# Eシリーズストレージを使用したネットアップのBeeGFS

## BeeGFS on NetApp with E-Series Storage

NetApp
January 27, 2026

# 目次

# Eシリーズストレージを使用したネットアップのBeeGFS

# はじめに

## このサイトに含まれている情報

このサイトには、NetApp Verified Architectures（NVA）とカスタムアーキテクチャの両方を使用して、ネットアップのBeeGFSを導入して管理する方法が記載されています。NVA設計は徹底的にテストされており、導入リスクを最小限に抑え、製品化サイクルを短縮するためのリファレンス構成とサイジングに関するガイダンスをお客様に提供します。また、ネットアップのハードウェアで実行されるカスタムのBeeGFSアーキテクチャもサポートするため、お客様やパートナーはさまざまな要件に合わせてファイルシステムを柔軟に設計できます。どちらのアプローチも導入にAnsibleを活用しているため、柔軟なハードウェア範囲であらゆる規模のBeeGFSを管理するアプライアンスのようなアプローチを提供します。

## 用語と概念

以下の用語と概念は、NetApp解決策 のBeeGFSに適用されます。

"BeeGFSクラスタの管理"BeeGFSハイアベイラビリティ（HA）クラスタとのやり取りに固有の用語や概念の詳細については、を参照してください。

| 期間 | 説明 |
| --- | --- |
| AI | 人工知能 |
| Ansibleコントロールノード | Ansible CLIの実行に使用する物理マシンまたは仮想マシン。 |
| Ansible のインベントリ | 目的のBeeGFS HAクラスタを記述するYAMLファイルを含むディレクトリ構造。 |
| BMC の場合 | ベースボード管理コントローラ：サービスプロセッサと呼ばれることもあります。 |
| ブロックノード | Eシリーズストレージシステム： |
| クライアント | HPCクラスタ内のノードで、ファイルシステムを利用する必要のあるアプリケーションを実行しています。コンピューティングノードまたはGPUノードと呼ぶこともあります。 |
| DL | ディープラーニング。 |
| ファイルノード | BeeGFSファイルサーバ： |

| 期間 | 説明 |
| --- | --- |
| 高可用性 | 高可用性： |
| HIC | ホストインターフェイスカード。 |
| HPC | ハイパフォーマンスコンピューティング。 |
| HPCスタイルのワークロード | HPCスタイルのワークロードの特徴は、通常、複数のコンピューティングノードまたはGPUがあり、すべてが同じデータセットに並行してアクセスする必要があることです。これにより、分散型のコンピューティングジョブやトレーニングジョブを円滑に進めることができます。多くの場合、このデータセットは大容量ファイルで構成されており、複数の物理ストレージノードにまたがってストライピングする必要があります。これにより、従来のハードウェアのボトルネックによって、単一ファイルへの同時アクセスが妨げられることがなくなります。 |
| ml | 機械学習。 |
| NLP | 自然言語処理。 |
| NLU | 自然言語の理解。 |
| NVA | NetApp Verified Architecture（NVA）プログラムは、特定のワークロードとユースケースに対するリファレンス構成とサイジングに関するガイダンスを提供します。これらのソリューションは徹底的にテストされており、導入リスクを最小限に抑え、製品化サイクルを短縮するように設計されています。 |
| ストレージネットワーク/クライアントネットワーク | クライアントがBeeGFSファイルシステムと通信するためのネットワーク。このネットワークは、多くの場合、並列Message Passing Interface（MPI；メッセージ転送インターフェイス）やHPCクラスタノード間のその他のアプリケーション通信に使用されるネットワークと同じです。 |

# 検証済みアーキテクチャを使用

## 概要と要件

### 解決策の概要

NetApp解決策 のBeeGFSは、BeeGFS並列ファイルシステムとNetApp EF600ストレージシステムを組み合わせることで、信頼性と拡張性に優れた対費用効果の高いインフラを実現し、要件の厳しいワークロードに対応します。

#### NVAプログラム

NetApp解決策 上のBeeGFSは、NetApp Verified Architecture（NVA）プログラムの一部であり、特定のワークロードとユースケースについて、参考構成とサイジングに関するガイダンスを提供します。NVAソリューションは、導入リスクを最小限に抑え、製品化サイクルを短縮するように徹底的にテストと設計されています。

#### 設計の概要

BeeGFS on NetAppソリューションは、拡張性に優れたビルディングブロックアーキテクチャとして設計されており、要件の厳しいさまざまなワークロード向けに構成可能です。多数の小さなファイルを扱う場合でも、大規模なファイル操作を管理する場合でも、ハイブリッドワークロードの場合でも、これらのニーズに合わせてファイルシステムをカスタマイズできます。設計には高可用性が組み込まれており、2層のハードウェア構造を使用しています。これにより、複数のハードウェアレイヤで独立したフェイルオーバーが可能になり、システムが部分的に低下しても一貫したパフォーマンスが保証されます。BeeGFSファイルシステムは、さまざまなLinuxディストリビューションにわたってハイパフォーマンスで拡張性に優れた環境を実現し、クライアントにアクセスしやすい単一のストレージネームスペースを提供します。詳細については、を "アーキテクチャの概要"参照してください。

#### ユースケース

以下のユースケースは、NetApp解決策 のBeeGFSに適用されます。

- NVIDIA DGX SuperPODシステムには、A100、H100、H200、B200 GPU搭載のDGXが搭載されています。
- 人工知能（AI）（機械学習（ML）、ディープラーニング（DL）、大規模な自然言語処理（NLP）、自然言語理解（NLU）など）詳細については、を参照してください "BeeGFS for AI：事実とフィクション"。
- MPI（メッセージ・パッシング・インターフェイス）やその他の分散コンピューティング技術により高速化されたアプリケーションを含む、ハイパフォーマンス・コンピューティング（HPC）。詳細については、を参照してください "BeeGFSがHPCの枠を超えている理由"。
- 次の特徴を持つアプリケーションワークロード：
  - 1GBを超えるファイルの読み取りまたは書き込み
  - 複数のクライアント（10s、100s、1000s）による同じファイルの読み取りと書き込み
- 数テラバイトまたは数ペタバイトのデータセット。
- 単一のストレージネームスペースが必要な環境：大容量ファイルと小容量ファイルを混在させる場合に最適化可能です。

利点

ネットアップでBeeGFSを使用する主なメリットは次のとおりです。

- 検証済みハードウェア設計の可用性：ハードウェアとソフトウェアコンポーネントを完全に統合し、予測可能なパフォーマンスと信頼性を確保します。
- Ansibleを使用して導入と管理を行い、シンプルさと大規模な一貫性を実現します。
- EシリーズPerformance AnalyzerおよびBeeGFSプラグインを使用した監視と監視が可能です。詳細については、を参照してください "NetApp Eシリーズソリューション監視フレームワークのご紹介"。
- データの保持と可用性を提供する共有ディスクアーキテクチャを採用した高可用性。
- コンテナとKubernetesを使用した最新のワークロード管理とオーケストレーションをサポート詳細については、を参照してください "Kubernetes BeeGFSを導入すれば、将来のニーズにも対応できる投資が実現します"。

## アーキテクチャの概要

### NetApp解決策 のBeeGFSには、検証済みのワークロードをサポートするために必要な機器、ケーブル配線、構成を決定するためのアーキテクチャ設計に関する考慮事項が含まれます。

ビルディングブロックアーキテクチャ

BeeGFSファイルシステムは、ストレージ要件に応じてさまざまな方法で導入および拡張できます。たとえば、主に多数の小さなファイルを扱うユースケースでは、メタデータのパフォーマンスと容量を強化できますが、大容量ファイルが少ないユースケースでは、実際のファイル内容よりも多くのストレージ容量とパフォーマンスを優先的に使用できます。このような複数の考慮事項は並列ファイルシステム環境のさまざまな次元に影響するため、ファイルシステムの設計と導入が複雑になります。

このような課題に対応するために、ネットアップでは、これらの要素のそれぞれをスケールアウトするための標準的なビルディングブロックアーキテクチャを設計しました。通常、BeeGFSビルディングブロックは、次の3つの設定プロファイルのいずれかに配置されます。

- BeeGFSの管理、メタデータ、ストレージサービスなど、単一のベースとなるビルディングブロックです
- BeeGFSメタデータとストレージビルディングブロック
- BeeGFSストレージのみのビルディングブロック

これらの3つのオプション間のハードウェア変更は、BeeGFSメタデータに小さいドライブを使用することだけです。それ以外の場合は、すべての設定変更がソフトウェアを介して適用されます。また、導入エンジンとしてAnsibleを使用することで、特定のビルディングブロックに必要なプロファイルを設定することで、構成タスクを簡単に実行できます。

詳細については、を参照してください [ハードウェアの設計を確認した]。

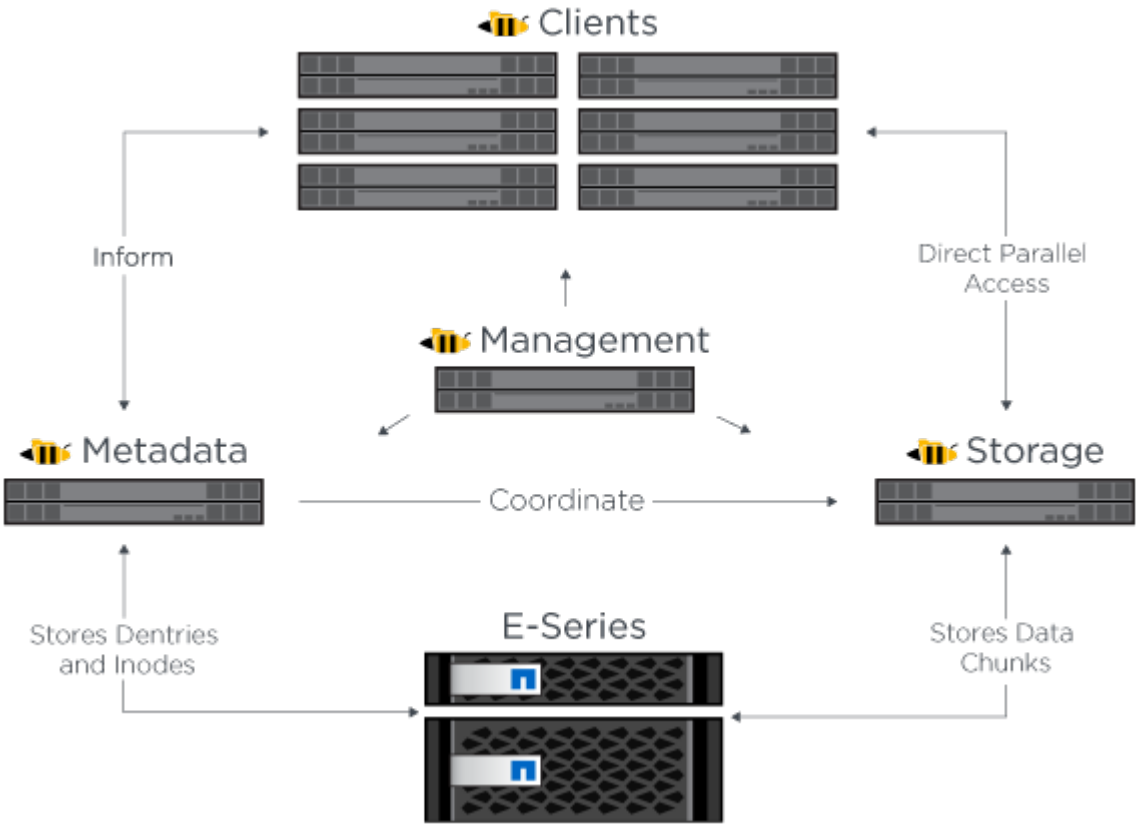
ファイルシステムサービス

BeeGFSファイルシステムには、次の主要サービスが含まれます。

- *管理サービス。*その他すべてのサービスを登録および監視します。

- *ストレージ・サービス。*データ・チャンク・ファイルと呼ばれる分散ユーザー・ファイルの内容を保存します。
- *メタデータサービス。*ファイルシステムのレイアウト、ディレクトリ、ファイル属性などを追跡します。
- *クライアント・サービス。*保存されたデータにアクセスするためのファイル・システムをマウントします。

次の図は、NetApp Eシリーズシステムで使用されるBeeGFS解決策 のコンポーネントと関係を示しています。




BeeGFSは、並列ファイルシステムとして、複数のサーバノードを介してファイルをストライプ化することで、読み取り/書き込みのパフォーマンスと拡張性を最大化します。サーバノードは連携して動作するため、ほかのサーバノード（一般に_clients__）から同時にマウントしてアクセスすることができる単一のファイルシステムを提供します。これらのクライアントは、分散ファイルシステムをNTFS、XFS、ext4などのローカルファイルシステムと同様に認識して使用できます。

これら4つの主要サービスは、サポートされている幅広いLinuxディストリビューションで動作し、InfiniBand（IB）、Omni-Path（OPA）、RDMA over Converged Ethernet（RoCE）など、すべてのTCP/IPまたはRDMA対応ネットワークを介して通信します。BeeGFSサーバサービス（管理'ストレージ'メタデータ）はユーザ空間デーモンであり'クライアントはネイティブカーネルモジュール（パッチレス）ですすべてのコンポーネントは、リブートせずにインストールまたは更新でき、同じノード上で任意の組み合わせのサービスを実行できます。

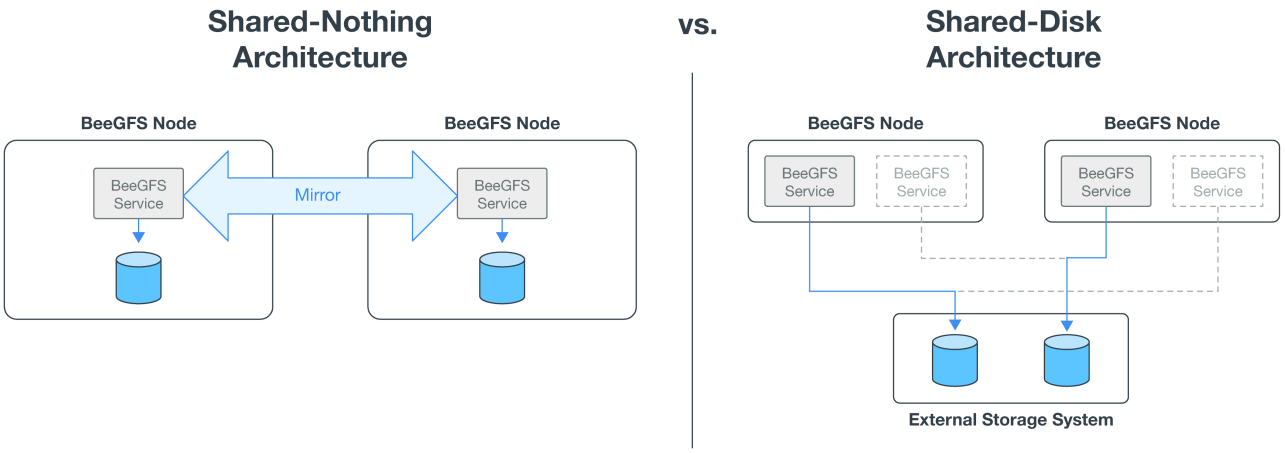
**HA**アーキテクチャ

BeeGFS on NetAppは、共有ディスクハイアベイラビリティ（HA）アーキテクチャを実現するネットアップハードウェアと完全に統合された解決策 を作成することで、BeeGFSエンタープライズエディションの機能

を拡張します。

> BeeGFSコミュニティエディションは無料でご利用いただけますが、エンタープライズエディションにはネットアップなどのパートナーからプロフェッショナルサポートサブスクリプション契約を購入する必要があります。エンタープライズエディションでは、耐障害性、クォータの適用、ストレージプールなど、いくつかの追加機能を使用できます。

次の図は、シェアードナッシングおよび共有ディスクHAアーキテクチャの比較です。




詳細については、を参照してください "ネットアップがサポートするBeeGFSの高可用性についてお知らせします"。

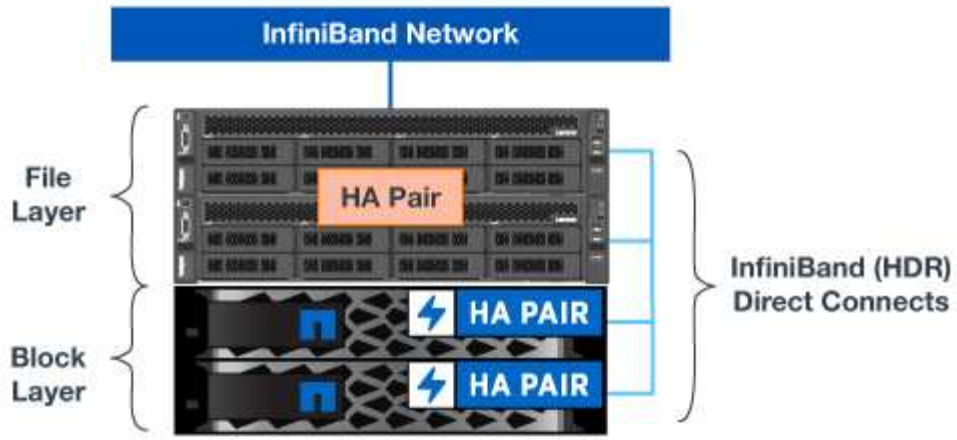
ノードを確認しました

BeeGFS on NetAppソリューションでは、以下のノードが検証されました。

| ノード | ハードウェア | 詳細 |
| --- | --- | --- |
| ブロック | NetApp EF600ストレージシステム | 要件の厳しいワークロード向けに設計された、ハイパフォーマンスなオールNVMe 2Uストレージアレイです。 |
| ファイル | Lenovo ThinkSystem SR665 V3サーバ | PCIe 5.0デュアルAMD EPYC 9124プロセッサを搭載した2ソケット2Uサーバ。Lenovo SR665 V3の詳細については、を参照してください "LenovoのWebサイト"。 |
| | Lenovo ThinkSystem SR665サーバ | PCIe 4.0、デュアルAMD EPYC 7003プロセッサを搭載した2ソケット2Uサーバ。Lenovo SR665の詳細については、を参照してください "LenovoのWebサイト"。 |

ハードウェアの設計を確認した

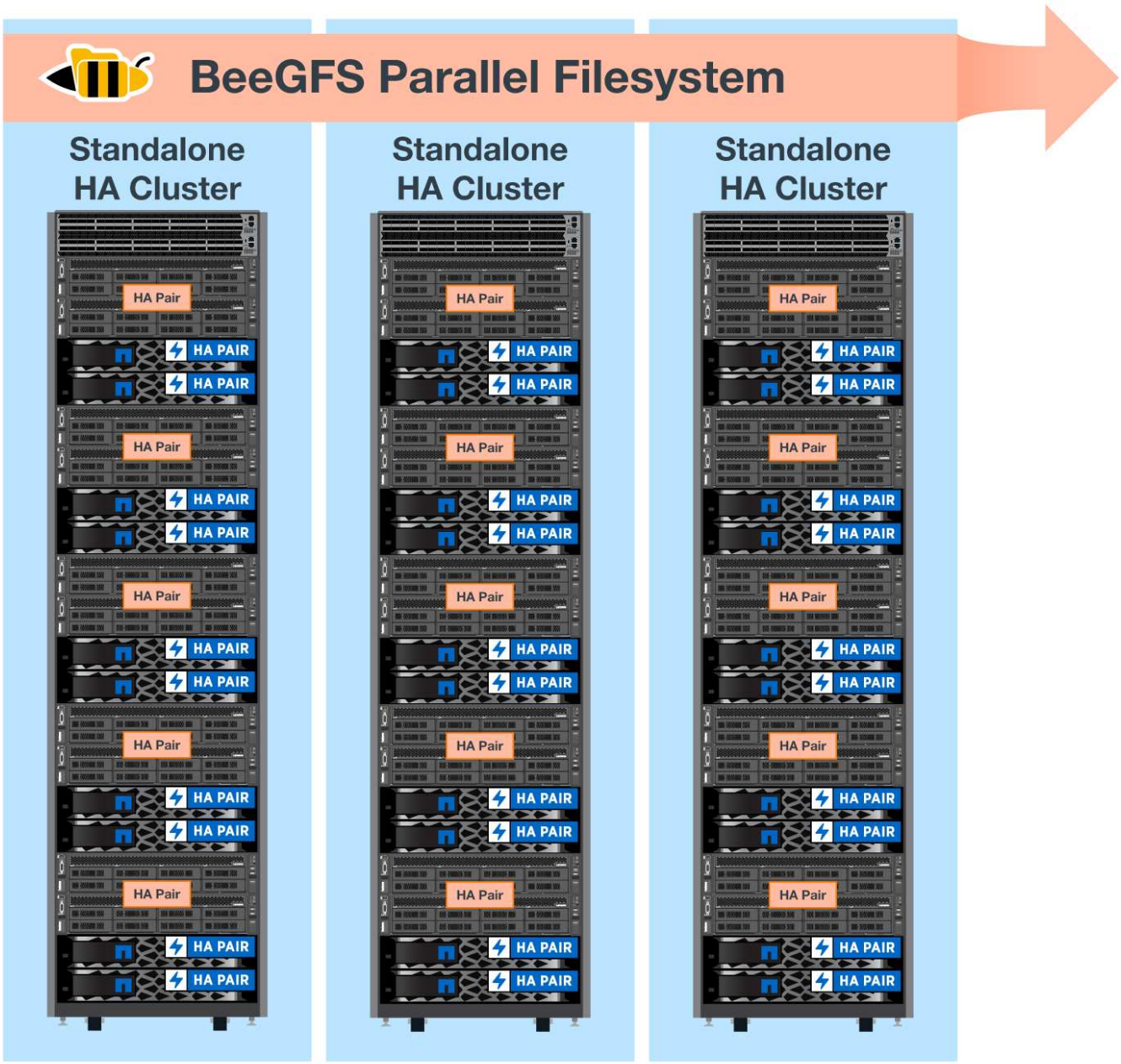
このソリューションのビルディングブロック（次の図を参照）では、BeeGFSファイルレイヤ用に検証済みファイルノードサーバを使用し、2台のEF600ストレージシステムをブロックレイヤとして使用します。




NetApp解決策 のBeeGFSは、導入環境のすべてのビルディングブロックで実行されます。最初に導入するビルディングブロックでは、BeeGFSの管理、メタデータ、ストレージサービス（基本ビルディングブロック）を実行する必要があります。後続のすべてのビルディングブロックは、ソフトウェアを使用して構成し、メタデータサービスとストレージサービスを拡張したり、ストレージサービスのみを提供したりできます。このモジュラ型アプローチにより、同じ基盤ハードウェアプラットフォームとビルディングブロック設計を使用しながら、ワークロードのニーズに合わせてファイルシステムを拡張できます。

最大5つのビルディングブロックを導入して、スタンドアロンのLinux HAクラスタを形成できます。これにより、Pacemakerによるリソース管理が最適化され、Corosyncとの効率的な同期が維持されます。これらのスタンドアロンBeeGFS HAクラスタの1つ以上が組み合わされてBeeGFSファイルシステムが作成され、クライアントは単一のストレージネームスペースとしてアクセスできます。ハードウェア側では、42Uラック1台に最大5つのビルディングブロックを収容でき、ストレージ/データネットワーク用に1U InfiniBandスイッチを2台搭載できます。視覚的な表現については、下の図を参照してください。

フェイルオーバークラスタでクォーラムを確立するには、少なくとも2つのビルディングブロックが必要です。2ノードクラスタには、フェイルオーバーの正常な実行を妨げる可能性がある制限があります。3つ目のデバイスをTiebreakerとして組み込むことで、2ノードクラスタを構成できますが、このドキュメントではその設計については説明していません。




**Ansible**

ネットアップのBeeGFSは、Ansible Automationを使用して提供および導入されます。この自動化はGitHubとAnsible Galaxy（BeeGFSコレクションはから入手できます "Ansible Galaxy" および "ネットアップのEシリーズGitHub"）。Ansibleは、主にBeeGFSビルディングブロックの構築に使用するハードウェアでテストされますが、サポートされているLinuxディストリビューションを使用して、ほぼすべてのx86ベースのサーバで実行するように設定できます。

詳細については、を参照してください "Eシリーズストレージを使用したBeeGFSの導入"。

## 技術要件

BeeGFS on NetAppソリューションを実装するには、本ドキュメントに記載されているテクノロジ要件を環境が満たしていることを確認してください。

ハードウェア要件

作業を開始する前に、BeeGFS on NetAppソリューションの1つの第2世代ビルディングブロック設計について、ハードウェアが次の仕様を満たしていることを確認してください。特定の導入に適したコンポーネントは、お客様の要件に応じて異なる場合があります。

| 数量 | ハードウェアコンポーネント | 要件 |
|---|---|---|
| 2. | BeeGFSファイルノード | 期待されるパフォーマンスを実現するには、各ファイルノードが推奨されるファイルノードの仕様を満たしている必要があります。<br><br>推奨されるファイルノードオプション：<br><br>• * Lenovo ThinkSystem SR665 V3*<br>◦ プロセッサー： AMD EPYC 9124 16C 3.0 GHz×2（2つのNUMAゾーンとして構成）。<br>◦ *メモリ:256 GB (16 x 16 GB TruDDR5 4800MHz RDIMM-A)<br>◦ * PCIe拡張：* 4つのPCIe Gen5 x16スロット（NUMAゾーンごとに2つ）<br>◦ その他:<br>▪ OS用RAID 1にドライブ2台（1TB 7.2K SATA以上）<br>▪ 1GbEポート（インバンドOS管理用）<br>▪ 1GbE BMCとRedfish APIによるアウトオブバンドサーバ管理<br>▪ デュアルホットスワップ電源装置とパフォーマンスファン |
| 2. | E-Seriesブロックノード（EF600アレイ） | メモリ： 256GB（コントローラあたり128GB）。アダプタ： 2ポート200GB/HDR（NVMe/IB）。*ドライブ：*必要なメタデータとストレージ容量に一致するように設定されています。 |
| 8. | InfiniBandホストカードアダプタ（ファイルノード用）。 | ホスト カード アダプターは、ファイル ノードのサーバー モデルによって異なる場合があります。検証済みファイルノードの推奨事項は次のとおりです。<br><br>• * Lenovo ThinkSystem SR665 V3サーバ：*<br>◦ MCX755106AS-HEAT ConnectX-7、NDR200、QSFP112、2ポート、PCIe Gen5 x16、 |
| 1. | ストレージネットワークスイッチ | ストレージネットワークスイッチの速度は200Gb/秒InfiniBandに対応している必要があります。推奨されるスイッチモデルは次のとおりです。<br><br>• * NVIDIA QM9700 Quantum 2 NDR InfiniBandスイッチ*<br>• * NVIDIA MQM8700 Quantum HDR InfiniBandスイッチ* |

ケーブル要件

ブロックノードからファイルノードへの直接接続。

| 数量 | パーツ番号 | 長さ |
| --- | --- | --- |
| 8. | MCP1650-H001E30（NVIDIAパッシブ銅ケーブル、QSFP56、200Gb/秒） | 100万 |

*ファイルノードからストレージネットワークスイッチへの接続。*InfiniBandストレージスイッチに応じて、次の表から適切なケーブルオプションを選択します。+推奨ケーブル長は2mですが、お客様の環境によって異なる場合があります。

| スイッチモデル | ケーブルタイプ | 数量 | パーツ番号 |
| --- | --- | --- | --- |
| NVIDIA QM9700 | アクティブファイバ（トランシーバを含む） | 2. | MMA4Z00-NS（マルチモード、IB/ETH、800GB/s 2×400Gb/sツインポートOSFP） |
| | | 4 | MFP7E20-Nxxx（マルチモード、4チャネル対2 2チャネルスプリッタファイバケーブル） |
| | | 8. | MMA1Z00-NS400（マルチモード、IB/ETH、400Gb/秒、シングルポートQSFP-112） |
| | パッシブカッパー | 2. | MCP7Y40-N002（NVIDIAパッシブ銅線スプリッタケーブル、InfiniBand 800Gb/s～4x200Gb/s、OSFP～4xQSFP112） |
| NVIDIA MQM8700 | アクティブファイバ | 8. | MFS1S00-H003E（NVIDIAアクティブファイバケーブル、InfiniBand 200Gb/s、QSFP56） |
| | パッシブカッパー | 8. | MCP1650-H002E26（NVIDIAパッシブ銅ケーブル、InfiniBand 200Gb/s、QSFP56） |

ソフトウェアとファームウェアの要件

予測可能なパフォーマンスと信頼性を確保するために、BeeGFS on NetAppソリューションのリリースでは、特定のバージョンのソフトウェアコンポーネントとファームウェアコンポーネントを使用してテストを実施しています。これらのバージョンは、ソリューションの実装に必要です。

ファイルのノード要件

| ソフトウェア | バージョン |
| --- | --- |
| Red Hat Enterprise Linux（RHEL） | 高可用性を備えた RHEL 9.4 物理サーバー (2 ソケット)。注: ファイル ノードには、有効な Red Hat Enterprise Linux Server サブスクリプションと Red Hat Enterprise Linux High Availability アドオンが必要です。 |
| Linuxカーネル | 5.14.0～427.42.1.el9_4.x86_64 |
| HCAファームウェア | **ConnectX-7 HCA**ファームウェア FW: 28.45.1200 + PXE: 3.7.0500 + UEFI: 14.38.0016<br>• ConnectX-6 HCAファームウェア* FW：20.43.2566 + PXE：3.7.0500 + UEFI：14.37.0013 |

**EF600**ブロックノードの要件

| ソフトウェア | バージョン |
| --- | --- |
| SANtricity OS の略 | 11.90R3 |
| NVSRAM | N6000-890834-D02.dlp |
| ドライブファームウェア | 使用中のドライブモデルで最新バージョンが提供されています。を参照してください"E-Seriesディスクファームウェアサイト"。 |

ソフトウェア導入の要件

次の表に、AnsibleベースのBeeGFS導入の一環として自動的に導入されるソフトウェア要件を示します。

| ソフトウェア | バージョン |
| --- | --- |
| BeeGFSの場合 | 7.4.6 |
| Corosync | 3.1.8-1 |
| ペースメーカー | 2.1.7-5.2 |
| PCS | 0.11.7-2 |
| フェンスエージェント（redfish/APC） | 4.10.0-62 |
| InfiniBand / RDMAドライバ | MLNX_OFED_Linux-23.10-3.2.2.1-LTS |

**Ansible**の制御ノード要件

NetApp解決策 のBeeGFSは、Ansible制御ノードから導入して管理します。詳細については、を参照してください "Ansibleのドキュメント"。

次の表に示すソフトウェア要件は、以下に記載するNetApp BeeGFSコレクションのバージョンに固有のものです。

| ソフトウェア | バージョン |
| --- | --- |
| Ansible | 10.x |
| Ansibleコア | 2.13.0以上 |
| Python | 3.10 |
| その他のPythonパッケージ | 暗号化- 43.0.0、netaddr-1.3.0、ipaddr-2.2.0 |
| NetApp E-Series BeeGFS Ansibleコレクション | 3.2.0 |

# 解決策 の設計を確認します

## 設計の概要

### BeeGFS並列ファイルシステムとNetApp EF600ストレージシステムを組み合わせ

## たNetApp解決策 上でBeeGFSをサポートするには、特定の機器、ケーブル配線、構成が必要です。

詳細はこちら。

- "ハードウェア構成"
- "ソフトウェア構成"
- "設計の検証"
- "サイジングガイドライン"
- "パフォーマンスの調整"

設計とパフォーマンスの違いを伴う派生アーキテクチャ：

- "大容量ビルディングブロック"

## ハードウェア構成

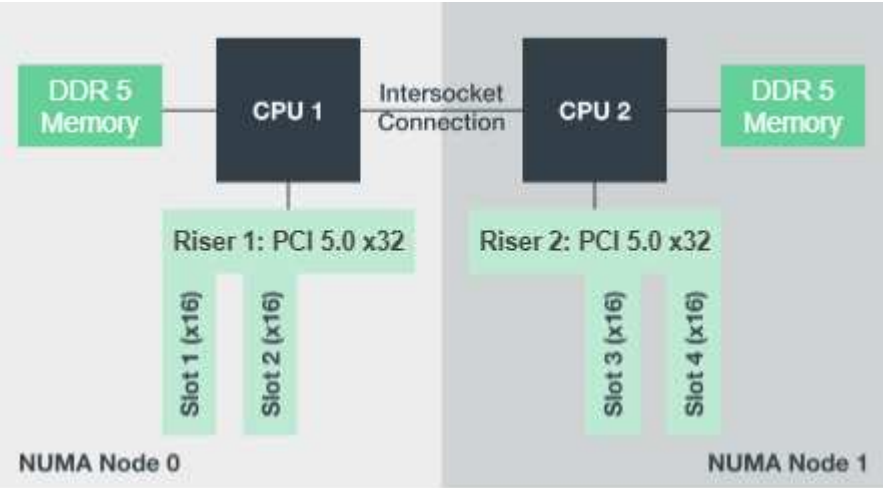
## ネットアップのBeeGFSのハードウェア構成には、ファイルノードとネットワークのケーブル配線が含まれます。

ファイルのノード構成

ファイルノードには、別々のNUMAゾーンとして構成された2つのCPUソケットがあり、同じ数のPCIeスロットとメモリへのローカルアクセスが含まれます。

InfiniBandアダプタは、適切なPCIライザーまたはスロットに装着する必要があります。これにより、使用可能なPCIeレーンとメモリチャネル間でワークロードが分散されます。個々のBeeGFSサービスの作業を特定のNUMAノードに完全に分離することで、ワークロードのバランスを調整します。目標は、各ファイルノードのパフォーマンスを、2つの独立したシングルソケットサーバと同様にすることです。

次の図は、ファイルノードのNUMA構成を示しています。




BeeGFSプロセスは、使用するインターフェイスが同じゾーン内にあることを確認するために、特定のNUMAゾーンに固定されます。この構成により、ソケット間接続を介したリモートアクセスが不要になります。ソケット間接続はQPIまたはGMI2リンクと呼ばれることもあります。最新のプロセッサアーキテクチャであって

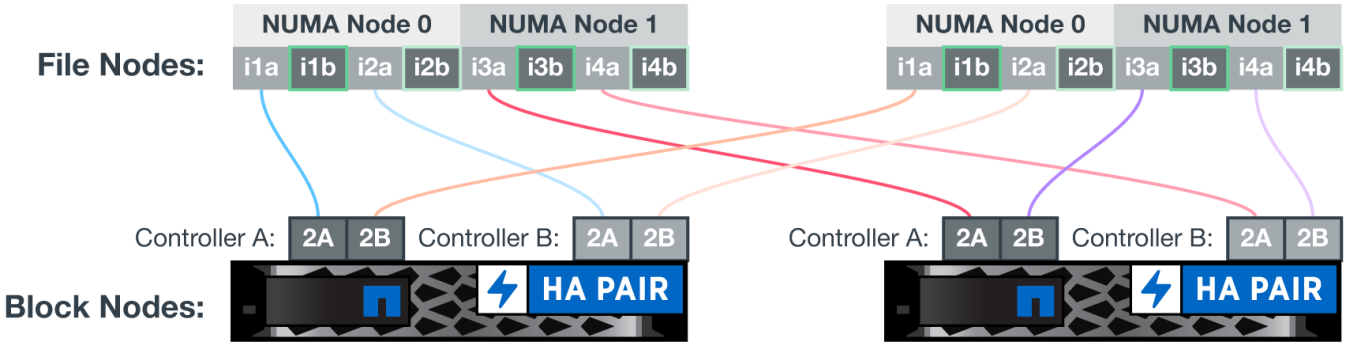
も、HDR InfiniBandなどの高速ネットワークを使用する場合はボトルネックになる可能性があります。

### ネットワークのケーブル構成

ビルディングブロック内では、各ファイルノードは合計4つの冗長InfiniBand接続を使用して2つのブロックノードに接続されます。また、各ファイルノードにInfiniBandストレージネットワークへの冗長接続が4つあります。

次の図に注意してください。

- 緑で示されているすべてのファイルノードポートは、ストレージファブリックへの接続に使用されます。他のすべてのファイルノードポートは、ブロックノードへの直接接続です。
- 特定のNUMAゾーン内の2つのInfiniBandポートは、同じブロックノードのAおよびBコントローラに接続します。
- NUMAノード0のポートは常に最初のブロックノードに接続します。
- NUMAノード1のポートは、2番目のブロックノードに接続します。




スプリッタケーブルを使用してストレージスイッチをファイルノードに接続する場合は、1本のケーブルが分岐して薄い緑色のポートに接続する必要があります。もう1本のケーブルが分岐し、濃い緑色で示されているポートに接続します。また、冗長スイッチを使用するストレージネットワークの場合、薄い緑色のポートは1つのスイッチに接続し、濃い緑色のポートは別のスイッチに接続します。

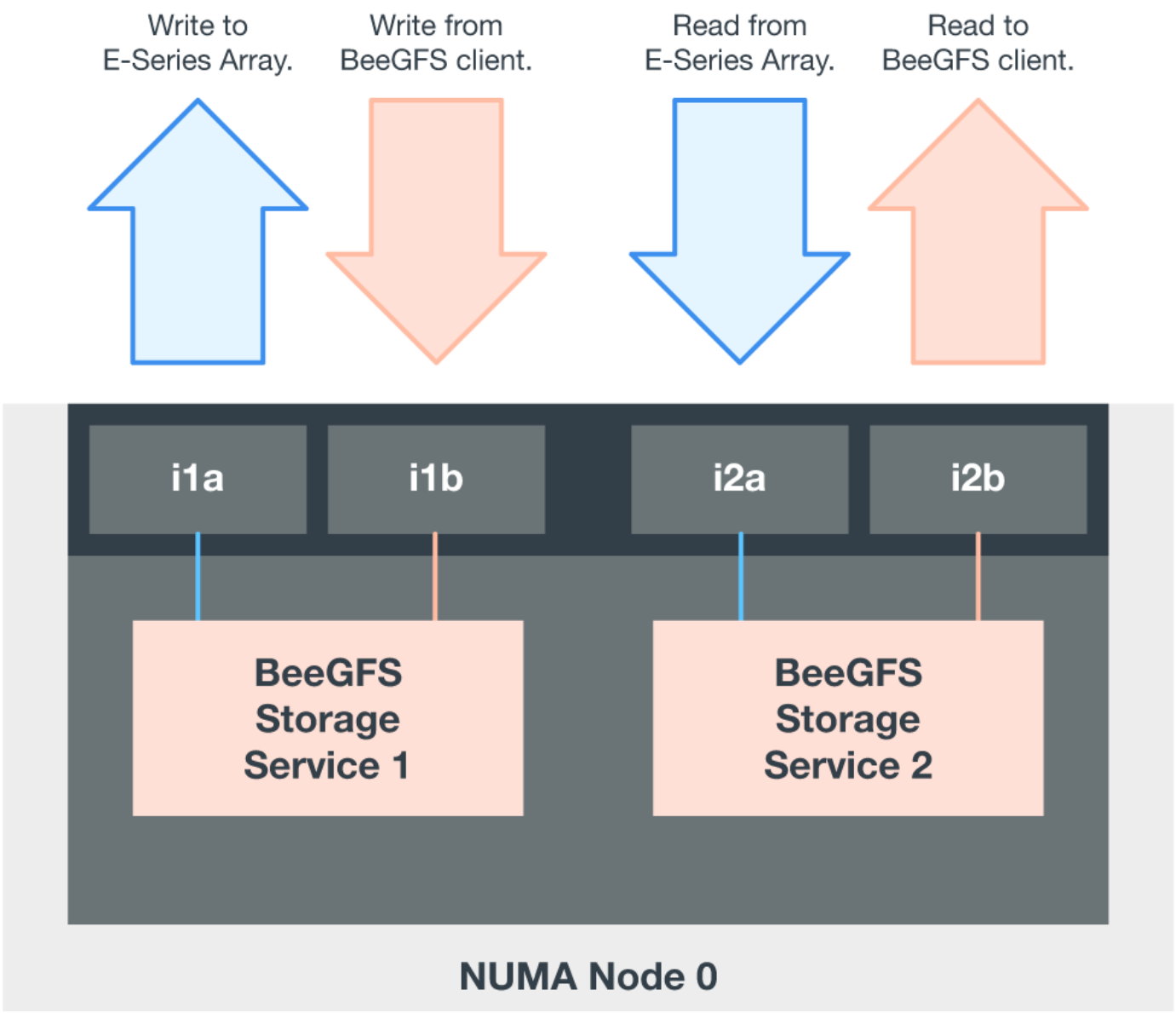
この図に示すケーブル構成では、BeeGFSサービスごとに次のことが可能です。

- BeeGFSサービスを実行しているファイルノードに関係なく、同じNUMAゾーンで実行します。
- 障害が発生した場所に関係なく、フロントエンドストレージネットワークおよびバックエンドブロックノードへのセカンダリの最適パスを確保する。
- ブロックノード内のファイルノードまたはコントローラのメンテナンスが必要な場合は、パフォーマンスへの影響を最小限に抑えます。

### 帯域幅を活用するためのケーブル接続

PCIeの完全な双方向帯域幅を利用するには、各InfiniBandアダプタの1つのポートをストレージファブリックに接続し、もう1つのポートをブロックノードに接続します。

