



Red Hat

Red Hat Enterprise Linux 8

ストレージデバイスの管理

ローカルおよびリモートのストレージデバイスの設定と管理

Red Hat Enterprise Linux 8 ストレージデバイスの管理

ローカルおよびリモートのストレージデバイスの設定と管理

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Red Hat Enterprise Linux (RHEL) は、ローカルおよびリモートのストレージオプションをいくつか提供します。利用可能なストレージオプションを使用すると、次のタスクを実行できます。要件に従ってディスクパーティションを作成します。ディスク暗号化を使用して、ブロックデバイス上のデータを保護します。RAID (Redundant Array of Independent Disks) を作成して、複数のドライブにデータを保存し、データ損失を回避します。iSCSI および NVMe over Fabrics を使用して、ネットワーク経由でストレージにアクセスします。

Table of Contents

RED HAT ドキュメントへのフィードバック (英語のみ)	7
第1章 利用可能なストレージオプションの概要	8
1.1. ローカルストレージの概要	8
1.2. リモートストレージの概要	9
1.3. GFS2 ファイルシステムの概要	10
第2章 ディスクパーティション	12
2.1. パーティションの概要	12
2.2. パーティションテーブルの種類の比較	12
2.3. MBR ディスクパーティション	13
2.4. 拡張 MBR パーティション	14
2.5. MBR パーティションタイプ	14
2.6. GUID パーティションテーブル	15
2.7. パーティションタイプ	17
2.8. パーティション命名スキーム	18
2.9. マウントポイントとディスクパーティション	19
第3章 パーティションの使用	20
3.1. PARTED でディスクにパーティションテーブルを作成	20
3.2. PARTED でパーティションテーブルの表示	21
3.3. PARTED を使用したパーティションの作成	22
3.4. FDISK でパーティションタイプの設定	24
3.5. PARTED でパーティションのサイズ変更	25
3.6. PARTED でパーティションの削除	26
第4章 ディスクを再設定するストラテジー	28
4.1. パーティションが分割されていない空き領域の使用	28
4.2. 未使用パーティションの領域の使用	28
4.3. アクティブなパーティションの空き領域の使用	29
第5章 永続的な命名属性の概要	33
5.1. 非永続的な命名属性のデメリット	33
5.2. ファイルシステムおよびデバイスの識別子	34
5.3. /DEV/DISK/ にある UDEV メカニズムにより管理されるデバイス名	34
5.4. DM MULTIPATH を使用した WORLD WIDE IDENTIFIER	36
5.5. UDEV デバイス命名規則の制約	37
5.6. 永続的な命名属性のリスト表示	38
5.7. 永続的な命名属性の変更	39
第6章 NVDIMM 永続メモリーストレージの使用	41
6.1. NVDIMM 永続メモリーテクノロジー	41
6.2. NVDIMM のインターリービングおよび地域	41
6.3. NVDIMM 名前空間	42
6.4. NVDIMM アクセスマード	42
6.5. NDCTL のインストール	43
6.6. ブロックデバイスとして動作する NVDIMM 上のセクターネーム空間の作成	43
6.7. NVDIMM でのデバイス DAX 名前空間の作成	47
6.8. NVDIMM でのファイルシステム DAX 名前空間の作成	52
6.9. S.M.A.R.T. を使用した NVDIMM 正常性 (ヘルス) の監視	58
6.10. 破損した NVDIMM デバイスの検出と交換	59
第7章 未使用ブロックの破棄	63

要件	63
7.1. ブロック破棄操作のタイプ	63
7.2. バッチブロック破棄の実行	63
7.3. オンラインブロック破棄の有効化	64
7.4. STORAGE RHEL システムロールを使用してオンラインブロック破棄を有効にする	64
7.5. 定期的なブロック破棄の有効化	66
第8章 iSCSI ターゲットの設定	67
8.1. TARGETCLI のインストール	67
8.2. iSCSI ターゲットの作成	68
8.3. iSCSI バックストア	69
8.4. FILEIO ストレージオブジェクトの作成	69
8.5. ブロックストレージオブジェクトの作成	70
8.6. PSCSI ストレージオブジェクトの作成	71
8.7. メモリーコピーの RAM ディスクストレージオブジェクトの作成	71
8.8. iSCSI ポータルの作成	72
8.9. iSCSI LUN の作成	73
8.10. 読み取り専用の iSCSI LUN の作成	74
8.11. iSCSI ACL の作成	75
8.12. ターゲットのチャレンジハンドシェイク認証プロトコルの設定	77
8.13. TARGETCLI ツールで iSCSI オブジェクトの削除	77
第9章 iSCSI イニシエーターの設定	79
9.1. iSCSI イニシエーターの作成	79
9.2. イニシエーター用のチャレンジハンドシェイク認証プロトコルの設定	80
9.3. ISCSIADM ユーティリティーを使用して iSCSI セッションを監視する	81
9.4. DM MULTIPATH によるデバイスのタイムアウトのオーバーライド	82
第10章 ファイバーチャネルデバイスの使用	83
10.1. LUN のサイズ変更後にファイバーチャネル論理ユニットを再スキャンする	83
10.2. ファイバーチャネルを使用したデバイスのリンク切れ動作の特定	83
10.3. ファイバーチャネル設定ファイル	84
第11章 FIBRE CHANNEL OVER ETHERNET の設定	86
11.1. RHEL でハードウェア FCOE HBA の使用	86
11.2. FCOE デバイスの設定	86
第12章 EH_DEADLINE を使用したストレージエラーからの回復における最大時間の設定	89
12.1. EH_DEADLINE パラメーター	89
12.2. EH_DEADLINE パラメーターの設定	90
第13章 スワップの使用	91
13.1. スワップ領域の概要	91
13.2. システムの推奨スワップ領域	91
13.3. スワップ用の LVM2 論理ボリュームの作成	92
13.4. スワップファイルの作成	93
13.5. STORAGE RHEL システムロールを使用してスワップボリュームを作成する	94
13.6. LVM2 論理ボリュームでのスワップ領域の拡張	95
13.7. LVM2 論理ボリュームでのスワップ領域の縮小	96
13.8. スワップ用の LVM2 論理ボリュームの削除	96
13.9. スワップファイルの削除	97
第14章 NVME OVER FABRIC デバイスの概要	98
第15章 NVME/RDMA を使用した NVME OVER FABRICS の設定	99

15.1. CONFIGFS を使用した NVME/RDMA コントローラーのセットアップ	99
15.2. NVMETCLI を使用した NVME/RDMA コントローラーのセットアップ	100
15.3. NVME/RDMA ホストの設定	101
15.4. 次のステップ	103
第16章 NVME/FC を使用した NVME OVER FABRICS の設定	104
16.1. BROADCOM アダプターの NVME ホストの設定	104
16.2. QLOGIC アダプターの NVME ホストの設定	106
16.3. 次のステップ	107
第17章 NVME デバイスでのマルチパスの有効化	108
17.1. ネイティブ NVME マルチパスと DM MULTIPATH	108
17.2. ネイティブ NVME マルチパスの実現	108
17.3. NVME デバイスでの DM MULTIPATH の有効化	110
第18章 ディスクスケジューラーの設定	113
18.1. 利用可能なディスクスケジューラー	113
18.2. 各種ユースケースで異なるディスクスケジューラー	114
18.3. デフォルトのディスクスケジューラー	114
18.4. アクティブなディスクスケジューラーの決定	114
18.5. TUNED を使用したディスクスケジューラーの設定	115
18.6. UDEV ルールを使用したディスクスケジューラーの設定	117
18.7. 特定ディスクに任意のスケジューラーを一時的に設定	118
第19章 リモートディスクレスシステムの設定	119
19.1. リモートディスクレスシステムの環境の準備	119
19.2. ディスクレスクライアントの TFTP サービスの設定	120
19.3. ディスクレスクライアントの DHCP サーバーの設定	121
19.4. ディスクレスクライアントのエクスポートしたファイルシステムの設定	122
19.5. リモートディスクレスシステムの再設定	124
19.6. リモートディスクレスシステムのロードに関する一般的な問題のトラブルシューティング	125
第20章 RAID の管理	127
20.1. RAID の概要	127
20.2. RAID のタイプ	127
20.3. RAID レベルとリニアサポート	129
20.4. サポート対象の RAID 変換	130
20.5. RAID サブシステム	133
20.6. インストール中のソフトウェア RAID の作成	133
20.7. インストール済みシステムでのソフトウェア RAID の作成	134
20.8. WEB コンソールで RAID の作成	135
20.9. WEB コンソールで RAID のフォーマット	136
20.10. WEB コンソールを使用した RAID 上のパーティションテーブルの作成	138
20.11. WEB コンソールを使用した RAID 上のパーティションの作成	138
20.12. WEB コンソールを使用した RAID 上のボリュームグループの作成	140
20.13. STORAGE RHEL システムロールを使用した RAID ボリュームの設定	140
20.14. RAID の拡張	142
20.15. RAID を縮小	142
20.16. インストール後にルートディスクを RAID1 に変換する	143
20.17. 高度な RAID デバイスの作成	144
20.18. RAID を監視するための電子メール通知の設定	144
20.19. RAID での障害のあるディスクの置き換え	145
20.20. RAID ディスクの修復	147
第21章 LUKS を使用したブロックデバイスの暗号化	149

21.1. LUKS ディスクの暗号化	149
21.2. RHEL の LUKS バージョン	150
21.3. LUKS2 再暗号化中のデータ保護のオプション	151
21.4. LUKS2 を使用したブロックデバイスの既存データの暗号化	152
21.5. 独立したヘッダーがある LUKS2 を使用してブロックデバイスの既存データの暗号化	154
21.6. LUKS2 を使用した空のブロックデバイスの暗号化	157
21.7. WEB コンソールでの LUKS パスフレーズの設定	158
21.8. WEB コンソールで LUKS パスフレーズの変更	159
21.9. コマンドラインを使用した LUKS パスフレーズの変更	160
21.10. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する	161
第22章 テープデバイスの管理	164
22.1. テープデバイスの種類	164
22.2. テープドライブ管理ツールのインストール	164
22.3. テープコマンド	164
22.4. 巻き戻しテープデバイスへの書き込み	165
22.5. 巻き戻しなしのテープデバイスへの書き込み	166
22.6. テープデバイスでのテープヘッドの切り替え	168
22.7. テープデバイスからのデータの復元	168
22.8. テープデバイスのデータの消去	169
第23章 ストレージデバイスの削除	170
23.1. ストレージデバイスの安全な削除	170
23.2. ブロックデバイスと関連メタデータの削除	170
第24章 STRATIS ファイルシステムの設定	174
24.1. STRATIS ファイルシステムのコンポーネント	174
24.2. STRATIS と互換性のあるブロックデバイス	175
24.3. STRATIS のインストール	175
24.4. 暗号化されていない STRATIS プールの作成	176
24.5. WEB コンソールを使用した暗号化されていない STRATIS プールの作成	177
24.6. カーネルキーリング内のキーを使用して暗号化された STRATIS プールを作成する	178
24.7. WEB コンソールを使用した暗号化された STRATIS プールの作成	179
24.8. WEB コンソールを使用した STRATIS プールの名前変更	181
24.9. STRATIS ファイルシステムでのオーバープロビジョニングモードの設定	181
24.10. STRATIS プールの NBDE へのバインド	182
24.11. STRATIS プールの TPM へのバインド	183
24.12. カーネルキーリングを使用した暗号化 STRATIS プールのロック解除	184
24.13. 補助暗号化からの STRATIS プールのバインド解除	184
24.14. STRATIS プールの開始および停止	185
24.15. STRATIS ファイルシステムの作成	186
24.16. WEB コンソールを使用した STRATIS プール上のファイルシステムの作成	187
24.17. STRATIS ファイルシステムのマウント	187
24.18. SYSTEMD サービスを使用した /ETC/FSTAB での非ルート STRATIS ファイルシステムの設定	188
第25章 追加のブロックデバイスで STRATIS プールを拡張する	190
25.1. STRATIS プールへのブロックデバイスの追加	190
25.2. WEB コンソールを使用した STRATIS プールへのブロックデバイスの追加	190
第26章 STRATIS ファイルシステムの監視	192
26.1. STRATIS ファイルシステムに関する情報の表示	192
26.2. WEB コンソールを使用した STRATIS プールの表示	193
第27章 STRATIS ファイルシステムでのスナップショットの使用	194
27.1. STRATIS スナップショットの特徴	194

27.2. STRATIS スナップショットの作成	194
27.3. STRATIS スナップショットのコンテンツへのアクセス	195
27.4. STRATIS ファイルシステムを以前のスナップショットに戻す	195
27.5. STRATIS スナップショットの削除	196
第28章 STRATIS ファイルシステムの削除	197
28.1. STRATIS ファイルシステムの削除	197
28.2. WEB コンソールを使用した STRATIS プールからのファイルシステムの削除	197
28.3. STRATIS プールの削除	198
28.4. WEB コンソールを使用した STRATIS プールの削除	199

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. [Jira](#) の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

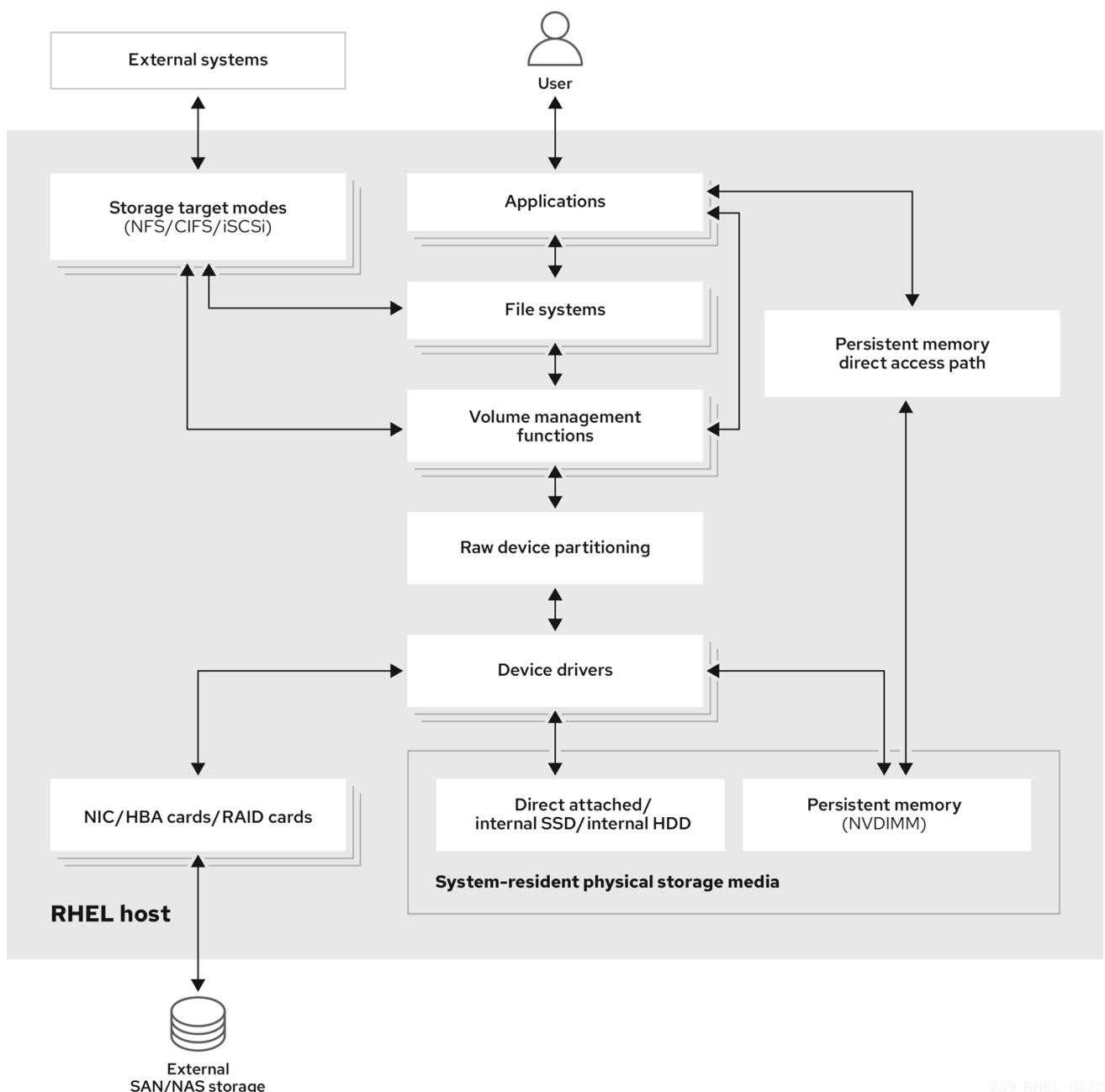
第1章 利用可能なストレージオプションの概要

RHEL 8 では、複数のローカル、リモート、およびクラスターベースのストレージオプションを利用できます。

ローカルストレージは、ストレージデバイスがシステムにインストールされているか、システムに直接接続されていることを意味します。

リモートストレージでは、LAN、インターネット、またはファイバーチャネルネットワークを介してデバイスにアクセスします。以下の Red Hat Enterprise Linux ストレージダイアグラムの概要では、さまざまなストレージオプションを説明します。

図1.1 Red Hat Enterprise Linux ストレージダイアグラム (概要)



269_RHEL_0822

1.1. ローカルストレージの概要

Red Hat Enterprise Linux 8 は、ローカルストレージオプションを複数提供します。

基本的なディスク管理:

parted と **fdisk** を使用して、ディスクパーティションの作成、変更、削除、および表示を行うことができます。パーティショニングレイアウトの標準は以下のようになります。

マスター・ブート・レコード (MBR)

BIOS ベースのコンピューターで使用されます。プライマリーパーティション、拡張パーティション、および論理パーティションを作成できます。

GUID パーティションテーブル (GPT)

Globally Unique Identifier (GUID) を使用し、一意のディスクおよびパーティション GUID を提供します。

ストレージ使用オプション

NVDIMM (Non-Volatile Dual In-line Memory Modules) の管理

メモリーとストレージの組み合わせです。システムに接続した NVDIMM デバイスで、さまざまな種類のストレージを有効にして管理できます。

ロックストレージ管理

各ブロックに固有の識別子を持つブロックの形式でデータを保存します。

ファイルストレージ

データは、ローカルシステムのファイルレベルに保存されます。これらのデータは、XFS (デフォルト) または ext4 を使用してローカルでアクセスしたり、NFS と SMB を使用してネットワーク上でアクセスできます。

論理ボリューム

論理ボリュームマネージャー (LVM)

物理デバイスから論理デバイスを作成します。論理ボリューム (LV) は、物理ボリューム (PV) とボリュームグループ (VG) の組み合わせです。

VDO (Virtual Data Optimizer)

重複排除、圧縮、およびシンプロビジョニングを使用して、データの削減に使用されます。VDO の下で論理ボリュームを使用すると、次のことができます。

- VDO ボリュームの拡張
- 複数のデバイスにまたがる VDO ボリューム

ローカルファイルシステム

XFS

デフォルトの RHEL ファイルシステム。

Ext4

レガシーファイルシステム。

Stratis

テクノロジープレビューとして利用可能になりました。Stratis は、高度なストレージ機能に対応する、ユーザーとカーネルのハイブリッドローカルストレージ管理システムです。

1.2. リモートストレージの概要

RHEL 8 で利用可能なリモートストレージオプションは次のとおりです。

ストレージの接続オプション

iSCSI

RHEL 8 は targetcli ツールを使用して、iSCSI ストレージの相互接続を追加、削除、表示、および監視します。

ファイバーチャネル (FC)

RHEL 8 は、以下のネイティブファイバーチャネルドライバーを提供します。

- **lpfc**
- **qla2xxx**
- **Zfcp**

NVMe (Non-volatile Memory Express)

ホストソフトウェアユーティリティーがソリッドステートドライブと通信できるようにするインターフェイス。NVMe over Fabrics を設定するには、次のタイプの fabric トランスポートを使用します。

- Remote Direct Memory Access (NVMe/RDMA) を使用する NVMe over Fabrics
- ファイバーチャネルを使用した NVMe over Fabrics (NVMe/FC)

Device Mapper Multipath (DM Multipath)

サーバーノードとストレージアレイ間の複数の I/O パスを 1 つのデバイスに設定できます。これらの I/O パスは、個別のケーブル、スイッチ、コントローラーを含むことができる物理的な SAN 接続です。

ネットワークファイルシステム

- NFS
- SMB

1.3. GFS2 ファイルシステムの概要

Red Hat Global File System 2 (GFS2) ファイルシステムは、64 ビットの対称クラスターファイルシステムで、共有名前空間を提供し、一般的なブロックデバイスを共有する複数のノード間の一貫性を管理します。GFS2 ファイルシステムは、ローカルファイルシステムに可能な限り近い機能セットを提供すると同時に、ノード間でクラスターの完全な整合性を強制することを目的としています。これを実現するため、ノードはファイルシステムリソースにクラスター全体のロックスキームを使用します。このロックスキームは、TCP/IP などの通信プロトコルを使用して、ロック情報を交換します。

場合によっては、Linux ファイルシステム API では、GFS2 のクラスター化された性質を完全に透過的にすることできません。たとえば、GFS2 で POSIX ロックを使用しているプログラムは、**GETLK** の使用を回避する必要があります。なぜなら、クラスター環境では、プロセス ID が、クラスター内の別のノードに対するものである可能性があるためです。ただし、ほとんどの場合、GFS2 ファイルシステムの機能は、ローカルファイルシステムのものと同じです。

Red Hat Enterprise Linux Resilient Storage Add-On は GFS2 を提供します。GFS2 が必要とするクラスター管理の提供は Red Hat Enterprise Linux High Availability Add-On により提供されます。

gfs2.ko カーネルモジュールは GFS2 ファイルシステムを実装し、GFS2 クラスターノードに読み込まれます。

GFS2 環境を最大限に利用するためにも、基礎となる設計に起因するパフォーマンス事情を考慮することが重要です。GFS2 では、ローカルファイルシステムと同様、ページキャッシングで、頻繁に使用されるデータのローカルキャッシングを行ってパフォーマンスを向上します。クラスターのノード間で一貫性を維持するために、`glock` ステートマシンでキャッシング制御が提供されます。

関連情報

- [GFS2 ファイルシステムの設定](#)

第2章 ディスクパーティション

ディスクを1つ以上の論理領域に分割するには、ディスクのパーティション設定ユーティリティーを使用します。これにより、各パーティションを個別に管理できます。

2.1. パーティションの概要

ハードディスクは、パーティションテーブルの各ディスクパーティションの場所とサイズに関する情報を保存します。オペレーティングシステムは、パーティションテーブルの情報を使用して、各パーティションを論理ディスクとして扱います。ディスクパーティション設定には、次のような利点があります。

- 物理ボリュームの管理上の見落としの可能性を減らす。
- 十分なバックアップを確保する。
- 効率的なディスク管理を提供する。

関連情報

- [直接または LVM を間に入れて、LUN でパーティション設定を使用するメリットとデメリットは何ですか？\(Red Hat ナレッジベース\)](#)

2.2. パーティションテーブルの種類の比較

デバイスでパーティションを有効にするには、さまざまな種類のパーティションテーブルでブロックデバイスをフォーマットします。次の表では、ブロックデバイスで作成できるさまざまな種類のパーティションテーブルのプロパティーを比較しています。



注記

このセクションでは、IBM Z アーキテクチャーに固有の DASD パーティションテーブルを説明しません。

表2.1パーティションテーブルの種類

パーティションテーブル	パーティションの最大数	パーティションの最大サイズ
マスターブートレコード (MBR)	4つのプライマリーパーティション、または3つのプライマリーパーティションと12の論理パーティションを持つ1つの拡張パーティション	512 b セクターのドライブを使用する場合は 2 TiB 4k セクターのドライブを使用する場合は 16 TiB
GUID パーティションテーブル (GPT)	128	512 b セクターのドライブを使用する場合は 8 ZiB 4k セクターのドライブを使用する場合は 64 ZiB

関連情報

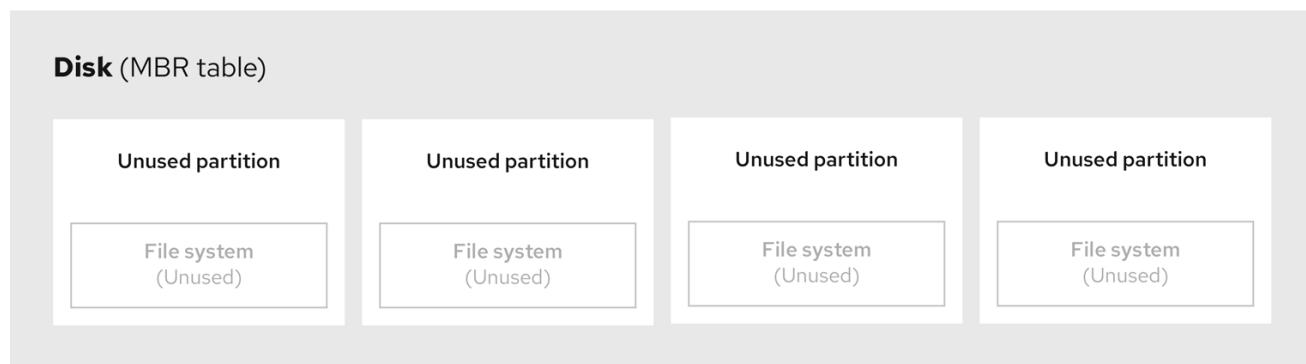
- [IBM ZへのLinuxインスタンスの設定](#)

- [What you should know about DASD](#)

2.3. MBR ディスクパーティション

パーティションテーブルはそのディスクの先頭部分となる、他のファイルシステムまたはユーザーデータの前に格納されています。わかりやすくするために、次の図ではパーティションテーブルを区切って示しています。

図2.1 MBR パーティションテーブルがあるディスク



269_RHEL_0822

上記の図で示したとおり、パーティションテーブルは使用していない4つのプライマリーパーティションの4つのセクションに分けられます。プライマリーパーティションは、論理ディスクドライブ（またはセクション）を1つだけ含むハードドライブのパーティションです。各論理ドライブは、1つのパーティションの定義に必要な情報を保持できます。つまり、パーティションテーブルで定義できるプライマリーパーティションは4つまでです。

各パーティションテーブルエントリーには、パーティションの重要な特徴が含まれています。

- ディスク上のパーティションの開始点と終了点
- パーティションの状態（アクティブとしてフラグを立てることができるのは1つのパーティションのみ）
- パーティションのタイプ

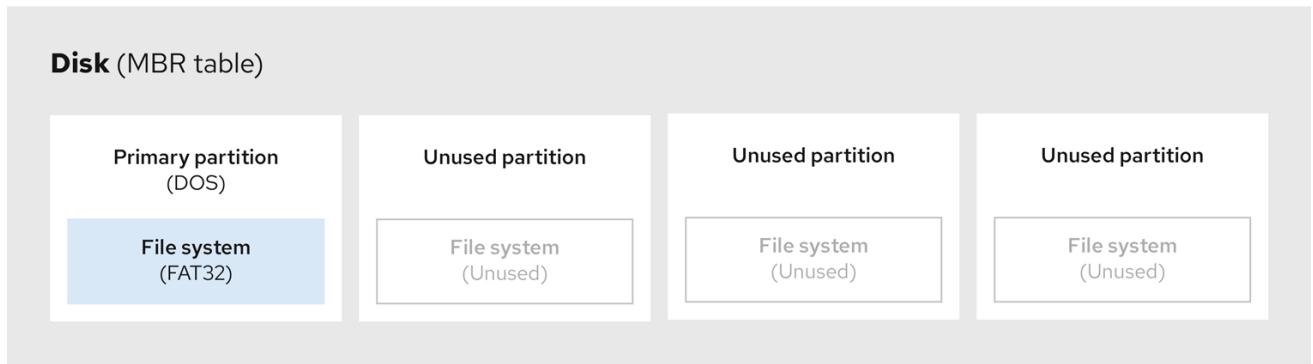
開始点と終了点は、ディスク上のパーティションのサイズと場所を定義します。一部のオペレーティングシステムブートローダーは、**active** フラグを使用します。つまり、"active" とマークされているパーティションのオペレーティングシステムが起動します。

タイプとは、パーティションの用途を識別する番号です。一部のオペレーティングシステムでは、パーティションの種類を使用して以下を行います。

- 特定のファイルシステムタイプを示します。
- 特定のオペレーティングシステムに関連付けられているパーティションにフラグを付けます。
- パーティションに起動可能なオペレーティングシステムが含まれていることを示します。

以下の図は、パーティションが1つあるドライブの例を示しています。この例では、最初のパーティションには **DOS** パーティションタイプのラベルが付けられています。

図2.2 1つのパーティションを持つディスク



269_RHEL_0822

関連情報

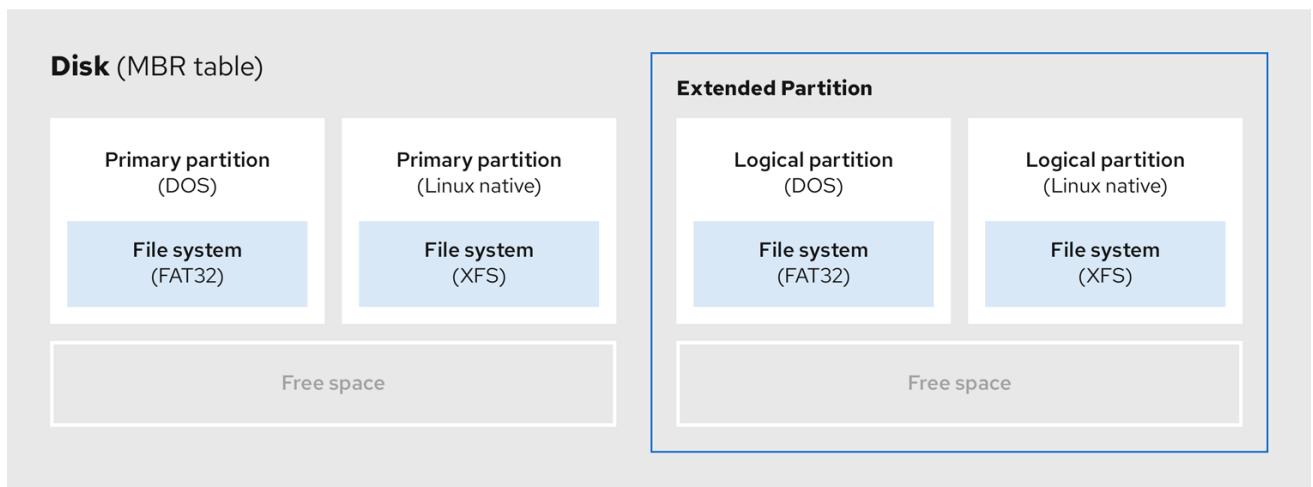
- [MBR パーティションタイプ](#)

2.4. 拡張 MBR パーティション

必要があれば、タイプを **extended** に設定して追加のパーティションを作成します。

拡張パーティションは、ディスクドライブに似ています。拡張パーティション内に完全に含まれる1つ以上の論理パーティションを指す独自のパーティションテーブルがあります。次の図は、2つのプライマリパーティションと、2つの論理パーティションを含む1つの拡張パーティションおよびいくつかの未パーティションの空き領域を備えたディスクドライブを示しています。

