A **Micron** Reference Architecture

Micron® 9300 MAX NVMe™ SSDs + Red Hat® Ceph Storage

Reference Architecture

John Mazzie, Senior Storage Solution Engineer
Tony Ansley, Principle Technical Marketing Engineer

systems software storage memory

Contents

Executive Summary

This document describes an example configuration of a performance-optimized Red Hat® Ceph Storage (RHCS) cluster using Micron® NVMe™ SSDs, standard x86 architecture rack-mount servers, and 100 GbE networking.

It details the hardware and software building blocks used to construct this reference architecture (including the Red Hat Enterprise Linux OS configuration, network switch configurations, and Ceph tuning parameters) and shows the performance test results and measurement techniques for a scalable 4-node RHCS architecture.

Optimized for block performance while also providing very high performance object storage, this all-NVMe solution provides a rack-efficient design to enable:

Faster deployment: The configuration has been pre-validated, optimized, and documented to enable faster deployment and faster performance than using default instructions and configuration.

Balanced design: The right combination of NVMe SSDs, DRAM, processors, and networking ensures subsystems are balanced for performance.

Broad use: Documenting complete tuning and performance characterization across multiple IO profiles for broad deployment across multiple uses.

Our testing illustrated exceptional performance results for 4KB random block workloads and 4MB object workloads.

4KB Random Block Performance		
IO Profile	IOPS	Avg. Latency
100% Read	2,099,444	1.5ms
70%/30% R/W	903,866	W: 6.85ms R: 2.11ms
100% Writes	477,029	6.7ms

4MB Object Performance		
IO Profile	GB/s	Avg. Latency
100% Sequential Read	48.36	52.16ms
100% Random Read	43.65	28.46ms
100% Random Writes	23.24	26.9ms

Tables 1a and 1b - Performance Summary

Why Micron for this Solution

Storage (SSDs and DRAM) represent a large portion of the value of today's advanced server/storage solutions. Micron's storage expertise starts at memory technology research, innovation, and design and extends to collaborating with customers on total data solutions. Micron develops and manufactures the storage and memory products that go into the enterprise solutions we architect.

Micron Reference Architectures

Micron Reference Architectures are optimized, pre-engineered, enterprise-leading solution templates for platforms that are co-developed between Micron and industry-leading hardware and software companies.

Designed and tested at Micron's Storage Solutions Center, they provide end users, system builders, independent software vendors (ISVs), and OEMs with a proven template to build next-generation solutions with reduced time investment and risk.

Ceph Distributed Architecture Overview

A Ceph storage cluster consists of a multiple of Ceph monitor nodes and data nodes for scalability, fault-tolerance, and performance. Ceph stores all data as objects regardless of the client interface used. Each node is based on industry-standard hardware and uses intelligent Ceph daemons that communicate with each other to:

- Store, retrieve, and replicate data objects
- Monitor and report on cluster health
- Redistribute data objects dynamically
- Ensure data object integrity
- Detect and recover from faults and failures

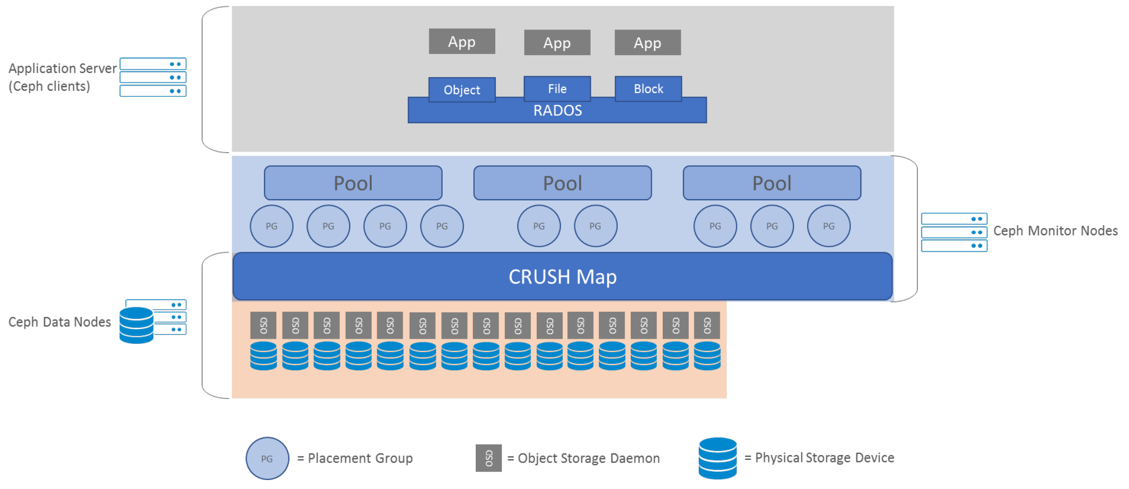

Figure 1 - Ceph Architecture

To the application host servers that read and write data, a Ceph storage cluster looks like a simple pool where data is stored. However, the storage cluster performs many complex operations in a manner that is completely transparent to the application server. Ceph clients and Ceph Object Storage Daemons (Ceph OSDs or OSDs) both use the Controlled Replication Under Scalable Hashing (CRUSH) algorithm for storage and retrieval of objects.

For a Ceph client, the storage cluster is very simple. When a Ceph client reads or writes data (referred to as an I/O context), it connects to a logical storage pool in the Ceph cluster. The figure above illustrates the overall Ceph architecture, with concepts described in the sections that follow.

Clients write to Ceph storage pools while the CRUSH ruleset determines how placement groups get distributed across OSDs.

Pools: Ceph clients store data objects in logical, dynamic partitions called pools. Administrators can create pools for various reasons such as for particular data types, to separate block, file and object usage, application isolation, or to separate user groups (multi-tenant hosting). The Ceph pool configuration dictates the number of object replicas and the number of placement groups (PGs) within the pool. Ceph storage pools can be either replicated or erasure coded, as appropriate for the application and cost model. Additionally, pools can “take root” at any position in the CRUSH hierarchy, allowing placement on groups of servers with differing performance characteristics, which allows for the optimization of different storage workloads.

Object storage daemons: Object storage daemons (OSDs) store data and handle data replication, recovery, backfilling, and rebalancing. They also provide some cluster state information to Ceph monitors by checking other Ceph OSDs with a heartbeat mechanism. A Ceph storage cluster configured to keep three replicas of every object requires a minimum of three OSDs, two of which need to be operational to process write requests successfully. Ceph OSDs roughly correspond to a file system on a physical hard disk drive.

Placement groups: Placement groups (PGs) are shards, or fragments, of a logical object pool that are composed of a group of Ceph OSD that are in a peering relationship. PGs provide a means of creating replication or erasure coding groups of coarser granularities than on a per-object basis. A larger number of placement groups (e.g., 200 per OSD or more) leads to better balancing.

CRUSH map: The CRUSH algorithm determines how to store and retrieve information from data nodes. CRUSH enables clients to communicate directly to OSDs on data nodes rather than through an intermediary service. By doing so, this removes a single point of failure from the cluster. The CRUSH map consists of a list of all OSDs and their physical location. Upon initial connection to a Ceph-hosted storage resource, the client contacts a Ceph monitor node for a copy of the CRUSH map, which is used by the client to directly manage the target OSDs.

Ceph monitors (MONs): Before Ceph clients can read or write data, they must contact a Ceph MON to obtain the current CRUSH map. A Ceph storage cluster can operate with a single MON, but this introduces a single point of failure. For added reliability and fault tolerance, Ceph supports an odd number of monitors in a quorum (typically three or five for small to mid-sized clusters). Consensus among various MON instances ensures consistent knowledge about the state of the cluster.

Reference Architecture – Overview

Micron designed this reference architecture on the Intel® Purley architecture using Xeon® Platinum 8168 processors. This combination provides a performance-optimized server platform while yielding an open, cost-effective software-defined storage (SDS) platform suitable for a wide variety of use cases such as OpenStack™ cloud, video distribution and transcoding, and big data storage.

The Micron 9300 MAX NVMe SSD offers tremendous performance with low latencies. Capacity per rack unit (RU) is maximized with ten 12.8TB NVMe SSDs per 1U storage node. This entire reference architecture takes up seven RUs consisting of three monitor nodes and four data nodes. Using this reference architecture as a starting point, administrators can add additional data nodes 1RU and 64TB at a time.

Two Mellanox ConnectX®-5 100 GbE network cards handle network throughput, one for the client/public network traffic and a second for the internal Ceph replication network. Mellanox ConnectX-4 50 GbE network cards installed in both the clients and monitor nodes connect to the storage network.

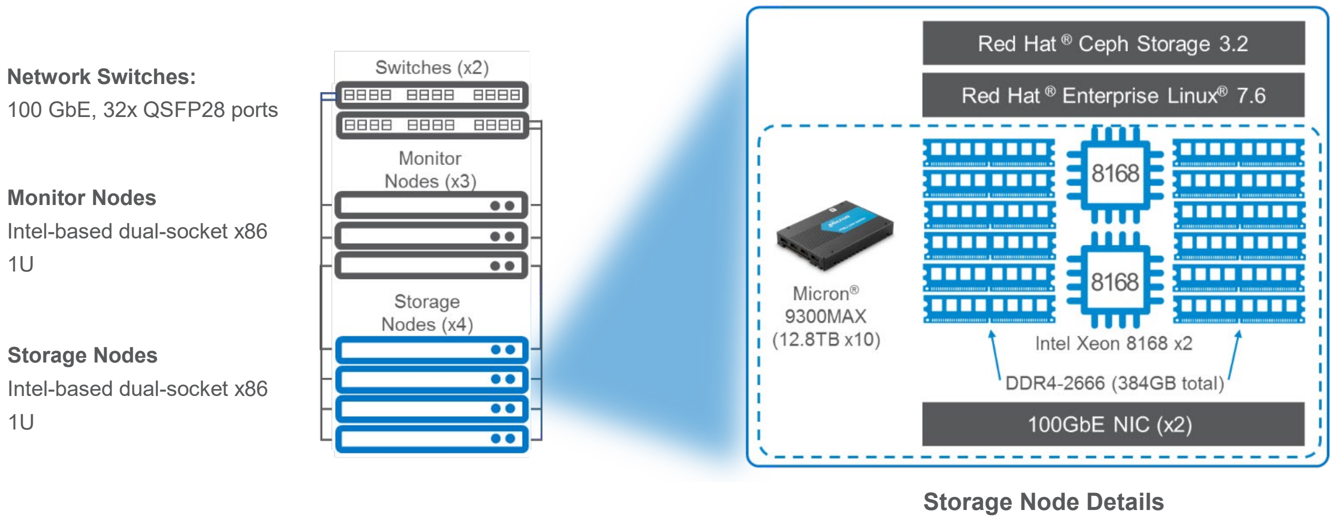

Figure 2 – Micron Reference Architecture

Software

This section details the software versions used in the reference architecture.