次の図に、PCIeの双方向帯域幅をフルに活用するためのケーブル配線の設計を示します。




BeeGFSサービスごとに、同じアダプタを使用して、クライアントトラフィックに使用する優先ポートと、そのサービスボリュームのプライマリ所有者であるブロックノードコントローラへのパスを接続します。詳細については、を参照してください "ソフトウェア構成"。

## ソフトウェア構成

ネットアップのBeeGFSのソフトウェア設定には、BeeGFSネットワークコンポーネント、EF600ブロックノード、BeeGFSファイルノード、リソースグループ、BeeGFSサービスが含まれます。

### BeeGFSネットワーク設定

BeeGFSネットワーク設定は、次のコンポーネントで構成されます。

- *フローティングIP *フローティングIPは、同じネットワーク内の任意のサーバーに動的にルーティングできる一種の仮想IPアドレスです。複数のサーバーが同じフローティングIPアドレスを所有できますが、一度にアクティブにできるのは1つのサーバーのみです。

各BeeGFSサーバサービスには、BeeGFSサーバサービスの実行場所に応じてファイルノード間を移動できる独自のIPアドレスがあります。このフローティングIP構成では、各サービスが他のファイルノードに独立してフェイルオーバーできます。クライアントは、特定のBeeGFSサービスのIPアドレスを知るだけで済み、そのサービスを現在実行しているファイルノードを認識する必要はありません。

- * BeeGFSサーバのマルチホーミング構成*解決策 の密度を高めるために'各ファイル・ノードには同じIPサブネットにIPが設定された複数のストレージ・インターフェイスがあります

  この設定がLinuxネットワークスタックで正常に機能するようにするには、追加の設定が必要です。これは、デフォルトでは、IPが同じサブネット内にある場合、1つのインターフェイスへの要求に別のインターフェイスで応答できるからです。他の欠点に加えて、このデフォルトの動作により、RDMA接続を適切に確立または維持することができなくなります。

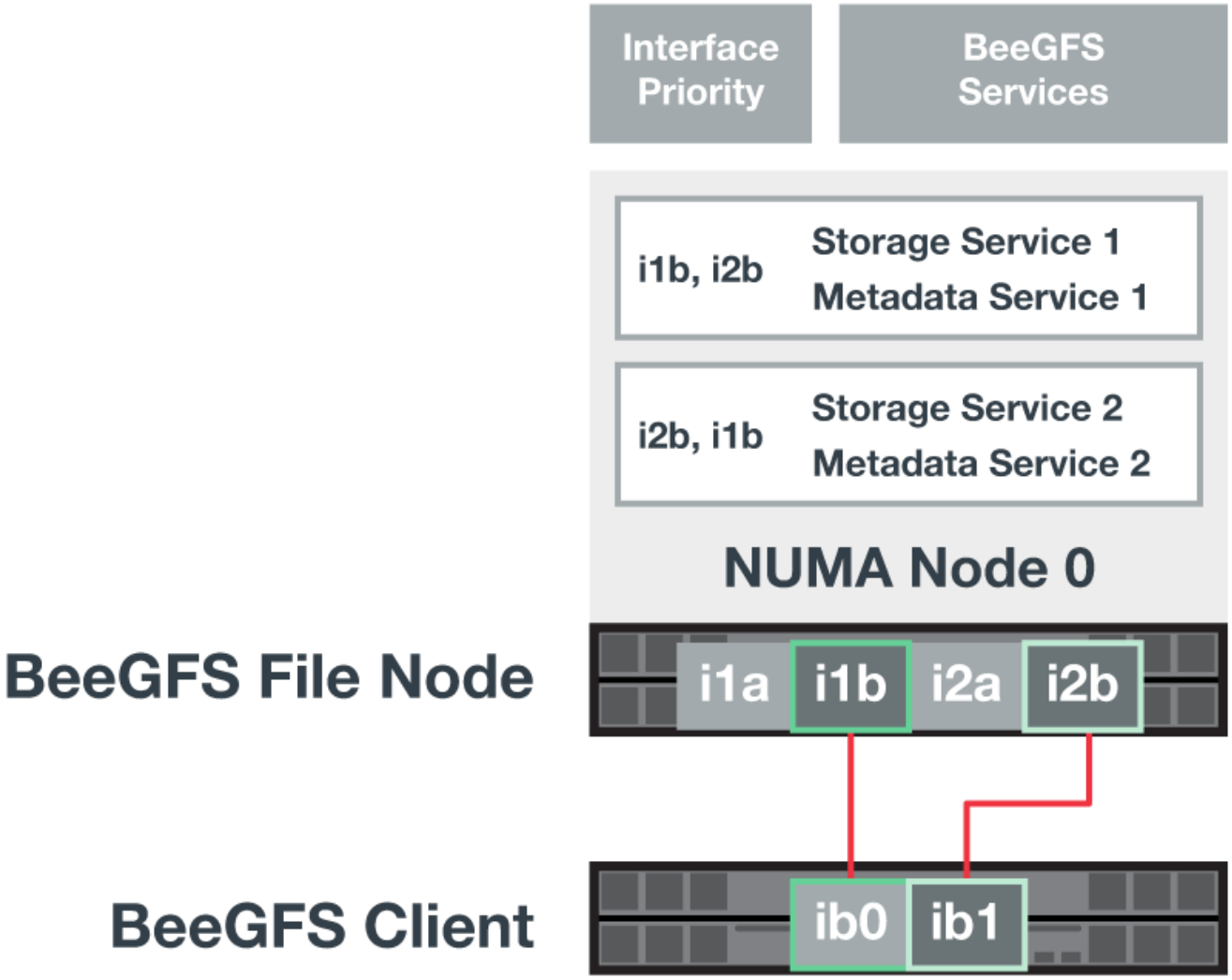
  Ansibleベースの導入では、逆方向パス（RP）とアドレス解決プロトコル（ARP）の動作の締め付けに加え、フローティングIPの開始と停止のタイミングを保証します。対応するIPルートとルールが動的に作成され、マルチホームネットワークの設定が適切に機能できるようになります。

- * BeeGFSクライアントのマルチレール構成*_Multi-rail_とは、アプリケーションが複数の独立したネットワーク接続（「レール」）を使用してパフォーマンスを向上させる機能を指します。

  BeeGFSはマルチレールサポートを実装し、1つのIPoIBサブネットで複数のIBインターフェイスを使用できるようにします。この機能により、RDMA NIC間での動的なロードバランシングなどの機能が有効になり、ネットワークリソースの使用が最適化されます。また、NVIDIA GPUDirect Storage（GDS）とも統合されているため、システム帯域幅が増加し、クライアントのCPUのレイテンシと使用率が低下します。

  このマニュアルでは、単一のIPoIBサブネット設定の手順について説明します。デュアルIPoIBサブネット構成はサポートされていますが、シングルサブネット構成と同じ利点はありません。

  次の図に、複数のBeeGFSクライアントインターフェイス間でのトラフィックの分散を示します。




BeeGFSの各ファイルは通常、複数のストレージサービスにまたがってストライピングされるため、マルチレール構成では、単一のInfiniBandポートでは実現できないスループットよりも高いスループットをクライアントに提供できます。たとえば、次のコード例は、クライアントが両方のインターフェイス間でトラフィックを分散できるようにする、一般的なファイルストライピング設定を示しています。

+

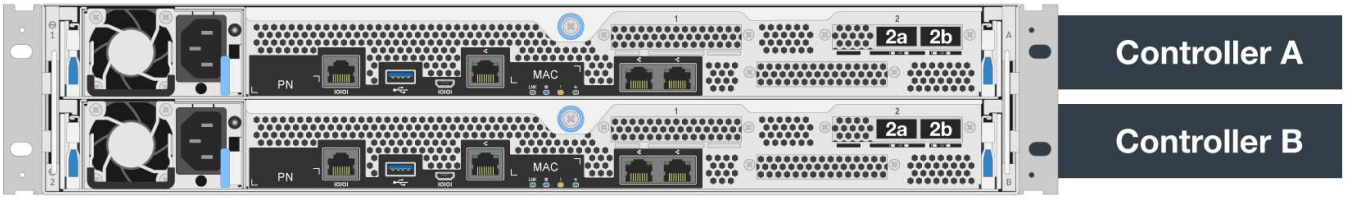
```
root@beegfs01:/mnt/beegfs# beegfs-ctl --getentryinfo myfile
Entry type: file
EntryID: 11D-624759A9-65
Metadata node: meta_01_tgt_0101 [ID: 101]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 1M
+ Number of storage targets: desired: 4; actual: 4
+ Storage targets:
  + 101 @ stor_01_tgt_0101 [ID: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

**EF600**ブロックノード構成

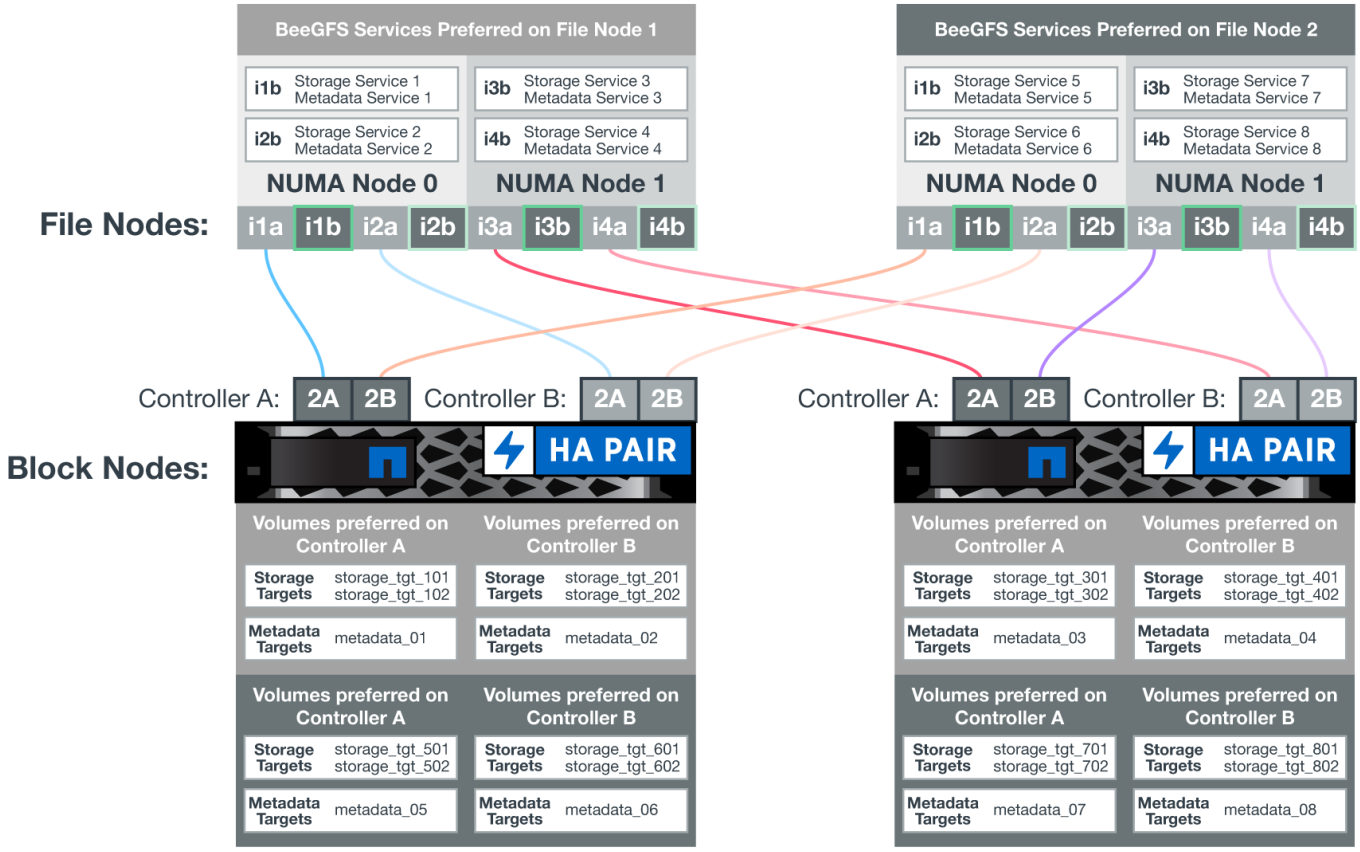
ブロックノードは、同じドライブセットへの共有アクセスを持つ2つのアクティブ/アクティブRAIDコントローラで構成されます。通常、各コントローラにはシステムに設定されているボリュームの半分が所有されますが、必要に応じてもう一方のコントローラがテイクオーバーできます。

ファイルノード上のマルチパスソフトウェアは、各ボリュームへのアクティブなパスと最適パスを決定し、ケーブル、アダプタ、またはコントローラに障害が発生した場合に自動的に代替パスに移動します。

次の図は、EF600ブロックノードのコントローラのレイアウトを示しています。




共有ディスクのHA解決策 を使用する場合、ボリュームが両方のファイルノードにマッピングされ、必要に応じて相互にテイクオーバーできるようになります。次の図は、パフォーマンスを最大化するためにBeeGFSサービスと優先ボリューム所有権を設定する例を示しています。各BeeGFSサービスの左側にあるインターフェイスは、クライアントとその他のサービスが接続に使用する優先インターフェイスを示します。




前の例では、クライアントとサーバサービスは、インターフェイスi1bを使用してストレージサービス1と通信することを好みます。ストレージサービス1は、最初のブロックノードのコントローラA上のボリューム（storage_tgt_101、102）と通信するための優先パスとしてインターフェイスi1aを使用します。この構成では、InfiniBandアダプタで使用できる完全な双方向PCIe帯域幅を使用し、PCIe 4.0では実現できないデュアルポートHDR InfiniBandアダプタよりも優れたパフォーマンスを実現します。

### BeeGFSファイルのノード設定

BeeGFSファイルノードは、複数のファイルノード間でBeeGFSサービスのフェイルオーバーを容易にするために、ハイアベイラビリティ（HA）クラスタに構成されます。

HAクラスタは、広く使用されている2つのLinux HAプロジェクトに基づいて設計されています。クラスタメンバーシップ用のCorosyncと、クラスタリソース管理用のPacemakerです。詳細については、を参照してください "高可用性アドオンに関するRed Hatトレーニング"。

複数のオープンクラスタフレームワーク（OCF）リソースエージェントがネットアップにオーサリングされ、拡張されました。このエージェントを使用すると、クラスタでBeeGFSリソースのインテリジェントな起動と監視が可能になります。

### BeeGFS HA clusters（BeeGFS HAクラスタ

通常、BeeGFSサービスを（HAの有無にかかわらず）開始するときは、次のようなリソースが必要になります。

- サービスに到達できるIPアドレス。通常はNetwork Managerによって設定されます。
- BeeGFSがデータを格納するためのターゲットとして使用する基盤となるファイルシステム。

これらは通常'/etc/fstabで定義され’システム・ディスクでマウントされます

- 他のリソースの準備ができたときにBeeGFSプロセスを開始するシステムdサービス。

  追加のソフトウェアがない場合、これらのリソースは単一のファイルノードからのみ開始されます。したがって、ファイルノードがオフラインになると、BeeGFSファイルシステムの一部にアクセスできなくなります。

複数のノードでBeeGFSサービスを起動できるため、ペースメーカーは各サービスと依存するリソースが一度に1つのノードでのみ動作していることを確認する必要があります。たとえば、2つのノードが同じBeeGFSサービスを起動しようとすると、どちらも基盤となるターゲット上の同じファイルに書き込みを試みると、データが破損するおそれがあります。この状況を回避するために、Pacemakerは、クラスタ全体の状態をすべてのノードで確実に同期し、クォーラムを確立するために、Corosyncに依存しています。

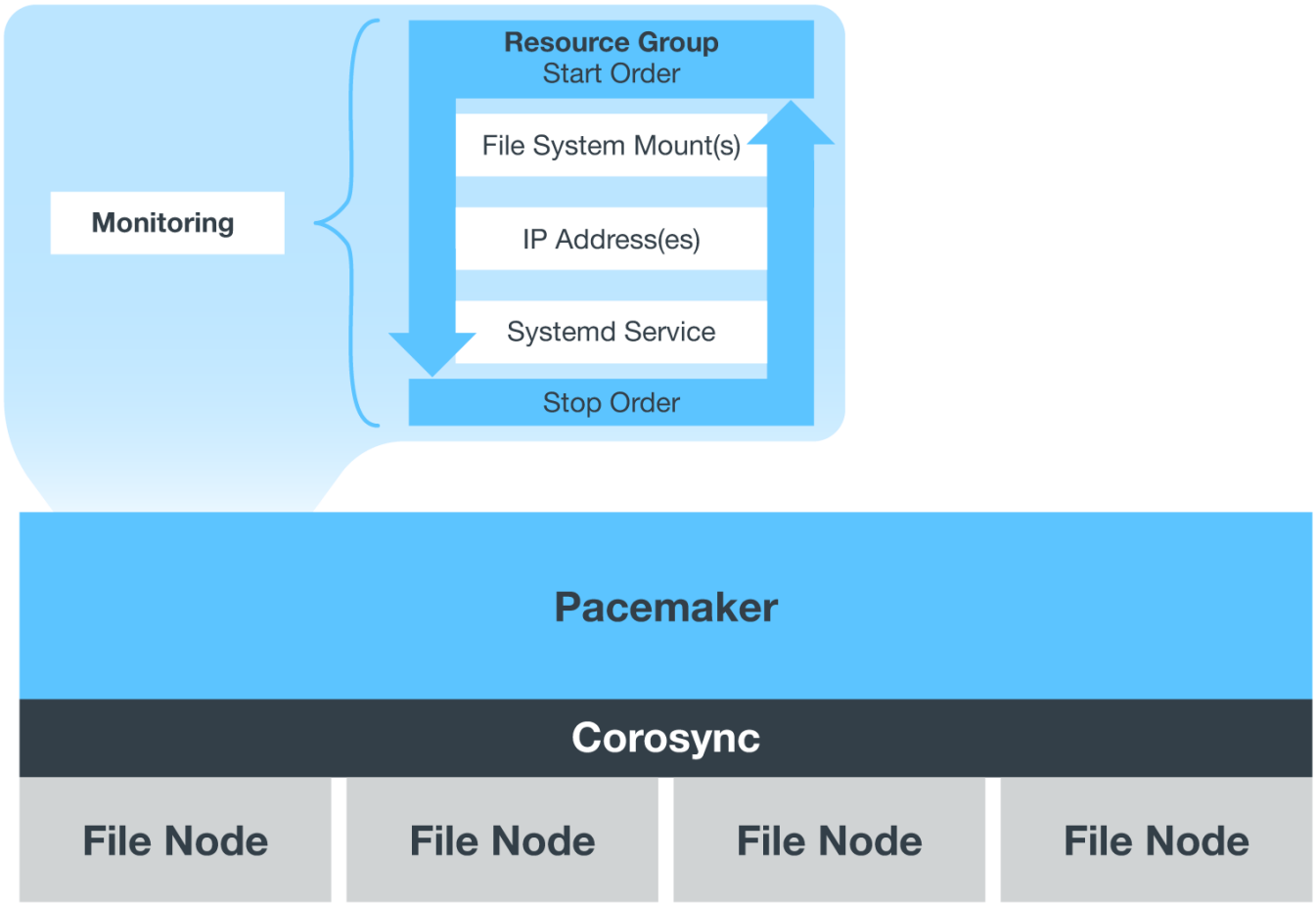
クラスタで障害が発生すると、Pacemakerは別のノードのBeeGFSリソースに反応して再起動します。一部の状況では、ペースメーカーが障害のある元のノードと通信できず、リソースが停止していることを確認できない場合があります。BeeGFSリソースを他の場所から再起動する前にノードが停止していることを確認するために、Pacemakerは障害のあるノードをフェンシングします。この場合、電源を切断します。

多くのオープンソースフェンシングエージェントを使用すると、Pacemakerは配電ユニット（PDU）を搭載したノードを遮断したり、RedfishなどのAPIを搭載したサーバベースボード管理コントローラ（BMC）を使用してノードを遮断したりできます。

HAクラスタでBeeGFSを実行している場合は、すべてのBeeGFSサービスと基盤となるリソースがペースメーカーによってリソースグループで管理されます。各BeeGFSサービスとそれが依存するリソースは、リソースグループに設定されます。これにより、リソースが正しい順序で開始および停止され、同じノードに配置されるようになります。

BeeGFSリソースグループごとに、PacemakerはカスタムのBeeGFSモニタリングリソースを実行します。このリソースは、障害状態を検出し、特定のノードでBeeGFSサービスがアクセスできなくなったときにフェイルオーバーをインテリジェントにトリガーします。

次の図に、Pacemaker制御のBeeGFSサービスと依存関係を示します。




同じタイプの複数のBeeGFSサービスが同じノードで起動するように、Pacemakerはマルチモード設定方式を使用してBeeGFSサービスを開始するように設定されます。詳細については、を参照してください "マルチモードでのBeeGFSのマニュアル"。

BeeGFSサービスは複数のノードで起動できる必要があるため'各サービスの構成ファイル（通常は/etc/beegfsにあります）は'そのサービスのBeeGFSターゲットとして使用されるEシリーズボリュームの1つに保存されますこれにより、特定のBeeGFSサービスのデータとともに、サービスの実行に必要なすべてのノードから設定へのアクセスが可能になります。

```
# 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
```

## 設計の検証

### NetApp解決策 のBeeGFSの第2世代設計は、3つのビルディングブロック構成プロファイルを使用して検証されました。

構成プロファイルには、次のものが含まれます。

- BeeGFSの管理、メタデータ、ストレージサービスなど、単一のベースとなるビルディングブロックです。
- BeeGFSメタデータとストレージビルディングブロック
- BeeGFSストレージ専用のビルディングブロック。

ビルディングブロックは、2台のNVIDIA Quantum InfiniBand（MQM8700）スイッチに接続されています。また、10個のBeeGFSクライアントもInfiniBandスイッチに接続し、総合ベンチマークユーティリティの実行に使用しました。

次の図は、NetApp解決策 でBeeGFSを検証するために使用するBeeGFS設定を示しています。


**BeeGFS**ファイルのストライピング

並列ファイルシステムの利点は、複数のストレージターゲットに個別のファイルをストライプすることです。これは、同一または異なる基盤となるストレージシステム上のボリュームを表すことができます。

BeeGFSでは'各ファイルに使用するターゲットの数を制御し'各ファイルストライプに使用するチャンクサイズ（またはブロックサイズ）を制御するために'ディレクトリ単位およびファイル単位でストライピングを構成できますこの構成では、サービスを再設定したり再開したりすることなく、ファイルシステムがさまざまなタイプのワークロードやI/Oプロファイルをサポートできます。ストライプ設定を適用するには'beegfs-ctl'コマンドラインツールを使用するか'ストライピングAPIを使用するアプリケーションを使用します詳細については、のBeeGFSのマニュアルを参照してください "ストライピング" および "ストライピングAPI"。

最高のパフォーマンスを得るために、テスト全体を通してストライプパターンを調整し、各テストに使用するパラメータを記録しました。

**IOR**帯域幅テスト：複数のクライアント

IOR帯域幅テストでは、OpenMPIを使用して、統合I/OジェネレータツールIOR（から入手可能）の並列ジョブを実行しました "HPC GitHub"）を使用して、10のすべてのクライアントノードから1つ以上のBeeGFSビルディングブロックにアクセスします。特に明記されていない限り：

- すべてのテストで、直接I/Oを使用し、転送サイズは1MiBに設定しました。
- BeeGFSファイルのストライピングは'1 MBのチャンクサイズと1ファイルあたり1つのターゲットに設定されました

IORでは次のパラメータを使用し、1つのビルディングブロックではアグリゲートのファイルサイズが5TiB、3つのビルディングブロックでは40TiBにセグメント数を調整しました。

```
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
```

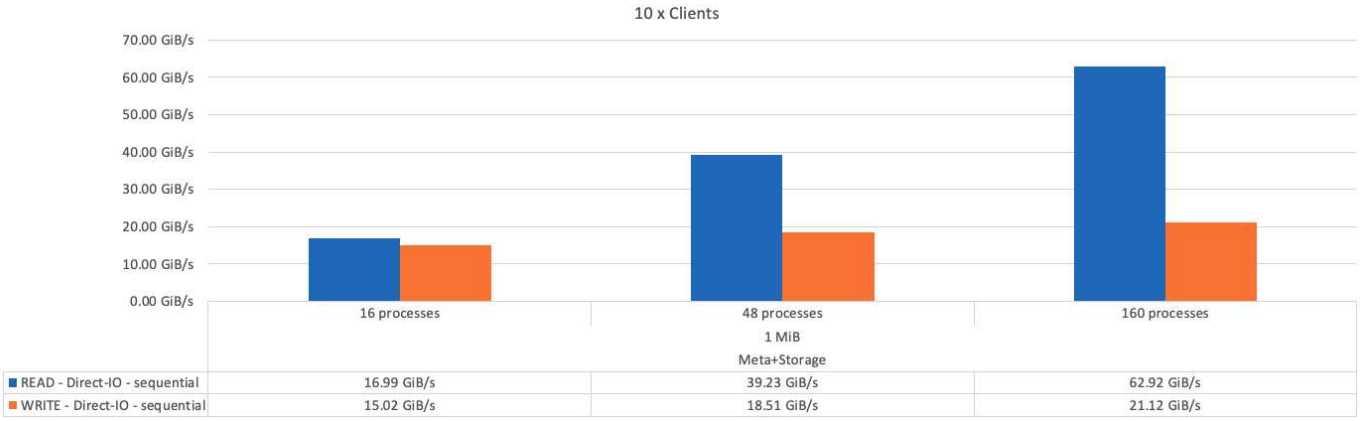
**BeeGFS**ベース（管理、メタデータ、ストレージ）ビルディングブロック**×1**

次の図に、1つのBeeGFSベース（管理、メタデータ、ストレージ）ビルディングブロックを使用したIORテ

スト結果を示します。


**BeeGFS**メタデータとストレージビルディングブロック

次の図は、1つのBeeGFSメタデータとストレージビルディングブロックを使用したIORテスト結果を示しています。

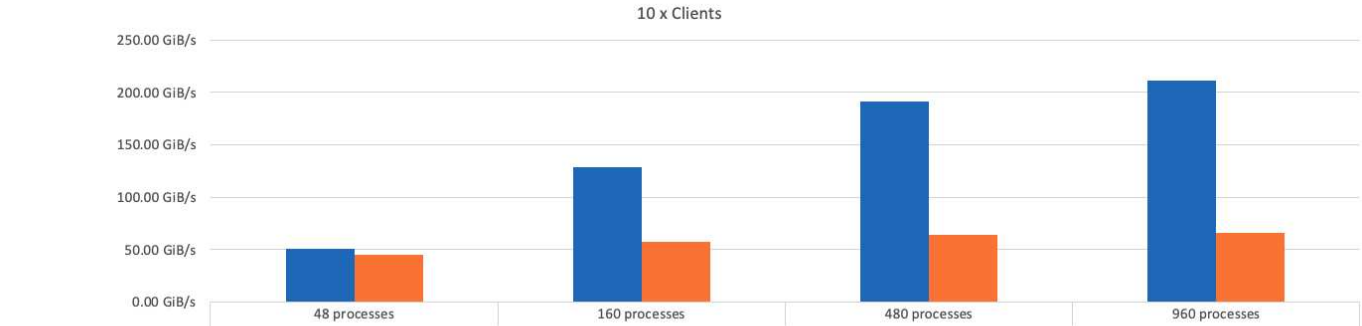



**BeeGFS**ストレージ専用のビルディングブロック

次の図に、1つのBeeGFSストレージ専用ビルディングブロックを使用したIORテスト結果を示します。


**3**つの**BeeGFS**ビルディングブロック

次の図に、3つのBeeGFSビルディングブロックを使用したIORテスト結果を示します。




基本となるビルディングブロックと後続のメタデータとストレージビルディングブロックのパフォーマンスの違いは、想定どおりごくわずかです。メタデータとストレージのビルディングブロックおよびストレージ専用のビルディングブロックを比較した場合、ストレージターゲットとして使用される追加のドライブが原因で読み取りパフォーマンスがわずかに向上します。ただし、書き込みパフォーマンスに大きな違いはありません。パフォーマンスを向上させるには、複数のビルディングブロックを1つに追加して、パフォーマンスを直線的に拡張します。

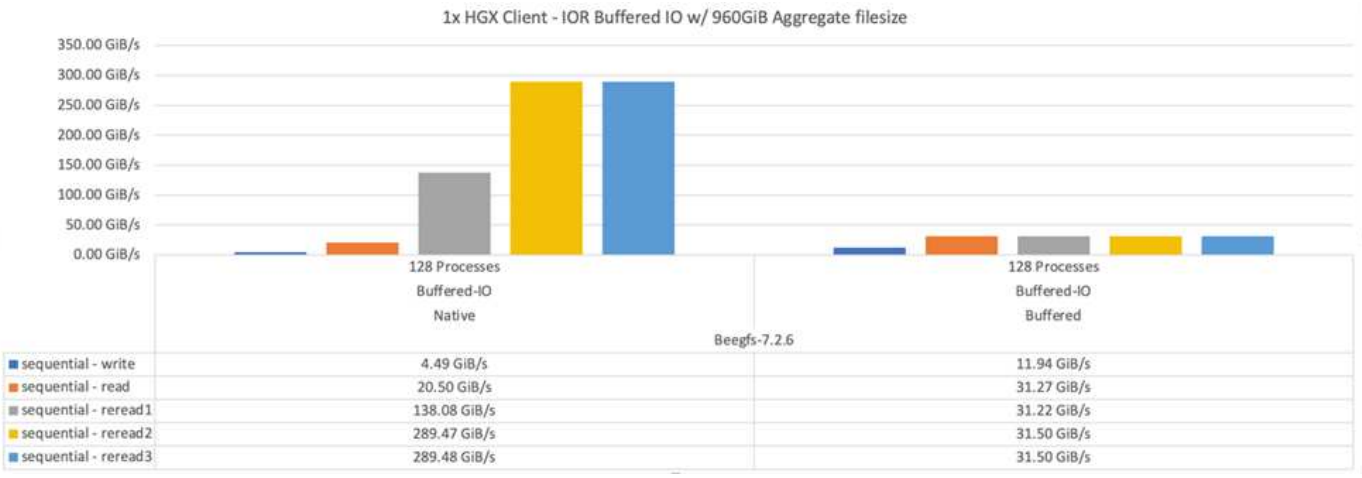
**IOR**帯域幅テスト：単一クライアント

IOR帯域幅テストでは、OpenMPIを使用して、1台の高性能GPUサーバを使用して複数のIORプロセスを実行し、1台のクライアントで実現可能なパフォーマンスを検証しました。

このテストでは'クライアントがLinuxカーネルのページキャッシュ(`tuneFileCacheType=native')を使用するように構成されている場合に'BeeGFSの再読み取り動作とパフォーマンスを'デフォルトのバッファ設定と比較します

ネイティブ・キャッシュ・モードでは'クライアント上のLinuxカーネル・ページ・キャッシュを使用するため'ネットワーク上で再送信されるのではなく'ローカル・メモリから再読み取り操作を行うことができます

次の図は、3つのBeeGFSビルディングブロックと1つのクライアントを使用したIORテスト結果を示しています。




これらのテストのBeeGFSストライピングは'ファイルあたり8つのターゲットを持つ1MBのチャンクサイズに設定されました

デフォルトのバッファモードを使用した場合の書き込みと初期の読み取りパフォーマンスは向上しますが、同じデータを複数回再読み取りするワークロードでは、ネイティブのキャッシュモードを使用するとパフォーマンスが大幅に向上します。ディープラーニングなどのワークロードで多くの期間にわたって同じデータセットを複数回再読み取りする場合は、読み取りパフォーマンスの向上が重要になります。

メタデータパフォーマンステスト

メタデータパフォーマンステストでは、MDTestツール（IORの一部として含まれる）を使用してBeeGFSのメタデータパフォーマンスを測定しました。テストでは、OpenMPIを使用して、10個のクライアントノードすべてで並列ジョブを実行しました。

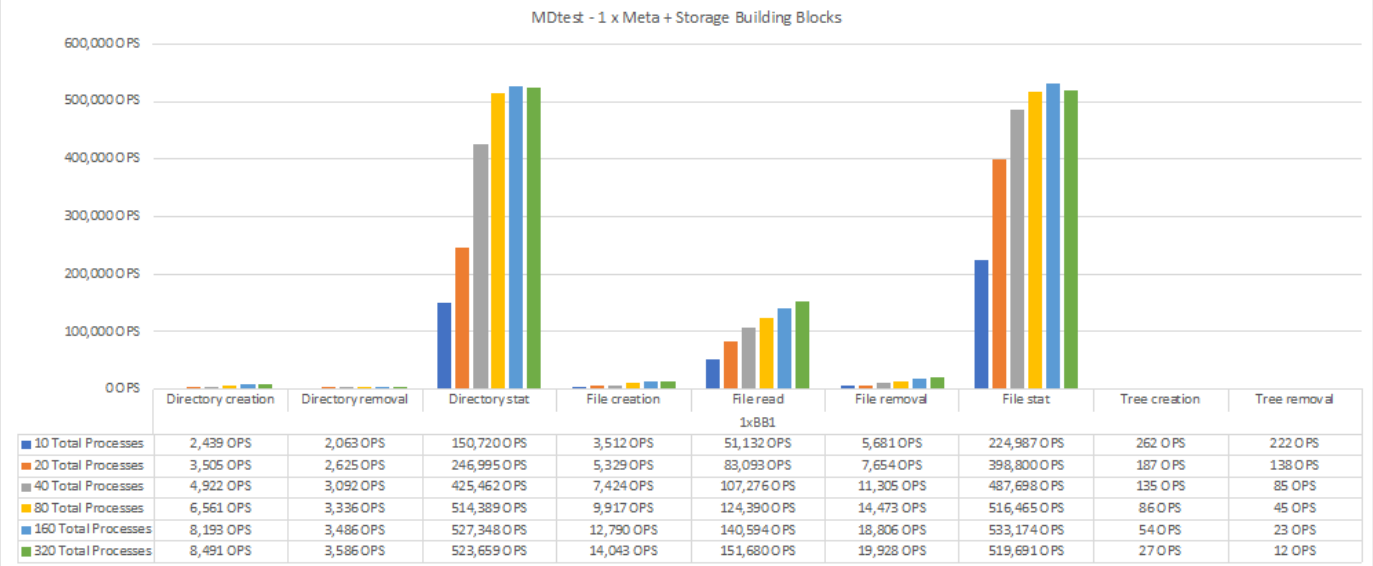
以下のパラメータを使用して、ステップ2xでステップ10から320まで、ファイルサイズ4Kのプロセスの合計数を使用して、ベンチマークテストを実行しました。

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

メタデータパフォーマンスは、まずメタデータとストレージのビルディングブロックを2つずつ使用して測定し、ビルディングブロックを追加してパフォーマンスがどのようにスケールアップするかを示しました。

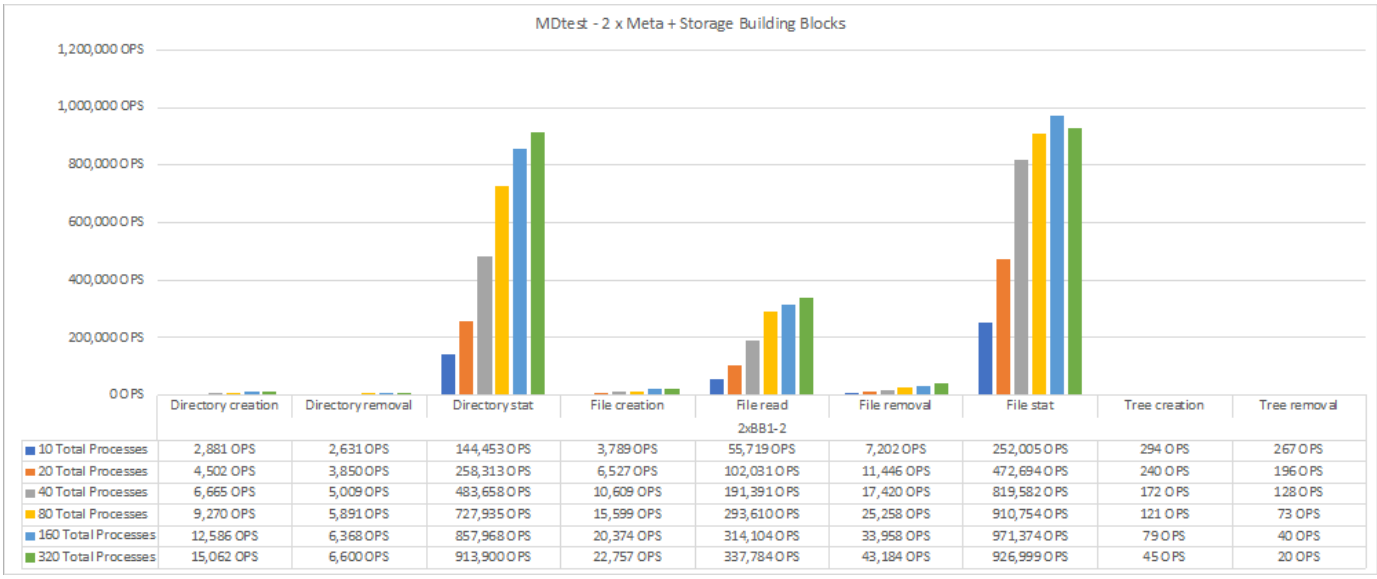
**BeeGFS**メタデータとストレージビルディングブロック**×1**

次の図は、BeeGFSメタデータとストレージビルディングブロックを1つずつ使用したMDTestの結果を示しています。




**BeeGFS**メタデータとストレージビルディングブロックが**2**つずつ搭載されています

次の図は、2つのBeeGFSメタデータとストレージビルディングブロックを使用したMDTestの結果を示しています。




| | 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 |

機能検証

このアーキテクチャの検証の一環として、ネットアップは次の機能テストをいくつか実施しました。

- スイッチポートを無効にして、単一のクライアントInfiniBandポートを障害状態にします。
- スイッチポートを無効にして、単一サーバのInfiniBandポートを障害状態にします。
- BMCを使用した即時サーバ電源オフのトリガー
- ノードを正常にスタンバイにし、別のノードにサービスをフェイルオーバーします。
- ノードを正常にオンラインに戻し、元のノードにサービスをフェイルバックします。
- PDUを使用している一方のInfiniBandスイッチの電源をオフにします。すべてのテストは、BeeGFSクライアントで設定された「sysSessionChecksEnabled：false」パラメータを使用して、ストレステストの実行中に実行されました。エラーやI/Oの中断は発生しませんでした。

既知の問題 がある（を参照） "変更ログ") BeeGFSクライアント/サーバRDMA接続が予期せず中断される場合は、プライマリインターフェイスの喪失（「connInterfacesFile」で定義）またはBeeGFSサーバの障害のいずれかによって、アクティブなクライアントI/Oが最大10分間ハングアップしてから再開します。この問題 は、計画的メンテナンスのためにBeeGFSノードが正常に配置され、スタンバイ状態から外れたとき、またはTCPが使用中のときは発生しません。

**NVIDIA DGX SuperPODとBasePODの検証**

ネットアップでは、3つのビルディングブロックにメタデータとストレージ構成プロファイルが適用されたBeeGFSファイルシステムを使用して、NVDIA DGX A100 SuperPOD向けのストレージ解決策 の検証を実施しました。認定には、このNVAで説明した解決策 を、さまざまなストレージ、機械学習、ディープラーニングのベンチマークを実行している20台のDGX A100 GPUサーバでテストすることが含まれます。NVIDIAのDGX A100 SuperPODで確立された検証に基づいて、BeeGFS on NetAppソリューションは、DGX SuperPOD H100、H200、B200システムでの使用が承認されました。この拡張は、NVIDIA DGX A100で検証された、以前に確立されたベンチマークとシステム要件を満たすことに基づいています。

詳細については、を参照してください "NVIDIA DGX SuperPODとネットアップ" および "NVIDIA DGX BasePOD"。

## サイジングガイドライン

# BeeGFS解決策 には、検証テストに基づくパフォーマンスと容量のサイジングに関する推奨事項が含まれます。

ビルディングブロックアーキテクチャの目的は、特定のBeeGFSシステムの要件を満たす複数のビルディングブロックを追加することで、サイズを簡単に決定できる解決策 を作成することです。以下のガイドラインを使用して、環境の要件を満たすために必要なBeeGFSビルディングブロックの数とタイプを見積もります。

これらの見積もりは、ケースのパフォーマンスが最も優れていることに留意してください。統合ベンチマークアプリケーションは、基盤となるファイルシステムの使用を最適化するために作成され、実際のアプリケーションでは使用されない可能性があります。

### パフォーマンスのサイジング

次の表に、推奨されるパフォーマンスサイジングを示します。

| 構成プロファイル | 読み取り：**1MiB** | **1MiB**の書き込み |
|---|---|---|
| メタデータ+ストレージ | 62GiBps | 21GiBps |
| ストレージのみ | 64GiBps | 21GiBps |

メタデータ容量のサイジングの推定値は、500GBの容量でBeeGFSに約1億5、000万ファイルを格納できる「経験則」に基づいています。（詳細については、のBeeGFSのマニュアルを参照してください "システム要件". ）

アクセス制御リストやディレクトリあたりのディレクトリ数やファイル数などの機能を使用すると、メタデータスペースの消費速度にも影響します。ストレージ容量の概算値は、使用可能なドライブ容量と、RAID 6およびXFSオーバーヘッドを考慮しています。

### メタデータおよびストレージのビルディングブロックの容量サイジング

次の表に、メタデータ用の推奨容量およびストレージのビルディングブロックを示します。

| ドライブサイズ（**2+2 RAID 1**）のメタデータボリュームグループ | メタデータ容量（ファイル数） | ドライブサイズ（**8+2 RAID 6**）ストレージボリュームグループ | ストレージ容量（ファイル内容） |
|---|---|---|---|
| 1.92TB | 1,938,577,200 | 1.92TB | 51.77TB |
| 3.84TB | 3,880,388,400 | 3.84TB | 103.55TB |
| 7.68TB | 8、125、278,000 | 7.68TB | 216.74TB |
| 15.3TB | 17、269、854、000 | 15.3TB | 460.60TB |

メタデータとストレージビルディングブロックのサイジングを行う場合、メタデータボリュームグループとストレージボリュームグループに使用するドライブを小型化することでコストを削減できます。

ストレージ専用のビルディングブロックの容量サイジング

次の表に、ストレージ専用のビルディングブロックの容量をルールベースの設定値で示します。

| ドライブサイズ（**10+2 RAID 6**）のストレージボリュームグループ | ストレージ容量（ファイル内容） |
|---|---|
| 1.92TB | 59.89TB |
| 3.84TB | 119.80TB |
| 7.68TB | 251.89TB |
| 15.3TB | 538.55TB |

グローバルファイルロックが有効になっていないかぎり、ベース（1番目）のビルディングブロックに管理サービスを含める場合のパフォーマンスと容量のオーバーヘッドは最小限です。

## パフォーマンスの調整

## BeeGFS解決策 には、検証テストに基づいたパフォーマンス調整の推奨事項が含まれます。

BeeGFSは設定不要で妥当なパフォーマンスを提供しますが、パフォーマンスを最大化するために推奨されるチューニングパラメータを一連開発しました。これらのパラメータには、基盤となるEシリーズのブロックノードの機能と、共有ディスクのHAアーキテクチャでBeeGFSを実行するために必要な特別な要件が考慮されています。

ファイルノードのパフォーマンス調整

設定可能なチューニングパラメータには、次のものがあります。

1. *ファイルノードのUEFI/BIOSのシステム設定。*パフォーマンスを最大化するには、ファイルノードとして使用するサーバーモデルのシステム設定を構成することをお勧めします。システム設定は、ベースボード管理コントローラ（BMC）が提供するセットアップユーティリティ（UEFI / BIOS）またはRedfish APIを使用してファイルノードをセットアップするときに設定します。

   システム設定は、ファイルノードとして使用するサーバーモデルによって異なります。使用中のサーバモデルに基づいて、設定を手動で行う必要があります。検証済みのLenovo SR665 V3ファイルノードのシステム設定を構成する方法については、以下を参照してください。 "パフォーマンスのファイルノードシステム設定を調整します" 。

2. *必須構成パラメータのデフォルト設定。*必要な構成パラメータは、BeeGFSサービスの構成方法、およびペースメーカーによるEシリーズボリューム（ブロックデバイス）のフォーマットとマウント方法に影響します。これらの必須設定パラメータには、次のものがあります。

   - BeeGFSサービスの設定パラメータ

     必要に応じて、設定パラメータのデフォルト設定を上書きできます。特定のワークロードやユースケースに合わせて調整できるパラメータについては、を参照して "BeeGFSサービスの設定パラメータ" ください。

   - ボリュームのフォーマットとマウントのパラメータは推奨されるデフォルトに設定されており、高度なユースケースでのみ調整する必要があります。デフォルト値では、次の処理が実行されます。

- ターゲットタイプ（管理、メタデータ、ストレージなど）、基盤となるボリュームのRAID構成とセグメントサイズに基づいて、ボリュームの初期フォーマットを最適化します。
- Pacemakerが各ボリュームをマウントする方法を調整し、変更がEシリーズのブロックノードにすぐにフラッシュされるようにします。これにより、アクティブな書き込みが進行中の状態でファイルノードに障害が発生してもデータが失われることはありませ

  特定のワークロードやユースケースに合わせて調整できるパラメータについては、を参照して "ボリュームのフォーマットと構成パラメータのマウント"ください。

3. *ファイルノードにインストールされているLinux OSのシステム設定。*の手順4でAnsibleインベントリを作成するときに、Linux OSのデフォルトのシステム設定を上書きできます "Ansibleインベントリを作成します"。

   デフォルトの設定を使用してNetApp解決策 のBeeGFSが検証されましたが、特定のワークロードやユースケースに合わせて調整するように変更することができます。変更可能なLinux OSのシステム設定には、次のようなものがあります。

   - Eシリーズのブロックデバイスの I/Oキュー。

     BeeGFSターゲットとして使用されるEシリーズのブロックデバイスでは、次の目的でI/Oキューを設定できます。

     - デバイスタイプ（NVMe、HDDなど）に基づいてスケジューリングアルゴリズムを調整します。
     - 未処理の要求数を増やします。
     - 要求サイズを調整します。
     - 先読み動作を最適化します。
   - 仮想メモリの設定。

     仮想メモリの設定を調整して、最適な持続ストリーミングパフォーマンスを得ることができます。

   - CPU設定。

     CPU周波数ガバナおよびその他のCPU構成を調整して、パフォーマンスを最大限に高めることができます。

   - 読み取り要求のサイズ

     NVIDIA HCAの最大読み取り要求サイズを増やすことができます。

ブロックノードのパフォーマンス調整

特定のBeeGFSビルディングブロックに適用される構成プロファイルに基づいて、ブロックノードに設定されたボリュームグループが少し変化します。たとえば、24ドライブのEF600ブロックノードがある場合、次のようになります。

- BeeGFSの管理、メタデータ、ストレージサービスなど、単一のベースビルディングブロックの場合は次の手順を実行します。
  - BeeGFS管理およびメタデータサービス用に2+2 RAID 10ボリュームグループが1つ
  - BeeGFSストレージサービス用に8+2 RAID 6ボリュームグループが2つ

- BeeGFSメタデータとストレージビルディングブロックの場合：
  - BeeGFSメタデータサービス用に2+2 RAID 10ボリュームグループが1つ
  - BeeGFSストレージサービス用に8+2 RAID 6ボリュームグループが2つ
- BeeGFSストレージのみのビルディングブロックの場合：
  - BeeGFSストレージサービス向けに10+2 RAID 6ボリュームグループが2つ

> BeeGFSでは管理とメタデータのストレージ容量がストレージよりも大幅に少ないため、RAID 10ボリュームグループには小さいドライブを使用する方法も1つあります。より小さいドライブは、最も外側のドライブスロットに取り付ける必要があります。詳細については、を参照してください "導入手順"。

これらはすべてAnsibleベースの導入で設定され、次のようなパフォーマンスや動作を最適化するために一般的に推奨されるいくつかの設定とともに使用されます。

- グローバルキャッシュブロックサイズを32KiBに調整し、デマンドベースのキャッシュフラッシュを80%に調整しています。
- 自動ロードバランシングを無効にする（コントローラのボリュームの割り当ては意図したとおりに維持する）。
- 読み取りキャッシュの有効化と先読みキャッシュの無効化
- ミラーリングあり、バッテリバックアップを必要とする書き込みキャッシュを有効にして、ブロックノードコントローラの障害後もキャッシュを維持できるようにします。
- ボリュームグループにドライブを割り当てる順序を指定し、使用可能なドライブチャネル間でI/Oのバランスを調整します。

## 大容量のビルディングブロック

標準のBeeGFS解決策 設計は、ハイパフォーマンスのワークロードを念頭に置いて構築されます。大容量のユースケースを検討している場合は、ここで紹介するような設計やパフォーマンスの特徴の違いを確認する必要があります。

### ハードウェアとソフトウェアの設定

大容量ビルディングブロック用のハードウェアとソフトウェアの設定は標準です。ただし、EF600コントローラをEF300コントローラに交換し、各ストレージアレイ用に60本のドライブを搭載した1 ~ 7個のIOM拡張トレイを取り付けることができます。ビルディングブロックあたり2～14台の拡張トレイを搭載。

大容量のビルディングブロック設計を導入する場合、使用されるのは、BeeGFSの管理、メタデータ、ストレージの各サービスで構成される基本ビルディングブロック形式の設定のみです。コスト効率を高めるために、大容量のストレージノードでは、EF300コントローラエンクロージャ内のNVMeドライブにメタデータボリュームをプロビジョニングし、拡張トレイ内のNL-SASドライブにストレージボリュームをプロビジョニングする必要があります。

[]

### サイジングガイドライン

これらのサイジングガイドラインでは、大容量のビルディングブロックが、ベースEF300エンクロージャ内の

メタデータ用に2+2のNVMe SSDボリュームグループ1つ、ストレージ用にIOM拡張トレイあたり6x8+2のNL-SASボリュームグループ1つで構成されていることを前提としています。

| ドライブサイズ（大容量**HDD**） | **BB**あたりの容量**(1**トレイ**)** | **BB**あたりの容量（トレイ**×2**） | **BB**あたりの容量（トレイ**×3**） | **BB**あたりの容量**(4**トレイ**)** |
|---|---|---|---|---|
| 4TB | 439TB | 878 TB | 1、317 TB | 1、1756 TB |
| 8 TB | 878 TB | 1、1756 TB | 2、634 TB | 3、512TB |
| 10 TB | 1、097 TB | 2195 TB | 3、3292 TB | 4390 TB |
| 12TB | 1、317 TB | 2、634 TB | 3951 TB | 5268 TB |
| 16 TB | 1、1756 TB | 3、512TB | 5268 TB | 7024 TB |
| 18 TB | 1975 TB | 3951 TB | 5927TB | 7902 TB |

# 解決策 を導入します

## 導入の概要

## NetApp上のBeeGFSは、NetAppのBeeGFSビルディングブロック設計とAnsibleを使用して、検証済みのファイルノードとブロックノードに導入できます。

### **Ansible**のコレクションとロール

BeeGFS on NetAppソリューションは、アプリケーションの導入を自動化する一般的なIT自動化エンジンであるAnsibleを使用して導入されます。Ansibleでは、総称してインベントリと呼ばれる一連のファイルを使用して、導入するBeeGFSファイルシステムをモデル化します。

Ansibleを使用すると、NetAppなどの企業は、Ansible Galaxyで利用可能なコレクションを使用して、組み込み機能を拡張できます（を参照 "NetApp EシリーズBeeGFSのコレクション"）。コレクションには、特定の機能やタスク（Eシリーズボリュームの作成など）を実行するモジュールや、複数のモジュールやその他のロールを呼び出すことができるロールが含まれます。この自動化されたアプローチにより、BeeGFSファイルシステムと基盤となるHAクラスタの導入に要する時間が短縮されます。さらに、クラスタとBeeGFSファイルシステムの保守と拡張が容易になります。

詳細については、を参照してください "Ansibleのインベントリを確認できます"。

NetApp解決策 へのBeeGFSの導入には多数の手順が含まれるため、解決策 の手動による導入はサポートされません。

### **BeeGFS**ビルディングブロックの構成プロファイル

導入手順では、次の設定プロファイルについて説明します。

- 管理、メタデータ、ストレージサービスを含む1つのベースとなるビルディングブロックです。
- メタデータとストレージサービスを含む2つ目のビルディングブロック。
- ストレージサービスのみを含む3つ目のビルディングブロック。

これらのプロファイルは、NetApp BeeGFSビルディングブロックに推奨されるすべての構成プロファイルを

示しています。メタデータとストレージのビルディングブロックまたはストレージサービスのみのビルディングブロックの数は、環境ごとに容量とパフォーマンスの要件に応じて変わる場合があります。

導入手順の概要

の導入では、次の作業を実行します。

ハードウェアの導入

1. 各ビルディングブロックを物理的に組み立てます。
2. ラックに設置してケーブルを配線する。詳細な手順については、を参照してください "ハードウェアを導入"。

ソフトウェアの導入

1. "ファイルノードとブロックノードをセットアップ"。
   - ファイルノードにBMCのIPを設定します
   - サポートされているオペレーティングシステムをインストールし、ファイルノードに管理ネットワークを設定します
   - ブロックノードに管理IPを設定します
2. "Ansibleコントロールノードをセットアップします"。
3. "パフォーマンスのシステム設定を調整します"。
4. "Ansibleインベントリを作成します"。
5. "BeeGFSビルディングブロックのAnsibleインベントリを定義します"。
6. "Ansibleを使用してBeeGFSを導入します"。
7. "BeeGFSクライアントを設定します"。

> ⓘ 展開手順には、テキストをファイルにコピーする必要があるいくつかの例が含まれています。特定のデプロイメントに合わせて変更する必要がある、または変更できる内容については、「#」または「//」文字で示されるインラインコメントに注意してください。例:
>
> ```
> `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"`
> ```

導入に関する推奨事項にバリエーションを伴う派生アーキテクチャ：

- "大容量ビルディングブロック"

## Ansibleのインベントリを確認できます

導入を開始する前に、Ansibleがどのように設定され、BeeGFS on NetAppソリューションの導入に使用されるかについて理解しておいてください。

Ansibleインベントリは、導入するBeeGFSファイルシステムのファイルノードとブロックノードをリストし

たディレクトリ構造です。これには、目的のBeeGFSファイルシステムを記述するホスト、グループ、および変数が含まれます。Ansibleインベントリは、Ansibleコントロールノードに格納する必要があります。コントロールノードとは、Ansibleプレイブックの実行に使用されるファイルノードとブロックノードにアクセスできる任意のマシンです。サンプルインベントリはからダウンロードできます "NetApp EシリーズBeeGFS GitHub"。

**Ansible**のモジュールとロール

Ansibleインベントリに記載されている構成を適用するには、エンドツーエンドのソリューションを導入するNetApp EシリーズAnsibleコレクション（から入手可能）に含まれているさまざまなAnsibleモジュールとロールを使用し "NetApp EシリーズBeeGFS GitHub"ます。

NetApp EシリーズAnsibleコレクションの各ロールは、NetApp解決策 上のBeeGFSをエンドツーエンドで導入します。これらのロールでは、NetApp E-Series SANtricity 、Host、およびBeeGFSの各コレクションを使用して、HA（High Availability）を使用してBeeGFSファイルシステムを設定できます。その後、ストレージをプロビジョニングしてマッピングし、クラスタストレージを使用できる状態にします。

ロールには詳細なドキュメントが含まれていますが、導入手順では、第2世代のBeeGFSビルディングブロック設計を使用して、ロールを使用してNetApp Verified Architectureを導入する方法について説明します。

> ⓘ 導入手順でAnsibleの使用経験が十分に細かい情報を提供できるようにすることは前提条件ではありませんが、Ansibleと関連する用語についてある程度理解している必要があります。

**BeeGFS HA**クラスタのインベントリレイアウト

Ansibleのインベントリ構造を使用してBeeGFS HAクラスタを定義します。

Ansibleの使用経験がある方は、BeeGFS HAロールには、各ホストに適用される変数（ファクト）を検出するためのカスタムメソッドが実装されていることに注意してください。この設計により、Ansibleインベントリの構造化が簡素化され、複数のサーバで実行できるリソースが記述されます。

Ansibleインベントリは通常、およびのファイルと `inventory.yml`、 `group_vars`ホストを特定のグループ（場合によっては他のグループ）に割り当てるファイルで構成され `host_vars`ます。

> ⓘ このサブセクションの内容を含むファイルは、例としてのみ作成しないでください。

この構成は構成プロファイルに基づいて事前定義されていますが、以下に示すように、すべてがAnsibleインベントリとしてどのようにレイアウトされるかについて一般的に理解しておく必要があります。