図2.3 2つのプライマリパーティションと拡張 MBR パーティションの両方を備えたディスク



269_RHEL_0822

最大4つのプライマリパーティションと拡張パーティションのみを使用できますが、論理パーティションの数に制限はありません。Linuxではパーティションへのアクセスに制限があり、1つのディスクドライブでは最大15個のパーティションが許可されます。

2.5. MBR パーティションタイプ¹

次の表は、最も一般的に使用される MBR パーティションタイプとそれらを表す16進数のリストです。

表2.2 MBRパーティションタイプ

MBRパーティションタイプ	値	MBRパーティションタイプ	値
Empty	00	Novell Netware 386	65
DOS 12 ビット FAT	01	PIC/IX	75
XENIX root	02	旧 MINIX	80
XENIX usr	03	Linux/MINUX	81
DOS 16 ビット (32M 以下)	04	Linux swap	82
Extended	05	Linux ネイティブ	83
DOS 16 ビット (32 以上)	06	Linux 拡張	85
OS/2 HPFS	07	Amoeba	93
AIX	08	Amoeba BBT	94
AIX ブート可能	09	BSD/386	a5
OS/2 Boot Manager	0a	OpenBSD	a6
Win95 FAT32	0b	NEXTSTEP	a7
Win95 FAT32 (LBA)	0c	BSDI fs	b7
Win95 FAT16 (LBA)	0e	BSDI swap	b8
Win95 Extended (LBA)	0f	Syrinx	c7
Venix 80286	40	CP/M	db
Novell	51	DOS アクセス	e1
PRep Boot	41	DOS R/O	e3
GNU HURD	63	DOS セカンダリー	f2
Novell Netware 286	64	BBT	ff

2.6. GUID パーティションテーブル

GUID パーティションテーブル (GPT) は、Globally Unique Identifier (GUID) に基づくパーティション設定スキームです。

GPT は、Master Boot Record (MBR) パーティションテーブルの制限に対処します。MBR パーティションテーブルは、約 2.2 TB に相当する 2 TiB を超えるストレージに対応できません。代わりに、GPT は大容量のハードディスクをサポートします。アドレス指定可能な最大ディスクサイズは、512b セクタードライブを使用する場合は 8 ZiB、4096b セクタードライブを使用する場合は 64 ZiB です。さらに、デフォルトで、GPT は最大 128 のプライマリーパーティションの作成をサポートします。パーティションテーブルにより多くの領域を割り当てて、プライマリーパーティションの最大量を拡張します。



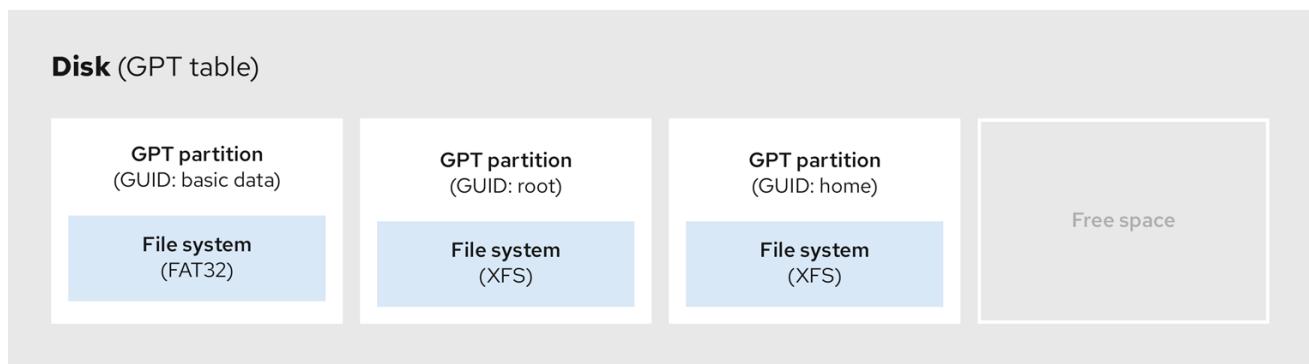
注記

GPT には GUID に基づくパーティションタイプがあります。特定のパーティションには特定の GUID が必要です。たとえば、Extensible Firmware Interface (EFI) ブートローダーのシステムパーティションには、GUID **C12A7328-F81F-11D2-BA4B-00A0C93EC93B** が必要です。

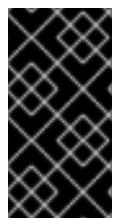
GPT ディスクは、論理ブロックアドレス指定 (LBA) とパーティションレイアウトを以下のように使用します。

- MBR ディスクとの下位互換性のために、システムは MBR データ用に GPT の最初のセクター (LBA 0) を予約し、"protective MBR" という名前を適用します。
- プライマリー GPT
 - ヘッダーは、デバイスの 2 番目の論理ブロック (LBA 1) から始まります。ヘッダーには、ディスク GUID、プライマリーパーティションテーブルの場所、セカンダリ GPT ヘッダーの場所、および CRC32 チェックサム、およびプライマリーパーティションテーブルが含まれます。また、テーブルにあるパーティションエントリーの数も指定します。
 - デフォルトでは、プライマリー GPT には 128 のパーティションエントリーが含まれます。各パーティションには、128 バイトのエントリーサイズ、パーティションタイプ GUID、一意のパーティション GUID があります。
- セカンダリ GPT
 - リカバリーの場合は、プライマリーパーティションテーブルが破損した場合にバックアップテーブルとして役立ちます。
 - ディスクの最後の論理セクターにはセカンダリ GPT ヘッダーが含まれており、プライマリーヘッダーが破損した場合に備えて GPT 情報を回復します。
 - 以下が含まれます。
 - ディスク GUID
 - セカンダリーパーティションテーブルとプライマリー GPT ヘッダーの場所
 - それ自体の CRC32 チェックサム
 - セカンダリーパーティションテーブル
 - 可能なパーティションエントリーの数

図2.4 GUIDパーティションテーブルを含むディスク



269_RHEL_0822



重要

GPTディスクにブートローダーを正常にインストールするには、BIOSブートパーティションが存在する必要があります。ディスクにすでにBIOSブートパーティションが含まれている場合にのみ、再利用が可能です。これには、Anacondaインストールプログラムによって初期化されたディスクが含まれます。

2.7. パーティションタイプ

パーティションタイプを管理する方法は複数あります。

- **fdisk** ユーティリティーは、16進数コードを指定することで、あらゆる種類のパーティションタイプに対応します。
- **systemd-gpt-auto-generator** はユニットジェネレーターユーティリティーで、パーティションタイプを使用してデバイスを自動的に識別し、マウントします。
- **parted** ユーティリティーは、フラグを使用してパーティションタイプをマップします。**parted** ユーティリティーは、LVM、swap、RAIDなど、特定のパーティションタイプのみを処理します。

parted ユーティリティーは、次のフラグの設定をサポートしています。

- **boot**
- **root**
- **swap**
- **hidden**
- **raid**
- **lvm**
- **lba**
- **legacy_boot**
- **irst**

- **esp**
- **palo**

parted ユーティリティーは、パーティションを作成するときにオプションでファイルシステムタイプ引数を受け付けます。必要な条件のリストについては、[parted を使用したパーティションの作成](#) を参照してください。値を使用して以下を行います。

- MBR にパーティションフラグを設定します。
- GPT にパーティションの UUID タイプを設定します。たとえば、ファイルシステムタイプの **swap**、**fat**、または **hfs** には、異なる GUID が設定されます。デフォルト値は Linux Data GUID です。

この引数では、パーティションのファイルシステムは変更されません。サポート対象フラグと GUID のみ区別します。

次のファイルシステムのタイプがサポートされています。

- **xfs**
- **ext2**
- **ext3**
- **ext4**
- **fat16**
- **fat32**
- **hfs**
- **hfs+**
- **linux-swap**
- **ntfs**
- **reiserfs**



注記

RHEL 8 で対応しているローカルファイルシステムは、**ext4** および **xfs** のみです。

2.8. パーティション命名スキーム

Red Hat Enterprise Linux は、**/dev/xxYN** 形式のファイル名を持つファイルベースの命名スキームを使用します。

デバイスおよびパーティション名は、以下の構造で構成されています。

/dev/

すべてのデバイスファイルが含まれるディレクトリーの名前。ハードディスクにはパーティションが含まれるため、すべてのパーティションを表すファイルは **/dev** にあります。

xx

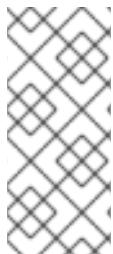
パーティション名の最初の 2 文字は、パーティションを含むデバイスのタイプを示します。

y

この文字は、パーティションを含む特定のデバイスを示します。たとえば、**/dev/sda** は最初のハードディスク、**/dev/sdb** は 2 番目のハードディスクです。ドライブの数が 26 を超えるシステムでは、さらに多くの文字を使用できます (例: **/dev/sdaa1**)。

N

最後の文字は、パーティションを表す数字を示します。最初の 4 つのパーティション (プライマリーまたは拡張) のパーティションには、1 から 4 までの番号が付けられます。論理パーティションは 5 から始まります。たとえば、**/dev/sda3** は 1 番目のハードディスクの 3 番目のプライマリーパーティションまたは拡張パーティションで、2 番目のハードディスク上の 2 番目の論理パーティション **/dev/sdb6** です。ドライブのパーティション番号は、MBR パーティションテーブルにのみ適用されます。N は常にパーティションを意味するものではないことに注意してください。

**注記**

Red Hat Enterprise Linux がすべてのタイプのディスクパーティションを識別して参照できる場合でも、ファイルシステムを読み取れないため、すべてのパーティショントイプに保存されているデータにアクセスできます。ただし、多くの場合、別のオペレーティングシステム専用のパーティション上にあるデータには問題なくアクセスすることができます。

2.9. マウントポイントとディスクパーティション

Red Hat Enterprise Linux では、各パーティションは、ファイルおよびディレクトリーの単一セットをサポートするのに必要なストレージの一部を形成します。パーティションをマウントすると、指定されたディレクトリー (マウントポイントと呼ばれる) を開始点としてそのパーティションのストレージが利用可能になります。

たとえば、パーティション **/dev/sda5** が **/usr/** にマウントされている場合、**/usr/** 下にあるすべてのファイルとディレクトリーは物理的に **/dev/sda5** 上に存在することになります。ファイル **/usr/share/doc/FAQ/txt/Linux-FAQ** は **/dev/sda5** にありますが、ファイル **/etc/gdm/custom.conf** はありません。

また、この例では、**/usr/** 以下の 1 つ以上のディレクトリーが他のパーティションのマウントポイントになる可能性もあります。たとえば、**/usr/local** にマウントされた **/dev/sda7** パーティションが含まれる場合、**/usr/local/man/whatis** は **/dev/sda5** ではなく **/dev/sda7** にあります。

第3章 パーティションの使用

ディスクパーティション設定を使用して、ディスクを1つ以上の論理領域に分割し、各パーティションで個別に作業できるようにします。ハードディスクは、パーティションテーブルの各ディスクパーティションの場所とサイズに関する情報を保存します。このテーブルを使用すると、各パーティションはオペレーティングシステムへの論理ディスクとして表示されます。その後、それらの個々のディスクで読み取りと書き込みを行うことができます。

ブロックデバイスでパーティションを使用する利点と欠点の概要は、Red Hat ナレッジベースソリューション [What are the advantages and disadvantages to using partitioning on LUNs, either directly or with LVM in between?](#) を参照してください。

3.1. PARTED でディスクにパーティションテーブルを作成

parted ユーティリティーを使用して、より簡単にパーティションテーブルでブロックデバイスをフォーマットできます。



警告

パーティションテーブルを使用してブロックデバイスをフォーマットすると、そのデバイスに保存されているすべてのデータが削除されます。

手順

1. インタラクティブな **parted** シェルを起動します。

```
# parted block-device
```

2. デバイスにパーティションテーブルがあるかどうかを確認します。

```
(parted) print
```

デバイスにパーティションが含まれている場合は、次の手順でパーティションを削除します。

3. 新しいパーティションテーブルを作成します。

```
(parted) mklabel table-type
```

- **table-type** を、使用するパーティションテーブルのタイプに置き換えます。
 - **msdos** (MBR の場合)
 - **gpt** (GPT の場合)

例3.1 GUID パーティションテーブル (GPT) テーブルの作成

ディスクに GPT テーブルを作成するには、次のコマンドを使用します。

```
(parted) mklabel gpt
```

このコマンドを入力すると、変更の適用が開始されます。

- パーティションテーブルを表示して、作成されたことを確認します。

(parted) print

- parted シェルを終了します。

(parted) quit

関連情報

- システム上の **parted(8)** man ページ

3.2. PARTED でパーティションテーブルの表示

ブロックデバイスのパーティションテーブルを表示して、パーティションレイアウトと個々のパーティションの詳細を確認します。parted ユーティリティーを使用して、ブロックデバイスのパーティションテーブルを表示できます。

手順

- parted ユーティリティーを起動します。たとえば、次の出力は、デバイス /dev/sda をリストします。

parted /dev/sda

- パーティションテーブルを表示します。

(parted) print

```
Model: ATA SAMSUNG MZNLN256 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
1	1049kB	269MB	268MB	primary	xfs	boot
2	269MB	34.6GB	34.4GB	primary		
3	34.6GB	45.4GB	10.7GB	primary		
4	45.4GB	256GB	211GB	extended		
5	45.4GB	256GB	211GB	logical		

- オプション: 次に調べるデバイスに切り替えます。

(parted) select **block-device**

print コマンドの出力の詳細は、以下を参照してください。

Model: ATA SAMSUNG MZNLN256 (scsi)

ディスクタイプ、製造元、モデル番号、およびインターフェイス。

Disk `/dev/sda: 256GB`

ブロックデバイスへのファイルパスとストレージ容量。

Partition Table: msdos

ディスクラベルの種類。

Number

パーティション番号。たとえば、マイナー番号1のパーティションは、`/dev/sda1` に対応します。

Start および End

デバイスにおけるパーティションの開始場所と終了場所。

Type

有効なタイプは、メタデータ、フリー、プライマリー、拡張、または論理です。

File system

ファイルシステムの種類。ファイルシステムの種類が不明な場合は、デバイスの **File system** フィールドに値が表示されません。**parted** ユーティリティーは、暗号化されたデバイスのファイルシステムを認識できません。

Flags

パーティションのフラグ設定リスト。利用可能なフラグは、`boot`、`root`、`swap`、`hidden`、`raid`、`lvm`、または `lba` です。

関連情報

- システム上の **parted(8)** man ページ

3.3. PARTED を使用したパーティションの作成

システム管理者は、**parted** ユーティリティーを使用してディスクに新しいパーティションを作成できます。



注記

必要なパーティションは、`swap`、`/boot/`、および `/(root)` です。

前提条件

- ディスクのパーティションテーブル。
- 2TiB を超えるパーティションを作成する場合は、GUID Partition Table (GPT) でディスクをフォーマットしておく。

手順

- parted** ユーティリティーを起動します。

```
# parted block-device
```

- 現在のパーティションテーブルを表示し、十分な空き領域があるかどうかを確認します。

```
(parted) print
```

- 十分な空き容量がない場合は、パーティションのサイズを変更してください。
 - パーティションテーブルから、以下を確認します。
 - 新しいパーティションの開始点と終了点
 - MBR で、どのパーティションタイプにすべきか
3. 新しいパーティションを作成します。

```
(parted) mkpart part-type name fs-type start end
```

- **part-type** を **primary**、**logical**、または **extended** に置き換えます。これは MBR パーティションテーブルにのみ適用されます。
- **name** を任意のパーティション名に置き換えます。これは GPT パーティションテーブルに必要です。
- **fs-type** を、**xfs**、**ext2**、**ext3**、**ext4**、**fat16**、**fat32**、**hfs**、**hfs+**、**linux-swap**、**ntfs**、または **reiserfs** に置き換えます。**fs-type** パラメーターは任意です。**parted** ユーティリティは、パーティションにファイルシステムを作成しないことに注意してください。
- **start** と **end** を、パーティションの開始点と終了点を決定するサイズに置き換えます(ディスクの開始からカウントします)。**512MiB**、**20GiB**、**1.5TiB** などのサイズ接尾辞を使用できます。デフォルトサイズの単位はメガバイトです。

例3.2 小さなプライマリーパーティションの作成

MBR テーブルに 1024MiB から 2048MiB までのプライマリーパーティションを作成するには、次のコマンドを使用します。

```
(parted) mkpart primary 1024MiB 2048MiB
```

コマンドを入力すると、変更の適用が開始されます。

4. パーティションテーブルを表示して、作成されたパーティションのパーティションタイプ、ファイルシステムタイプ、サイズが、パーティションテーブルに正しく表示されていることを確認します。

```
(parted) print
```

5. **parted** シェルを終了します。

```
(parted) quit
```

6. 新規デバイスノードを登録します。

```
# udevadm settle
```

7. カーネルが新しいパーティションを認識していることを確認します。

```
# cat /proc/partitions
```

関連情報

- システム上の **parted(8)** man ページ
- parted** でディスクにパーティションテーブルを作成
- parted** でパーティションのサイズ変更

3.4. FDISK でパーティションタイプの設定

fdisk ユーティリティーを使用して、パーティションタイプまたはフラグを設定できます。

前提条件

- ディスク上のパーティション。

手順

- インタラクティブな **fdisk** シェルを起動します。

```
# fdisk block-device
```

- 現在のパーティションテーブルを表示して、パーティションのマイナー番号を確認します。

```
Command (m for help): print
```

現在のパーティションタイプは **Type** 列で、それに対応するタイプ ID は **Id** 列で確認できます。

- パーティションタイプコマンドを入力し、マイナー番号を使用してパーティションを選択します。

```
Command (m for help): type
Partition number (1,2,3 default 3): 2
```

- オプション: リストを 16 進数コードで表示します。

```
Hex code (type L to list all codes): L
```

- パーティションタイプを設定します。

```
Hex code (type L to list all codes): 8e
```

- 変更を書き込み、**fdisk** シェルを終了します。

```
Command (m for help): write
The partition table has been altered.
Syncing disks.
```

- 変更を確認します。

```
# fdisk --list block-device
```

3.5. PARTED でパーティションのサイズ変更

parted ユーティリティーを使用して、パーティションを拡張して未使用のディスク領域を利用したり、パーティションを縮小してその容量をさまざまな目的に使用したりできます。

前提条件

- パーティションを縮小する前にデータをバックアップする。
- 2TiB を超えるパーティションを作成する場合は、GUID Partition Table (GPT) でディスクをフォーマットしておく。
- パーティションを縮小する場合は、サイズを変更したパーティションより大きくならないよう最初にファイルシステムを縮小しておく。



注記

XFS は縮小に対応していません。

手順

1. **parted** ユーティリティーを起動します。

```
# parted block-device
```

2. 現在のパーティションテーブルを表示します。

```
(parted) print
```

パーティションテーブルから、以下を確認します。

- パーティションのマイナー番号。
- 既存のパーティションの位置とサイズ変更後の新しい終了点。

3. パーティションのサイズを変更します。

```
(parted) resizepart 1 2GiB
```

- 1を、サイズを変更するパーティションのマイナー番号に置き換えます。
- 2を、サイズを変更するパーティションの新しい終了点を決定するサイズに置き換えます (ディスクの開始からカウントします)。512MiB、20GiB、1.5TiBなどのサイズ接尾辞を使用できます。デフォルトサイズの単位はメガバイトです。

4. パーティションテーブルを表示して、サイズ変更したパーティションのサイズが、パーティションテーブルで正しく表示されていることを確認します。

```
(parted) print
```

5. **parted** シェルを終了します。

```
(parted) quit
```

6. カーネルが新しいパーティションを登録していることを確認します。

```
# cat /proc/partitions
```

7. オプション: パーティションを拡張した場合は、そこにあるファイルシステムも拡張します。

関連情報

- システム上の **parted(8)** man ページ

3.6. PARTED でパーティションの削除

parted ユーティリティーを使用すると、ディスクパーティションを削除して、ディスク領域を解放できます。

手順

1. インタラクティブな **parted** シェルを起動します。

```
# parted block-device
```

- block-device** を、パーティションを削除するデバイスへのパス (例: **/dev/sda**) に置き換えます。

2. 現在のパーティションテーブルを表示して、削除するパーティションのマイナー番号を確認します。

```
(parted) print
```

3. パーティションを削除します。

```
(parted) rm minor-number
```

- minor-number** を、削除するパーティションのマイナー番号に置き換えます。

このコマンドを実行すると、すぐに変更の適用が開始されます。

4. パーティションテーブルからパーティションが削除されたことを確認します。

```
(parted) print
```

5. **parted** シェルを終了します。

```
(parted) quit
```

6. パーティションが削除されたことをカーネルが登録していることを確認します。

```
# cat /proc/partitions
```

7. パーティションが存在する場合は、**/etc/fstab** ファイルからパーティションを削除します。削除したパーティションを宣言している行を見つけ、ファイルから削除します。

8. システムが新しい **/etc/fstab** 設定を登録するように、マウントユニットを再生成します。

```
# systemctl daemon-reload
```

9. スワップパーティション、または LVM の一部を削除した場合は、カーネルコマンドラインからパーティションへの参照をすべて削除します。

- a. アクティブなカーネルオプションを一覧表示し、削除されたパーティションを参照するオプションがないか確認します。

```
# grubby --info=ALL
```

- b. 削除されたパーティションを参照するカーネルオプションを削除します。

```
# grubby --update-kernel=ALL --remove-args="option"
```

10. アーリーブートシステムに変更を登録するには、**initramfs** ファイルシステムを再構築します。

