E-Series 스토리지를 지원하는 NetApp 기반 BeeGFS

BeeGFS on NetApp with E-Series Storage

NetApp
January 27, 2026

목차

E-Series 스토리지를 지원하는 NetApp 기반 BeeGFS

시작하십시오

이 사이트에 포함된 내용

이 사이트에서는 NetApp NVA(Verified Architecture)와 맞춤형 아키텍처를 모두 사용하여 NetApp에서 BeeGFS를 구축 및 관리하는 방법을 설명합니다. NVA 설계는 철저한 테스트를 거쳐 고객에게 구축 위험을 최소화하고 시장 출시 기간을 단축할 수 있는 참조 구성 및 사이징 지침을 제공합니다. 또한 NetApp에서는 NetApp 하드웨어에서 실행되는 맞춤형 BeeGFS 아키텍처를 지원하므로 고객과 파트너가 다양한 요구사항에 맞춰 파일 시스템을 유연하게 설계할 수 있습니다. 두 접근 방식 모두 Ansible을 구축하여 다양한 하드웨어에서 모든 규모의 BeeGFS를 관리하는 어플라이언스 방식의 접근 방식을 제공합니다.

용어 및 개념

다음 용어와 개념은 NetApp 기반 BeeGFS 솔루션에 적용됩니다.

"BeeGFS 클러스터 관리"BeeGFS 고가용성(HA) 클러스터와 상호 작용하는 것과 관련된 용어 및 개념에 대한 자세한 내용은 섹션을 참조하십시오.

기간	설명
AI	인공 지능.
Ansible 컨트롤 노드	Ansible CLI 실행에 사용되는 물리적 또는 가상 머신
Ansible 인벤토리	원하는 BeeGFS HA 클러스터를 설명하는 데 사용되는 YAML 파일이 포함된 디렉토리 구조.
BMC	베이스보드 관리 컨트롤러. 서비스 프로세서라고도 합니다.
블록 노드	E-Series 스토리지 시스템
클라이언트	파일 시스템을 사용해야 하는 애플리케이션을 실행하는 HPC 클러스터의 노드 컴퓨팅 또는 GPU 노드라고도 합니다.
DL	딥 러닝.
파일 노드	BeeGFS 파일 서버
HA	고가용성.
HIC	호스트 인터페이스 카드.

기간	설명
HPC	고성능 컴퓨팅.
HPC 스타일의 워크로드	HPC 스타일 워크로드는 일반적으로 여러 컴퓨팅 노드 또는 GPU에서 동일한 데이터 세트에 병렬로 액세스하여 분산된 컴퓨팅 또는 교육 작업을 진행하는 것이 특징입니다. 이러한 데이터 세트는 단일 파일에 대한 동시 액세스를 방지하는 기존 하드웨어 병목 현상을 제거하기 위해 여러 물리적 스토리지 노드에 스트라이핑되어야 하는 대용량 파일로 구성되는 경우가 많습니다.
ML	머신 러닝.
NLP	자연어 처리.
NLU	자연어 이해.
NVA	NVA(NetApp Verified Architecture) 프로그램은 특정 워크로드 및 사용 사례에 대한 참조 구성 및 사이징 지침을 제공합니다. 이러한 솔루션은 철저한 테스트를 거쳤으며, 구축 위험을 최소화하고 출시 기간을 단축할 수 있도록 설계되었습니다.
스토리지 네트워크 /클라이언트 네트워크	클라이언트가 BeeGFS 파일 시스템과 통신하는 데 사용되는 네트워크입니다. 이 네트워크는 종종 병렬 MPI(Message Passing Interface)와 HPC 클러스터 노드 간의 기타 응용 프로그램 통신에 사용되는 동일한 네트워크입니다.

검증된 아키텍처를 사용합니다

개요 및 요구 사항

솔루션 개요

BeeGFS on NetApp 솔루션은 BeeGFS 병렬 파일 시스템을 NetApp EF600 스토리지 시스템과 결합하여 까다로운 워크로드에 대응할 수 있는 안정적이고 확장 가능하며 비용 효율적인 인프라를 제공합니다.

NVA 프로그램

NetApp 솔루션의 BeeGFS는 NVA(NetApp Verified Architecture) 프로그램에 포함되어 있으며, 특정 워크로드 및 사용 사례에 대한 참조 구성 및 사이징 지침을 고객에게 제공합니다. NVA 솔루션은 구축 위험을 최소화하고 시장 출시 기간을 단축할 수 있도록 철저한 테스트와 설계를 거쳤습니다.

설계 개요

NetApp 기반 BeeGFS 솔루션은 확장 가능한 구성 요소 아키텍처로 설계되었으며 다양한 까다로운 워크로드에 맞게 구성할 수 있습니다. 여러 개의 작은 파일을 처리하든, 대규모 파일 작업을 관리하든, 하이브리드 작업 부하를 처리하든 이러한 요구 사항을 충족하도록 파일 시스템을 사용자 지정할 수 있습니다. 고가용성은 다중 하드웨어 계층에서 독립적인 페일오버를 허용하고 부분적인 시스템 성능 저하 중에도 일관된 성능을 보장하는 2계층 하드웨어 구조를 사용하여 설계됩니다. BeeGFS 파일 시스템은 다양한 Linux 배포에서 확장형 고성능 환경을 지원하고 고객에게 쉽게 액세스할 수 있는 단일 스토리지 네임스페이스를 제공합니다. 자세한 내용은 을 "아키텍처 개요"참조하십시오.

사용 사례

NetApp 기반 BeeGFS 솔루션에는 다음 사용 사례가 적용됩니다.

- DGX와 A100, H100, H200 및 B200 GPU를 사용하는 NVIDIA DGX SuperPOD 시스템
- 머신 러닝(ML), 딥 러닝(DL), 대규모 자연어 처리(NLP), 자연어 이해(NLU)를 비롯한 인공 지능(AI) 자세한 내용은 을 참조하십시오 "AI용 BeeGFS: 팩션과 픽션 비교".
- MPI(메시지 전달 인터페이스) 및 기타 분산 컴퓨팅 기술에 의해 가속되는 응용 프로그램을 포함한 고성능 컴퓨팅(HPC). 자세한 내용은 을 참조하십시오 "BeeGFS가 HPC를 넘어서는 이유".
- 애플리케이션 워크로드의 특징:
  - 1GB 이상의 파일을 읽거나 쓰는 중입니다
  - 여러 클라이언트(10s, 100s 및 1000)에서 동일한 파일을 읽거나 쓰는 경우
- 테라바이트급 또는 페타바이트급의 데이터 세트
- 크기가 큰 파일과 작은 파일을 혼합하여 사용할 수 있도록 최적화되는 단일 스토리지 네임스페이스가 필요한 환경

이점

NetApp에서 BeeGFS를 사용할 때의 주요 이점은 다음과 같습니다.

- 검증된 하드웨어 설계를 통해 하드웨어 및 소프트웨어 구성요소를 완벽하게 통합하여 예측 가능한 성능과 안정성을 보장합니다.

- Ansible을 사용한 구축 및 관리로 단순성과 일관성 확보
- E-Series Performance Analyzer 및 BeeGFS 플러그인을 사용하여 제공되는 모니터링 및 관찰 가능성 자세한 내용은 을 참조하십시오 "NetApp E-Series 솔루션을 모니터링하는 프레임워크 소개".
- 데이터 내구성과 가용성을 제공하는 공유 디스크 아키텍처를 갖춘 고가용성
- 컨테이너 및 Kubernetes를 사용하여 최신 워크로드 관리 및 오케스트레이션 지원 자세한 내용은 을 참조하십시오 "Kubernetes에서 BeeGFS를 만나 보십시오. 미래 지향형 투자 이야기입니다".

아키텍처 개요

BeeGFS on NetApp 솔루션에는 검증된 워크로드를 지원하는 데 필요한 특정 장비, 케이블링, 구성을 결정하는 데 사용되는 아키텍처 설계 고려사항이 포함되어 있습니다.

빌딩 블록 아키텍처

스토리지 요구 사항에 따라 BeeGFS 파일 시스템을 다양한 방식으로 구축 및 확장할 수 있습니다. 예를 들어, 주로 작은 파일을 많이 사용하는 사용 사례에서는 메타데이터 성능과 용량이 더 큰 반면, 큰 파일이 적은 사용 사례에서는 실제 파일 콘텐츠에 대해 더 많은 스토리지 용량과 성능을 선호할 수 있습니다. 이러한 여러 가지 고려 사항은 병렬 파일 시스템 구축의 여러 가지 차원을 영향을 주므로 파일 시스템을 설계하고 구축하는 작업이 더 복잡해집니다.

이러한 문제를 해결하기 위해 NetApp은 이러한 각 차원을 확장하는 데 사용되는 표준 구성 요소 아키텍처를 설계했습니다. 일반적으로 BeeGFS 빌딩 블록은 다음 세 가지 구성 프로파일 중 하나로 구축됩니다.

- BeeGFS 관리, 메타데이터 및 스토리지 서비스를 포함한 단일 기본 구성 요소입니다
- BeeGFS 메타데이터 및 스토리지 구성 요소입니다
- BeeGFS 스토리지 전용 구성 요소입니다

이 세 가지 옵션 간의 하드웨어 변경 사항은 BeeGFS 메타데이터에 더 작은 드라이브를 사용하는 것입니다. 그렇지 않으면 모든 구성 변경 사항이 소프트웨어를 통해 적용됩니다. 또한 Ansible을 구축 엔진으로 사용하면 특정 구성 요소에 대해 원하는 프로필을 설정하여 구성 작업을 간단하게 수행할 수 있습니다.

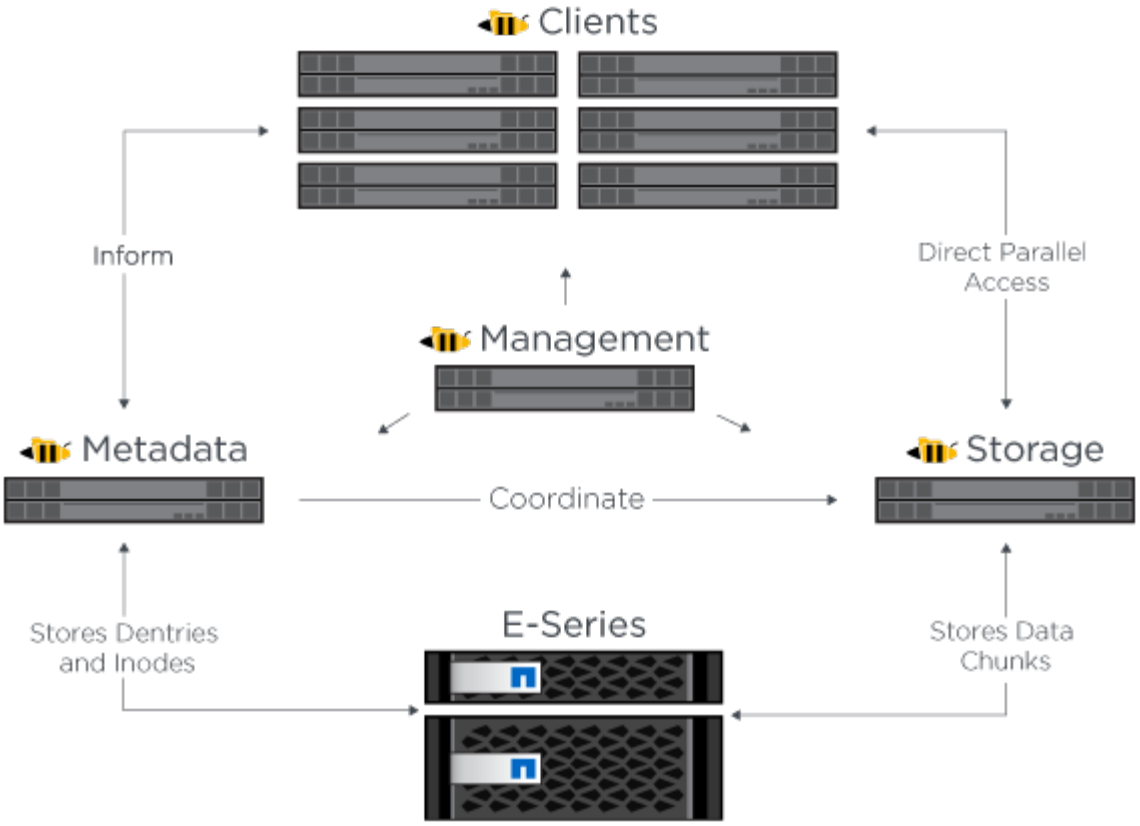
자세한 내용은 을 참조하십시오 검증된 하드웨어 설계.

파일 시스템 서비스

BeeGFS 파일 시스템에는 다음과 같은 주요 서비스가 포함됩니다.

- 관리 서비스.* 다른 모든 서비스를 등록하고 모니터링합니다.
- * 스토리지 서비스. * 데이터 청크 파일이라고 하는 분산 사용자 파일 콘텐츠를 저장합니다.
- * 메타데이터 서비스. * 파일 시스템 레이아웃, 디렉토리, 파일 특성 등을 추적합니다.
- * 클라이언트 서비스. * 파일 시스템을 마운트하여 저장된 데이터에 액세스합니다.

다음 그림에서는 NetApp E-Series 시스템에 사용된 BeeGFS 솔루션 구성 요소 및 관계를 보여 줍니다.



BeeGFS는 병렬 파일 시스템으로서 여러 서버 노드에서 파일을 스트라이핑하여 읽기/쓰기 성능과 확장성을 극대화합니다. 서버 노드는 함께 작동하여 일반적으로 _clients_라고 하는 다른 서버 노드에서 동시에 마운트하고 액세스할 수 있는 단일 파일 시스템을 제공합니다. 이러한 클라이언트는 NTFS, XFS 또는 ext4와 같은 로컬 파일 시스템과 유사하게 분산 파일 시스템을 보고 사용할 수 있습니다.

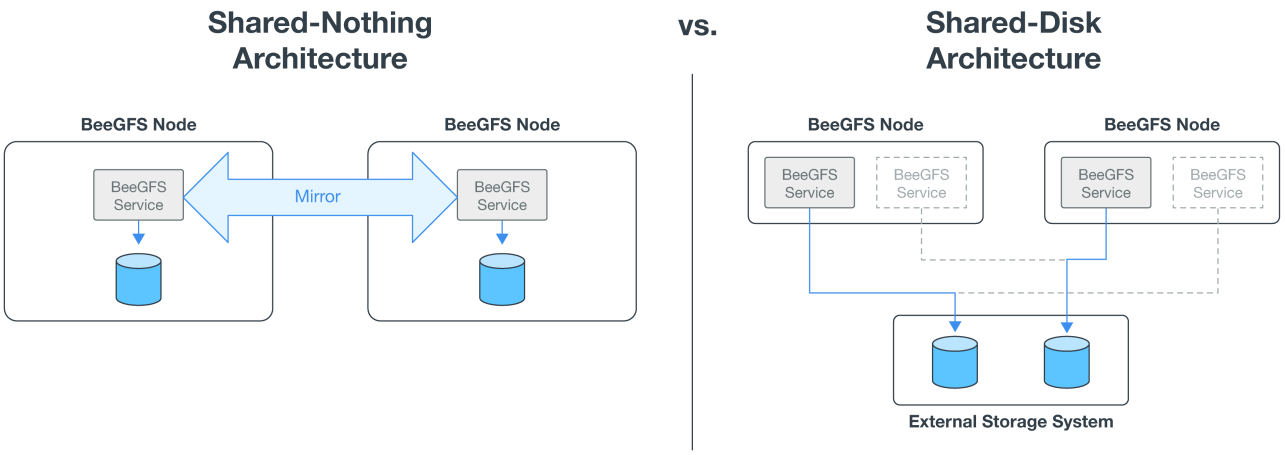
지원되는 다양한 Linux 배포판에서 4개의 기본 서비스를 실행하고 InfiniBand(IB), OPA(Omni-Path), RoCE(RDMA over Converged Ethernet)를 비롯한 TCP/IP 또는 RDMA 지원 네트워크를 통해 통신합니다. BeeGFS 서버 서비스(관리, 스토리지 및 메타데이터)는 사용자 공간 데몬이며, 클라이언트는 네이티브 커널 모듈(패치리스)입니다. 모든 구성요소를 재부팅하지 않고 설치 또는 업데이트할 수 있으며 동일한 노드에서 서비스 조합을 실행할 수 있습니다.

HA 아키텍처

NetApp 기반의 BeeGFS는 NetApp 하드웨어로 완벽하게 통합된 솔루션을 생성하여 공유 디스크 HA(고가용성) 아키텍처를 지원하여 BeeGFS 엔터프라이즈 에디션의 기능을 확장합니다.

BeeGFS 커뮤니티 에디션은 무료로 사용할 수 있지만, 이 엔터프라이즈 에디션은 NetApp과 같은 파트너로부터 프로페셔널 지원 구독 계약을 구매해야 합니다. Enterprise Edition에서는 복원력, 할당량 적용 및 스토리지 풀을 비롯한 몇 가지 추가 기능을 사용할 수 있습니다.

다음 그림에서는 공유 안 함 및 공유 디스크 HA 아키텍처를 비교하여 보여 줍니다.



자세한 내용은 을 참조하십시오 "NetApp에서 지원하는 BeeGFS에 대한 고가용성 발표".

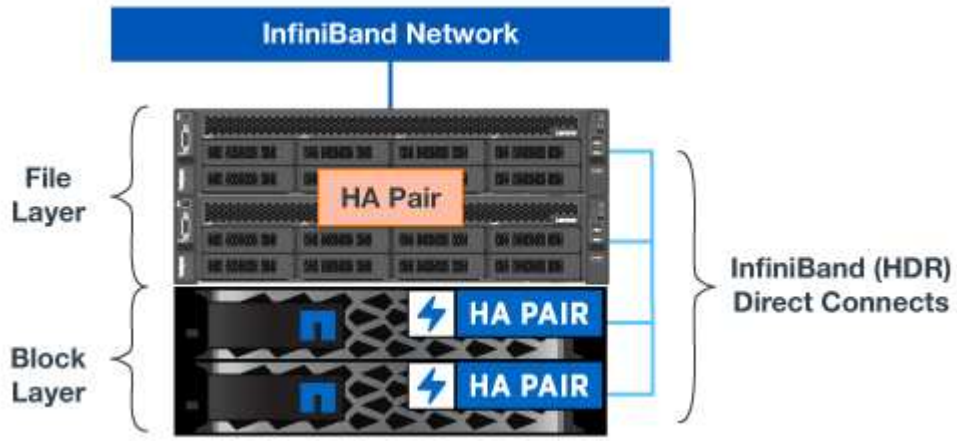
검증된 노드

NetApp 기반 BeeGFS 솔루션은 아래에 나열된 노드를 확인했습니다.

노드	하드웨어	세부 정보
블록	NetApp EF600 스토리지 시스템	까다로운 워크로드를 위해 설계된 고성능 All-NVMe 2U 스토리지 어레이입니다.
파일	Lenovo ThinkSystem SR665 V3 서버	PCIe 5.0, 듀얼 AMD EPYC 9124 프로세서를 탑재한 2소켓 2U 서버. Lenovo SR665 V3에 대한 자세한 내용은 를 참조하십시오 "Lenovo 웹 사이트".
	Lenovo ThinkSystem SR665 서버	PCIe 4.0, 듀얼 AMD EPYC 7003 프로세서가 탑재된 2소켓 2U 서버. Lenovo SR665에 대한 자세한 내용은 를 "Lenovo 웹 사이트"참조하십시오.

검증된 하드웨어 설계

솔루션의 구성 요소(다음 그림에 표시)는 BeeGFS 파일 계층에 대해 검증된 파일 노드 서버를 사용하고 EF600 스토리지 시스템 2개를 블록 계층으로 사용합니다.

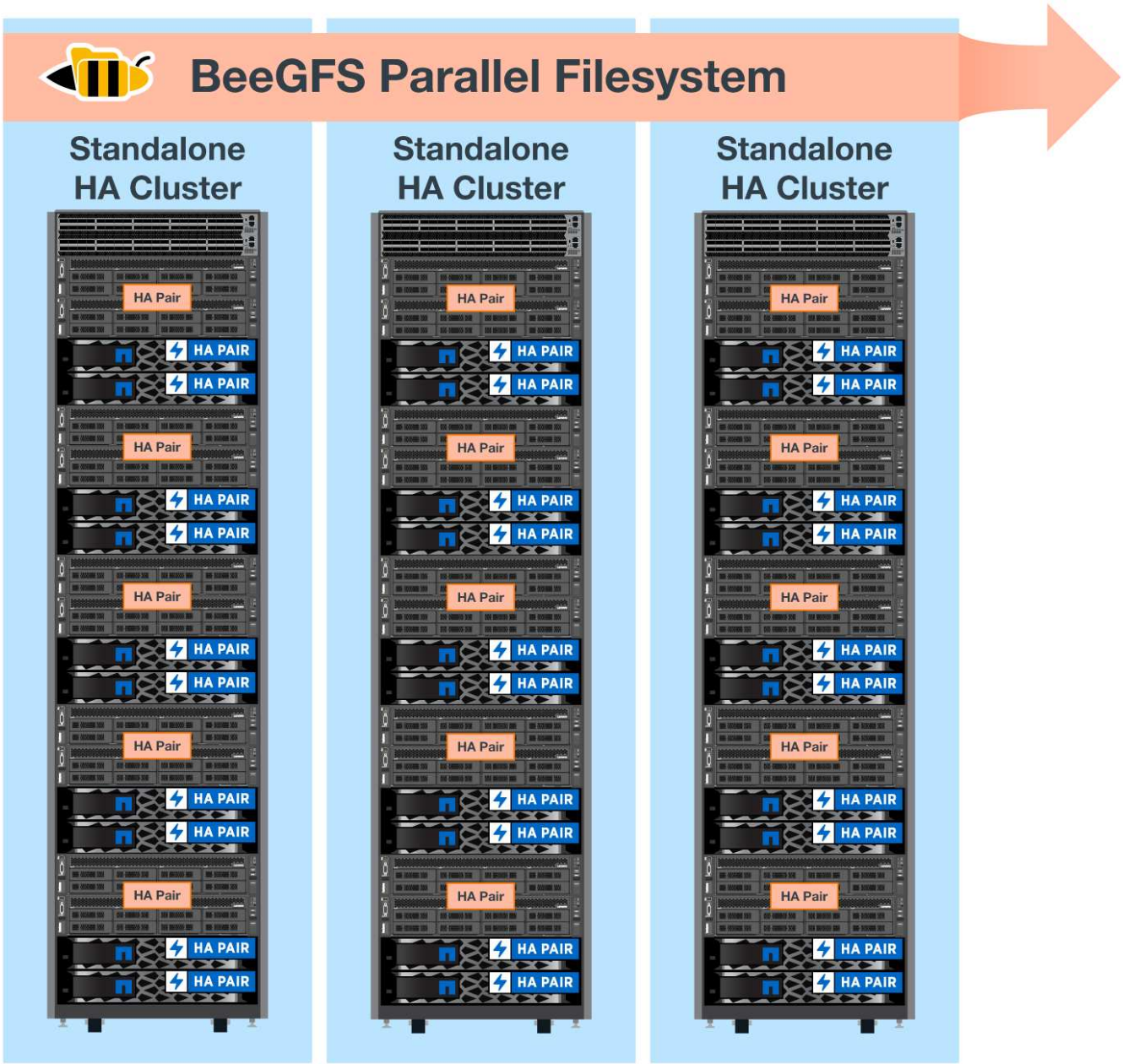


BeeGFS on NetApp 솔루션은 구축 환경의 모든 구성 요소에서 실행됩니다. 구축된 첫 번째 구성 요소는 BeeGFS 관리, 메타데이터 및 스토리지 서비스(기본 구성 요소라고 함)를 실행해야 합니다. 이후의 모든 구성 요소는

소프트웨어를 통해 메타데이터 및 스토리지 서비스를 확장하거나 스토리지 서비스만 제공하도록 구성할 수 있습니다. 이러한 모듈식 방식을 사용하면 동일한 기본 하드웨어 플랫폼 및 구성 요소 설계를 사용하면서 워크로드의 요구사항에 따라 파일 시스템을 확장할 수 있습니다.

최대 5개의 구성 요소를 구축하여 독립 실행형 Linux HA 클러스터를 구성할 수 있습니다. 따라서 심박조율기를 사용하여 리소스 관리가 최적화되고 Corosync와의 효율적인 동기화가 유지됩니다. 이러한 독립형 BeeGFS HA 클러스터 중 하나 이상이 결합되어 클라이언트가 단일 스토리지 네임스페이스로 액세스할 수 있는 BeeGFS 파일 시스템을 생성합니다. 하드웨어 측면에서 단일 42U 랙은 최대 5개의 빌딩 블록과 스토리지/데이터 네트워크용 1U InfiniBand 스위치 2개를 수용할 수 있습니다. 시각적 표현은 아래 그림을 참조하십시오.

장애 조치 클러스터에서 쿼럼을 설정하려면 최소 2개의 구성 요소가 필요합니다. 2노드 클러스터에는 성공적인 페일오버를 수행할 수 없는 제한 사항이 있습니다. 세 번째 장치를 Tiebreaker로 통합하여 2노드 클러스터를 구성할 수 있지만, 이 문서에서는 그러한 설계에 대해 설명하지 않습니다.



Ansible

NetApp 기반 BeeGFS는 GitHub 및 Ansible Galaxy(BeeGFS 컬렉션)에서 호스팅되는 Ansible 자동화를 통해 제공 및 구축됩니다 "Ansible 갤럭시" 및 "NetApp의 E-Series GitHub를 참조하십시오")를 클릭합니다. Ansible은 BeeGFS 구성 요소를 조립하는 데 사용되는 하드웨어에서 주로 테스트되지만, 지원되는 Linux 배포를 사용하여 거의 모든 x86 기반 서버에서 실행되도록 구성할 수 있습니다.

자세한 내용은 을 참조하십시오 "E-Series 스토리지를 통해 BeeGFS 구축".

기술 요구사항

NetApp 기반 BeeGFS 솔루션을 구현하려면 사용 환경이 본 문서에 설명된 기술 요구사항을 충족하는지 확인하십시오.

하드웨어 요구 사항

시작하기 전에 사용 중인 하드웨어가 NetApp 기반 BeeGFS 솔루션의 단일 2세대 구성 요소 설계에 대한 다음 사양을 충족하는지 확인하십시오. 특정 배포에 대한 정확한 구성 요소는 고객 요구 사항에 따라 다를 수 있습니다.

수량	하드웨어 구성 요소	요구 사항
2	BeeGFS 파일 노드	각 파일 노드가 권장 파일 노드의 사양을 충족하거나 초과해야 예상 성능을 얻을 수 있습니다. • 권장 파일 노드 옵션: * • * Lenovo ThinkSystem SR665 V3 * ◦ * 프로세서: * 2x AMD EPYC 9124 16C 3.0GHz(두 개의 NUMA 존으로 구성). ◦ * 메모리: * 256GB(16x 16GB TruDDR5 4800MHz RDIMM-A) ◦ * PCIe 확장: * PCIe Gen5 x16 슬롯 4개(NUMA 존당 2개) ◦ * 기타: * ▪ OS용 RAID 1에서 드라이브 2개(1TB 7.2K SATA 이상) ▪ 대역 내 OS 관리를 위한 1GbE 포트 ▪ 대역외 서버 관리를 위한 1GbE BMC 및 Redfish API ▪ 이중 핫 스왑 전원 공급 장치 및 성능 팬
2	E-Series 블록 노드(EF600 어레이)	• 메모리: * 256GB(컨트롤러당 128GB). * 어댑터: * 2포트 200GB/HDR(NVMe/IB). * 드라이브: * 원하는 메타데이터 및 스토리지 용량에 맞게 구성됩니다.

수량	하드웨어 구성 요소	요구 사항
8	InfiniBand 호스트 카드 어댑터(파일 노드용)	호스트 카드 어댑터는 파일 노드의 서버 모델에 따라 달라질 수 있습니다. 검증된 파일 노드에 대한 권장 사항은 다음과 같습니다. • * Lenovo ThinkSystem SR665 V3 서버: * ◦ MCX755106AS - 열 ConnectX-7, NDR200, QSFP112, 2포트, PCIe Gen5 x16, InfiniBand 어댑터
1	스토리지 네트워크 스위치	스토리지 네트워크 스위치는 200Gb/s InfiniBand 속도를 지원해야 합니다. 권장되는 스위치 모델은 다음과 같습니다. • * NVIDIA QM9700 Quantum 2 NDR InfiniBand 스위치 * • * NVIDIA MQM8700 Quantum HDR InfiniBand 스위치 *

케이블 요구 사항

- 블록 노드에서 파일 노드로 직접 연결. *

수량	부품 번호	길이
8	MCP1650-H001E30(NVIDIA 패시브 구리 케이블, QSFP56, 200GB/s)	1m

- 파일 노드에서 스토리지 네트워크 스위치로의 연결. * InfiniBand 스토리지 스위치에 따라 다음 표에서 해당 케이블 옵션을 선택합니다. + 권장 케이블 길이는 2m 이지만 고객의 환경에 따라 달라질 수 있습니다.

모델 전환	케이블 유형	수량	부품 번호
NVIDIA QM9700 를 참조하십시오	활성 파이버(트랜시버 포함)	2	MMA4Z00-NS(다중 모드, IB/ETH, 800GB/s 2x400Gb/s 2중 포트 OSFP)
		4	MFP7E20-Nxxx(다중 모드, 4채널 대 2채널 스플리터 파이버 케이블)
		8	MMA1Z00-NS400(다중 모드, IB/ETH, 400GB/s 단일 포트 QSFP-112)
	수동형 구리	2	MCP7Y40-N002(NVIDIA 패시브 구리 스플리터 케이블, InfiniBand 800GB/s ~ 4x 200GB/s, OSFP ~ 4x QSFP112)
NVIDIA MQM8700 를 참조하십시오	활성 파이버	8	MFS1S00-H003E(NVIDIA 활성 파이버 케이블, InfiniBand 200GB/s, QSFP56)
	수동형 구리	8	MCP1650-H002E26(NVIDIA 패시브 구리 케이블, InfiniBand 200GB/s, QSFP56)

소프트웨어 및 펌웨어 요구 사항

예측 가능한 성능 및 안정성을 보장하기 위해 NetApp 기반 BeeGFS 솔루션의 릴리스는 특정 버전의 소프트웨어 및 펌웨어 구성 요소를 사용하여 테스트했습니다. 이러한 버전은 솔루션을 구현하는 데 필요합니다.

파일 노드 요구 사항

소프트웨어	버전
Red Hat Enterprise Linux(RHEL)	고가용성(2소켓)을 갖춘 RHEL 9.4 물리적 서버. 참고**:** 파일 노드에는 유효한 Red Hat Enterprise Linux 서버 구독과 Red Hat Enterprise Linux 고가용성 애드온이 필요합니다.
Linux 커널	5.14.0-427.42.1.el9_4.x86_64
HCA 펌웨어	**ConnectX-7 HCA** 펌웨어 FW: 28.45.1200 + PXE: 3.7.0500 + UEFI: 14.38.0016 • ConnectX-6 HCA 펌웨어 * FW:20.43.2566 + PXE:3.7.0500 + UEFI:14.37.0013

EF600 블록 노드 요구사항

소프트웨어	버전
SANtricity OS를 참조하십시오	11.90R3
NVSRAM	N6000-890834-D02.DLP
드라이브 펌웨어	사용 중인 드라이브 모델에 대한 최신 버전입니다. 를 "E-Series 디스크 펌웨어 사이트입니다 "참조하십시오.

소프트웨어 배포 요구 사항

다음 표에는 Ansible 기반 BeeGFS 구축의 일부로 자동 구축되는 소프트웨어 요구사항이 나와 있습니다.

소프트웨어	버전
BeeGFS	7.4.6
Corosync 를 참조하십시오	3.1.8-1
심장박동기	2.1.7-5.2
PC(피씨)	0.11.7-2
펜스 에이전트(적목/APC)	4.10.0-62
InfiniBand/RDMA 드라이버	MLNX_OFED_Linux-23.10-3.2.2.1-LTS

Ansible 제어 노드 요구사항

NetApp 기반 BeeGFS 솔루션은 Ansible 제어 노드에서 구축 및 관리됩니다. 자세한 내용은 를 참조하십시오 "Ansible 설명서".

다음 표에 나와 있는 소프트웨어 요구사항은 아래 나열된 NetApp BeeGFS Ansible 컬렉션 버전과 관련이 있습니다.

소프트웨어	버전
Ansible	10.x를 참조하십시오
Ansible-코어	>= 2.13.0

소프트웨어	버전
파이썬	3.10
추가 Python 패키지	암호화 - 43.0.0, netaddr-1.3.0, ipaddr-2.2.0
NetApp E-Series BeeGFS Ansible 컬렉션	3.2.0

솔루션 설계를 검토합니다

설계 개요

BeeGFS 병렬 파일 시스템을 NetApp EF600 스토리지 시스템과 결합하는 NetApp 솔루션 기반 BeeGFS를 지원하려면 특정 장비, 케이블링, 구성이 필요합니다.

자세한 내용:

- "하드웨어 구성"
- "소프트웨어 구성"
- "설계 검증"
- "사이징 지침"
- "성능 튜닝"

설계 및 성능 면에서 다양한 파생 아키텍처:

- "고용량 빌딩 블록"

하드웨어 구성

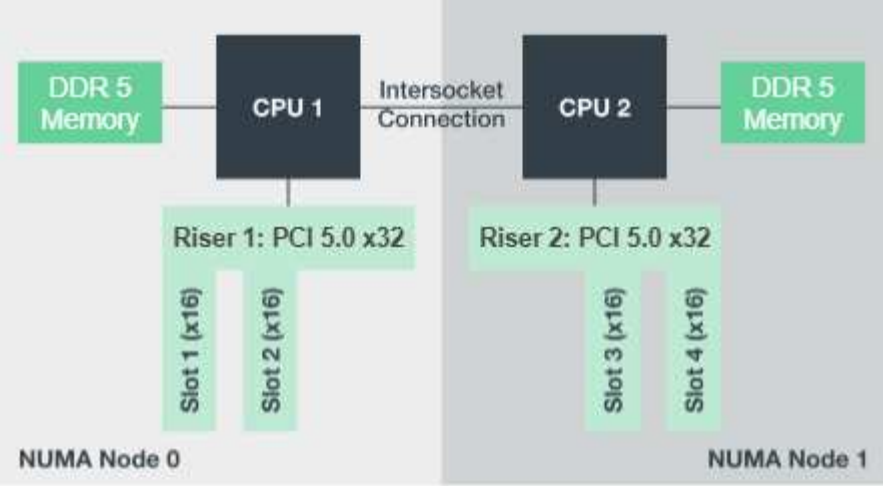
NetApp의 BeeGFS에 대한 하드웨어 구성에는 파일 노드 및 네트워크 케이블 연결이 포함됩니다.

파일 노드 구성

파일 노드에는 동일한 수의 PCIe 슬롯 및 메모리에 대한 로컬 액세스를 포함하는 별도의 NUMA 존으로 구성된 2개의 CPU 소켓이 있습니다.

InfiniBand 어댑터는 적절한 PCI 라이저 또는 슬롯에 설치되어야 사용 가능한 PCIe 레인 및 메모리 채널에 걸쳐 작업 부하가 분산됩니다. 개별 BeeGFS 서비스에 대한 작업을 특정 NUMA 노드에 완전히 격리하여 워크로드의 균형을 조정합니다. 목표는 두 개의 독립적인 단일 소켓 서버처럼 각 파일 노드에서 유사한 성능을 얻는 것입니다.

다음 그림에서는 파일 노드 NUMA 구성을 보여 줍니다.



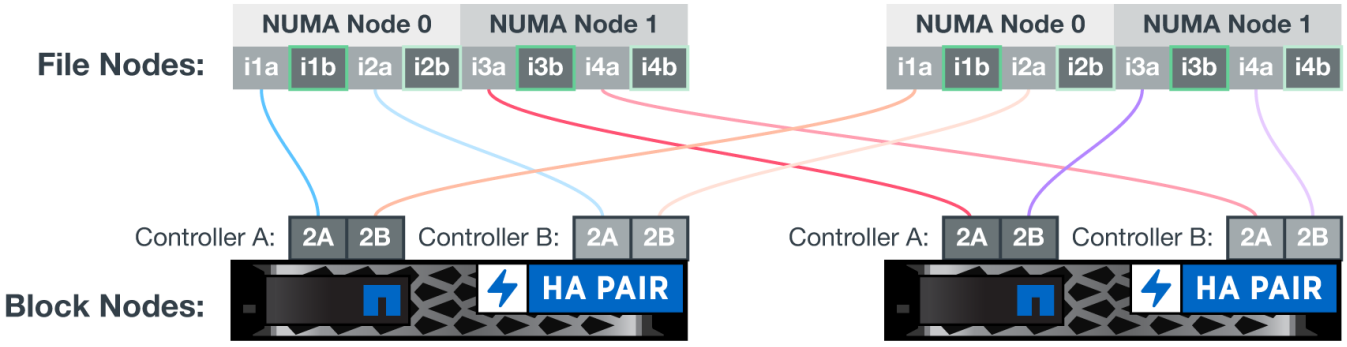
BeeGFS 프로세스는 사용된 인터페이스가 동일한 존에 있도록 특정 NUMA 존에 고정됩니다. 이렇게 구성하면 소켓 간 연결을 통한 원격 액세스가 필요하지 않습니다. 소켓 간 연결은 QPI 또는 GMI2 링크라고도 합니다. 최신 프로세서 아키텍처에서도 HDR InfiniBand와 같은 고속 네트워킹을 사용할 때 병목 현상이 발생할 수 있습니다.

네트워크 케이블 연결 구성

구성 요소 내에서 각 파일 노드는 총 4개의 중복 InfiniBand 연결을 사용하여 2개의 블록 노드에 연결됩니다. 또한 각 파일 노드에는 InfiniBand 스토리지 네트워크에 대한 4개의 이중화된 접속이 있습니다.

다음 그림에서 주목하십시오.

- 녹색으로 표시된 모든 파일 노드 포트는 스토리지 패브릭에 접속하는 데 사용되며, 다른 모든 파일 노드 포트는 블록 노드에 직접 연결됩니다.
- 특정 NUMA 존에 있는 2개의 InfiniBand 포트는 동일한 블록 노드의 A 및 B 컨트롤러에 연결됩니다.
- NUMA 노드 0의 포트는 항상 첫 번째 블록 노드에 연결됩니다.
- NUMA 노드 1의 포트는 두 번째 블록 노드에 연결됩니다.



Splitter 케이블을 사용하여 스토리지 스위치를 파일 노드에 연결하는 경우 케이블 하나가 분기되어 밝은 녹색으로 표시된 포트에 연결해야 합니다. 다른 케이블이 나와서 어두운 녹색으로 표시된 포트에 연결해야 합니다. 또한 중복 스위치가 있는 스토리지 네트워크의 경우 연한 녹색으로 표시된 포트가 하나의 스위치에 연결되고 진한 녹색의 포트는 다른 스위치에 연결해야 합니다.

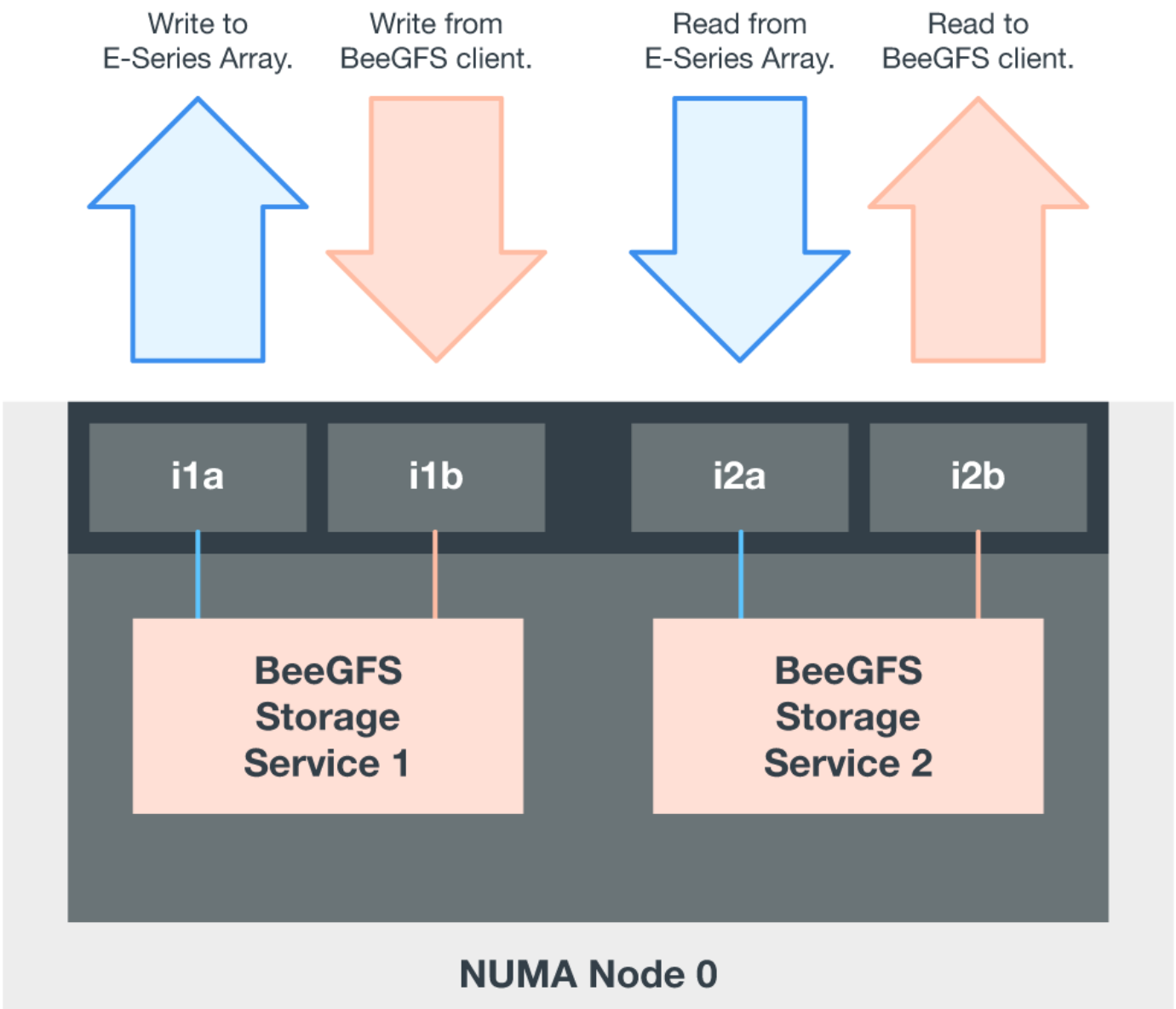
그림에 표시된 케이블 연결 구성을 통해 각 BeeGFS 서비스는 다음을 수행할 수 있습니다.

- BeeGFS 서비스를 실행 중인 파일 노드에 관계없이 동일한 NUMA 존에서 실행합니다.
- 장애 발생 위치에 관계없이 프런트엔드 스토리지 네트워크와 백엔드 블록 노드에 대한 보조 최적 경로 제공
- 블록 노드의 파일 노드 또는 컨트롤러에 유지 관리가 필요한 경우 성능 영향을 최소화합니다.

대역폭을 활용하기 위한 케이블 연결

PCIe 양방향 대역폭을 최대한 활용하려면 각 InfiniBand 어댑터의 포트 하나를 스토리지 패브릭에 연결하고 다른 포트는 블록 노드에 연결해야 합니다.

다음 그림은 전체 PCIe 양방향 대역폭을 활용하는 데 사용되는 케이블링 설계를 보여줍니다.



각 BeeGFS 서비스에 대해 동일한 어댑터를 사용하여 클라이언트 트래픽에 사용되는 기본 포트를 해당 서비스 볼륨의 기본 소유자인 블록 노드 컨트롤러의 경로와 연결합니다. 자세한 내용은 을 참조하십시오 "소프트웨어 구성".

소프트웨어 구성

NetApp 기반 BeeGFS에 대한 소프트웨어 구성에는 BeeGFS 네트워크 구성 요소, EF600 블록

노드, BeeGFS 파일 노드, 리소스 그룹, BeeGFS 서비스가 포함됩니다.

BeeGFS 네트워크 구성

BeeGFS 네트워크 구성은 다음과 같은 구성 요소로 이루어집니다.

- * 부동 IP * 부동 IP는 동일한 네트워크의 모든 서버로 동적으로 라우팅될 수 있는 일종의 가상 IP 주소입니다. 여러 서버가 동일한 부동 IP 주소를 소유할 수 있지만, 특정 시간에 한 서버에서만 활성화될 수 있습니다.

  각 BeeGFS 서버 서비스에는 BeeGFS 서버 서비스의 실행 위치에 따라 파일 노드 간에 이동할 수 있는 고유한 IP 주소가 있습니다. 이러한 부동 IP 구성을 통해 각 서비스가 다른 파일 노드로 독립적으로 페일오버할 수 있습니다. 클라이언트는 특정 BeeGFS 서비스의 IP 주소를 알고 있으면 됩니다. 이 경우 현재 해당 서비스를 실행 중인 파일 노드를 알 필요가 없습니다.

- * BeeGFS 서버 다중 홈 구성 * 솔루션의 밀도를 높이기 위해 각 파일 노드에는 동일한 IP 서브넷에 구성된 IP를 가진 여러 스토리지 인터페이스가 있습니다.

  기본적으로 하나의 인터페이스에 대한 요청은 동일한 서브넷에 있는 경우 다른 인터페이스에서 응답할 수 있기 때문에 Linux 네트워킹 스택에서 이 구성이 예상대로 작동하는지 확인하기 위해 추가 구성이 필요합니다. 다른 단점 외에도 이 기본 동작으로 인해 RDMA 연결을 적절하게 설정하거나 유지할 수 없습니다.

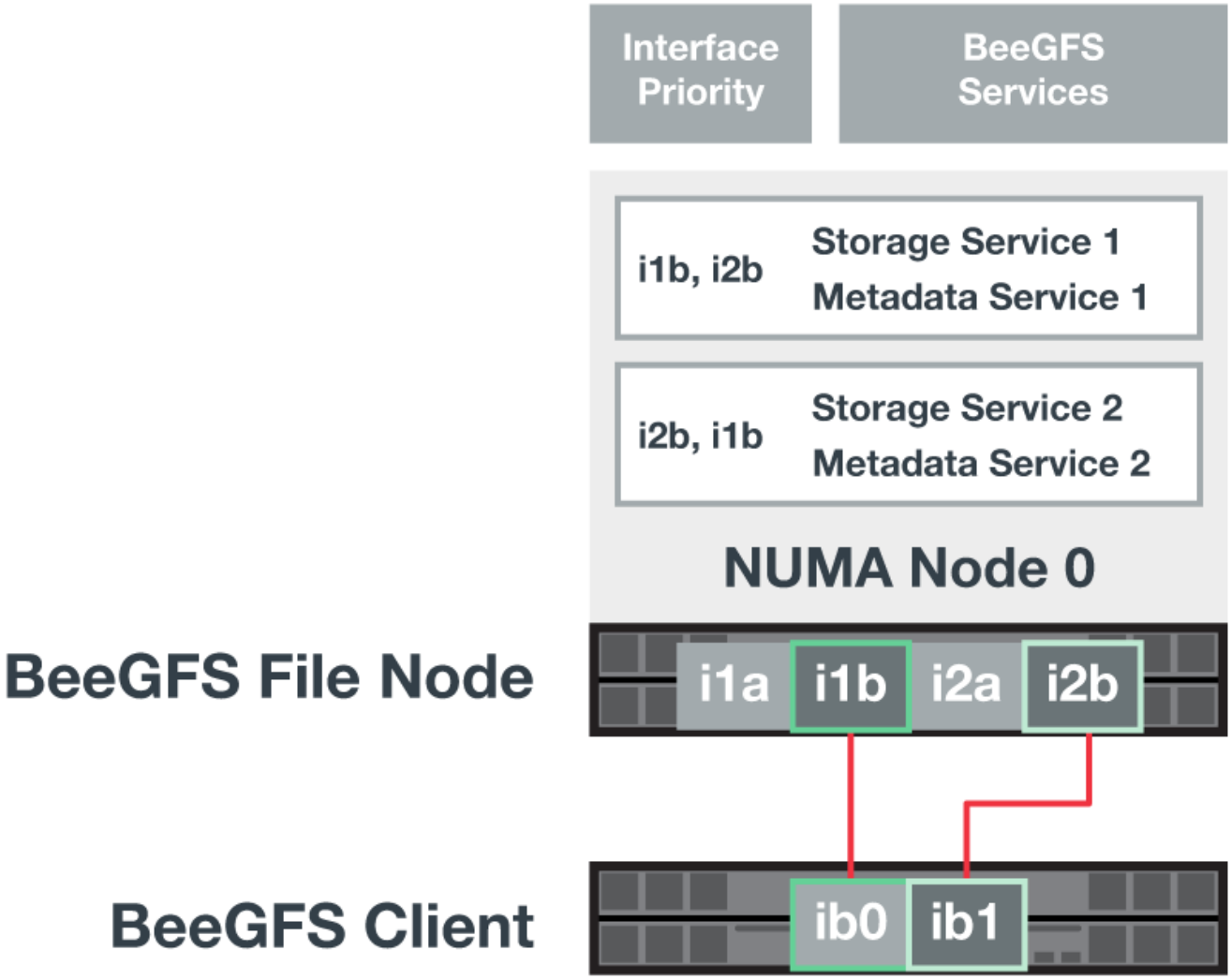
  Ansible 기반 배포에서는 부동 IP가 시작 및 중지되는 시기와 함께 RP(역방향 경로) 및 ARP(주소 해상도 프로토콜) 동작 조임을 처리하고, 다중 홈 네트워크 구성이 제대로 작동하도록 해당 IP 경로 및 규칙을 동적으로 생성합니다.

