WD

Application Guide

Implementing storage for your AI factory with Kubernetes

November 2026

Table of Contents

Notices 3

Introduction 4

- Exploring the function of storage in AI factories 4
- Why choose Kubernetes for your AI factory 4
- What you'll find in this guide 5

Prerequisites 6

- Knowledge and experience 6
- Tools and software 6
- Infrastructure 7

Step-by-step 8

- Phase 1: Prepare the environment and install the Rook operator 8
- Phase 2: Deploy the Ceph cluster 9
- Phase 3: Create pools and set data protection 11
- Phase 4: Expose the S3 object endpoint 12
- Phase 5: Enable block and file access through CSI 13
- Phase 6: Deploy the data-management layer and verify the end state 14

In conclusion 15

Resources and references 16

- Explore Ultrastar capabilities 16
- Explore the underlying software 16

Notices

This document is only a reference and does not replace the advice of a data professional to configure any system in the same way, nor does it imply any recommendation, support, contribution, or relationship from WD with any organization that represents the open-source software utilized in this exercise. The end result should not be used in production without the proper vetting from a storage professional.

©2026 Western Digital Corporation or its affiliates. All rights reserved.

WD, the WD design, and Ultrastar are registered trademarks or trademarks of Western Digital Corporation or its affiliates in the US and/or other countries. Ceph and the Ceph logo are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries. The NVMe word mark is a trademark of NVM Express, Inc. All other marks are the property of their respective owners.

Product specifications subject to change without notice. Pictures shown may vary from actual products. Not all products are available in all regions of the world.

Introduction

Exploring the function of storage in AI factories

AI factories put data to work at industrial scale: dense GPU clusters running training and inference continuously, fed by datasets that grow faster than anyone could have imagined a decade ago. Much attention has been paid to the compute layer, but storage can be just as significant a challenge. AI factories both ingest and generate data—and without sufficient storage, all those expensive accelerators sit idle.

There's no one right type of storage for an AI factory. High-speed flash storage is valuable and necessary for some functions of the infrastructure, like live training I/O. But flash storage, especially today, comes at a high cost. Using it to build a multi-petabyte data lake is not economically sensible. You need durable, reachable, affordable storage where you can keep the training corpus, datasets, checkpoints, intermediate outputs, and every prior version you might need to reproduce a result. This storage, where the data lives, is the capacity tier. Here, cost-efficient hard disk drives (HDDs) hold the data, which is staged up to the flash cache that feeds the GPUs when a job needs it.

HDDs remain central to AI infrastructure—even though they're not the fastest storage available—because they make it economically possible to keep all your data on hand. The gap is not marginal: At petabyte scale, the difference in cost per terabyte between hard drives and enterprise flash dominates the economics of the capacity tier, and it compounds every time the corpus grows or you retain a checkpoint set rather than discarding it.

That changes how to approach the design. Rather than asking how much of your data you can afford to keep on flash, ask which data requires flash performance. For the rest—which, in a capacity tier, is nearly all of it—HDDs deliver the lower total cost of ownership at performance the access pattern can absorb. This guide shows you how to build a capacity tier for your AI factory using WD® Ultrastar® conventional magnetic recording (CMR) hard drives.

Why choose Kubernetes for your AI factory

Kubernetes has become a control plane for AI infrastructure, changing how storage is delivered. It offers several advantages for AI factories that enable distributed computing:

- Kubernetes enables the storage tier to be managed by the same control plane as compute, so they work together smoothly as the workload mix changes.
- Kubernetes doesn't include a storage system. Instead, it ships a Container Storage Interface (CSI), and operators fill the gap. In our approach in this guide, the Rook operator deploys and manages a full Ceph cluster as a Kubernetes workload, which means a petabyte-scale data lake becomes simply one more declarative component of the cluster.
- Data consumption is native, with no external provisioning requirements. Workloads reach the data lake through an S3 endpoint (for objects) or through PersistentVolumeClaims (for block and file). Data loaders, catalog databases, and training jobs all use familiar interfaces.

