Brought to you by

Red Hat Storage

Storage Patterns for Kubernetes™

for dummies®
A Wiley Brand

Converge storage and applications

Solve complex storage needs

Understand architectural challenges

Erin Boyd
Sage Weil
Karena Angell

Red Hat Special Edition

About Red Hat

Red Hat is the world's leading provider of enterprise open source software solutions, using a community-powered approach to deliver reliable and high-performing Linux, hybrid cloud, container, and Kubernetes technologies. Red Hat helps customers integrate new and existing IT applications, develop cloud-native applications, standardize on its industry-leading operating system, and automate, secure, and manage complex environments. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500. As a strategic partner to cloud providers, system integrators, application vendors, customers, and open source communities, Red Hat can help organizations prepare for the digital future. Learn more at **https://www.redhat.com.**

About the Authors

Erin Boyd, Sr. Principal Engineer at Red Hat, is a Kubernetes contributor and an Apache Ambari committer. Erin is an active contributor to the Kubernetes Storage SIG and is currently the co-chair of the CNCF Storage SIG. Erin focuses on how to enable storage in a stateless, hybrid, multi-cloud environment.

Sage Weil is the lead architect and co-creator of the Ceph open source distributed storage system. He continues to refine the system with the goal of providing a stable, next generation distributed storage system for Linux. Sage co-founded Inktank in 2012, which was acquired by Red Hat in 2014, and he continues to improve Ceph and help shape Red Hat's overall storage strategy.

Karena Angell, Product Marketing Principal at Red Hat, focuses on data solutions for the hybrid cloud. She brings expertise in enterprise storage with experience across multiple departments, including Sales, Marketing, and Product Development. Karena delivers real-world solutions for many of today's emerging data-driven workloads.

Storage Patterns for Kubernetes

Red Hat Special Edition

by Erin Boyd, Sage Weil, and Karena Angell

with David Greenstein

for dummies®
A Wiley Brand

Storage Patterns for Kubernetes For Dummies®, Red Hat Special Edition

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

ISBN: 978-1-119-66632-5 (pbk); ISBN: 978-1-119-66637-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

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

Project Editor: Carrie Burchfield-Leighton

Acquisitions Editor: Ashley Coffey

Production Editor: Mohammed Zafar Ali

Editorial Manager: Rev Mengle

Business Development Representative: Molly Daugherty

Introduction

Have you run Docker containers and discovered that storage isn't as simple as mounting a directory? Perhaps you have exposure to Kubernetes and have discovered volumes but need more? The vast and flexible world of hyper-converged infrastructure and how that can be implemented with Kubernetes can help. Beyond understanding the pieces and parts, Kubernetes brings it all together with concrete examples of how your applications can benefit from enhanced storage capabilities.

About This Book

In this book, we provide you with an overview of Kubernetes storage and how it's implemented. You review the common storage mechanisms in Kubernetes and what they're used for. Building on the base storage capabilities, you discover the common software-defined storage systems that can run on top of Kubernetes. A brief look at some example architectural patterns also illustrate the benefits of using these technologies together.

You also get practical information to help you apply Kubernetes patterns to your applications, complete with pictures. And finally, we give you a brief glimpse at some of the emerging trends in Kubernetes storage.

Icons Used in This Book

To help guide you through the material, you find a few icons along the way. They're intended to grab your attention or to help direct you through the text.

TIP

The Tip icon gives you helpful hints or pointers to something that may assist you in understanding or implementing the technology being discussed.

REMEMBER

Remember icons help you recall information mentioned elsewhere in the text or give you important tidbits to remember.

WARNING

The warning icon calls your attention to information that may be a stumbling block or pitfall. As you look to discover or use bits of info from warning sections, pay extra attention to the detail.

TECHNICAL STUFF

These icons give you a heads-up that the topic immediately following is likely to be deeper in the technical "weeds" than other passages. You may find this information interesting but not completely necessary for a higher level understanding of Kubernetes storage patterns.

Beyond the Book

This book can help you discover more about Kubernetes storage, Rook, Ceph, and related technology, but if you want resources beyond what this book offers, we have some insight for you:

- Kubernetes documentation is always a good place to start when discovering its ever growing features. Visit `kubernetes.io/docs/home`.
- The Rook Quickstart Guide can help you deploy your own environment quickly and easily. Find details at `https://rook.io/docs/rook/v1.1/quickstart-toc.html`.
- Find helpful information and videos about NooBaa at `www.noobaa.io`.

IN THIS CHAPTER

» Discussing persistent storage for containerized applications

» Understanding volumes in Kubernetes

» Using tools to help with operations for volumes

Chapter 1 Introducing Storage in Kubernetes

Kubernetes has emerged as the most mature platform for managing, configuring, and deploying containers at scale. It provides you a declarative, API-centric way to describe container-based applications that can self-manage, self-heal, self-scale, and more. In this chapter, you explore the impact of expanding the automation and orchestration from applications and networking to also include storage, and you look at how you can apply these technologies and patterns to your applications. We also discuss where storage is heading in the Kubernetes ecosystem.