Red Hat Ceph Storage

Red Hat collaborates with the global open source Ceph community to develop new Ceph features, then packages changes into predictable, stable, enterprise-quality releases. Red Hat built Red Hat Ceph Storage on the open source Ceph community Luminous version 12.2, to which Red Hat was a leading code contributor.

As a self-healing, self-managing, unified storage platform with no single point of failure, Red Hat Ceph Storage decouples software from hardware to support block, object, and file storage on standard servers, using either HDDs and SSDs, significantly lowering the cost of storing enterprise data. OpenStack also uses Red Hat Ceph Storage along with services, including Nova, Cinder, Manila, Glance, Keystone, and Swift, and it offers user-driven storage lifecycle management. Voted the No. 1 storage option by

OpenStack users, Ceph is a highly tunable, extensible, and configurable architecture, well suited for archival, rich media, and cloud infrastructure environments.

Among many of its features, Red Hat Ceph Storage provides the following advantages to this reference architecture:

- Block storage integrated with OpenStack, Linux, and KVM hypervisor
- Data durability via erasure coding or replication
- Red Hat Ansible automation-based deployment
- Advanced monitoring and diagnostic information with an on-premise monitoring dashboard
- Availability of service-level agreement (SLA)-backed technical support
- Red Hat Enterprise Linux (included with subscription) and the backing of a global open source community

Red Hat Enterprise Linux

In need of a high-performance operating system environment, enterprises depend on Red Hat® Enterprise Linux® (RHEL) for scalable, fully supported, open source solutions. Micron used version 7.6 of Red Hat Enterprise Linux in this reference architecture due to its performance, reliability, and security, as well as its broad usage across many industries. Supported by leading hardware and software vendors, RHEL provides broad platform scalability (from workstations to servers, to mainframes) and consistent application environment across physical, virtual, and cloud deployments.

Software by Node Type

Table 2 shows the software and version numbers used in the Ceph monitor and storage nodes

Operating System	Red Hat Enterprise Linux	7.6
Storage Software	Red Hat Ceph Storage	3.2
NIC Driver	Mellanox OFED Driver	4.5-1.0.1.0

Table 2 – Software Deployed on Ceph Data and Monitor Nodes

The software used on the load generation client is the same as that used on the Ceph data and monitor nodes. The open source FIO storage load generation tool version 3.1.0 is also installed. FIO is configured with the `librbd` module.

Hardware by Node Type

Ceph Data Node

The Ceph data node in this RA is a x86 architecture server that hosts two or more OSDs and the physical storage devices used by those OSDs. While this RA used a server product available for purchase from one vendor, this RA does not make any recommendations regarding any specific server vendor or implementation, focusing on the overall solution architecture built around Intel® Xeon® processors and architecture. Table 3 below provides the details for the server architecture used for this Ceph data node role.

Server Type	x86 (dual-socket) 1U with PCIe Gen3 ("Purley")
CPU (x2)	Intel Xeon Platinum 8168 (24 cores, 2.7 GHz base)
DRAM (x12)	Micron 32GB DDR4-2666 MT/s, 384GB total per node
NVMe (x10)	Micron 9300 MAX NVMe SSDs, 12.8TB each
SATA (OS)	Micron 5100 SATA
Network	2x Mellanox ConnectX-5 100 GbE dual-port (MCX516A-CCAT)

Table 3 – Storage Nodes Hardware Details

Ceph Monitor Node

The Ceph monitor node in this RA is a x86 architecture server that hosts two or more OSDs and the physical storage devices used by those OSDs. As with the Ceph data node, this RA does not make any recommendations regarding any specific server vendor or implementation, focusing on the overall solution architecture built around Intel Xeon processors and architecture. Table 4 below provides the details for the server architecture used for this Ceph data node role.

Server Type	x86 (dual-socket) 1U with PCIe Gen3 ("Broadwell")
CPU (x2)	2x Intel Xeon 2690v4 (14 cores, 2.6 GHz base)
DRAM (x8)	Micron 16GB DDR4-2400 MT/s, 128GB total per node
SATA (OS)	Micron 5100 SATA
Network	1x Mellanox ConnectX-4 50GbE single-port (MC4X413A-GCAT)

Table 4 – Monitor Nodes Hardware Details

Micron Components Used

Micron 9300 MAX NVMe SSDs

The Micron 9300 series of NVMe SSDs is our flagship performance family and our third generation of NVMe SSDs. The 9300 family has the right capacity for demanding workloads, with capacities from 3.2TB to 15.36TB in mixed-use and read-intensive designs. The 9300 is also the first offering from Micron to provide "no compromise" read and write performance by offering balanced 3500 MB/s throughput on certain models. [1]

[1] Offered on 9300 PRO 7.68TB and 15.36TB and 9300 MAX 6.4TB and 12.8TB models.

This RA uses the 9300 MAX 6.4TB. Table 6 summarizes the 9300 MAX 12.8TB specifications.

Model	9300 MAX	Interface	PCIe Gen 3 x4
Form Factor	U.2	Capacity	12.8TB
NAND	Micron® 3D TLC	MTTF	2M device hours
Sequential Read	3.5 GB/s	Random Read	850,000 IOPS
Sequential Write	3.5 GB/s	Random Write	310,000 IOPS
Endurance	144.8 PB	Status	Production

Note: GB/s measured using 128K transfers, IOPS measured using 4K transfers. All data is steady state. Complete MTTF information can be provided by your Micron sales associate.

Table 5 – 9300 MAX 12.8TB Specifications Summary

Network Switches

We used one 100 GbE switch (32x QSFP28 ports each). For production purposes, we recommend that two separate network segments be used for redundancy purposes – one for the client network and the second for the Ceph storage network. The switches used are based on the Broadcom "Tomahawk" switch chip.

Model	Supermicro SSE-C3632SR
Software	Cumulus Linux 3.4.2

Table 6 – Network Switches (Hardware and Software)

The SSE-C3632S Layer 2/3 Ethernet Switch provides an Open Networking compliant solution, providing customers with the ability to maximize the efficient and flexible use of valuable data center resources while providing an ideal platform for managing and maintaining those resources in a manner in tune with the needs of the organization.

Mellanox ConnectX -5 EN Dual Port NICs

ConnectX-5 EN Network Controller offers dual ports of 10/25/50/100 Gb/s Ethernet connectivity and advanced offload capabilities while delivering high bandwidth, low latency, and high computation efficiency for high performance, data-intensive, and scalable HPC, cloud, data analytics, database, and storage platforms.

Planning Considerations

The following topics provide information to enhance the overall experience of the solution and ensure that the solution is scalable while maximizing performance.

Number of Ceph Storage Nodes

At least three (3) storage nodes must be present in a Ceph cluster to become eligible for Red Hat technical support. While this RA uses four data nodes, additional nodes can provide scalability and redundancy. Four (4) storage nodes represent a suitable starting point as a building block for scaling up to larger deployments.

Number of Ceph Monitor Nodes

A Ceph storage cluster should have at least three (3) monitor nodes on separate hardware for added resiliency. These nodes do not require high-performance CPUs. They do benefit from having SSDs to store the monitor map data.

Two OSDs per 9300 MAX SSD greatly reduce tail latency for 4KB random writes. Two per drive also improve 4KB random write IOPS at higher queue depths.

Replication Factor

NVMe SSDs have high reliability, with high MTTF and low bit error rate. Micron recommends using a minimum replication factor of two in production when deploying OSDs on NVMe versus a replication factor of three, which is common with legacy HDD-based storage.

CPU Sizing

Ceph OSD processes can consume large amounts of CPU while doing small block operations. Consequently, a higher CPU core count results in higher performance for I/O-intensive workloads. For throughput-intensive workloads characterized by large sequential I/O, Ceph performance is more likely to be bound by the maximum network bandwidth of the cluster and CPU sizing is less impactful.

Ceph Configuration Tuning

Tuning Ceph for NVMe devices can be complex. The `ceph.conf` settings used in this reference architecture optimize the solution for small block random performance (see Appendix A).

Networking

A 25 GbE network allows the solution to leverage the maximum block performance benefits of a NVMe-based Ceph cluster. For throughput-intensive workloads, Micron recommends 50 GbE or faster throughput connections.

Number of OSDs per Drive

Testing with one OSD per drive yields good performance that is roughly equivalent to the performance seen when using two OSDs per drive from an IOPS and average latency standpoint. However, running two OSDs per NVMe SSD provides **greatly** reduced tail (99.99%) latency for 4KB random writes as seen in Figure 3.

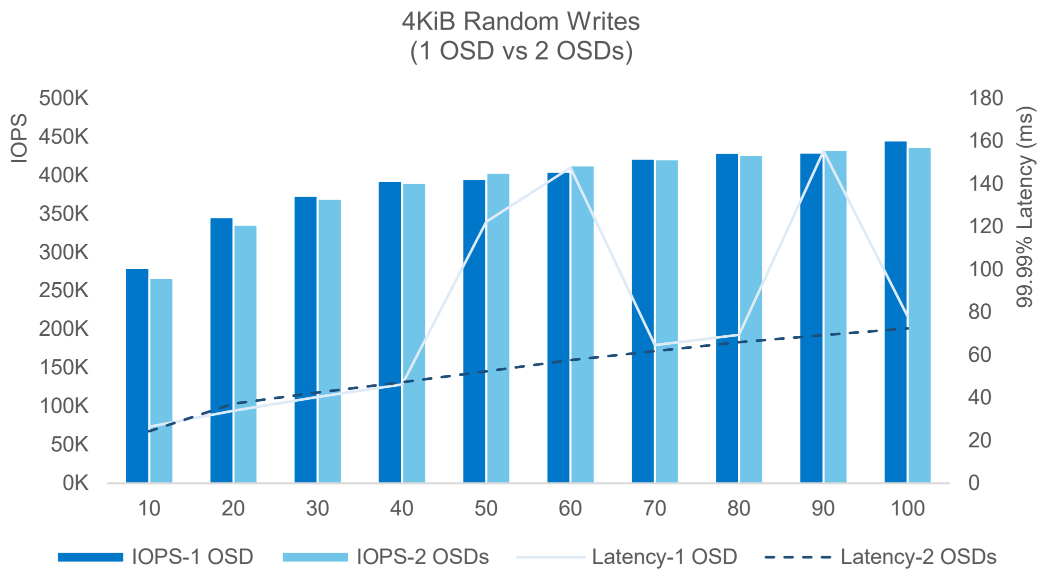

Figure 3 – Write Performance Comparison of 1 OSD vs. 2 OSDs per SSD

Figure 4 shows that using two OSDs per drive also increased 4KB random read IOPS at higher queue depths. Also, tail latency is consistently lower through queue depths up to 32.