What you'll find in this guide

In this guide, we'll walk you through deploying the capacity tier for an AI factory: a multi-petabyte-scale data lake, on WD Ultrastar CMR drives, presented to Kubernetes through Ceph. At the end, you'll have a working capacity tier for an AI factory that will look like this:

- A Ceph storage cluster deployed and managed by the Rook operator on an existing Kubernetes cluster
- WD Ultrastar CMR drives in a JBOD enclosure configured as Ceph OSDs, with metadata offloaded to NVMe storage
- An S3 object endpoint as the primary data-lake interface
- Block and file access through the Ceph CSI driver for workloads needing persistent volumes
- A PostgreSQL dataset catalog and a data-loader job reading from S3, showing how a real workflow finds and consumes data

Our focus in this guide is to offer practical advice for setting up HDD storage for your AI factory using Kubernetes. Note, however, that we do not cover deploying Kubernetes from scratch, deploying a high-performance fabric, racking or cabling hardware, setting up security solutions, configuring GPUs or AI accelerators, or selecting flash tiers. Every environment is different, and your experiences may differ depending on software versioning, network capacity, and other factors.

CMR and UltraSMR in the capacity tier

Hard drives reach high capacity in more than one way. CMR writes data in non-overlapping tracks. Shingled magnetic recording (SMR) overlaps tracks to raise areal density, and the WD UltraSMR implementation extends that approach with large block encoding and an advanced error correction algorithm that increases tracks per inch. The result is more capacity in the same drive footprint: Ultrastar UltraSMR drives reach 34 TB in fixed capacity, compared with 28 TB for WD CMR drives in the same 3.5-inch form factor. Read performance is comparable between the two formats, and UltraSMR is well established for data that is read frequently but rewritten rarely—a profile that fits many data lakes.

Each format has its place, and the right choice depends as much on the storage software as on the drives. This guide builds on Ceph, and Ceph zoned-storage support remains a development effort targeting non-overwrite workloads rather than a production-ready feature. A Rook-managed Ceph cluster is therefore the clearer fit for CMR today, which is why we specify CMR drives throughout this guide. Look for a future guide on UltraSMR deployment beneath a stack designed to take advantage of zoned devices.

Prerequisites

Knowledge and experience

We've written this guide for enterprise storage architects. The guide assumes working familiarity with:

- Linux administration
- Object storage
- The Amazon S3 API or a similar cloud object storage API
- High-level Ceph concepts (e.g., OSDs, pools, placement groups, and gateways)
- High-level Kubernetes concepts

You do not need to have run a Ceph cluster in production or have used Kubernetes for daily operations—if you've deployed workloads to a cluster and can read pod status and logs, you should be able to follow the discussion here. We'll also explore Kubernetes-specific concepts where relevant.

Tools and software

For this guide, we recommend using Rook to deploy most of the stack: You install the operator, and Rook runs Ceph as containers. Table 1 outlines the tools you'll need to move forward. We've highlighted up-to-date versions as of this writing in September 2026, but be sure to check the Rook release notes and the Ceph releases page for the latest.

Table 1: Tools you'll need for your AI factory with Kubernetes.

Tool	Purpose
kubectl	Cluster administration, configured with an admin context
Rook operator (v1.20.6 or v1.20.7)	Deployment and management of Ceph as a Kubernetes workload
Ceph (Tentacle v20.2.4 or later)	The storage system itself, run in containers by Rook
Ceph container images	From the official Ceph repository on Quay; pin a full version-and-build tag in production
Ceph toolbox pod	Ad hoc Ceph CLI access for validation and troubleshooting; match toolbox image to the deployed Ceph version
S3 client (AWS CLI or s3cmd)	Verification and use of the object endpoint
PostgreSQL	Dataset catalog, deployed in-cluster on Ceph block storage

How storage relates to Kubernetes

Kubernetes does not include a storage system. Instead, it defines an interface (the CSI) and lets storage platforms plug into it. Workloads request storage through a PersistentVolumeClaim, and a CSI driver provisions the actual capacity behind it. Kubernetes also supports operators, controllers that extend the cluster with new resource types and manage their lifecycle. The Rook operator deploys and runs Ceph inside Kubernetes as a managed workload. This enables the storage tier to work the same way as everything else in the Kubernetes cluster, instead of acting as an entirely separate system.

Infrastructure

Before moving further, make sure you have the following in place:

- A running Kubernetes cluster (control plane and worker nodes) with administrative access
  - If you need to stand up a vanilla cluster to follow along, we recommend referencing documentation from Kubernetes.
- Capacity drives (WD Ultrastar CMR HDDs), configured as one Ceph OSD per drive in Phase 2
- Dedicated storage nodes with a WD Ultrastar JBOD (Data60 or Data102) attached over SAS, with the HDDs and an NVMe metadata device visible to the operating system
  - Make sure the storage nodes are part of the Kubernetes cluster (or joinable to it), so Rook can schedule Ceph daemons onto them.
- A working network fabric with capacity for a separate Ceph cluster network

You may also want GPU compute and an NVMe flash tier either deployed or planned as a separate effort. While we don't cover these elements in this guide and they're not a hard requirement for a proof of concept, you'll likely need them to scale your AI factory to production.