- * BeeGFS 클라이언트 다중 레일 구성 * _Multi-rail_은 응용 프로그램이 여러 개의 독립 네트워크 연결 또는 "레일"을 사용하여 성능을 향상시키는 기능을 말합니다.

  BeeGFS는 단일 IPoIB 서브넷에서 여러 IB 인터페이스를 사용할 수 있도록 멀티레일 지원을 구현합니다. 이 기능을 통해 RDMA NIC 간에 동적 로드 밸런싱을 수행하여 네트워크 리소스 사용을 최적화할 수 있습니다. 또한 NVIDIA GDS(GPUDirect Storage)와 통합되어 시스템 대역폭을 높이고 클라이언트 CPU의 지연 시간 및 사용률을 줄여줍니다.

  이 문서에서는 단일 IPoIB 서브넷 구성에 대한 지침을 제공합니다. 이중 IPoIB 서브넷 구성이 지원되지만 단일 서브넷 구성과 동일한 이점을 제공하지 않습니다.

  다음 그림에서는 여러 BeeGFS 클라이언트 인터페이스에서 트래픽의 균형을 조정하는 방법을 보여 줍니다.



BeeGFS의 각 파일은 일반적으로 여러 스토리지 서비스에 걸쳐 스트라이핑되기 때문에 다중 레일 구성을 통해 클라이언트는 단일 InfiniBand 포트보다 더 많은 처리량을 달성할 수 있습니다. 예를 들어, 다음 코드 샘플은 클라이언트가 두 인터페이스 간에 트래픽의 균형을 조정할 수 있도록 하는 일반적인 파일 스트라이핑 구성을 보여 줍니다.

를 누릅니다

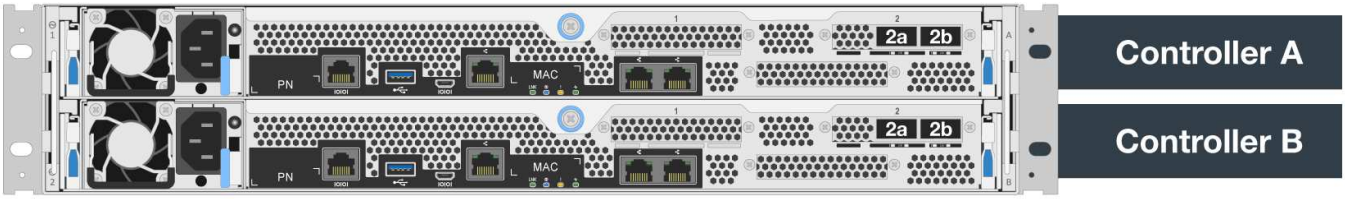
```
root@beegfs01:/mnt/beegfs# beegfs-ctl --getentryinfo myfile
Entry type: file
EntryID: 11D-624759A9-65
Metadata node: meta_01_tgt_0101 [ID: 101]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 1M
+ Number of storage targets: desired: 4; actual: 4
+ Storage targets:
  + 101 @ stor_01_tgt_0101 [ID: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

EF600 블록 노드 구성

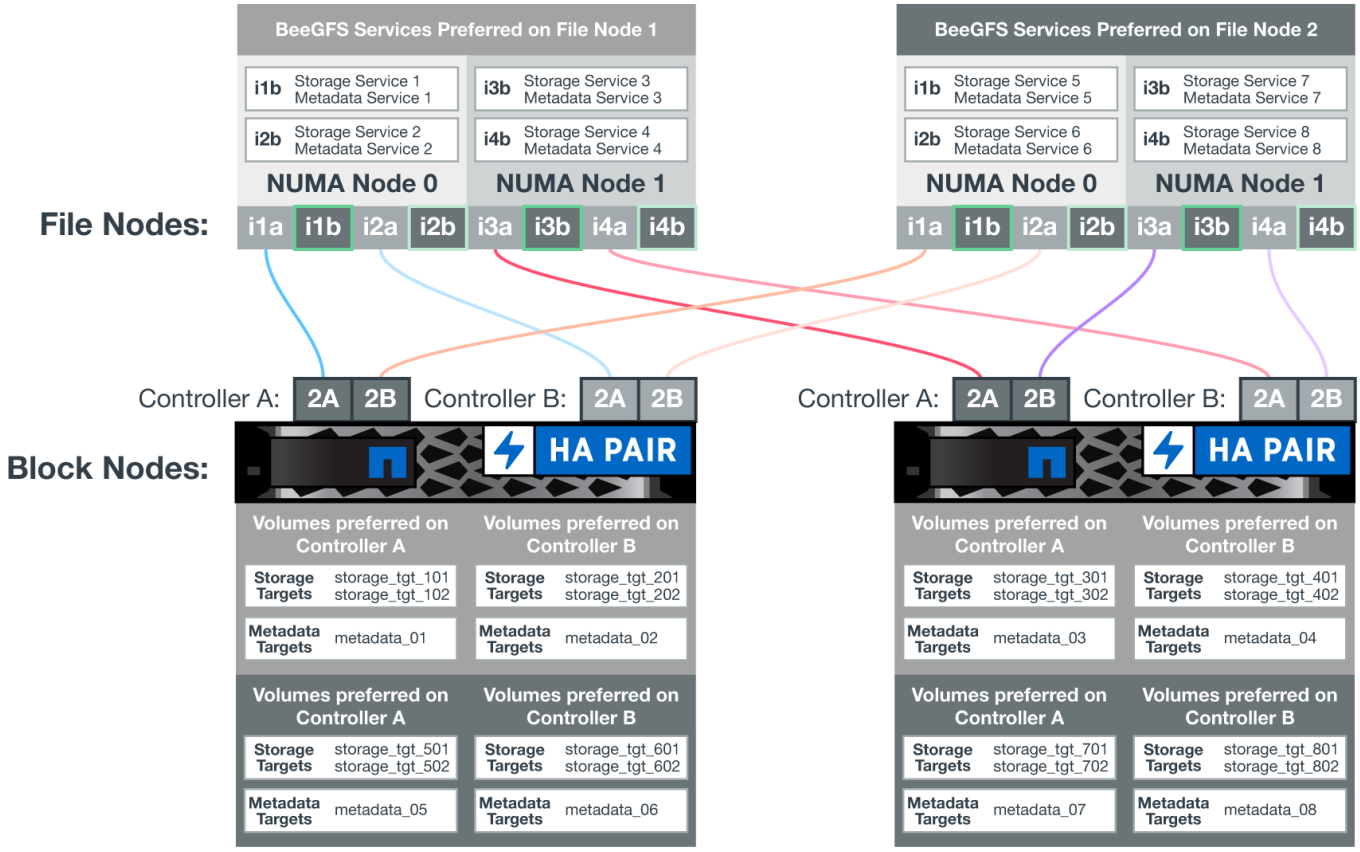
블록 노드는 동일한 드라이브 세트에 대한 공유 액세스를 가진 2개의 액티브/액티브 RAID 컨트롤러로 구성됩니다. 일반적으로 각 컨트롤러는 시스템에 구성된 볼륨의 절반을 소유하지만 필요에 따라 다른 컨트롤러를 인수할 수 있습니다.

파일 노드의 다중 경로 소프트웨어는 각 볼륨에 대한 최적화된 활성 경로를 결정하고 케이블, 어댑터 또는 컨트롤러에 장애가 발생할 경우 대체 경로로 자동으로 이동합니다.

다음 다이어그램은 EF600 블록 노드의 컨트롤러 레이아웃을 보여 줍니다.



공유 디스크 HA 솔루션을 지원하기 위해 볼륨은 두 파일 노드에 매핑되므로 필요에 따라 서로 테이크오버할 수 있습니다. 다음 다이어그램은 BeeGFS 서비스 및 기본 볼륨 소유권이 최대 성능을 위해 구성되는 방법의 예를 보여 줍니다. 각 BeeGFS 서비스 왼쪽에 있는 인터페이스는 클라이언트 및 기타 서비스가 연락하는 데 사용하는 기본 인터페이스를 나타냅니다.



앞의 예에서 클라이언트 및 서버 서비스는 인터페이스 i1b를 사용하여 스토리지 서비스 1과 통신하는 것을 선호합니다. 스토리지 서비스 1은 인터페이스 i1a를 기본 경로로 사용하여 첫 번째 블록 노드의 컨트롤러 A에 있는 해당 볼륨(storage_tgt_101, 102)과 통신합니다. 이 구성은 InfiniBand 어댑터에서 사용할 수 있는 양방향 PCIe 대역폭을 완벽하게 사용하고 PCIe 4.0에서 사용할 수 있는 것보다 듀얼 포트 HDR InfiniBand 어댑터에서 더 나은 성능을 제공합니다.

BeeGFS 파일 노드 구성

BeeGFS 파일 노드는 HA(High-Availability) 클러스터로 구성되어 여러 파일 노드 간에 BeeGFS 서비스의 페일오버를 지원합니다.

HA 클러스터 설계는 널리 사용되는 두 가지 Linux HA 프로젝트, 즉 클러스터 멤버십을 위한 Corosync 및 클러스터 리소스 관리를 위한 Pacemaker를 기반으로 합니다. 자세한 내용은 을 "고가용성 애드온을 위한 Red Hat 교육 "참조하십시오.

NetApp은 클러스터가 지능적으로 BeeGFS 리소스를 시작하고 모니터링할 수 있도록 여러 OCF(Open Cluster Framework) 리소스 에이전트를 저술하고 확장했습니다.

BeeGFS HA 클러스터

일반적으로, HA를 사용하거나 사용하지 않고 BeeGFS 서비스를 시작할 때 다음과 같은 몇 가지 리소스를 사용해야 합니다.

- 서비스에 연결할 수 있는 IP 주소이며 일반적으로 Network Manager에서 구성합니다.
- BeeGFS에서 데이터를 저장하기 위한 타겟으로 사용되는 기본 파일 시스템입니다.

  일반적으로 이러한 항목은 '/etc/fstab'에 정의되어 있으며 systemd에 의해 마운트됩니다.

- 다른 리소스가 준비되면 BeeGFS 프로세스를 시작하는 시스템 서비스입니다.

  추가 소프트웨어가 없으면 이러한 리소스는 단일 파일 노드에서만 시작됩니다. 따라서 파일 노드가 오프라인이 되면 BeeGFS 파일 시스템의 일부를 액세스할 수 없습니다.

여러 노드에서 각 BeeGFS 서비스를 시작할 수 있으므로, Pacemaker는 각 서비스와 종속 리소스가 한 번에 하나의 노드에서만 실행되도록 해야 합니다. 예를 들어, 두 노드가 동일한 BeeGFS 서비스를 시작하려고 하면 둘 다 기본 타겟의 동일한 파일에 쓰려고 하면 데이터 손상이 발생할 위험이 있습니다. 이러한 시나리오를 피하기 위해, 페이스메이커의 Corosync를 사용하여 전체 클러스터의 상태를 모든 노드에 걸쳐 안정적으로 유지하고 쿼럼을 설정합니다.

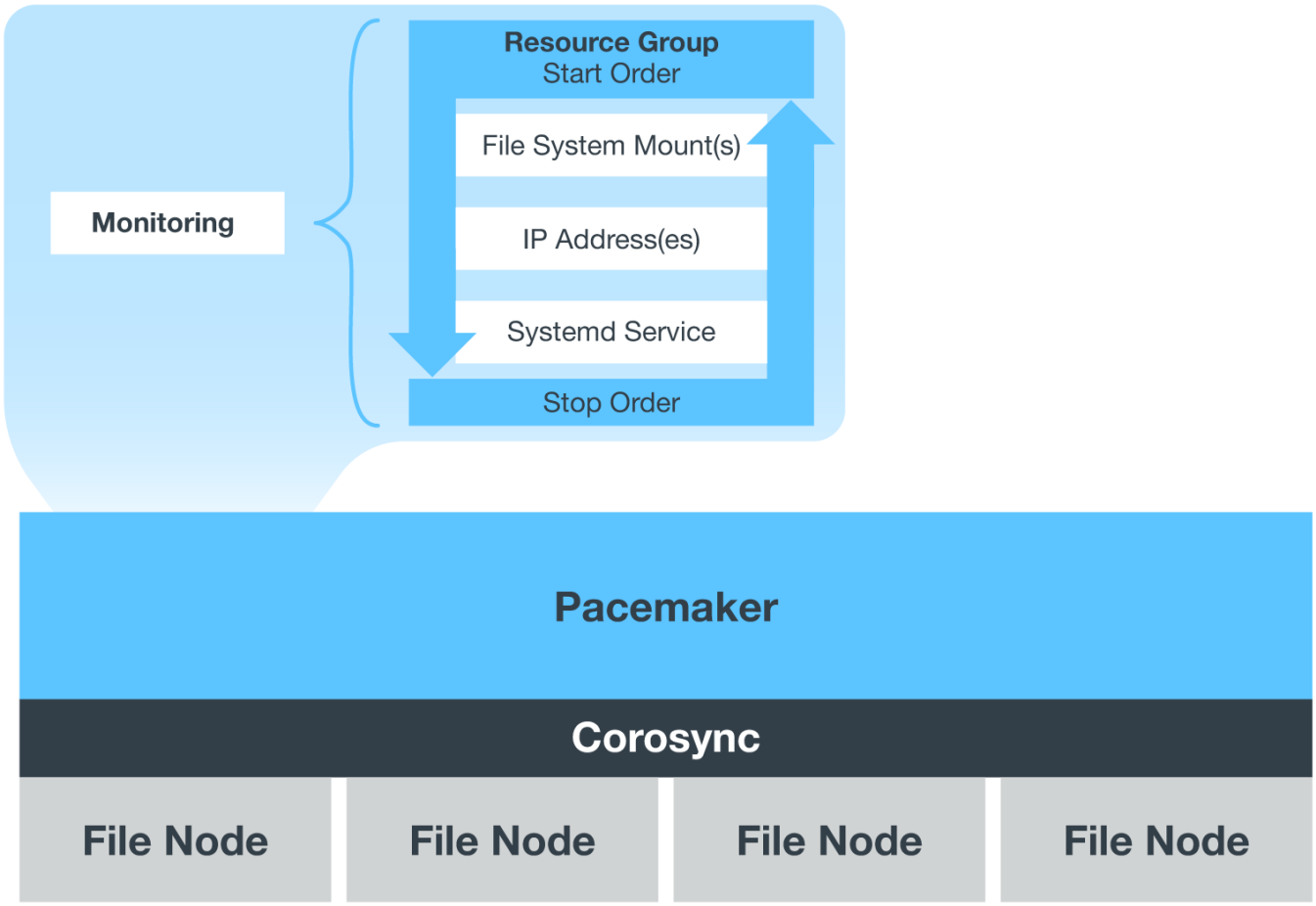
클러스터에서 장애가 발생하면 심장박동기가 반응하여 다른 노드에서 BeeGFS 리소스를 다시 시작합니다. 일부 시나리오에서는 심박조율기가 장애가 발생한 원래 노드와 통신하지 못하여 리소스가 중지되었는지 확인할 수 없습니다. 다른 곳에서 BeeGFS 리소스를 다시 시작하기 전에 노드가 다운되었는지 확인하려면 심장박동기가 장애가 있는 노드를 분리합니다. 즉, 전원을 제거하는 것이 좋습니다.

심박조율기가 PDU(Power Distribution Unit)를 사용하여 노드를 펜싱하거나 서버 BMC(Baseboard Management Controller)를 Redfish와 같은 API와 함께 사용하여 오픈 소스 펜싱 에이전트를 많이 사용할 수 있습니다.

BeeGFS가 HA 클러스터에서 실행 중인 경우 모든 BeeGFS 서비스 및 기본 리소스는 리소스 그룹의 페이스 메이커를 통해 관리됩니다. 각 BeeGFS 서비스 및 해당 서비스가 의존하는 리소스가 리소스 그룹으로 구성되어 리소스가 올바른 순서로 시작 및 중지되어 동일한 노드에 배치됩니다.

각 BeeGFS 리소스 그룹에 대해 심장박동기는 특정 노드에서 BeeGFS 서비스에 더 이상 액세스할 수 없을 때 장애 조건을 감지하고 페일오버를 지능적으로 트리거하는 사용자 지정 BeeGFS 모니터링 리소스를 실행합니다.

다음 그림에서는 심장박동기 제어 BeeGFS 서비스 및 종속성을 보여 줍니다.



동일한 유형의 여러 BeeGFS 서비스를 동일한 노드에서 시작할 수 있도록 다중 모드 구성 방법을 사용하여 BeeGFS 서비스를 시작하도록 페이스 메이커를 구성합니다. 자세한 내용은 를 참조하십시오 "멀티 모드에 대한 BeeGFS 문서".

BeeGFS 서비스는 여러 노드에서 시작할 수 있어야 하므로 각 서비스의 구성 파일('/etc/beegfs'에 있음)은 해당 서비스의 BeeGFS 타겟으로 사용되는 E-Series 볼륨 중 하나에 저장됩니다. 따라서 특정 BeeGFS 서비스에 대한 데이터와 함께 서비스를 실행해야 하는 모든 노드에서 해당 구성을 액세스할 수 있습니다.

```
# 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
```

설계 검증

BeeGFS on NetApp 솔루션의 2세대 설계는 3가지 구성 요소 구성 프로필을 사용하여 검증되었습니다.

구성 프로파일에는 다음이 포함됩니다.

- BeeGFS 관리, 메타데이터 및 스토리지 서비스를 포함한 단일 기본 구성 요소입니다.
- BeeGFS 메타데이터와 스토리지 구성 요소
- BeeGFS 스토리지 전용 구성 요소입니다.

빌딩 블록은 2개의 NVIDIA Quantum InfiniBand(MQM8700) 스위치에 연결되었습니다. 10개의 BeeGFS 클라이언트도 InfiniBand 스위치에 연결되었으며 통합 벤치마크 유틸리티를 실행하는 데 사용되었습니다.

다음 그림에서는 NetApp 솔루션의 BeeGFS 검증을 위해 사용되는 BeeGFS 구성을 보여 줍니다.

BeeGFS 파일 스트라이핑

병렬 파일 시스템의 이점은 여러 스토리지 대상 간에 개별 파일을 스트라이핑하는 기능입니다. 이 기능은 동일하거나 다른 기본 스토리지 시스템의 볼륨을 나타낼 수 있습니다.

BeeGFS에서는 디렉토리 및 파일별로 스트라이핑을 구성하여 각 파일에 사용되는 타겟 수를 제어하고 각 파일 스트라이프에 사용되는 청크 크기(또는 블록 크기)를 제어할 수 있습니다. 따라서 서비스를 재구성하거나 다시 시작할 필요 없이 파일 시스템에서 다양한 유형의 워크로드와 I/O 프로필을 지원할 수 있습니다. "begfs -ctl" 명령줄 도구 또는 스트라이핑 API를 사용하는 응용 프로그램을 사용하여 스트라이프 설정을 적용할 수 있습니다. 자세한 내용은 의 BeeGFS 설명서를 참조하십시오 "스트라이핑" 및 "스트라이핑 API".

최상의 성능을 얻기 위해 테스트 중에 스트라이프 패턴을 조정하고 각 테스트에 사용된 매개 변수를 기록하였습니다.

IOR 대역폭 테스트**:** 여러 클라이언트

IOR 대역폭 테스트는 OpenMPI를 사용하여 합성 I/O 생성기 툴 IOR(에서 제공)의 병렬 작업을 실행했습니다 "HPC GitHub를 참조하십시오") 10개 클라이언트 노드 전체에서 하나 이상의 BeeGFS 구성 요소로 이동합니다. 달리 명시되지 않은 한:

- 모든 테스트는 1MiB 전송 크기의 직접 I/O를 사용했습니다.
- BeeGFS 파일 스트라이핑은 1MB 청크 크기 및 파일당 하나의 타겟으로 설정되었습니다.

다음 매개 변수는 IOR에 사용되었습니다. 세그먼트 수는 구성 요소 1개의 경우 애그리게이트 파일 크기를 5TiB로, 3개의 구성 요소는 40TiB로 유지하도록 조정되었습니다.

```
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
```

하나의 **BeeGFS** 기본(관리, 메타데이터 및 스토리지) 구성 요소입니다

다음 그림에서는 단일 BeeGFS 기반(관리, 메타데이터 및 스토리지) 구성 요소를 사용한 IOR 테스트 결과를 보여 줍니다.

BeeGFS 메타데이터 **+** 스토리지 구성 요소입니다

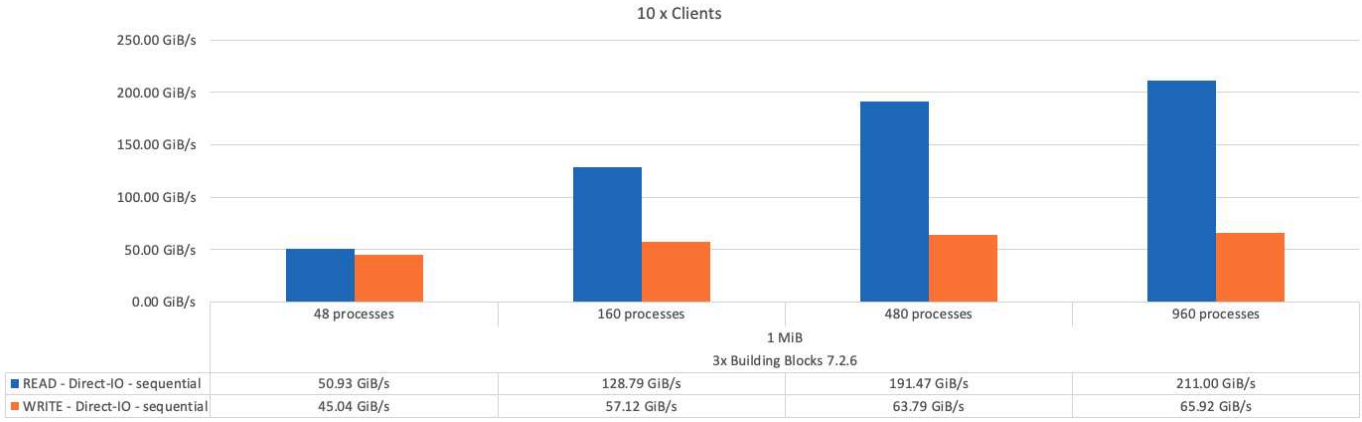
다음 그림에서는 단일 BeeGFS 메타데이터 + 스토리지 구성 요소를 사용한 IOR 테스트 결과를 보여 줍니다.

BeeGFS 스토리지 전용 구성 요소입니다

다음 그림에서는 단일 BeeGFS 스토리지 전용 구성 요소를 사용한 IOR 테스트 결과를 보여 줍니다.

BeeGFS 빌딩 블록 **3**개

다음 그림에서는 세 개의 BeeGFS 구성 요소가 포함된 IOR 테스트 결과를 보여 줍니다.



예상한 대로 기본 구성 요소과 후속 메타데이터 + 스토리지 구성 요소 간의 성능 차이는 무시할 수 있습니다. 메타데이터 + 스토리지 구성 요소 및 스토리지 전용 구성 요소를 비교하면 스토리지 대상으로 사용되는 추가 드라이브로 인해 읽기 성능이 약간 향상됩니다. 그러나 쓰기 성능에는 큰 차이가 없습니다. 더 높은 성능을 얻기 위해 여러 구성 요소를 함께 추가하여 성능을 선형 방식으로 확장할 수 있습니다.

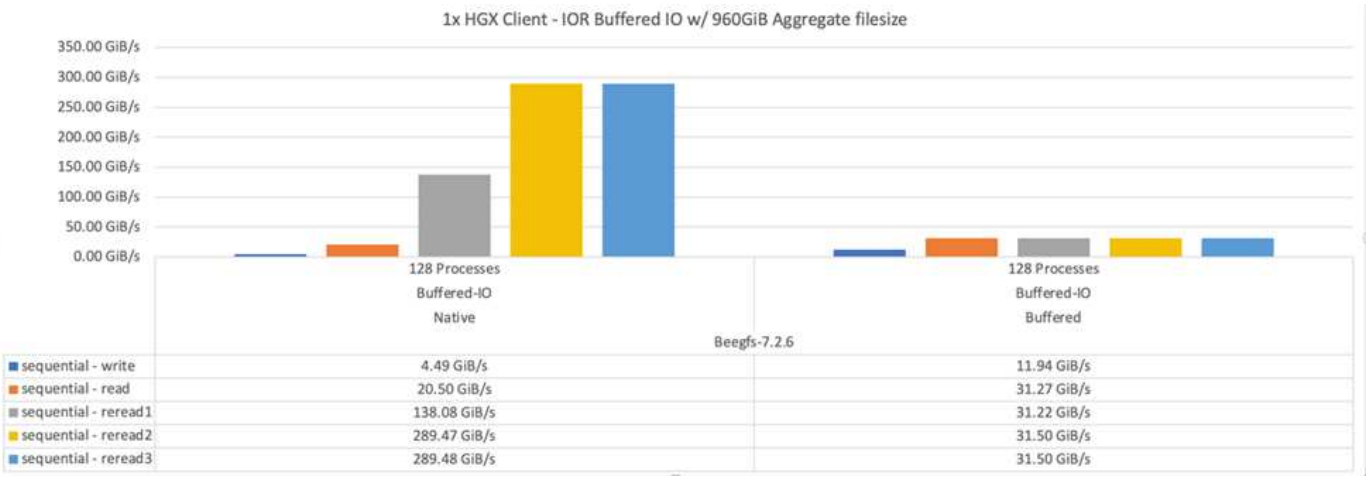
IOR 대역폭 테스트**:** 단일 클라이언트

IOR 대역폭 테스트는 OpenMPI를 사용하여 단일 고성능 GPU 서버를 사용하여 여러 IOR 프로세스를 실행하여 단일 클라이언트에서 얻을 수 있는 성능을 탐색했습니다.

이 테스트는 클라이언트가 Linux 커널 페이지 캐시('tuneFileCacheType=NATIVE')를 사용하도록 구성된 경우 BeeGFS의 다시 읽기 동작 및 성능을 기본 '버퍼링' 설정과 비교합니다.

네이티브 캐싱 모드는 클라이언트의 Linux 커널 페이지 캐시를 사용하므로 네트워크를 통해 다시 전송되는 것이 아니라 로컬 메모리에서 다시 읽기 작업을 수행할 수 있습니다.

다음 다이어그램은 BeeGFS 빌딩 블록 3개와 단일 클라이언트를 사용한 IOR 테스트 결과를 보여 줍니다.



이러한 테스트를 위한 BeeGFS 스트라이핑은 파일당 타겟 8개가 포함된 1MB 청크 크기로 설정되었습니다.

기본 버퍼링 모드에서 쓰기 및 초기 읽기 성능이 향상되지만, 동일한 데이터를 여러 번 다시 읽는 워크로드의 경우 네이티브 캐싱 모드에서 성능이 크게 향상됩니다. 이렇게 향상된 다시 읽기 성능은 여러 번의 Epoch에서 동일한 데이터 세트를 여러 번 다시 읽는 딥 러닝과 같은 워크로드에 중요합니다.

메타데이터 성능 테스트

Metadata 성능 테스트는 IOR의 일부로 포함된 MDTest 도구를 사용하여 BeeGFS의 메타데이터 성능을 측정했습니다. 이 테스트에서는 OpenMPI를 사용하여 10개의 클라이언트 노드 모두에서 병렬 작업을 실행했습니다.

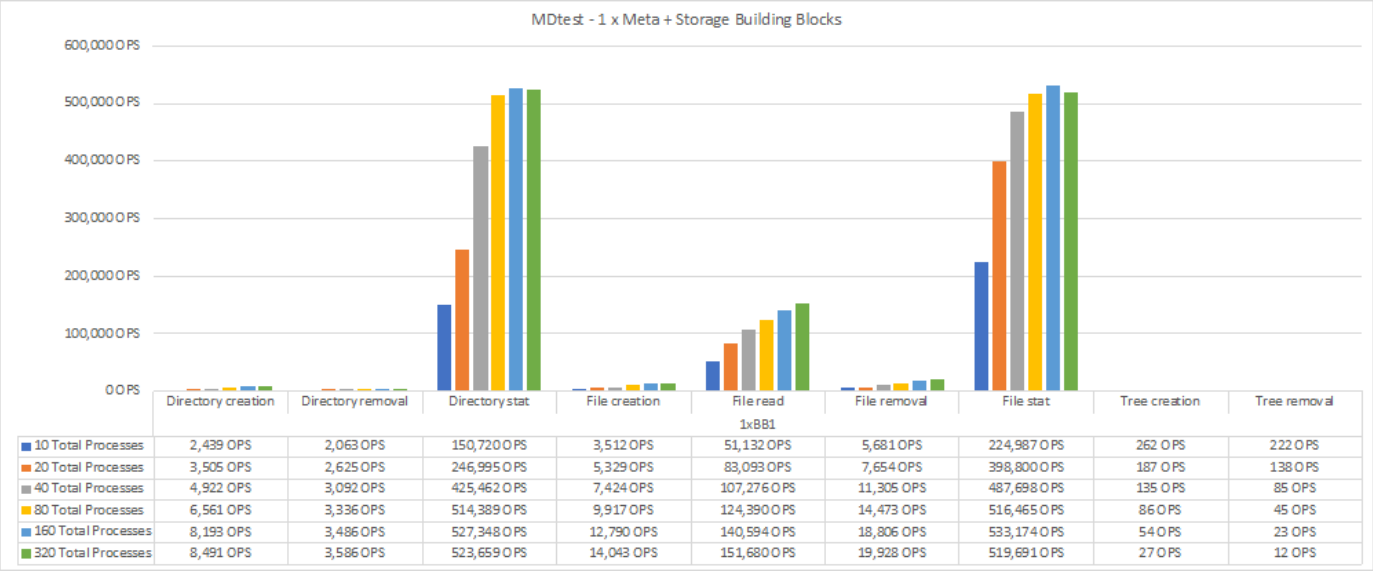
다음 매개 변수는 총 프로세스 수가 2배속 단계에서 10개에서 320으로 조정되고 파일 크기가 4K인 벤치마크 테스트를 실행하는 데 사용되었습니다.

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

메타데이터 성능은 먼저 메타데이터 + 스토리지 구성 요소 하나를 측정한 후 추가 구성 요소를 추가하여 성능이 얼마나 향상되는지를 보여 줍니다.

하나의 **BeeGFS** 메타데이터 **+** 스토리지 구성 요소입니다

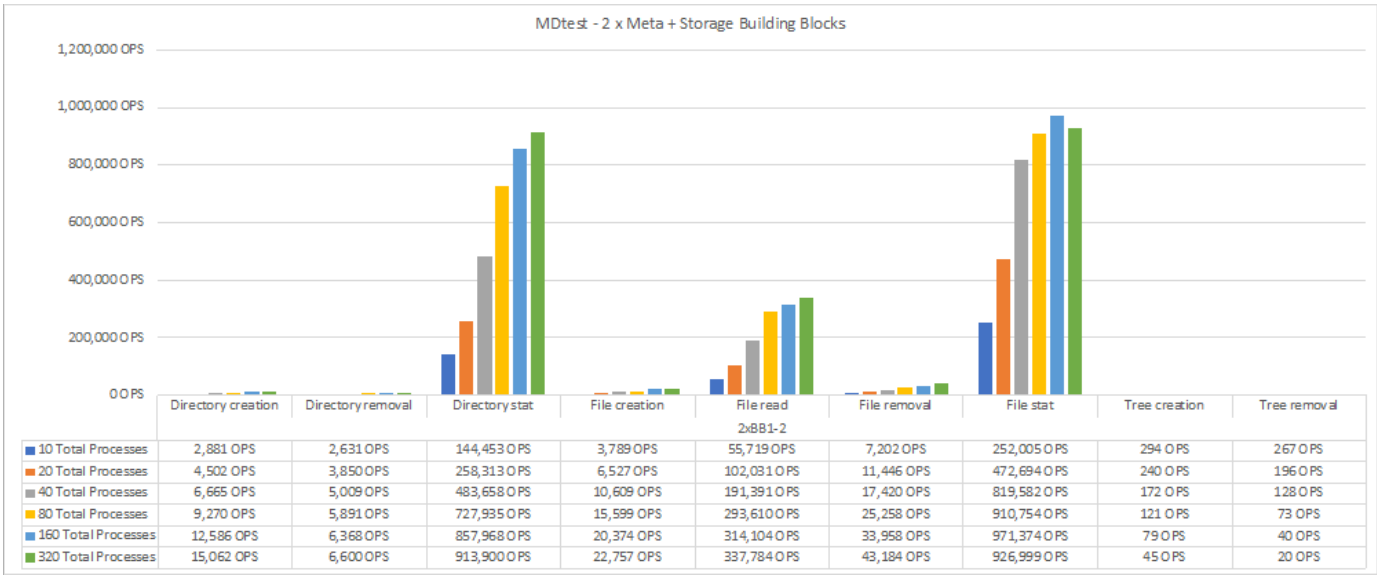
다음 다이어그램은 하나의 BeeGFS 메타데이터 + 스토리지 구성 요소가 포함된 MDTest 결과를 보여 줍니다.



	Directory creation	Directory removal	Directory stat	File creation	File read	File removal	File stat	Tree creation	Tree removal
					1xBB1				
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

BeeGFS 메타데이터 **+** 스토리지 구성 요소 **2**개

다음 다이어그램은 BeeGFS 메타데이터 + 스토리지 구성 요소 두 개가 포함된 MDTest 결과를 보여 줍니다.



	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

기능 검증

이 아키텍처의 검증 과정에서 NetApp은 다음을 비롯한 여러 기능 테스트를 수행했습니다.

- 스위치 포트를 비활성화하여 단일 클라이언트 InfiniBand 포트에 장애 발생
- 스위치 포트를 비활성화하여 단일 서버 InfiniBand 포트에 장애 발생
- BMC를 사용하여 즉시 서버 전원을 끕니다.
- 노드를 대기 노드에 배치하고 다른 노드에 대한 서비스 장애 조치를 원활히 합니다.
- 노드를 다시 온라인 상태로 전환하고 원래 노드에 서비스를 페일백합니다.
- PDU를 사용하여 InfiniBand 스위치 중 하나의 전원을 끕니다. BeeGFS 클라이언트에 설정된 'sysSessionChecksEnabled:false' 매개 변수를 사용하여 스트레스 테스트가 진행되는 동안 모든 테스트가 수행되었습니다. I/O에 대한 오류나 운영 중단이 관찰되지 않았습니다.

알려진 문제가 있습니다(참조) "변경 로그") 기본 인터페이스('connInterfacesFile'에 정의된 대로) 손실 또는 BeeGFS 서버 장애로 인해 BeeGFS 클라이언트/서버 RDMA 연결이 예기치 않게 중단되거나 활성 클라이언트 I/O가 최대 10분 동안 중단되어 다시 시작할 수 있습니다. 이 문제는 계획된 유지 관리를 위해 BeeGFS 노드가 정상적으로 대기 상태가 되거나 TCP가 사용 중인 경우 발생하지 않습니다.

NVIDIA DGX SuperPOD 및 **BasePOD** 검증

NetApp은 메타데이터와 스토리지 구성 프로필이 적용된 3개의 구성 블록으로 구성된 유사한 BeeGFS 파일 시스템을 사용하여 NVIDIAs DGX A100 SuperPOD에 대한 스토리지 솔루션을 검증했습니다. 검증 노력에는 다양한 스토리지, 머신 러닝 및 딥 러닝 벤치마크를 실행하는 20개의 DGX A100 GPU 서버를 통해 이 NVA에 의해 설명된 솔루션을 테스트하는 작업이 포함되었습니다. NVIDIA의 DGX A100 SuperPOD로 검증된 솔루션을 기반으로 구축된 NetApp 기반 BeeGFS 솔루션은 DGX SuperPOD H100, H200 및 B200 시스템에서 승인되었습니다. 이 확장은 NVIDIA DGX A100에서 검증된 이전 벤치마크 및 시스템 요구사항을 충족하는 것을 기반으로 합니다.

자세한 내용은 을 참조하십시오 "NetApp을 포함한 NVIDIA DGX SuperPOD" 및 "NVIDIA DGX 베이스POD".

사이징 지침

BeeGFS 솔루션에는 검증 테스트를 기반으로 한 성능 및 용량 사이징에 대한 권장 사항이 포함되어 있습니다.

빌딩 블록 아키텍처의 목표는 특정 BeeGFS 시스템의 요구 사항을 충족하기 위해 여러 빌딩 블록을 추가하여 간편하게 사이징할 수 있는 솔루션을 구축하는 것입니다. 아래 지침에 따라 환경 요구 사항을 충족하는 데 필요한 BeeGFS 빌딩 블록의 양과 유형을 예측할 수 있습니다.

이러한 추정치는 최상의 성능을 제공하는 것으로, 가상 벤치마킹 애플리케이션은 실제 애플리케이션이 사용할 수 없는 방식으로 기본 파일 시스템의 사용을 최적화하기 위해 작성 및 사용됩니다.

성능 사이징

다음 표에는 권장되는 성능 사이징이 나와 있습니다.

구성 프로파일	**1MiB** 읽기	**1MiB**의 쓰기입니다
메타데이터 + 스토리지	62GiBps	21GiBps
스토리지만 해당	64GiBps	21GiBps

메타데이터 용량 사이징 예상치는 "경험 규칙"을 기반으로 하며, 이 경우 500GB의 용량으로 BeeGFS에서 약 1억 5천만 개의 파일을 저장할 수 있습니다. (자세한 내용은 BeeGFS 설명서를 참조하십시오 "시스템 요구 사항"참조)

액세스 제어 목록, 디렉터리별 디렉토리 및 파일 수와 같은 기능을 사용하면 메타데이터 공간이 얼마나 빨리 소비되는지를 알 수 있습니다. 스토리지 용량 추정치는 RAID 6 및 XFS 오버헤드와 함께 사용 가능한 드라이브 용량을 고려합니다.

메타데이터 **+** 스토리지 구성 요소에 대한 용량 사이징

다음 표에는 메타데이터와 스토리지 구성 요소에 권장되는 용량 사이징이 나와 있습니다.

드라이브 크기**(2+2 RAID 1)** 메타데이터 볼륨 그룹	메타데이터 용량(파일 수)	드라이브 크기**(8 + 2 RAID 6)** 스토리지 볼륨 그룹	스토리지 용량(파일 콘텐츠)
1.92TB	1,938,577,200	1.92TB	51.77TB
3.84TB	3,880,388,400	3.84TB	103.55TB
7.68TB	8,125,278,000입니다	7.68TB	216.74TB
15.3TB	17,269,854,000	15.3TB	460.60TB

메타데이터 및 스토리지 구성 요소의 크기를 조정할 때 더 작은 드라이브를 메타데이터 볼륨 그룹에 사용하는 것과 스토리지 볼륨 그룹을 사용하는 것을 통해 비용을 절감할 수 있습니다.

스토리지 전용 구성 요소에 대한 용량 사이징

다음 표에는 스토리지 전용 구성 요소에 대한 경험 많은 용량 사이징이 나와 있습니다.

드라이브 크기(**10 + 2 RAID 6**) 스토리지 볼륨 그룹	스토리지 용량(파일 콘텐츠)
1.92TB	59.89TB
3.84TB	119.80TB
7.68TB	251.89TB
15.3TB	538.55TB

글로벌 파일 잠금이 활성화되지 않은 경우 기본(첫 번째) 구성 요소에서 관리 서비스를 포함할 때 발생하는 성능 및 용량 오버헤드가 최소화됩니다.

성능 튜닝

BeeGFS 솔루션에는 검증 테스트를 기반으로 한 성능 조정을 위한 권장 사항이 포함되어 있습니다.

BeeGFS가 즉시 사용 가능한 우수한 성능을 제공하기는 하지만, NetApp은 성능을 극대화하기 위해 일련의 권장 튜닝 매개 변수를 개발했습니다. 이 매개 변수는 기본 E-Series 블록 노드의 기능과 공유 디스크 HA 아키텍처에서 BeeGFS를 실행하는 데 필요한 특수 요구사항을 모두 고려합니다.

파일 노드의 성능 튜닝

구성할 수 있는 조정 매개변수는 다음과 같습니다.

1. * 파일 노드의 UEFI/BIOS에서 시스템 설정. * 성능을 극대화하려면 파일 노드로 사용하는 서버 모델에서 시스템 설정을 구성하는 것이 좋습니다. 시스템 설정(UEFI/BIOS) 또는 베이스보드 관리 컨트롤러(BMC)에서 제공하는 Redfish API를 사용하여 파일 노드를 설정할 때 시스템 설정을 구성합니다.

   시스템 설정은 파일 노드로 사용하는 서버 모델에 따라 달라집니다. 사용 중인 서버 모델에 따라 설정을 수동으로 구성해야 합니다. 검증된 Lenovo SR665 V3 파일 노드에 대한 시스템 설정을 구성하는 방법을 알아보려면 다음을 참조하세요. "성능을 위해 파일 노드 시스템 설정을 조정합니다" .

2. * 필수 구성 매개 변수에 대한 기본 설정 * 필수 구성 매개 변수는 BeeGFS 서비스의 구성 방법과 E-Series 볼륨(블록 장치)의 포맷 및 마운트에 영향을 미칩니다. 이러한 필수 구성 매개 변수는 다음과 같습니다.
   - BeeGFS 서비스 구성 매개 변수입니다

     필요에 따라 구성 매개 변수의 기본 설정을 재정의할 수 있습니다. 특정 워크로드 또는 사용 사례에 맞게 조정할 수 있는 매개 변수는 에서 확인하십시오. "BeeGFS 서비스 구성 매개 변수입니다"

   - 볼륨 포맷팅 및 마운팅 매개 변수는 권장 기본값으로 설정되며 고급 사용 사례에만 조정해야 합니다. 기본값은 다음을 수행합니다.
     - 기본 볼륨의 RAID 구성 및 세그먼트 크기와 함께 타겟 유형(예: 관리, 메타데이터 또는 스토리지)을 기준으로 초기 볼륨 포맷을 최적화합니다.
     - 심박동조율기가 각 볼륨을 마운트하는 방법을 조정하여 변경 사항이 즉시 E-시리즈 블록 노드로 플러시되도록 합니다. 이렇게 하면 활성 쓰기가 진행 중일 때 파일 노드가 실패할 때 데이터 손실을 방지할 수 있습니다.

       특정 워크로드 또는 사용 사례에 맞게 조정할 수 있는 매개 변수는 에서 확인하십시오. "볼륨 포맷 및 마운팅 구성 매개 변수"

3. * 파일 노드에 설치된 Linux OS의 시스템 설정. * 의 4단계에서 Ansible 인벤토리를 생성할 때 기본 Linux OS 시스템 설정을 재정의할 수 "Ansible 인벤토리를 작성합니다"있습니다.

   기본 설정을 사용하여 NetApp 기반 BeeGFS 솔루션을 검증했지만 특정 워크로드 또는 사용 사례에 맞게 수정할 수 있습니다. 변경할 수 있는 Linux OS 시스템 설정의 예는 다음과 같습니다.

   - E-Series 블록 장치의 I/O 큐

     BeeGFS 타겟으로 사용되는 E-Series 블록 디바이스에서 I/O 큐를 구성하여 다음을 수행할 수 있습니다.

     - 장치 유형(NVMe, HDD 등)에 따라 스케줄링 알고리즘을 조정합니다.
     - 미결 요청 수를 늘립니다.
     - 요청 크기를 조정합니다.
     - 미리 읽기 동작 최적화

   - 가상 메모리 설정.

     최적의 스트리밍 성능을 위해 가상 메모리 설정을 조정할 수 있습니다.

   - CPU 설정

     최대 성능을 위해 CPU 주파수 조절기 및 기타 CPU 구성을 조정할 수 있습니다.

   - 읽기 요청 크기입니다.

     NVIDIA HCA에 대한 최대 읽기 요청 크기를 늘릴 수 있습니다.

블록 노드의 성능 튜닝

특정 BeeGFS 빌딩 블록에 적용되는 구성 프로필을 기준으로 블록 노드에 구성된 볼륨 그룹이 약간 변경됩니다. 예를 들어, 24-드라이브 EF600 블록 노드는 다음과 같습니다.

- BeeGFS 관리, 메타데이터 및 스토리지 서비스를 비롯한 단일 기본 구성 요소:
  - BeeGFS 관리 및 메타데이터 서비스를 위한 1 x 2 + 2 RAID 10 볼륨 그룹
  - BeeGFS 스토리지 서비스용 2x 8+2 RAID 6 볼륨 그룹
- BeeGFS 메타데이터 + 스토리지 구성 요소:
  - BeeGFS 메타데이터 서비스용 2 + 2 RAID 10 볼륨 그룹 1개
  - BeeGFS 스토리지 서비스용 2x 8+2 RAID 6 볼륨 그룹
- BeeGFS 스토리지 전용 구성 요소:
  - BeeGFS 스토리지 서비스용 10+2 RAID 6 볼륨 그룹 2개

> BeeGFS는 스토리지 대비 관리 및 메타데이터를 위해 스토리지 공간이 훨씬 적게 요구되므로 RAID 10 볼륨 그룹에 더 작은 드라이브를 사용하는 옵션이 있습니다. 작은 드라이브는 가장 바깥쪽 드라이브 슬롯에 장착해야 합니다. 자세한 내용은 를 참조하십시오 "배포 지침".

이러한 모든 설정은 Ansible 기반 배포를 통해 구성되며, 다음은 성능/동작을 최적화하기 위해 일반적으로 권장되는 몇 가지 다른 설정과 함께 제공됩니다.

- 글로벌 캐시 블록 크기를 32KiB로 조정하고 요구 기반 캐시 플러시를 80%로 조정합니다.
- 자동 로드 밸런싱 비활성화(컨트롤러 볼륨 할당이 의도한 대로 유지되는지 확인)
- 읽기 캐싱 설정 및 미리 읽기 캐싱 해제
- 미러링으로 쓰기 캐시를 설정하고 배터리 백업이 필요하므로 블록 노드 컨트롤러의 장애가 발생해도 캐시가 유지됩니다.
- 드라이브가 볼륨 그룹에 할당되는 순서를 지정하여 사용 가능한 드라이브 채널 간에 I/O의 균형을 조정합니다.

고용량 구성 요소입니다

표준 BeeGFS 솔루션 설계는 고성능 워크로드를 염두에 두고 설계되었습니다. 고용량 사용 사례를 찾는 고객은 여기에 설명된 설계 및 성능 특성의 변화를 준수해야 합니다.

하드웨어 및 소프트웨어 구성

고용량 구성 요소에 대한 하드웨어 및 소프트웨어 구성은 EF300 컨트롤러를 각 스토리지 어레이당 60개의 드라이브로 1~7개의 IOM 확장 트레이를 연결하는 옵션과 함께 EF300 컨트롤러로 교체해야 한다는 점을 제외하고 표준입니다. 빌딩 블록당 총 2-14개의 확장 트레이.

대용량 구성 요소 설계를 구축하는 고객은 각 노드에 대해 BeeGFS 관리, 메타데이터 및 스토리지 서비스로 구성된 기본 구성 요소 스타일 구성만 사용할 수 있습니다. 비용 효율성을 위해 대용량 스토리지 노드는 EF300 컨트롤러 엔클로저의 NVMe 드라이브에 메타데이터 볼륨을 프로비저닝하고 확장 트레이의 NL-SAS 드라이브에 스토리지 볼륨을 프로비저닝해야 합니다.

[]

사이징 지침

이 사이징 지침은 대용량 구성 요소가 기본 EF300 엔클로저의 메타데이터용 2+2 NVMe SSD 볼륨 그룹 1개와 스토리지용 IOM 확장 트레이당 8개+2 NL-SAS 볼륨 그룹 6개로 구성되어 있다고 가정합니다.

드라이브 크기(용량 **HDD)**	**BB**당 용량(**1**트레이)	**BB**당 용량(**2**개의 트레이)	**BB**당 용량(**3**개의 트레이)	**BB**당 용량(**4**개의 트레이)
4TB	439TB	878TB	1,317TB	1756TB
8TB	878TB	1756TB	2,634TB	3512TB
10TB	1097TB	2195TB	3292TB	4390TB
12TB	1,317TB	2,634TB	3,951TB	5268TB
16TB	1756TB	3512TB	5268TB	7024TB
18TB	1975TB	3,951TB	5,927TB	7902TB

솔루션 구축

구축 개요

NetApp 기반 BeeGFS는 NetApp의 BeeGFS 구성 요소 설계와 함께 Ansible을 사용하여

검증된 파일 및 블록 노드에 배포할 수 있습니다.

Ansible 컬렉션 및 역할

NetApp 기반 BeeGFS 솔루션은 애플리케이션 배포를 자동화하는 주요 IT 자동화 엔진인 Ansible을 사용하여 배포됩니다. Ansible에서는 배포하려는 BeeGFS 파일 시스템을 모델링하는 인벤토리라고 하는 일련의 파일을 사용합니다.

Ansible을 사용하면 NetApp과 같은 회사에서 Ansible Galaxy에서 제공되는 컬렉션을 사용하여 기본 제공 기능을 확장할 수 있습니다(참조 "NetApp E-Series BeeGFS 컬렉션"). 컬렉션에는 특정 기능 또는 작업(예: E-Series 볼륨 생성)을 수행하는 모듈과 여러 모듈 및 기타 역할을 호출할 수 있는 역할이 포함됩니다. 이 자동화된 방식을 통해 BeeGFS 파일 시스템 및 기본 HA 클러스터를 구축하는 데 필요한 시간을 줄일 수 있습니다. 또한 클러스터와 BeeGFS 파일 시스템의 유지보수 및 확장을 간소화합니다.

자세한 내용은 을 참조하십시오 "Ansible 인벤토리에 대해 알아보십시오".

NetApp 솔루션에 BeeGFS를 구축하는 데 다양한 단계가 포함되므로 NetApp에서는 수동으로 솔루션 구축을 지원하지 않습니다.

BeeGFS 구성 요소에 대한 구성 프로필입니다

배포 절차는 다음과 같은 구성 프로파일을 다룹니다.

- 관리, 메타데이터 및 스토리지 서비스를 포함하는 하나의 기본 구성 요소입니다.
- 메타데이터와 스토리지 서비스가 포함된 두 번째 구성 요소입니다.
- 스토리지 서비스만 포함하는 세 번째 구성 요소입니다.

이러한 프로필을 통해 NetApp BeeGFS 구성 요소에 대한 권장 구성 프로필 전체 범위를 볼 수 있습니다. 각 구현 시 메타데이터 및 스토리지 구성 요소 또는 스토리지 서비스 전용 구성 요소의 수는 용량 및 성능 요구사항에 따라 달라질 수 있습니다.

배포 단계 개요

배포에는 다음과 같은 고급 작업이 포함됩니다.

하드웨어 구축

1. 각 구성 요소를 물리적으로 조립합니다.
2. 랙 및 케이블 하드웨어. 자세한 절차는 를 참조하십시오 "하드웨어 구축".

소프트웨어 구축

1. "파일 및 블록 노드 설정".
    - 파일 노드에서 BMC IP를 구성합니다
    - 지원되는 운영 체제를 설치하고 파일 노드에서 관리 네트워킹을 구성합니다
    - 블록 노드에서 관리 IP를 구성합니다
2. "Ansible 제어 노드를 설정합니다".
3. "성능을 위해 시스템 설정을 조정합니다".

4. "Ansible 인벤토리를 작성합니다".
5. "BeeGFS 구성 요소에 대한 Ansible 인벤토리를 정의합니다".
6. "Ansible을 사용하여 BeeGFS 구축".
7. "BeeGFS 클라이언트를 구성합니다".

> 배포 절차에는 텍스트를 파일로 복사해야 하는 몇 가지 예제가 포함되어 있습니다. 특정 배포에 맞게 수정해야 하거나 수정할 수 있는 모든 내용에 대해 "#" 또는 "//" 문자로 표시된 인라인 주석에 주의 깊게 접근하세요. 예를 들면 다음과 같습니다.
>
> ```
> `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"`
> ```

배포 권장 사항의 차이가 있는 파생 아키텍처:

- "고용량 빌딩 블록"

Ansible 인벤토리에 대해 알아보십시오

배포를 시작하기 전에 NetApp 기반 BeeGFS 솔루션을 배포하기 위해 Ansible을 구성 및 사용하는 방법에 대해 자세히 알아보십시오.

Ansible 인벤토리는 BeeGFS 시스템에 배포될 수 있는 파일 및 블록 노드를 나열하는 디렉토리 구조입니다. 여기에는 원하는 BeeGFS 파일 시스템을 설명하는 호스트, 그룹 및 변수가 포함됩니다. Ansible 인벤토리를 Ansible 제어 노드에 저장해야 합니다. Ansible 플레이북을 실행하는 데 사용되는 파일 및 블록 노드에 액세스할 수 있는 머신입니다. 샘플 재고는 에서 다운로드할 "NetApp E-Series BeeGFS GitHub를 참조하십시오" 수 있습니다.

Ansible 모듈 및 역할

Ansible 인벤토리에 설명된 구성을 적용하려면 엔드 투 엔드 솔루션을 구축하는 NetApp E-Series Ansible 컬렉션(에서 사용 가능)에 제공되는 다양한 Ansible 모듈 및 역할을 "NetApp E-Series BeeGFS GitHub를 참조하십시오" 사용하십시오.

NetApp E-Series Ansible 컬렉션에서 각 역할은 NetApp 솔루션 기반의 BeeGFS를 완벽하게 구축하는 데 있습니다. 이 역할은 NetApp E-Series SANtricity, 호스트 및 BeeGFS 컬렉션을 사용하여 HA(High Availability)를 통해 BeeGFS 파일 시스템을 구성할 수 있습니다. 그런 다음 스토리지를 프로비저닝하고 매핑하고 클러스터 스토리지를 사용할 준비가 되었는지 확인할 수 있습니다.

역할에 맞는 심층적인 문서가 제공되지만, 구축 절차에서는 제2세대 BeeGFS 구성 요소 설계를 사용하여 NetApp 검증 아키텍처를 구축하는 데 역할을 사용하는 방법에 대해 설명합니다.

> Ansible에 대한 사전 경험이 사전 필수 요소가 될 수 있도록 구축 단계에서 자세한 정보를 제공하려고 하지만, Ansible 및 관련 용어에 친숙해야 합니다.

BeeGFS HA 클러스터의 인벤토리 레이아웃

Ansible 인벤토리 구조를 사용하여 BeeGFS HA 클러스터를 정의합니다.

이전의 Ansible 경험을 보유한 사용자는 BeeGFS HA 역할이 각 호스트에 적용되는 변수(또는 사실)를 파악하기 위한 사용자 지정 방법을 구현한다는 점을 알아야 합니다. 이 설계는 Ansible 인벤토리를 구성하여 여러 서버에서 실행할 수 있는 리소스를 설명하는 과정을 간소화합니다.

Ansible 인벤토리는 일반적으로 및 `group_vars` 의 파일과 함께 `inventory.yml` 호스트를 특정 그룹(그리고 잠재적으로 다른 그룹에 할당하는 파일)으로 `host_vars` 구성됩니다.

본 하위 섹션의 내용을 포함하는 파일을 만들지 마십시오. 이 내용은 예제로만 제공됩니다.

이 구성은 구성 프로필을 기반으로 사전 결정됩니다. 하지만 다음과 같이 Ansible 인벤토리로 모든 내용을 레이아웃하는 방법을 전반적으로 이해해야 합니다.

