Einsatz verifizierter Architekturen

BeeGFS on NetApp with E-Series Storage

NetApp
June 09, 2025

Inhalt

Einsatz verifizierter Architekturen

Überblick und Anforderungen

Lösungsüberblick

Die BeeGFS auf NetApp Lösung kombiniert das parallele BeeGFS Filesystem mit NetApp EF600 Storage-Systemen und bietet so eine zuverlässige, skalierbare und kostengünstige Infrastruktur, die mit anspruchsvollen Workloads Schritt hält.

NVA-Programm

Die BeeGFS on NetApp Lösung ist Teil des NetApp Verified Architecture (NVA) Programms, das Kunden Referenzkonfigurationen und Orientierungshilfen zur Größenbestimmung für spezifische Workloads und Anwendungsfälle bietet. NVA-Lösungen werden ausführlich getestet und entwickelt, um Implementierungsrisiken zu minimieren und die Markteinführungszeit zu verkürzen.

Design-Übersicht

BeeGFS auf NetApp wurde als skalierbare Bausteinarchitektur konzipiert, die für eine Vielzahl anspruchsvoller Workloads konfigurierbar ist. Das Filesystem kann an diese Anforderungen angepasst werden – unabhängig davon, ob es um zahlreiche kleine Dateien, das Management umfangreicher Dateivorgänge oder um einen Hybrid-Workload geht. Hohe Verfügbarkeit ist in das Design mit der Verwendung einer zweistufigen Hardware-Struktur integriert, die ein unabhängiges Failover auf mehreren Hardware-Schichten ermöglicht und eine konsistente Performance auch bei teilweisen Systemabfällen gewährleistet. Das BeeGFS-Filesystem ermöglicht eine hochperformante und skalierbare Umgebung für verschiedene Linux-Distributionen. Es stellt Clients einen einzelnen, einfach zugänglichen Storage-Namespace zur Verfügung. Erfahren Sie mehr in der "Architekturübersicht".

Anwendungsfälle

Die folgenden Anwendungsfälle gelten für die BeeGFS auf NetApp Lösung:

- NVIDIA DGX SuperPOD-Systeme mit DGX's mit A100, H100, H200 und B200 GPUs
- Künstliche Intelligenz (KI), einschließlich Machine Learning (ML), Deep Learning (DL), großzügiger natürlicher Sprachverarbeitung (NLP) und NLU (Natural Language Understanding) Weitere Informationen finden Sie unter "BeeGFS for AI: Fakt versus Fiction".
- High-Performance Computing (HPC) einschließlich Applikationen, die mit MPI (Message Passing Interface) und anderen Distributed Computing-Techniken beschleunigt werden. Weitere Informationen finden Sie unter "Warum BeeGFS das HPC übertrifft".
- Applikations-Workloads zeichnen sich durch folgende Merkmale aus:
  - Lesen oder Schreiben auf Dateien mit einer Größe von mehr als 1 GB
  - Lesen oder Schreiben in dieselbe Datei durch mehrere Clients (10s, 100s und 1000s)
- Datensätze mit mehreren Terabyte oder mehreren Petabyte.
- Umgebungen, für die ein einziger Storage Namespace benötigt wird, der sich für eine Mischung aus großen und kleinen Dateien optimieren lässt

Vorteile

Die wichtigsten Vorteile von BeeGFS auf NetApp:

- Verfügbarkeit verifizierter Hardware-Designs mit vollständiger Integration von Hardware- und Softwarekomponenten, die zuverlässige Performance und Zuverlässigkeit gewährleisten.
- Implementierung und Management mit Ansible für Einfachheit und Konsistenz nach Maß
- Überwachung und Beobachtbarkeit mithilfe des E-Series Performance Analyzer und BeeGFS Plug-ins. Weitere Informationen finden Sie unter "Framework zur Überwachung von NetApp E-Series Lösungen".
- Hochverfügbarkeit dank einer Shared-Disk-Architektur für Datenaufbewahrungszeit und -Verfügbarkeit
- Unterstützung für modernes Workload-Management und moderne Orchestrierung mithilfe von Containern und Kubernetes Weitere Informationen finden Sie unter "Kubernetes Meet BeeGFS: Eine Geschichte zukunftssichere Investition".

Generationen im Design

Die BeeGFS on NetApp-Lösung befindet sich derzeit in der zweiten Generation.

Sowohl die erste als auch die zweite Generation verfügen über eine Basisarchitektur mit einem BeeGFS Filesystem und einem NVMe EF600 Storage-System. Die zweite Generation baut jedoch auf der ersten auf und beinhaltet folgende zusätzliche Vorteile:

- Verdoppeln Sie die Performance und Kapazität bei gleichzeitiger Ergänzung von nur 2 HE Rack-Stellfläche
- Hochverfügbarkeit (HA) auf Grundlage eines gemeinsam genutzten zwei-Tier-Hardwaredesigns
- Architektur, die für NVIDIA DGX SuperPOD A100, H100, H200 und B200 Systeme entwickelt wurde und zuvor auf einem dedizierten Abnahme-Cluster bei NVIDIA validiert wurde Lesen Sie mehr über NVIDIA DGX SuperPOD mit NetApp in der "Designleitfaden".

Design der zweiten Generation

Die zweite Generation von BeeGFS auf NetApp ist für die Performance-Anforderungen anspruchsvoller Workloads optimiert, darunter High-Performance-Computing (HPC), Machine Learning (ML), Deep Learning (DL) und andere Techniken für künstliche Intelligenz (KI). Durch die Integration einer Shared-Disk-Hochverfügbarkeits-Architektur (HA) sorgt dieses Design für die Langlebigkeit und Verfügbarkeit von Daten und eignet sich daher ideal für Unternehmen und andere Unternehmen, die Ausfallzeiten oder Datenverluste nicht leisten können. Das Design der zweiten Generation umfasst Komponenten wie PCIe gen5-Server und Unterstützung für NVIDIA® Quantum™ QM9700 400 GB/s InfiniBand-Switches. Diese Lösung wurde nicht nur von NetApp verifiziert, sondern hat auch die externe Qualifizierung als Storage-Option für das DGX™ A100 SuperPOD NVIDIA bestanden – mit erweiterter Zertifizierung für DGX SuperPOD H100-, H200- und B200-Systeme.

Design der ersten Generationen

Die erste Generation von BeeGFS auf NetApp wurde für Workloads für Machine Learning (ML) und künstliche Intelligenz (KI) mit NetApp EF600 NVMe-Storage-Systemen, dem parallelen Filesystem BeeGFS, NVIDIA DGX™ A100-Systemen und NVIDIA® Mellanox® Quantum™ QM8700 200 GB/s IB-Switches entwickelt. Dieses Design bietet zudem 200 GB/s InfiniBand (IB) für die Storage- und Computing-Cluster Interconnect Fabric und stellt so eine vollständig IB-basierte Architektur für hochperformante Workloads bereit.

Weitere Informationen zur ersten Generation finden Sie unter "NetApp EF-Series AI mit NVIDIA DGX A100 Systems und BeeGFS".

Überblick über die Architektur

Die BeeGFS on NetApp Lösung beinhaltet Design-Aspekte, die bei der Architekturentwicklung berücksichtigt werden, um die spezifischen Geräte, Kabel und Konfigurationen zu ermitteln, die für validierte Workloads erforderlich sind.

Modulare Architektur

Das BeeGFS-Dateisystem kann je nach Storage-Anforderungen unterschiedlich implementiert und skaliert werden. In Anwendungsfällen, die in erster Linie mehrere kleine Dateien enthalten, profitieren beispielsweise von der zusätzlichen Performance und Kapazität der Metadaten, während in Anwendungsfällen mit weniger großen Dateien mehr Storage-Kapazität und Performance für die tatsächlichen Dateiinhalte erforderlich wären. Diese verschiedenen Überlegungen wirken sich auf die verschiedenen Dimensionen der Implementierung paralleler Dateisysteme aus, was die Entwicklung und Implementierung des Filesystems weiter vereinfacht.

Zur Bewältigung dieser Herausforderungen hat NetApp eine standardmäßige Bausteinarchitektur entwickelt, mit der sich jede dieser Dimensionen skalieren lässt. BeeGFS-Bausteine werden in der Regel in einem von drei Konfigurationsprofilen bereitgestellt:

- Ein einzelner Baustein, einschließlich BeeGFS-Management, Metadaten und Storage-Services
- Ein BeeGFS Metadaten plus Storage-Baustein
- Ein BeeGFS-Lagergebäude

Die einzige Hardware-Änderung zwischen diesen drei Optionen ist die Verwendung kleinerer Laufwerke für BeeGFS-Metadaten. Andernfalls werden alle Konfigurationsänderungen durch die Software übernommen. Und mit Ansible als Implementierungs-Engine gestaltet sich die Einrichtung des gewünschten Profils für einen bestimmten Baustein die Konfigurationsaufgaben unkompliziert.

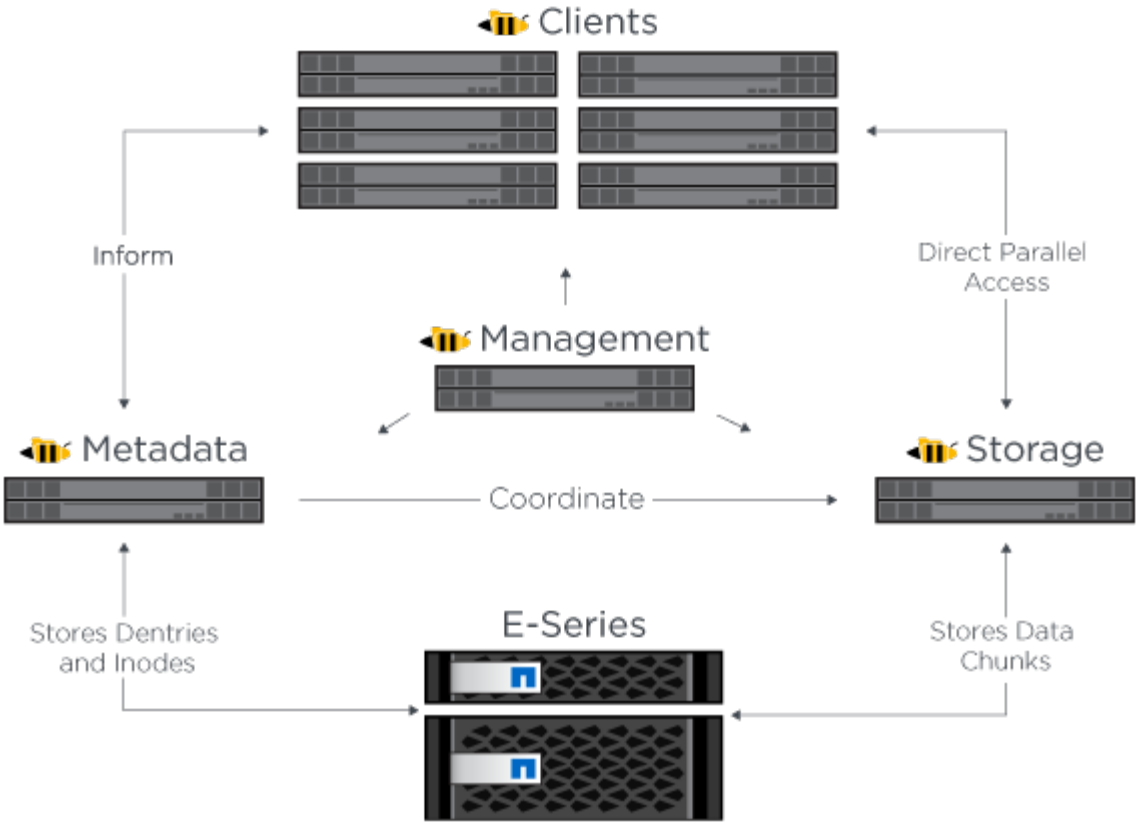
Weitere Informationen finden Sie unter Verifiziertes Hardwaredesign.

File-System-Services

Das BeeGFS-Dateisystem umfasst die folgenden Hauptdienste:

- **Management Service.** registriert und überwacht alle anderen Dienste.
- **Speicherdienst.** speichert den verteilten Inhalt der Benutzerdatei, bekannt als Datenblock-Dateien.
- **Metadatendienst.** verfolgt das Dateisystem-Layout, Verzeichnis, Dateiattribute und so weiter.
- **Client Service.** installiert das Dateisystem, um auf die gespeicherten Daten zuzugreifen.

Die folgende Abbildung zeigt die Komponenten und Beziehungen der BeeGFS-Lösung für NetApp E-Series Systeme.



Als paralleles Dateisystem verteilt BeeGFS seine Dateien auf mehrere Server-Nodes, um die Lese-/Schreib-Performance und Skalierbarkeit zu maximieren. Die Server-Knoten arbeiten zusammen, um ein einziges Dateisystem bereitzustellen, das gleichzeitig von anderen Server-Knoten, allgemein bekannt als *Clients*, gemountet werden kann. Diese Clients können das verteilte Dateisystem auf ähnliche Weise wie ein lokales Dateisystem wie NTFS, XFS oder ext4 sehen und nutzen.

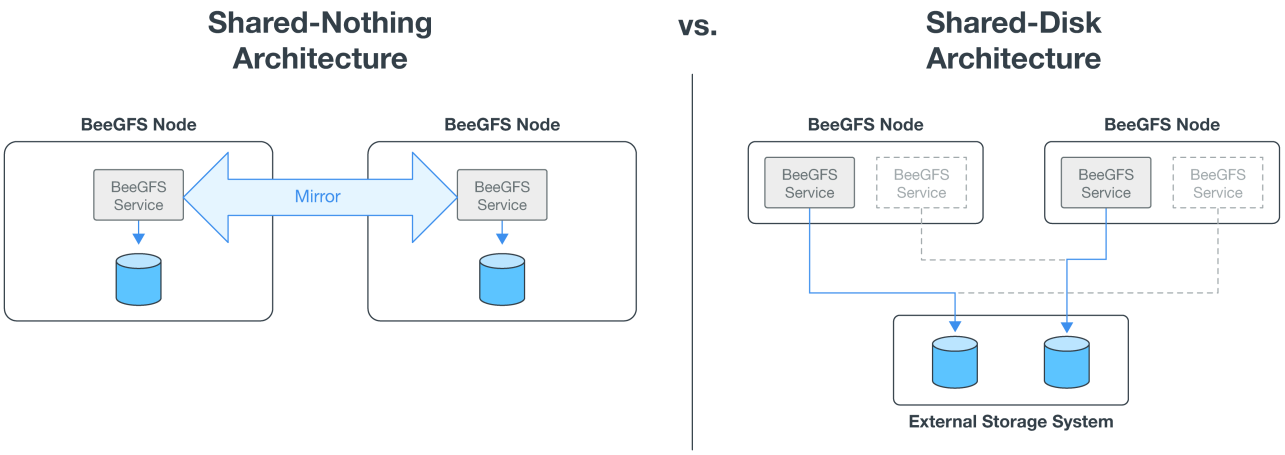
Die vier wichtigsten Services werden in einer Vielzahl von unterstützten Linux Distributionen ausgeführt und kommunizieren über jedes TCP/IP- oder RDMA-fähige Netzwerk, einschließlich InfiniBand (IB), Omni-Path (OPA) und RDMA over Converged Ethernet (RoCE). Die BeeGFS Server Services (Management, Speicherung und Metadaten) sind Benutzerspace-Dämonen, während der Client ein natives Kernel-Modul (patchless) ist. Alle Komponenten können ohne Neustart installiert oder aktualisiert werden. Sie können beliebige Kombinationen von Services auf demselben Node ausführen.

HA-Architektur

BeeGFS auf NetApp erweitert die Funktionalität der BeeGFS Enterprise Edition durch Entwicklung einer vollständig integrierten Lösung mit NetApp Hardware, die eine HA-Architektur (Shared Disk High Availability, Shared-Hochverfügbarkeit) ermöglicht.

> Die BeeGFS Community Edition kann zwar kostenlos genutzt werden, jedoch muss bei der Enterprise Edition ein Professional Support-Abonnementvertrag von einem Partner wie NetApp abgeschlossen werden. Die Enterprise-Version ermöglicht die Nutzung mehrerer zusätzlicher Funktionen wie Ausfallsicherheit, Kontingentzuverfolgung und Storage-Pools.

In der folgenden Abbildung werden die HA-Architekturen ohne Shared-Festplatten verglichen.



Weitere Informationen finden Sie unter "Ankündigung der Hochverfügbarkeit für BeeGFS mit Unterstützung von NetApp".

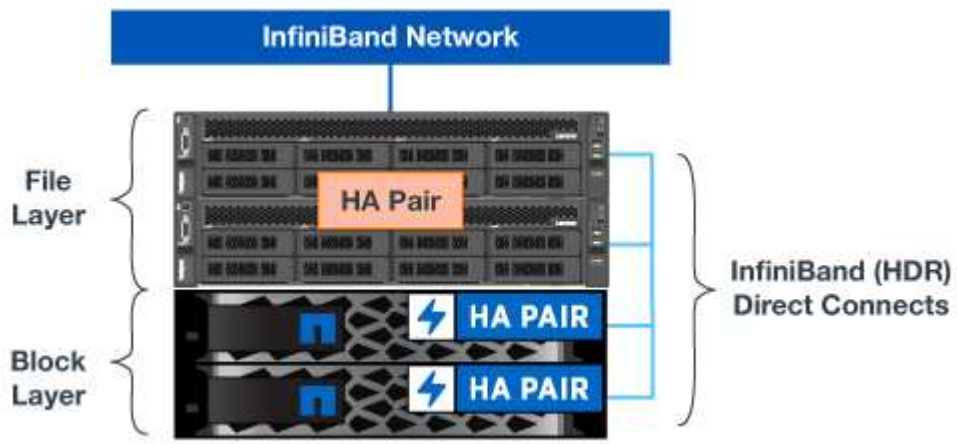
Verifizierte Nodes

Die BeeGFS auf NetApp-Lösung hat die unten aufgeführten Knoten verifiziert.

Knoten	Trennt	Details
Block-Storage	NetApp EF600 Storage-System	Dieses rein NVMe-basierte 2-HE-Storage-Array mit hoher Performance ist für anspruchsvolle Workloads konzipiert.
Datei	Lenovo ThinkSystem SR665 V3-Server	2-Socket-Server mit PCIe 5.0, zwei AMD EPYC 9124 Prozessoren. Weitere Informationen zum Lenovo SR665 V3 finden Sie unter "Lenovo Website".
	Lenovo ThinkSystem SR665 Server	2-Socket-Server mit PCIe 4.0, zwei AMD EPYC 7003 Prozessoren. Weitere Informationen zum Lenovo SR665 finden Sie unter "Lenovo Website".

Verifiziertes Hardwaredesign

Die Bausteine der Lösung (in der folgenden Abbildung dargestellt) verwenden die verifizierten File-Node-Server für die BeeGFS-Dateiebene und zwei EF600-Storage-Systeme als Block-Ebene.

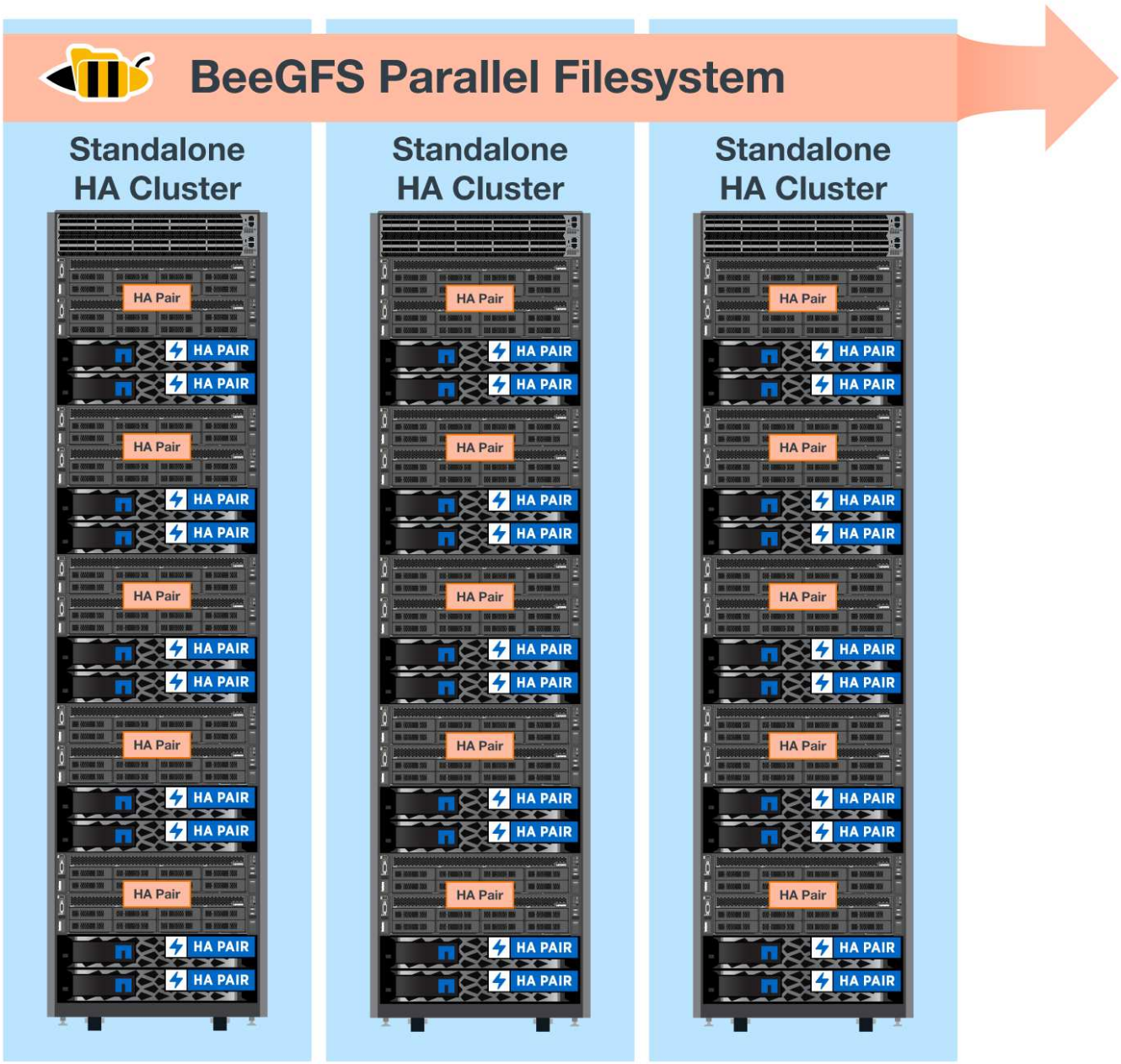


Die BeeGFS on NetApp Lösung läuft über alle Bausteine während der Implementierung hinweg. Auf dem ersten implementierten Baustein müssen BeeGFS-Management-, Metadaten- und Storage-Services (als

Basisbaustein bezeichnet) ausgeführt werden. Alle nachfolgenden Bausteine können über Software konfiguriert werden, um Metadaten und Storage-Services zu erweitern oder ausschließlich Storage-Services bereitzustellen. Mit diesem modularen Ansatz kann das Filesystem an die Anforderungen eines Workloads skaliert werden, während gleichzeitig dieselben zugrunde liegenden Hardware-Plattformen und dasselbe Bausteindesign verwendet werden.

Bis zu fünf Bausteine können als Standalone Linux HA-Cluster implementiert werden. Dies optimiert die Ressourcenverwaltung mit Pacemaker und sorgt für eine effiziente Synchronisierung mit Corosync. Mindestens ein dieser Standalone BeeGFS HA-Cluster wird kombiniert, um ein BeeGFS-Filesystem zu erstellen, das für Clients als einzelner Storage-Namespace zur Verfügung steht. Auf der Hardware-Seite kann ein einzelnes 42-HE-Rack bis zu fünf Bausteine zusammen mit zwei 1-HE-InfiniBand-Switches für das Storage-/Datennetzwerk aufnehmen. Eine visuelle Darstellung finden Sie in der folgenden Grafik.

Zum Herstellen von Quorum im Failover Cluster sind mindestens zwei Bausteine erforderlich. Ein Cluster mit zwei Nodes hat Einschränkungen, die ein erfolgreiches Failover verhindern können. Wenn Sie ein Cluster mit zwei Nodes konfigurieren, wird ein drittes Gerät als Tiebreaker integriert, dieses Design wird jedoch nicht in dieser Dokumentation beschrieben.



Ansible

BeeGFS auf NetApp wird mittels Ansible-Automatisierung bereitgestellt und implementiert. Das Hosting wird auf GitHub und Ansible Galaxy (die BeeGFS-Sammlung ist über verfügbar "Ansible-Galaxie" Und "NetApp E-Series GitHub"). Obwohl Ansible vor allem mit der Hardware getestet wird, die zum Zusammenbauen der BeeGFS-Bausteine verwendet wird, können Sie es so konfigurieren, dass es auf nahezu jedem x86-basierten Server unter Verwendung einer unterstützten Linux-Distribution ausgeführt wird.

Weitere Informationen finden Sie unter "Implementieren von BeeGFS mit E-Series Storage".

Technische Anforderungen

Stellen Sie zur Implementierung der Lösung BeeGFS auf NetApp sicher, dass Ihre Umgebung die in diesem Dokument beschriebenen Technologieanforderungen erfüllt.

Hardwareanforderungen

Stellen Sie zunächst sicher, dass Ihre Hardware die folgenden Spezifikationen für ein einziges Bausteindesign der zweiten Generation der BeeGFS auf NetApp-Lösung erfüllt. Die genauen Komponenten für eine bestimmte Implementierung können je nach den Anforderungen des Kunden variieren.

Menge	Hardwarekomponenten	Anforderungen
2	BeeGFS-Datei-Nodes	Jeder Datei-Node sollte die Spezifikationen der empfohlenen Datei-Nodes erfüllen oder übertreffen, um die erwartete Performance zu erreichen. **Empfohlene Dateiknoten-Optionen:** • * Lenovo ThinkSystem SR665 V3* ◦ **Prozessoren:** 2x AMD EPYC 9124 16C 3.0 GHz (konfiguriert als zwei NUMA Zonen). ◦ **Speicher:** 256 GB (16 x 16 GB TruDDR5 4800 MHz RDIMM-A) ◦ **PCIe-Erweiterung:** vier PCIe Gen5 x16-Steckplätze (zwei pro NUMA-Zone) ◦ **Verschiedenes:** ▪ Zwei Laufwerke in RAID 1 für OS (1 TB 7.200 SATA oder höher) ▪ 1-GbE-Port für in-Band-OS-Management ▪ 1GbE BMC mit Redfish API für Out-of-Band-Server-Management ▪ Zwei Hot-Swap-Netzteile und Lüfter mit hoher Leistung
2	E-Series-Block-Nodes (EF600 Array)	**Speicher:** 256 GB (128 GB pro Controller). **Adapter:** 2-Port 200 GB/HDR (NVMe/IB). **Laufwerke:** entsprechend den gewünschten Metadaten und Speicherkapazität konfiguriert.
8	InfiniBand-Host-Karten-Adapter (für Datei-Nodes)	Hostkartenadapter können je nach Servermodell des Dateiknotens variieren. Zu den Empfehlungen für verifizierte Datei-Nodes gehören: • * Lenovo ThinkSystem SR665 V3 Server:* ◦ MCX755106AS-HEAT ConnectX-7, NDR200, QSFP112, 2 Ports, PCIe Gen5 x16, InfiniBand-Adapter
1	Storage-Netzwerk-Switch	Der Storage-Netzwerk-Switch muss 200 GB/s InfiniBand-Geschwindigkeiten unterstützen. Empfohlene Switch-Modelle: • **NVIDIA QM9700 Quantum 2 NDR InfiniBand Switch** • **NVIDIA MQM8700 Quantum HDR InfiniBand Switch**

Verkabelungsanforderungen

Direkte Verbindungen von Block-Knoten zu File-Knoten.

