Lustre with NetApp E-Series storage

NetApp artificial intelligence solutions

NetApp
August 31, 2026

Table of Contents

Lustre with NetApp E-Series storage

Lustre with NetApp E-Series Storage - Introduction

Lustre with NetApp E-Series Storage is a validated parallel file system solution for AI and HPC workloads, built from repeatable building blocks of HA server pairs and E-Series all-flash arrays.

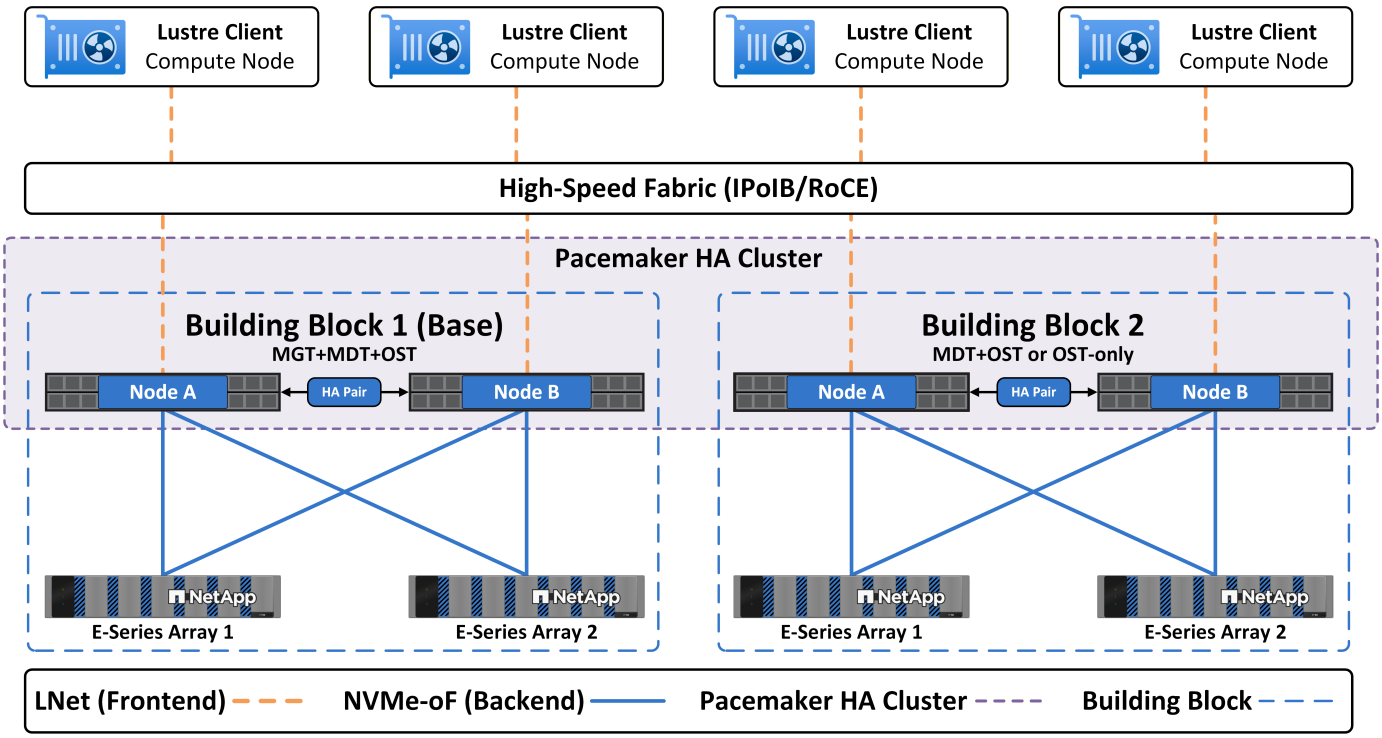
Solution overview

Lustre with NetApp E-Series Storage combines the open-source Lustre parallel file system with NetApp E-Series block storage to deliver a scalable, high-performance foundation for data-intensive workloads. Each building block pairs two OSS/MDS server nodes configured for high availability (HA) with two E-Series all-flash arrays. Backend NVMe-oF connectivity carries block I/O to the arrays and a separate LNet fabric connects clients and OSS/MDS server nodes for parallel file system access.

A base building block hosts the management server (MGS) and initial metadata targets (MDTs) and object storage targets (OSTs). Additional building blocks register against the base MGS to scale metadata capacity, data capacity, and aggregate throughput as workload demands grow.

The following figure shows an example deployment with multiple building blocks and Lustre clients connected over LNet.

Lustre with NetApp E-Series Storage solution overview



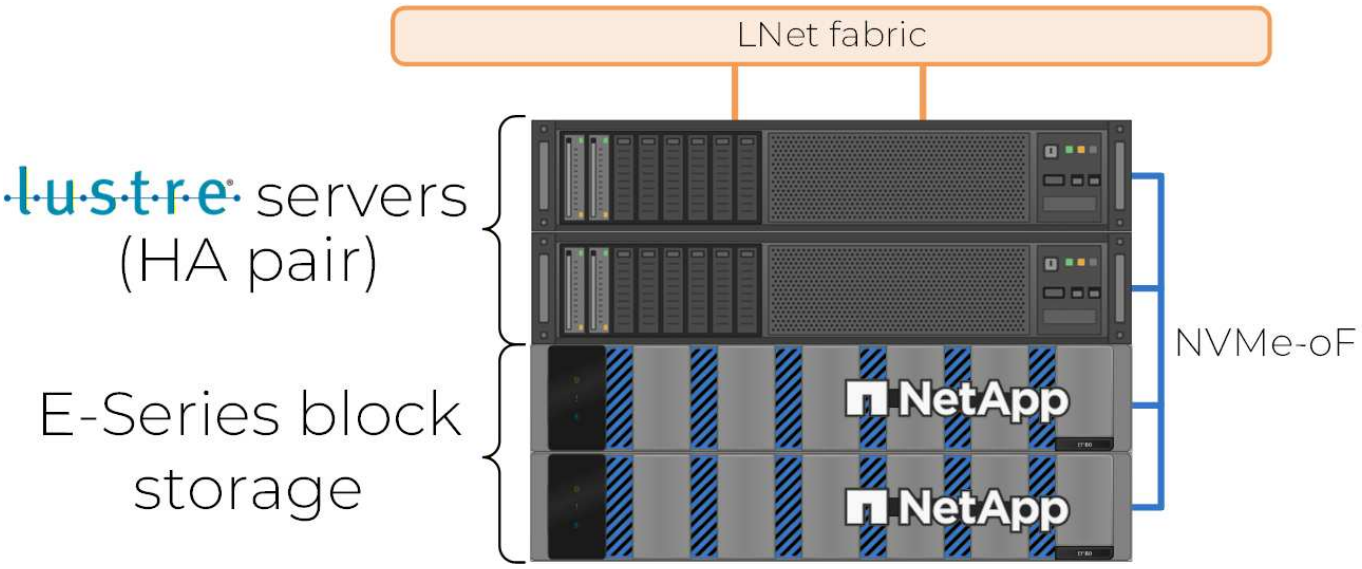
Lustre with NetApp E-Series Storage - Solution Architecture

Learn how Lustre with NetApp E-Series Storage uses building blocks, high availability, network topology, and clients so you can plan capacity, performance, and failover.

Building block design

A building block is the fundamental scalable unit of the NetApp E-Series Lustre solution. Each building block consists of two server nodes configured as a high availability (HA) pair that provide Lustre object storage server (OSS) and metadata server (MDS) functions, two NetApp E-Series all-flash arrays, dedicated backend NVMe over Fabrics (NVMe-oF) connectivity between servers and arrays, and dedicated frontend Lustre networking (LNet) connectivity for clients and inter-node communication. A Pacemaker and Corosync HA cluster can contain the server pair from one or more building blocks. The NetApp `ansible-lustre` collection uses Ansible automation to deploy and configure the Lustre services.

Lustre building block



Building blocks scale to match file system growth requirements. Every file system requires one base building block. The base building block hosts the management server (MGS) on a management target (MGT) and provides initial metadata target (MDT) and object storage target (OST) capacity. Additional building blocks extend the file system and register their MDT and OST targets against the base building block's MGS. The modular approach allows capacity and performance to scale independently based on workload needs.

NetApp recommends the following building block configuration profiles for most deployments. The MGS, MDT, and OST counts listed are recommended defaults that are optimized to maximize the performance and capacity of the E-Series storage arrays. Adjust target counts, volume sizes, and E-Series drive allocation in the Ansible inventory to suit workload needs. For example, deployments requiring greater metadata storage capacity can provision more MDTs and fewer OSTs within the same building block hardware.

The following table summarizes building block types and recommended target counts.

