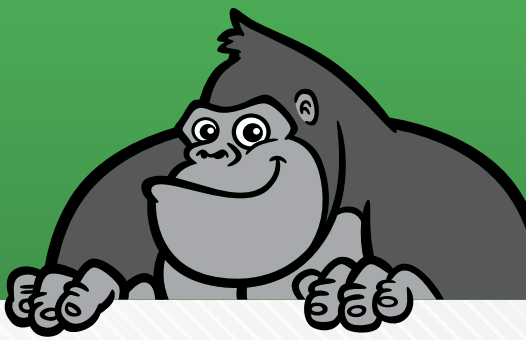
ActualTech Media

SCALITY

Hewlett Packard Enterprise

The Gorilla Guide to...®

Express Edition



# Modern Object Storage for Cloud-Native Applications

Dan Sullivan and Ed Tittel

## Inside the Guide

- A Primer on Cloud-Native and Kubernetes from a Storage Standpoint
- Storage Challenges for Containerized Applications Orchestrated by Kubernetes
- What Makes Object Storage Such a Great Fit for Cloud-Native and Kubernetes

THE GORILLA GUIDE TO...®

# Modern Object Storage for Cloud-Native Applications

## Express Edition

By Dan Sullivan and Ed Tittel



**ACTUALTECH MEDIA**

6650 Rivers Ave Ste 105 #22489
North Charleston, SC 29406-4829
www.actualtechmedia.com

# PUBLISHER'S ACKNOWLEDGEMENTS


**EDITORIAL DIRECTOR**
Keith Ward

**DIRECTOR OF CONTENT DELIVERY**
Wendy Hernandez

**CREATIVE DIRECTOR**
Olivia Thomson

**SENIOR DIRECTOR OF CONTENT**
Katie Mohr

**PARTNER AND VP OF CONTENT**
James Green


## ABOUT THE AUTHOR

Dan Sullivan is a software architect and engineer specializing in cloud architecture and analytics. He is the author of six books and hundreds of articles and white papers spanning a wide range of IT topics including cloud, monitoring, security, data architecture, and machine learning.

Ed Tittel is a 30-plus year veteran of the IT industry who writes regularly about cloud computing, networking, security, and Windows topics. Perhaps best known as the creator of the *Exam Cram* series of certification prep books in the late 1990s, Ed writes and blogs regularly for GoCertify.com, TechTarget, ComputerWorld, and other sites. For more information about Ed, including a resume and list of publications, please visit EdTittel.com.

# ENTERING THE JUNGLE

# CALLOUTS USED IN THIS BOOK


The Gorilla is the professorial sort that enjoys helping people learn. In the School House callout, you'll gain insight into topics that may be outside the main subject but are still important.


This is a special place where you can learn a bit more about ancillary topics presented in the book.


When we have a great thought, we express them through a series of grunts in the Bright Idea section.


Takes you into the deep, dark depths of a particular topic.


Discusses items of strategic interest to business leaders.

# ICONS USED IN THIS BOOK

## DEFINITION

Defines a word, phrase, or concept.

## KNOWLEDGE CHECK

Tests your knowledge of what you've read.

## PAY ATTENTION

We want to make sure you see this!

## GPS

We'll help you navigate your knowledge to the right place.

## WATCH OUT!

Make sure you read this so you don't make a critical error!

## TIP

A helpful piece of advice based on what you've read.

# The New Business Imperatives

Welcome to The Gorilla Guide To...® Modern Object Storage for Cloud-Native Applications, Express Edition.

Today's businesses work under increasing pressure to build applications faster and more efficiently. At the same time, digital transformation initiatives fundamentally change how those businesses deliver products and services.

This creates unprecedented demand for developers, IT staff, and underlying infrastructures to support such initiatives. Big data analytics and data science keep improving the amount of information and insight we can extract from data, and informs profound emphasis on data-driven decision making.

Ultimately, digital transformation and the insights and innovation that data can bring, relies on an organization's ability to collect, integrate, and analyze large volumes of data. Machine learning (ML) and artificial intelligence (AI) lets developers build applications to handle tasks and solve problems that, in the past, would have demanded copious human time and effort. And in fact, the ability to scale such intelligent processing is what drives ever-increasing adoption and use of ML and AI.