Menge	Teilenummer	Länge
8	MCP1650-H001E30 (passives NVIDIA-Kupferkabel, QSFP56, 200 GB/s)	1 m

Verbindungen von Dateiknoten zum Speichernetzwerk-Switch. Wählen Sie je nach InfiniBand-Speicherschalter die entsprechende Kabeloption aus der folgenden Tabelle aus. + die empfohlene Kabellänge beträgt 2 m, dies kann jedoch je nach Umgebung des Kunden variieren.

Switch-Modell	Kabeltyp	Menge	Teilenummer
NVIDIA QM9700	Aktive Glasfaser (einschließlich Transceiver)	2	MMA4Z00-NS (Multimode, IB/ETH, 800 GB/s 2x400 GB/s Twin-Port OSFP)
		4	MFP7E20-Nxxx (Multimode, 4-Kanal-zu-zwei 2-Kanal-Splitter-Glasfaserkabel)
		8	MMA1Z00-NS400 (Multimode, IB/ETH, 400 GB/s Single-Port QSFP-112)
	Passives Kupfer	2	MCP7Y40-N002 (passives NVIDIA-Kupferverteilerkabel, InfiniBand 800 GB/s bis 4 x 200 GB/s, OSFP auf 4 x QSFP112)
NVIDIA MQM8700	Aktive Glasfaser	8	MFS1S00-H003E (aktives NVIDIA-Glasfaserkabel, InfiniBand 200 GB/s, QSFP56)
	Passives Kupfer	8	MCP1650-H002E26 (passives NVIDIA-Kupferkabel, InfiniBand 200 GB/s, QSFP56)

Software- und Firmware-Anforderungen zu erfüllen

Um eine vorhersehbare Performance und Zuverlässigkeit zu gewährleisten, werden Versionen der BeeGFS auf NetApp Lösung mit bestimmten Versionen der Software- und Firmware-Komponenten getestet. Diese Versionen sind für die Implementierung der Lösung erforderlich.

Anforderungen an Datei-Nodes

Software	Version
Red hat Enterprise Linux (RHEL)	RHEL 9.4 Server physisch mit hoher Verfügbarkeit (2 Sockel). **Hinweis:** Für Dateiknoten sind ein gültiges Red Hat Enterprise Linux Server-Abonnement und das Red Hat Enterprise Linux High Availability Add-On erforderlich.
Linux-Kernel	5.14.0-427.42.1.el9_4.x86_64
HCA-Firmware	**ConnectX-7 HCA-Firmware** FW: 28.43.1014 + PXE: 3.7.0500 + UEFI: 14.36.0016 **ConnectX-6 HCA-Firmware** FW: 20.43.2566 + PXE: 3.7.0500 + UEFI: 14.37.0013

Anforderungen der EF600 Block-Nodes

Software	Version
SANtricity OS	11.90R1
NVSRAM	N6000-890834-D02.dlp

Software	Version
Festplatten-Firmware	Neueste verfügbar für die verwendeten Antriebsmodelle. Siehe "E-Series Festplatten-Firmware-Website".

Anforderungen an die Softwareimplementierung

In der folgenden Tabelle sind die automatisch bereitgestellten Softwareanforderungen im Rahmen der Ansible-basierten BeeGFS-Implementierung aufgeführt.

Software	Version
BeeGFS	7.4.6
Corosync	3.1.8-1
Schrittmacher	2.1.7-5,2
PCS	0.11.7-2
Zaunmittel (Rotbarsch/apc)	4.10.0-62
InfiniBand-/RDMA-Treiber	MLNX_OFED_LINUX-23.10-3.2.2.1-LTS

Ansible-Control-Node-Anforderungen

Die BeeGFS auf NetApp Lösung wird über einen Ansible-Kontroll-Node implementiert und gemanagt. Weitere Informationen finden Sie im "Ansible-Dokumentation".

Die in den folgenden Tabellen aufgeführten Software-Anforderungen beziehen sich speziell auf die unten aufgeführte Version der NetApp BeeGFS Ansible Sammlung.

Software	Version
Ansible	10.x
Ansible-Core	>= 2.13.0
Python	3,10
Zusätzliche Python-Pakete	Kryptographie-43.0.0, netaddr-1.3.0, ipaddr-2.2.0
NetApp E-Series BeeGFS Ansible Sammlung	3.2.0

Überprüfen des Lösungsdesigns

Designübersicht

Spezifische Geräte, Kabel und Konfigurationen sind erforderlich, um die BeeGFS auf NetApp Lösung zu unterstützen, die das parallele Filesystem BeeGFS mit den NetApp EF600 Storage-Systemen kombiniert.

Weitere Informationen:

- "Hardwarekonfiguration"
- "Softwarekonfiguration"
- "Design-Überprüfung"
- "Richtlinien für die Dimensionierung"
- "Performance-Optimierung"

Derivative Architekturen mit Variationen in Design und Performance:

- "Baustein Mit Hoher Kapazität"

Hardwarekonfiguration

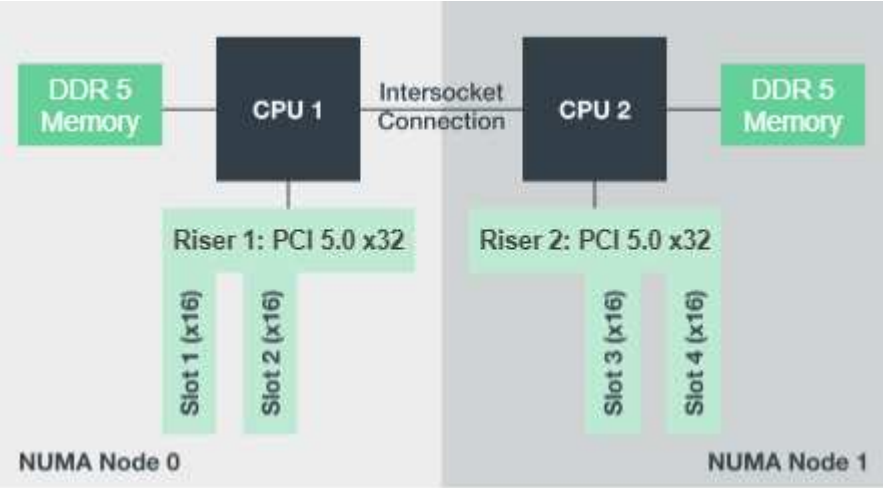
Die Hardware-Konfiguration für BeeGFS auf NetApp umfasst Datei-Nodes und Netzwerkverkabelung.

Konfiguration der Datei-Nodes

File-Nodes haben zwei CPU-Sockets als separate NUMA-Zonen konfiguriert, die lokalen Zugriff auf eine gleiche Anzahl von PCIe-Steckplätzen und Arbeitsspeicher enthalten.

InfiniBand-Adapter müssen in die entsprechenden PCI-Risers oder Steckplätze gefüllt sein, damit die Workload über die verfügbaren PCIe-Lanes und Speicherkanäle ausgeglichen ist. Sie balancieren den Workload aus, indem einzelne BeeGFS-Services vollständig auf einen bestimmten NUMA-Node isoliert werden. Das Ziel besteht darin, bei jedem Datei-Node eine ähnliche Performance zu erreichen, als ob es sich um zwei unabhängige Single-Socket-Server handelte.

Die folgende Abbildung zeigt die NUMA-Konfiguration des Dateiknotens.



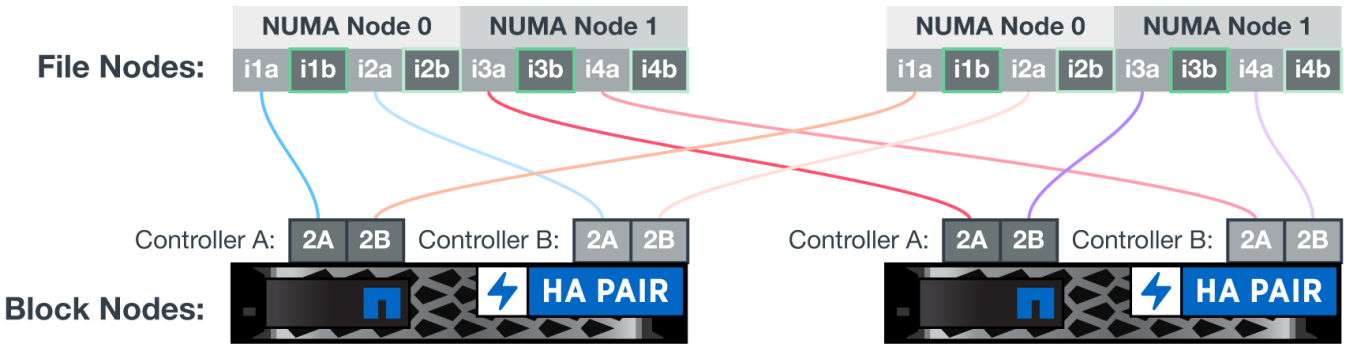
Die BeeGFS-Prozesse sind an eine bestimmte NUMA-Zone gebunden, um sicherzustellen, dass sich die verwendeten Schnittstellen in der gleichen Zone befinden. Diese Konfiguration vermeidet den Remote-Zugriff über die Verbindung zwischen den Sockets. Die Verbindung zwischen den Sockeln wird manchmal als QPI oder GMI2 Link bezeichnet; selbst in modernen Prozessorarchitekturen können sie einen Engpass darstellen, wenn High-Speed-Netzwerke wie HDR InfiniBand eingesetzt werden.

Konfiguration der Netzwerkverkabelung

Jeder Datei-Node ist innerhalb eines Bausteins mit zwei Block-Nodes verbunden. Dabei werden insgesamt vier redundante InfiniBand-Verbindungen verwendet. Zusätzlich verfügt jeder Datei-Node über vier redundante Verbindungen zum InfiniBand-Storage-Netzwerk.

Beachten Sie in der folgenden Abbildung Folgendes:

- Alle grün dargestellten Datei-Node-Ports werden zur Verbindung mit der Storage-Fabric verwendet. Alle anderen Datei-Node-Ports sind die direkten Verbindungen zu den Block-Nodes.
- Zwei InfiniBand-Ports in einer bestimmten NUMA-Zone werden mit den A- und B-Controllern desselben Block-Nodes verbunden.
- Ports im NUMA-Knoten 0 stellen immer eine Verbindung zum ersten Block-Knoten her.
- Die Ports im NUMA-Knoten 1 verbinden sich mit dem zweiten Block-Knoten.



> Wenn Sie Splitterkabel verwenden, um den Speicher-Switch mit den Dateiknoten zu verbinden, sollte ein Kabel abzweigen und mit den hellgrünen Ports verbunden werden. Ein anderes Kabel sollte abzweigen und an die dunkelgrünen Ports anschließen. Außerdem sollten bei Speichernetzwerken mit redundanten Switches die hellgrünen Ports mit einem Switch verbunden werden, während dunkelgrüne Ports mit einem anderen Switch verbunden sein sollten.

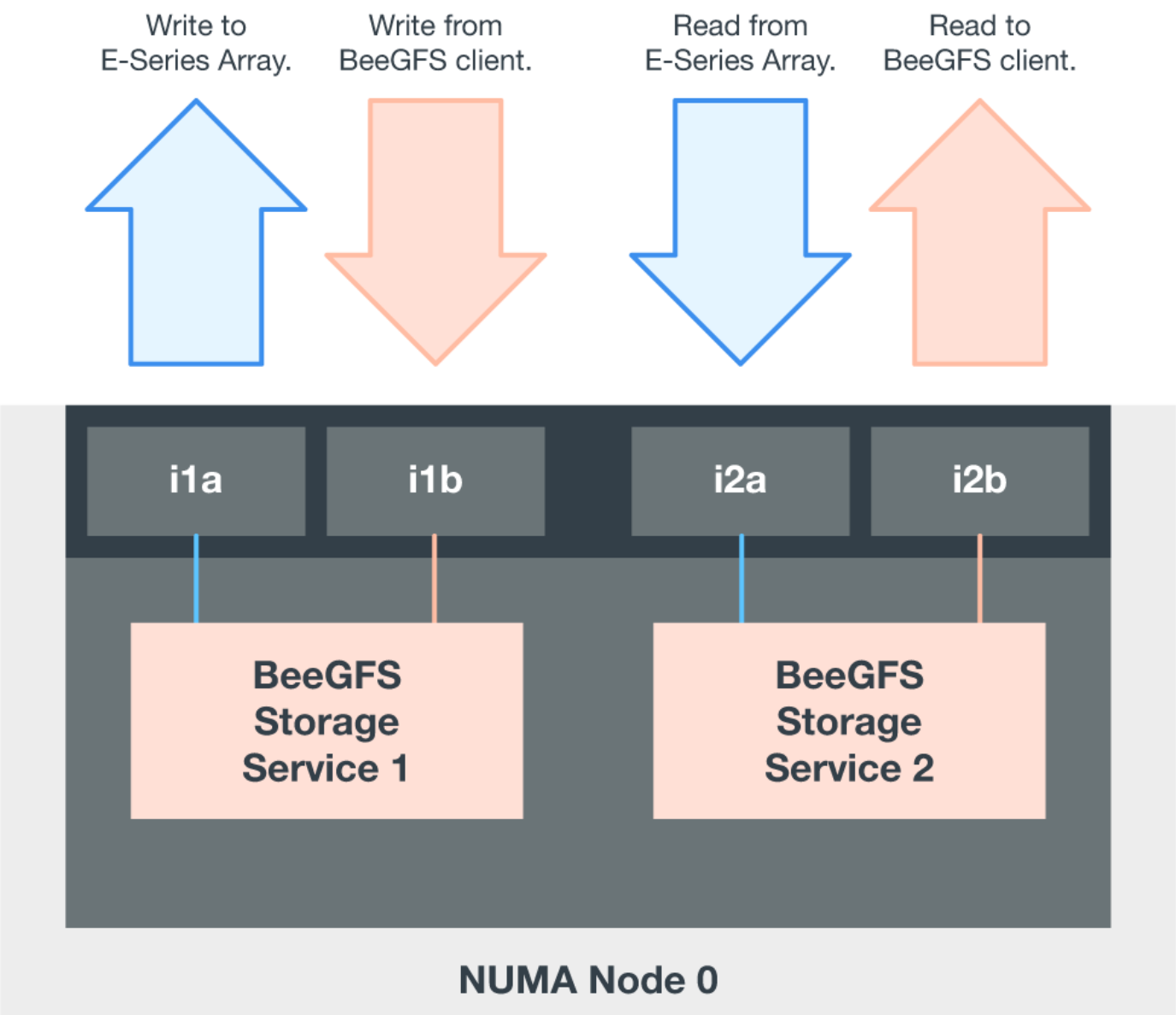
Die in der Abbildung dargestellte Verkabelungskonfiguration ermöglicht jedem BeeGFS-Dienst Folgendes:

- Laufen Sie in derselben NUMA-Zone, unabhängig davon, auf welchem Dateiknoten der BeeGFS-Service ausgeführt wird.
- Sekundäre optimale Pfade zum Front-End-Storage-Netzwerk und zu den Back-End-Block-Nodes sind vorhanden, unabhängig davon, wo ein Ausfall auftritt.
- Minimale Auswirkungen auf die Performance, wenn ein Datei-Node oder Controller in einem Block-Node gewartet werden muss

Verkabelung zur Nutzung der Bandbreite

Um die vollständige bidirektionale PCIe-Bandbreite zu nutzen, stellen Sie sicher, dass ein Port an jedem InfiniBand-Adapter mit der Storage-Fabric verbunden ist, und der andere Port mit einem Block-Node verbunden wird.

Die folgende Abbildung zeigt das Verkabelungsdesign, mit dem die vollständige bidirektionale PCIe-Bandbreite genutzt werden kann.



Verwenden Sie für jeden BeeGFS-Service denselben Adapter, um den bevorzugten Port für Client-Datenverkehr mit dem Pfad zu dem Block-Nodes-Controller zu verbinden, der der primäre Eigentümer dieser Service-Volumes ist. Weitere Informationen finden Sie unter "Softwarekonfiguration".

Softwarekonfiguration

Die Software-Konfiguration für BeeGFS auf NetApp umfasst BeeGFS-Netzwerkkomponenten, EF600 Block-Nodes, BeeGFS-Datei-Nodes, Ressourcengruppen und BeeGFS-Services.

BeeGFS-Netzwerkkonfiguration

Die BeeGFS-Netzwerkkonfiguration besteht aus den folgenden Komponenten.

- **Schwimmende IPs** schwimmende IPs sind eine Art virtueller IP-Adresse, die dynamisch an jeden Server im selben Netzwerk weitergeleitet werden kann. Mehrere Server können dieselbe Floating-IP-Adresse besitzen, sie kann jedoch nur auf einem Server zu einem bestimmten Zeitpunkt aktiv sein.

Jeder BeeGFS-Serverdienst hat eine eigene IP-Adresse, die sich je nach Ausführungsort des BeeGFS-Serverdienstes zwischen Datei-Nodes verschieben kann. Diese fließende IP-Konfiguration ermöglicht jedem Service ein unabhängiges Failover auf den anderen File-Node. Der Client muss einfach die IP-Adresse für einen bestimmten BeeGFS-Service kennen. Es muss nicht wissen, welcher Datei-Node derzeit diesen Service ausführt.

- **BeeGFS-Server Multi-Homing-Konfiguration** um die Dichte der Lösung zu erhöhen, verfügt jeder Dateiknoten über mehrere Speicherschnittstellen mit IPs, die im gleichen IP-Subnetz konfiguriert sind.

  Es ist eine zusätzliche Konfiguration erforderlich, um sicherzustellen, dass diese Konfiguration wie erwartet mit dem Linux-Netzwerk-Stack funktioniert, da standardmäßig Anfragen an eine Schnittstelle auf einer anderen Schnittstelle beantwortet werden können, wenn ihre IPs im selben Subnetz sind. Neben anderen Nachteilen ist es bei diesem Standardverhalten unmöglich, RDMA-Verbindungen ordnungsgemäß einzurichten oder zu pflegen.

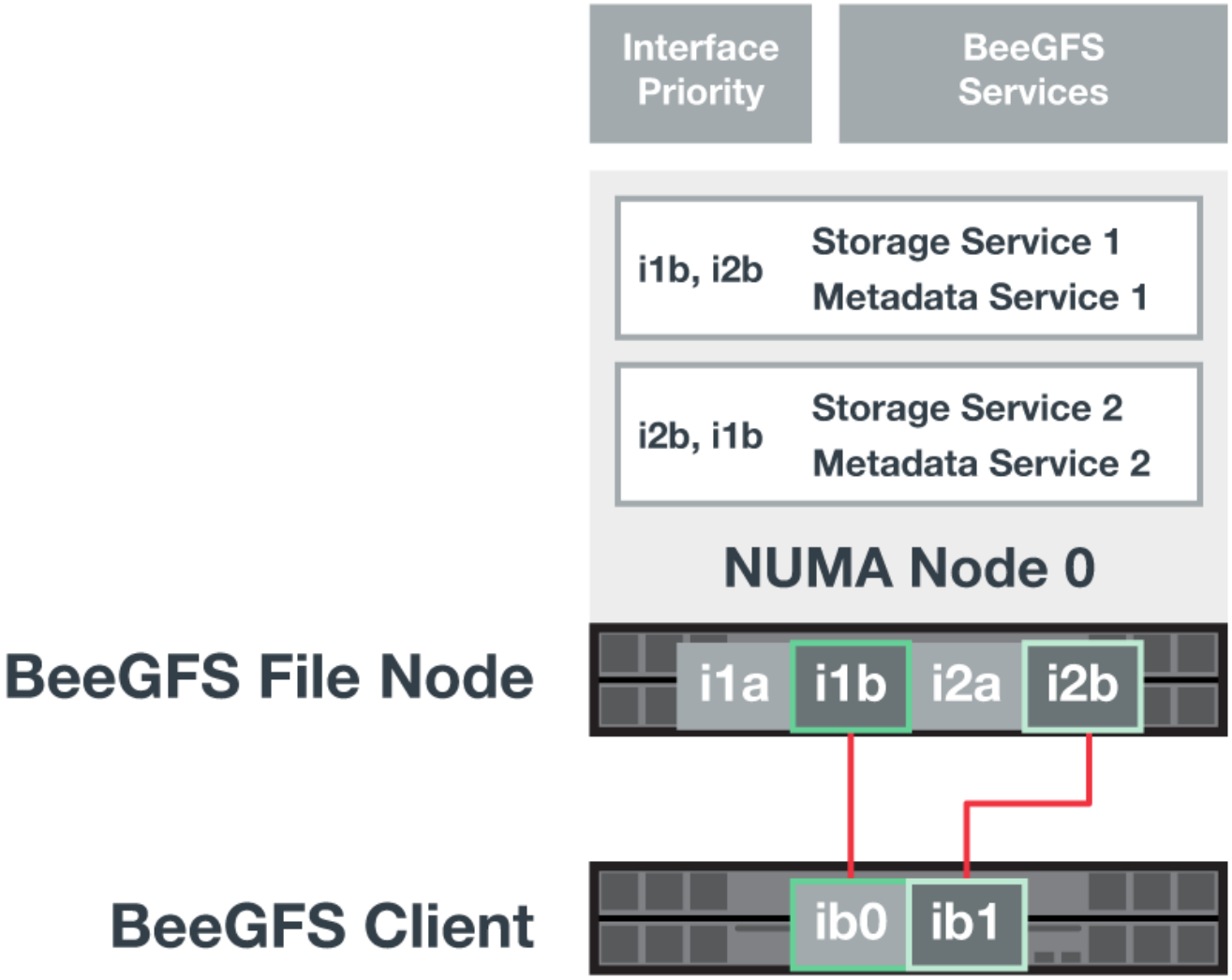
  Die Ansible-basierte Implementierung verarbeitet das Anziehen des Reverse Path (RP) und das ARP-Verhalten (Address Resolution Protocol) und stellt sicher, dass Floating IPs gestartet und gestoppt werden. Die entsprechenden IP-Routen und -Regeln werden dynamisch erstellt, damit die Multihomed-Netzwerkkonfiguration ordnungsgemäß funktioniert.

- **BeeGFS-Client-Multi-Rail-Konfiguration** *Multi-Rail* bezieht sich auf die Fähigkeit einer Anwendung, mehrere unabhängige Netzwerkverbindungen, oder "Schienen", zu verwenden, um die Leistung zu erhöhen.

  BeeGFS implementiert Multi-Rail-Unterstützung, um die Verwendung mehrerer IB-Schnittstellen in einem einzigen IPoIB-Subnetz zu ermöglichen. Diese Funktion ermöglicht Funktionen wie den dynamischen Lastausgleich über RDMA NICs und optimiert damit die Auslastung von Netzwerkressourcen. Die Integration in NVIDIA GPUDirect Storage (GDS) sorgt für eine höhere Systembandbreite sowie geringere Latenz und Auslastung der Client-CPU.

  Diese Dokumentation enthält Anweisungen für einzelne IPoIB-Subnetzkonfigurationen. Duale IPoIB-Subnetzkonfigurationen werden unterstützt, bieten jedoch nicht die gleichen Vorteile wie Konfigurationen mit einem einzigen Subnetz.

  Die folgende Abbildung zeigt die Verteilung des Datenverkehrs auf mehrere BeeGFS-Client-Schnittstellen.



Da jede Datei in BeeGFS normalerweise über mehrere Storage-Services verteilt wird, kann der Client dank der Multi-Rail-Konfiguration einen höheren Durchsatz erzielen als mit einem einzelnen InfiniBand-Port möglich. Die folgende Codeprobe zeigt beispielsweise eine allgemeine File-Striping-Konfiguration, die es dem Client ermöglicht, den Datenverkehr über beide Schnittstellen auszugleichen:

+

```
root@beegfs01:/mnt/beegfs# beegfs-ctl --getentryinfo myfile
Entry type: file
EntryID: 11D-624759A9-65
Metadata node: meta_01_tgt_0101 [ID: 101]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 1M
+ Number of storage targets: desired: 4; actual: 4
+ Storage targets:
  + 101 @ stor_01_tgt_0101 [ID: 101]
  + 102 @ stor_01_tgt_0101 [ID: 101]
  + 201 @ stor_02_tgt_0201 [ID: 201]
  + 202 @ stor_02_tgt_0201 [ID: 201]
```

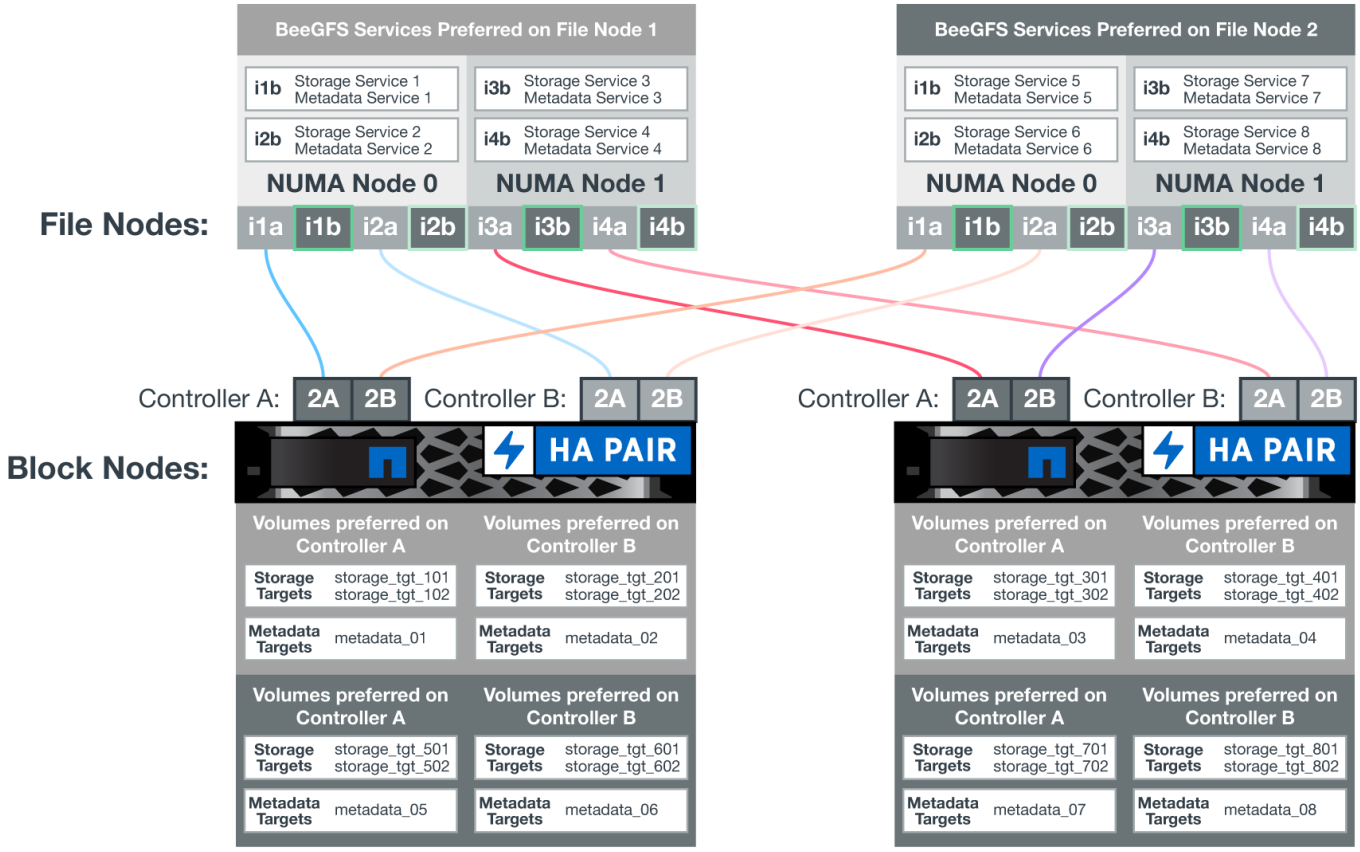
Konfiguration der EF600 Block-Node

Block-Nodes bestehen aus zwei aktiv/aktiv-RAID-Controllern mit gemeinsamem Zugriff auf denselben Satz an Laufwerken. In der Regel besitzt jeder Controller die Hälfte der im System konfigurierten Volumes, jedoch kann für den anderen Controller nach Bedarf die Aufgaben übernehmen.