```
# 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.
```

각 서비스에 대해 해당 구성을 설명하는 group_vars 아래에 추가 파일이 생성됩니다.

```
# 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
```

이 레이아웃을 통해 각 리소스에 대한 BeeGFS 서비스, 네트워크 및 스토리지 구성을 단일 위치에서 정의할 수 있습니다. BeeGFS 역할은 이러한 인벤토리 구조를 기반으로 각 파일 및 블록 노드에 필요한 구성을 집계합니다.

> 각 서비스의 BeeGFS 숫자 및 문자열 노드 ID는 그룹 이름을 기준으로 자동으로 구성됩니다. 따라서 그룹 이름이 고유해야 하는 일반적인 Ansible 요구 사항 외에도 BeeGFS 서비스를 나타내는 그룹은 해당 그룹이 나타내는 BeeGFS 서비스 유형에 고유한 번호로 끝나야 합니다. 예를 들어, meta_01 및 stor_01은 허용되지만 metadata_01 및 meta_01은 허용되지 않습니다.

모범 사례를 검토합니다

NetApp 솔루션에 BeeGFS를 구축할 때는 모범 사례 지침을 따르십시오.

표준 규약

Ansible 인벤토리 파일을 물리적으로 조립하고 생성할 때는 다음 표준 규칙을 따르십시오(자세한 내용은 을 참조하십시오 "Ansible 인벤토리를 작성합니다")를 클릭합니다.

- 파일 노드 호스트 이름은 랙 상단에 더 적은 숫자가 있고 하단에 더 높은 숫자가 있는 순서대로 번호가 지정됩니다(H01-HN).

  예를 들어 명명 규칙은 `[location][row][rack]hN` 다음과 같습니다 `beegfs_01`.

- 각 블록 노드는 각각 고유한 호스트 이름을 가진 두 개의 스토리지 컨트롤러로 구성됩니다.

스토리지 어레이 이름은 Ansible 인벤토리의 일부로 전체 블록 스토리지 시스템을 나타내는 데 사용됩니다. 스토리지 배열 이름은 순서대로 번호(A01-AN)로 지정되어야 하며, 개별 컨트롤러의 호스트 이름은 해당 명명 규칙에서 파생됩니다.

예를 들어, 이라는 이름의 블록 노드는 `ictad22a01` 일반적으로 및 와 같은 각 컨트롤러에 대해 구성된 호스트 이름을 가질 수 `ictad22a01-a ictad22a01-b`있지만, `Ansible` 인벤토리에서 로 지칭됩니다 `netapp_01`.

- 동일한 빌딩 블록 내의 파일 및 블록 노드는 동일한 번호 지정 체계를 공유하며, 랙의 서로 인접해 있으며 두 파일 노드 모두 위에 있고 두 블록 노드 바로 아래에 있습니다.

  예를 들어 첫 번째 빌딩 블록에서 파일 노드 H01 및 H02는 모두 블록 노드 A01 및 A02에 직접 연결됩니다. 위에서 아래로 호스트 이름은 H01, H02, A01 및 A02입니다.

- 빌딩 블록은 호스트 이름을 기준으로 순차적으로 설치되므로 번호가 낮은 호스트 이름은 랙 상단에, 번호가 높은 호스트 이름은 하단에 표시됩니다.

  이는 랙 스위치 상단으로 연결되는 케이블의 길이를 최소화하고 문제 해결을 단순화하기 위한 표준 배포 방법을 정의하는 것입니다. 랙 안정성 문제로 인해 이것이 허용되지 않는 데이터 센터의 경우, 맨 아래부터 랙을 채우는 역작업이 허용됩니다.

InfiniBand 스토리지 네트워크 구성

각 파일 노드의 InfiniBand 포트 중 절반은 블록 노드에 직접 연결하는 데 사용됩니다. 나머지 절반은 InfiniBand 스위치에 연결되며 BeeGFS 클라이언트-서버 연결에 사용됩니다. BeeGFS 클라이언트 및 서버에 사용되는 IPoIB 서브넷의 크기를 결정할 때 예상되는 컴퓨팅/GPU 클러스터 및 BeeGFS 파일 시스템 확장을 고려해야 합니다. 권장 IP 범위를 벗어나야 하는 경우, 단일 빌딩 블록의 각 직접 접속은 고유한 서브넷을 가지며 클라이언트-서버 접속에 사용되는 서브넷과 중복되지 않는다는 점에 유의하십시오.

직접 연결

각 빌딩 블록 내의 파일 및 블록 노드는 항상 직접 연결에 다음 표의 IP를 사용합니다.

> ⓘ 이 주소 지정 체계는 다음 규칙을 따릅니다. 세 번째 옥텟은 항상 홀수이거나 짝수이며, 이는 파일 노드가 홀수인지 아니면 짝수인지에 따라 다릅니다.

파일 노드	**IB** 포트	**IP** 주소입니다	블록 노드	**IB** 포트	물리적 **IP**	가상 **IP**
홀수(h1)	i1a	192.168.1.10	홀수(C1)	2A	192.168.1.100	192.168.1.101
홀수(h1)	i2a	192.168.3.10	홀수(C1)	2A	192.168.3.100	192.168.3.101
홀수(h1)	i3a	192.168.5.10	짝수(C2)	2A	192.168.5.100	192.168.5.101
홀수(h1)	i4a	192.168.7.10	짝수(C2)	2A	192.168.7.100	192.168.7.101
짝수(H2)	i1a	192.168.2.10	홀수(C1)	2B	192.168.2.100	192.168.2.101
짝수(H2)	i2a	192.168.4.10	홀수(C1)	2B	192.168.4.100	192.168.4.101
짝수(H2)	i3a	192.168.6.10	짝수(C2)	2B	192.168.6.100	192.168.6.101
짝수(H2)	i4a	192.168.8.10	짝수(C2)	2B	192.168.8.100	192.168.8.101

BeeGFS 클라이언트-서버 IPoIB 주소 지정 체계

각 파일 노드에서 여러 BeeGFS 서버 서비스(관리, 메타데이터 또는 스토리지)를 실행합니다. 각 서비스가 다른 파일 노드로 독립적으로 페일오버할 수 있도록 각 서비스마다 고유한 IP 주소가 구성되며 이 주소는 두 노드 간에 자유롭게 움직일 수 있습니다(LIF라고도 함).

필수 사항은 아니지만 이 구축 환경에서 이러한 연결에 다음 IPoIB 서브넷 범위가 사용 중인 것으로 가정하며 다음 규칙을 적용하는 표준 주소 지정 체계를 정의합니다.

- 두 번째 옥텟은 파일 노드 InfiniBand 포트가 홀수인지 또는 짝수인지에 따라 항상 홀수이거나 짝수 입니다.
- BeeGFS 클러스터 IP는 항상 xxx입니다. 127.100.yyy 또는 xxx.128.100.yyy.

> 대역 내 OS 관리에 사용되는 인터페이스 외에도 클러스터 심장 박동 및 동기화를 위한 Corosync에서 추가 인터페이스를 사용할 수 있습니다. 따라서 단일 인터페이스가 손실되어도 전체 클러스터가 다운되지 않습니다.

- BeeGFS Management 서비스는 항상 xxx.yyy.101.0 또는 xxx.yyy.102.0 중 입니다.
- BeeGFS 메타데이터 서비스는 항상 xxx.yyy.101.zzz 또는 xxx.yyy.102.zzz입니다.
- BeeGFS 스토리지 서비스는 항상 `xxx.yyy.103.zzz` 또는 `xxx.yyy.104.zzz`입니다.
- 100.xxx.1.1 ~ 100.xxx.99.255 범위의 주소는 고객용으로 예약되어 있습니다.

IPoIB 단일 서브넷 주소 지정 체계

이 배포 가이드에서는 에 나와 있는 이점을 감안하여 단일 서브넷 스키마를 "소프트웨어 아키텍처"활용합니다.

서브넷**: 100.127.0.0/16**

다음 표에는 단일 서브넷의 범위가 나와 있습니다. 100.127.0.0/16.

목적	**InfiniBand** 포트입니다	**IP** 주소 또는 범위입니다
BeeGFS 클러스터 IP입니다	i1b 또는 i4b	100.127.100.1-100.127.100.255
BeeGFS 관리	i1b	100.127.101.0
	i2b	100.127.102.0
BeeGFS 메타데이터	i1b 또는 i3b	100.127.101.1 - 100.127.101.255
	i2b 또는 i4b	100.127.102.1 - 100.127.102.255
BeeGFS 스토리지	i1b 또는 i3b	100.127.103.1 - 100.127.103.255
	i2b 또는 i4b	100.127.104.1 - 100.127.104.255
BeeGFS 클라이언트	(클라이언트에 따라 다름)	100.127.1.1 - 100.127.99.255

IPoIB 두 개의 서브넷 주소 지정 체계

두 개의 서브넷 주소 지정 체계는 더 이상 권장되지 않지만 여전히 구현할 수 있습니다. 권장되는 두 개의 서브넷 구성표는 아래 표를 참조하십시오.

서브넷 **A: 100.127.0.0/16**

다음 표에는 서브넷 A:100.127.0.0/16의 범위가 나와 있습니다.

목적	**InfiniBand** 포트입니다	**IP** 주소 또는 범위입니다
BeeGFS 클러스터 IP입니다	i1b	100.127.100.1-100.127.100.255
BeeGFS 관리	i1b	100.127.101.0
BeeGFS 메타데이터	i1b 또는 i3b	100.127.101.1 - 100.127.101.255
BeeGFS 스토리지	i1b 또는 i3b	100.127.103.1 - 100.127.103.255
BeeGFS 클라이언트	(클라이언트에 따라 다름)	100.127.1.1 - 100.127.99.255

서브넷 **B: 100.128.0.0/16**

다음 표에는 서브넷 B:100.128.0.0/16의 범위가 나와 있습니다.

목적	**InfiniBand** 포트입니다	**IP** 주소 또는 범위입니다
BeeGFS 클러스터 IP입니다	i4b	100.128.100.1-100.128.100.255
BeeGFS 관리	i2b	100.128.102.0
BeeGFS 메타데이터	i2b 또는 i4b	100.128.102.1-100.128.102.255
BeeGFS 스토리지	i2b 또는 i4b	100.128.104.1 - 100.128.104.255
BeeGFS 클라이언트	(클라이언트에 따라 다름)	100.128.1.1-100.128.99.255

위 범위에 있는 모든 IP가 이 NetApp 검증 아키텍처에 사용되는 것은 아닙니다. 또한 IP 주소를 사전 할당하여 일관된 IP 주소 지정 체계를 사용하여 파일 시스템을 쉽게 확장할 수 있는 방법을 보여 줍니다. 이 스키마에서는 BeeGFS 파일 노드 및 서비스 ID가 잘 알려진 IP 범위의 네 번째 옥텟과 일치합니다. 필요한 경우 파일 시스템을 255개 노드 또는 서비스 이상으로 확장할 수 있습니다.

하드웨어 구축

각 구성 요소는 HDR(200GB) InfiniBand 케이블을 사용하여 두 블록 노드에 직접 연결된 검증된 x86 파일 노드 2개로 구성됩니다.

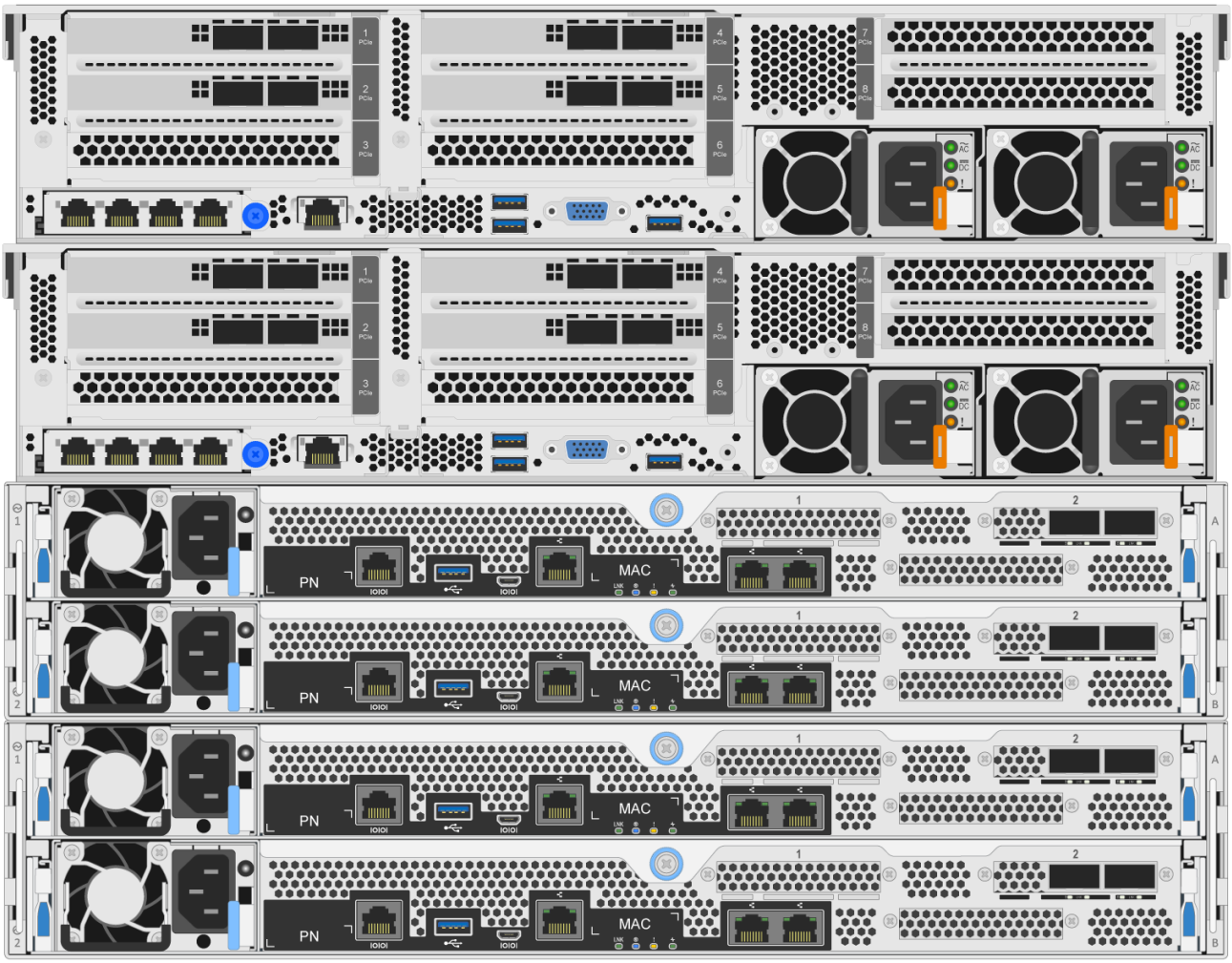
장애 조치 클러스터에서 쿼럼을 설정하려면 최소 2개의 구성 요소가 필요합니다. 2노드 클러스터에는 성공적인 페일오버를 수행할 수 없는 제한 사항이 있습니다. 세 번째 장치를 Tiebreaker로 통합하여 2노드 클러스터를 구성할 수 있지만, 이 문서에서는 그러한 설계에 대해 설명하지 않습니다.

BeeGFS 메타데이터와 스토리지 서비스를 모두 실행하는 데 사용되는지 아니면 스토리지 서비스만 실행하는 데 사용되는지에 관계없이 클러스터의 각 구성 요소에 대해 다음 단계는 동일합니다.

단계

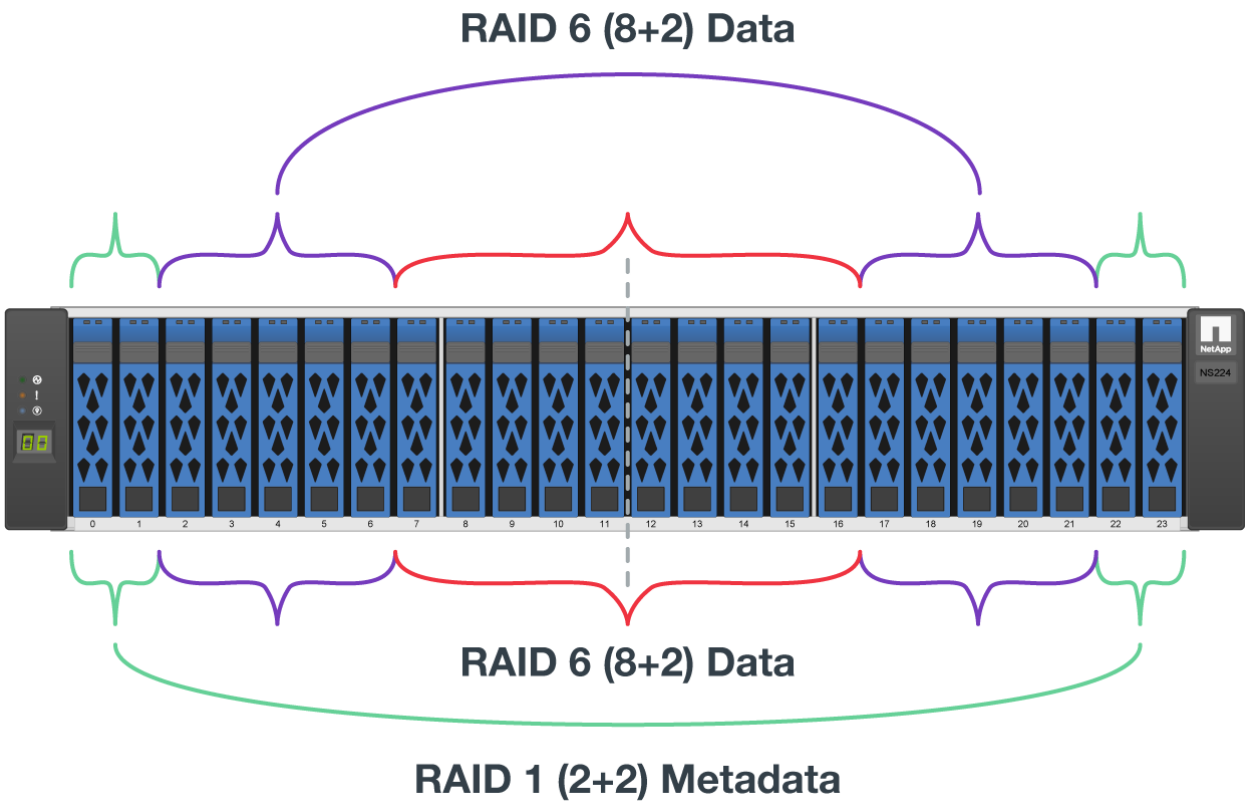
1. 에 지정된 모델을 사용하여 4개의 HCA(호스트 채널 어댑터)로 각 BeeGFS 파일 노드를 "기술 요구사항" 설정합니다. 아래 사양에 따라 파일 노드의 PCIe 슬롯에 HCA를 삽입합니다.
   - * Lenovo ThinkSystem SR665 V3 서버: * PCIe 슬롯 1, 2, 4 및 5를 사용합니다.
   - * Lenovo ThinkSystem SR665 서버: * PCIe 슬롯 2, 3, 5 및 6을 사용합니다.
2. 이중 포트 200GB 호스트 인터페이스 카드(HIC)로 각 BeeGFS 블록 노드를 구성하고 두 스토리지 컨트롤러 각각에 HIC를 설치합니다.

두 개의 BeeGFS 파일 노드가 BeeGFS 블록 노드 위에 있도록 구성 요소를 랙에 설치하십시오. 다음 그림은 Lenovo ThinkSystem SR665 V3 서버를 파일 노드로 사용하는 BeeGFS 구성 요소의 올바른 하드웨어 구성을 보여줍니다(후면).

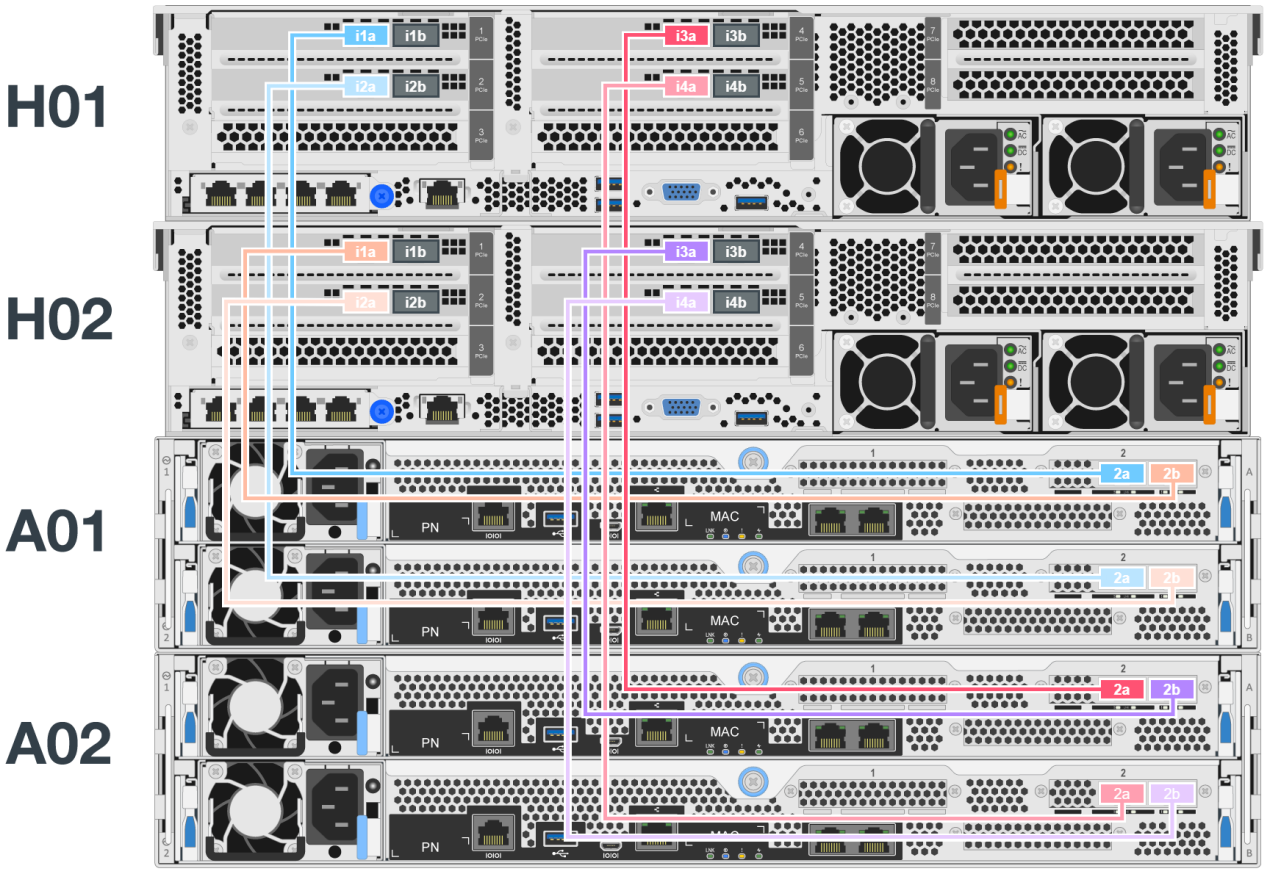


운영 활용 사례에 대한 전원 공급 장치 구성은 일반적으로 중복 PSU를 사용해야 합니다.

3. 필요한 경우 각 BeeGFS 블록 노드에 드라이브를 설치합니다.
   a. 빌딩 블록을 사용하여 BeeGFS 메타데이터 및 스토리지 서비스를 실행하고 작은 드라이브를 메타데이터 볼륨에 사용하는 경우 아래 그림과 같이 가장 바깥쪽 드라이브 슬롯에 채워졌는지 확인합니다.
   b. 모든 구성 요소 구성에서 드라이브 엔클로저가 완전히 채워지지 않은 경우 최적의 성능을 위해 슬롯 0–11 및 12–23에 동일한 수의 드라이브가 채워졌는지 확인하십시오.



4. 다음 그림에 표시된 토폴로지와 일치하도록 를 사용하여 블록 및 파일 노드를 "1m InfiniBand HDR 200GB 직접 연결 구리 케이블"연결합니다.

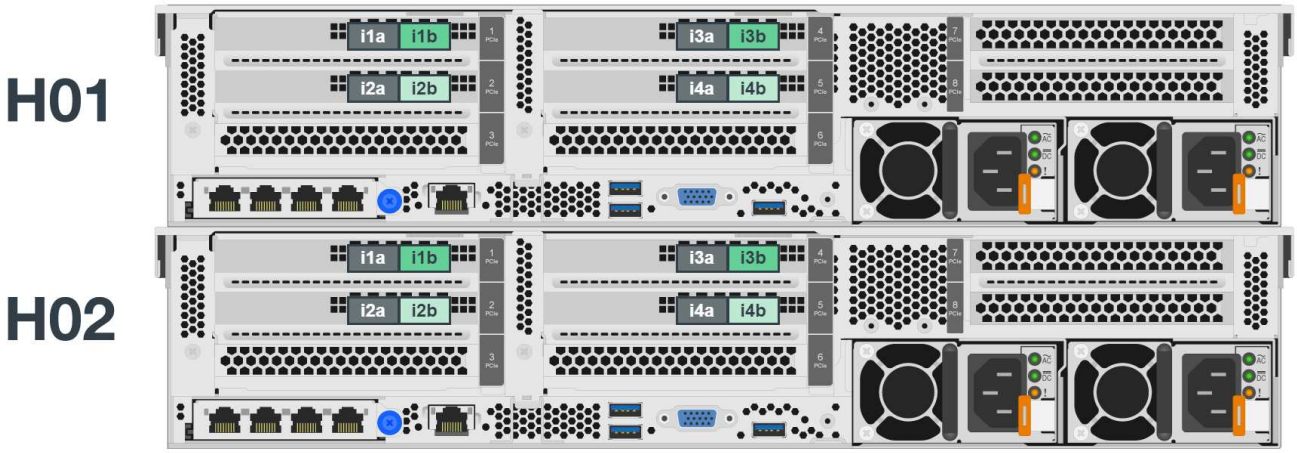


여러 빌딩 블록의 노드는 직접 연결되지 않습니다. 각 구성 요소는 독립형 장치로 취급해야 하며 구성 요소 간의 모든 통신은 네트워크 스위치를 통해 이루어집니다.

5. InfiniBand 스토리지 스위치에 해당하는 를 사용하여 파일 노드의 나머지 InfiniBand 포트를 스토리지 네트워크의 InfiniBand 스위치에 "2M InfiniBand 케이블" 연결합니다.

   Splitter 케이블을 사용하여 스토리지 스위치를 파일 노드에 연결하는 경우 케이블 하나가 스위치에서 분기되어 밝은 녹색으로 표시된 포트에 연결되어야 합니다. 다른 스플리터 케이블이 스위치에서 나와서 어두운 녹색으로 표시된 포트에 연결해야 합니다.

   또한 중복 스위치가 있는 스토리지 네트워크의 경우 연한 녹색으로 표시된 포트가 하나의 스위치에 연결되고 진한 녹색의 포트는 다른 스위치에 연결해야 합니다.



6. 필요한 경우 동일한 케이블 연결 지침에 따라 추가 구성 요소를 조립합니다.

   단일 랙에 구축할 수 있는 총 구성 요소 수는 각 사이트의 사용 가능한 전력 및 냉각에 따라 다릅니다.

소프트웨어 배포

파일 노드 및 블록 노드 설정

대부분의 소프트웨어 구성 작업은 NetApp에서 제공하는 Ansible 컬렉션을 사용하여 자동화되지만, 각 서버의 BMC(베이스보드 관리 컨트롤러)에서 네트워킹을 구성하고 각 컨트롤러의 관리 포트를 구성해야 합니다.

파일 노드 설정

1. 각 서버의 BMC(베이스보드 관리 컨트롤러)에서 네트워킹을 구성합니다.

   검증된 Lenovo SR665 V3 파일 노드에 대한 네트워킹을 구성하는 방법은 을 참조하십시오 "Lenovo ThinkSystem 설명서".

서비스 프로세서라고도 하는 베이스보드 관리 컨트롤러(BMC)는 다양한 서버 플랫폼에 내장되어 운영 체제가 설치되어 있지 않거나 액세스할 수 없는 경우에도 원격 액세스를 제공할 수 있는 대역외 관리 기능의 일반 이름입니다. 공급업체는 일반적으로 고유한 브랜딩으로 이 기능을 마케팅합니다. 예를 들어, Lenovo SR665에서 BMC는 _Lenovo XClarity Controller(XCC)_라고 합니다.

2. 최대 성능을 위해 시스템 설정을 구성합니다.

   UEFI 설정(이전의 BIOS)을 사용하거나 많은 BMC에서 제공하는 Redfish API를 사용하여 시스템 설정을 구성합니다. 시스템 설정은 파일 노드로 사용되는 서버 모델에 따라 달라집니다.

   검증된 Lenovo SR665 V3 파일 노드에 대한 시스템 설정을 구성하는 방법을 알아보려면 다음을 참조하세요. "성능을 위해 시스템 설정을 조정합니다" .

3. Red Hat Enterprise Linux(RHEL) 9.4를 설치하고 Ansible 제어 노드에서 SSH 연결을 포함하여 운영 체제를 관리하는 데 사용되는 호스트 이름과 네트워크 포트를 구성합니다.

   지금은 InfiniBand 포트에 IP를 구성하지 마십시오.

   엄밀히 요구되지는 않지만, 이후의 섹션에서는 호스트 이름이 순차적으로 번호가 매겨진 것으로 간주하고(예: h1-hn) 홀수 호스트와 짝수 번호의 호스트에서 완료해야 하는 작업을 참조합니다.

4. Red Hat Subscription Manager를 사용하여 시스템을 등록하고 구독하면 공식 Red Hat 저장소에서 필요한 패키지를 설치할 수 있고 지원되는 Red Hat 버전으로 업데이트를 제한할 수 있습니다. `subscription-manager release --set=9.4` . 자세한 내용은 및 을 "RHEL 시스템을 등록하고 가입하는 방법" "업데이트 제한 방법"참조하십시오.

5. 고가용성을 위해 필요한 패키지가 포함된 Red Hat 리포지토리를 활성화합니다.

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

6. 모든 HCA 펌웨어를 "파일 노드 어댑터 펌웨어를 업데이트합니다"가이드 사용에 권장되는 버전으로 "기술 요구 사항"업데이트합니다.

블록 노드 설정

각 컨트롤러의 관리 포트를 구성하여 EF600 블록 노드를 설정합니다.

1. 각 EF600 컨트롤러의 관리 포트를 구성합니다.

   포트 구성에 대한 자세한 내용은 를 "E-Series 문서 센터 를 참조하십시오" 참조하십시오.

2. 필요에 따라 각 시스템의 스토리지 어레이 이름을 설정합니다.

   이름을 설정하면 이후 섹션에서 각 시스템을 쉽게 참조할 수 있습니다. 배열 이름 설정에 대한 자세한 내용은 을 "E-Series 문서 센터 를 참조하십시오" 참조하십시오.

엄밀히 요구되지는 않지만, 후속 주제는 스토리지 배열 이름이 순차적으로 번호가 매겨진 것으로 간주하고(예: C1-CN) 홀수 대 짝수 번호의 시스템에서 완료해야 하는 단계를 참조합니다.

성능을 최대화하려면 파일 노드로 사용하는 서버 모델에서 시스템 설정을 구성하는 것이 좋습니다.

시스템 설정은 파일 노드로 사용하는 서버 모델에 따라 달라집니다. 이 항목에서는 검증된 Lenovo ThinkSystem SR665 서버 파일 노드에 대한 시스템 설정을 구성하는 방법에 대해 설명합니다.

UEFI 인터페이스를 사용하여 시스템 설정을 조정합니다

Lenovo SR665 V3 서버의 시스템 펌웨어에는 UEFI 인터페이스를 통해 설정할 수 있는 다양한 튜닝 매개변수가 포함되어 있습니다. 이러한 튜닝 매개 변수는 서버의 작동 방식 및 서버 성능에 미치는 모든 측면에 영향을 줄 수 있습니다.

UEFI 설정 > 시스템 설정 * 에서 다음 시스템 설정을 조정합니다.

작동 모드 메뉴

* 시스템 설정 *	* 로 변경합니다
작동 모드	맞춤형
cTDP	수동
cTDP 설명서	350
패키지 전력 제한	수동
효율성 모드	사용 안 함
Global-Cstate-Control	사용 안 함
SOC P 상태	P0
DF C 상태	사용 안 함
P - 상태	사용 안 함
메모리 전원 끄기 활성화	사용 안 함
소켓당 NUMA 노드	NPS1

Device and I/O ports(장치 및 **I/O** 포트**)** 메뉴

* 시스템 설정 *	* 로 변경합니다
IOMMU	사용 안 함

전원 메뉴

* 시스템 설정 *	* 로 변경합니다
PCIe 전원 브레이크	사용 안 함

프로세서 메뉴

* 시스템 설정 *	* 로 변경합니다
글로벌 C 상태 제어	사용 안 함
DF C 상태	사용 안 함
SMT 모드	사용 안 함
CPPC	사용 안 함

Redfish API를 사용하여 시스템 설정을 조정합니다

UEFI 설정 외에도 Redfish API를 사용하여 시스템 설정을 변경할 수 있습니다.

```
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"
}
}
'
```

Redfish 스키마에 대한 자세한 내용은 를 참조하십시오 "DMTF 웹 사이트".

Ansible 제어 노드를 설정합니다

Ansible 제어 노드를 설정하려면 NetApp 솔루션 기반 BeeGFS 솔루션용으로 구축된 모든 파일 및 블록 노드에 대한 네트워크 액세스가 가능한 가상 머신 또는 물리적 머신을 지정해야 합니다.

권장되는 패키지 버전 목록은 를 "기술 요구사항"참조하십시오. 다음 단계는 Ubuntu 22.04에서 테스트되었습니다. 선호하는 Linux 배포와 관련된 단계는 를 "Ansible 설명서"참조하십시오.

1. Ansible 제어 노드에서 다음 Python 및 Python 가상 환경 패키지를 설치합니다.

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

2. Python 가상 환경을 만듭니다.

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

3. 가상 환경을 활성화합니다.

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

4. 활성화된 가상 환경 내에 필요한 Python 패키지를 설치합니다.

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

5. Ansible Galaxy를 사용하여 BeeGFS 컬렉션을 설치합니다.

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

6. 설치된 Ansible, Python 및 BeeGFS 컬렉션 버전이 과 일치하는지 확인합니다"기술 요구사항".

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

7. Ansible이 Ansible 제어 노드에서 원격 BeeGFS 파일 노드에 액세스할 수 있도록 암호 없는 SSH를 설정합니다.
   a. 필요한 경우 Ansible 제어 노드에서 공용 키 쌍을 생성합니다.

```
ssh-keygen
```

   b. 각 파일 노드에 암호 없는 SSH를 설정합니다.

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

블록 노드에 암호 없는 SSH를 * 설정하지 마십시오. 이 작업은 지원되거나 필요하지 않습니다.

Ansible 인벤토리를 작성합니다

파일 및 블록 노드의 구성을 정의하려면 구축할 BeeGFS 파일 시스템을 나타내는 Ansible 인벤토리를 생성합니다. 인벤토리는 원하는 BeeGFS 파일 시스템을 설명하는 호스트, 그룹 및 변수를 포함합니다.

1단계**:** 모든 빌딩 블록에 대한 설정을 정의합니다

개별적으로 적용할 수 있는 구성 프로파일에 관계없이 모든 구성 요소에 적용되는 구성을 정의합니다.

시작하기 전에

- 배포에 사용할 서브넷 주소 지정 체계를 선택합니다. 에 나와 있는 이점 때문에 "소프트웨어 아키텍처"단일 서브넷 주소 지정 체계를 사용하는 것이 좋습니다.

단계

1. Ansible 제어 노드에서 Ansible 인벤토리 및 플레이북 파일을 저장하는 데 사용할 디렉토리를 식별하십시오.

   별도로 언급하지 않는 한, 이 단계에서 만든 모든 파일과 디렉터리와 다음 단계는 이 디렉터리를 기준으로 생성됩니다.

2. 다음 하위 디렉터리를 만듭니다.

   HOST_VAR'입니다

   group_vars입니다

   '패키지'

3. 클러스터 암호에 대한 하위 디렉토리를 생성하고 Ansible Vault로 암호화하여 파일을 보호합니다(참조 "Ansible Vault로 콘텐츠 암호화").

   a. 하위 디렉터리를 `group_vars/all`만듭니다.

   b. `group_vars/all`디렉토리에서 레이블이 지정된 암호 파일을 `passwords.yml` 만듭니다.

   c. 구성에 따라 모든 사용자 이름 및 암호 매개 변수를 대체하여 를 다음과 같이 입력합니다 `passwords.yml file`.

```
# 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. 실행 `ansible-vault encrypt passwords.yml` 후 메시지가 표시되면 볼트 암호를 설정합니다.

2단계**:** 개별 파일 및 블록 노드에 대한 설정을 정의합니다

개별 파일 노드 및 개별 구성 요소 노드에 적용되는 구성을 정의합니다.

1. 'host_vars/'에서 다음 내용으로 이름이 '<HOSTNAME>.yml'인 각 BeeGFS 파일 노드에 대한 파일을 만듭니다. BeeGFS 클러스터 IP 및 호스트 이름에 대해 채울 콘텐츠에 대한 메모는 홀수와 짝수로 끝나는 것이 좋습니다.

   처음에는 파일 노드 인터페이스 이름이 여기에 나열된 것과 일치합니다(예: ib0 또는 ibs1f0). 이러한 사용자 정의 이름은 에 구성되어 있습니다 4단계: 모든 파일 노드에 적용할 구성을 정의합니다.

```
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
```

BeeGFS 클러스터를 이미 구축한 경우, NVMe/IB에 사용되는 클러스터 IP 및 IP를 포함하여 정적으로 구성된 IP 주소를 추가하거나 변경하기 전에 클러스터를 중지해야 합니다. 이러한 변경 사항이 적절히 적용되고 클러스터 작업을 방해하지 않도록 이 작업이 필요합니다.

2. 'host_vars/'에서 '<HOSTNAME>.yml'이라는 이름의 각 BeeGFS 블록 노드에 대한 파일을 생성하고 다음 내용으로 채웁니다.

   홀수와 짝수로 끝나는 스토리지 배열 이름에 대한 내용을 입력할 때 특히 주의해야 합니다.

   각 블록 노드에 대해 하나의 파일을 생성하고 두 컨트롤러 중 하나의 "<management_ip>"를 지정합니다 (일반적으로 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
```

3단계**:** 모든 파일 및 블록 노드에 적용되어야 하는 설정을 정의합니다

그룹에 해당하는 파일 이름으로 group_vars 아래에 있는 호스트 그룹에 공통된 구성을 정의할 수 있습니다. 이렇게 하면 여러 위치에서 공유 구성이 반복되지 않습니다.

이 작업에 대해

호스트는 둘 이상의 그룹에 있을 수 있으며 런타임 시 Ansible은 변수 우선 순위 규칙에 따라 특정 호스트에 적용되는 변수를 선택합니다. (이 규칙에 대한 자세한 내용은 용 Ansible 설명서를 참조하십시오 "변수 사용"참조)

호스트 대 그룹 지정은 이 절차의 마지막을 위해 생성되는 실제 Ansible 인벤토리 파일에 정의됩니다.

단계

Ansible에서는 모든 호스트에 적용할 구성을 '모두'라는 그룹으로 정의할 수 있습니다. 다음 내용으로 group_vars/all.yml 파일을 만듭니다.

```
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"
```

4단계**:** 모든 파일 노드에 적용할 구성을 정의합니다

파일 노드의 공유 구성은 ha_cluster라는 그룹에 정의됩니다. 이 섹션의 단계에서는 group_vars/ha_cluster.yml 파일에 포함되어야 하는 구성을 작성합니다.

단계

1. 파일 맨 위에서 파일 노드의 'SUDO' 사용자로 사용할 암호를 포함하여 기본값을 정의합니다.

```
### 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"
```

> 가 이미 있는 `root` 경우 `ansible_ssh_user` 필요에 따라 를 생략하고 플레이북을 실행할 때 옵션을 지정할 `--ask-become-pass` 수 있습니다 `ansible_become_password`.

2. 필요에 따라 고가용성(HA) 클러스터의 이름을 구성하고 클러스터 내 통신을 위한 사용자를 지정합니다.

   전용 IP 주소 지정 체계를 수정하는 경우 기본 "begfs_ha_mgmtd_floating_ip"도 업데이트해야 합니다. 나중에 BeeGFS 관리 리소스 그룹에 대해 구성한 것과 일치해야 합니다.

"begfs_ha_alert_email_list"를 사용하여 클러스터 이벤트에 대한 경고를 수신할 e-메일을 하나 이상 지정합니다.

```
### 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
```

중복된 것처럼 보이지만 BeeGFS 파일 시스템을 단일 HA 클러스터 이상으로 확장하는 경우 "begfs_ha_mgmtd_floating_ip"가 중요합니다. 이후 HA 클러스터는 추가 BeeGFS 관리 서비스 없이 구축되고 첫 번째 클러스터에서 제공하는 관리 서비스를 가리키도록 구축됩니다.

3. 펜싱 에이전트를 구성합니다. (자세한 내용은 을 참조하십시오 "Red Hat High Availability 클러스터에서 펜싱을 구성합니다".) 다음 출력에서는 일반적인 펜싱 에이전트를 구성하는 예를 보여 줍니다. 다음 옵션 중 하나를 선택합니다.

   이 단계에서는 다음 사항에 유의하십시오.

- 기본적으로 펜싱은 활성화되어 있지만 fencing_agent_를 구성해야 합니다.
- pcmk_host_map 또는 pcmk_host_list에 지정된 '<HOSTNAME>'은(는) Ansible 인벤토리의 호스트 이름과 일치해야 합니다.
- 특히 운영 환경에서는 펜싱 없이 BeeGFS 클러스터를 실행할 수 없습니다. 이는 주로 블록 디바이스와 같은 리소스 종속성이 포함된 BeeGFS 서비스가 문제로 인해 페일오버될 때 파일 시스템 손상 또는 기타 바람직하지 않거나 예기치 않은 동작으로 이어질 수 있는 여러 노드에 의한 동시 액세스 위험이 발생하지 않도록 하기 위한 것입니다. 펜싱을 비활성화해야 하는 경우 BeeGFS HA 역할의 시작 가이드의 일반 참고를 참조하여 ha_cluster_crm_config_options ["STONITH -enabled"]"를 false 로 설정합니다.
- 사용 가능한 노드 레벨 펜싱 장치가 여러 개 있으며 BeeGFS HA 역할은 Red Hat HA 패키지 리포지토리에서 사용 가능한 펜싱 에이전트를 구성할 수 있습니다. 가능한 경우 무정전 전원 공급 장치(UPS) 또는 랙 배전 장치(rPDU)를 통해 작동하는 펜싱 에이전트를 사용합니다. BMC(베이스보드 관리 컨트롤러) 또는 서버에 내장된 기타 표시등 출력 장치와 같은 일부 펜싱 에이전트가 특정 장애 시나리오에서 Fence 요청에 응답하지 않을 수 있기 때문입니다.

```
### 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. Linux OS에서 권장되는 성능 조정을 활성화합니다.