Software engineering practices and operations management practices that have served businesses well when working with mainframes and on-premises, bare-metal, and virtualized servers aren't well-suited to addressing the demands of modern, hybrid cloud-based application development and deployment.

Fortunately, a new, more effective, approach to application development and deployment has evolved. It's known as cloud-native applications. These applications are designed to scale with demand and run in public and private clouds, while being resilient to failures.

Such applications make extensive use of platform-agnostic container technologies such as Kubernetes and Docker to keep them portable, flexible, and agile. They also use platform-agnostic storage services and APIs to support containerized apps, persistent volumes, storage management and migration, and more.

In this Gorilla Guide, we'll take a look at how these tools support organizations and developers, and the importance of integrated development and operations. Let's start with a deeper dive into the rise of cloud-native applications, and how they're changing the game.

CHAPTER 1

# Cloud-Native Applications: Building on Containers and Microservices

Accessing, using, and interacting with cloud-based applications, services, and resources puts organizations in a complex and interesting situation.

Behind the scenes, cloud platforms and services employ a veritable ecosystem to support ready access to virtualized applications, services, networks, platforms, and even entire infrastructures. This is orchestrated via software and configuration data running in (and across) one or more provider's data centers.

If used correctly, numerous specific technologies come into play to give organizations flexibility and interoperability when bringing cloud-based services, storage, and networking into play. There are several key elements that anchor and inform this deliberate implementation, deployment, and management strategy:

- **Containers:** These lightweight runtime constructs function as discrete and separate process and resource handlers within which one or more applications or services can run. By design, containers include only the resources necessary for those applications and services. Thus, more containers can run on any given server or cluster than traditional virtual machines, because containers don't include a full operating system or instantiate services, protocols, libraries, and functions they won't use, unlike VMs.

- **Kubernetes:** Kubernetes is the leading platform for container orchestration. While there are other container orchestration products, Kubernetes should be seen as the de facto standard. It's open source, portable, and extensible, and manages containerized workloads and services with a large and growing ecosystem of tools.

- **Microservices:** This software development method focuses on building single-function modules, each with its own well-defined interfaces and operations. These modules are then assembled and combined to build applications or services. Small and simple by design, microservices require less time and work to implement, test, maintain, and adapt. And because any microservice can be updated, tested, and deployed independently of the others, ongoing development is simpler and faster. Modern microservices are containerized, so they can run on any OS or cloud platform that supports that container type. This is a profound benefit, and explains how microservices workloads and their data can migrate among data centers, private, and public clouds with relative ease and dispatch.

- **DevOps:** This term represents the conflation of development and operations under a single overarching development methodology. DevOps seeks to shorten the development lifecycle while also delivering features, fixes, updates, and enhancements frequently to better meet business or organizational objectives. DevOps practitioners often refer to "CI/CD," which stands for "continuous integration/continuous delivery (or deployment, in some cases)". Continuous integration is the process of making small updates to software and committing the changes to a centralized repository, sometimes as often as daily, to improve the product bit by bit over time. Continuous delivery is the next step in that sequence. It refers to automating application delivery into the various infrastructure pipelines for eventual release into the wild, wherever that wild is. See **Figure 1**.

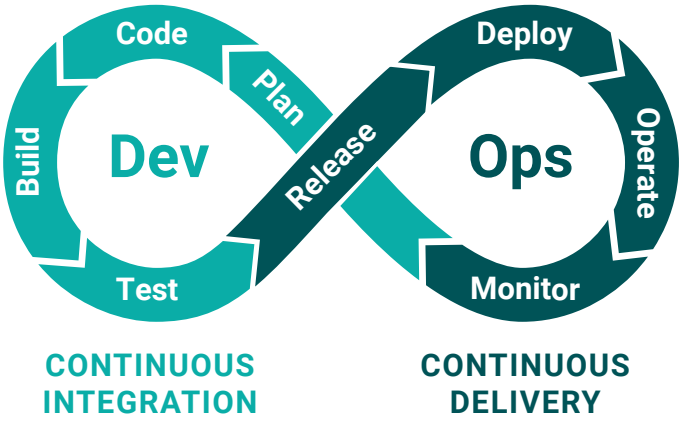



**Figure 1:** A typical CI/CD process pipeline

## Cloud-Native Architectures and Technologies

Cloud-native approaches to development use containers to define and build microservices-based architectures. Because such architectures consist of re-usable modules and components that can be assembled (and later adjusted or recomposed) to deliver applications and services to end users, they're not only immediately useful, but also flexible and adaptable in the face of change.