Multipathing-Software auf den Datei-Knoten bestimmt den aktiven und optimierten Pfad zu jedem Volume und verschiebt sich automatisch nach einem Kabel-, Adapter- oder Controller-Ausfall zum alternativen Pfad.

Das folgende Diagramm zeigt das Controller-Layout in EF600 Block-Nodes.

Um eine Shared-Disk-HA-Lösung zu erleichtern, werden Volumes beiden Datei-Nodes zugeordnet, sodass sie sich bei Bedarf gegenseitig übernehmen können. Das folgende Diagramm zeigt ein Beispiel, wie BeeGFS-Service und die bevorzugte Volumeneigentümer für maximale Performance konfiguriert sind. Die Schnittstelle links von jedem BeeGFS-Dienst gibt die bevorzugte Schnittstelle an, mit der die Clients und andere Dienste Kontakt aufnehmen.



Im vorherigen Beispiel kommunizieren Clients und Server Services lieber mit Storage Service 1 über Schnittstelle i1b. Storage Service 1 verwendet Schnittstelle i1a als bevorzugten Pfad zur Kommunikation mit seinen Volumes (Storage_tgt_101, 102) auf Controller A des ersten Block-Node. Diese Konfiguration nutzt die volle bidirektionale PCIe-Bandbreite, die dem InfiniBand-Adapter zur Verfügung steht, und erreicht mit einem Dual-Port-HDR-InfiniBand-Adapter eine bessere Leistung als bei PCIe 4.0.

BeeGFS-Dateiknoten-Konfiguration

Die BeeGFS Datei-Nodes sind in einem HA-Cluster (High Availability) konfiguriert, um den Failover von BeeGFS-Services zwischen mehreren Datei-Nodes zu ermöglichen.

Das HA Cluster Design basiert auf zwei weit verbreiteten Linux HA Projekten: Corosync für Cluster-Mitgliedschaft und Pacemaker für Cluster Resource Management. Weitere Informationen finden Sie unter "Red hat Training für Add-ons mit hoher Verfügbarkeit".

NetApp hat mehrere Open Cluster Framework (OCF) Resource Agents entwickelt und erweitert, damit das Cluster die BeeGFS Ressourcen intelligent starten und überwachen kann.

BeeGFS HA Cluster

Wenn Sie einen BeeGFS-Dienst starten (mit oder ohne HA), müssen in der Regel einige Ressourcen vorhanden sein:

- IP-Adressen, in denen der Dienst erreichbar ist, werden normalerweise von Network Manager konfiguriert.
- Zugrunde liegende Dateisysteme, die als Ziele für BeeGFS zum Speichern von Daten verwendet werden.

  Diese werden typischerweise in definiert `/etc/fstab` Und montiert von systemd.

- Ein systemd-Service, der für den Start von BeeGFS-Prozessen verantwortlich ist, wenn die anderen Ressourcen bereit sind.

  Ohne zusätzliche Software werden diese Ressourcen nur auf einem einzelnen Datei-Node gestartet. Wenn der Datei-Node offline geschaltet wird, kann auf einen Teil des BeeGFS-Dateisystems zugegriffen werden.

Da mehrere Nodes jeden BeeGFS-Service starten können, muss Pacemaker sicherstellen, dass jeder Service und die abhängigen Ressourcen nur auf jeweils einem Node ausgeführt werden. Wenn beispielsweise zwei Knoten versuchen, denselben BeeGFS-Service zu starten, besteht das Risiko einer Datenbeschädigung, wenn beide versuchen, auf dieselben Dateien auf dem zugrunde liegenden Ziel zu schreiben. Um dieses Szenario zu vermeiden, setzt Pacemaker auf Corosync, um den Zustand des gesamten Clusters zuverlässig über alle Knoten hinweg zu synchronisieren und Quorum zu schaffen.

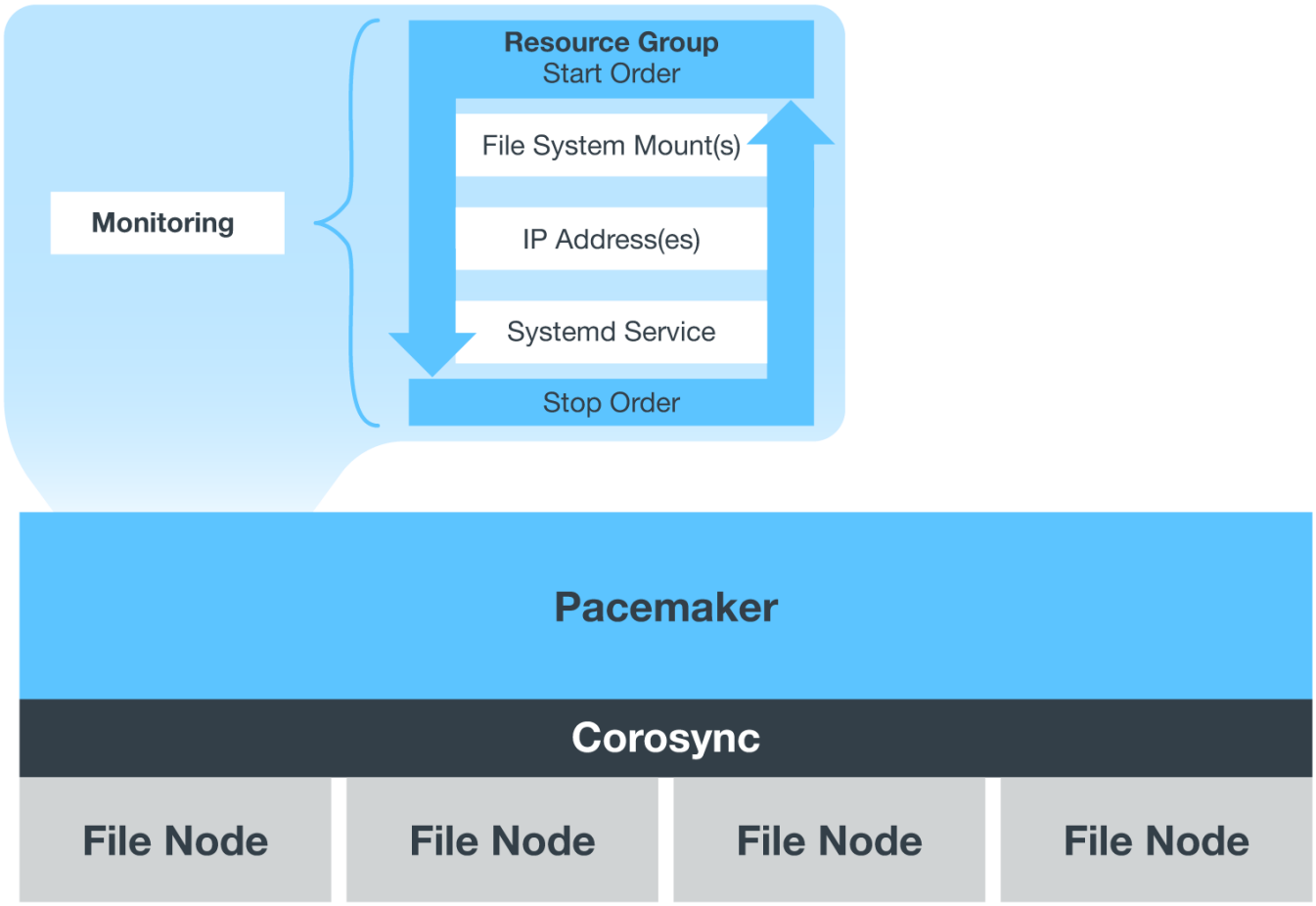
Wenn ein Fehler im Cluster auftritt, reagiert Pacemaker und startet BeeGFS-Ressourcen auf einem anderen Knoten neu. In einigen Fällen kann Pacemaker möglicherweise nicht mit dem ursprünglichen fehlerhaften Knoten kommunizieren, um zu bestätigen, dass die Ressourcen angehalten werden. Um zu überprüfen, ob der Knoten ausgefallen ist, bevor BeeGFS-Ressourcen an anderer Stelle neu gestartet werden, isoliert Pacemaker den fehlerhaften Knoten, idealerweise indem er die Stromversorgung entfernt.

Es stehen zahlreiche Open-Source-Fechten-Agenten zur Verfügung, die es Pacemaker ermöglichen, einen Knoten mit einer Stromverteilungs-Einheit (PDU) oder den Server-Baseboard-Management-Controller (BMC) mit APIs wie Redfish zu Zaun.

Wenn BeeGFS in einem HA-Cluster ausgeführt wird, werden alle BeeGFS-Services und zugrunde liegenden Ressourcen von Pacemaker in Ressourcengruppen gemanagt. Jeder BeeGFS-Service und die Ressourcen, auf die er angewiesen ist, werden in einer Ressourcengruppe konfiguriert, die sicherstellt, dass Ressourcen in der richtigen Reihenfolge gestartet und gestoppt werden und auf demselben Node zusammengelegt werden.

Für jede BeeGFS-Ressourcengruppe führt Pacemaker eine benutzerdefinierte BeeGFS-Überwachungsressource aus, die für die Erkennung von Fehlerbedingungen und die intelligente Auslösung von Failover verantwortlich ist, wenn auf einem bestimmten Knoten kein BeeGFS-Dienst mehr verfügbar ist.

Die folgende Abbildung zeigt die Pacemaker-gesteuerten BeeGFS-Dienste und -Abhängigkeiten.



Damit mehrere BeeGFS-Dienste desselben Typs auf demselben Knoten gestartet werden, wird Pacemaker so konfiguriert, dass BeeGFS-Dienste mit der Multi-Mode-Konfigurationsmethode gestartet werden. Weitere Informationen finden Sie im "BeeGFS-Dokumentation im Multi-Modus".

Da BeeGFS-Dienste auf mehreren Nodes starten können müssen, muss die Konfigurationsdatei für jeden Dienst (normalerweise bei gefunden `/etc/beegfs`) Wird auf einem der E-Series Volumes gespeichert, die als BeeGFS-Ziel für diesen Service verwendet werden. Damit sind die Konfiguration zusammen mit den Daten für einen bestimmten BeeGFS Service für alle Nodes zugänglich, die den Service möglicherweise ausführen müssen.

```
# tree stor_01_tgt_0101/ -L 2
stor_01_tgt_0101/
├── data
│   ├── benchmark
│   ├── buddymir
│   ├── chunks
│   ├── format.conf
│   ├── lock.pid
│   ├── nodeID
│   ├── nodeNumID
│   ├── originalNodeID
│   ├── targetID
│   └── targetNumID
└── storage_config
    ├── beegfs-storage.conf
    ├── connInterfacesFile.conf
    └── connNetFilterFile.conf
```

Design-Überprüfung

Das Design der zweiten Generation für die BeeGFS auf NetApp Lösung wurde mithilfe von drei Bausteinkonfigurationsprofilen verifiziert.

Die Konfigurationsprofile umfassen Folgendes:

- Ein einzelner Baustein, einschließlich BeeGFS-Management, Metadaten und Storage-Services
- Ein BeeGFS Metadaten plus ein Storage-Baustein.
- Ein BeeGFS-Speicherbaustein.

Die Bausteine wurden mit zwei NVIDIA Quantum InfiniBand (MQM8700) Switches verbunden. Zehn BeeGFS Clients wurden auch an die InfiniBand Switches angeschlossen und zur Ausführung von Synthetic Benchmark Utilities verwendet.

Die folgende Abbildung zeigt die BeeGFS-Konfiguration, die zur Validierung der BeeGFS auf NetApp Lösung verwendet wird.

BeeGFS-Datei-Striping

Ein Vorteil paralleler Dateisysteme besteht darin, einzelne Dateien über mehrere Storage-Ziele zu verteilen, wodurch Volumes auf demselben oder verschiedenen zugrunde liegenden Storage-Systemen dargestellt werden können.

In BeeGFS können Sie Striping auf Verzeichnisbasis und pro Datei konfigurieren, um die Anzahl der für jede Datei verwendeten Ziele zu steuern und die für jeden Dateistripe verwendete Chunksize (oder Blockgröße) zu steuern. Bei dieser Konfiguration kann das Filesystem verschiedene Workload- und I/O-Profile unterstützen, ohne Services neu konfigurieren oder neu starten zu müssen. Sie können Stripe-Einstellungen mit dem anwenden `beegfs-ctl` Befehlszeilen-Tool oder Anwendungen, die die Striping-API verwenden. Weitere Informationen finden Sie in der BeeGFS-Dokumentation für "Striping" Und "Striping-API".

Um eine optimale Leistung zu erzielen, wurden während des Tests Streifenmuster angepasst und die für jeden Test verwendeten Parameter werden notiert.

IOR-Bandbreitentests: Mehrere Clients

Die IOR-Bandbreitentests verwendeten OpenMPI, um parallele Jobs des synthetischen E/A-Generatorwerkzeugs IOR auszuführen (verfügbar unter "HPC GitHub") Über alle 10 Client-Knoten zu einem oder mehreren BeeGFS-Bausteinen. Sofern nicht anders angegeben:

- Alle Tests verwendeten einen direkten I/O mit einer Übertragungsgröße von 1 MiB.
- BeeGFS-Datei-Striping wurde auf 1 MB Chunksize und ein Ziel pro Datei eingestellt.

Für IOR wurden die folgenden Parameter verwendet, wobei die Segmentanzahl angepasst wurde, um die aggregierte Dateigröße für einen Baustein auf 5 tib und 40 tib für drei Bausteine zu halten.

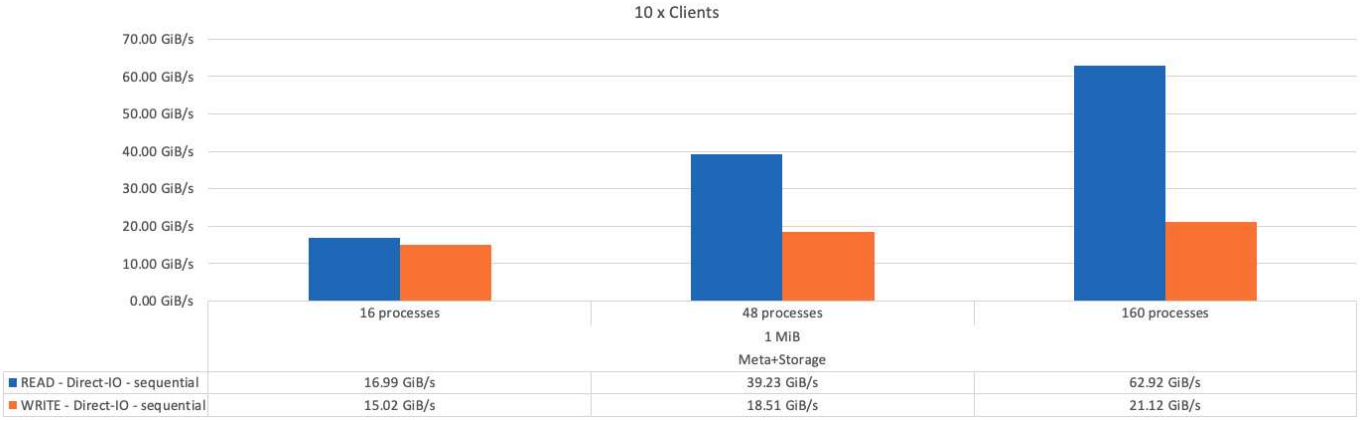
```
mpirun --allow-run-as-root --mca btl tcp -np 48 -map-by node -hostfile
10xnodes ior -b 1024k --posix.odirect -e -t 1024k -s 54613 -z -C -F -E -k
```

Baustein für eine BeeGFS-Basis (Management, Metadaten und Storage

Die folgende Abbildung zeigt die IOR-Testergebnisse mit einem einzelnen BeeGFS-Basisspeicher (Management, Metadaten und Storage).

BeeGFS Metadaten + Storage-Baustein

Die folgende Abbildung zeigt die IOR-Testergebnisse mit einem einzelnen BeeGFS-Metadaten + einem Storage-Baustein.

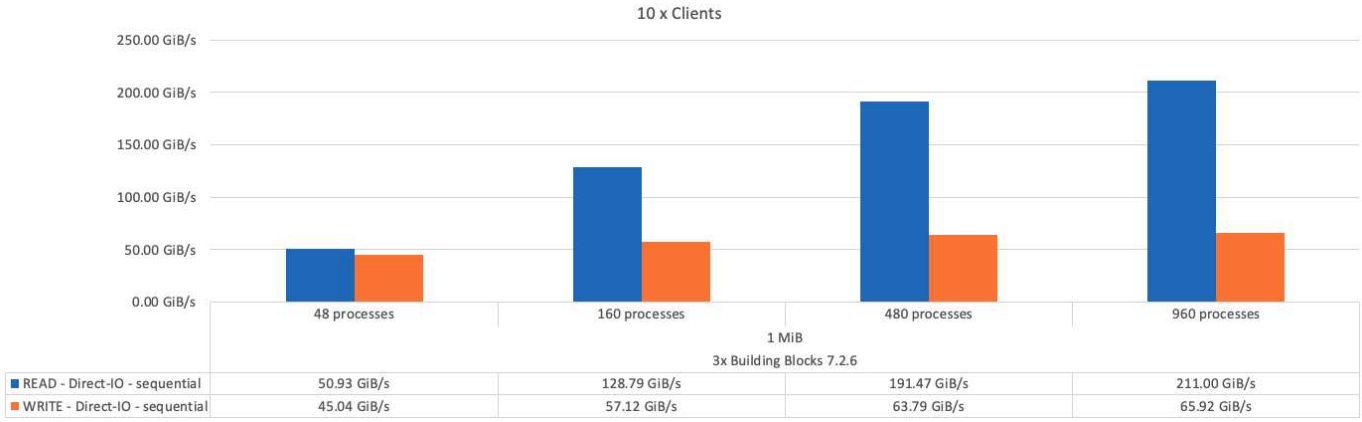


BeeGFS-Speicherbaustein

Die folgende Abbildung zeigt die IOR-Testergebnisse mit einem einzelnen BeeGFS-Storage-only-Baustein.

Drei BeeGFS-Bausteine

Die folgende Abbildung zeigt die IOR-Testergebnisse mit drei BeeGFS-Bausteinen.



Wie erwartet, ist der Performance-Unterschied zwischen dem Basis-Baustein und den nachfolgenden Metadaten + dem Storage-Baustein vernachlässigbar. Ein Vergleich zwischen Metadaten und Storage-Bausteinen und einem ausschließlich Storage-Baustein zeigt einen leichten Anstieg der Lese-Performance aufgrund der zusätzlichen Laufwerke, die als Storage-Ziele verwendet werden. Allerdings gibt es keinen wesentlichen Unterschied in der Schreib-Performance. Um eine höhere Performance zu erzielen, können Sie mehrere Bausteine gleichzeitig hinzufügen, um die Performance linear zu skalieren.

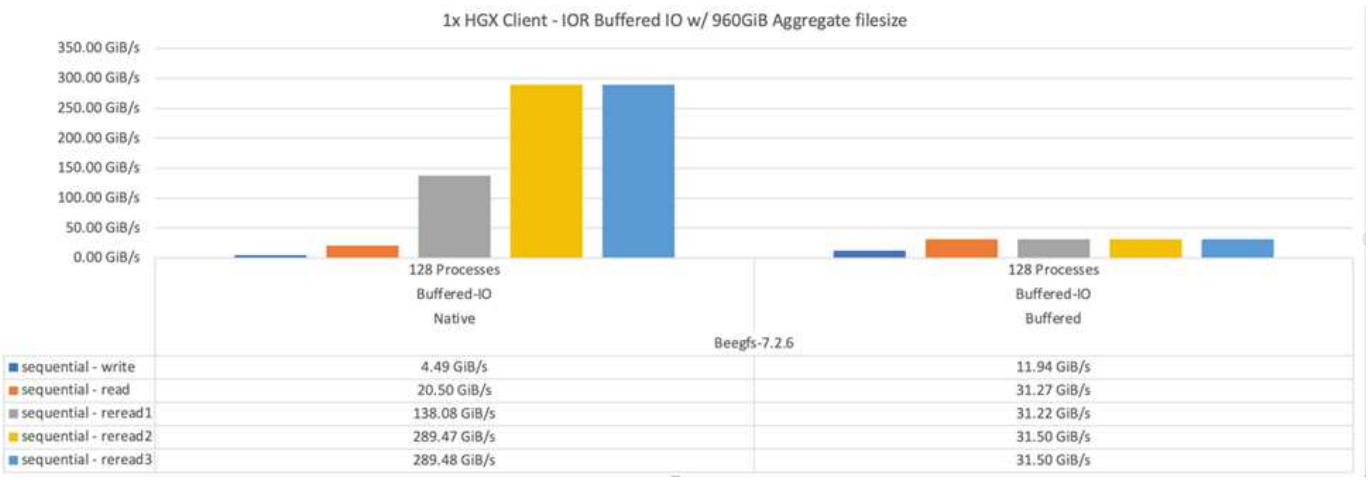
IOR-Bandbreitentests: Einzelner Client

Der IOR-Bandbreitentest nutzte OpenMPI, um mehrere IOR-Prozesse mithilfe eines einzigen leistungsstarken GPU-Servers auszuführen, um die Performance zu untersuchen, die für einen einzelnen Client erreichbar ist.

Dieser Test vergleicht auch das Verhalten und die Leistung von BeeGFS, wenn der Client so konfiguriert ist, dass er den Linux-Kernel-Page-Cache verwendet (`tuneFileCacheType = native`) Im Vergleich zum Standard `buffered` Einstellung.

Der native Caching-Modus verwendet den Linux-Kernel-Page-Cache auf dem Client, sodass die Readervorgänge nicht über das Netzwerk übertragen werden, sondern aus dem lokalen Speicher stammen.

Das folgende Diagramm zeigt die IOR-Testergebnisse mit drei BeeGFS-Bausteinen und einem einzelnen Client.



BeeGFS Striping für diese Tests wurde auf 1 MB Chunksize mit acht Zielen pro Datei eingestellt.

Obwohl die Performance bei den Schreibzugriffen und beim ersten Lesen im standardmäßigen gepufferten Modus höher ist, werden bei Workloads, die dieselben Daten mehrmals lesen, eine deutliche Performance-

Steigerung im nativen Caching-Modus erzielt. Diese verbesserte Performance bei erneuten Lesevorgängen ist für Workloads wie Deep Learning wichtig, die denselben Datensatz mehrmals in vielen Epoch-Durchläufen lesen.

Metadaten-Performance-Test

Bei den Metadaten-Performance-Tests wurde das MDTest-Tool (im Rahmen von IOR enthalten) verwendet, um die Metadaten-Performance von BeeGFS zu messen. Die Tests verwendeten OpenMPI, um parallele Jobs auf allen zehn Client-Knoten auszuführen.

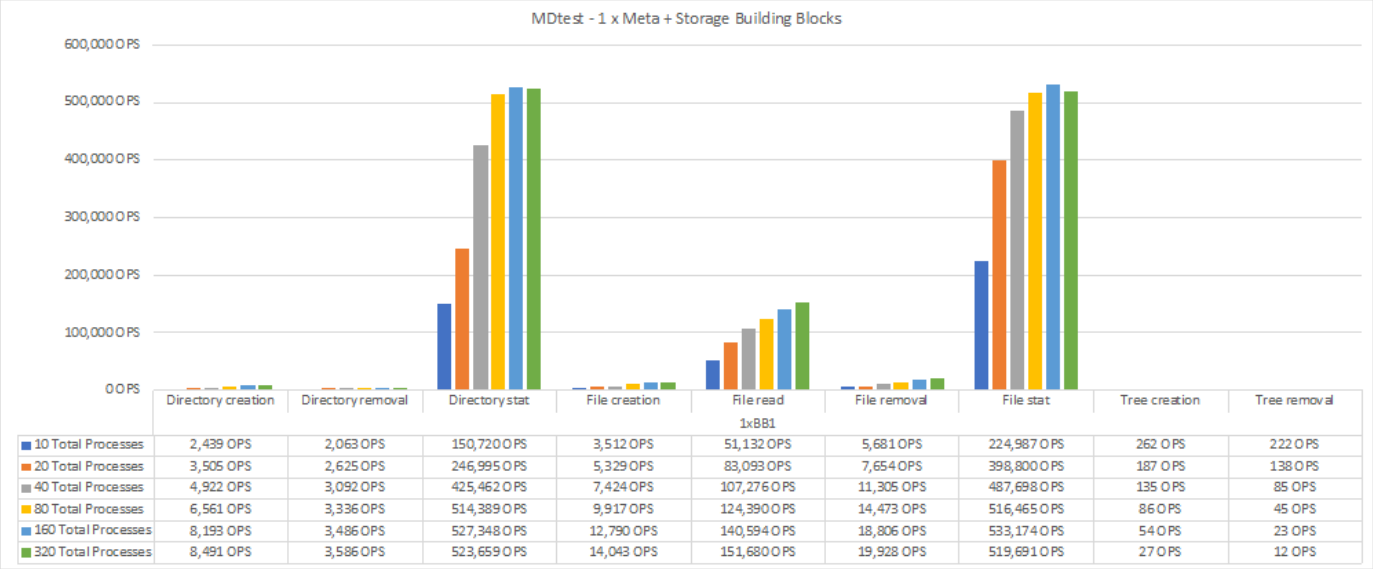
Die folgenden Parameter wurden verwendet, um den Benchmark-Test mit der Gesamtzahl der Prozesse von 10 auf 320 in Schritt 2 x und mit einer Dateigröße von 4k durchgeführt.

```
mpirun -h 10xnodes –map-by node np $processes mdtest -e 4k -w 4k -i 3 -I
16 -z 3 -b 8 -u
```

Die Metadaten-Performance wurde zuerst mit ein bis zwei Metadaten + Storage-Bausteinen gemessen, um die Performance durch das Hinzufügen weiterer Bausteine zu verdeutlichen.