```
# 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.
```

サービスごとに'構成を記述するgroup_varsの下に追加ファイルが作成されます

```
# 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
```

このレイアウトでは、各リソースのBeeGFSサービス、ネットワーク、ストレージの構成を1か所で定義できます。バックグラウンドでは、BeeGFSロールは、このインベントリ構造に基づいて各ファイルおよびブロックノードに必要な設定を集約します。

> 各サービスのBeeGFS数値と文字列ノードIDは、グループ名に基づいて自動的に設定されます。したがって、グループ名を一意にするための一般的なAnsibleの要件に加えて、BeeGFSサービスを表すグループは、グループが表すBeeGFSサービスのタイプに一意の番号で終わる必要があります。たとえば、meta_01とstor_01は許可されますが、meta_01とmeta_01は許可されません。

## ベストプラクティスを確認

### NetApp解決策 にBeeGFSを導入する際は、ベストプラクティスのガイドラインに従ってください。

標準規則

Ansibleインベントリファイルを物理的に構築して作成する場合は、以下の標準的な規則に従ってください（詳細については、を参照してください） "Ansibleインベントリを作成します"）。

- ファイル・ノードのホスト名は'順番に番号が付けられ（H01-HN）'ラックの一番上に番号が小さい'下に数字が大きい

  たとえば、命名規則は `[location][row][rack]hN` 次のようになります。 `beegfs_01`

- 各ブロックノードは2台のストレージコントローラで構成され、コントローラごとに独自のホスト名が付けられます。

  ストレージアレイ名は、Ansibleインベントリの一部としてブロックストレージシステム全体を参照するために使用されます。ストレージアレイ名は順番に（A01-AN）番号を付け、各コントローラのホスト名はこの命名規則に基づいて付けます。

  たとえば、という名前のブロックノードは `ictad22a01` 通常、コントローラごとにホスト名を設定できます（やなど）があります `ictad22a01-a` `ictad22a01-b`が、Ansibleインベントリではと呼ばれます。 `netapp_01`

- 同じビルディングブロック内のファイルノードとブロックノードは、同じ番号方式を共有し、ラック内で互いに隣接しています。一番上に両方のファイルノードがあり、その下に両方のブロックノードが直接配置されています。

  たとえば、最初のビルディングブロックでは、ファイルノードH01とH02がどちらもブロックノードA01とA02に直接接続されています。ホスト名は、H01、H02、A01、およびA02です。

- ビルディングブロックは、ホスト名に基づいて順に配置されるため、番号の小さいホスト名はラックの上部に、番号の大きいホスト名は下部に配置されます。

  ここでは、ケーブルをラックスイッチの上部まで配線する時間を最小限に抑え、トラブルシューティングを簡単にするための標準的な導入方法を定義します。ラックの安定性に問題があるためにこれが許可されないデータセンターでは、逆の場合は、下から上にラックにデータを入力することができます。

### InfiniBandストレージネットワーク構成

各ファイルノードの半分のInfiniBandポートをブロックノードに直接接続します。残りの半分はInfiniBandスイッチに接続され、BeeGFSクライアント/サーバ接続に使用されます。BeeGFSクライアントおよびサーバに使用するIPoIBサブネットのサイズを決定する際には、コンピューティング/ GPUクラスタとBeeGFSファイルシステムの予想される増加を考慮する必要があります。推奨されるIP範囲から外れる必要がある場合は、1つのビルディングブロック内の各直接接続に一意のサブネットがあり、クライアント/サーバ接続に使用されるサブネットと重複しないことに注意してください。

#### 直接接続

各ビルディングブロック内のファイルノードとブロックノードは、常に次の表のIPを使用して直接接続されます。

> このアドレス指定方式は、次のルールに従っています。3番目のオクテットは常に奇数または偶数で、ファイルノードが奇数であるか偶数であるかによって異なります。

| ファイルノード | IBポート | IP アドレス | ブロックノード | IBポート | 物理IP | 仮想IP |
|---|---|---|---|---|---|---|
| 奇数（h1） | i1a | 192.168.1.10 | 奇数（c1） | 2A | 192.168.1.100 | 192.168.1.101 |
| 奇数（h1） | i2a | 192.168.3.10 | 奇数（c1） | 2A | 192.168.3.100 | 192.168.3.101 |
| 奇数（h1） | i3a | 192.168.5.10 | 偶数（C2） | 2A | 192.168.5.100 | 192.168.5.101 |
| 奇数（h1） | i4a | 192.168.7.10 | 偶数（C2） | 2A | 192.168.7.100 | 192.168.7.101 |
| 偶数（h2） | i1a | 192.168.2.10 | 奇数（c1） | 2B | 192.168.2.100 | 192.168.2.101 |

| ファイルノード | IBポート | IP アドレス | ブロックノード | IBポート | 物理IP | 仮想IP |
|---|---|---|---|---|---|---|
| 偶数（h2） | i2a | 192.168.4.10 | 奇数（c1） | 2B | 192.168.4.100 | 192.168.4.101 |
| 偶数（h2） | i3a | 192.168.6.10 | 偶数（C2） | 2B | 192.168.6.100 | 192.168.6.101 |
| 偶数（h2） | i4a | 192.168.8.10 | 偶数（C2） | 2B | 192.168.8.100 | 192.168.8.101 |

**BeeGFS**クライアント/サーバ**IPoIB**アドレス方式

各ファイルノードは、複数のBeeGFSサーバサービス（管理、メタデータ、またはストレージ）を実行します。各サービスを他のファイルノードに個別にフェイルオーバーできるようにするために、各サービスには両方のノード間で共有できる一意のIPアドレス（論理インターフェイスまたはLIFとも呼ばれます）が設定されます。

必須ではありませんが、この配置では、次のIPoIBサブネット範囲がこれらの接続に使用されていることが前提となり、次の規則を適用する標準アドレッシング方式を定義します。

- 2番目のオクテットは、ファイルノードのInfiniBandポートが奇数であるか偶数であるかに基づいて常に奇数になります。
- BeeGFSクラスタIPは常にxxxです。127.100.yyyまたは'xxx.128.100.yyy

> インバンドOS管理に使用されるインターフェイスに加えて、Corosyncでクラスタのハートビートおよび同期に追加のインターフェイスを使用できます。これにより、1つのインターフェイスが停止してもクラスタ全体が停止することはありません。

- BeeGFS管理サービスは'常に'xxx.yyy.101.0'または'xxx.yyy.102.0`になります
- BeeGFSメタデータ・サービスは常に「xxx.yyy.101.zzz」または「xxx.yyy.102.zzz」です。
- BeeGFSストレージサービスは、常に `xxx.yyy.103.zzz` または `xxx.yyy.104.zzz`です。
- アドレス範囲は'100.xxx.1.1'～'100.xxx.99.255'で'クライアント用に予約されています

**IPoIB**単一サブネットアドレッシング方式

この導入ガイドでは、に記載されている利点を考慮して、単一のサブネットスキーマを使用し "ソフトウェアアーキテクチャ"ます。

サブネット：**100.127.0.0/16**

次の表に、単一のサブネットの範囲（100.127.0.0/16）を示します。

| 目的 | InfiniBandポート | IPアドレスまたはIP範囲 |
|---|---|---|
| BeeGFS Cluster IP（BeeGFSクラスタIP） | i1bまたはi4b | 100.127.100.1-100.127.100.255 |
| BeeGFS Managementの略 | i1b | 100.127.101.0 |
| | i2b | 100.127.102.0 |
| BeeGFSメタデータ | i1bまたはi3b | 100.127.101.1-100.127.101.255 |
| | i2bまたはi4b | 100.127.102.1～100.127.102.255 |

| 目的 | **InfiniBand**ポート | **IP**アドレスまたは**IP**範囲 |
|---|---|---|
| BeeGFS Storage（BeeGFSストレージ） | i1bまたはi3b | 100.127.103.1-100.127.103.255 |
| | i2bまたはi4b | 100.127.104.1～100.127.104.255 |
| BeeGFSクライアント | （クライアントによって異なる） | 100.127.1.1～100.127.99.255 |

**IPoIB 2**サブネットアドレッシング方式

2つのサブネットアドレッシング方式は推奨されなくなりましたが、実装は可能です。推奨される2つのサブネット方式については、次の表を参照してください。

サブネット**A**：**100.127.0.0.0/16**

次の表に、サブネットAの範囲を示します。100.127.0.0.0/16

| 目的 | **InfiniBand**ポート | **IP**アドレスまたは**IP**範囲 |
|---|---|---|
| BeeGFS Cluster IP（BeeGFSクラスタIP） | i1b | 100.127.100.1-100.127.100.255 |
| BeeGFS Managementの略 | i1b | 100.127.101.0 |
| BeeGFSメタデータ | i1bまたはi3b | 100.127.101.1-100.127.101.255 |
| BeeGFS Storage（BeeGFSストレージ） | i1bまたはi3b | 100.127.103.1-100.127.103.255 |
| BeeGFSクライアント | （クライアントによって異なる） | 100.127.1.1～100.127.99.255 |

サブネット**B**：**100.128.0.0/16**

次の表に、サブネットBの範囲を示します。100.128.0.0/16

| 目的 | **InfiniBand**ポート | **IP**アドレスまたは**IP**範囲 |
|---|---|---|
| BeeGFS Cluster IP（BeeGFSクラスタIP） | i4b | 100.128.100.1~100.128.100.255 |
| BeeGFS Managementの略 | i2b | 100.128.102.0 |
| BeeGFSメタデータ | i2bまたはi4b | 100.128.102.1～100.128.102.255 |
| BeeGFS Storage（BeeGFSストレージ） | i2bまたはi4b | 100.128.104.1～100.128.104.255 |
| BeeGFSクライアント | （クライアントによって異なる） | 100.128.1.1～100.128.99.255 |

このNetApp Verified Architectureでは、上記の範囲のすべてのIPが使用されているわけではありません。一貫したIPアドレッシング方式を使用してファイルシステムを簡単に拡張できるように、IPアドレスを事前に割り当てる方法を示します。この方式では、BeeGFSファイルノードとサービスIDは既知のIP範囲の4番目のオクテットに対応します。ファイルシステムは、必要に応じて255ノード以上のノードやサービスを拡張できます。

## ハードウェアを導入

## 各ビルディングブロックは、HDR（200GB）InfiniBandケーブルを使用して2つのブロッ

クノードに直接接続された、検証済みの2つのx86ファイルノードで構成されます。

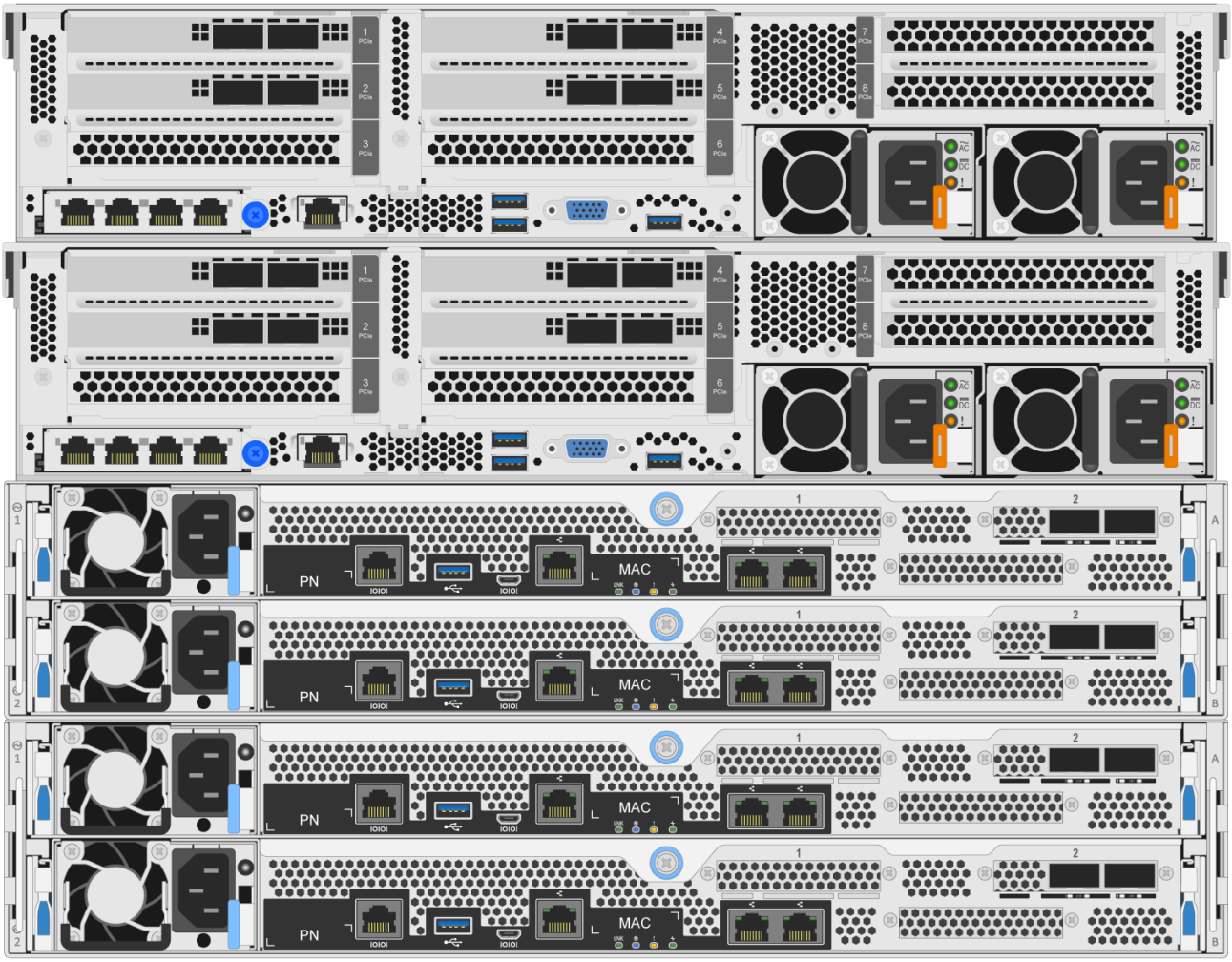
> フェイルオーバークラスタでクォーラムを確立するには、少なくとも2つのビルディングブロックが必要です。2ノードクラスタには、フェイルオーバーの正常な実行を妨げる可能性がある制限があります。3つ目のデバイスをTiebreakerとして組み込むことで、2ノードクラスタを構成できますが、このドキュメントではその設計については説明していません。

次の手順は、特に記載がないかぎり、クラスタ内の各ビルディングブロックについて同じです。これは、このビルディングブロックを使用してBeeGFSメタデータサービスとストレージサービスの両方を実行するか、ストレージサービスだけを実行するかに関係ありません。

手順

1. で指定したモデルを使用して、4つのホストチャネルアダプタ（HCA）で各BeeGFSファイルノードをセットアップします。 "技術要件"以下の仕様に従って、HCAをファイルノードのPCIeスロットに挿入します。
   - * Lenovo ThinkSystem SR665 V3サーバー：PCIeスロット1、2、4、5を使用します。
   - * Lenovo ThinkSystem SR665サーバー：PCIeスロット2、3、5、6を使用します。
2. デュアルポートの200GBホストインターフェイスカード（HIC）で各BeeGFSブロックノードを設定し、2台の各ストレージコントローラにHICを取り付けます。

   2つのBeeGFSファイルノードがBeeGFSブロックノードの上になるようにビルディングブロックをラックに配置します。次の図は、Lenovo ThinkSystem SR665 V3サーバをファイルノードとして使用するBeeGFSビルディングブロックの正しいハードウェア構成を示しています（背面図）。

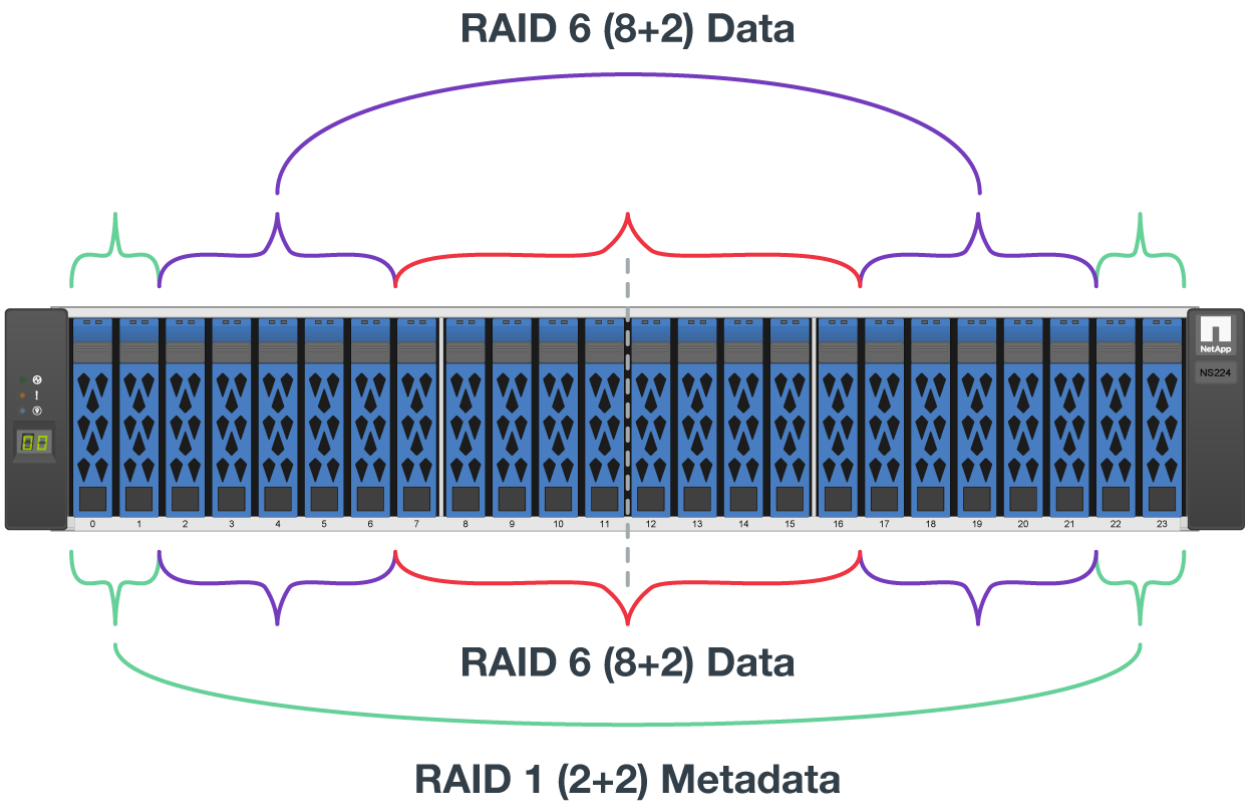



一般に、本番環境では電源装置を冗長PSUにする必要があります。

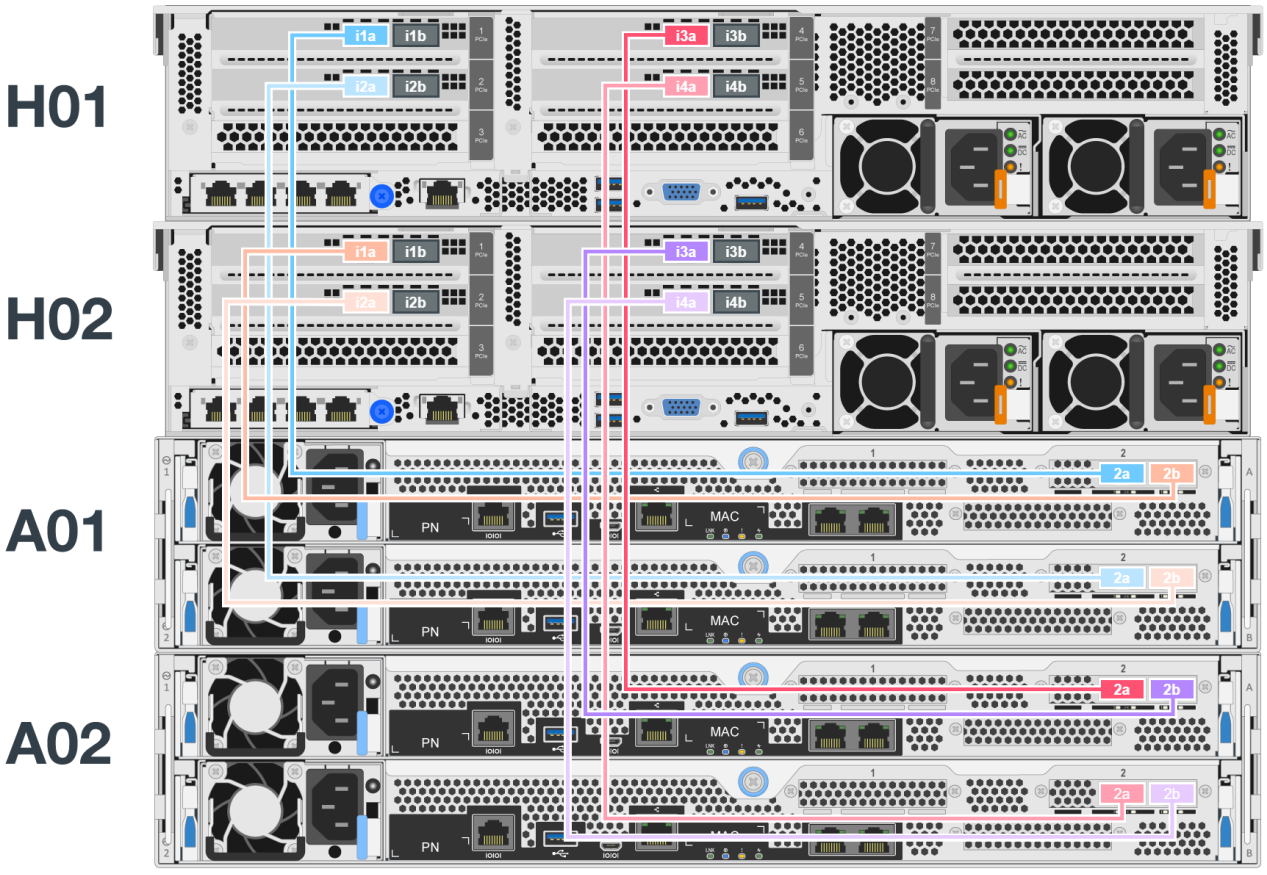
3. 必要に応じて、BeeGFSブロックノードのそれぞれにドライブを取り付けます。

   a. ビルディングブロックを使用してBeeGFSメタデータとストレージサービスを実行し、さらに小さいドライブをメタデータボリュームに使用する場合は、次の図に示すように、最も外側のドライブスロットにそれらが搭載されていることを確認します。

   b. すべてのビルディングブロック構成で、ドライブエンクロージャにフル装備されていない場合は、最適なパフォーマンスを得るために、同じ数のドライブがスロット0$_{11}$および$_{12}$23に装着されていることを確認してください。




4. 次の図に示すトポロジと一致するように、を使用してブロックノードとファイルノードを接続します "1M InfiniBand HDR 200GB直接接続銅ケーブル"。

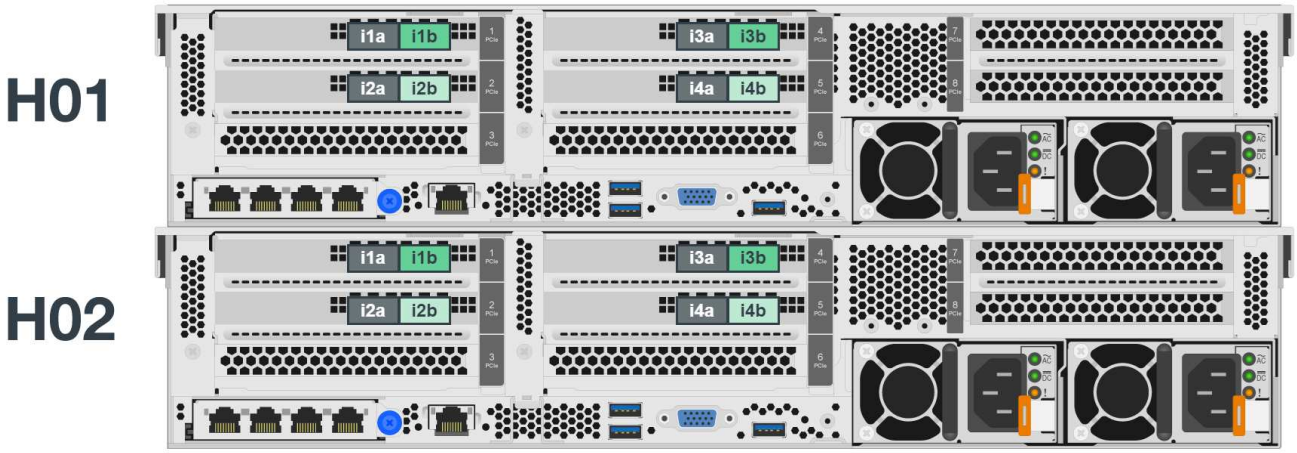


複数のビルディングブロックを横断するノードが直接接続されることはありません。各ビルディングブロックはスタンドアロンユニットとして扱われ、ビルディングブロック間のすべての通信はネットワークスイッチを介して行われます。

5. InfiniBandストレージスイッチに固有のを使用して、ファイルノードの残りのInfiniBandポートをストレージネットワークのInfiniBandスイッチに接続します "2M InfiniBandケーブル" 。

   スプリッタケーブルを使用してストレージスイッチとファイルノードを接続する場合は、スイッチから1本のケーブルが分岐し、ライトグリーンで示されているポートに接続する必要があります。別のスプリッタケーブルがスイッチから分岐し、濃い緑色で示されているポートに接続する必要があります。

   また、冗長スイッチを使用するストレージネットワークの場合、薄い緑色のポートは1つのスイッチに接続し、濃い緑色のポートは別のスイッチに接続します。




6. 必要に応じて、同じケーブル配線ガイドラインに従って追加のビルディングブロックをアセンブルします。

1台のラックに導入できるビルディングブロックの総数は、各サイトで利用可能な電力と冷却量によって異なります。

## ソフトウェアを導入

### ファイルノードとブロックノードをセットアップ

ほとんどのソフトウェア設定タスクはネットアップが提供するAnsibleコレクションを使用して自動化されていますが、各サーバのBaseboard Management Controller（BMC；ベースボード管理コントローラ）でネットワークを設定し、各コントローラの管理ポートを設定する必要があります。

#### ファイルノードをセットアップします

1. 各サーバのベースボード管理コントローラ（BMC）のネットワークを設定します。

   検証済みLenovo SR665 V3ファイルノードのネットワーク設定方法については、を参照してください "Lenovo ThinkSystemのドキュメント"。

ベースボード管理コントローラ（BMC）は、サービスプロセッサとも呼ばれ、オペレーティングシステムがインストールされていない場合やアクセスできない場合でもリモートアクセスを提供できるさまざまなサーバプラットフォームに組み込まれているアウトオブバンド管理機能の一般的な名前です。ベンダーは通常、この機能を独自のブランドで販売しています。たとえば、Lenovo SR665では、BMCは_Lenovo XClarity Controller（XCC）_と呼ばれています。

2. 最大のパフォーマンスを得るためにシステムを設定します。

   システム設定は、UEFIセットアップ（旧BIOS）を使用するか、多くのBMCが提供するRedfish APIを使用して設定します。システム設定は、ファイルノードとして使用するサーバモデルによって異なります。

   検証済みのLenovo SR665 V3ファイルノードのシステム設定を構成する方法については、以下を参照してください。 "パフォーマンスのシステム設定を調整します" 。

3. Red Hat Enterprise Linux (RHEL) 9.4 をインストールし、Ansible コントロール ノードからの SSH 接続を含むオペレーティング システムの管理に使用するホスト名とネットワーク ポートを構成します。

   この時点では、InfiniBandポートにIPを設定しないでください。

   厳密には必須ではありませんが、以降のセクションでは、ホスト名には順番に番号が付けられ（h1-hNなど）、奇数ホストと偶数ホストで完了する必要のあるタスクを参照するようにしています。

4. Red Hat Subscription Manager を使用してシステムを登録およびサブスクライブし、公式 Red Hat リポジトリからの必要なパッケージのインストールを許可し、サポートされているバージョンの Red Hat への更新を制限します。 `subscription-manager release --set=9.4` 。手順については、およびを参照してください "RHELシステムを登録および登録する方法" "更新を制限する方法"。

5. ハイアベイラビリティに必要なパッケージを含むRed Hatリポジトリを有効にします。

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

6. ガイドの使用"ファイルノードアダプタファームウェアの更新"で推奨されているバージョンにすべてのHCAファームウェアを更新します"テクノロジ要件"。

ブロックノードをセットアップする

各コントローラの管理ポートを設定してEF600ブロックノードをセットアップします。

1. 各EF600コントローラに管理ポートを設定します。

   ポートの設定手順については、を参照して "Eシリーズドキュメントセンター"ください。

2. 必要に応じて、各システムのストレージアレイ名を設定します。

   名前を設定すると、以降のセクションで各システムを簡単に参照できるようになります。アレイ名の設定手順については、を参照して "Eシリーズドキュメントセンター"ください。

必須ではありませんが、以降のトピックでは、ストレージアレイ名には必ず順番に番号を付け（c1-CNなど）、奇数のシステムでも偶数のシステムでも完了する必要がある手順を参照してください。

パフォーマンスのファイルノードシステム設定を調整します

# パフォーマンスを最大化するには、ファイルノードとして使用するサーバーモデルでシステム設定を構成することをお勧めします。

システム設定は、ファイルノードとして使用するサーバーモデルによって異なります。この項では、検証済みLenovo ThinkSystem SR665サーバーファイルノードのシステム設定を構成する方法について説明します。

### UEFIインターフェイスを使用して、システム設定を調整します

Lenovo SR665 V3 サーバーのシステム ファームウェアには、UEFI インターフェイスを通じて設定できる多数のチューニング パラメーターが含まれています。これらのチューニングパラメータは、サーバの機能とサーバのパフォーマンスのすべての側面に影響を与える可能性があります。

UEFI Setup（UEFIセットアップ）> System Settings（システム設定）*で、次のシステム設定を調整します。

**Operating Mode**（操作モード）メニュー

| システム設定 | 「」に変更します |
|---|---|
| 動作モード | カスタム |
| CTDP | 手動 |
| CTDPマニュアル | 350 |
| パッケージの電力制限 | 手動 |
| 効率モード | 無効にします |
| GLOBAL-Cstate-Controlの略 | 無効にします |
| SOCの状態 | P0 |
| DF C -状態 | 無効にします |
| P状態 | 無効にします |
| Memory Power Down Enable (メモリの電源オフ有効 | 無効にします |
| ソケットごとのNUMAノード | NPS1 |

デバイスと**I/O**ポートのメニュー

| システム設定 | 「」に変更します |
| --- | --- |
| IOMMUを使用します | 無効にします |

電源メニュー

| システム設定 | 「」に変更します |
| --- | --- |
| PCIeパワーブレーキ | 無効にします |

**[**プロセッサ**]**メニュー

| システム設定 | 「」に変更します |
| --- | --- |
| グローバルCステートコントロール | 無効にします |
| DF C -状態 | 無効にします |
| SMTモード | 無効にします |
| CPPC | 無効にします |

**Redfish API**を使用して、システム設定を調整します

UEFIセットアップのほかに、Redfish APIを使用してシステム設定を変更することもできます。