   일반적으로 성능 매개 변수에 대한 기본 설정은 대부분의 사용자가 찾지만 선택적으로 특정 작업 부하에 대한 기본 설정을 변경할 수 있습니다. 따라서 이러한 권장 사항은 BeeGFS 역할에 포함되지만 기본적으로 설정되어 있지 않으므로 사용자가 파일 시스템에 적용된 튜닝에 대해 알 수 있습니다.

   성능 조정을 활성화하려면 다음을 지정하십시오.

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

5. (선택 사항) 필요에 따라 Linux OS에서 성능 조정 매개 변수를 조정할 수 있습니다.

   조정할 수 있는 사용 가능한 튜닝 매개 변수의 전체 목록은 에서 BeeGFS HA 역할의 성능 튜닝 기본값 섹션을 참조하십시오 "E-Series BeeGFS GitHub 사이트". 이 파일의 클러스터에 있는 모든 노드 또는 개별 노드의 파일에 대해 기본값을 재정의할 수 `host_vars` 있습니다.

6. 블록 노드와 파일 노드 간에 전체 200GB/HDR 연결을 허용하려면 NVIDIA Open Fabrics Enterprise Distribution(MLNX_OFED)의 OpenSM(Open Subnet Manager) 패키지를 사용하십시오. 나열된 MLNX_OFED 버전은 "파일 노드 요구 사항" 권장 OpenSM 패키지와 함께 제공됩니다. Ansible을 사용한 배포가 지원되지만, 먼저 모든 파일 노드에 MLNX_OFED 드라이버를 설치해야 합니다.

   a. group_vars/ha_cluster.yml에 다음 파라미터를 입력합니다(필요에 따라 패키지 조정).

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

7. 논리적 InfiniBand 포트 식별자를 기본 PCIe 디바이스에 일관되게 매핑하도록 'udev' 규칙을 구성합니다.

   udev 규칙은 BeeGFS 파일 노드로 사용되는 각 서버 플랫폼의 PCIe 토폴로지에 고유해야 합니다.

   검증된 파일 노드에 대해 다음 값을 사용합니다.

```
### 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. (선택 사항) 메타데이터 대상 선택 알고리즘을 업데이트합니다.

```
beegfs_ha_beegfs_meta_conf_ha_group_options:
  tuneTargetChooser: randomrobin
```

검증 테스트에서는 일반적으로 성능 벤치마킹 중에 테스트 파일이 모든 BeeGFS 스토리지 대상에 고르게 분산되도록 하기 위해 "랜덤 로빈"이 사용되었습니다(벤치마킹을 위한 자세한 내용은 BeeGFS 사이트 참조) "BeeGFS 시스템을 벤치마킹합니다")를 클릭합니다. 실제 환경에서 사용하면 낮은 번호의 대상이 높은 번호의 목표보다 빠르게 채워질 수 있습니다. 기본 '무작위 배정' 값을 사용하기만 하면 사용 가능한 모든 대상을 활용하는 동시에 우수한 성능을 제공하는 것으로 나타났습니다.

5단계**:** 공통 블록 노드에 대한 구성을 정의합니다

블록 노드의 공유 구성은 eseries_storage_systems라는 그룹에 정의되어 있습니다. 이 섹션의 단계에서는 group_vars/eseries_storage_systems.yml 파일에 포함되어야 하는 구성을 작성합니다.

단계

1. Ansible 연결을 로컬로 설정하고 시스템 암호를 제공하며 SSL 인증서를 확인해야 하는지 여부를 지정합니다. (일반적으로 Ansible은 SSH를 사용하여 관리 호스트에 연결하지만, 블록 노드로 사용되는 NetApp E-Series 스토리지 시스템의 경우 모듈은 통신에 REST API를 사용합니다.) 파일 맨 위에 다음을 추가합니다.

```
### 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. 최적의 성능을 보장하기 위해 에 블록 노드에 대해 나열된 버전을 설치합니다 "기술 요구사항".

   에서 해당 파일을 다운로드합니다 "NetApp Support 사이트". 수동으로 업그레이드하거나 Ansible 제어 노드의 'packages/' 디렉토리에 추가한 다음, Ansible을 사용하여 업그레이드하려면 "eseries_storage_systems.yml"에 다음 매개 변수를 입력합니다.

```
# 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. 에서 블록 노드에 설치된 드라이브에 사용할 수 있는 최신 드라이브 펌웨어를 "NetApp Support 사이트 "다운로드하여 설치합니다. 수동으로 업그레이드하거나 Ansible 제어 노드의 디렉토리에 포함시킨 다음 Ansible을 사용하여 업그레이드하려면 에 다음 매개 변수를 채울 수 있습니다 `packages/eseries_storage_systems.yml` .

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

> eseries_drive_firmware_upgrade_drives_online을 "false"로 설정하면 업그레이드 속도가 빨라지지만 BeeGFS가 구축되기 전에는 수행할 수 없습니다. 이 설정은 응용 프로그램 오류를 방지하기 위해 업그레이드 전에 드라이브에 대한 모든 I/O를 중지하도록 하기 때문입니다. 볼륨을 구성하기 전에 온라인 드라이브 펌웨어 업그레이드를 수행하는 것이 여전히 빠르지만 나중에 문제가 발생하지 않도록 항상 이 값을 "참"으로 설정하는 것이 좋습니다.

4. 성능을 최적화하려면 글로벌 구성을 다음과 같이 변경합니다.

```
# 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. 최적의 볼륨 프로비저닝 및 동작을 위해 다음 매개 변수를 지정합니다.

```
# 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"
```

'eseries_storage_pool_usable_drives'에 지정된 값은 NetApp EF600 블록 노드에만 해당되며 드라이브가 새 볼륨 그룹에 할당되는 순서를 제어합니다. 이 주문을 통해 각 그룹에 대한 입출력이 백엔드 드라이브 채널에 균등하게 분산됩니다.

BeeGFS 구성 요소에 대한 **Ansible** 인벤토리를 정의합니다

일반 Ansible 인벤토리 구조를 정의한 후 BeeGFS 파일 시스템의 각 구성 요소에 대한 구성을 정의합니다.

이러한 구축 지침은 관리, 메타데이터 및 스토리지 서비스를 포함한 기본 구성 요소로 구성된 파일 시스템, 메타데이터 및 스토리지 서비스를 포함하는 두 번째 구성 요소, 세 번째 스토리지 전용 구성 요소로 이루어진 파일 시스템을 구축하는 방법을 보여 줍니다.

이 단계에서는 전체 BeeGFS 파일 시스템의 요구 사항을 충족하도록 NetApp BeeGFS 구성 요소를 구성하는 데 사용할 수 있는 모든 일반 구성 프로필을 보여 주기 위한 것입니다.

이 섹션과 후속 섹션에서 필요에 따라 조정하여 구축할 BeeGFS 파일 시스템을 나타내는 인벤토리를 생성합니다. 특히, 스토리지 네트워크에 대해 각 블록 또는 파일 노드를 나타내는 Ansible 호스트 이름과 원하는 IP 주소 지정 스키마를 사용하여 BeeGFS 파일 노드 및 클라이언트의 수에 맞게 확장할 수 있도록 합니다.

1단계**: Ansible** 인벤토리 파일을 만듭니다

단계

1. 새 'inventory.yml' 파일을 만든 다음, 필요에 따라 'eseries_storage_systems' 아래에 있는 호스트를 대체하여 구축의 블록 노드를 나타냅니다. 이름은 host_VAR/<filename>.yml에 사용되는 이름과 일치해야 합니다.

```
# 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:
```

다음 섹션에서는 클러스터에서 실행할 BeeGFS 서비스를 나타내는 "ha_cluster" 아래에 Ansible 그룹을 추가로 생성합니다.

2단계**:** 관리**,** 메타데이터 및 스토리지 구성 요소에 대한 인벤토리를 구성합니다

클러스터 또는 기본 구성 요소에서 첫 번째 구성 요소는 메타데이터 및 스토리지 서비스와 함께 BeeGFS 관리 서비스를 포함해야 합니다.

단계

1. inventory.yml에서 ha_cluster:Children 아래에 다음 매개 변수를 입력합니다.

```
      # 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. 'group_vars/mgmt.yml' 파일을 생성하고 다음을 포함합니다.

```
# 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. group_vars/ 아래에서 다음 템플릿을 사용하여 META_08을 통해 자원 그룹 META_01에 대한 파일을 만든 다음 아래 표를 참조하여 각 서비스에 대한 자리 표시자 값을 입력합니다.

```
# 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>
```

볼륨 크기는 전체 스토리지 풀(볼륨 그룹이라고도 함)의 백분율로 지정됩니다. SSD 오버 프로비저닝을 위해 각 풀에 여유 용량을 남겨 두는 것이 좋습니다(자세한 내용은 참조) "NetApp EF600 어레이 소개")를 클릭합니다. 스토리지 풀 'begfs_m1_m2_m5_m6'도 관리 서비스를 위해 풀 용량의 1%를 할당합니다. 따라서 스토리지 풀의 메타데이터 볼륨에 대해 1.92TB 또는 3.84TB 드라이브를 사용할 때 Beegfs_M1_m2_M5_M6의 경우 이 값을 21.25로 설정하고, 7.65TB 드라이브의 경우 이 값을 22.25로 설정하고, 15.3TB 드라이브의 경우 이 값을 23.75로 설정합니다.

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
meta_01.yml	8015	i1b:100.127.101.1 / 16 i2b:100.127.102.1/16	0	netapp_01를 참조하십시오	Beegfs_m1_m2_m5_m6	A
meta_02.yml	8025	i2b:100.127.102.2 / 16 i1b:100.127.101.2/16	0	netapp_01를 참조하십시오	Beegfs_m1_m2_m5_m6	B
meta_03.yml	8035	i3b:100.127.101.3 / 16 i4b:100.127.102.3/16	1	netapp_02를 참조하십시오	Beegfs_m3_M4_M7_M8	A
meta_04.yml	8045	i4b:100.127.102.4 / 16 i3b:100.127.101.4/16	1	netapp_02를 참조하십시오	Beegfs_m3_M4_M7_M8	B

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
meta_05.yml	8055	i1b:100.127.101.5 / 16 i2b:100.127.102.5/16	0	netapp_01를 참조하십시오	Beegfs_m1_m2_m5_m6	A
meta_06.yml	8065	i2b:100.127.102.6 / 16 i1b:100.127.101.6/16	0	netapp_01를 참조하십시오	Beegfs_m1_m2_m5_m6	B
meta_07.yml	8075	i3b:100.127.101.7 / 16 i4b:100.127.102.7/16	1	netapp_02를 참조하십시오	Beegfs_m3_M4_M7_M8	A
META_08.월	8085	i4b:100.127.102.8 / 16 i3b:100.127.101.8/16	1	netapp_02를 참조하십시오	Beegfs_m3_M4_M7_M8	B

4. group_vars/ 아래에서 다음 템플릿을 사용하여 'tor_08'을 통해 리소스 그룹 tor_01에 대한 파일을 만든 다음 예제를 참조하여 각 서비스의 자리 표시자 값을 입력합니다.

```
# 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>
```

올바른 크기는 을 참조하십시오 "권장되는 스토리지 풀 오버 프로비저닝 비율".

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
STOR_01.대칭	8013	i1b:100.127.103.1 / 16 i2b:100.127.104.1/16	0	netapp_01를 참조하십시오	Beegfs_s1_s2	A
STOR_02.월	8023	i2b:100.127.104.2 / 16 i1b:100.127.103.2/16	0	netapp_01를 참조하십시오	Beegfs_s1_s2	B
STOR_03.월	8033	i3b:100.127.103.3 / 16 i4b:100.127.104.3/16	1	netapp_02를 참조하십시오	Beegfs_S3_S4	A
STOR_04.yml	8043	i4b:100.127.104.4 / 16 i3b:100.127.103.4/16	1	netapp_02를 참조하십시오	Beegfs_S3_S4	B
STOR_05.월	8053	i1b:100.127.103.5 / 16 i2b:100.127.104.5/16	0	netapp_01를 참조하십시오	Beegfs_S5_S6	A
STOR_06.대칭	8063	i2b:100.127.104.6 / 16 i1b:100.127.103.6/16	0	netapp_01를 참조하십시오	Beegfs_S5_S6	B
STOR_07.월	8073	i3b:100.127.103.7 / 16 i4b:100.127.104.7/16	1	netapp_02를 참조하십시오	Beegfs_S7_s8	A
STOR_08.월	8083	i4b:100.127.104.8 / 16 i3b:100.127.103.8/16	1	netapp_02를 참조하십시오	Beegfs_S7_s8	B

3단계**:** 메타데이터 **+** 스토리지 구성 요소에 대한 인벤토리를 구성합니다

다음 단계에서는 BeeGFS 메타데이터 + 스토리지 구성 요소에 대한 Ansible 인벤토리를 설정하는 방법을 설명합니다.

단계

1. 'inventory.yml'에서 기존 설정 아래에 다음 파라미터를 입력합니다.

```
        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. group_vars/ 아래에서 다음 템플릿을 사용하여 META_16을 통해 자원 그룹 META_09 파일을 만든 다음 예제를 참조하여 각 서비스의 자리 표시자 값을 입력합니다.

```
# 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>
```

올바른 크기는 을 참조하십시오 "권장되는 스토리지 풀 오버 프로비저닝 비율".

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
META_09.대칭	8015	i1b:100.127.101.9 / 16 i2b:100.127.102.9/16	0	netapp_03를 참조하십시오	Beegfs_m9_M10_M13_M14	A
META_10.월	8025	i2b:100.127.102.10 / 16 i1b:100.127.101.10/16	0	netapp_03를 참조하십시오	Beegfs_m9_M10_M13_M14	B
meta_11.yml	8035	i3b:100.127.101.11 / 16 i4b:100.127.102.11/16	1	netapp_04를 참조하십시오	Beegfs_M11_M12_M15_M16	A
META_12.월	8045	i4b:100.127.102.12 / 16 i3b:100.127.101.12/16	1	netapp_04를 참조하십시오	Beegfs_M11_M12_M15_M16	B
META_13.월	8055	i1b:100.127.101.13 / 16 i2b:100.127.102.13/16	0	netapp_03를 참조하십시오	Beegfs_m9_M10_M13_M14	A
meta_14.yml	8065	i2b:100.127.102.14 / 16 i1b:100.127.101.14/16	0	netapp_03를 참조하십시오	Beegfs_m9_M10_M13_M14	B
META_15.월	8075	i3b:100.127.101.15 / 16 i4b:100.127.102.15/16	1	netapp_04를 참조하십시오	Beegfs_M11_M12_M15_M16	A
meta_16.yml	8085	i4b:100.127.102.16 / 16 i3b:100.127.101.16/16	1	netapp_04를 참조하십시오	Beegfs_M11_M12_M15_M16	B

3. group_vars/ 아래에서 다음 템플릿을 사용하여 'tor_16'을 통해 리소스 그룹 tor_09에 대한 파일을 만든 다음 예제를 참조하여 각 서비스의 자리 표시자 값을 입력합니다.

```
# 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>
```

사용할 올바른 크기를 보려면 다음을 참조하세요. "권장되는 스토리지 풀 오버 프로비저닝 비율" ..

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
STOR_09.대칭	8013	i1b:100.127.103.9 / 16 i2b:100.127.104.9/16	0	netapp_03를 참조하십시오	Beegfs_S9_S10	A
STOR_10.월	8023	i2b:100.127.104.10 / 16 i1b:100.127.103.10/16	0	netapp_03를 참조하십시오	Beegfs_S9_S10	B
STOR_11.월	8033	i3b:100.127.103.11 / 16 i4b:100.127.104.11/16	1	netapp_04를 참조하십시오	Beegfs_s11_s12	A
STOR_12.월	8043	i4b:100.127.104.12 / 16 i3b:100.127.103.12/16	1	netapp_04를 참조하십시오	Beegfs_s11_s12	B

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
STOR_13.월	8053	i1b:100.127.103.13 / 16 i2b:100.127.104.13/16	0	netapp_03를 참조하십시오	Beegfs_S13_s14	A
STOR_14.월	8063	i2b:100.127.104.14 / 16 i1b:100.127.103.14/16	0	netapp_03를 참조하십시오	Beegfs_S13_s14	B
STOR_15.월	8073	i3b:100.127.103.15 / 16 i4b:100.127.104.15/16	1	netapp_04를 참조하십시오	Beegfs_s15_S16	A
STOR_16.월	8083	i4b:100.127.104.16 / 16 i3b:100.127.103.16/16	1	netapp_04를 참조하십시오	Beegfs_s15_S16	B

4단계**:** 스토리지 전용 구성 요소에 대한 인벤토리를 구성합니다

다음 단계에서는 BeeGFS 스토리지 전용 구성 요소에 대한 Ansible 인벤토리를 설정하는 방법을 설명합니다. 메타데이터 + 스토리지에 대한 구성을 설정하는 것과 스토리지 전용 구성 블록에 대한 구성을 설정하는 것의 주된 차이점은 모든 메타데이터 리소스 그룹이 생략되고 각 스토리지 풀에 대해 "criteria_drive_count"를 10에서 12로 변경하는 것입니다.

단계

1. 'inventory.yml'에서 기존 설정 아래에 다음 파라미터를 입력합니다.

```
# 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. group_vars/ 아래에서 다음 템플릿을 사용하여 'tor_24'를 통해 리소스 그룹 tor_17에 대한 파일을 만든 다음 예제를 참조하여 각 서비스의 자리 표시자 값을 입력합니다.

```
# 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>
```

사용할 올바른 크기를 보려면 다음을 참조하세요. "권장되는 스토리지 풀 오버 프로비저닝 비율" .

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
STOR_17.월	8013	i1b:100.127.103.17 / 16 i2b:100.127.104.17/16	0	netapp_05를 참조하십시오	Beegfs_S17_s18	A
STOR_18.월	8023	i2b:100.127.104.18 / 16 i1b:100.127.103.18/16	0	netapp_05를 참조하십시오	Beegfs_S17_s18	B
STOR_19.대칭	8033	i3b:100.127.103.19 / 16 i4b:100.127.104.19/16	1	netapp_06를 참조하십시오	Beegfs_S19_S20	A
STOR_20.월	8043	i4b:100.127.104.20 / 16 i3b:100.127.103.20/16	1	netapp_06를 참조하십시오	Beegfs_S19_S20	B

파일 이름입니다	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
STOR_21.대칭	8053	i1b:100.127.103.21 / 16 i2b:100.127.104.21/16	0	netapp_05를 참조하십시오	Beegfs_s21_S22	A
STOR_22.월	8063	i2b:100.127.104.22 / 16 i1b:100.127.103.22/16	0	netapp_05를 참조하십시오	Beegfs_s21_S22	B
STOR_23.월	8073	i3b:100.127.103.23 / 16 i4b:100.127.104.23/16	1	netapp_06를 참조하십시오	Beegfs_S23_S24	A
STOR_24.월	8083	i4b:100.127.104.24 / 16 i3b:100.127.103.24/16	1	netapp_06를 참조하십시오	Beegfs_S23_S24	B

BeeGFS 구축

구성 배포 및 관리에는 Ansible이 실행해야 하는 작업이 포함된 하나 이상의 플레이북을 실행해야 전체 시스템을 원하는 상태로 되돌릴 수 있습니다.

모든 작업이 단일 Playbook에 포함될 수 있지만 복잡한 시스템의 경우 이를 관리하기가 매우 복잡해집니다. Ansible을 사용하면 재사용 가능한 플레이북 및 관련 콘텐츠(예: 기본 변수, 작업, 처리기)를 패키지하는 방법으로 역할을 만들고 배포할 수 있습니다. 자세한 내용은 용 Ansible 설명서를 참조하십시오 "역할".

역할은 종종 관련 역할 및 모듈이 포함된 Ansible 컬렉션의 일부로 배포됩니다. 따라서, 이러한 플레이북은 다양한 NetApp E-Series Ansible 컬렉션에서 주로 다양한 역할을 하지만

> 2노드 클러스터를 사용하여 쿼럼을 설정할 때 문제를 완화하기 위해 별도의 쿼럼 장치를 Tiebreaker로 구성하지 않는 한 현재 BeeGFS를 구축하려면 최소 2개의 구성 요소(4개의 파일 노드)가 필요합니다.

단계

1. 새로운 '플레이북.yml' 파일을 생성하고 다음을 포함합니다.

```
# 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
```

ⓘ 이 플레이북에서는 파일 노드에 Python 3이 설치되어 있는지 확인하는 몇 가지 "pre_tasks"를 실행하고 제공되는 Ansible 태그가 지원되는지 확인합니다.

2. BeeGFS를 배포할 준비가 되면 재고 및 플레이북 파일과 함께 'Ansible-Playbook' 명령을 사용하십시오.

   구축 과정에서 모든 "pre_tasks"를 실행한 다음 실제 BeeGFS 구축을 진행하기 전에 사용자 확인을 묻는 메시지가 표시됩니다.

   다음 명령을 실행하여 필요에 따라 포크 수를 조정합니다(아래 참고 참조).

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

   ⓘ 대규모 배포의 경우 `forks` 매개 변수를 사용하여 기본 포크 수(5)를 재정의하면 Ansible이 병렬로 구성하는 호스트 수를 늘리는 것이 좋습니다. (자세한 내용은 을 참조하십시오 "플레이북 실행 제어".) 최대값 설정은 Ansible 제어 노드에서 사용할 수 있는 처리 능력에 따라 다릅니다. 위의 20개 예는 CPU 4개(인텔® 제온® 골드 6146 CPU @ 3.20GHz)가 장착된 가상 Ansible 컨트롤 노드에서 실행되었습니다.

   Ansible 제어 노드와 BeeGFS 파일 및 블록 노드 간의 구축 및 네트워크 성능 크기에 따라 구축 시간이 달라질 수 있습니다.

BeeGFS 클라이언트를 구성합니다

컴퓨팅 또는 GPU 노드와 같이 BeeGFS 파일 시스템에 액세스해야 하는 모든 호스트에 BeeGFS 클라이언트를 설치하고 구성해야 합니다. 이 작업에서는 Ansible 및 BeeGFS 컬렉션을 사용할 수 있습니다.

단계

1. 필요한 경우, Ansible 제어 노드에서 BeeGFS 클라이언트로 구성하려는 각 호스트에 대해 암호 없는 SSH를 설정합니다.

   'ssh-copy-id <user>@<HOSTNAME_OR_IP>'를 참조하십시오

2. 'host_vars/'에서 다음 내용으로 이름이 '<HOSTNAME>.yml'인 각 BeeGFS 클라이언트에 대한 파일을 만들어 사용자 환경에 맞는 올바른 정보로 자리 표시자 텍스트를 채웁니다.

```
# 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>
```

두 개의 서브넷 주소 지정 체계를 사용하여 구축하는 경우 각 클라이언트에서 두 개의 스토리지 IPoIB 서브넷에 각각 하나씩 두 개의 InfiniBand 인터페이스를 구성해야 합니다. 여기에 나열된 각 BeeGFS 서비스에 대해 예제 서브넷과 권장 범위를 사용하는 경우, 클라이언트에는 ~ 의 범위에서 인터페이스 하나와 ~ 의 범위로 구성된 인터페이스가 있어야 `100.127.1.0` `100.127.99.255` `100.128.1.0` `100.128.99.255`합니다.

3. 새 파일 'client_inventory.yml'를 만든 다음 맨 위에 다음 매개 변수를 입력합니다.

```
# 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.
```

암호를 일반 텍스트로 저장하지 마십시오. 대신 Ansible Vault를 사용하십시오(의 Ansible 설명서 참조) "Ansible Vault로 콘텐츠 암호화") 또는 플레이북이 실행될 때 '--Ask-when-pass' 옵션을 사용합니다.

4. 'client_inventory.yml' 파일에서 Beegfs_clients' 그룹 아래에 BeeGFS 클라이언트로 구성해야 하는 모든 호스트를 나열한 다음 BeeGFS 클라이언트 커널 모듈을 구축하는 데 필요한 추가 구성을 지정합니다.

```
  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.
```

NVIDIA OFED 드라이버를 사용하는 경우 가 `beegfs_client_ofed_include_path` Linux 설치에 대한 올바른 "헤더 포함 경로"를 가리키는지 확인합니다. 자세한 내용은 의 BeeGFS 설명서를 "RDMA 지원"참조하십시오.

5. client_inventory.yml 파일에 미리 정의된 VAR의 하단에 마운트할 BeeGFS 파일 시스템을 나열합니다.

```
        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
```

Beegfs_client_config는 테스트된 설정을 나타냅니다. 모든 옵션에 대한 종합적인 개요는 netapp_eseries.beegfs` 컬렉션의 "begfs_client" 역할에 포함된 설명서를 참조하십시오. 여기에는 여러 개의 BeeGFS 파일 시스템을 마운트하거나 동일한 BeeGFS 파일 시스템을 여러 번 마운트하는 방법에 대한 세부 정보가 포함됩니다.

6. 새 'client_Playbook.yml' 파일을 만든 후 다음 매개 변수를 입력합니다.

```
# 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
```

필요한 IB/RDMA 드라이버와 IP를 해당 IPoIB 인터페이스에 이미 설치한 경우 'NetApp_eseries.host' 수집 및 'IPoIB' 역할을 가져오지 마십시오.

7. 클라이언트를 설치 및 구축하고 BeeGFS를 마운트하려면 다음 명령을 실행합니다.

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

8. BeeGFS 파일 시스템을 운영 환경에 배치하기 전에 모든 클라이언트에 로그인하고 "begfs-fsck—checkfs"를 실행하여 모든 노드에 연결할 수 있고 보고된 문제가 없는지 확인하는 것이 좋습니다.

5가지 구성 요소 이상으로 확장

5개의 빌딩 블록(10개의 파일 노드) 이상으로 확장하기 위해 페이스 메이커 및 Corosync를 구성할 수 있습니다. 그러나 더 큰 클러스터에는 결점이 있으며, 결국 심장박동기 및 Corosync로 인해 최대 32개의 노드가 필요합니다.

NetApp은 최대 10개 노드까지 BeeGFS HA 클러스터를 테스트했습니다. 따라서 개별 클러스터를 이 제한을 초과하여 확장하는 것은 권장되거나 지원되지 않습니다. 하지만 BeeGFS 파일 시스템은 여전히 10개 노드 이상으로 확장되어야 하며 NetApp은 BeeGFS on NetApp 솔루션에서 이 점을 고려했습니다.

각 파일 시스템에서 구성 요소의 하위 집합이 포함된 여러 HA 클러스터를 구축함으로써 기본 HA 클러스터링 메커니즘에 대한 권장 제한 또는 하드 제한과는 별개로 전체 BeeGFS 파일 시스템을 확장할 수 있습니다. 이 시나리오에서는 다음을 수행합니다.

- 추가 HA 클러스터를 나타내는 새 Ansible 인벤토리를 생성한 다음 다른 관리 서비스 구성 생략합니다. 대신 각 추가 클러스터 ha_cluster.yml의 "begfs_ha_mgmtd_floating_ip" 변수를 첫 번째 BeeGFS 관리 서비스의 IP에 지정합니다.
- 동일한 파일 시스템에 HA 클러스터를 추가할 때는 다음 사항을 확인하십시오.
  - BeeGFS 노드 ID는 고유합니다.

- group_vars의 각 서비스에 해당하는 파일 이름은 모든 클러스터에서 고유합니다.
- BeeGFS 클라이언트 및 서버 IP 주소는 모든 클러스터에서 고유합니다.
- 추가 클러스터를 구축 또는 업데이트하기 전에 BeeGFS 관리 서비스가 포함된 첫 번째 HA 클러스터가 실행되고 있습니다.

- 각 HA 클러스터에 대한 인벤토리를 각각 자체 디렉토리 트리에서 유지 관리합니다.

  하나의 디렉토리 트리에서 여러 클러스터의 인벤토리 파일을 혼합하려고 하면 BeeGFS HA 역할이 특정 클러스터에 적용된 구성을 집계하는 방식에 문제가 발생할 수 있습니다.

> 새 HA 클러스터를 생성하기 전에 각 HA 클러스터를 5개의 구성 요소로 확장할 필요가 없습니다. 대부분의 경우 클러스터당 더 적은 수의 구성 요소를 사용하는 것이 더 쉽게 관리할 수 있습니다. 한 가지 접근 방식은 각 단일 랙의 구성 요소를 HA 클러스터로 구성하는 것입니다.

권장되는 스토리지 풀 오버 프로비저닝 비율

2세대 구성 요소에 대해 스토리지 풀당 표준 볼륨 4개를 따르는 경우 다음 표를 참조하십시오.

이 표에는 각 BeeGFS 메타데이터 또는 스토리지 타겟에 대한 "eseries_storage_pool_configuration"의 볼륨 크기로 사용할 권장 비율이 나와 있습니다.

드라이브 크기	크기
1.92TB	18
3.84TB	21.5
7.68TB	22.5
15.3TB	24

> 위의 지침은 관리 서비스가 포함된 스토리지 풀에 적용되지 않으며, 관리 데이터에 대해 스토리지 풀의 1%를 할당하기 위해 위의 크기를 .25%까지 줄여야 합니다.

이러한 값이 어떻게 결정되었는지 확인하려면 을 참조하십시오 "TR-4800: 부록 A: SSD 내구성 및 오버 프로비저닝 이해".

고용량 구성 요소입니다

표준 BeeGFS 솔루션 구축 가이드에서는 고성능 워크로드 요구 사항에 대한 절차 및 권장 사항을 간략하게 설명합니다. 고용량 요구 사항을 충족하려는 고객은 여기에 설명된 구축 및 권장 사항의 변화를 관찰해야 합니다.

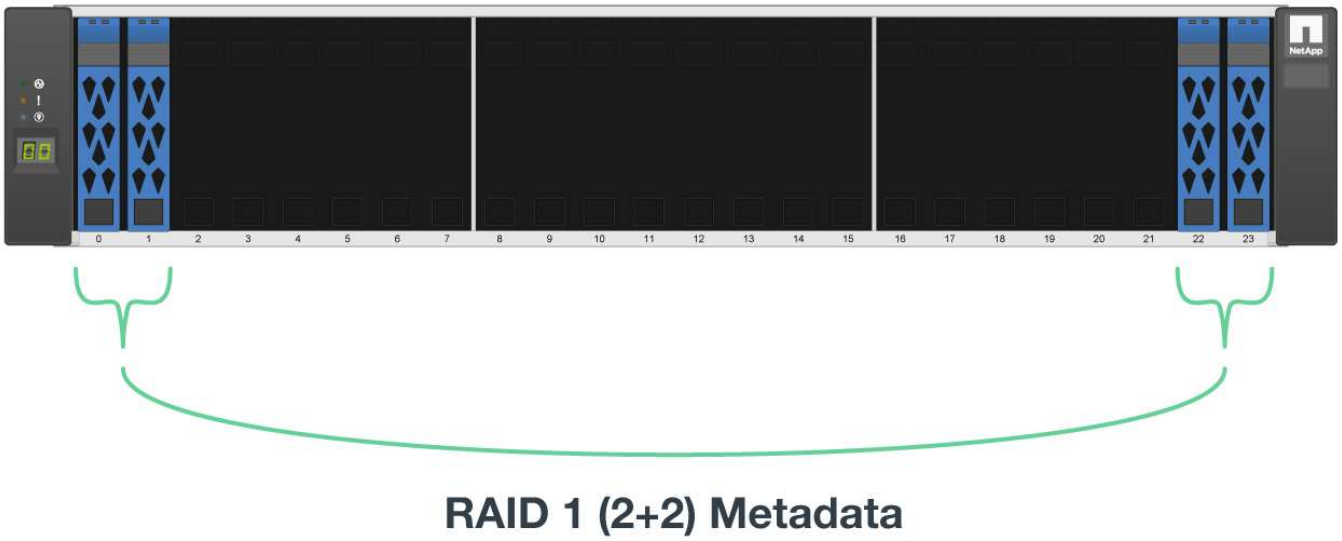
컨트롤러

고용량 구성 요소의 경우 EF600 컨트롤러를 EF300 컨트롤러로 교체해야 하며, 각 컨트롤러는 SAS 확장을 위해 Cascade HIC를 설치합니다. 각 블록 노드는 스토리지 엔클로저에 BeeGFS 메타데이터 스토리지를 위한 최소한의 NVMe SSD를 포함하고 BeeGFS 스토리지 볼륨용 NL-SAS HDD로 채워진 확장 셸프에 연결됩니다.

File Node to Block 노드 구성은 동일하게 유지됩니다.

드라이브 배치

BeeGFS 메타데이터 스토리지를 위해 각 블록 노드에 최소 4개의 NVMe SSD가 필요합니다. 이러한 드라이브는 인클로저의 가장 바깥쪽 슬롯에 위치해야 합니다.



확장 트레이

스토리지 어레이당 1-7, 60 드라이브 확장 트레이를 사용하여 대용량 구성 요소를 사이징할 수 있습니다.

각 확장 트레이 케이블 연결 지침은 "드라이브 쉘프의 EF300 케이블 연결을 참조하십시오".

맞춤형 아키텍처를 사용합니다

개요 및 요구 사항

Ansible을 사용하여 BeeGFS 고가용성 클러스터를 구축할 때 모든 NetApp E/EF-Series 스토리지 시스템을 BeeGFS 블록 노드 및 x86 서버로 BeeGFS 파일 노드로 사용할 수 있습니다.

이 섹션에서 사용되는 용어에 대한 정의는 "용어 및 개념"페이지에서 확인할 수 있습니다.

소개

"NetApp 검증 아키텍처"사전 정의된 참조 구성 및 사이징 지침을 제공하는 한편, 일부 고객 및 파트너는 특정 요구 사항이나 하드웨어 선호도에 보다 적합한 맞춤형 아키텍처를 설계하기를 원할 수 있습니다. NetApp에서 BeeGFS를 선택할 때의 주요 이점 중 하나는 Ansible을 사용하여 BeeGFS 공유 디스크 HA 클러스터를 구축하여 NetApp에서 작성한 HA 구성요소를 통해 클러스터 관리를 단순화하고 안정성을 높일 수 있다는 것입니다. NetApp에 맞춤형 BeeGFS 아키텍처를 구축할 때는 Ansible을 사용하여 유연한 하드웨어 제품군에서 어플라이언스 같은 접근 방식을 유지 관리합니다.

이 섹션에서는 NetApp 하드웨어에서 BeeGFS 파일 시스템을 구축하고 Ansible을 사용하여 BeeGFS 파일 시스템을 구성하는 데 필요한 일반적인 단계를 간략하게 설명합니다. BeeGFS 파일 시스템 설계와 관련된 모범 사례 및 최적화된 예제에 대한 자세한 "NetApp 검증 아키텍처"내용은 섹션을 참조하십시오.

구축 개요

일반적으로 BeeGFS 파일 시스템을 구축하려면 다음 단계를 수행해야 합니다.

- 초기 설정:
  - 하드웨어 설치/케이블 연결
  - 파일 및 블록 노드 설정
  - Ansible 제어 노드를 설정합니다.
- BeeGFS 파일 시스템을 Ansible 인벤토리로 정의합니다.
- 파일 및 블록 노드 기준으로 Ansible을 실행하여 BeeGFS 구축
  - 선택적으로 클라이언트를 설정하고 BeeGFS를 마운트합니다.

다음 섹션에서는 이러한 단계에 대해 자세히 설명합니다.

Ansible은 다음을 비롯한 모든 소프트웨어 프로비저닝 및 구성 작업을 처리합니다.

- 블록 노드에서 볼륨 생성/매핑
- 파일 노드에서 볼륨 포맷/튜닝
- 파일 노드에 소프트웨어 설치/구성
- HA 클러스터 설정 및 BeeGFS 리소스 및 파일 시스템 서비스 구성

요구 사항

Ansible에서 BeeGFS에 대한 지원은 에서 제공됩니다 "Ansible 갤럭시" BeeGFS HA 클러스터의 포괄적인 구축 및 관리를 자동화하는 역할 및 모듈 모음입니다.

BeeGFS 자체는 <major>.<minor>.<patch> 버전 관리 체계에 따라 버전이 지정되며, 이 컬렉션은 지원되는 각 <major>.<minor> 버전의 BeeGFS에 대한 역할을 유지합니다. 예를 들어 BeeGFS 7.2 또는 BeeGFS 7.3과 같은 BeeGFS 버전이 있습니다. 컬렉션에 대한 업데이트가 릴리스되면 각 역할의 패치 버전이 해당 릴리즈 지점에 대해 사용 가능한 최신 BeeGFS 버전을 가리키도록 업데이트됩니다(예: 7.2.8). 각 버전의 컬렉션은 특정 Linux 배포판과 버전에서도 테스트되어 지원됩니다. 현재 파일 노드의 경우 Red Hat이고, 클라이언트의 경우 Red Hat과 Ubuntu입니다. 다른 배포판 실행이 지원되지 않으며 다른 버전(특히 다른 주요 버전)을 실행하는 것은 권장되지 않습니다.

Ansible 컨트롤 노드

이 노드에는 BeeGFS 관리에 사용되는 인벤토리 및 Playbook이 포함됩니다. 다음과 같은 요구 사항이 있습니다.

- Ansible 6.x(Ansible-코어 2.13)
- Python 3.6 (이상)
- Python(PIP) 패키지: ipaddr 및 netaddr

또한 제어 노드에서 모든 BeeGFS 파일 노드 및 클라이언트로 암호 없는 SSH를 설정하는 것이 좋습니다.

BeeGFS 파일 노드

파일 노드는 Red Hat Enterprise Linux(RHEL) 9.4를 실행해야 하며, 필수 패키지(pacemaker, corosync, fence-agents-all, resource-agents)가 포함된 HA 저장소에 액세스할 수 있어야 합니다. 예를 들어, 다음 명령을 실행하여 RHEL 9에서 해당 저장소를 활성화할 수 있습니다.

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

BeeGFS 클라이언트 노드

BeeGFS 클라이언트 Ansible 역할을 사용하여 BeeGFS 클라이언트 패키지를 설치하고 BeeGFS 마운트를 관리할 수 있습니다. 이 역할은 RHEL 9.4 및 Ubuntu 22.04에서 테스트되었습니다.

Anabilities를 사용하여 BeeGFS 클라이언트를 설정하고 BeeGFS를 마운트하지 않는 경우 "BeeGFS는 Linux 배포 및 커널을 지원했습니다" 사용할 수 있습니다.

초기 설정

하드웨어 설치 및 케이블 연결

NetApp에서 BeeGFS를 실행하는 데 사용되는 하드웨어를 설치하고 케이블을 연결하는 데 필요한 단계입니다.

설치 계획

각 BeeGFS 파일 시스템은 몇 개의 블록 노드에서 제공하는 백엔드 스토리지를 사용하여 BeeGFS 서비스를 실행하는 몇 개의 파일 노드로 구성됩니다. 파일 노드는 하나 이상의 고가용성 클러스터로 구성되어 BeeGFS 서비스에 대한 내결함성을 제공합니다. 각 블록 노드는 이미 액티브/액티브 HA 쌍입니다. 각 HA 클러스터에서 지원되는 파일 노드의 최소 수는 3개이고, 각 클러스터에서 지원되는 파일 노드의 최대 수는 10개입니다. BeeGFS 파일 시스템은 함께 작동하여 단일 파일 시스템 네임스페이스를 제공하는 여러 독립 HA 클러스터를 구축함으로써 10개 노드 이상으로 확장할 수 있습니다.

일반적으로 각 HA 클러스터는 몇 개의 파일 노드(x86 서버)가 일부 블록 노드(일반적으로 E-Series 스토리지 시스템)에 직접 연결되는 일련의 "구성 요소"로 구축됩니다. 이 구성은 비대칭형 클러스터를 생성하며, BeeGFS 서비스는 BeeGFS 타겟에 사용되는 백엔드 블록 스토리지에 대한 액세스 권한이 있는 특정 파일 노드에서만 실행할 수 있습니다. 각 구성 요소에서 파일-블록 노드와 직접 연결에 사용되는 스토리지 프로토콜의 균형은 특정 설치 요구 사항에 따라 달라집니다.

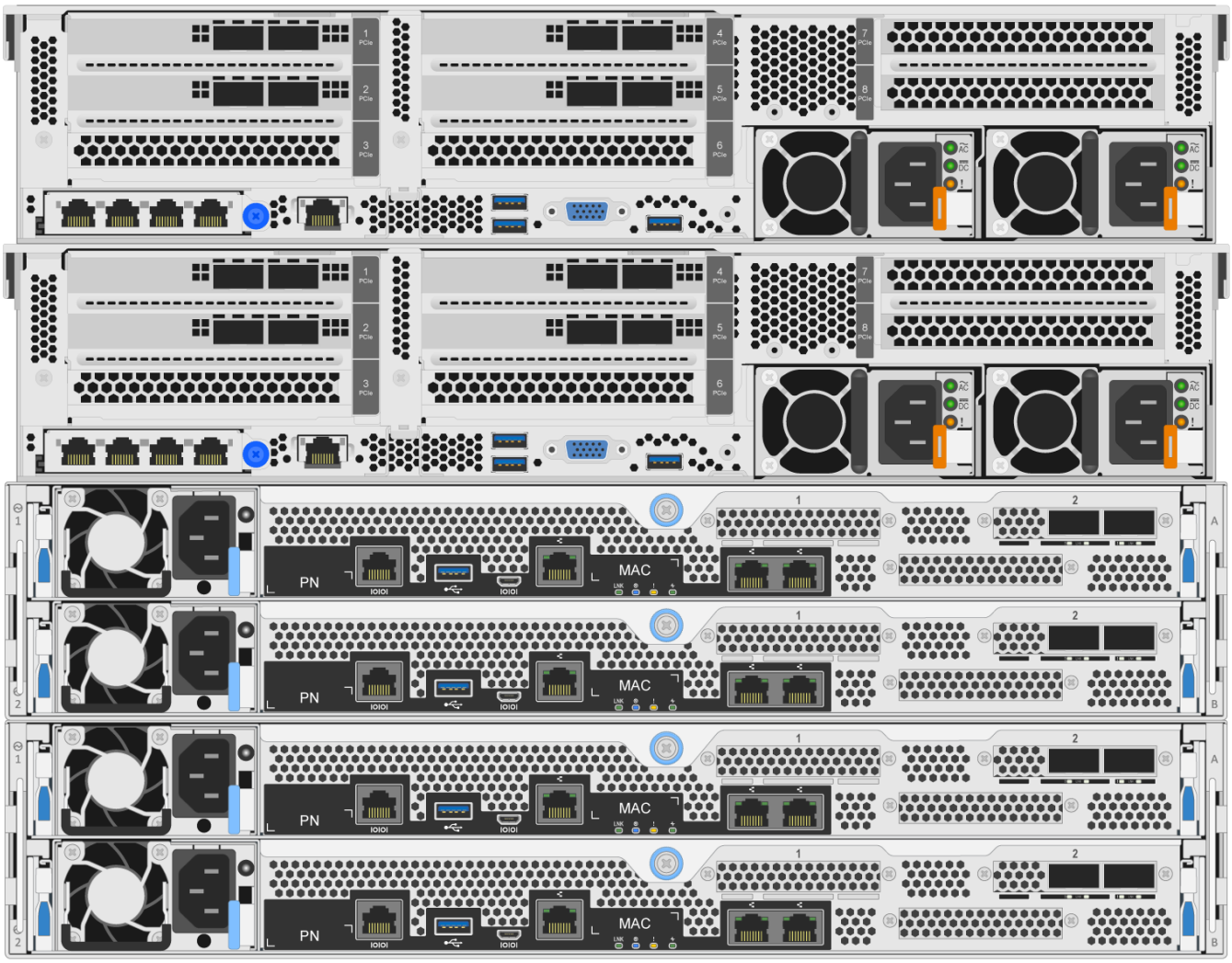
대체 HA 클러스터 아키텍처는 파일 노드와 블록 노드 간에 스토리지 패브릭(SAN이라고도 함)을 사용하여 대칭 클러스터를 설정합니다. 따라서 특정 HA 클러스터의 모든 파일 노드에서 BeeGFS 서비스를 실행할 수 있습니다. 일반적으로 대칭형 클러스터는 추가 SAN 하드웨어로 인해 비용 효율적이지 않기 때문에 이 문서에서는 하나 이상의 빌딩 블록으로 구성된 일련의 비대칭 클러스터를 사용한다고 가정합니다.

설치를 계속하기 전에 특정 BeeGFS 구축에 대해 원하는 파일 시스템 아키텍처를 충분히 이해해야 합니다.

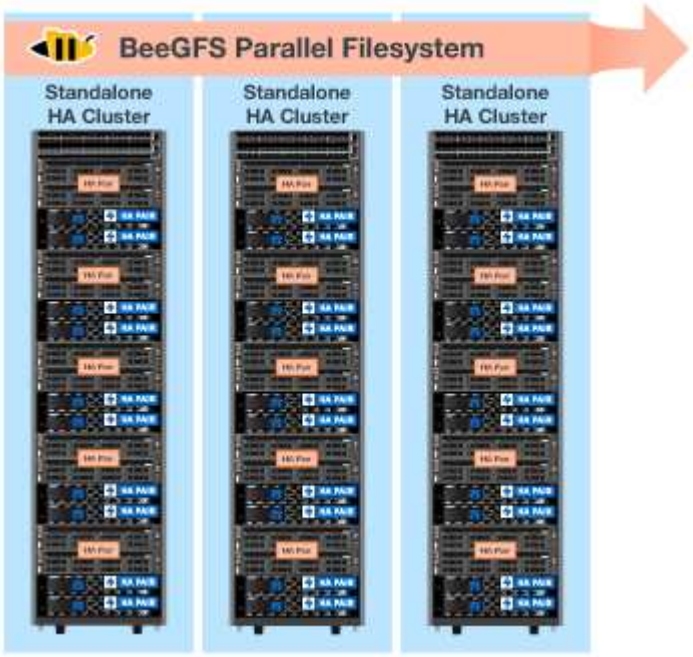
랙 하드웨어

설치를 계획할 때 각 구성 요소에 있는 모든 장비가 인접한 랙 유닛에 랙에 장착되어 있어야 합니다. 각 구성 요소에서 파일 노드를 바로 블록 노드 위에 랙에 마운트하는 것이 모범 사례입니다. 파일 및 의 모델에 대한 설명서를 따릅니다 "블록" 레일 및 하드웨어를 랙에 설치할 때 사용 중인 노드입니다.

단일 구성 요소 예:

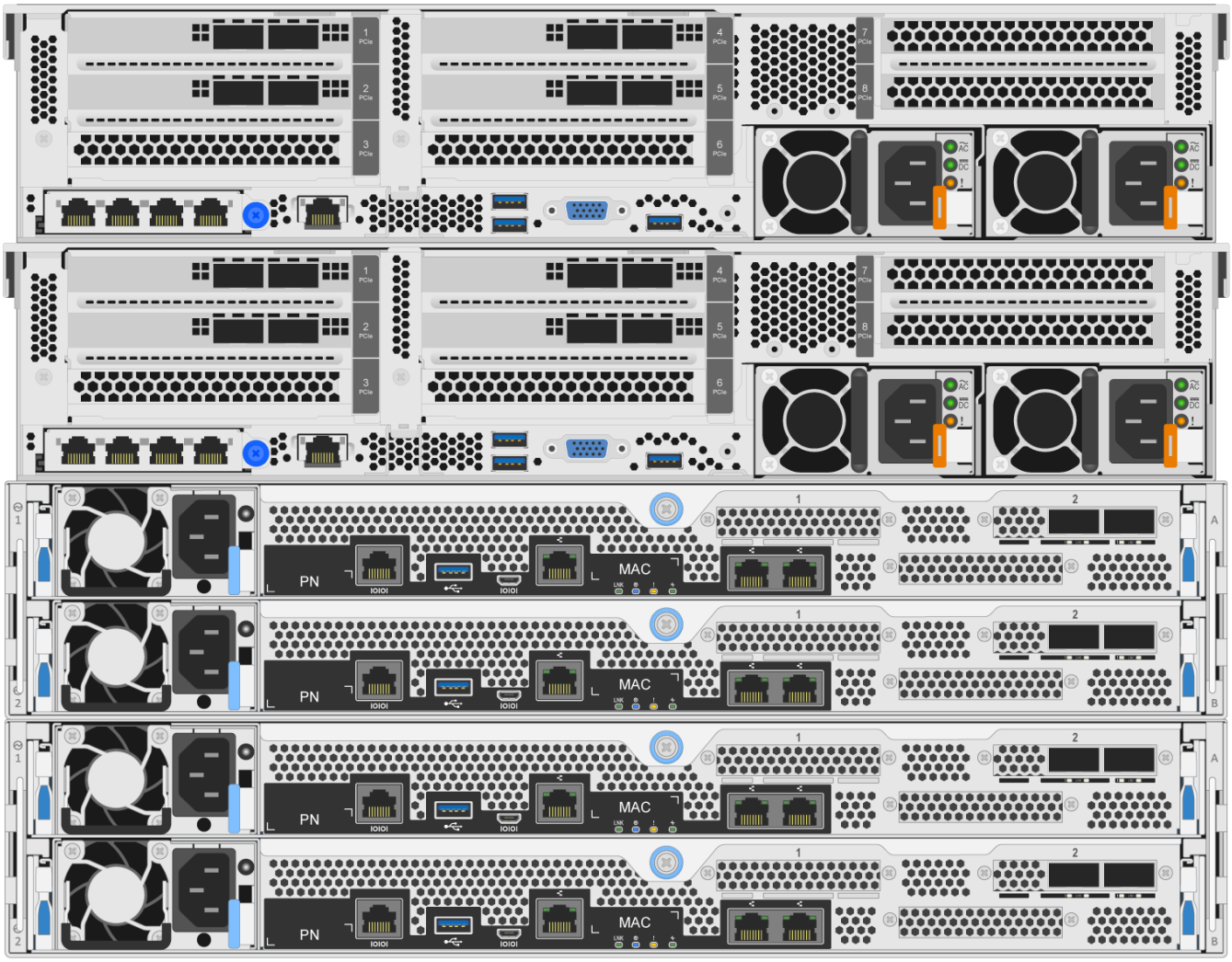


각 HA 클러스터에 여러 개의 구성 요소가 있고 파일 시스템에 여러 HA 클러스터가 있는 대규모 BeeGFS 설치의 예:



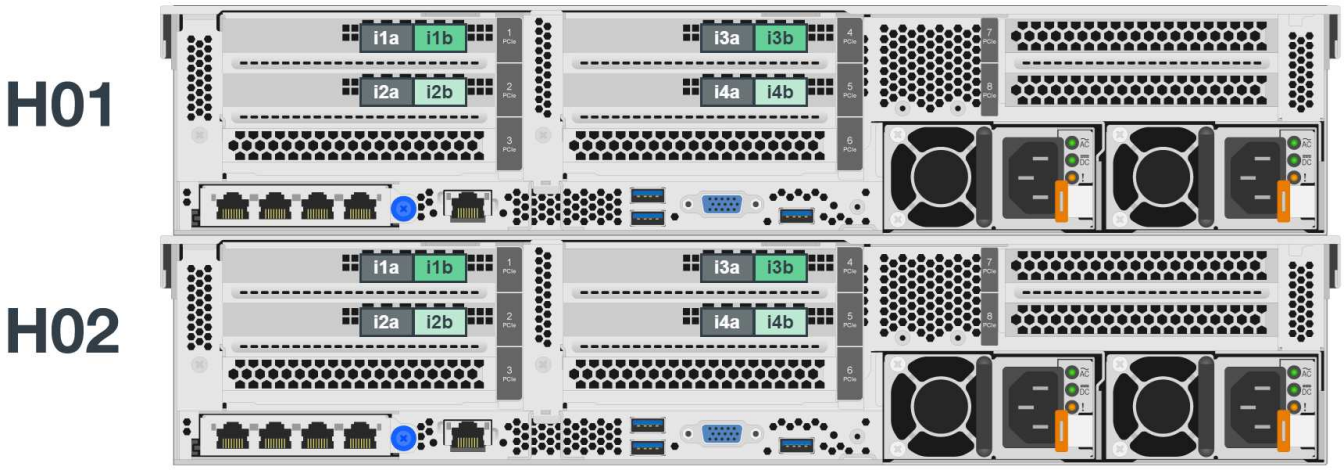
케이블 파일 및 블록 노드

일반적으로 E-Series 블록 노드의 HIC 포트를 파일 노드의 지정된 호스트 채널 어댑터(InfiniBand 프로토콜의 경우) 또는 호스트 버스 어댑터(파이버 채널 및 기타 프로토콜의 경우) 포트에 직접 연결합니다. 이러한 접속을 설정하는 정확한 방법은 원하는 파일 시스템 아키텍처에 따라 다릅니다. 예를 들면 다음과 "NetApp 검증 아키텍처에서 2세대 BeeGFS 기반"같습니다.