1 BeeGFS Metadaten + Storage-Baustein

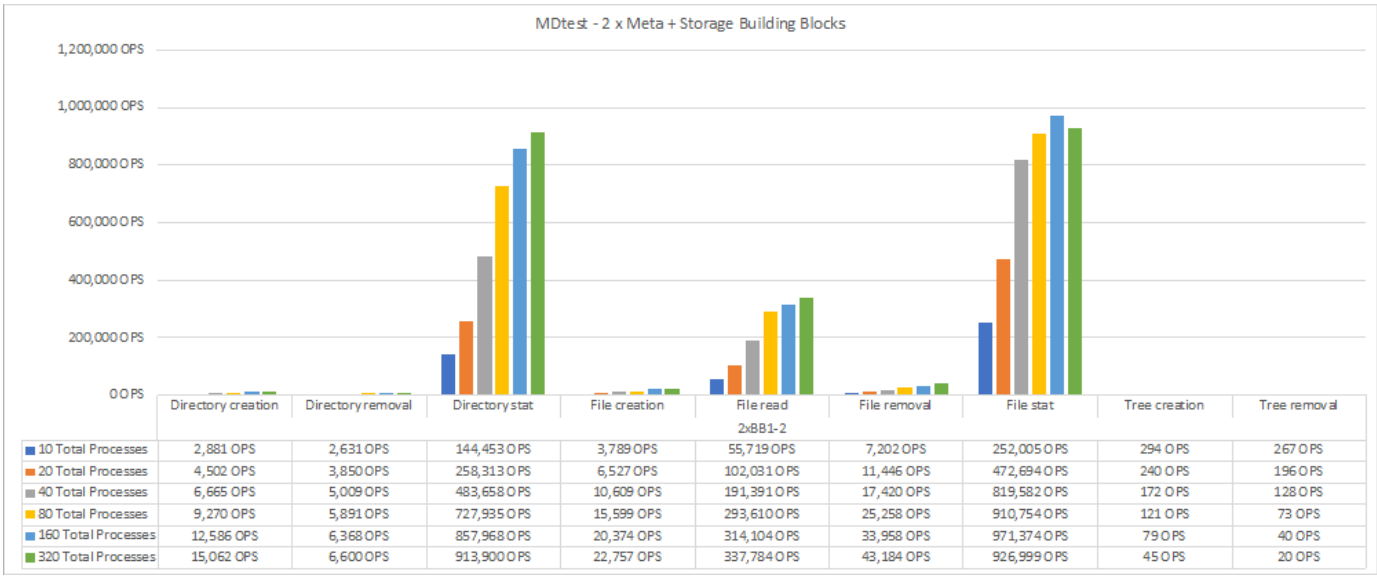
Das folgende Diagramm zeigt die MDTest-Ergebnisse mit einer BeeGFS-Metadaten + Speicherbausteinen.



1xBB1	Directory creation	Directory removal	Directory stat	File creation	File read	File removal	File stat	Tree creation	Tree removal
10 Total Processes	2,439 OPS	2,063 OPS	150,720 OPS	3,512 OPS	51,132 OPS	5,681 OPS	224,987 OPS	262 OPS	222 OPS
20 Total Processes	3,505 OPS	2,625 OPS	246,995 OPS	5,329 OPS	83,093 OPS	7,654 OPS	398,800 OPS	187 OPS	138 OPS
40 Total Processes	4,922 OPS	3,092 OPS	425,462 OPS	7,424 OPS	107,276 OPS	11,305 OPS	487,698 OPS	135 OPS	85 OPS
80 Total Processes	6,561 OPS	3,336 OPS	514,389 OPS	9,917 OPS	124,390 OPS	14,473 OPS	516,465 OPS	86 OPS	45 OPS
160 Total Processes	8,193 OPS	3,486 OPS	527,348 OPS	12,790 OPS	140,594 OPS	18,806 OPS	533,174 OPS	54 OPS	23 OPS
320 Total Processes	8,491 OPS	3,586 OPS	523,659 OPS	14,043 OPS	151,680 OPS	19,928 OPS	519,691 OPS	27 OPS	12 OPS

Zwei BeeGFS Metadaten + Storage-Bausteine

Das folgende Diagramm zeigt die MDTest-Ergebnisse mit zwei BeeGFS-Metadaten + Speicherbausteinen.



	Directory creation	Directory removal	Directory stat	File creation	File read	File removal	File stat	Tree creation	Tree removal
					2xBB1-2				
10 Total Processes	2,881 OPS	2,631 OPS	144,453 OPS	3,789 OPS	55,719 OPS	7,202 OPS	252,005 OPS	294 OPS	267 OPS
20 Total Processes	4,502 OPS	3,850 OPS	258,313 OPS	6,527 OPS	102,031 OPS	11,446 OPS	472,694 OPS	240 OPS	196 OPS
40 Total Processes	6,665 OPS	5,009 OPS	483,658 OPS	10,609 OPS	191,391 OPS	17,420 OPS	819,582 OPS	172 OPS	128 OPS
80 Total Processes	9,270 OPS	5,891 OPS	727,935 OPS	15,599 OPS	293,610 OPS	25,258 OPS	910,754 OPS	121 OPS	73 OPS
160 Total Processes	12,586 OPS	6,368 OPS	857,968 OPS	20,374 OPS	314,104 OPS	33,958 OPS	971,374 OPS	79 OPS	40 OPS
320 Total Processes	15,062 OPS	6,600 OPS	913,900 OPS	22,757 OPS	337,784 OPS	43,184 OPS	926,999 OPS	45 OPS	20 OPS

Funktionsprüfung

Im Rahmen der Validierung dieser Architektur führte NetApp mehrere Funktionstests durch, darunter:

- Ausfall eines einzelnen InfiniBand-Ports des Clients durch Deaktivieren des Switch-Ports
- Ausfall eines InfiniBand-Ports mit einem einzelnen Server durch Deaktivieren des Switch-Ports
- Sofortige Abschaltung des Servers mithilfe des BMC.
- Anmutig Platzierung eines Node im Standby-Modus und Failover-Betrieb zu einem anderen Node
- Anmutig Setzen eines Node wieder online und Failback-Services auf den ursprünglichen Node.
- Schalten Sie einen der InfiniBand-Switches mithilfe der PDU aus. Alle Tests wurden durchgeführt, während Belastungstests mit dem durchgeführt wurden `sysSessionChecksEnabled: false` Parameter auf BeeGFS-Clients gesetzt. Es wurden keine Fehler oder Störungen bei I/O festgestellt.

> Es ist ein bekanntes Problem aufgetreten (siehe "Changelog") Wenn BeeGFS-Client/Server-RDMA-Verbindungen unerwartet unterbrochen werden, entweder durch Ausfall der primären Schnittstelle (wie in definiert `connInterfacesFile`) Oder ein BeeGFS-Server fällt aus. Aktive Client-I/O kann bis zu zehn Minuten lang aufhängen, bevor der Vorgang fortgesetzt wird. Dieses Problem tritt nicht auf, wenn BeeGFS-Knoten ordnungsgemäß für geplante Wartung in den Standby-Modus versetzt oder TCP verwendet wird.

Validierung von NVIDIA DGX SuperPOD und BasePOD

NetApp validierte eine Storage-Lösung für IANVIDDGX A100 SuperPOD unter Verwendung eines ähnlichen BeeGFS Filesystem, das aus drei Bausteinen mit den Metadaten und angewandtem Storage-Konfigurationsprofil besteht. Die Qualifizierung bestand darin, die von dieser NVA beschriebene Lösung mit zwanzig DGX A100 GPU-Servern zu testen, auf denen eine Vielzahl von Storage-, Machine-Learning- und Deep-Learning-Benchmarks ausgeführt wurden. Aufbauend auf der Validierung mit DGX A100 SuperPOD von NVIDIA wurde die BeeGFS auf NetApp Lösung für DGX SuperPOD H100-, H200- und B200-Systeme genehmigt. Diese Erweiterung basiert auf der Erfüllung der zuvor mit dem NVIDIA DGX A100 validierten Benchmarks und Systemanforderungen.

Weitere Informationen finden Sie unter "NVIDIA DGX SuperPOD mit NetApp" Und "NVIDIA DGX BasePOD".

Richtlinien für die Dimensionierung

Die BeeGFS Lösung enthält Empfehlungen für das Performance- und Kapazitäts-Sizing, die auf Verifizierungstests basieren.

Die Bausteinarchitektur verfolgt das Ziel, eine Lösung zu entwickeln, die sich einfach dimensionieren lässt. Dazu werden mehrere Bausteine hinzugefügt, die die Anforderungen für ein bestimmtes BeeGFS-System erfüllen. Mithilfe der nachstehenden Richtlinien können Sie die Anzahl und die Arten von BeeGFS-Bausteinen schätzen, die für die Anforderungen Ihrer Umgebung benötigt werden.

Beachten Sie, dass es sich bei diesen Schätzungen um die beste Performance handelt. Benchmark-Applikationen werden geschrieben und verwendet, um die Nutzung der zugrunde liegenden Filesysteme auf eine Weise zu optimieren, die reale Applikationen vielleicht nicht erreichen.

Performance-Dimensionierung

Die folgende Tabelle enthält ein empfohlene Performance-Sizing.

Konfigurationsprofil	1 MiB Lesevorgänge	1 MiB Schreibvorgänge
Metadaten + Storage	62Gibps	21Gibps
Nur Storage	64Gibps	21Gibps

Die Schätzungen zur Größe der Metadaten-Kapazität basieren auf der „Faustregel“, dass eine Kapazität von 500 GB für etwa 150 Millionen Dateien in BeeGFS ausreichend ist. (Weitere Informationen finden Sie in der BeeGFS-Dokumentation für "Systemanforderungen".)

Die Verwendung von Funktionen wie Zugriffssteuerungslisten und die Anzahl der Verzeichnisse und Dateien pro Verzeichnis wirkt sich auch darauf aus, wie schnell der Metadatenspeicherplatz verbraucht wird. Schätzungen zur Storage-Kapazität berücksichtigen neben RAID 6 und XFS-Overhead die nutzbare Laufwerkskapazität.

Kapazitätsdimensionierung für Metadaten + Storage-Bausteine

Die folgende Tabelle enthält eine empfohlene Kapazitätsdimensionierung für Metadaten und Storage-Bausteine.

Laufwerksgröße (2+2 RAID 1) Metadaten-Volume-Gruppen	Metadaten-Kapazität (Anzahl der Dateien)	Laufwerksgröße (8+2 RAID 6) Storage-Volume-Gruppen	Speicherkapazität (File-Inhalte)
1,92 TB	1,938,577,200	1,92 TB	51,77 TB
3,84 TB	3,880,388,400	3,84 TB	103,5 TB
7,68 TB	8,125,278,000	7,68 TB	216,74 TB
15,3 TB	17,269,854,000	15,3 TB	460 TB

Bei der Größenbestimmung von Metadaten und Storage-Bausteinen können die Kosten reduziert werden, da weniger Laufwerke für Metadaten-Volume-Gruppen anstelle von Storage-Volume-Gruppen verwendet werden.

Kapazitätsdimensionierung für reine Storage-Bausteine

Die folgende Tabelle zeigt die Größenanpassung der Kapazität für reine Storage-Bausteine.

Laufwerksgröße (10+2 RAID 6) Storage-Volume-Gruppen	Speicherkapazität (File-Inhalte)
1,92 TB	59,89 TB
3,84 TB	119,80 TB
7,68 TB	251,89 TB
15,3 TB	538,5 TB

Die Performance- und der Kapazitäts-Overhead, die durch das Einbinden des Management-Service in den Basis-Baustein (erster) verursacht werden, sind minimal, es sei denn, die globale Dateisperrung ist aktiviert.

Performance-Optimierung

Die BeeGFS-Lösung enthält Empfehlungen für Performance-Tuning, die auf Verifikationstests basieren.

Obwohl BeeGFS Out-of-the-Box-Performance bietet, hat NetApp eine Reihe von empfohlenen Tuning-Parametern entwickelt, um die Performance zu maximieren. Diese Parameter berücksichtigen die Funktionen der zugrunde liegenden Block-Nodes der E-Series und alle speziellen Anforderungen, die für die Ausführung von BeeGFS in einer HA-Architektur mit Shared-Festplatten erforderlich sind.

Performance-Tuning für Datei-Nodes

Zu den verfügbaren Tuning-Parametern, die Sie konfigurieren können, gehören folgende:

1. **Systemeinstellungen im UEFI/BIOS von Datei-Nodes.** um die Leistung zu maximieren, empfehlen wir die Konfiguration der Systemeinstellungen auf dem Server-Modell, das Sie als Datei-Knoten verwenden. Sie konfigurieren die Systemeinstellungen, wenn Sie die Datei-Nodes einrichten, indem Sie entweder das System-Setup (UEFI/BIOS) oder die Redfish-APIs verwenden, die vom Baseboard-Management-Controller (BMC) bereitgestellt werden.

   Die Systemeinstellungen variieren je nach Servermodell, das Sie als Dateiknoten verwenden. Die Einstellungen müssen manuell auf der Grundlage des verwendeten Servermodells konfiguriert werden. Informationen zum Konfigurieren der Systemeinstellungen für die validierten Lenovo SR665 V3-Dateiknoten finden Sie unter "Optimieren Sie die System-Einstellungen des File Node für die Performance" .

2. **Standardeinstellungen für erforderliche Konfigurationsparameter.** die erforderlichen Konfigurationsparameter beeinflussen die Konfiguration von BeeGFS-Diensten und die Formatierung und Bereitstellung von E-Series-Volumes (Blockgeräte) durch Pacemaker. Die folgenden Konfigurationsparameter sind erforderlich:
   - BeeGFS Service-Konfigurationsparameter

     Sie können die Standardeinstellungen für die Konfigurationsparameter bei Bedarf überschreiben. Informationen zu den Parametern, die Sie an Ihre spezifischen Workloads oder Anwendungsfälle anpassen können, finden Sie im "Parameter für BeeGFS-Service-Konfiguration".

- Für die Volume-Formatierung und die Montage-Parameter gelten die empfohlenen Standardeinstellungen, die nur für erweiterte Anwendungsfälle angepasst werden sollten. Die Standardwerte führen folgende Schritte aus:
  - Optimieren Sie die ursprüngliche Volume-Formatierung auf Basis des Zieltyps (beispielsweise Management, Metadaten oder Storage), zusammen mit der RAID-Konfiguration und Segmentgröße des zugrunde liegenden Volumes.
  - Passen Sie an, wie Pacemaker die einzelnen Lautstärkerahmen einstellt, um sicherzustellen, dass Änderungen sofort an Block-Nodes der E-Series gespült werden. So wird Datenverlust verhindert, wenn bei aktiven Schreibvorgängen Datei-Nodes ausfallen.

    Informationen zu den Parametern, die Sie an Ihre spezifischen Workloads oder Anwendungsfälle anpassen können, finden Sie im "Volume-Formatierung und Montage der Konfigurationsparameter".

3. **Systemeinstellungen im Linux-Betriebssystem, das auf den Dateiknoten installiert ist.** Sie können die standardmäßigen Linux OS-Systemeinstellungen überschreiben, wenn Sie den Ansible-Bestand in Schritt 4 von erstellen "Erstellen des Ansible-Inventars".

   Die Standardeinstellungen wurden zur Validierung der BeeGFS auf NetApp Lösung verwendet. Sie können sie jedoch an Ihre spezifischen Workloads oder Anwendungsfälle anpassen. Einige Beispiele für die Linux-Betriebssystemeinstellungen, die Sie ändern können, sind:

   - I/O-Warteschlangen an Block-Geräten der E-Series.

     Auf den als BeeGFS-Ziele verwendeten E-Series-Block-Geräten können Sie I/O-Warteschlangen folgendermaßen konfigurieren:

     - Passen Sie den Planungsalgorithmus basierend auf dem Gerätetyp (NVMe, HDD usw.) an.
     - Erhöhen Sie die Anzahl der ausstehenden Anfragen.
     - Passen Sie Anfragegrößen an.
     - Optimierung des Verhaltens beim Lesen.
   - Einstellungen für virtuellen Speicher.

     Sie können die Einstellungen für den virtuellen Speicher für eine optimale Streaming-Leistung anpassen.

   - CPU-Einstellungen.

     Sie können den CPU-Frequenzregler und andere CPU-Konfigurationen für maximale Leistung anpassen.

   - Anfragegröße lesen

     Sie können die maximale Größe für Leseanforderungen für NVIDIA-HCAs erhöhen.

Performance-Tuning für Block-Nodes

Basierend auf den Konfigurationsprofilen, die auf einen bestimmten BeeGFS-Baustein angewendet werden, ändern sich die auf den Block-Nodes konfigurierten Volume-Gruppen leicht. Beispiel mit einem EF600 Block-Node mit 24 Laufwerken:

- Für den einzelnen Basis-Block, einschließlich BeeGFS-Management, Metadaten und Storage-Services:

    - 1x 2+2 RAID 10 Volume-Gruppe für BeeGFS Management und Metadaten-Services
    - 2 x 8+2 RAID 6-Volume-Gruppen für BeeGFS-Storage-Services
- Für BeeGFS Metadaten + Storage-Baustein:
    - 1x 2+2 RAID 10 Volume-Gruppe für BeeGFS Metadaten-Services
    - 2 x 8+2 RAID 6-Volume-Gruppen für BeeGFS-Storage-Services
- Nur für BeeGFS-Lagergebäude:
    - 2 x 10+2 RAID 6-Volume-Gruppen für BeeGFS-Storage-Services

> Da BeeGFS für Management und Metadaten im Gegensatz zu Storage erheblich weniger Speicherplatz benötigt, besteht die Möglichkeit, für die RAID 10-Volume-Gruppen kleinere Laufwerke zu verwenden. Kleinere Laufwerke sollten in den äußeren Laufwerksteckplätzen befüllt werden. Weitere Informationen finden Sie im "Implementierungsanleitungen".

Diese werden alle durch die Ansible-basierte Implementierung konfiguriert, und verschiedene andere allgemein empfohlene Einstellungen für Performance-/Verhaltensoptimierung:

- Anpassung der globalen Cache-Blockgröße auf 32 KiB und Anpassung der bedarfsorientierten Cache-Flush auf 80 %.
- Deaktivieren des automatischen Load-Balancing (sicherstellen, dass die Controller-Volume-Zuweisungen wie vorgesehen bleiben).
- Aktivieren von Lese-Caching und Deaktivieren des Read-Ahead-Caching
- Aktivieren des Schreib-Caches mit Spiegelung und Bedarf an Akku-Backups, sodass Caches während des Ausfalls eines Block-Node-Controllers bestehen.
- Festlegen der Reihenfolge, in der Laufwerke Volume-Gruppen zugewiesen werden, und Ausgleich der I/O-Vorgänge über verfügbare Laufwerkskanäle

Baustein mit hoher Kapazität

Bei der Entwicklung der BeeGFS-Standardlösung wurde auf Workloads mit hoher Performance gedenkt. Kunden, die nach Anwendungsfällen mit hoher Kapazität suchen, sollten die hier beschriebenen Design- und Performance-Unterschiede beobachten.

Hardware- und Softwarekonfiguration

Hardware- und Softwarekonfiguration für den Baustein mit hoher Kapazität ist Standard. Es sei denn, die EF600 Controller sollten durch einen EF300 Controller ersetzt werden. Sie können zwischen 1 und 7 IOM-Erweiterungsfächern mit jeweils 60 Laufwerken pro Storage-Array anschließen, Insgesamt 2 bis 14 Erweiterungsfächer pro Baustein.

Unternehmen, die ein High-Capacity-Baustein-Design implementieren, verwenden wahrscheinlich nur die Basis-Bausteinkonfiguration, die aus BeeGFS-Management, Metadaten und Storage-Services für jeden Node besteht. Um die Kosteneffizienz zu steigern, sollten Storage-Nodes mit hoher Kapazität Metadaten-Volumes auf den NVMe-Laufwerken im EF300-Controller-Gehäuse bereitstellen und Storage-Volumes für die NL-SAS-Laufwerke in den Erweiterungsfächern bereitstellen.

[]

Richtlinien für die Dimensionierung

Bei diesen Richtlinien zur Dimensionierung ist davon auszugehen, dass Bausteine mit hoher Kapazität mit einer NVMe-SSD-Volume-Gruppe von 2+2 für Metadaten im Basis-EF300-Gehäuse und 6 x 8+2 NL-SAS-Volume-Gruppen pro IOM-Erweiterungsfach für Storage konfiguriert sind.

Laufwerkgröße (Kapazitäts-HDDs)	Kapazität pro BB (1 Fach)	Kapazität pro BB (2 Einschübe)	Kapazität pro BB (3 Einschübe)	Kapazität pro BB (4 Einschübe)
4 TB	439 TB	878 TB	1317 TB	1756 TB
8 TB	878 TB	1756 TB	2634 TB	3512 TB
10 TB	1097 TB	2195 TB	3292 TB	4390 TB
12 TB	1317 TB	2634 TB	3951 TB	5268 TB
16 TB	1756 TB	3512 TB	5268 TB	7024 TB
18 TB	1975 TB	3951 TB	5927 TB	7902 TB

Implementieren der Lösung

Implementierungsübersicht

BeeGFS auf NetApp kann auf validierte Datei- und Block-Nodes mithilfe von Ansible mit dem BeeGFS-Baustein-Design von NetApp implementiert werden.

Ansible-Sammlungen und -Funktionen

Die BeeGFS auf NetApp-Lösung wird mithilfe von Ansible implementiert, einer beliebten IT-Automatisierungs-Engine, die Applikationsimplementierungen automatisiert. Ansible verwendet eine Reihe von Dateien, die gemeinsam als Inventar bezeichnet werden. Hierbei wird das BeeGFS-Filesystem modelliert, das Sie implementieren möchten.

Ansible ermöglicht Unternehmen wie NetApp die Erweiterung auf integrierte Funktionen mithilfe von Sammlungen, die auf Ansible Galaxy verfügbar sind (siehe "NetApp E-Series BeeGFS Sammlung"). Sammlungen umfassen Module, die bestimmte Funktionen oder Aufgaben (wie das Erstellen eines E-Series Volumes) ausführen, sowie Rollen, die mehrere Module und andere Rollen aufrufen können. Dieser automatisierte Ansatz reduziert die Zeit für die Implementierung des BeeGFS-Filesystems und des zugrunde liegenden HA-Clusters. Darüber hinaus vereinfacht es die Wartung und Erweiterung des Clusters und des BeeGFS-Dateisystems.

Weitere Informationen finden Sie unter "Weitere Informationen zum Ansible Inventar".

Da zahlreiche Schritte an der Implementierung der BeeGFS auf NetApp-Lösung beteiligt sind, unterstützt NetApp die manuelle Bereitstellung der Lösung nicht.

Konfigurationsprofile für BeeGFS-Bausteine

Die Implementierungsverfahren umfassen die folgenden Konfigurationsprofile:

- Ein einziger Baustein, der Management-, Metadaten- und Storage-Services umfasst
- Ein zweiter Baustein, der Metadaten und Storage-Services umfasst.

- Ein dritter Baustein, der nur Storage-Services umfasst.

Diese Profile veranschaulichen die gesamte Palette der empfohlenen Konfigurationsprofile für die NetApp BeeGFS-Bausteine. Bei jeder Implementierung kann die Anzahl der Metadaten und Storage-Bausteine oder nur-Storage-Services-Bausteine je nach Kapazitäts- und Performance-Anforderungen variieren.

Übersicht über die einzelnen Implementierungsschritte

Die Bereitstellung umfasst folgende allgemeine Aufgaben:

Hardwarebereitstellung

1. Stellen Sie jeden Baustein physisch zusammen.
2. Rack-und Kabelhardware: Ausführliche Verfahren finden Sie unter "Implementierung von Hardware".

Softwareimplementierung

1. "Richten Sie Datei- und Block-Nodes ein".
   - Konfigurieren Sie BMC-IPs auf Datei-Knoten
   - Installieren Sie ein unterstütztes Betriebssystem und konfigurieren Sie Managementnetzwerk auf Datei-Knoten
   - Konfiguration der Management-IPs auf Block-Nodes
2. "Richten Sie einen Ansible-Steuerungsknoten ein".
3. "Passen Sie die Systemeinstellungen für die Performance an".
4. "Erstellen des Ansible-Inventars".
5. "Definieren Sie den Ansible-Bestand für BeeGFS-Bausteine".
6. "Implementieren Sie BeeGFS mit Ansible".
7. "Konfigurieren Sie BeeGFS-Clients".

> Die Bereitstellungsverfahren umfassen mehrere Beispiele, in denen Text in eine Datei kopiert werden muss. Achten Sie besonders auf Inline-Kommentare, die durch die Zeichen „#“ oder „//“ gekennzeichnet sind und auf alles hinweisen, was für eine bestimmte Bereitstellung geändert werden sollte oder kann. Beispiel:
>
> ```
> `beegfs_ha_ntp_server_pools:  # THIS IS AN EXAMPLE OF A COMMENT!
>   - "pool 0.pool.ntp.org iburst maxsources 3"
>   - "pool 1.pool.ntp.org iburst maxsources 3"`
> ```

Derivative Architekturen mit Variationen bei Implementierungsempfehlungen:

- "Baustein Mit Hoher Kapazität"

Weitere Informationen zum Ansible Inventar

Machen Sie sich vor der Implementierung mit der Konfiguration von Ansible vertraut und werden Sie zur Implementierung der BeeGFS auf NetApp Lösung verwendet.

Der Ansible-Bestand ist eine Verzeichnisstruktur mit den Datei- und Block-Nodes für das zu implementierende

BeeGFS-Filesystem. Es enthält Hosts, Gruppen und Variablen, die das gewünschte BeeGFS-Dateisystem beschreiben. Die Ansible-Bestandsaufnahme muss auf dem Ansible-Steuerungsknoten gespeichert werden, bei dem es sich um jeden Computer mit Zugriff auf die Datei- und Block-Nodes handelt, mit denen das Ansible-Playbook ausgeführt wird. Probenbestände können von der heruntergeladen werden "NetApp E-Series BeeGFS GitHub".