Type	MGS	MDTs	OSTs	Use
Base	1	8	32	First building block in a file system. Hosts the MGS and initial MDT and OST capacity.

Type	MGS	MDTs	OSTs	Use
MDT+OST	0	8	32	Adds metadata and data capacity. Register against the base building block MGS.
OST-only	0	0	32	Adds data capacity only. Register against the base building block MGS.

- **Drive type and pool layout:** E-Series supports traditional RAID volume groups and Dynamic Disk Pools (DDP). For TLC NVMe drives, use RAID 6 volume groups for OST storage and RAID 1 volume groups for MGS and MDT storage. For QLC NVMe drives, use DDP for the shared drive pool.
- **Target distribution:** Lustre targets in each building block are balanced across both OSS/MDS server nodes and both E-Series arrays. The layout spreads I/O across server CPUs and array controllers to improve NVMe-oF path performance and maintain redundancy. See target distribution and NVMe-oF connectivity in Hardware components.

Supported operating systems, Lustre versions, array firmware, and related building block components are listed in NetApp Interoperability Matrix Tool (IMT) under **E-Series Lustre**. For detailed volume counts, drive layouts, and sizing guidelines, see Hardware components. For software components, see Software components.

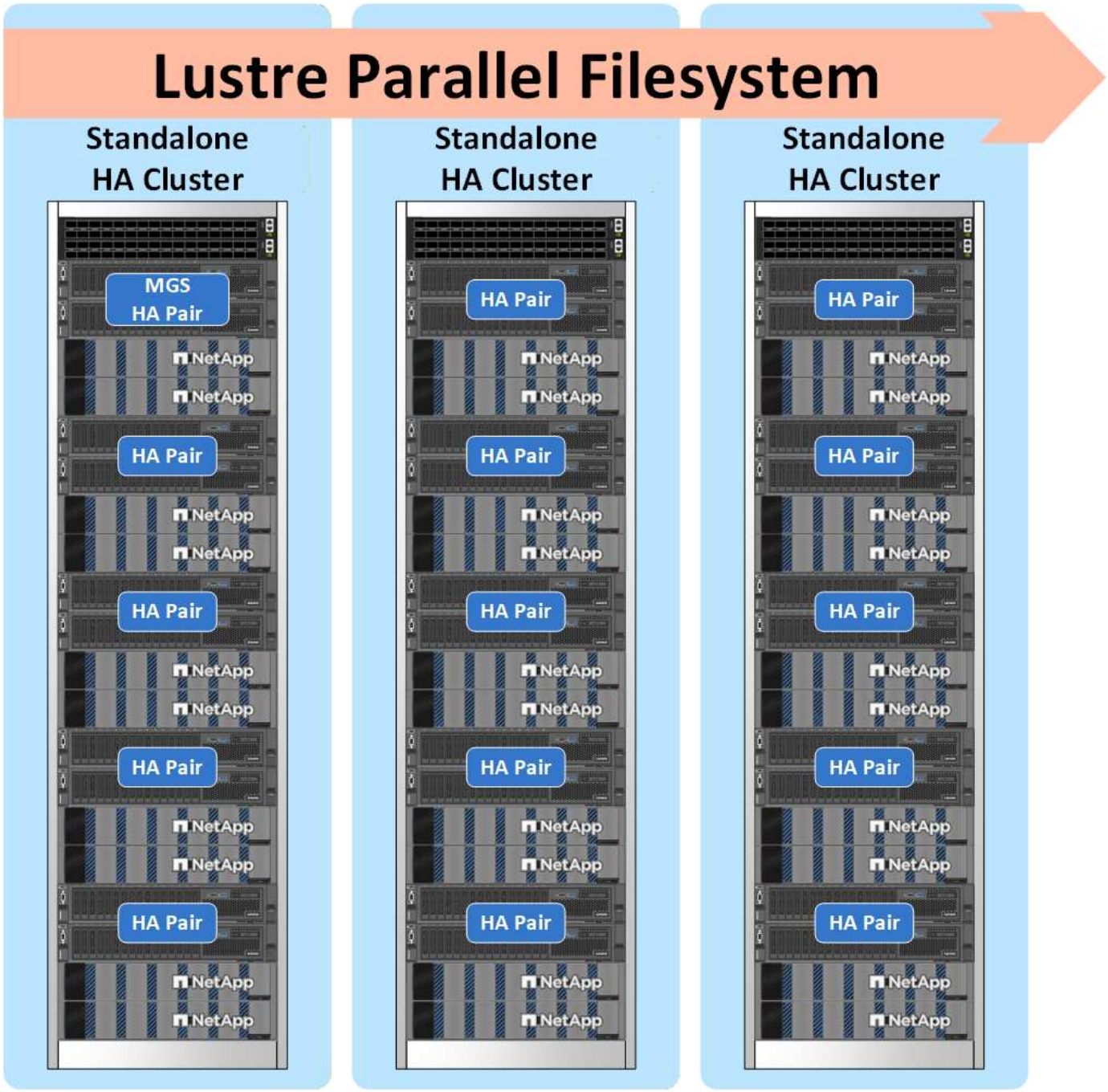
Scaling recommendations

The Lustre file system scales by adding building blocks when metadata capacity, data capacity, or both become limiting. Scale MDTs based on file count and metadata IOPS; scale OSTs based on capacity and aggregate throughput requirements.

1. **One base building block per file system:** Deploy exactly one base building block; it hosts the MGS and initial MDT and OST targets.
2. **Additional building block profile:** Add an OST-only building block when existing MDT capacity is sufficient and data capacity or throughput must increase. Add an MDT+OST building block when metadata performance or namespace capacity becomes the bottleneck.
3. **HA cluster limit:** Limit each Pacemaker and Corosync HA cluster to five building blocks (ten Lustre server nodes). Split larger deployments evenly into multiple HA clusters to avoid resource constraints in large configurations.

The following figure shows up to five building blocks stacked in a 42U rack (one Pacemaker HA cluster per rack). Multiple racks can participate in a single Lustre file system; only the base building block hosts the MGS.

Lustre parallel file system rack scaling



For sizing examples, see Sizing guidance.

HA architecture

Lustre with NetApp E-Series Storage uses a shared-disk high availability (HA) architecture integrated with NetApp E-Series storage. Pacemaker manages the HA resources, and Corosync provides cluster membership and messaging. Together, they manage Lustre target failover between the two OSS/MDS server nodes in each building block. Multipath NVMe-oF connects both OSS/MDS server nodes to the same E-Series volumes, so either node can take over target services when required.

Each MGS, MDT, and OST is configured as an HA resource with its dependencies in a Pacemaker resource group. Pacemaker ensures resources start and stop in the correct order, remain collocated on the same node, and run on only one node at a time. Concurrent access to the same target from both nodes could corrupt the underlying file system.

- **Target failover:** Both OSS/MDS server nodes are peers in the cluster, but each Lustre target is active on only one node at a time. A Pacemaker monitoring resource watches the health of each target and its dependencies and triggers a failover when a target becomes unavailable on its current node. On failure, Pacemaker restarts the affected resource group on the partner node in the correct order, and clients transparently reconnect to the target at its new location once services resume. Because each target activates on a single node, the file system is never exposed to concurrent writes from both nodes.
- **Shared storage access:** Multipath NVMe-oF paths connect each OSS/MDS server node to all E-Series controllers in the building block, so every target volume is reachable from either node. If a server node fails, the partner node already has active paths to the same volumes and can assume ownership of the targets without reconfiguring storage connectivity. If an array controller or path fails, multipath redirects I/O to a surviving controller. This dual redundancy lets Lustre targets fail over between OSS/MDS server nodes or array controllers without losing access to backing volumes.
- **STONITH fencing:** When a failure occurs, Pacemaker sometimes cannot communicate with the faulty node to confirm its targets are stopped. Before restarting those targets elsewhere, Pacemaker fences the faulty node, ideally by removing power, to ensure it is down. Fencing prevents a split-brain scenario in which both nodes access the same target concurrently and corrupt the underlying file system, preserving data integrity during failover. NetApp recommends `fence_redfish` for servers with Redfish-capable baseboard management controllers (BMCs). Other fencing agents, such as `fence_apc`, are also supported.

For cluster administration and fencing configuration, see the HA user guide in the ansible-lustre collection.

Network architecture

The solution uses separate network paths for storage backend traffic and LNet frontend traffic.

