, in turn, must provide security mechanisms at the level of the physical node, the VM, the container, and so on.

At the container level, services must be able to communicate securely with SDS. Kubernetes provides mechanisms for controlling the access privileges of containers, limiting resource usage, and preventing containers from taking dangerous or unwanted actions. The storage layer must do its part by encrypting data both at rest and in motion, restricting access to specific clients, and using application and API knowledge to permit only sensible data transactions.

CHAPTER 4

Data Protection for Kubernetes

Data protection encompasses a broad array of practices and concepts including high availability, backup, disaster recovery, and other processes that support business continuity. Every enterprise maintains and tests data protection policies and programs to minimize downtime and ensure that operations can continue after a disruption. Over the past several years, data protection has also become an important component of compliance. In addition, data protection strategies must provide data privacy as regulations begin to address the substantial amount of sensitive personal data that companies handle from day to day.

Kubernetes Data Protection Challenges

Traditional data protection was focused at the level of a physical or virtual machine, protecting applications and data by securing the server itself. This approach is effective for applications that run on a single host. For containerized applications, however, protection at the server level is insufficiently granular. Targeting the entire server makes it impossible to separate applications, storage, and configuration, and commingles applications that require different data protection policies.

Only by providing data protection at the container level is it possible to apply policies by application, container, or individual unit of Kubernetes storage. In a Kubernetes environment, data protection must be available at the Kubernetes resource level and must include authentication and authorization for distributed data layers

throughout the cluster. Backup and disaster recovery must be integrated into the container orchestration environment itself.

Because each application is complex, and because of the large number of applications running on an enterprise cluster, manual backup strategies are impractical. The only way forward is automation; but here, too, complexity reigns.

Scale

Containerized applications are all about scale, including large amounts of data that must be processed at high velocity. Data at scale requires backup solutions that can gather data from a wide array of sources in diverse environments, managing and aggregating them efficiently. These needs include the ability to back up multinode and multicontainer applications, including not just the data but application configuration and state as well.

Distributed Architecture

Cloud native architecture makes data protection challenging. Because applications comprise loosely coupled microservices managed in containers, traditional machine-focused data protection strategies don't suffice. Traditional backup solutions simply capture the entire state of the VM where an application runs. With distributed, containerized applications this doesn't work, because the state itself is distributed. Figure 4-1 shows the difference between traditional applications running on VMs and cloud native, containerized applications running on Kubernetes.

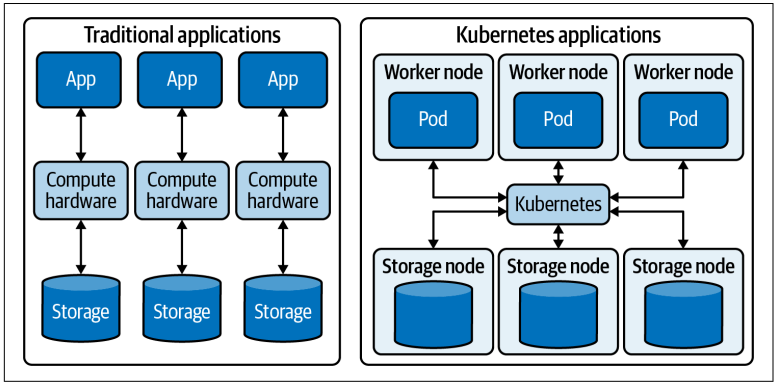

Figure 4-1. Traditional applications versus containerized applications

Cloud native applications are highly distributed, often across multiple clouds in multiple geographical locations. Containerized application deployments frequently span public and private clouds, sometimes on premises as well. Data protection in these environments must be container aware and able to integrate with container orchestration frameworks.

Namespaces

Kubernetes *namespaces* are a mechanism for partitioning a single cluster into different isolated groups, either to allocate resources to different business units or to group applications together. An IT admin might need to back up all applications running in a particular namespace at the same time. Because a namespace can contain a large number of pods, it is often not practical to attempt this manually. But traditional backup tools have no awareness of the Kubernetes namespace model and can't integrate with the Kubernetes API, leaving no alternative to taking a manual backup of every machine.

Recovery Point Objective and Recovery Time Objective

Some applications work with data that is critical in the moment. Other applications might require long-term data fidelity while tolerating momentary interruptions, or might work with data that is neither time sensitive nor mission critical. It is sensible to treat these applications differently, setting policies for each based on compliance, importance, and other aspects of their significance to the business.

Each application, and each type of data, might need its own data protection strategy, with its own recovery point objective (RPO) and recovery time objective (RTO):

- The RPO determines backup frequency, and represents the recency of recoverable data.
- The RTO is the maximum amount of time allowed to resume operations after a disruption.