Ansible-Module und -Rollen

Um die im Ansible Inventar beschriebene Konfiguration anzuwenden, verwenden Sie die verschiedenen Ansible Module und Rollen aus der NetApp E-Series Ansible Sammlung (verfügbar über "NetApp E-Series BeeGFS GitHub"), die die End-to-End-Lösung implementieren.

Jede Rolle der NetApp E-Series Ansible Sammlung ist eine vollständige End-to-End-Implementierung der BeeGFS auf NetApp Lösung. Die Rollen verwenden die Sammlungen NetApp E-Series SANtricity, Host und BeeGFS, mit denen Sie das BeeGFS Filesystem mit HA (High Availability, Hochverfügbarkeit) konfigurieren können. Anschließend können Sie Storage bereitstellen und zuordnen und den Cluster-Storage betriebsbereit machen.

Die Rollen verfügen über eine ausführliche Dokumentation. In den Implementierungsverfahren wird beschrieben, wie die Rolle bei der Implementierung einer NetApp Verified Architecture mit dem BeeGFS-Bausteindesign der zweiten Generation eingesetzt wird.

> Obwohl die Implementierungsschritte versucht, genügend Details bereitzustellen, sodass frühere Erfahrungen mit Ansible nicht erforderlich sind, sollten Sie mit Ansible und der zugehörigen Terminologie vertraut sein.

Bestandslayout für BeeGFS HA-Cluster

Definieren Sie ein BeeGFS HA-Cluster mit der Ansible-Bestandsstruktur.

Jeder mit früheren Ansible-Erfahrungen sollte sich bewusst sein, dass die BeeGFS-HA-Rolle eine benutzerdefinierte Methode implementiert, um zu ermitteln, welche Variablen (oder Fakten) für jeden Host gelten. Dieses Design vereinfacht die Strukturierung des Ansible-Bestands, um Ressourcen zu beschreiben, die auf mehreren Servern ausgeführt werden können.

Ein Ansible-Inventar besteht in der Regel aus den Dateien in `host_vars` und `group_vars` sowie einer `inventory.yml` Datei, die Hosts spezifischen Gruppen (und potenziell Gruppen anderen Gruppen) zuweist.

> Erstellen Sie keine Dateien mit dem Inhalt in diesem Unterabschnitt, der nur als Beispiel gedacht ist.

Obwohl diese Konfiguration anhand des Konfigurationsprofils vorab festgelegt ist, sollten Sie wie folgt allgemeine Kenntnisse darüber haben, wie alles als Ansible-Inventar ausgelegt ist:

```
# BeeGFS HA (High Availability) cluster inventory.
all:
  children:
    # Ansible group representing all block nodes:
    eseries_storage_systems:
      hosts:
        netapp01:
        netapp02:
    # Ansible group representing all file nodes:
    ha_cluster:
      children:
        meta_01:  # Group representing a metadata service with ID 01.
          hosts:
            beegfs_01:  # This service is preferred on the first file
node.
            beegfs_02:  # And can failover to the second file node.
        meta_02:  # Group representing a metadata service with ID 02.
          hosts:
            beegfs_02:  # This service is preferred on the second file
node.
            beegfs_01: # And can failover to the first file node.
```

Für jeden Dienst wird unter eine zusätzliche Datei erstellt `group_vars` Beschreibung der Konfiguration:

```
# meta_01 - BeeGFS HA Metadata Resource Group
beegfs_ha_beegfs_meta_conf_resource_group_options:
  connMetaPortTCP: 8015
  connMetaPortUDP: 8015
  tuneBindToNumaZone: 0
floating_ips:
  - i1b: <IP>/<SUBNET_MASK>
  - i2b: <IP>/<SUBNET_MASK>
# Type of BeeGFS service the HA resource group will manage.
beegfs_service: metadata # Choices: management, metadata, storage.
# What block node should be used to create a volume for this service:
beegfs_targets:
  netapp01:
    eseries_storage_pool_configuration:
      - name: beegfs_m1_m2_m5_m6
        raid_level: raid1
        criteria_drive_count: 4
        common_volume_configuration:
          segment_size_kb: 128
        volumes:
          - size: 21.25
            owning_controller: A
```

Mit diesem Layout können BeeGFS-Service-, Netzwerk- und Storage-Konfigurationen für jede Ressource an einem Ort definiert werden. Hinter den Kulissen aggregiert die Rolle BeeGFS basierend auf dieser Bestandsstruktur die erforderliche Konfiguration für jede Datei und jeden Block-Node.

> Die numerische BeeGFS- und String-Node-ID für jeden Dienst wird automatisch auf Basis des Gruppennamens konfiguriert. Zusätzlich zur allgemeinen Ansible-Anforderung, dass Gruppennamen eindeutig sein sollen, müssen Gruppen, die einen BeeGFS-Service darstellen, in einer Zahl enden, die für den BeeGFS-Service eindeutig ist, für den diese Gruppe repräsentiert. Zum Beispiel sind meta_01 und stor_01 zulässig, aber Metadaten_01 und meta_01 sind nicht.

Besprechen der Best Practices

Beachten Sie bei der Implementierung der BeeGFS auf NetApp-Lösung die Best Practice-Richtlinien.

Standardkonventionen

Folgen Sie beim physischen Zusammenbau und Erstellen der Ansible-Bestandsdatei diesen Standardkonventionen (weitere Informationen finden Sie unter "Erstellen des Ansible-Inventars").

- Host-Namen der Dateiknoten werden nacheinander nummeriert (h01-HN) mit niedrigeren Zahlen oben im Rack und höheren Zahlen unten.

  Die Namenskonvention sieht beispielsweise `[location][row][rack]hN` wie folgt aus: `beegfs_01`.

- Jeder Block-Node besteht aus zwei Storage-Controllern, die jeweils über einen eigenen Host-Namen verfügen.

  Mit einem Storage-Array-Namen wird im Rahmen eines Ansible-Inventars das gesamte Block-Storage-System bezeichnet. Die Namen des Speicher-Arrays sollten nacheinander nummeriert sein (a01 - an), und die Hostnamen für einzelne Controller werden aus dieser Namenskonvention abgeleitet.

  Beispielsweise kann bei einem Block-Node mit dem Namen `ictad22a01` normalerweise Hostnamen für jeden Controller wie und konfiguriert `ictad22a01-a ictad22a01-b`sein, in einem Ansible-Inventar jedoch als bezeichnet werden `netapp_01`.

- File- und Block-Nodes innerhalb desselben Bausteins teilen sich das gleiche Nummerierungsschema und sind im Rack nebeneinander, wobei beide Datei-Nodes oben und beide Block-Nodes direkt darunter liegen.

  Im ersten Baustein sind beispielsweise die Datei-Nodes h01 und h02 direkt mit den Block-Nodes a01 und a02 verbunden. Von oben nach unten sind die Hostnamen h01, h02, a01 und a02.

- Bausteine werden in sequenzieller Reihenfolge auf der Grundlage ihrer Hostnamen installiert, sodass sich die niedrigeren Host-Namen oben im Rack befinden und die höheren nummerierten Host-Namen sich unten befinden.

  Ziel ist es, die Länge des Kabels zu minimieren, das oben auf den Rack Switches läuft, und eine standardisierte Implementierungspraxis zu definieren, um die Fehlerbehebung zu vereinfachen. Für Rechenzentren, in denen dies nicht erlaubt ist, aufgrund von Bedenken um die Rack-Stabilität, ist die umgekehrte sicherlich erlaubt, das Befüllen des Racks von unten nach oben.

InfiniBand-Storage-Netzwerkkonfiguration

Die Hälfte der InfiniBand-Ports an jedem Datei-Node werden für eine direkte Verbindung mit Block-Nodes verwendet. Die andere Hälfte ist mit den InfiniBand-Switches verbunden und wird für die BeeGFS-Client-Server-Konnektivität verwendet. Beim Bestimmen der Größe der IPoIB-Subnetze, die für BeeGFS-Clients und -Server verwendet werden, müssen Sie das erwartete Wachstum Ihres Compute/GPU-Clusters und BeeGFS-Dateisystems berücksichtigen. Wenn Sie von den empfohlenen IP-Bereichen abweichen müssen, beachten Sie, dass jede direkte Verbindung in einem einzelnen Baustein ein eigenes Subnetz hat und es keine Überschneidung mit Subnetzen gibt, die für die Client-Server-Konnektivität verwendet werden.

Direkte Verbindungen

Datei- und Block-Nodes innerhalb jedes Bausteins verwenden für ihre direkten Verbindungen immer die IPs in der folgenden Tabelle.

Dieses Adressprogramm entspricht der folgenden Regel: Das dritte Oktett ist immer ungerade oder gerade, was davon abhängt, ob der Datei-Node ungerade oder gerade ist.

Datei-Node	IB-Port	IP-Adresse	Block-Node	IB-Port	Physische IP-Adresse	Virtuelle IP
ODD (h1)	i1a	192.168.1.10	Ungerade (c1)	2 a	192.168.1.100	192.168.1.101
ODD (h1)	i2a	192.168.3.10	Ungerade (c1)	2 a	192.168.3.100	192.168.3.101
ODD (h1)	i3a	192.168.5.10	Gleichmäßig (c2)	2 a	192.168.5.100	192.168.5.101

Datei-Node	IB-Port	IP-Adresse	Block-Node	IB-Port	Physische IP-Adresse	Virtuelle IP
ODD (h1)	I4a	192.168.7.10	Gleichmäßig (c2)	2 a	192.168.7.100	192.168.7.101
Gleichmäßig (h2)	i1a	192.168.2.10	Ungerade (c1)	2b	192.168.2.100	192.168.2.101
Gleichmäßig (h2)	i2a	192.168.4.10	Ungerade (c1)	2b	192.168.4.100	192.168.4.101
Gleichmäßig (h2)	i3a	192.168.6.10	Gleichmäßig (c2)	2b	192.168.6.100	192.168.6.101
Gleichmäßig (h2)	I4a	192.168.8.10	Gleichmäßig (c2)	2b	192.168.8.100	192.168.8.101

IPoIB-Adressierungsschemata für BeeGFS-Client-Server

Auf jedem Datei-Node werden mehrere BeeGFS-Serverservices ausgeführt (Management, Metadaten oder Storage). Damit jeder Service unabhängig vom anderen Datei-Node ein Failover durchführen kann, verfügt jeder über eindeutige IP-Adressen, die zwischen beiden Nodes schweben können (auch als logische Schnittstelle oder LIF bezeichnet).

Diese Bereitstellung setzt zwar nicht zwingend voraus, dass für diese Verbindungen folgende IPoIB-Subnetzbereiche verwendet werden und definiert ein Standard-Adressierungsschema, das folgende Regeln anwendet:

- Das zweite Oktett ist immer ungerade oder sogar, basierend darauf, ob der InfiniBand-Port des Datei-Nodes ungerade oder sogar ungerade ist.
- BeeGFS Cluster-IPs sind immer `xxx. 127.100.yyy` Oder `xxx.128.100.yyy`.

> Zusätzlich zur Schnittstelle, die für die bandinterne Betriebssystemverwaltung verwendet wird, können zusätzliche Schnittstellen von Corosync für Cluster Heart-Schläge und Synchronisation verwendet werden. So wird sichergestellt, dass der Verlust einer einzelnen Schnittstelle das gesamte Cluster nicht in den Fall bringt.

- Der BeeGFS Management Service ist immer im Betrieb `xxx.yyy.101.0` Oder `xxx.yyy.102.0`.
- BeeGFS Metadatendienste sind immer dabei `xxx.yyy.101.zzz` Oder `xxx.yyy.102.zzz`.
- BeeGFS Storage-Services finden sich immer bei `xxx.yyy.103.zzz` oder `xxx.yyy.104.zzz`.
- Adressen im Bereich `100.xxx.1.1` Bis `100.xxx.99.255` Sind für Kunden reserviert.

IPoIB-Adressierungsschema für ein einzelnes Subnetz

In diesem Bereitstellungshandbuch wird ein einziges Subnetz-Schema verwendet, da die in aufgeführten Vorteile im aufgeführt "Softwarearchitektur"sind.

Subnetz: 100.127.0.0/16

Die folgende Tabelle enthält den Bereich für ein einzelnes Subnetz: 100.127.0.0/16.

Zweck	InfiniBand-Port	IP-Adresse oder Bereich
BeeGFS Cluster-IP	i1b oder i4b	100.127.100.1 - 100.127.100.255
BeeGFS Management	i1b	100.127.101.0
	i2b	100.127.102.0
BeeGFS-Metadaten	i1b oder i3b	100.127.101.1 - 100.127.101.255
	i2b oder i4b	100.127.102.1 - 100.127.102.255
BeeGFS-Speicherung	i1b oder i3b	100.127.103.1 - 100.127.103.255
	i2b oder i4b	100.127.104.1 - 100.127.104.255
BeeGFS-Clients	(Je nach Kunde)	100.127.1.1 - 100.127.99.255

IPoIB zwei Subnetz-Adressierungsschema

Ein zwei-Subnetz-Adressierungsschema wird nicht mehr empfohlen, kann aber trotzdem implementiert werden. In den folgenden Tabellen finden Sie ein empfohlenes zwei-Subnetz-Schema.

Subnetz A: 100.127.0.0/16

In der folgenden Tabelle ist der Bereich für Subnetz A angegeben: 100.127.0.0/16.

Zweck	InfiniBand-Port	IP-Adresse oder Bereich
BeeGFS Cluster-IP	i1b	100.127.100.1 - 100.127.100.255
BeeGFS Management	i1b	100.127.101.0
BeeGFS-Metadaten	i1b oder i3b	100.127.101.1 - 100.127.101.255
BeeGFS-Speicherung	i1b oder i3b	100.127.103.1 - 100.127.103.255
BeeGFS-Clients	(Je nach Kunde)	100.127.1.1 - 100.127.99.255

Subnetz B: 100.128.0.0/16

In der folgenden Tabelle ist der Bereich für Subnetz B angegeben: 100.128.0.0/16.

Zweck	InfiniBand-Port	IP-Adresse oder Bereich
BeeGFS Cluster-IP	I4b	100.128.100.1 - 100.128.100.255
BeeGFS Management	i2b	100.128.102.0
BeeGFS-Metadaten	i2b oder i4b	100.128.102.1 - 100.128.102.255
BeeGFS-Speicherung	i2b oder i4b	100.128.104.1 - 100.128.104.255
BeeGFS-Clients	(Je nach Kunde)	100.128.1.1 - 100.128.99.255

In dieser NetApp Verified Architecture werden nicht alle IPs in den oben genannten Bereichen verwendet. Sie zeigen, wie IP-Adressen vorzugewiesen werden können, um eine einfache Erweiterung des Dateisystems mit einem konsistenten IP-Adressierungschema zu ermöglichen. In diesem Schema entsprechen BeeGFS-Datei-Knoten und Service-IDs dem vierten Oktett eines bekannten IP-Bereichs. Das Filesystem konnte bei Bedarf auf jeden Fall über 255 Nodes oder Services skaliert werden.

Implementierung von Hardware

Jeder Baustein besteht aus zwei validierten x86-Datei-Nodes, die mithilfe von HDR-Kabeln (200 GB) direkt mit zwei Block-Nodes verbunden sind.

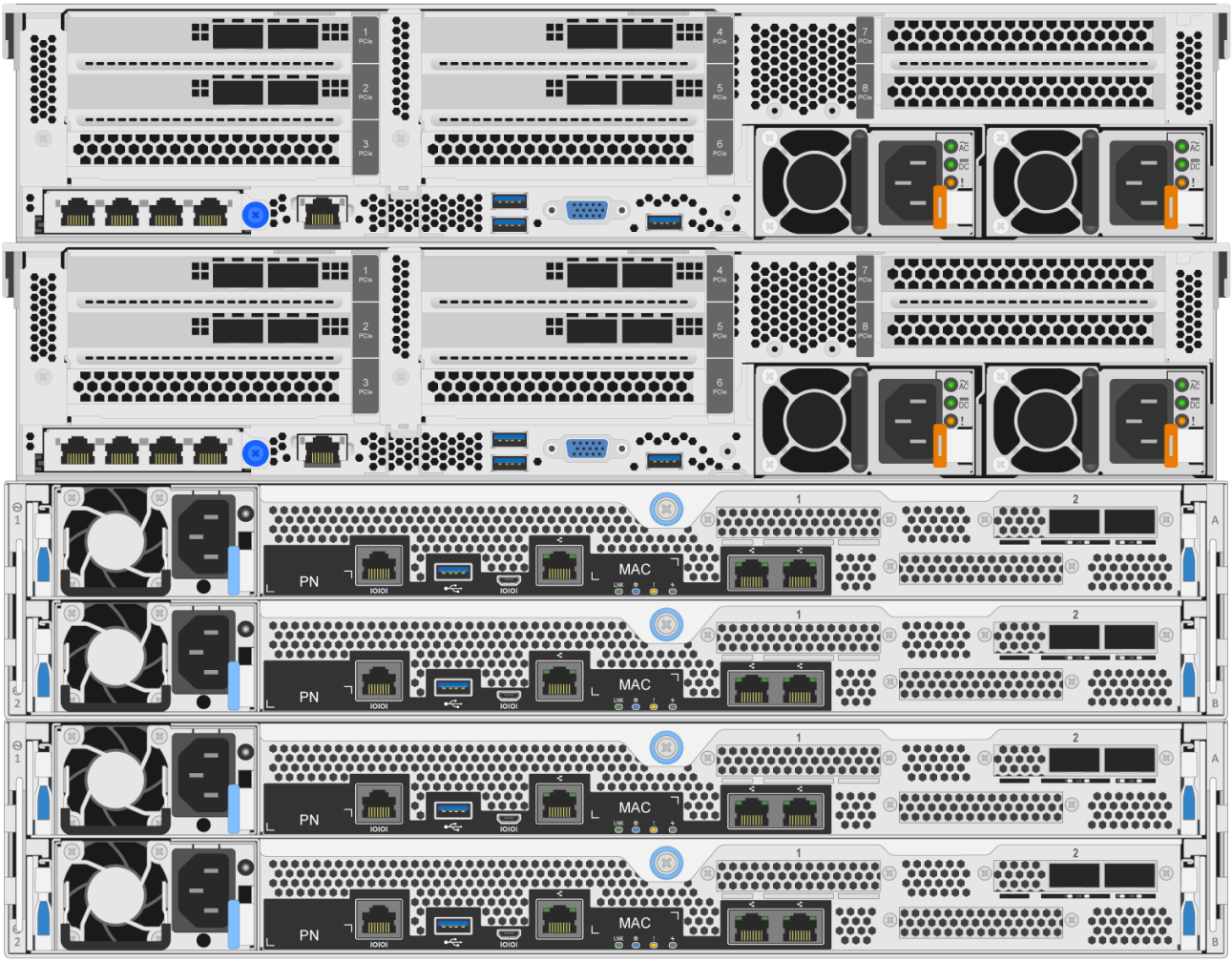
> Zum Herstellen von Quorum im Failover Cluster sind mindestens zwei Bausteine erforderlich. Ein Cluster mit zwei Nodes hat Einschränkungen, die ein erfolgreiches Failover verhindern können. Wenn Sie ein Cluster mit zwei Nodes konfigurieren, wird ein drittes Gerät als Tiebreaker integriert, dieses Design wird jedoch nicht in dieser Dokumentation beschrieben.

Die folgenden Schritte sind für jeden Baustein im Cluster identisch, unabhängig davon, ob er sowohl für die Ausführung von BeeGFS-Metadaten- und Storage-Services als auch nur für Storage-Services eingesetzt wird, sofern nicht anders angegeben.

Schritte

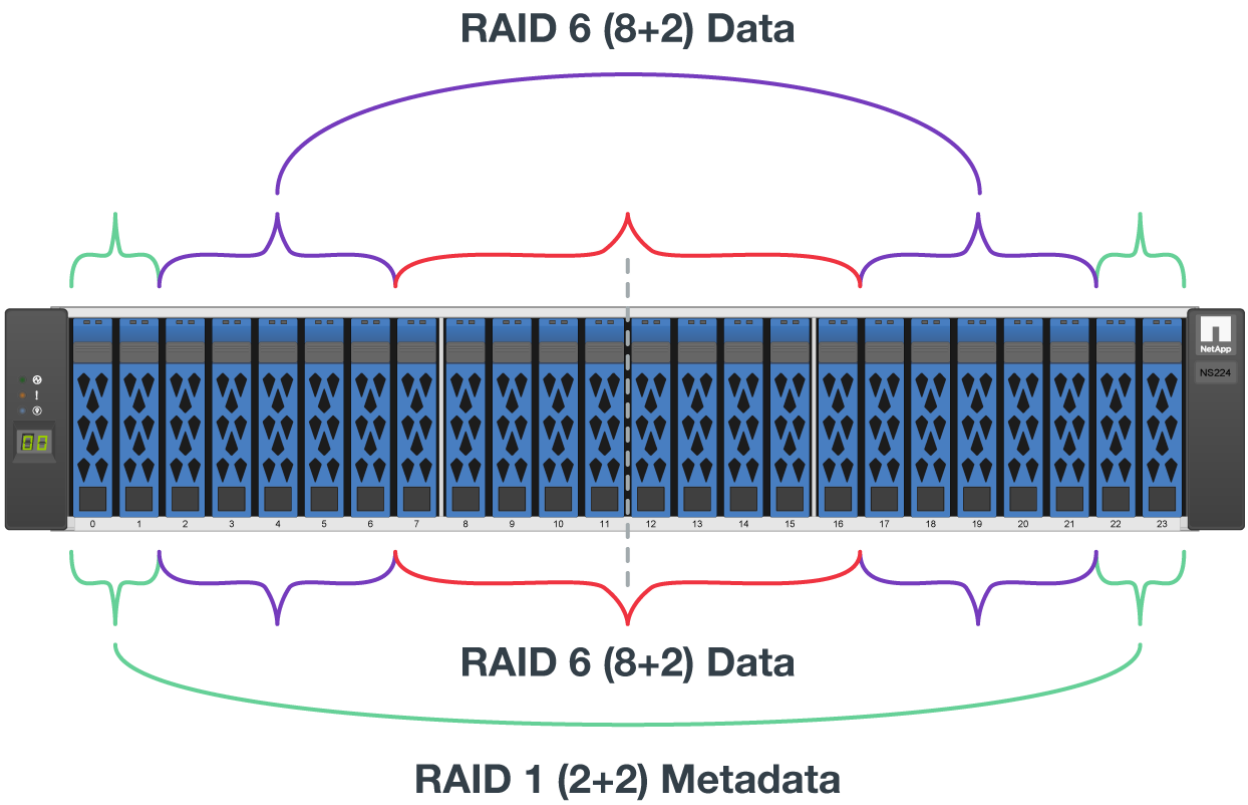
1. Richten Sie jeden BeeGFS-Dateiknoten mit vier Host-Channel-Adaptern (HCAs) mithilfe der in der angegebenen Modelle "Technische Anforderungen"ein. Legen Sie die HCAs gemäß den folgenden Spezifikationen in die PCIe-Steckplätze des Dateiknotens ein:
   - * Lenovo ThinkSystem SR665 V3 Server:* Verwenden Sie die PCIe-Steckplätze 1, 2, 4 und 5.
   - * Lenovo ThinkSystem SR665 Server:* Verwenden Sie die PCIe-Steckplätze 2, 3, 5 und 6.
2. Konfigurieren Sie jeden BeeGFS-Block-Node mit einer 200-GB-Host-Schnittstellenkarte (HIC) mit zwei Ports, und installieren Sie die HIC in jedem ihrer beiden Storage Controller.

   Stellen Sie die Bausteine so ein, dass die beiden BeeGFS-Datei-Nodes über den BeeGFS-Block-Nodes liegen. Die folgende Abbildung zeigt die richtige Hardwarekonfiguration für den BeeGFS-Baustein, der Lenovo ThinkSystem SR665 V3-Server als Dateiknoten verwendet (Rückansicht).

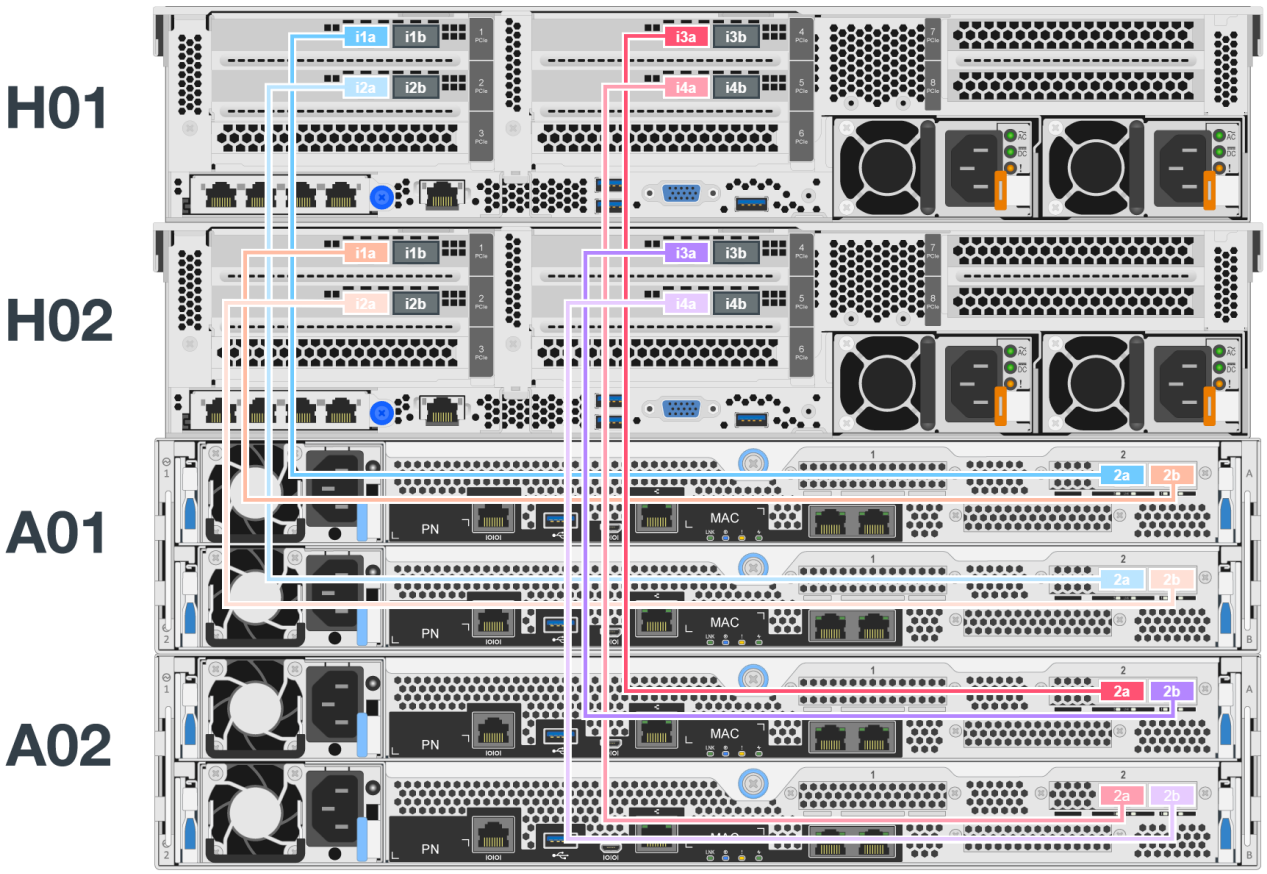


Die Konfiguration der Stromversorgung für Produktionsanwendungsfälle sollte in der Regel redundante Netzteile verwenden.