Persistent Storage in an Ephemeral World

Containers have changed the way we build, ship, and run applications. This transformative technology affords greater velocity and application portability but has also brought new challenges to technology teams.

Challenges of persisting data in a transient system

Kubernetes is the leading orchestrator for container workloads. When a container stops for one reason or another, Kubernetes is able to start a new instance almost immediately, making availability and recoverability something that happens in seconds as opposed to minutes or even hours. But this new instance of the application may not have access to the file system from the old instance, especially in the case of local file systems, so additional orchestration is required to ensure that the stateful component of the application is bound to the instance at restart.

This focus on stateless applications and ephemeral systems has prevented certain stateful and complex applications from being a good fit for this revolutionary technology. As companies look to bring applications into Kubernetes, the inability to persist data through a container's life cycle is undesirable. People need a way of dealing with stateful applications and persisting data, all without inhibiting the positive aspects of the self-healing capabilities of the Kubernetes platform.

Preparing for enterprise workloads

To get a better sense of how to address challenges in Kubernetes, in this section, you take a look at one of the earliest challenges that has been successfully addressed: handling log data. The Kubernetes community found a successful way to address log data by sending it to an external system. By using solutions such as the ElasticSearch, Logstash, and Kibana (ELK) Stack, you have a pattern of leveraging an agent in a container alongside applications. This process enables the developer to send logs to a standardized end point, which makes it portable and flexible.

In the enterprise, moving storage into external systems isn't something new or unique. The use of a Storage Area Network (SAN) has been a staple practice of enterprises for a number of years, but typically it has been implemented by delivering a preconfigured, dedicated volume to a virtual or physical machine.

So, how do you provide a stable storage solution for any type of content, with an equally composable and portable convention as the ELK pattern? I'm glad you asked. You don't want to inhibit your IT team's abilities to backup, recover, or secure the data either. Fortunately, that's where Kubernetes steps in. Its features and capabilities address this challenge, and they're rapidly growing and maturing. Kubernetes began with storage features as part of the Pod API and soon expanded these into standalone interfaces. These include

- Volumes
- Persistent volumes
- Storage classes
- Container Storage Interface (CSI)

TIP

A Kubernetes *pod* is the smallest schedulable resource in Kubernetes. It's a logical collection of containers, which ensures the encapsulated containers share resources and are hosted on the same node in your cluster. Typically, a pod consists of a single container and is an instance of an application.

Volumes in Kubernetes

The Kubernetes volume in its simplest form is a directory that's attached to a pod and mounted to one or more containers in the pod. Such coupling is tight and is a foundational construct for how data can be presented to a container.

Because of this relationship with the pod, Kubernetes volumes are designed to have the same life cycle as the pod. If an ephemeral volume is defined by the pod, for example, its contents don't survive a container restart. When a pod is removed from Kubernetes, any volume relationships are broken and ephemeral volumes will be destroyed.

REMEMBER

Kubernetes volumes are typed, which is how you define the storage mechanism backing the volume. Volume characteristics may be defined via volume type-specific parameters. This may be a physical disk or path on the Kubernetes node, a SAN, network attached storage (NAS), or even a storage service from a cloud provider — to name a few.

Volumes allow applications to have a defined storage path that stays consistent through the redeployment of a container. This layer of abstraction helps but on its own doesn't solve all persistent storage concerns.

Special volumes

Kubernetes makes the following special volumes available for use:

- » emptyDir
- » configMap
- » secret

When a pod is scheduled to run on a node, a volume is created on that node and will remain there until the pod is deleted or moved to another node. This volume is an emptyDir volume that begins empty and is backed by the storage of the host itself. Given the nature of this volume type, it's not regularly backed up and should be treated as ephemeral or temporary storage in most cases, such as caching or working objects. Every pod can make use of an emptyDir.

The other two types, configMap and secret, are volumes whose content is actually stored by Kubernetes. This mechanism is often used to pass configuration information, such as credentials, into a container and ensure they're available wherever the pod is scheduled to run.

TIP

Additional information about configMaps and secrets may be found at `kubectl.docs.kubernetes.io/pages/app_management/secrets_and_configmaps.html`.

Volumes on nodes

Much like the emptyDir (see the preceding section), you can also define a hostPath volume or a local volume. Both volume types provide a way for you to specify a local path or storage device on a node and mount it in a path defined in a container.

hostPath volumes

WARNING

While hostPath volumes allow you to access the file system of the Kubernetes node it's scheduled on, it also means your pod can get access to other containers running on the host, certificates of the kubelet, and other sensitive files. Some critical limitations of hostPath volumes to keep in mind include the following:

- The same path may be absent or contain a different version of the referenced file from node to node, causing irregular behavior in an application.
- Kubernetes can't account for the storage available for hostPath volumes, which means your available local storage is in a blind spot of your orchestrator.
- Typically, only the root user has the ability to write to files and directories in your hostPath and forces you to run your container as a privileged container or your processes as root. This is a bad practice because it's at odds with the principle of least privilege (PoLP).
- Each node that can be targeted by a hostPath pod must be able to support these pods.