A note on hardware topology

As Figure 1 shows, this guide uses SAS-attached JBODs, which keep drive capacity and server compute independently scalable. The same deployment also works on storage servers with internal drive bays, where the drives are local to the node instead of in an external enclosure. Which approach you choose will depend on your specific infrastructure and needs.

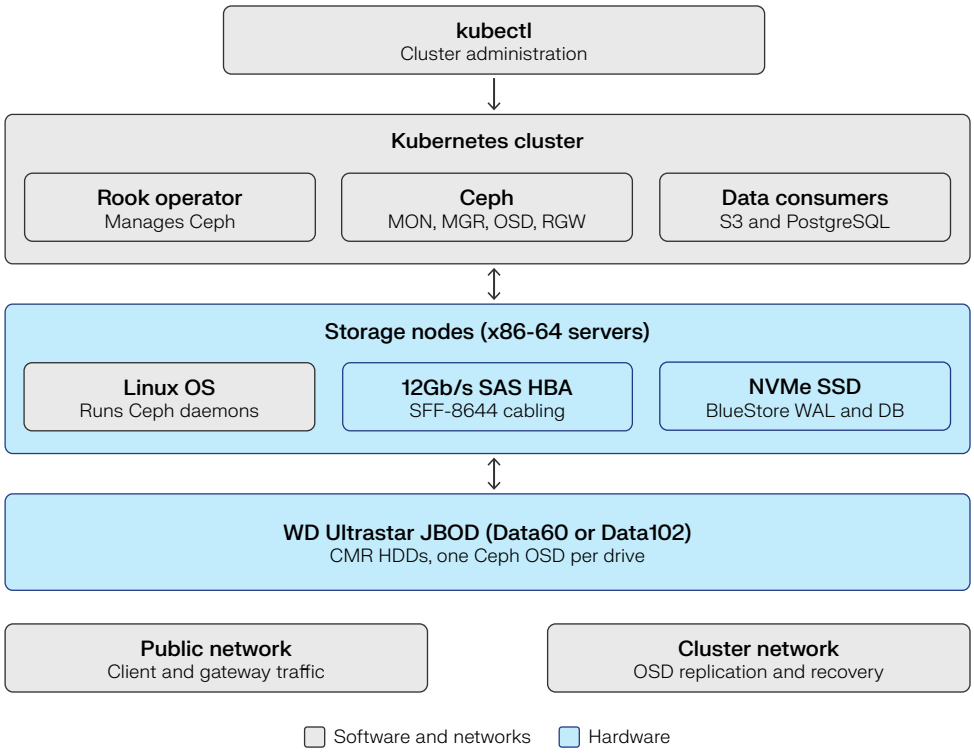

Figure 1: Topology of the solution for this how-to guide.

Step-by-step

In this section, we'll explore how to build the capacity tier in six phases. Each phase explains what you are creating and why, then walks you through the steps and offers a checkpoint to confirm that the phase is complete. Where you'll need to make design decisions for your environment, such as data protection and metadata placement, we call them out so you can adapt them to your own infrastructure.

How Rook works

In this guide, you will not install Ceph itself. Instead, you will install an operator, then describe the storage cluster you want as Kubernetes resources. The operator will read those descriptions and do the work: formatting drives, starting daemons, forming quorum, and continuously reconciling the running cluster against what you declared. If a daemon dies, the operator restarts it, and if you change the declaration, the operator converges toward the new state. This is why most of what follows is applying YAML rather than running installers.

Phase 1: Prepare the environment and install the Rook operator

Why does this phase matter?

Rook is conservative about consuming disks: It will only use devices that are completely unformatted, with no partition table, filesystem signature, or logical volume metadata. This protects you from an operator wiping a drive you meant to keep, but it means Rook can silently skip any drives that were previously used for a different purpose. Many failed Rook deployments trace back to this problem. By taking a few minutes to verify the drives first, you reduce your chances of staring at a confusing empty cluster down the line.

Complete the steps

Step 1: Confirm the drives are visible and clean.