3. Installieren Sie bei Bedarf die Laufwerke in jedem BeeGFS-Block-Knoten.
   a. Wenn der Baustein zur Ausführung von BeeGFS-Metadaten und Speicherdiensten verwendet wird und kleinere Laufwerke für Metadaten-Volumes verwendet werden, vergewissern Sie sich, dass diese in den äußeren Laufwerksschächten gefüllt sind, wie in der Abbildung unten gezeigt.
   b. Wenn ein Laufwerkgehäuse nicht vollständig bestückt ist, stellen Sie bei allen Bausteinkonfigurationen sicher, dass eine gleiche Anzahl an Laufwerken in den Steckplätzen 0–11 und 12–23 gefüllt ist, um eine optimale Performance zu erzielen.



4. Verbinden Sie die Block- und File-Knoten mit dem "1 m InfiniBand HDR 200 GB Direct-Attach-Kupferkabel", so dass sie mit der in der folgenden Abbildung gezeigten Topologie übereinstimmen.

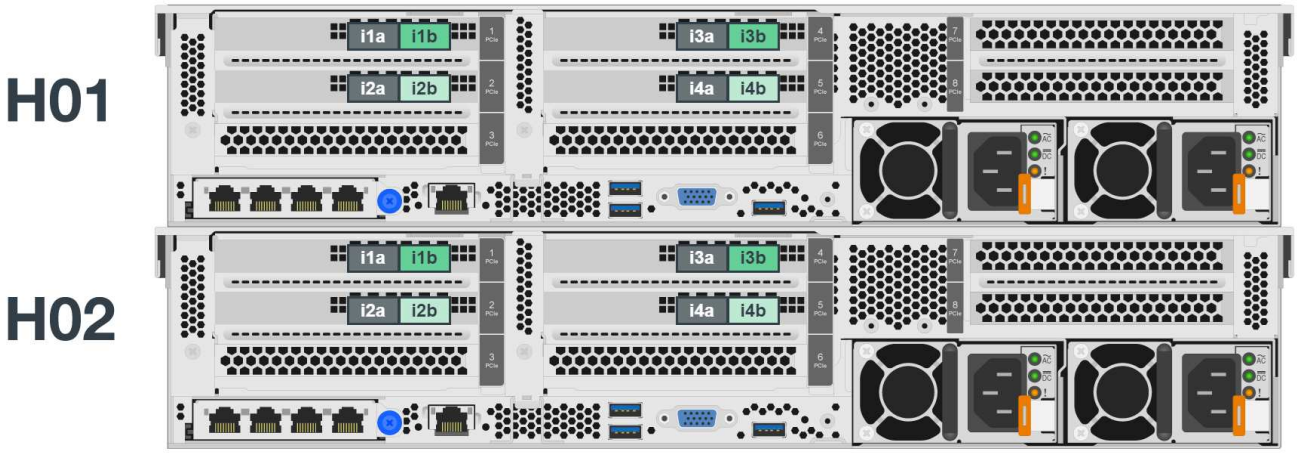


Die Nodes sind über mehrere Bausteine hinweg nie direkt miteinander verbunden. Jeder Baustein sollte als eigenständige Einheit behandelt werden und alle Kommunikation zwischen Bausteinen erfolgt über Netzwerk-Switches.

5. Verbinden Sie die übrigen InfiniBand-Ports auf dem Datei-Node mithilfe des spezifischen InfiniBand-Speicherswitters des Speichernetzwerks mit dem "2 m InfiniBand-Kabel" InfiniBand-Switch.

   Wenn Sie Splitterkabel verwenden, um den Speicher-Switch mit Dateiknoten zu verbinden, sollte ein Kabel vom Switch abzweigen und mit den hellgrünen Ports verbunden werden. Ein anderes Splitterkabel sollte sich vom Switch abzweigen und an die dunkelgrünen Ports anschließen.

   Außerdem sollten bei Speichernetzwerken mit redundanten Switches die hellgrünen Ports mit einem Switch verbunden werden, während dunkelgrüne Ports mit einem anderen Switch verbunden sein sollten.



6. Montieren Sie bei Bedarf weitere Bausteine gemäß den gleichen Verkabelungsrichtlinien.

Die Gesamtzahl der Bausteine, die in einem einzigen Rack implementiert werden können, hängt von der verfügbaren Stromversorgung und Kühlung an jedem Standort ab.

Implementierung von Software

Einrichten von Datei-Nodes und Block-Nodes

Während die meisten Software-Konfigurationsaufgaben mithilfe der von NetApp zur Verfügung gestellten Ansible Sammlungen automatisiert werden, müssen Sie das Netzwerk auf dem Baseboard Management Controller (BMC) jedes Servers konfigurieren und den Management-Port auf jedem Controller konfigurieren.

Richten Sie die Datei-Nodes ein

1. Konfigurieren Sie das Netzwerk auf dem Baseboard Management Controller (BMC) jedes Servers.

   Informationen zum Konfigurieren der Netzwerkkonfiguration für die validierten Lenovo SR665 V3-Dateiknoten finden Sie unter "Lenovo ThinkSystem Dokumentation".

Ein Baseboard Management Controller (BMC), der manchmal als Service-Prozessor bezeichnet wird, ist der generische Name für die Out-of-Band-Management-Funktion, die in verschiedenen Server-Plattformen integriert ist, die Remote-Zugriff bieten können, selbst wenn das Betriebssystem nicht installiert ist oder nicht zugänglich ist. Anbieter vermarkten diese Funktionalität in der Regel mit ihrem eigenen Branding. Auf dem Lenovo SR665 wird beispielsweise der BMC als *Lenovo XClarity Controller (XCC)* bezeichnet.

2. Konfigurieren Sie die Systemeinstellungen für maximale Performance.

   Sie konfigurieren die Systemeinstellungen über das UEFI-Setup (früher BIOS) oder über die Redfish APIs, die von vielen BMCs bereitgestellt werden. Die Systemeinstellungen variieren je nach Servermodell, das als Dateiknoten verwendet wird.

   Informationen zum Konfigurieren der Systemeinstellungen für die validierten Lenovo SR665 V3-Dateiknoten finden Sie unter "Passen Sie die Systemeinstellungen für die Performance an" .

3. Installieren Sie Red Hat Enterprise Linux (RHEL) 9.4 und konfigurieren Sie den Hostnamen und den Netzwerkport, die zur Verwaltung des Betriebssystems verwendet werden, einschließlich der SSH-Konnektivität vom Ansible-Steuerknoten.

   Konfigurieren Sie derzeit keine IPs auf einem der InfiniBand-Ports.

   Die nachfolgenden Abschnitte gehen davon aus, dass die Hostnamen sequenziell nummeriert sind (z. B. h1-HN), und beziehen sich auf Aufgaben, die auf ungeraden oder gar nummerierten Hosts ausgeführt werden sollten.

4. Verwenden Sie den Red Hat Subscription Manager, um das System zu registrieren und zu abonnieren, damit die erforderlichen Pakete aus den offiziellen Red Hat-Repositorys installiert werden können und um Updates auf die unterstützte Version von Red Hat zu beschränken: `subscription-manager release --set=9.4` . Anweisungen hierzu finden Sie unter "Registrieren und Abonnieren eines RHEL Systems" und "Einschränken von Aktualisierungen".
5. Aktivieren Sie das Red hat Repository mit den für hohe Verfügbarkeit erforderlichen Paketen.

   ```
   subscription-manager repo-override --repo=rhel-9-for-x86_64
   -highavailability-rpms --add=enabled:1
   ```

6. Aktualisieren Sie alle HCA-Firmware auf die in Verwendung des "Aktualisiert die Datei-Node-Adapter-Firmware"Handbuchs empfohlene Version"Technologieanforderungen erfüllt".

Richten Sie Block-Nodes ein

Richten Sie die EF600 Block-Nodes ein, indem Sie den Managementport pro Controller konfigurieren.

1. Konfigurieren Sie den Managementport an jedem EF600 Controller.

   Anweisungen zum Konfigurieren von Ports finden Sie im "E-Series Documentation Center".

2. Legen Sie optional den Speicher-Array-Namen für jedes System fest.

   Durch das Festlegen eines Namens kann es einfacher sein, in den nachfolgenden Abschnitten auf jedes System zu verweisen. Anweisungen zum Festlegen des Arraynamens finden Sie im "E-Series Documentation Center".

Folgende Themen setzen voraus, dass die Namen der Speicherarrays nacheinander nummeriert sind (z. B. c1 - CN), und beziehen sich auf die Schritte, die auf ungeraden und nicht gerade nummerierten Systemen ausgeführt werden sollten.

Optimieren Sie die System-Einstellungen des File Node für die Performance

Um die Leistung zu maximieren, empfehlen wir, die Systemeinstellungen auf dem Servermodell zu konfigurieren, das Sie als Dateiknoten verwenden.

Die Systemeinstellungen variieren je nach Servermodell, das Sie als Dateiknoten verwenden. In diesem Thema wird beschrieben, wie die Systemeinstellungen für die validierten Lenovo ThinkSystem SR665-Serverdateiknoten konfiguriert werden.

Über die UEFI-Schnittstelle können Sie die Systemeinstellungen anpassen

Die System-Firmware des Lenovo SR665 V3-Servers enthält zahlreiche Tuning-Parameter, die über die UEFI-Schnittstelle eingestellt werden können. Diese Optimierungsparameter können sich auf alle Aspekte der Serverfunktionen und die Leistung des Servers auswirken.

Passen Sie unter **UEFI Setup > Systemeinstellungen** die folgenden Systemeinstellungen an:

Menü „Betriebsmodus“

Systemeinstellung	Wechseln Sie zu
Betriebsmodus	Individuell
CTDP	Manuell
CTDP-Handbuch	350
Maximale Leistung Des Pakets	Manuell
Effizienzmodus	Deaktivieren
Global-Cstate-Control	Deaktivieren
SOC P-Staaten	P0
DF-C-Staaten	Deaktivieren
P-Zustand	Deaktivieren
Speicherabschaltstrom Aktivieren	Deaktivieren
NUMA-Knoten pro Socket	NPS1

Menü „Geräte“ und „E/A-Anschlüsse“

Systemeinstellung	Wechseln Sie zu
IOMMU	Deaktivieren

Ein-/aus-Menü

Systemeinstellung	Wechseln Sie zu
PCIe-Power-Brake	Deaktivieren

Menü „Prozessoren“

Systemeinstellung	Wechseln Sie zu
Global C-State Control	Deaktivieren
DF-C-Staaten	Deaktivieren
SMT-Modus	Deaktivieren
CPPC	Deaktivieren

Verwenden Sie die Redfish-API, um die Systemeinstellungen anzupassen

Zusätzlich zur Verwendung von UEFI Setup können Sie die Redfish API verwenden, um Systemeinstellungen zu ändern.

```
curl --request PATCH \
  --url https://<BMC_IP_ADDRESS>/redfish/v1/Systems/1/Bios/Pending \
  --user <BMC_USER>:<BMC- PASSWORD> \
  --header 'Content-Type: application/json' \
  --data '{
"Attributes": {
"OperatingModes_ChooseOperatingMode": "CustomMode",
"Processors_cTDP": "Manual",
"Processors_PackagePowerLimit": "Manual",
"Power_EfficiencyMode": "Disable",
"Processors_GlobalC_stateControl": "Disable",
"Processors_SOCP_states": "P0",
"Processors_DFC_States": "Disable",
"Processors_P_State": "Disable",
"Memory_MemoryPowerDownEnable": "Disable",
"DevicesandIOPorts_IOMMU": "Disable",
"Power_PCIePowerBrake": "Disable",
"Processors_GlobalC_stateControl": "Disable",
"Processors_DFC_States": "Disable",
"Processors_SMTMode": "Disable",
"Processors_CPPC": "Disable",
"Memory_NUMANodesperSocket":"NPS1"
}
}
'
```

Ausführliche Informationen zum Schema Redfish finden Sie im "DMTF-Website".

Richten Sie einen Ansible-Steuerungsknoten ein

Zum Einrichten eines Ansible-Steuerknotens müssen Sie eine virtuelle oder physische Maschine mit Netzwerkzugriff auf alle Datei- und Block-Nodes zuweisen, die für die BeeGFS auf NetApp Lösung implementiert werden.

Eine Liste der empfohlenen Paketversionen finden Sie im"Technische Anforderungen". Die folgenden Schritte wurden auf Ubuntu 22.04 getestet. Für spezifische Schritte zu Ihrer bevorzugten Linux-Distribution, siehe "Ansible-Dokumentation".

1. Installieren Sie über Ihren Ansible Control Node die folgenden Pakete für Python und Python Virtual Environment.

   ```
   sudo apt-get install python3 python3-pip python3-setuptools python3.10-venv
   ```

2. Erstellen Sie eine virtuelle Python-Umgebung.

```
python3 -m venv ~/pyenv
```

3. Aktivieren Sie die virtuelle Umgebung.

```
source ~/pyenv/bin/activate
```

4. Installieren Sie die erforderlichen Python-Pakete in der aktivierten virtuellen Umgebung.

```
pip install ansible netaddr cryptography passlib
```

5. Installieren Sie die BeeGFS-Sammlung mit Ansible Galaxy.

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

6. Überprüfen Sie, ob die installierten Versionen von Ansible, Python und BeeGFS-Sammlung mit der übereinstimmen"Technische Anforderungen".

```
ansible --version
ansible-galaxy collection list netapp_eseries.beegfs
```

7. Richten Sie passwortloses SSH ein, damit Ansible vom Ansible-Steuerungsknoten aus auf die Remote-BeeGFS-Datei-Nodes zugreifen kann.
   a. Generieren Sie auf dem Ansible-Steuerungsknoten, falls erforderlich, ein Paar öffentlicher Schlüssel.

```
ssh-keygen
```

   b. Richten Sie passwortloses SSH für jeden der Dateiknoten ein.

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

Richten Sie bei den Blockknoten eine passwortlose SSH ein. Dies wird weder unterstützt noch erforderlich.

Erstellen des Ansible-Inventars

Um die Konfiguration für Datei- und Block-Nodes zu definieren, erstellen Sie einen Ansible-Bestand für das BeeGFS-Dateisystem, das bereitgestellt werden soll. Der Bestand umfasst Hosts, Gruppen und Variablen, die das gewünschte BeeGFS-Dateisystem beschreiben.

Schritt 1: Konfiguration für alle Bausteine definieren

Legen Sie die Konfiguration fest, die für alle Bausteine gilt, unabhängig davon, welches Konfigurationsprofil Sie für sie einzeln anwenden können.

Bevor Sie beginnen

- Wählen Sie ein Subnetz-Adressierungsschema für Ihre Bereitstellung aus. Aufgrund der im aufgeführten Vorteile "Softwarearchitektur"wird empfohlen, ein einziges Subnetz-Adressierungsschema zu verwenden.

Schritte

1. Geben Sie auf dem Ansible-Steuerungsknoten ein Verzeichnis an, das Sie zum Speichern der Bestands- und Playbook-Dateien in Ansible verwenden möchten.

   Sofern nicht anders angegeben, werden alle in diesem Schritt erstellten Dateien und Verzeichnisse und die folgenden Schritte relativ zu diesem Verzeichnis erstellt.

2. Folgende Unterverzeichnisse erstellen:

   `host_vars`

   `group_vars`

   `packages`

3. Erstellen Sie ein Unterverzeichnis für Cluster-Passwörter und sichern Sie die Datei durch Verschlüsselung mit Ansible Vault (siehe "Verschlüsseln von Inhalten mit Ansible Vault"):

   a. Erstellen Sie das Unterverzeichnis `group_vars/all`.

   b. Erstellen Sie im `group_vars/all` Verzeichnis eine Passwortdatei mitder Bezeichnung `passwords.yml`.

   c. Füllen Sie den `passwords.yml file` mit den folgenden Angaben aus, und ersetzen Sie alle Benutzernamen- und Kennwortparameter entsprechend Ihrer Konfiguration:

```
# Credentials for storage system's admin password
eseries_password: <PASSWORD>

# Credentials for BeeGFS file nodes
ssh_ha_user: <USERNAME>
ssh_ha_become_pass: <PASSWORD>

# Credentials for HA cluster
ha_cluster_username: <USERNAME>
ha_cluster_password: <PASSWORD>
ha_cluster_password_sha512_salt: randomSalt

# Credentials for fencing agents
# OPTION 1: If using APC Power Distribution Units (PDUs) for fencing:
# Credentials for APC PDUs.
apc_username: <USERNAME>
apc_password: <PASSWORD>

# OPTION 2: If using the Redfish APIs provided by the Lenovo XCC (and
other BMCs) for fencing:
# Credentials for XCC/BMC of BeeGFS file nodes
bmc_username: <USERNAME>
bmc_password: <PASSWORD>
```

d. Führen Sie aus `ansible-vault encrypt passwords.yml`, und legen Sie ein Vault-Kennwort fest, wenn Sie dazu aufgefordert werden.

Schritt: Konfiguration für einzelne Datei- und Block-Nodes definieren

Legen Sie die Konfiguration für einzelne Datei-Nodes und einzelne Baustein-Nodes fest.

1. Unter `host_vars/`Erstellen Sie für jeden BeeGFS-Dateiknoten eine Datei mit dem Namen `<HOSTNAME>.yml` Mit dem folgenden Inhalt, besondere Aufmerksamkeit auf die Notizen über den Inhalt für BeeGFS Cluster-IPs und Host-Namen enden in ungerade oder gerade Zahlen.

   Zunächst stimmen die Schnittstellennamen der Dateiknoten mit dem überein, was hier aufgeführt ist (z. B. ib0 oder ibs1f0). Diese benutzerdefinierten Namen werden in konfiguriert die für alle Datei-Knoten gelten soll.

```
ansible_host: “<MANAGEMENT_IP>”
eseries_ipoib_interfaces:  # Used to configure BeeGFS cluster IP
addresses.
  - name: i1b
    address: 100.127.100. <NUMBER_FROM_HOSTNAME>/16
  - name: i4b
    address: 100.127.100. <NUMBER_FROM_HOSTNAME>/16
beegfs_ha_cluster_node_ips:
  - <MANAGEMENT_IP>
  - <i1b_BEEGFS_CLUSTER_IP>
  - <i4b_BEEGFS_CLUSTER_IP>
# NVMe over InfiniBand storage communication protocol information
# For odd numbered file nodes (i.e., h01, h03, ..):
eseries_nvme_ib_interfaces:
  - name: i1a
    address: 192.168.1.10/24
    configure: true
  - name: i2a
    address: 192.168.3.10/24
    configure: true
  - name: i3a
    address: 192.168.5.10/24
    configure: true
  - name: i4a
    address: 192.168.7.10/24
    configure: true
# For even numbered file nodes (i.e., h02, h04, ..):
# NVMe over InfiniBand storage communication protocol information
eseries_nvme_ib_interfaces:
  - name: i1a
    address: 192.168.2.10/24
    configure: true
  - name: i2a
    address: 192.168.4.10/24
    configure: true
  - name: i3a
    address: 192.168.6.10/24
    configure: true
  - name: i4a
    address: 192.168.8.10/24
    configure: true
```

Wenn Sie bereits das BeeGFS-Cluster implementiert haben, müssen Sie das Cluster beenden, bevor Sie statisch konfigurierte IP-Adressen hinzufügen oder ändern, einschließlich Cluster-IPs und IPs für NVMe/IB. Dies ist erforderlich, damit diese Änderungen ordnungsgemäß wirksam werden und Cluster-Vorgänge nicht unterbrechen.

2. Unter `host_vars/`Erstellen Sie für jeden BeeGFS-Block-Knoten eine Datei mit dem Namen `<HOSTNAME>.yml` Und geben Sie den folgenden Inhalt ein.

   Achten Sie besonders auf die Hinweise zum Inhalt, die für Speicher-Array-Namen ausgefüllt werden müssen, die mit ungeraden oder geraden Zahlen enden.

   Erstellen Sie für jeden Block-Node eine Datei, und geben Sie den an `<MANAGEMENT_IP>` Für einen der beiden Controller (normalerweise A).

```
eseries_system_name: <STORAGE_ARRAY_NAME>
eseries_system_api_url: https://<MANAGEMENT_IP>:8443/devmgr/v2/
eseries_initiator_protocol: nvme_ib
# For odd numbered block nodes (i.e., a01, a03, ..):
eseries_controller_nvme_ib_port:
  controller_a:
    - 192.168.1.101
    - 192.168.2.101
    - 192.168.1.100
    - 192.168.2.100
  controller_b:
    - 192.168.3.101
    - 192.168.4.101
    - 192.168.3.100
    - 192.168.4.100
# For even numbered block nodes (i.e., a02, a04, ..):
eseries_controller_nvme_ib_port:
  controller_a:
    - 192.168.5.101
    - 192.168.6.101
    - 192.168.5.100
    - 192.168.6.100
  controller_b:
    - 192.168.7.101
    - 192.168.8.101
    - 192.168.7.100
    - 192.168.8.100
```

Schritt 3: Definieren Sie die Konfiguration, die für alle Datei- und Block-Nodes gelten soll

Unter können Sie die gemeinsame Konfiguration für eine Gruppe von Hosts definieren `group_vars` In einem Dateinamen, der der Gruppe entspricht. Dadurch wird verhindert, dass eine gemeinsame Konfiguration an mehreren Orten wiederholt wird.

Über diese Aufgabe

Hosts können sich in mehr als einer Gruppe befinden. Ansible zur Laufzeit wählt Ansible aus, welche Variablen auf Basis seiner variablen Rangfolge für einen bestimmten Host gelten. (Weitere Informationen zu diesen Regeln finden Sie in der Ansible-Dokumentation für "Variablen verwenden".)

Host-zu-Gruppe-Zuweisungen werden in der tatsächlichen Ansible-Bestandsdatei definiert, die gegen Ende dieses Vorgangs erstellt wird.

Schritt

In Ansible können alle Konfigurationen, die auf alle Hosts angewendet werden sollen, in einer Gruppe mit dem Namen definiert werden `All`. Erstellen Sie die Datei `group_vars/all.yml` Mit folgenden Inhalten:

```
ansible_python_interpreter: /usr/bin/python3
beegfs_ha_ntp_server_pools:  # Modify the NTP server addressess if
desired.
  - "pool 0.pool.ntp.org iburst maxsources 3"
  - "pool 1.pool.ntp.org iburst maxsources 3"
```

Schritt 4: Definieren Sie die Konfiguration, die für alle Datei-Knoten gelten soll

Die gemeinsame Konfiguration für Dateiknoten ist in einer Gruppe mit dem Namen definiert `ha_cluster`. In den Schritten in diesem Abschnitt wird die Konfiguration erstellt, die in der enthalten sein sollte `group_vars/ha_cluster.yml` Datei:

Schritte

1. Legen Sie oben in der Datei die Standardeinstellungen fest, einschließlich des Kennworts, das als verwendet werden soll `sudo` Benutzer auf den Datei-Nodes.

```
### ha_cluster Ansible group inventory file.
# Place all default/common variables for BeeGFS HA cluster resources
below.
### Cluster node defaults
ansible_ssh_user: {{ ssh_ha_user }}
ansible_become_password: {{ ssh_ha_become_pass }}
eseries_ipoib_default_hook_templates:
  - 99-multihoming.j2   # This is required for single subnet
deployments, where static IPs containing multiple IB ports are in the
same IPoIB subnet. i.e: cluster IPs, multirail, single subnet, etc.
# If the following options are specified, then Ansible will
automatically reboot nodes when necessary for changes to take effect:
eseries_common_allow_host_reboot: true
eseries_common_reboot_test_command: "! systemctl status
eseries_nvme_ib.service || systemctl --state=exited | grep
eseries_nvme_ib.service"
eseries_ib_opensm_options:
  virt_enabled: "2"
  virt_max_ports_in_process: "0"
```

> Wenn der `ansible_ssh_user` bereits ist `root`, können Sie optional die auslassen und beim Ausführen des Playbook die `ansible_become_password` Option angeben `--ask -become-pass`.

2. Konfigurieren Sie optional einen Namen für den Hochverfügbarkeits-Cluster und geben Sie einen Benutzer für die Cluster-interne Kommunikation an.

   Wenn Sie das private IP-Adressschema ändern, müssen Sie auch die Standardeinstellung aktualisieren `beegfs_ha_mgmtd_floating_ip`. Dies muss mit dem übereinstimmen, was Sie später für die BeeGFS Management Ressourcengruppe konfigurieren.

   Geben Sie eine oder mehrere E-Mails an, die Warnmeldungen für Cluster-Ereignisse mit empfangen sollen `beegfs_ha_alert_email_list`.

```
### Cluster information
beegfs_ha_firewall_configure: True
eseries_beegfs_ha_disable_selinux: True
eseries_selinux_state: disabled
# The following variables should be adjusted depending on the desired
configuration:
beegfs_ha_cluster_name: hacluster                    # BeeGFS HA cluster
name.
beegfs_ha_cluster_username: "{{ ha_cluster_username }}" # Parameter for
BeeGFS HA cluster username in the passwords file.
beegfs_ha_cluster_password: "{{ ha_cluster_password }}" # Parameter for
BeeGFS HA cluster username's password in the passwords file.
beegfs_ha_cluster_password_sha512_salt: "{{
ha_cluster_password_sha512_salt }}" # Parameter for BeeGFS HA cluster
username's password salt in the passwords file.
beegfs_ha_mgmtd_floating_ip: 100.127.101.0          # BeeGFS management
service IP address.
# Email Alerts Configuration
beegfs_ha_enable_alerts: True
beegfs_ha_alert_email_list: ["email@example.com"]  # E-mail recipient
list for notifications when BeeGFS HA resources change or fail.  Often a
distribution list for the team responsible for managing the cluster.
beegfs_ha_alert_conf_ha_group_options:
      mydomain: "example.com"
# The mydomain parameter specifies the local internet domain name. This
is optional when the cluster nodes have fully qualified hostnames (i.e.
host.example.com).
# Adjusting the following parameters is optional:
beegfs_ha_alert_timestamp_format: "%Y-%m-%d %H:%M:%S.%N" #%H:%M:%S.%N
beegfs_ha_alert_verbosity: 3
#  1) high-level node activity
#  3) high-level node activity + fencing action information + resources
(filter on X-monitor)
#  5) high-level node activity + fencing action information + resources
```

Während scheinbar redundant, `beegfs_ha_mgmtd_floating_ip` Ist wichtig, wenn Sie das BeeGFS-Dateisystem über einen einzelnen HA-Cluster hinaus skalieren. Nachfolgende HA-Cluster werden ohne zusätzlichen BeeGFS-Managementservice bereitgestellt und Punkt am Managementservice des ersten Clusters.