- **Backend (NVMe-oF):** OSS/MDS server nodes connect to E-Series arrays over NVMe-oF using NVMe/InfiniBand or NVMe/RoCE. The NVMe-oF path carries block I/O between Lustre target services and E-Series volumes. For EF80 six-HCA backend cabling, see Rack and cable Lustre hardware. For preferred target placement across nodes and arrays, see target distribution in Hardware components.
- **Frontend (LNet):** Lustre clients and OSS/MDS server nodes communicate over LNet using the `@o2ib` network type with the NVIDIA OFED stack. InfiniBand and RoCE are both validated frontend transports. The ansible-lustre collection configures all frontend interfaces for multi-rail LNet, which aggregates multiple ports to increase bandwidth and provide path redundancy for client and inter-node traffic.
- **Management and cluster:** An out-of-band management network provides server management, BMC access, and array management. STONITH fencing agents, such as `fence_redfish`, use this network to reach the server BMCs and remove power from a failed node. Corosync can also run cluster communication over this network as an additional ring alongside the frontend fabric. Configuring Corosync with multiple rings adds redundancy for cluster messaging and reduces the risk that a single network failure disrupts quorum.

Lustre clients

A Lustre client loads the client kernel module, establishes LNet connectivity to OSS/MDS server nodes, and mounts the file system to present a single, coherent, POSIX-compliant namespace to applications. I/O is distributed in parallel across active OSTs. Client nodes are external to a building block and use the file system through the frontend LNet fabric. Client operating systems are not listed in IMT for this solution; use the Lustre Support Matrix for tested client distributions and kernel versions supported by the deployed Lustre release (for example, Red Hat Enterprise Linux 9, SUSE Linux Enterprise Server 15, and Ubuntu 24.04).

Client nodes require connectivity to the same LNet fabric and network type as the OSS/MDS server nodes. If required, a LNet router can bridge separate LNet subnets or fabrics.

NetApp provides prebuilt Lustre server RPMs for Rocky Linux 9.8 and Red Hat Enterprise Linux 9.8. NetApp does not provide prebuilt client packages. Build client packages for the client operating system and kernel from the Lustre source code in the netapp-lustre repository.

After the client packages are installed, the optional `lustre_client` role in the ansible-lustre collection configures client network interfaces and LNet, enables multi-rail LNet when selected, validates connectivity, and manages a persistent systemd mount unit. The role does not build Lustre. It installs client packages only when `lustre_client_packages` is populated. If desired, clients can be configured manually. For step-by-step procedures, see Deploy the solution and the lustre_client role documentation.

Lustre with NetApp E-Series Storage - Hardware Components

Use these hardware requirements for servers, NetApp E-Series arrays, storage layout, and networking when you size and order Lustre with NetApp E-Series Storage. For software components, see Software components.

Server requirements

The solution uses a bring-your-own-server (BYOS) model. Each building block requires two OSS/MDS server nodes that meet or exceed the following specifications.

Node specifications

The following table lists the minimum server requirements per OSS/MDS node.

Component	Requirement
Quantity	2 nodes per building block
CPU	AMD EPYC or Intel Xeon, 32 cores or higher
Memory	256 GB DDR5 (minimum)
Network HCAs	6 dual-port 200Gb HCAs (see HCA connectivity)
PCIe slots	2× PCIe Gen5 x16 and 4× PCIe Gen5 x8 (see PCIe slot requirements)
Boot drives	2× drives in RAID 1 (software or hardware RAID recommended)

NetApp validated the solution using Lenovo ThinkSystem SR665 V3 servers. Equivalent servers from other vendors are supported when they meet these requirements.

HCA connectivity

The following table lists HCA, LNet frontend port, and NVMe-oF path requirements per OSS/MDS server node.

HCAs per node	LNet ports per Lustre node	NVMe-oF paths per Lustre node	Example HCA
6 (12 ports)	4	8 total (4 to each array)	MCX755106AS-HEAT (Dual port 200Gb PCIe gen5 card)

NetApp recommends six HCAs per node to maximize the bandwidth of the EF80 storage arrays.

PCIe slot requirements

Each OSS/MDS server node in an EF80 building block holds six dual-port HCAs: two for LNet and four for NVMe-oF. Slot width determines the bandwidth available to each HCA, so confirm the electrical width of every slot and riser rather than the width of the card alone. A Gen5 x16 card installed in a Gen5 x8 slot runs at x8.

- **LNet HCAs:** Install in PCIe Gen5 x16 slots to reach full frontend throughput.
- **NVMe-oF HCAs:** Install in PCIe Gen5 x8 or PCIe Gen5 x16 slots.
- **NUMA balance:** Divide the HCAs evenly between the two NUMA zones so that each zone hosts two LNet interfaces and four NVMe-oF interfaces.

The following table lists the minimum slots for each OSS/MDS server node in an EF80 building block.

Traffic type	HCAs per node	Minimum slot width	Slots per NUMA zone
LNet	2	PCIe Gen5 x16	1
NVMe-oF	4	PCIe Gen5 x8	2

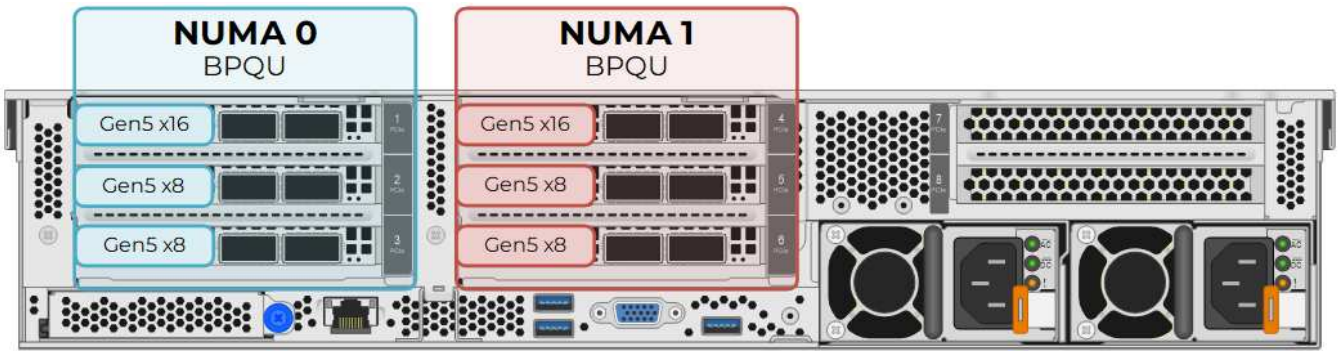
Optionally, you can split the ports on a dual-port HCA so that one port carries LNet traffic and the other carries NVMe-oF traffic. Install all four of these HCAs in PCIe Gen5 x16 slots so that every LNet port reaches full throughput, then add two more HCAs in Gen5 x8 or Gen5 x16 slots for the remaining NVMe-oF ports.

▼ Example riser configurations for Lenovo ThinkSystem SR665 V3

Both of the following riser combinations meet the slot requirements for an EF80 building block.

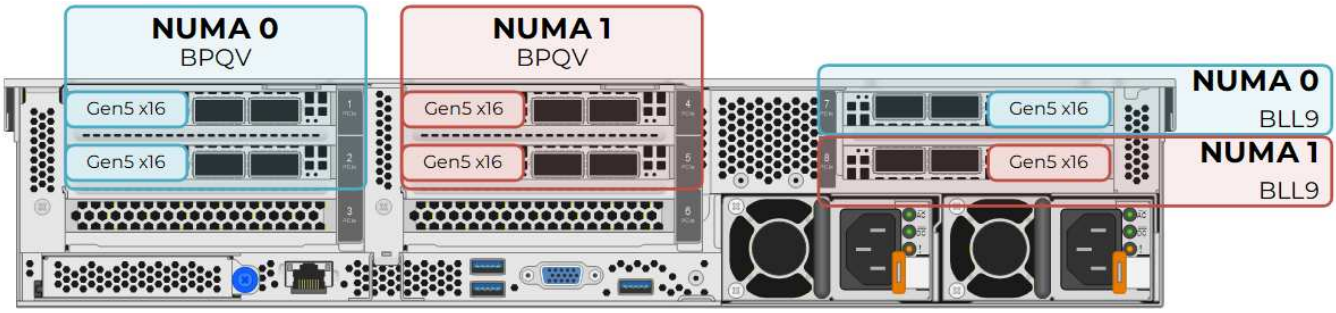
Minimum slot widths

Install a BPQU riser in riser positions 1 and 2. Each BPQU riser provides one PCIe Gen5 x16 slot and two PCIe Gen5 x8 slots. Together, the risers provide two Gen5 x16 slots for LNet and four Gen5 x8 slots for NVMe-oF, evenly divided between the NUMA zones.



All Gen5 x16 slots

Install a BPQV riser in riser positions 1 and 2, and a BLL9 riser in riser position 3. Each BPQV riser provides two PCIe Gen5 x16 slots. The BLL9 riser provides slot 7 on NUMA zone 0 and slot 8 on NUMA zone 1, both PCIe Gen5 x16. This combination provides six Gen5 x16 slots, three per NUMA zone, and supports splitting LNet and NVMe-oF traffic across the ports of the same HCA.