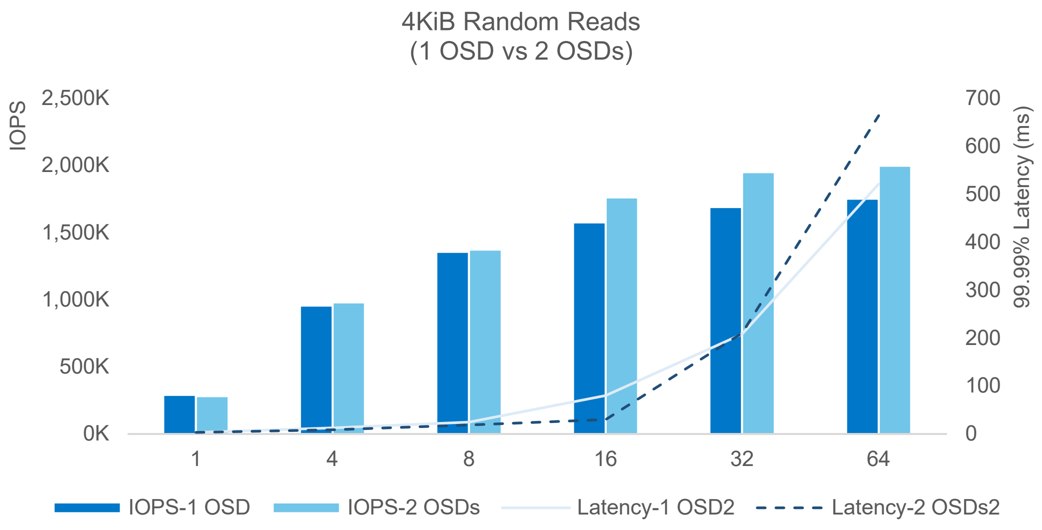

Figure 4 – Read Performance Comparison of 1 OSD vs. 2 OSDs per SSD

Based on this analysis, this reference architecture recommends and uses two OSDs per NVMe device. Appendix A contains the method used for configuring two OSDs per drive.

OS Tuning / NUMA

This RA used Ceph-Ansible for OS tuning and applied the following OS settings:

```
disable_transparent_hugepage: true
kernel.pid_max, value: 4,194,303
fs.file-max, value: 26,234,859
vm.zone_reclaim_mode, value: 0
vm.swappiness, value: 1
vm.min_free_kbytes, value: 1,000,000
net.core.rmem_max, value: 268,435,456
net.core.wmem_max, value: 268,435,456
net.ipv4.tcp_rmem, value: 4096 87,380 134,217,728
net.ipv4.tcp_wmem, value: 4096 65,536 134,217,728
ceph_tcmalloc_max_total_thread_cache: 134,217,728
```

Due to the unbalanced nature of the servers concerning PCIe lane assignments (four NVMe devices and both NICs attach to CPU 1 while the other six NVMe devices attach CPU 2), this RA did not use any NUMA tuning during testing.

`Irqbalance` was active for all tests and did a reasonable job balancing across CPUs.

Measuring Performance

4KB Random Workloads

Small block testing used the FIO synthetic IO generation tool and the Ceph RADOS Block Device (RBD) driver to generate 4KB random I/O workloads.

The test configuration consisted of initially creating 100 RBD images resulting in each RBD image size of 75GB and a total of 7.5TB of data. Implementing a 2x replicated pool resulted in 15TB of total data stored within the cluster.

The four data nodes have a combined total of 1.5TB of DRAM, which is 6.7% of the dataset size.

Random write tests scaled the number of FIO clients running against the Ceph cluster at a fixed queue depth of 32. A client is a single instance of FIO running on a load generation server. Using a queue depth of 32 simulates an active RDB image consumer and allows tests to scale up to a high client count. The number of clients scale from 10 clients to 100 clients. The test used 10 load generation servers with an equal number of FIO instances on each load generation server.

Random reads and 70/30 read/write tests all 100 FIO clients and their associated RBD images, scaling the queue depth per FIO client from 1 to 32. It is important to use all 100 FIO clients for these tests to ensure that tests accessed the entire 15TB dataset; otherwise, Linux filesystem caching can skew results resulting in a false report of higher performance.

Three test iterations executed for 10 minutes with a 2-minute ramp up time for a total of 12-minute test runs. Before each iteration, the test script clears all Linux filesystem caches. The results reported are the averages across all test runs.

4MB Object Workloads

Object testing utilizes the RADOS Bench tool provided as part of the Ceph package to measure object I/O performance. This benchmark reports throughput performance in MB/s and represents the best-case object performance. Object I/O uses a RADOS gateway service operating on each application server. The configuration of RADOS gateway is beyond the scope of this RA.

To measure object write throughput, each test executed RADOS Bench with a “threads” value of 16 on a load generation server writing directly to a Ceph storage pool using 4MB objects. RADOS Bench executed on a varying number of load generation servers scaled between 2 to 10.

To measure object read throughput, each test generates 15TB of data into a 2x replicated pool using 20 RADOS Bench instances. Once the test data load is complete, all 20 RADOS Bench instances execute 4MB object reads against the storage pool while scaling RADOS Bench thread count between four threads and 32 threads.

Three test iterations executed for 10 minutes. Before each iteration, the test script clears all Linux filesystem caches. The results reported are the averages across all test runs.

Ten Micron 9300 MAX SSDs deliver 7 million 4KB random read IOPS and 2.9 million 4KB random write IOPS in baseline FIO testing on a single storage server.

Baseline Performance Test Methodology

Storage and network performance are baseline tested without Ceph software to determine the theoretical hardware performance maximums using FIO and iPerf benchmark tools. Each storage test executes one locally run FIO instance per NVMe drive (10 total NVMe drives) simultaneously. Each network test executes four concurrent `iperf3` instances from each data node and monitor node to each other and from each client to each data server. The results represent the expected maximum performance possible using the specific server and network components in the test environment.

Storage Baseline Results

The baseline block storage test executed FIO across all 10 9300 MAX NVMe SSDs on each node. Eight FIO instances executed 4KB random writes at a queue depth of 10 per FIO instance. Eight FIO instances executed 4KB random reads at a queue depth of 10 per FIO instance. Table 7 provides the average IOPS and latency for all storage baseline testing.

Storage Node	4KB Workloads: FIO on 10x 9300 MAX NVMe SSDs			
	Write IOPS	Write Avg. Latency	Read IOPS	Read Avg. Latency
Node 1	6,474,768	0.12ms	5,956,554	0.13ms
Node 2	6,474,373	0.11ms	5,938,871	0.13ms
Node 3	6,474,321	0.12ms	5,952,282	0.13ms
Node 4	6,474,683	0.12ms	5,932,787	0.13ms

Table 7: Baseline FIO 4KB Random Workloads

The baseline object storage test executed FIO across all 10 9300 MAX NVMe SSDs on each node. Eight FIO instances executed 4MB sequential writes at a queue depth of eight per FIO instance. Eight FIO instances executed 4MB sequential reads at a queue depth of eight per FIO instance. Table 8 provides the average throughput and latency results.

Storage Node	4MB Workloads: FIO on 10x 9300 MAX NVMe SSDs			
	Write Throughput	Write Avg. Latency	Read Throughput	Read Avg. Latency
Node 1	30.46 GiB/s	0.31ms	30.69 GiB/s	0.31ms
Node 2	30.42 GiB/s	0.31ms	30.69 GiB/s	0.31ms
Node 3	30.45 GiB/s	0.31ms	30.69 GiB/s	0.31ms
Node 4	30.43 GiB/s	0.31ms	30.69 GiB/s	0.31ms

Table 8: Baseline FIO 4MB Workloads

Network Baseline Results

Each server's network connectivity was tested using two concurrent `iPerf3` running for one minute. Each iPerf3 instance on each server transmitted data to all other servers.

All storage nodes with 100 GbE NICs averaged 96+ Gb/s during testing. Monitor nodes and clients with 50 GbE NICs averaged 45+ Gb/s during testing.

Test Results and Analysis

The results detailed below are based on a 2x replicated storage pool using version 3.2 Red Hat Ceph Storage with 8192 placement groups.

Small Block Random Workload Testing

The following sections describe the resulting performance measured for random 4KiB (4.0x2^{10} byte) block read, write, and mixed read/write I/O tests.

4KiB 100% Random Write Workloads

Write performance reached a maximum of 477K 4KiB IOPS. Average latency showed a linear increase as the number of clients increased, reaching a maximum average latency of of 6.71ms at 100 clients. Tail (99.99%) latency increased linearly up to 70 clients, then spiked upward, going from 59.38ms at 70 clients to 98.45ms at 80 clients – an increase of over 65% (Figure 5).

Ceph data nodes depend heavily on CPU for performance. Low client load shows CPU utilization starting at 59% with 10 clients and increasing steadily to over 86% at a load of 70 clients. Above 70 clients, average CPU utilization is flat and below 90% at 100 clients (Figure 6).

Figures 5-6 and Table 9 summarize the solution's write performance. Based on the observed behavior, write-centric small-block workloads should focus on sizing for no more than 85% CPU utilization. The actual number of clients attained before reaching this level of utilization will depend on the CPU model chosen. This RA's choice of a Xeon Platinum 8168 indicates that the target sizing should be 70 clients.