클라이언트 네트워크에 파일 노드를 케이블로 연결합니다

각 파일 노드에는 BeeGFS 클라이언트 트래픽용으로 지정된 몇 개의 InfiniBand 또는 이더넷 포트가 있습니다. 아키텍처에 따라 각 파일 노드는 고성능 클라이언트/스토리지 네트워크에 하나 이상의 연결을 갖게 되며, 이중화 및 대역폭 향상을 위해 여러 스위치에 연결할 수 있습니다. 다음은 이중화된 네트워크 스위치를 사용하는 클라이언트 케이블 연결의 예입니다. 이 때 어두운 녹색으로 강조 표시된 포트와 밝은 녹색으로 강조 표시된 포트는 별도의 스위치에 연결됩니다.



연결 관리 네트워킹 및 전원

대역내 및 대역외 네트워크에 필요한 네트워크 연결을 설정합니다.

각 파일 및 블록 노드가 중복성을 위해 여러 전원 분배 장치에 연결되어 있는지 확인하기 위해 모든 전원 공급 장치를 연결합니다(사용 가능한 경우).

파일 및 블록 노드 설정

Ansible을 실행하기 전에 파일 및 블록 노드를 설정하는 데 필요한 수동 단계

파일 노드

베이스보드 관리 컨트롤러(BMC) 구성

서비스 프로세서라고도 하는 베이스보드 관리 컨트롤러(BMC)는 다양한 서버 플랫폼에 내장되어 운영 체제가 설치되어 있지 않거나 액세스할 수 없는 경우에도 원격 액세스를 제공할 수 있는 대역외 관리 기능의 일반 이름입니다. 공급업체는 일반적으로 고유한 브랜딩으로 이 기능을 마케팅합니다. 예를 들어, Lenovo SR665에서 BMC는 Lenovo XClarity Controller(XCC)라고 합니다.

서버 공급업체의 설명서에 따라 이 기능에 액세스하는 데 필요한 모든 라이센스를 활성화하고 BMC가 네트워크에 연결되고 원격 액세스에 맞게 구성되었는지 확인합니다.

> Redfish를 사용하여 BMC 기반 펜싱을 사용하려면 Redfish가 활성화되어 있고 파일 노드에 설치된 OS에서 BMC 인터페이스에 액세스할 수 있어야 합니다. BMC와 운영 체제가 동일한 물리적 네트워크 인터페이스를 공유하는 경우 네트워크 스위치에 특별한 구성이 필요할 수 있습니다.

시스템 설정을 조정합니다

시스템 설정(BIOS/UEFI) 인터페이스를 사용하여 성능을 최대화하도록 설정이 설정되어 있는지 확인합니다. 정확한 설정과 최적의 값은 사용 중인 서버 모델에 따라 달라집니다. 에 대한 지침이 "파일 노드 모델을 확인했습니다"제공되며, 그렇지 않은 경우 모델에 따라 서버 공급업체의 설명서 및 모범 사례를 참조하십시오.

운영 체제를 설치합니다

나열된 파일 노드 요구 사항에 따라 지원되는 운영 체제를 "여기"설치합니다. Linux 배포판에 따라 아래의 추가 단계를 참조하십시오.

Red Hat

Red Hat Subscription Manager를 사용하여 시스템을 등록하고 구독하면 공식 Red Hat 저장소에서 필요한 패키지를 설치할 수 있고 지원되는 Red Hat 버전으로 업데이트를 제한할 수 있습니다. `subscription-manager release --set=<MAJOR_VERSION>.<MINOR_VERSION>` . 지침은 다음을 참조하세요. "RHEL 시스템을 등록하고 가입하는 방법" 그리고 "업데이트 제한 방법" .

고가용성을 위해 필요한 패키지가 포함된 Red Hat 리포지토리를 활성화합니다.

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

관리 네트워크를 구성합니다

운영 체제의 대역 내 관리를 허용하는 데 필요한 네트워크 인터페이스를 구성합니다. 정확한 단계는 사용 중인 특정 Linux 배포 및 버전에 따라 다릅니다.

SSH가 활성화되어 있고 Ansible 제어 노드에서 모든 관리 인터페이스에 액세스할 수 있는지 확인합니다.

HCA 및 **HBA** 펌웨어를 업데이트합니다

모든 HBA 및 HCA가 에 나열된 지원되는 펌웨어 버전을 "NetApp 상호 운용성 매트릭스"실행하고 있는지 확인하고 필요한 경우 업그레이드합니다. NVIDIA ConnectX 어댑터에 대한 추가 권장 사항을 찾을 "여기"수 있습니다.

블록 노드

의 단계를 따릅니다 "E-Series와 함께 가동 및 운영합니다" 각 블록 노드 컨트롤러에서 관리 포트를 구성하고 선택적으로 각 시스템의 스토리지 어레이 이름을 설정합니다.

Ansible 제어 노드에서 모든 블록 노드에 액세스할 수 있도록 보장하는 추가 구성은 필요하지 않습니다. 나머지 시스템 구성은 Ansible을 사용하여 적용/유지합니다.

Ansible Control Node 설정

파일 시스템을 배포 및 관리하기 위해 Ansible 제어 노드를 설정합니다.

개요

Ansible 제어 노드는 클러스터를 관리하는 데 사용되는 물리적 또는 가상 Linux 시스템입니다. 다음 요구 사항을 충족해야 합니다.

- "요구 사항"Ansible, Python 및 추가 Python 패키지의 설치 버전을 비롯한 BeeGFS HA 역할을 소개합니다.
- 공무원도 만나세요 "Ansible 제어 노드 요구사항" 운영 체제 버전을 포함합니다.
- 모든 파일 및 블록 노드에 대한 SSH 및 HTTPS 액세스 권한 보유

자세한 설치 단계를 찾을 수 "여기"있습니다.

BeeGFS 파일 시스템을 정의합니다

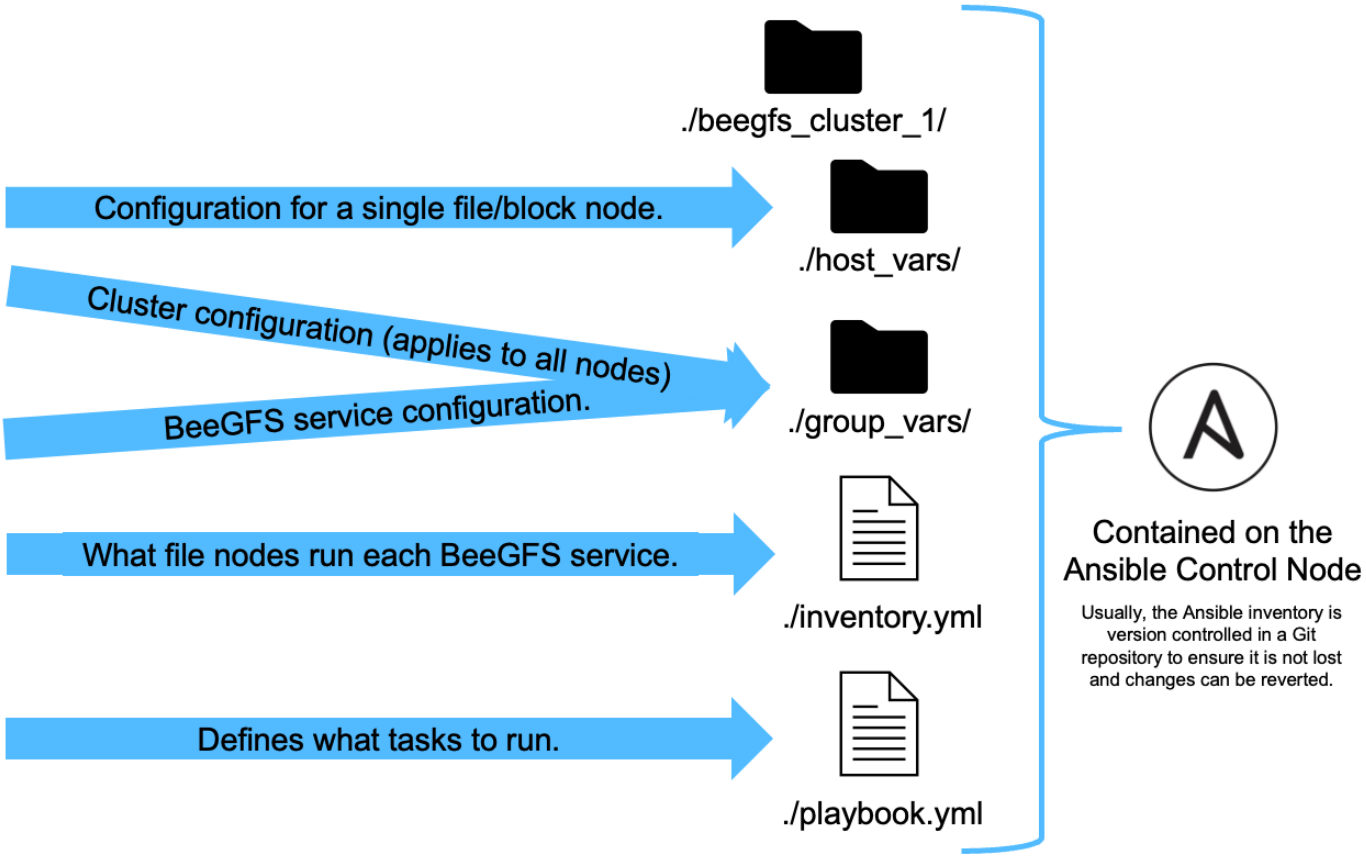
Ansible 인벤토리 개요

Ansible 인벤토리는 원하는 BeeGFS HA 클러스터를 정의하는 구성 파일 세트입니다.

개요

을 구성하기 위한 표준 Ansible 관행을 따르는 것이 좋습니다 "인벤토리"을 참조하십시오 "하위 디렉토리/파일" 전체 재고를 한 파일에 저장하는 대신

단일 BeeGFS HA 클러스터의 Ansible 인벤토리는 다음과 같이 구성됩니다.



단일 BeeGFS 파일 시스템이 여러 HA 클러스터에 걸쳐 있을 수 있으므로 대규모 설치의 경우 여러 Ansible 재고가 있을 수 있습니다. 일반적으로 문제를 피하기 위해 여러 HA 클러스터를 단일 Ansible 인벤토리로 정의하지 않는 것이 좋습니다.

단계

1. Ansible 제어 노드에서 구축할 BeeGFS 클러스터의 Ansible 인벤토리가 포함된 빈 디렉토리를 생성합니다.
   a. 파일 시스템에 여러 HA 클러스터가 포함될 수 있는 경우 먼저 파일 시스템에 대한 디렉토리를 생성한 다음 각 HA 클러스터를 나타내는 인벤토리에 대한 하위 디렉토리를 생성하는 것이 좋습니다. 예를 들면 다음과 같습니다.

```
beegfs_file_system_1/
    beegfs_cluster_1/
    beegfs_cluster_2/
    beegfs_cluster_N/
```

2. 구축할 HA 클러스터의 인벤토리가 포함된 디렉토리에서 두 개의 디렉토리를 생성합니다 `group_vars` 및 `host_vars` 두 개의 파일이 있습니다 `inventory.yml` 및 `playbook.yml`.

다음 섹션에서는 이러한 각 파일의 내용을 정의하는 방법을 설명합니다.

파일 시스템 계획

Ansible 재고를 구축하기 전에 파일 시스템 배포를 계획하십시오.

개요

파일 시스템을 구축하기 전에 클러스터에서 실행 중인 모든 파일 노드, 블록 노드 및 BeeGFS 서비스에 필요한 IP 주소, 포트 및 기타 구성을 정의해야 합니다. 정확한 구성은 클러스터의 아키텍처에 따라 다르지만 이 섹션에서는 일반적으로 적용되는 모범 사례와 후속 단계를 정의합니다.

단계

1. IP 기반 스토리지 프로토콜(예: iSER, iSCSI, NVMe/IB 또는 NVMe/RoCE)을 사용하여 파일 노드를 블록 노드에 연결하는 경우, 각 구성 요소에 대해 다음 워크시트를 작성하십시오. 단일 빌딩 블록의 각 직접 접속은 고유한 서브넷을 가져야 하며, 클라이언트-서버 접속에 사용되는 서브넷과 중복되지 않아야 합니다.

파일 노드	IB 포트	IP 주소입니다	블록 노드	IB 포트	물리적 IP	가상 IP(HDR IB를 사용하는 EF600용)
<HOSTNAME>	<PORT>	<IP/SUBNET>	<HOSTNAME>	<PORT>	<IP/SUBNET>	<IP/SUBNET>

> 각 빌딩 블록의 파일 및 블록 노드가 직접 연결된 경우 여러 빌딩 블록에 동일한 IP/스키마를 재사용할 수 있습니다.

2. 스토리지 네트워크에 InfiniBand 또는 RoCE(RDMA over Converged Ethernet)를 사용하는 경우, 다음 워크시트를 작성하여 HA 클러스터 서비스, BeeGFS 파일 서비스 및 클라이언트가 통신할 수 있는 IP 범위를 결정하십시오.

목적	**InfiniBand** 포트입니다	**IP** 주소 또는 범위입니다
BeeGFS 클러스터 IP	<INTERFACE(s)>	<RANGE>
BeeGFS 관리	<INTERFACE(s)>	<IP(s)>
BeeGFS 메타데이터	<INTERFACE(s)>	<RANGE>
BeeGFS 스토리지	<INTERFACE(s)>	<RANGE>
BeeGFS 클라이언트	<INTERFACE(s)>	<RANGE>

a. 단일 IP 서브넷을 사용하는 경우 워크시트가 하나만 필요합니다. 그렇지 않으면 두 번째 서브넷에 대한 워크시트도 작성하십시오.

3. 위의 내용을 토대로 클러스터의 각 구성 요소에 대해 실행할 BeeGFS 서비스를 정의하는 다음 워크시트를 작성하십시오. 각 서비스에 대해 기본 설정/보조 파일 노드, 네트워크 포트, 부동 IP, NUMA 영역 할당(필요한 경우) 및 대상에 사용할 블록 노드를 지정합니다. 워크시트를 작성할 때 다음 지침을 참조하십시오.

   a. BeeGFS 서비스 중 하나를 지정합니다 `mgmt.yml`, `meta_<ID>.yml`, 또는 `storage_<ID>.yml` 여기서 ID는 이 파일 시스템에서 해당 유형의 모든 BeeGFS 서비스에 대한 고유 번호입니다. 이 규칙은 각 서비스를 구성하는 파일을 생성하는 동안 다음 섹션에서 이 워크시트를 간단하게 참조할 수 있도록 합니다.

   b. BeeGFS 서비스의 포트는 특정 구성 요소 전체에서 고유해야 합니다. 포트 충돌을 방지하기 위해 동일한 포트 번호의 서비스를 동일한 파일 노드에서 실행할 수 없도록 합니다.

   c. 필요한 서비스가 둘 이상의 블록 노드 및/또는 스토리지 풀의 볼륨을 사용할 수 있는 경우(모든 볼륨을 동일한 컨트롤러가 소유할 필요는 없음) 또한 여러 서비스에서 동일한 블록 노드 및/또는 스토리지 풀 구성을 공유할 수 있습니다(개별 볼륨은 이후 섹션에서 정의).

BeeGFS 서비스(파일 이름)	파일 노드	포트	유동 **IP**	**NUMA** 영역	블록 노드	스토리지 풀	소유 컨트롤러
<SERVICE TYPE>_<ID>.대칭	<PREFERRED FILE NODE> <SECONDARY FILE NODE(s)>	<PORT>	<INTERFACE>:<IP/SUBNET> <INTERFACE>:<IP/SUBNET>	<NUMA NODE/ZONE>	<BLOCK NODE>	<STORAGE POOL/VOLUME GROUP>	<A OR B>

표준 규칙, 모범 사례 및 작성된 예제 워크시트에 대한 자세한 내용은 "모범 사례""BeeGFS 구성 요소를 정의합니다 "NetApp 검증 아키텍처에서 BeeGFS의 및 섹션을 참조하십시오.

파일 및 블록 노드 정의

개별 파일 노드를 구성합니다

호스트 변수(host_vars)를 사용하여 개별 파일 노드의 구성을 지정합니다.

개요

이 섹션에서는 를 채우는 방법을 설명합니다 `host_vars/<FILE_NODE_HOSTNAME>.yml` 클러스터에 있는 각 파일 노드에 대한 파일입니다. 이러한 파일에는 특정 파일 노드에 고유한 구성만 포함되어야 합니다. 여기에는 일반적으로 다음이 포함됩니다.

- Ansible에서 노드에 연결하는 데 사용해야 하는 IP 또는 호스트 이름 정의
- 다른 파일 노드와 통신하기 위해 HA 클러스터 서비스(박동조율기 및 Corosync)에 사용되는 추가 인터페이스 및 클러스터 IP를 구성합니다. 기본적으로 이러한 서비스는 관리 인터페이스와 동일한 네트워크를 사용하지만 이중화를 위해 추가 인터페이스를 사용할 수 있어야 합니다. 일반적으로 추가 클러스터 또는 관리 네트워크가 필요하지 않도록 스토리지 네트워크에 추가 IP를 정의하는 것이 좋습니다.
  - 클러스터 통신에 사용되는 모든 네트워크의 성능은 파일 시스템 성능에 중요하지 않습니다. 기본 클러스터 구성에서 일반적으로 1Gb/s 이상의 네트워크는 노드 상태 동기화 및 클러스터 리소스 상태 변경 조정과 같은 클러스터 작업에 충분한 성능을 제공합니다. 느리거나 사용량이 많은 네트워크는 리소스 상태 변경이 평소보다 오래 걸릴 수 있으며, 극단적인 경우 적절한 시간 내에 하트비트를 전송할 수 없는 경우 클러스터에서 노드가

제거될 수 있습니다.

- 원하는 프로토콜을 통해 블록 노드에 연결하는 데 사용되는 인터페이스 구성(예: iSCSI/iSER, NVMe/IB, NVMe/RoCE, FCP 등)

단계

"파일 시스템 계획"섹션에 정의된 IP 주소 지정 체계를 참조하여 클러스터의 각 파일 노드에 대해 파일을 `host_vars/<FILE_NODE_HOSTNAME>/yml` 생성하고 다음과 같이 채웁니다.

1. 맨 위에서 Ansible이 노드에 SSH를 통해 사용하고 관리해야 하는 IP 또는 호스트 이름을 지정합니다.

```
ansible_host: "<MANAGEMENT_IP>"
```

2. 클러스터 트래픽에 사용할 수 있는 추가 IP 구성:
   a. 네트워크 유형이 인 경우 "InfiniBand(IPoIB 사용)":

```
eseries_ipoib_interfaces:
- name: <INTERFACE>  # Example: ib0 or i1b
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   b. 네트워크 유형이 인 경우 "RoCE(RDMA over Converged Ethernet)":

```
eseries_roce_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   c. 네트워크 유형이 인 경우 "이더넷(TCP 전용, RDMA 없음)":

```
eseries_ip_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

3. 클러스터 트래픽에 사용해야 하는 IP를 표시하고 기본 IP가 더 높게 나열됨:

```
beegfs_ha_cluster_node_ips:
- <MANAGEMENT_IP> # Including the management IP is typically but not
required.
- <IP_ADDRESS>    # Ex: 100.127.100.1
- <IP_ADDRESS>    # Additional IPs as needed.
```

2단계에서 구성된 IPS는 에 포함되지 않는 한 클러스터 IP로 사용되지 않습니다 `beegfs_ha_cluster_node_ips` 목록. 따라서 필요한 경우 다른 목적으로 사용할 수 있는 Ansible을 사용하여 추가 IP/인터페이스를 구성할 수 있습니다.

4. 파일 노드가 IP 기반 프로토콜을 통해 블록 노드와 통신해야 하는 경우 IP를 적절한 인터페이스와 해당 프로토콜에 필요한 모든 패키지를 설치/구성해야 합니다.

   a. 를 사용하는 경우 "iSCSI":

   ```
   eseries_iscsi_interfaces:
   - name: <INTERFACE>  # Example: eth0.
     address: <IP/SUBNET> # Example: 100.127.100.1/16
   ```

   b. 를 사용하는 경우 "iSER":

   ```
   eseries_ib_iser_interfaces:
   - name: <INTERFACE>  # Example: ib0.
     address: <IP/SUBNET> # Example: 100.127.100.1/16
     configure: true # If the file node is directly connected to the
   block node set to true to setup OpenSM.
   ```

   c. 를 사용하는 경우 "NVMe/IB":

   ```
   eseries_nvme_ib_interfaces:
   - name: <INTERFACE>  # Example: ib0.
     address: <IP/SUBNET> # Example: 100.127.100.1/16
     configure: true # If the file node is directly connected to the
   block node set to true to setup OpenSM.
   ```

   d. 를 사용하는 경우 "NVMe/RoCE":

   ```
   eseries_nvme_roce_interfaces:
   - name: <INTERFACE>  # Example: eth0.
     address: <IP/SUBNET> # Example: 100.127.100.1/16
   ```

   e. 기타 프로토콜:

i. 를 사용하는 경우 "NVMe/FC", 개별 인터페이스를 구성할 필요가 없습니다. BeeGFS 클러스터 배포는 자동으로 프로토콜을 감지하고 필요에 따라 요구 사항을 설치/구성합니다. 패브릭을 사용하여 파일 및 블록 노드를 연결하는 경우, 스위치가 NetApp과 스위치 공급업체의 모범 사례에 따라 적절히 조닝되었는지 확인하십시오.

ii. FCP 또는 SAS를 사용하는 경우 추가 소프트웨어를 설치하거나 구성할 필요가 없습니다. FCP를 사용하는 경우 스위치가 다음에 적절하게 조닝(zoning)되어 있는지 확인합니다 "넷엡" 스위치 공급업체의 모범 사례를 소개합니다.

iii. 현재 IB SRP 사용은 권장되지 않습니다. E-Series 블록 노드에서 지원하는 것에 따라 NVMe/IB 또는 iSER을 사용합니다.

을 클릭합니다 "여기" 단일 파일 노드를 나타내는 전체 인벤토리 파일의 예

고급: 이더넷과 **InfiniBand** 모드 간에 **NVIDIA ConnectX VPI** 어댑터를 전환합니다

NVIDIA ConnectX-Virtual Protocol Interconnect & reg;(VPI) 어댑터는 전송 계층으로 InfiniBand와 이더넷을 모두 지원합니다. 모드 간 전환은 자동으로 조정되지 않으며 에 포함된 오픈 소스 패키지인 에 포함된 도구를 사용하여 구성해야 `mstconfig` `mstflint` "NVIDIA 펌웨어 도구(MFT)"합니다. 어댑터 모드 변경은 한 번만 수행하면 됩니다. 이 작업을 수동으로 수행하거나 인벤토리 섹션을 사용하여 구성된 인터페이스의 일부로 Ansible 인벤토리에 포함시켜 자동으로 확인/적용할 수 `eseries-[ib|ib_iser|ipoib|nvme_ib|nvme_roce|roce]_interfaces:` 있습니다.

예를 들어, InfiniBand 모드에서 인터페이스 전류를 이더넷으로 변경하여 RoCE에 사용할 수 있습니다.

1. 구성할 각 인터페이스에 대해 지정합니다 `mstconfig` 를 지정하는 매핑(또는 사전)으로 지정합니다 `LINK_TYPE_P<N>` 위치 `<N>` 인터페이스에 대한 HCA의 포트 번호로 결정됩니다. 를 클릭합니다 `<N>` 값은 를 실행하여 확인할 수 있습니다 `grep PCI_SLOT_NAME /sys/class/net/<INTERFACE_NAME>/device/uevent` PCI 슬롯 이름의 성에 1을 추가하고 10진수로 변환합니다.

   a. 예를 들어, 를 입력합니다 `PCI_SLOT_NAME=0000:2f:00.2` (2+1 → HCA 포트 3) → `LINK_TYPE_P3: eth`:

```
eseries_roce_interfaces:
- name: <INTERFACE>
  address: <IP/SUBNET>
  mstconfig:
    LINK_TYPE_P3: eth
```

자세한 내용은 를 참조하십시오 "NetApp E-Series 호스트 컬렉션의 문서입니다" 사용 중인 인터페이스 유형 /프로토콜의 경우.

개별 블록 노드를 구성합니다

호스트 변수(host_vars)를 사용하여 개별 블록 노드의 구성을 지정합니다.

개요

이 섹션에서는 를 채우는 방법을 설명합니다 `host_vars/<BLOCK_NODE_HOSTNAME>.yml` 클러스터에 있는 각 블록 노드에 대한 파일입니다. 이러한 파일에는 특정 블록 노드에 고유한 설정만 포함되어야 합니다. 여기에는

일반적으로 다음이 포함됩니다.

- 시스템 이름(System Manager에 표시됨)
- 컨트롤러 중 하나에 대한 HTTPS URL(REST API를 사용하여 시스템을 관리하는 데 사용됨)
- 이 블록 노드에 연결하기 위해 사용하는 스토리지 프로토콜 파일 노드
- IP 주소(필요한 경우)와 같은 HIC(호스트 인터페이스 카드) 포트 구성

단계

"파일 시스템 계획"섹션에 정의된 IP 주소 지정 체계를 참조하여 클러스터의 각 블록 노드에 대해 파일을 `host_vars/<BLOCK_NODE_HOSTNAME>/yml` 생성하고 다음과 같이 채웁니다.

1. 맨 위에서 컨트롤러 중 하나에 대한 시스템 이름과 HTTPS URL을 지정합니다.

```
eseries_system_name: <SYSTEM_NAME>
eseries_system_api_url:
https://<MANAGEMENT_HOSTNAME_OR_IP>:8443/devmgr/v2/
```

2. 를 선택합니다 "프로토콜" 파일 노드는 를 사용하여 이 블록 노드에 접속합니다.
   a. 지원되는 프로토콜: `auto`, `iscsi`, `fc`, `sas`, `ib_srp`, `ib_iser`, `nvme_ib`, `nvme_fc`, `nvme_roce`.

```
eseries_initiator_protocol: <PROTOCOL>
```

3. 사용 중인 프로토콜에 따라 HIC 포트는 추가 구성이 필요할 수 있습니다. 필요한 경우 HIC 포트 구성을 정의하여 각 컨트롤러 구성의 상위 항목이 각 컨트롤러의 물리적 가장 왼쪽 포트 및 하단 포트가 가장 오른쪽 포트와 대응하도록 해야 합니다. 모든 포트는 현재 사용되지 않는 경우에도 유효한 구성이 필요합니다.

> EF600 블록 노드와 함께 HDR(200GB) InfiniBand 또는 200GB RoCE를 사용하는 경우에는 아래 섹션도 참조하십시오.

   a. iSCSI의 경우:

```
eseries_controller_iscsi_port:
  controller_a:         # Ordered list of controller A channel
definition.
    - state:            # Whether the port should be enabled.
Choices: enabled, disabled
      config_method:    # Port configuration method Choices: static,
dhcp
      address:          # Port IPv4 address
      gateway:          # Port IPv4 gateway
      subnet_mask:      # Port IPv4 subnet_mask
      mtu:              # Port IPv4 mtu
    - (...)             # Additional ports as needed.
  controller_b:         # Ordered list of controller B channel
definition.
    - (...)             # Same as controller A but for controller B

# Alternatively the following common port configuration can be
defined for all ports and omitted above:
eseries_controller_iscsi_port_state: enabled         # Generally
specifies whether a controller port definition should be applied
Choices: enabled, disabled
eseries_controller_iscsi_port_config_method: dhcp    # General port
configuration method definition for both controllers. Choices:
static, dhcp
eseries_controller_iscsi_port_gateway:               # General port
IPv4 gateway for both controllers.
eseries_controller_iscsi_port_subnet_mask:           # General port
IPv4 subnet mask for both controllers.
eseries_controller_iscsi_port_mtu: 9000              # General port
maximum transfer units (MTU) for both controllers. Any value greater
than 1500 (bytes).
```

b. iSER의 경우:

```
eseries_controller_ib_iser_port:
  controller_a:    # Ordered list of controller A channel address
definition.
    -              # Port IPv4 address for channel 1
    - (...)        # So on and so forth
  controller_b:    # Ordered list of controller B channel address
definition.
```

c. NVMe/IB의 경우:

```
eseries_controller_nvme_ib_port:
  controller_a:    # Ordered list of controller A channel address
definition.
    -              # Port IPv4 address for channel 1
    - (...)        # So on and so forth
  controller_b:    # Ordered list of controller B channel address
definition.
```

d. NVMe/RoCE의 경우:

```
eseries_controller_nvme_roce_port:
  controller_a:         # Ordered list of controller A channel
definition.
    - state:            # Whether the port should be enabled.
      config_method:    # Port configuration method Choices: static,
dhcp
      address:          # Port IPv4 address
      subnet_mask:      # Port IPv4 subnet_mask
      gateway:          # Port IPv4 gateway
      mtu:              # Port IPv4 mtu
      speed:            # Port IPv4 speed
  controller_b:         # Ordered list of controller B channel
definition.
    - (...)             # Same as controller A but for controller B

# Alternatively the following common port configuration can be
defined for all ports and omitted above:
eseries_controller_nvme_roce_port_state: enabled         # Generally
specifies whether a controller port definition should be applied
Choices: enabled, disabled
eseries_controller_nvme_roce_port_config_method: dhcp    # General
port configuration method definition for both controllers. Choices:
static, dhcp
eseries_controller_nvme_roce_port_gateway:               # General
port IPv4 gateway for both controllers.
eseries_controller_nvme_roce_port_subnet_mask:           # General
port IPv4 subnet mask for both controllers.
eseries_controller_nvme_roce_port_mtu: 4200              # General
port maximum transfer units (MTU). Any value greater than 1500
(bytes).
eseries_controller_nvme_roce_port_speed: auto            # General
interface speed. Value must be a supported speed or auto for
automatically negotiating the speed with the port.
```

e. FC 및 SAS 프로토콜은 추가 구성이 필요하지 않습니다. SRP가 올바르게 권장되지 않습니다.

iSCSI CHAP 구성 기능을 포함하여 HIC 포트 및 호스트 프로토콜을 구성하는 추가 옵션은 을 참조하십시오 "문서화" SANtricity 컬렉션에 포함되어 있습니다. 참고 BeeGFS를 구축할 때 스토리지 풀, 볼륨 구성 및 스토리지 용량 할당의 기타 측면은 다른 위치에 구성되며 이 파일에 정의하면 안 됩니다.

을 클릭합니다 "여기" 단일 블록 노드를 나타내는 전체 인벤토리 파일의 예

HDR(200GB) InfiniBand 또는 **200GB RoCE**와 **NetApp EF600** 블록 노드 사용**:**

EF600에서 HDR(200GB) InfiniBand를 사용하려면 각 물리적 포트에 대해 두 번째 "가상" IP를 구성해야 합니다. 다음은 이중 포트 InfiniBand HDR HIC가 장착된 EF600을 구성하는 올바른 방법의 예입니다.

```
eseries_controller_nvme_ib_port:
  controller_a:
    - 192.168.1.101   # Port 2a (virtual)
    - 192.168.2.101   # Port 2b (virtual)
    - 192.168.1.100   # Port 2a (physical)
    - 192.168.2.100   # Port 2b (physical)
  controller_b:
    - 192.168.3.101   # Port 2a (virtual)
    - 192.168.4.101   # Port 2b (virtual)
    - 192.168.3.100   # Port 2a (physical)
    - 192.168.4.100   # Port 2b (physical)
```

일반 파일 노드 구성을 지정합니다

그룹 변수(group_vars)를 사용하여 일반 파일 노드 구성을 지정합니다.

개요

모든 파일 노드에 대해 사과해야 하는 구성은 에 정의되어 있습니다 `group_vars/ha_cluster.yml`. 일반적으로 다음과 같은 기능이 있습니다.

- 각 파일 노드에 연결 및 로그인하는 방법에 대한 세부 정보
- 공통 네트워킹 구성
- 자동 재부팅이 허용되는지 여부
- 방화벽 및 SELinux 상태를 구성하는 방법
- 경고 및 펜싱을 포함한 클러스터 구성
- 성능 튜닝:
- 공통 BeeGFS 서비스 구성

이 파일에 설정된 옵션은 혼합 하드웨어 모델을 사용 중이거나 각 노드에 대해 다른 암호를 사용하는 경우와 같이 개별 파일 노드에서도 정의할 수 있습니다. 개별 파일 노드의 구성은 이 파일의 구성보다 우선합니다.

단계

파일을 만듭니다 `group_vars/ha_cluster.yml` 다음과 같이 채웁니다.

1. Ansible Control 노드가 원격 호스트에서 인증해야 하는 방법을 나타냅니다.

```
ansible_ssh_user: root
ansible_become_password: <PASSWORD>
```

> 특히 프로덕션 환경에서는 암호를 일반 텍스트로 저장하지 마십시오. 대신 Ansible Vault를 사용하십시오(참조 "Ansible Vault로 콘텐츠 암호화") 또는 을 누릅니다 `--ask-become-pass` 옵션을 클릭합니다. 를 누릅니다 `ansible_ssh_user` 이(가) 이미 루트이므로 필요에 따라 를 생략할 수 있습니다 `ansible_become_password`.

2. 이더넷 또는 InfiniBand 인터페이스(예: 클러스터 IP)에서 정적 IP를 구성하고 여러 인터페이스가 동일한 IP 서브넷에 있는 경우(예: ib0이 192.168.1.10/24를 사용하고 ib1이 192.168.1.11/24를 사용 중인 경우) 멀티홈 지원이 제대로 작동하려면 추가 IP 라우팅 테이블 및 규칙을 설정해야 합니다. 다음과 같이 제공된 네트워크 인터페이스 구성 후크를 활성화하기만 하면 됩니다.

```
eseries_ip_default_hook_templates:
  - 99-multihoming.j2
```

3. 클러스터를 구축할 때 스토리지 프로토콜에 따라 노드를 재부팅하여 원격 블록 장치(E-Series 볼륨)를 쉽게 검색하거나 다른 구성 요소를 적용해야 할 수 있습니다. 기본적으로 노드를 재부팅하기 전에 프롬프트가 표시되지만 다음을 지정하여 노드를 자동으로 다시 시작할 수 있습니다.

```
eseries_common_allow_host_reboot: true
```

   a. 재부팅 후 기본적으로 블록 장치 및 기타 서비스가 준비되도록 하려면 Ansible이 시스템이 준비될 때까지 기다립니다 `default.target` 에 도달한 후 배포를 계속합니다. NVMe/IB가 사용 중인 일부 시나리오에서는 이 시간이 부족하여 원격 장치를 초기화, 검색 및 연결할 수 없습니다. 이로 인해 자동화된 배포가 조기에 계속 진행되어 실패할 수 있습니다. NVMe/IB를 사용할 때 이를 방지하려면 다음을 정의합니다.

```
eseries_common_reboot_test_command: "! systemctl status
eseries_nvme_ib.service || systemctl --state=exited | grep
eseries_nvme_ib.service"
```

4. BeeGFS 및 HA 클러스터 서비스가 통신하려면 다양한 방화벽 포트가 필요합니다. 첫 번째 명령을 수동으로 구성하지 않는 한(권장하지 않음) 필요한 방화벽 영역을 만들고 포트를 자동으로 열리도록 다음을 지정합니다.

```
beegfs_ha_firewall_configure: True
```

5. 이때 SELinux는 지원되지 않으며, 특히 RDMA를 사용하는 경우 충돌을 피하기 위해 상태를 비활성화로 설정하는 것이 좋습니다. SELinux가 비활성화되었는지 확인하려면 다음을 설정합니다.

```
eseries_beegfs_ha_disable_selinux: True
eseries_selinux_state: disabled
```

6. 파일 노드가 통신할 수 있도록 인증을 구성하고 조직의 정책에 따라 필요에 따라 기본값을 조정합니다.

```
beegfs_ha_cluster_name: hacluster                    # BeeGFS HA cluster
name.
beegfs_ha_cluster_username: hacluster                # BeeGFS HA cluster
username.
beegfs_ha_cluster_password: hapassword               # BeeGFS HA cluster
username's password.
beegfs_ha_cluster_password_sha512_salt: randomSalt # BeeGFS HA cluster
username's password salt.
```

7. "파일 시스템 계획"섹션을 기반으로 이 파일 시스템에 대한 BeeGFS 관리 IP를 지정합니다.

```
beegfs_ha_mgmtd_floating_ip: <IP ADDRESS>
```

> ⓘ 중복된 것처럼 보이지만 BeeGFS 파일 시스템을 단일 HA 클러스터 이상으로 확장하는 경우 "begfs_ha_mgmtd_floating_ip"가 중요합니다. 이후 HA 클러스터는 추가 BeeGFS 관리 서비스 없이 구축되고 첫 번째 클러스터에서 제공하는 관리 서비스를 가리키도록 구축됩니다.

8. 원하는 경우 e-메일 알림 활성화:

```
beegfs_ha_enable_alerts: True
# E-mail recipient list for notifications when BeeGFS HA resources
change or fail.
beegfs_ha_alert_email_list: ["<EMAIL>"]
# This dictionary is used to configure postfix service
(/etc/postfix/main.cf) which is required to set email alerts.
beegfs_ha_alert_conf_ha_group_options:
      # This parameter specifies the local internet domain name. This is
optional when the cluster nodes have fully qualified hostnames (i.e.
host.example.com)
      mydomain: <MY_DOMAIN>
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
```

9. 펜싱을 사용하는 것이 좋습니다. 그렇지 않으면 기본 노드에 장애가 발생할 때 보조 노드에서 서비스가 시작되지

않도록 차단할 수 있습니다.

a. 다음을 지정하여 펜싱을 전역적으로 활성화합니다.

```
beegfs_ha_cluster_crm_config_options:
  stonith-enabled: True
```

i. 참고 필요한 경우 여기에서 지원되는 모든 항목을 "클러스터 속성" 지정할 수도 있습니다. BeeGFS HA의 역할은 잘 테스트된 많은 테스트를 거치므로 이러한 조정이 필요하지 않습니다. "기본값"

b. 그런 다음 펜싱 에이전트를 선택하고 구성합니다.

i. 옵션 1: APC PDU(Power Distribution Unit)를 사용하여 펜싱 활성화하기:

```
beegfs_ha_fencing_agents:
  fence_apc:
    - ipaddr: <PDU_IP_ADDRESS>
      login: <PDU_USERNAME>
      passwd: <PDU_PASSWORD>
      pcmk_host_map:
"<HOSTNAME>:<PDU_PORT>,<PDU_PORT>;<HOSTNAME>:<PDU_PORT>,<PDU_PORT>
"
```

ii. 옵션 2: Lenovo XCC(및 기타 BMC)에서 제공하는 Redfish API를 사용하여 펜싱을 활성화하려면

```
redfish: &redfish
  username: <BMC_USERNAME>
  password: <BMC_PASSWORD>
  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
```

iii. 다른 펜싱 에이전트 구성에 대한 자세한 내용은 을 "Red Hat 문서"참조하십시오.

10. BeeGFS HA 역할은 다양한 튜닝 매개 변수를 적용하여 성능을 더욱 최적화할 수 있습니다. 여기에는 커널 메모리 활용도 최적화 및 블록 디바이스 입출력 등이 포함됩니다. 이 역할은 NetApp E-Series 블록 노드를 사용한 테스트를 기반으로 하는 적절한 집합을 "기본값" 제공하지만, 기본적으로 다음을 지정하지 않으면 적용되지 않습니다.

```
beegfs_ha_enable_performance_tuning: True
```

a. 필요한 경우 여기에서 기본 성능 튜닝에 대한 변경 사항도 지정합니다. 자세한 내용은 전체 설명서를 "성능 튜닝 매개 변수" 참조하십시오.

11. BeeGFS 서비스에 사용되는 부동 IP 주소(논리 인터페이스라고도 함)가 파일 노드 간에 페일오버할 수 있도록 모든 네트워크 인터페이스의 이름이 일관되게 지정되어야 합니다. 기본적으로 네트워크 인터페이스 이름은 동일한 PCIe 슬롯에 네트워크 어댑터가 설치된 동일한 서버 모델에서도 일관된 이름을 생성한다는 보장이 없는 커널에 의해 생성됩니다. 이 기능은 장비를 구축하고 생성된 인터페이스 이름을 알 수 있도록 하기 전에 인벤토리를 생성할 때도 유용합니다. 서버 또는 의 블록 다이어그램을 기반으로 일관된 장치 이름을 보장합니다 `lshw -class network -businfo` 출력에서 원하는 PCIe 주소-논리 인터페이스 매핑을 다음과 같이 지정합니다.

a. InfiniBand(IPoIB) 네트워크 인터페이스의 경우:

```
eseries_ipoib_udev_rules:
  "<PCIe ADDRESS>": <NAME> # Ex: 0000:01:00.0: i1a
```

b. 이더넷 네트워크 인터페이스의 경우:

```
eseries_ip_udev_rules:
  "<PCIe ADDRESS>": <NAME> # Ex: 0000:01:00.0: e1a
```

인터페이스의 이름을 바꿀 때(이름을 바꿀 수 없음) 충돌을 방지하려면 eth0, ens9f0, ib0 또는 ibs4f0과 같은 잠재적인 기본 이름을 사용하지 않아야 합니다. 일반적인 명명 규칙은 이더넷 또는 InfiniBand의 경우 'e' 또는 'i'를 사용하고 그 뒤에 PCIe 슬롯 번호와 해당 포트를 나타내는 문자를 사용하는 것입니다. 예를 들어 슬롯 3에 설치된 InfiniBand 어댑터의 두 번째 포트는 i3b입니다.

검증된 파일 노드 모델을 사용하는 경우 를 클릭합니다 "여기" PCIe 주소와 논리적 포트 매핑의 예

12. 선택적으로 클러스터의 모든 BeeGFS 서비스에 적용할 구성을 지정합니다. 기본 구성 값을 찾을 수 "여기"있으며 서비스별 구성은 다른 곳에 지정됩니다.

a. BeeGFS 관리 서비스:

```
beegfs_ha_beegfs_mgmtd_conf_ha_group_options:
  <OPTION>: <VALUE>
```

b. BeeGFS 메타데이터 서비스:

```
beegfs_ha_beegfs_meta_conf_ha_group_options:
  <OPTION>: <VALUE>
```

c. BeeGFS 스토리지 서비스:

```
beegfs_ha_beegfs_storage_conf_ha_group_options:
  <OPTION>: <VALUE>
```

13. BeeGFS 7.2.7 및 7.3.1 "연결 인증" 구성 또는 명시적으로 비활성화해야 합니다. Ansible 기반 배포를 사용하여 다음과 같은 몇 가지 방법으로 이를 구성할 수 있습니다.

a. 기본적으로 배포는 연결 인증을 자동으로 구성하고 을 생성합니다 `connauthfile` 모든 파일 노드에 배포되고 BeeGFS 서비스와 함께 사용됩니다. 이 파일은 또한 의 Ansible 제어 노드에 배치/유지됩니다 `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile` 이 파일 시스템을 액세스해야 하는 클라이언트에서 재사용하기 위해 안전하게 유지해야 하는 경우

i. 새 키 지정을 생성하려면 다음을 지정합니다 `-e "beegfs_ha_conn_auth_force_new=True` Ansible 플레이북을 실행할 때, 참고 의 경우 이 작업은 무시됩니다 `beegfs_ha_conn_auth_secret` 정의됩니다.

ii. 고급 옵션은 에 포함된 전체 기본값 목록을 "BeeGFS HA 역할입니다"참조하십시오.

b. 에서 다음을 정의하여 사용자 지정 암호를 사용할 수 있습니다 `ha_cluster.yml`:

```
beegfs_ha_conn_auth_secret: <SECRET>
```

c. 연결 인증은 완전히 비활성화할 수 있습니다(권장하지 않음).

```
beegfs_ha_conn_auth_enabled: false
```

을 클릭합니다 "여기" 일반 파일 노드 구성을 나타내는 전체 인벤토리 파일의 예

NetApp EF600 블록 노드에서 HDR(200GB) InfiniBand 사용:

EF600에서 HDR(200GB) InfiniBand를 사용하려면 서브넷 관리자가 가상화를 지원해야 합니다. 스위치를 사용하여 파일 및 블록 노드를 연결하는 경우 전체 패브릭의 서브넷 관리자 관리자에서 이 기능을 활성화해야 합니다.

블록 및 파일 노드가 InfiniBand를 사용하여 `opensm` 직접 연결된 경우 블록 노드에 직접 연결된 각 인터페이스에 대해 각 파일 노드에서 의 인스턴스를 구성해야 합니다. 이 작업은 `configure: true` 시기를 지정하여"파일 노드 스토리지 인터페이스를 구성하는 중입니다" 수행합니다.

현재 지원되는 Linux 배포판과 함께 제공된 의 받은 편지함 버전은 `opensm` 가상화를 지원하지 않습니다. 대신 OFED(NVIDIA OpenFabrics Enterprise Distribution)에서 의 버전을 설치하고 구성해야 `opensm` 합니다. Ansible을 사용한 구축도 여전히 지원되지만, 몇 가지 추가 단계가 필요합니다.

1. curl 또는 원하는 툴을 사용하여 NVIDIA 웹 사이트에서 디렉토리로 섹션에 나열된 OpenSM 버전의 패키지를 다운로드합니다. "기술 요구 사항" `<INVENTORY>/packages/` 예를 들면 다음과 같습니다.

```
curl -o packages/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
https://linux.mellanox.com/public/repo/mlnx_ofed/23.10-
3.2.2.0/rhel9.4/x86_64/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
curl -o packages/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
https://linux.mellanox.com/public/repo/mlnx_ofed/23.10-
3.2.2.0/rhel9.4/x86_64/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
```

2. 아래에서 `group_vars/ha_cluster.yml` 다음 구성을 정의합니다.