Local volumes

A local volume varies slightly from the hostPath in two ways:

- It's a direct reference to a local storage device on a node — not just a directory.
- Local volumes provide a more portable solution because the Kubernetes scheduler is able to calculate what node the pod should run on based on characteristics of the volume itself. This is in contrast to the manual process necessary for hostPath volumes (see the preceding section).

Externally backed volumes

As the relevance and utility of local storage options are exhausted, the need arises to leverage external storage. Individual nodes may have storage provided by a SAN, but that's not the same as being able to orchestrate the external storage along with the rest of the instructions for managing an application.

Kubernetes provides a Persistent Volume Claim (PVC), which is a request for storage. The request defines characteristics of the storage that are desired by your application, such as

- Access and volume modes
- Size of volume

- Selectors
- Storage class

TIP

Find details about the PVC elements at `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims`.

In conjunction with PVCs, Kubernetes provides a Persistent Volume (PV) resource. A PV has historically been a preprovisioned storage volume but more commonly is provisioned from a Storage Class dynamically and made available to the cluster for use by a pod.

In a Kubernetes cluster, when you create a storage class, you define

- **Provisioners:** The utility or component responsible for interacting with the backend storage system
- **Parameters:** Settings or values that you can pass to the provisioner
- **reclaimPolicy:** Defines how the volume should be processed when a pod no longer needs it

TIP

The reclaimPolicy instructs Kubernetes to either retain or delete the volume when it's released. If your volume is retained, you may need to perform your own volume cleanup and maintenance.

Extending volume support

To make storage system support more extensible, plugin-based storage driver mechanisms are needed. Instead of only developing and shipping storage drivers as part of Kubernetes itself, storage vendors and users wanted a way to provide and use the drivers they need, and to omit the ones they don't. So along came CSI and FlexVolume, which are the out-of-tree extension points for storage drivers.

Both plugin types allow for administrators to deploy storage drivers that can be loaded dynamically by Kubernetes nodes, and both provide the ability to provision volumes of any number of storage types. FlexVolume was introduced first but CSI is emerging as a more dynamic and comprehensive specification.

TIP

FlexVolume is included for completeness; however, the convention is being deprecated. Only use it if you must.

The CSI specification outlines a number of storage operations in a way that can extend to additional container orchestration platforms.

IN THIS CHAPTER

- » Discovering real-world Kubernetes storage examples
- » Moving storage solutions into your Kubernetes cluster
- » Seeing the practical side of converged storage systems for your applications

Chapter 2 Looking at the Convergence of Storage and Applications

In Chapter 1, you discover the numerous capabilities in Kubernetes to address both ephemeral and persistent storage. In this chapter, you find out about systems and patterns that can be added to Kubernetes to solve even more complex storage needs and architectural challenges.

Facing Storage Concerns in the Real World

When you think about adopting a new application architecture, such as cloud native, it seems logical to do it with a new application. More times than not, however, you don't find yourself only developing new applications. If you're like most folks, you have to blend the traditional needs of applications and the functions they serve, while applying emerging application architectures to improve speed, adaptability, and costs. At this point, Kubernetes and the real world collide, and storage is a big part of this transformation.

Containers need storage

You build business applications in containers. They implement business logic, automate decisions, and set out to make the lives of your users easier. Your applications also produce data — it may be in the form of files, data tables, reports, and so on. Building small functional bits of logic into containers is a good decision, but what do you do with the data?

REMEMBER

Without storing data in a persistent volume, the file system of a container is also ephemeral. When a container restarts, the file system is reset to whatever the file system state is in the container's image and files not stored in a persistent volume don't survive the restart.

You need a sustainable and supportable way to store data and objects in flight or as output from your applications, while taking advantage of the portability, scalability, and recoverability of containers. Instead of fighting the natural tendency of containers to be stateless and lightweight, the pattern looks to augment Kubernetes to facilitate the behaviors we want.

If your Kubernetes cluster has access to or is in a public cloud provider, such as AWS, you have platform services that can provide the following:

- **Block devices:** Elastic Block Store (EBS)
- **File system as a service:** Elastic File System (EFS)
- **Databases as a service:** Relational Database Service (RDS)
- **Object storage:** Simple Storage Service (S3)

REMEMBER

Each of these offerings has individual performance characteristics, pros, and cons. You must align your application and workload with the proper type of storage for its needs. This alignment requires multiple unique methods and processes for orchestrating the various storage mechanisms, and not all are aligned with the way the application is orchestrated by Kubernetes.

Software-defined systems include storage

Orchestrating stateless, containerized applications requires adopting a declarative model of describing your application deployment. The desired state of your deployment is described as YAML code, and Kubernetes works to make the deployed system match that desired state.

So logically, you leverage Kubernetes to provide a consistent interface between applications and compute, achieving portability of the application without needing to know the nuances of individual hardware. The same can be said for storage services from the cloud service provider (CSP).