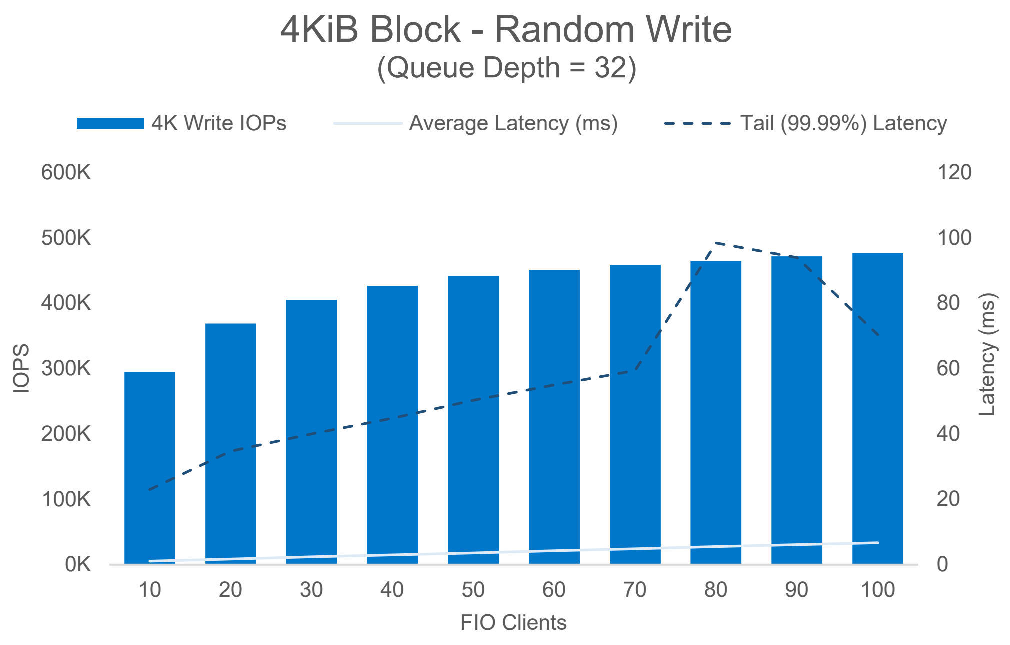

Figure 5 – 4KB Random Write Performance vs. Average and Tail Latency

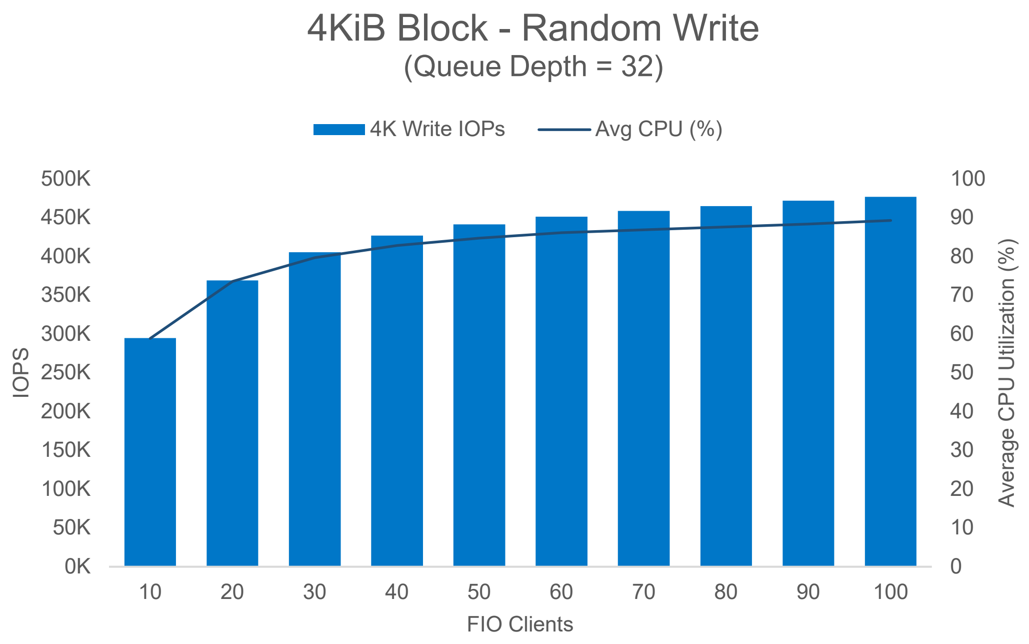

Figure 6 – 4KB Random Write IOPS vs. CPU Utilization

	4KiB Random Write Performance: FIO RBD @ Queue Depth 32				
FIO Clients	IOPS	Average Latency	95% Latency	99.99% Latency	Avg. CPU Utilization
10 Clients	294,714	1.08ms	1.48ms	22.97ms	58.8%
20 Clients	369,092	1.73ms	2.60ms	34.75ms	73.6%
30 Clients	405,353	2.36ms	4.09ms	40.03ms	79.6%
40 Clients	426,876	3.00ms	6.15ms	44.84ms	82.8%
50 Clients	441,391	3.62ms	8.40ms	50.31ms	84.7%
60 Clients	451,308	4.25ms	10.61ms	55.00ms	86.1%
70 Clients	458,809	4.88ms	12.63ms	59.38ms	86.8%
80 Clients	464,905	5.51ms	14.46ms	98.45ms	87.6%
90 Clients	471,696	6.11ms	16.21ms	93.93ms	88.3%
100 Clients	477,029	6.71ms	17.98ms	70.40ms	89.3%

Table 9: 4KB Random Write Summary

4KB Random Read Workload Analysis

Read performance of 100 FIO clients reached a maximum of 2.1M 4KiB IOPS. Average latency showed an increase as the queue depth increased, reaching a maximum average latency of only 0.33ms at queue depth 32. Tail (99.99%) latency increased steadily up to a queue depth of 16, then spiked upward at queue depth of 32, going from 35.4ms at queue depth of 16 to 240ms at queue depth of 32 — an increase of over 580% (Figure 7).

Low queue depth shows CPU utilization starting at 14% at queue depth of 1 and increasing steadily to over 89% at a queue depth of 16. Above queue depth 16, average CPU utilization continues to grow at a slower pace to a value of just over 93% at queue depth of 32 (Figure 8).

Figures 7-8 and Table 10 summarize the solution’s read performance. Based on the observed behavior, read-centric small-block workloads should focus on sizing for no more than 90% CPU utilization. The actual queue depth and client load attained before reaching this level of utilization will depend on the CPU model chosen. This RA’s choice of a Xeon Platinum 8168 indicates that the target sizing should be a queue depth of 16.

Tail latency spikes above queue depth 16 / 2.1 million IOPS due to CPU saturation.

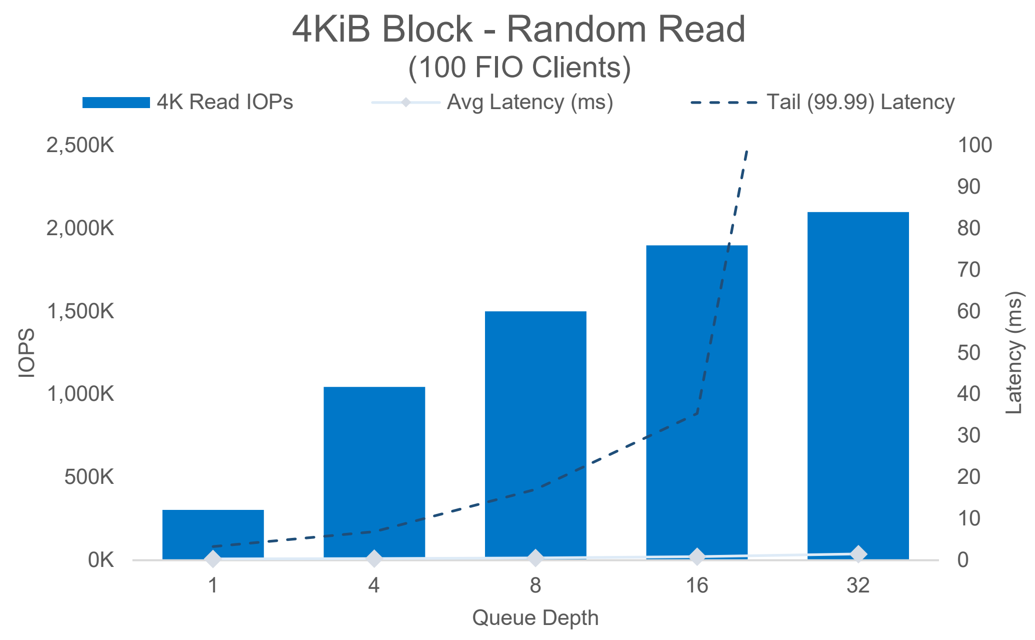

Figure 7 – 4KiB Random Read IOPS vs. Average and Tail (99.99%) Latency

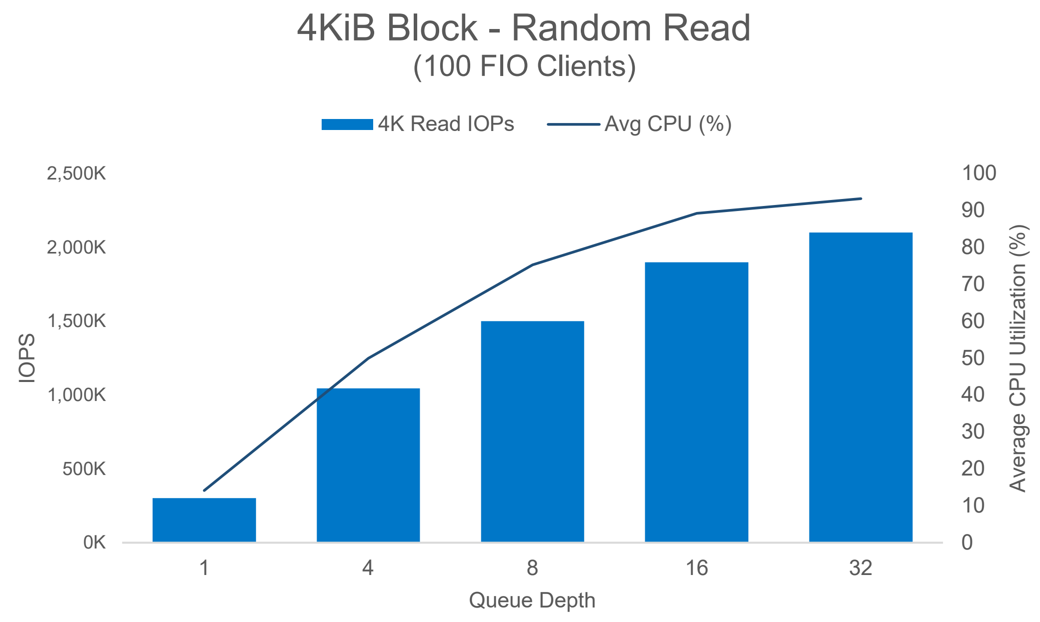

Figure 8 – 4KiB Random Read IOPS vs. CPU Utilization

	4KB Random Read Performance: 100 FIO RBD Clients at Varied QD				
Queue Depth	IOPS	Average Latency	95% Latency	99.99% Latency	Avg CPU Utilization
QD 1	302,598	0.33ms	2.00ms	3.30ms	14.1%
QD 4	1,044,447	0.38ms	2.52ms	6.92ms	49.9%
QD 8	1,499,703	0.53ms	3.96ms	17.10ms	75.3%
QD 16	1,898,738	0.84ms	8.87ms	35.41ms	89.3%
QD 32	2,099,444	1.52ms	20.80ms	240.86ms	93.2%