```
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スキーマの詳細については、を参照してください "DMTFのWebサイト"。

**Ansible**コントロールノードをセットアップします

## Ansible制御ノードをセットアップするには、BeeGFS on NetAppソリューション用に導入されたすべてのファイルノードとブロックノードにネットワークアクセスできる仮想マシンまたは物理マシンを指定する必要があります。

推奨されるパッケージバージョンのリストについては、を参照して"技術要件"ください。次の手順はUbuntu 22.04でテストされました。使用しているLinuxディストリビューションに固有の手順については、を参照してください "Ansibleのドキュメント"。

1. Ansibleコントロールノードから、次のPythonおよびPython仮想環境パッケージをインストールします。

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

2. Python 仮想環境を作成します。

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

3. 仮想環境をアクティブ化します。

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

4. アクティブ化された仮想環境内に必要なPythonパッケージをインストールします。

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

5. Ansible Galaxyを使用してBeeGFSコレクションをインストールします。

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

6. インストールされているAnsible、Python、BeeGFSコレクションのバージョンがに一致することを確認します"技術要件"。

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

7. パスワードレスSSHを設定して、AnsibleがAnsibleコントロールノードからリモートBeeGFSファイルノードにアクセスできるようにします。
   a. Ansibleコントロールノードで、必要に応じて公開鍵のペアを生成します。

```
ssh-keygen
```

   b. 各ファイルノードへのパスワードレスSSHを設定します。

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

ブロックノードにパスワードなしのSSHを設定しないでください。サポートされていません。

**Ansible**インベントリを作成します

ファイルノードとブロックノードの設定を定義するには、導入するBeeGFSファイルシステムを表すAnsibleインベントリを作成します。インベントリには、目的のBeeGFSファイルシステムを記述するホスト、グループ、および変数が含まれます。

手順**1**：すべてのビルディングブロックの構成を定義します

どの構成プロファイルを個別に適用できるかに関係なく、環境 のすべての構成ブロックを定義します。

作業を開始する前に

- 導入環境に適したサブネットアドレッシング方式を選択します。に記載されている利点のため、 "ソフトウェアアーキテクチャ"単一のサブネットアドレッシング方式を使用することを推奨します。

手順

1. Ansibleの制御ノードで、Ansibleのインベントリファイルとプレイブックファイルの格納に使用するディレクトリを特定します。

   特に記載がないかぎり、この手順および以降の手順で作成するすべてのファイルとディレクトリは、このディレクトリを基準にして作成されます。

2. 次のサブディレクトリを作成します。

   「host_vars」

   'group_vars'

   「パッケージ」

3. クラスタパスワード用のサブディレクトリを作成し、Ansible Vaultでファイルを暗号化して保護します（を参照） "Ansible Vaultを使用したコンテンツの暗号化"。

   a. サブディレクトリを作成し `group_vars/all`ます。

   b. `group_vars/all`ディレクトリに、という名前のパスワードファイルを作成し `passwords.yml`ます。

   c. 設定に応じて、すべてのユーザ名およびパスワードパラメータを置き換えて、に次の情報を入力 `passwords.yml file`します。

```
# 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. プロンプトが表示されたら、を実行して `ansible-vault encrypt passwords.yml`ボルトパスワードを設定します。

手順**2**：個々のファイルノードとブロックノードの設定を定義する

環境 の個々のファイルノードおよび個々のビルディングブロックノードの構成を定義します。

1. 「host_vars/`」で、「<hostname>.yml」という名前のBeeGFSファイルノードごとに次の内容のファイルを作成します。BeeGFSクラスタのIPおよび奇数で終わるホスト名と偶数で終わるホスト名には、内容に関する注意を払って入力してください。

   最初は、ファイルノードのインターフェイス名が、ここに記載されている名前と一致しています（ib0やibs1f0など）。これらのカスタム名は、で設定します [手順4：すべてのファイルノードに適用する設定を定義する]。

```
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クラスタをすでに導入している場合は、静的に設定されたIPアドレス（NVMe/IBで使用するクラスタIPやIPなど）を追加または変更する前に、クラスタを停止する必要があります。これは、変更が適切に反映され、クラスタの処理が中断されないようにするために必要です。

2. 「host_vars/`」で、「<hostname>.yml」という名前のBeeGFSブロックノードごとにファイルを作成し、次の内容を入力します。

   ストレージアレイ名の末尾が奇数で偶数である場合は、内容に特に注意してください。

   ブロックノードごとに1つのファイルを作成し、2つのコントローラ（通常はA）のうちの1つに「<MANAGEMENT _IP>」を指定します。

```
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
```

手順**3**：すべてのファイルノードとブロックノードに適用する設定を定義する

グループに対応するファイル名に'GROLE_vars'の下にあるホストのグループに共通する構成を定義できますこれにより、複数の場所で共有設定を繰り返す必要がなくなります。

このタスクについて

ホストは複数のグループに含めることができ、実行時に、Ansibleは、変数の優先順位ルールに基づいて、特定のホストに適用する変数を選択します。（これらのルールの詳細については、Ansibleのドキュメントを参照してください "変数を使用します". ）

ホストとグループの割り当ては、実際のAnsibleインベントリファイルに定義されます。このファイルは、この手順 の末尾に作成されます。

ステップ

Ansibleでは、すべてのホストに適用する構成は「all」というグループで定義できます。次の内容で'ファイル'group_vars/all.yml'を作成します

```
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"
```

手順**4**：すべてのファイルノードに適用する設定を定義する

ファイル・ノードの共有構成は'ha_cluster'というグループで定義されますこのセクションの手順では'group_vars/ha_cluster.yml`ファイルに含める必要がある構成を構築します

手順

1. ファイルの最上部で'ファイルノードのsudoユーザーとして使用するパスワードを含むデフォルトを定義します

```
### 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"
```

> がすでに存在する `root`場合は `ansible_ssh_user`、必要に応じてを省略し、Playbookの実行時にオプションを指定 `--ask-become-pass`できます `ansible_become_password`。

2. 必要に応じて、ハイアベイラビリティ（HA）クラスタの名前を設定し、クラスタ内通信用のユーザを指定します。

   プライベートIPアドレッシング方式を変更する場合は、デフォルトの「beegfs_ha_mgmtd_floating_ip」も更新する必要があります。これは、後でBeeGFS Managementリソースグループに設定する内容と一致している必要があります。

   「beegfs_alert_email_list」を使用して、クラスタ・イベントのアラートを受信する電子メールを1つ以上指定します。

```
### 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
```

一見冗長に見えても’beegfs_ha_gmtd_floating_ip’は'1つのHAクラスタを超えてBeeGFSファイルシステムを拡張する場合に重要です以降のHAクラスタは、BeeGFS管理サービスを追加せずに導入され、最初のクラスタが提供する管理サービスをポイントします。

3. フェンシングエージェントを設定します。（詳細については、を参照してください "Red Hatハイアベイラビリティクラスタでフェンシングを設定します"）。次の出力は、一般的なフェンシングエージェントの設定例を示しています。次のいずれかのオプションを選択します。

   この手順では、次の点に注意してください。

   ◦ フェンシングはデフォルトで有効になっていますが、フェンシングエージェント_を設定する必要があ

ります。

- 'pcmk_host_map'または'pcmk_host_listに指定されている`<hostname>は'Ansibleインベントリ内のホスト名に対応している必要があります
- フェンシングなしでBeeGFSクラスタを実行することは、特に本番環境ではサポートされません。これは、ブロックデバイスなどのリソース依存関係を含むBeeGFSサービスが問題 によってフェイルオーバーする際に、ファイルシステムの破損やその他の望ましくない動作や予期しない動作を引き起こす複数のノードによる同時アクセスのリスクがないことを主に保証するためです。フェンシングを無効にする必要がある場合は'BeeGFS HAロールの入門ガイドの一般的な注意事項を参照して'ha_cluster.ymlで'beegfs_cluster_crm_config_options[stonith -enabled "]をfalseに設定します
- 複数のノードレベルのフェンシングデバイスがあり、BeeGFS HAロールでは、Red Hat HAパッケージリポジトリで使用可能なフェンシングエージェントを設定できます。可能な場合は、無停電電源装置（UPS）またはラック配電装置（rPDU）を経由するフェンシングエージェントを使用します。 ベースボード管理コントローラ（BMC）などの一部のフェンシングエージェントや、サーバに組み込まれているその他のライトアウトデバイスは、特定の障害シナリオではフェンス要求に応答しない場合があります。

```
### 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 OSで推奨されるパフォーマンス調整を有効にします。

   多くのユーザはパフォーマンスパラメータのデフォルト設定を確認できますが、特定のワークロードのデフォルト設定は必要に応じて変更できます。そのため、これらの推奨事項はBeeGFSロールに含まれますが、デフォルトでは有効になっていないため、ユーザーはファイルシステムに適用された調整を認識できません。

   パフォーマンス・チューニングを有効にするには’次のように指定

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

5. （オプション）Linux OSのパフォーマンス調整パラメータを必要に応じて調整できます。

   調整可能なチューニングパラメータの包括的なリストについては、のBeeGFS HAロールの「Performance Tuning Defaults」セクションを参照してください "EシリーズBeeGFS GitHubサイト"。 デフォルト値は、このファイルのクラスタ内のすべてのノードまたは個 々 のノードのファイルで上書きできます `host_vars` 。

6. ブロックノードとファイルノード間の200GB/HDR接続を完全に許可するには、NVIDIA Open Fabrics Enterprise Distribution（MLNX_OFED）のOpen Subnet Manager（OpenSM）パッケージを使用します。に記載されているMLNX_OFEDバージョンは "ファイルノードの要件" 、推奨されるOpenSMパッケージにバンドルされています。Ansibleを使用した導入もサポートされていますが、最初にすべてのファイルノードにMLNX_OFEDドライバをインストールする必要があります。

   a. 'group_vars/ha_cluster.yml'の次のパラメータを入力します(必要に応じてパッケージを調整します)

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

7. 論理InfiniBandポート識別子と基盤となるPCIeデバイスとのマッピングが一貫して行われるように'udev'ルールを設定します

   udevルールは'BeeGFSファイル・ノードとして使用される各サーバ・プラットフォームのPCIeトポロジーに固有のものである必要があります

   検証済みファイルノードには、次の値を使用します。

```
### 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. （オプション）メタデータターゲット選択アルゴリズムを更新します。

```
beegfs_ha_beegfs_meta_conf_ha_group_options:
  tuneTargetChooser: randomrobin
```

検証テストでは'通常'randomrobinを使用して'パフォーマンス・ベンチマーク中にテスト・ファイルがすべてのBeeGFSストレージ・ターゲットに均等に分散されるようにしました（ベンチマークの詳細については'BeeGFSのサイトを参照してください "BeeGFSシステムのベンチマーク"）。実際に使用されている場合は、原因 の番号が小さいターゲットが、番号の大きいターゲットよりも早くいっぱいになる可能性があります。「randomrobin」を省略し、デフォルトの「randomized」値を使用するだけで、利用可能なすべてのターゲットを利用しながら、優れたパフォーマンスを提供できるようになりました。

手順**5**：共通ブロックノードの設定を定義する

ブロック・ノードの共有構成は'eseries_storage_systems'というグループで定義されますこのセクションの手順では'group_vars/eseries_storage_systems.yml`ファイルに含める必要がある構成を構築します

手順

1. Ansible接続をローカルに設定し、システムパスワードを指定して、SSL証明書を検証するかどうかを指定します。（通常、AnsibleはSSHを使用して管理対象ホストに接続しますが、NetApp Eシリーズストレージシステムがブロックノードとして使用されている場合、モジュールはREST APIを使用して通信します

）。 ファイルの上部に、次の情報を追加します。

```
### 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. 最適なパフォーマンスを確保するには、に記載されているバージョンをブロックノードにインストールします "技術要件"。

   対応するファイルをからダウンロードします "ネットアップサポートサイト"。これらを手動でアップグレードするか'Ansibleコントロール・ノードのパッケージ/ディレクトリに含めてから'eseries_storage_systemesyml'に以下のパラメータを入力して'Ansibleを使用してアップグレードできます

```
# 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. から、ブロックノードに取り付けられているドライブで使用可能な最新のドライブファームウェアをダウンロードしてインストールし "ネットアップサポートサイト"ます。手動でアップグレードするか、Ansible制御ノードのディレクトリに追加してから、の次のパラメータを入力してAnsibleを使用してアップグレードできます `packages/ eseries_storage_systems.yml` 。

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

> eseries_drive_firmware_upgrade_drivesonlineを'false'に設定すると'アップグレードが高速化されますが'BeeGFSが導入されるまでは実行しないでくださいこれは、アプリケーションエラーを回避するために、アップグレード前にドライブへのすべてのI/Oを停止する必要があるためです。ボリュームを構成する前にオンライン・ドライブ・ファームウェア・アップグレードを実行しても問題が発生しないようにするには'この値を常にtrueに設定することを推奨します

4. パフォーマンスを最適化するには、グローバル構成に対して次の変更を行います。

```
# 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. ボリュームのプロビジョニングと動作を最適化するには、次のパラメータを指定します。

```
# 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」に指定する値はNetApp EF600ブロックノードに固有であり、新しいボリュームグループにドライブを割り当てる順序を制御します。この順序により、各グループへのI/Oがバックエンドドライブチャネル間で均等に分散されます。

**BeeGFS**ビルディングブロックの**Ansible**インベントリを定義します

## 一般的なAnsibleのインベントリ構造を定義したら、BeeGFSファイルシステムの各ビルディングブロックの設定を定義します。

導入手順では、管理、メタデータ、ストレージサービスなどの基本ビルディングブロックで構成されるファイルシステム、メタデータとストレージサービスを提供する2つ目のビルディングブロック、およびストレージ専用の3つ目のビルディングブロックで構成されるファイルシステムの導入方法を示します。

以下の手順は、BeeGFSファイルシステム全体の要件を満たすようにNetApp BeeGFSビルディングブロックを設定する際に使用する代表的な構成プロファイルをすべて示しています。

ⓘ このセクションと以降のセクションで、必要に応じて調整して、導入するBeeGFSファイルシステムを表すインベントリを作成します。特に、各ブロックまたはファイルノードを表すAnsibleホスト名と、ストレージネットワークに必要なIPアドレス指定方式を使用して、BeeGFSファイルノードとクライアントの数に合わせて拡張できます。

手順**1**：**Ansible**インベントリファイルを作成する

手順

1. 新しい'inventory.yml`ファイルを作成し'以下のパラメータを挿入します配置されたブロック・ノードを表すために'必要に応じて'eseries_storage_systems'の下のホストを置き換えますこれらの名前は'host_vars/<filename>.yml`に使用する名前に対応していなければなりません

```
# 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:
```

以降のセクションでは、「ha_cluster」の下に、クラスタで実行するBeeGFSサービスを表すAnsibleグループを追加で作成します。

手順**2**：管理、メタデータ、ストレージのビルディングブロックのインベントリを設定する

クラスタまたはベースビルディングブロックの最初のビルディングブロックには、メタデータサービスおよびストレージサービスとともにBeeGFS管理サービスが含まれている必要があります。

手順

1. 'inventory.yml'で'ha_cluster: children'の下に次のパラメータを入力します

```
      # 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. ファイル'group_vars/mgmt.yml'を作成し'以下を含めます

```
# 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/`」の下で、次のテンプレートを使用して「meta_01」から「meta_08」までのリソースグループのファイルを作成し、以下の表を参照する各サービスのプレースホルダ値を入力します。

```
# 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>
```

ボリュームサイズは、ストレージプール（ボリュームグループとも呼ばれる）全体に対する割合で指定します。SSDのオーバープロビジョニングのためのスペースを確保するために、各プールにある程度の空き容量を確保することを強く推奨します（詳細については、を参照してください） "NetApp EF600アレイの概要"）。ストレージプール'beegfs_m1_m2_m3_m6"は'管理サービス用のプールの容量の1%も割り当てますしたがって'ストレージ・プール内のメタデータ・ボリュームでは'beegfs_m1_m2_m5_m6'1.92TBまたは3.84TBのドライブを使用している場合'この値を21.25'7.65TBドライブの場合は22.25'15.3TBドライブの場合は'23.75'に設定します

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| meta_01.yml | 8015 | i1b：100.127.101.1/16 i2b：100.127.102.1 /16 | 0 | netapp_01 | beegfs_m1_m2_m5_m6 | A |
| meta_02.yml | 8025 | i2b：100.127.102.2/16 i1b：100.127.101.2 /16 | 0 | netapp_01 | beegfs_m1_m2_m5_m6 | B |

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| meta_03.yml | 8035 | i3b：100.127.101.3/16 i4b：100.127.102.3 /16 | 1. | netapp_02 | beegfs_m3_m4_m7_M8 | A |
| meta_04.yml | 8045 | i4b：100.127.102.4/16 i3b：100.127.101.4 /16 | 1. | netapp_02 | beegfs_m3_m4_m7_M8 | B |
| meta_05.yml | 8055 | i1b：100.127.101.5/16 i2b：100.127.102.5 /16 | 0 | netapp_01 | beegfs_m1_m2_m5_m6 | A |
| meta_06.yml | 8065 | i2b：100.127.102.6/16 i1b：100.127.101.6 /16 | 0 | netapp_01 | beegfs_m1_m2_m5_m6 | B |
| meta_07.yml | 8075 | i3b：100.127.101.7/16 i4b：100.127.102.7 /16 | 1. | netapp_02 | beegfs_m3_m4_m7_M8 | A |
| meta_08.yml | 8085 | i4b：100.127.102.8/16 i3b：100.127.101.8 /16 | 1. | netapp_02 | beegfs_m3_m4_m7_M8 | B |

4. 「group_vars/`」の下で、以下のテンプレートを使用して「stor_01」から「stor_08」のリソースグループ用のファイルを作成し、例を参照する各サービスのプレースホルダ値を入力します。

```
# 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>
```

正しいサイズについては、を参照してください "ストレージプールのオーバープロビジョニングの割合を推奨します"。

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_01.yml | 8013 | i1b：100.127.103.1/16 i2b：100.127.104.1 /16 | 0 | netapp_01 | beegfs_s1_s2 | A |
| STOR_02.yml | 8023 | i2b：100.127.104.2/16 i1b：100.127.103.2 /16 | 0 | netapp_01 | beegfs_s1_s2 | B |
| STOR_03.yml | 8033 | i3b：100.127.103.3/16 i4b：100.127.104.3 /16 | 1. | netapp_02 | beegfs_s3_s4 | A |

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_04.yml | 8043 | i4b：100.127.104.4/16 i3b：100.127.103.4 /16 | 1. | netapp_02 | beegfs_s3_s4 | B |
| STOR_05.yml | 8053 | i1b：100.127.103.5/16 i2b：100.127.104.5 /16 | 0 | netapp_01 | beegfs_s5_s6 | A |
| STOR_06.yml | 8063 | i2b：100.127.104.6/16 i1b：100.127.103.6 /16 | 0 | netapp_01 | beegfs_s5_s6 | B |
| STOR_07.yml | 8073 | i3b：100.127.103.7/16 i4b：100.127.104.7 /16 | 1. | netapp_02 | beegfs_s7_s8 | A |
| STOR_08.yml | 8083 | i4b：100.127.104.8/16 i3b：100.127.103.8 /16 | 1. | netapp_02 | beegfs_s7_s8 | B |

手順**3**：メタデータとストレージのビルディングブロックのインベントリを設定する

以下の手順では、BeeGFSメタデータとストレージビルディングブロックにAnsibleインベントリを設定する方法について説明します。

手順

1. 'inventory.yml'で'既存の構成の下に次のパラメータを入力します

```
        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/`」の下で、次のテンプレートを使用して「meta_09」から「meta_16」までのリソースグループのファイルを作成し、例を参照する各サービスのプレースホルダ値を入力します。

```
# 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>
```

正しいサイズについては、を参照してください "ストレージプールのオーバープロビジョニングの割合を推奨します"。

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| meta_09.yml | 8015 | i1b：100.127.101.9/16 i2b：100.127.102.9 /16 | 0 | netapp_03 | beegfs_m9_m10_m13_M14 | A |

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| meta_10.yml | 8025 | i2b：100.127.102.10/16 i1b：100.127.101.10 /16 | 0 | netapp_03 | beegfs_m9_m10_m13_M14 | B |
| meta_11.yml | 8035 | i3b：100.127.101.11/16 i4b：100.127.102.11 /16 | 1. | netapp_04 | BEegfs_M11_M12_M15_M16 | A |
| meta_12.yml | 8045 | i4b：100.127.102.12/16 i3b：100.127.101.12 /16 | 1. | netapp_04 | BEegfs_M11_M12_M15_M16 | B |
| meta_13.yml | 8055 | i1b：100.127.101.13/16 i2b：100.127.102.13 /16 | 0 | netapp_03 | beegfs_m9_m10_m13_M14 | A |
| meta_14.yml | 8065 | i2b：100.127.102.14/16 i1b：100.127.101.14 /16 | 0 | netapp_03 | beegfs_m9_m10_m13_M14 | B |
| meta_15.yml | 8075 | i3b：100.127.101.15/16 i4b：100.127.102.15 /16 | 1. | netapp_04 | BEegfs_M11_M12_M15_M16 | A |
| meta_16.yml | 8085 | i4b：100.127.102.16/16 i3b：100.127.101.16 /16 | 1. | netapp_04 | BEegfs_M11_M12_M15_M16 | B |

3. 「group_vars/」の下で、「stor_09」から「stor_16」までのリソースグループ用のファイルを作成し、例を参照する各サービスのプレースホルダ値を入力します。

```
# 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>
```

適切なサイズについては、 "ストレージプールのオーバープロビジョニングの割合を推奨します" ..

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_09.yml | 8013 | i1b：100.127.103.9/16 i2b：100.127.104.9 /16 | 0 | netapp_03 | beegfs_s9_s10 | A |
| STOR_10.yml | 8023 | i2b：100.127.104.10/16 i1b：100.127.103.10 /16 | 0 | netapp_03 | beegfs_s9_s10 | B |
| STOR_11.yml | 8033 | i3b：100.127.103.11/16 i4b：100.127.104.11 /16 | 1. | netapp_04 | beegfs_s11_s12を指定します | A |

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_12.yml | 8043 | i4b：100.127.104.12/16 i3b：100.127.103.12 /16 | 1. | netapp_04 | beegfs_s11_s12を指定します | B |
| STOR_13.yml | 8053 | i1b：100.127.103.13/16 i2b：100.127.104.13 /16 | 0 | netapp_03 | beegfs_S13_s14 | A |
| STOR_14.yml | 8063 | i2b：100.127.104.14/16 i1b：100.127.103.14 /16 | 0 | netapp_03 | beegfs_S13_s14 | B |
| STOR_15.yml | 8073 | i3b：100.127.103.15/16 i4b：100.127.104.15 /16 | 1. | netapp_04 | beegfs_s15_s16 | A |
| STOR_16.yml | 8083 | i4b：100.127.104.16/16 i3b：100.127.103.16 /16 | 1. | netapp_04 | beegfs_s15_s16 | B |

手順**4**：ストレージ専用のビルディングブロックのインベントリを設定する

以下の手順では、BeeGFSストレージ専用ビルディングブロックのAnsibleインベントリを設定する方法について説明します。メタデータとストレージのみのビルディング・ブロックの構成を設定する場合の主な違いは'すべてのメタデータ・リソース・グループを省略し'各ストレージ・プールの基準ドライブ数を10から12に変更することです

手順

1. 'inventory.yml'で'既存の構成の下に次のパラメータを入力します

```
# 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/」の下で、以下のテンプレートを使用して「`stor_17~`stor_24`のリソースグループのファイルを作成し、例を参照する各サービスのプレースホルダ値を入力します。

```
# 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>
```

適切なサイズについては、 "ストレージプールのオーバープロビジョニングの割合を推奨します"。

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_17.yml | 8013 | i1b：100.127.103.17/16 i2b：100.127.104.17 /16 | 0 | netapp_05 | beegfs_s17_s18 | A |
| STOR_18.yml | 8023 | i2b：100.127.104.18/16 i1b：100.127.103.18 /16 | 0 | netapp_05 | beegfs_s17_s18 | B |
| STOR_19.yml | 8033 | i3b：100.127.103.19/16 i4b：100.127.104.19 /16 | 1. | netapp_06 | beegfs_s19_s20 | A |

| ファイル名 | ポート | フローティングIP | NUMAゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|
| STOR_20.yml | 8043 | i4b：100.127.104.20/16 i3b：100.127.103.20 /16 | 1. | netapp_06 | beegfs_s19_s20 | B |
| STOR_21.yml | 8053 | i1b：100.127.103.21/16 i2b：100.127.104.21 /16 | 0 | netapp_05 | beegfs_S21_s22 | A |
| STOR_22.yml | 8063 | i2b：100.127.104.22/16 i1b：100.127.103.22 /16 | 0 | netapp_05 | beegfs_S21_s22 | B |
| STOR_23.yml | 8073 | i3b：100.127.103.23/16 i4b：100.127.104.23 /16 | 1. | netapp_06 | beegfs_S23_s24 | A |
| STOR_24.yml | 8083 | i4b：100.127.104.24/16 i3b：100.127.103.24 /16 | 1. | netapp_06 | beegfs_S23_s24 | B |

**BeeGFS**を導入します

## 構成の導入と管理には、Ansibleで実行するタスクが含まれた1つ以上のプレイブックを実行し、システム全体を目的の状態にする必要があります。

すべてのタスクを1つのプレイブックに含めることができますが、複雑なシステムでは、この作業が管理しにくくなります。Ansibleを使用すると、再利用可能なプレイブックと関連コンテンツ（デフォルトの変数、タスク、ハンドラなど）をパッケージ化する方法でロールを作成して配布できます。詳細については、Ansibleのドキュメントを参照してください "ロール"。

多くの場合、ロールは関連するロールとモジュールを含むAnsibleコレクションの一部として配布されます。このため、このプレイブックは、主に、NetApp Eシリーズの各種Ansibleコレクションに分散された複数のロールをインポートするだけです。

現在、2ノードクラスタとのクォーラムの確立時に問題が発生しないように、別のクォーラムデバイスをTiebreakerとして設定している場合を除き、BeeGFSを導入するには少なくとも2つのビルディングブロック（4つのファイルノード）が必要です。

手順

1. 新しい'playbook.yml`ファイルを作成し'次のものを含めます

```
# 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
```

このプレイブックは、Python 3がファイルノードにインストールされていることを確認し、提供されたAnsibleタグがサポートされていることを確認するいくつかの「pre_ttasks」を実行します。

2. BeeGFSを導入する準備ができたら'Ansibleプレイブックコマンドを使用してインベントリとプレイブックファイルを作成します

   配備ではすべての「pre_tasks」が実行され、ユーザーの確認を求めるプロンプトが表示された後、実際のBeeGFS配備に進みます。

   次のコマンドを実行して、必要に応じてフォークの数を調整します（以下の注記を参照）。

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

   特に大規模な環境では、 `forks`Ansibleが並行して構成するホストの数を増やすために、パラメータを使用してデフォルトのフォーク数（5）を上書きすることを推奨します。（詳細については、を参照して "プレイブックの実行を制御する"ください）。最大値の設定は、Ansibleコントロールノードで使用可能な処理能力によって異なります。上記の例では、CPUを4つ搭載した仮想Ansibleコントロールノード（インテル（R）Xeon（R）Gold 6146 CPU @ 3.20GHz）上で20を実行しています。

導入のサイズと、Ansible制御ノードとBeeGFSファイルおよびブロックノードの間のネットワークパフォ

ーマンスによって、導入時間が異なる場合があります。

### BeeGFSクライアントを設定します

コンピューティングノードやGPUノードなど、BeeGFSファイルシステムにアクセスする必要のあるホストにBeeGFSクライアントをインストールして設定する必要があります。このタスクでは、AnsibleとBeeGFSコレクションを使用できます。

手順

1. 必要に応じて、Ansibleコントロールノードから、BeeGFSクライアントとして設定する各ホストにパスワードなしのSSHを設定します。

   「ssh-copy-id」<user>@<hostname_or_ip>`

2. 「host_vars/`」の下で、「<hostname>.yml」という名前のBeeGFSクライアントごとに、次の内容でファイルを作成し、環境に適した情報をプレースホルダテキストに入力します。

```
# 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>
```

   2つのサブネットアドレス方式を導入している場合は、2つのストレージIPoIBサブネットそれぞれに1つずつ、各クライアントに2つのInfiniBandインターフェイスを設定する必要があります。ここに示す各BeeGFSサービスのサブネットの例と推奨範囲を使用する場合は、クライアントのインターフェイスの1つをの範囲で設定し、もう1つをの範囲で設定する必要があります 100.127.1.0 100.127.99.255 100.128.1.0 100.128.99.255。

3. 新しいファイル'client_inventory.yml'を作成し'上部に次のパラメータを設定します

```
# 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.
```

パスワードをプレーンテキストで保存しないでください。代わりにAnsible Vaultを使用します（のAnsibleのドキュメントを参照してください） "Ansible Vaultを使用したコンテンツの暗号化")または'プレイブックを実行するときに'--Ask -bece-pass`オプションを使用します

4. 「client_inventory.yml」ファイルで、「beegfs_clients」グループの下にBeeGFSクライアントとして設定する必要があるすべてのホストを一覧表示し、BeeGFSクライアントカーネルモジュールの構築に必要な追加の設定を指定します。

```
  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ドライバを使用する場合は、がLinuxのインストールに適した「ヘッダインクルードパス」を指していることを確認して `beegfs_client_ofed_include_path` ください。詳細については、BeeGFSのドキュメントを参照してください "RDMAのサポート"。

5. 'client_inventory.yml'ファイルに'以前に定義したすべての変数の一番下にマウントするBeeGFSファイル・システムを一覧表示します

```
        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_config」は、テストされた設定を表します。すべてのオプションの包括的な概要については'netapp_eseries.beegfs`コレクションのbeegfs_client'ロールに付属のマニュアルを参照してくださいこれには、複数のBeeGFSファイルシステムのマウントまたは同じBeeGFSファイルシステムの複数回のマウントに関する詳細が含まれます。

6. 新しい'client_playbook.yml`ファイルを作成し'次のパラメータを設定します

```
# 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
```

必要なIB/RDMAドライバをインストールし、適切なIPoIBインターフェイスにIPを設定している場合は、「NetApp_eseries.host」コレクションと「IPoIB」ロールのインポートを省略します。

7. クライアントをインストールしてビルドし、BeeGFSをマウントするには、次のコマンドを実行します。

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

8. BeeGFSファイル・システムを本番環境に配置する前に'任意のクライアントにログインし'beegfs -ffsck --checkfs'を実行して'すべてのノードにアクセスできることと'問題が報告されないことを確認することを強くお勧めします

## 5つのビルディングブロックを超えた拡張性

PacemakerとCorosyncを5つ以上のビルディングブロック（10個のファイルノード）に拡張できるように設定できます。ただし、大規模なクラスタには欠点があるため、最終的にはPacemakerとCorosyncでは最大32個のノードが必要になります。

ネットアップでは、最大10個のノードについてBeeGFS HAクラスタのみをテストしています。この制限を超える個々のクラスタの拡張は推奨もサポートもされません。ただし、BeeGFSファイルシステムは依然として10ノードをはるかに超える規模に拡張する必要があり、ネットアップはNetApp解決策 のBeeGFSでこれを計上しています。

各ファイルシステムのビルディングブロックのサブセットを含む複数のHAクラスタを導入することで、基盤となるHAクラスタリングメカニズムの推奨制限やハード制限とは無関係に、BeeGFSファイルシステム全体を拡張できます。このシナリオでは、次の手順を実行します。

- 追加のHAクラスタを表す新しいAnsibleインベントリを作成し、別の管理サービスの設定は省略します。代わりに'ha_cluster.yml'追加する各クラスタのbeegfs_ha_gmtd_floating_ip'変数を最初のBeeGFS管理サービスのIPに指定します

- 同じファイルシステムにHAクラスタを追加する場合は、次の点を確認してください。
  - BeeGFSノードIDは一意です。
  - 「group_vars」の下の各サービスに対応するファイル名は、すべてのクラスタで一意です。
  - BeeGFSクライアントとサーバのIPアドレスは'すべてのクラスタで一意です
  - 追加のクラスタを導入または更新する前に、BeeGFS管理サービスを含む最初のHAクラスタが実行されています。
- 各HAクラスタのインベントリを、それぞれのディレクトリツリーで個別に維持します。

  複数のクラスタのインベントリファイルを1つのディレクトリツリーに混在させようとすると、BeeGFS HAロールで特定のクラスタに適用される構成を集約する方法で原因 の問題が発生することがあります。

新しいビルディングブロックを作成する前に、HAクラスタごとに5つのビルディングブロックを拡張する必要はありません。多くの場合、クラスタあたりの使用するビルディングブロック数が少なくて済むため、管理が容易です。1つは、各ラックのビルディングブロックをHAクラスタとして構成する方法です。

## ストレージプールのオーバープロビジョニングの割合を推奨します

### 第2世代のビルディングブロックでストレージプールあたりの標準の4ボリューム構成に従う場合は、次の表を参照してください。

次の表に、BeeGFSメタデータまたはストレージ・ターゲットごとの'eseries_storage_pool_configuration'でボリューム・サイズとして使用する推奨パーセンテージを示します。

| ドライブサイズ | サイズ |
|---|---|
| 1.92TB | 18 |
| 3.84TB | 21.5 |
| 7.68TB | 22.5インチ |
| 15.3TB | 24 |

上記のガイダンスは、管理サービスが含まれるストレージプールには適用されません。この場合、管理データ用にストレージプールの1%を割り当てるために、25%上のサイズを縮小する必要があります。

これらの値の決定方法については、を参照してください "TR-4800『Appendix A：Understanding SSD持久力とオーバープロビジョニング』"。

## 大容量のビルディングブロック

### 標準のBeeGFS解決策 導入ガイドには、ハイパフォーマンスなワークロードの要件に関する手順と推奨事項が記載されています。大容量の要件を満たすことを検討しているお客様は、ここで紹介する導入方法や推奨事項の違いを確認する必要があります。

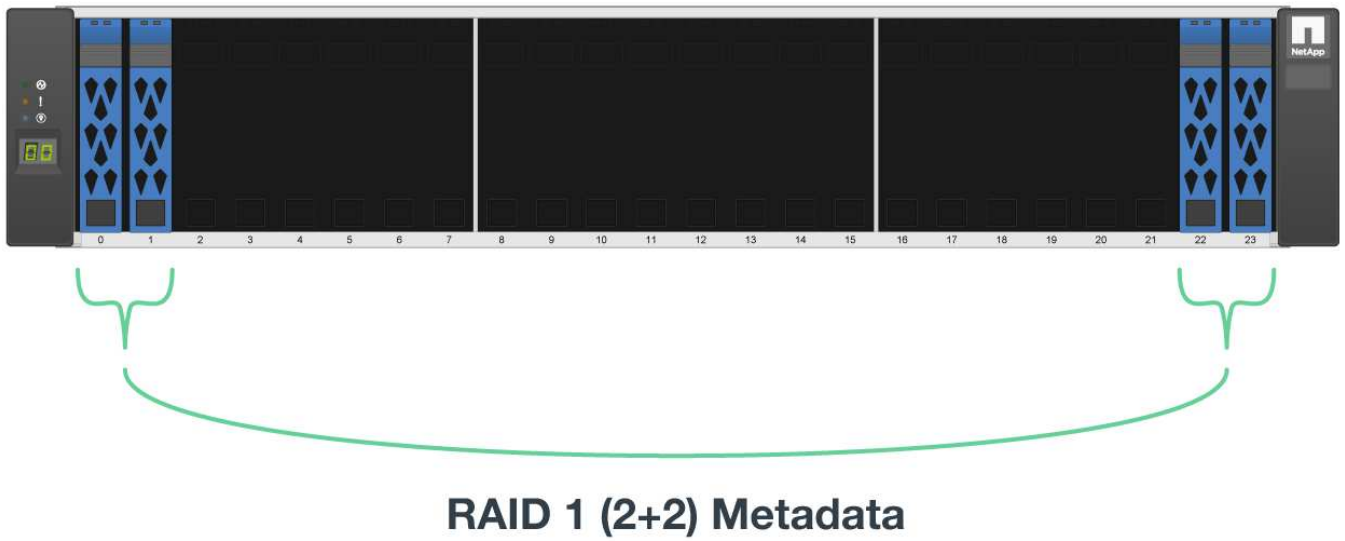

コントローラ

大容量ビルディングブロックの場合は、EF600コントローラをEF300コントローラに交換し、それぞれのコントローラにSAS拡張用のカスケードHICを取り付ける必要があります。各ブロックノードには、BeeGFSメタデータストレージ用の最小限の数のNVMe SSDがアレイエンクロージャに搭載され、BeeGFSストレージボリューム用にNL-SAS HDDが搭載された拡張シェルフに接続されます。