Let's compare S3 and Google Cloud Storage. Both have a lot of similar characteristics and capabilities, such as providing means to create storage buckets, store and list objects, and so on. The API for interacting with each (the methods and properties), however, aren't identical. An application would need to be modified if it was moved between the two offerings. Likewise, the provisioning and security conventions of the two offerings are also different. This is in contrast to the use of block or file-based storage from Kubernetes, which uses the familiar file system interface provided by Linux and containers.

GCP, AWS, and Azure all have storage drivers in Kubernetes that help offload the concerns of managing and provisioning the platform storage services and provide a common interface in PVCs and Storage Classes. But how do you achieve parity for clusters running inside corporate data centers, or for storage services you wish to use when a consistent experience in Kubernetes doesn't exist yet? One solution, which is to host your own storage system, is covered in the next section.

Bringing Storage Orchestration Solutions into Kubernetes

Capabilities on-premises and across different clouds are different enough for storage and application orchestration that it can leave you overwhelmed. Converged infrastructure is an architecture where you orchestrate your compute and networking in one platform. In this section, you look at bringing storage into the mix as well and how doing so with Kubernetes can help you run the same platform on premise and across multiple cloud providers.

Hyper-converged storage systems

Hyper-converged infrastructure (HCI) is the combination of compute, networking, and storage, all implemented as software, and all deployed on the same nodes. This software-defined approach introduces immense flexibility in the underlying hardware, eliminating dependence on specific vendors and specialized components. It also affords IT teams the ability to achieve portability and consistency of management and configuration.

Rook

Rook is a storage orchestrator for Kubernetes. It uses the declarative nature of Kubernetes to make storage services self-healing, self-scaling, and managed with similar conventions as your containerized applications. Rook uses a plugin-extensible architecture to help you automate storage administrative tasks such as deployment, bootstrapping, configuring, scaling, provisioning, and monitoring.

What does this mean to you? Rook allows you to use a declarative style of management that empowers you to achieve "as code" capabilities for a multitude of storage types. It implements the new necessary Kubernetes objects so you have a common interface to provision storage software that provides object, block, and file system storage types.

Rook goes beyond primitive storage types and also enables you to orchestrate databases and other storage types in a claims-based fashion. This common way of dealing with volumes and now other storage types in Kubernetes is one way it is growing to orchestrate all storage types.

REMEMBER

On its own, Rook isn't a storage system; it must be coupled with a portable storage subsystem or service. Rook has operators to support the following storage backends:

- Cassandra
- **Ceph**

- CockroachDB
- **EdgeFS**
- Minio
- NFS
- YugabyteDB

Ceph and EdgeFS are the most mature and are actively supported by the Rook community. The other operators are mentioned for completeness and are in various stages of development.

Ceph

Ceph is a high-performance, distributed storage platform designed to bring similar capabilities found in GCP and AWS to any cloud platform, such as Kubernetes. It provides object storage, block storage, and distributed file systems backed by a single, reliable storage cluster running on commodity server hardware.

TECHNICAL STUFF

Ceph is designed to be extremely scalable. It's based on the Reliable Autonomic Distributed Object Store (RADOS), a self-healing system that distributes and replicates data safely across storage nodes. It then layers a distributed file system (CephFS), block storage service (RBD, or RADOS Block Device), and S3-compatible object storage service (RGW, or RADOS Gateway) on top of RADOS.

Because Ceph is open source and purely software, it can provide its rich set of storage capabilities and features when deployed across almost any server hardware and either hard drives, SSDs, or any combination of the two.

Ceph is widely used in a range of environments, especially private cloud deployments, but it can be complicated to properly provision and connect a Ceph system to Kubernetes. Rook simplifies the deployment and management of Ceph itself and also provides an extremely simple and Kubernetes-native way to provision storage resources (persistent volumes for consumption by containers). If you want to learn more about Ceph and its components, visit `https://ceph.io`.

Longhorn

Longhorn is another software-defined storage platform, providing Kubernetes with a storage subsystem which rounds out the HCI capabilities. Unlike Rook-Ceph, it is a single solution encompassing features of both of those projects, but only implements a distributed block storage solution, which means volumes can only be consumed by a single pod at a time.

Architectural patterns

If you've been reading this chapter up to this point, it may be helpful to see how Rook-Ceph fits into your Kubernetes cluster. Take a look at Figure 2-1. You see the relationship of Rook, Ceph, and Kubernetes components.

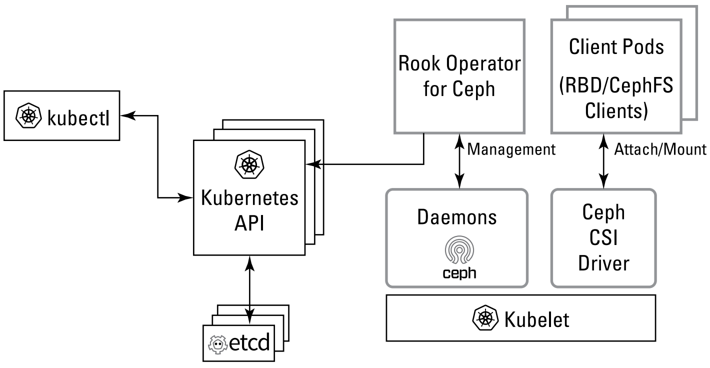

FIGURE 2-1: HCI with Kubernetes + Rook-Ceph.