Table 11: 4KB Random Read Results

4KB Random 70% Read/30% Write Workload Analysis

Mixed read and write (70% read/30% write) performance of 100 FIO clients reached a maximum of almost 904K 4KiB IOPS. Both read and write average latency showed an increase as the queue depth increased, reaching a maximum average read latency of 2.11ms and a maximum average write latency of 6.85ms at queue depth 32. Tail (99.99%) latency for both read and write increased steadily up to a queue depth of 16, then spiked upward at queue depth of 32, with read latency going from 36.6ms at queue depth of 16 to 257ms at queue depth of 32 — an increase of over 602% — and with write latency increasing from 72.6ms at queue depth 16 to 262ms at queue depth of 32 — an increase of 260% (Figure 9).

Increasing load from queue depth 16 to 32 increases IOPS by 12% but triples average latency and increases tail latency by 260%.

Low queue depth shows CPU utilization starting at 19.4% at queue depth of 1 and increasing steadily to over 89% at a queue depth of 16. Above queue depth 16, average CPU utilization continues to grow at a slower pace to a value of just under 93% at queue depth 32 (Figure 10).

Figures 9-10 and Table 11 summarize the solution's 70%/30% read/write performance. Based on the observed behavior, mixed I/O small-block workloads should focus on sizing for no more than 90% CPU utilization. The actual queue depth and client load attained before reaching this level of utilization will depend on the CPU model chosen and the actual read/write mix. This RA's choice of a Xeon Platinum 8168 and 70%/30% read/write mix indicates that the target sizing should be a queue depth of 16.

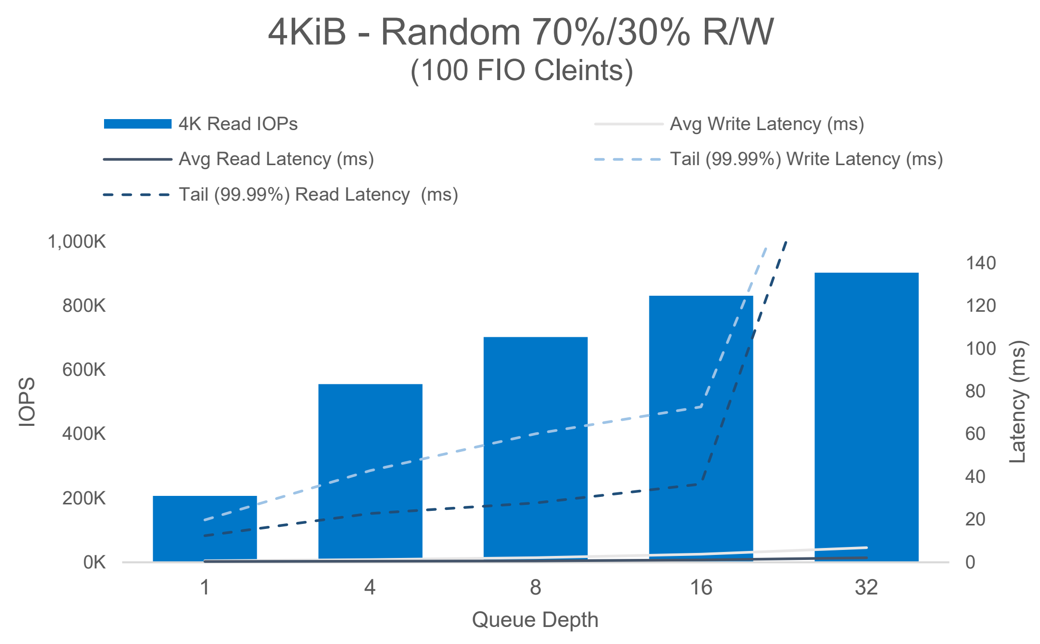

Figure 9 – 4KB Random 70%/30% R/W IOPS vs. Average and Tail (99.99%) Latency

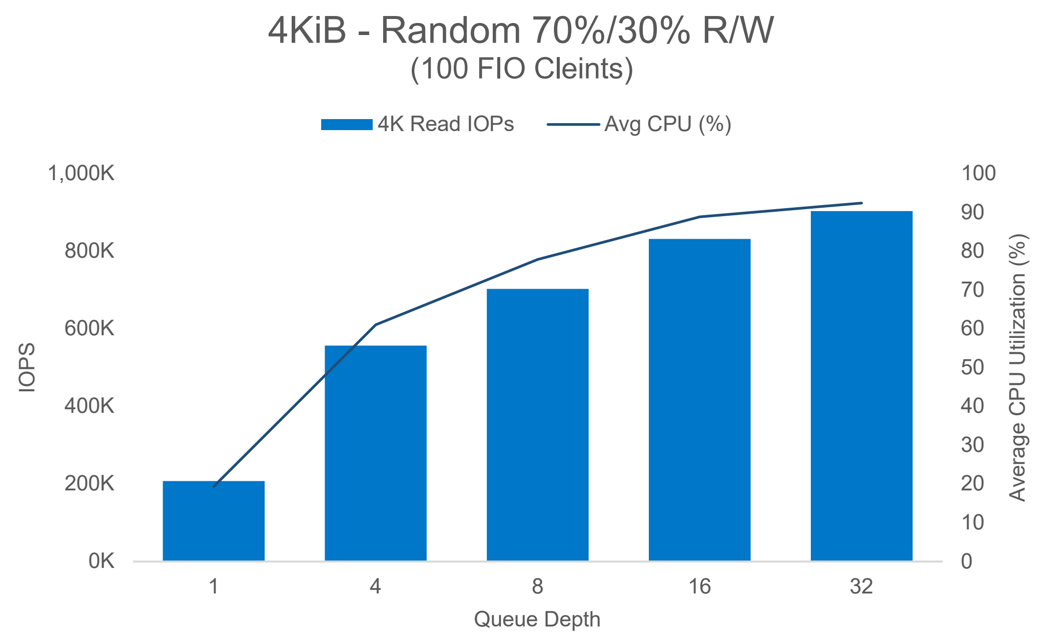

Figure 10 – 4KB Random 70%/30% R/W IOPS vs. CPU Utilization

	4KB Random 70/30 R/W Performance: 100 FIO RBD Clients at Varied QD					
Queue Depth	IOPS	Avg. Read Latency	Avg. Write Latency	99.99% Read Latency	99.99% Write Latency	Avg. CPU Utilization
QD 1	207,703	0.72ms	0.37ms	19.83ms	12.45ms	19.38%
QD 4	556,369	1.23ms	0.49ms	42.86ms	22.91ms	61.00%
QD 8	702,907	2.12ms	0.71ms	60.13ms	27.77ms	77.92%
QD 16	831,611	3.77ms	1.13ms	72.67ms	36.56ms	88.86%
QD 32	903,866	6.85ms	2.11ms	261.93ms	257.25ms	92.42%

Table 12: 4KB 70%/30% Random Read/Write Results

4MB Object Workloads

The following sections describe the resulting performance measured for random, 4MiB (4.0x2^{20} byte) object read and write data in both sequential (read and write) and random (reads only) I/O scenarios.

Object Write Workload Analysis

Object write performance reached a maximum throughput of 24 GB/s with an average latency of 26.9ms at a workload level of ten clients. Latency growth was consistent as workload increased indicating that there were no apparent cases of resource constraints (Figure 13).

CPU utilization was extremely low for this test. Previous RAs have experienced CPU utilization of near 50% using similar hardware configurations. Average CPU utilization for this RA never reached higher than 22% (Figure 14).