ファイルノードからブロックノードへの構成は変更されません。

ドライブの配置

BeeGFSメタデータストレージの各ブロックノードには、少なくとも4本のNVMe SSDが必要です。これらのドライブは、エンクロージャの一番外側のスロットに取り付ける必要があります。




拡張トレイ

大容量ビルディングブロックのサイズは、ストレージアレイごとに1~7、60本のドライブ拡張トレイを使用して設定できます。

各拡張トレイのケーブル接続手順については、 "ドライブシェルフのEF300ケーブル接続を参照してください"。

# カスタムアーキテクチャを使用

## 概要と要件

Ansibleを使用してBeeGFSハイアベイラビリティクラスタを導入する場合は、任意のNetApp E / EFシリーズストレージシステムをBeeGFSブロックノードとして使用し、x86サーバをBeeGFSファイルノードとして使用します。

このセクションで使用される用語の定義については、"用語と概念"ページを参照してください。

### はじめに

"NetApp Verified Architectureレポート"事前定義されたリファレンス構成とサイジングガイダンスが提供されますが、一部のお客様やパートナー様は、特定の要件やハードウェアの好みにより適したカスタムアーキテクチャを設計したい場合があります。ネットアップでBeeGFSを選択する主なメリットの1つは、Ansibleを使用してBeeGFS共有ディスクHAクラスタを導入できることです。これにより、クラスタ管理が簡易化され、ネットアップがオーサリングするHAコンポーネントによって信頼性が向上します。ネットアップへのカスタムBeeGFSアーキテクチャの導入はAnsibleでも実行されるため、アプライアンスと同様のアプローチでハードウェアを柔軟に選択することができます。

このセクションでは、ネットアップハードウェアにBeeGFSファイルシステムを導入し、Ansibleを使用してBeeGFSファイルシステムを設定するための一般的な手順を説明します。BeeGFSファイルシステムの設計に関するベストプラクティスと最適化された例の詳細については、"NetApp Verified Architectureレポート"セクションを参照してください。

### 導入の概要

通常、BeeGFSファイルシステムを導入するには、次の手順を実行します。

- 初期セットアップ：
  - ハードウェアの設置/ケーブル接続
  - ファイルノードとブロックノードをセットアップ
  - Ansibleコントロールノードをセットアップします。
- BeeGFSファイルシステムをAnsibleインベントリとして定義します。
- ファイルノードとブロックノードに対してAnsibleを実行して、BeeGFSを導入します。
  - 必要に応じて、クライアントをセットアップし、BeeGFSをマウントします。

以降のセクションでは、これらの手順について詳しく説明します。

Ansibleは、ソフトウェアのプロビジョニングタスクと設定タスクをすべて処理します。

- ブロックノードでのボリュームの作成/マッピング
- ファイルノードでのボリュームのフォーマットと調整
- ファイルノードへのソフトウェアのインストール/設定
- HAクラスタを確立し、BeeGFSリソースとファイルシステムサービスを設定します。

## 要件

AnsibleでBeeGFSがサポートされるようになりました "Ansible Galaxy" BeeGFS HAクラスタのエンドツーエンドの導入と管理を自動化するロールとモジュールの集合として。

BeeGFS自体は、<major> .BeeGFSバージョンに準拠してバージョン管理されます。<patch> バージョンのバージョン管理スキームとコレクションでは、サポートされる<major> ごとに役割が維持されます。たとえば、BeeGFS 7.2やBeeGFS 7.3などのBeeGFSバージョンの<minor> <minor> バージョンです。コレクションの更新がリリースされると、各ロールのパッチバージョンが更新され、そのリリースブランチで利用可能な最新のBeeGFSバージョン（例：7.2.8）が示されます。コレクションの各バージョンは、特定の Linux ディストリビューションおよびバージョンでもテストおよびサポートされており、現在、ファイル ノードの場合は Red Hat、クライアントの場合は Red Hat と Ubuntu です。他のディストリビューションを実行することはサポートされていません。また、他のバージョン（特に他のメジャーバージョン）を実行することはお勧めしません。

### **Ansible**コントロールノード

このノードには、BeeGFSの管理に使用するインベントリとプレイブックが含まれます。次のものが必要です。

- Ansible 6.x（Ansibleコア2.13）
- Python 3.6（またはそれ以降）
- python（pip）パッケージ：ipaddrおよびnetaddr

また、制御ノードからすべてのBeeGFSファイルノードとクライアントにパスワードなしのSSHを設定することを推奨します。

### **BeeGFS**ファイルノード

ファイルノードはRed Hat Enterprise Linux (RHEL) 9.4を実行し、必要なパッケージ（pacemaker、corosync、fence-agents-all、resource-agents）を含むHAリポジトリにアクセスできる必要があります。例えば、RHEL 9で適切なリポジトリを有効にするには、以下のコマンドを実行します。

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

### **BeeGFS**クライアントノード

BeeGFSクライアントのAnsibleロールを使用して、BeeGFSクライアントパッケージをインストールし、BeeGFSマウントを管理できます。このロールは、RHEL 9.4 および Ubuntu 22.04 でテストされていま

す。

Ansibleを使用してBeeGFSクライアントをセットアップしない場合は、BeeGFSをマウントします "BeeGFSはLinuxディストリビューションとカーネルをサポートしています" を使用できます。

# 初期セットアップ

## ハードウェアの設置とケーブル接続

### ネットアップでBeeGFSを実行するために使用するハードウェアを設置してケーブルを配線するための手順を説明します。

#### インストールを計画します

各BeeGFSファイルシステムは、いくつかのブロックノードで提供されるバックエンドストレージを使用して、BeeGFSサービスを実行するいくつかのファイルノードで構成されます。BeeGFSサービスにフォールトトレランスを提供するために、ファイルノードは1つ以上のハイアベイラビリティクラスタに構成されます。各ブロックノードはすでにアクティブ/アクティブHAペアです。各HAクラスタでサポートされるファイルノードの最小数は3で、各クラスタでサポートされるファイルノードの最大数は10です。BeeGFSファイルシステムは、単一のファイルシステムネームスペースを提供するために連携する複数の独立したHAクラスタを導入することで、10ノードを超える規模に拡張できます。

一般に、各HAクラスタは一連の「ビルディングブロック」として導入されます。この場合、一部の数のファイルノード（x86サーバ）がいくつかのブロックノード（通常はEシリーズストレージシステム）に直接接続されます。この設定では、非対称クラスタが作成されます。BeeGFSサービスは、BeeGFSターゲットに使用されるバックエンドブロックストレージにアクセスできる特定のファイルノードでのみ実行できます。各ビルディングブロック内のファイルとブロックのノードと、直接接続に使用されるストレージプロトコルのバランスは、特定のインストール要件によって異なります。

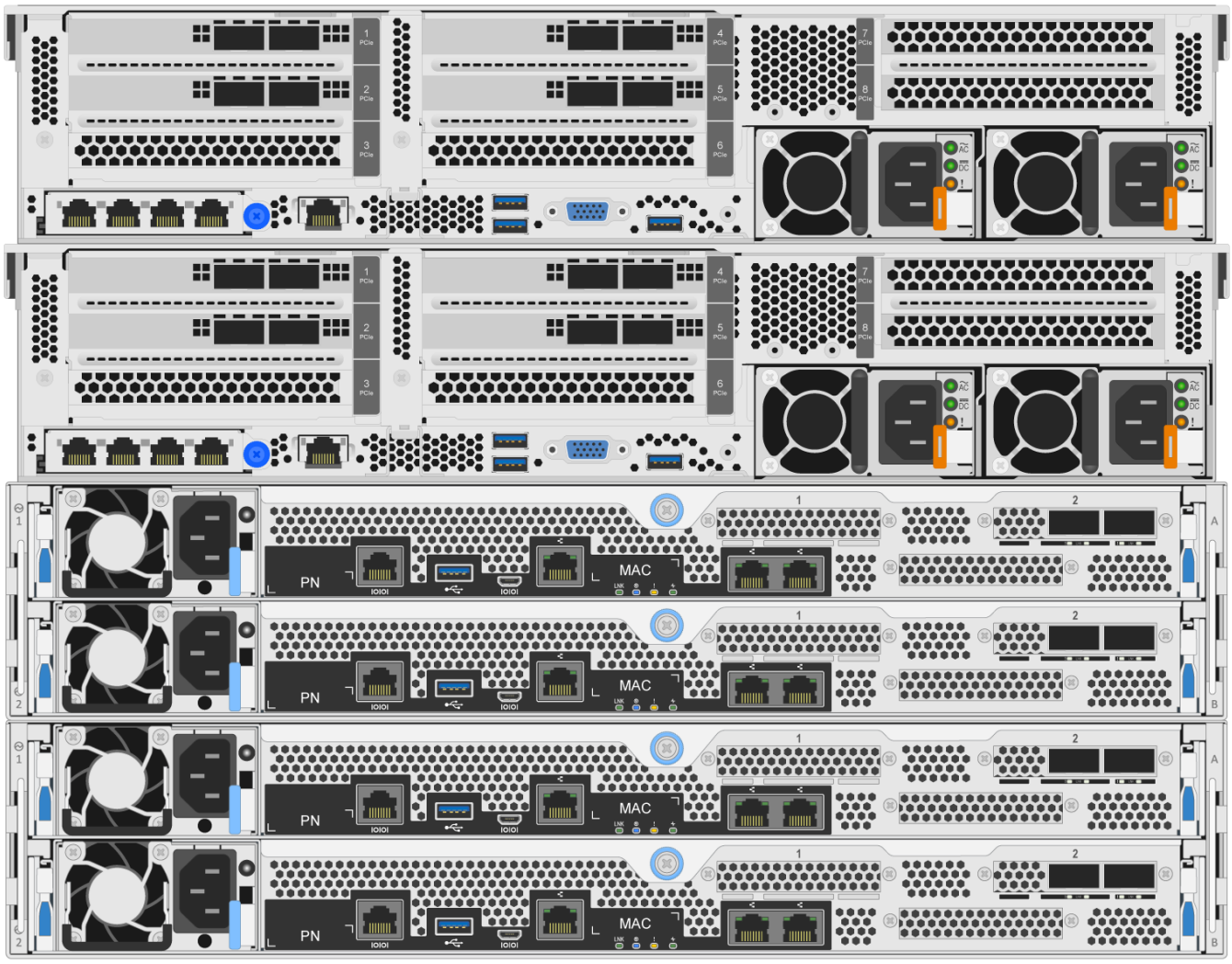
別のHAクラスタアーキテクチャでは、ファイルノードとブロックノードの間にストレージファブリック（ストレージエリアネットワークまたはSANとも呼ばれます）を使用して対称型クラスタを確立します。これにより、BeeGFSサービスを特定のHAクラスタ内の任意のファイルノードで実行できるようになります。一般に対称クラスタは、追加のSANハードウェアによってコスト効率が高くないため、このドキュメントでは、非対称クラスタを1つ以上のビルディングブロックとして配置することを前提としています。

> インストールを開始する前に、特定のBeeGFS導入に必要なファイルシステムアーキテクチャを十分に理解しておく必要があります。

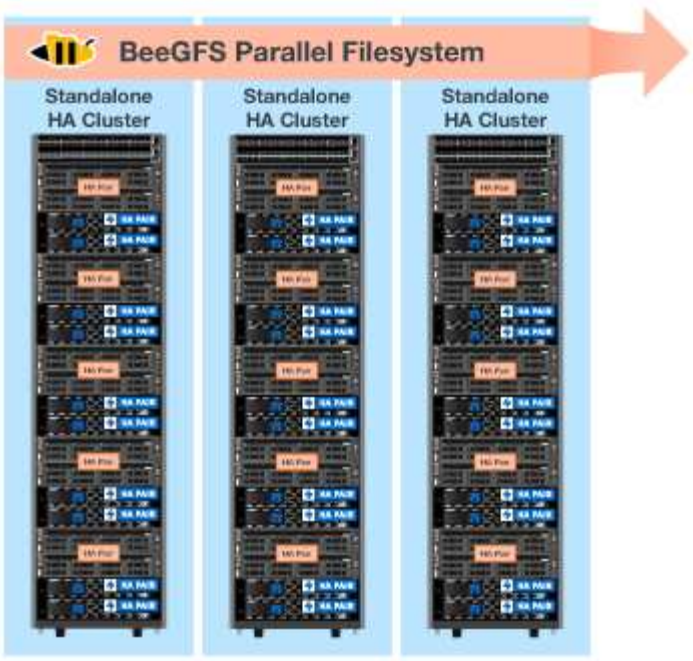
#### ラックハードウェア

設置を計画する場合、各ビルディングブロック内のすべての機器が隣接するラックユニットにラックに設置されることが重要です。ベストプラクティスとして、各ビルディングブロック内のブロックノードのすぐ上にファイルノードがラックに設置されるようにすることを推奨します。ファイルとのモデルについては、のマニュアルを参照してください "ブロック" ラックにレールとハードウェアを設置するときに使用するノード。

単一のビルディングブロックの例：

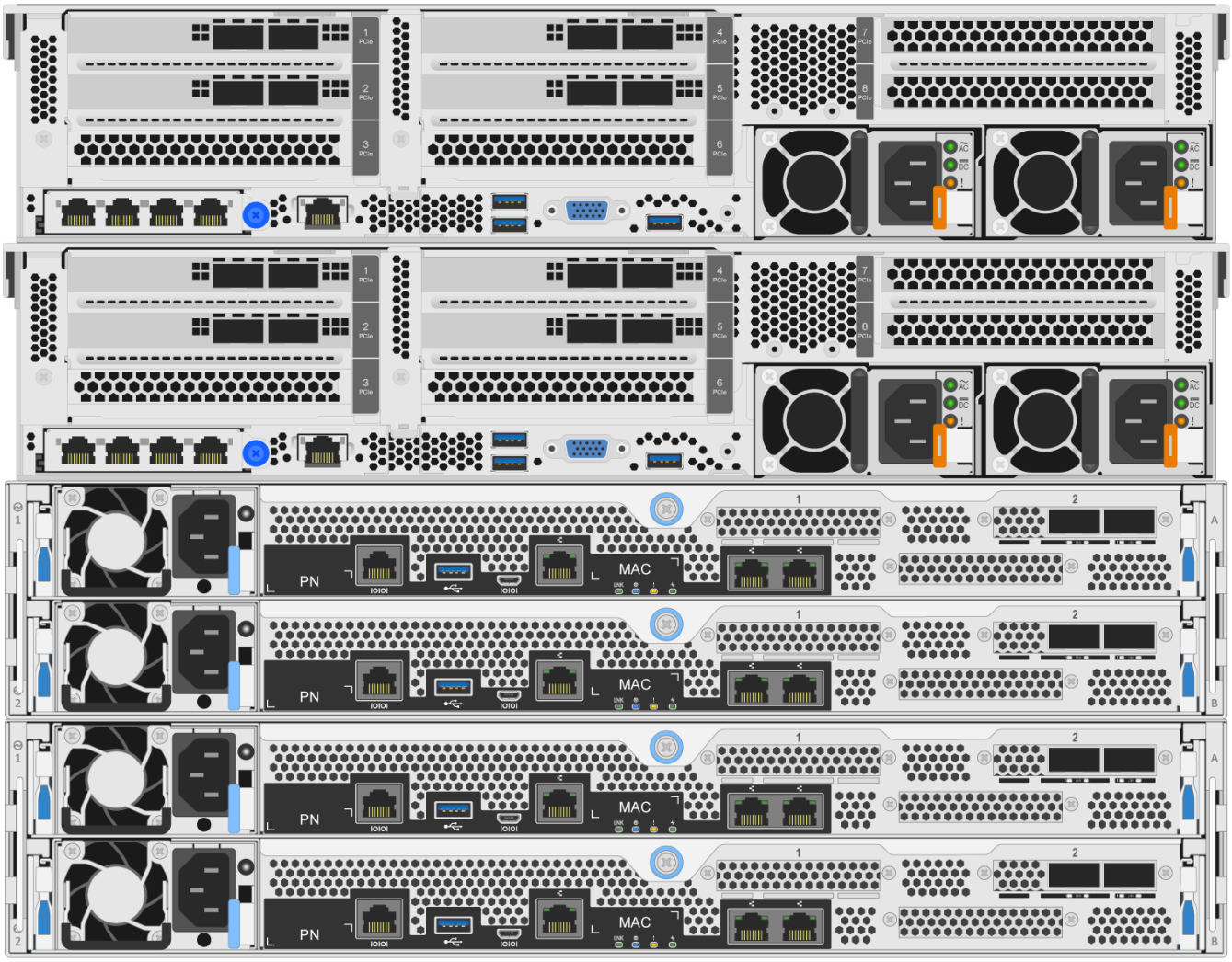



大規模なBeeGFSインストールの例では、各HAクラスタに複数のビルディングブロックがあり、ファイルシステムに複数のHAクラスタがある場合を示します。



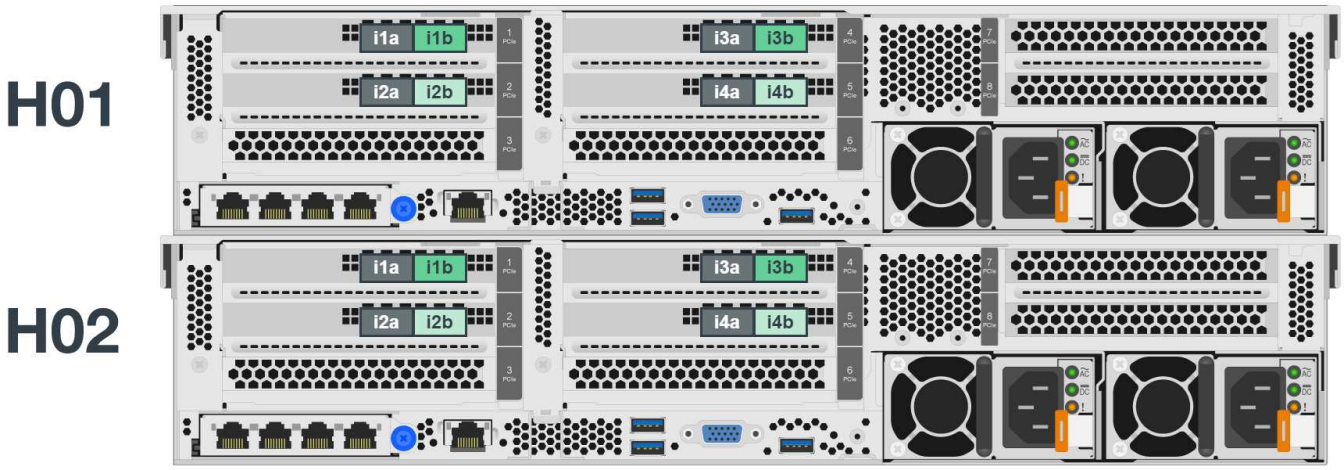

ファイルノードとブロックノードをケーブル接続します

通常、Eシリーズのブロックノードの HIC ポートは、ファイルノードの指定のホストチャネルアダプタ（InfiniBand プロトコルの場合）ポートまたはホストバスアダプタ（ファイバチャネルおよびその他のプロトコルの場合）ポートに直接接続します。これらの接続を確立する正確な方法は、目的のファイルシステムアーキテクチャによって異なります。次に例を示し"NetApp Verified Architecture上の第2世代BeeGFSに基づく"ます。




ファイルノードをクライアントネットワークにケーブル接続します

各ファイルノードには、BeeGFSクライアントトラフィックに指定されたいくつかのInfiniBandポートまたはイーサネットポートがあります。アーキテクチャによっては、各ファイルノードは高パフォーマンスのクライアント/ストレージネットワークに1つ以上の接続を持ち、潜在的に複数のスイッチに接続して冗長性を確保し、帯域幅を増やします。次に、冗長ネットワークスイッチを使用したクライアントケーブル接続の例を示します。濃い緑で強調表示されたポートは別 々 のスイッチに接続されています。




管理ネットワークと電源を接続します

インバンドおよびアウトオブバンドネットワークに必要なネットワーク接続を確立します。

すべての電源装置を接続して、各ファイルノードとブロックノードが複数の配電ユニットに接続して冗長性を確保します（使用可能な場合）。

## ファイルノードとブロックノードをセットアップします

Ansibleを実行する前に、ファイルノードとブロックノードを手動でセットアップする必要があります。

### ファイルノード

#### ベースボード管理コントローラ（**BMC**）の設定

ベースボード管理コントローラ（BMC）は、サービスプロセッサとも呼ばれ、オペレーティングシステムがインストールされていない場合やアクセスできない場合でもリモートアクセスを提供できるさまざまなサーバプラットフォームに組み込まれているアウトオブバンド管理機能の一般的な名前です。ベンダーは通常、この機能を独自のブランドで販売しています。たとえば、Lenovo SR665では、BMCはLenovo XClarity Controller（XCC）と呼ばれています。

サーバベンダーのマニュアルに従って、この機能へのアクセスに必要なライセンスを有効にし、BMCがネットワークに接続され、リモートアクセス用に適切に設定されていることを確認してください。

Redfishを使用したBMCベースのフェンシングが必要な場合は、Redfishが有効になっており、BMCインターフェイスにファイルノードにインストールされているOSからアクセスできることを確認します。BMCと動作環境が同じ物理ネットワークインターフェイスを共有している場合は、ネットワークスイッチで特別な設定が必要になることがあります。

#### システム設定を調整します

セットアップユーティリティ（BIOS/UEFI）インターフェイスを使用して、パフォーマンスを最大化するように設定されていることを確認します。正確な設定と最適な値は、使用しているサーバーモデルによって異なります。ガイダンスはのために提供されて"ファイルノードモデルを確認しました"います。それ以外の場合は、サーバベンダーのドキュメントと、お使いのモデルに基づいたベストプラクティスを参照してください。

オペレーティングシステムをインストールします

リストされているファイルノードの要件に基づいて、サポートされているオペレーティングシステムをインストールし"こちらをご覧ください"ます。Linuxディストリビューションに基づいて、以下の追加手順を参照してください。

**Red Hat**

、 "RHELシステムを登録および登録する方法"そして "更新を制限する方法" 。

ハイアベイラビリティに必要なパッケージを含むRed Hatリポジトリを有効にします。

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

管理ネットワークを設定します

オペレーティングシステムのインバンド管理に必要なネットワークインターフェイスを設定します。具体的な手順は、使用しているLinuxのディストリビューションとバージョンによって異なります。

SSHが有効になっていて、Ansibleコントロールノードからすべての管理インターフェイスにアクセスできることを確認します。

**HCAとHBA**ファームウェアを更新します

すべてのHBAおよびHCAでに記載されているサポート対象のファームウェアバージョンが実行されていることを確認し"NetApp Interoperability Matrix を参照してください"、必要に応じてアップグレードします。NVIDIA ConnectXアダプタに関するその他の推奨事項については"こちらをご覧ください"、こちらを参照してください。

ブロックノード

手順~を実行します "Eシリーズの運用を開始" 各ブロックノードコントローラに管理ポートを設定し、必要に応じて各システムのストレージアレイ名を設定します。

Ansible制御ノードからすべてのブロックノードにアクセスできるようにする以外に、追加の設定は必要ありません。残りのシステム構成はAnsibleで適用/管理されます。

## Ansible Control Nodeをセットアップします

ファイルシステムを導入および管理するためのAnsibleコントロールノードをセットアップします。

概要

Ansible制御ノードは、クラスタの管理に使用される物理または仮想Linuxマシンです。次の要件を満たしている必要があります。

- "要件"BeeGFS HAロール（インストールされているAnsible、Python、その他のPythonパッケージなど）

のを確認します。

- 公式情報をご確認ください "Ansibleの制御ノード要件" オペレーティングシステムのバージョンも含まれます。
- すべてのファイルノードとブロックノードに、SSHとHTTPSでアクセスできます。

詳細なインストール手順が"こちらをご覧ください"記載されています。

# BeeGFSファイルシステムを定義します

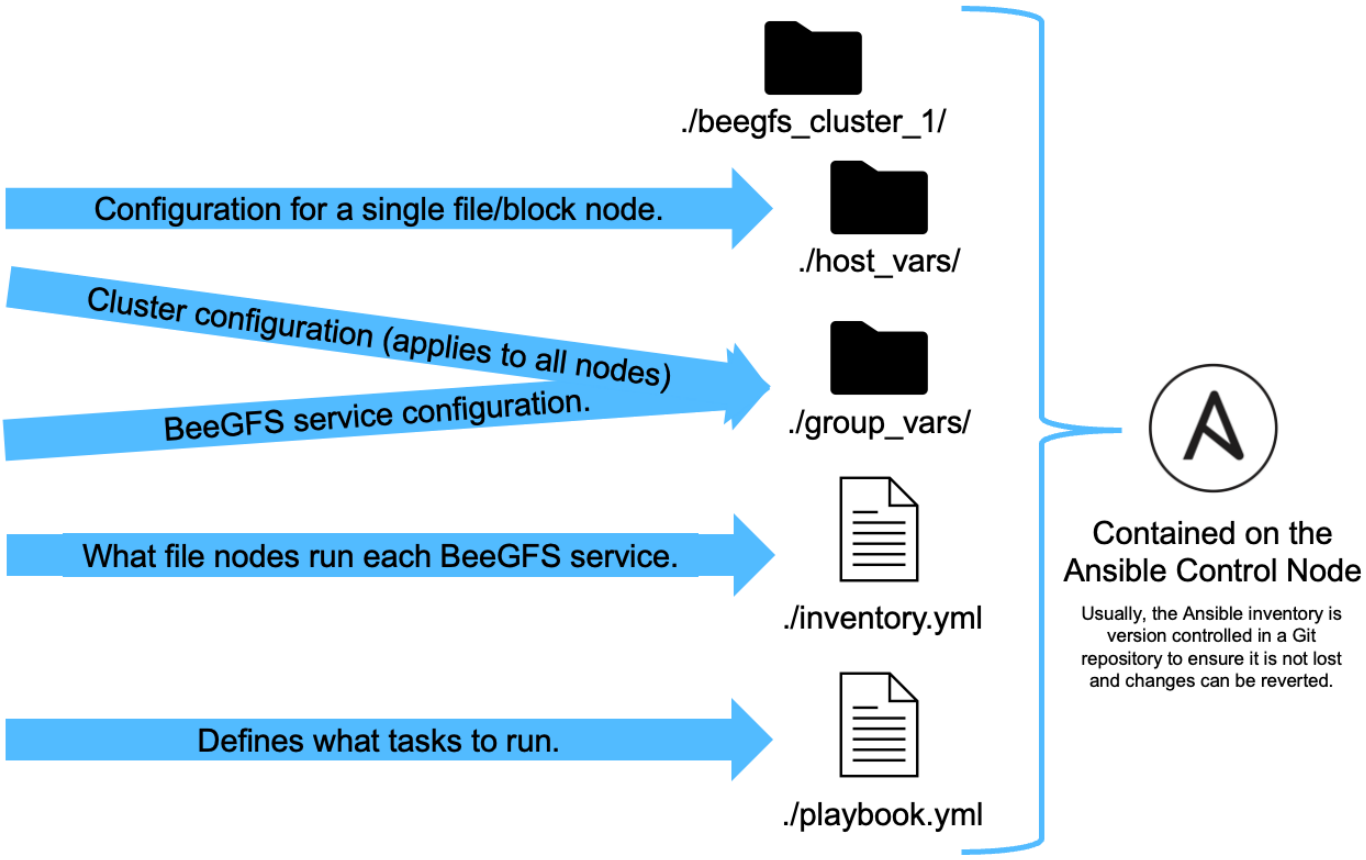
## Ansibleのインベントリの概要

Ansibleインベントリは、必要なBeeGFS HAクラスタを定義する一連の構成ファイルです。

概要

の編成については、Ansibleの標準的な手法に従うことを推奨します "在庫"の使用を含む "サブディレクトリ/ファイル" インベントリ全体を1つのファイルに格納する必要はありません。

単一のBeeGFS HAクラスタのAnsibleインベントリは、次のように構成されます。



1つのBeeGFSファイルシステムは複数のHAクラスタにまたがることができるため、大規模なインストールで複数のAnsibleインベントリを使用することが可能です。一般に、問題を回避するために複数のHAクラスタを単一のAnsibleインベントリとして定義することは推奨されません。

手順

1. Ansibleコントロールノードで、導入するBeeGFSクラスタのAnsibleインベントリを含む空のディレクトリを作成します。
   a. ファイルシステムに最終的に複数のHAクラスタが含まれる場合は、まずファイルシステムのディレクトリを作成し、そのあとに各HAクラスタを表すインベントリのサブディレクトリを作成することを推奨します。例：

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

2. 導入するHAクラスタのインベントリが格納されているディレクトリに、2つのディレクトリを作成します `group_vars` および `host_vars` 2つのファイルがあります `inventory.yml` および `playbook.yml`。

以降のセクションでは、これらの各ファイルの内容を定義する手順を説明します。

## ファイルシステムを計画

## Ansibleインベントリを構築する前に、ファイルシステムの導入を計画します。

概要

ファイルシステムを導入する前に、クラスタ内で実行されているすべてのファイルノード、ブロックノード、およびBeeGFSサービスで必要となるIPアドレス、ポート、およびその他の設定を定義する必要があります。具体的な構成はクラスタのアーキテクチャによって異なりますが、ここでは、一般に適用されるベストプラクティスと手順について説明します。

手順

1. IPベースのストレージプロトコル（iSER、iSCSI、NVMe/IB、NVMe/RoCEなど）を使用してファイルノードをブロックノードに接続する場合は、ビルディングブロックごとに次のワークシートに記入します。1つのビルディングブロック内の各直接接続には、一意のサブネットが必要であり、クライアント/サーバ接続に使用されるサブネットと重複しないようにする必要があります。

| ファイルノード | IBポート | IP アドレス | ブロックノード | IBポート | 物理IP | 仮想IP（HDR IBを使用するEF600のみ） |
|---|---|---|---|---|---|---|
| <HOSTNAME> | <PORT> | <IP/SUBNET> | <HOSTNAME> | <PORT> | <IP/SUBNET> | <IP/SUBNET> |

各ビルディングブロックのファイルノードとブロックノードが直接接続されている場合、複数のビルディングブロックで同じIP/スキームを再利用することがよくあります。

2. ストレージネットワークにInfiniBandまたはRDMA over Converged Ethernet（RoCE）を使用しているかどうかに関係なく、次のワークシートに記入して、HAクラスタサービス、BeeGFSファイルサービス、およびクライアントが通信するために使用するIP範囲を決定します。

| 目的 | **InfiniBand**ポート | **IP**アドレスまたは**IP**範囲 |
|---|---|---|
| BeeGFSクラスタIP | <INTERFACE(s)> | <RANGE> |
| BeeGFS Managementの略 | <INTERFACE(s)> | <IP(s)> |
| BeeGFSメタデータ | <INTERFACE(s)> | <RANGE> |
| BeeGFS Storage（BeeGFSストレージ） | <INTERFACE(s)> | <RANGE> |
| BeeGFSクライアント | <INTERFACE(s)> | <RANGE> |

   a. 単一のIPサブネットを使用する場合は、ワークシートが1つだけ必要です。それ以外の場合は、2つ目のサブネットのワークシートにも記入してください。

3. 上記に基づいて、クラスタ内の各ビルディングブロックに対して、実行するBeeGFSサービスを定義する次のワークシートに記入します。各サービスについて、優先/セカンダリファイルノード、ネットワークポート、フローティングIP、NUMAゾーン割り当て（必要な場合）、およびターゲットに使用するブロックノードを指定します。ワークシートに記入する際は、次のガイドラインを参照してください。

   a. BeeGFSサービスをいずれかとして指定します `mgmt.yml`、 `meta_<ID>.yml`または `storage_<ID>.yml` ここで'IDは'このファイルシステム内のすべてのBeeGFSサービスの一意の番号を表しますこの規則により、以降のセクションでは、各サービスを設定するためのファイルを作成する際に、このワークシートを簡単に参照できます。

   b. BeeGFSサービスのポートは、特定のビルディングブロック全体で一意である必要があります。ポートの競合を回避するために、同じポート番号のサービスを同じファイルノード上で実行することはできません。

   c. 必要なサービスが複数のブロックノード/ストレージプール（すべてのボリュームを同じコントローラに所有する必要はない）のボリュームを使用できる場合。複数のサービスで同じブロックノードやストレージプール構成を共有することもできます（個 々 のボリュームはこのあとのセクションで定義します）。

| **BeeGFS**サービス（ファイル名） | ファイルノード | ポート | フローティング**IP** | **NUMA**ゾーン | ブロックノード | ストレージプール | 所有コントローラ |
|---|---|---|---|---|---|---|---|
| <SERVICE TYPE>_<ID> .yml | <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> |

標準的な規則、ベストプラクティス、記入例のワークシートの詳細については"ベストプラクティス""BeeGFS

ビルディングブロックを定義します"、『BeeGFS on NetApp Verified Architecture』のおよびのセクションを参照してください。

## ファイルノードとブロックノードを定義します

### 個 々 のファイルノードを設定します

## ホスト変数（host_vars）を使用して、個 々 のファイルノードの設定を指定します。

#### 概要

このセクションでは、の入力手順について説明します `host_vars/<FILE_NODE_HOSTNAME>.yml` クラスタ内の各ファイルノードのファイル。これらのファイルには、特定のファイルノードに固有の設定のみを含める必要があります。これには、次のような一般

- AnsibleでIPまたはホスト名を定義して、ノードへの接続に使用する必要があります。
- HAクラスタサービス（PacemakerとCorosync）で他のファイルノードとの通信に使用するインターフェイスおよびクラスタIPを追加で設定しています。デフォルトでは、これらのサービスは管理インターフェイスと同じネットワークを使用しますが、冗長性を確保するために追加のインターフェイスを使用できる必要があります。一般的には、ストレージネットワークに追加のIPを定義して、クラスタまたは管理ネットワークを追加する必要を回避します。
  - クラスタ通信に使用されるネットワークのパフォーマンスは、ファイルシステムのパフォーマンスにとっては重要ではありません。デフォルトのクラスタ構成では、通常、少なくとも1Gb/秒ネットワークを使用すると、ノード状態の同期やクラスタリソース状態の変更の調整など、クラスタ処理に十分なパフォーマンスが提供されます。低速/ビジーなネットワークでは、原因 リソースの状態が通常よりも長くなる可能性があります。また、ノードが妥当な時間内にハートビートを送信できない場合、ノードがクラスタから削除されることがあります。
- 目的のプロトコルを介したブロックノードへの接続に使用するインターフェイスの設定（iSCSI / iSER、NVMe/IB、NVMe/RoCE、FCPなど）

#### 手順

"ファイルシステムを計画"セクションで定義したIPアドレス指定方式を参照して、クラスタ内のファイルノードごとにファイルを作成し `host_vars/<FILE_NODE_HOSTNAME>/yml`、次のように設定します。

1. 上部に、ノードへのSSHとノードの管理にAnsibleで使用するIPまたはホスト名を指定します。

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

2. クラスタトラフィックに使用できる追加のIPを設定します。
   a. ネットワークタイプがの場合 "InfiniBand（IPoIBを使用）"：

```
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. ネットワークタイプがの場合 "RDMA over Converged Ethernet（RoCE）"：

```
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. ネットワークタイプがの場合 "イーサネット（TCPのみ、RDMAなし）"：

```
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. 優先IPが高い順に、クラスタトラフィックに使用するIPを指定します。

```
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.
```

ステップ2で設定したIPSは、に含まれていないかぎり、クラスタIPとして使用されません `beegfs_ha_cluster_node_ips` リストこれにより、必要に応じて他の目的にも使用できるAnsibleを使用して追加のIP /インターフェイスを設定できます。

4. IPベースのプロトコルを使用してノードをブロックするためにファイルノードが通信する必要がある場合は、IPを適切なインターフェイスに設定し、そのプロトコルのインストールまたは設定に必要なパッケージをすべて設定する必要があります。

a. を使用する場合 "iSCSI"：

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

b. を使用する場合 "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. を使用する場合 "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. を使用する場合 "NVMe/RoCE"：

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

e. その他のプロトコル：

i. を使用する場合 "NVMe/FC"個 々 のインターフェイスを設定する必要はありません。BeeGFSクラスタの導入により、プロトコルが自動的に検出され、必要に応じて要件がインストール/設定されます。ファブリックを使用してファイルノードとブロックノードを接続する場合は、ネットアップとスイッチベンダーのベストプラクティスに従ってスイッチを適切にゾーニングしてください。

ii. FCPまたはSASを使用する場合、追加のソフトウェアをインストールまたは設定する必要はありません。FCPを使用する場合は、次に示す手順でスイッチが適切にゾーニングされていることを確認 "ネットアップ" スイッチベンダーのベストプラクティスを確認してください。

iii. 現時点では、IB SRPの使用は推奨されていません。Eシリーズのブロックノードでサポートされているものに応じて、NVMe/IBまたはiSERを使用します。

をクリックします "こちらをご覧ください" たとえば、単一のファイルノードを表す完全なインベントリファイルなどです。

**Advanced**：イーサネットと**InfiniBand**モードの間で**NVIDIA ConnectX VPI**アダプタを切り替えます

NVIDIA ConnectX-Virtual Protocol Interconnect ®（VPI）アダプタは、InfiniBandとイーサネットの両方をトランスポートレイヤとしてサポートします。モード間の切り替えは自動的にはネゴシエートされないため、に含まれているオープンソースパッケージを使用して設定する必要があります <code>mstconfig</code> <code>mstflint</code> <a href="https://docs.nvidia.com/networking/display/mftv4270/mft+supported+configurations+and+parameters" target="_blank">"NVIDIAファームウェアツール(MFT)"</a>。アダプタのモードを変更する必要があるのは一度だけです。これは手動で行うことも、インベントリのセクションを使用して設定されたインターフェイスの一部としてAnsibleインベントリに含めることもでき <code>eseries-[ib|ib_iser|ipoib|nvme_ib|nvme_roce|roce]_interfaces:</code> 、自動的にチェック/適用されます。

たとえば、InfiniBandモードのインターフェイスをイーサネットに変更して、RoCEに使用できるようにするには、次のコマンドを実行します。

1. 設定する各インターフェイスについて、を指定します `mstconfig` を指定するマッピング（またはディクショナリ）として指定します `LINK_TYPE_P<N>` ここで、 `<N>` は、インターフェイスのHCAのポート番号で決まります。。 `<N>` の値はを実行して確認できます `grep PCI_SLOT_NAME /sys/class/net/<INTERFACE_NAME>/device/uevent` PCIスロット名の最後の数字に1を追加し、10進数に変換します。

   a. たとえば、を指定します `PCI_SLOT_NAME=0000:2f:00.2` （2+1 → HCAポート3）→ `LINK_TYPE_P3: eth`：

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

詳細については、を参照してください "NetApp Eシリーズホストコレクションのドキュメント" をクリックします。

個 々 のブロックノードを設定します

## ホスト変数（host_vars）を使用して個 々 のブロックノードの設定を指定します。

概要

このセクションでは、の入力手順について説明します `host_vars/<BLOCK_NODE_HOSTNAME>.yml` クラスタ内のブロックノードごとにファイルを作成します。これらのファイルに含まれるのは、特定のブロックノードに固有の設定のみである必要があります。これには、次のような一般

- システム名（System Managerに表示）。
- いずれかのコントローラのHTTPS URL（REST APIを使用したシステムの管理に使用）。
- このブロックノードへの接続に使用するストレージプロトコルファイルノード。
- IPアドレスなどのホストインターフェイスカード（HIC）ポートを設定する（必要な場合）。

手順

"ファイルシステムを計画"セクションで定義したIPアドレス指定方式を参照して、クラスタ内のブロックノードごとにファイルを作成し `host_vars/<BLOCK_NODE_HOSTNAME>/yml`、次のように設定します。

1. 上部で、いずれかのコントローラのシステム名とHTTPS URLを指定します。

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

2. を選択します "プロトコル" ファイルノードはこのブロックノードへの接続に使用します。
   a. サポートされるプロトコル： `auto、 iscsi、 fc、 sas、 ib_srp、 ib_iser、 nvme_ib、 nvme_fc、 nvme_roce。`