The RPO and RTO policies are determined by the potential impact to the business of downtime, and how much data the business can realistically afford to lose, based on the application and type of data concerned.

In a distributed application at scale, providing high availability and recovery to meet a specific RPO and RTO often means maintaining extra data clusters or moving high volumes of data very quickly. To meet these goals in a modern, cloud native environment, you need automated and application-aware tools that can work with Kubernetes abstractions and APIs, protecting data at the container level.

The continuous integration/continous delivery (CI/CD) pipeline is a good example of an environment that can tolerate a moderate RPO and RTO. Development tools are important, but they don't immediately impact the customer experience. For these applications, some data loss or interruption to workflow is acceptable.

Data protection must be stricter for applications the customer touches directly, especially those that handle sensitive customer data. For these applications, both RPO and RTO are important. It's key to be able to recover data to a very recent point, and to resolve interruptions quickly. This sometimes requires disaster recovery across data centers, or failover from one active data center to another.

The most critical applications are those that can't afford any data loss and can tolerate only very brief interruptions: transaction processing tools, for example. This means a very short RTO time and an RPO of zero, which are demanding requirements to meet. This can only be accomplished with a combination of high availability within each cluster, backup and recovery with a compliance focus, and multiple data centers kept in sync to enable high availability and failover across geographical areas.

Data Protection by Application Type

Different applications not only store different data types and formats, but also employ different data policies. Database applications, for example, store the bulk of their data in tables, but also maintain application state, write application logs, and consume configuration files. For a traditional application running on a VM, the solution is easy: a backup of the VM preserves everything, including application state. For distributed applications, this is not the case. Backup, disaster recovery, and other data protection processes for Kubernetes must be aware of the specific needs of applications and APIs rather than taking a generic approach.

Because application state is distributed, backup solutions must be aware of how each application uses data, and must be able to find and protect all the different kinds of data the application uses. For example, an application-aware backup tool might know that a particular application keeps a queue of pending writes in a cache, and might ask the application to write them to disk before taking a snapshot so that it can capture a complete view of application state.

Strategies for Kubernetes Data Protection

A Kubernetes data protection strategy must provide high availability, backup and recovery, and failover both within and across data centers. These functions must be automated, application aware, and scaled for the enterprise. Data protection must be granular to the container level and aware of Kubernetes topology, storage, and abstractions. Finally, any solution must be able to support the different applications, business requirements, data, and SLAs an enterprise might require.

Container-Aware Backup and Recovery

Traditional backup and recovery methods protect applications and data at the machine level. By backing up the machine, it is possible to restore the previous running state of the applications on the machine. This approach works well for an application that runs discreetly on a single host, but it doesn't work for a microservices-based application distributed in containers that span multiple nodes in a cluster. For this reason, you must be able to locate the application's data across the cluster to create an *application-consistent* backup, meaning that the process makes copies of all the application's volumes and state at once. Application-consistent backups require domain-specific knowledge of the application to locate all volumes and capture application state properly. Failure to back up data in an application-aware way can lead to data corruption and loss.

A Kubernetes backup tool must be able to integrate with the Kubernetes API, be aware of Kubernetes compute and storage resources, and have the ability to map cluster topology so that it can back up groups of pods or entire namespaces. Finally, a Kubernetes backup tool must be aware of the different requirements for different applications and be able to capture and restore not just persistent storage

but application state and configuration as well. In other words, the system must know how to treat different Kubernetes objects that comprise the application, rather than attempting to back up the nodes themselves.

As cloud native architecture takes hold, data protection is moving from IT to a shared responsibility among multiple teams, including application owners. Where traditional backup was centralized, container-aware backup provides application owners with role-based self-service capabilities, including the ability to set their own backup policies and rules to ensure backups are application consistent.

Containerized applications are designed for scale, and backup solutions must be able to scale with them. A single application can scale to thousands of objects, and an enterprise might run hundreds or thousands of such applications. A Kubernetes backup solution must handle many thousands of objects and storage volumes.

Data Protection Within a Single Data Center

Data protection in a single Kubernetes cluster entails ensuring high availability of the Kubernetes components and applications, and maintaining the appropriate data replication. This mainly means setting up Kubernetes to avoid a single point of failure for any service or volume.