Organizations follow DevOps principles to guide them in designing, building, maintaining, and delivering cloud-native, containerized, and microservices-based applications and services. This approach enables organizations to meet current business objectives through streamlined, lean product development and delivery processes.

It also helps them adapt quickly to changes as they occur, to accommodate market changes, organizational change, or new tools and technologies to improve productivity and profitability.

**The Cloud Native Computing Foundation defines cloud-native in this way:** Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal muss and fuss.

## Stateless vs. Stateful Applications

In general, the distinction between stateless and stateful refers to persistence of data or memory between transactions or instantiations. When it comes to containerized applications, for example, stateless applications do not store data, whereas stateful applications include storage access so that they can acquire prior state and data, if any, when they start up, and save existing state and data when they pause or stop.

Maintaining state allows applications to work from information, knowledge, and data acquired or generated during prior activity. Stateless applications use transitory data, where state must typically be stored in a separate backend service such as a database.

For stateless applications, storage is ephemeral. That means its contents disappear if the container stops running, or gets restarted. When they first adopt containers, organizations tend to use stateless applications because they are easily implemented and adapted in

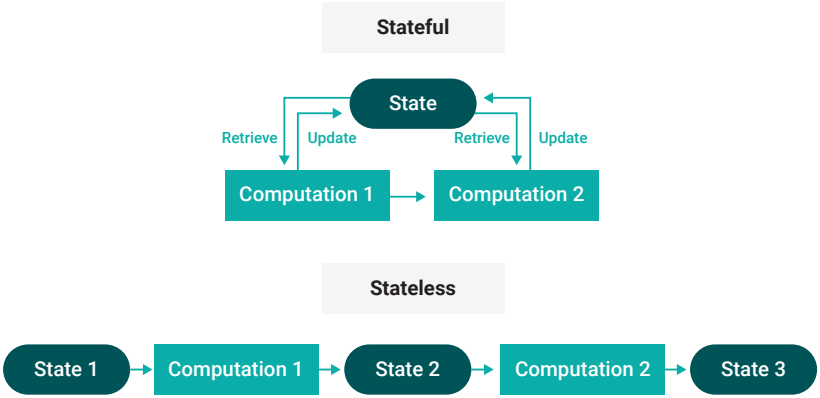



**Figure 2:** A comparison of a stateful vs. a stateless application

cloud-native architectures. Because they do not typically embody or incorporate more traditional monolithic code, stateless containerized apps built on microservices are also easier to move around and scale.

Stateful applications typically involve transactions, where the server processes requests based on data they provide but also uses information stored from previous requests. Thus, the server must be able to store and retain state information from the past, as well as respond to current requests on demand (see **Figure 2**).

Orchestration for stateful applications requires identifying the best location to run the container (or container collection) involved in its execution to best handle the application's needs for storage, networking, and to maintain a consistent and workable I/O path. In some cases, orchestration for stateful applications might also ensure high availability, by moving containers or remounting storage volumes without making (or needing) any changes to application code.

# Making Stateful Applications Work

Stateful applications generally work with an underlying storage layer through a set of application programming interfaces, or APIs. In fact, storage attributes required in a stateful containerized app pose important design and implementation decisions for organization who build and use them.

In a cloud-based environment, storage needs the same attributes that apply to the rest of that environment. As a container moves around a cluster or through the cloud, it must maintain its connection to its storage volume(s). Thus, a software layer between the application and its container, and the underlying storage layer, can automatically manage such connections, and change locations as needed (within or across clusters, availability zones, and even multiple clouds).

Building cloud-native applications generally involves dense conglomerations of microservices and their data. Making this work depends on a flexible and elastic software layer to mediate between those microservices and the underlying native storage (either on-premises or in the cloud).