```
eseries_initiator_protocol: <PROTOCOL>
```

3. 使用するプロトコルによっては、HICポートの設定を追加する必要があります。HICポートの設定は、必要に応じて各コントローラの設定の一番上のエントリが各コントローラの物理的な左端のポートに対応し、一番下のポートが最も右のポートになるように定義する必要があります。現在使用していないポートでも、すべてのポートで有効な設定が必要です。

> EF600ブロックノードでHDR（200GB）InfiniBandまたは200GBのRoCEを使用している場合は、次のセクションも参照してください。

   a. iSCSIの場合：

```
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. iSERの場合：

```
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/IB：

```
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/RoCEの場合：

```
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プロトコルとSASプロトコルについては、追加の設定は必要ありません。SRPの使用は推奨されません。

iSCSI CHAPの設定など、HICポートとホストプロトコルを設定するその他のオプションについては、を参照してください "ドキュメント" SANtricity コレクションに含まれています。注：BeeGFSを導入する場合は'ストレージ・プール'ボリューム構成'その他のプロビジョニング・ストレージの設定は他の場所で行いますこのファイルでは定義しないでください

をクリックします "こちらをご覧ください" たとえば、1つのブロックノードを表す完全なインベントリファイルなどです。

**NetApp EF600**ブロックノードで**HDR（200GB）InfiniBand**または**200GB RoCE**を使用：

EF600でHDR（200GB）InfiniBandを使用するには、物理ポートごとに2つ目の「仮想」IPを設定する必要があります。以下は、デュアルポートInfiniBand HDR HICを搭載したEF600の正しい設定方法の例です。

```
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)
```

**Common File Node Configuration**を指定します

## グループ変数（group_vars）を使用して共通ファイルノード設定を指定します。

概要

Appleがすべてのファイルノードに適用される構成は、で定義されます `group_vars/ha_cluster.yml`。一般的には次のものが含ま

- 各ファイルノードに接続してログインする方法の詳細。
- 一般的なネットワーク構成。
- 自動リブートが許可されるかどうか。
- ファイアウォールとSELinuxの状態を設定する方法。
- アラートやフェンシングなどのクラスタ構成。
- パフォーマンスの調整。
- Common BeeGFSサービスの設定

このファイルで設定したオプションは、たとえば、異なるハードウェアモデルを混在させる場合や、ノードごとに異なるパスワードを設定する場合など、個 々 のファイルノードに定義することもできます。個 々 のファイルノードの設定は、このファイルの設定よりも優先されます。

手順

ファイルを作成します `group_vars/ha_cluster.yml` 次のように入力します。

1. リモートホストでAnsible Controlノードがどのように認証されるかを指定します。

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

特に本番環境では、パスワードをプレーンテキストで保存しないでください。代わりにAnsible Vaultを使用します（を参照） "Ansible Vaultを使用したコンテンツの暗号化"）またはをクリックします `--ask-become-pass` プレイブックを実行する際のオプション。状況に応じて `ansible_ssh_user` はすでにrootであるため、必要に応じてを省略できます `ansible_become_password`。

2. イーサネットインターフェイスまたはInfiniBandインターフェイス（クラスタIPなど）に静的IPを設定していて、複数のインターフェイスが同じIPサブネットにある場合（たとえば、ib0が192.168.1.10/24を使用し、ib1が192.168.1.11/24を使用している場合）、マルチホームサポートが正常に機能するためには、追加のIPルーティングテーブルとルールを設定する必要があります。提供されているネットワークインターフェイス設定フックを次のように有効にします。

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

3. クラスタを導入する際は、ストレージプロトコルによっては、リモートブロックデバイスを検出しやすくするためにノードをリブートしたり（Eシリーズボリューム）、構成の他の要素を適用したりする必要があります。デフォルトでは、ノードはリブート前にプロンプトを表示しますが、次の項目を指定することでノードの自動再起動を許可できます。

```
eseries_common_allow_host_reboot: true
```

   a. リブート後のデフォルトでは、ブロックデバイスやその他のサービスの準備ができていることを確認するために、Ansibleはsystemdまで待機します `default.target` は、導入を続行する前に到達しています。NVMe/IBを使用する場合は、リモートデバイスの初期化、検出、および接続に時間がかかることがあります。その結果、導入の途中で自動化が失敗し続ける可能性があります。NVMe/IBを使用する場合にこの問題を回避するには、以下の条件も定義します。

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

4. BeeGFSサービスとHAクラスタサービスが通信するためには、多数のファイアウォールポートが必要です。firwallを手動で設定する場合を除き（非推奨）、必要なファイアウォールゾーンを作成し、ポートを自動的に開くように次のように指定します。

```
beegfs_ha_firewall_configure: True
```

5. SELinuxは現時点でサポートされていないため、競合を回避するために（特にRDMAを使用している場合）状態をdisabledに設定することを推奨します。SELinuxが無効になっていることを確認するには、次のように設定

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

6. ファイルノードが通信できるように認証を設定し、組織のポリシーに基づいてデフォルト設定を必要に応じて調整します。

```
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. "ファイルシステムを計画"セクションに基づいて、このファイルシステムのBeeGFS管理IPを指定します。

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

> 一見冗長に見えても'beegfs_ha_gmtd_floating_ip'は'1つのHAクラスタを超えてBeeGFSファイルシステムを拡張する場合に重要です以降のHAクラスタは、BeeGFS管理サービスを追加せずに導入され、最初のクラスタが提供する管理サービスをポイントします。

8. 必要に応じてEメールアラートを有効にします。

```
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. フェンシングを有効にすることを強く推奨します。そうしないと、プライマリノードで障害が発生したときに、セカンダリノードでサービスが開始されないようにブロックされます。
   a. 次の項目を指定して、フェンシングをグローバルに有効にします

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

      i. メモ必要に応じて、サポートされているものを "クラスタ・プロパティ" ここで指定することもできます。BeeGFS HAロールには十分にテストされた機能が多数付属しているため、これらの調整は通常は必要ありません "デフォルト値です"。

   b. 次に、フェンシングエージェントを選択して構成します。
      i. オプション1：APC Power Distribution Unit（PDU;配電ユニット）を使用してフェンシングをイネーブルにするには、次の手順

```
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. オプション2：Lenovo XCC（および他のBMC）が提供するRedfish APIを使用してフェンシングを有効にするには、次の手順を実行します。

```
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. 他のフェンシングエージェントの設定の詳細については、を参照してください "Red Hat ドキュメント"。

10. BeeGFS HAロールでは、パフォーマンスをさらに最適化するために、さまざまなチューニングパラメータを適用できます。これには、カーネルメモリ使用率の最適化や、ブロックデバイスI/Oなどのパラメータが含まれます。このロールには、NetApp E-Seriesブロックノードを使用したテストに基づく合理的なセットが付属してい "デフォルト値です" ますが、デフォルトでは次を指定しない限り、これらは適用されません。

```
beegfs_ha_enable_performance_tuning: True
```

a. 必要に応じて、ここでデフォルトのパフォーマンス調整に変更を加えます。詳細については、完全なドキュメントを参照して "パフォーマンス調整パラメータ" ください。

11. BeeGFSサービスに使用されるフローティングIPアドレス（論理インターフェイスとも呼ばれます）がファイルノード間でフェイルオーバーできるようにするには、すべてのネットワークインターフェイスに一貫した名前を付ける必要があります。デフォルトでは、ネットワークインターフェイス名はカーネルによって生成されます。これは、同じPCIeスロットにネットワークアダプタが搭載された同一のサーバモデルであっても、一貫した名前が生成される保証はありません。これは、装置が展開され、生成されたインターフェイス名が認識される前にインベントリを作成する場合にも役立ちます。サーバまたはのブロック図に基づいて、一貫したデバイス名を使用できるようにします `lshw -class network -businfo` 出力で、目的のPCIeアドレスと論理インターフェイスのマッピングを次のように指定します。

a. InfiniBand（IPoIB）ネットワークインターフェイスに対応しています。

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

b. イーサネットネットワークインターフェイスの場合：

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

インターフェイスの名前を変更したときの競合を回避するには（名前を変更できないようにするため）、eth0、ens9f0、ib0、ibs4f0などの潜在的なデフォルト名は使用しないでください。一般的な命名規則としては、イーサネットまたはInfiniBandには「e」または「i」を使用し、続いてPCIeスロット番号とポートを示す文字を使用します。たとえば、スロット3にInfiniBandアダプタの2番目のポートはi3bとなります。

検証済みファイルノードモデルを使用している場合は、をクリックします "こちらをご覧ください" PCIeアドレスと論理ポートのマッピングの例

12. 必要に応じて、クラスタ内のすべてのBeeGFSサービスに適用する設定を指定します。デフォルトの設定値が表示され "こちらをご覧ください"、サービス単位の設定は他の場所で指定されます。

    a. BeeGFS管理サービス：

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

    b. BeeGFSメタデータサービス：

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

    c. BeeGFSストレージサービス：

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

13. BeeGFS 7.2.7および7.3.1以降 "接続認証" 設定または明示的に無効にする必要があります。Ansibleベースの導入を使用してこれを設定するには、いくつかの方法があります。

    a. デフォルトでは、展開によって自動的に接続認証が設定され、が生成されます `connauthfile` これはすべてのファイルノードに配布され、BeeGFSサービスとともに使用されます。このファイルは、Ansibleの制御ノードにも配置/管理されます `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile` このファイルシステムにアクセスする必要のあるクライアントで再利用できるように、安全に保管する必要があります。

        i. 新しいキーを生成するには、をクリックします `-e "beegfs_ha_conn_auth_force_new=True` Ansibleプレイブックを実行している場合。注：これは、がの場合は無視されます `beegfs_ha_conn_auth_secret` が定義されている。

        ii. 詳細オプションについては、に付属のデフォルトの一覧を参照して "BeeGFS HAロール"ください。

    b. カスタムシークレットを使用するには、で以下を定義します `ha_cluster.yml`：

```
beegfs_ha_conn_auth_secret: <SECRET>
```

c. 接続認証は完全に無効にできます（非推奨）。

```
beegfs_ha_conn_auth_enabled: false
```

をクリックします "こちらをご覧ください" 一般的なファイルノード設定を表す完全なインベントリファイルの例を次に示します。

**NetApp EF600**ブロックノードで**HDR（200GB）InfiniBand**を使用：

EF600でHDR（200GB）InfiniBandを使用するには、サブネットマネージャが仮想化をサポートしている必要があります。ファイルノードとブロックノードがスイッチを使用して接続されている場合は、ファブリック全体に対してサブネットマネージャで有効にする必要があります。

ブロックノードとファイルノードがInfiniBandを使用して直接接続されている場合は `opensm`、ブロックノードに直接接続されているインターフェイスごとに、各ファイルノードでのインスタンスを設定する必要があります。そのためには、 `configure: true`whenを指定し"ファイルノードストレージインターフェイスを設定しています"ます。

現在、サポートされているLinuxディストリビューションに同梱されているの受信トレイバージョンで `opensm` は、仮想化はサポートされていません。代わりに、NVIDIA OpenFabrics Enterprise Distribution（OFED）からのバージョンをインストールして設定する必要があります `opensm` 。Ansibleによる導入もサポートされていますが、いくつかの追加手順が必要です。

1. curlまたは任意のツールを使用して、セクションに記載されているOpenSMのバージョンのパッケージをNVIDIAのWebサイトからディレクトリにダウンロードし "テクノロジ要件" `<INVENTORY>/packages/` ます。例：

```
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. の下 `group_vars/ha_cluster.yml` 次の設定を定義します。

```
### 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"
```

**Common Block Node Configuration**を指定します

## グループ変数（group_vars）を使用して共通ブロックノード設定を指定します。

概要

すべてのブロックノードに対してAppleが実施する必要がある設定は、で定義します `group_vars/eseries_storage_systems.yml`。一般的には次のものが含ま

- Ansible制御ノードが、ブロックノードとして使用されるEシリーズストレージシステムに接続する方法の詳細。
- ノードで実行するファームウェア、NVSRAM、およびドライブファームウェアのバージョン。

- キャッシュ設定、ホスト構成、ボリュームのプロビジョニング方法に関する設定を含むグローバル構成。

> このファイルで設定したオプションは、個 々 のブロックノードに定義することもできます。たとえば、異なるハードウェアモデルを混在させる場合や、ノードごとに異なるパスワードを設定する場合などです。個 々 のブロックノードの設定は、このファイルの設定よりも優先されます。

手順

ファイルを作成します `group_vars/eseries_storage_systems.yml` 次のように入力します。

1. Ansibleは、SSHを使用してブロックノードに接続するのではなく、REST APIを使用します。これを実現するには、以下を設定する必要があります

```
ansible_connection: local
```

2. 各ノードを管理するためのユーザ名とパスワードを指定してください。ユーザ名はオプションで省略できます（デフォルトはadmin）。それ以外の場合はadmin権限を持つ任意のアカウントを指定できます。また、SSL証明書を検証するかどうかを指定します。無視するかどうかを指定します。

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

> プレーンテキストでパスワードを一覧表示することは推奨されません。Ansibleバックアップツールを使用するか、を提供します `eseries_system_password` --extra -varsを使用してAnsibleを実行している場合。

3. 必要に応じて、ノードにインストールするコントローラファームウェア、NVSRAM、ドライブファームウェアを指定します。これらのファイルは、にダウンロードする必要があります `packages/` Ansibleを実行する前のディレクトリ。Eシリーズコントローラのファームウェアとnvsramをダウンロードできます "こちらをご覧ください" ドライブファームウェアを定義できます "こちらをご覧ください"：

```
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".
```

この設定を指定すると、コントローラのリブート（必要な場合）を含むすべてのファームウェアがAnsibleで自動的に更新され、追加のプロンプトは表示されません。これはBeeGFS/ホストI/Oに影響しないものと想定されていますが、原因 によってパフォーマンスが一時的に低下する可能性があります。

4. グローバルシステム構成のデフォルトを調整します。ここに示すオプションと値は、ネットアップのBeeGFSには一般的に推奨される設定ですが、必要に応じて調整することもできます。

```
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. グローバルなボリュームプロビジョニングをデフォルトに設定ここに示すオプションと値は、ネットアップのBeeGFSには一般的に推奨される設定ですが、必要に応じて調整することもできます。

```
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. 必要に応じて、次のベストプラクティスに留意しながら、ストレージプールとボリュームグループ用のドライブがAnsibleで選択される順序を調整します。

   a. 管理ボリューム/メタデータボリュームに使用する（小容量の可能性がある）ドライブから先に、ストレージボリュームを最後にリストします。

   b. ディスクシェルフ/ドライブエンクロージャのモデルに基づいて、使用可能なドライブチャネル間でドライブ選択順序を分散してください。たとえば、EF600で拡張が行われていない場合、ドライブ$0_{11}$はドライブチャネル1に、ドライブ$_{12}$23はドライブチャネルに配置されます。したがって、ドライブ選択のバランスを取るための戦略は、を選択することです `disk shelf:drive` 99：0、99：23、99：1、99：22などエンクロージャが複数ある場合は、1桁目の数字がドライブシェルフIDを表します。

```
# 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"
```

をクリックします "こちらをご覧ください" に、一般的なブロックノード構成を表す完全なインベントリファイルの例を示します。

# BeeGFSサービスを定義します

## BeeGFS管理サービスを定義します

## BeeGFSサービスは、グループ変数（group_vars）を使用して設定します。

### 概要

このセクションでは、BeeGFS管理サービスの定義について説明します。HAクラスタには、このタイプのサービスを特定のファイルシステムに対して1つだけ配置する必要があります。このサービスには、次の定義が含まれます。

- サービスタイプ（管理）。
- このBeeGFSサービスにのみ適用する設定を定義します。
- このサービスに到達できる1つ以上のフローティングIP（論理インターフェイス）の設定。
- このサービス（BeeGFS管理ターゲット）のデータを格納する場所と方法を指定します。

### 手順

新しいファイルを作成し `group_vars/mgmt.yml`、"ファイルシステムを計画"セクションを参照して次のように入力します。

1. BeeGFS管理サービスの設定を表すファイルを指定します。

```
beegfs_service: management
```

2. このBeeGFSサービスにのみ適用する設定を定義します。通常、からサポートされている設定パラメータを指定してクォータを有効にする必要がないかぎり、この設定は管理サービスには必要ありません `beegfs-mgmtd.conf` 含めることができます。次のパラメータは、自動的に設定されますが、ここでは指定しないでください。 `storeMgmtdDirectory`、 `connAuthFile`、 `connDisableAuthentication`、 `connInterfacesFile`および `connNetFilterFile`。

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

3. 他のサービスやクライアントがこのサービスへの接続に使用する1つまたは複数のフローティングIPを設定します（これにより、自動的にBeeGFSが設定されます） `connInterfacesFile` オプション）：

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

4. 必要に応じて、発信通信に使用できるIPサブネットを1つ以上指定します（これにより、BeeGFSが自動的に設定されます） `connNetFilterFile` オプション）：

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

5. 次のガイドラインに従って、このサービスがデータを保存するBeeGFS管理ターゲットを指定します。
   a. 複数のBeeGFSサービス/ターゲットに同じストレージプール名またはボリュームグループ名を使用できます。必ず同じ名前を使用してください `name`、 `raid_level`、 `criteria_*`および `common_*` それぞれの構成（サービスごとに表示されるボリュームは異なるはずです）。
   b. ボリュームサイズは、ストレージプール/ボリュームグループの割合として指定します。また、特定のストレージプール/ボリュームグループを使用するすべてのサービス/ボリュームで、合計サイズが100を超えないようにします。注SSDを使用する場合は、SSDのパフォーマンスと寿命を最大限にするために、ボリュームグループに空きスペースをいくらか残しておくことを推奨します（クリックして"こちらをご覧ください"詳細を確認）。
   c. をクリックします "こちらをご覧ください" で使用可能なすべての設定オプションのリストを表示するには、を参照してください `eseries_storage_pool_configuration`。などのオプションに注意してください `state`、 `host`、 `host_type`、 `workload_name`および `workload_metadata` ボリューム名は自動的に生成されるため、ここでは指定しないでください。

```
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
```

をクリックします "こちらをご覧ください" たとえば、BeeGFS管理サービスを表す完全なインベントリファイルの例を示します。

**BeeGFS**メタデータサービスを定義します

## BeeGFSサービスは、グループ変数（group_vars）を使用して設定します。

概要

このセクションでは、BeeGFSメタデータサービスの定義手順を説明します。このタイプのサービスは、特定のファイルシステムのHAクラスタに少なくとも1つ存在する必要があります。このサービスには、次の定義が含まれます。

- サービスのタイプ（メタデータ）。

- このBeeGFSサービスにのみ適用する設定を定義します。
- このサービスに到達できる1つ以上のフローティングIP（論理インターフェイス）の設定。
- このサービス（BeeGFSメタデータターゲット）のデータを格納する場所と方法を指定します。

手順

"ファイルシステムを計画"セクションを参照し、 `group_vars/meta_<ID>.yml`クラスタ内の各メタデータサービスについてファイルをに作成し、次のように設定します。

1. BeeGFSメタデータサービスの設定を表すファイルを指定します。

```
beegfs_service: metadata
```

2. このBeeGFSサービスにのみ適用する設定を定義します。でサポートされる設定パラメータは、最小で目的のTCPポートとUDPポートを指定する必要があります `beegfs-meta.conf` このほか、次のパラメータは、自動的に設定されますが、ここでは指定しないでください。 `sysMgmtdHost`、 `storeMetaDirectory`、 `connAuthFile`、 `connDisableAuthentication`、 `connInterfacesFile`および `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. 他のサービスやクライアントがこのサービスへの接続に使用する1つまたは複数のフローティングIPを設定します（これにより、自動的にBeeGFSが設定されます） `connInterfacesFile` オプション）：

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

4. 必要に応じて、発信通信に使用できるIPサブネットを1つ以上指定します（これにより、BeeGFSが自動的に設定されます） `connNetFilterFile` オプション）：

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

5. 次のガイドラインに従って、このサービスがデータを格納するBeeGFSメタデータターゲットを指定します（これにより、が自動的に設定されます） `storeMetaDirectory` オプション）：

   a. 複数のBeeGFSサービス/ターゲットに同じストレージプール名またはボリュームグループ名を使用できます。必ず同じ名前を使用してください `name`、 `raid_level`、 `criteria_*`および

`common_*` それぞれの構成（サービスごとに表示されるボリュームは異なるはずです）。

b. ボリュームサイズは、ストレージプール/ボリュームグループの割合として指定します。また、特定のストレージプール/ボリュームグループを使用するすべてのサービス/ボリュームで、合計サイズが100を超えないようにします。注SSDを使用する場合は、SSDのパフォーマンスと寿命を最大限にするために、ボリュームグループに空きスペースをいくらか残しておくことを推奨します（クリックして"こちらをご覧ください"詳細を確認）。

c. をクリックします "こちらをご覧ください" で使用可能なすべての設定オプションのリストを表示するには、を参照してください `eseries_storage_pool_configuration`。などのオプションに注意してください `state`、 `host`、 `host_type`、 `workload_name`および `workload_metadata` ボリューム名は自動的に生成されるため、ここでは指定しないでください。

```
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
```

をクリックします "こちらをご覧ください" たとえば、BeeGFSメタデータサービスを表す完全なインベントリファイルの例を示します。

**BeeGFS**ストレージサービスを定義します

## BeeGFSサービスは、グループ変数（group_vars）を使用して設定します。

概要

このセクションでは、BeeGFSストレージサービスの定義手順を説明します。このタイプのサービスは、特定のファイルシステムのHAクラスタに少なくとも1つ存在する必要があります。このサービスには、次の定義が含まれます。

- サービスのタイプ（storage）。
- このBeeGFSサービスにのみ適用する設定を定義します。
- このサービスに到達できる1つ以上のフローティングIP（論理インターフェイス）の設定。
- このサービス（BeeGFSストレージターゲット）のデータを格納する場所/方法を指定します。

手順

"ファイルシステムを計画"セクションを参照して、 `group_vars/stor_<ID>.yml`クラスタ内の各ストレージサ

ービスのファイルをに作成し、次のように設定します。

1. BeeGFSストレージサービスの設定を表すファイルを指定します。

```
beegfs_service: storage
```

2. このBeeGFSサービスにのみ適用する設定を定義します。でサポートされる設定パラメータは、最小で目的のTCPポートとUDPポートを指定する必要があります `beegfs-storage.conf` このほか、次のパラメータは、自動的に設定されますが、ここでは指定しないでください。 `sysMgmtdHost`、 `storeStorageDirectory`、 `connAuthFile`、 `connDisableAuthentication`、 `connInterfacesFile`および `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. 他のサービスやクライアントがこのサービスへの接続に使用する1つまたは複数のフローティングIPを設定します（これにより、自動的にBeeGFSが設定されます） `connInterfacesFile` オプション）：

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

4. 必要に応じて、発信通信に使用できるIPサブネットを1つ以上指定します（これにより、BeeGFSが自動的に設定されます） `connNetFilterFile` オプション）：

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

5. 次のガイドラインに従って、このサービスがデータを保存するBeeGFSストレージターゲットを指定します（これにより、も自動的に設定されます） `storeStorageDirectory` オプション）：

   a. 複数のBeeGFSサービス/ターゲットに同じストレージプール名またはボリュームグループ名を使用できます。必ず同じ名前を使用してください `name`、 `raid_level`、 `criteria_*`および `common_*` それぞれの構成（サービスごとに表示されるボリュームは異なるはずです）。

   b. ボリュームサイズは、ストレージプール/ボリュームグループの割合として指定します。また、特定のストレージプール/ボリュームグループを使用するすべてのサービス/ボリュームで、合計サイズが100を超えないようにします。注SSDを使用する場合は、SSDのパフォーマンスと寿命を最大限にするために、ボリュームグループに空きスペースをいくらか残しておくことを推奨します（クリックして"こちらをご覧ください"詳細を確認）。

   c. をクリックします "こちらをご覧ください" で使用可能なすべての設定オプションのリストを表示する

には、を参照してください `eseries_storage_pool_configuration`。などのオプションに注意してください `state`、 `host`、 `host_type`、 `workload_name`および `workload_metadata` ボリューム名は自動的に生成されるため、ここでは指定しないでください。

```
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
```

をクリックします "こちらをご覧ください" たとえば、BeeGFSストレージサービスを表す完全なインベントリファイルの例を示します。

## BeeGFSサービスをファイルノードにマッピングします

### を使用して、各BeeGFSサービスを実行できるファイルノードを指定します `inventory.yml` ファイル。

#### 概要

このセクションでは、を作成する方法について説明します `inventory.yml` ファイル。これには、すべてのブロックノードのリストを表示し、各BeeGFSサービスを実行できるファイルノードを指定することも含まれます。

#### 手順

ファイルを作成します `inventory.yml` 次のように入力します。

1. ファイルの上部から、標準のAnsibleインベントリ構造を作成します。

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

2. このHAクラスタに含まれるすべてのブロックノードを含むグループを作成します。

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

3. クラスタ内のすべてのBeeGFSサービスとそれらを実行するファイルノードを含むグループを作成します。

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

4. クラスタ内のBeeGFSサービスごとに、そのサービスを実行する優先ファイルノードとセカンダリファイルノードを定義します。

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

をクリックします "こちらをご覧ください" 完全なインベントリファイルの例を示します。

# BeeGFSファイルシステムを導入します

## Ansible Playbookの概要

### Ansibleを使用したBeeGFS HAクラスタの導入と管理

#### 概要

前のセクションでは、BeeGFS HAクラスタを表すAnsibleインベントリを構築するために必要な手順を説明しました。このセクションでは、ネットアップがクラスタの導入と管理を行うAnsibleによる自動化を紹介します。

### Ansible：重要な概念

開始する前に、Ansibleの主要な概念を理解しておくと役立ちます。

- Ansibleインベントリに対して実行されるタスクは、* Playbook *と呼ばれるもので定義されています。
  - Ansibleのほとんどのタスクは*べき等値*となるように設計されているため、何度も実行して、必要な構成や状態が適用されていることを確認することができます。その際、作業を中断したり、不要な更新を加える必要はありません。
- Ansibleで実行される最小単位は*モジュール*です。
  - 一般的なプレイブックでは、複数のモジュールを使用
    - 例：パッケージのダウンロード、構成ファイルの更新、サービスの開始/有効化
  - NetApp Eシリーズシステムを自動化するために、モジュールを配布
- 複雑な自動化はロールとしてより適切にパッケージ化されています。
  - 基本的には、再利用可能なプレイブックを配布するための標準形式です。
  - LinuxホストとBeeGFSファイルシステムに役割を配布します。

### AnsibleのBeeGFS HAロール：主な概念

ネットアップ上のBeeGFSの各バージョンの導入と管理に必要なすべての自動化機能がAnsibleのロールとしてパッケージ化され、の一部として提供されます "BeeGFSに対応したNetApp EシリーズAnsibleコレクション"：

- この役割は、BeeGFS用の*インストーラ*と最新の*導入/管理*エンジンの間にあると考えることができます。
  - コードの手法や理念として最新のインフラを活用し、あらゆる規模のストレージインフラをシンプルに管理できます。
  - この"久保スプレー"プロジェクトでは、スケールアウトコンピューティングインフラ向けにKubernetesディストリビューション全体を導入/保守できるようになります。
- この役割は、ネットアップのソリューションでBeeGFSをパッケージ化、配布、保守するためにネットアップが使用する*ソフトウェア定義*形式です。
  - Linuxディストリビューション全体や大きなイメージを配布することなく、「アプライアンスのような」エクスペリエンスを実現できるように努力してください。
  - カスタムのBeeGFSターゲットとIPアドレスに対応したネットアップがオーサリングしたOpen Cluster Framework（OCF）準拠のクラスタリソースエージェントで構成され、高度なPacemakerとBeeGFSを統合するための監視機能が提供されます。
- この役割は、単に導入を「自動化」するものではなく、以下を含むファイルシステムのライフサイクル全体を管理することを目的としています。
  - サービス単位またはクラスタ全体の設定変更および更新を適用する。
  - ハードウェアの問題が解決されたあとのクラスタの修復とリカバリの自動化
  - BeeGFSとネットアップのボリュームを使用した広範なテストに基づいてデフォルト値を設定することで、パフォーマンスの調整を簡易化
  - 構成のずれの検証と修正

ネットアップは、向けのAnsibleのロールも提供しています "BeeGFSクライアント"必要に応じて、BeeGFSの

インストールとファイルシステムのマウントを行い、/GPU/ログインノードを計算します。

## BeeGFS HAクラスタを導入します

### プレイブックを使用してBeeGFS HAクラスタを導入するために実行するタスクを指定します。

#### 概要

このセクションでは、ネットアップでBeeGFSを導入/管理するために使用する標準的なプレイブックを組み立てる方法について説明します。

#### 手順

##### Ansible Playbookを作成

ファイルを作成します `playbook.yml` 次のように入力します。

1. 最初に、一連のタスクを定義します（一般的には、と呼ばれます） "再生"）が実行されるのはNetApp Eシリーズのブロックノードだけです。インストールを実行する前に確認を求めて（誤ってプレイブックが実行されないように）、をインポートします `nar_santricity_management` ロール。このロールは、で定義されている一般的なシステム構成の適用を処理します `group_vars/eseries_storage_systems.yml` または個人 `host_vars/<BLOCK NODE>.yml` ファイル。

```
- 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. すべてのファイルノードおよびブロックノードに対して実行する再生を定義します。

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

3. このアプローチでは、必要に応じて、HAクラスタを導入する前に実行する一連の「事前タスク」を定義できます。これは、Pythonなどの前提条件を確認してインストールするのに役立ちます。また、提供されたAnsibleタグがサポートされていることを確認するなど、任意のプリフライトチェックを実行することもできます。

```
  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. 最後に、導入するBeeGFSのバージョンに応じてBeeGFS HAロールをインポートします。

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

> BeeGFS HAロールは、サポートされるメジャーマイナーバージョンのBeeGFSごとに維持されます。これにより、ユーザはメジャー/マイナーバージョンをいつアップグレードするかを選択できます。現在、BeeGFS 7.3.x(`beegfs_7_3`）またはBeeGFS 7.2.x(`beegfs_7_2`）のいずれかがサポートされています。デフォルトでは、どちらのロールでも最新のBeeGFSパッチバージョンがリリース時に導入されますが、必要に応じてこれを上書きして最新のパッチを導入することもできます。"アップグレードガイド"詳細については、最新のを参照してください。

5. オプション：追加のタスクを定義する場合は、タスクの指示を考慮してください `all` ホスト（Eシリーズストレージシステムを含む）またはファイルノードのみ。必要に応じて、を使用して、ファイルノードを対象とした新しいプレイを定義します `- hosts: ha_cluster`。

をクリックします "こちらをご覧ください" に、完全なPlaybookファイルの例を示します。

#### NetApp Ansibleコレクションをインストールします

AnsibleのBeeGFSコレクションとすべての依存関係は維持されます "Ansible Galaxy"。Ansibleコントロールノードで次のコマンドを実行して最新バージョンをインストールします。

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

通常は推奨されませんが、コレクションの特定のバージョンをインストールすることもできます。

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

#### Playbookを実行してください

を含むAnsibleコントロールノードのディレクトリから `inventory.yml` および `playbook.yml` ファイルでは、次のようにプレイブックを実行します。

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

クラスタのサイズによっては、初期導入に20分以上かかることがあります。何らかの理由で導入が失敗した場合は、問題を修正し（ケーブルの接続ミス、ノードの起動など）、Ansibleプレイブックを再起動するだけです。

を指定するときに"共通ファイルノード構成"、接続ベースの認証をAnsibleで自動的に管理するデフォルトオプションを選択した場合、 `connAuthFile`共有シークレットとして使用されているが `<playbook_dir>/files/beegfs/<sysMgmtdHost>_connAuthFile`（デフォルト）に表示されるようになりました。ファイルシステムにアクセスする必要があるクライアントは、この共有シークレットを使用する必要があります。これは、クライアントがを使用して設定されている場合に自動的に処理され"BeeGFSクライアントの役割"ます。

## BeeGFSクライアントを導入します

また、Ansibleを使用してBeeGFSクライアントを設定し、ファイルシステムをマウントすることもできます。

#### 概要

BeeGFSファイルシステムにアクセスするには、ファイルシステムをマウントする必要のある各ノードにBeeGFSクライアントをインストールして設定する必要があります。このセクションでは、使用可能なを使用してこれらのタスクを実行する方法について説明します "Ansibleのロール"。

#### 手順

##### クライアントインベントリファイルを作成します

1. 必要に応じて、Ansibleコントロールノードから、BeeGFSクライアントとして設定する各ホストにパスワードなしのSSHを設定します。

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

2. の下 `host_vars/`をクリックし、という名前のBeeGFSクライアントごとにファイルを作成します `<HOSTNAME>.yml` 次の内容を使用して、プレースホルダテキストに環境に適した情報を入力します。

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

3. NetApp Eシリーズホストコレクションのロールを使用して、クライアントがBeeGFSファイルノードに接続するためのInfiniBandインターフェイスまたはイーサネットインターフェイスを設定する場合は、オプションで次のいずれかを指定します。

   a. ネットワークタイプがの場合 "InfiniBand（IPoIBを使用）"：

   ```
   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. ネットワークタイプがの場合 "RDMA over Converged Ethernet（RoCE）"：

   ```
   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. ネットワークタイプがの場合 "イーサネット（TCPのみ、RDMAなし）"：

   ```
   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. 新しいファイルを作成します `client_inventory.yml` さらに、Ansibleが各クライアントに接続するために使用するユーザを指定します。また、パスワードがAnsibleで権限の昇格（これにはが必要です `ansible_ssh_user` rootにするか、sudo権限を持っているか）：

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

パスワードをプレーンテキストで保存しないでください。代わりにAnsible Vaultを使用します（を参照してください） "Ansibleのドキュメント" Ansible Vaultを使用してコンテンツを暗号化する場合）またはを使用します `--ask-become-pass` プレイブックを実行する際のオプション。

5. を参照してください `client_inventory.yml` ファイルに、の下でBeeGFSクライアントとして設定する必要があるすべてのホストをリストします `beegfs_clients` グループ化し、インラインコメントを参照して、BeeGFSクライアントカーネルモジュールをシステムに構築するために必要な追加設定のコメントを外します。

```
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ドライバを使用する場合は、beegfs_client_ofed_include_pathが、使用しているLinuxのインストールに適した「ヘッダーインクルードパス」を指定していることを確認してください。詳細については、BeeGFSのドキュメントを参照してください "RDMAのサポート"。

6. を参照してください `client_inventory.yml` ファイルで、以前に定義した任意の下にマウントするBeeGFSファイルシステムを一覧表示します `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および7.3.1以降で"接続認証"は、設定または明示的に無効にする必要があります。を指定するときに接続ベースの認証を設定する方法によっては"共通ファイルノード構成"、クライアント設定の調整が必要になる場合があります。

   a. デフォルトでは、HAクラスタ環境で自動的に接続認証が設定され、が生成されます `connauthfile` に配置/管理されます `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile`。デフォルトでは、BeeGFSクライアントの役割は、で定義したクライアントにこのファイルを読み取り/配布するように設定されています `client_inventory.yml`追加のアクションは必要ありません。

      i. 詳細オプションについては、に付属のすべてのデフォルト設定を参照してください "BeeGFSクライアントの役割"。

   b. でカスタムシークレットを指定する場合は、を使用します `beegfs_ha_conn_auth_secret` で指定します `client_inventory.yml` ファイルも同様：

```
beegfs_ha_conn_auth_secret: <SECRET>
```

   c. で接続ベースの認証を完全に無効にする場合は、を使用します `beegfs_ha_conn_auth_enabled` で、を指定します `client_inventory.yml` ファイルも同様：

```
beegfs_ha_conn_auth_enabled: false
```

サポートされるパラメータの一覧およびその他の詳細については、を参照してください "BeeGFSクライアントの完全なドキュメント"。クライアントインベントリの完全な例については、をクリックしてください "こちらをご覧ください"。

#### BeeGFS Client Playbookファイルを作成します

1. 新しいファイルを作成します `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. オプション：NetApp Eシリーズホストコレクションのロールを使用して、クライアントがBeeGFSファイルシステムに接続するためのインターフェイスを設定する場合は、設定するインターフェイスタイプに対応するロールをインポートします。
   a. InfiniBand（IPoIB）を使用している場合は、次の手順を実行します。

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

   b. を使用している環境でRDMA over Converged Ethernet（RoCE）を使用している場合：

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

   c. 使用しているネットワークがイーサネット（TCPのみ、RDMAはなし）の場合：

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

3. 最後に、BeeGFSクライアントの役割をインポートしてクライアントソフトウェアをインストールし、フ

ファイルシステムをマウントします。

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

クライアントのプレイブックの完全な例については、をクリックしてください "こちらをご覧ください"。

#### BeeGFS Client Playbookを実行します

クライアントをインストール/ビルドしてBeeGFSをマウントするには、次のコマンドを実行します。

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

## BeeGFSの導入を確認します

### ファイルシステムを本番環境に導入する前に、ファイルシステムの導入を確認してください。

#### 概要

BeeGFSファイルシステムを本番環境に移行する前に、いくつかの検証チェックを実行します。

#### 手順

1. クライアントにログインして次のコマンドを実行し、想定されるすべてのノードが存在するか到達可能であり、不整合やその他の問題が報告されていないことを確認します。

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

2. クラスタ全体をシャットダウンし、再起動します。任意のファイルノードから、次のコマンドを実行します。

```
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. 各ノードをスタンバイにし、BeeGFSサービスがセカンダリノードにフェイルオーバーできることを確認します。このログインを任意のファイルノードに行うには、次のコマンドを実行します。

```
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やMDTestなどのパフォーマンスベンチマークツールを使用して、ファイルシステムのパフォーマンスが期待どおりであることを確認します。BeeGFSで使用される一般的なテストとパラメータの例については"設計検証"、「BeeGFS on NetApp Verified Architecture」を参照してください。

追加テストは、特定のサイト/設置環境に対して定義された受け入れ基準に基づいて実施する必要があります。

# 機能と統合を展開する

## BeeGFS CSI ドライバー

## BeeGFS v8のTLS暗号化を設定する

BeeGFS v8 管理サービスとクライアント間の通信を保護するために TLS 暗号化を構成します。

### 概要

BeeGFS v8では、管理ツール（`beegfs`コマンドラインユーティリティなど）とBeeGFSサーバーサービス（ManagementやRemoteなど）間のネットワーク通信を暗号化するためのTLSサポートが導入されました。このガイドでは、3つのTLS設定方法を使用して、BeeGFSクラスターでTLS暗号化を設定する方法について説明します：

- 信頼できる証明機関の使用： BeeGFS クラスターで既存の CA 署名証明書を使用します。
- ローカル認証局の作成： ローカル認証局を作成し、それを使用してBeeGFSサービスの証明書に署名します。このアプローチは、外部CAに依存せずに独自の信頼チェーンを管理したい環境に適しています。
- **TLS** 無効： 暗号化が不要な環境やトラブルシューティングを行う場合は、TLS を完全に無効にしてください。内部ファイルシステムの構造や設定に関する機密情報が平文で公開される可能性があるため、この方法は推奨されません。

環境と組織のポリシーに最適な方法を選択してください。詳細については、"BeeGFS TLS"ドキュメントを参照してください。

> ⓘ `beegfs-client`サービスを実行しているマシンでは、`BeeGFS` ファイルシステムをマウントするために `TLS` は必要ありません。`BeeGFS CLI` や、リモートや同期などの他の `beegfs` サービスを利用するには、`TLS` を設定する必要があります。

### 信頼できる証明機関の使用

信頼できる証明機関（CA）によって発行された証明書（社内のエンタープライズ CA またはサードパーティプロバイダーの CA）にアクセスできる場合は、自己署名証明書を生成する代わりに、これらの CA 署名証明書を使用するように BeeGFS v8 を構成できます。

#### 新しい **BeeGFS v8** クラスターのデプロイ

新しい BeeGFS v8 クラスターのデプロイメントでは、Ansible インベントリの `user_defined_params.yml`ファイルを設定して CA 署名付き証明書を参照します：

```
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`が空でない場合、Ansibleは指定されたalt_namesを証明書のサブジェクト別名（SAN）として使用し、自己署名TLS証明書と鍵を自動的に生成します。 `beegfs_ha_tls_cert_src_path`および`beegfs_ha_tls_key_src_path`で指定された独自のカスタムTLS証明書と鍵を使用するには、`beegfs_ha_tls_config_options`セクション全体をコメントアウトするか削除する必要があります。そうしないと、自己署名証明書の生成が優先され、カスタム証明書と鍵は使用されません。

既存の **BeeGFS v8** クラスターの構成

既存の BeeGFS v8 クラスターの場合は、BeeGFS 管理サービスの設定ファイル内のパスをファイルノードの CA 署名証明書に設定します：

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

**CA**署名証明書を使用した**BeeGFS v8**クライアントの構成

BeeGFS v8クライアントがシステムの証明書プールを使用してCA署名証明書を信頼するように設定するには、各クライアントの設定で tls-cert-file = "" を設定します。システム証明書プールを使用していない場合は、tls-cert-file = <local cert> を設定してローカル証明書へのパスを指定します。この設定により、クライアントはBeeGFS管理サービスによって提示された証明書を認証できるようになります。

## ローカル認証局の作成

BeeGFSクラスタ用に独自の証明書インフラストラクチャを構築する場合は、ローカル証明機関（CA）を作成して、BeeGFSクラスタ用の証明書の発行と署名を行うことができます。このアプローチでは、BeeGFS管理サービスの証明書に署名するCAを作成し、署名された証明書をクライアントに配布して信頼チェーンを確立します。以下の手順に従ってローカルCAを設定し、既存または新規のBeeGFS v8クラスタに証明書を展開してください。

新しい **BeeGFS v8** クラスターのデプロイ

BeeGFS v8の新規デプロイメントでは、 `beegfs_8` Ansibleロールがコントロールノード上のローカルCAの作成と、管理サービスに必要な証明書の生成を処理します。これは、Ansibleインベントリの `user_defined_params.yml` ファイルで以下のパラメータを設定することで有効化できます：