Rook deploys an operator for Ceph, which is responsible for managing the state of the Ceph storage cluster. It will start, stop, or schedule the necessary Ceph services and components to provide the desired storage types.

Running your storage ecosystem inside your Kubernetes cluster means the same technology used to orchestrate your applications is now being used to orchestrate and ensure the availability of your storage systems for your applications. Native Kubernetes objects will be used to represent and manage the storage provider for your applications, such as Custom Resource Definitions (CRDs) to represent storage pools, object stores, file systems and other objects which are all part of the Rook-Ceph system — now in your cluster.

REMEMBER

CRDs are portable objects in the Kubernetes API that describe user-defined components without modifying Kubernetes itself.

Practical Application of Converged Storage Patterns

Armed with an understanding of Kubernetes storage and how Rook-Ceph extends those capabilities, this section helps you understand how you can make good use of the combined technologies. Putting HCI to work for you and your applications can make you more nimble and resilient.

Highlighting critical use cases

In this section, you take a look at actual use cases:

- » Using a cloud provider storage service to externalize data from the container
- » Using a software-defined storage system to provide a consistent storage service across different cloud environments and on-premises

Starting in Figure 2-2, you see the implementation of the application and storage patterns in a progressive fashion. You can use each of these application and storage patterns in the HCI system; we discuss this in the earlier section "Bringing Storage Orchestration Solutions into Kubernetes." In Figure 2-2, you see an application that consumes storage via a persistent volume (PV). If you move from AWS to a different provider such as Google, you can provision the application in the same way (although your data won't move with it — see Chapter 3). However, if you're running Kubernetes on-premises, there's no preexisting storage service to consume.

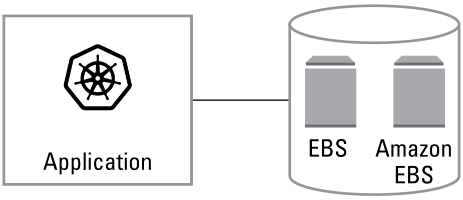

FIGURE 2-2: An application attached to a PV backed by Amazon EBS.

In Figure 2-3, the application can use storage provided by Rook-Ceph (or another software-defined storage solution), which is in turn consuming compute and storage resources from any cloud provider.

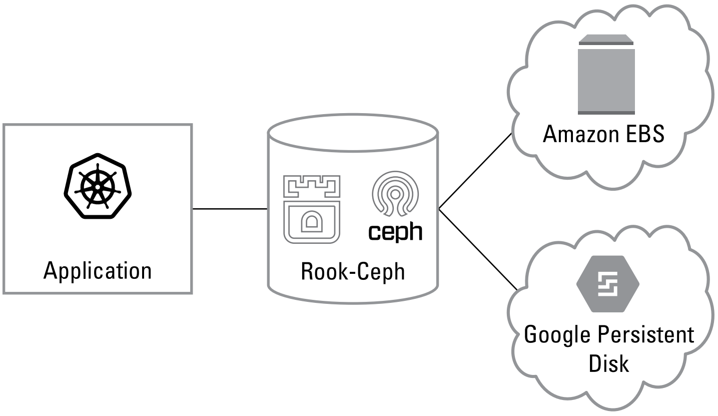

FIGURE 2-3: An application that consumes PVs provided by Rook-Ceph in any environment.

Finally, Figure 2-4 shows your application deployed to a hyper-converged cluster. The diagram illustrates your ability to make your applications portable across similar HCI clusters hosted in two different providers. It's critical to point out that this is accomplished with only one set of orchestrations to manage applications and storage thanks to the common interfaces and consistency provided by Kubernetes + Rook-Ceph. You can move your application as-is between providers now.

Keeping abstractions aligned

Normalizing how your applications (and your IT teams) interact with the entire stack enables you to achieve this goal. With a single set of instructions, you can run a hyper-converged Kubernetes cluster in any cloud or on-premises — yet not change any of your application manifests. Kubernetes is the single, common component that enables you to interact the same way with all providers.