EF80 standard building block

The EF80 is the validated standard platform for this solution release.

Array specifications

The following table lists EF80 array specifications for a building block (two arrays).

Component	Specification
Model	NetApp EF80
Form factor	2U base chassis, 24 internal NVMe SSD slots
Controllers	Dual controllers (A and B)
Drives	24× NVMe SSD per array
I/O connectivity	8× 200Gb NVMe/IB or NVMe/RoCE host ports per array in the validated Lustre design

The EF80 platform supports up to twelve host ports per array when three two-port host I/O modules are installed in each controller. The validated Lustre design uses host I/O modules in slots 1 and 2 for eight host ports per array.

Each EF80 array also uses the dedicated slot 4 I/O module for inter-controller mirroring. Cable controller A port 4a to controller B port 4a, and controller A port 4b to controller B port 4b. These connections provide cache mirroring and I/O shipping and are not used for host NVMe-oF traffic. See Cable the EF50 and EF80 inter-controller mirroring connections.

For full EF-Series specifications across all models, see the NetApp EF-Series all-flash array datasheet.

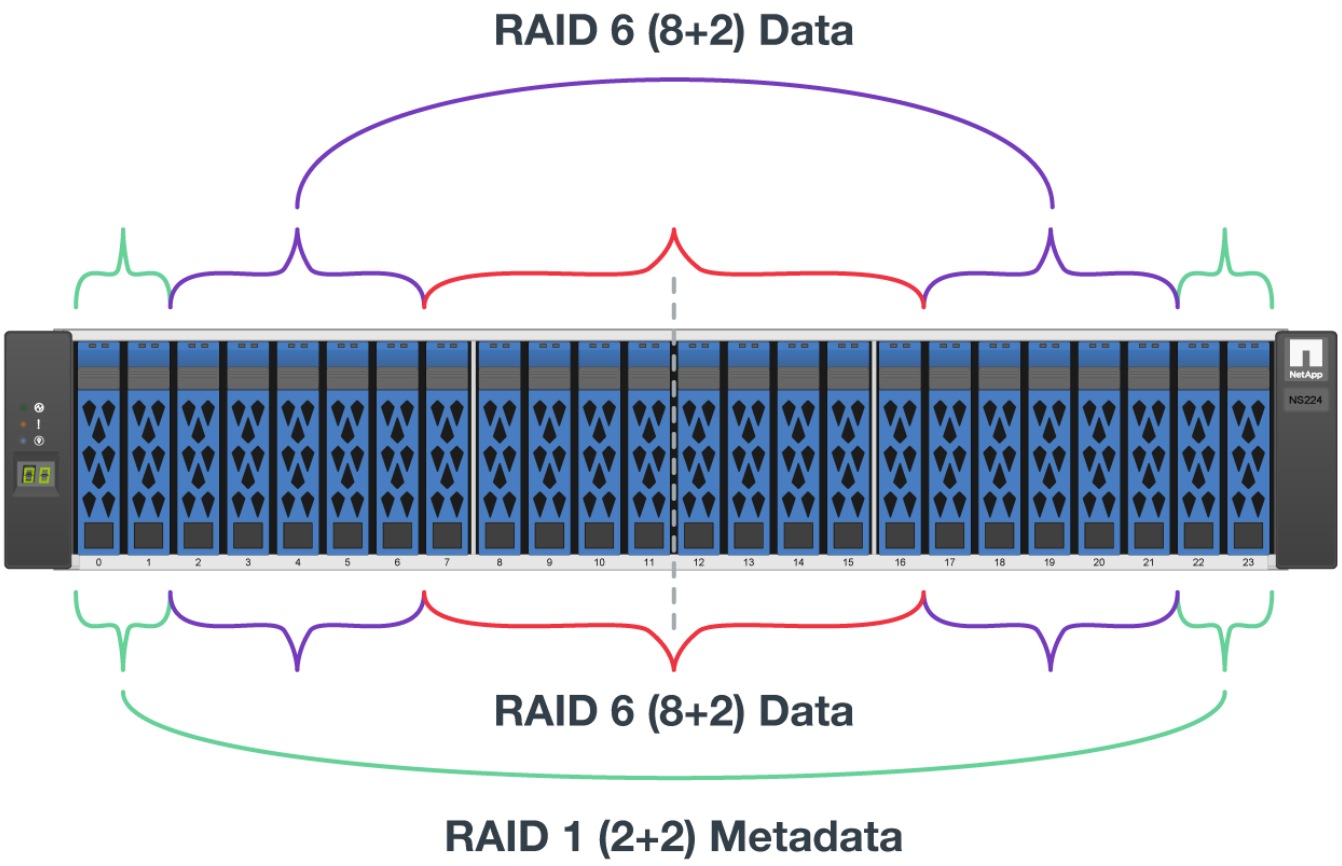
Drive layout (24 drives per array)

Each EF80 array in a base building block uses all twenty-four NVMe drive slots:

- 4× drives in RAID 1 for MGS/MDT storage (shared volume group or DDP allocation)
- 10× drives in RAID 6 for OST storage (first OST pool)
- 10× drives in RAID 6 for OST storage (second OST pool)

This layout applies when using 3.84 TB, 7.68 TB, or 15.3 TB drives with traditional volume groups. When using 30.7 TB or 61.4 TB Capacity Flash (QLC) drives, provision a single Dynamic Disk Pool across all twenty-four drives and create RAID 1 volumes inside the pool for MGS and MDT volumes, while using default RAID 6 volumes for OSTs.

1.92 TB drives are not currently recommended for this solution. Use one of the validated drive capacities listed above.



Volume and target counts (base building block)

The following table lists MGS, MDT, and OST counts for a base building block.

Target type	Count per BB	RAID / pool	Notes
MGS	1	RAID 1	Base building block only; array 1
MDT	8	RAID 1	4 per array
OST	32	RAID 6 (TLC) or DDP (QLC)	16 per array

Use the following volume sizing guidelines as a starting point in the Ansible inventory.

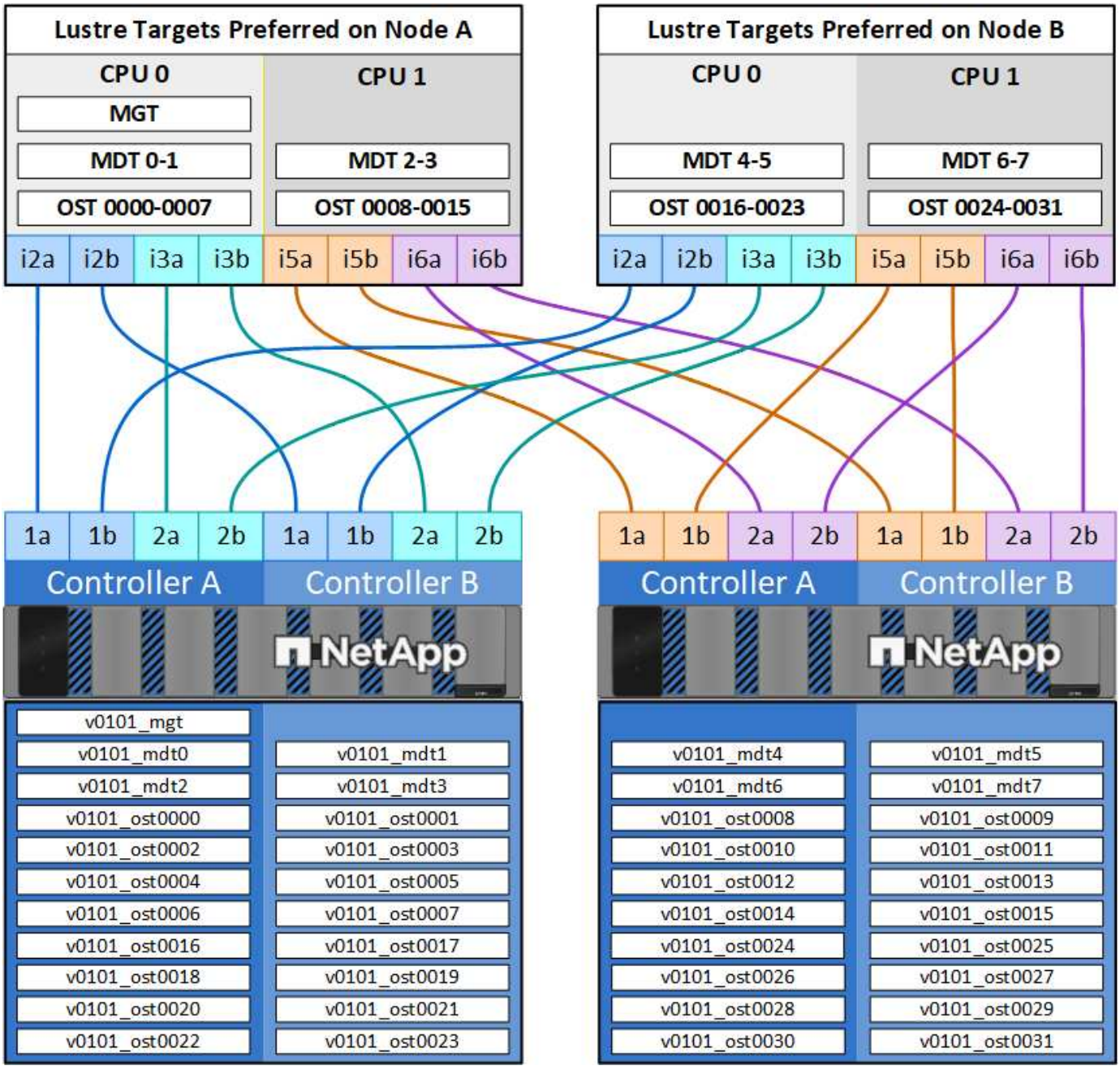
Volume	Recommended size	Notes
MGS	5–10 GiB	Configuration data only