Replication strategies vary by application type and business requirements. Simple applications, which don't handle their own replication, rely on the underlying storage layer to be available without fail. When the data those applications handle is ephemeral and of low value, it may be acceptable to keep only one copy, on the assumption that failure of the node where the volume resides will not have much impact. For more important data, of course, the storage layer should be configured to provide adequate replication, both for data protection and for availability. For data that is important and in high demand, more replicas across more clusters can serve a larger number of clients with lower latency. Some applications handle their own replication. For these applications, the job of the storage is mainly to provide replacement storage in the event that a volume (or its node) fails.

Disaster Recovery Across Data Centers

A comprehensive data protection strategy should complement protection within the data center with failover and disaster recovery to secondary or standby clusters located in different data centers. In this scenario, an active cluster replicates data and configuration to a standby cluster using a timeline determined by RPO and RTO requirements, keeping it in sync so that it can take over if the active cluster fails. Figure 4-2 shows replication from one data center to another.

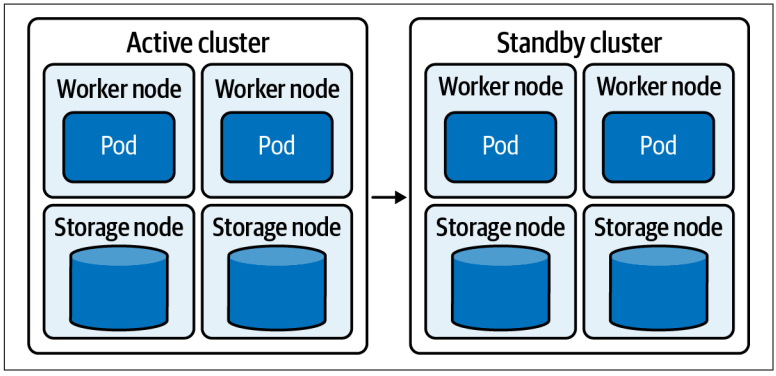

Figure 4-2. Replication from an active cluster to a standby cluster in a separate data center

When replicating data to another data center, it's important to be aware of the receiving cluster's topology so that replicas are distributed appropriately among nodes, for both performance and security reasons. The replicated data must provide highly available, performant access while guarding against loss in case of node failures.

There are two strategies for keeping the data centers in sync: synchronous replication, for data that requires stringent RPO and RTO times or immediate failover; and asynchronous replication, for data that can tolerate some loss or unavailability.

Synchronous replication ensures that all of the data written to the active cluster is replicated in the standby cluster. A write to the active cluster is considered complete only when the write to the standby cluster is complete as well. This approach is challenging because it requires very low latency to maintain write performance, but the benefit is that it enables an RPO of zero to support

mission-critical data such as transactions. In environments where sufficiently low latency is impossible, you must be able to set migration policies at the level of individual volumes and objects, allowing less critical data to replicate asynchronously to save bandwidth.

Asynchronous replication replicates data from one cluster to another on a schedule, which doesn't guarantee that the clusters are completely in sync but saves network bandwidth. Applications that can tolerate moderate amounts of data loss or downtime can use asynchronous replication to achieve the appropriate RPO and RTO. The RPO depends on the recency of data that is guaranteed to be copied from the active cluster to the standby cluster. The RTO depends on how much time it takes to restore the application to full functioning in the case that the active cluster becomes unavailable.

CHAPTER 5
Moving to Kubernetes

In today's world, consumers and business customers alike demand increasing speed. Complex, modern applications must deliver better results more quickly in areas like personalization, artificial intelligence, and fraud detection across verticals. Centrally managed and human-operated applications are a drain on struggling IT teams and provide diminishing returns; traditional application architecture can't keep up with today's huge data volume, global deployment, and demand for horizontal scale.

PaaS and SaaS are paradigms that work best when they are elastic, scaling up and down to meet unpredictable demand. Unlike traditional monolithic on-premises applications, PaaS and SaaS must scale horizontally, adding physical or virtual resources immediately when needed. Because adding physical on-premises servers is often a slow process with a long lead time, this means that modern applications must be able to run anywhere: in on-premises environments, in cloud environments, or in hybrid environments that consist of a combination of the two.

The answer that has emerged is to break up applications into microservices running in containers. This introduces complexity beyond what IT teams can be asked to manage manually, necessitating automated management, provisioning, load balancing, and fault resolution. These requirements are essentially the feature list of Kubernetes. Kubernetes is a platform that orchestrates collections of containers in a variety of environments, making it possible to build cloud native, microservices-based apps and run them anywhere. For

this reason, Kubernetes has become the dominant way that companies deploy applications at scale.

Even when DevOps teams have all the skills they need to run a production Kubernetes cluster, data storage remains challenging. Every application and service has its own storage needs, metadata, and way of handling state. Every kind of data has its own requirements around security, scalability, compliance, and availability. If that isn't enough, the sheer number of applications and data types required for the enterprise means exponentially more complexity.