```
### OpenSM package and configuration information
eseries_ib_opensm_allow_upgrades: true
eseries_ib_opensm_skip_package_validation: true
eseries_ib_opensm_rhel_packages: []
eseries_ib_opensm_custom_packages:
  install:
    - files:
        add:
          "packages/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm": "/tmp/"
          "packages/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm": "/tmp/"
    - packages:
        add:
          - /tmp/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
          - /tmp/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
  uninstall:
    - packages:
        remove:
          - opensm
          - opensm-libs
      files:
        remove:
          - /tmp/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
          - /tmp/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm

eseries_ib_opensm_options:
  virt_enabled: "2"
```

일반 블록 노드 구성을 지정합니다

그룹 변수(group_vars)를 사용하여 일반 블록 노드 구성을 지정합니다.

개요

모든 블록 노드에 대해 사과해야 하는 구성은 에 정의되어 있습니다
`group_vars/eseries_storage_systems.yml`. 일반적으로 다음과 같은 기능이 있습니다.

- Ansible 제어 노드를 블록 노드로 사용되는 E-Series 스토리지 시스템에 연결하는 방법에 대한 자세한 정보
- 노드에서 실행해야 하는 펌웨어, NVSRAM 및 드라이브 펌웨어 버전

- 캐시 설정, 호스트 구성 및 볼륨 프로비저닝 방법에 대한 설정을 포함한 글로벌 구성

> 이 파일에 설정된 옵션은 혼합 하드웨어 모델을 사용 중이거나 각 노드에 대해 다른 암호를 사용하는 경우와 같이 개별 블록 노드에서도 정의할 수 있습니다. 개별 블록 노드의 구성은 이 파일의 구성보다 우선합니다.

단계

파일을 만듭니다 `group_vars/eseries_storage_systems.yml` 다음과 같이 채웁니다.

1. Ansible은 SSH를 사용하여 블록 노드에 연결하지 않고 REST API를 사용합니다. 이를 위해 다음을 설정해야 합니다.

```
ansible_connection: local
```

2. 각 노드를 관리할 사용자 이름과 암호를 지정합니다. 사용자 이름은 선택적으로 생략할 수 있으며 기본적으로 admin이 됩니다. 그렇지 않으면 관리자 권한이 있는 계정을 지정할 수 있습니다. 또한 SSL 인증서를 확인해야 하는지 무시해야 하는지 여부를 지정합니다.

```
eseries_system_username: admin
eseries_system_password: <PASSWORD>
eseries_validate_certs: false
```

> 암호를 일반 텍스트로 나열하는 것은 권장되지 않습니다. Ansible 볼트를 사용하거나 을 제공합니다 `eseries_system_password` VAR을(를) 사용하여 Ansible을 실행하는 경우

3. 노드에 설치할 컨트롤러 펌웨어, NVSRAM 및 드라이브 펌웨어를 선택적으로 지정합니다. 이러한 파일은 로 다운로드해야 합니다 `packages/` Ansible을 실행하기 전 디렉토리: E-Series 컨트롤러 펌웨어 및 NVSRAM을 다운로드할 수 있습니다 "여기" 및 드라이브 펌웨어를 업데이트합니다 "여기":

```
eseries_firmware_firmware: "packages/<FILENAME>.dlp" # Ex.
"packages/RCB_11.80GA_6000_64cc0ee3.dlp"
eseries_firmware_nvsram: "packages/<FILENAME>.dlp" # Ex.
"packages/N6000-880834-D08.dlp"
eseries_drive_firmware_firmware_list:
  - "packages/<FILENAME>.dlp"
  # Additional firmware versions as needed.
eseries_drive_firmware_upgrade_drives_online: true # Recommended unless
BeeGFS hasn't been deployed yet, as it will disrupt host access if set
to "false".
```

> 이 구성을 지정하면 Ansible이 추가 프롬프트 없이 컨트롤러 재부팅(필요한 경우)을 비롯한 모든 펌웨어를 자동으로 업데이트합니다. 이는 BeeGFS/호스트 입출력에 영향을 줄 수 있지만 일시적으로 성능이 저하될 수 있습니다.

4. 글로벌 시스템 구성 기본값을 조정합니다. 여기에 나열된 옵션과 값은 NetApp 기반의 BeeGFS에 일반적으로 권장되지만 필요한 경우 조정할 수 있습니다.

```
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 by default.
```

5. 글로벌 볼륨 프로비저닝 기본값을 구성합니다. 여기에 나열된 옵션과 값은 NetApp 기반의 BeeGFS에 일반적으로 권장되지만 필요한 경우 조정할 수 있습니다.

```
eseries_volume_size_unit: pct # Required by default. This allows volume
capacities to be specified as a percentage, simplifying putting together
the inventory.
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
```

6. 필요한 경우 Ansible에서 스토리지 풀 및 볼륨 그룹을 위한 드라이브를 선택하는 순서를 조정하고 다음 모범 사례를 염두에 두십시오.

   a. 관리 및/또는 메타데이터 볼륨에 먼저 사용되어야 하는 (잠재적으로 작은) 드라이브와 스토리지 볼륨을 나열합니다.

   b. 디스크 쉘프/드라이브 엔클로저 모델을 기준으로 사용 가능한 드라이브 채널 간에 드라이브 선택 순서를 조정해야 합니다. 예를 들어 EF600과 확장 없는 경우 드라이브 0-11은 드라이브 채널 1에 있고 드라이브 12-23은 드라이브 채널에 있습니다. 따라서 드라이브 선택의 균형을 맞추는 전략은 선택입니다 `disk shelf:drive` 99:0, 99:23, 99:1, 99:22 등 하나 이상의 엔클로저가 있는 경우 첫 번째 숫자는 드라이브 쉘프 ID를 나타냅니다.

```
# Optimal/recommended order for the EF600 (no expansion):
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"
```

을 클릭합니다 "여기" 일반 블록 노드 구성을 나타내는 전체 인벤토리 파일의 예

BeeGFS 서비스를 정의합니다

BeeGFS 서비스는 그룹 변수(group_VAR)를 사용하여 구성됩니다.

개요

이 섹션에서는 BeeGFS 관리 서비스 정의에 대해 설명합니다. 특정 파일 시스템에 대한 HA 클러스터에는 이 유형의 서비스 하나만 있어야 합니다. 이 서비스를 구성하는 데는 다음 사항이 포함됩니다.

- 서비스 유형(관리)
- 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다.
- 이 서비스에 연결할 수 있는 하나 이상의 부동 IP(논리 인터페이스)를 구성합니다.
- 이 서비스에 대한 데이터를 저장할 볼륨 위치/방법 지정(BeeGFS 관리 타겟)

단계

새 파일을 `group_vars/mgmt.yml` 만들고 "파일 시스템 계획"섹션을 참조하여 다음과 같이 채웁니다.

1. 이 파일이 BeeGFS 관리 서비스의 구성을 나타냅니다.

```
beegfs_service: management
```

2. 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다. 할당량을 설정해야 하는 경우가 아니면 일반적으로 관리 서비스에 필요하지 않지만 에서 지원되는 구성 매개 변수는 필요하지 않습니다 `beegfs-mgmtd.conf` 포함될 수 있습니다. 참고 다음 매개 변수는 자동으로/다른 위치에 구성되며 여기에서 지정하면 안 됩니다. `storeMgmtdDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, 및 `connNetFilterFile`.

```
beegfs_ha_beegfs_mgmtd_conf_resource_group_options:
  <beegfs-mgmt.conf:key>:<beegfs-mgmt.conf:value>
```

3. 다른 서비스 및 클라이언트가 이 서비스에 연결하는 데 사용할 하나 이상의 부동 IP를 구성합니다. 이렇게 하면 BeeGFS가 자동으로 설정됩니다 `connInterfacesFile` 옵션):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.0/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. 선택적으로, 나가는 통신에 사용할 수 있는 하나 이상의 허용된 IP 서브넷을 지정합니다(이 경우 BeeGFS가 자동으로 설정됩니다) `connNetFilterFile` 옵션):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. 다음 지침에 따라 이 서비스가 데이터를 저장할 BeeGFS 관리 대상을 지정합니다.
   a. 여러 BeeGFS 서비스/타겟에 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용할 수 있으므로 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용하기만 하면 됩니다 `name`, `raid_level`, `criteria_*`, 및 `common_*` 각 서비스에 대한 구성(각 서비스에 대해 나열된 볼륨은 서로 달라야 함)
   b. 볼륨 크기는 스토리지 풀/볼륨 그룹의 백분율로 지정해야 하며, 특정 스토리지 풀/볼륨 그룹을 사용하는 모든 서비스/볼륨에서 합계가 100을 초과해서는 안 됩니다. 참고 SSD를 사용할 경우 SSD 성능 및 마모 수명을 최대화하기 위해 볼륨 그룹에 여유 공간을 두는 것이 좋습니다("여기"자세한 내용을 보려면 클릭).
   c. 을 클릭합니다 "여기" 에서 사용할 수 있는 전체 구성 옵션 목록을 확인하십시오 `eseries_storage_pool_configuration`. 과 같은 일부 옵션을 확인합니다 `state`, `host`, `host_type`, `workload_name`, 및 `workload_metadata` 및 볼륨 이름은 자동으로 생성되며 여기에서 지정할 수 없습니다.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_m1_m2_m5_m6
        raid_level: <LEVEL> # One of: raid1, raid5, raid6, raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

을 클릭합니다 "여기" BeeGFS 관리 서비스를 나타내는 전체 인벤토리 파일의 예

BeeGFS 메타데이터 서비스를 정의합니다

BeeGFS 서비스는 그룹 변수(group_VAR)를 사용하여 구성됩니다.

개요

이 섹션에서는 BeeGFS 메타데이터 서비스 정의에 대해 설명합니다. 특정 파일 시스템에 대한 HA 클러스터에 이 유형의 서비스가 하나 이상 있어야 합니다. 이 서비스를 구성하는 데는 다음 사항이 포함됩니다.

- 서비스 유형(메타데이터)
- 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다.
- 이 서비스에 연결할 수 있는 하나 이상의 부동 IP(논리 인터페이스)를 구성합니다.
- 이 서비스에 대한 데이터를 저장할 볼륨 위치/방법 지정(BeeGFS 메타데이터 타겟)

단계

"파일 시스템 계획"섹션을 참조하여 `group_vars/meta_<ID>.yml` 클러스터의 각 메타데이터 서비스에 대해 에서 파일을 생성하고 다음과 같이 채웁니다.

1. 이 파일이 BeeGFS 메타데이터 서비스에 대한 구성을 나타냅니다.

```
beegfs_service: metadata
```

2. 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다. 최소한 원하는 TCP 및 UDP 포트를 지정해야 하지만 에서 지원되는 구성 매개 변수는 모두 지정해야 합니다 `beegfs-meta.conf` 또한 포함될 수 있습니다. 참고 다음 매개 변수는 자동으로/다른 위치에 구성되며 여기에서 지정하면 안 됩니다. `sysMgmtdHost`, `storeMetaDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, 및 `connNetFilterFile`.

```
beegfs_ha_beegfs_meta_conf_resource_group_options:
  connMetaPortTCP: <TCP PORT>
  connMetaPortUDP: <UDP PORT>
  tuneBindToNumaZone: <NUMA ZONE> # Recommended if using file nodes with
multiple CPU sockets.
```

3. 다른 서비스 및 클라이언트가 이 서비스에 연결하는 데 사용할 하나 이상의 부동 IP를 구성합니다. 이렇게 하면 BeeGFS가 자동으로 설정됩니다 `connInterfacesFile` 옵션):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.1/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. 선택적으로, 나가는 통신에 사용할 수 있는 하나 이상의 허용된 IP 서브넷을 지정합니다(이 경우 BeeGFS가 자동으로 설정됩니다) `connNetFilterFile` 옵션):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. 다음 지침에 따라 이 서비스가 데이터를 저장할 BeeGFS 메타데이터 타겟을 지정합니다. 이렇게 하면 가 자동으로 구성됩니다 `storeMetaDirectory` 옵션):

   a. 여러 BeeGFS 서비스/타겟에 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용할 수 있으므로 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용하기만 하면 됩니다 `name`, `raid_level`, `criteria_*`, 및 `common_*` 각 서비스에 대한 구성(각 서비스에 대해 나열된 볼륨은 서로 달라야 함)

   b. 볼륨 크기는 스토리지 풀/볼륨 그룹의 백분율로 지정해야 하며, 특정 스토리지 풀/볼륨 그룹을 사용하는 모든 서비스/볼륨에서 합계가 100을 초과해서는 안 됩니다. 참고 SSD를 사용할 경우 SSD 성능 및 마모 수명을 최대화하기 위해 볼륨 그룹에 여유 공간을 두는 것이 좋습니다("여기"자세한 내용을 보려면 클릭).

c. 을 클릭합니다 "여기" 에서 사용할 수 있는 전체 구성 옵션 목록을 확인하십시오 `eseries_storage_pool_configuration`. 과 같은 일부 옵션을 확인합니다 `state`, `host`, `host_type`, `workload_name`, 및 `workload_metadata` 및 볼륨 이름은 자동으로 생성되며 여기에서 지정할 수 없습니다.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_m1_m2_m5_m6
        raid_level: <LEVEL> # One of: raid1, raid5, raid6, raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

을 클릭합니다 "여기" BeeGFS 메타데이터 서비스를 나타내는 전체 인벤토리 파일의 예

BeeGFS 스토리지 서비스를 정의합니다

BeeGFS 서비스는 그룹 변수(group_VAR)를 사용하여 구성됩니다.

개요

이 섹션에서는 BeeGFS 스토리지 서비스 정의를 안내합니다. 특정 파일 시스템에 대한 HA 클러스터에 이 유형의 서비스가 하나 이상 있어야 합니다. 이 서비스를 구성하는 데는 다음 사항이 포함됩니다.

- 서비스 유형(스토리지)
- 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다.
- 이 서비스에 연결할 수 있는 하나 이상의 부동 IP(논리 인터페이스)를 구성합니다.
- 이 서비스에 대한 데이터를 저장할 볼륨 위치/방법 지정(BeeGFS 스토리지 타겟)

단계

"파일 시스템 계획"섹션을 참조하여 `group_vars/stor_<ID>.yml` 클러스터의 각 스토리지 서비스에 대해 에서 파일을 생성하고 다음과 같이 채웁니다.

1. 이 파일이 BeeGFS 스토리지 서비스에 대한 구성을 나타냅니다.

```
beegfs_service: storage
```

2. 이 BeeGFS 서비스에만 적용해야 하는 구성을 정의합니다. 최소한 원하는 TCP 및 UDP 포트를 지정해야 하지만 에서 지원되는 구성 매개 변수는 모두 지정해야 합니다 `beegfs-storage.conf` 또한 포함될 수 있습니다. 참고

다음 매개 변수는 자동으로/다른 위치에 구성되며 여기에서 지정하면 안 됩니다. `sysMgmtdHost`, `storeStorageDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, 및 `connNetFilterFile`.

```
beegfs_ha_beegfs_storage_conf_resource_group_options:
  connStoragePortTCP: <TCP PORT>
  connStoragePortUDP: <UDP PORT>
  tuneBindToNumaZone: <NUMA ZONE> # Recommended if using file nodes with
multiple CPU sockets.
```

3. 다른 서비스 및 클라이언트가 이 서비스에 연결하는 데 사용할 하나 이상의 부동 IP를 구성합니다. 이렇게 하면 BeeGFS가 자동으로 설정됩니다 `connInterfacesFile` 옵션):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.1/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. 선택적으로, 나가는 통신에 사용할 수 있는 하나 이상의 허용된 IP 서브넷을 지정합니다(이 경우 BeeGFS가 자동으로 설정됩니다) `connNetFilterFile` 옵션):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. 다음 지침에 따라 이 서비스가 데이터를 저장할 BeeGFS 스토리지 타겟을 지정합니다. 이 경우에도 가 자동으로 구성됩니다 `storeStorageDirectory` 옵션):

   a. 여러 BeeGFS 서비스/타겟에 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용할 수 있으므로 동일한 스토리지 풀 또는 볼륨 그룹 이름을 사용하기만 하면 됩니다 `name`, `raid_level`, `criteria_*`, 및 `common_*` 각 서비스에 대한 구성(각 서비스에 대해 나열된 볼륨은 서로 달라야 함)

   b. 볼륨 크기는 스토리지 풀/볼륨 그룹의 백분율로 지정해야 하며, 특정 스토리지 풀/볼륨 그룹을 사용하는 모든 서비스/볼륨에서 합계가 100을 초과해서는 안 됩니다. 참고 SSD를 사용할 경우 SSD 성능 및 마모 수명을 최대화하기 위해 볼륨 그룹에 여유 공간을 두는 것이 좋습니다("여기"자세한 내용을 보려면 클릭).

   c. 을 클릭합니다 "여기" 에서 사용할 수 있는 전체 구성 옵션 목록을 확인하십시오 `eseries_storage_pool_configuration`. 과 같은 일부 옵션을 확인합니다 `state`, `host`, `host_type`, `workload_name`, 및 `workload_metadata` 및 볼륨 이름은 자동으로 생성되며 여기에서 지정할 수 없습니다.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_s1_s2
        raid_level: <LEVEL> # One of: raid1, raid5, raid6,
raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
        # Multiple storage targets are supported / typical:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

을 클릭합니다 "여기" BeeGFS 스토리지 서비스를 나타내는 전체 인벤토리 파일의 예

BeeGFS 서비스를 파일 노드에 매핑합니다

를 사용하여 각 BeeGFS 서비스를 실행할 수 있는 파일 노드를 지정합니다 `inventory.yml` 파일.

개요

이 섹션에서는 을 생성하는 방법을 안내합니다 `inventory.yml` 파일. 여기에는 모든 블록 노드를 나열하고 각 BeeGFS 서비스를 실행할 수 있는 파일 노드를 지정하는 작업이 포함됩니다.

단계

파일을 만듭니다 `inventory.yml` 다음과 같이 채웁니다.

1. 파일 상단에서 표준 Ansible 인벤토리 구조를 생성합니다.

```
# BeeGFS HA (High_Availability) cluster inventory.
all:
  children:
```

2. 이 HA 클러스터에 참여하는 모든 블록 노드를 포함하는 그룹을 생성합니다.

```
    # Ansible group representing all block nodes:
    eseries_storage_systems:
      hosts:
        <BLOCK NODE HOSTNAME>:
        <BLOCK NODE HOSTNAME>:
        # Additional block nodes as needed.
```

3. 클러스터의 모든 BeeGFS 서비스를 포함할 그룹과 해당 서비스를 실행할 파일 노드를 생성합니다.

```
    # Ansible group representing all file nodes:
    ha_cluster:
      children:
```

4. 클러스터의 각 BeeGFS 서비스에 대해 해당 서비스를 실행해야 하는 기본 파일 노드 및 보조 파일 노드를 정의합니다.

```
        <SERVICE>: # Ex. "mgmt", "meta_01", or "stor_01".
          hosts:
            <FILE NODE HOSTNAME>:
            <FILE NODE HOSTNAME>:
            # Additional file nodes as needed.
```

을 클릭합니다 "여기" 전체 재고 파일의 예

BeeGFS 파일 시스템을 구축합니다

Ansible 플레이북 개요

Ansible을 사용하여 BeeGFS HA 클러스터 구축 및 관리

개요

이전 섹션에서는 BeeGFS HA 클러스터를 나타내는 Ansible 재고를 구축하는 데 필요한 단계를 안내했습니다. 이 섹션에서는 클러스터를 구축 및 관리하기 위해 NetApp에서 개발한 Ansible 자동화를 소개합니다.

Ansible: 주요 개념

진행하기 전에 Ansible의 몇 가지 주요 개념을 숙지하는 것이 좋습니다.

- Ansible 인벤토리에 대해 실행할 작업은 * 플레이북 * 으로 알려진 내용에 정의됩니다.
  - Ansible에서 수행하는 대부분의 작업은 * idemtiple * 으로 설계되어 여러 번 실행할 수 있으므로, 원하는 구성/상태가 중단 없이 적용되었는지, 불필요한 업데이트를 하지 않고 그대로 적용되는지를 확인할 수 있습니다.
- Ansible에서 실행할 수 있는 가장 작은 단위는 * 모듈 * 입니다.

    - 일반적인 플레이북에서는 여러 모듈을 사용합니다.
        - 예: 패키지 다운로드, 구성 파일 업데이트, 서비스 시작/활성화
    - NetApp은 모듈을 분산하여 NetApp E-Series 시스템을 자동화합니다.
- 복잡한 자동화는 하나의 역할로 더 잘 패키징됩니다.
    - 기본적으로 재사용 가능한 플레이북을 배포하기 위한 표준 형식입니다.
    - NetApp은 Linux 호스트 및 BeeGFS 파일 시스템에 대한 역할을 분산합니다.

Ansible용 BeeGFS HA 역할: 주요 개념

NetApp에서 각 버전의 BeeGFS를 구축 및 관리하는 데 필요한 모든 자동화 기능이 Ansible 역할로 패키지되어 의 일부로 배포됩니다 "BeeGFS용 NetApp E-Series Ansible 컬렉션":

- 이 역할은 * 설치 프로그램 * 과 BeeGFS용 최신 * 구축/관리 * 엔진 사이의 어딘가에 있다고 생각할 수 있습니다.
    - 최신 인프라를 코드 사례 및 철학으로 적용하여 모든 규모의 스토리지 인프라 관리를 단순화합니다.
    - "구베기도"사용자가 스케일아웃 컴퓨팅 인프라를 위한 전체 Kubernetes 배포를 배포/유지 관리할 수 있는 프로젝트와 비슷합니다.
- NetApp에서 NetApp 솔루션에서 BeeGFS를 패키징, 배포 및 유지 관리하는 데 사용하는 * 소프트웨어 정의 * 형식입니다.
    - 전체 Linux 배포판이나 큰 이미지를 배포할 필요 없이 "어플라이언스와 유사한" 환경을 조성하기 위해 노력합니다.
    - 지능형 Pacemaker/BeeGFS 통합을 제공하는 맞춤형 BeeGFS 타겟, IP 주소, 모니터링을 위해 NetApp에서 작성한 OCF(Open Cluster Framework) 규격 클러스터 리소스 에이전트가 포함되어 있습니다.
- 이 역할은 단순히 "자동화"를 구축하는 것이 아니라 다음을 포함한 전체 파일 시스템 수명주기를 관리하는 데 사용됩니다.
    - 서비스별 또는 클러스터 전체 구성 변경 및 업데이트 적용
    - 하드웨어 문제가 해결된 후 클러스터 복구 및 복구 자동화
    - BeeGFS 및 NetApp 볼륨을 사용한 광범위한 테스트 결과를 기준으로 설정된 기본값으로 성능 조정을 단순화합니다.
    - 구성 드리프트 확인 및 수정

NetApp은 또한 Ansible 역할을 제공합니다 "BeeGFS 클라이언트"필요에 따라 BeeGFS를 설치하고 파일 시스템을 컴퓨팅/GPU/로그인 노드에 마운트하는 데 사용할 수 있습니다.

BeeGFS HA 클러스터를 구축합니다

Playbook을 사용하여 BeeGFS HA 클러스터를 구축하기 위해 실행해야 할 작업을 지정합니다.

개요

이 섹션에서는 NetApp에서 BeeGFS를 구축/관리하는 데 사용되는 표준 플레이북을 취합하는 방법에 대해 설명합니다.

단계

파일을 만듭니다 `playbook.yml` 다음과 같이 채웁니다.

1. 먼저 작업 집합(일반적으로 라고 함)을 정의합니다 "재생") NetApp E-Series 블록 노드에서만 실행되어야 합니다. 일시 중지 작업을 사용하여 설치를 실행하기 전에 메시지를 표시한 다음(우발적인 플레이북 실행을 피하기 위해) 을(를) 가져옵니다 `nar_santricity_management` 역할. 이 역할은 에 정의된 모든 일반 시스템 구성을 적용하는 작업을 처리합니다 `group_vars/eseries_storage_systems.yml` 있습니다 `host_vars/<BLOCK NODE>.yml` 파일.

```
- hosts: eseries_storage_systems
  gather_facts: false
  collections:
    - netapp_eseries.santricity
  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: Configure NetApp E-Series block nodes.
      import_role:
        name: nar_santricity_management
```

2. 모든 파일 및 블록 노드에 대해 실행할 플레이를 정의합니다.

```
- hosts: all
  any_errors_fatal: true
  gather_facts: false
  collections:
    - netapp_eseries.beegfs
```

3. 이 플레이에서는 HA 클러스터를 구축하기 전에 실행해야 하는 "사전 작업" 세트를 선택적으로 정의할 수 있습니다. Python과 같은 필수 구성 요소를 확인/설치하는 데 유용할 수 있습니다. 제공된 Ansible 태그가 지원되는지 확인하는 등 비행 전 점검을 삽입할 수도 있습니다.

```
  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 }}"
```

4. 마지막으로, 이 플레이는 구축할 BeeGFS 버전에 대한 BeeGFS HA 역할을 가져옵니다.

```
tasks:
  - name: Verify the BeeGFS HA cluster is properly deployed.
    import_role:
      name: beegfs_ha_7_4 # Alternatively specify: beegfs_ha_7_3.
```

지원되는 각 주요 버전 BeeGFS에 대해 BeeGFS HA 역할이 유지됩니다. 따라서 사용자는 주/부 버전을 업그레이드할 시기를 선택할 수 있습니다. 현재 BeeGFS 7.3.x(`beegfs_7_3`) 또는 BeeGFS 7.2.(`beegfs_7_2`x)가 지원됩니다. 기본적으로 두 역할 모두 릴리즈 시점에 최신 BeeGFS 패치 버전을 구축합니다. 하지만 사용자가 원할 경우 이를 무시하고 최신 패치를 배포할 수 있습니다. "업그레이드 가이드"자세한 내용은 최신 을 참조하십시오.

5. 선택 사항: 추가 작업을 정의하려면 작업이 에 지정되어야 하는지 여부를 염두에 두십시오 `all` 호스트(E-Series 스토리지 시스템 포함) 또는 파일 노드만 포함됩니다. 필요한 경우 를 사용하여 파일 노드를 대상으로 하는 새로운 플레이를 정의합니다 `- hosts: ha_cluster`.

을 클릭합니다 "여기" 전체 플레이북 파일의 예

NetApp Ansible Collections를 설치합니다

Ansible용 BeeGFS 컬렉션 및 모든 종속 항목이 에 유지됩니다 "Ansible 갤럭시". Ansible 제어 노드에서 다음 명령을 실행하여 최신 버전을 설치합니다.

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

일반적으로 권장하지는 않지만 컬렉션의 특정 버전을 설치할 수도 있습니다.

```
ansible-galaxy collection install netapp_eseries.beegfs:
==<MAJOR>.<MINOR>.<PATCH>
```

Playbook을 실행합니다

이 포함된 Ansible 제어 노드의 디렉토리에서 `inventory.yml` 및 `playbook.yml` 파일을 실행하고 다음과 같이 플레이북을 실행합니다.

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

클러스터의 크기에 따라 초기 구축에는 20분 이상 걸릴 수 있습니다. 어떠한 이유로든 구축에 실패하는 경우, 잘못된 케이블 연결, 노드 시작 등 문제를 해결하고 Ansible 플레이북을 다시 시작하십시오.

를 지정할 때 "공통 파일 노드 구성"Ansible에서 연결 기반 인증을 자동으로 관리하도록 기본 옵션을 선택한 경우 `connAuthFile` 공유 암호로 사용되는 을 `<playbook_dir>/files/beegfs/<sysMgmtdHost>_connAuthFile` (기본적으로) 에서 찾을 수 있습니다. 파일 시스템에 액세스해야 하는 모든 클라이언트는 이 공유 암호를 사용해야 합니다. 이 작업은 클라이언트를 를 사용하여 구성한 경우 자동으로 "BeeGFS 클라이언트 역할입니다"처리됩니다.

BeeGFS 클라이언트를 구축합니다

선택적으로 Ansible을 사용하여 BeeGFS 클라이언트를 구성하고 파일 시스템을 마운트할 수 있습니다.

개요

BeeGFS 파일 시스템을 액세스하려면 파일 시스템을 마운트해야 하는 각 노드에서 BeeGFS 클라이언트를 설치 및 구성해야 합니다. 이 섹션에서는 사용 가능한 를 사용하여 이러한 작업을 수행하는 방법을 설명합니다 "Ansible 역할".

단계

클라이언트 인벤토리 파일을 생성합니다

1. 필요한 경우, Ansible 제어 노드에서 BeeGFS 클라이언트로 구성하려는 각 호스트에 대해 암호 없는 SSH를 설정합니다.

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

2. 아래에서 `host_vars/`에서 각 `BeeGFS` 클라이언트에 대한 파일을 생성합니다 `<HOSTNAME>.yml` 다음 콘텐츠를 사용하여 환경에 맞는 올바른 정보로 자리 표시자 텍스트를 입력합니다.

```
# BeeGFS Client
ansible_host: <MANAGEMENT_IP>
```

3. 선택적으로 NetApp E-Series 호스트 컬렉션의 역할을 사용하여 클라이언트가 BeeGFS 파일 노드에 연결할 수 있도록 InfiniBand 또는 이더넷 인터페이스를 구성하려면 다음 중 하나를 포함합니다.

   a. 네트워크 유형이 인 경우 "InfiniBand(IPoIB 사용)":

```
eseries_ipoib_interfaces:
- name: <INTERFACE>  # Example: ib0 or i1b
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   b. 네트워크 유형이 인 경우 "RoCE(RDMA over Converged Ethernet)":

```
eseries_roce_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

c. 네트워크 유형이 인 경우 "이더넷(TCP 전용, RDMA 없음)":

```
eseries_ip_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

4. 새 파일을 만듭니다 `client_inventory.yml` 그리고 Ansible이 각 클라이언트에 연결하는 데 사용해야 하는 사용자 지정과 Ansible이 권한 에스컬레이션을 위해 사용해야 하는 암호(이 경우 필요)를 지정합니다 `ansible_ssh_user` 루트 또는 sudo 권한 보유):

```
# BeeGFS client inventory.
all:
  vars:
    ansible_ssh_user: <USER>
    ansible_become_password: <PASSWORD>
```

암호를 일반 텍스트로 저장하지 마십시오. 대신 Ansible Vault를 사용하십시오(참조 "Ansible 설명서" Ansible Vault로 콘텐츠 암호화)를 사용하거나 을 사용합니다 `--ask-become-pass` 옵션을 클릭합니다.

5. 에 있습니다 `client_inventory.yml` File(파일): 에 BeeGFS 클라이언트로 구성해야 하는 모든 호스트를 나열합니다 `beegfs_clients` 그룹화한 다음 인라인 주석을 참조하여 시스템에서 BeeGFS 클라이언트 커널 모듈을 구축하는 데 필요한 추가 구성을 제거합니다.

```
children:
    # Ansible group representing all BeeGFS clients:
    beegfs_clients:
      hosts:
        <CLIENT HOSTNAME>:
        # Additional clients as needed.

      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.
```

NVIDIA OFED 드라이버를 사용하는 경우 beegfs_client_OFED_include_path가 Linux 설치에 대한 올바른 "헤더 포함 경로"를 가리키는지 확인하십시오. 자세한 내용은 의 BeeGFS 설명서를 "RDMA 지원"참조하십시오.

6. 에 있습니다 `client_inventory.yml` 파일, 이전에 정의한 모든 파일 아래에 마운트할 BeeGFS 파일 시스템을 나열합니다 `vars`:

```
        beegfs_client_mounts:
          - sysMgmtdHost: <IP ADDRESS>  # 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
        # Specify additional file system mounts for this or other file
systems.
```

7. BeeGFS 7.2.7 및 7.3.1부터 "연결 인증"구성하거나 명시적으로 사용하지 않도록 설정해야 합니다. 를 지정할 때 연결 기반 인증을 구성하는 방법에 따라 "공통 파일 노드 구성"클라이언트 구성을 조정해야 할 수도 있습니다.

   a. 기본적으로 HA 클러스터 배포는 연결 인증을 자동으로 구성하고 를 생성합니다 `connauthfile` 이 정보는 에서 Ansible 제어 노드에 배치/유지됩니다 `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile`. 기본적으로 BeeGFS 클라이언트 역할은 에 정의된 클라이언트에 이 파일을 읽고 배포하도록 설정되어 있습니다 `client_inventory.yml`추가 조치가 필요하지 않습니다.

      i. 고급 옵션은 에 포함된 기본값 전체 목록을 참조하십시오 "BeeGFS 클라이언트 역할입니다".

   b. 을 사용하여 사용자 지정 암호를 지정하도록 선택한 경우 `beegfs_ha_conn_auth_secret` 에서 지정합니다 `client_inventory.yml` 파일 또한:

```
beegfs_ha_conn_auth_secret: <SECRET>
```

   c. 을 사용하여 연결 기반 인증을 완전히 사용하지 않도록 선택하는 경우 `beegfs_ha_conn_auth_enabled`에서 를 지정합니다 `client_inventory.yml` 파일 또한:

```
beegfs_ha_conn_auth_enabled: false
```

지원되는 매개 변수의 전체 목록과 추가 세부 정보는 를 참조하십시오 "전체 BeeGFS 클라이언트 문서". 클라이언트 인벤토리의 전체 예제를 보려면 을 클릭합니다 "여기".

BeeGFS Client Playbook File을 생성합니다

1. 새 파일을 만듭니다 `client_playbook.yml`

```
# BeeGFS client playbook.
- hosts: beegfs_clients
  any_errors_fatal: true
  gather_facts: true
  collections:
    - netapp_eseries.beegfs
    - netapp_eseries.host
  tasks:
```

2. 선택 사항: NetApp E-Series Host Collection의 역할을 사용하여 클라이언트가 BeeGFS 파일 시스템에 연결할 수 있도록 인터페이스를 구성하려면 구성 중인 인터페이스 유형에 해당하는 역할을 가져옵니다.

   a. InfiniBand(IPoIB)를 사용하는 경우:

```
    - name: Ensure IPoIB is configured
      import_role:
        name: ipoib
```

   b. RoCE(RDMA over Converged Ethernet)를 사용 중인 경우:

```
    - name: Ensure IPoIB is configured
      import_role:
        name: roce
```

   c. 를 사용 중인 경우 이더넷(TCP 전용, RDMA 없음)을 사용합니다.

```
    - name: Ensure IPoIB is configured
      import_role:
        name: ip
```

3. 마지막으로 BeeGFS 클라이언트 역할을 가져와 클라이언트 소프트웨어를 설치하고 파일 시스템 마운트를 설정합니다.

```
    # REQUIRED: Install the BeeGFS client and mount the BeeGFS file
system.
    - name: Verify the BeeGFS clients are configured.
      import_role:
        name: beegfs_client
```

클라이언트 플레이북의 전체 예제를 보려면 을 클릭합니다 "여기".

BeeGFS Client Playbook을 실행합니다

클라이언트를 설치/구축하고 BeeGFS를 마운트하려면 다음 명령을 실행합니다.

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

BeeGFS 구축을 확인합니다

시스템을 운영 환경에 배치하기 전에 파일 시스템 구축을 확인하십시오.

개요

BeeGFS 파일 시스템을 운영 환경에 배치하기 전에 몇 가지 검증 검사를 수행하십시오.

단계

1. 모든 클라이언트에 로그인하고 다음을 실행하여 모든 예상 노드가 존재하고 연결 가능한지, 불일치 또는 보고된 다른 문제가 없는지 확인합니다.

```
beegfs-fsck --checkfs
```

2. 전체 클러스터를 종료한 다음 재시작합니다. 모든 파일 노드에서 다음을 실행합니다.

```
pcs cluster stop --all # Stop the cluster on all file nodes.
pcs cluster start --all  # Start the cluster on all file nodes.
pcs status # Verify all nodes and services are started and no failures
are reported (the command may need to be reran a few times to allow time
for all services to start).
```

3. 각 노드를 스탠바이에 배치하고 BeeGFS 서비스가 보조 노드로 페일오버할 수 있는지 확인합니다. 이 작업을 수행하려면 파일 노드에 로그인하고 다음을 실행합니다.

```
pcs status # Verify the cluster is healthy at the start.
pcs node standby <FILE NODE HOSTNAME> # Place the node under test in
standby.
pcs status # Verify services are started on a secondary node and no
failures are reported.
pcs node unstandby <FILE NODE HOSTNAME> # Take the node under test out
of standby.
pcs status # Verify the file node is back online and no failures are
reported.
pcs resource relocate run # Move all services back to their preferred
nodes.
pcs status # Verify services have moved back to the preferred node.
```

4. IOR 및 MDTest와 같은 성능 벤치마킹 툴을 사용하여 파일 시스템 성능이 기대에 부합하는지 확인합니다. BeeGFS와 함께 사용되는 일반 테스트 및 매개 변수의 예는 "설계 Verification"NetApp 검증 아키텍처의 BeeGFS 섹션에서 찾을 수 있습니다.

특정 현장/설치에 대해 정의된 수용 기준에 따라 추가 테스트를 수행해야 합니다.

기능 및 통합 배포

BeeGFS CSI 드라이버

BeeGFS v8용 TLS 암호화 구성

BeeGFS v8 관리 서비스와 클라이언트 간의 통신을 보호하기 위해 TLS 암호화를 구성하십시오.

개요

BeeGFS v8은 관리 도구(예: `beegfs` 명령줄 유틸리티)와 Management 또는 Remote와 같은 BeeGFS 서버 서비스 간의 네트워크 통신을 암호화하기 위한 TLS 지원을 도입했습니다. 이 가이드에서는 세 가지 TLS 구성 방법을 사용하여 BeeGFS 클러스터에서 TLS 암호화를 구성하는 방법을 설명합니다.

- **신뢰할 수 있는 인증 기관 사용:** BeeGFS 클러스터에서 기존 CA 서명 인증서를 사용하십시오.
- **로컬 인증 기관 생성:** 로컬 인증 기관을 생성하고 이를 사용하여 BeeGFS 서비스용 인증서에 서명합니다. 이 방법은 외부 CA에 의존하지 않고 자체 신뢰 체인을 관리하려는 환경에 적합합니다.
- **TLS 비활성화:** 암호화가 필요하지 않은 환경이나 문제 해결을 위해 TLS를 완전히 비활성화할 수 있습니다. 하지만 내부 파일 시스템 구조 및 구성에 대한 잠재적으로 민감한 정보가 평문으로 노출될 수 있으므로 권장하지 않습니다.

귀사의 환경 및 조직 정책에 가장 적합한 방법을 선택하십시오. 자세한 내용은 "BeeGFS TLS" 문서를 참조하십시오.

> `beegfs-client` 서비스를 실행하는 머신은 BeeGFS 파일 시스템을 마운트하기 위해 TLS가 필요하지 않습니다. BeeGFS CLI 및 원격, 동기화와 같은 다른 beegfs 서비스를 사용하려면 TLS를 설정해야 합니다.

신뢰할 수 있는 인증 기관 사용

내부 기업 CA 또는 타사 공급업체에서 발급한 신뢰할 수 있는 인증 기관(CA)의 인증서에 액세스할 수 있는 경우 BeeGFS v8이 자체 서명 인증서를 생성하는 대신 이러한 CA 서명 인증서를 사용하도록 구성할 수 있습니다.

새로운 **BeeGFS v8** 클러스터 배포

새로운 BeeGFS v8 클러스터 배포를 위해 Ansible 인벤토리의 `user_defined_params.yml` 파일에서 CA 서명 인증서를 참조하도록 구성하십시오.

```
beegfs_ha_tls_enabled: true

beegfs_ha_ca_cert_src_path: files/beegfs/cert/ca_cert.pem

beegfs_ha_tls_cert_src_path: files/beegfs/cert/mgmtd_tls_cert.pem

beegfs_ha_tls_key_src_path: files/beegfs/cert/mgmtd_tls_key.pem
```

`beegfs_ha_tls_config_options.alt_names`이 비어 있지 않으면 Ansible은 제공된 `alt_names`를 인증서의 SAN(Subject Alternative Names)으로 사용하여 자체 서명된 TLS 인증서와 키를 자동으로 생성합니다. 사용자 지정 TLS 인증서와 키(`beegfs_ha_tls_cert_src_path` 및 `beegfs_ha_tls_key_src_path`에 지정된 대로)를 사용하려면 `beegfs_ha_tls_config_options` 섹션 전체를 주석 처리하거나 삭제해야 합니다. 그렇지 않으면 자체 서명된 인증서 생성이 우선시되어 사용자 지정 인증서와 키가 사용되지 않습니다.

기존 **BeeGFS v8** 클러스터 구성

기존 BeeGFS v8 클러스터의 경우 BeeGFS 관리 서비스의 구성 파일에서 파일 노드의 CA 서명 인증서 경로를 설정하십시오.

```
tls-cert-file = /path/to/cert.pem
tls-key-file = /path/to/key.pem
```

CA 서명 인증서를 사용하여 **BeeGFS v8** 클라이언트 구성

BeeGFS v8 클라이언트가 시스템 인증서 풀을 사용하여 CA 서명 인증서를 신뢰하도록 구성하려면 각 클라이언트 구성에서 tls-cert-file = ""로 설정하십시오. 시스템 인증서 풀을 사용하지 않는 경우 tls-cert-file = <local cert>로 설정하여 로컬 인증서의 경로를 제공하십시오. 이 설정을 통해 클라이언트는 BeeGFS 관리 서비스에서 제공하는 인증서를 인증할 수 있습니다.

로컬 인증 기관 생성

조직에서 BeeGFS 클러스터용 자체 인증서 인프라를 구축하려면 로컬 인증 기관(CA)을 생성하여 BeeGFS 클러스터용 인증서를 발급하고 서명할 수 있습니다. 이 방법은 BeeGFS 관리 서비스용 인증서에 서명하는 CA를 생성한 다음, 해당 인증서를 클라이언트에 배포하여 신뢰 체인을 구축하는 방식입니다. 다음 지침에 따라 로컬 CA를 설정하고 기존 또는 신규 BeeGFS v8 클러스터에 인증서를 배포하십시오.

새로운 **BeeGFS v8** 클러스터 배포

새로운 BeeGFS v8 배포의 경우, `beegfs_8` Ansible 역할은 제어 노드에 로컬 CA를 생성하고 관리 서비스에 필요한 인증서를 생성합니다. 이 기능은 Ansible 인벤토리의 `user_defined_params.yml` 파일에 다음 매개변수를 설정하여 활성화할 수 있습니다:

```
beegfs_ha_tls_enabled: true

beegfs_ha_ca_cert_src_path: files/beegfs/cert/local_ca_cert.pem

beegfs_ha_tls_cert_src_path: files/beegfs/cert/mgmtd_tls_cert.pem

beegfs_ha_tls_key_src_path: files/beegfs/cert/mgmtd_tls_key.pem

beegfs_ha_tls_config_options:
  alt_names: [<mgmt_service_ip>]
```

`beegfs_ha_tls_config_options.alt_names`가 제공되지 않으면 Ansible은 지정된 인증서/키 경로에 있는 기존 인증서를 사용하려고 시도합니다.

기존 **BeeGFS v8** 클러스터 구성

기존 BeeGFS 클러스터의 경우 로컬 인증 기관을 생성하고 관리 서비스에 필요한 인증서를 생성하여 TLS를 통합할 수 있습니다. BeeGFS 관리 서비스의 구성 파일에서 경로를 업데이트하여 새로 생성된 인증서를 가리키도록 하십시오.

이 섹션의 지침은 참고용으로 사용됩니다. 개인 키와 인증서를 다룰 때는 적절한 보안 예방 조치를 취해야 합니다.

인증 기관 생성

신뢰할 수 있는 시스템에서 로컬 인증 기관(CA)을 생성하여 BeeGFS 관리 서비스용 인증서를 서명하십시오. 생성된 CA 인증서는 클라이언트에 배포되어 신뢰를 구축하고 BeeGFS 서비스와의 안전한 통신을 가능하게 합니다.

다음 지침은 RHEL 기반 시스템에서 로컬 Certificate Authority를 생성하기 위한 참조 자료입니다.

1. OpenSSL이 아직 설치되어 있지 않은 경우 설치하십시오.

```
dnf install openssl
```

2. 인증서 파일을 저장할 작업 디렉터리를 생성합니다:

```
mkdir -p ~/beegfs_tls && cd ~/beegfs_tls
```

3. CA 개인 키를 생성합니다.

```
openssl genrsa -out ca_key.pem 4096
```

4. `ca.cnf`라는 이름의 CA 구성 파일을 생성하고 고유 이름 필드를 조직에 맞게 조정하십시오.

```
[ req ]
default_bits        = 4096
distinguished_name  = req_distinguished_name
x509_extensions     = v3_ca
prompt              = no

[ req_distinguished_name ]
C  = <Country>
ST = <State>
L  = <City>
O  = <Organization>
OU = <OrganizationalUnit>
CN = BeeGFS-CA

[ v3_ca ]
basicConstraints = critical,CA:TRUE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
```

5. CA 인증서를 생성하십시오. 이 인증서는 시스템 수명 동안 유효해야 하며, 그렇지 않으면 만료 전에 인증서를 다시 생성해야 합니다. 인증서가 만료되면 일부 구성 요소 간의 통신이 불가능해지며, TLS 인증서를 업데이트하려면 일반적으로 서비스를 재시작해야 합니다.

   다음 명령은 1년 동안 유효한 CA 인증서를 생성합니다:

```
openssl req -new -x509 -key ca_key.pem -out ca_cert.pem -days 365
-config ca.cnf
```

   이 예시에서는 간단하게 하기 위해 1년 유효 기간을 사용하지만, 조직의 보안 요구 사항에 따라 `-days` 매개변수를 조정하고 인증서 갱신 프로세스를 수립해야 합니다.

관리 서비스 인증서 생성

BeeGFS 관리 서비스용 인증서를 생성하고 생성한 CA로 서명하십시오. 이러한 인증서는 BeeGFS 관리 서비스가 실행되는 파일 노드에 설치됩니다.

1. 관리 서비스 개인 키를 생성합니다:

```
openssl genrsa -out mgmtd_tls_key.pem 4096
```

2. 모든 관리 서비스 IP 주소에 대한 주체 대체 이름(SAN)을 포함하는 `tls_san.cnf`이라는 이름의 인증서 구성 파일을 생성합니다:

```
[ req ]
default_bits        = 4096
distinguished_name  = req_distinguished_name
req_extensions      = req_ext
prompt              = no

[ req_distinguished_name ]
C  = <Country>
ST = <State>
L  = <City>
O  = <Organization>
OU = <OrganizationalUnit>
CN = beegfs-mgmt

[ req_ext ]
subjectAltName = @alt_names

[ v3_ca ]
subjectAltName = @alt_names
basicConstraints = CA:FALSE

[ alt_names ]
IP.1 = <beegfs_mgmt_service_ip_1>
IP.2 = <beegfs_mgmt_service_ip_2>
```

고유 이름 필드를 CA 구성에 맞게 업데이트하고 `IP.1` 및 `IP.2` 값을 관리 서비스 IP 주소로 업데이트하십시오.

3. 인증서 서명 요청(CSR) 생성:

```
openssl req -new -key mgmtd_tls_key.pem -out mgmtd_tls_csr.pem -config
tls_san.cnf
```

4. CA로 인증서에 서명(1년간 유효):

```
openssl x509 -req -in mgmtd_tls_csr.pem -CA ca_cert.pem -CAkey
ca_key.pem -CAcreateserial -out mgmtd_tls_cert.pem -days 365 -sha256
-extensions v3_ca -extfile tls_san.cnf
```

조직의 보안 정책에 따라 인증서 유효 기간(`-days 365`을 조정하십시오. 많은 조직에서는 1~2년마다 인증서를 갱신하도록 요구합니다.

5. 인증서가 올바르게 생성되었는지 확인합니다.

```
openssl x509 -in mgmtd_tls_cert.pem -text -noout
```

주체 대체 이름 섹션에 모든 관리 IP 주소가 포함되어 있는지 확인하십시오.

파일 노드에 인증서 배포

CA 인증서와 관리 서비스 인증서를 해당 파일 노드 및 클라이언트에 배포합니다.

1. CA 인증서와 관리 서비스 인증서 및 키를 관리 서비스를 실행하는 파일 노드에 복사합니다.

```
scp ca_cert.pem mgmtd_tls_cert.pem mgmtd_tls_key.pem
user@beegfs_01:/etc/beegfs/
scp ca_cert.pem mgmtd_tls_cert.pem mgmtd_tls_key.pem
user@beegfs_02:/etc/beegfs/
```

관리 서비스가 **TLS** 인증서를 가리키도록 설정

TLS를 활성화하고 생성된 TLS 인증서를 참조하도록 BeeGFS 관리 서비스 구성을 업데이트하십시오.

1. BeeGFS 관리 서비스가 실행 중인 파일 노드에서 관리 서비스 구성 파일을 편집합니다(예: `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml`). 다음 TLS 관련 매개변수를 추가하거나 업데이트합니다:

```
tls-disable = false
tls-cert-file = "/etc/beegfs/mgmtd_tls_cert.pem"
tls-key-file = "/etc/beegfs/mgmtd_tls_key.pem"
```

2. 변경 사항을 적용하려면 BeeGFS 관리 서비스를 안전하게 다시 시작하는 적절한 조치를 취하십시오:

```
systemctl restart beegfs-mgmtd
```

3. 관리 서비스가 성공적으로 시작되었는지 확인합니다.

```
journalctl -xeu beegfs-mgmtd
```

성공적인 TLS 초기화 및 인증서 로딩을 나타내는 로그 항목을 찾으십시오.

```
Successfully initialized certificate verification library.
Successfully loaded license certificate: TMP-XXXXXXXXXX
```

BeeGFS v8 클라이언트에 대한 TLS 구성

BeeGFS 관리 서비스와의 통신이 필요한 모든 BeeGFS 클라이언트에 로컬 CA에서 서명한 인증서를 생성하고 배포합니다.