```
# dracut --force --verbose
```

関連情報

- システム上の **parted(8)** man ページ

第4章 ディスクを再設定するストラテジー

ディスクのパーティションを再設定する方法は複数あります。これには以下が含まれます。

- パーティションが分割されていない空き領域が利用できる。
- 未使用のパーティションが利用可能である。
- アクティブに使用されているパーティションの空き領域が利用可能である。



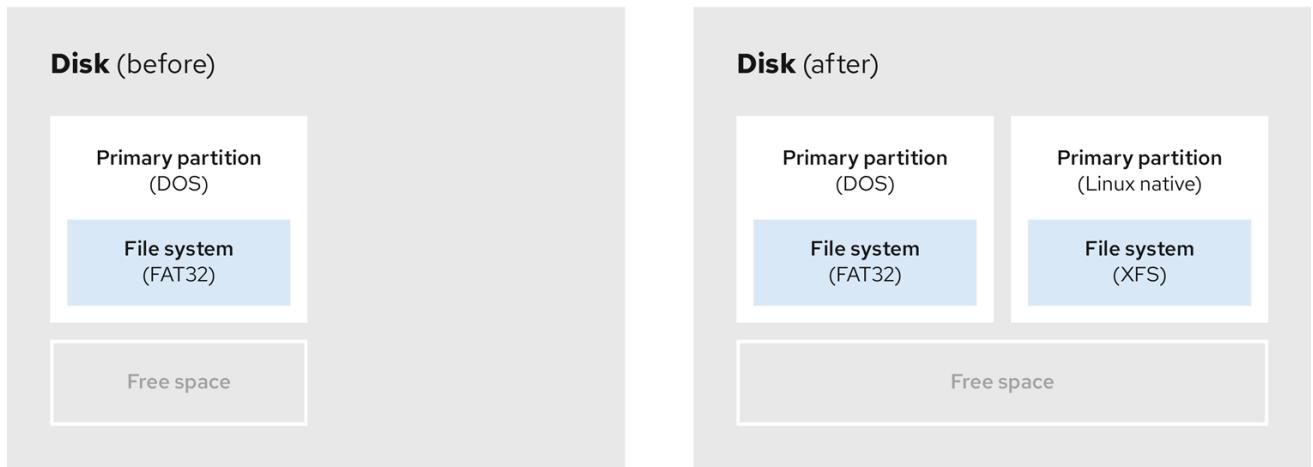
注記

以下の例は、わかりやすくするために単純化されており、実際に Red Hat Enterprise Linux をインストールするときの正確なパーティションレイアウトは反映していません。

4.1. パーティションが分割されていない空き領域の使用

すでに定義されているパーティションはハードディスク全体にまたがらないため、定義されたパーティションには含まれない未割り当ての領域が残されます。次の図は、これがどのようになるかを示しています。

図4.1パーティションが分割されていない空き領域があるディスク



269_RHEL_0822

最初の図は、1つのプライマリパーティションと未割り当て領域のある未定義のパーティションを持つディスクを表しています。2番目の図は、スペースが割り当てられた2つの定義済みパーティションを持つディスクを表しています。

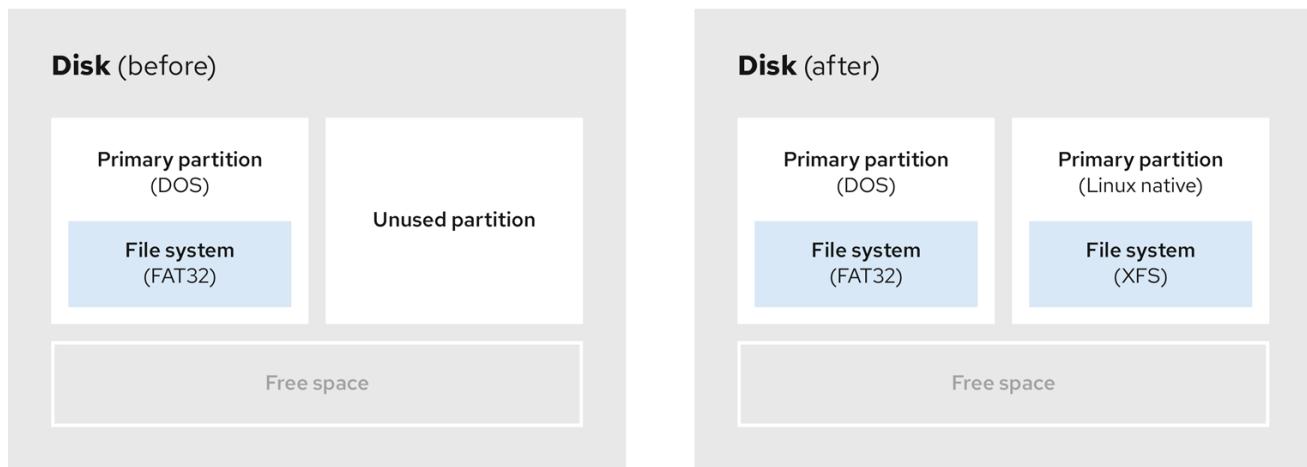
未使用のハードディスクもこのカテゴリーに分類されます。唯一の違いは、すべての領域が定義されたパーティションの一部ではないことです。

新しいディスクでは、未使用の領域から必要なパーティションを作成できます。ほとんどのオペレーティングシステムは、ディスクドライブ上の利用可能な領域をすべて取得するように設定されています。

4.2. 未使用パーティションの領域の使用

次の例の最初の図は、未使用のパーティションを持つディスクを表しています。2番目の図は、Linuxの未使用パーティションの再割り当てを表しています。

図4.2 未使用のパーティションがあるディスク

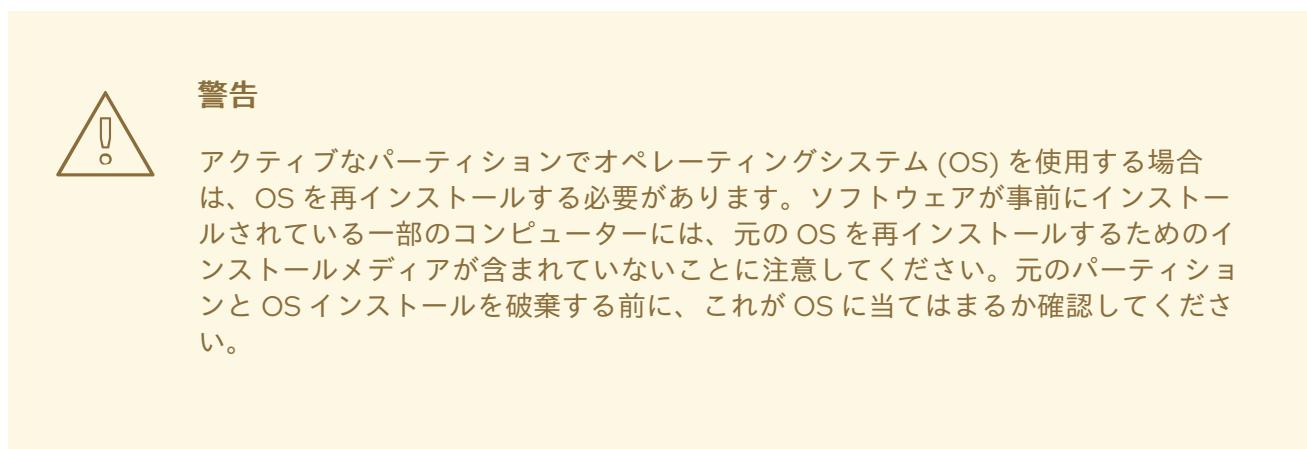


269_RHEL_0822

未使用のパーティションに割り当てられた領域を使用するには、パーティションを削除してから、代わりに適切な Linux パーティションを作成します。または、インストールプロセス時に未使用のパーティションを削除し、新しいパーティションを手動で作成します。

4.3. アクティブなパーティションの空き領域の使用

すでに使用されているアクティブなパーティションには、必要な空き領域が含まれているため、このプロセスの管理は困難な場合があります。ほとんどの場合、ソフトウェアが事前にインストールされているコンピューターのハードディスクには、オペレーティングシステムとデータを保持する大きなパーティションが1つ含まれます。



使用可能な空き領域の使用を最適化するには、破壊的または非破壊的なパーティション再設定の方法を使用できます。

4.3.1. 破壊的な再設定

破壊的なパーティション再設定は、ハードドライブのパーティションを破棄し、代わりにいくつかの小さなパーティションを作成します。この方法は完全にコンテンツを削除するため、元のパーティションから必要なデータをバックアップします。

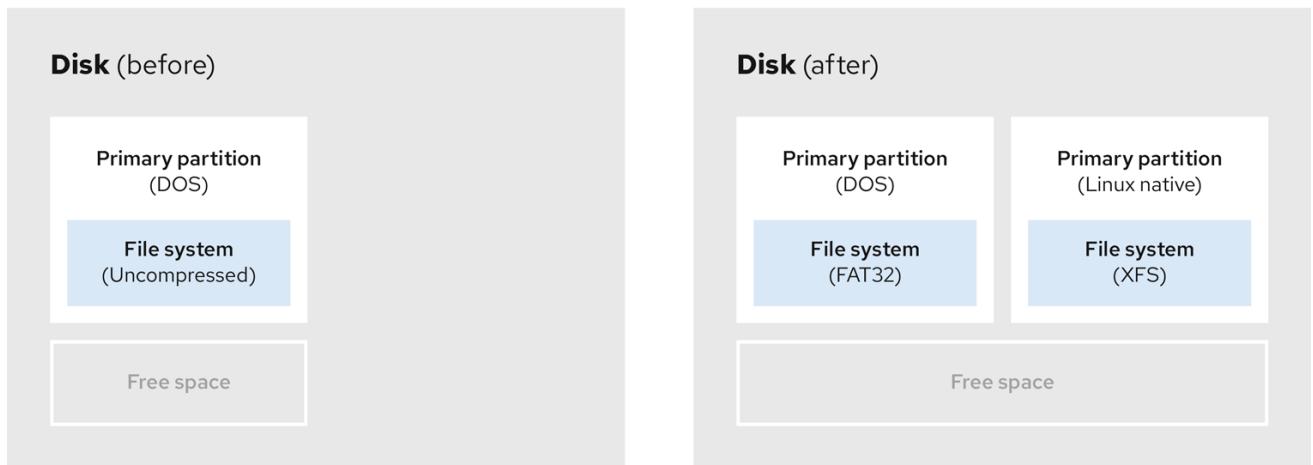
既存のオペレーティングシステム用に小規模なパーティションを作成すると、以下が可能になります。

- ソフトウェアの再インストール。

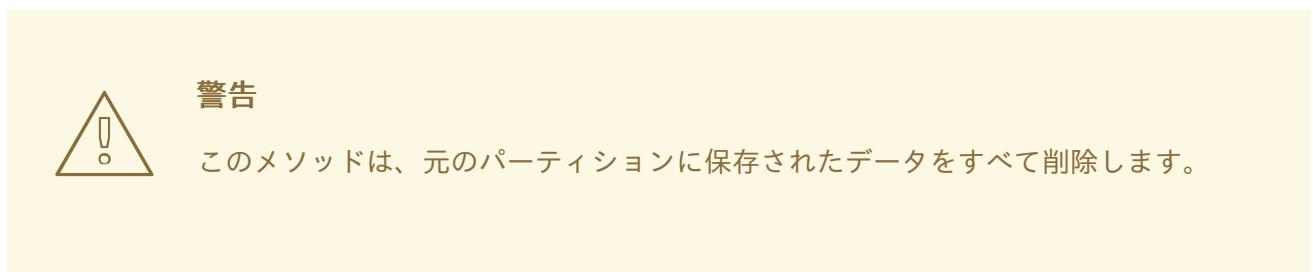
- データの復元。
- Red Hat Enterprise Linux インストールの開始。

以下の図は、破壊的なパーティション再設定の方法を使用を簡潔に示しています。

図4.3 ディスク上での破壊的な再パーティション処理



269_RHEL_0822



4.3.2. 非破壊的な再パーティション

非破壊的なパーティション再設定では、データの損失なしにパーティションのサイズを変更します。この方法は信頼性できますが、大きなドライブでは処理に時間がかかります。

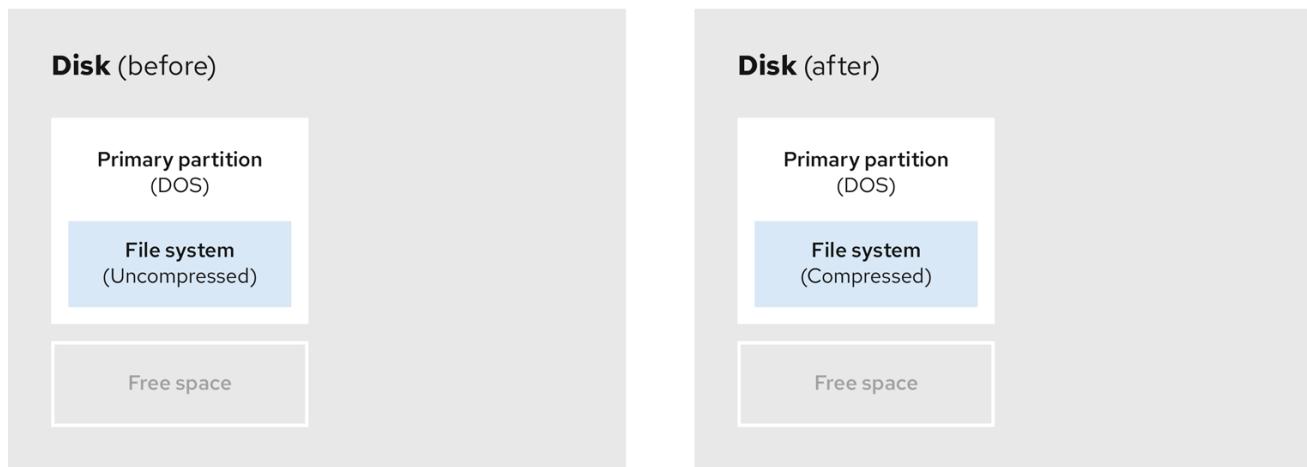
以下は、破壊的なパーティション再設定の開始に役立つメソッドのリストです。

- 既存データの圧縮

一部のデータの保存場所は変更できません。これにより、必要なサイズへのパーティションのサイズ変更が妨げられ、最終的に破壊的なパーティション再設定プロセスが必要になる可能性があります。既存のパーティションでデータを圧縮すると、必要に応じてパーティションのサイズを変更できます。また、使用可能な空き容量を最大化することもできます。

以下の図は、このプロセスを簡略化したものです。

図4.4 ディスク上でのデータ圧縮



269_RHEL_0822

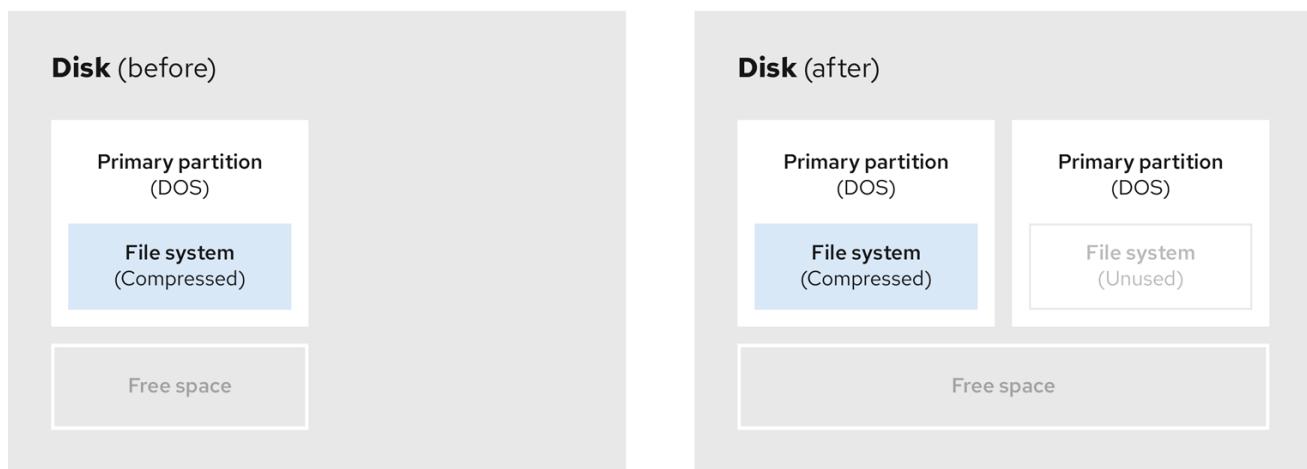
データ損失の可能性を回避するには、圧縮プロセスを続行する前にバックアップを作成します。

- 既存パーティションのサイズ変更

既存のパーティションのサイズを変更すると、より多くの領域を解放できます。結果は、サイズ変更ソフトウェアにより異なります。多くの場合、元のパーティションと同じタイプのフォーマットされていない新しいパーティションを作成できます。

サイズ変更後の手順は、使用するソフトウェアにより異なります。以下の例では、新しい DOS (Disk Operating System) パーティションを削除し、代わりに Linux パーティションを作成することを推奨します。サイズ変更プロセスを開始する前に、何がディスクに最適か確認してください。

図4.5 ディスク上でのパーティションのサイズ変更



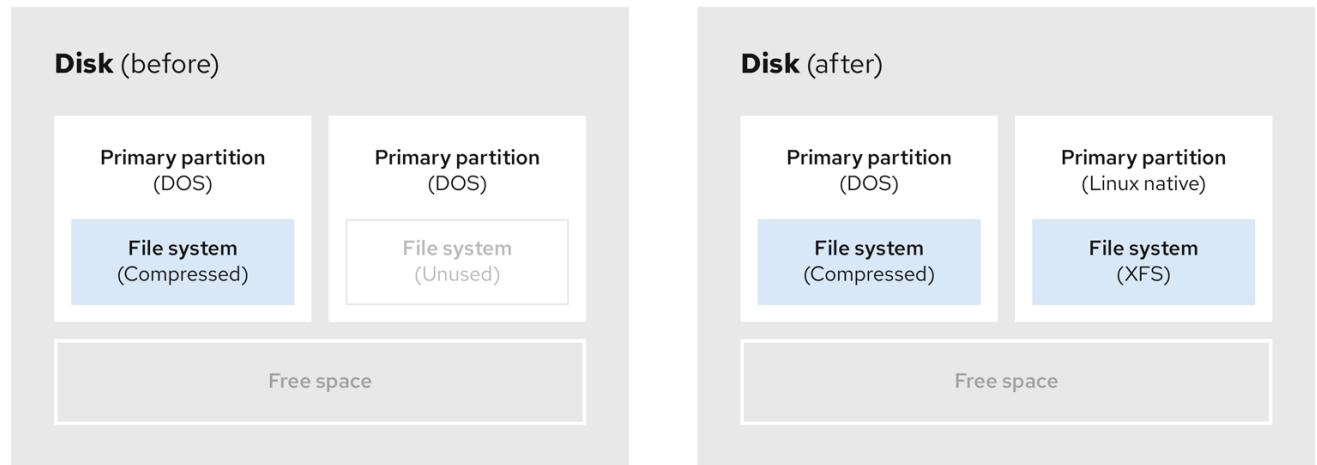
269_RHEL_0822

- オプション: 新規パーティションの作成

一部のサイズ変更ソフトウェアは、Linux ベースのシステムをサポートしています。この場合、サイズ変更後に新たに作成されたパーティションを削除する必要はありません。新しいパーティションの作成方法は、使用するソフトウェアによって異なります。

以下の図は、新しいパーティションを作成する前後のディスクの状態を示しています。

図4.6 最終パーティション設定のディスク



269_RHEL_0822

第5章 永続的な命名属性の概要

システム管理者は、永続的な命名属性を使用してストレージボリュームを参照し、再起動を何度も行つても信頼できるストレージ設定を構築する必要があります。

5.1. 非永続的な命名属性のデメリット

Red Hat Enterprise Linux では、ストレージデバイスを識別する方法が複数あります。特にドライブへのインストール時やドライブの再フォーマット時に誤ったデバイスにアクセスしないようにするために、適切なオプションを使用して各デバイスを識別することが重要になります。

従来、**/dev/sd(メジャー番号)(マイナー番号)** の形式の非永続的な名前は、ストレージデバイスを参照するために Linux 上で使用されます。メジャー番号とマイナー番号の範囲、および関連する **sd** 名は、検出されると各デバイスに割り当てられます。つまり、デバイスの検出順序が変わると、メジャー番号とマイナー番号の範囲、および関連する **sd** 名の関連付けが変わる可能性があります。

このような順序の変更は、以下の状況で発生する可能性があります。

- システム起動プロセスの並列化により、システム起動ごとに異なる順序でストレージデバイスが検出された場合。
- ディスクが起動しなかったり、SCSI コントローラに応答しなかった場合。この場合は、通常のデバイスプローブにより検出されません。ディスクはシステムにアクセスできなくなり、後続のデバイスは関連する次の **sd** 名が含まれる、メジャー番号およびマイナー番号の範囲があります。たとえば、通常 **sdb** と呼ばれるディスクが検出されないと、**sdc** と呼ばれるディスクが **sdb** として代わりに表示されます。
- SCSI コントローラ (ホストバスアダプターまたは HBA) が初期化に失敗し、その HBA に接続されているすべてのディスクが検出されなかった場合。後続のプローブされた HBA に接続しているディスクは、別のメジャー番号およびマイナー番号の範囲、および関連する別の **sd** 名が割り当てられます。
- システムに異なるタイプの HBA が存在する場合は、ドライバー初期化の順序が変更する可能性があります。これにより、HBA に接続されているディスクが異なる順序で検出される可能性があります。また、HBA がシステムの他の PCI スロットに移動した場合でも発生する可能性があります。
- ストレージアレイや干渉するスイッチの電源が切れた場合など、ストレージデバイスがプローブされたときに、ファイバーチャネル、iSCSI、または FCoE アダプターを持つシステムに接続されたディスクがアクセスできなくなる可能性があります。システムが起動するまでの時間よりもストレージアレイがオンラインになるまでの時間の方が長い場合に、電源の障害後にシステムが再起動すると、この問題が発生する可能性があります。一部のファイバーチャネルドライバーは WWPN マッピングへの永続 SCSI ターゲット ID を指定するメカニズムをサポートしますが、メジャー番号およびマイナー番号の範囲や関連する **sd** 名は予約されず、一貫性のある SCSI ターゲット ID 番号のみが提供されます。

そのため、**/etc/fstab** ファイルなどにあるデバイスを参照するときにメジャー番号およびマイナー番号の範囲や関連する **sd** 名を使用することは望ましくありません。誤ったデバイスがマウントされ、データが破損する可能性があります。

しかし、場合によっては他のメカニズムが使用される場合でも **sd** 名の参照が必要になる場合もあります (デバイスによりエラーが報告される場合など)。これは、Linux カーネルはデバイスに関するカーネルメッセージで **sd** 名 (および SCSI ホスト、チャネル、ターゲット、LUN タプル) を使用するためです。

5.2. ファイルシステムおよびデバイスの識別子

ファイルシステムの識別子は、ファイルシステム自体に関連付けられます。一方、デバイスの識別子は、物理ブロックデバイスに紐付けられます。適切なストレージ管理を行うには、その違いを理解することが重要です。

ファイルシステムの識別子

ファイルシステムの識別子は、ブロックデバイス上に作成された特定のファイルシステムに関連付けられます。識別子はファイルシステムの一部としても格納されます。ファイルシステムを別のデバイスにコピーしても、ファイルシステム識別子は同じです。ただし、**mkfs** ユーティリティーでフォーマットするなどしてデバイスを書き換えると、デバイスはその属性を失います。

ファイルシステムの識別子に含まれるのは、次のとおりです。

- 一意識別子 (UUID)
- ラベル

デバイスの識別子

デバイス識別子は、ブロックデバイス (ディスクやパーティションなど) に関連付けられます。**mkfs** ユーティリティーでフォーマットするなどしてデバイスを書き換えた場合、デバイスはファイルシステムに格納されていないため、属性を保持します。

デバイスの識別子に含まれるのは、次のとおりです。

- World Wide Identifier (WWID)
- パーティション UUID
- シリアル番号

推奨事項

- 論理ボリュームなどの一部のファイルシステムは、複数のデバイスにまたがっています。Red Hat は、デバイスの識別子ではなくファイルシステムの識別子を使用してこのファイルシステムにアクセスすることを推奨します。

5.3. /DEV/DISK/ にある UDEV メカニズムにより管理されるデバイス名

udev メカニズムは、Linux のすべてのタイプのデバイスに使用され、ストレージデバイスだけに限定されません。/dev/disk/ ディレクトリーにさまざまな種類の永続的な命名属性を提供します。ストレージデバイスの場合、Red Hat Enterprise Linux には /dev/disk/ ディレクトリーにシンボリックリンクを作成する **udev** ルールが含まれています。これにより、次の方法でストレージデバイスを参照できます。

- ストレージデバイスのコンテンツ
- 一意識別子
- シリアル番号

udev の命名属性は永続的なものですが、システムを再起動しても自動的には変更されないため、設定可能なこともあります。

5.3.1. ファイルシステムの識別子

/dev/disk/by-uuid/ の UUID 属性

このディレクトリーのエントリーは、デバイスに格納されているコンテンツ(つまりデータ)内の一意識別子(UUID)によりストレージデバイスを参照するシンボリック名を提供します。以下に例を示します。

/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6

次の構文を使用することで、UUID を使用して /etc/fstab ファイルのデバイスを参照できます。

UUID=3e6be9de-8139-11d1-9106-a43f08d823a6

ファイルシステムを作成する際に UUID 属性を設定できます。後で変更することもできます。

/dev/disk/by-label/ のラベル属性

このディレクトリーのエントリーは、デバイスに格納されているコンテンツ(つまりデータ)内の ラベルにより、ストレージデバイスを参照するシンボリック名を提供します。

以下に例を示します。

/dev/disk/by-label/Boot

次の構文を使用することで、ラベルを使用して /etc/fstab ファイルのデバイスを参照できます。

LABEL=Boot

ファイルシステムを作成するときにラベル属性を設定できます。また、後で変更することもできます。

5.3.2. デバイスの識別子

/dev/disk/by-id/ の WWID 属性

World Wide Identifier (WWID) は永続的で、SCSI 規格によりすべての SCSI デバイスが必要とするシステムに依存しない識別子です。各ストレージデバイスの WWID 識別子は一意となることが保証され、デバイスのアクセスに使用されるパスに依存しません。この識別子はデバイスのプロパティーですが、デバイスのコンテンツ(つまりデータ)には格納されません。

この識別子は、SCSI Inquiry を発行して Device Identification Vital Product Data (0x83 ページ) または Unit Serial Number (0x80 ページ) を取得することにより獲得できます。

Red Hat Enterprise Linux では、WWID ベースのデバイス名から、そのシステムの現在の /dev/sd 名への正しいマッピングを自動的に維持します。デバイスへのパスが変更したり、別のシステムからそのデバイスへのアクセスがあった場合にも、アプリケーションはディスク上のデータ参照に /dev/disk/by-id/ を使用できます。

例5.1 WWID マッピング

WWID シンボリックリンク	非永続的なデバイス	備考
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	/dev/sda	ページ 0x83 の識別子を持つデバイス

WWID シンボリックリンク	非永続的なデバイス	備考
<code>/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6</code>	<code>/dev/sdb</code>	ページ <code>0x80</code> の識別子を持つデバイス
<code>/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3</code>	<code>/dev/sdc3</code>	ディスクパーティション

システムにより提供される永続的な名前のほかに、**udev** ルールを使用して独自の永続的な名前を実装し、ストレージの WWID にマップすることもできます。

`/dev/disk/by-partuuid` のパーティション UUID 属性

パーティション UUID (PARTUUID) 属性は、GPT パーティションテーブルにより定義されているパーティションを識別します。

例5.2 パーティション UUID のマッピング

PARTUUID シンボリックリンク	非永続的なデバイス
<code>/dev/disk/by-partuuid/4cd1448a-01</code>	<code>/dev/sda1</code>
<code>/dev/disk/by-partuuid/4cd1448a-02</code>	<code>/dev/sda2</code>
<code>/dev/disk/by-partuuid/4cd1448a-03</code>	<code>/dev/sda3</code>

`/dev/disk/by-path/` のパス属性

この属性は、デバイスへのアクセスに使用される **ハードウェアパス** がストレージデバイスを参照するシンボル名を提供します。

ハードウェアパス (PCI ID、ターゲットポート、LUN 番号など) の一部が変更されると、パス属性に失敗します。このため、パス属性は信頼性に欠けます。ただし、パス属性は以下のいずれかのシナリオで役に立ちます。

- 後で置き換える予定のディスクを特定する必要があります。
- 特定の場所にあるディスクにストレージサービスをインストールする予定です。

5.4. DM MULTIPATH を使用した WORLD WIDE IDENTIFIER

Device Mapper (DM) Multipath を設定して、World Wide Identifier (WWID) と非永続的なデバイス名をマッピングできます。

システムからデバイスへのパスが複数ある場合、DM Multipath はこれを検出するために WWID を使用します。その後、DM Multipath は `/dev/mapper/wwid` ディレクトリー (例: `/dev/mapper/3600508b400105df70000e00000ac0000`) に単一の "疑似デバイス" を表示します。

コマンド **multipath -l** は、非永続的な識別子へのマッピングを示します。

- **Host:Channel:Target:LUN**
- **/dev/sd** 名
- **major:minor** 数値

例5.3 マルチパス設定での WWID マッピング

multipath -l コマンドの出力例:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwandler=0][rw]
\ round-robin 0 [prio=0][active]
 \ 5:0:1:1 sdc 8:32 [active][undef]
 \ 6:0:1:1 sdg 8:96 [active][undef]
 \ round-robin 0 [prio=0][enabled]
 \ 5:0:0:1 sdb 8:16 [active][undef]
 \ 6:0:0:1 sdf 8:80 [active][undef]
```

DM Multipath は、各 WWID ベースのデバイス名から、システムで対応する **/dev/sd** 名への適切なマッピングを自動的に維持します。これらの名前は、パスが変更しても持続し、他のシステムからデバイスにアクセスする際に一貫性を保持します。

DM Multipath の **user_friendly_names** 機能を使用すると、WWID は **/dev/mapper/mpathN** 形式の名前にマップされます。デフォルトでは、このマッピングは **/etc/multipath/bindings** ファイルに保持されています。これらの **mpathN** 名は、そのファイルが維持されている限り永続的です。



重要

user_friendly_names を使用する場合は、クラスター内で一貫した名前を取得するためには追加の手順が必要です。

5.5. UDEV デバイス命名規則の制約

udev 命名規則の制約の一部は次のとおりです。

- **udev** イベントに対して **udev** ルールが処理されるときに、**udev** メカニズムはストレージデバイスをクエリーする機能に依存する可能性があるため、クエリーの実行時にデバイスにアクセスできない可能性があります。これは、ファイバーチャネル、iSCSI、または FCoEストレージデバイスといった、デバイスがサーバーシャーシにない場合に発生する可能性が高くなります。
- カーネルは **udev** イベントをいつでも送信する可能性があるため、デバイスにアクセスできない場合に **/dev/disk/by-*/** リンクが削除される可能性があります。
- **udev** イベントが生成されそのイベントが処理されるまでに遅延が生じる場合があります（大量のデバイスが検出され、ユーザー空間の **udev** サービスによる各デバイスのルールを処理するのにある程度の時間がかかる場合など）。これにより、カーネルがデバイスを検出してから、**/dev/disk/by-*/** の名前が利用できるようになるまでに遅延が生じる可能性があります。

- ルールに呼び出される **blkid** などの外部プログラムによってデバイスが短期間開き、他の目的でデバイスにアクセスできなくなる可能性があります。
- /dev/disk/** の **udev** メカニズムで管理されるデバイス名は、メジャーリリース間で変更される可能性があるため、リンクの更新が必要になる場合があります。

5.6. 永続的な命名属性のリスト表示

非永続ストレージデバイスの永続的な命名属性を確認できます。

手順

- UUID 属性とラベル属性をリスト表示するには、**lsblk** ユーティリティーを使用します。

```
$ lsblk --fs storage-device
```

以下に例を示します。

例5.4 ファイルシステムの UUID とラベルの表示

```
$ lsblk --fs /dev/sda1
```

NAME	FSTYPE	LABEL	UUID	MOUNTPOINT
sda1	xfs	Boot	afa5d5e3-9050-48c3-acc1-bb30095f3dc4	/boot

- PARTUUID 属性をリスト表示するには、**--output +PARTUUID** オプションを指定して **lsblk** ユーティリティーを使用します。

```
$ lsblk --output +PARTUUID
```

以下に例を示します。

例5.5 パーティションの PARTUUID 属性の表示

```
$ lsblk --output +PARTUUID /dev/sda1
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT	PARTUUID
sda1	8:1	0	512M	0	part	/boot	4cd1448a-01

- WWID 属性をリスト表示するには、**/dev/disk/by-id/** ディレクトリーのシンボリックリンクのターゲットを調べます。以下に例を示します。

例5.6 システムにある全ストレージデバイスの WWID の表示

```
$ file /dev/disk/by-id/*
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2
```

```

symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root
symbolic link to ../../dm-0
/dev/disk/by-id/dm-name-rhel_rhel8-swap
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrlRLhzfMyOKd
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-wu4QQm
symbolic link to ../../sda2

```

5.7. 永続的な命名属性の変更

ファイルシステムの UUID またはラベルの永続的な命名属性を変更できます。



注記

udev 属性の変更はバックグラウンドで行われ、時間がかかる場合があります。 **udevadm settle** コマンドは、変更が完全に登録されるまで待機します。そのため、その後のコマンドで確実に新しい属性を正しく使用できます。

以下のコマンドでは、次を行います。

- **new-uuid** を、設定する UUID (例: **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**) に置き換えます。 **uuidgen** コマンドを使用して UUID を生成できます。
- **new-label** を、ラベル (例: **backup_data**) に置き換えます。

前提条件

- XFS ファイルシステムをアンマウントしている (XFS ファイルシステムの属性を変更する場合)。

手順

- XFS ファイルシステムの UUID またはラベル属性を変更するには、**xfs_admin** ユーティリティを使用します。


```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```
- ext4 ファイルシステム、ext3 ファイルシステム、ext2 ファイルシステムの UUID またはラベル属性を変更するには、**tune2fs** ユーティリティを使用します。


```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```
- スワップボリュームの UUID またはラベル属性を変更するには、**swaponlabel** ユーティリティを使用します。

```
# swaplabel --uuid new-uuid --label new-label swap-device  
# udevadm settle
```

第6章 NVDIMM 永続メモリーストレージの使用

システムに接続した NVDIMM (Non-Volatile Dual In-line Memory Modules) デバイス上にあるさまざまなタイプのストレージの有効化および管理を行うことができます。

NVDIMM ストレージに Red Hat Enterprise Linux 8 をインストールする場合は、[NVDIMM デバイスへのインストール](#) を参照してください。

6.1. NVDIMM 永続メモリーテクノロジー

Non-Volatile Dual In-line Memory Modules (NVDIMM) 永続メモリーは、ストレージクラスメモリーまたは **pmem** とも呼ばれ、メモリーとストレージの組み合わせです。

NVDIMM は、ストレージの耐久性に加え、低アクセスレイテンシーと動的な DRAM の広帯域幅を採用しています。NVDIMM を使用するその他の利点は次のとおりです。

- NVDIMM ストレージはバイトアドレス指定可能です。つまり、CPU ロードおよびストア命令を使用してアクセスできます。従来のブロックベースのストレージへのアクセスに必要なシステムコール `read()` および `write()` の他に、NVDIMM はダイレクトロードとストアプログラミングモデルにも対応しています。
- NVDIMM のパフォーマンス特性は、アクセスレイテンシーが非常に低い DRAM と似ています。通常、数万から数十万ナノ秒です。
- NVDIMM に保存されたデータは、永続メモリーと同様に、電源がオフになっても保持されます。
- ダイレクトアクセス (DAX) テクノロジーを使用すると、システムページキャッシュを経由せずにメモリーマップストレージへのアプリケーションを直接実行できます。これにより、別の目的で DRAM を解放します。

NVDIMM は、次のようなユースケースでメリットがあります。

データベース

NVDIMM でのストレージアクセスレイテンシーの短縮により、データベースのパフォーマンスが向上します。

高速な再起動

高速な再起動は、ウォームキャッシュ効果とも呼ばれます。たとえば、ファイルサーバーは起動後、メモリー内にファイルコンテンツを持ちません。クライアントがデータを接続して読み書きすると、そのデータはページキャッシュにキャッシュされます。最終的に、キャッシュには、ほとんどのホットデータが含まれます。再起動後、システムは従来のストレージで再度プロセスを開始する必要があります。

NVDIMM を使用すると、アプリケーションが適切に設計されていれば、再起動後もウォームキャッシュを維持できます。この例には、ページキャッシュは含まれません。アプリケーションは、永続メモリーに直接データをキャッシュします。

高速書き込みキャッシュ

多くの場合、ファイルサーバーは、データが耐久性のあるメディアに保存されるまで、クライアントの書き込み要求を認識しません。NVDIMM を高速な書き込みキャッシュとして使用すると、ファイルサーバーが書き込み要求をすばやく認識できるようになり、遅延が少くなります。

6.2. NVDIMM のインターリービングおよび地域

不揮発性デュアルインラインメモリーモジュール (NVDIMM) デバイスは、インターリープ領域へのグループ化をサポートしています。

NVDIMM デバイスは、通常のダイナミック RAM (DRAM) と同じ方法でインターリープセットにグループ化できます。インターリープセットは、複数の DIMM にまたがる RAID 0 レベル (ストライプ) 設定と似ています。インターリープセットは、リージョンとも呼ばれます。

インターリービングには、以下の利点があります。

- NVDIMM は、インターリープセットに設定するとパフォーマンスが向上します。
- インターリービングは、複数の小さな NVDIMM デバイスを大きな論理デバイスに組み合わせます。

NVDIMM インターリープセットは、システムの BIOS または UEFI ファームウェアで設定されます。Red Hat Enterprise Linux は、各インターリープセットにリージョンデバイスを作成します。

6.3. NVDIMM 名前空間

不揮発性デュアルインラインメモリーモジュール (NVDIMM) 領域は、ラベル領域のサイズに応じて 1 つ以上の名前空間に分割できます。名前空間を使用すると、**sector**、**fsdax**、**devdax**、**raw** などの名前空間のアクセスモードに基づいて、さまざまな方法でデバイスにアクセスできます。詳細については、[NVDIMM アクセスマード](#) を参照してください。

一部の NVDIMM デバイスは、任意のリージョンでの複数の名前空間に対応していません。

- お使いの NVDIMM デバイスがラベルに対応している場合は、リージョンを名前空間に分割できます。
- NVDIMM デバイスがラベルに対応していない場合は、リージョンに名前空間を 1 つだけ追加できます。この場合、Red Hat Enterprise Linux は、リージョン全体に対応するデフォルトの名前空間を作成します。

6.4. NVDIMM アクセスマード

Non-Volatile Dual In-line Memory Modules (NVDIMM) 名前空間を設定して、次のいずれかのモードを使用できます。

sector

ストレージを高速ブロックデバイスとして示します。このモードは、NVDIMM ストレージを使用するように変更されていないレガシーアプリケーションや、Device Mapper を含む完全な I/O スタックを使用するアプリケーションに役立ちます。

セクター デバイスは、システム上のその他のブロックデバイスと同じ方法で使用できます。そこではパーティションやファイルシステムを作成し、ソフトウェア RAID セットの一部として作成したり、**dm-cache** のキャッシュデバイスとして使用できます

このモードのデバイスは、**/dev/pmemNs** として利用できます。名前空間を作成したら、リストされている **blockdev** 値を確認します。

devdax またはデバイスダイレクトアクセス (DAX)

devdax を使用すると、NVDIMM デバイスは、Storage Networking Industry Association (SNIA) Non-Volatile Memory (NVM) Programming Model 仕様で説明されているように、直接アクセスプログラミングをサポートします。このモードでは、I/O はカーネルのストレージスタックを回避します。したがって、デバイスマッパードライバーは使用できません。

デバイス DAX は、DAX キャラクター デバイス ノードを使用して NVDIMM ストレージへの raw アクセスを提供します。CPU キャッシュのフラッシュ命令とフェンシング命令を使用して、**devdax** デバイスのデータを作成できます。特定のデータベースおよび仮想マシンのハイパーバイザーは、このモードの利点を得られます。**devdax** デバイスにファイルシステムを作成することはできません。

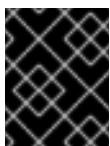
このモードのデバイスは **/dev/daxN.M** として利用できます。名前空間を作成したら、リストされた **chardev** 値を確認します。

fsdax またはファイルシステムダイレクトアクセス (DAX)

fsdax を使用すると、NVDIMM デバイスは、Storage Networking Industry Association (SNIA) Non-Volatile Memory (NVM) Programming Model 仕様で説明されているように、直接アクセス プログラミングをサポートします。このモードでは、I/O はカーネルのストレージ スタックを回避するため、多くのデバイスマッパードライバーが使用できなくなります。

ファイルシステム DAX デバイスにファイルシステムを作成できます。

このモードのデバイスは、**/dev/pmemN** として利用できます。名前空間を作成したら、リストされている **blockdev** 値を確認します。



重要

ファイルシステムの DAX テクノロジーはテクノロジープレビューとしてのみ提供されるため、Red Hat では対応していません。

raw

DAX に対応していないメモリーディスクを示します。このモードでは、名前空間にいくつかの制限があるため、使用すべきではありません。

このモードのデバイスは、**/dev/pmemN** として利用できます。名前空間を作成したら、リストされている **blockdev** 値を確認します。

6.5. NDCTL のインストール

ndctl ユーティリティーをインストールして、不揮発性デュアルインラインメモリーモジュール (NVDIMM) デバイスを設定および監視できます。

手順

- **ndctl** ユーティリティーをインストールします。