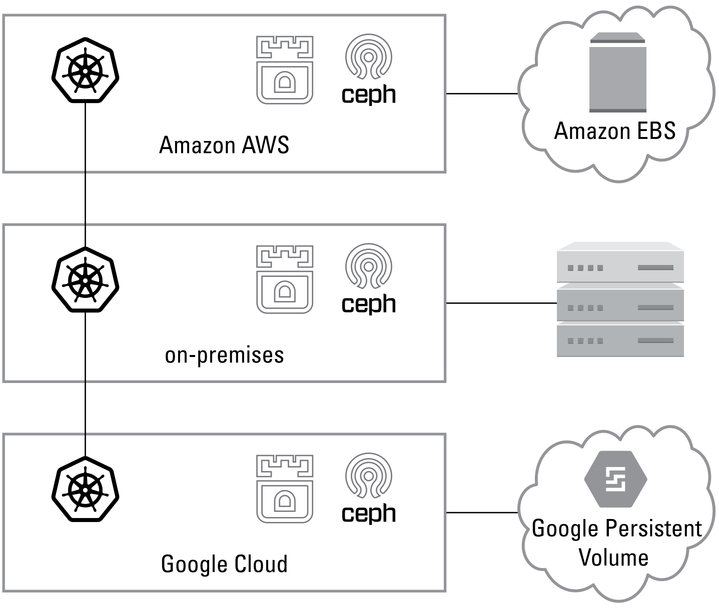

FIGURE 2-4: Application and storage deployment that is portable between HCI platforms on-premises and different cloud environments.

Having Kubernetes as the common interface solves challenges such as

- Normalizing configuration logic
- Using the same declarative code to deploy everywhere
- Providing a consistent security and compliance model across boundaries or providers

IN THIS CHAPTER

» **Discovering persistent storage for your applications**

» **Planning for and averting disaster in your storage platforms**

» **Migrating applications and their storage between platforms**

Chapter 3 Multi-Cloud and Hybrid Cloud Considerations

In this chapter, you examine different patterns for dealing with the complexity of running stateless apps that produce and use stateful data.

Portability Concerns around Block Storage Services

The Kubernetes Persistent Volume claim pattern provides a powerful storage abstraction that isolates your application from the details of the storage system — but only to a point. In practice, capabilities, performance, cost, and security vary among cloud storage services, traditional storage providers, and software-defined solutions.

For example, if you want to run Kubernetes in AWS, you may be using EBS volumes for block devices. A few limitations in using that technology include the following:

» Provisioning and attaching EBS volumes dynamically can be slow (for example, minutes), depending on the size of the volume. This has a direct impact on your application start-up and fail over times.

- Performance of EBS volumes is directly tied to the size of the volume. Such a relationship can lead to over-provisioning of EBS, which has a proportionate financial increase to size (and speed).
- EBS is built in a way that it's scoped to exist in a specific availability zone (AZ), inhibiting your ability to automatically fail over to another zone.

Software-defined storage platforms like Rook-Ceph implement a block device service whose capabilities are determined by the way the software is deployed and configured, and not by which server hardware or cloud platforms they're deployed on. It may seem strange to deploy a system like Ceph on top of a public cloud when analogous services are provided by the cloud provider, but it may be necessary to avoid limitations in the native service, like those for EBS.

Perhaps more importantly, software-defined solutions can provide consistency across cloud providers and on-premises deployments, so operators and application developers don't need to worry about inconsistencies. Furthermore, newer multi-cluster disaster recovery and replication features — such as asynchronous replication of volumes across data centers and between different cloud providers — may be available.

Looking at Shared File Services

Shared file system services are generally more demanding on the infrastructure provider because of the complexity of providing a consistent view of your data from multiple containers or nodes. For this reason, shared file services (for example, those that provide NFS-based storage) aren't always available from all cloud providers, and when they are, they tend to be very expensive.

It is also worth noting that capabilities between shared file services tend to have more variance in behavior and performance than block-based storage. The consistency model for NFS, for example, when data is accessed between multiple nodes is much weaker than a local file system or many other distributed file systems like CephFS or Lustre.

For these reasons, it's often smart to provision your own software-defined file solution by using a system like Rook-Ceph and CephFS.

Object Storage

TIP

If your application stores a lot of mostly static content, it may be best suited to use object storage services as opposed to block or file storage.

The advent of the public cloud, and in particular Amazon's S3, have popularized the use of object storage. Today, object storage is a critical part of storage infrastructure for most cloud-native applications. Historically, Kubernetes hasn't addressed object storage in any meaningful way — it's up to each application to manage object storage resources and handle connectivity to object storage services.

Today, Google, Azure, and AWS all provide object storage as a service. However, these services all implement slightly different APIs and security models, and provide slightly different management capabilities.

The complexity of your application, if it needed to be able to discover how and when to use each potential API, would be unsustainable. Solving this challenge is precisely what the storage abstractions in Kubernetes are designed to improve for development teams.

ObjectBuckets and ObjectBucketClaims

The Rook and lib-bucket-provisioner projects have recently brought the same simplicity of on-demand persistent volume provisioning to object storage. With persistent volumes, an application's declared state can specify a claim for storage that's matched against an existing storage class, and the storage class provisioner handles the platform-specific details of managing the life cycle and attachment of the volume to a pod.

The same pattern allows applications to request object storage with an ObjectBucketClaim (OBC) object that's matched to a storage class. That storage class may represent S3 on AWS, Blob storage on Azure, or object storage from Rook-Ceph — whatever it is, a bucket is provisioned, and access credentials and URLs are passed to the pod so it knows how to connect to the storage.