Volume	Recommended size	Notes
MDT	RAID 1 capacity ÷ MDT count	Approximately 1–2 TiB each (typical)
OST	RAID 6 or DDP capacity ÷ OST count per pool	Scales with drive capacity; see Sizing guidance

Primary building block volume distribution

The following figure shows preferred Lustre target placement and NVMe-oF connectivity across OSS/MDS server nodes and E-Series arrays in a base EF80 building block. The diagram labels the management target as **MGT** (management target); one MGT hosts the MGS (management server) service referenced elsewhere in this document.

Base building block target distribution (EF80)



The following table lists how the volumes are distributed across the two arrays and which server each target prefers.

Target type	Array 1	Array 2	Server 1	Server 2	Notes
MGS	1	0	1	0	Base building block only
MDT	4	4	4	4	Each server prefers 2 MDTs from each array
OST	16	16	16	16	8 per RAID 6 pool × 2 pools per array, or equivalent DDP allocation
Total volumes	**21**	**20**	**21**	**20**	

Storage pool selection: TLC vs QLC

Choose the E-Series pool type based on the NVMe drive capacity in the arrays:

- **3.84 TB, 7.68 TB, and 15.3 TB drives:** Use **RAID 6 volume groups** for OST volumes and **RAID 1 volume groups** for MGS and MDT volumes, or use DDP for the shared drive pool.
- **30.7 TB and 61.4 TB Capacity Flash (QLC) drives:** Use **Dynamic Disk Pools (DDP)** only. Create RAID 1 volumes inside the DDP for MGS and MDT storage. Create RAID 6 volumes for OST storage from the DDP.

For per-building-block usable capacity estimates by drive size and layout, see Sizing guidance.

Network requirements

Backend (NVMe-oF)

- NVMe/InfiniBand or NVMe/RoCE between each OSS/MDS node and each E-Series array
- MTU 9000 on NVMe/RoCE backend interfaces (typical)
- Eight paths from each Lustre node to the storage arrays (four to each array)
- EF80 uses six HCAs per node (i2, i3, i5, and i6 for NVMe-oF; i1 and i4 for LNet). Cable Node A to controller ports whose labels end in `a`, such as `1a` and `2a`. Cable Node B to controller ports whose labels end in `b`, such as `1b` and `2b`. See EF80 six-HCA backend cabling.

Frontend (LNet)

- IPoIB or RoCE for LNet (`@o2ib` network type)
- MTU 9000 on RoCE frontend interfaces (typical)
- Multi-rail LNet recommended when four frontend ports are available (EF80: i1 and i4 HCAs, both ports)
- Dedicated InfiniBand or RoCE switch fabric connecting the OSS/MDS server nodes and Lustre clients
- Lossless RoCE (PFC) recommended on RoCE fabrics

Management

- Out-of-band management network for server BMCs and array management ports

- Dedicated Corosync network or shared frontend fabric (site-dependent)

Lustre with NetApp E-Series Storage - Software Components

Review the software stack and Ansible automation that deploy Lustre with NetApp E-Series Storage. For servers, storage, and networks, see Hardware components.

Building block software

The following table lists the software components used by E-Series arrays and OSS/MDS server nodes. Use the NetApp Interoperability Matrix Tool (IMT) as the authoritative source for supported versions and component combinations. Search for **E-Series Lustre** and confirm the operating system, Lustre, SANtricity OS, DOCA-OFED, HCA firmware, and HA package versions before deployment.

Component	Function
SANtricity OS	Runs on each E-Series array and provides SANtricity System Manager, dual-active controller high availability with automated I/O path failover, automatic load balancing and path monitoring, Dynamic Disk Pools and traditional RAID, automatic drive rebuilds, mirrored cache protection, Data Assurance, proactive drive health monitoring, and online software, firmware, and configuration changes.
Lustre	Presents clients with a single POSIX-compliant namespace while distributing metadata and file data across multiple targets for parallel access. The management server (MGS) stores file system configuration, metadata targets (MDTs) manage the namespace, and object storage targets (OSTs) store file data on E-Series volumes. This separation enables performance and capacity to scale by adding building blocks.
Red Hat Enterprise Linux (RHEL) or Rocky Linux	Provides the kernel, system services, and package framework for Lustre target services, HA clustering, and storage and network connectivity. The supported operating system repositories also supply Pacemaker, Corosync, fence agents, NVMe-oF, and multipath packages that are tested together as a solution stack.
High-availability cluster services (Pacemaker, Corosync, and fence agents)	Together, these services provide coordinated high availability for Lustre targets across OSS/MDS server nodes. They maintain cluster membership and quorum, monitor resources, orchestrate ordered failover, and isolate unresponsive nodes before activating shared targets elsewhere. This coordination prevents split-brain access to E-Series volumes and protects file system integrity during failures.
NVIDIA DOCA-OFED	Provides HCA drivers, RDMA libraries, and management utilities for InfiniBand and RoCE fabrics. The same software stack supports low-latency backend NVMe-oF access to E-Series arrays and high-throughput frontend LNet communication with Lustre clients.

NetApp provides prebuilt Lustre server RPMs for Rocky Linux 9.8 and Red Hat Enterprise Linux 9.8.

Ansible automation

Lustre with NetApp E-Series Storage is delivered and deployed using Ansible automation from a control node with management access to the E-Series arrays and OSS/MDS server nodes. An Ansible inventory models the desired solution, including arrays, servers, storage assignments, network interfaces, and HA cluster resources.

The NetApp E-Series Lustre Ansible collection orchestrates the end-to-end deployment by using the SANtricity and Host collections. Together, these collections provision and map E-Series storage, prepare the server operating system, configure NVMe-oF and LNet connectivity, deploy Lustre targets, and create Pacemaker resources. This automation makes the configuration repeatable and simplifies adding building blocks to an existing file system.

The following table summarizes the principal software packages used on the Ansible control node. See the NetApp E-Series Lustre Ansible collection for current package dependencies.

Package	Dependency or purpose
Python	Provides the runtime for Ansible and required Python modules.
ansible-core	Requires Python and runs the NetApp E-Series Ansible collections.
Additional Python packages	`cryptography`, `ipaddr`, `netaddr`, `passlib`, `resolvelib`, `jinja2`, and `pyyaml`
NetApp E-Series Lustre Ansible collection	Configures Lustre, LNet, NVMe-oF host connectivity, and Pacemaker resources.
NetApp E-Series SANtricity Ansible collection	Configures E-Series arrays, storage pools, volumes, and host mappings through the SANtricity Web Services API.
NetApp E-Series Host Ansible collection	Prepares OSS/MDS server nodes for LNet and NVMe-oF connectivity to E-Series arrays.

Lustre client software

Lustre client software enables a system to access the parallel file system through the frontend LNet fabric. Each client needs Lustre kernel modules and Lustre client utilities that are compatible with both its running kernel and the Lustre release deployed on the servers.

Use the Lustre Support Matrix to find a compatible client operating system and kernel for your Lustre version. NetApp does not provide prebuilt Lustre client packages. Build client packages for the client operating system and kernel from the NetApp-provided Lustre source code in the netapp-lustre repository. After the Lustre client packages are installed, the `lustre_client` role can configure network interfaces, LNet, and create a persistent systemd mount unit for the Lustre file system.

For client deployment steps, see Deploy the solution.