1. 위의 관리 서비스 인증서와 동일한 프로세스를 사용하여 클라이언트용 인증서를 생성하되, 주체 대체 이름(SAN) 필드에 클라이언트의 IP 주소 또는 호스트 이름을 입력합니다.
2. 클라이언트의 인증서를 안전한 원격 복사 방식으로 클라이언트에 복사하고 클라이언트에서 인증서 이름을 `cert.pem`로 변경합니다.

```
scp client_cert.pem user@client:/etc/beegfs/cert.pem
```

3. 모든 클라이언트에서 BeeGFS 클라이언트 서비스를 다시 시작하십시오.

```
systemctl restart beegfs-client
```

4. `beegfs CLI` 명령어를 실행하여 클라이언트 연결을 확인하십시오. 예:

```
beegfs health check
```

TLS 비활성화

TLS는 문제 해결을 위해 또는 사용자가 원할 경우 비활성화할 수 있습니다. 이는 내부 파일 시스템 구조 및 구성에 대한 잠재적으로 민감한 정보가 평문 형태로 노출될 수 있으므로 권장하지 않습니다. 기존 또는 새 BeeGFS v8 클러스터에서 TLS를 비활성화하려면 다음 지침을 따르십시오.

새로운 BeeGFS v8 클러스터 배포

새로운 BeeGFS 클러스터를 배포할 때, Ansible 인벤토리의 `user_defined_params.yml` 파일에 다음 매개변수를 설정하여 TLS를 비활성화한 상태로 클러스터를 배포할 수 있습니다:

```
beegfs_ha_tls_enabled: false
```

기존 BeeGFS v8 클러스터 구성

기존 BeeGFS v8 클러스터의 경우 관리 서비스 구성 파일을 편집합니다. 예를 들어, `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml` 경로에 있는 파일을 편집하고 다음과 같이 설정합니다:

```
tls-disable = true
```

변경 사항을 적용하려면 관리 서비스를 안전하게 재시작하기 위한 적절한 조치를 취하십시오.

BeeGFS 클러스터 관리

개요, 주요 개념 및 용어

구축된 BeeGFS HA 클러스터를 관리하는 방법에 대해 알아보십시오.

개요

이 섹션은 구축된 BeeGFS HA 클러스터를 관리해야 하는 클러스터 관리자를 대상으로 합니다. Linux HA 클러스터에 익숙한 사람조차도 이 가이드를 완전히 읽어야 합니다. Ansible을 사용하여 재구성을 수행할 때는 특히 클러스터 관리 방법에 여러 가지 차이점이 있기 때문입니다.

주요 개념

이러한 개념 중 일부는 메인 "용어 및 개념" 페이지에 소개되었지만 BeeGFS HA 클러스터의 컨텍스트에서 다시 소개하면 도움이 됩니다.

- 클러스터 노드:** 심장박동기 및 Corosync 서비스를 실행하고 HA 클러스터에 참여하는 서버.
- 파일 노드:** 하나 이상의 BeeGFS 관리, 메타데이터 또는 스토리지 서비스를 실행하는 데 사용되는 클러스터 노드입니다.
- 블록 노드:** 파일 노드에 블록 스토리지를 제공하는 NetApp E-Series 스토리지 시스템 이러한 노드는 자체 독립형 HA 기능을 제공하므로 BeeGFS HA 클러스터에 참여하지 않습니다. 각 노드는 블록 계층에서 고가용성을 제공하는 2개의 스토리지 컨트롤러로 구성됩니다.
- BeeGFS 서비스:** BeeGFS 관리, 메타데이터 또는 스토리지 서비스 각 파일 노드는 블록 노드의 볼륨을 사용하여 데이터를 저장하는 하나 이상의 서비스를 실행합니다.
- 빌딩 블록:** BeeGFS 파일 노드, E-Series 블록 노드 및 BeeGFS 서비스를 표준화된 방식으로 구축하여 NetApp 검증 아키텍처에 따라 BeeGFS HA 클러스터/파일 시스템을 간편하게 확장할 수 있습니다. 맞춤형 HA 클러스터도 지원되지만, 확장을 단순화하기 위해 유사한 구성 요소 접근 방식을 따르는 경우가 많습니다.
- BeeGFS HA Cluster:** 블록 노드에서 지원하는 BeeGFS 서비스를 실행하는 데 사용되는 확장 가능한 수의 파일 노드를 통해 BeeGFS 데이터를 고가용성 방식으로 저장합니다. 패키징 및 배포를 위해 Ansible을 사용하여 업계에서 검증된 오픈 소스 구성 요소 페이스 메이커 및 Corosync를 기반으로 합니다.
- 클러스터 서비스: ** 클러스터에 참여하는 각 노드에서 실행 중인 심장박동기 및 Corosync 서비스를 나타냅니다. 참고: 노드가 BeeGFS 서비스를 실행하지 않고 단지 두 개의 파일 노드만 필요한 경우 "Tiebreaker" 노드로 클러스터에 참여할 수 있습니다.
- 클러스터 리소스:** 클러스터에서 실행 중인 각 BeeGFS 서비스에 대해 BeeGFS 모니터링 리소스와 BeeGFS 타겟, IP 주소(부동 IP) 및 BeeGFS 서비스 자체에 대한 리소스가 포함된 리소스 그룹이 표시됩니다.

Ansible:** 소프트웨어 프로비저닝, 구성 관리 및 애플리케이션 배포를 위한 도구로 인프라를 코드로 구현할 수 있습니다. BeeGFS HA 클러스터를 패키지화하여 NetApp에서 BeeGFS 구축, 재구성 및 업데이트 프로세스를 간소화합니다.

- PCS:** 클러스터의 파일 노드에서 사용할 수 있는 명령줄 인터페이스입니다. 이 인터페이스는 클러스터의 노드 및 리소스 상태를 쿼리하고 제어하는 데 사용됩니다.

일반 용어

- 장애 조치:** 각 BeeGFS 서비스에는 해당 노드에 장애가 발생하지 않는 한 실행되는 기본 파일 노드가 있습니다. 비기본/보조 파일 노드에서 BeeGFS 서비스를 실행하는 경우 페일오버 중인 것으로 표시됩니다.
- 페일백:** 비기본 파일 노드에서 기본 설정 노드로 BeeGFS 서비스를 이동하는 동작
- HA 쌍:** 동일한 블록 노드 세트에 액세스할 수 있는 두 개의 파일 노드를 HA 쌍이라고도 합니다. 이는 NetApp 전체에서 사용되는 일반적인 용어로 서로 "이어갈" 수 있는 2개의 스토리지 컨트롤러 또는 노드를 참조하는 데 사용됩니다.
- 유지 관리 모드: ** 모든 리소스 모니터링을 비활성화하고 심장박동기 장치가 클러스터의 리소스를 이동하거나 관리하지 못하게 합니다(의 섹션 참조"유지보수 모드").
- HA 클러스터:** 클러스터의 여러 노드 간에 페일오버하여 가용성이 높은 BeeGFS 파일 시스템을 생성할 수 있는 BeeGFS 서비스를 실행하는 하나 이상의 파일 노드. 파일 노드는 클러스터에서 BeeGFS 서비스의 하위 집합을 실행할 수 있는 HA 쌍으로 구성되는 경우가 많습니다.

Ansible과 PCS 도구를 사용해야 하는 경우

HA 클러스터를 관리하기 위해 Ansible과 PCS 명령줄 도구를 사용해야 하는 경우는 언제입니까?

모든 클러스터 배포 및 재구성 작업은 외부 Ansible 제어 노드의 Ansible을 사용하여 완료해야 합니다. 클러스터 상태의 일시적인 변경(예: 대기 모드 내외부로 노드 배치)은 일반적으로 클러스터의 한 노드(성능이 저하되지 않거나 유지 보수를 수행하려고 하지 않는 노드)에 로그인하고 PCS 명령줄 도구를 사용하여 수행됩니다.

리소스, 제약, 속성 및 BeeGFS 서비스 자체를 비롯한 클러스터 구성을 수정하려면 항상 Ansible을 사용해야 합니다. Ansible 인벤토리 및 플레이북의 최신 복사본(소스 제어 기능을 통해 변경 사항을 추적하는 것이 이상적)을 유지하는 것은 클러스터를 유지하는 데 필요한 부분입니다. 구성을 변경해야 하는 경우 인벤토리를 업데이트하고 BeeGFS HA 역할을 가져오는 Ansible 플레이북을 다시 실행합니다.

HA 역할은 클러스터를 유지 관리 모드로 설정한 다음 필요한 변경을 수행한 후 BeeGFS 또는 클러스터 서비스를 다시 시작하여 새 구성을 적용하는 작업을 처리합니다. 전체 노드 재부팅은 일반적으로 초기 구현 이후에 필요하지 않기 때문에 Ansible을 다시 실행하는 것은 일반적으로 "안전한" 절차로 간주되지만, BeeGFS 서비스를 다시 시작해야 할 경우 유지보수 시간이나 근무 시간 외에 실행하는 것이 좋습니다. 이러한 재시작으로 인해 일반적으로 응용 프로그램 오류가 발생하지는 않지만 성능이 저하될 수 있습니다(일부 응용 프로그램이 다른 응용 프로그램보다 더 잘 처리할 수 있음).

또한 Ansible을 다시 실행하는 것은 전체 클러스터를 최적의 상태로 되돌리는 옵션이며, 경우에 따라 PC를 사용할 때보다 클러스터의 상태를 더 쉽게 복구할 수 있습니다. 특히 어떤 이유로 클러스터가 중단된 비상 상황에서는 모든 노드가 다시 실행 중인 Ansible을 사용하면 PC를 사용하는 것보다 신속하고 안정적으로 클러스터를 복구할 수 있습니다.

클러스터의 상태를 검사합니다

PC를 사용하여 클러스터의 상태를 봅니다.

개요

실행 중입니다 `pcs status` 모든 클러스터 노드에서 클러스터의 전체 상태와 각 리소스의 상태(예: BeeGFS 서비스 및 해당 종속성)를 확인하는 가장 쉬운 방법입니다. 이 섹션에서는 의 출력에서 확인할 수 있는 내용을 설명합니다 `pcs`

`status` 명령.

의 출력 이해 `pcs status`

실행 `pcs status` 클러스터 서비스(박동조율기 및 Corosync)가 시작되는 모든 클러스터 노드에서. 출력 맨 위에는 클러스터의 요약이 표시됩니다.

```
[root@beegfs_01 ~]# pcs status
Cluster name: hacluster
Cluster Summary:
  * Stack: corosync
  * Current DC: beegfs_01 (version 2.0.5-9.el8_4.3-ba59be7122) - partition
with quorum
  * Last updated: Fri Jul  1 13:37:18 2022
  * Last change:  Fri Jul  1 13:23:34 2022 by root via cibadmin on
beegfs_01
  * 6 nodes configured
  * 235 resource instances configured
```

아래 섹션에는 클러스터의 노드가 나열됩니다.

```
Node List:
  * Node beegfs_06: standby
  * Online: [ beegfs_01 beegfs_02 beegfs_04 beegfs_05 ]
  * OFFLINE: [ beegfs_03 ]
```

이는 대기 또는 오프라인 상태인 노드를 나타냅니다. 대기 상태의 노드는 여전히 클러스터에 참여하고 있지만 리소스 실행 부적격으로 표시되어 있습니다. 오프라인 상태인 노드는 노드가 수동으로 중지되었거나 노드가 재부팅/종료되었기 때문에 해당 노드에서 클러스터 서비스가 실행되고 있지 않음을 나타냅니다.

노드가 처음 시작될 때 클러스터 서비스가 중지되며, 비정상적인 노드로 리소스가 실수로 페일백되지 않도록 수동으로 시작해야 합니다.

노드가 비관리적 이유(예: 실패)로 인해 대기 또는 오프라인 상태인 경우 노드의 상태 옆에 괄호 안에 추가 텍스트가 표시됩니다. 예를 들어 펜싱을 사용하지 않도록 설정하고 리소스에 장애가 발생하면 표시됩니다 `Node <HOSTNAME>: standby (on-fail)`. 다른 가능한 상태는 입니다 `Node <HOSTNAME>: UNCLEAN (offline)`펜싱이 실패하여 클러스터가 노드 상태를 확인할 수 없음을 나타내는 경우(다른 노드에서 리소스가 시작되는 것을 차단할 수 있음) 잠시 노드가 펜싱되는 것으로 나타나지만 지속합니다.

다음 섹션에는 클러스터의 모든 리소스 목록과 해당 상태가 표시됩니다.

```
Full List of Resources:
  * mgmt-monitor    (ocf::eseries:beegfs-monitor):   Started beegfs_01
  * Resource Group: mgmt-group:
    * mgmt-FS1  (ocf::eseries:beegfs-target):    Started beegfs_01
    * mgmt-IP1  (ocf::eseries:beegfs-ipaddr2):   Started beegfs_01
    * mgmt-IP2  (ocf::eseries:beegfs-ipaddr2):   Started beegfs_01
    * mgmt-service  (systemd:beegfs-mgmtd):  Started beegfs_01
[...]
```

노드와 마찬가지로 리소스에 문제가 있는 경우 괄호 안의 리소스 상태 옆에 추가 텍스트가 표시됩니다. 예를 들어 페이스메이커의 리소스 중지를 요청했으나 할당된 시간 내에 완료되지 못한 경우 심장박동기는 노드를 울타리로 만들려고 시도합니다. 펜싱이 비활성화되거나 펜싱 작업이 실패하는 경우 리소스 상태는 입니다 `FAILED <HOSTNAME> (blocked)` 다른 노드에서 심장박동조율기를 시작할 수 없습니다.

BeeGFS HA 클러스터가 여러 BeeGFS에 최적화된 맞춤형 OCF 리소스 에이전트를 사용하는 것을 주목할 필요가 있습니다. 특히 BeeGFS 모니터는 특정 노드의 BeeGFS 리소스를 사용할 수 없을 때 페일오버를 트리거합니다.

HA 클러스터와 BeeGFS를 재구성합니다

Ansible을 사용하여 클러스터를 재구성합니다.

개요

일반적으로 BeeGFS HA 클러스터의 모든 측면을 재구성하려면 Ansible 인벤토리를 업데이트하고 `ansible-playbook` 명령을 다시 실행해야 합니다. 여기에는 알림 업데이트, 영구 펜싱 구성 변경 또는 BeeGFS 서비스 구성 조정 등이 포함됩니다. 이러한 옵션은 `group_vars/ha_cluster.yml` 파일을 사용하여 조정되며 전체 옵션 목록은 "일반 파일 노드 구성을 지정합니다"섹션에서 확인할 수 있습니다.

유지 관리를 수행하거나 클러스터를 서비스할 때 관리자가 알아야 하는 구성 옵션에 대한 자세한 내용은 아래를 참조하십시오.

펜싱 비활성화 및 활성화 방법

클러스터를 설정할 때 펜싱은 기본적으로 활성화/필요합니다. 경우에 따라 펜싱을 일시적으로 비활성화하여 운영 체제 업그레이드와 같은 특정 유지보수 작업을 수행할 때 노드가 실수로 종료되지 않도록 하는 것이 좋습니다. 이 기능은 수동으로 비활성화할 수 있지만 관리자가 주의해야 할 단점이 있습니다.

옵션 **1: Ansible**을 사용하여 펜싱 비활성화(권장)

Ansible을 사용하여 펜싱을 비활성화하면 BeeGFS 모니터의 장애 발생 시 동작이 "fence"에서 "standby"로 변경됩니다. 즉, BeeGFS 모니터가 장애를 감지하면 노드를 대기 상태로 두고 모든 BeeGFS 서비스를 페일오버하려고 합니다. 활성 상태의 문제 해결/테스트 이외의 경우 일반적으로 옵션 2보다 더 권장됩니다. 단점은 리소스가 원래 노드에서 중지되지 않을 경우 다른 위치에서 시작하는 것이 차단된다는 것입니다(즉, 펜싱은 일반적으로 운영 클러스터에 필요합니다).

1. 에 있는 Ansible 인벤토리에 들어 있습니다 `groups_vars/ha_cluster.yml` 다음 구성을 추가합니다.

```
beegfs_ha_cluster_crm_config_options:
    stonith-enabled: False
```

2. Ansible 플레이북을 다시 실행하여 클러스터의 변경 사항을 적용합니다.

옵션 **2:** 수동으로 펜싱을 비활성화합니다**.**

경우에 따라 Ansible을 다시 실행하지 않고 펜싱을 일시적으로 비활성화할 수 있으며, 클러스터에 대한 문제 해결 또는 테스트를 용이하게 할 수 있습니다.

> 이 구성에서 BeeGFS 모니터가 장애를 감지하면 클러스터가 해당 리소스 그룹을 중지하려고 시도합니다. 전체 페일오버를 트리거하거나 영향을 받는 리소스 그룹을 다시 시작하거나 다른 호스트로 이동하려고 시도하지 않습니다. 복구하려면 문제를 해결한 다음 를 실행하십시오 `pcs resource cleanup` 또는 노드를 수동으로 대기 상태로 전환합니다.

단계:

1. 펜싱(STONITH)이 전역적으로 활성화 또는 비활성화되었는지 확인하려면 다음을 실행하십시오. `pcs property show stonith-enabled`
2. 펜싱 실행을 비활성화하려면: `pcs property set stonith-enabled=false`
3. 펜싱 실행을 활성화하려면: `pcs property set stonith-enabled=true`

> 다음에 Ansible 플레이북을 실행하면 이 설정이 재정의됩니다.

HA 클러스터 구성 요소를 업데이트합니다

BeeGFS 서비스 업그레이드

Ansible을 사용하여 HA 클러스터에서 실행 중인 BeeGFS 버전을 업데이트하세요.

개요

BeeGFS는 `major.minor.patch` 버전 관리 체계를 따릅니다. BeeGFS HA Ansible 역할은 지원되는 각 `major.minor` 버전(예: `beegfs_ha_7_2` 및 `beegfs_ha_7_3`)에 대해 제공됩니다. 각 HA 역할은 Ansible 컬렉션 릴리즈 시점에 제공되는 최신 BeeGFS 패치 버전에 고정되어 있습니다.

BeeGFS의 주요 버전, 마이너 버전 및 패치 버전 간 이동을 포함한 모든 BeeGFS 업그레이드에는 Ansible을 사용해야 합니다. BeeGFS를 업데이트하려면 먼저 BeeGFS Ansible 컬렉션을 업데이트해야 하며, 이 과정에서 배포/관리 자동화 및 기본 HA 클러스터에 대한 최신 수정 사항과 개선 사항도 함께 가져와집니다. 컬렉션을 최신 버전으로 업데이트한 후에도 `-e "beegfs_ha_force_upgrade=true"`가 설정된 상태에서 `ansible-playbook`을 실행하기 전까지는 BeeGFS가 업그레이드되지 않습니다. 각 업그레이드에 대한 자세한 내용은 현재 버전의 "BeeGFS 업그레이드 설명서"을 참조하십시오.

> BeeGFS v8로 업그레이드하는 경우 "BeeGFS v8로 업그레이드" 절차를 참조하십시오.

테스트된 업그레이드 경로

다음 업그레이드 경로는 테스트 및 검증을 완료했습니다.

원본 버전	버전 업그레이드	멀티 레일	세부 정보
7.2.6	7.3.2	예	v3.0.1에서 v3.1.0으로 Beegfs 컬렉션을 업그레이드하는 중, 멀티레일이 추가되었습니다
7.2.6	7.2.8	아니요	v3.0.1에서 v3.1.0으로 Beegfs 컬렉션 업그레이드
7.2.8	7.3.1에서 포함	예	begfs 컬렉션 v3.1.0을 사용하여 업그레이드하십시오. 멀티레일이 추가되었습니다
7.3.1에서 포함	7.3.2	예	Beegfs 컬렉션 v3.1.0을 사용하여 업그레이드합니다
7.3.2	7.4.1	예	Beegfs 컬렉션 v3.2.0을 사용하여 업그레이드합니다
7.4.1	7.4.2	예	Beegfs 컬렉션 v3.2.0을 사용하여 업그레이드합니다
7.4.2	7.4.6	예	Beegfs 컬렉션 v3.2.0을 사용하여 업그레이드합니다
7.4.6	8.0	예	"BeeGFS v8로 업그레이드" 절차의 지침을 사용하여 업그레이드하십시오.
7.4.6	8.1	예	"BeeGFS v8로 업그레이드" 절차의 지침을 사용하여 업그레이드하십시오.
7.4.6	8.2	예	"BeeGFS v8로 업그레이드" 절차의 지침을 사용하여 업그레이드하십시오.

BeeGFS 업그레이드 단계

다음 섹션에서는 BeeGFS Ansible 컬렉션 및 BeeGFS 자체를 업데이트하는 단계를 제공합니다. BeeGFS 주 버전 또는 부 버전 업데이트에 대한 추가 단계에 특히 주의하십시오.

1단계**: BeeGFS** 컬렉션 업그레이드

에 액세스하여 컬렉션 업그레이드용 "Ansible 갤럭시"에서 다음 명령을 실행합니다.

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

오프라인 컬렉션 업그레이드의 경우 에서 컬렉션을 다운로드하십시오 "Ansible 갤럭시" 원하는 을 클릭합니다 `Install Version` 그리고 나서 `Download tarball`. 타볼을 Ansible 제어 노드로 전송하고 다음 명령을 실행합니다.

```
ansible-galaxy collection install netapp_eseries-beegfs-<VERSION>.tar.gz
--upgrade
```

을 참조하십시오 "컬렉션 설치 중" 를 참조하십시오.

2단계: **Ansible** 인벤토리 업데이트

클러스터의 Ansible 인벤토리 파일에 필요한 또는 원하는 업데이트를 수행하십시오. 특정 업그레이드 요구 사항에 대한 자세한 내용은 아래 버전 업그레이드 참고 사항 섹션을 참조하십시오. BeeGFS HA 인벤토리 구성에 대한 일반 정보는 "Ansible 인벤토리 개요" 섹션을 참조하십시오.

3단계: **Ansible** 플레이북 업데이트(주 버전 또는 부 버전을 업데이트하는 경우에만)

주 버전 또는 부 버전 간에 이동하는 경우 `playbook.yml` 클러스터를 배포하고 유지 관리하는 데 사용되는 파일에서 원하는 버전을 반영하도록 역할의 이름을 `beegfs_ha_<VERSION>` 업데이트합니다. 예를 들어 BeeGFS 7.4를 배포하려는 경우 다음과 같은 이점이 `beegfs_ha_7_4`있습니다.

```
- hosts: all
  gather_facts: false
  any_errors_fatal: true
  collections:
    - netapp_eseries.beegfs
  tasks:
    - name: Ensure BeeGFS HA cluster is setup.
      ansible.builtin.import_role:  # import_role is required for tag
availability.
        name: beegfs_ha_7_4
```

이 플레이북 파일의 내용에 대한 자세한 "BeeGFS HA 클러스터를 구축합니다"내용은 섹션을 참조하십시오.

4단계: **BeeGFS** 업그레이드를 실행합니다

BeeGFS 업데이트 적용하기:

```
ansible-playbook -i inventory.yml beegfs_ha_playbook.yml -e
"beegfs_ha_force_upgrade=true" --tags beegfs_ha
```

BeeGFS HA 역할에서 다루는 비하인드 스토리:

- 각 BeeGFS 서비스가 기본 노드에 위치하도록 클러스터가 최적의 상태인지 확인합니다.
- 클러스터를 유지보수 모드로 전환합니다.
- HA 클러스터 구성 요소를 업데이트합니다(필요한 경우).
- 다음과 같이 각 파일 노드를 한 번에 하나씩 업그레이드합니다.
  - 대기 노드에 배치하고 서비스를 보조 노드로 페일오버합니다.
  - BeeGFS 패키지를 업그레이드합니다.
  - 서비스 대체.
- 클러스터를 유지보수 모드 외부로 이동합니다.

버전 업그레이드 참고 사항

BeeGFS 버전 **7.2.6** 또는 **7.3.0**에서 업그레이드

연결 기반 인증에 대한 변경 사항

BeeGFS 버전 7.3.2 이상에서는 연결 기반 인증이 구성되어 있어야 합니다. 다음 중 하나라도 구성되지 않으면 서비스가 시작되지 않습니다.

- `connAuthFile` 또는 를 지정합니다.
- 서비스의 구성 파일에서 `connDisableAuthentication=true` 설정.

보안을 위해 연결 기반 인증을 활성화하는 것을 적극 권장합니다. 자세한 내용은 "BeeGFS 연결 기반 인증"을 참조하십시오.

`beegfs_ha*` 역할은 인증 파일을 자동으로 생성하여 다음 위치로 배포합니다.

- 클러스터의 모든 파일 노드
- `<playbook_directory>/files/beegfs/<beegfs_mgmt_ip_address>_connAuthFile`의 Ansible 제어 노드

`beegfs_client` 역할은 파일이 존재할 경우 자동으로 감지하여 클라이언트에 적용합니다.

`beegfs_client` 역할을 사용하여 클라이언트를 구성하지 않은 경우 각 클라이언트에 인증 파일을 수동으로 배포하고 `beegfs-client.conf` 파일에서 `connAuthFile` 설정을 구성해야 합니다. 연결 기반 인증이 없는 BeeGFS 버전에서 업그레이드하는 경우 `group_vars/ha_cluster.yml`에서 `beegfs_ha_conn_auth_enabled: false`를 설정하여 업그레이드 중에 연결 기반 인증을 비활성화하지 않으면 클라이언트가 액세스 권한을 잃게 됩니다(권장하지 않음).

자세한 내용 및 다른 구성 옵션은 "일반 파일 노드 구성을 지정합니다" 섹션의 연결 인증 구성 단계를 참조하십시오.

BeeGFS v8로 업그레이드

다음 단계를 따라 BeeGFS HA 클러스터를 버전 7.4.6에서 BeeGFS v8로 업그레이드하십시오.

개요

BeeGFS v8은 BeeGFS v7에서 업그레이드하기 전에 추가 설정이 필요한 몇 가지 중요한 변경 사항을 도입했습니다. 이 문서에서는 BeeGFS v8의 새로운 요구 사항에 맞게 클러스터를 준비하고 BeeGFS v8로 업그레이드하는 과정을 안내합니다.

BeeGFS v8로 업그레이드하기 전에 시스템에 BeeGFS 7.4.6 이상이 설치되어 있는지 확인하십시오. BeeGFS 7.4.6 이전 버전을 실행 중인 클러스터는 이 BeeGFS v8 업그레이드 절차를 진행하기 전에 먼저 "버전 7.4.6으로 업그레이드"해야 합니다.

BeeGFS v8의 주요 변경 사항

BeeGFS v8에서는 다음과 같은 주요 변경 사항이 도입되었습니다.

- 라이선스 시행: BeeGFS v8은 스토리지 풀, 원격 스토리지 대상, BeeOND 등과 같은 프리미엄 기능을 사용하기 위해 라이선스가 필요합니다. 업그레이드하기 전에 BeeGFS 클러스터에 대한 유효한 라이선스를 확보하십시오. 필요한 경우 "BeeGFS 라이선스 포털"에서 임시 BeeGFS v8 평가판 라이선스를 받을 수 있습니다.
- 관리 서비스 데이터베이스 마이그레이션: BeeGFS v8의 새로운 TOML 기반 형식으로 구성을 활성화하려면 BeeGFS v7 관리 서비스 데이터베이스를 업데이트된 BeeGFS v8 형식으로 수동으로 마이그레이션해야 합니다.
- **TLS** 암호화: BeeGFS v8은 서비스 간의 안전한 통신을 위해 TLS를 도입했습니다. 업그레이드 과정에서 BeeGFS 관리 서비스와 `beegfs` 명령줄 유틸리티에 사용할 TLS 인증서를 생성하고 배포해야 합니다.

BeeGFS 8의 자세한 내용 및 추가 변경 사항은 "BeeGFS v8.0.0 업그레이드 가이드"를 참조하십시오.

> ⓘ BeeGFS v8로 업그레이드하려면 클러스터 가동 중지 시간이 필요합니다. 또한 BeeGFS v7 클라이언트는 BeeGFS v8 클러스터에 연결할 수 없습니다. 운영에 미치는 영향을 최소화하려면 클러스터와 클라이언트 간의 업그레이드 시기를 신중하게 조율하십시오.

업그레이드를 위해 BeeGFS 클러스터를 준비하세요

업그레이드를 시작하기 전에 원활한 전환과 다운타임 최소화를 위해 환경을 신중하게 준비하십시오.

1. 클러스터가 정상 상태인지, 모든 BeeGFS 서비스가 기본 노드에서 실행 중인지 확인하십시오. BeeGFS 서비스가 실행 중인 파일 노드에서 모든 Pacemaker 리소스가 기본 노드에서 실행 중인지 확인하십시오.

```
pcs status
```

2. 클러스터 구성을 기록하고 백업합니다.
   a. 클러스터 구성 백업에 대한 지침은 "BeeGFS 백업 설명서"을 참조하십시오.
   b. 기존 관리 데이터 디렉터리를 백업합니다.

```
cp -r /mnt/mgmt_tgt_mgmt01/data
/mnt/mgmt_tgt_mgmt01/data_beegfs_v7_backup_$(date +%Y%m%d)
```

   c. beegfs 클라이언트에서 다음 명령을 실행하고 출력을 참조용으로 저장합니다.

```
beegfs-ctl --getentryinfo --verbose /path/to/beegfs/mountpoint
```

   d. 미러링을 사용하는 경우 자세한 상태 정보를 수집하십시오.

```
beegfs-ctl --listtargets --longnodes --state --spaceinfo
--mirrorgroups --nodetype=meta
beegfs-ctl --listtargets --longnodes --state --spaceinfo
--mirrorgroups --nodetype=storage
```

3. 클라이언트의 다운타임을 준비하고 `beegfs-client`서비스를 중지하십시오. 각 클라이언트에 대해 다음을 실행하십시오.

```
systemctl stop beegfs-client
```

4. 각 Pacemaker 클러스터에 대해 STONITH를 비활성화하십시오. 이렇게 하면 불필요한 노드 재부팅을 트리거하지 않고 업그레이드 후 클러스터의 무결성을 검증할 수 있습니다.

```
pcs property set stonith-enabled=false
```

5. BeeGFS 네임스페이스의 모든 Pacemaker 클러스터에 대해 PCS를 사용하여 클러스터를 중지합니다.

```
pcs cluster stop --all
```

BeeGFS 패키지 업그레이드

클러스터의 모든 파일 노드에 사용 중인 Linux 배포판에 맞는 BeeGFS v8 패키지 저장소를 추가하십시오. 공식 BeeGFS 저장소 사용 방법은 "BeeGFS 다운로드 페이지"에서 확인할 수 있습니다. 그렇지 않은 경우 로컬 beegfs 미러 저장소를 적절히 구성하십시오.

다음 단계는 RHEL 9 파일 노드에서 공식 BeeGFS 8.2 리포지토리를 사용하는 방법을 안내합니다. 클러스터의 모든 파일 노드에서 다음 단계를 수행하십시오.

1. BeeGFS GPG 키를 가져옵니다:

```
rpm --import https://www.beegfs.io/release/beegfs_8.2/gpg/GPG-KEY-beegfs
```

2. BeeGFS 리포지토리를 가져옵니다.

```
curl -L -o /etc/yum.repos.d/beegfs-rhel9.repo
https://www.beegfs.io/release/beegfs_8.2/dists/beegfs-rhel9.repo
```

새로운 BeeGFS v8 리포지토리와의 충돌을 방지하려면 이전에 구성된 BeeGFS 리포지토리를 모두 제거하십시오.

3. 패키지 관리자 캐시를 정리하세요:

```
dnf clean all
```

4. 모든 파일 노드에서 BeeGFS 패키지를 BeeGFS 8.2로 업데이트하십시오.

```
dnf update beegfs-mgmtd beegfs-storage beegfs-meta libbeegfs-ib
```

표준 클러스터에서 `beegfs-mgmtd` 패키지는 처음 두 개의 파일 노드에서만 업데이트됩니다.

관리 데이터베이스 업그레이드

BeeGFS 관리 서비스가 실행 중인 파일 노드 중 하나에서 다음 단계를 수행하여 관리 데이터베이스를 BeeGFS v7에서 v8로 마이그레이션합니다.

1. 모든 NVMe 디바이스를 나열하고 관리 타겟을 기준으로 필터링합니다.

```
nvme netapp smdevices | grep mgmt_tgt
```

   a. 출력에서 장치 경로를 확인하십시오.

   b. 기존 관리 대상 마운트 지점에 관리 대상 장치를 마운트합니다(`/dev/nvmeXnY`를 장치 경로로 대체):

```
mount /dev/nvmeXnY /mnt/mgmt_tgt_mgmt01/
```

2. 다음을 실행하여 BeeGFS 7 관리 데이터를 새 데이터베이스 형식으로 가져옵니다.

```
/opt/beegfs/sbin/beegfs-mgmtd --import-from
-v7=/mnt/mgmt_tgt_mgmt01/data/
```

   예상 출력:

```
Created new database version 3 at "/var/lib/beegfs/mgmtd.sqlite".
Successfully imported v7 management data from
"/mnt/mgmt_tgt_mgmt01/data/".
```

BeeGFS v8의 강화된 유효성 검사 요구 사항으로 인해 자동 가져오기가 모든 경우에 성공하지 못할 수 있습니다. 예를 들어, 대상이 존재하지 않는 스토리지 풀에 할당된 경우 가져오기가 실패합니다. 마이그레이션이 실패하면 업그레이드를 진행하지 마십시오. 데이터베이스 마이그레이션 문제 해결을 위해 NetApp 지원팀에 문의하십시오. 임시 해결책으로, 문제가 해결될 때까지 BeeGFS v8 패키지를 다운그레이드하고 BeeGFS v7을 계속 실행할 수 있습니다.

3. 생성된 SQLite 파일을 관리 서비스 마운트로 이동합니다.

```
mv /var/lib/beegfs/mgmtd.sqlite /mnt/mgmt_tgt_mgmt01/data/
```

4. 생성된 `beegfs-mgmtd.toml`을 관리 서비스 마운트로 이동합니다.

```
mv /etc/beegfs/beegfs-mgmtd.toml /mnt/mgmt_tgt_mgmt01/mgmt_config/
```

`beegfs-mgmtd.toml` 구성 파일 준비는 다음 섹션의 라이선싱 및 TLS 구성 단계를 완료한 후에 수행됩니다.

라이선싱 구성

1. beegfs 관리 서비스를 실행하는 모든 노드에 beegfs 라이선스 패키지를 설치하십시오. 일반적으로 클러스터의 첫 번째 두 노드입니다.

```
dnf install libbeegfs-license
```

2. BeeGFS v8 라이선스 파일을 관리 노드에 다운로드하여 다음 위치에 배치하십시오.

```
/etc/beegfs/license.pem
```

TLS 암호화 구성

BeeGFS v8은 관리 서비스와 클라이언트 간의 안전한 통신을 위해 TLS 암호화를 필요로 합니다. 관리 서비스와 클라이언트 서비스 간의 네트워크 통신에 TLS 암호화를 구성하는 방법은 세 가지가 있습니다. 권장되는 가장 안전한 방법은 신뢰할 수 있는 인증 기관(CA)에서 서명한 인증서를 사용하는 것입니다. 또는 자체 로컬 CA를 생성하여 BeeGFS 클러스터용 인증서를 서명할 수도 있습니다. 암호화가 필요하지 않은 환경이나 문제 해결을 위한 경우에는 TLS를 완전히 비활성화할 수 있지만, 이 경우 민감한 정보가 네트워크에 노출되므로 권장하지 않습니다.

진행하기 전에 "BeeGFS 8용 TLS 암호화 구성" 가이드의 지침에 따라 환경에 맞는 TLS 암호화를 설정하십시오.

업데이트 관리 서비스 구성

BeeGFS v7 구성 파일의 설정을 `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml` 파일로 수동으로 전송하여 BeeGFS v8 관리 서비스 구성 파일을 준비합니다.

1. 관리 대상이 마운트된 관리 노드에서 BeeGFS 7용 관리 서비스 파일 `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.conf`을 참조한 다음 모든 설정을 `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml` 파일로 전송합니다. 기본 설정의 경우 `beegfs-mgmtd.toml`은 다음과 같을 수 있습니다.

```
beemsg-port = 8008
grpc-port = 8010
log-level = "info"
node-offline-timeout = "900s"
quota-enable = false
auth-disable = false
auth-file = "/etc/beegfs/<mgmt_service_ip>_connAuthFile"
db-file = "/mnt/mgmt_tgt_mgmt01/data/mgmtd.sqlite"
license-disable = false
license-cert-file = "/etc/beegfs/license.pem"
tls-disable = false
tls-cert-file = "/etc/beegfs/mgmtd_tls_cert.pem"
tls-key-file = "/etc/beegfs/mgmtd_tls_key.pem"
interfaces = ['i1b:mgmt_1', 'i2b:mgmt_2']
```

필요에 따라 모든 경로를 조정하여 사용자의 환경 및 TLS 구성과 일치시키십시오.

2. 관리 서비스를 실행하는 각 파일 노드에서 systemd 서비스 파일을 수정하여 새 구성 파일 위치를 가리키도록 합니다.

```
sudo sed -i 's|ExecStart=.*|ExecStart=nice -n -3
/opt/beegfs/sbin/beegfs-mgmtd --config-file
/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml|'
/etc/systemd/system/beegfs-mgmtd.service
```

   a. systemd를 다시 로드합니다.

```
systemctl daemon-reload
```

3. 관리 서비스를 실행하는 각 파일 노드에 대해 관리 서비스의 gRPC 통신을 위해 포트 8010을 엽니다.

   a. beegfs 영역에 포트 8010/tcp를 추가합니다.

```
sudo firewall-cmd --zone=beegfs --permanent --add-port=8010/tcp
```

   b. 변경 사항을 적용하려면 방화벽을 다시 로드하십시오.

```
sudo firewall-cmd --reload
```

BeeGFS 모니터 스크립트 업데이트

Pacemaker `beegfs-monitor` OCF 스크립트는 새로운 TOML 구성 형식과 systemd 서비스 관리를 지원하도록 업데이트해야 합니다. 클러스터의 한 노드에서 스크립트를 업데이트한 다음, 업데이트된 스크립트를 다른 모든 노드로 복사하십시오.

1. 현재 스크립트의 백업을 생성합니다:

```
cp /usr/lib/ocf/resource.d/eseries/beegfs-monitor
/usr/lib/ocf/resource.d/eseries/beegfs-monitor.bak.$(date +%F)
```

2. 관리 구성 파일 경로를 `.conf`에서 `.toml`(으)로 업데이트하십시오:

```
sed -i 's|mgmt_config/beegfs-mgmtd\.conf|mgmt_config/beegfs-mgmtd.toml|'
/usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

또는 스크립트에서 다음 블록을 수동으로 찾습니다.

```
case $type in
  management)
    conf_path="${configuration_mount}/mgmt_config/beegfs-mgmtd.conf"
    ;;
```

그리고 다음으로 바꾸세요:

```
case $type in
  management)
    conf_path="${configuration_mount}/mgmt_config/beegfs-mgmtd.toml"
    ;;
```

3. `get_interfaces()` 및 `get_subnet_ips()` 함수를 업데이트하여 TOML 구성을 지원합니다.
   a. 텍스트 편집기에서 스크립트를 엽니다.

```
vi /usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

   b. 다음 두 함수를 찾으세요: `get_interfaces()` 및 `get_subnet_ips()`.
   c. `get_interfaces()`에서 시작하여 `get_subnet_ips()`의 끝까지 두 함수 전체를 삭제하세요.
   d. 다음 업데이트된 함수를 복사하여 해당 위치에 붙여넣으세요.

```
# Return network communication interface name(s) from the BeeGFS
resource's connInterfaceFile
get_interfaces() {
  # Determine BeeGFS service network IP interfaces.
  if [ "$type" = "management" ]; then
    interfaces_line=$(grep "^interfaces =" "$conf_path")
    interfaces_list=$(echo "$interfaces_line" | sed "s/.*= \[\(.*
\)\]/\1/")
    interfaces=$(echo "$interfaces_list" | tr -d "'" | tr -d " " | tr
',' '\n')

    for entry in $interfaces; do
      echo "$entry" | cut -d ':' -f 1
    done
  else
    connInterfacesFile_path=$(grep "^connInterfacesFile" "$conf_path"
| tr -d "[:space:]" | cut -f 2 -d "=")

    if [ -f "$connInterfacesFile_path" ]; then
      while read -r entry; do
        echo "$entry" | cut -f 1 -d ':'
      done < "$connInterfacesFile_path"
    fi
  fi
}

# Return list containing all the BeeGFS resource's usable IP
addresses. *Note that these are filtered by the connNetFilterFile
entries.
get_subnet_ips() {
  # Determine all possible BeeGFS service network IP addresses.
  if [ "$type" != "management" ]; then
    connNetFilterFile_path=$(grep "^connNetFilterFile" "$conf_path" |
tr -d "[:space:]" | cut -f 2 -d "=")

    filter_ips=""
    if [ -n "$connNetFilterFile_path" ] && [ -e
$connNetFilterFile_path ]; then
      while read -r filter; do
        filter_ips="$filter_ips $(get_ipv4_subnet_addresses $filter)"
      done < $connNetFilterFile_path
    fi

    echo "$filter_ips"
  fi
}
```

e. 텍스트 편집기를 저장하고 종료합니다.

f. 진행하기 전에 다음 명령을 실행하여 스크립트의 구문 오류를 확인하십시오. 출력이 없으면 스크립트의 구문이 올바릅니다.

```
bash -n /usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

4. 일관성을 유지하기 위해 업데이트된 `beegfs-monitor` OCF 스크립트를 클러스터의 다른 모든 노드에 복사하십시오.

```
scp /usr/lib/ocf/resource.d/eseries/beegfs-monitor
user@node:/usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

클러스터를 다시 온라인 상태로 전환

1. 이전의 모든 업그레이드 단계를 완료한 후 모든 노드에서 BeeGFS 서비스를 시작하여 클러스터를 다시 온라인 상태로 전환하십시오.

```
pcs cluster start --all
```

2. `beegfs-mgmtd` 서비스가 성공적으로 시작되었는지 확인하십시오.

```
journalctl -xeu beegfs-mgmtd
```

예상 출력에는 다음과 같은 줄이 포함됩니다:

```
Started Cluster Controlled beegfs-mgmtd.
Loaded config file from "/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-
mgmtd.toml"
Successfully initialized certificate verification library.
Successfully loaded license certificate: TMP-113489268
Opened database at "/mnt/mgmt_tgt_mgmt01/data/mgmtd.sqlite"
Listening for BeeGFS connections on [::]:8008
Serving gRPC requests on [::]:8010
```

저널 로그에 오류가 나타나면 관리 구성 파일 경로를 검토하고 BeeGFS 7 구성 파일에서 모든 값이 올바르게 전송되었는지 확인하십시오.

3. `pcs status`를 실행하고 클러스터가 정상 상태이며 서비스가 기본 설정 노드에서 시작되었는지 확인합니다.
4. 클러스터가 정상 상태임을 확인한 후 STONITH를 다시 활성화하십시오.

```
pcs property set stonith-enabled=true
```

5. 클러스터의 BeeGFS 클라이언트를 업그레이드하고 BeeGFS 클러스터의 상태를 확인하려면 다음 섹션으로 이동하십시오.

BeeGFS 클라이언트 업그레이드

클러스터를 BeeGFS v8로 성공적으로 업그레이드한 후에는 모든 BeeGFS 클라이언트도 업그레이드해야 합니다.

다음 단계는 Ubuntu 기반 시스템에서 BeeGFS 클라이언트를 업그레이드하는 과정을 설명합니다.

1. 아직 하지 않았다면 BeeGFS 클라이언트 서비스를 중지하십시오.

```
systemctl stop beegfs-client
```

2. 사용하는 Linux 배포판에 맞는 BeeGFS v8 패키지 저장소를 추가하세요. 공식 BeeGFS 저장소 사용 방법은 "^BeeGFS 다운로드 페이지"에서 확인할 수 있습니다. 그렇지 않은 경우 로컬 BeeGFS 미러 저장소를 적절히 구성하세요.

   다음 단계에서는 Ubuntu 기반 시스템에서 공식 BeeGFS 8.2 리포지토리를 사용합니다.

3. BeeGFS GPG 키를 가져옵니다:

```
wget https://www.beegfs.io/release/beegfs_8.2/gpg/GPG-KEY-beegfs -O
/etc/apt/trusted.gpg.d/beegfs.asc
```

4. 리포지토리 파일을 다운로드하세요:

```
wget https://www.beegfs.io/release/beegfs_8.2/dists/beegfs-noble.list -O
/etc/apt/sources.list.d/beegfs.list
```

   새로운 BeeGFS v8 리포지토리와의 충돌을 방지하려면 이전에 구성된 BeeGFS 리포지토리를 모두 제거하십시오.

5. BeeGFS 클라이언트 패키지를 업그레이드합니다.

```
apt-get update
apt-get install --only-upgrade beegfs-client
```

6. 클라이언트에 TLS를 구성하십시오. BeeGFS CLI를 사용하려면 TLS가 필요합니다. "BeeGFS 8용 TLS 암호화 구성" 절차를 참조하여 클라이언트에서 TLS를 구성하십시오.
7. BeeGFS 클라이언트 서비스를 시작합니다.

```
systemctl start beegfs-client
```

업그레이드 확인

BeeGFS v8로 업그레이드를 완료한 후 다음 명령을 실행하여 업그레이드가 성공했는지 확인하십시오.

1. 루트 inode가 이전과 동일한 메타데이터 노드에 의해 소유되는지 확인하십시오. 관리 서비스에서 `import-from-v7` 기능을 사용한 경우 이 작업은 자동으로 수행됩니다.

```
beegfs entry info /mnt/beegfs
```

2. 모든 노드와 대상이 온라인 상태이고 양호한 상태인지 확인하십시오.

```
beegfs health check
```

"Available Capacity" 검사에서 대상의 여유 공간이 부족하다는 경고가 표시되면 `beegfs-mgmtd.toml` 파일에 정의된 "capacity pool" 임계값을 환경에 맞게 조정할 수 있습니다.

HA 클러스터의 페이스 메이커 및 Corosync 패키지를 업그레이드합니다

HA 클러스터의 심장박동기 및 Corosync 패키지를 업그레이드하려면 다음 단계를 수행하십시오.

개요

페이스 메이커와 Corosync를 업그레이드하면 새로운 기능, 보안 패치 및 성능 향상의 이점을 누릴 수 있습니다.

업그레이드 접근 방식

클러스터를 업그레이드하는 데에는 롤링 업그레이드 또는 전체 클러스터 종료라는 두 가지 권장 방법이 있습니다. 각 접근 방식에는 고유한 장점과 단점이 있습니다. 업그레이드 절차는 심장박동기 릴리스 버전에 따라 다를 수 있습니다. ClusterLabs의 "심장박동기 클러스터 업그레이드"설명서를 참조하여 사용할 방법을 결정합니다. 업그레이드 방법을 따르기 전에 다음 사항을 확인하십시오.

- NetApp BeeGFS 솔루션 내에서 새로운 페이스 메이커 및 Corosync 패키지가 지원됩니다.
- BeeGFS 파일 시스템 및 Pacemaker 클러스터 구성에 유효한 백업이 있습니다.
- 클러스터가 정상 상태입니다.

롤링 업그레이드

이 방법은 클러스터에서 각 노드를 제거하고 업그레이드한 다음 모든 노드에서 새 버전이 실행될 때까지 클러스터에 다시 도입하는 것입니다. 이렇게 하면 클러스터가 운영 상태로 유지되므로 대규모 HA 클러스터에 이상적이지만, 프로세스 중에 혼합 버전을 실행할 위험이 있습니다. 2노드 클러스터에서 이 접근 방식은 사용하지 않아야 합니다.

1. 각 BeeGFS 서비스가 1차 노드에서 실행되고 있는 상태에서 클러스터가 최적의 상태인지 확인합니다. 자세한 내용은 을 "클러스터의 상태를 검사합니다" 참조하십시오.
2. 업그레이드할 노드를 대기 모드로 전환하여 모든 BeeGFS 서비스를 방전하거나 이동합니다.

```
pcs node standby <HOSTNAME>
```

3. 다음을 실행하여 노드의 서비스가 소진되었는지 확인합니다.

```
pcs status
```

대기 상태인 노드에 대해 보고된 서비스가 `Started` 없는지 확인합니다.

> ⓘ 클러스터 크기에 따라 서비스가 자매 노드로 이동하는 데 몇 초 또는 몇 분이 걸릴 수 있습니다. BeeGFS 서비스가 자매 노드에서 시작되지 않는 경우 를 참조하십시오"문제 해결 설명서".

4. 노드에서 클러스터를 종료합니다.

```
pcs cluster stop <HOSTNAME>
```

5. 노드에서 심장박동기, Corosync 및 PCS 패키지를 업그레이드합니다.

> ⓘ 패키지 관리자 명령은 운영 체제에 따라 다릅니다. 다음은 RHEL 8 이상을 실행하는 시스템에 대한 명령입니다.

```
dnf update pacemaker-<version>
```

```
dnf update corosync-<version>
```

```
dnf update pcs-<version>
```

6. 노드에서 심장박동기 클러스터 서비스를 시작합니다.

```
pcs cluster start <HOSTNAME>
```

7. 패키지가 업데이트된 경우 `pcs` 클러스터를 사용하여 노드를 다시 인증합니다.

```
pcs host auth <HOSTNAME>
```

8. 박동조율기 구성이 여전히 도구에 유효한지 `crm_verify` 확인합니다.

   클러스터 업그레이드 중에 한 번만 확인하면 됩니다.

```
crm_verify -L -V
```

9. 노드를 대기 모드에서 해제합니다.

```
pcs node unstandby <HOSTNAME>
```

10. 모든 BeeGFS 서비스를 기본 노드로 다시 재배치:

```
pcs resource relocate run
```

11. 모든 노드가 원하는 페이스 메이커, Corosync 및 PCS 버전을 실행할 때까지 클러스터의 각 노드에 대해 이전 단계를 반복합니다.
12. 마지막으로, `pcs status` 클러스터를 실행하고 클러스터 상태가 양호한지 확인하고 가 `Current DC` 원하는 페이스 메이커 버전을 보고합니다.