OBCs solve the provisioning part of the portability problem, but now the API part: Each cloud provider still implements its own object storage API. However, most have a mode that's compatible with S3, at least for the most commonly used operations. More importantly, virtually all object storage solutions used on premises — including Ceph — implement the S3 API.

TIP

NooBaa is an up-and-coming project that implements the S3 bucket protocol, using the Object Bucket (OB) and OB claim pattern. It normalizes how your applications interact with object storage, while affording you the ability to use multiple cloud providers on the back end. On the front end, it presents to applications and consumers an S3-compatible API. In addition, NooBaa has developed an operator for Rook making deployment a breeze. By providing such storage abstraction, NooBaa is able so solve many data portability challenges for hybrid and multi-cloud. Lastly, NooBaa exposes a method for expressing data policy that, as we move into complex hybrid cloud and multi cloud management, will be critical at scale. Discover more about NooBaa at `www.noobaa.io`.

TECHNICAL STUFF

When you save an object in NooBaa, the file is fragmented into chunks. The chunks are then processed to deduplicate, compress, and encrypt. Finally, the processed data chunks are stored in your configured backend storage systems.

lib-bucket-provisioner

TIP

lib-bucket-provisioner is a library for Kubernetes that introduces the OB and OBC, OB/OBC objects, and associated workflows, interfaces, and so on. Discover the details about this project at `github.com/kube-object-storage/lib-bucket-provisioner`.

There are examples of the OB/OBC pattern implemented in the library, the AWS S3 Operator, and Rook-Ceph. Members of the Kubernetes community who are working on these patterns and technologies are seeking broader alignment and consensus on how and when to implement in a more native way.

Challenges faced in implementing the OB/OBC pattern in a more generic fashion are how to address data policy management in an appropriate way. As these challenges and others are being looked at, you should recognize that providing a native way to deal with object storage rounds out the most common storage scenarios directly in the platform. Being able to describe and dynamically orchestrate, in an API-first fashion, block devices, file systems, and object stores solve a great number of challenges for developers and administrators alike.

Concerning Disaster Recovery

You don't want to think about it, but failure and disaster does occur. Kubernetes provides a great deal of application and workload resiliency, self-healing, and management improvements, but you need more than just that. As you implement Kubernetes and grow its usage in your environment, you need to address continuity at the application, cluster, and even provider level. This section describes the capabilities and tools available and how you can use them.

Application stack configuration state

If your application were completely ephemeral, there would only be the need to extract the application configuration from the old cluster and apply to the new cluster; however, we know a lot of business applications also require storage.

TIP

Store or stash your Kubernetes objects in a system external to your cluster, such as a Git repository. When coupled with CI/CD as part of a DevOps or GitOps methodology, your pipeline becomes an invaluable tool for deploying your app stack to your platform of choice.

If you wanted to make your Kubernetes objects even easier to manage and reuse, you'd leverage Helm charts or Operators. Their inherent templating capabilities make installing your application easier in any target environment, modifying only the environment specific values as necessary.

TIP

Learn more about Helm charts at `https://helm.sh` or Operators at `https://operatorhub.io/what-is-an-operator`.

Coupling of application stack and storage state

Traditionally, disaster recovery (DR) is implemented in one of the following patterns:

- Back up and restore, where you reconstruct your environment in a different data center, provider, or region from backups
- Active-passive, in which case you have a "cold" or offline set of components that need to be brought to the same data state as your current cluster before you start using it
- Active-active, which relies on a parallel set of components, but state and data are replicated so you may fail over immediately

In the absence of Kubernetes, managing data and applications across multiple platforms is difficult to do efficiently as the requirements for each are unique. Hyper-converged storage patterns and consistent platforms provide a means of standardizing the strategies for DR. By maintaining your data and application stacks as Kubernetes native elements, you reduce the complexity and thus the pressure in the event of a disaster.

In terms of offline backup strategies, Velero is a tool to help you back up and restore cluster objects in Kubernetes, including your persistent volumes. It plugs into the underlying storage systems snapshot capabilities to keep a point-in-time backup of content and objects for you.

By leveraging Velero's capabilities, you can

- Back up and restore your applications and associated data. (It's *always* a good idea to back up before an upgrade or major release.)
- Migrate whole sets of workloads between clusters.
- Replicate your cluster for testing, troubleshooting, and so on.

Velero consists of a server and a command line interface (CLI). Coupling Velero with Rook-Ceph or NooBaa can be an all-inclusive buffet of storage capabilities that will be self-hosted in your Kubernetes cluster. Velero triggers snapshots of your volumes, captures the objects from the Kubernetes API, and stores them all together in its own space in the storage provider.

Migrating Applications and Their State

Moving your apps, their configurations, and associated content between providers and platforms is a complex challenge that doesn't yet have an elegant solution in Kubernetes itself. Workload migration commonly happens as a manual process that includes several steps. This process typically forces an outage to the application and represents a fair amount of risk in the margins of error.