```
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`が指定されていない場合、Ansibleは指定された証明書/キーパス内の既存の証明書の使用を試みます。

既存の **BeeGFS v8** クラスターの構成

既存のBeeGFSクラスタでは、ローカル証明機関を作成し、管理サービスに必要な証明書を生成することでTLSを統合できます。BeeGFS管理サービスの設定ファイル内のパスを、新しく作成した証明書を指すように更新してください。

このセクションの手順は参考としてご利用ください。秘密鍵と証明書を扱う際には、適切なセキュリティ対策を講じてください。

認証局を作成する

信頼できるマシンに、BeeGFS管理サービスの証明書に署名するためのローカル証明機関（CA）を作成します。CA証明書はクライアントに配布され、信頼関係を確立し、BeeGFSサービスとの安全な通信を可能にします。

次の手順は、RHEL ベースのシステムでローカル認証局を作成するためのリファレンスです。

1. OpenSSL がまだインストールされていない場合はインストールします：

```
dnf install openssl
```

2. 証明書ファイルを保存するための作業ディレクトリを作成します：

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

3. CA 秘密キーを生成します：

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

4. `ca.cnf`という名前の CA 構成ファイルを作成し、識別名フィールドを組織に合わせて調整します：

```
[ 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証明書を生成します。この証明書はシステムの運用期間中有効である必要があります。有効でない場合は、有効期限が切れる前に証明書を再生成する必要があります。証明書の有効期限が切れると、一部のコンポーネント間の通信が不可能になり、TLS証明書の更新を完了するには通常、サービスの再起動が必要になります。

   次のコマンドは、1 年間有効な CA 証明書を生成します：

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

> この例では、簡潔にするために 1 年間の有効期間を使用していますが、組織のセキュリティ要件に応じて `-days`パラメータを調整し、証明書の更新プロセスを確立する必要があります。

管理サービス証明書を作成する

BeeGFS管理サービス用の証明書を生成し、作成したCAで署名します。これらの証明書は、BeeGFS管理サービスを実行するファイルノードにインストールされます。

1. 管理サービスの秘密キーを生成します：

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

2. `tls_san.cnf`という名前の証明書構成ファイルを作成し、すべての管理サービス IP アドレスに対してサブジェクト別名（SAN）を設定します：

```
[ 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>
```

識別名フィールドを更新して、CA 構成と `IP.1`および `IP.2`値を管理サービスの IP アドレスと一致させます。

3. 証明書署名要求（CSR）を生成します：

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

4. CA を使用して証明書に署名します（有効期間 1 年）：

```
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
```

組織のセキュリティポリシーに基づいて証明書の有効期間(`-days 365`を調整してください。多くの組織では、1〜2年ごとに証明書のローテーションが必要です。

5. 証明書が正しく作成されたことを確認します：

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

Subject Alternative Name セクションにすべての管理 IP アドレスが含まれていることを確認します。

#### ファイルノードに証明書を配布する

CA 証明書と管理サービス証明書を適切なファイル ノードとクライアントに配布します。

1. CA 証明書と管理サービス証明書およびキーを管理サービスを実行しているファイル ノードにコピーします：

```
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/
```

#### 管理サービスを**TLS**証明書に指定する

BeeGFS 管理サービスの構成を更新して TLS を有効にし、作成された TLS 証明書を参照します。

1. BeeGFS 管理サービスを実行しているファイルノードから、管理サービス構成ファイル（例：`/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml`）を編集します。次の TLS 関連パラメータを追加または更新します：

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

2. 変更を有効にするには、適切なアクションを実行して BeeGFS 管理サービスを安全に再起動してください：

```
systemctl restart beegfs-mgmtd
```

3. 管理サービスが正常に開始されたことを確認します：

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

TLS の初期化と証明書の読み込みが成功したことを示すログエントリを探します。

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

#### BeeGFS v8クライアントのTLSを設定する

BeeGFS 管理サービスとの通信を必要とするすべての BeeGFS クライアントに、ローカル CA によって署名された証明書を作成して配布します。

1. 上記の管理サービス証明書と同じプロセスを使用してクライアントの証明書を生成しますが、Subject Alternative Name（SAN）フィールドにクライアントの IP アドレスまたはホスト名を指定します。
2. クライアントの証明書をクライアントに安全にリモートコピーし、クライアント上で証明書の名前を `cert.pem`に変更します：

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

3. すべてのクライアントで BeeGFS クライアント サービスを再起動します：

```
systemctl restart beegfs-client
```

4. `beegfs CLI`コマンドを実行して、クライアントの接続を確認します。次に例を示します：

```
beegfs health check
```

## TLSの無効化

TLSはトラブルシューティングのため、またはユーザーの希望に応じて無効にすることができます。ただし、内部ファイルシステムの構造や設定に関する機密情報が平文で公開される可能性があるため、無効にすることは推奨されません。既存または新規のBeeGFS v8クラスターでTLSを無効にするには、以下の手順に従ってください。

#### 新しい BeeGFS v8 クラスターのデプロイ

新しい BeeGFS クラスターのデプロイメントでは、Ansible インベントリの `user_defined_params.yml`ファイルで次のパラメーターを設定することで、TLS を無効にしてクラスターをデプロイできます：

```
beegfs_ha_tls_enabled: false
```

#### 既存の BeeGFS v8 クラスターの構成

既存のBeeGFS v8クラスタの場合は、管理サービスの設定ファイルを編集します。例えば、`/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml`のファイルを編集し、以下のように設定します：

```
tls-disable = true
```

変更を有効にするには、適切な処置を行って管理サービスを安全に再起動してください。

# BeeGFSクラスタの管理

## 概要、主要な概念、用語

BeeGFS HAクラスタの導入後の管理方法について説明します。

### 概要

このセクションは、BeeGFS HAクラスタを導入したあとに管理する必要があるクラスタ管理者を対象としています。Linux HAクラスタに詳しい場合でも、このガイドをよくお読みください。クラスタの管理方法には、特にAnsibleによる再設定に関していくつかの違いがあるためです。

### 主な概念

これらの概念の一部はメイン"用語と概念"ページで紹介されていますが、BeeGFS HAクラスタのコンテキストで再紹介すると便利です。

クラスタノード: PacemakerサービスとCorosyncサービスを実行しており、HAクラスタに参加しているサーバ。

ファイルノード: 1つ以上のBeeGFS管理'メタデータ'またはストレージサービスを実行するために使用されるクラスタノード

ブロックノード:ファイルノードにブロックストレージを提供するNetApp Eシリーズストレージシステム。これらのノードは独自のスタンドアロンHA機能を提供するため、BeeGFS HAクラスタには参加しません。各ノードは、ブロックレイヤで高可用性を提供する2台のストレージコントローラで構成されます。

- BeeGFSサービス** BeeGFS管理'メタデータ'ストレージ・サービス各ファイルノードは、ブロックノード上のボリュームを使用してデータを格納する1つ以上のサービスを実行します。

ビルディングブロック： BeeGFSファイルノード、EシリーズブロックノードおよびBeeGFSサービスを標準で導入し、NetApp Verified Architectureに基づくBeeGFS HAクラスタ/ファイルシステムのスケールアウトを簡易化します。カスタムHAクラスタもサポートされますが、多くの場合、拡張を簡易化するビルディングブロック方式が採用されています。

- BeeGFS HA Cluster：**ブロックノードによってサポートされるBeeGFSサービスの実行に使用される拡張可能なファイルノード数。BeeGFSデータを高可用性で保存します。業界で実証済みのオープンソースコンポーネントPacemakerとCorosyncにAnsibleを使用してパッケージと導入を行いました。

クラスタサービスは、クラスタに参加している各ノードでPacemakerサービスおよびCorosyncサービスを実行していることを意味します。注：ノードでBeeGFSサービスを実行せずに、2つのファイルノードしか必要としない場合は「Tiebreaker」ノードとしてクラスタに参加するだけで済みます。

クラスタリソース:クラスタ内で実行されているBeeGFSサービスごとに、BeeGFSモニタリソースと、BeeGFSターゲットのリソース、IPアドレス（フローティングIP）、BeeGFSサービスそのものを含むリソースグループが表示されます。

- Ansible：**ソフトウェアのプロビジョニング、構成管理、アプリケーション導入のためのツールで、コードとしてのインフラストラクチャを実現します。BeeGFS HAクラスタをパッケージ化することで、ネットアップのBeeGFSの導入、再構成、更新を簡易化します。

- PC:**クラスタ内の任意のファイルノードから使用可能なコマンドラインインターフェイス。クラスタ内のノードおよびリソースの状態を照会および制御するために使用します。

### 一般的な用語

フェールオーバー:各BeeGFSサービスには、そのノードで障害が発生しない限り、実行される優先ファイルノードがあります。BeeGFSサービスが非優先/セカンダリファイルノードで実行されている場合は'フェイルオーバー中と呼ばれます

フェイルバック:優先されないファイルノードから優先ノードにBeeGFSサービスを移動する動作。

- HAペア：**同じブロックノードのセットにアクセスできる2つのファイルノードは、HAペアと呼ばれることもあります。ネットアップ全体で、相互にテイクオーバーできる2台のストレージコントローラまたはノードを指します。

**Maintenance Mode**：すべてのリソース監視を無効にし、Pacemakerによるクラスタ内のリソースの移動や管理を防止します（の項も参照"メンテナンスモード"）。

- HAクラスタ:**クラスタ内の複数のノード間でフェイルオーバー可能なBeeGFSサービスを実行する1つ以上のファイルノードが'高可用性BeeGFSファイルシステムを作成します多くの場合、ファイルノードはHAペアに構成され、クラスタ内のBeeGFSサービスのサブセットを実行できるようになります。

# AnsibleとPCSツールを使用するタイミング

## HAクラスタの管理にAnsibleとPCコマンドラインツールを使用する必要があるのはどのような場合ですか？

外部のAnsibleコントロールノードからAnsibleを使用して、クラスタの導入と再設定のすべてのタスクを完了する必要があります。クラスタの状態の一時的な変更（ノードのスタンバイへの切り替えなど）は、通常、クラスタの1つのノード（デグレード状態でないノードやメンテナンスを予定しているノード）にログインし、PCSコマンドラインツールを使用して実行します。

リソース、制約、プロパティ、BeeGFSサービスなどのクラスタ構成を変更するには、必ずAnsibleを使用します。Ansibleのインベントリとプレイブックの最新のコピーを維持すること（変更を追跡するためのソース管理に理想的）は、クラスタの管理の一部です。設定に変更を加える必要がある場合は、インベントリを更新してAnsibleプレイブックを再実行し、BeeGFS HAロールをインポートします。

HAロールでは、クラスタをメンテナンスモードにしてから、BeeGFSまたはクラスタサービスを再起動して新しい設定を適用する前に必要な変更を行います。通常、最初の導入ではノードの完全なリブートは必要ありませんが、Ansibleの再実行は一般に「安全な」手順 とみなされます。ただし、メンテナンス時間中や、BeeGFSサービスの再起動が必要になった場合は、オフ時間帯に行うことを推奨します。このような再起動によって、通常は原因 アプリケーションエラーが発生することはありませんが、パフォーマンスが低下する可能性があります（一部のアプリケーションは他のアプリケーションよりも処理能力が高い）。

Ansibleの再実行も、クラスタ全体を最適な状態に戻したい場合に選択できます。場合によっては、PCを使用する場合よりも簡単にクラスタの状態をリカバリできる可能性があります。特に、何らかの理由でクラスタが停止した緊急時に、すべてのノードが稼働状態に戻ってからAnsibleを再実行すると、PCを使用するよりも迅速かつ確実にクラスタをリカバリできます。

# クラスタの状態を確認します

PCを使用してクラスタの状態を表示します。

## 概要

実行中です `pcs status` いずれかのクラスタノードから、クラスタの全体的な状態と各リソースのステータス（BeeGFSサービスやその依存関係など）を確認する最も簡単な方法があります。このセクションでは、の出力で確認できる内容について説明します `pcs status` コマンドを実行します

## からの出力を理解する `pcs status`

を実行します `pcs status` クラスタサービス（PacemakerとCorosync）が開始されているクラスタノードで実行します。出力の一番上にクラスタの概要が表示されます。

```
[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
```

次のセクションには、クラスタ内のノードが表示されます。

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

これは、スタンバイまたはオフラインのノードを示します。スタンバイ状態のノードは引き続きクラスタに参加していますが、リソースを実行する資格がありません。ノードがオフラインの場合は、そのノードでクラスタサービスが実行されていないことを示します。これは、手動で停止されているか、ノードがリブートまたはシャットダウンされたことが原因です。

ノードの初回起動時にクラスタサービスが停止し、リソースが異常なノードに誤ってフェイルバックされないように手動で開始する必要があります。

管理者以外の理由（障害など）によりノードがスタンバイまたはオフラインになっている場合、ノードの状態の横に括弧内に追加のテキストが表示されます。たとえば、フェンシングが無効で、リソースに障害が発生した場合などです `Node <HOSTNAME>: standby (on-fail)`。もう1つの状態はです `Node <HOSTNAME>:

UNCLEAN (offline)`を使用すると、ノードがフェンシングされていると短時間だけ表示されますが、クラスタがノードの状態を確認できないことを示すフェンシングが失敗した場合も維持されます（これにより、リソースが他のノードで開始されない場合があります）。

次のセクションでは、クラスタ内のすべてのリソースとその状態のリストを示します。

```
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
[...]
```

ノードと同様に、リソースに問題がある場合は、リソースの状態の横にかっこで囲んだテキストが追加で表示されます。たとえば、Pacemakerがリソースの停止を要求し、割り当てられた時間内に完了できなかった場合、Pacemakerはノードの遮断を試みます。フェンシングが無効になっているか、フェンシング処理が失敗した場合、リソースの状態はになります `FAILED <HOSTNAME> (blocked)` また、Pacemakerは別のノードで起動できません。

BeeGFS HAクラスタでは、最適化されたいくつかのBeeGFSカスタムOCFリソースエージェントを使用することに注意してください。特に、BeeGFSモニタは、特定のノードのBeeGFSリソースを使用できない場合にフェイルオーバーをトリガーします。

# HAクラスタとBeeGFSを再設定します

Ansibleを使用してクラスタを再構成します。

## 概要

通常、BeeGFS HAクラスタの再設定は、Ansibleインベントリを更新して `ansible-playbook`コマンドを再実行することで行う必要があります。これには、アラートの更新、永続的なフェンシング設定の変更、BeeGFSサービス設定の調整などが含まれます。これらは `group_vars/ha_cluster.yml`ファイルを使用して調整され、オプションの完全なリストは"Common File Node Configurationを指定します"セクションにあります。

一部の設定オプションについては、以下を参照してください。これらのオプションを選択した場合、管理者は、クラスタのメンテナンスやサービスを行う際に注意が必要になります。

## フェンシングを無効にして有効にする方法

フェンシングは、クラスタのセットアップ時にデフォルトで有効/必要になります。場合によっては、フェンシングを一時的に無効にして、特定のメンテナンス処理（オペレーティングシステムのアップグレードなど）の実行時にノードが誤ってシャットダウンされないようにすることが望ましい場合もあります。この機能は手動で無効にできますが、管理者が注意する必要があるトレードオフがあります。

オプション**1**：**Ansible**によるフェンシングを無効にします（推奨）。

Ansibleを使用してフェンシングを無効にすると、BeeGFSモニタの失敗時のアクションが「フェンス」から「standby」に変更されます。つまり、BeeGFSモニタが障害を検出すると、ノードがスタンバイ状態になり、すべてのBeeGFSサービスをフェイルオーバーしようとします。一般的に、オプション2よりも、外部のアクティブなトラブルシューティング/テストの方が望ましい。短所は、あるリソースが元のノードで停止しないと、そのリソースが他の場所からブロックされる場合です（そのため、通常は本番環境のクラスタでフェンシングが必要となります）。

1. Ansibleのインベントリで `groups_vars/ha_cluster.yml` 次の構成を追加します。

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

2. Ansibleプレイブックを再実行して、クラスタに変更を適用します。

オプション**2**：フェンシングを手動で無効にする

場合によっては、Ansibleなしでフェンシングを一時的に無効にすることもできます。たとえば、クラスタのトラブルシューティングやテストを実施する場合などです。

この設定では、BeeGFSモニタが障害を検出すると、クラスタは対応するリソースグループの停止を試みます。フルフェイルオーバーはトリガーされず、影響を受けるリソースグループを再起動したり別のホストに移動したりすることもありません。リカバリするには、問題を解決してからを実行します `pcs resource cleanup` または、手動でノードをスタンバイにします。

手順

1. フェンシング（stonith）がグローバルに有効になっているか無効になっているかを確認するには、次のコマンドを実行 `pcs property show stonith-enabled`
2. フェンシングをディセーブルにするには、 `pcs property set stonith-enabled=false`
3. フェンシングを有効にするには、次の `pcs property set stonith-enabled=true`

この設定は、次回 Ansible プレイブックを実行するときに上書きされます。

# HAクラスタコンポーネントの更新

## BeeGFS サービスのアップグレード

Ansible を使用して、HA クラスターで実行されている BeeGFS バージョンを更新します。

概要

BeeGFSには `major.minor.patch`バージョン管理方式が採用されています。`BeeGFS HA Ansible`のロールは、サポートされる `major.minor`バージョンごとに用意されています（ `beegfs_ha_7_2`やなど

`beegfs_ha_7_3`）。各HAロールは、Ansibleコレクションのリリース時に利用可能な最新のBeeGFSパッチバージョンに固定されています。

BeeGFSのすべてのアップグレードにはAnsibleを使用する必要があります。これには、BeeGFSのメジャーバージョン、マイナーバージョン、パッチバージョン間の移行も含まれます。BeeGFSをアップデートするには、まずBeeGFS Ansibleコレクションをアップデートする必要があります。これにより、デプロイメント/管理自動化と基盤となるHAクラスタの最新の修正と機能強化も取得されます。コレクションを最新バージョンにアップデートした後でも、 `-e "beegfs_ha_force_upgrade=true"`を設定して `ansible-playbook`を実行するまでBeeGFSはアップグレードされません。各アップグレードの詳細については、現在のバージョンの"BeeGFSアップグレードのドキュメント"を参照してください。

> BeeGFS v8 にアップグレードする場合は、代わりに"BeeGFS v8にアップグレード"手順を参照してください。

テスト済みのアップグレードパス

次のアップグレード パスがテストおよび検証されています：

| 元のバージョン | アップグレードバージョン | マルチレール | 詳細 |
|---|---|---|---|
| 7.2.6. | 7.3.2の場合 | はい。 | beegfsコレクションをv3.0.1からv3.1.0にアップグレードすると'マルチレールが追加されました |
| 7.2.6. | 7.2.8 | いいえ | beegfsコレクションをv3.0.1からv3.1.0にアップグレードしています |
| 7.2.8 | 7.3.1 | はい。 | beegfs collection v3.1.0を使用してアップグレードすると、マルチレールが追加されました |
| 7.3.1 | 7.3.2の場合 | はい。 | beegfs collection v3.1.0を使用してアップグレードします |
| 7.3.2の場合 | 7.4.1 | はい。 | beegfs collection v3.2.0を使用してアップグレードします |
| 7.4.1 | 7.4.2 | はい。 | beegfs collection v3.2.0を使用してアップグレードします |
| 7.4.2 | 7.4.6 | はい。 | beegfs collection v3.2.0を使用してアップグレードします |
| 7.4.6 | 8.0 | はい。 | "BeeGFS v8にアップグレード"手順の指示に従ってアップグレードします。 |
| 7.4.6 | 8.1 | はい。 | "BeeGFS v8にアップグレード"手順の指示に従ってアップグレードします。 |
| 7.4.6 | 8.2 | はい。 | "BeeGFS v8にアップグレード"手順の指示に従ってアップグレードします。 |

**BeeGFS**のアップグレード手順

次のセクションでは、BeeGFS AnsibleコレクションとBeeGFS自体を更新する手順を示します。BeeGFSのメジャーバージョンまたはマイナーバージョンを更新する際は、特に注意してください。

手順**1**：**BeeGFS**コレクションをアップグレードする

へのアクセスによる収集のアップグレード "Ansible Galaxy"を使用して、次のコマンドを実行します。

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

オフラインでの収集アップグレードの場合は、から収集をダウンロードします "Ansible Galaxy" 目的のをクリックします `Install Version` 次に `Download tarball`。tarballをAnsibleコントロールノードに転送し、次のコマンドを実行します。

```
ansible-galaxy collection install netapp_eseries-beegfs-<VERSION>.tar.gz
--upgrade
```

を参照してください "コレクションのインストール" を参照してください。

#### ステップ2：Ansibleインベントリを更新する

クラスタのAnsibleインベントリファイルに必要な更新や希望する更新を行ってください。具体的なアップグレード要件の詳細については、以下の[バージョンのアップグレードに関する注意事項]セクションをご覧ください。BeeGFS HAインベントリの設定に関する一般的な情報については、"Ansibleのインベントリの概要"セクションをご覧ください。

#### 手順3：Ansible Playbookを更新する（メジャーバージョンまたはマイナーバージョンを更新する場合のみ）

メジャーバージョンとマイナーバージョンを切り替える場合は、クラスタの導入とメンテナンスに使用するファイルで、 `playbook.yml` ロールの名前を目的のバージョンに合わせて更新し `beegfs_ha_<VERSION>` ます。たとえば、BeeGFS 7.4を導入する場合は、次のようになり `beegfs_ha_7_4`ます。

```
- 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
```

このPlaybookファイルの内容の詳細については、を参照してください"BeeGFS HAクラスタを導入します"。

#### 手順4：BeeGFSのアップグレードを実行する

BeeGFSアップデートを適用するには：

```
ansible-playbook -i inventory.yml beegfs_ha_playbook.yml -e
"beegfs_ha_force_upgrade=true" --tags beegfs_ha
```

BeeGFS HAの役割では、次の処理が行われます。

- 各BeeGFSサービスが優先ノードに配置された状態で、クラスタが最適な状態であることを確認します。
- クラスタをメンテナンスモードにします。
- HAクラスタのコンポーネントを更新します（必要な場合）。
- 各ファイルノードを次のように1つずつアップグレードします。
  - スタンバイにし、サービスをセカンダリノードにフェイルオーバーします。
  - BeeGFSパッケージをアップグレードします。
  - フォールバックサービス。
- クラスタをメンテナンスモードから切り替えます。

バージョンのアップグレードに関する注意事項

**BeeGFS**バージョン**7.2.6**または**7.3.0**からアップグレードしています

接続ベースの認証に対する変更

BeeGFS バージョン 7.3.2 以降では、接続ベースの認証を設定する必要があります。次のいずれかを行わないと、サービスは起動しません：

- `connAuthFile`の指定、または
- サービスの構成ファイルで `connDisableAuthentication=true`を設定する。

セキュリティのため、接続ベースの認証を有効にすることを強くお勧めします。詳細については、"BeeGFS Connection Based Authenticationの略"をご覧ください。

`beegfs_ha*`ロールは認証ファイルを自動的に生成し、次の場所に配布します：

- クラスター内のすべてのファイルノード
- `<playbook_directory>/files/beegfs/<beegfs_mgmt_ip_address>_connAuthFile`のAnsibleコントロールノード

`beegfs_client`ロールは、このファイルが存在する場合、それを自動的に検出してクライアントに適用します。

`beegfs_client`ロールを使用してクライアントを設定しなかった場合は、各クライアントに認証ファイルを手動で配布し、 `beegfs-client.conf`ファイル内の `connAuthFile`設定を構成する必要があります。接続ベースの認証がないBeeGFSバージョンからアップグレードする場合、 `group_vars/ha_cluster.yml`で `beegfs_ha_conn_auth_enabled: false`を設定してアップグレード中に接続ベースの認証を無効にしない限り、クライアントはアクセスできなくなります（推奨されません）。

追加の詳細と代替構成オプションについては、"Common File Node Configurationを指定します" セクションの接続認証構成手順を参照してください。

## BeeGFS v8にアップグレード

BeeGFS HA クラスターをバージョン 7.4.6 から BeeGFS v8 にアップグレードするには、次の手順に従います。

### 概要

BeeGFS v8では、BeeGFS v7からアップグレードする前に追加の設定が必要となる重要な変更がいくつか導入されています。このドキュメントでは、BeeGFS v8の新しい要件に合わせてクラスターを準備し、BeeGFS v8にアップグレードする方法について説明します。

> ⓘ BeeGFS v8にアップグレードする前に、システムが少なくともBeeGFS 7.4.6を実行していることを確認してください。BeeGFS 7.4.6より前のリリースを実行しているクラスターは、このBeeGFS v8アップグレード手順を進める前に、まず"バージョン7.4.6にアップグレード"を行う必要があります。

### BeeGFS v8の主な変更点

BeeGFS v8 では、次の主要な変更が導入されています：

- ライセンスの適用： BeeGFS v8では、ストレージプール、リモートストレージターゲット、BeeONDなどのプレミアム機能を使用するにはライセンスが必要です。アップグレード前に、BeeGFSクラスターの有効なライセンスを取得してください。必要に応じて、一時的なBeeGFS v8評価ライセンスを"BeeGFSライセンスポータル"から取得できます。
- 管理サービス データベースの移行： BeeGFS v8 の新しい TOML ベース形式による構成を有効にするには、BeeGFS v7 管理サービス データベースを更新された BeeGFS v8 形式に手動で移行する必要があります。
- **TLS**暗号化： BeeGFS v8では、サービス間の安全な通信のためにTLSが導入されています。アップグレードの一環として、BeeGFS管理サービスと `beegfs`コマンドラインユーティリティ用のTLS証明書を生成して配布する必要があります。

BeeGFS 8 の詳細と追加の変更については、"BeeGFS v8.0.0 アップグレードガイド"を参照してください。

> ⓘ BeeGFS v8へのアップグレードには、クラスタのダウンタイムが必要です。また、BeeGFS v7クライアントはBeeGFS v8クラスタに接続できません。運用への影響を最小限に抑えるため、クラスタとクライアント間のアップグレードタイミングを慎重に調整してください。

### BeeGFS クラスターをアップグレード用に準備する

アップグレードを開始する前に、スムーズな移行を実現し、ダウンタイムを最小限に抑えるために環境を慎重に準備してください。

1. クラスターが正常な状態であり、すべてのBeeGFSサービスがそれぞれの優先ノードで実行されていることを確認してください。BeeGFSサービスを実行しているファイルノードから、すべてのPacemakerリソースがそれぞれの優先ノードで実行されていることを確認します：

```
pcs status
```

2. クラスター構成を記録してバックアップします。
   a. クラスタ構成のバックアップ手順については、"BeeGFS Backupのドキュメント"を参照してください。
   b. 既存の管理データ ディレクトリをバックアップします：

```
cp -r /mnt/mgmt_tgt_mgmt01/data
/mnt/mgmt_tgt_mgmt01/data_beegfs_v7_backup_$(date +%Y%m%d)
```

   c. beegfs クライアントから次のコマンドを実行し、出力を参照用に保存します：

```
beegfs-ctl --getentryinfo --verbose /path/to/beegfs/mountpoint
```

   d. ミラーリングを使用する場合は、詳細な状態情報を収集します：

```
beegfs-ctl --listtargets --longnodes --state --spaceinfo
--mirrorgroups --nodetype=meta
beegfs-ctl --listtargets --longnodes --state --spaceinfo
--mirrorgroups --nodetype=storage
```

3. クライアントのダウンタイムに備えて `beegfs-client`サービスを停止します。各クライアントで以下を実行します：

```
systemctl stop beegfs-client
```

4. 各Pacemakerクラスタで、STONITHを無効にします。これにより、不要なノードの再起動をトリガーすることなく、アップグレード後にクラスタの整合性を検証できます。

```
pcs property set stonith-enabled=false
```

5. BeeGFS 名前空間内のすべての Pacemaker クラスターでは、PCS を使用してクラスターを停止します：

```
pcs cluster stop --all
```

**BeeGFS**パッケージをアップグレードする

クラスター内のすべてのファイルノードに、Linuxディストリビューション用のBeeGFS v8パッケージリポジトリを追加してください。公式BeeGFSリポジトリの使用方法については、"BeeGFSダウンロードページ"をご覧ください。それ以外の場合は、ローカルのbeegfsミラーリポジトリを適切に設定してください。

以下の手順は、RHEL 9 ファイルノード上の公式 BeeGFS 8.2 リポジトリを使用した手順です。クラスター内のすべてのファイルノードで以下の手順を実行してください：

1. BeeGFS GPG キーをインポートします：

```
rpm --import https://www.beegfs.io/release/beegfs_8.2/gpg/GPG-KEY-beegfs
```

2. BeeGFS リポジトリをインポートします：

```
curl -L -o /etc/yum.repos.d/beegfs-rhel9.repo
https://www.beegfs.io/release/beegfs_8.2/dists/beegfs-rhel9.repo
```

> ⓘ 新しい BeeGFS v8 リポジトリとの競合を避けるため、以前に設定された BeeGFS リポジトリを削除します。

3. パッケージ マネージャーのキャッシュを消去します：

```
dnf clean all
```

4. すべてのファイル ノードで、BeeGFS パッケージを BeeGFS 8.2 に更新します。

```
dnf update beegfs-mgmtd beegfs-storage beegfs-meta libbeegfs-ib
```

> ⓘ 標準クラスターでは、 `beegfs-mgmtd`パッケージは最初の2つのファイル ノードでのみ更新されます。

管理データベースをアップグレードする

BeeGFS 管理サービスを実行しているファイルノードの 1 つで、次の手順を実行して、管理データベースを BeeGFS v7 から v8 に移行します。

1. すべての NVMe デバイスを一覧表示し、管理対象をフィルタリングします：

```
nvme netapp smdevices | grep mgmt_tgt
```

   a. 出力からデバイスパスをメモします。

   b. 管理対象デバイスを既存の管理対象マウント ポイントにマウントします（ `/dev/nvmeXnY`をデバイス パスに置き換えます）：

```
mount /dev/nvmeXnY /mnt/mgmt_tgt_mgmt01/
```

2. 次のコマンドを実行して、BeeGFS 7 管理データを新しいデータベース形式にインポートします：

```
/opt/beegfs/sbin/beegfs-mgmtd --import-from
-v7=/mnt/mgmt_tgt_mgmt01/data/
```

期待される出力：

```
Created new database version 3 at "/var/lib/beegfs/mgmtd.sqlite".
Successfully imported v7 management data from
"/mnt/mgmt_tgt_mgmt01/data/".
```

BeeGFS v8では検証要件が厳格化されているため、自動インポートが必ずしも成功するとは限りません。例えば、ターゲットが存在しないストレージプールに割り当てられている場合、インポートは失敗します。移行に失敗した場合は、アップグレードを続行しないでください。データベース移行問題の解決については、NetAppサポートにお問い合わせください。当面の解決策として、BeeGFS v8パッケージをダウングレードし、問題が解決するまでBeeGFS v7を引き続き実行することができます。

3. 生成された SQLite ファイルを管理サービス マウントに移動します：

```
mv /var/lib/beegfs/mgmtd.sqlite /mnt/mgmt_tgt_mgmt01/data/
```

4. 生成された `beegfs-mgmtd.toml`を管理サービスマウントに移動します：

```
mv /etc/beegfs/beegfs-mgmtd.toml /mnt/mgmt_tgt_mgmt01/mgmt_config/
```

`beegfs-mgmtd.toml`構成ファイルの準備は、次のセクションのライセンスおよび TLS 構成手順を完了した後に行われます。

ライセンスを構成する

1. beegfs管理サービスを実行するすべてのノードにbeegfsライセンスパッケージをインストールします。通常はクラスタの最初の2つのノードです：

```
dnf install libbeegfs-license
```

2. BeeGFS v8 ライセンス ファイルを管理ノードにダウンロードし、次の場所に配置します：

```
/etc/beegfs/license.pem
```

**TLS暗号化を構成する**

BeeGFS v8では、管理サービスとクライアント間の安全な通信のためにTLS暗号化が必要です。管理サービスとクライアントサービス間のネットワーク通信でTLS暗号化を設定するには、3つのオプションがあります。推奨される最も安全な方法は、信頼できる証明機関によって署名された証明書を使用することです。あるいは、独自のローカルCAを作成して、BeeGFSクラスターの証明書に署名することもできます。暗号化が不要な環境やトラブルシューティングの場合は、TLSを完全に無効にすることもできますが、機密情報がネットワークに公開されるため、これは推奨されません。

続行する前に、"BeeGFS 8 の TLS 暗号化を設定する"ガイドの指示に従って、環境のTLS暗号化を設定してください。

更新管理サービスの構成

BeeGFS v7 構成ファイルから設定を手動で `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml`ファイルに転送して、BeeGFS v8 管理サービス構成ファイルを準備します。

1. 管理ターゲットがマウントされている管理ノードで、 `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.conf`BeeGFS 7の管理サービスファイルを参照し、すべての設定を `/mnt/mgmt_tgt_mgmt01/mgmt_config/beegfs-mgmtd.toml`ファイルに転送します。基本的な設定では、`beegfs-mgmtd.toml`は以下のようになります：

   ```
   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']
   ```

   必要に応じて、環境と TLS 構成に合わせてすべてのパスを調整します。

2. 管理サービスを実行している各ファイル ノードで、新しい構成ファイルの場所を指すように systemd サービス ファイルを変更します。

   ```
   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. systemdをリロードします：

```
systemctl daemon-reload
```

3. 管理サービスを実行している各ファイル ノードに対して、管理サービスの gRPC 通信用にポート 8010 を開きます。

a. ポート 8010/tcp を beegfs ゾーンに追加します：

```
sudo firewall-cmd --zone=beegfs --permanent --add-port=8010/tcp
```

b. 変更を適用するには、ファイアウォールをリロードします：

```
sudo firewall-cmd --reload
```

**BeeGFS**モニタースクリプトを更新する

Pacemaker `beegfs-monitor` OCFスクリプトは、新しいTOML構成形式とsystemdサービス管理をサポートするために更新が必要です。クラスター内の1つのノードでスクリプトを更新し、更新したスクリプトを他のすべてのノードにコピーしてください。

1. 現在のスクリプトのバックアップを作成します：

```
cp /usr/lib/ocf/resource.d/eseries/beegfs-monitor
/usr/lib/ocf/resource.d/eseries/beegfs-monitor.bak.$(date +%F)
```

2. 管理構成ファイルのパスを `.conf`から `.toml`に更新します：

```
sed -i 's|mgmt_config/beegfs-mgmtd\.conf|mgmt_config/beegfs-mgmtd.toml|'
/usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

または、スクリプト内で次のブロックを手動で見つけます：

```
case $type in
  management)
    conf_path="${configuration_mount}/mgmt_config/beegfs-mgmtd.conf"
    ;;
```

これを次のように置き換えます：

```
case $type in
  management)
    conf_path="${configuration_mount}/mgmt_config/beegfs-mgmtd.toml"
    ;;
```

3. `get_interfaces()`および `get_subnet_ips()`関数を更新して、TOML構成をサポートします：
   a. テキスト エディターでスクリプトを開きます：

   ```
   vi /usr/lib/ocf/resource.d/eseries/beegfs-monitor
   ```

   b. 2 つの関数 `get_interfaces()`と `get_subnet_ips()`を見つけます。
   c. `get_interfaces()`から `get_subnet_ips()`の終わりまで、両方の関数全体を削除します。
   d. 次の更新された関数をコピーして、その場所に貼り付けます：

```
# 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. テキストエディターを保存して終了します。

f. 続行する前に、次のコマンドを実行してスクリプトの構文エラーを確認してください。出力がない場合は、スクリプトの構文が正しいことを示します。

```
bash -n /usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

4. 更新された `beegfs-monitor` OCF スクリプトをクラスター内の他のすべてのノードにコピーして一貫性を確保します：

```
scp /usr/lib/ocf/resource.d/eseries/beegfs-monitor
user@node:/usr/lib/ocf/resource.d/eseries/beegfs-monitor
```

クラスタをオンラインに戻す

1. これまでのアップグレード手順がすべて完了したら、すべてのノードで BeeGFS サービスを開始して、クラスターをオンラインに戻します。

```
pcs cluster start --all
```

2. `beegfs-mgmtd`サービスが正常に開始されたことを確認します：

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

予想される出力には次のような行が含まれます：

```
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
```

ジャーナル ログにエラーが表示される場合は、管理構成ファイルのパスを確認し、すべての値が BeeGFS 7 構成ファイルから正しく転送されていることを確認します。

3. 実行 `pcs status`して、クラスターが正常であり、優先ノードでサービスが開始されていることを確認します。

4. クラスターが正常であることが確認されたら、STONITH を再度有効にします：

```
pcs property set stonith-enabled=true
```

5. 次のセクションに進み、クラスター内の BeeGFS クライアントをアップグレードし、BeeGFS クラスターの健全性を確認します。

### BeeGFSクライアントのアップグレード

クラスターを BeeGFS v8 に正常にアップグレードした後、すべての BeeGFS クライアントもアップグレードする必要があります。

次の手順では、Ubuntu ベースのシステムで BeeGFS クライアントをアップグレードするプロセスの概要を説明します。

1. まだ行っていない場合は、BeeGFS クライアント サービスを停止します：

```
systemctl stop beegfs-client
```

2. お使いのLinuxディストリビューションにBeeGFS v8パッケージリポジトリを追加してください。公式BeeGFSリポジトリの使い方については、"BeeGFSダウンロードページ"をご覧ください。それ以外の場合は、ローカルのBeeGFSミラーリポジトリを適切に設定してください。

   次の手順では、Ubuntu ベースのシステム上の公式 BeeGFS 8.2 リポジトリを使用します：

3. BeeGFS GPG キーをインポートします：

```
wget https://www.beegfs.io/release/beegfs_8.2/gpg/GPG-KEY-beegfs -O
/etc/apt/trusted.gpg.d/beegfs.asc
```

4. リポジトリ ファイルをダウンロードします：

```
wget https://www.beegfs.io/release/beegfs_8.2/dists/beegfs-noble.list -O
/etc/apt/sources.list.d/beegfs.list
```

> ⓘ 新しい BeeGFS v8 リポジトリとの競合を避けるため、以前に設定された BeeGFS リポジトリを削除します。

5. BeeGFS クライアント パッケージをアップグレードします：

```
apt-get update
apt-get install --only-upgrade beegfs-client
```

6. クライアントのTLSを設定します。BeeGFS CLIを使用するにはTLSが必要です。"BeeGFS 8 の TLS 暗号化を設定する"の手順を参照して、クライアントでTLSを設定してください。

7. BeeGFS クライアント サービスを開始します：

```
systemctl start beegfs-client
```

アップグレードを確認する

BeeGFS v8 へのアップグレードが完了したら、次のコマンドを実行して、アップグレードが成功したことを確認します。

1. ルートinodeが以前と同じメタデータノードによって所有されていることを確認します。管理サービスで`import-from-v7`機能を使用した場合、これは自動的に行われるはずです：

```
beegfs entry info /mnt/beegfs
```

2. すべてのノードとターゲットがオンラインで、良好な状態であることを確認します：

```
beegfs health check
```

「使用可能な容量」チェックでターゲットの空き容量が少ないという警告が表示された場合は、`beegfs-mgmtd.toml`ファイルに定義されている「容量プール」しきい値を調整して、環境に適したものにすることができます。

## HAクラスタでのPacemakerおよびCorosyncパッケージのアップグレード

HAクラスタ内のPacemakerおよびCorosyncパッケージをアップグレードする手順は、次のとおりです。

概要

PacemakerとCorosyncをアップグレードすることで、クラスタは新機能、セキュリティパッチ、およびパフォーマンスの向上の恩恵を受けることができます。

アップグレードアプローチ

クラスタのアップグレードには、ローリングアップグレードとクラスタの完全なシャットダウンの2つの方法が推奨されます。各アプローチには独自の利点と欠点があります。アップグレード手順は、Pacemakerのリリースバージョンによって異なる場合があります。使用するアプローチを決定するには、ClusterLabsのドキュメントを参照して"Pacemakerクラスタのアップグレード"ください。アップグレードアプローチに従う前に、次のことを確認してください。

- 新しいPacemakerおよびCorosyncパッケージは、NetApp BeeGFSソリューション内でサポートされています。
- BeeGFSファイルシステムおよびPacemakerクラスタ構成に対して有効なバックアップが存在します。
- クラスタは正常な状態です。

この方法では、各ノードをクラスタから削除してアップグレードし、すべてのノードで新しいバージョンが実行されるまでクラスタに再導入します。この方法ではクラスタの運用を維持できるため、大規模なHAクラスタには最適ですが、処理中にバージョンが混在するリスクがあります。この方法は2ノードクラスタでは使用しないでください。

1. 各BeeGFSサービスが優先ノードで実行され、クラスタが最適な状態であることを確認します。詳細については、を参照してください "クラスタの状態を確認します" 。
2. ノードをアップグレードするには、ノードをスタンバイモードにして、すべてのBeeGFSサービスを停止（または移動）します。