Lustre with NetApp E-Series Storage - Sizing Guidance

Size Lustre with NetApp E-Series Storage building blocks by capacity and metadata needs, decide when to add capacity, and review the factors that affect performance.

Capacity sizing

A base building block with two EF80 arrays provides the following Lustre target layout. For preferred target

placement and NVMe-oF paths, see Primary building block volume distribution in Hardware components.

Component	Count	Typical volume size (per target)
MGS	1	5-10 GiB
MDT	8	Size varies by drive capacity and RAID layout
OST	32	Size varies by drive capacity and RAID layout

Each EF80 array uses twenty-four NVMe drives. Supported capacities are 3.84 TB, 7.68 TB, and 15.3 TB with traditional volume groups (RAID 1 for MGS/MDT and RAID 6 for OST) or DDP, and 30.7 TB or 61.4 TB Capacity Flash (QLC) drives with DDP only. 1.92 TB drives are not currently recommended for this solution. For drive slot layout and pool selection, see Hardware components.

The following table lists approximate usable capacity for an EF80 building block (two arrays, 48 drives). Array usable capacity is not the same as final Lustre file system usable capacity. MDT inode allocation, journals, overprovisioning, and inventory volume percentages reduce capacity available to applications.

Drive capacity	Layout (per array)	Approximate usable (building block)
3.84 TB	RAID 1 (2+2) + RAID 6 (10) + RAID 6 (10)	138.08 TB
3.84 TB	DDP (24 drives, 2 reserved)	133.63 TB
7.68 TB	RAID 1 (2+2) + RAID 6 (10) + RAID 6 (10)	276.35 TB
7.68 TB	DDP (24)	267.47 TB
15.3 TB	RAID 1 (2+2) + RAID 6 (10) + RAID 6 (10)	552.88 TB
15.3 TB	DDP (24)	535.15 TB
30.7 TB	DDP only	1,070.35 TB
61.4 TB	DDP only	2,162.70 TB

Size metadata and data separately, then set volume sizes in the Ansible inventory. The deployment templates format targets with ldiskfs; MDT uses the Lustre default inode ratio, and OST templates set `-i 4096`.

- **Metadata (MDT):** Size MDTs by expected file count. Plan roughly twice the expected file count for growth. An undersized MDT can prevent new file creates even when OST capacity remains.
- **Data (OST):** Size OSTs from usable capacity and throughput needs.
- **MGS:** Use a small management target (5-10 GiB) for file system configuration only.

Adjust MDT and OST volume sizes in the inventory for the selected drive capacity. Change `format_options.mkfsoptions` in `eseries_lustre_filesystem_mdt` or `eseries_lustre_filesystem_ost` only when a site needs a different inode ratio. For background on inode ratios, see the Lustre Wiki: Lustre Tuning. For when to add building blocks, see Scaling.

Scaling

Add building blocks when capacity or metadata load requires it:

- **OST-only:** File count and metadata load are within existing MDT capacity, and you need more data capacity or aggregate throughput. Adds 32 OSTs and two OSS/MDS server nodes.
- **MDT+OST:** File count or metadata rate is approaching MDT capacity, or you need both more metadata throughput and data capacity. Adds 8 MDTs and 32 OSTs.

Use `mdt.*.md_stats` on existing MDTs to confirm whether metadata is saturated. Limit each Pacemaker/Corosync cluster to five building blocks (ten OSS/MDS server nodes). For larger deployments, split building blocks evenly across multiple HA clusters. See Solution architecture for scaling rules.

The following example shows a file system with a base building block plus an OST-only building block in one HA cluster.

Building block	Type	Servers	Arrays	MGS	MDT	OST
BB1	Base	2	2	1	8	32
BB2	OST-only	2	2	0	0	32
Total		4	4	1	8	64

BB2 registers its OST targets against the MGS in BB1 using `mgsnode=`. OST indices for BB2 are assigned globally (for example, OST0032 through OST0063) so that no index conflicts with BB1.

Performance

Formal validation uses IOR, mdtest, and fio from Lustre clients to characterize relative throughput, IOPS, and metadata behavior, and to validate HA failover. These tests do not publish guaranteed performance numbers. Synthetic benchmarks measure best-case behavior and may not reflect real application performance.

Performance scales approximately with the number of building blocks. Realized results depend on drive type and pool layout, NVMe-oF path count, LNet fabric and client count, and the mix of data versus metadata I/O.

Contact your NetApp account team for workload-specific sizing and performance recommendations.

For drive layout and volume counts, see Hardware components. For deployment steps, see Deploy the solution. For field-level inventory guidance, see Customize the Lustre Ansible inventory.

Deploy the solution

Deploy Lustre with NetApp E-Series Storage

Deploy Lustre with NetApp E-Series Storage by completing the hardware, environment preparation, inventory, Ansible deployment, and verification tasks in order.

Prerequisites

Before you start the deployment, ensure the following:

- You followed the sizing and building-block guidance in Sizing guidance and selected hardware that matches Hardware components.
- You confirmed that the server, HCA, E-Series, operating system, Lustre, SANtricity OS, and DOCA-OFED combination is listed for **E-Series Lustre** in the NetApp Interoperability Matrix Tool.

- All OSS/MDS server nodes and E-Series arrays for the planned building blocks are on site and ready to rack.
- You have credentials and network access for server BMCs, array management ports, and the host that will serve as the Ansible control node.

NetApp supports deploying the Lustre HA cluster and Lustre clients with the NetApp E-Series Lustre Ansible collection. Do not configure Lustre targets, LNet, NVMe-oF host connectivity, or Pacemaker resources manually.

Deployment workflow

Complete the following tasks in order:

1. Rack and cable Lustre hardware.

   Rack the Lustre servers and EF80 arrays, then cable the backend NVMe-oF and frontend LNet fabrics.

2. Prepare the Lustre deployment environment.

   Configure the arrays and servers, then install Ansible and its dependencies on the control node.

3. Customize the Lustre Ansible inventory.

   Select the template for the site's fabric and populate the servers, arrays, networks, building blocks, and credentials.

4. Deploy the Lustre HA cluster and clients.

   Prepare the server operating system, install NVIDIA DOCA-OFED, run the main deployment playbook, and configure Lustre clients.

5. Verify and expand the Lustre deployment.

   Confirm cluster health, mounts, and capacity before production use. Extend the existing inventory when the file system needs another building block.

Related information

- NetApp E-Series Lustre Ansible collection
- NetApp Lustre source repository
- HA administration user guide
- Performance tuning guide
- Lustre client deployment templates

Rack and Cable Lustre Hardware

Rack the Lustre servers and NetApp EF80 arrays, then cable the backend NVMe-oF and frontend LNet fabrics for each building block.

Before you begin

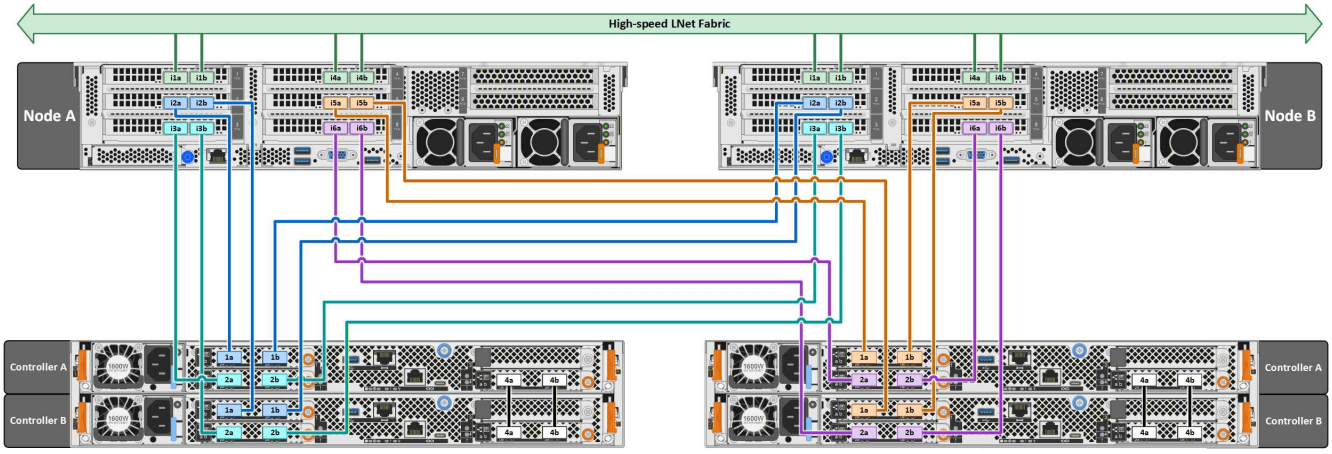
Review the HCA counts, PCIe slot requirements, path requirements, and MTU guidance in Hardware components.