On each storage node, enumerate the block devices:

```
lsblk -o NAME,SIZE,ROTA,TYPE,FSTYPE,MOUNTPOINT
```

You are looking for three things. Every HDD in the JBOD should appear, and the count should match the enclosure population. The ROTA column should read 1 for the HDDs and 0 for the NVMe metadata device, which confirms the OS is identifying the media type correctly. The FSTYPE column should be empty for every device you intend to hand to Ceph. If a device shows a filesystem or partition, it will not be consumed until it is wiped.

Step 2: Label the storage nodes.

Ceph daemons should run only where the drives are. Without a label, Rook has no way to distinguish a storage node from a general worker, and the operator may try to place daemons on hosts with no disks:

```
kubectl label node <storage-node-name> role=storage-node
```

Step 3: Install the operator.

We recommend a Helm chart, since it gives you a clearly versioned, upgradeable release:

```
helm repo add rook-release https://charts.rook.io/release
helm repo update
kubectl create namespace rook-ceph
helm install rook-ceph rook-release/rook-ceph \
  --namespace rook-ceph --create-namespace
```

This installs the Custom Resource Definitions, the RBAC rules the operator needs, and the operator deployment itself. The operator is now watching for a CephCluster resource to act on.

Checkpoint.

Every intended OSD device and metadata device is accounted for and unformatted on each node, and the operator pod is Running with no restarts:

```
kubectl -n rook-ceph get pods
```

Phase 2: Deploy the Ceph cluster

Why does this phase matter?

This is where the hardware becomes a storage system. To move forward, you'll need to understand three types of daemons that will come up in the following pages:

- **Monitors (MONs)** maintain the authoritative map of the cluster: which OSDs exist, which are up, and how data is distributed. Clients consult the monitors to learn where data lives. Monitors operate as a quorum, which is why you deploy an odd number, typically three, across separate hosts.
- **Managers (MGRs)** handle metrics, the dashboard, and orchestration modules. They are not in the data path, but we recommend running two so your monitoring infrastructure can keep running in the event of a host failure.
- **Object storage daemons (OSDs)** are the workers: one OSD per drive, each managing its disk directly through Ceph's BlueStore backend. BlueStore writes to the raw block device instead of through a filesystem, which removes a layer of overhead that matters a great deal on rotational media.

One place to use NVMe SSDs

BlueStore keeps two things apart from the data itself: a write-ahead log (WAL) and an internal metadata database (DB) holding object locations, checksums, and allocation state. On an HDD, every metadata update becomes a seek, which competes with the sequential reads and writes you need your storage to handle. To resolve this problem, you can place the WAL and DB on an NVMe device instead, letting the HDD spend its time on data.

Size this metadata device to a few percent of the OSD capacity it serves, accounting for the fact that one NVMe device typically backs many OSDs on the same node. The downside of this approach is that the metadata device is a shared failure domain: if it fails, every OSD depending on it also goes down. Ceph will take some time to rebuild because the rebuild is larger than a single-drive failure. This is worth factoring into your failure-domain planning.

Complete the steps

Step 1: Define the cluster.

The CephCluster resource describes the topology:

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
 name: rook-ceph
 namespace: rook-ceph
spec:
 cephVersion:
  image: quay.io/ceph/ceph:v20.2.4-20260818
 dataDirHostPath: /var/lib/rook
 mon:
  count: 3
  allowMultiplePerNode: false
 mgr:
  count: 2
 storage:
  useAllNodes: false
  useAllDevices: false
  nodes:
   - name: <storage-node-name>
     deviceFilter: "^sd."
     config:
       metadataDevice: "nvme0n1"
```

We recommend setting useAllNodes and useAllDevices to false, because it means you list what Ceph may consume rather than letting it discover everything—preventing surprises as the cluster grows. We also recommend pinning a full version-and-build image tag, so every daemon runs an identical Ceph version.

Next, the operator starts monitors, waits for quorum, and starts managers. Then, it runs a provisioning job on each storage node that inspects the listed devices, prepares each eligible disk with BlueStore, and starts an OSD for it. On a densely populated JBOD, this can take time.

Step 2: Deploy the toolbox.

The toolbox is a pod with the Ceph CLI available. This is how you inspect the cluster from here on:

```
kubectl apply -f https://raw.githubusercontent.com/rook/rook/master/deploy/examples/toolbox.yaml
```

The operator doesn't upgrade the toolbox for you, so be sure to match the toolbox image to your deployed Ceph version. A mismatched CLI can report misleading results.

Checkpoint.

From inside the toolbox, confirm the cluster is healthy and every drive became an OSD:

```
ceph status
ceph osd tree
```

You want HEALTH_OK, a monitor quorum of three, and an OSD count matching your drive count with all OSDs up and in. The OSD tree also shows the CRUSH hierarchy, the map Ceph uses to decide where data goes. Confirm your storage nodes appear as distinct hosts.

Phase 3: Create pools and set data protection

Why does this phase matter?

When you create pools, you decide how much of your raw storage capacity converts into usable capacity, and how much failure the cluster can survive. This is an important decision, and therefore one you should adapt to your own organization's requirements.

Ceph offers two possible protection schemes:

- **Replication** stores complete copies of every object. As one example, three-way replication will survive two simultaneous failures and yield roughly 33 percent usable capacity. Recovery is generally fast, because rebuilding means copying an existing replica, and read performance is good, because any replica can serve a request.
- **Erasure coding** splits each object into data chunks and computes parity chunks, distributing all of them across failure domains. For example, an 8+3 profile splits data into eight chunks with three parity chunks, survives three simultaneous failures, and yields roughly 73 percent usable capacity. In exchange for higher usable capacity, you spend some CPU cycles on encoding and decoding, see higher latency on small objects, and increase the cost of recovery, since rebuilding a lost chunk requires reading from many devices rather than copying one replica.

For a petabyte-scale AI data lake holding large, sequentially accessed objects, erasure coding is usually the right default due to the dramatic increase in capacity. On the same raw hardware, 8+3 delivers more than twice the usable space of three-way replication.

If you choose erasure coding, you'll need to select the right profile for your node count. An 8+3 profile produces eleven chunks, and for host-level failure tolerance, each chunk needs a separate host. That means you'll need eleven storage nodes, with additional headroom so the cluster can recover after a host loss rather than sitting degraded. If your deployment is smaller, use a narrower profile such as 4+2, or start with replication and migrate later.

Replicating metadata pools

If you choose erasure coding, make sure your RADOS Gateway's (RGW) index and metadata pools still use three-way replication. Metadata is a requirement for rebuilding erasure coding data, so you'll need full copies of that data available quickly. Fortunately, because the metadata pools are small, the capacity cost is negligible.

Complete the steps

Step 1: Create the object store.

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: ai-datalake
  namespace: rook-ceph
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPool:
    failureDomain: host
    erasureCoded:
      dataChunks: 8
      codingChunks: 3
    deviceClass: hdd
  gateway:
    port: 80
    instances: 2
```

Setting failureDomain: host tells Controlled Replication Under Scalable Hashing (CRUSH) to distribute chunks so no two land on the same server, which is what makes host-level failure survivable. Setting deviceClass: hdd keeps data on the capacity drives.

Running two gateway instances gives you availability and lets you spread client load. We recommend scaling this with your ingest and retrieval demands, since the gateway can be a throughput bottleneck in an object deployment.

Checkpoint.

Confirm the pools exist with the expected protection, and that the cluster returned to health after placement groups settled:

```
ceph osd pool ls detail
ceph df
ceph status
```

Placement group rebalancing takes time after pool creation, and transient warnings during that period are expected. What matters is that the cluster converges back to HEALTH_OK and that ceph df reports usable capacity consistent with your protection scheme. If usable capacity looks wrong, the profile is not doing what you intended—go ahead and fix that now, instead of waiting until the lake is full.

Phase 4: Expose the S3 object endpoint

Why does this phase matter?

Ceph's native protocol is RADOS, which AI tooling generally does not speak. Instead, nearly every data loader, training framework, and catalog tool utilizes S3. For this reason, we use the RGW as the translation layer. The RGW presents a standard S3 API and stores objects in the pools you just created, turning a Ceph cluster into a functional data lake.

RGW is also the reason the previous phase separated metadata from bulk data. The gateway maintains small, constantly updated bucket indexes and user metadata alongside large, comparatively static object data. Those two types of workloads require different pools.

Complete the step

Step 1: Create a user for access credentials.