```
# yum install ndctl
```

6.6. ブロックデバイスとして動作する NVDIMM 上のセクターネーム空間の作成

非揮発性デュアルインラインメモリーモジュール (NVDIMM) デバイスをセクターモード (レガシーモードとも呼ばれます) で設定して、従来のブロックベースのストレージをサポートできます。

次のいずれかになります。

- 既存の名前空間をセクターモードに再設定

- 新規セクター名前空間を作成 (利用可能な領域がある場合)

前提条件

- NVDIMM デバイスがシステムに接続されている。

6.6.1. 既存の NVDIMM 名前空間のセクターモードへの再設定

Non-Volatile Dual In-line Memory Modules (NVDIMM) 名前空間をセクターモードに再設定して、高速ブロックデバイスとして使用できます。



警告

名前空間を再設定すると、名前空間に以前に保存されたデータが削除されます。

前提条件

- `ndctl` ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。

手順

- 既存の名前空間を表示します。

```
# ndctl list --namespaces --idle
[
  {
    "dev": "namespace1.0",
    "mode": "raw",
    "size": 34359738368,
    "state": "disabled",
    "numa_node": 1
  },
  {
    "dev": "namespace0.0",
    "mode": "raw",
    "size": 34359738368,
    "state": "disabled",
    "numa_node": 0
  }
]
```

- 選択した名前空間をセクターモードに再設定します。

```
# ndctl create-namespace --force --reconfig=namespace-ID --mode=sector
```

例6.1 セクターモードでの namespace1.0 の再設定

```
# ndctl create-namespace --force --reconfig=namespace1.0 --mode=sector
```

```
{
  "dev":"namespace1.0",
  "mode":"sector",
  "size":"755.26 GiB (810.95 GB)",
  "uuid":"2509949d-1dc4-4ee0-925a-4542b28aa616",
  "sector_size":4096,
  "blockdev":"pmem1s"
}
```

再設定された名前空間は、`/dev` ディレクトリーの下で `/dev/pmem1s` ファイルとして利用できるようになりました。

検証

- システム上の既存の名前空間が再設定されているかどうかを確認します。

```
# ndctl list --namespace namespace1.0
[
  {
    "dev":"namespace1.0",
    "mode":"sector",
    "size":810954706944,
    "uuid":"2509949d-1dc4-4ee0-925a-4542b28aa616",
    "sector_size":4096,
    "blockdev":"pmem1s"
  }
]
```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.6.2. セクターモードでの新たな NVDIMM 名前空間の作成

領域に利用可能なスペースがある場合、高速ブロックデバイスとして使用するために、不揮発性デュアルオンラインメモリーモジュール (NVDIMM) 名前空間をセクターモードで作成できます。

前提条件

- ndctl** ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。
- NVDIMM デバイスは、リージョン内に複数の名前空間を作成するためのラベルをサポートしています。これは、次のコマンドを使用して確認できます。

```
# ndctl read-labels nmem0 >/dev/null
read 1 nmem
```

これは、1つの NVDIMM デバイスのラベルを読み取ったことを示しています。値が **0** の場合、デバイスがラベルをサポートしていないことを意味します。

手順

- 利用可能な領域があるシステムの **pmem** リージョンのリストを表示します。以下の例では、**region1** リージョンと **region0** リージョンの領域が利用できます。

```
# ndctl list --regions
[
  {
    "dev": "region1",
    "size": 2156073582592,
    "align": 16777216,
    "available_size": 2117418876928,
    "max_available_extent": 2117418876928,
    "type": "pmem",
    "iset_id": -9102197055295954944,
    "badblock_count": 1,
    "persistence_domain": "memory_controller"
  },
  {
    "dev": "region0",
    "size": 2156073582592,
    "align": 16777216,
    "available_size": 2143188680704,
    "max_available_extent": 2143188680704,
    "type": "pmem",
    "iset_id": 736272362787276936,
    "badblock_count": 3,
    "persistence_domain": "memory_controller"
  }
]
```

- 利用可能な領域のいずれかに、1つ以上の名前空間を割り当てます。

```
# ndctl create-namespace --mode=sector --region=regionN --size=namespace-size
```

例6.2 region0 に 36 GiB セクターの名前空間を作成する

```
# ndctl create-namespace --mode=sector --region=region0 --size=36G
{
  "dev": "namespace0.1",
  "mode": "sector",
  "size": "35.96 GiB (38.62 GB)",
  "uuid": "ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
  "sector_size": 4096,
  "blockdev": "pmem0.1s"
}
```

新しい名前空間が **/dev/pmem0.1s** として利用できるようになりました。

検証

- 新しい名前空間がセクターモードで作成されているかどうかを確認します。

```
# ndctl list -RN -n namespace0.1
{
```

```

"regions": [
  {
    "dev": "region0",
    "size": 2156073582592,
    "align": 16777216,
    "available_size": 2104533975040,
    "max_available_extent": 2104533975040,
    "type": "pmem",
    "iset_id": 736272362787276936,
    "badblock_count": 3,
    "persistence_domain": "memory_controller",
    "namespaces": [
      {
        "dev": "namespace0.1",
        "mode": "sector",
        "size": 38615912448,
        "uuid": "ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
        "sector_size": 4096,
        "blockdev": "pmem0.1s"
      }
    ]
  }
]
}

```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.7. NVDIMM でのデバイス DAX 名前空間の作成

システムに接続されている NVDIMM デバイスをデバイス DAX モードで設定して、ダイレクトアクセス機能を備えたキャラクターストレージをサポートします。

次のオプションを検討してください。

- 既存の名前空間をデバイス DAX モードに再設定する。
- 利用可能な領域がある場合は、新しいデバイスの DAX 名前空間を作成する。

6.7.1. デバイスのダイレクトアクセスモードの NVDIMM

デバイスダイレクトアクセス (デバイス DAX である **devdax**) は、ファイルシステムの関与なしで、アプリケーションがストレージに直接アクセスできる手段を提供します。デバイス DAX の利点は、**ndctl** ユーティリティーの **--align** オプションを使用して設定できる、保証されたフォールトの粒度を提供することです。

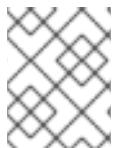
Intel 64 アーキテクチャーおよび AMD64 アーキテクチャーでは、以下のフォールト粒度に対応しています。

- 4 KiB
- 2 MiB
- 1 GiB

デバイス DAX ノードは、以下のシステム呼び出しにのみ対応しています。

- **open()**
- **close()**
- **mmap()**

ndctl list --human --capabilities コマンドを使用して、NVDIMM デバイスのサポートされているアライメントを表示できます。たとえば、region0 デバイスについて表示するには、**ndctl list --human --capabilities -r region0** コマンドを使用します。



注記

デバイスの DAX ユースケースは SNIA 不揮発性メモリープログラミングモデルに関連付けられているため、**read()** および **write()** システムコールはサポートされていません。

6.7.2. 既存の NVDIMM 名前空間をデバイス DAX モードに再設定

既存の不揮発性デュアルインラインメモリーモジュール (NVDIMM) 名前空間をデバイス DAX モードに再設定できます。



警告

名前空間を再設定すると、名前空間に以前に保存されたデータが削除されます。

前提条件

- **ndctl** ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。

手順

1. システムにある名前空間のリストを表示します。

```
# ndctl list --namespaces --idle

[
  {
    "dev": "namespace1.0",
    "mode": "raw",
    "size": 34359738368,
    "uuid": "ac951312-b312-4e76-9f15-6e00c8f2e6f4",
    "state": "disabled",
    "numa_node": 1
  },
  {
    "dev": "namespace0.0",
    "mode": "raw",
    "size": 38615912448,
    "uuid": "00000000-0000-0000-0000-000000000000"
  }
]
```

```

    "uuid":"ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
    "state":"disabled",
    "numa_node":0
  }
]
```

- 名前空間を再設定します。

```
# ndctl create-namespace --force --mode=devdax --reconfig=namespace-ID
```

例6.3 名前空間をデバイス DAX として再設定

次のコマンドは、DAX に対応するデータストレージ用に **namespace0.1** を再設定します。オペレーティングシステムが一度に 2 MiB ページでフォールトされるように、2 MiB フォールトの粒度に合わせて調整されます。

```

# ndctl create-namespace --force --mode=devdax --align=2M --reconfig=namespace0.1
{
  "dev":"namespace0.1",
  "mode":"devdax",
  "map":"dev",
  "size":"35.44 GiB (38.05 GB)",
  "uuid":"426d6a52-df92-43d2-8cc7-046241d6d761",
  "daxregion":{
    "id":0,
    "size":"35.44 GiB (38.05 GB)",
    "align":2097152,
    "devices":[
      {
        "chardev":"dax0.1",
        "size":"35.44 GiB (38.05 GB)",
        "target_node":4,
        "mode":"devdax"
      }
    ]
  },
  "align":2097152
}
```

名前空間は、**/dev/dax0.1** パスで利用できます。

検証

- システム上の既存の名前空間が再設定されているかどうかを確認します。

```

# ndctl list --namespace namespace0.1
[
  {
    "dev":"namespace0.1",
    "mode":"devdax",
    "map":"dev",
    "size":38048628736,
    "uuid":"426d6a52-df92-43d2-8cc7-046241d6d761",

```

```

        "chardev":"dax0.1",
        "align":2097152
    }
]
```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.7.3. デバイス DAX モードでの新しい NVDIMM 名前空間の作成

リージョンに空き容量がある場合は、Non-Volatile Dual In-line Memory Modules (NVDIMM) デバイスに新しいデバイス DAX 名前空間を作成できます。

前提条件

- ndctl** ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。
- NVDIMM デバイスは、リージョン内に複数の名前空間を作成するためのラベルをサポートしています。これは、次のコマンドを使用して確認できます。

```
# ndctl read-labels nmem0 >/dev/null
read 1 nmem
```

これは、1つの NVDIMM デバイスのラベルを読み取ったことを示しています。値が **0** の場合、デバイスがラベルをサポートしていないことを意味します。

手順

- 利用可能な領域があるシステムの **pmem** リージョンのリストを表示します。以下の例では、**region1** リージョンと **region0** リージョンの領域が利用できます。

```
# ndctl list --regions
[
{
    "dev":"region1",
    "size":2156073582592,
    "align":16777216,
    "available_size":2117418876928,
    "max_available_extent":2117418876928,
    "type":"pmem",
    "iset_id":9102197055295954944,
    "badblock_count":1,
    "persistence_domain":"memory_controller"
},
{
    "dev":"region0",
    "size":2156073582592,
    "align":16777216,
    "available_size":2143188680704,
    "max_available_extent":2143188680704,
    "type":"pmem",
    "iset_id":736272362787276936,
```

```

        "badblock_count":3,
        "persistence_domain":"memory_controller"
    }
]

```

- 利用可能な領域のいずれかに、1つ以上の名前空間を割り当てます。

```
# ndctl create-namespace --mode=devdax --region=regionN --size=namespace-size
```

例6.4 リージョンへの名前空間の作成

次のコマンドは、region0 に 36-GiB のデバイス DAX 名前空間を作成します。オペレーティングシステムが一度に 2 MiB ページでフォールトされるように、2 MiB フォールトの粒度に合わせて調整されます。

```

# ndctl create-namespace --mode=devdax --region=region0 --align=2M --size=36G
{
    "dev":"namespace0.2",
    "mode":"devdax",
    "map":"dev",
    "size":"35.44 GiB (38.05 GB)",
    "uuid":"89d13f41-be6c-425b-9ec7-1e2a239b5303",
    "daxregion":{
        "id":0,
        "size":"35.44 GiB (38.05 GB)",
        "align":2097152,
        "devices":[
            {
                "chardev":"dax0.2",
                "size":"35.44 GiB (38.05 GB)",
                "target_node":4,
                "mode":"devdax"
            }
        ]
    },
    "align":2097152
}

```

名前空間は **/dev/dax0.2** として利用できるようになりました。

検証

- 新しい名前空間がデバイス DAX モードで作成されたかどうかを確認します。

```

# ndctl list -RN -n namespace0.2
{
    "regions":[
        {
            "dev":"region0",
            "size":2156073582592,
            "align":16777216,
            "available_size":2065879269376,
            "max_available_extent":2065879269376,

```

```

  "type":"pmem",
  "iset_id":736272362787276936,
  "badblock_count":3,
  "persistence_domain":"memory_controller",
  "namespaces":[
    {
      "dev":"namespace0.2",
      "mode":"devdax",
      "map":"dev",
      "size":38048628736,
      "uuid":"89d13f41-be6c-425b-9ec7-1e2a239b5303",
      "chardev":"dax0.2",
      "align":2097152
    }
  ]
}

```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.8. NVDIMM でのファイルシステム DAX 名前空間の作成

システムに接続されている NVDIMM デバイスをファイルシステム DAX モードで設定して、ダイレクトアクセス機能を備えたファイルシステムをサポートします。

次のオプションを検討してください。

- ファイルシステムの DAX モードに既存の名前空間を再設定する。
- 新しいファイルシステムの DAX 名前空間を作成する (利用可能な領域がある場合)。



重要

ファイルシステムの DAX テクノロジーはテクノロジープレビューとしてのみ提供されるため、Red Hat では対応していません。

6.8.1. ファイルシステムの直接アクセスモードの NVDIMM

NVDIMM デバイスがファイルシステムダイレクトアクセス (ファイルシステム DAX、**fsdax**) モードで設定されている場合、その上にファイルシステムを作成できます。このファイルシステムのファイルで **mmap()** 操作を実行するアプリケーションは、ストレージに直接アクセスします。これにより、NVDIMM 上のプログラミングモデルに直接アクセスできます。

次の新しい **-o dax** オプションが利用できるようになりました。必要に応じて、ファイル属性を介して直接アクセスの動作を制御できます。

-o dax=inode

これは、ファイルシステムのマウント時に dax オプションを指定しない場合のデフォルトオプションです。このオプションを使用すると、ファイルに属性フラグを設定して、dax モードをアクティブにできるかどうかを制御できます。必要に応じて、個々のファイルにこのフラグを設定できます。

このフラグをディレクトリーに設定することもでき、そのディレクトリー内のすべてのファイルが同じフラグで作成されます。この属性フラグは、`xfs_io -c 'chattr +x' directory-name` コマンドを使用して設定できます。

-o **dax=never**

このオプションを使用すると、dax フラグが **inode** モードに設定されていても、dax モードは有効になりません。これは、inode ごとの dax 属性フラグが無視され、このフラグが設定されたファイルは直接アクセスが有効にならないことを意味します。

-o **dax=always**

このオプションは、古い **-o dax** の動作と同等です。このオプションを使用すると、dax 属性フラグに関係なく、ファイルシステム上の任意のファイルに対して直接アクセスモードを有効にできます。



警告

今後のリリースでは、**-o dax** がサポートされなくなる可能性があります。必要に応じて、代わりに **-o dax=always** を使用できます。このモードでは、すべてのファイルが直接アクセスモードになる可能性があります。

ページごとのメタデータ割り当て

このモードでは、システム DRAM または NVDIMM デバイス自体でページごとのメタデータを割り当てる必要があります。このデータ構造のオーバーヘッドは、4 KiB ページごとに 64 バイトです。

- 小さいデバイスでは、問題なく DRAM に収まるのに十分なオーバーヘッド量があります。たとえば、16 GiB の名前区間のページ構造に必要なのは 256 MiB だけです。NVDIMM デバイスは通常小さくて高価であるため、ページトラッキングデータ構造を DRAM に格納することが推奨されます。
- テラバイト以上のサイズの NVDIMM デバイスの場合は、ページトラッキングデータ構造の格納に必要なメモリーの量がシステム内の DRAM の量を超える可能性があります。NVDIMM の 1 TiB に対して、ページ構造だけで 16 GiB が必要です。したがって、このような場合には、NVDIMM 自体にデータ構造を保存することが推奨されます。名前空間の設定時に **--map** オプションを使用して、ページごとのメタデータを保存する場所を設定できます。
- システム RAM に割り当てるには、**--map=mem** を使用します。
- NVDIMM に割り当てるには、**--map=dev** を使用します。

6.8.2. ファイルシステム DAX モードへの既存の NVDIMM 名前空間の再設定

既存の不揮発性デュアルインラインメモリーモジュール (NVDIMM) 名前空間をファイルシステム DAX モードに再設定できます。



警告

名前空間を再設定すると、名前空間に以前に保存されたデータが削除されます。

前提条件

- **ndctl** ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。

手順

1. システムにある名前空間のリストを表示します。

```
# ndctl list --namespaces --idle
[
  {
    "dev": "namespace1.0",
    "mode": "raw",
    "size": 34359738368,
    "uuid": "ac951312-b312-4e76-9f15-6e00c8f2e6f4",
    "state": "disabled",
    "numa_node": 1
  },
  {
    "dev": "namespace0.0",
    "mode": "raw",
    "size": 38615912448,
    "uuid": "ff5a0a16-3495-4ce8-b86b-f0e3bd9d1817",
    "state": "disabled",
    "numa_node": 0
  }
]
```

2. 名前空間を再設定します。

```
# ndctl create-namespace --force --mode=fsdax --reconfig=namespace-ID
```

例6.5 ファイルシステム DAX としての名前空間の再設定

DAX に対応するファイルシステムに **namespace0.0** を使用するには、次のコマンドを使用します。

```
# ndctl create-namespace --force --mode=fsdax --reconfig=namespace0.0
{
  "dev": "namespace0.0",
  "mode": "fsdax",
  "map": "dev",
  "size": "11.81 GiB (12.68 GB)",
  "uuid": "f8153ee3-c52d-4c6e-bc1d-197f5be38483",
  "sector_size": 512,
```

```

    "align":2097152,
    "blockdev":"pmem0"
}
```

名前空間は **/dev/pmem0** パスで利用できるようになりました。

検証

- システム上の既存の名前空間が再設定されているかどうかを確認します。

```

# ndctl list --namespace namespace0.0
[
  {
    "dev":"namespace0.0",
    "mode":"fsdax",
    "map":"dev",
    "size":12681478144,
    "uuid":"f8153ee3-c52d-4c6e-bc1d-197f5be38483",
    "sector_size":512,
    "align":2097152,
    "blockdev":"pmem0"
  }
]
```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.8.3. ファイルシステム DAX モードで新しい NVDIMM 名前空間の作成

リージョンに空き容量がある場合は、Non-Volatile Dual In-line Memory Modules (NVDIMM) デバイスに新しいファイルシステム DAX 名前空間を作成できます。

前提条件

- ndctl** ユーティリティーがインストールされている。詳細は、[ndctl のインストール](#) を参照してください。
- NVDIMM デバイスは、リージョン内に複数の名前空間を作成するためのラベルをサポートしています。これは、次のコマンドを使用して確認できます。

```

# ndctl read-labels nmem0 >/dev/null
read 1 nmem
```

これは、1つの NVDIMM デバイスのラベルを読み取ったことを示しています。値が **0** の場合、デバイスがラベルをサポートしていないことを意味します。

手順

- 利用可能な領域があるシステムの **pmem** リージョンのリストを表示します。以下の例では、**region1** リージョンと **region0** リージョンの領域が利用できます。

```
# ndctl list --regions
[
  {
    "dev": "region1",
    "size": 2156073582592,
    "align": 16777216,
    "available_size": 2117418876928,
    "max_available_extent": 2117418876928,
    "type": "pmem",
    "iset_id": -9102197055295954944,
    "badblock_count": 1,
    "persistence_domain": "memory_controller"
  },
  {
    "dev": "region0",
    "size": 2156073582592,
    "align": 16777216,
    "available_size": 2143188680704,
    "max_available_extent": 2143188680704,
    "type": "pmem",
    "iset_id": 736272362787276936,
    "badblock_count": 3,
    "persistence_domain": "memory_controller"
  }
]
```

- 利用可能な領域のいずれかに、1つ以上の名前空間を割り当てます。

```
# ndctl create-namespace --mode=fsdax --region=regionN --size=namespace-size
```

例6.6 リージョンへの名前空間の作成

次のコマンドは、region0 で 36 GiB のファイルシステム DAX 名前空間を作成します。

```
# ndctl create-namespace --mode=fsdax --region=region0 --size=36G
{
  "dev": "namespace0.3",
  "mode": "fsdax",
  "map": "dev",
  "size": "35.44 GiB (38.05 GB)",
  "uuid": "99e77865-42eb-4b82-9db6-c6bc9b3959c2",
  "sector_size": 512,
  "align": 2097152,
  "blockdev": "pmem0.3"
}
```

名前空間は **/dev/pmem0.3** として利用できるようになりました。

検証

- 新しい名前空間がファイルシステム DAX モードで作成されたかどうかを確認します。

```
# ndctl list -RN -n namespace0.3
```

```
{
  "regions": [
    {
      "dev": "region0",
      "size": 2156073582592,
      "align": 16777216,
      "available_size": 2027224563712,
      "max_available_extent": 2027224563712,
      "type": "pmem",
      "iset_id": 736272362787276936,
      "badblock_count": 3,
      "persistence_domain": "memory_controller",
      "namespaces": [
        {
          "dev": "namespace0.3",
          "mode": "fsdax",
          "map": "dev",
          "size": 38048628736,
          "uuid": "99e77865-42eb-4b82-9db6-c6bc9b3959c2",
          "sector_size": 512,
          "align": 2097152,
          "blockdev": "pmem0.3"
        }
      ]
    }
  ]
}
```

関連情報

- システム上の **ndctl-create-namespace(1)** man ページ

6.8.4. ファイルシステム DAX デバイスでのファイルシステムの作成

ファイルシステム DAX デバイス上にファイルシステムを作成し、ファイルシステムをマウントできます。ファイルシステムを作成した後、アプリケーションは永続メモリーを使用して **mount-point** ディレクトリーにファイルを作成し、ファイルを開き、**mmap** 操作を使用して直接アクセスできるようにファイルをマップできます。

Red Hat Enterprise Linux 8 では、テクノロジープレビューとして、XFS および ext4 ファイルシステムの両方を NVDIMM にできます。

手順

- オプション: ファイルシステム DAX デバイス上にパーティションを作成します。詳細は、[parted を使用したパーティションの作成](#) を参照してください。



注記

fsdax デバイスにパーティションを作成する場合、パーティションはページの境界に調整する必要があります。Intel 64 アーキテクチャーおよび AMD64 アーキテクチャーでは、パーティションの開始と終了に最低 4 KiB のアライメントが必要です。2 MiB が優先されるアライメントです。

parted ツールは、デフォルトでは 1 MiB の境界にパーティションをそろえます。最初のパーティションには、パーティションの開始部分として 2 MiB を指定します。パーティションのサイズが 2 MiB の倍数である場合は、他のすべてのパーティションもそろえられます。

- パーティションまたは NVDIMM デバイスに XFS または ext4 ファイルシステムを作成します。

```
# mkfs.xfs -d su=2m,sw=1 fsdax-partition-or-device
```



注記

dax 対応ファイルと reflinked ファイルは、ファイルシステム上で共存できるようになりました。ただし、個々のファイルの場合、dax と reflink は相互に排他的です。

XFS の場合、dax マウントオプションと互換性がないため、共有コピーオンライトのデータエクステントを無効にします。また、大規模ページのマッピングの可能性を高めるために、ストライプユニットとストライプ幅を設定してください。

- ファイルシステムをマウントします。

```
# mount f_sdax-partition-or-device mount-point_
```

直接アクセスモードを有効にするために dax オプションを使用してファイルシステムをマウントする必要はありません。マウント時に dax オプションを指定しない場合、ファイルシステムは **dax=inode** モードになります。直接アクセスモードをアクティブにする前に、ファイルに dax オプションを設定します。

関連情報

- システム上の **mkfs.xfs(8)** man ページ
- ファイルシステムの直接アクセスモードの [NVDIMM](#)

6.9. S.M.A.R.T. を使用した NVDIMM 正常性 (ヘルス) の監視

一部の不揮発性デュアルインラインメモリーモジュール (NVDIMM) デバイスは、ヘルス情報を取得するためのセルフモニタリング、分析、レポートテクノロジー (S.M.A.R.T.) インターフェイスをサポートしています。



重要

NVDIMM 正常性を定期的に監視して、データの損失を防ぎます。S.M.A.R.T. が NVDIMM デバイスのヘルステータスに関する問題を報告した場合は、[Detecting and replacing a broken NVDIMM device](#) の説明に従って交換します。

前提条件

- オプション: 一部のシステムでは、次のコマンドを使用して **acpi_ipmi** ドライバーをアップロードし、ヘルス情報を取得します。

```
# modprobe acpi_ipmi
```

手順

- ヘルス情報にアクセスします。

```
# ndctl list --dimms --health
[
  {
    "dev": "nmem1",
    "id": "8089-a2-1834-00001f13",
    "handle": 17,
    "phys_id": 32,
    "security": "disabled",
    "health": {
      "health_state": "ok",
      "temperature_celsius": 36.0,
      "controller_temperature_celsius": 37.0,
      "spares_percentage": 100,
      "alarm_temperature": false,
      "alarm_controller_temperature": false,
      "alarm_spares": false,
      "alarm_enabled_media_temperature": true,
      "temperature_threshold": 82.0,
      "alarm_enabled_ctrl_temperature": true,
      "controller_temperature_threshold": 98.0,
      "alarm_enabled_spares": true,
      "spares_threshold": 50,
      "shutdown_state": "clean",
      "shutdown_count": 4
    }
  },
  [...]
]
```

関連情報

- システム上の **ndctl-list(1)** man ページ

6.10. 破損した NVDIMM デバイスの検出と交換

不揮発性デュアルインラインメモリーモジュール (NVDIMM) に関するエラーメッセージがシステムログまたは S.M.A.R.T. によって報告されている場合は、NVDIMM デバイスに障害が発生している可能性があります。この場合は、以下を行なう必要があります。

1. NVDIMM デバイスがエラーしていることを検出
2. そこに格納されているデータをバックアップ

3. デバイスを物理的に交換

手順

1. 壊れたデバイスを検出します。

```
# ndctl list --dimms --regions --health
{
  "dimms": [
    {
      "dev": "nmem1",
      "id": "8089-a2-1834-00001f13",
      "handle": 17,
      "phys_id": 32,
      "security": "disabled",
      "health": {
        "health_state": "ok",
        "temperature_celsius": 35.0,
        [...]
      }
    [...]
  }
}
```

2. 破損した NVDIMM の **phys_id** 属性を見つけます。

```
# ndctl list --dimms --human
```

前述の例では、**nmem0** が破損した NVDIMM になります。したがって、**nmem0** の **phys_id** 属性を確認します。

例6.7 NVDIMMs の **phys_id** 属性

以下の例では、**phys_id** は **0x10** です。

```
# ndctl list --dimms --human
[
  {
    "dev": "nmem1",
    "id": "XXXX-XX-XXXX-XXXXXXX",
    "handle": "0x120",
    "phys_id": "0x1c"
  },
  {
    "dev": "nmem0",
    "id": "XXXX-XX-XXXX-XXXXXXX",
    "handle": "0x20",
    "phys_id": "0x10",
    "flag_failed_flush": true,
    "flag_smart_event": true
  }
]
```

3. 壊れた NVDIMM のメモリースロットを見つけます。

```
# dmidecode
```

出力において、Handle 識別子が、破損した NVDIMM の **phys_id** 属性と一致するエントリーを確認します。Locator フィールドは、破損した NVDIMM が使用するメモリースロットの一覧を表示します。

例6.8 NVDIMM メモリースロットリスト

以下の例では、**nmem0** デバイスが **0x0010** の識別子に一致し、**DIMM-XXX-YYYY** メモリースロットを使用します。

```
# dmidecode
...
Handle 0x0010, DMI type 17, 40 bytes
Memory Device
  Array Handle: 0x0004
  Error Information Handle: Not Provided
  Total Width: 72 bits
  Data Width: 64 bits
  Size: 125 GB
  Form Factor: DIMM
  Set: 1
  Locator: DIMM-XXX-YYYY
  Bank Locator: Bank0
  Type: Other
  Type Detail: Non-Volatile Registered (Buffered)
...
```

4. NVDIMM 上の名前空間にある全データのバックアップを作成します。NVDIMM を交換する前にデータのバックアップを作成しないと、システムから NVDIMM を削除したときにデータが失われます。



警告

時折、NVDIMM が完全に破損すると、バックアップが失敗することがあります。

これを防ぐには、[Monitoring NVDIMM health using S.M.A.R.T.](#) で説明されているように、S.M.A.R.T. を使用して NVDIMM デバイスを定期的に監視し、故障した NVDIMM を破損する前に交換します。

5. NVDIMM の名前空間を一覧表示します。

```
# ndctl list --namespaces --dimm=DIMM-ID-number
```

例6.9 NVDIMM 名前空間のリスト表示

以下の例では、**nmem0** デバイスには、バックアップが必要な名前空間の **namespace0.0** と **namespace0.2** が含まれます。

```
# ndctl list --namespaces --dimm=0

[
  {
    "dev": "namespace0.2",
    "mode": "sector",
    "size": 67042312192,
    "uuid": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size": 4096,
    "blockdev": "pmem0.2s",
    "numa_node": 0
  },
  {
    "dev": "namespace0.0",
    "mode": "sector",
    "size": 67042312192,
    "uuid": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "raw_uuid": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "sector_size": 4096,
    "blockdev": "pmem0s",
    "numa_node": 0
  }
]
```

6. 破損した NVDIMM を物理的に交換します。

関連情報

- システム上の **ndctl-list(1)** および **dmidecode(8)** man ページ

第7章 未使用ブロックの破棄

破棄操作に対応するブロックデバイスで破棄操作を実行するか、そのスケジュールを設定できます。ブロック破棄操作では、マウントされたファイルシステムによって使用されなくなったファイルシステムブロックを、基盤となるストレージに伝達します。ブロック破棄操作により、SSD はガベージコレクションルーチンを最適化でき、シンプロビジョニングされたストレージに未使用の物理ブロックを再利用するように通知できます。

要件

- ファイルシステムの基礎となるブロックデバイスは、物理的な破棄操作に対応している必要があります。
- `/sys/block/<device>/queue/discard_max_bytes` ファイルの値がゼロではない場合は、物理的な破棄操作はサポートされます。

7.1. ブロック破棄操作のタイプ

以下のような、さまざまな方法で破棄操作を実行できます。

バッチ破棄

これは、ユーザーによって明示的にトリガーされ、選択したファイルシステム内の未使用のブロックをすべて破棄します。

オンライン破棄

これは、マウント時に指定され、ユーザーの介入なしにリアルタイムでトリガーされます。オンライン破棄操作は、`used` から `free` 状態に移行中のブロックのみを破棄します。

定期的な破棄

`systemd` サービスが定期的に実行するバッチ操作です。

すべてのタイプは、XFS ファイルシステムおよび ext4 ファイルシステムでサポートされます。

推奨事項

Red Hat は、バッチ破棄または周期破棄を使用することを推奨します。

以下の場合にのみ、オンライン破棄を使用してください。

- システムのワークロードでバッチ破棄が実行できない場合
- パフォーマンス維持にオンライン破棄操作が必要な場合

7.2. バッチブロック破棄の実行

バッチブロック破棄操作を実行して、マウントされたファイルシステムの未使用ブロックを破棄することができます。

前提条件

- ファイルシステムがマウントされている。
- ファイルシステムの基礎となるブロックデバイスが物理的な破棄操作に対応している。

手順

- `fstrim` ユーティリティーを使用します。

- 選択したファイルシステムでのみ破棄を実行するには、次のコマンドを使用します。