As we outline in the preceding section, assume you can freely move the application itself around — what do you do about the storage? One strategy is

1. **Stop your application where it's currently running.**
2. **Move your data manually, either cross region or cross providers through backup or copy operations.**
3. **Update your application configuration and cluster to appropriately associate the new storage to the target cluster.**
4. **Redeploy your application in the target environment and validate its operation.**

If your storage is more of a fixed point in the equation, such as when you're using NooBaa or any common external storage, you may take a more active-active migration approach. Stand up your application in the target environment while the original application is still in service, and simply reconfigure your entry point to direct users to the new environment or cluster.

TIP

Your applications and environmental decisions dictate the best approach, so plan as far ahead as you can for what exit strategy and continuity requirements you must implement.

Managing Storage in a Multi-Cluster/ Hybrid Cloud Environment

Many of the patterns we discuss in this book center on methods for managing your deployments for a relatively small number of clusters. As we scale, the complexity increases not only for ephemeral applications but also even more so for applications that create and utilize storage.

We discuss in Chapter 2 how software-defined storage can facilitate creating a consistent storage interface across clouds and on-premises, but what technologies are being developed to interact with the API to make deploying and managing systems easier? A new project, Crossplane, looks at addressing these challenges.

Crossplane improves multi-cluster environments. More than just Kubernetes, the entirety of the Crossplane project encompasses four main feature areas that may be used independently:

- **Crossplane services:** Provision managed services from kubectl
- **Crossplane stacks:** Extends Crossplane with new functionality
- **Crossplane workloads:** Defines complete applications and schedules across clusters, regions, and clouds
- **Crossplane clusters:** Manages multiple Kubernetes clusters from a single control plane

This combination of capabilities points toward being able to orchestrate and manage your applications and associated storage systems, including native provider services, from a common control plane. Find out more at `https://crossplane.io`.

IN THIS CHAPTER

» **Getting involved in the Kubernetes community**

» **Keeping an eye on the storage drivers**

» **Maintaining your learning curve**

Chapter 4 Ten Items for Your Kubernetes Storage Checklist

In this chapter, we give you ten critical items to keep in mind as you tackle storage in your Kubernetes clusters:

- **Get involved in the Kubernetes community.** If you haven't already become involved in the vibrant Kubernetes community, you should. Open-source software is made and tested by its community members. Ensure your interests are represented and join in the community. Learn more at `kubernetes.io/community`.
- **Consider using a converged storage platform.** If you want to run your application platform on any cloud, complete your capabilities by including storage in your cluster. If you think about portability as applying to individual workloads, embrace the most portable and commonly supported capabilities to achieve your goals.
- **Make good use of PersistentVolumeClaims (PVC) in your application bundles.** This is a critical abstraction layer that's highly portable but allows developers to describe critical

aspects of their volumes such as read/write and size. Expecting each cluster to be able to satisfy your PVC via a storage class isn't an unreasonable assumption.

- **Embrace the ObjectBucketClaim model for provisioning object storage.** Using the OB/OBC claim-based model insulates you from the different object storage provider backends much like PVCs do.
- **Understand the limits of emptyDir volume in your applications.** Leveraging ephemeral volumes still provides persistence beyond the container lifetime and is useful when you need ramdisks and the like.

WARNING

- **When possible, be API driven.** With the use of both the Kubernetes API and Custom Resource Definitions (CRDs) users can create a consistent model for defining their resources that make their deployments more portable in the future.
- **Keep an eye on the storage drivers you use in your environments.** As more storage drivers are converting to Container Storage Interface (CSI), you want to embrace the go-forward pattern as soon as you're able to.
- **Mind your Kubernetes versions.** As you look at branching out into multi-cloud deployments, mixed versions will have varying levels of support for the same features. For example, volume snapshotting is still an Alpha feature and was introduced in Kubernetes version 1.12. Alpha features aren't enabled by default, so not all 1.12+ clusters will have this available to you.
- **Get running quickly with operators and Helm charts for Rook-Ceph, NooBaa, and so on.** You can gain a lot of capability for less effort than deploying these solutions manually. Critical tools are available at the following websites:
  - `operatorhub.io/operator/noobaa-operator`
  - `operatorhub.io/operator/rook-ceph`
  - `github.com/helm/charts/tree/master/stable/velero`
- **Never stop learning.** Kubernetes is one of the best technologies we've ever worked with. It's rapidly evolving, with quarterly releases. As the ecosystem continues to become more plugin-centric, the components and tools will continue to change and grow exponentially. You can't stop reading and learning to stay on top of the rate of change to remain current.

Benefit from enhanced storage capabilities

In this book, you get an overview of Kubernetes storage and how to use it. You take a look at common storage mechanisms in Kubernetes and what they're used for. Build on your base storage capabilities with software-defined storage systems that can run on top of Kubernetes. Apply Kubernetes patterns to your applications and take look at the emerging trends in Kubernetes storage today.

Inside...

- Real-world Kubernetes storage examples
- Discover persistent storage
- Converged storage systems for your applications
- Avert disaster in your storage platforms
- Migrate applications and their storage

Red Hat Storage

ISBN: 978-1-119-66632-5
Not For Resale

9 781119 666325

for dummies® A Wiley Brand