```
pcs node standby <HOSTNAME>
```

3. 次のコマンドを実行して、ノードのサービスが削除されたことを確認します。

```
pcs status
```

スタンバイのノードでサービスがとして報告されていないことを確認します `Started`。

> クラスタのサイズによっては、サービスが姉妹ノードに移動するまでに数秒から数分かかることがあります。姉妹ノードでBeeGFSサービスが開始されない場合は、を参照してください"トラブルシューティングガイド"。

4. ノードのクラスタをシャットダウンします。

```
pcs cluster stop <HOSTNAME>
```

5. ノードのPacemaker、Corosync、およびPCsパッケージをアップグレードします。

> パッケージマネージャのコマンドは、オペレーティングシステムによって異なります。次のコマンドは、RHEL 8以降を実行するシステム用です。

```
dnf update pacemaker-<version>
```

```
dnf update corosync-<version>
```

```
dnf update pcs-<version>
```

6. ノードでPacemakerクラスタサービスを開始します。

```
pcs cluster start <HOSTNAME>
```

7. パッケージが更新された場合は `pcs`、クラスタでノードを再認証します。

```
pcs host auth <HOSTNAME>
```

8. ツールを使用して、ペースメーカーの設定がまだ有効であることを確認します `crm_verify`。

> この検証は、クラスタのアップグレード時に1回だけ実行します。

```
crm_verify -L -V
```

9. ノードのスタンバイを解除します。

```
pcs node unstandby <HOSTNAME>
```

10. すべてのBeeGFSサービスを優先ノードに再配置します。

```
pcs resource relocate run
```

11. クラスタ内の各ノードで上記の手順を繰り返して、すべてのノードで目的のバージョンのペースメーカー、Corosync、およびPCが実行されるようにします。
12. 最後に、を実行し `pcs status`てクラスタが正常であることを確認し、で `Current DC`目的のPacemakerバージョンが報告されます。

> 「mixed-version」と表示される場合 `Current DC`は、クラスタ内のノードが以前のバージョンのPacemakerで実行されているため、アップグレードが必要です。アップグレードしたノードがクラスタに再参加できない場合、またはリソースが起動しない場合は、クラスタログを確認し、アップグレードの既知の問題についてPacemakerのリリースノートまたはユーザガイドを参照してください。

クラスタのシャットダウン後の処理

この方法では、すべてのクラスタノードとリソースをシャットダウンし、ノードをアップグレードしてから、クラスタを再起動します。この方法は、PacemakerバージョンとCorosyncバージョンが混在した構成をサポートしていない場合に必要です。

1. 各BeeGFSサービスが優先ノードで実行され、クラスタが最適な状態であることを確認します。詳細については、を参照してください "クラスタの状態を確認します" 。
2. すべてのノードでクラスタソフトウェア（PacemakerおよびCorosync）をシャットダウンします。

クラスタのサイズによっては、クラスタ全体が停止するまでに数秒から数分かかることがあります。

```
pcs cluster stop --all
```

3. すべてのノードでクラスタサービスがシャットダウンしたら、要件に応じて各ノードのPacemaker、Corosync、およびPCsパッケージをアップグレードします。

パッケージマネージャのコマンドは、オペレーティングシステムによって異なります。次のコマンドは、RHEL 8以降を実行するシステム用です。

```
dnf update pacemaker-<version>
```

```
dnf update corosync-<version>
```

```
dnf update pcs-<version>
```

4. すべてのノードをアップグレードしたら、すべてのノードでクラスタソフトウェアを起動します。

```
pcs cluster start --all
```

5. `pcs`パッケージが更新された場合は、クラスタ内の各ノードを再認証します。

```
pcs host auth <HOSTNAME>
```

6. 最後に、を実行し `pcs status`てクラスタが正常であることを確認し、で `Current DC`正しいPacemakerバージョンが報告されます。

「mixed-version」と表示される場合 `Current DC`は、クラスタ内のノードが以前のバージョンのPacemakerで実行されているため、アップグレードが必要です。

## ファイルノードアダプタファームウェアの更新

次の手順に従って、ファイルノードのConnectX-7アダプタを最新のファームウェアに更新します。

### 概要

新しいMLNX_OFEDドライバのサポート、新機能の有効化、バグの修正には、ConnectX-7アダプタファーム

ウェアの更新が必要になる場合があります。このガイドでは、使いやすさと効率性を考慮して、アダプタのアップデートにNVIDIAのユーティリティを使用し `mlxfwmanager`ます。

### アップグレード時の考慮事項

このガイドでは、ConnectX-7アダプタのファームウェアを更新する2つの方法（ローリング更新と2ノードクラスタ更新）について説明します。クラスタのサイズに応じて、適切な更新方法を選択します。ファームウェアの更新を実行する前に、次のことを確認します。

- サポートされているMLNX_OFEDドライバがインストールされている場合は、を参照してください。"テクノロジ要件"
- BeeGFSファイルシステムおよびPacemakerクラスタ構成に対して有効なバックアップが存在します。
- クラスタは正常な状態です。

### ファームウェアアップデートの準備

NVIDIAのユーティリティを使用して、NVIDIAのMLNX_OFEDドライバにバンドルされているノードのアダプタファームウェアを更新することを推奨し `mlxfwmanager`ます。アップデートを開始する前に、アダプタのファームウェアイメージをからダウンロードし"NVIDIAのサポートサイト"、各ファイルノードに保存します。

Lenovo ConnectX-7アダプタの場合は、NVIDIAのページにあるツールを"OEM ファームウェア"使用します `mlxfwmanager_LES`。

### ローリング更新アプローチ

ノードが3つ以上のHAクラスタでは、この方法が推奨されます。このアプローチでは、一度に1つのファイルノードのアダプタファームウェアを更新して、HAクラスタが要求の処理を継続できるようにしますが、この間はI/Oの処理を行わないことを推奨します。

1. 各BeeGFSサービスが優先ノードで実行され、クラスタが最適な状態であることを確認します。詳細については、を参照してください "クラスタの状態を確認します" 。
2. 更新するファイルノードを選択し、スタンバイモードにします。スタンバイモードでは、そのノードからすべてのBeeGFSサービスが削除（または移動）されます。

   ```
   pcs node standby <HOSTNAME>
   ```

3. 次のコマンドを実行して、ノードのサービスが削除されたことを確認します。

   ```
   pcs status
   ```

   スタンバイ状態のノードでサービスがとして報告されていないことを確認します `Started`。

   クラスタのサイズによっては、BeeGFSサービスが姉妹ノードに移動するまでに数秒から数分かかることがあります。姉妹ノードでBeeGFSサービスが開始されない場合は、を参照してください"トラブルシューティングガイド"。

4. を使用してアダプタファームウェアを更新し `mlxfwmanager`ます。

```
mlxfwmanager -i <path/to/firmware.bin> -u
```

ファームウェアのアップデートを受信している各アダプタのをメモします `PCI Device Name`。

5. ユーティリティを使用して各アダプタをリセットし、 `mlxfwreset`新しいファームウェアを適用します。

> 一部のファームウェアアップデートでは、アップデートを適用するために再起動が必要になる場合があります。詳細については、を参照してください"NVIDIAのmlxfwresetの制限事項"。リブートが必要な場合は、アダプタをリセットする代わりにリブートを実行します。

a. opensmサービスを停止します。

```
systemctl stop opensm
```

b. 前述の各コマンドについて、次のコマンドを実行し `PCI Device Name`ます。

```
mlxfwreset -d <pci_device_name> reset -y
```

c. opensmサービスを開始します。

```
systemctl start opensm
```

d. 再起動する `eseries_nvme_ib.service`。

```
systemctl restart eseries_nvme_ib.service
```

e. E シリーズ ストレージ アレイのボリュームが存在することを確認します。

```
multipath -ll
```

1. を実行し `ibstat`、すべてのアダプタが目的のファームウェアバージョンで実行されていることを確認します。

```
ibstat
```

2. ノードでPacemakerクラスタサービスを開始します。

```
pcs cluster start <HOSTNAME>
```

3. ノードのスタンバイを解除します。

```
pcs node unstandby <HOSTNAME>
```

4. すべてのBeeGFSサービスを優先ノードに再配置します。

```
pcs resource relocate run
```

すべてのアダプタが更新されるまで、クラスタ内のファイルノードごとに上記の手順を繰り返します。

**2**ノードクラスタ更新アプローチ

この方法は、ノードが2つだけのHAクラスタの場合に推奨されます。このアプローチはローリング更新に似ていますが、1つのノードのクラスタサービスが停止しているときにサービスのダウンタイムを回避するための手順が追加されています。

1. 各BeeGFSサービスが優先ノードで実行され、クラスタが最適な状態であることを確認します。詳細については、を参照してください "クラスタの状態を確認します" 。
2. 更新するファイルノードを選択し、ノードをスタンバイモードにします。スタンバイモードでは、そのノードからすべてのBeeGFSサービスが削除（または移動）されます。

```
pcs node standby <HOSTNAME>
```

3. 次のコマンドを実行して、ノードのリソースが枯渇したことを確認します。

```
pcs status
```

スタンバイ状態のノードでサービスがとして報告されていないことを確認します `Started`。

クラスタのサイズによっては、BeeGFSサービスが姉妹ノードとして報告されるまでに数秒から数分かかることがあります `Started`。BeeGFSサービスが起動しない場合は、を参照してください"トラブルシューティングガイド"。

4. クラスタをメンテナンスモードにします。

```
pcs property set maintenance-mode=true
```

5. を使用してアダプタファームウェアを更新し `mlxfwmanager`ます。

```
mlxfwmanager -i <path/to/firmware.bin> -u
```

ファームウェアのアップデートを受信している各アダプタのをメモします `PCI Device Name`。

6. ユーティリティを使用して各アダプタをリセットし、 `mlxfwreset`新しいファームウェアを適用します。

> 一部のファームウェアアップデートでは、アップデートを適用するために再起動が必要になる場合があります。詳細については、を参照してください"NVIDIAのmlxfwresetの制限事項"。リブートが必要な場合は、アダプタをリセットする代わりにリブートを実行します。

a. opensmサービスを停止します。

```
systemctl stop opensm
```

b. 前述の各コマンドについて、次のコマンドを実行し `PCI Device Name`ます。

```
mlxfwreset -d <pci_device_name> reset -y
```

c. opensmサービスを開始します。

```
systemctl start opensm
```

7. を実行し `ibstat`、すべてのアダプタが目的のファームウェアバージョンで実行されていることを確認します。

```
ibstat
```

8. ノードでPacemakerクラスタサービスを開始します。

```
pcs cluster start <HOSTNAME>
```

9. ノードのスタンバイを解除します。

```
pcs node unstandby <HOSTNAME>
```

10. クラスタのメンテナンスモードを終了します。

```
pcs property set maintenance-mode=false
```

11. すべてのBeeGFSサービスを優先ノードに再配置します。

```
pcs resource relocate run
```

すべてのアダプタが更新されるまで、クラスタ内のファイルノードごとに上記の手順を繰り返します。

## E-Seriesストレージアレイのアップグレード

### HAクラスタのEシリーズストレージアレイのコンポーネントをアップグレードする手順は、次のとおりです。

概要

HAクラスタのNetApp Eシリーズストレージアレイを最新のファームウェアで最新の状態に保つことで、最適なパフォーマンスとセキュリティが確保されます。ストレージアレイのファームウェア更新は、SANtricity OS、NVSRAM、およびドライブファームウェアファイルを使用して適用されます。

ストレージアレイはHAクラスタをオンラインにしたままアップグレードできますが、すべてのアップグレードでクラスタをメンテナンスモードにすることを推奨します。

ブロックノードのアップグレード手順

次の手順は `Netapp_Eseries.Santricity`、Ansibleコレクションを使用してストレージアレイのファームウェアを更新する方法の概要です。次の手順に進む前に、を参照して"アップグレード時の考慮事項"E-Seriesシステムを更新してください。

SANtricity OS 11.80以降のリリースへのアップグレードは、11.70.5P1以降でのみ可能です。以降のアップグレードを適用する前に、ストレージアレイを11.70.5P1にアップグレードしておく必要があります。

1. Ansibleコントロールノードが最新のSANtricity Ansibleコレクションを使用していることを確認します。
   - へのアクセスによる収集のアップグレード "Ansible Galaxy"を使用して、次のコマンドを実行します。

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

   - オフラインアップグレードの場合は、からcollection tarballをダウンロードし"Ansible Galaxy"、制御ノードに転送して、次のコマンドを実行します。

     ```
     ansible-galaxy collection install netapp_eseries-santricity-
     <VERSION>.tar.gz --upgrade
     ```

     を参照してください "コレクションのインストール" を参照してください。
2. ストレージアレイとドライブの最新のファームウェアを入手します。

a. ファームウェアファイルをダウンロードします。

   - * SANtricity OSとNVSRAM：*に移動し、"NetApp Support Site"お使いのストレージアレイモデルに対応した最新リリースのSANtricity OSとNVSRAMをダウンロードします。
   - *ドライブファームウェア：*に移動し"E-Seriesディスクファームウェアサイト"、ストレージアレイの各ドライブモデルに対応する最新のファームウェアをダウンロードします。

b. SANtricity OS、NVSRAM、およびドライブファームウェアのファイルをAnsibleの制御ノードの `<inventory_directory>/packages`ディレクトリに格納します。

3. 必要に応じて、クラスタのAnsibleインベントリファイルを更新して、更新が必要なすべてのストレージアレイ（ブロックノード）を含めます。手順については、を参照してください"Ansibleのインベントリの概要"。
4. 各BeeGFSサービスが優先ノードに配置され、クラスタが最適な状態であることを確認します。詳細については、を参照してください "クラスタの状態を確認します" 。
5. の手順に従って、クラスタをメンテナンスモードにし"クラスタをメンテナンスモードにします"ます。
6. という名前の新しいAnsibleプレイブックを作成し `update_block_node_playbook.yml`ます。Playbookに次の情報を入力し、SANtricity OS、NVSRAM、およびドライブファームウェアのバージョンを目的のアップグレードパスに置き換えます。

```
- 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. 更新を開始するには、Ansibleコントロールノードから次のコマンドを実行します。

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

8. プレイブックが完了したら、各ストレージアレイが最適な状態になっていることを確認します。
9. クラスタをメンテナンスモードから切り替え、各BeeGFSサービスが優先ノードに配置され、クラスタが最適な状態であることを確認します。

# サービスとメンテナンス

## フェイルオーバーサービスとフェイルバックサービス

## クラスタノード間でBeeGFSサービスを移動します。

### 概要

BeeGFSサービスは、ノード間でフェイルオーバーして、ノードに障害が発生した場合でもクライアントが引き続きファイルシステムにアクセスできるようにすることも、計画的なメンテナンスを実行することもできます。このセクションでは、障害からのリカバリ後、またはノード間でサービスを手動で移動したあとに、クラスタを修復するさまざまな方法について説明します。

### 手順

#### フェイルオーバーとフェイルバック

#### フェイルオーバー（計画的）

通常、保守のために1つのファイルノードをオフラインにする必要がある場合は、そのノードからすべてのBeeGFSサービスを移動（または削除）する必要があります。そのためには、最初にノードをスタンバイにします。

```
pcs node standby <HOSTNAME>
```

使用状況を確認したら `pcs status` すべてのリソースが代替ファイルノードで再起動されました。必要に応じて、ノードをシャットダウンしたり他の変更を加えたりできます。

#### フェイルバック（計画的フェイルオーバーのあと）

BeeGFSサービスを優先ノードにリストアする準備ができたら、最初に実行します `pcs status` ノードリストで、ステータスがstandbyになっていることを確認します。ノードをリブートした場合、クラスタサービスをオンラインにするまではオフラインと表示されます。

```
pcs cluster start <HOSTNAME>
```

ノードがオンラインになったら、次のコマンドを使用して、ノードをスタンバイ状態から解除します。

```
pcs node unstandby <HOSTNAME>
```

最後に、次のコマンドを使用してすべてのBeeGFSサービスを優先ノードに再配置します。

```
pcs resource relocate run
```

フェイルバック（計画外フェイルオーバー後）

ノードでハードウェアやその他の障害が発生した場合、HAクラスタが正常なノードに自動的に対応してサービスを移動し、管理者が適切に対処できるようにする必要があります。作業を進める前に、"トラブルシューティング"セクションを参照してフェイルオーバーの原因を特定し、未解決の問題を解決してください。ノードの電源がオンになり正常に戻ったら、フェイルバックを続行できます。

計画外（または計画的）リブート後にノードがブートした場合、クラスタサービスは自動的には開始されないため、最初にでノードをオンラインにする必要があります。

```
pcs cluster start <HOSTNAME>
```

次に、リソース障害をクリーンアップし、ノードのフェンシング履歴をリセットします。

```
pcs resource cleanup node=<HOSTNAME>
pcs stonith history cleanup <HOSTNAME>
```

で確認します `pcs status` ノードはオンラインで正常な状態です。デフォルトでは、BeeGFSサービスは、リソースを誤って正常なノードに戻すことを防ぐために、自動的にフェイルバックを行いません。準備ができたら、を指定してクラスタ内のすべてのリソースを優先ノードに戻します。

```
pcs resource relocate run
```

個 々 の**BeeGFS**サービスを代替ファイルノードに移動します

**BeeGFS**サービスを新しいファイルノードに永続的に移動します

個 々 のBeeGFSサービスの優先ファイルノードを永続的に変更するには、Ansibleのインベントリを調整して優先ノードがリストされるようにしてから、Ansibleプレイブックを再実行します。

たとえば、次のサンプルファイルでは `inventory.yml` 、beegfs_01がBeeGFS管理サービスの実行に優先されるファイルノードです。

```
        mgmt:
          hosts:
            beegfs_01:
            beegfs_02:
```

この順序を逆にすると、beegfs_02で管理サービスが優先されます原因 。

```
        mgmt:
          hosts:
            beegfs_02:
            beegfs_01:
```

**BeeGFS**サービスを一時的に代替ファイルノードに移動します

通常、ノードのメンテナンス中に[フェイルオーバーとフェイルバックの手順]（フェイルオーバーとフェイルバックの手順）を使用して、すべてのサービスをそのノードから移動します。

何らかの理由で、個 々 のサービスを別のファイルノードに移動する必要がある場合は、次のコマンドを実行します。

```
pcs resource move <SERVICE>-monitor <HOSTNAME>
```

> 個 々 のリソースまたはリソースグループを指定しないでください。再配置するBeeGFSサービスのモニタ名を必ず指定しますたとえば、BeeGFS管理サービスをbeegfs_02に移動するには、次のコマンドを実行し `pcs resource move mgmt-monitor beegfs_02` ます。このプロセスを繰り返して、1つ以上のサービスを優先ノードから移動できます。サービスが新しいノードで再配置/開始されたことを確認します `pcs status` 。

BeeGFSサービスを優先ノードに戻すには'最初に一時的なリソースの制約を解除します（複数のサービスの場合はこの手順を繰り返します）

```
pcs resource clear <SERVICE>-monitor
```

次に、サービスを実際に優先ノードに戻す準備ができたら、次のコマンドを実行します。

```
pcs resource relocate run
```

このコマンドは、優先ノードに配置されていない一時的なリソース制約のないサービスを再配置します。

## クラスタをメンテナンスモードにします

## 環境内の意図した変更にHAクラスタが誤って対応しないようにします。

概要

クラスタをメンテナンスモードにすると、リソースの監視がすべて無効になり、ペースメーカーによるクラスタ内のリソースの移動や管理ができなくなります。アクセスを妨げる一時的な障害が発生した場合でも、すべてのリソースは元のノードで実行されたままとなります。次のような場合に推奨/有用です。

- ファイルノードとBeeGFSサービス間の接続を一時的に中断する可能性のあるネットワークメンテナンス。

- ブロックノードのアップグレード。
- ファイルノードのオペレーティングシステム、カーネル、またはその他のパッケージの更新。

通常、クラスタを手動で保守モードにする唯一の理由は、環境内の外部の変更にクラスタが対応できないようにするためです。クラスタ内の個 々 のノードで物理的な修復が必要な場合は、保守モードを使用せずに、そのノードを上記の手順 に従ってスタンバイにします。Ansibleを再実行するとクラスタが自動的に保守モードになり、アップグレードや設定の変更など、ほとんどのソフトウェアメンテナンスが容易になります。

手順

クラスタがメンテナンスモードかどうかを確認するには、次のコマンドを実行します。

```
pcs property config
```

`maintenance-mode`クラスタが正常に動作している場合、プロパティは表示されません。クラスタが現在メンテナンスモードの場合、プロパティはとして報告されます `true`。メンテナンスモードの実行を有効にするには、次の

```
pcs property set maintenance-mode=true
```

PCステータスを実行し、すべてのリソースに「（管理対象外）」と表示されていることを確認することで確認できます。クラスタの保守モードを解除するには、次のコマンドを実行します。

```
pcs property set maintenance-mode=false
```

## クラスタを停止して起動します

## HAクラスタを正常に停止および起動します。

概要

ここでは、BeeGFSクラスタを正常にシャットダウンして再起動する方法について説明します。これが必要なシナリオの例としては、電気的なメンテナンスやデータセンター間またはラック間の移行などが挙げられます。

手順

何らかの理由でBeeGFSクラスタ全体を停止し、すべてのサービスをシャットダウンする必要がある場合は、次の手順を実行します。

```
pcs cluster stop --all
```

個 々 のノードでクラスタを停止することもできます（これにより、サービスが別のノードに自動的にフェイルオーバーされます）"フェイルオーバー"。ただし、最初にノードをスタンバイ状態にすることを推奨します（を参照）。

```
pcs cluster stop <HOSTNAME>
```

すべてのノードでクラスタサービスとリソースを開始するには、次のコマンドを実行します。

```
pcs cluster start --all
```

または、次のコマンドを使用して特定のノードでサービスを開始します。

```
pcs cluster start <HOSTNAME>
```

この時点でを実行します `pcs status` すべてのノードでクラスタサービスとBeeGFSサービスが開始され、必要なノードでサービスが実行されていることを確認します。

クラスタのサイズによっては、クラスタ全体が停止するまでに数秒から数分かかる場合やで開始されたと表示される場合があり `pcs status`ます。が5分以上ハングする場合 `pcs cluster <COMMAND>`は、「Ctrl+C」を実行してコマンドをキャンセルする前に、クラスタの各ノードにログインし、を使用して `pcs status`そのノードでクラスタサービス（Corosync / Pacemaker）が実行されているかどうかを確認します。クラスタがまだアクティブになっているノードから、クラスタをブロックしているリソースを確認できます。問題 に手動で対処する必要があります。コマンドを完了するか、再実行して残りのサービスを停止できます。

## ファイルノードを交換します

## 元のサーバに障害がある場合のファイルノードの交換

### 概要

クラスタ内のファイルノードを交換するために必要な手順の概要を以下に示します。これらの手順は、ハードウェア問題 が原因でファイルノードに障害が発生し、新しい同一のファイルノードに置き換えられたことを前提としています。

### 手順

1. ファイルノードを物理的に交換し、すべてのケーブルをブロックノードおよびストレージネットワークに接続します。
2. Red Hatサブスクリプションの追加を含め、ファイルノードにオペレーティングシステムを再インストールします。
3. ファイルノードで管理ネットワークとBMCネットワークを設定します。
4. ホスト名、IP、PCIeと論理インターフェイスのマッピング、または新しいファイルノードに関する変更があれば、Ansibleインベントリを更新します。通常この作業は、ノードを同じサーバハードウェアに交換し、元のネットワーク構成を使用している場合は必要ありません。

a. たとえば、ホスト名が変更された場合は、ノードのインベントリファイルを作成（または名前を変更）します (`host_vars/<NEW_NODE>.yml`）をクリックしてから、Ansibleインベントリファイルを使用してください (`inventory.yml`）で、古いノードの名前を新しいノード名に置き換えます。

```
all:
    ...
    children:
    ha_cluster:
        children:
        mgmt:
            hosts:
            node_h1_new:    # Replaced "node_h1" with "node_h1_new"
            node_h2:
```

5. クラスタ内の他のいずれかのノードから古いノードを削除します。 `pcs cluster node remove <HOSTNAME>`。

   この手順を実行する前に、次に進まないでください。

6. Ansibleコントロールノードで：
   a. 次のコマンドを使用して古いSSHキーを削除します。

```
`ssh-keygen -R <HOSTNAME_OR_IP>`
```

   b. 交換用ノードにパスワードなしのSSHを設定します。

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

7. Ansibleプレイブックを再実行してノードを設定し、クラスタに追加します。

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

8. この時点で、を実行します `pcs status` 交換したノードが表示され、サービスが実行されていることを確認します。

## クラスタを拡張または縮小します

## クラスタのビルディングブロックを追加または削除します。

### 概要

ここでは、BeeGFS HAクラスタのサイズを調整するためのさまざまな考慮事項とオプションについて説明します。通常、クラスタのサイズはビルディングブロックを追加または削除することで調整されます。ビルディ

ングブロックは通常、HAペアとして2つのファイルノードがセットアップされる構成です。必要に応じて、個々 のファイルノード（または他のタイプのクラスタノード）を追加または削除することもできます。

## クラスタへのビルディングブロックの追加

### 考慮事項

ビルディングブロックを追加してクラスタを拡張するプロセスは簡単です。個 々 のHAクラスタ内のクラスタノードの最小数と最大数に関する制限に注意して、既存のHAクラスタにノードを追加するか、新しいHAクラスタを作成するかを決定する必要があります。通常、各ビルディングブロックは2つのファイルノードで構成されますが、クラスタあたりの最小ノード数は3（クォーラムを確立するため）、推奨される最大ノード数は10（テスト済み）です。高度なシナリオでは、2ノードクラスタの導入時にBeeGFSサービスを実行しない「Tiebreaker」ノードを1つ追加できます。このような導入を検討される場合は、ネットアップサポートにお問い合わせください。

クラスタの拡張方法を決定する際には、これらの制限事項と、想定される将来のクラスタの拡張について留意してください。たとえば、6ノードクラスタの場合、ノードを4つ追加する必要があるときは、新しいHAクラスタを開始するだけで十分です。

1つのBeeGFSファイルシステムを複数の独立したHAクラスタで構成できますこれにより、基盤となるHAクラスタコンポーネントの推奨される制限やハード制限をはるかに超えてファイルシステムを継続的に拡張できます。

### 手順

クラスタにビルディングブロックを追加する場合は `host_vars`、新しいファイルノードおよびブロックノード（E-Seriesアレイ）ごとにファイルを作成する必要があります。これらのホストの名前は、作成する新しいリソースとともにインベントリに追加する必要があります。 `group_vars`新しいリソースごとに対応するファイルを作成する必要があります。"カスタムアーキテクチャの使用"詳細については、を参照してください。

正しいファイルを作成したら、コマンドを使用して自動化を再実行するだけです。

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

## クラスタからのビルディングブロックの削除

ビルディングブロックをリタイアさせる必要がある場合は、いくつかの考慮事項に留意する必要があります。次に例を示します。

- このビルディングブロックで実行されているBeeGFSサービスは何ですか。
- ファイルノードが撤去され、ブロックノードが新しいファイルノードに接続されるだけですか。
- ビルディングブロック全体が廃止される場合、データを新しいビルディングブロックに移動したり、クラスタ内の既存のノードに分散させたり、新しいBeeGFSファイルシステムやその他のストレージシステムに移動したりする必要がありますか。
- これはシステム停止中に発生するか、またはシステムを停止せずに実行する必要がありますか？
- ビルディングブロックがアクティブに使用されているか、またはアクティブではなくなったデータが主に含まれているか。

導入にあたっては、さまざまな出発点や希望する最終状態が考えられるため、ネットアップのサポートにご連

絡ください。ネットアップでは、お客様の環境と要件に基づいて最適な戦略を特定し、導入するお手伝いをいたします。

# トラブルシューティングを行う

## BeeGFS HAクラスタのトラブルシューティング

### 概要

このセクションでは、BeeGFS HAクラスタの運用中に発生する可能性のあるさまざまな障害やその他のシナリオを調査してトラブルシューティングする方法を説明します。

### トラブルシューティングガイド

予期しないフェイルオーバーを調査してい

ノードが予期せずフェンシングされていてサービスが別のノードに移動された場合は、の下部でクラスタがリソース障害を示していないかどうかを最初に確認する必要があります `pcs status`。通常、フェンシングが正常に完了し、リソースが別のノードで再起動された場合、何も表示されません。

通常、次の手順では、を使用してシステムログを検索します `journalctl` 残りのファイルノードのいずれか（Pacemakerログは、すべてのノードで同期されます）。障害が発生した時刻がわかっている場合は、障害が発生する直前に検索を開始できます（通常は10分前までに実行することをお勧めします）。

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>"
```

以降のセクションでは、ログに含まれているをgrepで検索できる一般的なテキストを示します。これにより、調査をさらに絞り込むことができます。

#### 調査/解決の手順

手順**1**：**BeeGFS**モニタで障害が検出されたかどうかを確認します。

BeeGFSモニタによってフェイルオーバーがトリガーされた場合は、エラーが表示されます（次の手順に進みない場合）。

```
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
```

このインスタンスでは、何らかの理由でBeeGFSサービスMETA_08が停止しました。トラブルシューティングを続行するには、beegfs_02を起動し、でサービスのログを確認する必要があります `/var/log/beegfs-meta-meta_08_tgt_0801.log`。たとえば、BeeGFSサービスで内部問題 やノードの問題が原因でアプリケーションエラーが発生した可能性があります。

Pacemakerのログとは異なり、BeeGFSサービスのログはクラスタ内のすべてのノードに分散されるわけではありません。これらのタイプの障害を調査するには、障害が発生した元のノードのログが必要です。

モニタで報告される可能性がある問題は次のとおりです。

- ターゲットにアクセスできません。
  - 概要 ：ブロックボリュームにアクセスできなかったことを示します。
  - トラブルシューティング：
    - 代替ファイルノードでサービスも開始できない場合は、ブロックノードが正常な状態であることを確認してください。
    - このファイルノードからブロックノードにアクセスできなくなる可能性がある物理的な問題がないかどうかを確認します。たとえば、InfiniBandアダプタまたはケーブルに障害がある場合などです。
- ネットワークに到達できません。
  - 概要 ：このBeeGFSサービスへの接続にクライアントが使用するアダプタがオンラインではありませんでした。
  - トラブルシューティング：
    - 複数またはすべてのファイルノードが影響を受けた場合は、BeeGFSクライアントとファイルシステムの接続に使用されるネットワークに障害が発生していないかどうかを確認します。
    - InfiniBandアダプタやケーブルの障害など、このファイルノードからクライアントにアクセスできなくなる物理的な問題がないかどうかを確認してください。
- BeeGFSサービスはアクティブではありません。
  - 概要 ：BeeGFSサービスが予期せず停止しました。
  - トラブルシューティング：
    - エラーが報告されたファイルノードで、影響を受けたBeeGFSサービスのログを調べて、クラッシュが報告されたかどうかを確認します。この場合は、ネットアップサポートに問い合わせてクラッシュを調査できるようにケースをオープンしてください。
    - BeeGFSログにエラーが報告されていない場合は、ジャーナルログを調べて、サービスが停止した理由がシステムに記録されているかどうかを確認します。一部のシナリオでは、プロセスが終了する前（たとえば、誰かが実行された場合）にBeeGFSサービスがメッセージをログに記録できなかった可能性があります `kill -9 <PID>`）。

手順**2**：ノードがクラスタから予期せず離れていないかを確認します

ノードで壊滅的なハードウェア障害が発生した場合（システム基板が故障した場合など）、またはカーネルパニックまたは同様のソフトウェア問題 が発生した場合、BeeGFSモニタはエラーを報告しません。代わりにホスト名を検索し、Pacemakerからのメッセージで、ノードが予期せずに失われたことを示すメッセージが表示されます。

```
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
```

手順3：**Pacemaker**がノードを遮断できたことを確認します

すべてのシナリオで、ノードが実際にオフラインであることを確認するためにペースメーカーがノードを遮断しようとすると考えられます（正確なメッセージはフェンシングの原因 によって異なる場合があります）。

```
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
```

フェンシング処理が正常に完了すると、次のようなメッセージが表示されます。

```
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
```

何らかの理由でフェンシングアクションが失敗した場合、データ破損のリスクを回避するために別のノードでBeeGFSサービスを再起動できなくなります。たとえば、フェンシングデバイス（PDUまたはBMC）にアクセスできなかったり、誤って設定されていた場合、問題 は個別に調査します。

**Address Failed Resource Actions**（**PC**ステータスの下部にある）

BeeGFSサービスの実行に必要なリソースに障害が発生すると、BeeGFSモニタによってフェイルオーバーがトリガーされます。この場合は、の下部に[Failed Resource Actions]が表示されない可能性があり `pcs status` ます。[How to]の手順を参照してください。"計画外フェイルオーバー後のフェイルバック"

そうしないと、通常、「Failed Resource Actions」というメッセージが表示されるシナリオは2つだけになります。

シナリオ**1**：フェンシングエージェントで一時的または永続的な問題 が検出され、再起動されたか、別のノードに移動された。

フェンシングエージェントの中には、他のエージェントよりも信頼性の高いものがあり、それぞれがフェンシングデバイスの準備が整ったことを確認する独自の監視方法を実装しています。特に、Redfishフェンシングエージェントは、次のような失敗したリソースアクションを報告するようになりましたが、まだ開始されています。

```
  * 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
```

特定のノードで失敗したリソースアクションを報告するフェンシングエージェントは、そのノードで実行されているBeeGFSサービスのフェールオーバーをトリガーする必要はありません。同じノードまたは別のノードで自動的に再起動されるだけです。

解決手順：

1. フェンシングエージェントが、すべてのノードまたは一部のノードでの実行を常に拒否している場合は、それらのノードがフェンシングエージェントに接続できるかどうかを確認し、フェンシングエージェントがAnsibleインベントリで正しく設定されていることを確認します。
   a. たとえば、Redfish（BMC）フェンシングエージェントがフェンシングを担当するノードで実行されており、OS管理とBMC IPが同じ物理インターフェイス上にある場合、一部のネットワークスイッチ構成では2つのインターフェイス間の通信が許可されないため（ネットワークループが回避されます）、デフォルトでは、HAクラスタは、フェンシングを担当するノードにフェンシングエージェントを配置しないようにしますが、これは一部のシナリオや構成で発生する可能性があります。
2. すべての問題が解決したら（または問題 が一時的なものと思われる場合）、を実行します `pcs resource cleanup` 失敗したリソースアクションをリセットします。

シナリオ**2**：**BeeGFS**モニタが問題 を検出してフェイルオーバーをトリガーしましたが、何らかの理由でセカンダリノードでリソースを起動できませんでした。

フェンシングが有効で、リソースが元のノードで停止しないようにブロックされていない場合（「standby」（on -ffail）のトラブルシューティングのセクションを参照）、最も可能性の高い理由は、次のような理由からセカンダリノードでリソースを起動する際の問題です。

- セカンダリノードはすでにオフラインでした。
- 物理構成または論理構成の問題 によって、セカンダリはBeeGFSターゲットとして使用されるブロックボリュームにアクセスできなくなりました。

解決手順：

1. 失敗したリソースアクションの各エントリについて、次の手順を実行します。
   a. 失敗したリソースアクションが開始操作であることを確認します。
   b. 指定したリソースと、失敗したリソースアクションで指定されたノードに基づきます。
      i. ノードが指定したリソースを起動できないような外部の問題がないかどうかを確認して解決しま

す。たとえば、BeeGFS IP address (floating IP) failed to startの場合は、必要なインターフェイスの少なくとも1つが接続/オンラインであり、適切なネットワークスイッチにケーブル接続されていることを確認します。BeeGFSターゲット（ブロックデバイス/Eシリーズボリューム）に障害が発生した場合は、バックエンドブロックノードへの物理接続が正常に接続されていることを確認し、ブロックノードが正常であることを確認します。

c. 明らかな外部の問題がなく、このインシデントに対するrootの原因 が必要な場合は、以下の手順を進める前にネットアップサポートの調査ケースをオープンして原因 分析（RCA）を実施することを検討することを推奨します。

2. 外部の問題を解決したあと：

a. Ansibleのinventory.ymlファイルから機能しないノードをコメント化し、完全なAnsibleプレイブックを再実行して、すべての論理構成がセカンダリノードで正しくセットアップされていることを確認します。

i. 注：ノードが正常でフェイルバックの準備ができたら、これらのノードのコメントを解除してプレイブックを再実行してください。

b. または、クラスタのリカバリを手動で実行することもできます。

i. 次のコマンドを使用して、オフラインのノードをオンラインに戻します。 `pcs cluster start <HOSTNAME>`

ii. 障害が発生したすべてのリソースアクションをクリアするには、 `pcs resource cleanup`

iii. PCステータスを実行し、すべてのサービスが期待どおりに開始されることを確認します。

iv. 必要に応じてを実行します `pcs resource relocate run` をクリックして、リソースを優先ノードに戻します（使用可能な場合）。

## 一般的な問題

**BeeGFS**サービスは、要求されたときにフェイルオーバーやフェイルバックを行いません

可能性の高い問題 ： `pcs resource relocate` 実行コマンドは実行されましたが、正常に終了しませんでした。

*確認方法:*実行 `pcs constraint --full` IDがの場所の制約がないかどうかを確認します `pcs-relocate-<RESOURCE>`。

*解決方法:*実行 `pcs resource relocate clear` 再実行します `pcs constraint --full` 追加の拘束が除去されたことを確認します。

フェンシングが無効な場合、**PC**ステータスの一方のノードに「**standby (on-fail)**」と表示されます

考えられる問題 ： Pacemakerは、障害が発生したノードですべてのリソースが停止していることを正常に確認できませんでした。

解決方法:

1. を実行します `pcs status` および出力の一番下に「started」または「エラーが表示されていないリソースがないかどうかを確認し、問題を解決します。

2. ノードをオンラインに戻すには、次の手順を実行します `pcs resource cleanup --node=<HOSTNAME>`。

想定外のフェイルオーバーが発生すると、フェンシングが有効になっている場合、**PC**のステータスに「**started**（**on-fail**）」と表示されます

*問題 の可能性：*フェールオーバーをトリガーしたが、Pacemakerがノードをフェンシングしていることを確認できなかった問題が発生しました。フェンシングが正しく設定されていないか、フェンシングエージェントを含む問題 が存在することが原因で発生します（例：PDUがネットワークから切断されています）。

解決方法:

1. ノードの電源がオフになっていることを確認します。

> 指定したノードが実際にはオフになっておらず、クラスタのサービスやリソースを実行している場合は、データの破損やクラスタ障害が発生します。

2. フェンシングを手動で確認する場合： `pcs stonith confirm <NODE>`

この時点で、サービスのフェイルオーバーが完了し、別の正常なノードで再開されます。

## 一般的なトラブルシューティングタスク

**BeeGFS**サービスを個別に再起動します

通常、BeeGFSサービスを再起動（設定変更を容易にするためなど）する必要がある場合は、Ansibleインベントリを更新してプレイブックを再実行します。一部のシナリオでは、個 々 のサービスを再起動して迅速なトラブルシューティングを実現したい場合があります。たとえば、プレイブック全体の実行を待たずにログレベルを変更する場合などです。

> Ansibleインベントリに手動で変更を追加しないかぎり、次回Ansibleプレイブックが実行されたときに変更が元に戻されます。

オプション**1**：**systemd**で制御された再起動

新しい設定でBeeGFSサービスが適切に再起動しないリスクがある場合は、まずクラスタをメンテナンスモードにして、BeeGFSモニタがサービスを停止して不要なフェイルオーバーをトリガーしないようにします。

```
pcs property set maintenance-mode=true
```

必要に応じて、でサービス設定を変更します `/mnt/<SERVICE_ID>/_config/beegfs-.conf` （例： `/mnt/meta_01_tgt_0101/metadata_config/beegfs-meta.conf`）次にsystemdを使用して再起動します。

```
systemctl restart beegfs-*@<SERVICE_ID>.service
```

例 `systemctl restart beegfs-meta@meta_01_tgt_0101.service`

オプション**2**：ペースメーカーの再起動を制御

新しい設定で原因 サービスが予期せず停止する（ロギングレベルの変更など）か、メンテナンス時間になっ

ていてダウンタイムが気にならない場合は、再起動するサービスのBeeGFSモニタを再起動するだけです。

```
pcs resource restart <SERVICE>-monitor
```

たとえば、BeeGFS管理サービスを再起動するには、次の手順を実行します。 `pcs resource restart mgmt-monitor`

# 法的通知

著作権に関する声明、商標、特許などにアクセスできます。

## 著作権

"https://www.netapp.com/company/legal/copyright/"

## 商標

NetApp 、 NetApp のロゴ、および NetApp の商標ページに記載されているマークは、 NetApp, Inc. の商標です。その他の会社名および製品名は、それぞれの所有者の商標である場合があります。

"https://www.netapp.com/company/legal/trademarks/"

## 特許

ネットアップが所有する特許の最新リストは、次のサイトで入手できます。

https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf

## プライバシーポリシー

"https://www.netapp.com/company/legal/privacy-policy/"

## オープンソース

通知ファイルには、ネットアップソフトウェアで使用されるサードパーティの著作権およびライセンスに関する情報が記載されています。

"E シリーズ / EF シリーズ SANtricity OS に関する通知です"

## 著作権に関する情報



## 商標に関する情報