3. Konfigurieren Sie einen Fechtagenten. (Weitere Informationen finden Sie unter "Konfigurieren Sie Fechten in einem Red hat High Availability Cluster".) Die folgende Ausgabe zeigt Beispiele für die Konfiguration gängiger Fencing-Agenten. Wählen Sie eine dieser Optionen.

   Beachten Sie bei diesem Schritt Folgendes:

- Standardmäßig ist Fechten aktiviert, Sie müssen jedoch einen Fechten_Agent_ konfigurieren.
- Der `<HOSTNAME>` Angegeben in `pcmk_host_map` Oder `pcmk_host_list` Der Hostname in der Ansible-Bestandsaufnahme entspricht.
- Das BeeGFS-Cluster ohne Fencing wird insbesondere in der Produktion nicht unterstützt. Dies soll weitgehend sicherstellen, wenn BeeGFS-Services, einschließlich aller Ressourcenabhängigkeiten wie Blockgeräte, Failover aufgrund eines Problems durchführen, es besteht keine Möglichkeit des gleichzeitigen Zugriffs durch mehrere Nodes, die zu einer Beschädigung des Filesystems oder anderen unerwünschten oder unerwarteten Verhalten führen. Wenn das Fechten deaktiviert werden muss, lesen Sie die allgemeinen Hinweise in der BeeGFS HA-Rolle „erste Schritte“-Anleitung und „Set“ `beegfs_ha_cluster_crm_config_options["stonith-enabled"]` Mit FALSE innen `ha_cluster.yml`.
- Es sind mehrere Fechtgeräte auf Node-Ebene verfügbar, und die BeeGFS HA-Rolle kann jeden Fechtagenten konfigurieren, der im Red hat HA Package Repository verfügbar ist. Wenn möglich, verwenden Sie einen Zaunsagenten, der über die unterbrechungsfreie Stromversorgung (USV) oder die Rack-Stromverteilereinheit (rPDU) arbeitet. Da einige Fechten-Agenten wie der Baseboard-Management-Controller (BMC) oder andere Lights-Out-Geräte, die in den Server integriert sind, möglicherweise nicht auf die Zaunanforderung unter bestimmten Ausfallszenarien reagieren.

```
### Fencing configuration:
# OPTION 1: To enable fencing using APC Power Distribution Units
(PDUs):
beegfs_ha_fencing_agents:
 fence_apc:
   - ipaddr: <PDU_IP_ADDRESS>
     login: "{{ apc_username }}" # Parameter for APC PDU username in
the passwords file.
     passwd: "{{ apc_password }}" # Parameter for APC PDU password in
the passwords file.
     pcmk_host_map:
"<HOSTNAME>:<PDU_PORT>,<PDU_PORT>;<HOSTNAME>:<PDU_PORT>,<PDU_PORT>"
# OPTION 2: To enable fencing using the Redfish APIs provided by the
Lenovo XCC (and other BMCs):
redfish: &redfish
  username: "{{ bmc_username }}" # Parameter for XCC/BMC username in
the passwords file.
  password: "{{ bmc_password }}" # Parameter for XCC/BMC password in
the passwords file.
    ssl_insecure: 1 # If a valid SSL certificate is not available
specify “1”.
beegfs_ha_fencing_agents:
  fence_redfish:
    - pcmk_host_list: <HOSTNAME>
      ip: <BMC_IP>
      <<: *redfish
    - pcmk_host_list: <HOSTNAME>
      ip: <BMC_IP>
      <<: *redfish
# For details on configuring other fencing agents see
https://access.redhat.com/documentation/en-
us/red_hat_enterprise_linux/9/html/configuring_and_managing_high_avai
lability_clusters/assembly_configuring-fencing-configuring-and-
managing-high-availability-clusters.
```

4. Aktivieren Sie die empfohlene Performance-Optimierung im Linux-Betriebssystem.

   Viele Benutzer finden die Standardeinstellungen für die Performance-Parameter zwar im Allgemeinen gut, Sie können jedoch optional die Standardeinstellungen für einen bestimmten Workload ändern. Daher sind diese Empfehlungen in die BeeGFS-Rolle enthalten, jedoch sind sie nicht standardmäßig aktiviert, um sicherzustellen, dass Benutzer die auf ihr Dateisystem angewendete Einstellung kennen.

   Um das Performance-Tuning zu aktivieren, geben Sie Folgendes an:

```
### Performance Configuration:
beegfs_ha_enable_performance_tuning: True
```

5. (Optional) Sie können die Leistungsparameter im Linux-Betriebssystem nach Bedarf anpassen.

   Eine umfassende Liste der verfügbaren Tuning-Parameter, die Sie anpassen können, finden Sie im Abschnitt Performance Tuning Defaults der BeeGFS HA-Rolle in "E-Series BeeGFS GitHub-Website". Die Standardwerte können für alle Knoten im Cluster in dieser Datei oder für die Datei eines einzelnen Knotens überschrieben werden `host_vars` .

6. Um vollständige 200 GB/HDR-Konnektivität zwischen Block- und Dateiknoten zu ermöglichen, verwenden Sie das OpenSM-Paket (Open Subnetz Manager) aus der NVIDIA Open Fabrics Enterprise Distribution (MLNX_OFED). Die MLNX_OFED-Version in der Liste wird im Lieferumfang der "Anforderungen an den Datei-Node" empfohlenen OpenSM-Pakete enthalten. Obwohl die Implementierung mit Ansible unterstützt wird, müssen Sie zuerst den MLNX_OFED-Treiber auf allen Datei-Nodes installieren.

   a. Füllen Sie die folgenden Parameter in aus `group_vars/ha_cluster.yml` (Passen Sie Pakete nach Bedarf an):

      ```
      ### OpenSM package and configuration information
      eseries_ib_opensm_options:
        virt_enabled: "2"
        virt_max_ports_in_process: "0"
      ```

7. Konfigurieren Sie die `udev` Regel zur Sicherstellung einer konsistenten Zuordnung von logischen InfiniBand-Port-IDs zu zugrunde liegenden PCIe-Geräten.

   Der `udev` Die Regel muss für die PCIe-Topologie jeder Serverplattform, die als BeeGFS-Datei-Node verwendet wird, eindeutig sein.

   Für verifizierte Dateiknoten folgende Werte verwenden:

```
### Ensure Consistent Logical IB Port Numbering
# OPTION 1: Lenovo SR665 V3 PCIe address-to-logical IB port mapping:
eseries_ipoib_udev_rules:
  "0000:01:00.0": i1a
  "0000:01:00.1": i1b
  "0000:41:00.0": i2a
  "0000:41:00.1": i2b
  "0000:81:00.0": i3a
  "0000:81:00.1": i3b
  "0000:a1:00.0": i4a
  "0000:a1:00.1": i4b

# OPTION 2: Lenovo SR665 PCIe address-to-logical IB port mapping:
eseries_ipoib_udev_rules:
  "0000:41:00.0": i1a
  "0000:41:00.1": i1b
  "0000:01:00.0": i2a
  "0000:01:00.1": i2b
  "0000:a1:00.0": i3a
  "0000:a1:00.1": i3b
  "0000:81:00.0": i4a
  "0000:81:00.1": i4b
```

8. (Optional) Aktualisieren des Metadaten-Zielauswahlalgorithmus.

```
beegfs_ha_beegfs_meta_conf_ha_group_options:
  tuneTargetChooser: randomrobin
```

Während der Verifizierungstests `randomrobin` Wurde in der Regel verwendet, um sicherzustellen, dass Testdateien während des Performance-Benchmarking gleichmäßig auf alle BeeGFS-Speicherziele verteilt wurden (weitere Informationen zu Benchmarking finden Sie auf der BeeGFS-Website für "Benchmarking eines BeeGFS-Systems"). Bei der realen Welt könnte dies dazu führen, dass sich die niedrigeren nummerierten Ziele schneller füllen als die höher nummerierten Ziele. Auslassung `randomrobin` Und nur mit dem Standard `randomized` Der Wert zeigt sich, dass er eine gute Leistung bietet und gleichzeitig alle verfügbaren Ziele nutzt.

Schritt 5: Definieren Sie die Konfiguration für den gemeinsamen Block-Node

Die gemeinsame Konfiguration für Block-Knoten wird in einer Gruppe mit dem Namen definiert `eseries_storage_systems`. In den Schritten in diesem Abschnitt wird die Konfiguration erstellt, die in der enthalten sein sollte `group_vars/ eseries_storage_systems.yml` Datei:

Schritte

1. Setzen Sie die Ansible-Verbindung auf Local, geben Sie das Systemkennwort ein und geben Sie an, ob

SSL-Zertifikate verifiziert werden sollen. (Normalerweise verwendet Ansible SSH für die Verbindung zu gemanagten Hosts. Bei Storage-Systemen der NetApp E-Series, die als Block-Nodes verwendet werden, verwenden die Module JEDOCH die REST-API für die Kommunikation.) Fügen Sie oben in der Datei Folgendes hinzu:

```
### eseries_storage_systems Ansible group inventory file.
# Place all default/common variables for NetApp E-Series Storage Systems
here:
ansible_connection: local
eseries_system_password: {{ eseries_password }} # Parameter for E-Series
storage array password in the passwords file.
eseries_validate_certs: false
```

2. Installieren Sie die für Block-Nodes in aufgeführten Versionen, um eine optimale Performance zu gewährleisten "Technische Anforderungen".

   Laden Sie die entsprechenden Dateien aus dem herunter "NetApp Support Website". Sie können sie entweder manuell aktualisieren oder sie in das einbeziehen `packages/` Verzeichnis des Ansible-Steuerungsknotens, und füllen Sie dann die folgenden Parameter in aus `eseries_storage_systems.yml` So führen Sie ein Upgrade mit Ansible durch:

   ```
   # Firmware, NVSRAM, and Drive Firmware (modify the filenames as needed):
   eseries_firmware_firmware: "packages/RCB_11.80GA_6000_64cc0ee3.dlp"
   eseries_firmware_nvsram: "packages/N6000-880834-D08.dlp"
   ```

3. Laden Sie die neueste Laufwerksfirmware herunter, die für die in Ihren Blockknoten installierten Laufwerke verfügbar ist, und installieren Sie sie im "NetApp Support Website". Sie können sie entweder manuell aktualisieren oder in das Verzeichnis des Ansible-Steuerknotens aufnehmen `packages/` . Dann füllen Sie die folgenden Parameter in aus `eseries_storage_systems.yml` , um das Upgrade mit Ansible auszuführen:

   ```
   eseries_drive_firmware_firmware_list:
     - "packages/<FILENAME>.dlp"
   eseries_drive_firmware_upgrade_drives_online: true
   ```

   > Einstellung `eseries_drive_firmware_upgrade_drives_online` Bis `false` Beschleunigt das Upgrade, sollte aber erst nach dem Einsatz von BeeGFS durchgeführt werden. Der Grund dafür ist, dass bei dieser Einstellung sämtliche I/O-Vorgänge auf den Laufwerken vor dem Upgrade angehalten werden müssen, um Applikationsfehler zu vermeiden. Obwohl ein Online-Laufwerk-Firmware-Upgrade vor der Konfiguration von Volumes noch schnell durchgeführt wird, empfehlen wir Ihnen, diesen Wert immer auf zu setzen `true` Um später Probleme zu vermeiden.

4. Nehmen Sie zur Optimierung der Leistung folgende Änderungen an der globalen Konfiguration vor:

```
# Global Configuration Defaults
eseries_system_cache_block_size: 32768
eseries_system_cache_flush_threshold: 80
eseries_system_default_host_type: linux dm-mp
eseries_system_autoload_balance: disabled
eseries_system_host_connectivity_reporting: disabled
eseries_system_controller_shelf_id: 99 # Required.
```

5. Um eine optimale Bereitstellung und ein optimales Verhalten von Volumes zu gewährleisten, geben Sie folgende Parameter an:

```
# Storage Provisioning Defaults
eseries_volume_size_unit: pct
eseries_volume_read_cache_enable: true
eseries_volume_read_ahead_enable: false
eseries_volume_write_cache_enable: true
eseries_volume_write_cache_mirror_enable: true
eseries_volume_cache_without_batteries: false
eseries_storage_pool_usable_drives:
"99:0,99:23,99:1,99:22,99:2,99:21,99:3,99:20,99:4,99:19,99:5,99:18,99:6,
99:17,99:7,99:16,99:8,99:15,99:9,99:14,99:10,99:13,99:11,99:12"
```

Der für angegebene Wert `eseries_storage_pool_usable_drives` Gibt einen spezifischen Block-Node der NetApp EF600 an und steuert die Reihenfolge, in der Laufwerke neuen Volume-Gruppen zugewiesen werden. Durch diese Bestellung wird sichergestellt, dass der I/O zu jeder Gruppe gleichmäßig über die Kanäle des Backend-Laufwerks verteilt wird.

Definieren Sie den Ansible-Bestand für BeeGFS-Bausteine

Definieren Sie nach der Definition der allgemeinen Ansible-Bestandsstruktur die Konfiguration für jeden Baustein im BeeGFS-Dateisystem.

Diese Implementierungsanleitungen zeigen, wie Sie ein Filesystem implementieren, das aus einem Grundbaustein besteht, einschließlich Management-, Metadaten- und Storage-Services, einem zweiten Baustein mit Metadaten und Storage-Services und einem dritten Baustein nur für Storage.

Diese Schritte sollen den gesamten Bereich typischer Konfigurationsprofile anzeigen, mit denen Sie NetApp BeeGFS-Bausteine konfigurieren können, um die Anforderungen des gesamten BeeGFS-Dateisystems zu erfüllen.

Passen Sie in diesen und nachfolgenden Abschnitten nach Bedarf an, um den Bestand zu erstellen, der das BeeGFS-Dateisystem darstellt, das Sie bereitstellen möchten. Verwenden Sie insbesondere Ansible-Hostnamen, die jeden Block- oder Datei-Node darstellen, und das gewünschte IP-Adressschema für das Storage-Netzwerk, um sicherzustellen, dass es auf die Anzahl der BeeGFS-Datei-Nodes und -Clients skaliert werden kann.

Schritt: Die Ansible-Bestandsdatei erstellen

Schritte

1. Erstellen Sie eine neue `inventory.yml` Datei, und fügen Sie dann die folgenden Parameter ein, ersetzen Sie die Hosts unter `eseries_storage_systems` Nach Bedarf zur Darstellung der Block-Nodes in Ihrer Implementierung. Die Namen sollten dem Namen entsprechen, für den sie verwendet werden `host_vars/<FILENAME>.yml`.

```
# BeeGFS HA (High Availability) cluster inventory.
all:
  children:
    # Ansible group representing all block nodes:
    eseries_storage_systems:
      hosts:
        netapp_01:
        netapp_02:
        netapp_03:
        netapp_04:
        netapp_05:
        netapp_06:
    # Ansible group representing all file nodes:
    ha_cluster:
      children:
```

   In den nachfolgenden Abschnitten werden unter weitere Ansible-Gruppen erstellt `ha_cluster` Die die BeeGFS-Dienste darstellen, die im Cluster ausgeführt werden sollen.

Schritt: Inventar für einen Management-, Metadaten- und Storage-Baustein konfigurieren

Der erste Baustein im Cluster- oder Basis-Baustein muss den BeeGFS-Managementservice sowie Metadaten- und Storage-Services umfassen:

Schritte

1. In `inventory.yml`, Befüllen Sie die folgenden Parameter unter `ha_cluster: children`:

```
      # beegfs_01/beegfs_02 HA Pair (mgmt/meta/storage building block):
        mgmt:
          hosts:
            beegfs_01:
            beegfs_02:
        meta_01:
          hosts:
            beegfs_01:
            beegfs_02:
        stor_01:
          hosts:
```

```
      beegfs_01:
      beegfs_02:
  meta_02:
    hosts:
      beegfs_01:
      beegfs_02:
  stor_02:
    hosts:
      beegfs_01:
      beegfs_02:
  meta_03:
    hosts:
      beegfs_01:
      beegfs_02:
  stor_03:
    hosts:
      beegfs_01:
      beegfs_02:
  meta_04:
    hosts:
      beegfs_01:
      beegfs_02:
  stor_04:
    hosts:
      beegfs_01:
      beegfs_02:
  meta_05:
    hosts:
      beegfs_02:
      beegfs_01:
  stor_05:
    hosts:
      beegfs_02:
      beegfs_01:
  meta_06:
    hosts:
      beegfs_02:
      beegfs_01:
  stor_06:
    hosts:
      beegfs_02:
      beegfs_01:
  meta_07:
    hosts:
      beegfs_02:
      beegfs_01:
```

```
        stor_07:
          hosts:
            beegfs_02:
            beegfs_01:
        meta_08:
          hosts:
            beegfs_02:
            beegfs_01:
        stor_08:
          hosts:
            beegfs_02:
            beegfs_01:
```

2. Erstellen Sie die Datei `group_vars/mgmt.yml` Und geben Sie Folgendes an:

```
# mgmt - BeeGFS HA Management Resource Group
# OPTIONAL: Override default BeeGFS management configuration:
# beegfs_ha_beegfs_mgmtd_conf_resource_group_options:
#  <beegfs-mgmt.conf:key>:<beegfs-mgmt.conf:value>
floating_ips:
  - i1b: 100.127.101.0/16
  - i2b: 100.127.102.0/16
beegfs_service: management
beegfs_targets:
  netapp_01:
    eseries_storage_pool_configuration:
      - name: beegfs_m1_m2_m5_m6
        raid_level: raid1
        criteria_drive_count: 4
        common_volume_configuration:
          segment_size_kb:  128
        volumes:
          - size: 1
            owning_controller: A
```

3. Unter `group_vars/`, Dateien für Ressourcengruppen erstellen `meta_01` Bis `meta_08` Verwenden Sie die folgende Vorlage und füllen Sie dann die Platzhalterwerte für jeden Service aus, indem Sie auf die folgende Tabelle verweisen:

```
# meta_0X - BeeGFS HA Metadata Resource Group
beegfs_ha_beegfs_meta_conf_resource_group_options:
  connMetaPortTCP: <PORT>
  connMetaPortUDP: <PORT>
  tuneBindToNumaZone: <NUMA ZONE>
floating_ips:
  - <PREFERRED PORT:IP/SUBNET> # Example: i1b:192.168.120.1/16
  - <SECONDARY PORT:IP/SUBNET>
beegfs_service: metadata
beegfs_targets:
  <BLOCK NODE>:
    eseries_storage_pool_configuration:
      - name: <STORAGE POOL>
        raid_level: raid1
        criteria_drive_count: 4
        common_volume_configuration:
          segment_size_kb:  128
        volumes:
          - size: 21.25 # SEE NOTE BELOW!
            owning_controller: <OWNING CONTROLLER>
```

Die Volume-Größe wird als Prozentsatz des gesamten Storage-Pools angegeben (auch als Volume-Gruppe bezeichnet). NetApp empfiehlt, freie Kapazitäten in jedem Pool zu belassen, um Platz für die SSD-Überprovisionierung zu haben (weitere Informationen finden Sie unter "Einführung in das NetApp EF600 Array"). Der Storage-Pool, `beegfs_m1_m2_m5_m6`, Weist auch 1% der Kapazität des Pools für den Management-Service. Somit für Metadaten-Volumes im Storage-Pool `beegfs_m1_m2_m5_m6`, Wenn 1,92-TB- oder 3,84-TB-Laufwerke verwendet werden, setzen Sie diesen Wert auf `21.25`; Für 7,5-TB-Laufwerke setzen Sie diesen Wert auf `22.25`; Und für 15,3-TB-Laufwerke ist dieser Wert auf festgelegt `23.75`.

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
meta_01.yml	8015	i1b:100.127.101.1/16 i2b:100.127.102.1/16	0	netapp_01	Beegfs_m1_m2_m5_m6	A
meta_02.yml	8025	i2b:100.127.102.2/16 i1b:100.127.101.2/16	0	netapp_01	Beegfs_m1_m2_m5_m6	B
meta_03.yml	8035	i3b:100.127.101.3/16 i4b:100.127.102.3/16	1	netapp_02	Beegfs_m3_m4_m7_m8	A

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
meta_04.yml	8045	l4b:100.127.102.4/16 i3b:100.127.101.4/16	1	netapp_02	Beegfs_m3_m4_m7_m8	B
meta_05.yml	8055	i1b:100.127.101.5/16 i2b:100.127.102.5/16	0	netapp_01	Beegfs_m1_m2_m5_m6	A
meta_06.yml	8065	i2b:100.127.102.6/16 i1b:100.127.101.6/16	0	netapp_01	Beegfs_m1_m2_m5_m6	B
meta_07.yml	8075	i3b:100.127.101.7/16 i4b:100.127.102.7/16	1	netapp_02	Beegfs_m3_m4_m7_m8	A
meta_08.yml	8085	l4b:100.127.102.8/16 i3b:100.127.101.8/16	1	netapp_02	Beegfs_m3_m4_m7_m8	B

4. Unter `group_vars/`, Dateien für Ressourcengruppen erstellen `stor_01` Bis `stor_08` Füllen Sie anschließend die Platzhalterwerte für jeden Service aus, indem Sie auf das Beispiel verweisen:

```
# stor_0X - BeeGFS HA Storage Resource
Groupbeegfs_ha_beegfs_storage_conf_resource_group_options:
  connStoragePortTCP: <PORT>
  connStoragePortUDP: <PORT>
  tuneBindToNumaZone: <NUMA ZONE>
floating_ips:
  - <PREFERRED PORT:IP/SUBNET>
  - <SECONDARY PORT:IP/SUBNET>
beegfs_service: storage
beegfs_targets:
  <BLOCK NODE>:
    eseries_storage_pool_configuration:
      - name: <STORAGE POOL>
        raid_level: raid6
        criteria_drive_count: 10
        common_volume_configuration:
          segment_size_kb: 512        volumes:
          - size: 21.50 # See note below!             owning_controller:
<OWNING CONTROLLER>
          - size: 21.50           owning_controller: <OWNING
CONTROLLER>
```

Informationen zur richtigen Größe finden Sie unter "Empfohlene Prozentsätze für die Überprovisionierung von Storage-Pools".

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
stor_01.yml	8013	i1b:100.127.103.1/16 i2b:100.127.104.1/16	0	netapp_01	Beegfs_s1_s2	A
stor_02.yml	8023	i2b:100.127.104.2/16 i1b:100.127.103.2/16	0	netapp_01	Beegfs_s1_s2	B
stor_03.yml	8033	i3b:100.127.103.3/16 i4b:100.127.104.3/16	1	netapp_02	Beegfs_s3_s4	A
stor_04.yml	8043	l4b:100.127.104.4/16 i3b:100.127.103.4/16	1	netapp_02	Beegfs_s3_s4	B

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
stor_05.yml	8053	i1b:100.127.103.5/16 i2b:100.127.104.5/16	0	netapp_01	Beegfs_s5_s6	A
stor_06.yml	8063	i2b:100.127.104.6/16 i1b:100.127.103.6/16	0	netapp_01	Beegfs_s5_s6	B
stor_07.yml	8073	i3b:100.127.103.7/16 i4b:100.127.104.7/16	1	netapp_02	Beegfs_s7_s8	A
stor_08.yml	8083	l4b:100.127.104.8/16 i3b:100.127.103.8/16	1	netapp_02	Beegfs_s7_s8	B

Schritt 3: Konfigurieren Sie den Bestand für einen Baustein Metadaten + Speicher

In diesen Schritten wird beschrieben, wie ein Ansible-Inventar für BeeGFS-Metadaten + Storage-Baustein eingerichtet wird.

Schritte

1. In `inventory.yml`, Befüllen Sie die folgenden Parameter unter der vorhandenen Konfiguration:

```
        meta_09:
          hosts:
            beegfs_03:
            beegfs_04:
        stor_09:
          hosts:
            beegfs_03:
            beegfs_04:
        meta_10:
          hosts:
            beegfs_03:
            beegfs_04:
        stor_10:
          hosts:
            beegfs_03:
            beegfs_04:
        meta_11:
          hosts:
            beegfs_03:
```

```
        beegfs_04:
    stor_11:
      hosts:
        beegfs_03:
        beegfs_04:
    meta_12:
      hosts:
        beegfs_03:
        beegfs_04:
    stor_12:
      hosts:
        beegfs_03:
        beegfs_04:
    meta_13:
      hosts:
        beegfs_04:
        beegfs_03:
    stor_13:
      hosts:
        beegfs_04:
        beegfs_03:
    meta_14:
      hosts:
        beegfs_04:
        beegfs_03:
    stor_14:
      hosts:
        beegfs_04:
        beegfs_03:
    meta_15:
      hosts:
        beegfs_04:
        beegfs_03:
    stor_15:
      hosts:
        beegfs_04:
        beegfs_03:
    meta_16:
      hosts:
        beegfs_04:
        beegfs_03:
    stor_16:
      hosts:
        beegfs_04:
        beegfs_03:
```

2. Unter `group_vars/`, Dateien für Ressourcengruppen erstellen `meta_09` Bis `meta_16` Füllen Sie anschließend die Platzhalterwerte für jeden Service aus, indem Sie auf das Beispiel verweisen:

```
# meta_0X - BeeGFS HA Metadata Resource Group
beegfs_ha_beegfs_meta_conf_resource_group_options:
  connMetaPortTCP: <PORT>
  connMetaPortUDP: <PORT>
  tuneBindToNumaZone: <NUMA ZONE>
floating_ips:
  - <PREFERRED PORT:IP/SUBNET>
  - <SECONDARY PORT:IP/SUBNET>
beegfs_service: metadata
beegfs_targets:
  <BLOCK NODE>:
    eseries_storage_pool_configuration:
      - name: <STORAGE POOL>
        raid_level: raid1
        criteria_drive_count: 4
        common_volume_configuration:
          segment_size_kb: 128
        volumes:
          - size: 21.5 # SEE NOTE BELOW!
            owning_controller: <OWNING CONTROLLER>
```

Informationen zur richtigen Größe finden Sie unter "Empfohlene Prozentsätze für die Überprovisionierung von Storage-Pools".