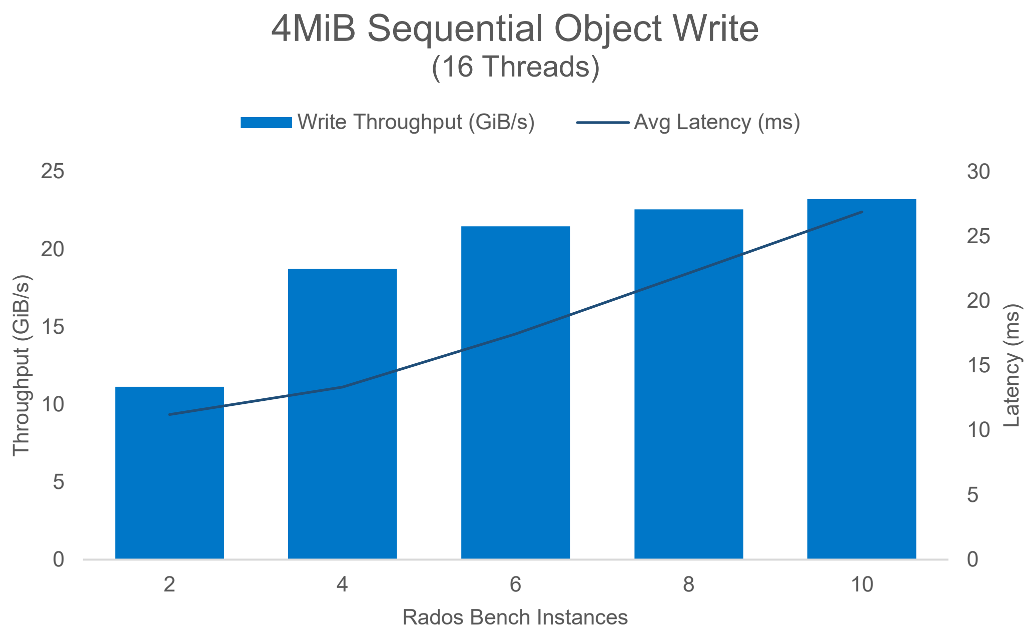

Figure 13 – 4MB Object Write Throughput vs. Average Latency

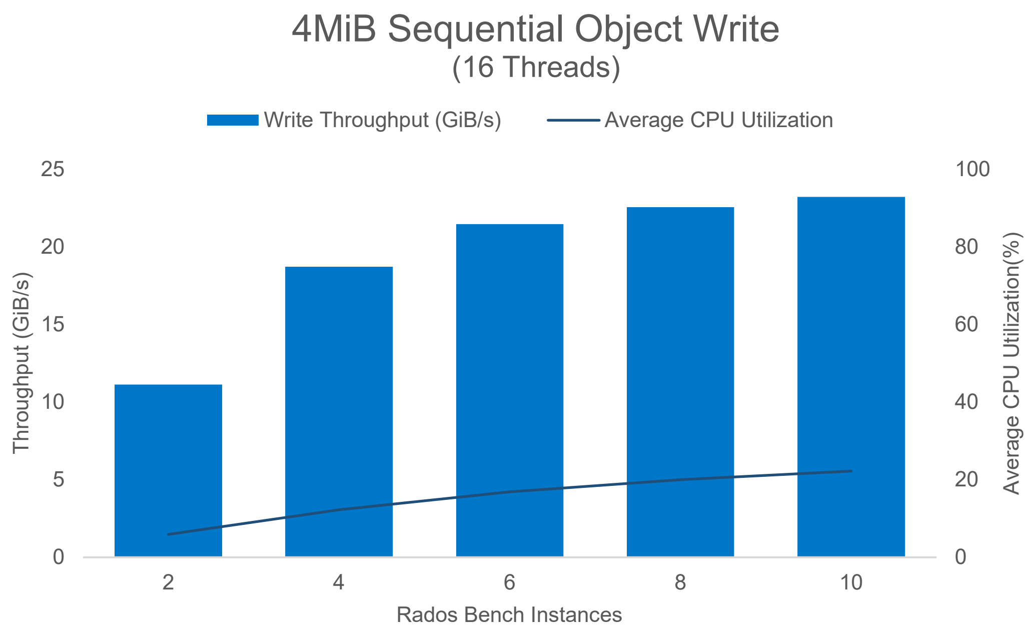

Figure 14 – 4MB Object Write Throughput vs. Average CPU Utilization

4MB Object Write Performance: RADOS Bench			
Clients	Write Throughput	Average Latency	Average CPU Utilization
2 Clients	11.14 GiB/s	11.22ms	5.90%
4 Clients	18.74 GiB/s	13.34ms	12.25%
6 Clients	21.48 GiB/s	17.46ms	16.91%
8 Clients	22.57 GiB/s	22.15ms	20.04%
10 Clients	23.24 GiB/s	26.90ms	22.23%

Table 13: 4MB Object Write Results

Object Read Workload Analysis

Object read performance reached a maximum sequential throughput of 48.4 GB/s with an average latency of 26.5ms attained at 16 threads; while maximum random throughput of 43.7 GB/s was achieved with an average latency of 52.16ms at 32 threads. Sequential read performance was slightly lower at 16 threads (4% lower), but average latency was one-half the 52ms measured at 32 threads. Latency growth was consistent as workload increased indicating that there were no apparent cases of resource constraints (Figure 14 and Table 14).

CPU utilization was extremely low for this test. Average CPU utilization for this RA never reached higher than 13% (Figure 15 and Table 14).

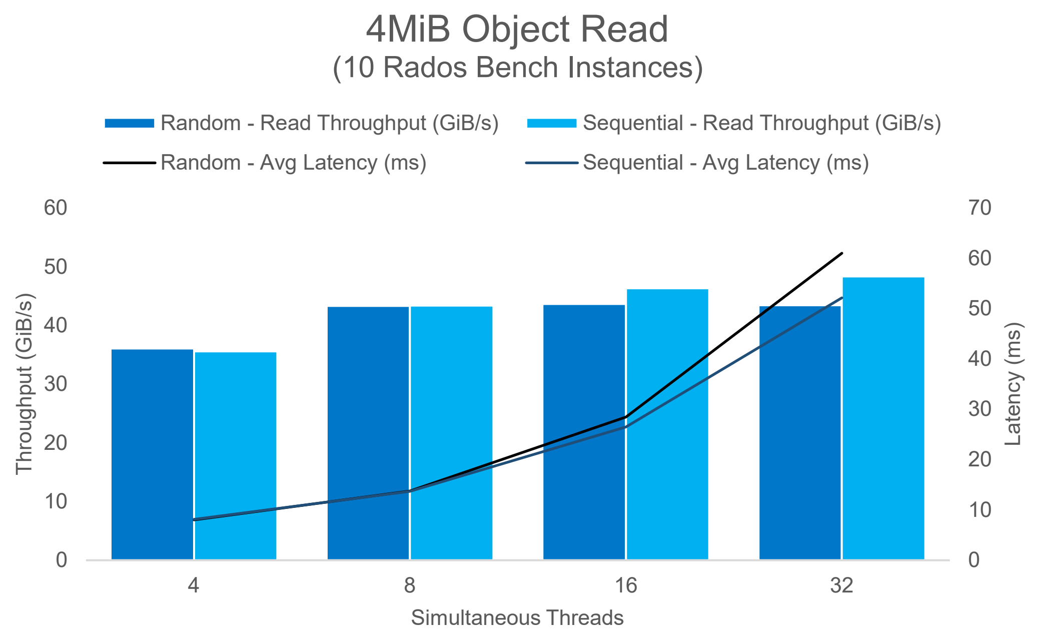

Figure 14 – 4MB Object Read Throughput vs. Average Latency

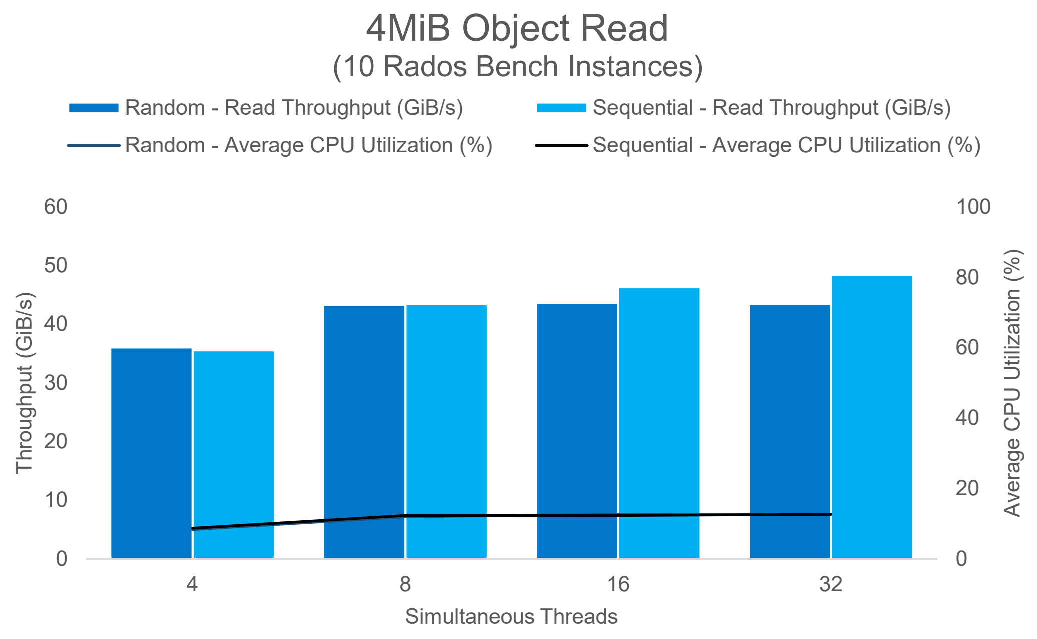

Figure 15 – 4MB Object Read Throughput vs. Average CPU Utilization

Simultaneous Threads	4MB Object Read Performance: RADOS Bench			
	Random Read Throughput	Average CPU Utilization	Sequential Read Throughput	Average CPU Utilization
4	36.08 GiB/s	8.38%	35.57 GiB/s	8.75%
8	43.32 GiB/s	11.99%	43.40 GiB/s	12.34%
16	43.65 GiB/s	12.75%	46.32 GiB/s	12.35%
32	43.45 GiB/s	12.77%	48.36 GiB/s	12.68%

Table 14: 4MB Object Read Results

Summary

Micron designed this reference architecture for small block random workloads. With over 2 million 4KiB random reads and 477,000 4KiB random writes in a compact design based on four 1RU data nodes and three 1RU monitor nodes and 256TB of total storage, this is the most performant Ceph architecture we've tested to date.

Micron's enterprise NVMe SSDs provide massive performance and provide a suitable solution for many different types of storage solutions such as Red Hat Ceph Storage software-defined storage. Whether your needs are general purpose use cases or demand ultra-fast responses for transactional workloads or large, fast data analytics solutions, Micron has taken the guesswork out of building the right solution for your needs.