```
# fstrim mount-point
```

- マウントされているすべてのファイルシステムで破棄を実行するには、次のコマンドを使用します。

```
# fstrim --all
```

fstrim コマンドを以下のいずれかで実行している場合は、

- 破棄操作に対応していないデバイス
- 複数のデバイスから構成され、そのデバイスの1つが破棄操作に対応していない論理デバイス (LVM または MD)

次のメッセージが表示されます。

```
# fstrim /mnt/non_discard
fstrim: /mnt/non_discard: the discard operation is not supported
```

関連情報

- システム上の **fstrim(8)** man ページ

7.3. オンラインブロック破棄の有効化

オンラインブロック破棄操作を実行して、サポートしているすべてのファイルシステムで未使用のブロックを自動的に破棄できます。

手順

- マウント時のオンライン破棄を有効にします。

- ファイルシステムを手動でマウントするには、**-o discard** マウントオプションを追加します。

```
# mount -o discard device mount-point
```

- ファイルシステムを永続的にマウントするには、**/etc/fstab** ファイルのマウントエントリーに **discard** オプションを追加します。

関連情報

- システム上の **mount(8)** および **fstab(5)** man ページ

7.4. STORAGE RHEL システムロールを使用してオンラインブロック破棄を有効にする

オンラインブロック破棄オプションを使用すると、XFS ファイルシステムをマウントし、未使用のブロックを自動的に破棄できます。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: `~/playbook.yml`) を作成します。

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Enable online block discard
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
    vars:
      storage_volumes:
        - name: barefs
          type: disk
          disks:
            - sdb
          fs_type: xfs
          mount_point: /mnt/data
          mount_options: discard
```

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.storage/README.md` ファイルを参照してください。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- オンラインブロック破棄オプションが有効になっていることを確認します。

```
# ansible managed-node-01.example.com -m command -a 'findmnt /mnt/data'
```

関連情報

- `/usr/share/ansible/roles/rhel-system-roles.storage/README.md` ファイル

- `/usr/share/doc/rhel-system-roles/storage/` ディレクトリー

7.5. 定期的なブロック破棄の有効化

systemd タイマーを有効にして、サポートしているすべてのファイルシステムで未使用ブロックを定期的に破棄できます。

手順

- **systemd** タイマーを有効にして起動します。

```
# systemctl enable --now fstrim.timer
Created symlink /etc/systemd/system/timers.target.wants/fstrim.timer →
/usr/lib/systemd/system/fstrim.timer.
```

検証

- タイマーのステータスを確認します。

```
# systemctl status fstrim.timer
fstrim.timer - Discard unused blocks once a week
   Loaded: loaded (/usr/lib/systemd/system/fstrim.timer; enabled; vendor preset: disabled)
   Active: active (waiting) since Wed 2023-05-17 13:24:41 CEST; 3min 15s ago
     Trigger: Mon 2023-05-22 01:20:46 CEST; 4 days left
   Docs: man:fstrim

May 17 13:24:41 localhost.localdomain systemd[1]: Started Discard unused blocks once a
week.
```

第8章 iSCSI ターゲットの設定

Red Hat Enterprise Linux では、コマンドラインインターフェイスとして **targetcli** シェルを使用し、以下の操作を行います。

- iSCSI ハードウェアを使用できるように iSCSI ストレージ相互接続を追加、削除、表示、監視します。
- ファイル、ボリューム、ローカル SCSI デバイス、またはリモートシステムへの RAM ディスクで対応しているローカルストレージリソースをエクスポートします。

targetcli ツールには、組み込みタブ補完、自動補完サポート、インラインドキュメントなどのツリーベースのレイアウトがあります。

8.1. TARGETCLI のインストール

targetcli ツールをインストールして、iSCSI ストレージの相互接続を追加、監視、削除します。

手順

1. **targetcli** ツールをインストールします。

```
# yum install targetcli
```

2. ターゲットサービスを起動します。

```
# systemctl start target
```

3. システムの起動時にターゲットサービスが起動するように設定するには、次のコマンドを実行します。

```
# systemctl enable target
```

4. ファイアウォールの **3260** ポートを開き、ファイアウォール設定を再読み込みします。

```
# firewall-cmd --permanent --add-port=3260/tcp
Success
```

```
# firewall-cmd --reload
Success
```

検証

- **targetcli** レイアウトを表示します。

```
# targetcli
/> ls
o- /.....[...]
o- backstores.....[...]
| o- block.....[Storage Objects: 0]
| o- fileio.....[Storage Objects: 0]
| o- pscsi.....[Storage Objects: 0]
```

```
| o- ramdisk.....[Storage Objects: 0]
o- iscsi.....[Targets: 0]
o- loopback.....[Targets: 0]
```

関連情報

- システム上の **targetcli(8)** man ページ

8.2. iSCSI ターゲットの作成

iSCSI ターゲットを作成すると、クライアントの iSCSI イニシエーターがサーバー上のストレージデバイスにアクセスできるようになります。ターゲットとイニシエーターにはどちらも一意の識別名があります。

前提条件

- targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。

手順

- iSCSI ディレクトリーに移動します。 **cd** コマンドを使用して iSCSI ディレクトリーに移動することもできます。

```
/> iscsi/
```

- iSCSI ターゲットを作成するには、以下のいずれかのオプションを使用します。

- デフォルトのターゲット名を使用した iSCSI ターゲットの作成:

```
/iscsi> create

Created target
iqn.2003-01.org.linux-iscsi.hostname.x8664:sn.78b473f296ff
Created TPG1
```

- 特定の名前を使用した iSCSI ターゲットの作成:

```
/iscsi> create iqn.2006-04.com.example:444

Created target iqn.2006-04.com.example:444
Created TPG1
Here iqn.2006-04.com.example:444 is target_iqn_name
```

iqn.2006-04.com.example:444 を、特定のターゲット名に置き換えます。

- 新たに作成されたターゲットを確認します。

```
/iscsi> ls

o- iscsi.....[1 Target]
o- iqn.2006-04.com.example:444.....[1 TPG]
o- tpg1.....[enabled, auth]
o- acls.....[0 ACL]
```

- o- luns.....[0 LUN]
- o- portals.....[0 Portal]

関連情報

- システム上の **targetcli(8)** man ページ

8.3. iSCSI バックストア

iSCSI バックストアは、エクスポートした LUN のデータをローカルマシンに保存するさまざまな方法に対応します。ストレージオブジェクトを作成して、バックストアが使用するリソースを定義します。

管理者は、LIO (Linux-IO) が対応する以下のバックストアデバイスのいずれかを選択できます。

fileio バックストア

ローカルファイルシステム上の通常のファイルをディスクイメージとして使用する場合は、**fileio**ストレージオブジェクトを作成します。**fileio** バックストアの作成は、[fileio ストレージオブジェクトの作成](#)を参照してください。

block バックストア

ローカルのブロックデバイスおよび論理デバイスを使用している場合には、ブロックストレージオブジェクトを作成します。**block** バックストアの作成は、[ブロックストレージオブジェクトの作成](#)を参照してください。

pscsi バックストア

ストレージオブジェクトが SCSI コマンドの直接パススルーに対応している場合は、**pscsi**ストレージオブジェクトを作成します。**pscsi** バックストアの作成は、[pscsi ストレージオブジェクトの作成](#)を参照してください。

ramdisk バックストア

一時的な RAM 対応デバイスを作成する場合は、**ramdisk**ストレージオブジェクトを作成します。**ramdisk** バックストアの作成は、[メモリーコピーの RAM ディスクストレージオブジェクトの作成](#)を参照してください。

関連情報

- システム上の **targetcli(8)** man ページ

8.4. FILEIO ストレージオブジェクトの作成

fileio ストレージオブジェクトは、**write_back** または **write_thru** 操作のいずれかをサポートできます。**write_back** 操作では、ローカルファイルシステムキャッシュが有効になります。これにより、パフォーマンスが向上しますが、データの損失のリスクが高まります。

write_thru 操作を優先させるために、**write_back=false** を使用して **write_back** 操作を無効にすることが推奨されます。

前提条件

- **targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#)を参照してください。

手順

1. **backstores**/ ディレクトリーから **fileio**/ に移動します。

```
/> backstores/fileio
```

2. **fileio** ストレージオブジェクトを作成します。

```
/backstores/fileio> create file1 /tmp/disk1.img 200M write_back=false
```

```
Created fileio file1 with size 209715200
```

検証

- 作成された **fileio** ストレージオブジェクトを確認します。

```
/backstores/fileio> ls
```

関連情報

- システム上の **targetcli(8)** man ページ

8.5. ブロックストレージオブジェクトの作成

ブロックドライバーを使用すると、**/sys/block/** ディレクトリーにあるブロックデバイスを LIO (Linux-IO) で使用できます。これには、HDD、SSD、CD、DVD などの物理デバイス、およびソフトウェアやハードウェアの RAID ボリューム、LVM ボリュームなどの論理デバイスが含まれます。

前提条件

- **targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。

手順

1. **backstores**/ ディレクトリーから **block**/ に移動します。

```
/> backstores/block/
```

2. **block** バックストアを作成します。

```
/backstores/block> create name=block_backend dev=/dev/sdb
```

```
Generating a wwn serial.
```

```
Created block storage object block_backend using /dev/sdb.
```

検証

- 作成された **block** ストレージオブジェクトを確認します。

```
/backstores/block> ls
```

関連情報

- システム上の **targetcli(8)** man ページ

8.6. PSCSI ストレージオブジェクトの作成

SCSI エミュレーションがない場合、かつ **/proc/scsi/scsi** に **lsscsi** で表示される基盤の SCSI デバイスが存在する場合には、SCSI コマンドの直接パスルートをサポートするストレージオブジェクトであれば、どれでもバックストアとして設定できます。たとえば、SAS ハードドライブなどが該当します。このサブシステムでは、SCSI-3 以降に対応しています。

警告



pscsci は、上級ユーザーのみが使用してください。非対称論理ユニット割り当て (ALUA) や永続予約 (VMware ESX や vSphere で使用される永続予約など) は、通常はデバイスのファームウェアに実装されず、誤作動やクラッシュが発生する原因となることがあります。確信が持てない場合は、実稼働の設定に **block** バックストアを使用してください。

前提条件

- targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。

手順

- backstores/** ディレクトリーから **pscsci/** に移動します。

```
/> backstores/pscsci/
```

- この例では、**/dev/sr0** を使用して物理 SCSI デバイスである TYPE_ROM デバイスの **pscsci** バックストアを作成します。

```
/backstores/pscsci> create name=pscsci_backend dev=/dev/sr0
```

```
Generating a wwn serial.  
Created pscsi storage object pscsi_backend using /dev/sr0
```

検証

- 作成した **pscsci** ストレージオブジェクトを確認します。

```
/backstores/pscsci> ls
```

関連情報

- システム上の **targetcli(8)** man ページ

8.7. メモリーコピーの RAM ディスクストレージオブジェクトの作成

メモリーコピー RAM ディスク (**ramdisk**) は、完全な SCSI エミュレーションと、イニシエーターのメモリーコピーを使用した個別のメモリーマッピングが含まれる RAM ディスクを提供します。これにより、マルチセッションの機能を利用できます。これは、特に実稼働環境での高速で不揮発性の大容量ストレージで有用です。

前提条件

- **targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。

手順

1. **backstores**/ ディレクトリーから **ramdisk/** に移動します。

```
/> backstores/ramdisk/
```

2. 1GB RAM ディスクバックストアを作成します。

```
/backstores/ramdisk> create name=rd_backend size=1GB
```

```
Generating a wwn serial.  
Created rd_mcp ramdisk rd_backend with size 1GB.
```

検証

- 作成した **ramdisk** ストレージオブジェクトを確認します。

```
/backstores/ramdisk> ls
```

関連情報

- システム上の **targetcli(8)** man ページ

8.8. iSCSI ポータルの作成

iSCSI ポータルを作成できます。これにより、IP アドレスとポートがターゲットに追加され、ターゲットが有効な状態に維持されます。

前提条件

- **targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。
- ターゲットポータルグループ (TPG) に関連付けられた iSCSI ターゲット。詳細は、[iSCSI ターゲットの作成](#) を参照してください。

手順

1. TPG ディレクトリーに移動します。

```
/iscsi> iqn.2006-04.com.example:444/tpg1/
```

2. iSCSI ポータルを作成するには、以下のいずれかのオプションを使用します。

- a. デフォルトポータルを作成するには、デフォルトの iSCSI ポート **3260** を使用し、ターゲットがそのポートのすべての IP アドレスをリッスンできるようにします。

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create
```

```
Using default IP port 3260
Binding to INADDR_Any (0.0.0.0)
Created network portal 0.0.0.0:3260
```

- b. 特定の IP アドレスを使用したポータルの作成:

```
/iscsi/iqn.20...mple:444/tpg1> portals/ create 192.168.122.137
```

```
Using default IP port 3260
Created network portal 192.168.122.137:3260
```

検証

- 新たに作成されたポータルを確認します。

```
/iscsi/iqn.20...mple:444/tpg1> ls
```

```

o- tpg..... [enabled, auth]
o- acls ..... [0 ACL]
o- luns ..... [0 LUN]
o- portals ..... [1 Portal]
o- 192.168.122.137:3260..... [OK]

```

関連情報

- システム上の **targetcli(8)** man ページ

8.9. iSCSI LUN の作成

論理ユニット番号 (LUN) は、iSCSI バックストアで対応している物理デバイスです。各 LUN には固有の番号があります。

前提条件

- **targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。
- ターゲットポータルグループ (TPG) に関する情報。詳細は、[iSCSI ターゲットの作成](#) を参照してください。
- 作成したストレージオブジェクト。詳細は、[iSCSI バックストア](#) を参照してください。

手順

1. 作成したストレージオブジェクトの LUN を作成します。

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/ramdisk/rd_backend
Created LUN 0.
```

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/block/block_backend
Created LUN 1.
```

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/fileio/file1
Created LUN 2.
```

2. 作成した LUN を確認します。

```
/iscsi/iqn.20...mple:444/tpg1> ls

o- tpg..... [enabled, auth]
  o- acls ..... [0 ACL]
  o- luns ..... [3 LUNs]
    | o- lun0..... [ramdisk/ramdisk1]
    | o- lun1..... [block/block1 (/dev/vdb1)]
    | o- lun2..... [fileio/file1 (/foo.img)]
  o- portals ..... [1 Portal]
    o- 192.168.122.137:3260..... [OK]
```

デフォルトの LUN 名は 0 から始まります。



重要

デフォルトでは、読み書きパーミッションを持つ LUN が作成されます。ACL の作成後に新しい LUN が追加されると、LUN は自動的に利用可能なすべての ACL にマッピングされ、セキュリティー上のリスクが発生します。読み取り専用権限を持つ LUN の作成については、[Creating a read-only iSCSI LUN](#) を参照してください。

3. ACL を設定します。詳細は、[iSCSI ACL の作成](#) を参照してください。

関連情報

- システム上の **targetcli(8)** man ページ

8.10. 読み取り専用の iSCSI LUN の作成

デフォルトでは、読み書きパーミッションを持つ LUN が作成されます。読み取り専用の LUN を作成できます。

前提条件

- targetcli** をインストールして、実行している。詳細は、[targetcli のインストール](#) を参照してください。
- ターゲットポータルグループ (TPG) に関連付けられた iSCSI ターゲット。詳細は、[iSCSI ターゲットの作成](#) を参照してください。
- 作成したストレージオブジェクト。詳細は、[iSCSI バックストア](#) を参照してください。

ナビ

- 読み取り専用パーミッションを設定します。

```
/> set global auto_add_mapped_luns=false
```

```
Parameter auto_add_mapped_luns is now 'false'.
```

これにより、LUN が既存の ACL へ自動的にマッピングされないようになり、LUN を手動でマッピングできるようになります。

- initiator_iqn_name ディレクトリーに移動します。

```
/> /iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name/
```

- LUN を作成します。

```
/iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name> create
mapped_lun=next_sequential_LUN_number tpg_lun_or_backstore=backstore
write_protect=1
```

たとえば、以下のようになります。

```
/iscsi/target_iqn_name/tpg1/acls/2006-04.com.example:888> create mapped_lun=1
tpg_lun_or_backstore=/backstores/block/block2 write_protect=1
```

```
Created LUN 1.
Created Mapped LUN 1.
```

- 作成した LUN を確認します。

```
/iscsi/target_iqn_name/tpg1/acls/2006-04.com.example:888> ls
o- 2006-04.com.example:888 .. [Mapped LUNs: 2]
| o- mapped_lun0 ..... [lun0 block/disk1 (rw)]
| o- mapped_lun1 ..... [lun1 block/disk2 (ro)]
```

(mapped_lun0 の (rw) とは異なり) mapped_lun1 行の最後に (ro) が表示されますが、これは、読み取り専用であることを表しています。

- ACL を設定します。詳細は、[iSCSI ACL の作成](#) を参照してください。

関連情報

- システム上の **targetcli(8)** man ページ

8.11. iSCSI ACL の作成

targetcli サービスは、アクセスコントロールリスト (ACL) を使用してアクセスルールを定義し、各イニシエーターに論理ユニット番号 (LUN) へのアクセスを許可します。

ターゲットとイニシエーターにはどちらも一意の識別名があります。ACL を設定するには、イニシエーターの一意の名前を知っている必要があります。**iscsi-initiator-utils** パッケージによって提供される **/etc/iscsi/initiatorname.iscsi** ファイルには、iSCSI イニシエーター名が含まれています。

前提条件

- **targetcli** サービスが [インストール](#) され、実行されている。
- ターゲットポータルグループ (TPG) に関連付けられた [iSCSI ターゲット](#)。

手順

1. オプション: ACL への LUN の自動マッピングを無効にするには、[読み取り専用の iSCSI LUN の作成](#) を参照してください。
2. acls ディレクトリーへ移動します。

```
/> iscsi/target_iqn_name/tpg_name/acls/
```

3. ACL を作成するには、以下のいずれかのオプションを使用します。
 - イニシエーターの `/etc/iscsi/initiatorname.iscsi` ファイルの `initiator_iqn_name` を使用します。

```
iscsi/target_iqn_name/tpg_name/acls> create initiator_iqn_name
Created Node ACL for initiator_iqn_name
Created mapped LUN 2.
Created mapped LUN 1.
Created mapped LUN 0.
```

- `Custom_name` を使用し、それに一致するようにイニシエーターを更新します。

```
iscsi/target_iqn_name/tpg_name/acls> create custom_name
Created Node ACL for custom_name
Created mapped LUN 2.
Created mapped LUN 1.
Created mapped LUN 0.
```

イニシエーター名の更新については、[iSCSI イニシエーターの作成](#) を参照してください。

検証

- 作成した ACL を確認します。

```
iscsi/target_iqn_name/tpg_name/acls> ls
o- acls .....[1 ACL]
  o- target_iqn_name ....[3 Mapped LUNs, auth]
    o- mapped_lun0 .....[lun0 ramdisk/ramdisk1 (rw)]
    o- mapped_lun1 .....[lun1 block/block1 (rw)]
    o- mapped_lun2 .....[lun2 fileio/file1 (rw)]
```

関連情報

- システム上の `targetcli(8)` man ページ

8.12. ターゲットのチャレンジハンドシェイク認証プロトコルの設定

Challenge-Handshake Authentication Protocol (CHAP) を使用すると、パスワードでターゲットを保護できます。イニシエーターは、このパスワードでターゲットに接続できることを認識している必要があります。

前提条件

- iSCSI ACL を作成している。詳細は、[iSCSI ACL の作成](#) を参照してください。

手順

1. 属性認証を設定します。

```
/iscsi/iqn.20...mple:444/tpg1> set attribute authentication=1
```

```
Parameter authentication is now '1'.
```

2. **userid** と **password** を設定します。

```
/tpg1> set auth userid=redhat
```

```
Parameter userid is now 'redhat'.
```

```
/iscsi/iqn.20...689dcbb3/tpg1> set auth password=redhat_passwd
```

```
Parameter password is now 'redhat_passwd'.
```

3. **acls** ディレクトリーへ移動します。

```
/> iscsi/target_iqn_name/tpg1/acls/initiator_iqn_name/
```

4. 属性認証を設定します。

```
/iscsi/iqn.20...:605fcc6a48be> set attribute authentication=1
```

```
Parameter authentication is now '1'.
```

5. **userid** と **password** を設定します。

```
/iscsi/iqn.20...:605fcc6a48be> set auth userid=redhat
```

```
Parameter userid is now 'redhat'.
```

```
/iscsi/iqn.20...:605fcc6a48be> set auth password=redhat_passwd
```

```
Parameter password is now 'redhat_passwd'.
```

関連情報

- システム上の **targetcli(8)** man ページ

8.13. TARGETCLI ツールで iSCSI オブジェクトの削除

targetcli ツールを使用して iSCSI オブジェクトを削除できます。

手順

1. ターゲットからログオフします。

```
# iscsiadadm -m node -T iqn.2006-04.com.example:444 -u
```

ターゲットへのログイン方法は、[iSCSI イニシエーターの作成](#) を参照してください。

2. ACL、LUN、およびポータルのすべてを含め、ターゲット全体を削除します。

```
/> iscsi/ delete iqn.2006-04.com.example:444
```

iqn.2006-04.com.example:444 を target_iqn_name に置き換えます。

- iSCSI バックストアを削除するには、次のコマンドを実行します。

```
/> backstores/backstore-type/ delete block_backend
```

backstore-type を fileio、block、pscsci、または ramdisk に置き換えます。

block_backend を、削除する バックストア名 に置き換えます。

- ACL などの iSCSI ターゲットの一部を削除するには、次のコマンドを実行します。

```
/> /iscsi/iqn-name/tpg/acls/ delete iqn.2006-04.com.example:444
```

検証

- 変更を表示します。

```
/> iscsi/ ls
```

関連情報

- システム上の **targetcli(8)** man ページ

第9章 iSCSI イニシエーターの設定

iSCSI イニシエーターは iSCSI ターゲットに接続するセッションを形成します。デフォルトでは、iSCSI サービスは起動に時間がかかり、**iscsiadm** コマンドの実行後にサービスが起動します。root が iSCSI デバイスがない場合や、**node.startup = automatic** でマークされたノードがない場合は、**iscsiadm** コマンドが実行するまで iSCSI サービスが起動しなくなります。これには、カーネルモジュール **iscsid** または **iscsi** の起動が必要になります。

iscsid サービスを強制的に実行し、iSCSI カーネルモジュールをロードするには、root として **systemctl start iscsid** コマンドを実行します。

9.1. iSCSI イニシエーターの作成

サーバー上のストレージデバイスにアクセスするために、iSCSI ターゲットに接続するための iSCSI イニシエーターを作成します。

前提条件

- iSCSI ターゲットのホスト名と IP アドレスがあります。
 - 外部ソフトウェアが作成したストレージターゲットに接続している場合は、ストレージ管理者からターゲットのホスト名と IP アドレスを取得します。
 - iSCSI ターゲットを作成する場合は、[iSCSI ターゲットの作成](#) を参照してください。

手順

1. クライアントマシンに **iscsi-initiator-utils** をインストールします。

```
# yum install iscsi-initiator-utils
```

2. **iscsid** サービスを再起動します。

```
# systemctl start iscsid
```

3. イニシエーター名を確認します。

```
# cat /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2006-04.com.example:888
```

4. [iSCSI ACL の作成](#) で ACL にカスタム名を指定した場合は、ACL と一致するようにイニシエーター名を更新します。

- a. **/etc/iscsi/initiatorname.iscsi** ファイルを開き、イニシエーター名を変更します。

```
# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=custom-name
```

- b. **iscsid** サービスを再起動します。

```
# systemctl restart iscsid
```

5. ターゲットを検出し、表示されたターゲット IQN でターゲットにログインします。

```
# iscsiadadm -m discovery -t st -p 10.64.24.179
10.64.24.179:3260,1 iqn.2006-04.com.example:444

# iscsiadadm -m node -T iqn.2006-04.com.example:444 -
Logging in to [iface: default, target: iqn.2006-04.com.example:444, portal:
10.64.24.179,3260] (multiple)
Login to [iface: default, target: iqn.2006-04.com.example:444, portal: 10.64.24.179,3260]
successful.
```

10.64.24.179 を、target-ip-address に置き換えます。

この手順では、[iSCSI ACL の作成](#) で説明されているように、それぞれのイニシエーター名が ACL に追加されている場合は、同じターゲットに接続されている任意の数のイニシエーターに対してこの手順を使用できます。

6. iSCSI ディスク名を確認して、この iSCSI ディスクにファイルシステムを作成します。

```
# grep "Attached SCSI" /var/log/messages

# mkfs.ext4 /dev/disk_name
```

disk_name を、[/var/log/messages](#) ファイルに記載されている iSCSI ディスク名に置き換えます。

7. ファイルシステムをマウントします。

```
# mkdir /mount/point

# mount /dev/disk_name /mount/point
```

/mount/point を、パーティションのマウントポイントに置き換えます。

8. システムの起動時にファイルシステムを自動的にマウントするように [/etc/fstab](#) を編集します。

```
# vi /etc/fstab

/dev/disk_name /mount/point ext4 _netdev 0 0
```

disk_name を iSCSI ディスク名に置き換え、/mount/point を、パーティションのマウントポイントに置き換えます。

関連情報

- システム上の [targetcli\(8\)](#) および [iscsiadm\(8\)](#) man ページ

9.2. イニシエーター用のチャレンジハンドシェイク認証プロトコルの設定

Challenge-Handshake Authentication Protocol (CHAP) を使用すると、パスワードでターゲットを保護できます。イニシエーターは、このパスワードでターゲットに接続できることを認識している必要があります。

前提条件

- iSCSI イニシエーターを作成しました。 詳細は、[iSCSI イニシエーターの作成](#) を参照してください。
- ターゲットの **CHAP** を設定します。 詳細は、[ターゲットのチャレンジハンドシェイク認証プロトコルの設定](#) を参照してください。

手順

1. **iscsid.conf** ファイルで CHAP 認証を有効にします。

```
# vi /etc/iscsi/iscsid.conf
node.session.auth.authmethod = CHAP
```

デフォルトでは、**node.session.auth.authmethod** は **None** に設定されています。

2. ターゲットの **username** と **password** を **iscsid.conf** ファイルに追加します。

```
node.session.auth.username = redhat
node.session.auth.password = redhat_passwd
```

3. **iscsid** サービスを再起動します。

```
# systemctl restart iscsid
```

関連情報

- システム上の **iscsiadm(8)** man ページ

9.3. ISCSIADM ユーティリティーを使用して iSCSI セッションを監視する

iscsiadm ユーティリティーを使用して iSCSI セッションを監視できます。

デフォルトでは、iSCSI サービスは起動に時間がかかり、**iscsiadm** コマンドの実行後にサービスが起動します。root が iSCSI デバイスにない場合や、**node.startup = automatic** でマークされたノードがない場合は、**iscsiadm** コマンドが実行するまで iSCSI サービスが起動しなくなります。これには、カーネルモジュール **iscsid** または **iscsi** の起動が必要になります。

iscsid サービスを強制的に実行し、iSCSI カーネルモジュールをロードするには、root として **systemctl start iscsid** コマンドを使用します。

手順

1. クライアントマシンに **iscsi-initiator-utils** をインストールします。

```
# yum install iscsi-initiator-utils
```

2. 実行中のセッションに関する情報を検索します。

```
# iscsiadm -m session -P 3
```

このコマンドは、セッションまたはデバイスの状態、セッション ID (sid)、いくつかのネゴシエートしたパラメーター、およびセッション経由でアクセス可能な SCSI デバイスを表示します。

- より短い出力 (たとえば **sid-to-node** 間のマッピングのみの表示) には、次のコマンドを実行します。

```
# iscsiadadm -m session -P 0
or
# iscsiadadm -m session

tcp [2] 10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
tcp [3] 10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

このコマンドは、**driver [sid] target_ip:port,target_portal_group_tag proper_target_name** の形式で実行中のセッションのリストを表示します。

関連情報

- [/usr/share/doc/iscsi-initiator-utils-version/README](#) ファイル
- システム上の **iscsiadm(8)** man ページ

9.4. DM MULTIPATH によるデバイスのタイムアウトのオーバーライド

sysfs オプションの **recovery_tmo** は、特定の iSCSI デバイスのタイムアウトを制御します。次のオプションは、**recovery_tmo** 値をグローバルにオーバーライドします。

- replacement_timeout** 設定オプションは、すべての iSCSI デバイスの **recovery_tmo** 値をグローバルにオーバーライドします。
- DM Multipath の **fast_io_fail_tmo** オプションは、DM Multipath によって管理されるすべての iSCSI デバイスの **recovery_tmo** 値をグローバルにオーバーライドします。
DM Multipath の **fast_io_fail_tmo** オプションは、ファイバーチャネルデバイスの **fast_io_fail_tmo** オプションもオーバーライドします。

DM Multipath の **fast_io_fail_tmo** オプションは、**replacement_timeout** よりも優先されます。**multipathd** サービスがリロードされるたびに、**recovery_tmo** が **fast_io_fail_tmo** 設定オプションの値にリセットされます。DM Multipath によって管理されるデバイスの **recovery_tmo** をオーバーライドするには、DM Multipath の **fast_io_fail_tmo** 設定オプションを使用してください。

第10章 ファイバーチャネルデバイスの使用

Red Hat Enterprise Linux 8 は、以下のネイティブファイバーチャネルドライバーを提供します。

- **lpfc**
- **qla2xxx**
- **zfcp**

10.1. LUN のサイズ変更後にファイバーチャネル論理ユニットを再スキャンする

外部ストレージの論理ユニット番号 (LUN) のサイズを変更した場合は、**echo** コマンドを使用してカーネルのサイズのビューを更新します。

手順

1. **multipath** 論理ユニットのパスとなるデバイスを特定します。

```
# multipath -ll
```

2. マルチパスを使用するシステムで、ファイバーチャネル論理ユニットを再スキャンします。

```
$ echo 1 > /sys/block/<device_ID>/device/rescan
```

<device_ID> は、デバイスの ID (例: **sda**) に置き換えます。

関連情報

- システム上の **multipath(8)** man ページ

10.2. ファイバーチャネルを使用したデバイスのリンク切れ動作の特定

ドライバーがトランスポートの **dev_loss_tmo** コールバックを実装している場合、トランスポートの問題が検出されるとリンクを経由したデバイスへのアクセス試行がブロックされます。

手順

- リモートポートの状態を判断します。