Behind the scenes, cloud-native containerized environments must provide mechanisms to create persistent container storage volumes. This kind of capability involves integrating a persistent storage layer with container orchestration that uses a dynamic storage platform.

Such a platform should also comply with data security, protection, and resilience requirement for application deployment. This creates what might be called a software-defined storage platform, which microservices and their parent containers can access abstractly, while the orchestrator manages the details and the connections in the background. This also lets developers, IT, and even users (with self-service portal access) provision storage on their own without involving a storage administrator.

# Kubernetes Storage Initiatives: CSI and COSI

Persistent volumes (PV) is the construct through which Kubernetes exposes permanent storage to applications and services (and their users). PV resources are available cluster-wide, and are often backed with attached external storage. In fact, Kubernetes uses control plane interfaces through orchestration to link with external storage, so storage vendors must provide volume plug-ins that work with the Kubernetes codebase (called in-tree volume plug-ins).

Such plug-ins can pose issues for storage vendors and Kubernetes developers alike. From the vendor side, it means their code has to be compiled, packaged and shipped within a Kubernetes distribution. Not only does this expose their code, it also ties it to the Kubernetes release cycle. In turn this can pose testing issues for would-be users. From the users' perspective, it also limits their storage options to those plug-ins included with Kubernetes code base.

***Software-defined storage*** **distinguishes between the storage hardware where the bits reside and the storage controller software, which manages access to storage addresses,** reading and writing bits (or blocks, as is typical on solid-state devices), and integrity checks (and associated bad block lists, over provisioning, and so forth). Software-defined storage lets the storage system define and expose various types of storage to applications, such as object, block, and filesystem storage. It also manages the details behind the scenes to provide a consistent logical view of storage for application use, while handling data about where the data resides, in what format, what kind of storage units it uses, and so on.

To address these issues, the Kubernetes community introduce its Container Storage Interface (CSI) in 2017. CSI is a standard through which arbitrary block and file storage systems may be accessed within containerized workloads running on Kubernetes (or other orchestrator using CSI).

CSI makes the Kubernetes storage layer open and extensible. Third-party storage providers or vendors can use it to create and share volume plug-ins to expose their storage to Kubernetes. They no longer need to include those plug-ins with the Kubernetes code base, either.

CSI does a great job with block and file storage, but as the COSI GitHub status page asserts, "primitives for file/block storage do not extend well to object storage." (See the reference for a list of reasons why.) Thus, COSI—the Container Object Storage Interface—defines a set of abstractions to provision and manage object storage, defining a common object storage layer across multiple vendors.

The design is modeled on CSI, and has garnered support from makers of numerous open source and commercial storage systems. COSI defines a set of resources to work with object "buckets" (which are to objects as volumes are to blocks and files), to provision and manage object buckets across the data and application lifecycles. Using COSI, Kubernetes can manage object stores in a standard, native way. Storage vendors can expose their object stores via COSI, independent of the Kubernetes codebase. It's a win-win situation.

## Storage for Containers vs. Storage in Containers

Storage for containers exposes storage to a container or group of containers through an external mount point over a network. Sometimes known as container-ready storage, it can work with systems based on software-defined storage (SDS), network-attached storage (NAS), or storage-area networks (SANs). Container-ready storage is typically accessed via a vendor-defined or standard API.

It's important to understand that container-ready storage may not be an ideal solution for containers and their constituent apps and services, label notwithstanding. That's because relatively few such storage platforms have APIs that Kubernetes can use for dynamic provisioning and storage delivery.

Storage inside containers, deployed alongside containerized applications or services, works to the benefit of developers and IT admins alike. This approach containerizes storage services so they can be managed using Kubernetes orchestration and control. This approach leaves admins with less housekeeping to do (automation will handle that for them both quickly and accurately).

Because admins can run the storage platform, applications, and services on a uniform infrastructure, there's less learning curve involved (the same tools, commands, and automation applies across the board, rather than having to learn multiple sets of same). Often there's also less expense involved, because it's cheaper, easier, and less time-consuming to procure more of one kind of infrastructure than less of two or more kinds.