   `Current DC'혼합 버전'이 보고되면 클러스터의 노드가 이전 페이스 메이커 버전과 함께 실행되고 있으므로 업그레이드해야 합니다. 업그레이드된 노드가 클러스터에 다시 연결할 수 없거나 리소스가 시작되지 않는 경우, 클러스터 로그를 확인하고 알려진 업그레이드 문제에 대해서는 Pacemaker 릴리즈 노트 또는 사용자 가이드를 참조하십시오.

클러스터 종료를 완료합니다

이 접근 방식에서는 모든 클러스터 노드 및 리소스가 종료되고 노드가 업그레이드된 다음 클러스터가 다시 시작됩니다. 이 방법은 페이스 메이커 및 Corosync 버전이 혼합 버전 구성을 지원하지 않는 경우에 필요합니다.

1. 각 BeeGFS 서비스가 1차 노드에서 실행되고 있는 상태에서 클러스터가 최적의 상태인지 확인합니다. 자세한 내용은 을 "클러스터의 상태를 검사합니다" 참조하십시오.
2. 모든 노드에서 클러스터 소프트웨어(심장박동기 및 Corosync)를 종료합니다.

   클러스터 크기에 따라 전체 클러스터를 중지하는 데 몇 초 또는 몇 분이 걸릴 수 있습니다.

```
pcs cluster stop --all
```

3. 모든 노드에서 클러스터 서비스가 종료되면 요구 사항에 따라 각 노드의 심장박동기, Corosync 및 PCS 패키지를 업그레이드합니다.

   패키지 관리자 명령은 운영 체제에 따라 다릅니다. 다음은 RHEL 8 이상을 실행하는 시스템에 대한 명령입니다.

```
dnf update pacemaker-<version>
```

```
dnf update corosync-<version>
```

```
dnf update pcs-<version>
```

4. 모든 노드를 업그레이드한 후 모든 노드에서 클러스터 소프트웨어를 시작합니다.

```
pcs cluster start --all
```

5. 패키지가 업데이트된 경우 `pcs` 클러스터의 각 노드를 다시 인증합니다.

```
pcs host auth <HOSTNAME>
```

6. 마지막으로, `pcs status` 클러스터를 실행하고 클러스터의 상태가 양호한지 확인하고 가 `Current DC` 올바른 심장박동기 버전을 보고합니다.

> ⓘ `Current DC'혼합 버전'이 보고되면 클러스터의 노드가 이전 페이스 메이커 버전과 함께 실행되고 있으므로 업그레이드해야 합니다.

파일 노드 어댑터 펌웨어를 업데이트합니다

다음 단계에 따라 파일 노드의 ConnectX-7 어댑터를 최신 펌웨어로 업데이트합니다.

개요

새로운 MLNX_OFED 드라이버를 지원하거나 새로운 기능을 활성화하거나 버그를 수정하려면 ConnectX-7 어댑터 펌웨어를 업데이트해야 할 수 있습니다. 이 설명서에서는 NVIDIA의 유틸리티를 사용하여 어댑터를 업데이트할 수 있습니다. 이 `mlxfwmanager` 유틸리티는 사용 편의성과 효율성이 우수합니다.

업그레이드 고려 사항

이 가이드에서는 ConnectX-7 어댑터 펌웨어를 업데이트하는 두 가지 방법, 즉 롤링 업데이트와 2노드 클러스터 업데이트에 대해 설명합니다. 클러스터 크기에 따라 적절한 업데이트 방법을 선택합니다. 펌웨어 업데이트를 수행하기 전에 다음 사항을 확인하십시오.

- 지원되는 MLNX_OFED 드라이버가 설치되어 있으면 을 참조하십시오."기술 요구 사항"
- BeeGFS 파일 시스템 및 Pacemaker 클러스터 구성에 유효한 백업이 있습니다.
- 클러스터가 정상 상태입니다.

펌웨어 업데이트 준비

NVIDIA의 MLNX_OFED 드라이버와 함께 번들로 제공되는 노드의 어댑터 펌웨어를 업데이트하려면 NVIDIA의 유틸리티를 사용하는 것이 좋습니다 `mlxfwmanager`. 업데이트를 시작하기 전에 에서 어댑터의 펌웨어 이미지를 "NVIDIA의 지원 사이트"다운로드하여 각 파일 노드에 저장합니다.

Lenovo ConnectX-7 어댑터의 경우 `mlxfwmanager_LES` NVIDIA 페이지에서 사용할 수 있는 도구를 "OEM 펌웨어"사용합니다.

롤링 업데이트 접근 방식

이 접근 방식은 3개 이상의 노드가 있는 HA 클러스터에 권장됩니다. 이 접근 방식에는 한 번에 하나의 파일 노드에서 어댑터 펌웨어를 업데이트하여 HA 클러스터가 서비스 요청을 처리할 수 있습니다. 하지만 이 시간 동안 I/O를 처리하지 않는 것이 좋습니다.

1. 각 BeeGFS 서비스가 1차 노드에서 실행되고 있는 상태에서 클러스터가 최적의 상태인지 확인합니다. 자세한 내용은 을 "클러스터의 상태를 검사합니다" 참조하십시오.
2. 업데이트할 파일 노드를 선택하고 대기 모드로 전환하면 해당 노드에서 모든 BeeGFS 서비스를 드레이닝(또는 이동)합니다.

```
pcs node standby <HOSTNAME>
```

3. 다음을 실행하여 노드의 서비스가 방전되었는지 확인합니다.

```
pcs status
```

대기 상태인 노드에 대해 보고하는 서비스가 없는지 확인합니다 `Started`.

클러스터 크기에 따라 BeeGFS 서비스가 자매 노드로 이동하는 데 몇 초 또는 몇 분이 걸릴 수 있습니다. BeeGFS 서비스가 자매 노드에서 시작되지 않는 경우 를 참조하십시오"문제 해결 설명서".

4. 을 사용하여 어댑터 펌웨어를 `mlxfwmanager` 업데이트합니다.

```
mlxfwmanager -i <path/to/firmware.bin> -u
```

펌웨어 업데이트를 수신하는 각 어댑터에 대해 를 `PCI Device Name` 확인합니다.

5. 유틸리티를 사용하여 각 어댑터를 재설정하여 `mlxfwreset` 새 펌웨어를 적용합니다.

일부 펌웨어 업데이트에서는 업데이트를 적용하기 위해 재부팅해야 할 수 있습니다. 지침은 을 "NVIDIA의 mlxfwreset 제한 사항"참조하십시오. 재부팅이 필요한 경우 어댑터를 재설정하는 대신 재부팅을 수행하십시오.

a. OpenSM 서비스를 중지합니다.

```
systemctl stop opensm
```

b. 앞서 언급한 각 명령에 대해 다음 명령을 `PCI Device Name` 실행합니다.

```
mlxfwreset -d <pci_device_name> reset -y
```

c. OpenSM 서비스를 시작합니다.

```
systemctl start opensm
```

d. 다시 시작하세요 `eseries_nvme_ib.service` .

```
systemctl restart eseries_nvme_ib.service
```

e. E-시리즈 스토리지 어레이의 볼륨이 있는지 확인하세요.

```
multipath -ll
```

1. `ibstat`다음을 실행하여 모든 어댑터가 원하는 펌웨어 버전에서 실행되고 있는지 확인합니다.

```
ibstat
```

2. 노드에서 심장박동기 클러스터 서비스를 시작합니다.

```
pcs cluster start <HOSTNAME>
```

3. 노드를 대기 모드에서 해제합니다.

```
pcs node unstandby <HOSTNAME>
```

4. 모든 BeeGFS 서비스를 기본 노드로 다시 재배치:

```
pcs resource relocate run
```

모든 어댑터가 업데이트될 때까지 클러스터의 각 파일 노드에 대해 이 단계를 반복합니다.

2노드 클러스터 업데이트 접근 방식

이 접근 방식은 2개의 노드만 있는 HA 클러스터에 권장됩니다. 이 방법은 롤링 업데이트와 유사하지만 한 노드의 클러스터 서비스가 중지될 때 서비스 다운타임을 방지하기 위한 추가 단계가 포함되어 있습니다.

1. 각 BeeGFS 서비스가 1차 노드에서 실행되고 있는 상태에서 클러스터가 최적의 상태인지 확인합니다. 자세한 내용은 을 "클러스터의 상태를 검사합니다" 참조하십시오.
2. 업데이트할 파일 노드를 선택하고 노드를 대기 모드로 전환하면 해당 노드에서 모든 BeeGFS 서비스를 압축(또는 이동)합니다.

```
pcs node standby <HOSTNAME>
```

3. 다음을 실행하여 노드의 리소스가 소모되었는지 확인합니다.

```
pcs status
```

대기 상태인 노드에 대해 보고하는 서비스가 없는지 확인합니다 `Started`.

클러스터 크기에 따라 BeeGFS 서비스가 자매 노드로 보고되려면 몇 초 또는 몇 분이 걸릴 수 `Started` 있습니다. BeeGFS 서비스를 시작하지 못하는 경우 를 참조하십시오"문제 해결 설명서".

4. 클러스터를 유지보수 모드로 전환합니다.

```
pcs property set maintenance-mode=true
```

5. 을 사용하여 어댑터 펌웨어를 `mlxfwmanager` 업데이트합니다.

```
mlxfwmanager -i <path/to/firmware.bin> -u
```

펌웨어 업데이트를 수신하는 각 어댑터에 대해 를 `PCI Device Name` 확인합니다.

6. 유틸리티를 사용하여 각 어댑터를 재설정하여 `mlxfwreset` 새 펌웨어를 적용합니다.

일부 펌웨어 업데이트에서는 업데이트를 적용하기 위해 재부팅해야 할 수 있습니다. 지침은 을 "NVIDIA의 mlxfwreset 제한 사항"참조하십시오. 재부팅이 필요한 경우 어댑터를 재설정하는 대신 재부팅을 수행하십시오.

a. OpenSM 서비스를 중지합니다.

```
systemctl stop opensm
```

b. 앞서 언급한 각 명령에 대해 다음 명령을 `PCI Device Name` 실행합니다.

```
mlxfwreset -d <pci_device_name> reset -y
```

c. OpenSM 서비스를 시작합니다.

```
systemctl start opensm
```

7. `ibstat`다음을 실행하여 모든 어댑터가 원하는 펌웨어 버전에서 실행되고 있는지 확인합니다.

```
ibstat
```

8. 노드에서 심장박동기 클러스터 서비스를 시작합니다.

```
pcs cluster start <HOSTNAME>
```

9. 노드를 대기 모드에서 해제합니다.

```
pcs node unstandby <HOSTNAME>
```

10. 클러스터를 유지보수 모드에서 해제합니다.

```
pcs property set maintenance-mode=false
```

11. 모든 BeeGFS 서비스를 기본 노드로 다시 재배치:

```
pcs resource relocate run
```

모든 어댑터가 업데이트될 때까지 클러스터의 각 파일 노드에 대해 이 단계를 반복합니다.

E-Series 스토리지 시스템을 업그레이드합니다

HA 클러스터의 E-Series 스토리지 어레이 구성 요소를 업그레이드하려면 다음 단계를 따르십시오.

개요

최신 펌웨어로 HA 클러스터의 NetApp E-Series 스토리지 어레이를 최신 상태로 유지하면 최적의 성능과 향상된 보안을 보장할 수 있습니다. 스토리지 어레이용 펌웨어 업데이트는 SANtricity OS, NVSRAM 및 드라이브 펌웨어 파일을 통해 적용됩니다.

스토리지 어레이는 HA 클러스터를 온라인으로 업그레이드할 수 있지만 모든 업그레이드를 위해 클러스터를 유지보수 모드로 전환하는 것이 좋습니다.

블록 노드 업그레이드 단계

다음 단계에서는 `Netapp_Eseries.Santricity` Ansible 컬렉션을 사용하여 스토리지 어레이의 펌웨어를 업데이트하는 방법을 간략히 설명합니다. 계속하기 전에 "업그레이드 고려 사항"E-Series 시스템 업데이트에 대한 를 검토하십시오.

SANtricity OS 11.80 이상 릴리즈로 업그레이드하는 것은 11.70.5P1부터 가능합니다. 추가 업그레이드를 적용하기 전에 먼저 스토리지 어레이를 11.70.5P1로 업그레이드해야 합니다.

1. Ansible 제어 노드에서 최신 SANtricity Ansible Collection을 사용하고 있는지 확인합니다.
   - 에 액세스하여 컬렉션 업그레이드용 "Ansible 갤럭시"에서 다음 명령을 실행합니다.

     ```
     ansible-galaxy collection install netapp_eseries.santricity --upgrade
     ```

   - 오프라인 업그레이드의 경우 에서 컬렉션 타르볼을 다운로드하여 "Ansible 갤럭시"컨트롤 노드로 전송한 후 다음을 실행합니다.

     ```
     ansible-galaxy collection install netapp_eseries-santricity-
     <VERSION>.tar.gz --upgrade
     ```

     을 참조하십시오 "컬렉션 설치 중" 를 참조하십시오.

2. 스토리지 배열 및 드라이브에 대한 최신 펌웨어를 가져옵니다.
   a. 펌웨어 파일을 다운로드합니다.
      - * SANtricity OS 및 NVSRAM: * "NetApp Support 사이트"스토리지 어레이 모델에 맞는 SANtricity OS 및 NVSRAM 최신 릴리스로 이동하여 다운로드합니다.
      - * 드라이브 펌웨어: * 로 "E-Series 디스크 펌웨어 사이트입니다"이동하여 각 스토리지 배열의 드라이브 모델에 대한 최신 펌웨어를 다운로드합니다.

   b. SANtricity OS, NVSRAM 및 드라이브 펌웨어 파일을 Ansible 제어 노드의 `<inventory_directory>/packages` 디렉토리에 저장합니다.
3. 필요한 경우 업데이트가 필요한 모든 스토리지 어레이(블록 노드)를 포함하도록 클러스터의 Ansible 인벤토리 파일을 업데이트합니다. 지침은 "Ansible 인벤토리 개요"섹션을 참조하십시오.
4. 1차 노드에서 각 BeeGFS 서비스를 통해 클러스터가 최적의 상태로 유지되도록 합니다. 자세한 내용은 을 "클러스터의 상태를 검사합니다" 참조하십시오.
5. 의 지침에 따라 클러스터를 유지보수 모드로 "클러스터를 유지보수 모드로 전환합니다"전환합니다.
6. 이라는 새 Ansible 플레이북을 `update_block_node_playbook.yml`생성합니다. SANtricity OS, NVSRAM 및 드라이브 펌웨어 버전을 원하는 업그레이드 경로로 대체하여 플레이북에 다음 콘텐츠를 채웁니다.

```
- hosts: eseries_storage_systems
  gather_facts: false
  any_errors_fatal: true
  collections:
    - netapp_eseries.santricity
  vars:
    eseries_firmware_firmware: "packages/<SantricityOS>.dlp"
    eseries_firmware_nvsram: "packages/<NVSRAM>.dlp"
    eseries_drive_firmware_firmware_list:
      - "packages/<drive_firmware>.dlp"
    eseries_drive_firmware_upgrade_drives_online: true

  tasks:
    - name: Configure NetApp E-Series block nodes.
      import_role:
        name: nar_santricity_management
```

7. 업데이트를 시작하려면 Ansible 컨트롤 노드에서 다음 명령을 실행합니다.

```
ansible-playbook -i inventory.yml update_block_node_playbook.yml
```

8. 플레이북이 완료된 후 각 스토리지 어레이가 최적의 상태인지 확인합니다.
9. 클러스터를 유지보수 모드에서 해제하고 각 BeeGFS 서비스가 기본 노드에 있을 때 클러스터가 최적의 상태인지 확인합니다.

서비스 및 유지 관리

장애 조치 및 장애 복구 서비스

클러스터 노드 간에 BeeGFS 서비스 이동

개요

BeeGFS 서비스는 클러스터에서 노드 간에 페일오버를 수행하여 노드에 장애가 발생하거나 계획된 유지 관리를 수행해야 하는 경우 클라이언트가 파일 시스템에 계속 액세스할 수 있도록 합니다. 이 섹션에서는 장애를 복구한 후 관리자가 클러스터를 복구하거나 노드 간에 서비스를 수동으로 이동할 수 있는 다양한 방법에 대해 설명합니다.

단계

페일오버 및 페일백

페일오버(계획됨)

일반적으로 유지 관리를 위해 단일 파일 노드를 오프라인으로 전환해야 하는 경우 해당 노드에서 모든 BeeGFS 서비스를 이동(또는 드레이닝)해야 합니다. 먼저 노드를 대기 상태로 전환하여 이 작업을 수행할 수 있습니다.

```
pcs node standby <HOSTNAME>
```

를 사용하여 확인한 후 `pcs status` 모든 리소스가 대체 파일 노드에서 다시 시작되었습니다. 노드를 종료하거나 필요에 따라 변경할 수 있습니다.

페일백(계획된 페일오버 후)

BeeGFS 서비스를 기본 노드 첫 번째 실행으로 복구할 준비가 되면 `pcs status` "노드 목록"에서 상태가 대기 상태인지 확인합니다. 노드가 재부팅된 경우 클러스터 서비스를 온라인 상태로 전환할 때까지 오프라인 상태로 표시됩니다.

```
pcs cluster start <HOSTNAME>
```

노드가 온라인 상태가 되면 다음을 사용하여 대기 모드에서 나오게 합니다.

```
pcs node unstandby <HOSTNAME>
```

마지막으로 다음을 통해 모든 BeeGFS 서비스를 기본 노드에 다시 재배치하십시오.

```
pcs resource relocate run
```

페일백(계획되지 않은 페일오버 후)

노드에 하드웨어 또는 기타 장애가 발생할 경우 HA 클러스터가 자동으로 대응하고 서비스를 정상 노드로 이동하여 관리자에게 수정 조치를 취할 수 있는 시간을 제공해야 합니다. 계속하기 전에 "문제 해결"섹션을 참조하여 장애 조치의 원인을 확인하고 미해결 문제를 해결하십시오. 노드 전원이 다시 켜지고 정상 상태가 되면 페일백을 진행할 수 있습니다.

예정되지 않은(또는 계획된) 재부팅 후 노드가 부팅될 때 클러스터 서비스가 자동으로 시작되도록 설정되지 않으므로 먼저 를 사용하여 노드를 온라인 상태로 가져와야 합니다.

```
pcs cluster start <HOSTNAME>
```

그런 다음 리소스 장애를 정리하고 노드의 펜싱 기록을 재설정합니다.

```
pcs resource cleanup node=<HOSTNAME>
pcs stonith history cleanup <HOSTNAME>
```

에서 확인하십시오 `pcs status` 노드가 온라인 상태이고 정상 상태입니다. 기본적으로 BeeGFS 서비스는 실수로 리소스가 정상 상태가 아닌 노드로 다시 이동하는 것을 방지하기 위해 자동으로 페일백하지 않습니다. 준비되면 클러스터의 모든 리소스를 원하는 노드로 다시 돌려볼 수 있는 방법은 다음과 같습니다.

```
pcs resource relocate run
```

BeeGFS 서비스를 새 파일 노드로 영구적으로 이동합니다

개별 BeeGFS 서비스에 대해 선호하는 파일 노드를 영구적으로 변경하려면 선호하는 노드가 먼저 나열되도록 Ansible 인벤토리를 조정하고 Ansible 플레이북을 다시 실행하십시오.

예를 들어, 이 샘플 `inventory.yml` 파일에서 beegfs_01은 BeeGFS 관리 서비스를 실행하는 데 사용되는 파일 노드입니다.

```
        mgmt:
          hosts:
            beegfs_01:
            beegfs_02:
```

순서를 반대로 하면 beegfs_02에서 관리 서비스가 선호됩니다.

```
        mgmt:
          hosts:
            beegfs_02:
            beegfs_01:
```

BeeGFS 서비스를 대체 파일 노드로 임시 이동합니다

일반적으로 노드가 유지 관리 중인 경우 [failover and failback steps](#failover-and-failback)를 사용하여 해당 노드에서 모든 서비스를 이동하려고 합니다.

어떤 이유로 개별 서비스를 다른 파일 노드로 이동해야 하는 경우 다음을 실행합니다.

```
pcs resource move <SERVICE>-monitor <HOSTNAME>
```

> 개별 리소스 또는 리소스 그룹을 지정하지 마십시오. 재배치할 BeeGFS 서비스에 대한 모니터 이름을 항상 지정합니다. 예를 들어 BeeGFS 관리 서비스를 beegfs_02 실행으로 이동하려면 다음을 실행합니다 `pcs resource move mgmt-monitor beegfs_02`. 이 프로세스를 반복하여 하나 이상의 서비스를 원하는 노드에서 이동할 수 있습니다. 서비스를 사용하여 새 노드에서 재배치 /시작되었는지 `pcs status` 확인합니다.

BeeGFS 서비스를 기본 노드로 다시 이동하려면 먼저 임시 리소스 제약 조건을 해제합니다(여러 서비스에 필요한 경우 이 단계를 반복).

```
pcs resource clear <SERVICE>-monitor
```

그런 다음 실제로 서비스를 원하는 노드로 다시 이동할 준비가 되면 다음을 실행합니다.

```
pcs resource relocate run
```

참고 이 명령은 더 이상 임시 리소스 제약 조건이 없는 서비스를 기본 노드에 배치하지 않습니다.

클러스터를 유지보수 모드로 전환합니다

HA 클러스터가 운영 환경의 의도된 변경 사항에 실수로 반응하는 것을 방지합니다.

개요

클러스터를 유지보수 모드로 전환하면 모든 리소스 모니터링이 비활성화되고 박동기가 클러스터 리소스를 이동하거나 관리하는 것을 방지할 수 있습니다. 액세스를 방해하는 일시적인 장애 조건이 있더라도 모든 리소스는 원래 노드에서 계속 실행됩니다. 권장/유용한 시나리오는 다음과 같습니다.

- 파일 노드와 BeeGFS 서비스 간의 연결이 일시적으로 중단될 수 있는 네트워크 유지 보수
- 블록 노드 업그레이드
- 파일 노드 운영 체제, 커널 또는 기타 패키지 업데이트.

일반적으로 클러스터를 유지 관리 모드로 수동으로 설정하는 유일한 이유는 해당 클러스터가 환경의 외부 변경에 반응하지 않도록 하기 위해서입니다. 클러스터의 개별 노드에 물리적 복구가 필요한 경우 유지보수 모드를 사용하지 말고 위의 절차에 따라 해당 노드를 대기 모드에 두십시오. Ansible을 다시 실행하면 업그레이드 및 구성 변경을 포함하여 대부분의 소프트웨어 유지보수를 수행할 수 있도록 클러스터가 유지보수 모드로 자동으로 전환됩니다.

단계

클러스터가 유지보수 모드인지 확인하려면 다음을 실행합니다.

```
pcs property config
```

`maintenance-mode`클러스터가 정상적으로 작동하는 경우에는 속성이 나타나지 않습니다. 클러스터가 현재 유지 관리 모드에 있는 경우 속성은 로 보고됩니다. `true` 유지보수 모드를 활성화하려면 다음을 실행하십시오.

```
pcs property set maintenance-mode=true
```

PC 상태를 실행하고 모든 리소스에 "(관리되지 않음)"이 표시되는지 확인하여 확인할 수 있습니다. 클러스터를 유지보수 모드에서 제외하려면 다음을 실행하십시오.

```
pcs property set maintenance-mode=false
```

클러스터를 중지하고 시작합니다

정상적으로 HA 클러스터를 중지 및 시작합니다.

개요

이 섹션에서는 BeeGFS 클러스터를 정상적으로 종료하고 다시 시작하는 방법에 대해 설명합니다. 이러한 작업이 필요할 수 있는 시나리오에는 전기 유지보수 또는 데이터 센터 또는 랙 간 마이그레이션이 포함됩니다.

단계

어떤 이유로든 전체 BeeGFS 클러스터를 중지하고 모든 서비스를 종료해야 하는 경우 다음을 실행합니다.

```
pcs cluster stop --all
```

또한 개별 노드에서 클러스터를 중지할 수도 있습니다(다른 노드로 자동으로 서비스 페일오버). 먼저 노드를 대기 상태로 두는 것이 좋지만("페일오버"섹션 참조):

```
pcs cluster stop <HOSTNAME>
```

모든 노드에서 클러스터 서비스 및 리소스를 시작하려면 다음을 실행합니다.

```
pcs cluster start --all
```

또는 다음 특정 노드에서 서비스를 시작합니다.

```
pcs cluster start <HOSTNAME>
```

이 시점에서 를 실행합니다 `pcs status` 모든 노드에서 클러스터와 BeeGFS 서비스가 시작되는지, 그리고 원하는 노드에서 서비스가 실행되고 있는지 확인합니다.

클러스터 크기에 따라 전체 클러스터가 중지되거나 에서 시작된 것으로 표시되는 데 몇 초 또는 몇 분이 걸릴 수 `pcs status` 있습니다. 가 5분 이상 중단된 경우 `pcs cluster <COMMAND>` "Ctrl+C"를 실행하여 명령을 취소하기 전에 클러스터의 각 노드에 로그인하여 `pcs status` 클러스터 서비스(Corosync/Pacemaker)가 해당 노드에서 계속 실행되고 있는지 확인하십시오. 클러스터가 여전히 활성 상태인 모든 노드에서 클러스터를 차단하는 리소스를 확인할 수 있습니다. 문제를 수동으로 해결하고 명령을 완료하거나 다시 실행하여 나머지 서비스를 중지할 수 있습니다.

파일 노드를 바꿉니다

원래 서버에 오류가 있는 경우 파일 노드를 교체합니다.

개요

다음은 클러스터의 파일 노드를 교체하는 데 필요한 단계에 대한 개요입니다. 다음 단계에서는 하드웨어 문제로 인해 파일 노드가 실패했으며 동일한 새 파일 노드로 교체된다고 가정합니다.

단계:

1. 파일 노드를 물리적으로 교체하고 블록 노드 및 스토리지 네트워크에 대한 모든 케이블 연결을 복원합니다.
2. Red Hat 서브스크립션 추가를 포함하여 파일 노드에 운영 체제를 다시 설치합니다.
3. 파일 노드에서 관리 및 BMC 네트워킹을 구성합니다.
4. 호스트 이름, IP, PCIe-논리 인터페이스 매핑 또는 새 파일 노드에 대해 변경된 사항이 있는 경우 Ansible 인벤토리를 업데이트합니다. 일반적으로 노드가 동일한 서버 하드웨어로 교체되었고 원래 네트워크 구성을 사용하는 경우에는 필요하지 않습니다.
   a. 예를 들어 호스트 이름이 변경된 경우 노드의 인벤토리 파일을 생성하거나 이름을 변경합니다 (`host_vars/<NEW_NODE>.yml`)를 선택한 다음 Ansible 인벤토리 파일에 저장합니다 (`inventory.yml`)에서 이전 노드의 이름을 새 노드 이름으로 바꿉니다.

```
all:
    ...
    children:
    ha_cluster:
        children:
        mgmt:
            hosts:
            node_h1_new:    # Replaced "node_h1" with "node_h1_new"
            node_h2:
```

5. 클러스터의 다른 노드 중 하나에서 이전 노드를 제거합니다. `pcs cluster node remove <HOSTNAME>`.

   ⚠ 이 단계를 실행하기 전에 진행하지 마십시오.

6. Ansible 제어 노드에서:
   a. 다음을 사용하여 이전 SSH 키를 제거합니다.

```
`ssh-keygen -R <HOSTNAME_OR_IP>`
```

   b. 바꾸기 노드에 대해 암호 없는 SSH를 다음으로 구성:

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

7. Ansible 플레이북을 다시 실행하여 노드를 구성하고 클러스터에 추가합니다.

```
ansible-playbook -i <inventory>.yml <playbook>.yml
```

8. 이때 를 실행합니다 `pcs status` 교체된 노드가 나열되고 서비스가 실행 중인지 확인합니다.

클러스터를 확장 또는 축소합니다

클러스터에서 구성 요소를 추가하거나 제거합니다.

개요

이 섹션에서는 BeeGFS HA 클러스터의 크기를 조정하는 다양한 고려 사항 및 옵션에 대해 설명합니다. 일반적으로 클러스터 크기는 구성 요소를 추가 또는 제거하여 조정합니다. 구성 요소는 일반적으로 2개의 파일 노드를 HA 쌍으로 설정합니다. 필요한 경우 개별 파일 노드(또는 다른 유형의 클러스터 노드)를 추가하거나 제거할 수도 있습니다.

클러스터에 빌딩 블록 추가

고려 사항

추가 구성 요소를 추가하여 클러스터를 늘리는 것은 간단한 프로세스입니다. 시작하기 전에 각 개별 HA 클러스터의 최소 및 최대 클러스터 노드 수에 대한 제한을 염두에 두고, 기존 HA 클러스터에 노드를 추가하거나 새로운 HA 클러스터를 생성해야 하는지 확인합니다. 일반적으로 각 구성 요소는 2개의 파일 노드로 구성되지만, 3개의 노드는 쿼럼(quorum)을 설정하기 위해 클러스터당 최소 노드 수이고 10개는 권장(테스트)입니다. 고급 시나리오의 경우 2노드 클러스터를 구축할 때 BeeGFS 서비스를 실행하지 않는 단일 "Tiebreaker" 노드를 추가할 수 있습니다. 이러한 배포를 고려 중인 경우 NetApp 지원에 문의하십시오.

클러스터를 확장하는 방법을 결정할 때는 이러한 제한 사항과 향후 클러스터 확장에 대한 예상에 유의하십시오. 예를 들어, 6노드 클러스터가 있고 4노드를 더 추가해야 하는 경우 새 HA 클러스터를 시작하는 것이 좋습니다.

단일 BeeGFS 파일 시스템은 여러 독립 HA 클러스터로 구성될 수 있습니다. 따라서 파일 시스템은 기본 HA 클러스터 구성 요소의 권장/하드 제한보다 훨씬 높은 확장성을 유지할 수 있습니다.

단계

구성 요소를 클러스터에 추가할 때 `host_vars` 각 새 파일 노드 및 블록 노드(E-Series 스토리지)에 대한 파일을 생성해야 합니다. 이러한 호스트의 이름은 생성할 새 리소스와 함께 인벤토리에 추가해야 합니다. `group_vars`각 새 리소스에 대해 해당 파일을 생성해야 합니다. "사용자 지정 아키텍처 사용"자세한 내용은 섹션을 참조하십시오.

올바른 파일을 생성한 후 다음 명령을 사용하여 자동화를 다시 실행해야 합니다.

```
ansible-playbook -i <inventory>.yml <playbook>.yml
```

클러스터에서 빌딩 블록 제거

구성 요소를 폐기해야 할 경우에는 다음과 같은 여러 가지 사항을 고려해야 합니다.

- 이 빌딩 블록에서 실행 중인 BeeGFS 서비스는 무엇입니까?
- 파일 노드만 사용 중지되며 블록 노드를 새 파일 노드에 연결해야 합니까?

- 전체 빌딩 블록이 사용 중단된 경우 데이터를 새 빌딩 블록으로 이동하거나, 클러스터의 기존 노드로 분산하거나, 새로운 BeeGFS 파일 시스템 또는 다른 스토리지 시스템으로 이동해야 합니까?
- 이 문제는 중단 중에 발생할 수 있습니까? 아니면 중단 없이 수행해야 합니까?
- 구성 요소를 적극적으로 사용하고 있습니까, 아니면 주로 더 이상 활성 상태가 아니지 않은 데이터가 포함되어 있습니까?

가능한 다양한 시작 지점과 원하는 종료 상태 때문에 NetApp 지원에 문의하여 고객 환경과 요구 사항에 따라 최상의 전략을 파악하고 구현할 수 있습니다.

문제 해결

BeeGFS HA 클러스터 문제 해결

개요

이 섹션에서는 BeeGFS HA 클러스터를 운영할 때 발생할 수 있는 다양한 장애 및 기타 시나리오를 조사하고 해결하는 방법을 설명합니다.

문제 해결 설명서

예상치 못한 장애 조치 조사

노드가 예기치 않게 펜싱되고 해당 서비스가 다른 노드로 이동된 경우 첫 번째 단계는 클러스터가 하단에 리소스 장애를 나타내는지 확인하는 것입니다 `pcs status`. 펜싱이 성공적으로 완료되고 리소스가 다른 노드에서 다시 시작된 경우에는 일반적으로 아무 것도 표시되지 않습니다.

일반적으로 다음 단계는 를 사용하여 시스템 로그를 검색하는 것입니다 `journalctl` 나머지 파일 노드 중 하나에서(심장박동기 로그는 모든 노드에서 동기화됨) 오류가 발생한 시간을 알고 있는 경우 장애가 발생하기 바로 직전에 검색을 시작할 수 있습니다(일반적으로 10분 이상 전에 검색을 시작하는 것이 좋습니다).

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>"
```

다음 섹션에서는 조사 범위를 더욱 좁히기 위해 로그에 표시할 수 있는 일반적인 텍스트를 보여 줍니다.

조사/해결 단계

1단계: **BeeGFS** 모니터에서 장애를 감지했는지 확인합니다**.**

BeeGFS 모니터에 의해 페일오버가 트리거된 경우 오류가 표시됩니다(다음 단계로 진행되지 않는 경우).

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>" | grep -i unexpected
[...]
Jul 01 15:51:03 beegfs_01 pacemaker-schedulerd[9246]:  warning: Unexpected
result (error: BeeGFS service is not active!) was recorded for monitor of
meta_08-monitor on beegfs_02 at Jul  1 15:51:03 2022
```

이 경우 BeeGFS 서비스 META_08이 어떤 이유로 중지되었습니다. 문제 해결을 계속하려면 beegfs_02를 부팅하고 에서 서비스에 대한 로그를 검토해야 합니다 `/var/log/beegfs-meta-meta_08_tgt_0801.log`. 예를 들어, BeeGFS 서비스에서 노드의 내부 문제 또는 문제로 인해 애플리케이션 오류가 발생했을 수 있습니다.

페이스 메이커의 로그와 달리 BeeGFS 서비스의 로그는 클러스터의 모든 노드에 배포되지 않습니다. 이러한 유형의 장애를 조사하려면 장애가 발생한 원래 노드의 로그가 필요합니다.

모니터에서 보고할 수 있는 문제는 다음과 같습니다.

- 대상에 액세스할 수 없습니다!
  - 설명: 블록 볼륨을 액세스할 수 없음을 나타냅니다.
  - 문제 해결:
    - 대체 파일 노드에서 서비스도 시작하지 못한 경우 블록 노드가 정상 상태인지 확인합니다.
    - 이 파일 노드에서 블록 노드에 액세스하지 못하게 하는 물리적 문제(예: InfiniBand 어댑터 또는 케이블 장애)가 있는지 확인합니다.
- 네트워크에 연결할 수 없습니다!
  - 설명: 클라이언트가 이 BeeGFS 서비스에 연결하는 데 사용하는 어댑터 중 온라인 어댑터가 없습니다.
  - 문제 해결:
    - 여러 파일 노드/모든 파일 노드에 영향을 받은 경우 BeeGFS 클라이언트 및 파일 시스템을 연결하는 데 사용된 네트워크에 장애가 있는지 확인합니다.
    - 이 파일 노드에서 클라이언트에 액세스하지 못하게 하는 물리적 문제(예: InfiniBand 어댑터 또는 케이블 장애)가 있는지 확인합니다.
- BeeGFS 서비스가 활성 상태가 아닙니다.
  - 설명: BeeGFS 서비스가 예기치 않게 중지되었습니다.
  - 문제 해결:
    - 오류를 보고한 파일 노드에서 영향을 받는 BeeGFS 서비스의 로그를 확인하여 충돌이 보고되었는지 확인합니다. 이 경우 NetApp Support에서 케이스를 접수하여 충돌을 조사하십시오.
    - BeeGFS 로그에 보고된 오류가 없는 경우 저널 로그를 확인하여 시스템이 서비스가 중지된 이유를 기록했는지 확인합니다. 일부 시나리오에서 BeeGFS 서비스는 프로세스가 종료되기 전에(예: 누군가 를 실행한 경우) 메시지를 기록할 수 있는 기회를 제공하지 않았을 수 있습니다 `kill -9 <PID>`)를 클릭합니다.

2단계**:** 노드가 예기치 않게 클러스터를 종료했는지 확인합니다

노드에 심각한 하드웨어 장애가 발생하거나(예: 시스템 보드가 작동하지 않음) 커널 패닉 또는 유사한 소프트웨어 문제가 발생한 경우 BeeGFS 모니터에서 오류를 보고하지 않습니다. 대신 호스트 이름을 찾으십시오. 심장박동기에서 노드가 예기치 않게 손실되었음을 나타내는 메시지가 표시됩니다.

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>" | grep -i <HOSTNAME>
[...]
Jul 01 16:18:01 beegfs_01 pacemaker-attrd[9245]:  notice: Node beegfs_02
state is now lost
Jul 01 16:18:01 beegfs_01 pacemaker-controld[9247]:  warning:
Stonith/shutdown of node beegfs_02 was not expected
```

3단계: 심장박동기가 노드를 울타리로 만들 수 있는지 확인합니다

모든 시나리오에서 노드가 실제로 오프라인 상태인지 확인하기 위해 심장박동기 펜스(pencing)를 시도하는 것을 볼 수 있습니다(정확한 메시지는 펜싱 원인에 따라 다를 수 있음).

```
Jul 01 16:18:02 beegfs_01 pacemaker-schedulerd[9246]:  warning: Cluster
node beegfs_02 will be fenced: peer is no longer part of the cluster
Jul 01 16:18:02 beegfs_01 pacemaker-schedulerd[9246]:  warning: Node
beegfs_02 is unclean
Jul 01 16:18:02 beegfs_01 pacemaker-schedulerd[9246]:  warning: Scheduling
Node beegfs_02 for STONITH
```

펜싱 작업이 성공적으로 완료되면 다음과 같은 메시지가 표시됩니다.

```
Jul 01 16:18:14 beegfs_01 pacemaker-fenced[9243]:  notice: Operation 'off'
[2214070] (call 27 from pacemaker-controld.9247) for host 'beegfs_02' with
device 'fence_redfish_2' returned: 0 (OK)
Jul 01 16:18:14 beegfs_01 pacemaker-fenced[9243]:  notice: Operation 'off'
targeting beegfs_02 on beegfs_01 for pacemaker-
controld.9247@beegfs_01.786df3a1: OK
Jul 01 16:18:14 beegfs_01 pacemaker-controld[9247]:  notice: Peer
beegfs_02 was terminated (off) by beegfs_01 on behalf of pacemaker-
controld.9247: OK
```

어떤 이유로 펜싱 작업이 실패한 경우 데이터 손상을 방지하기 위해 다른 노드에서 BeeGFS 서비스를 다시 시작할 수 없습니다. 예를 들어 펜싱 장치(PDU 또는 BMC)에 액세스할 수 없거나 잘못 구성된 경우 별도로 조사하는 것이 문제입니다.

실패한 리소스 작업 해결**(PCS** 상태 하단에 있음**)**

BeeGFS 서비스를 실행하는 데 필요한 리소스에 장애가 발생하면 BeeGFS 모니터에서 페일오버가 트리거됩니다. 이 경우 의 하단에 "실패한 리소스 작업"이 나열되지 않을 수 `pcs status` 있으므로 방법 단계를 참조해야 "계획되지 않은 페일오버 후 페일백"합니다.

그렇지 않으면 일반적으로 "실패한 리소스 작업"이 표시되는 두 가지 시나리오만 있어야 합니다.

시나리오 **1:** 펜싱 에이전트에서 일시적 또는 영구적인 문제가 감지되어 이를 다시 시작하거나 다른 노드로 이동했습니다**.**

일부 펜싱 에이전트는 다른 펜싱 장치보다 신뢰성이 높으며 각 펜싱 장치가 준비되도록 자체 모니터링 방법을 구현합니다. 특히 Redfish 펜싱 에이전트가 여전히 started로 표시되더라도 다음과 같은 실패한 리소스 작업을 보고하는 것으로 나타났습니다.

```
  * fence_redfish_2_monitor_60000 on beegfs_01 'not running' (7):
call=2248, status='complete', exitreason='', last-rc-change='2022-07-26
08:12:59 -05:00', queued=0ms, exec=0ms
```

특정 노드에서 장애가 발생한 리소스 작업을 보고하는 펜싱 에이전트가 해당 노드에서 실행되는 BeeGFS 서비스의 페일오버를 트리거하지 않습니다. 동일한 노드 또는 다른 노드에서 자동으로 다시 시작하기만 하면 됩니다.

해결 단계:

1. 펜싱 에이전트가 노드 전체 또는 하위 집합에서 지속적으로 실행을 거부하는 경우 해당 노드가 펜싱 에이전트에 연결할 수 있는지 확인하고 펜싱 에이전트가 Ansible 인벤토리에서 올바르게 구성되었는지 확인합니다.
   a. 예를 들어, BMC(Redfish) 펜싱 에이전트가 펜싱을 담당하는 동일한 노드에서 실행되고 있고 OS 관리 및 BMC IP가 동일한 물리적 인터페이스에 있는 경우 일부 네트워크 스위치 구성에서는 두 인터페이스 간의 통신을 허용하지 않습니다(네트워크 루프 방지). 기본적으로 HA 클러스터는 펜싱을 담당하는 노드에 펜싱 에이전트를 배치하는 것을 피하려고 하지만 일부 시나리오/구성에서는 이러한 문제가 발생할 수 있습니다.
2. 모든 문제가 해결되거나 문제가 일시적인 것으로 나타나는 경우 를 실행합니다 `pcs resource cleanup` 실패한 리소스 작업을 재설정합니다.

시나리오 **2: BeeGFS** 모니터가 문제를 감지하여 페일오버를 트리거했지만**,** 어떤 이유로 보조 노드에서 리소스를 시작할 수 없습니다**.**

펜싱이 활성화되고 리소스가 원래 노드에서 정지하는 것을 차단하지 않은 경우("대기(장애 발생 시)"의 문제 해결 섹션 참조), 보조 노드에서 리소스를 시작하는 데 다음과 같은 문제가 원인일 수 있습니다.

- 보조 노드가 이미 오프라인 상태입니다.
- 물리적 또는 논리적 구성 문제로 인해 보조 시스템에서 BeeGFS 타겟으로 사용되는 블록 볼륨에 액세스하지 못했습니다.

해결 단계:

1. 실패한 리소스 작업의 각 항목에 대해 다음을 수행합니다.
   a. 실패한 리소스 작업이 시작 작업인지 확인합니다.
   b. 표시된 리소스와 실패한 리소스 작업에 지정된 노드를 기반으로 합니다.
      i. 노드가 지정된 리소스를 시작하지 못하는 외부 문제를 찾아 해결합니다. 예를 들어 BeeGFS IP 주소(부동 IP)를 시작하지 못한 경우 필요한 인터페이스 중 하나 이상이 온라인으로 연결되어 있고 올바른 네트워크 스위치에 케이블로 연결되어 있는지 확인합니다. BeeGFS 타겟(블록 디바이스/E-Series 볼륨)에 장애가 발생한 경우 백엔드 블록 노드에 대한 물리적 접속이 예상대로 접속되어 있는지 확인하고 블록 노드가 정상 상태인지 확인합니다.
   c. 명확한 외부 문제가 없고 이 인시던트에 대한 근본 원인이 필요한 경우, 다음 단계로 인해 근본 원인 분석

(RCA)이 어렵거나 불가능할 수 있으므로 계속하기 전에 NetApp Support에서 케이스를 열어 조사하는 것이 좋습니다.

2. 외부 문제 해결 후:
   a. Anabilities inventory.yml 파일에서 작동하지 않는 노드를 모두 제거하고 전체 Ansible 플레이북을 다시 실행하여 모든 논리적 구성이 보조 노드에 올바르게 설정되었는지 확인합니다.
      i. 참고: 노드 상태가 양호하고 페일백할 준비가 되면 이러한 노드의 주석을 해제하고 플레이북을 다시 실행하십시오.
   b. 또는 클러스터를 수동으로 복구할 수도 있습니다.
      i. 다음을 사용하여 오프라인 노드를 다시 온라인 상태로 전환: `pcs cluster start <HOSTNAME>`
      ii. 다음을 사용하여 실패한 모든 리소스 작업을 지웁니다. `pcs resource cleanup`
      iii. PCS 상태를 실행하고 모든 서비스가 예상대로 시작되는지 확인합니다.
      iv. 필요한 경우 실행합니다 `pcs resource relocate run` 리소스를 원하는 노드로 다시 이동하려면 (사용 가능한 경우)

일반적인 문제

BeeGFS 서비스는 요청 시 페일오버 또는 페일백을 수행하지 않습니다

- 가능성 높은 문제: * `pcs resource relocate` 실행 명령이 실행되었지만 성공적으로 완료되지 않았습니다.
- 확인 방법: * 실행 `pcs constraint --full` ID가 인 위치 제약 조건이 있는지 확인합니다 `pcs-relocate-<RESOURCE>`.
- 해결 방법: * 실행 `pcs resource relocate clear` 그런 다음 다시 실행합니다 `pcs constraint --full` 추가 구속조건이 제거되었는지 확인합니다.

펜싱이 비활성화된 경우 **PCS** 상태의 노드 중 하나에 **"STANDBY(ON-FAIL)"**가 표시됩니다

- 가능성 높은 문제: * 심장박동기가 실패한 노드에서 모든 리소스가 중지되었는지 확인할 수 없습니다.
- 해결 방법: *
  1. 실행 `pcs status` 그리고 출력 하단에 "시작"되지 않은 리소스 또는 오류가 표시되는지 확인하고 모든 문제를 해결합니다.
  2. 노드를 다시 온라인 상태로 전환하려면 다음을 수행합니다 `pcs resource cleanup --node=<HOSTNAME>`.

예기치 않은 장애 조치 후 펜싱이 활성화된 경우 **PCS** 상태에 **"started (on-fail)"**가 표시됩니다

- 가능성 높은 문제: * 장애 조치를 트리거한 문제가 발생했지만 심장박동기가 노드 펜싱되었는지 확인할 수 없었습니다. 펜싱이 잘못 구성되었거나 펜싱 에이전트(예: 네트워크에서 PDU 연결이 끊어짐)에 문제가 있기 때문에 이 문제가 발생할 수 있습니다.
- 해결 방법: *
  1. 노드의 전원이 실제로 꺼져 있는지 확인합니다.

지정하는 노드가 실제로 꺼져 있지 않지만 클러스터 서비스 또는 리소스를 실행하는 경우 데이터 손상/클러스터 장애가 발생합니다.

2. 다음을 사용하여 펜싱을 수동으로 확인합니다. `pcs stonith confirm <NODE>`

이 시점에서 서비스는 장애 조치를 완료하고 다른 정상 노드에서 다시 시작해야 합니다.

일반적인 문제 해결 작업

개별 **BeeGFS** 서비스를 다시 시작합니다

일반적으로 BeeGFS 서비스를 다시 시작해야 하는 경우(예: 구성 변경을 용이하게 함) Ansible 인벤토리를 업데이트하고 플레이북을 다시 실행하여 이 작업을 수행해야 합니다. 경우에 따라 전체 Playbook을 실행할 때까지 기다릴 필요 없이 로깅 수준을 변경하는 등 더 빠른 문제 해결을 위해 개별 서비스를 다시 시작하는 것이 좋습니다.

수동 변경 사항도 Ansible 인벤토리에 추가되지 않으면 다음 번에 Ansible 플레이북을 실행할 때 되돌릴 수 있습니다.

옵션 **1:** 시스템 **d**가 재시작을 제어했습니다

BeeGFS 서비스가 새 구성으로 제대로 재시작되지 않을 위험이 있는 경우 먼저 클러스터를 유지 관리 모드로 전환하여 BeeGFS 모니터가 서비스를 감지하지 못하게 하고 원치 않는 페일오버를 트리거하는 것을 방지하십시오.

```
pcs property set maintenance-mode=true
```

필요한 경우 에서 서비스 구성을 변경합니다 `/mnt/<SERVICE_ID>/_config/beegfs-.conf` (예: `/mnt/meta_01_tgt_0101/metadata_config/beegfs-meta.conf`) 그런 다음 systemd를 사용하여 다시 시작합니다.

```
systemctl restart beegfs-*@<SERVICE_ID>.service
```

예: `systemctl restart beegfs-meta@meta_01_tgt_0101.service`

옵션 **2:** 심장박동기 제어 재시작

새로운 구성으로 인해 서비스가 예기치 않게 중지되거나(예: 로깅 수준 변경) 유지 보수 기간에 있고 다운타임이 염려되지 않는 경우 다시 시작할 서비스에 대해 BeeGFS 모니터를 다시 시작하면 됩니다.

```
pcs resource restart <SERVICE>-monitor
```

예를 들어 BeeGFS 관리 서비스를 다시 시작하려면 `pcs resource restart mgmt-monitor`

법적 고지

법적 고지 사항은 저작권 선언, 상표, 특허 등에 대한 액세스를 제공합니다.

저작권

"https://www.netapp.com/company/legal/copyright/"

상표

NetApp, NetApp 로고, NetApp 상표 페이지에 나열된 마크는 NetApp Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.

"https://www.netapp.com/company/legal/trademarks/"

특허

NetApp 소유 특허 목록은 다음 사이트에서 확인할 수 있습니다.

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

개인 정보 보호 정책

"https://www.netapp.com/company/legal/privacy-policy/"

오픈 소스

통지 파일은 NetApp 소프트웨어에 사용된 타사의 저작권 및 라이센스에 대한 정보를 제공합니다.

"E-Series/EF-Series SANtricity OS에 대한 알림"