```
$ cat /sys/class/fc_remote_ports/rport-host:bus:remote-port/port_state
```

このコマンドは、次のいずれかの出力を返します。

- リモートポートからアクセスしたデバイスとともにリモートポートがブロックされると **Blocked** となります。
- リモートポートが正常に動作しているときには **Online** となります
dev_loss_tmo 秒以内に問題が解決されない場合は、**rport** およびデバイスのブロックが解除されます。そのデバイスで実行しているすべての I/O は、そのデバイスに送信された新しい I/O とともにすべて失敗します。

リンクロスが **dev_loss_tmo** を超えると、**scsi_device** デバイスおよび **sd_N_** デバイスが削除されます。通常、ファイバーチャネルクラスがデバイスを変更することはありません。たとえば、**/dev/sda** は **/dev/sda** のままであります。これは、ターゲットバインディングがファイバーチャネルドライバーによって保存されており、ターゲットポートが復帰した際に SCSI アドレスが忠実に再作成されるためです。ただし、これは保証されません。ストレージボックス内の LUN 設定に追加の変更が加えられていない場合にのみ、デバイスが復元されます。

関連情報

- システム上の **multipath.conf(5)** man ページ
- [Recommended tuning at scsi,multipath and at application layer while configuring Oracle RAC cluster \(Red Hat ナレッジベース\)](#)

10.3. ファイバーチャネル設定ファイル

以下は、ユーザー空間の API をファイバーチャネルに提供する **/sys/class/** ディレクトリーの設定ファイルのリストです。

項目は以下の変数を使用します。

H

ホスト番号

B

バス番号

T

ターゲット

L

論理ユニット (LUN)

-R

リモートポート番号



重要

システムでマルチパスソフトウェアを使用している場合は、このセクションで説明されている値を変更する前にハードウェアベンダーにお問い合わせください。

/sys/class/fc_transport/targetH:B:T/ のトランスポート設定

port_id

24 ビットのポート ID/アドレス

node_name

64 ビットのノード名

port_name

64 ビットのポート名

/sys/class/fc_remote_ports/rport-H:B-R/ のリモートポート設定

- port_id**

- **node_name**

- **port_name**

- **dev_loss_tmo**

scsi デバイスがシステムから削除されるタイミングを制御します。**dev_loss_tmo** がトリガーされると、scsi デバイスが削除されます。**multipath.conf** ファイルでは、**dev_loss_tmo** を **infinity** に設定できます。

Red Hat Enterprise Linux 8 では、**fast_io_fail_tmo** オプションを設定しないと、**dev_loss_tmo** の上限が **600** 秒になります。デフォルトでは、**multipathd** サービスが実行している場合は、Red Hat Enterprise Linux 8 の **fast_io_fail_tmo** が **5** 秒に設定されています。それ以外の場合は **off** に設定されています。

- **fast_io_fail_tmo**

リンクに "bad" のマークが付くまでの待機秒数を指定します。リンクに bad のマークが付けられると、対応するパス上の既存の実行中の I/O または新しい I/O が失敗します。

I/O がブロックされたキューに存在する場合は、**dev_loss_tmo** の期限が切れ、キューのブロックが解除されるまでエラーを起こしません。

fast_io_fail_tmo を **off** 以外の値に設定すると、**dev_loss_tmo** は取得されません。**fast_io_fail_tmo** を **off** に設定すると、システムからデバイスが削除されるまで I/O は失敗します。**fast_io_fail_tmo** に数値を設定すると、**fast_io_fail_tmo** タイムアウトが発生するとすぐに I/O が失敗します。

/sys/class/fc_host/hostH のホスト設定

- **port_id**

- **node_name**

- **port_name**

- **issue_lip**

リモートポートを再検出するようにドライバーに指示します。

第11章 FIBRE CHANNEL OVER ETHERNET の設定

IEEE T11 FC-BB-5 標準に基づく Fibre Channel over Ethernet (FCoE) は、イーサネットネットワーク上でファイバーチャネルフレームを送信するためのプロトコルです。通常、データセンターには専用の LAN および Storage Area Network (SAN) があり、各個別設定に分けられます。FCoE は、これらのネットワークを単一のネットワーク設定とコンバージドネットワーク設定に統合します。FCoE の利点 (ハードウェアおよび電力の低さなど) の利点は以下のとおりです。

11.1. RHEL でハードウェア FCOE HBA の使用

RHEL では、ハードウェアの Fibre Channel over Ethernet (FCoE) Host Bus Adapter (HBA) を使用できます。これは、以下のドライバーでサポートされています。

- **qedf**
- **bnx2fc**
- **fnic**

このような HBA を使用する場合は、HBA の設定で FCoE を設定します。詳細は、アダプターのドキュメントを参照してください。

HBA を設定すると、Storage Area Network (SAN) からエクスポートした論理ユニット番号 (LUN) が、**/dev/sd*** デバイスとして RHEL で自動的に利用可能になります。ローカルストレージデバイスと同様に、これらのデバイスを使用できます。

11.2. FCOE デバイスの設定

ソフトウェア FCoE デバイスを使用すると、FCoE オフロードを部分的にサポートするイーサネットアダプターを使用し、FCoE を介して論理ユニット番号 (LUN) にアクセスできます。



重要

RHEL は、**fcoe.ko** カーネルモジュールを必要とするソフトウェア FCoE デバイスに対応していません。

この手順を完了すると、ストレージエリアネットワーク (SAN) からのエクスポート LUN は、**/dev/sd*** デバイスとして RHEL で利用可能になります。このようなデバイスは、ローカルストレージデバイスと同様に使用できます。

前提条件

- VLAN に対応するようにネットワークスイッチを設定している。
- SAN は VLAN を使用して、通常のイーサネットトラフィックからストレージトラフィックを分離します。
- サーバーの HBA が BIOS で設定されている。
- HBA がネットワークに接続されており、リンクが起動している。詳細は、HBA のドキュメントを参照してください。

手順

1. **fcoe-utils** パッケージをインストールします。

```
# yum install fcoe-utils
```

2. **/etc/fcoe/cfg-ethx** テンプレートファイルを **/etc/fcoe/cfg-interface_name** にコピーします。たとえば、FCoE を使用するように **enp1s0** インターフェイスを設定する場合は、以下のコマンドを実行します。

```
# cp /etc/fcoe/cfg-ethx /etc/fcoe/cfg-enp1s0
```

3. **fcoe** サービスを有効にして起動します。

```
# systemctl enable --now fcoe
```

4. インターフェイス **enp1s0** で FCoE VLAN を検出し、検出された VLAN のネットワークデバイスを作成して、イニシエーターを起動します。

```
# fipvlan -s -c enp1s0
Created VLAN device enp1s0.200
Starting FCoE on interface enp1s0.200
Fibre Channel Forwarders Discovered
interface | VLAN | FCF MAC
-----
enp1s0 | 200 | 00:53:00:a7:e7:1b
```

5. 必要に応じて、検出されたターゲット、LUN、および LUN に関連付けられたデバイスの詳細を表示します。

```
# fcoeadm -t
Interface: enp1s0.200
Roles: FCP Target
Node Name: 0x500a0980824acd15
Port Name: 0x500a0982824acd15
Target ID: 0
MaxFrameSize: 2048 bytes
OS Device Name: rport-11:0-1
FC-ID (Port ID): 0xba00a0
State: Online

LUN ID Device Name Capacity Block Size Description
----- -----
0 sdb 28.38 GiB 512 NETAPP LUN (rev 820a)
...
```

この例では、SAN からの LUN 0 が **/dev/sdb** デバイスとしてホストに割り当てられていることを示しています。

検証

- アクティブなすべての FCoE インターフェイスの情報を表示します。

```
# fcoeadm -i
Description: BCM57840 NetXtreme II 10 Gigabit Ethernet
```

Revision: 11
Manufacturer: Broadcom Inc. and subsidiaries
Serial Number: 000AG703A9B7

Driver: bnx2x Unknown
Number of Ports: 1

Symbolic Name: bnx2fc (QLogic BCM57840) v2.12.13 over enp1s0.200
OS Device Name: host11
Node Name: 0x2000000af70ae935
Port Name: 0x2001000af70ae935
Fabric Name: 0x20c8002a6aa7e701
Speed: 10 Gbit
Supported Speed: 1 Gbit, 10 Gbit
MaxFrameSize: 2048 bytes
FC-ID (Port ID): 0xba02c0
State: Online

関連情報

- システム上の **fcoeadm(8)** man ページ
- /usr/share/doc/fcoe-utils/README**
- [ファイバーチャネルデバイスの使用](#)

第12章 EH_DEADLINE を使用したストレージエラーからの回復における最大時間の設定

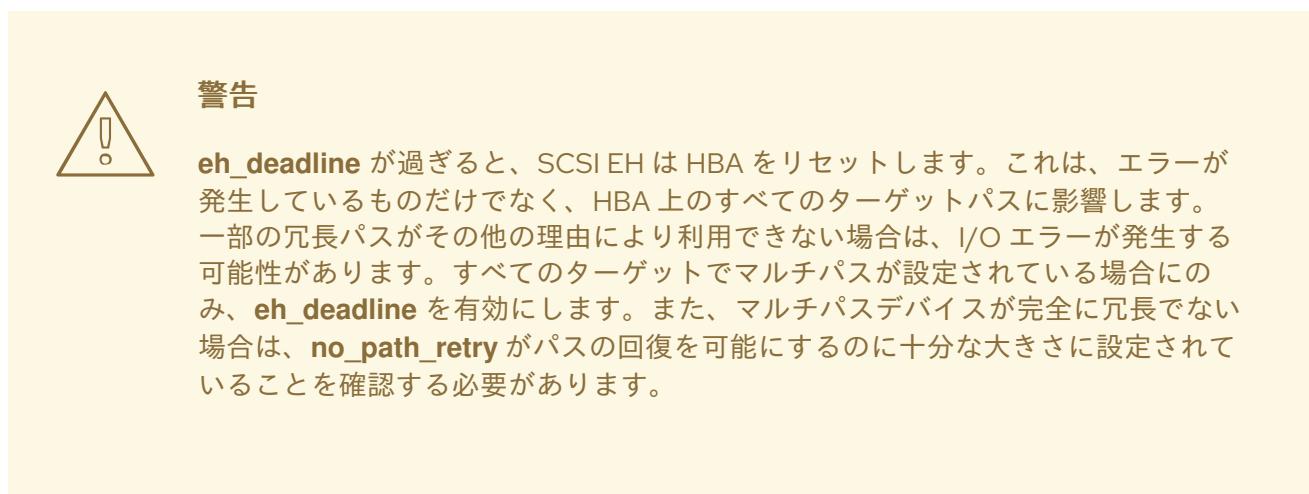
障害が発生した SCSI デバイスを復旧するのに許容できる最大時間を設定できます。この設定は、ストレージハードウェアが不具合により応答しなくなっても、I/O 応答時間を保証します。

12.1. EH_DEADLINE パラメーター

SCSI エラー処理 (EH) メカニズムは、障害が発生した SCSI デバイスでエラーからの復旧の実行を試行します。SCSI ホストオブジェクト **eh_deadline** パラメーターでは、復旧時間の最大量を設定できます。設定した時間が過ぎると、SCSI EH は、ホストバスアダプター (HBA) 全体を停止してリセットします。

eh_deadline を使用すると、以下のいずれかの時間を短縮できます。

- エラーのあるパスのシャットオフ
- パスの切り替え
- RAID スライスの無効化



eh_deadline パラメーターの値は秒単位で指定されます。デフォルト設定は **off** で、時間制限が無効になり、すべてのエラー復旧が行われるようになります。

eh_deadline が便利なシナリオ

多くの場合、**eh_deadline** を有効にする必要はありません。**eh_deadline** を使用すると、特定のシナリオで役立つ場合があります。たとえば、ファイバーチャネル (FC) スイッチとターゲットポート間でリンクが失われ、HBA が Registered State Change Notifications (RSCN) を受信しない場合などです。このような場合、I/O 要求やエラーからの復旧コマンドは、エラーに遭遇することなく、すべてタイムアウトになります。この環境で **eh_deadline** を設定すると、リカバリー時間に上限が課せられます。これにより、DM Multipath により、利用できる別のパスで不具合の発生した I/O の再試行が可能になります。

以下の条件下では、**eh_deadline** パラメーターは、これ以上のメリットをもたらしません。その理由は、DM Multipath の再試行を可能にする I/O とエラー復旧コマンドがすぐに失敗するためです。

- RSCN が有効になっている場合
- HBA が利用できなくなっているリンクを登録しない場合

12.2. EH_DEADLINE パラメーターの設定

eh_deadline パラメーターの値を設定することで、SCSI 最大復旧時間を制限できます。

手順

- **eh_deadline** は、以下のいずれかの方法で設定できます。
 - **multipath.conf** ファイルの **defaults** セクション
multipath.conf ファイルの **defaults** セクションから、**eh_deadline** パラメーターを必要な秒数に設定します。

```
# eh_deadline 300
```



注記

RHEL 8.4 以降、**multipath.conf** ファイルの **defaults** セクションを使用して **eh_deadline** パラメーターを設定することが推奨されます。

このメソッドで **eh_deadline** パラメーターをオフにするには、**eh_deadline** を **off** に設定します。

- **sysfs**
/sys/class/scsi_host/host<host-number>/eh_deadline ファイルに秒数を書き込みます。たとえば、SCSI ホスト 6 の **sysfs** を介して **eh_deadline** パラメーターを設定するには、次のようにします。

```
# echo 300 > /sys/class/scsi_host/host6/eh_deadline
```

このメソッドで **eh_deadline** パラメーターをオフにするには、**echo off** を使用します。

- カーネルパラメーター
すべての SCSI HBA のデフォルト値は **scsi_mod.eh_deadline** カーネルパラメーターを使用して設定します。

```
# echo 300 > /sys/module/scsi_mod/parameters/eh_deadline
```

このメソッドで **eh_deadline** パラメーターをオフにするには、**echo -1** を使用します。

関連情報

- [How to set eh_deadline and eh_timeout persistently, using a udev rule](#) (Red Hat ナレッジベース)

第13章 スワップの使用

スワップ領域を使用して、非アクティブなプロセスとデータに一時的なストレージを提供し、物理メモリーがいっぱいになった場合に発生するメモリー不足エラーを防ぎます。スワップ領域は物理メモリーの拡張として機能し、物理メモリーが使い果たされた場合でもシステムがスムーズに動作し続けることを可能にします。スワップ領域を使用するとシステムのパフォーマンスが低下する可能性があるため、スワップ領域を利用する前に物理メモリーの使用を最適化するほうが望ましい場合があることに注意してください。

13.1. スワップ領域の概要

Linux の **スワップ領域** は、物理メモリー (RAM) が不足すると使用されます。システムに多くのメモリリソースが必要で、RAM が不足すると、メモリーの非アクティブなページがスワップ領域に移動します。スワップ領域は、RAM が少ないマシンで役に立ちますが、RAM の代わりに使用しないようにしてください。

スワップ領域はハードドライブにあり、そのアクセス速度は物理メモリーに比べると遅くなります。スワップ領域の設定は、専用のスワップパーティション (推奨)、スワップファイル、またはスワップパーティションとスワップファイルの組み合せが考えられます。

過去数年、推奨されるスワップ領域のサイズは、システムの RAM サイズに比例して増加していました。しかし、最近のシステムには通常、数百ギガバイトの RAM が含まれます。結果として、推奨されるスワップ領域は、システムのメモリーではなく、システムメモリーのワークロードの機能とみなされます。

13.2. システムの推奨スワップ領域

推奨されるスワップパーティションのサイズは、システムの RAM の容量と、システムをハイバネートするのに十分なメモリーが必要かどうかによって異なります。推奨されるスワップパーティションのサイズは、インストール時に自動的に設定されます。ハイバネートを可能にするには、カスタムのパーティション分割段階でスワップ領域を編集する必要があります。

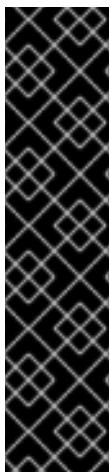
以下の推奨は、1GB 以下など、メモリーが少ないシステムで特に重要です。このようなシステムで十分なスワップ領域を割り当てられないと、システムが不安定になったり、インストールしたシステムが起動できなくなったりする可能性があります。

表13.1 推奨されるスワップ領域

システム内の RAM の容量	推奨されるスワップ領域	ハイバネートを許可する場合に推奨されるスワップ領域
≤ 2 GB	RAM 容量の 2 倍	RAM 容量の 3 倍
> 2 GB - 8 GB	RAM 容量と同じ	RAM 容量の 2 倍
> 8 GB - 64 GB	最低 4GB	RAM 容量の 1.5 倍
> 64 GB	最低 4GB	ハイバネートは推奨されない

システム RAM が 2 GB、8 GB、または 64 GB などの境界値の場合は、必要に応じてスワップサイズを選択してください。システムリソースに余裕がある場合は、スワップ領域を増やすとパフォーマンスが向上することがあります。

高速のドライブ、コントローラー、およびインターフェイスを搭載したシステムでは、複数のストレージデバイスにスワップ領域を分散すると、スワップ領域のパフォーマンスも向上します。



重要

スワップ領域として割り当てたファイルシステムおよびLVM2ボリュームは、変更時に使用しないでください。システムプロセスまたはカーネルがスワップ領域を使用していると、スワップの修正に失敗します。**free** コマンドおよび**cat /proc/swaps** コマンドを使用して、スワップの使用量と、使用中の場所を確認します。

スワップ領域のサイズを変更するには、システムから一時的にスワップ領域を削除する必要があります。これは、実行中のアプリケーションが追加のスワップ領域に依存し、メモリーが不足する可能性がある場合に問題になる可能性があります。できれば、レスキューモードからスワップのサイズ変更を実行してください。[デバッグ起動オプション](#)を参照してください。ファイルシステムをマウントするように指示されたら、スキップを選択します。

13.3. スワップ用の LVM2 論理ボリュームの作成

スワップ用の LVM2 論理ボリュームを作成できます。ここでは、追加するスワップボリュームを **/dev/VolGroup00/LogVol02** とします。

前提条件

- 十分なディスク領域がある。

手順

- サイズが 2 GB の LVM2 論理ボリュームを作成します。

```
# lvcreate VolGroup00 -n LogVol02 -L 2G
```

- 新しいスワップ領域をフォーマットします。

```
# mkswap /dev/VolGroup00/LogVol02
```

- 次のエントリーを **/etc/fstab** ファイルに追加します。

```
/dev/VolGroup00/LogVol02 none swap defaults 0 0
```

- システムが新しい設定を登録するように、マウントユニットを再生成します。

```
# systemctl daemon-reload
```

- 論理ボリュームでスワップをアクティブにします。

```
# swapon -v /dev/VolGroup00/LogVol02
```

検証

- スワップ論理ボリュームが正常に作成され、アクティブになったかをテストするには、次のコマンドを使用して、アクティブなスワップ領域を調べます。

```
# cat /proc/swaps
total used free shared buff/cache available
Mem: 30Gi 1.2Gi 28Gi 12Mi 994Mi 28Gi
Swap: 22Gi 0B 22Gi

# free -h
total used free shared buff/cache available
Mem: 30Gi 1.2Gi 28Gi 12Mi 995Mi 28Gi
Swap: 17Gi 0B 17Gi
```

13.4. スワップファイルの作成

システムのメモリーが不足しているときに、スワップファイルを作成して、ソリッドステートドライブまたはハードディスク上に一時的なストレージ領域を作成できます。

前提条件

- 十分なディスク領域がある。

手順

- 新しいスワップファイルのサイズをメガバイト単位で指定してから、そのサイズに1024をかけてブロック数を指定します。たとえばスワップファイルのサイズが64 MBの場合は、ブロック数が65536になります。
- 空のファイルの作成:

```
# dd if=/dev/zero of=/swapfile bs=1024 count=65536
```

65536を、必要なブロックサイズと同じ値に置き換えます。

- 次のコマンドでスワップファイルをセットアップします。

```
# mkswap /swapfile
```

- スワップファイルのセキュリティーを変更して、全ユーザーで読み込みができないようにします。

```
# chmod 0600 /swapfile
```

- システムの起動時にスワップファイルを有効にするには、次のエントリーを使用して **/etc/fstab** ファイルを編集します。

```
/swapfile none swap defaults 0 0
```

次にシステムが起動すると新しいスワップファイルが有効になります。

- システムが新しい **/etc/fstab** 設定を登録するように、マウントユニットを再生成します。

```
# systemctl daemon-reload
```

- すぐにスワップファイルをアクティブにします。

```
# swapon /swapfile
```

検証

- 新しいスワップファイルが正常に作成され、有効になったかをテストするには、次のコマンドを使用して、アクティブなスワップ領域を調べます。

```
$ cat /proc/swaps
```

```
$ free -h
```

13.5. STORAGE RHEL システムロールを使用してスワップボリュームを作成する

このセクションでは、Ansible Playbook の例を示します。この Playbook は、**storage** ロールを適用し、デフォルトのパラメーターを使用して、ブロックデバイスにスワップボリュームが存在しない場合は作成し、スワップボリュームがすでに存在する場合はそれを変更します。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

- 次の内容を含む Playbook ファイル (例: `~/playbook.yml`) を作成します。

```
---
- name: Create a disk device with swap
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: swap_fs
        type: disk
        disks:
          - /dev/sdb
        size: 15 GiB
        fs_type: swap
```

現在、ボリューム名 (この例では `swap_fs`) は任意です。**storage** ロールは、**disks:** 属性にリスト表示されているディスクデバイスでボリュームを特定します。

- Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

- Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

関連情報

- [/usr/share/ansible/roles/rhel-system-roles.storage/README.md](#) ファイル
- [/usr/share/doc/rhel-system-roles/storage/](#) ディレクトリー

13.6. LVM2 論理ボリュームでのスワップ領域の拡張

既存の LVM2 論理ボリューム上のスワップ領域を拡張できます。ここでは、2 GB 拡張するボリュームを `/dev/VolGroup00/LogVol01` とします。

前提条件

- 十分なディスク領域がある。

手順

- 関連付けられている論理ボリュームのスワップ機能を無効にします。

```
# swapoff -v /dev/VolGroup00/LogVol01
```

- LVM2 論理ボリュームのサイズを 2 GB 増やします。

```
# lvresize /dev/VolGroup00/LogVol01 -L +2G
```

- 新しいスワップ領域をフォーマットします。

```
# mkswap /dev/VolGroup00/LogVol01
```

- 拡張論理ボリュームを有効にします。

```
# swapon -v /dev/VolGroup00/LogVol01
```

検証

- スワップの論理ボリュームの拡張に成功したかどうかをテストするには、アクティブなスワップ容量を調べます。

```
# cat /proc/swaps
Filename      Type  Size   Used  Priority
/dev/dm-1    partition 16322556     0      -2
/dev/dm-4    partition  7340028     0      -3
```

```
# free -h
total   used   free   shared  buff/cache  available
```

Mem:	30Gi	1.2Gi	28Gi	12Mi	994Mi	28Gi
Swap:	22Gi	0B	22Gi			

13.7. LVM2 論理ボリュームでのスワップ領域の縮小

LVM2 論理ボリュームのスワップ領域を縮小できます。ここでは、縮小するボリュームを `/dev/VolGroup00/LogVol01` とします。

手順

- 関連付けられている論理ボリュームのスワップ機能を無効にします。

```
# swapoff -v /dev/VolGroup00/LogVol01
```

- スワップ署名を削除します。

```
# wipefs -a /dev/VolGroup00/LogVol01
```

- LVM2 論理ボリュームのサイズを変更して 512 MB 削減します。

```
# lvreduce /dev/VolGroup00/LogVol01 -L -512M
```

- 新しいスワップ領域をフォーマットします。

```
# mkswap /dev/VolGroup00/LogVol01
```

- 論理ボリュームでスワップをアクティブにします。

```
# swapon -v /dev/VolGroup00/LogVol01
```

検証

- スワップ論理ボリュームが正常に削減されたかをテストするには、次のコマンドを使用して、アクティブなスワップ領域を調べます。

```
$ cat /proc/swaps
```

```
$ free -h
```

13.8. スワップ用の LVM2 論理ボリュームの削除

スワップ用の LVM2 論理ボリュームを削除できます。削除するスワップボリュームを `/dev/VolGroup00/LogVol02` とします。

手順

- 関連付けられている論理ボリュームのスワップ機能を無効にします。

```
# swapoff -v /dev/VolGroup00/LogVol02
```

2. LVM2 論理ボリュームを削除します。

```
# lvremove /dev/VolGroup00/LogVol02
```

3. 次の関連エントリーを **/etc/fstab** ファイルから削除します。

```
/dev/VolGroup00/LogVol02 none swap defaults 0 0
```

4. マウントユニットを再生成して新しい設定を登録します。

```
# systemctl daemon-reload
```

検証

- 論理ボリュームが正常に削除されたかどうかをテストし、次のコマンドを使用してアクティブなスワップ領域を調べます。

```
$ cat /proc/swaps
```

```
$ free -h
```

13.9. スワップファイルの削除

スワップファイルを削除できます。

手順

- /swapfile** スワップファイルを無効にします。

```
# swapoff -v /swapfile
```

- /etc/fstab** ファイルからエントリーを削除します。

- システムが新しい設定を登録するように、マウントユニットを再生成します。

```
# systemctl daemon-reload
```

- 実際のファイルを削除します。

```
# rm /swapfile
```

第14章 NVME OVER FABRIC デバイスの概要

Non-volatile Memory Express™ (NVMe™) は、ホストソフトウェアユーティリティがソリッドストレードライブと通信できるようにするインターフェイスです。

次の種類のファブリックトランスポートを使用して、NVMe over fabric デバイスを設定します。

NVMe over Remote Direct Memory Access (NVMe/RDMA)

NVMe™/RDMA の設定方法については、[NVMe/RDMA を使用した NVMe over Fabric の設定](#) を参照してください。

NVMe over Fibre Channel (NVMe/FC)

NVMe™/FC の設定方法については、[NVMe/FC を使用した NVMe over Fabric の設定](#) を参照してください。

ファブリック上で NVMe を使用する場合、ソリッドストレートドライブはシステムに対してローカルである必要はありません。NVMe over Fabrics デバイスを介してリモートで設定できます。

第15章 NVME/RDMA を使用した NVME OVER FABRICS の設定

Non-volatile Memory Express™ (NVMe™) over RDMA (NVMe™/RDMA) セットアップでは、NVMe コントローラーと NVMe イニシエーターを設定します。

15.1. CONFIGFS を使用した NVME/RDMA コントローラーのセットアップ

configfs を使用して、Non-volatile Memory Express™ (NVMe™) over RDMA (NVMe™/RDMA) コントローラーを設定できます。

前提条件

- **nvmet** サブシステムに割り当てるブロックデバイスがあることを確認する。

手順

1. **nvmet-rdma** サブシステムを作成します。

```
# modprobe nvmet-rdma
# mkdir /sys/kernel/config/nvmet/subsystems/testnqn
# cd /sys/kernel/config/nvmet/subsystems/testnqn
```

testnqn を、サブシステム名に置き換えます。

2. すべてのホストがこのコントローラーに接続できるようにします。

```
# echo 1 > attr_allow_any_host
```

3. namespace を設定します。

```
# mkdir namespaces/10
# cd namespaces/10
```

10 を、namespace の数値に置き換えます。

4. NVMe デバイスへのパスを設定します。

```
# echo -n /dev/nvme0n1 > device_path
```

5. namespace を有効にします。

```
# echo 1 > enable
```

6. NVMe ポートでディレクトリーを作成します。

```
# mkdir /sys/kernel/config/nvmet/ports/1
# cd /sys/kernel/config/nvmet/ports/1
```

7. mlx5_ib0 の IP アドレスを表示します。

```
# ip addr show mlx5_ib0

8: mlx5_ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 4092 qdisc mq state UP
  group default qlen 256
    link/infiniband 00:00:06:2f:fe:80:00:00:00:00:00:e4:1d:2d:03:00:e7:0f:f6 brd
    00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:ff:ff:ff:ff
      inet 172.31.0.202/24 brd 172.31.0.255 scope global noprefixroute mlx5_ib0
        valid_lft forever preferred_lft forever
      inet6 fe80::e61d:2d03:e7:ff6/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

8. コントローラーのトランスポートアドレスを設定します。

```
# echo -n 172.31.0.202 > addr_traddr
```

9. RDMA をトランスポートタイプとして設定します。

```
# echo rdma > addr_trtype
# echo 4420 > addr_trsvcid
```

10. ポートのアドレスファミリーを設定します。

```
# echo ipv4 > addr_adrfam
```

11. ソフトリンクを作成します。

```
# ln -s /sys/kernel/config/nvmet/subsystems/testnqn
  /sys/kernel/config/nvmet/ports/1/subsystems/testnqn
```

検証

- NVMe コントローラーが指定されたポートでリッスンしていて、接続要求の準備ができていることを確認します。

```
# dmesg | grep "enabling port"
[1091.413648] nvmet_rdma: enabling port 1 (172.31.0.202:4420)
```

関連情報

- システム上の **nvme(1)** man ページ

15.2. NVMETCLI を使用した NVME/RDMA コントローラーのセットアップ

nvmetcli ユーティリティーを使用して、Non-volatile Memory Express™ (NVMe™) over RDMA (NVMe™/RDMA) コントローラーを設定できます。**nvmetcli** ユーティリティーには、コマンドラインと対話式のシェルオプションが用意されています。

前提条件

- **nvmet** サブシステムに割り当てるブロックデバイスがあることを確認する。
- root で、以下の **nvmetcli** 操作を実行する。

手順

1. **nvmetcli** パッケージをインストールします。

```
# yum install nvmetcli
```

2. **rdma.json** ファイルをダウンロードします。

```
# wget
http://git.infradead.org/users/hch/nvmetcli.git/blob_plain/0a6b088db2dc2e5de11e6f23f
1e890e4b54fee64:/rdma.json
```

3. **rdma.json** ファイルを編集して、**traddr** の値を **172.31.0.202** に変更します。

4. NVMe コントローラー設定ファイルをロードして、コントローラーをセットアップします。

```
# nvmetcli restore rdma.json
```



注記

NVMe コントローラー設定ファイル名を指定しない場合は、**nvmetcli** が **/etc/nvmet/config.json** ファイルを使用します。

検証

- NVMe コントローラーが指定されたポートでリッスンしていて、接続要求の準備ができていることを確認します。

```
# dmesg | tail -1
[4797.132647] nvmet_rdma: enabling port 2 (172.31.0.202:4420)
```

- オプション: 現在の NVMe コントローラーをクリアします。

```
# nvmetcli clear
```

関連情報

- システム上の **nvmetcli** および **nvme(1)** man ページ

15.3. NVME/RDMA ホストの設定

NVMe 管理コマンドラインインターフェイス (**nvme-cli**) ツールを使用して、Non-volatile Memory Express™ (NVMe™) over RDMA (NVMe™/RDMA) ホストを設定できます。

手順

1. **nvme-cli** ツールをインストールします。

■

```
# yum install nvme-cli
```

2. **nvme-rdma** モジュールが読み込まれていない場合は、読み込みます。

```
# modprobe nvme-rdma
```

3. NVMe コントローラーで使用可能なサブシステムを検出します。

```
# nvme discover -t rdma -a 172.31.0.202 -s 4420
```

```
Discovery Log Number of Records 1, Generation counter 2
=====Discovery Log Entry 0=====
trtype: rdma
adrifam: ipv4
subtype: nvme subsystem
treq: not specified, sq flow control disable supported
portid: 1
trsvcid: 4420
subnqn: testnqn
traddr: 172.31.0.202
rdma_prtype: not specified
rdma_qptype: connected
rdma_cms: rdma-cm
rdma_pkey: 0x0000
```