Steps

1. Rack and cable all Lustre servers and E-Series storage arrays according to the building block topology.

2. For each EF80 array, cable controller A port 4a to controller B port 4a, and controller A port 4b to controller B port 4b. These dedicated connections provide inter-controller mirroring and are not used for host traffic. See Cable the EF50 and EF80 inter-controller mirroring connections.
3. Cable the backend NVMe-oF network between servers and arrays according to the EF80 six-HCA topology.

EF80 six-HCA backend cabling (RoCE or InfiniBand)



4. Connect the four frontend LNet interfaces (the i1 and i4 HCAs) on each Lustre server and the Lustre clients to the dedicated InfiniBand or RoCE switch fabric.
5. When using RoCE, configure the fabric for lossless operation with priority flow control (PFC). The Ansible playbooks configure lossless RoCE on all interfaces in the building block.

After you finish

Prepare the servers, arrays, and Ansible control node.

Prepare the Lustre Deployment Environment

Configure management access and base settings on each EF80 array and Lustre server, then prepare an Ansible control node to deploy the solution.

Prepare E-Series arrays

Steps

1. Configure the management interface on each controller and verify that SANtricity System Manager is reachable.
2. Set the array name and administrative credentials that the Ansible inventory will use.
3. Confirm that the array is in an optimal state and that all drives are present and reporting correctly.

For instructions on configuring the management connection, see the E-Series documentation.

Prepare Lustre servers

Steps

1. Configure the baseboard management controller (BMC) on each server for out-of-band access, and set the UEFI system settings for maximum performance.
2. Install a supported operating system (RHEL or Rocky Linux), then configure the host name and the management network port.
3. Prepare all firmware and software packages to meet the requirements for the **E-Series Lustre** solution, as described in Software components.

Prepare the Ansible control node

Designate a virtual or physical Linux system with management network access to all Lustre servers and E-Series arrays to be the Ansible control node.

The following steps use Ubuntu 24.04. For instructions for another Linux distribution, see the Ansible installation documentation.

Steps

1. Install Python, pip, and the Python virtual environment package:

```
sudo apt-get install python3 python3-pip python3-setuptools python3-venv
```

2. Create and activate a Python virtual environment:

```
python3 -m venv ~/pyenv
source ~/pyenv/bin/activate
```

3. Install Ansible and the required Python packages in the activated virtual environment:

```
pip install "ansible-core>=2.19" cryptography jinja2 ipaddr netaddr \
  passlib pyyaml resolvelib
```

4. Install the NetApp E-Series Lustre Ansible collection. The collection installation also installs its dependent collections, including the SANtricity and Host collections:

```
ansible-galaxy collection install netapp_eseries.lustre
```

5. Verify the installed Ansible, Python, and collection versions:

```
ansible --version
python3 --version
ansible-galaxy collection list netapp_eseries.lustre
```

6. Configure passwordless SSH from the control node to each Lustre server. Replace `<ip_or_hostname>` with the management IP address or host name of a server:

```
ssh-keygen
ssh-copy-id <ip_or_hostname>
```

Repeat `ssh-copy-id` for each Lustre server.

Do not configure passwordless SSH to the E-Series arrays. Ansible manages the arrays through the SANtricity Web Services API by using the credentials defined in the inventory.

After you finish

Customize the Ansible inventory.

Customize the Lustre Ansible Inventory

Create a working copy of an EF80 deployment template and customize its inventory for your servers, arrays, networks, and Lustre building blocks.

Keep the selected inventory under its fabric and platform directory. Retain the shared `playbooks` directory and `passwords.yml` file at the root of `building_blocks`.

Select a deployment template

Start from the template that matches the site's network fabric:

- **InfiniBand:** IB_H2_IB_DAC_A2_EF80/lustre_building_block_cluster_1

- **RoCE:** ROCE_H2_ROCE_DAC_A2_EF80/lustre_building_block_cluster_1

Configure site-specific settings

Set the site-specific values in the inventory, along with the local repository and udev-rule settings. Line references in the table point to the RoCE EF80 template files in the `ansible-lustre release/1.0.0` tag. The InfiniBand template uses the same variable names and structure except for the LNet interface variable, which is noted in the table.

Field	Ansible variable	Template file (`release/1.0.0`)	Scope	Notes
File system name	`eseries_lustre_filesystem_mgt.format_options.fsname`, `eseries_lustre_filesystem_mdt.format_options.fsname`, `eseries_lustre_filesystem_ost.format_options.fsname`	ha_cluster.yml#L76, #L90, #L110	Cluster-wide	Nested under `format_options` for the MGT, MDT, and OST maps. Maximum 8 characters. Use the same value in all three locations. A top-level `fsname` key has no effect.
MGS NIDs for target registration	`eseries_lustre_filesystem_mdt.format_options.mgsnode`, `eseries_lustre_filesystem_ost.format_options.mgsnode`	ha_cluster.yml#L98, #L118	Cluster-wide	Nested under MDT and OST `format_options`. Include the frontend LNet NIDs from both nodes in the HA pair. List the NIDs from the node configured as the preferred MGS owner first, followed by the partner node's NIDs.
Pacemaker cluster name	`eseries_lustre_pacemaker_cib_properties_overrides.cluster_properties.cluster-name`	ha_cluster.yml#L228	Cluster-wide	Nested under Pacemaker CIB property overrides. Unique name for the HA cluster.
BMC fencing addresses	`eseries_lustre_pacemaker_fencing_agents.fence_redfish.ip`	ha_cluster.yml#L211, #L214, #L217, #L220	Repeat per Lustre server	Redfish BMC IP address for each server node.
Alert email recipients	`eseries_lustre_alert_email_list`	ha_cluster.yml#L233	Cluster-wide	Recipients for HA resource and failure notifications.
Mail domain	`eseries_lustre_mail_service_overrides.conf.mydomain`	ha_cluster.yml#L240	Cluster-wide	Postfix domain used to send alert email.

Field	Ansible variable	Template file (`release/1.0.0`)	Scope	Notes
NTP time sources	`eseries_lustre_time_sync_overrides.server_pools`	ha_cluster.yml#L250	Cluster-wide	Time sources for the cluster nodes. Specify an NTP pool or one or more individual NTP servers. Include the required `pool` or `server` directive and options such as `iburst`.
Interface PCI-to-name udev map	`eseries_ip_udev_rules`	ha_cluster.yml#L172	Repeat per HCA slot layout	Map each PCI slot to a persistent interface name (values at L182). Collect with `lspci -nnvmm`.
Local package repository (optional)	`eseries_common_yum_repos`	ha_cluster.yml#L256, #L261	Cluster-wide	For air-gapped installs, uncomment the block and set the repository `url` (L264).
Server management IP	`ansible_host`	host_vars/lustre_01.yml#L10	Repeat per Lustre server	One `host_vars/lustre_NN.yml` file per server.
NVMe-oF backend interfaces	`eseries_nvme_roce_interfaces` (RoCE) / `eseries_nvme_ib_interfaces` (InfiniBand)	RoCE host_vars/lustre_01.yml#L21, InfiniBand host_vars/lustre_01.yml#L17	Repeat per Lustre server	Backend interface addresses used for NVMe-oF storage traffic to the E-Series arrays.
LNet cluster interfaces	`eseries_roce_interfaces` (RoCE) / `eseries_ipoib_interfaces` (InfiniBand)	RoCE host_vars/lustre_01.yml#L47, InfiniBand host_vars/lustre_01.yml#L38	Repeat per Lustre server	Frontend LNet interface addresses (RoCE values at L56; InfiniBand values at L47).
Corosync node IPs	`eseries_lustre_corosync_node_ips`	host_vars/lustre_01.yml#L58	Repeat per Lustre server	List all node IPs in the HA cluster (values at L62).
Lustre target service NIDs	`lustre_target_service_nodes`	host_vars/lustre_01.yml#L64	Repeat per Lustre server	Include frontend LNet NIDs (`@o2ib`) from both nodes in the HA pair so that target services remain reachable after failover (values at L67).
Array management IP	`ansible_host`	host_vars/netapp_01.yml#L13	Repeat per E-Series array	Management IP of one controller; the API URL derives from it.

Field	Ansible variable	Template file (`release/1.0.0`)	Scope	Notes
Building-block group membership	`mgs` / `mds_NN` / `oss_NN`	lustre_inventory.yml	Per building block	Each inventory hostname must match the basename of its corresponding `host_vars` file. For example, host `lustre_01` uses `host_vars/lustre_01.yml`, and `netapp_01` uses `host_vars/netapp_01.yml`. Include the MGS group only in the first building block. Include MDS groups in the base building block and each MDT+OST expansion building block; omit MDS groups only from OST-only expansion building blocks. Register expansion targets against the existing MGS with `mgsnode`.