For developers, benefits come from self-service: Rather than working through storage admins to provision their applications and services with storage, they can provision friendly and elastic containerized storage services themselves. Additionally, its APIs are usually well-defined and understood, and easy to work with, test, and deploy.

Storage in containers involves a growing set of storage classes that range across many use cases. These include boot volumes, log files (circular and linear), transactional databases, and application data using traditional file and emerging object APIs, along with backup datasets, images and snapshots plus archival holdings.

Since object data alone covers an enormous number of data types and serves analytics applications that include Elastic, Cloudera, Spark, Splunk, Vertica, Weka and more, containerized apps need storage

access more than ever. CSI covers file and block storage access and management, while COSI handles object storage. But calls for storage access and services in containers is never-ending and nearly unlimited in size, scope and variety.

CHAPTER 2

# Storage Challenges for Containerized Apps

Because stateless apps carry the data and state information they need to do their jobs, storage challenges for containerized apps fall mostly on those of the stateful variety. That said, organizations should adopt storage layer software that is either open source or that works with the various cloud platforms they use (or would like to use).

This layer is least likely to pose interoperability or access problems if it's open source. Either way, storage layer software provides the ability to position containers where it makes best sense, and to move containers around if and when a change of location (and possibly, platform) is warranted.

## Persistent Storage for Stateful Apps

As we've discussed, stateful apps need persistent storage. In fact, few applications or services can do anything useful or interesting without some means for data storage and retrieval. This is a challenge for containers, which are by nature ephemeral and transient. They might live on one server for a while, then move over to a different server after that if an admin or an orchestrator dictates a move.

While containers keep their software and dependencies intact wherever they go, they deliberately don't store data—this helps them stay compact and predictable in size. VMs don't have such limitations: They operate as images that can be modified, then snapshotted

and saved. Containers work much the same way, except for data persistence. If the container hiccups or gets restarted, all data associated with its constituent applications or services gets lost, unless it has connections to a storage layer where such data can persist independently, but in close association to the container (wherever it may reside).

Though containers may have access to local storage, that may not be enough. Stateful applications require state, data, and configuration to persist across time and space. Thus, a database container needs a persistent store for its data—in fact, that's where the actual content of the database lives. In general, stateful applications require data to survive independently of the container itself (which can come and go quite frequently).

Local storage isn't enough, either: If the container moves to another location, it loses it connection to local storage (and the data it contains). In a nutshell, that's why stateful applications require access to a storage layer to provide them with the ability to keep state and configuration information around, along with the data that stateful applications and services expect and need to have at their beck and call.

## On-Demand, Dynamic Provisioning

Dynamic provisioning has shown itself to be a major improvement for containerized storage. Static provisioning was the order of the day before dynamic provisioning came along, but it had two major waste issues: time and storage space.

Static provisioning requires an administrator to work with a storage provider to obtain more storage space (additional volumes). The same thing applied to developers, who first had to calculate how much storage they might need, then request it from an administrator.

Developers creating stateful containerized applications have two major hurdles to jump. First and foremost, they must be able to provision

the storage their application or service needs both easily and quickly. Second, they must be sure that this application or service can access that storage so that the state information, configuration info, and data will persist as and when it must.

A proper containerized storage framework lets administrators provision volumes as needed from storage platforms that may reside on-premises in a public or private cloud. Kubernetes, through CSI and COSI, supports plug-ins to permit a container to mount the volumes it needs, after which it can start that container and tie that mounted volume to some directory accessible to the container. The same is true for object buckets and block stores—it all depends on what the containerized application or service needs and uses.

Dynamism comes into play as containers are instantiated and moved. The containerized applications tell the orchestrator what kinds of storage resources they need to run. The orchestrator examines the storage layer to identify the resources, obtain access, and expose the volumes or buckets needed while the application or service is running.

Should the application pause or restart, the orchestrator keeps the storage connection data handy, so when it resumes they, too, can carry on where they left off. The same general principle applies if an application or service workload moves from one cluster or cloud to another, except that storage units may need to be copied to another location, to meet associated performance, security, or compliance requirements. This should all be transparent to the end user.