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
meta_09.yml	8015	i1b:100.127.101.9/16 i2b:100.127.102.9/16	0	netapp_03	Beegfs_m9_m10_m13_m14	A
meta_10.yml	8025	i2b:100.127.102.10/16 i1b:100.127.101.10/16	0	netapp_03	Beegfs_m9_m10_m13_m14	B
meta_11.yml	8035	i3b:100.127.101.11/16 i4b:100.127.102.11/16	1	netapp_04	Beegfs_m11_m12_m15_m16	A
meta_12.yml	8045	l4b:100.127.102.12/16 i3b:100.127.101.12/16	1	netapp_04	Beegfs_m11_m12_m15_m16	B

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
meta_13.yml	8055	i1b:100.127.101.13/16 i2b:100.127.102.13/16	0	netapp_03	Beegfs_m9_m10_m13_m14	A
meta_14.yml	8065	i2b:100.127.102.14/16 i1b:100.127.101.14/16	0	netapp_03	Beegfs_m9_m10_m13_m14	B
meta_15.yml	8075	i3b:100.127.101.15/16 i4b:100.127.102.15/16	1	netapp_04	Beegfs_m11_m12_m15_m16	A
meta_16.yml	8085	l4b:100.127.102.16/16 i3b:100.127.101.16/16	1	netapp_04	Beegfs_m11_m12_m15_m16	B

3. Unter `group_vars/`, Dateien für Ressourcengruppen erstellen `stor_09` Bis `stor_16` Füllen Sie anschließend die Platzhalterwerte für jeden Service aus, indem Sie auf das Beispiel verweisen:

```
# stor_0X - BeeGFS HA Storage Resource Group
beegfs_ha_beegfs_storage_conf_resource_group_options:
  connStoragePortTCP: <PORT>
  connStoragePortUDP: <PORT>
  tuneBindToNumaZone: <NUMA ZONE>
floating_ips:
  - <PREFERRED PORT:IP/SUBNET>
  - <SECONDARY PORT:IP/SUBNET>
beegfs_service: storage
beegfs_targets:
  <BLOCK NODE>:
    eseries_storage_pool_configuration:
      - name: <STORAGE POOL>
        raid_level: raid6
        criteria_drive_count: 10
        common_volume_configuration:
          segment_size_kb: 512        volumes:
          - size: 21.50 # See note below!
            owning_controller: <OWNING CONTROLLER>
          - size: 21.50            owning_controller: <OWNING
CONTROLLER>
```

Die richtige Größe finden Sie unter "Empfohlene Prozentsätze für die Überprovisionierung von Storage-Pools" ..

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
stor_09.yml	8013	i1b:100.127.103.9/16 i2b:100.127.104.9/16	0	netapp_03	Beegfs_s9_s10	A
stor_10.yml	8023	i2b:100.127.104.10/16 i1b:100.127.103.10/16	0	netapp_03	Beegfs_s9_s10	B
stor_11.yml	8033	i3b:100.127.103.11/16 i4b:100.127.104.11/16	1	netapp_04	Beegfs_s11_s12	A
stor_12.yml	8043	l4b:100.127.104.12/16 i3b:100.127.103.12/16	1	netapp_04	Beegfs_s11_s12	B
stor_13.yml	8053	i1b:100.127.103.13/16 i2b:100.127.104.13/16	0	netapp_03	Beegfs_s13_s14	A
stor_14.yml	8063	i2b:100.127.104.14/16 i1b:100.127.103.14/16	0	netapp_03	Beegfs_s13_s14	B
stor_15.yml	8073	i3b:100.127.103.15/16 i4b:100.127.104.15/16	1	netapp_04	Beegfs_s15_s16	A
stor_16.yml	8083	l4b:100.127.104.16/16 i3b:100.127.103.16/16	1	netapp_04	Beegfs_s15_s16	B

Schritt 4: Konfigurieren Sie den Bestand für einen nur-Storage-Baustein

In diesen Schritten wird beschrieben, wie Sie einen Ansible-Bestand für einen einzigen BeeGFS-Baustein einrichten. Der Hauptunterschied zwischen der Konfiguration für Metadaten + Storage und einem rein Storage-basierten Baustein besteht darin, dass alle Metadaten-Ressourcengruppen und Änderungen nicht mehr berücksichtigt werden `criteria_drive_count` Von 10 bis 12 für jeden Speicherpool.

Schritte

1. In `inventory.yml`, Befüllen Sie die folgenden Parameter unter der vorhandenen Konfiguration:

```
# beegfs_05/beegfs_06 HA Pair (storage only building block):
  stor_17:
    hosts:
      beegfs_05:
      beegfs_06:
  stor_18:
    hosts:
      beegfs_05:
      beegfs_06:
  stor_19:
    hosts:
      beegfs_05:
      beegfs_06:
  stor_20:
    hosts:
      beegfs_05:
      beegfs_06:
  stor_21:
    hosts:
      beegfs_06:
      beegfs_05:
  stor_22:
    hosts:
      beegfs_06:
      beegfs_05:
  stor_23:
    hosts:
      beegfs_06:
      beegfs_05:
  stor_24:
    hosts:
      beegfs_06:
      beegfs_05:
```

2. Unter `group_vars/`, Dateien für Ressourcengruppen erstellen `stor_17` Bis `stor_24` Füllen Sie anschließend die Platzhalterwerte für jeden Service aus, indem Sie auf das Beispiel verweisen:

```
# stor_0X - BeeGFS HA Storage Resource Group
beegfs_ha_beegfs_storage_conf_resource_group_options:
  connStoragePortTCP: <PORT>
  connStoragePortUDP: <PORT>
  tuneBindToNumaZone: <NUMA ZONE>
floating_ips:
  - <PREFERRED PORT:IP/SUBNET>
  - <SECONDARY PORT:IP/SUBNET>
beegfs_service: storage
beegfs_targets:
  <BLOCK NODE>:
    eseries_storage_pool_configuration:
      - name: <STORAGE POOL>
        raid_level: raid6
        criteria_drive_count: 12
        common_volume_configuration:
          segment_size_kb: 512
        volumes:
          - size: 21.50 # See note below!
            owning_controller: <OWNING CONTROLLER>
          - size: 21.50
            owning_controller: <OWNING CONTROLLER>
```

Die richtige Größe finden Sie unter "Empfohlene Prozentsätze für die Überprovisionierung von Storage-Pools" .

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
stor_17.yml	8013	i1b:100.127.103.17/16 i2b:100.127.104.17/16	0	netapp_05	Beegfs_s17_s18	A
stor_18.yml	8023	i2b:100.127.104.18/16 i1b:100.127.103.18/16	0	netapp_05	Beegfs_s17_s18	B
stor_19.yml	8033	i3b:100.127.103.19/16 i4b:100.127.104.19/16	1	netapp_06	Beegfs_s19_s20	A
stor_20.yml	8043	l4b:100.127.104.20/16 i3b:100.127.103.20/16	1	netapp_06	Beegfs_s19_s20	B

Dateiname	Port	Fließende IPs	NUMA-Zone	Block-Node	Storage-Pool	Controller, der die LUN besitzt
stor_21.yml	8053	i1b:100.127.103.21/16 i2b:100.127.104.21/16	0	netapp_05	Beegfs_s21_s22	A
stor_22.yml	8063	i2b:100.127.104.22/16 i1b:100.127.103.22/16	0	netapp_05	Beegfs_s21_s22	B
stor_23.yml	8073	i3b:100.127.103.23/16 i4b:100.127.104.23/16	1	netapp_06	Beegfs_s23_s24	A
stor_24.yml	8083	l4b:100.127.104.24/16 i3b:100.127.103.24/16	1	netapp_06	Beegfs_s23_s24	B

BeeGFS bereitstellen

Zur Implementierung und zum Management der Konfiguration werden ein oder mehrere Playbooks ausgeführt, die die Aufgaben enthalten, die Ansible ausführen muss, und das gesamte System in den gewünschten Zustand bringen.

Zwar können alle Aufgaben in einem einzigen Playbook enthalten sein, doch bei komplexen Systemen ist dies schnell und schwerfällig. Mit Ansible können Sie Rollen erstellen und verteilen, um wiederverwendbare Playbooks und verwandte Inhalte (z. B. Standardvariablen, Aufgaben und Handler) zu verpacken. Weitere Informationen finden Sie in der Ansible-Dokumentation für "Rollen".

Rollen werden häufig im Rahmen einer Ansible Sammlung mit zugehörigen Rollen und Modulen verteilt. Daher importieren diese Playbooks in erster Linie mehrere Rollen, die in den verschiedenen NetApp E-Series Ansible Sammlungen verteilt sind.

> Derzeit sind mindestens zwei Bausteine (vier Datei-Nodes) für die Bereitstellung von BeeGFS erforderlich, es sei denn, ein separates Quorum-Gerät ist als Tiebreaker konfiguriert, um Probleme beim Einrichten von Quorum mit einem Cluster mit zwei Nodes zu minimieren.

Schritte

1. Erstellen Sie eine neue `playbook.yml` Datei und schließen Sie Folgendes ein:

```
# BeeGFS HA (High Availability) cluster playbook.
- hosts: eseries_storage_systems
  gather_facts: false
  collections:
    - netapp_eseries.santricity
  tasks:
```

```
    - name: Configure NetApp E-Series block nodes.
      import_role:
        name: nar_santricity_management
- hosts: all
  any_errors_fatal: true
  gather_facts: false
  collections:
    - netapp_eseries.beegfs
  pre_tasks:
    - name: Ensure a supported version of Python is available on all
file nodes.
      block:
        - name: Check if python is installed.
          failed_when: false
          changed_when: false
          raw: python --version
          register: python_version
        - name: Check if python3 is installed.
          raw: python3 --version
          failed_when: false
          changed_when: false
          register: python3_version
          when: 'python_version["rc"] != 0 or (python_version["stdout"]
| regex_replace("Python ", "")) is not version("3.0", ">=")'
        - name: Install python3 if needed.
          raw: |
            id=$(grep "^ID=" /etc/*release* | cut -d= -f 2 | tr -d '"')
            case $id in
              ubuntu) sudo apt install python3 ;;
              rhel|centos) sudo yum -y install python3 ;;
              sles) sudo zypper install python3 ;;
            esac
          args:
            executable: /bin/bash
          register: python3_install
          when: python_version['rc'] != 0 and python3_version['rc'] != 0
          become: true
        - name: Create a symbolic link to python from python3.
          raw: ln -s /usr/bin/python3 /usr/bin/python
          become: true
          when: python_version['rc'] != 0
      when: inventory_hostname not in
groups[beegfs_ha_ansible_storage_group]
    - name: Verify any provided tags are supported.
      fail:
        msg: "{{ item }} tag is not a supported BeeGFS HA tag. Rerun
```

```
your playbook command with --list-tags to see all valid playbook tags."
      when: 'item not in ["all", "storage", "beegfs_ha",
"beegfs_ha_package", "beegfs_ha_configure",
"beegfs_ha_configure_resource", "beegfs_ha_performance_tuning",
"beegfs_ha_backup", "beegfs_ha_client"]'
      loop: "{{ ansible_run_tags }}"
  tasks:
    - name: Verify before proceeding.
      pause:
        prompt: "Are you ready to proceed with running the BeeGFS HA
role? Depending on the size of the deployment and network performance
between the Ansible control node and BeeGFS file and block nodes this
can take awhile (10+ minutes) to complete."
    - name: Verify the BeeGFS HA cluster is properly deployed.
      ansible.builtin.import_role:
        name: netapp_eseries.beegfs.beegfs_ha_7_4
```

In diesem Playbook wird ein paar ausgeführt `pre_tasks` Überprüfen Sie, ob Python 3 auf den Datei-Nodes installiert ist, und überprüfen Sie, ob die angegebenen Ansible-Tags unterstützt werden.

2. Verwenden Sie die `ansible-playbook` Befehl mit den Inventar- und Playbook-Dateien, wenn Sie bereit sind BeeGFS zu implementieren.

   Die Bereitstellung wird komplett ausgeführt `pre_tasks`, Und dann zur Bestätigung des Benutzers aufgefordert, bevor mit der tatsächlichen BeeGFS-Bereitstellung.

   Führen Sie den folgenden Befehl aus, indem Sie die Anzahl der Gabeln nach Bedarf anpassen (siehe Hinweis unten):

   ```
   ansible-playbook -i inventory.yml playbook.yml --forks 20
   ```

   Insbesondere bei größeren Bereitstellungen `forks` wird empfohlen, die Standardanzahl von Gabeln (5) mit dem Parameter zu überschreiben, um die Anzahl der Hosts zu erhöhen, die Ansible parallel konfiguriert. (Weitere Informationen finden Sie unter "Kontrolle der Playbook-Ausführung".) Die Einstellung für den maximalen Wert hängt von der Verarbeitungsleistung ab, die auf dem Ansible-Steuerungsknoten verfügbar ist. Das oben genannte Beispiel von 20 wurde auf einem virtuellen Ansible-Steuerungsknoten mit 4 CPUs (Intel® Xeon® Gold 6146 CPU @ 3,20 GHz) ausgeführt.

   Je nach Größe der Implementierung und Netzwerk-Performance zwischen dem Ansible Control Node und BeeGFS File- und Block-Nodes kann die Implementierungszeit variieren.

Konfigurieren Sie BeeGFS-Clients

Sie müssen den BeeGFS-Client auf allen Hosts installieren und konfigurieren, die Zugriff auf das BeeGFS-Dateisystem benötigen, z. B. Compute- oder GPU-Nodes. Für diese

Aufgabe können Sie Ansible und die BeeGFS-Sammlung verwenden.

Schritte

1. Richten Sie bei Bedarf über den Ansible-Steuerungsknoten passwortlose SSH für jeden Host ein, den Sie als BeeGFS-Clients konfigurieren möchten:

   ```
   ssh-copy-id <user>@<HOSTNAME_OR_IP>
   ```

2. Unter `host_vars/`Erstellen Sie für jeden BeeGFS-Client eine Datei mit dem Namen `<HOSTNAME>.yml` Füllen Sie den Platzhaltertext mit den korrekten Informationen für Ihre Umgebung aus:

   ```
   # BeeGFS Client
   ansible_host: <MANAGEMENT_IP>
   # OPTIONAL: If you want to use the NetApp E-Series Host Collection's
   IPoIB role to configure InfiniBand interfaces for clients to connect to
   BeeGFS file systems:
   eseries_ipoib_interfaces:
     - name: <INTERFACE>
       address: <IP>/<SUBNET_MASK> # Example: 100.127.1.1/16
     - name: <INTERFACE>
       address: <IP>/<SUBNET_MASK>
   ```

   Bei der Bereitstellung mit zwei Subnetzadressierungsschemata müssen auf jedem Client zwei InfiniBand-Schnittstellen konfiguriert werden, eine in jedem der beiden Storage-IPoIB-Subnetze. Wenn Sie die Beispiel-Subnetze und empfohlenen Bereiche für jeden hier aufgeführten BeeGFS-Dienst verwenden, sollten Clients eine Schnittstelle im Bereich von bis und die andere in bis konfigurieren `100.127.1.0` `100.127.99.255` `100.128.1.0` `100.128.99.255`.

3. Erstellen Sie eine neue Datei `client_inventory.yml`, Und dann füllen Sie die folgenden Parameter an der Spitze:

   ```
   # BeeGFS client inventory.
   all:
     vars:
       ansible_ssh_user: <USER> # This is the user Ansible should use to
   connect to each client.
       ansible_become_password: <PASSWORD> # This is the password Ansible
   will use for privilege escalation, and requires the ansible_ssh_user be
   root, or have sudo privileges.
   The defaults set by the BeeGFS HA role are based on the testing
   performed as part of this NetApp Verified Architecture and differ from
   the typical BeeGFS client defaults.
   ```

Speichern Sie Passwörter nicht im Klartext. Verwenden Sie stattdessen den Ansible Vault (siehe Ansible-Dokumentation für) "Verschlüsseln von Inhalten mit Ansible Vault") Oder verwenden Sie die `--ask-become-pass` Option beim Ausführen des Playbooks.

4. Im `client_inventory.yml` Datei, Listen Sie alle Hosts auf, die als BeeGFS-Clients unter dem konfiguriert werden sollen `beegfs_clients` Gruppe, und geben Sie dann alle zusätzlichen Konfigurationen an, die zum Erstellen des BeeGFS-Client-Kernelmoduls erforderlich sind.

```
  children:
    # Ansible group representing all BeeGFS clients:
    beegfs_clients:
      hosts:
        beegfs_01:
        beegfs_02:
        beegfs_03:
        beegfs_04:
        beegfs_05:
        beegfs_06:
        beegfs_07:
        beegfs_08:
        beegfs_09:
        beegfs_10:
      vars:
        # OPTION 1: If you're using the NVIDIA OFED drivers and they are
already installed:
        eseries_ib_skip: True # Skip installing inbox drivers when using
the IPoIB role.
        beegfs_client_ofed_enable: True
        beegfs_client_ofed_include_path:
"/usr/src/ofa_kernel/default/include"
        # OPTION 2: If you're using inbox IB/RDMA drivers and they are
already installed:
        eseries_ib_skip: True # Skip installing inbox drivers when using
the IPoIB role.
        # OPTION 3: If you want to use inbox IB/RDMA drivers and need
them installed/configured.
        eseries_ib_skip: False # Default value.
        beegfs_client_ofed_enable: False # Default value.
```

Wenn Sie die NVIDIA OFED-Treiber verwenden, stellen Sie sicher, dass `beegfs_client_ofed_include_path` sie auf den richtigen „Header include path“ für Ihre Linux-Installation verweist. Weitere Informationen finden Sie in der BeeGFS-Dokumentation für "RDMA-Unterstützung".

5. Im `client_inventory.yml` Datei, Listen Sie die BeeGFS-Dateisysteme auf, die am unteren Rand eines zuvor definierten gemountet werden sollen `vars`.

```
        beegfs_client_mounts:
          - sysMgmtdHost: 100.127.101.0 # Primary IP of the BeeGFS
management service.
            mount_point: /mnt/beegfs    # Path to mount BeeGFS on the
client.
            connInterfaces:
              - <INTERFACE> # Example: ibs4f1
              - <INTERFACE>
            beegfs_client_config:
              # Maximum number of simultaneous connections to the same
node.
              connMaxInternodeNum: 128 # BeeGFS Client Default: 12
              # Allocates the number of buffers for transferring IO.
              connRDMABufNum: 36 # BeeGFS Client Default: 70
              # Size of each allocated RDMA buffer
              connRDMABufSize: 65536 # BeeGFS Client Default: 8192
              # Required when using the BeeGFS client with the shared-
disk HA solution.
              # This does require BeeGFS targets be mounted in the
default "sync" mode.
              # See the documentation included with the BeeGFS client
role for full details.
              sysSessionChecksEnabled: false
```

Der `beegfs_client_config` Stellt die Einstellungen dar, die getestet wurden. Lesen Sie die Dokumentation im `netapp_eseries.beegfs` Kollektion's `beegfs_client` Rolle für einen umfassenden Überblick über alle Optionen. Dies enthält Details zum Mounten mehrerer BeeGFS-Dateisysteme oder zum mehrere Male das gleiche BeeGFS-Dateisystem.

6. Erstellen Sie eine neue `client_playbook.yml` Datei, und füllen Sie dann die folgenden Parameter aus:

```
# BeeGFS client playbook.
- hosts: beegfs_clients
  any_errors_fatal: true
  gather_facts: true
  collections:
    - netapp_eseries.beegfs
    - netapp_eseries.host
  tasks:
    - name: Ensure IPoIB is configured
      import_role:
        name: ipoib
    - name: Verify the BeeGFS clients are configured.
      import_role:
        name: beegfs_client
```

Beim Importieren des weglassen `netapp_eseries.host` Sammlung und `ipoib` Rolle, wenn Sie bereits die erforderlichen IB/RDMA-Treiber und IP-Adressen auf den entsprechenden IPoIB-Schnittstellen installiert haben.

7. Führen Sie den folgenden Befehl aus, um den Client zu installieren und zu erstellen und BeeGFS zu mounten:

```
ansible-playbook -i client_inventory.yml client_playbook.yml
```

8. Bevor Sie das BeeGFS-Dateisystem in Produktion setzen, empfehlen wir * dringend*, sich bei allen Clients anzumelden und zu starten `beegfs-fsck --checkfs` Um sicherzustellen, dass alle Knoten erreichbar sind und keine Probleme gemeldet werden.

Skalierung auf mehr als fünf Bausteine

Pacemaker und Corosync können so konfiguriert werden, dass sie über fünf Bausteine (10 Datei-Knoten) hinausgehen. Allerdings gibt es Nachteile zu größeren Clustern, und schließlich Pacemaker und Corosync auferlegen ein Maximum von 32 Knoten.

NetApp hat nur BeeGFS HA Cluster für bis zu 10 Nodes getestet. Es wird nicht empfohlen, einzelne Cluster über dieses Limit hinaus zu skalieren. BeeGFS-Filesysteme müssen jedoch immer noch weit über 10 Nodes skalieren, und NetApp hat dies in der BeeGFS on NetApp Lösung berücksichtigt.

Durch die Implementierung mehrerer HA-Cluster mit einer Teilmenge der Bausteine in jedem Filesystem können Sie das gesamte BeeGFS Filesystem unabhängig von empfohlenen oder festen Grenzwerten für die zugrunde liegenden HA-Clustering-Mechanismen skalieren. Gehen Sie in diesem Szenario wie folgt vor:

- Erstellen Sie einen neuen Ansible-Bestand, der die zusätzlichen HA-Cluster darstellt, und geben Sie dann das Konfigurieren eines anderen Managementservices nicht ein. Zeigen Sie stattdessen auf `beegfs_ha_mgmtd_floating_ip` Variable in jedem zusätzlichen Cluster enthalten `ha_cluster.yml` An die IP für den ersten BeeGFS Management Service.

- Wenn Sie dem selben Dateisystem zusätzliche HA-Cluster hinzufügen, stellen Sie Folgendes sicher:
  - Die BeeGFS-Knoten-IDs sind eindeutig.
  - Die Dateinamen, die den einzelnen Diensten unter entsprechen `group_vars` Ist für alle Cluster eindeutig.
  - Die BeeGFS-Client- und Server-IP-Adressen sind für alle Cluster eindeutig.
  - Das erste HA-Cluster mit dem BeeGFS-Managementservice wird ausgeführt, bevor versucht wird, zusätzliche Cluster zu implementieren oder zu aktualisieren.
- Inventarisierung für jedes HA-Cluster getrennt in der eigenen Verzeichnisstruktur

  Wenn Sie versuchen, die Bestandsdateien für mehrere Cluster in einem Verzeichnisbaum zu mischen, kann dies zu Problemen führen, wie die BeeGFS HA-Rolle die auf ein bestimmtes Cluster angewendete Konfiguration aggregiert.

> Es ist nicht erforderlich, dass jedes HA-Cluster auf fünf Bausteine skaliert werden kann, bevor ein neues erstellt wird. In vielen Fällen lässt sich das Management mit weniger Bausteinen pro Cluster vereinfachen. Ein Ansatz besteht darin, die Bausteine in jedem einzelnen Rack als HA-Cluster zu konfigurieren.

Empfohlene Prozentsätze für die Überprovisionierung von Storage-Pools

Wenn Sie den standardmäßigen vier Volumes pro Storage-Pool-Konfiguration für Bausteine der zweiten Generation folgen, finden Sie in der folgenden Tabelle.

Diese Tabelle enthält empfohlene Prozentsätze, die als Volume-Größe im verwendet werden können `eseries_storage_pool_configuration` Für jede BeeGFS-Metadaten oder jedes Storage-Ziel:

Laufwerkgröße	Größe
1,92 TB	18
3,84 TB	21.5
7,68 TB	22.5
15,3 TB	24

> Die oben stehende Anleitung gilt nicht für den Speicherpool, der den Management-Service enthält. Dieser sollte die Größe von über 25 % verringern, um 1 % des Speicherpools für Management-Daten zuzuweisen.

Um zu verstehen, wie diese Werte ermittelt wurden, lesen Sie "TR-4800: Anhang A: Verständnis von SSD-Ausdauer und -Überprovisionierung".

Baustein mit hoher Kapazität

Im Standard BeeGFS Solution Deployment Guide werden Verfahren und Empfehlungen für High-Performance-Workload-Anforderungen beschrieben. Kunden, die hohe Kapazitätsanforderungen erfüllen möchten, sollten die hier aufgeführten Variationen bei Implementierung und Empfehlungen beobachten.

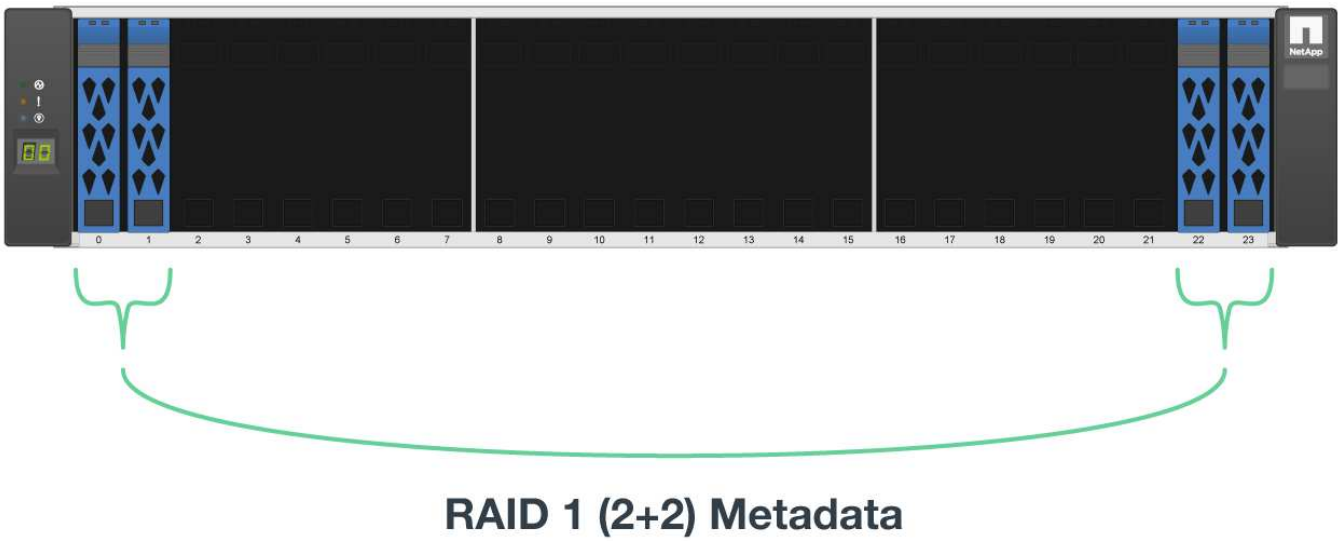
Controller

Wenn Sie Bausteine mit hoher Kapazität benötigen, sollten EF600 Controller durch EF300 Controller ersetzt werden, wobei jeweils eine Cascade HIC zur SAS-Erweiterung installiert ist. Jeder Block-Node verfügt über eine minimale Anzahl von NVMe-SSDs, die im Array-Gehäuse für BeeGFS-Metadaten-Storage befüllt sind, und wird an Erweiterungs-Shelfs mit NL-SAS-HDDs für BeeGFS-Storage-Volumes angeschlossen.

Die Konfiguration des Datei-Node zu Block-Nodes bleibt unverändert.

Laufwerkplatzierung

In jedem Block-Node sind mindestens 4 NVMe-SSDs für BeeGFS-Metadaten-Storage erforderlich. Diese Laufwerke sollten in den äußeren Steckplätzen des Gehäuses platziert werden.



Erweiterungsfächer

Der Baustein mit hoher Kapazität kann mit 1-7, 60 Laufwerkserweiterungsfächern pro Speicher-Array dimensioniert werden.

Anweisungen zum Kabelanschluss der einzelnen Erweiterungsfächer finden Sie unter "Siehe EF300-Verkabelung für Laufwerk-Shelfs".