Repeat the settings for each host and building block. Create one `host_vars/lustre_NN.yml` file per Lustre server and one `host_vars/netapp_NN.yml` file per array, and repeat the inventory groups for each additional building block.

The deployment playbooks load array, Pacemaker, and BMC credentials from the shared `building_blocks/passwords.yml` file. Populate this file and encrypt it with Ansible Vault before deployment, or override the values securely at runtime with `--extra-vars`. Do not commit plaintext credentials to source control.

IP addresses, hostnames, interface names, and volume sizing in the deployment templates are examples. Modify all inventory files for the environment before running any playbook.

After you finish

Deploy the Lustre HA cluster and clients.

Deploy the Lustre HA Cluster and Clients

Prepare the server operating system, install NVIDIA DOCA-OFED, deploy the Lustre HA cluster, and configure Lustre clients by using NetApp Ansible playbooks.

Before you begin, complete and secure the inventory described in Customize the Lustre Ansible inventory.

Deploy the Lustre HA cluster

Steps

1. From the `building_blocks/playbooks` directory in your working copy, prepare the operating system

on the Lustre servers. Replace `<inventory_path>` with the path to your customized inventory directory:

```
ansible-playbook -i <inventory_path>/lustre_inventory.yml
deploy_lustre_os/deploy_lustre_os_playbook.yml
```

2. Install NVIDIA DOCA-OFED on each Lustre server. Build the kernel modules against the kernel that the previous step installed. Follow the procedure for your kernel in NVIDIA DOCA-Host installation and upgrade. The following example shows a DOCA-OFED installation on RHEL or Rocky Linux.
   a. Install the DOCA host repository package and refresh the package cache. Replace `<doca_host_package>` with the DOCA host package for your operating system and architecture:

   ```
   dnf install -y wget tar
   wget https://www.mellanox.com/downloads/DOCA/<doca_host_package>.rpm
   rpm -i <doca_host_package>.rpm
   dnf clean all
   dnf makecache
   ```

   b. Build the DOCA kernel modules for the running kernel. The script prints the path of the package it creates:

   ```
   dnf install -y doca-extra
   /opt/mellanox/doca/tools/doca-kernel-support
   ```

   c. Install the kernel module repository package that the script created, then refresh the package cache. Replace `<doca_kernel_repo>` with the path from the previous step:

   ```
   rpm -Uvh <doca_kernel_repo>.rpm
   dnf makecache
   ```

   d. Install the DOCA-OFED user space, kernel, and NVMe packages. Replace `<kernel_version>` with the running kernel version:

   ```
   dnf install -y doca-ofed-userspace
   dnf install -y --disablerepo=doca doca-kernel-<kernel_version>
   dnf install -y kmod-mlnx-nvme
   ```

3. Run the main deployment playbook:

```
ansible-playbook -i <inventory_path>/lustre_inventory.yml
lustre_playbook.yml
```

Adjust `--forks` for the size of the deployment to control how many hosts Ansible configures in parallel. For example, add `--forks 20` for a larger cluster.

The playbook provisions E-Series volume groups or DDP pools and volumes, configures NVMe-oF host connectivity, formats and mounts the Lustre targets, configures LNet, applies performance tuning, and configures the Pacemaker HA resources.

A deployment with one building block forms a two-node Pacemaker cluster. Node A is given an extra vote in the case of failover. To achieve quorum in a two-node cluster, consider configuring a qdevice. When a deployment contains multiple building blocks, each building block contributes a two-node server pair to the larger Pacemaker cluster, up to the documented cluster limit. Review the fencing and quorum configuration in the HA administration user guide so the cluster can safely recover targets during a node failure.

Deploy Lustre clients

Deploy Lustre clients on any system that requires file system access by customizing the client deployment template and running the client playbook.

Steps

1. Select a compatible client operating system and kernel from the Lustre Support Matrix. Build and install the Lustre client packages for that kernel from netapp-lustre if they are not already installed.
2. Create a working copy of the `clients` template directory, including its shared `passwords.yml` file, and select the template that matches your fabric:
   - **InfiniBand:** IB_lustre_cluster_clients
   - **RoCE:** ROCE_lustre_cluster_clients
3. Customize `lustre_client_inventory.yml`, `host_vars`, and `group_vars` for your clients. Line references in the table point to the RoCE client template files in the `ansible-lustre release/1.0.0` tag; the InfiniBand client template uses the same variables except where noted.

Field	Ansible variable	Template file (`release/1.0.0`)	Scope	Notes
Client hosts	`lustre_clients`	lustre_client_inventory.yml#L4	Per client	List each client host name.
Client SSH user and become password	`ssh_client_user` / `ssh_client_become_pass`	passwords.yml#L2	Cluster-wide	Set the become password only if the SSH user is not `root`.
LNet interfaces	`eseries_lustre_lnet`	group_vars/all.yml#L7	Cluster-wide	Client-side LNet interface names (values at L13).
Mount MGS NIDs	`lustre_client_mounts.mgsnode`	group_vars/all.yml#L17	Cluster-wide	MGS NIDs used to mount the file system (values at L20).
File system name	`lustre_client_mounts.fsname`	group_vars/all.yml#L21	Cluster-wide	Must match the cluster `fsname`.

Field	Ansible variable	Template file (`release/1.0.0`)	Scope	Notes
Mount point	`lustre_client_mounts.mount_point`	group_vars/all.yml#L22	Cluster-wide	Client mount directory.
Client management IP	`ansible_host`	host_vars/client_01.yml#L4	Repeat per client	One `host_vars/client_NN.yml` file per client.
Storage fabric interfaces	`eseries_roce_interfaces`	host_vars/client_01.yml#L10	Repeat per client	Frontend interface addresses (values at L15). The InfiniBand template uses `eseries_ipoib_interfaces` here.

Repeat the settings for each client. Create one `host_vars/client_NN.yml` file for each client and add the corresponding host to `lustre_client_inventory.yml`. Populate the shared `clients/passwords.yml` file and encrypt it with Ansible Vault before running the client playbook.

4. Run the client playbook from the client template directory:

```
ansible-playbook -i lustre_client_inventory.yml
lustre_client_playbook.yml
```

The client playbook configures the client network interfaces and LNet, and can create a persistent systemd mount unit for the Lustre file system.

After you finish

Verify the deployment.

Verify and Expand the Lustre Deployment

Confirm cluster health, Lustre mounts, and file system capacity before production use, then use the same inventory to add building blocks when needed.

Verify the deployment

Steps

1. From a Lustre server, confirm that all Pacemaker resources are started on the expected nodes:

```
pcs status
```

2. From a Lustre client, confirm that the Lustre file system is mounted:

```
mount -t lustre
```

3. From a Lustre client, confirm file system capacity and that MDT and OST targets are visible:

```
lfs df -h
```

For performance validation with IOR, mdtest, and fio, see Sizing guidance.

For controlled target migration or HA failover testing, follow the procedures in the HA administration user guide.

Expand the file system

To add capacity or metadata performance, extend your existing inventory with an additional building block. Do not create a separate inventory for the new building block. The playbook automatically assigns unique MDT and OST indices and registers the new targets against the existing MGS.

Steps

1. Add the new building block hosts, arrays, and resource groups (MDT+OST or OST-only) to the same inventory and `host_vars` and `group_vars` files you used for the original deployment.
2. Rerun the main deployment playbook against the updated inventory.

See Solution architecture for scaling rules and Sizing guidance for guidance on when to add each building block type.

Lustre with NetApp E-Series Storage - Related resources

Use these links for related documentation, tools, and software as you plan or expand a Lustre with NetApp E-Series Storage deployment. For sizing and deployment steps, see Sizing guidance and Deploy the solution.

NetApp resources

- NetApp Interoperability Matrix Tool. Search for **E-Series Lustre**.
- NetApp EF-Series all-flash array datasheet
- NetApp SANtricity System Manager documentation
- NetApp ansible-lustre collection. For inventory field guidance, see Customize the Lustre Ansible inventory.
- NetApp Lustre source repository. NetApp provides prebuilt Lustre server RPMs for Rocky Linux 9.8 and Red Hat Enterprise Linux 9.8. Build Lustre client packages for the client operating system and kernel from the NetApp source.

Lustre community

- Lustre file system
- Lustre Support Matrix for client operating systems and kernels
- Lustre Wiki: Lustre Tuning

Related NetApp AI infrastructure solutions

- BeeGFS on NetApp with E-Series Storage
- Deploying IBM Spectrum Scale with NetApp E-Series Storage
- NVIDIA DGX SuperPOD with NetApp EF-Series

Copyright information

Trademark information