4. 検出されたサブシステムに接続します。

```
# nvme connect -t rdma -n testnqn -a 172.31.0.202 -s 4420
```

```
# lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda            8:0    0 465.8G  0 disk
  └─sda1         8:1    0   1G  0 part /boot
  └─sda2         8:2    0 464.8G  0 part
    ├─rhel_rdma--virt--03-root 253:0  0   50G  0 lvm /
    ├─rhel_rdma--virt--03-swap 253:1  0   4G  0 lvm [SWAP]
    └─rhel_rdma--virt--03-home 253:2  0 410.8G  0 lvm /home
nvme0n1
```

```
# cat /sys/class/nvme/nvme0/transport
rdma
```

testnqn を NVMe サブシステム名に置き換えます。

172.31.0.202 をコントローラーの IP アドレスに置き換えます。

4420 を、ポート番号に置き換えます。

検証

- 現在接続されている NVMe デバイスのリストを表示します。

```
# nvme list
```

- オプション: コントローラーから切断します。

```
# nvme disconnect -n testnqn
NQN:testnqn disconnected 1 controller(s)

# lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda            8:0    0 465.8G  0 disk
├─sda1          8:1    0   1G  0 part /boot
└─sda2          8:2    0 464.8G  0 part
  ├─rhel_rdma--virt--03-root 253:0  0  50G  0 lvm /
  ├─rhel_rdma--virt--03-swap 253:1  0   4G  0 lvm [SWAP]
  └─rhel_rdma--virt--03-home 253:2  0 410.8G  0 lvm /home
```

関連情報

- システム上の **nvme(1)** man ページ

15.4. 次のステップ

- NVMe デバイスでのマルチパスの有効化

第16章 NVME/FC を使用した NVME OVER FABRICS の設定

Non-volatile Memory Express™ (NVMe™) over Fibre Channel (NVMe™/FC) トランスポートは、特定の Broadcom Emulex および Marvell Qlogic ファイバーチャネルアダプターと共に使用する場合、ホスト モードで完全にサポートされます。

16.1. BROADCOM アダプターの NVME ホストの設定

NVMe 管理コマンドラインインターフェイス (**nvme-cli**) ユーティリティーを使用して、Broadcom アダプターで Non-volatile Memory Express™ (NVMe™) ホストを設定できます。

手順

1. **nvme-cli** ユーティリティーをインストールします。

```
# yum install nvme-cli
```

これにより、**/etc/nvme/** ディレクトリーに **hostnqn** ファイルが作成されます。**hostnqn** ファイルは、NVMe ホストを識別します。

2. ローカルポートとリモートポートのワールドワイドノード名 (WWNN) とワールドワイドポート名 (WWPN) 識別子を見つけます。

```
# cat /sys/class/scsi_host/host*/nvme_info
```

```
NVME Host Enabled
XRI Dist lpfco Total 6144 IO 5894 ELS 250
NVME LPORT lpfco WWPN x10000090fae0b5f5 WWNN x20000090fae0b5f5 DID x010f00
ONLINE
NVME RPORT    WWPN x204700a098cbcac6 WWNN x204600a098cbcac6 DID x01050e
TARGET DISCSRVC ONLINE

NVME Statistics
LS: Xmt 00000000e Cmpl 00000000e Abort 00000000
LS XMIT: Err 00000000 CMPL: xb 00000000 Err 00000000
Total FCP Cmpl 00000000000008ea Issue 00000000000008ec OutIO 0000000000000002
    abort 00000000 noxri 00000000 nondlp 00000000 qdepth 00000000 wqerr 00000000 err
00000000
FCP CMPL: xb 00000000 Err 00000000
```

これらの **host-traddr** と **traddr** の値を使用して、Subsystem NVMe Qualified Name (NQN) を検索します。

```
# nvme discover --transport fc \
--traddr nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 \
--host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5
```

```
Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0=====
trtype: fc
adrfrm: fibre-channel
subtype: nvme subsystem
treq: not specified
portid: 0
```

```

trsvcid: none
subnqn: nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
traddr: nn-0x204600a098cbcac6:pn-0x204700a098cbcac6

```

nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 を、 **traddr** に置き換えます。

nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 を、 **host-traddr** に置き換えます。

3. **nvme-cli** を使用して NVMe コントローラーに接続します。

```

# nvme connect --transport fc \
    --traddr nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 \
    --host-traddr nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 \
    -n nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1 \
    -k 5

```



注記

接続時間がデフォルトの keep-alive タイムアウト値を超えると **keep-alive timer (5 seconds) expired!** と表示される場合は、**-k** オプションを使用して値を増やします。たとえば、**-k 7** を使用できます。

ここでは、以下のようになります。

nn-0x204600a098cbcac6:pn-0x204700a098cbcac6 を、 **traddr** に置き換えます。

nn-0x20000090fae0b5f5:pn-0x10000090fae0b5f5 を、 **host-traddr** に置き換えます。

nqn.1992-
08.com.netapp:sn.e18bfca87d5e11e98c0800a098cbcac6:subsystem.st14_nvme_ss_1_1
を、 **subnqn** に置き換えます。

5 を keep-alive タイムアウト値(秒単位)に置き換えます。

検証

- 現在接続されている NVMe デバイスのリストを表示します。

```

# nvme list
  Node      SN           Model                 Namespace Usage
  Format    FW Rev
  -----
  /dev/nvme0n1  80BgLFM7xMJbAAAAAAAC NetApp ONTAP Controller      1
  107.37 GB / 107.37 GB   4 KiB + 0 B  FFFFFFFF

# lsblk |grep nvme
  nvme0n1          259:0  0  100G  0 disk

```

関連情報

- システム上の **nvme(1)** man ページ

16.2. QLOGIC アダプターの NVME ホストの設定

NVMe 管理コマンドラインインターフェイス (**nvme-cli**) ユーティリティーを使用して、Qlogic アダプターで Non-volatile Memory Express™ (NVMe™) ホストを設定できます。

手順

- nvme-cli** ユーティリティーをインストールします。

```
# yum install nvme-cli
```

これにより、**/etc/nvme/** ディレクトリーに **hostnqn** ファイルが作成されます。**hostnqn** ファイルは、NVMe ホストを識別します。

- qla2xxx** を再読み込みします。

```
# modprobe -r qla2xxx
# modprobe qla2xxx
```

- ローカルポートとリモートポートのワールドワイドノード名 (WWNN) とワールドワイドポート名 (WWPN) 識別子を見つけます。

```
# dmesg |grep traddr
[ 6.139862] qla2xxx [0000:04:00.0]-ffff:0: register_localport: host-traddr=nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 on portID:10700
[ 6.241762] qla2xxx [0000:04:00.0]-2102:0: qla_nvme_register_remote: traddr=nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 PortID:01050d
```

これらの **host-traddr** と **traddr** の値を使用して、Subsystem NVMe Qualified Name (NQN) を検索します。

```
# nvme discover --transport fc \
--traddr nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 \
--host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62

Discovery Log Number of Records 2, Generation counter 49530
=====Discovery Log Entry 0=====
trtype: fc
adrfaam: fibre-channel
subtype: nvme subsystem
treq: not specified
portid: 0
trsvcid: none
subnqn: nqn.1992-08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsystem_468
traddr: nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6
```

nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 を、**traddr** に置き換えます。

nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 を、**host-traddr** に置き換えます。

4. **nvme-cli** ツールを使用して NVMe コントローラーに接続します。

```
# nvme connect --transport fc \
    --traddr nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 \
    --host-traddr nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 \
    -n nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsystem_468 \
    -k 5
```



注記

接続時間がデフォルトの keep-alive タイムアウト値を超えると **keep-alive timer (5 seconds) expired!** と表示される場合は、**-k** オプションを使用して値を増やします。たとえば、**-k 7** を使用できます。

ここでは、以下のようになります。

nn-0x203b00a098cbcac6:pn-0x203d00a098cbcac6 を、**traddr** に置き換えます。

nn-0x20000024ff19bb62:pn-0x21000024ff19bb62 を、**host-traddr** に置き換えます。

nqn.1992-
08.com.netapp:sn.c9ecc9187b1111e98c0800a098cbcac6:subsystem.vs_nvme_multipath_1_subsystem_468 を、**subnqn** に置き換えます。

5 を keep-live タイムアウト値(秒単位)に置き換えます。

検証

- 現在接続されている NVMe デバイスのリストを表示します。

```
# nvme list
Node      SN           Model           Namespace Usage
Format    FW Rev
-----
-----
/dev/nvme0n1  80BgLFM7xMjbAAAAAAAC NetApp ONTAP Controller      1
107.37 GB / 107.37 GB   4 KiB + 0 B  FFFFFFFF

# lsblk |grep nvme
nvme0n1          259:0  0  100G 0 disk
```

関連情報

- システム上の **nvme(1)** man ページ

16.3. 次のステップ

- NVMe デバイスでのマルチパスの有効化

第17章 NVME デバイスでのマルチパスの有効化

ファイバーチャネル (FC) などのファブリックトランスポートを介して、システムに接続されている Non-volatile Memory Express™ (NVMe™) デバイスをマルチパスすることができます。複数のマルチパスソリューションを選択することができます。

17.1. ネイティブ NVME マルチパスと DM MULTIPATH

Non-volatile Memory Express™ (NVMe™) デバイスは、ネイティブなマルチパス機能をサポートしています。NVMe にマルチパスを設定する場合、標準の DM Multipath フレームワークと NVMe のネイティブなマルチパスのどちらかを選択できます。

DM Multipath と NVMe のネイティブマルチパスは、どちらも NVMe デバイスのマルチパス方式である ANA(Asymmetric Namespace Access) に対応しています。ANA は、コントローラーとホスト間の最適化されたパスを特定し、パフォーマンスを向上させます。

ネイティブ NVMe マルチパスを有効にすると、すべての NVMe デバイスにグローバルに適用されます。より高いパフォーマンスを提供できますが、DM Multipath が提供するすべての機能は含まれていません。例えば、ネイティブの NVMe マルチパスは、**numa** と **round-robin** のパス選択方法のみをサポートしています。

Red Hat は、Red Hat Enterprise Linux 8 の DM Multipath をデフォルトのマルチパスソリューションとして使用することを推奨します。

17.2. ネイティブ NVME マルチパスの実現

nvme_core.multipath オプションのデフォルトのカーネル設定は **N** に設定されています。これは、ネイティブ Non-volatile Memory Express™ (NVMe™) マルチパスが無効であることを意味します。ネイティブ NVMe マルチパスソリューションを使用して、ネイティブ NVMe マルチパスを有効にすることができます。

前提条件

- NVMe デバイスがシステムに接続されていることを確認します。詳細は、[NVMe over fabric デバイスの概要](#) を参照してください。

手順

1. カーネルでネイティブ NVMe マルチパスが有効になっているかどうかを確認します。

```
# cat /sys/module/nvme_core/parameters/multipath
```

コマンドは以下のいずれかを表示します。

N

ネイティブ NVMe マルチパスは無効です。

Y

ネイティブ NVMe マルチパスは有効です。

2. ネイティブ NVMe マルチパスが無効になっている場合は、次のいずれかの方法を使用して有効にします。

- カーネルオプションの使用

a. **nvme_core.multipath=Y** オプションをコマンドラインに追加します。

```
# grubby --update-kernel=ALL --args="nvme_core.multipath=Y"
```

b. 64 ビットの IBM Z アーキテクチャーでは、ブートメニューを更新します。

```
# zipl
```

c. システムを再起動します。

- カーネルモジュール設定ファイルの使用

a. 以下の内容で **/etc/modprobe.d/nvme_core.conf** 設定ファイルを作成します。

```
options nvme_core multipath=Y
```

b. **initramfs** ファイルをバックアップします。

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +\%m-\%d-\%H\%M\%S).img
```

c. **initramfs** を再構築します。

```
# dracut --force --verbose
```

d. システムを再起動します。

3. オプション: 実行中のシステムで、NVMe デバイスの I/O ポリシーを変更して、利用可能なすべてのパスに I/O を分散させます。

```
# echo "round-robin" > /sys/class/nvme-subsystem/nvme-subsy0/iopolicy
```

4. オプション: **udev** ルールを使用して I/O ポリシーを永続的に設定します。以下の内容で **/etc/udev/rules.d/71-nvme-io-policy.rules** ファイルを作成します。

```
ACTION=="add|change", SUBSYSTEM=="nvme-subsystem", ATTR{iopolicy}="round-robin"
```

検証

1. システムが NVMe デバイスを認識しているかどうかを確認します。次の例は、2 つの NVMe 名前空間を持つ NVMe over fabrics ストレージサブシステムが接続されていることを想定しています:

```
# nvme list
```

Node	SN	Model	Namespace	Usage
Format	FW Rev			
<hr/>				
/dev/nvme0n1	a34c4f3a0d6f5cec	Linux	1	250.06 GB /
250.06 GB	512 B + 0 B	4.18.0-2		
/dev/nvme0n2	a34c4f3a0d6f5cec	Linux	2	250.06 GB /
250.06 GB	512 B + 0 B	4.18.0-2		

- 接続されているすべての NVMe サブシステムをリストアップします。

```
# nvme list-subsy

nvme-subsy0 - NQN=testnqn
 \
 +- nvme0 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme1 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-0x20000090fac7e1dd:pn-0x10000090fac7e1dd live
 +- nvme2 fc traddr=nn-0x20000090fadd5979:pn-0x10000090fadd5979 host_traddr=nn-0x20000090fac7e1de:pn-0x10000090fac7e1de live
 +- nvme3 fc traddr=nn-0x20000090fadd597a:pn-0x10000090fadd597a host_traddr=nn-0x20000090fac7e1de:pn-0x10000090fac7e1de live
```

アクティブトランSPORTタイプを確認します。例えば、**nvme0 fc** はファイバーチャネルトランSPORTで接続されていることを示し、**nvme tcp** は TCP で接続されていることを示しています。

- カーネルオプションを編集した場合は、カーネルコマンドラインでネイティブ NVMe マルチパスが有効になっているかどうかを確認します。

```
# cat /proc/cmdline

BOOT_IMAGE=[...] nvme_core.multipath=Y
```

- I/O ポリシーを変更した場合は、NVMe デバイス上で **round-robin** がアクティブな I/O ポリシーであるかどうかを確認します。

```
# cat /sys/class/nvme-subsystem/nvme-subsy0/iopolicy

round-robin
```

関連情報

- カーネルコマンドラインパラメーターの設定

17.3. NVME デバイスでの DM MULTIPATH の有効化

ネイティブ NVMe マルチパスを無効にすることで、接続された NVMe デバイスで DM マルチパスを有効にできます。

前提条件

- NVMe デバイスがシステムに接続されていることを確認します。詳細は、[NVMe over fabric デバイスの概要](#) を参照してください。

手順

- ネイティブ NVMe マルチパスが無効になっているかどうかを確認します。

```
# cat /sys/module/nvme_core/parameters/multipath
```

コマンドは以下のいずれかを表示します。

N

ネイティブ NVMe マルチパスは無効です。

Y

ネイティブ NVMe マルチパスは有効です。

2. ネイティブ NVMe マルチパスが有効になっている場合は、次のいずれかの方法を使用して無効にします。

- カーネルオプションの使用

a. カーネルのコマンドラインから **nvme_core.multipath=Y** オプションを削除しました。

```
# grubby --update-kernel=ALL --remove-args="nvme_core.multipath=Y"
```

b. 64 ビットの IBM Z アーキテクチャーでは、ブートメニューを更新します。

```
# zipl
```

c. システムを再起動します。

- カーネルモジュール設定ファイルの使用

a. **/etc/modprobe.d/nvme_core.conf** ファイルに **nvme_core multipath=Y** オプションの行が存在する場合は、それを削除します。

b. **initramfs** ファイルをバックアップします。

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%-m%-%d-%H%M%S).img
```

c. **initramfs** を再構築します。

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%-m%-%d-%H%M%S).img
# dracut --force --verbose
```

d. システムを再起動します。

3. DM マルチパスを有効にします。

```
# systemctl enable --now multipathd.service
```

4. 利用可能なすべてのパスに I/O を分配します。**/etc/multipath.conf** ファイルに以下の内容を追加します。

```
devices {
  device {
    vendor "NVME"
    product "."
  }
}
```

```

    }
    path_grouping_policy  group_by_prio
}

```



注記

DM Multipath が NVMe デバイスを管理する場合、**/sys/class/nvme-subsystem/nvme-subsy0/iopolicy** 設定ファイルは I/O ディストリビューションには影響を与えません。

5. 設定の変更を適用するために、**multipathd** サービスをリロードします。

```
# multipath -r
```

検証

- ネイティブ NVMe マルチパスが無効になっているかどうかを確認します。

```
# cat /sys/module/nvme_core/parameters/multipath
N
```

- DM Multipath が NVMe デバイスを認識しているかどうかを確認します。

```
# multipath -l

eui.00007a8962ab241100a0980000d851c8 dm-6 NVME,NetApp E-Series
size=20G features='0' hwhandler='0' wp=rw
`-- policy='service-time 0' prio=0 status=active
  |- 0:10:2:2 nvme0n2 259:3 active undef running
  `-- policy='service-time 0' prio=0 status=enabled
    |- 4:11:2:2 nvme4n2 259:28 active undef running
    `-- policy='service-time 0' prio=0 status=enabled
      |- 5:32778:2:2 nvme5n2 259:38 active undef running
    `-- policy='service-time 0' prio=0 status=enabled
      |- 6:32779:2:2 nvme6n2 259:44 active undef running
```

関連情報

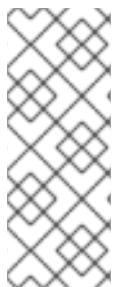
- カーネルコマンドラインパラメーターの設定
- DM Multipath の設定

第18章 ディスクスケジューラーの設定

ディスクスケジューラーは、ストレージデバイスに送信された I/O 要求を順序付けます。

スケジューラーは以下の複数の方法で設定できます。

- [Setting the disk scheduler using TuneD](#) の説明に従って、TuneD を使用してスケジューラーを設定します。
- [udev ルールを使用したディスクスケジューラーの設定](#) で説明されているように、udev を使用してスケジューラーを設定します。
- [特定ディスクに任意のスケジューラーを一時的に設定](#) で説明されているように、実行中のシステムのスケジューラーを一時的に変更します。



注記

Red Hat Enterprise Linux 8 では、ブロックデバイスはマルチキュースケジューリングのみに対応します。これにより、ブロックレイヤーのパフォーマンスを高速ソリッドステートドライブ (SSD) およびマルチコアシステムで適切に拡張できます。

Red Hat Enterprise Linux 7 以前のバージョンで利用できた従来のシングルキュースケジューラーが削除されました。

18.1. 利用可能なディスクスケジューラー

Red Hat Enterprise Linux 8 では、以下のマルチキューディスクスケジューラーに対応しています。

none

FIFO (First-in First-out) スケジューリングアルゴリズムを実装します。これにより、汎用のブロック層で単純な last-hit キャッシュを介して要求がマージされます。

mq-deadline

これにより、要求がスケジューラーに到達した時点からの要求のレイテンシーが保証されます。

mq-deadline スケジューラーは、キュー待ちの I/O リクエストを読み取りバッチまたは書き込みバッチに分類します。そして、論理ブロックアドレス (LBA) を増大順に実行するためのスケジュール設定を行います。デフォルトでは、アプリケーションは読み取り I/O 操作でブロックする可能性の方が高いため、読み取りバッチの方が書き込みバッチより優先されます。**mq-deadline** がバッチを処理すると、このプロセスは書き込み動作が待機している長さを確認して、次の読み取りバッチまたは書き込みバッチをスケジュールします。

このスケジューラーはほとんどのユースケースに適していますが、必要に応じて特に書き込み動作より読み取り動作の方が頻繁に起こるユースケースに適しています。

bfq

デスクトップシステムおよび対話式のタスクを対象とします。

bfq スケジューラーは、単一のアプリケーションがすべての帯域幅を使用しないようにします。これにより、ストレージデバイスがアイドル状態であるかのように常に応答できるようになります。デフォルトの設定では、**bfq** は、最大スループットを実現するのではなく、レイテンシーを最小限に抑えることに焦点を合わせています。

bfq は **cfq** コードに基づいています。一定のタイムスライスごとにディスクを各プロセスに付与するのではなく、セクター数単位で測定された **バジェット** をプロセスに割り当てます。

このスケジューラーは大きなファイルをコピーする際に適しており、この場合、システムが応答しなくなることはありません。

kyber

スケジューラーは、ブロック I/O レイヤーに送信されたすべての I/O 要求のレイテンシーを計算することで、レイテンシーゴールを達成するために自身を調整します。cache-misses の場合、読み込み/同期書き込みリクエストにターゲットレイテンシーを設定できます。

このスケジューラーは、NVMe、SSD などの低レイテンシーデバイスなど、高速なデバイスに適しています。

18.2. 各種ユースケースで異なるディスクスケジューラー

システムが実行するタスクに応じて、分析タスクおよびチューニングタスクの前に、以下のディスクスケジューラーがベースラインとして推奨されます。

表18.1 各種ユースケースのディスクスケジューラー

ユースケース	ディスクスケジューラー
SCSI インターフェイスを備えた従来の HDD	mq-deadline または bfq を使用します。
高速ストレージで高パフォーマンスの SSD または CPU がバインドされたシステム	特にエンタープライズアプリケーションを実行する場合は none を使用します。または kyber を使用します。
デスクトップまたはインタラクティブなタスク	bfq を使用します。
仮想ゲスト	mq-deadline を使用します。マルチキューに対応しているホストバスアダプター (HBA) ドライバーでは、 none を使用します。

18.3. デフォルトのディスクスケジューラー

ブロックデバイスは、別のスケジューラーを指定しない限り、デフォルトのディスクスケジューラーを使用します。



注記

NVMe (Non-volatile Memory Express) ブロックデバイスの場合、デフォルトのスケジューラーは **none** であり、Red Hat ではこれを変更しないことを推奨します。

カーネルは、デバイスのタイプに基づいてデフォルトのディスクスケジューラーを選択します。自動的に選択されたスケジューラーは、通常、最適な設定です。別のスケジューラーが必要な場合は、Red Hat では、**udev** ルールまたは **TuneD** アプリケーションを使用して設定することを推奨しています。選択したデバイスを一致させ、それらのデバイスのスケジューラーのみを切り替えます。

18.4. アクティブなディスクスケジューラーの決定

この手順では、特定のブロックデバイスで現在アクティブなディスクスケジューラーを確認します。

手順

- `/sys/block/device/queue/scheduler` ファイルの内容を読み取ります。

```
# cat /sys/block/device/queue/scheduler
```

```
[mq-deadline] kyber bfq none
```

ファイル名の `device` を、 `sdc` などのブロックデバイス名に置き換えます。

アクティブなスケジューラーは、角括弧([])にリスト表示されます。

18.5. TUNED を使用したディスクスケジューラーの設定

この手順では、選択したブロックデバイスに特定のディスクスケジューラーを設定するTuned プロファイルを作成して有効にします。この設定は、システムを再起動しても持続します。

以下のコマンドと設定で、以下の内容を置き換えます。

- `device` をブロックデバイスの名前に置き換えます (例: `sdf`)。
- `selected-scheduler` を、デバイスに設定するディスクスケジューラーに置き換えます (例: `bfq`)。

前提条件

- Tuned サービスがインストールされ、有効になっている。詳細は、 [Tuned のインストールと有効化](#) を参照してください。

手順

1. 必要に応じて、プロファイルのベースとなる既存のTunedプロファイルを選択します。利用可能なプロファイルのリストは、 [RHELとともに配布される Tuned プロファイル](#) を参照してください。

現在アクティブなプロファイルを確認するには、次のコマンドを実行します。

```
$ tuned-adm active
```

2. Tuned プロファイルを保持する新しいディレクトリーを作成します。

```
# mkdir /etc/tuned/my-profile
```

3. 選択したブロックデバイスのシステム固有の識別子を見つけます。

```
$ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'
```

```
ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic- SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```



注記

この例のコマンドは、指定したブロックデバイスに関連付けられた World Wide Name (WWN) またはシリアル番号として識別されるすべての値を返します。WWN を使用することが推奨されますが、WWN は特定のデバイスで常に利用できる訳ではなく、コマンド例で返される値は、**デバイスのシステム固有の ID**として使用することが許容されます。

4. **/etc/tuned/my-profile/tuned.conf** 設定ファイルを作成します。このファイルで、以下のオプションを設定します。

- 必要に応じて、既存のプロファイルを追加します。

```
[main]
include=existing-profile
```

- WWN 識別子に一致するデバイスに対して選択したディスクスケジューラーを設定します。

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

ここでは、以下のようになります。

- **IDNAME** を、使用されている識別子名に置き換えます (例:**ID_WWN**)。
 - **device system unique id** を、選択した識別子の値に置き換えます (例:**0x5002538d00000000**)。
- devices_udev_regex** オプションで複数のデバイスに一致させるには、識別子を括弧で囲み、垂直バーで区切れます。

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. プロファイルを有効にします。

```
# tuned-adm profile my-profile
```

検証

1. TuneD プロファイルがアクティブで、適用されていることを確認します。

```
$ tuned-adm active
```

```
Current active profile: my-profile
```

```
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile.
See TuneD log file ('/var/log/tuned/tuned.log') for details.
```

2. **/sys/block/device/queue/scheduler** ファイルの内容を読み取ります。

```
# cat /sys/block/device/queue/scheduler
```

```
[mq-deadline] kyber bfq none
```

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

アクティブなスケジューラーは、角括弧([])にリスト表示されます。

関連情報

- [TuneD プロファイルのカスタマイズ](#)

18.6. UDEV ルールを使用したディスクスケジューラーの設定

この手順では、**udev** ルールを使用して、特定ブロックデバイスに、特定のディスクスケジューラーを設定します。この設定は、システムを再起動しても持続します。

以下のコマンドと設定で、以下の内容を置き換えます。

- **device** をブロックデバイスの名前に置き換えます (例: **sdf**)。
- **selected-scheduler** を、デバイスに設定するディスクスケジューラーに置き換えます (例: **bfq**)。

手順

1. ブロックデバイスのシステム固有の識別子を見つけます。

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'  
E: ID_WWN=0x5002538d00000000  
E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0  
E: ID_SERIAL_SHORT=20120501030900000
```



注記

この例のコマンドは、指定したブロックデバイスに関連付けられた World Wide Name (WWN) またはシリアル番号として識別されるすべての値を返します。WWN を使用することが推奨されますが、WWN は特定のデバイスで常に利用できる訳ではなく、コマンド例で返される値は、デバイスのシステム固有の ID として使用することが許容されます。

2. **udev** ルールを設定します。以下の内容で **/etc/udev/rules.d/99-scheduler.rules** ファイルを作成します。

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

ここでは、以下のようになります。

- **IDNAME** を、使用されている識別子名に置き換えます (例: **ID_WWN**)。
- **device system unique id** を、選択した識別子の値に置き換えます (例: **0x5002538d00000000**)。

3. **udev** ルールを再読み込みします。

```
# udevadm control --reload-rules
```

4. スケジューラー設定を適用します。

```
# udevadm trigger --type=devices --action=change
```

検証

- アクティブなスケジューラーを確認します。

```
# cat /sys/block/device/queue/scheduler
```

18.7. 特定ディスクに任意のスケジューラーを一時的に設定

この手順では、特定のブロックデバイスに、特定のディスクスケジューラーを設定します。この設定は、システムを再起動すると元に戻ります。

手順

- 選択したスケジューラーの名前を、**/sys/block/device/queue/scheduler** ファイルに書き込みます。

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

検証

- スケジューラーがデバイスでアクティブになっていることを確認します。

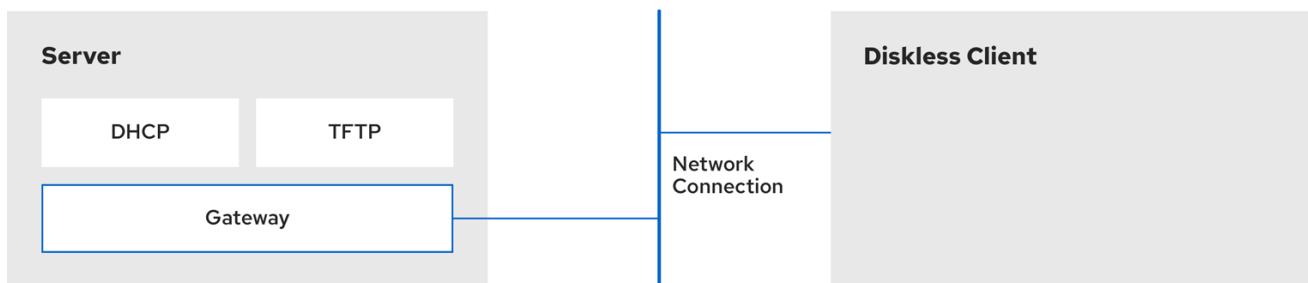
```
# cat /sys/block/device/queue/scheduler
```

第19章 リモートディスクレスシステムの設定

ネットワーク環境では、リモートディスクレスシステムをデプロイすることで、同一の設定で複数のクライアントをセットアップできます。現在の Red Hat Enterprise Linux サーバーバージョンを使用すると、これらのクライアントのハードドライブのコストを節約し、別のサーバーにゲートウェイを設定できます。

次の図は、Dynamic Host Configuration Protocol (DHCP) および Trivial File Transfer Protocol (TFTP) サービスを介したディスクレスクライアントとサーバーの接続を示しています。

図19.1 リモートディスクレスシステム設定のダイアグラム



269_RHEL_0822

19.1. リモートディスクレスシステムの環境の準備

リモートディスクレスシステムの実装を続行できるように環境を準備します。リモートディスクレスシステムの起動には次のサービスが必要です。

- `tftp-server` によって提供される Trivial File Transfer Protocol (TFTP) サービス。システムは、`tftp` サービスを使用して、Preboot Execution Environment (PXE) ローダーを通じてネットワーク経由でカーネルイメージと初期 RAM ディスク `initrd` を取得します。
- `dhcpc` によって提供される Dynamic Host Configuration Protocol (DHCP) サービス。

前提条件

- `xinetd` パッケージをインストールしている。
- ネットワーク接続が設定されている。

手順

1. `dracut-network` パッケージをインストールします。

```
# yum install dracut-network
```

2. `/etc/dracut.conf.d/network.conf` ファイルに次の行を追加します。

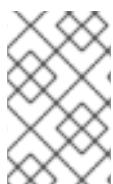
```
add_dracutmodules+=" nfs "
```

3. 次の順序でサービスを設定して、環境内のリモートディスクレスシステムが正しく機能するようにします。

- a. TFTP サービスを設定します。詳細は、[ディスクレスクライアントの TFTP サービスの設定](#)を参照してください。
- b. DHCP サーバーを設定します。詳細は、[ディスクレスクライアントの DHCP サーバーの設定](#)を参照してください。
- c. ネットワークファイルシステム (NFS) とエクスポートしたファイルシステムを設定します。詳細は、[ディスクレスクライアントのエクスポートしたファイルシステムの設定](#)を参照してください。

19.2. ディスクレスクライアントの TFTP サービスの設定

リモートディスクレスシステムを環境内で正しく機能させるには、まずディスクレスクライアントの Trivial File Transfer Protocol (TFTP) サービスを設定する必要があります。



注記

この設定は、Unified Extensible Firmware Interface (UEFI) 経由では起動しません。UEFI ベースのインストールの場合は、[UEFI ベースのクライアント向けに TFTP サーバーを設定する](#)を参照してください。

前提条件

- 次のパッケージがインストールされている。
 - **tftp-server**
 - **syslinux**
 - **xinetd**

手順

1. TFTP サービスを有効にします。

```
# systemctl enable --now tftp
```

2. tftp のルートディレクトリーに pxelinux ディレクトリーを作成します。