Dynamic provisioning and association for containerized applications and services also means that as containers move or scale out, associated storage components move or scale out with them. This is built into the orchestrator, and lets developers and users take advantage of what the containerized environment can deliver without having to worry overmuch about the details involved in pauses, restarts, handoffs, and so forth.

# Automation for DevOps

Following DevOps approaches and practices for containerized apps and services and their storage means that organizations adopt CI/CD as both mantra and method. Automation is at the heart of this process, and provides these benefits:

- **Speed:** Computers can do things faster than humans. This isn't breaking news, but it's important to keep in mind for DevOps. Automation takes advantage of this characteristic, responding at computer speeds to alarms, alerts, and other events that require quick or immediate action.
- **Accuracy:** Automation, once tested and vetted, fumbles no further fingers at the keyboard. If it's right once, it's always right thereafter. Human input often includes input errors that can vary from simple typos to invalid instructions to potentially damaging mistakes, misconfigurations, or deletions. Automation is vastly more reliable and accurate than humans on the loose.
- **Scalability:** Without automation, cloud environments wouldn't scale, either up or down, period. Automation makes the kind of configurations, provisioning, and workload migration needed to support scaling usable and practical. The sheer scope and scale of the cloud, and its incredible uptake, all testify to that.
- **Agility:** Agility enables rapid provisioning of computer resources, so that cloud environments can spin up new containerized apps or services, VMs, or virtualized platforms, and storage in minutes. Automation applies across the lifecycle in a DevOps world, and results in shortened development cycles. It also offers more "what-ifs" or A/B tests to improve and enhance data, services, and the overall user experience.

- **Portability:** Portability is provided through standard, widely used APIs, protocols, tools and technologies. Portability lets workloads and data move where they provide the best value. Portability also brings flexibility—including the choice of cloud platforms and services based purely on merit and cost, with no fear of vendor lock-in—to the party.

CHAPTER 3

# Why Object Storage Is the Best Choice for Cloud-Native Development

As already stated, object storage provides a set of tools to accommodate huge volumes of data in many forms and formats, and covers the gamut from unstructured to semi-structured to structured data. Because it can handle all the data types that comes its way, object storage is an excellent fit for cloud-native, Kubernetes-based containerized applications and services (and the storage that supports them).

Kubernetes is a powerful container orchestration tool and provides a robust set of features, including storage solutions. Object storage, in particular, is an efficient and cost-effective way to provide storage for cloud-native applications.

## Object Storage Benefits

Object storage that can be deployed and managed by Kubernetes fits well with the needs of cloud-native applications. Object storage from the cluster can also reduce the need to configure and manage NAS, SAN, direct-attached storage (DAS), and other forms of persistent storage. These resource-intensive chores aren't needed when object storage is available, and explains why it's getting so much traction and buy-in in today's increasingly cloud-native world.

## Microservices

There are many other reasons why object storage makes the most sense for cloud-native applications and services, including microservices.

Microservices are predicated on APIs, so the RESTful API object storage is based on is a great fit for incorporating and using storage within these basic building blocks for containerized applications and services.

## Kubernetes

Kubernetes is all about orchestration and automation, independent of any geographic location. The means there is no anchor point keeping it weighed down and destroying application and data portability.

This, too, makes the familiar HTTP/HTML-based RESTful API object storage is known for a great fit as it allows anywhere access. Portability is preserved, which is crucial for cloud-based development and operations management.

## Web Scale

Cloud-native architecture can handle data at web scale, which is a key differentiator from block and file storage. The inability of those storage types to scale anywhere near as well as object storage disqualifies them for the most part. This is increasingly important as data creation across the globe explodes with the cloud, edge computing, the Internet of Things (IoT), and so on. The simple fact is that block and file storage can't keep up, leaving object storage as the default answer.

## The Data Tsunami


The best current estimates show that about 2.5 quintillion bytes of data are created every day, and that about 90% of the world's data has been created in the last two years alone.

In addition, information from Domo breaks down some of the chief types of data being created. Every day, for example, Zoom hosts 208,333 participants in video meetings; Instagram users post 347,222 stories; YouTube users upload 500 hours of video; Amazon ships 6,659 packages; and Netflix users stream 404,444 hours of video.