Appendix A
Ceph.conf

```
[client]
rbd_cache = False
rbd_cache_writethrough_until_flush = False

# Please do not change this file directly
since it is managed by Ansible and will be
overwritten
[global]
auth client required = none
auth cluster required = none
auth service required = none
auth supported = none
cephx require signatures = False
cephx sign messages = False
cluster network = 192.168.1.0/24
debug asok = 0/0
debug auth = 0/0
debug bluefs = 0/0
debug bluestore = 0/0
debug buffer = 0/0
debug client = 0/0
debug context = 0/0
debug crush = 0/0
debug filer = 0/0
debug filestore = 0/0
debug finisher = 0/0
debug hadoop = 0/0
debug heartbeatmap = 0/0
debug journal = 0/0
debug journaler = 0/0
debug lockdep = 0/0
debug log = 0
debug mds = 0/0
debug mds_balancer = 0/0
debug mds_locker = 0/0
debug mds_log = 0/0
debug mds_log_expire = 0/0
debug mds_migrator = 0/0
debug mon = 0/0
debug monc = 0/0
debug ms = 0/0
debug objclass = 0/0
debug objectcacher = 0/0
debug objecter = 0/0
debug optracker = 0/0
debug osd = 0/0
debug paxos = 0/0
debug perfcounter = 0/0
debug rados = 0/0
debug rbd = 0/0
debug rgw = 0/0
debug rocksdb = 0/0
debug throttle = 0/0
debug timer = 0/0
debug tp = 0/0
debug zs = 0/0
fsid = 36a9e9ee-a7b8-4c41-a3e5-0b575f289379
mon host =
192.168.0.200,192.168.0.201,192.168.0.202
mon pg warn max per osd = 800
mon_allow_pool_delete = True
mon_max_pg_per_osd = 800
ms type = async
ms_crc_data = False
ms_crc_header = True
osd objectstore = bluestore
osd_pool_default_size = 2
perf = True
public network = 192.168.0.0/24
rocksdb_perf = True

[mon]
mon_max_pool_pg_num = 166496
mon_osd_max_split_count = 10000

[osd]
bluestore_csum_type = none
bluestore_extent_map_shard_max_size = 200
bluestore_extent_map_shard_min_size = 50
bluestore_extent_map_shard_target_size = 100
bluestore_rocksdb_options =
compression=kNoCompression,max_write_buffer_
number=64,min_write_buffer_number_to_merge=3
2,recycle_log_file_num=64,compaction_style=k
CompactionStyleLevel,write_buffer_size=4MB,t
arget_file_size_base=4MB,max_background_comp
actions=64,level0_file_num_compaction_trigge
r=64,level0_slowdown_writes_trigger=128,leve
l0_stop_writes_trigger=256,max_bytes_for_lev
el_base=6GB,compaction_threads=32,flusher_th
reads=8,compaction_readahead_size=2MB
osd memory target = 14150664191
osd_max_pg_log_entries = 10
osd_min_pg_log_entries = 10
osd_pg_log_dups_tracked = 10
osd_pg_log_trim_min = 10
```

Ceph-Ansible Configuration

All.yml

```
---
dummy:
mon_group_name: mons
osd_group_name: osds
rgw_group_name: rgws
mds_group_name: mdss
nfs_group_name: nfss
restapi_group_name: restapis
rbdmirror_group_name: rbdmirrors
client_group_name: clients
iscsi_gw_group_name: iscsigws
mgr_group_name: mgrs
ntp_service_enabled: false
ceph_origin: repository
ceph_repository: rhcs
ceph_rhcs_version: "{{
ceph_stable_rh_storage_version | default(3)
}}"
ceph_repository_type: cdn
fsid: 36a9e9ee-a7b8-4c41-a3e5-0b575f289379
generate_fsid: false
cephx: false
rbd_cache: "false"
rbd_cache_writethrough_until_flush: "false"
monitor_interface: enp131s0.501
ip_version: ipv4
public_network: 192.168.0.0/24
cluster_network: 192.168.1.0/24
osd_mkfs_type: xfs
osd_mkfs_options_xfs: -f -i size=2048
osd_mount_options_xfs:
noatime,largeio,inode64,swalloc
osd_objectstore: bluestore
ceph_conf_overrides:
 global:
  auth client required: none
  auth cluster required: none
  auth service required: none
  auth supported: none
  osd objectstore: bluestore
  cephx require signatures: False
  cephx sign messages: False
  mon_allow_pool_delete: True
  mon_max_pg_per_osd: 800
  mon pg warn max per osd: 800
  ms_crc_header: True
  ms_crc_data: False
  ms type: async
  perf: True
  rocksdb_perf: True
  osd_pool_default_size: 2
  debug asok: 0/0
  debug auth: 0/0
  debug bluefs: 0/0
  debug bluestore: 0/0
  debug buffer: 0/0
  debug client: 0/0
  debug context: 0/0
  debug crush: 0/0
  debug filer: 0/0
  debug filestore: 0/0
  debug finisher: 0/0
  debug hadoop: 0/0
  debug heartbeatmap: 0/0
  debug journal: 0/0
  debug journaler: 0/0
  debug lockdep: 0/0
  debug log: 0
  debug mds: 0/0
  debug mds_balancer: 0/0
  debug mds_locker: 0/0
  debug mds_log: 0/0
  debug mds_log_expire: 0/0
  debug mds_migrator: 0/0
  debug mon: 0/0
  debug monc: 0/0
  debug ms: 0/0
  debug objclass: 0/0
  debug objectcacher: 0/0
  debug objecter: 0/0
  debug optracker: 0/0
  debug osd: 0/0
  debug paxos: 0/0
  debug perfcounter: 0/0
  debug rados: 0/0
  debug rbd: 0/0
  debug rgw: 0/0
  debug rocksdb: 0/0
  debug throttle: 0/0
  debug timer: 0/0
  debug tp: 0/0
  debug zs: 0/0
 mon:
  mon_max_pool_pg_num: 166496
  mon_osd_max_split_count: 10000
 client:
  rbd_cache: false
  rbd_cache_writethrough_until_flush: false
 osd:
  osd_min_pg_log_entries: 10
  osd_max_pg_log_entries: 10
  osd_pg_log_dups_tracked: 10
  osd_pg_log_trim_min: 10
  bluestore_csum_type: none
  bluestore_extent_map_shard_min_size: 50
  bluestore_extent_map_shard_max_size: 200
  bluestore_extent_map_shard_target_size:
100
  bluestore_rocksdb_options:
compression=kNoCompression,max_write_buffer_
number=64,min_write_buffer_number_to_merge=3
2,recycle_log_file_num=64,compaction_style=k
CompactionStyleLevel,write_buffer_size=4MB,t
arget_file_size_base=4MB,max_background_comp
actions=64,level0_file_num_compaction_trigge
r=64,level0_slowdown_writes_trigger=128,leve
l0_stop_writes_trigger=256,max_bytes_for_lev
el_base=6GB,compaction_threads=32,flusher_th
reads=8,compaction_readahead_size=2MB
```

```
disable_transparent_hugepage: true
os_tuning_params:
  - { name: kernel.pid_max, value: 4194303 }
  - { name: fs.file-max, value: 26234859 }
  - { name: vm.zone_reclaim_mode, value: 0 }
  - { name: vm.swappiness, value: 1 }
  - { name: vm.min_free_kbytes, value:
1000000 }
  - { name: net.core.rmem_max, value:
268435456 }
  - { name: net.core.wmem_max, value:
268435456 }
  - { name: net.ipv4.tcp_rmem, value: 4096
87380 134217728 }
  - { name: net.ipv4.tcp_wmem, value: 4096
65536 134217728 }
ceph_tcmalloc_max_total_thread_cache:
134217728
containerized_deployment: False
```

Osds.yml

```
---
dummy:
osd_scenario: lvm

lvm_volumes:
   - data: data-lv1
     data_vg: vg_nvme0n1
     db: db-lv1
     db_vg: vg_nvme0n2
   - data: data-lv1
     data_vg: vg_nvme0n3
     db: db-lv1
     db_vg: vg_nvme0n4
   - data: data-lv1
     data_vg: vg_nvme1n1
     db: db-lv1
     db_vg: vg_nvme1n2
   - data: data-lv1
     data_vg: vg_nvme1n3
     db: db-lv1
     db_vg: vg_nvme1n4
   - data: data-lv1
     data_vg: vg_nvme2n1
     db: db-lv1
     db_vg: vg_nvme2n2
   - data: data-lv1
     data_vg: vg_nvme2n3
     db: db-lv1
     db_vg: vg_nvme2n4
   - data: data-lv1
     data_vg: vg_nvme3n1
     db: db-lv1
     db_vg: vg_nvme3n2
   - data: data-lv1
     data_vg: vg_nvme3n3
     db: db-lv1
     db_vg: vg_nvme3n4
   - data: data-lv1
     data_vg: vg_nvme4n1
     db: db-lv1
     db_vg: vg_nvme4n2
   - data: data-lv1
     data_vg: vg_nvme4n3
     db: db-lv1
     db_vg: vg_nvme4n4
   - data: data-lv1
     data_vg: vg_nvme5n1
     db: db-lv1
     db_vg: vg_nvme5n2
   - data: data-lv1
     data_vg: vg_nvme5n3
     db: db-lv1
     db_vg: vg_nvme5n4
   - data: data-lv1
     data_vg: vg_nvme6n1
     db: db-lv1
     db_vg: vg_nvme6n2
   - data: data-lv1
     data_vg: vg_nvme6n3
     db: db-lv1
     db_vg: vg_nvme6n4
   - data: data-lv1
     data_vg: vg_nvme7n1
     db: db-lv1
     db_vg: vg_nvme7n2
   - data: data-lv1
     data_vg: vg_nvme7n3
     db: db-lv1
     db_vg: vg_nvme7n4
   - data: data-lv1
     data_vg: vg_nvme8n1
     db: db-lv1
     db_vg: vg_nvme8n2
   - data: data-lv1
     data_vg: vg_nvme8n3
     db: db-lv1
     db_vg: vg_nvme8n4
   - data: data-lv1
     data_vg: vg_nvme9n1
     db: db-lv1
     db_vg: vg_nvme9n2
   - data: data-lv1
     data_vg: vg_nvme9n3
     db: db-lv1
     db_vg: vg_nvme9n4
```

```
---
dummy:
osd_scenario: collocated
```