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: datalake-user
  namespace: rook-ceph
spec:
  store: ai-datalake
  displayName: "AI data lake user"
```

Rook generates the access and secret keys and places them in a Kubernetes Secret, which is how workloads will consume them without credentials being written into manifests.

Checkpoint.

Verify a full round trip from an S3 client: create a bucket, upload an object, list it, and download it back. A successful put and get confirms the entire stack is working: You have a data lake.

Phase 5: Enable block and file access through CSI

Why does this phase matter?

Object storage such as S3 is right for datasets, but wrong for a database. PostgreSQL databases need a POSIX filesystem on a block device with low-latency random writes. That's why we have the CSI.

Rook deploys the Ceph CSI drivers alongside the cluster, enabling the same Ceph deployment to serve block volumes and shared filesystems, in addition to objects, through the standard Kubernetes provisioning path. A workload requests storage with a PersistentVolumeClaim, the CSI driver provisions it in Ceph, and the volume is attached. This approach allows us to move forward without a separate storage system or the need for manual provisioning.

Complete the steps

Step 1: Create a replicated block pool and its StorageClass.

Block volumes backing a database should use replication rather than erasure coding, for the same reason metadata pools do: the access pattern is small and random.

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicated-block-pool
  namespace: rook-ceph
spec:
  failureDomain: host
  replicated:
    size: 3
  deviceClass: hdd
```

Then create a StorageClass referencing that pool, which is what workloads will name in their claims.

Step 2 (optional): Create a shared filesystem.

If your workloads need concurrent read-write access from multiple pods, a CephFilesystem provides that. However, it introduces metadata server (MDS) daemons, so deploy it only if a workload requires shared access.

Checkpoint.

Create a small throwaway PVC against the block StorageClass and confirm it reaches Bound, then delete it. A bound claim proves dynamic provisioning works end to end.

Phase 6: Deploy the data-management layer and verify the end state

Why does this phase matter?

A data lake without a catalog is just a bucket full of files nobody can find. In practice, training pipelines don't enumerate object storage to discover datasets; they query a catalog that records what exists, which version it is, where it lives, and what it contains. This phase builds a minimal version of that pattern, making the deployment usable.

Complete the steps

Step 1: Deploy PostgreSQL as the catalog.

Run it as a StatefulSet with a PersistentVolumeClaim against the block StorageClass from Phase 5. By using StatefulSet rather than Deployment, you give the database a stable identity and keep its volume attached across restarts.

Step 2: Create the catalog schema.

Keep it small: a dataset table with a name, version, S3 bucket and prefix, object count, total size, checksum, and timestamp. This catalog stores pointers and metadata, never the data itself. Datasets stay in object storage, where capacity is cheap, while the catalog holds the index that makes them findable.

Step 3: Deploy the data-loader job.

The loader is a Kubernetes Job that reads its S3 credentials from the Secret Rook generated in Phase 4, connects to the RGW endpoint, enumerates the objects in a dataset prefix, computes size and object count, and writes a catalog row. Run it against a sample dataset to populate the catalog. This step replicates the real-world training workflow in miniature.

Step 4: Verify the deployed end state.

Walk the full chain in order: query the catalog and confirm it returns the dataset record, take the S3 location from that record and retrieve the objects from the gateway, then confirm the cluster reports HEALTH_OK and that ceph df shows capacity consumed consistent with what you uploaded. That last check closes the loop, confirming the bytes are on the CMR drives under the protection scheme you configured.

Checkpoint.

The catalog is queryable, its rows resolve to real objects in the lake, retrieval succeeds through the S3 endpoint, and the cluster is healthy.

In conclusion

If you've followed this guide, you should have a functional Kubernetes deployment with a data lake—congratulations. While validation, metrics, and troubleshooting are out of scope for this guide, you'll want to test your deployment thoroughly before putting it to work in production. When the time comes, though, you'll be ready to implement your AI factory and reap the benefits of a modern AI infrastructure.

Resources and references

Explore Ultrastar capabilities

- Ultrastar DC HC590 product page
- Ultrastar DC HC590 data sheet
- Ultrastar JBOD platforms (Data60 / Data102) enclosure overview
- Enterprise data center storage hub

Explore the underlying software

- Rook documentation
- Ceph documentation

5601 Great Oaks Parkway
San Jose, CA 95119, USA
www.wd.com

©2026 Western Digital Corporation or its affiliates. All rights reserved.

Western Digital, the Western Digital design, and the Western Digital logo are registered trademarks or trademarks of Western Digital Corporation or its affiliates in the US and/or other countries. microSD is a trademark of SD-3C, LLC. All other marks are the property of their respective owners.