Most of that data will be kept in object storage, highlighting its importance now and into the future.

## Persistence

As discussed in the previous chapter, persistent storage for stateful apps can pose development and implementation challenges. While CSI drivers offered a workaround that can accommodate block and file storage, object storage needs minimal to no adaptation layer to deliver robust, reliable, web-scale storage.

In fact, object storage can achieve the same results as block and file storage. But it does so with fewer required resources (compute, networking, and storage), reduced hurdles (fewer roadblocks or constraints for sharing, synchronization, replication, snapshotting, and I/O activity), and at lower risk. Where file and block storage provide cloud-*ready* storage, object storage functions as *true* cloud-native storage.

## Native Architecture

Furthermore, object storage that's architected from the ground up to run as containerized services would be directly managed by Kubernetes. It can either be co-hosted with the app or hosted somewhere else, as needs, costs, and performance considerations dictate.

By contrast, cloud-ready block and file storage is commonly deployed independently and not managed by Kubernetes. This helps explain why object storage, managed, and run inside Kubernetes, offers so much more in terms of flexibility, automation, and integration.

## DevOps Flexibility

Yet another consideration is that some DevOps teams don't need or want compute and storage co-located together. In such a case, the perfect object storage solution is Kubernetes-based, but pre-packaged within its own Kubernetes distribution.

In fact, this method of delivery supports one-click install on bare-metal servers. The point here is that object storage provides options, and that's always a good thing for IT staff, as they can tune the solution for their specific operating environment.

## The SDS Advantage

When delivered as SDS, object storage runs on commodity, industry-standard servers, and is both hardware-independent and hardware-agnostic. This is the same blueprint that drove the design of containers and Kubernetes clusters, and still serves today as necessary pre-requisite for running Kubernetes clusters and containerized apps and services.

This independence has other benefits as well, including the avoidance of vendor lock-in and all the attendant problems of high costs, limited flexibility, and less ability to customize for infrastructure optimization.

# The Scality Solution

By now, you've seen that object storage is the best solution for a great many cloud-native development environments. It's the only technology that's truly scalable, usable, and affordable enough to accommodate web-scale data sets.

Throughout this Gorilla Guide, you've learned how modern object storage supports and enables cloud-based apps. You've also seen how cloud-native applications are designed around microservices that can leverage the advantages of object storage.

An ideal object storage system is lightweight and easy to operate, even in the case of edge deployments. Modern applications are not limited to running in a single cloud, so their object storage systems should be multi-cloud-capable.

As you explore options for object storage, put Scality at the top of your search list. They specialize in object storage for cloud-native scenarios, and have solutions that will help you get the most out of your cloud investments.

To learn more about modern object storage for cloud-native applications, check out the portfolio of Scality ARTESCA solutions with HPE.

# ABOUT SCALITY & HPE

Scality® storage propels companies to unify data management no matter where data lives—from edge to core to cloud. Our market-leading file and object storage software protects data on-premises and in hybrid and multi-cloud environments. With RING and ARTESCA, Scality's approach to managing data across the enterprise accelerates business insight for sound decision-making and maximum return on investment. To compete in a data-driven economy, IT leaders and application developers trust Scality to build sustainable, adaptable solutions. Scality is recognized as a leader by Gartner and IDC. Follow us @scality and LinkedIn. Visit www.scality.com, or subscribe to our company blog.

Hewlett Packard Enterprise

Hewlett Packard Enterprise is the global edge-to-cloud platform-as-a-service company that helps organizations accelerate outcomes by unlocking value from all of their data, everywhere. Built on decades of reimagining the future and innovating to advance the way people live and work, HPE delivers unique, open and intelligent technology solutions, with a consistent experience across all clouds and edges, to help customers develop new business models, engage in new ways, and increase operational performance. For more information, visit: www.hpe.com.

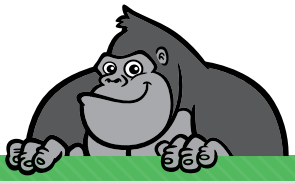
# ABOUT ACTUALTECH MEDIA

ActualTech Media

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

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

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



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