Creating Multiple Namespaces

The Ceph recommendation for the volume storing the OSD database is no less than 4% of the size of the OSD data volume, which is the number that we used.

```
python create_namespaces.py -ns 12002338736 500097448 12002338736 500097448 -ls 512 -d
/dev/nvme0 /dev/nvme1 /dev/nvme2 /dev/nvme3 /dev/nvme4 /dev/nvme5 /dev/nvme6
/dev/nvme7 /dev/nvme8 /dev/nvme9
```

create_namespaces.py

```
import argparse
import time
from subprocess import Popen, PIPE

def parse_arguments():
    parser = argparse.ArgumentParser(description='This file creates namespaces across
NVMe devices')
    parser.add_argument('-ns', '--namespace-size', nargs='+', required=True, type=str,
                        help='List of size of each namespace in number of LBAs.
Specifying more than one will create '
                             'multiple namespaces on each device.')
    parser.add_argument('-ls', '--lba-size', default='512', choices=['512', '4096'],
required=False, type=str,
                        help='Size of LBA in bytes. Valid options are 512 and 4096
(Default: 512)')
    parser.add_argument('-d', '--devices', nargs='+', required=True, type=str,
                        help='List of data devices to create OSDs on.')


    return {k: v for k, v in vars(parser.parse_args()).items() }

def execute_command(cmd):
    process = Popen(cmd, stdout=PIPE, stderr=PIPE)
    stdout, stderr = process.communicate()

    if stderr not in ('', None):
        print stdout
        raise Exception(stderr)
    else:
        return stdout


def remove_namespaces(devices, **_):
    for dev in devices:
        cmd = ['nvme', 'list-ns', dev]
        namespaces = [int(value[value.find('[') + 1:-value.find(']')].strip())
                      for value in execute_command(cmd=cmd).strip().split('\n')]
        for namespace in namespaces:
            cmd = ['nvme', 'detach-ns', dev, '-n', str(namespace+1), '-c', '1']
            print execute_command(cmd=cmd)

            cmd = ['nvme', 'delete-ns', dev, '-n', str(namespace+1)]
            print execute_command(cmd=cmd)
            time.sleep(0.1)


def create_namespaces(devices, namespace_size, lba_size):
    lba_key = '2' if lba_size == '4096' else '0'
    for dev in devices:
        for size in namespace_size:
            cmd = ['nvme', 'create-ns', dev, '-f', lba_key, '-s', size, '-c', size]
            print execute_command(cmd=cmd)

    attach_namespaces(devices=devices, num_namespaces=len(namespace_size))


def attach_namespaces(devices, num_namespaces):
    for namespace in range(1,num_namespaces+1):
        for dev in devices:
            cmd = ['nvme', 'attach-ns', dev, '-n', str(namespace), '-c', '1']
            print execute_command(cmd=cmd)


def run_test():
    arguments = parse_arguments()

    remove_namespaces(**arguments)
    create_namespaces(**arguments)


if __name__=='__main__':
    run_test()
```

Partitioning Drives for OSDs

python create_ceph_partitions.py -h for help message

```
python create_ceph_partitions.py --osds-per-device 1 --data-devices
/dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1 /dev/nvme4n1 /dev/nvme5n1
/dev/nvme6n1 /dev/nvme7n1 /dev/nvme8n1 /dev/nvme9n1 /dev/nvme0n3 /dev/nvme1n3
/dev/nvme2n3 /dev/nvme3n3 /dev/nvme4n3 /dev/nvme5n3 /dev/nvme6n3 /dev/nvme7n3
/dev/nvme8n3 /dev/nvme9n3 –cache-devices /dev/nvme0n2 /dev/nvme1n2
/dev/nvme2n2 /dev/nvme3n2 /dev/nvme4n2 /dev/nvme5n2 /dev/nvme6n2 /dev/nvme7n2
/dev/nvme8n2 /dev/nvme9n2 /dev/nvme0n4 /dev/nvme1n4 /dev/nvme2n4 /dev/nvme3n4
/dev/nvme4n4 /dev/nvme5n4 /dev/nvme6n4 /dev/nvme7n4 /dev/nvme8n4 /dev/nvme9n4
```

The value of 1 was used for OSDs-per-device because we used multiple namespaces on the same physical device. Each namespace only has one OSD.

create_ceph_partitions.py

```
import argparse
import os
from subprocess import Popen, PIPE

def parse_arguments():
    parser = argparse.ArgumentParser(description='This file partitions devices for
ceph storage deployment')
    parser.add_argument('-o', '--osds-per-device', required=True, type=int,
help='Number of OSDs per data device')
    parser.add_argument('-d', '--data-devices', nargs='+', required=True, type=str,
                        help='Space separated list of data devices to create OSDs
on.')
    parser.add_argument('-c', '--cache-devices', nargs='+', required=False, type=str,
                        help='Space separated list of cache devices to store BlueStore
RocksDB and/or write-ahead log')
    parser.add_argument('-ws', '--wal-sz', required=False, type=int,
                        help='Size of each write-ahead log on specified cache devices
in GiB')
    parser.add_argument('-dnr', '--do-not-remove', action='store_true',
                        help='Do Not remove old volumes (Disabled by default)')
    parser.add_argument('-dnc', '--do-not-create', action='store_true',
                        help='Do not create new volumes (Disabled by default)')

    return {k: v for k, v in vars(parser.parse_args()).items() }

def execute_command(cmd):
    process = Popen(cmd, stdout=PIPE, stderr=PIPE)
    stdout, stderr = process.communicate()

    if stderr not in ('', None):
        print stdout
        raise Exception(stderr)
    else:
        return stdout


def remove_lvm_volumes():
    dev_path = '/dev'
    cache_prefix = 'vg_nvm'
    data_prefix = 'vg_sd'

    for device in os.listdir(dev_path):
        path = os.path.join(dev_path, device)
        if device.startswith(cache_prefix) or device.startswith(data_prefix):

            # Remove Logical Volumes
            for item in os.listdir(path):
                cmd = ['lvremove','-y',os.path.join(path, item)]
                print execute_command(cmd=cmd)

            # Remove Volume Group
            cmd = ['vgremove', '-y', device]
            print execute_command(cmd=cmd)

            # Remove Physical Volume
            pv_name = device[3:]
            cmd = ['pvremove', '-y', '/dev/{}'.format(pv_name)]
            print execute_command(cmd=cmd)

            # Wipe FS
            cmd = ['wipefs', '-a', '/dev/{}'.format(pv_name)]
            print execute_command(cmd=cmd)

            # Create GPT
            cmd = ['sudo', 'sgdisk', '-Z', '/dev/{}'.format(pv_name)]
            print execute_command(cmd=cmd)


def create_partitions(data_devices, osds_per_device, cache_devices, wal_sz, **_):
    # Create cache partitions
    if cache_devices:
        db_partitions = len(data_devices) * osds_per_device / len(cache_devices)
        create_cache_device_volumes(cache_devices=cache_devices, wal_sz=wal_sz,
db_partitions=db_partitions)

    # Create data partitions
    create_data_device_volumes(data_devices=data_devices,
osds_per_device=osds_per_device)


def create_cache_device_volumes(cache_devices, wal_sz, db_partitions):
    for dev in cache_devices:
        cmd = ['pvcreate', dev]
        print execute_command(cmd=cmd)

        vg_name = 'vg_{}'.format(os.path.basename(dev))
        cmd = ['vgcreate', vg_name, dev]
        print execute_command(cmd=cmd)

        gb_total = get_total_size(vg_name=vg_name)

        # If WAL was given
        if not wal_sz:
            wal_sz = 0

        sz_per_db = (gb_total / db_partitions) - wal_sz

        for i in range(1, db_partitions+1):
            cmd = ['lvcreate', '-y', '--name', 'db-lv{}'.format(i), '--size',
'{}G'.format(sz_per_db), vg_name]
            print execute_command(cmd=cmd)
            if wal_sz:
                cmd = ['lvcreate', '-y', '--name', 'wal-lv{}'.format(i), '--size',
'{}G'.format(wal_sz), vg_name]
                print execute_command(cmd=cmd)


def create_data_device_volumes(data_devices, osds_per_device):
    for dev in data_devices:
        cmd = ['pvcreate', dev]
        print
        execute_command(cmd=cmd)

        vg_name = 'vg_{}'.format(os.path.basename(dev))
        cmd = ['vgcreate', vg_name, dev]
        print
        execute_command(cmd=cmd)

        gb_total = get_total_size(vg_name=vg_name)

        sz_per_osd = gb_total / osds_per_device

        for i in range(1, osds_per_device+1):
            cmd = ['lvcreate', '-y', '--name', 'data-lv{}'.format(i), '--size',
'{}G'.format(sz_per_osd), vg_name]
            print execute_command(cmd=cmd)


def get_total_size(vg_name):
    cmd = ['vgdisplay', vg_name]
    stdout = execute_command(cmd=cmd)
```

```
        for line in stdout.split('\n'):
            if 'Total PE' in line:
                total_pe = int(line.split()[2])
            elif 'PE Size' in line:
                pe_size = int(float(line.split()[2]))

        gb_total = total_pe * pe_size / 1024
        return gb_total

    def run_test():
        arguments = parse_arguments()

        if not arguments['do_not_remove']:
            # Remove All Old LVM Volumes
            remove_lvm_volumes()

        if not arguments['do_not_create']:
            create_partitions(**arguments)


    if __name__ == '__main__':
        run_test()
```

Drive Readahead Settings

For 4KB random tests, the following was run on the storage nodes to set readahead to 0:

```
for i in {0..9};do sudo echo 0 > /sys/block/nvme${i}n1/queue/read_ahead_kb;done
```

For 4MB object tests, the following was run on the storage nodes to set readahead to 128:

```
for i in {0..9};do sudo echo 128 > /sys/block/nvme${i}n1/queue/read_ahead_kb;done
```

Spectre/Meltdown Patch Settings

Due to the potential performance impact and changing release status of Spectre/Meltdown updates, the variant #2 (Spectre) and variant #3 (Meltdown) patches were disabled (see the instructions in this article https://access.redhat.com/articles/3311301).

echo 0 > /sys/kernel/debug/x86/pti_enabled

echo 0 > /sys/kernel/debug/x86/ibrs_enabled

echo 0 > /sys/kernel/debug/x86/retp_enabled

About Micron

Micron Technology (Nasdaq: MU) is a world leader in innovative memory solutions. Through our global brands — Micron, Crucial® and Ballistix® — our broad portfolio of high-performance memory technologies, including DRAM, NAND, NOR Flash and 3D XPoint™memory, is transforming how the world uses information to enrich life. Backed by 40 years of technology leadership, our memory and storage solutions enable disruptive trends, including artificial intelligence, machine learning and autonomous vehicles, in key market segments like data center, networking, automotive, industrial, mobile, graphics and client. Our common stock is traded on the Nasdaq under the MU symbol.

About Red Hat

Red Hat is the world's leading provider of open source software solutions, using a community-powered approach to provide reliable and high-performing cloud, Linux, middleware, storage, and virtualization technologies. Red Hat also offers award-winning support, training, and consulting services. As a connective hub in a global network of enterprises, partners, and open source communities, Red Hat helps create relevant, innovative technologies that liberate resources for growth and prepare customers for the future of IT.

About Ceph Storage

Ceph is an open source distributed object store and file system designed to provide excellent performance, reliability, and scalability. It can:

- Free you from the expensive lock-in of proprietary, hardware-based storage solutions.
- Consolidate labor and storage costs into 1 versatile solution.
- Introduce cost-effective scalability on self-healing clusters based on standard servers and disks

micron.com