Challenges of Kubernetes Adoption

Few enterprises are starting from scratch. A typical enterprise runs large numbers of legacy applications originally designed to run on physical servers or VMs. While cloud native, containerized applications are the clear path forward, it isn't practical to immediately abandon all the legacy applications that are powering the organization.

Adopting Kubernetes means investing in cloud native, containerized applications while supporting more traditional application architectures. Providing separate storage for both types of software is impractical both because of operational cost and because it slows down data processing. There are two approaches that can help: running containers inside VMs, or running VMs inside containers. An advantage of the latter approach is that it allows Kubernetes to manage the VMs where traditional applications are running, moving them as needed.

Persistent storage is paramount for every organization as data volume grows. The enterprise requires seamless management of vast quantities of data, with stringent needs around data security, high availability, and disaster recovery. Data must be available in multiple locations, geographically controllable for compliance, and highly performant. Designed originally for stateless processes, Kubernetes isn't natively suited to the data needs of enterprise companies.

To run applications at scale, the enterprise needs Kubernetes. But to run Kubernetes at scale, companies also need strategies to provide an underlying storage layer, decoupled from compute containers, which can support the needs of complex, large-scale business applications.

Running Large-Scale Systems on Kubernetes

The key to running Kubernetes at scale is the ability to optimize resource allocation as demand changes, automatically scaling resources and services up and down as needed. Although Kubernetes lets you declaratively automate cluster management and compute resources, you still need additional tools to efficiently manage storage.

An important factor is to ensure that storage is allocated properly and not wasted. Because block storage was not originally designed for use by massive numbers of containers, resource efficiency in Kubernetes can be challenging. Out of the box, Kubernetes offers a limited number of volumes per host and doesn't provide a wide range of tools for capabilities such as failover, backup, and disaster recovery. This often leads to overprovisioning of block storage by a factor of two or three, and a resultant decrease in application efficiency.

On the public cloud, providers generally limit the number of block devices that can be attached to a single VM, leading to another kind of overprovisioning. If a VM has enough compute power to serve more containers than the provider allows, it becomes necessary at some point to provision an additional VM for storage. This drives up compute cost and increases management overhead.

Data portability becomes a more important part of managing storage resources at scale. To maximize operational storage efficiency, it must be possible to move workloads to the most cost-effective storage. This means the enterprise must be able to move data and applications nimbly between environments, across providers, or from one team to another. Hyperconverged topology can help reduce cost and complexity, making it easier to scale.

Software-defined storage, which abstracts away individual hardware and presents all available storage as a single pool, is one important component in running Kubernetes efficiently at scale, because it can

support traditional applications as well as containerized workloads. You still need to make decisions about the underlying compute and storage hardware. Just running the compute and storage platforms requires a minimum set of resources on every node. Above that floor, you must make decisions about how much work each node should be capable of, such as:

- How many apps will run on each node
- How many containers each node will support
- How much CPU each container gets

For the same total cluster workload, you must decide whether to run a smaller number of more powerful nodes, or a larger number of less powerful nodes.

Cloud and hybrid environments make these questions easier to answer, because they allow you to start small and grow as needed by provisioning more compute and storage resources. There is no way to anticipate the needs of every possible application and workload. It's impossible to plan a cluster that will work for everyone. By provisioning only what's needed and scaling as you grow, you can continually tailor Kubernetes at scale to meet the needs of the enterprise.

One of the most time-intensive and disruptive aspects of Kubernetes in production is storage capacity management. In highly dynamic environments, storage needs can be extremely volatile, making storage nearly impossible to manage without sophisticated tools and automation. The right storage management solution can help you provision storage intelligently to avoid waste while scaling on demand.

Ultimately, the goal of running large-scale Kubernetes deployments is to serve enterprise applications, including PaaS and SaaS. These deployments must meet demands for data security, portability, compliance, locality, and availability, among others. Kubernetes can't meet these business-critical needs without help. To meet the data needs of the marketplace, you need a coordinated storage layer that can manage the reliability and performance of container volumes, providing high availability, replication, storage tiering, disaster recovery, backup, and the other features that the modern enterprise application demands.

About the Author

Peter Conrad is a technical writer with diverse content development experience, ranging from consumer electronics and telecommunications to IoT and enterprise software, in both hardware and software environments. He has cultivated strong interpersonal skills from interviewing and collaborating with diverse subject matter experts; from showcasing technology effectively for different audiences; as a liaison obtaining reviews and approvals of documentation; and as a user advocate.