```
# mkdir -p /var/lib/tftpboot/pxelinux/
```

3. /usr/share/syslinux/pxelinux.0 ファイルを /var/lib/tftpboot/pxelinux/ ディレクトリーにコピーします。

```
# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/pxelinux/
```

4. /usr/share/syslinux/ldlinux.c32 を /var/lib/tftpboot/pxelinux/ にコピーします。

```
# cp /usr/share/syslinux/ldlinux.c32 /var/lib/tftpboot/pxelinux/
```

5. tftp のルートディレクトリーに pxelinux.cfg ディレクトリーを作成します。

```
# mkdir -p /var/lib/tftpboot/pxelinux/pxelinux.cfg/
```

検証

- サービス **tftp** のステータスを確認します。

```
# systemctl status tftp
...
Active: active (running)
...
```

19.3. ディスクレスクライアントの DHCP サーバーの設定

リモートディスクレスシステムを正しく機能させるには、いくつかのサービスをあらかじめインストールしておく必要があります。

前提条件

- Trivial File Transfer Protocol (TFTP) サービスがインストールされている。
- 次のパッケージがインストールされている。
 - dhcp-server**
 - xinetd**
- ディスクレスクライアントの **tftp** サービスが設定されている。詳細は、[ディスクレスクライアントの TFTP サービスの設定](#) を参照してください。

手順

- /etc/dhcp/dhcpd.conf** ファイルに次の設定を追加して、DHCP サーバーをセットアップし、ブート用の Preboot Execution Environment (PXE) を有効にします。

```
option space pxelinux;
option pxelinux.magic code 208 = string;
option pxelinux.configfile code 209 = text;
option pxelinux.pathprefix code 210 = text;
option pxelinux.reboottime code 211 = unsigned integer 32;
option architecture-type code 93 = unsigned integer 16;

subnet 192.168.205.0 netmask 255.255.255.0 {
    option routers 192.168.205.1;
    range 192.168.205.10 192.168.205.25;

    class "pxeclients" {
        match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
        next-server 192.168.205.1;

        if option architecture-type = 00:07 {
            filename "BOOTX64.efi";
        } else {
            filename "pxelinux/pxelinux.0";
        }
    }
}
```

DHCP 設定は、リース時間や固定アドレスの設定など、環境によって異なる場合があります。詳細は、[DHCP サービスの提供](#) を参照してください。



注記

libvirt 仮想マシンをディスクレスクライアントとして使用する場合、**libvirt** デーモンが DHCP サービスを提供し、スタンドアロン DHCP サーバーは使用されません。この状況では、**libvirt** ネットワーク設定の **virsh net-edit** で **bootp file=<filename>** オプションを使用して、ネットワークブートを有効にする必要があります。

2. **dhcpd.service** を有効にします。

```
# systemctl enable --now dhcpd.service
```

検証

- サービス **dhcpd.service** のステータスを確認します。

```
# systemctl status dhcpd.service
...
Active: active (running)
...
```

19.4. ディスクレスクライアントのエクスポートしたファイルシステムの設定

環境にリモートディスクレスシステムを設定する一環として、ディスクレスクライアント用にエクスポートしたファイルシステムを設定する必要があります。

前提条件

- ディスクレスクライアントの **tftp** サービスが設定されている。[ディスクレスクライアントの TFTP サービスの設定](#) セクションを参照してください。
- Dynamic Host Configuration Protocol (DHCP) サーバーが設定されている。[ディスクレスクライアントの DHCP サーバーの設定](#) セクションを参照してください。

手順

- /etc/exports** ディレクトリーにルートディレクトリーを追加して、ルートディレクトリーをエクスポートするようにネットワークファイルシステム (NFS) サーバーを設定します。手順の詳細は、[NFS サーバーのデプロイ](#) を参照してください。
- 完全にディスクレスのクライアントに対応できるように、Red Hat Enterprise Linux の完全なバージョンをルートディレクトリーにインストールします。これを行うには、新しいベースシステムをインストールするか、既存のインストールのクローンを作成します。
 - exported-root-directory** をエクスポートしたファイルシステムへのパスに置き換えて、エクスポートした場所に Red Hat Enterprise Linux をインストールします。

```
# yum install @Base kernel dracut-network nfs-utils --installroot=exported-root-directory --releasever=/
```

- **releasever** オプションを `/` に設定すると、`releasever` がホスト `(/)` システムから検出されます。

- **rsync** ユーティリティーを使用して、実行中のシステムと同期します。

```
# rsync -a -e ssh --exclude='/proc/' --exclude='/sys/' example.com:/ exported-root-directory
```

- `example.com` は、`rsync` ユーティリティーで同期する実行中のシステムのホスト名に置き換えます。
- `exported-root-directory` を、エクスポートしたファイルシステムへのパスに置き換えます。
このオプションには、実行中の別のシステムが必要です。これは、このコマンドでサーバーにクローンを作成します。

3. ファイルシステムをディスクレスクライアントで使用する前に、エクスポートの準備が完了しているファイルシステムを設定します。

- a. ディスクレスクライアントがサポートするカーネル (`vmlinuz-kernel-version_pass:attributes`) を `tftp` ブートディレクトリーにコピーします。

```
# cp /exported-root-directory/boot/vmlinuz-kernel-version /var/lib/tftpboot/pixelinux/
```

- b. `initramfs-kernel-version.img` ファイルをローカルに作成し、NFS をサポートするエクスポートされたルートディレクトリーに移動します。

```
# dracut --add nfs initramfs-kernel-version.img kernel-version
```

以下に例を示します。

```
# dracut --add nfs /exports/root/boot/initramfs-5.14.0-202.el9.x86_64.img 5.14.0-202.el9.x86_64
```

現在実行中のカーネルバージョンを使用し、既存のイメージを上書きして `initrd` を作成する例を以下に示します。

```
# dracut -f --add nfs "boot/initramfs-$(uname -r).img" "$(uname -r)"
```

- c. `initrd` のファイル権限を **0644** に変更します。

```
# chmod 0644 /exported-root-directory/boot/initramfs-kernel-version.img
```

警告



`initrd` のファイル権限を変更しないと、`pxelinux.0` ブートローダーが "file not found" エラーを表示して失敗します。

- d. 作成された **initramfs-kernel-version.img** ファイルを **tftp** ブートディレクトリーにコピーします。

```
# cp /exported-root-directory/boot/initramfs-kernel-version.img
/var/lib/tftpboot/pxelinux/
```

- e. **/var/lib/tftpboot/pxelinux/pxelinux.cfg/default** ファイルに次の設定を追加して、**initrd** とカーネルを使用するためのデフォルトのブート設定を編集します。

```
default menu.c32
prompt 0
menu title PXE Boot Menu
ontimeout rhel8-over-nfsv4.2
timeout 120
label rhel8-over-nfsv4.2
menu label Install diskless rhel8{} nfsv4.2{}
kernel $vmlinuz
append initrd=$initramfs root=nfs4:$nfsserv:/vers=4.2,rw rw panic=60 ipv6.disable=1
console=tty0 console=ttyS0,115200n8
label rhel8-over-nfsv3
menu label Install diskless rhel8{} nfsv3{}
kernel $vmlinuz
append initrd=$initramfs root=nfs:$nfsserv:$nfsroot:vers=3,rw rw panic=60
ipv6.disable=1 console=tty0 console=ttyS0,115200n8
```

この設定は、ディスクレスクライアントのルートに、**/exported-root-directory** エクスポートファイルシステムを読み取り/書き込み形式でマウントするように指示します。

- a. オプション: **/var/lib/tftpboot/pxelinux/pxelinux.cfg/default** ファイルを次の設定で編集して、ファイルシステムを **read-only** 形式でマウントします。

```
default rhel8

label rhel8
kernel vmlinuz-kernel-version
append initrd=initramfs-kernel-version.img root=nfs:server-ip:/exported-root-
directory ro
```

- b. NFS サーバーを再起動します。

```
# systemctl restart nfs-server.service
```

これで、NFS 共有をディスクレスクライアントにエクスポートできるようになりました。これらのクライアントは、Preboot Execution Environment (PXE) 経由でネットワーク経由で起動できます。

19.5. リモートディスクレスシステムの再設定

パッケージのインストール、サービスの再起動、または問題のデバッグを行う場合は、システムを再設定できます。

前提条件

- エクスポートしたファイルシステムで **no_root_squash** オプションが有効になっている。

手順

- ユーザーパスワードを変更します。
 - コマンドラインを `/exported/root/directory` に変更します。

```
# chroot /exported/root/directory /bin/bash
```

- 必要なユーザーのパスワードを変更します。

```
# passwd <username>
```

`<username>` は、パスワードを変更する実際のユーザーに置き換えます。

- コマンドラインを終了します。
- リモートディスクレスシステムにソフトウェアをインストールします。

```
# yum install <package> --installroot=/exported/root/directory --releasever=/ --config /etc/dnf/dnf.conf --setopt=reposdir=/etc/yum.repos.d/
```

`<package>` を、インストールする実際のパッケージに置き換えます。

- 2つの個別のエクスポートを設定して、リモートディスクレスシステムを `/usr` と `/var` に分割します。詳細は、[NFS サーバーのデプロイ](#) を参照してください。

19.6. リモートディスクレスシステムのロードに関する一般的な問題のトラブルシューティング

以前の設定を利用すると、リモートディスクレスシステムのロード中に問題が発生する可能性があります。以下に、Red Hat Enterprise Linux サーバーで最も一般的な問題とそのトラブルシューティングの例をいくつか示します。

例19.1 クライアントが IP アドレスを取得しない

1. サーバー上で Dynamic Host Configuration Protocol (DHCP) サービスが有効になっているかどうかを確認します。

- a. `dhcp.service` が実行しているかどうかを確認します。

```
# systemctl status dhcpcd.service
```

- b. `dhcp.service` が非アクティブな場合は、有効にして起動します。

```
# systemctl enable dhcpcd.service
# systemctl start dhcpcd.service
```

- c. ディスクレスクライアントを再起動します。

- d. DHCP 設定ファイル `/etc/dhcp/dhcpcd.conf` を確認します。詳細は、[ディスクレスクライアントの DHCP サーバーの設定](#) を参照してください。

2. ファイアウォールポートが開いているかどうかを確認します。

- a. **dhcp.service** がアクティブなサービスにリストされているかどうかを確認します。

```
# firewall-cmd --get-active-zones
# firewall-cmd --info-zone=public
```

- b. **dhcp.service** がアクティブなサービスにリストされていない場合は、リストに追加します。

```
# firewall-cmd --add-service=dhcp --permanent
```

- c. **nfs.service** がアクティブなサービスに記載されているかどうかを確認します。

```
# firewall-cmd --get-active-zones
# firewall-cmd --info-zone=public
```

- d. **nfs.service** がアクティブなサービスに記載されていない場合は、これをリストに追加します。

```
# firewall-cmd --add-service=nfs --permanent
```

例19.2 リモートディスクレスシステムの起動時にファイルを使用できない

1. ファイルが **/var/lib/tftpboot/** ディレクトリーにあるかどうかを確認します。
2. ファイルがディレクトリー内にある場合は、ファイルに次の権限があるかどうかを確認します。

```
# chmod 644 pxelinux.0
```

3. ファイアウォールポートが開いているかどうかを確認します。

例19.3 kernel/initrd のロード後にシステムの起動に失敗した

1. サーバーで NFS サービスが有効になっているかどうかを確認します。

- a. **nfs.service** が実行中かどうかを確認します。

```
# systemctl status nfs.service
```

- b. **nfs.service** が非アクティブな場合は、それを起動して有効にする必要があります。

```
# systemctl start nfs.service
# systemctl enable nfs.service
```

2. **/var/lib/tftpboot/pxelinux.cfg/** ディレクトリーでパラメーターが正しいかどうかを確認してください。詳細は、[ディスクレスクライアントのエクスポートしたファイルシステムの設定](#)を参照してください。

3. ファイアウォールポートが開いているかどうかを確認します。

第20章 RAID の管理

Redundant Array of Independent Disks (RAID) を使用して、複数のドライブにデータを保存できます。ドライブに障害が発生した場合に、データの損失を回避することができます。

20.1. RAID の概要

RAID では、HDD、SSD、または NVMe などの複数のデバイスをアレイに組み合わせて、1つの大型で高価なドライブでは達成できないパフォーマンスまたは冗長性の目標を達成することができます。このデバイスのアレイは、1つの論理ストレージユニットまたはドライブとしてコンピューターに表示されます。

RAID は、レベル 0、1、4、5、6、10、リニアなどのさまざまな設定に対応します。RAID は、ディスクストライピング (RAID レベル 0)、ディスクミラーリング (RAID レベル 1)、およびパリティーによるディスクストライピング (RAID レベル 4、5、および 6) などの技術を使用して、冗長性、低遅延、帯域の増加、ハードディスクのクラッシュからの回復能力の最大化を達成します。

RAID は、データを一貫して使用するチャンク (通常は 256KB または 512KB、他の値は受け入れ可能) に分割することで、アレイ内の各デバイスにデータを分散します。採用されている RAID レベルに従って、これらのチャンクを RAID アレイ内のハードドライブに書き込みます。データの読み取り中はプロセスが逆になり、アレイ内の複数のデバイスが実際には1つの大きなドライブであるかのように見えます。

RAID テクノロジーは、大量のデータを管理するユーザーにとって有益です。RAID を導入する主な理由は次のとおりです。

- 速度を高める
- 1台の仮想ディスクを使用してストレージ容量を増やす
- ディスク障害によるデータ損失を最小限に抑える
- RAID レイアウトおよびレベルのオンライン変換

20.2. RAID のタイプ

RAID のタイプとして考えられるのは以下の通りです。

ファームウェア RAID

ファームウェア RAID (ATARAID とも呼ばれる) とは、ソフトウェア RAID の種類で、ファームウェアベースのメニューを使用して RAID セットを設定できます。このタイプの RAID で使用されるファームウェアは BIOS にもフックされるため、その RAID セットから起動できます。異なるベンダーは、異なるオンディスクメタデータ形式を使用して、RAID セットのメンバーをマークします。Intel Matrix RAID は、ファームウェア RAID システムの一例を示しています。

ハードウェア RAID

ハードウェアベースのアレイは、RAID サブシステムをホストとは別に管理します。ホストに対して RAID アレイごとに複数のデバイスが表示される場合があります。

ハードウェア RAID デバイスは、システムの内部または外部になる場合があります。内部デバイスは、一般的には、RAID タスクをオペレーティングシステムに対して透過的に処理する専用のコントローラーカードで構成されています。外部デバイスは、一般的には SCSI、ファイバーチャネル、iSCSI、InfiniBand などの高速ネットワーク相互接続を介してシステムに接続し、システムへの論理ユニットなどのボリュームを提示します。

RAID コントローラーカードは、オペレーティングシステムへの SCSI コントローラーのように動作

し、実際のドライブ通信をすべて処理します。ドライブを通常の SCSI コントローラーと同様に RAID コントローラーに接続し、RAID コントローラーの設定に追加できます。オペレーティングシステムはこの違いを認識できません。

ソフトウェア RAID

ソフトウェア RAID は、カーネルブロックデバイスコード内にさまざまな RAID レベルを実装します。高価なディスクコントローラーカードやホットスワップシャーシが不要なため、可能な限り安価なソリューションを提供します。ホットスワップシャーシを使用すると、システムの電源を切らずにハードドライブを取り外すことができます。ソフトウェア RAID は、SATA、SCSI、NVMe などの Linux カーネルが対応しているブロックストレージでも機能します。現在高速 CPU では、ハイエンドのストレージデバイスを使用する場合以外は、ソフトウェア RAID は通常ハードウェア RAID の規模を上回します。

Linux カーネルにはマルチデバイス (MD) ドライバーが含まれているため、RAID ソリューションは完全にハードウェアに依存しません。ソフトウェアベースのアレイのパフォーマンスは、サーバーの CPU パフォーマンスと負荷によって異なります。

Linux ソフトウェア RAID スタックの主な機能は次のとおりです。

- マルチスレッド設計
- 再構築なしで Linux マシン間でのアレイの移植性
- アイドルシステムリソースを使用したバックグラウンドのアレイ再構築
- ホットスワップドライブのサポート
- CPU の自動検出は、ストリーミング SIMD (Single Instruction Multiple Data) サポートなどの特定の CPU 機能を利用するための自動 CPU 検出
- アレイ内のディスク上にある不良セクターの自動修正
- RAID データの整合性を定期的にチェックしアレイの健全性を確保
- 重要なイベントが発生すると、指定された電子メールアドレスに送信される電子メールアラートによるアレイのプロアクティブな監視
- システムのクラッシュ後にアレイ全体を再同期する代わりに、ディスクのどの部分を再同期する必要があるかをカーネルが正確に把握できるようにすることで、再同期イベントの速度を大幅に向上させる書き込みが集中しているビットマップ



注記

再同期は、既存の RAID 内のデバイス間でデータを同期して冗長性を実現するプロセスです。

- チェックポイントを再同期して、再同期中にコンピューターを再起動すると、起動時に再同期が中断したところから再開され、最初からやり直すことはありません。
- インストール後にアレイのパラメーターを変更する機能は、再形成と呼ばれます。たとえば、新しいデバイスを追加しても、4つのディスクの RAID5 アレイを5つのディスク RAID5 アレイに増大させることができます。この拡張操作はライブで行うため、新しいアレイで再インストールする必要はありません。
- 再成形は、RAID アルゴリズム、RAID アレイタイプのサイズ (RAID4、RAID5、RAID6、RAID10 など) の変更に対応しています。

- テイクオーバーは、RAID0 から RAID6 などの RAID レベルの変換をサポートしています。
- クラスターのストレージソリューションである Cluster MD は、RAID1 ミラーリングの冗長性をクラスターに提供します。現在、RAID1 のみがサポートされています。

20.3. RAID レベルとリニアサポート

レベル 0、1、4、5、6、10、リニアなど、RAID 別の対応設定は以下のとおりです。

レベル 0

ストライピングとも呼ばれる RAID レベル 0 は、パフォーマンス指向のストライピングデータマッピング技術です。これは、アレイに書き込まれるデータがストライプに分割され、アレイのメンバーディスク全体に書き込まれることを意味します。これにより低い固有コストで高い I/O パフォーマンスを実現できますが、冗長性は提供されません。

RAID レベル 0 実装は、アレイ内の最小デバイスのサイズまで、メンバーデバイス全体にだけデータをストライピングします。つまり、複数のデバイスのサイズが少し異なる場合、それぞれのデバイスは最小ドライブと同じサイズであるかのように処理されます。したがって、レベル 0 アレイの共通ストレージ容量は、すべてのディスクの合計容量です。メンバーディスクのサイズが異なる場合、RAID0 は使用可能なゾーンを使用して、それらのディスクのすべての領域を使用します。

レベル 1

RAID レベル 1(ミラーリング)は、アレイの各メンバーディスクに同一のデータを書き込み、ミラーリングされたコピーを各ディスクに残すことによって冗長性を提供します。ミラーリングは、データの可用性の単純化と高レベルにより、いまでも人気があります。レベル 1 は 2 つ以上のディスクと連携して、非常に優れたデータ信頼性を提供し、読み取り集中型のアプリケーションに対してパフォーマンスが向上しますが、比較的コストが高くなります。

RAID レベル 1 は、アレイ内のすべてのディスクに同じ情報を書き込むためコストがかかります。これにより、データの信頼性が提供されますが、レベル 5 などのパリティーベースの RAID レベルよりもスペース効率が大幅に低下します。ただし、この領域の非効率性にはパフォーマンス上の利点があります。パリティーベースの RAID レベルは、パリティーを生成するためにかなり多くの CPU 電力を消費しますが、RAID レベル 1 は単に同じデータを、CPU オーバーヘッドが非常に少ない複数の RAID メンバーに複数回書き込むだけです。そのため、RAID レベル 1 は、ソフトウェア RAID が使用されているマシンや、マシンの CPU リソースが一貫して RAID アクティビティー以外の操作でアレイ化されます。

レベル 1 アレイのストレージ容量は、ハードウェア RAID 内でミラーリングされている最小サイズのハードディスクの容量と同じか、ソフトウェア RAID 内でミラーリングされている最小のパーティションと同じ容量になります。レベル 1 の冗長性は、すべての RAID タイプの中で最も高いレベルであり、アレイは 1 つのディスクのみで動作できます。

レベル 4

レベル 4 は、1 つのディスクドライブでパリティー連結を使用して、データを保護します。パリティー情報は、アレイ内の残りのメンバーディスクのコンテンツに基づいて計算されます。この情報は、アレイ内のいずれかのディスクに障害が発生した場合にデータの再構築に使用できます。その後、再構築されたデータを使用して、交換前に失敗したディスクに I/O 要求に対応でき、交換後に失敗したディスクを接続します。

パリティー専用ディスクは、RAID アレイへのすべての書き込みトランザクションにおいて固有のボトルネックとなるため、ライトバックキャッシングなどの付随する技術なしにレベル 4 が使用されることはありません。または、システム管理者が意図的にこのボトルネックを考慮してソフトウェア RAID デバイスを設計している特定の状況下で使用されます。たとえば、アレイにデータが格納されると書き込みトランザクションがほとんどないようなアレイです。RAID レベル 4 にはほとんど使用されないため、Anaconda ではこのオプションとしては使用できません。ただし、実際には必要な場合は、ユーザーが手動で作成できます。

ハードウェア RAID レベル 4 のストレージ容量は、最小メンバーパーティションの容量にパーティションの数を掛けて 1 を引いた値に等しくなります。RAID レベル 4 アレイのパフォーマンスは常に非対称です。つまり、読み込みは書き込みを上回ります。これは、パリティーを生成するときに書き込み操作が余分な CPU リソースとメインメモリー帯域幅を消費し、実際のデータをディスクに書き込むときに余分なバス帯域幅も消費するためです。これは、データだけでなくパリティーも書き込むためです。読み取り操作は、アレイが劣化状態にない限り、データを読み取るだけでパリティーを読み取る必要はありません。その結果、読み取り操作では、通常の操作条件下で同じ量のデータ転送を行う場合でも、ドライブおよびコンピューターのバス全体に生成されるトラフィックが少なくなります。

レベル 5

これは RAID の最も一般的なタイプです。RAID レベル 5 は、アレイのすべてのメンバーディスクドライブにパリティーを分散することにより、レベル 4 に固有の書き込みボトルネックを排除します。パリティー計算プロセス自体のみがパフォーマンスのボトルネックです。最近の CPU はパリティーを非常に高速に計算できます。しかし、RAID 5 アレイに多数のディスクを使用していて、すべてのデバイスの合計データ転送速度が十分に高い場合、パリティー計算がボトルネックになる可能性があります。

レベル 5 のパフォーマンスは非対称であり、読み取りは書き込みよりも大幅に優れています。RAID レベル 5 のストレージ容量は、レベル 4 と同じです。

レベル 6

パフォーマンスではなくデータの冗長性と保存が最重要事項であるが、レベル 1 の領域の非効率性が許容できない場合は、これが RAID の一般的なレベルです。レベル 6 では、複雑なパリティースキームを使用して、アレイ内の 2 つのドライブから失われたドライブから復旧できます。複雑なパリティースキームにより、ソフトウェア RAID デバイスで CPU 幅が大幅に高くなり、書き込みトランザクションの際に増大度が高まります。したがって、レベル 6 はレベル 4 や 5 よりもパフォーマンスにおいて、非常に非対称です。

RAID レベル 6 アレイの合計容量は、RAID レベル 5 および 4 と同様に計算されますが、デバイス数から追加パリティーストレージ領域用に 2 つのデバイス (1 ではなく) を引きます。

レベル 10

この RAID レベルでは、レベル 0 のパフォーマンスとレベル 1 の冗長性を組み合わせます。また、2 台以上のデバイスを使用するレベル 1 アレイの無駄なスペースをある程度削減することができます。レベル 10 では、たとえば、データごとに 2 つのコピーのみを格納するように設定された 3 ドライブアレイを作成することができます。これにより、全体用のアレイサイズを最小デバイスのみと同じサイズ (3 つのデバイス、レベル 1 アレイなど) ではなく、最小デバイスのサイズの 1.5 倍にすることができます。これにより、CPU プロセスの使用量が RAID レベル 6 のようにパリティーを計算するのを防ぎますが、これは領域効率が悪くなります。

RAID レベル 10 の作成は、インストール時には対応していません。インストール後に手動で作成できます。

リニア RAID

リニア RAID は、より大きな仮想ドライブを作成するドライブのグループ化です。

リニア RAID では、あるメンバードライブからチャunkが順次割り当てられます。最初のドライブが完全に満杯になったときにのみ次のドライブに移動します。これにより、メンバードライブ間の I/O 操作が分割される可能性はないため、パフォーマンスの向上は見られません。リニア RAID は冗長性がなく、信頼性は低下します。メンバードライブが 1 台でも故障すると、アレイ全体が使用できなくなり、データが失われる可能性があります。容量はすべてのメンバーディスクの合計になります。

20.4. サポート対象の RAID 変換

RAID レベルを別のレベルに変換することができます。たとえば、RAID5 から RAID10 への変換はできますが、RAID10 から RAID5 への変換はできません。次の表は、サポートされている RAID 変換を示しています。

RAID 変換レベル	変換手順	注記
RAID レベル 0 から RAID レベル 4	<pre># mdadm --grow /dev/md0 --level=4 -n3 --add /dev/vdd</pre>	MD アレイに少なくとも 3 つのディスクが必要なため、ディスクを追加する必要があります。
RAID レベル 0 から RAID レベル 5	<pre># mdadm --grow /dev/md0 --level=5 -n3 --add /dev/vdd</pre>	MD アレイに少なくとも 3 つのディスクが必要なため、ディスクを追加する必要があります。
RAID レベル 0 から RAID レベル 10	<pre># mdadm --grow /dev/md0 --level 10 -n 4 --add /dev/vd[ef]</pre>	MD アレイに 2 つのディスクを追加する必要があります。
RAID レベル 1 から RAID レベル 0	<pre># mdadm --grow /dev/md0 -l0</pre>	
RAID レベル 1 から RAID レベル 5	<pre># mdadm --grow /dev/md0 --level=5</pre>	
RAID レベル 4 から RAID レベル 0	<pre># mdadm --grow /dev/md0 --level=0</pre>	
RAID レベル 4 から RAID レベル 5	<pre># mdadm --grow /dev/md0 --level=5</pre>	
RAID レベル 5 から RAID レベル 0	<pre># mdadm --grow /dev/md0 --level=0</pre>	

RAID 変換レベル	変換手順	注記
RAID レベル 5 から RAID レベル 1	<pre># mdadm -CR /dev/md0 -l5 -n3 /dev/sd[abc] --assume-clean --size 1G # mdadm -D /dev/md0 grep Level</pre> <pre># mdadm --grow /dev/md0 --array-size 1048576</pre> <pre># mdadm --grow -n 2 /dev/md0 -- backup=internal</pre> <pre># mdadm --grow -l1 /dev/md0</pre> <pre># mdadm -D /dev/md0 grep Level</pre>	
RAID レベル 5 から RAID レベル 4	<pre># mdadm --grow /dev/md0 --level=4</pre>	
RAID レベル 5 から RAID レベル 6	<pre># mdadm --grow /dev/md0 --level=6 --add /dev/vde</pre>	
RAID レベル 5 から RAID レベル 10	<pre># mdadm --grow /dev/md0 --level=0 # mdadm --grow /dev/md0 --level=10 --add /dev/vde /dev/vdf</pre>	<p>RAID レベル 5 から RAID レベル 10 への変換は、次の 2 つのステップで行います。</p> <ol style="list-style-type: none"> 1. RAID レベル 0 に変換します。 2. RAID10 に変換するときに 2 つのディスクを追加します。
RAID レベル 6 から RAID レベル 5	<pre># mdadm --grow /dev/md0 --level=5</pre>	
RAID レベル 10 から RAID レベル 0	<pre># mdadm --grow /dev/md0 --level=0</pre>	



注記

RAID 5 から RAID0 および RAID4 への変換は、**ALGORITHM_PARITY_N** レイアウトでのみ可能です。

RAID レベルを変換した後、**mdadm --detail /dev/md0** または **cat /proc/mdstat** コマンドを使用して変換を確認します。

関連情報

- システム上の **mdadm(8)** man ページ

20.5. RAID サブシステム

RAID は次のサブシステムで構成されます。

ハードウェア RAID コントローラードライバー

ハードウェア RAID コントローラーに固有の RAID サブシステムはありません。特別な RAID チップセットを使用するため、ハードウェア RAID コントローラには独自のドライバが付属しています。これらのドライバーを使用すると、システムは RAID セットを通常のディスクとして検出します。

mdraid

mdraid サブシステムはソフトウェア RAID ソリューションとして設計されました。これは、Red Hat Enterprise Linux のソフトウェア RAID の推奨ソリューションでもあります。このサブシステムでは独自のメタデータ形式が使用され、通常はネイティブの MD メタデータと呼ばれます。

mdraid は、外部メタデータとして知られる他のメタデータ形式にも対応しています。Red Hat Enterprise Linux 8 は **mdraid** と外部メタデータを使用して、Intel Rapid Storage (ISW) または Intel Matrix Storage Manager (IMSM) セットと Storage Networking Industry Association (SNIA) Disk Drive Format (DDF) にアクセスします。**mdraid** サブシステムセットは、**mdadm** ユーティリティによって設定および制御されます。

20.6. インストール中のソフトウェア RAID の作成

Redundant Arrays of Independent Disks (RAID) デバイスは、パフォーマンスを向上させ、一部の設定ではより優れたフォールトトレランスを提供するように配置された複数のストレージデバイスから構築されます。RAID デバイスの作成は 1 つのステップで終わり、必要に応じてディスクを追加または削除できます。システムでは、1 つの物理ディスクに 1 つの RAID パーティションが作成できるため、インストールプログラムで使用できるディスク数により、利用できる RAID デバイスのレベルが決定します。たとえば、システムにディスクが 2 つある場合は、**RAID 10** デバイスを作成することはできません。少なくともディスクが 3 つ必要になるためです。RHEL は、システムのストレージパフォーマンスと信頼性を最適化するために、インストールされたシステムにストレージを設定するための LVM および LVM シンプロビジョンングを使用したソフトウェア **RAID 0**、**RAID 1**、**RAID 4**、**RAID 5**、**RAID 6**、および **RAID 10** タイプをサポートしています。



注記

64 ビットの IBM Z では、ストレージサブシステムが RAID を透過的に使用します。ソフトウェア RAID を手動で設定する必要はありません。

前提条件

- RAID 設定オプションは、インストール用に複数のディスクを選択している場合にのみ表示される。作成する RAID タイプに応じて、少なくとも 2 つのディスクが必要です。
- マウントポイントを作成している。マウントポイントを設定して、RAID デバイスを設定します。
- インストール先 画面で カスタム ラジオボタンを選択している。

手順

- 手動パーティション設定 画面の左側のペインで、必要なパーティションを選択します。
- デバイスセクションの下にある 修正 をクリックします。マウントポイントの設定 ダイアログ ボックスが開きます。
- RAID デバイスに追加するディスクを選択して、選択 をクリックします。
- デバイスタイプ ドロップダウンメニューをクリックして、RAID を選択します。
- ファイルシステム のドロップダウンメニューをクリックして、目的のファイルシステムタイプを選択します。
- RAID レベル ドロップダウンメニューをクリックして、目的の RAID レベルを選択します。
- Update Settings をクリックして、変更を保存します。
- 完了 をクリックして設定を適用し、インストールの概要 ウィンドウに戻ります。

関連情報

- DM 整合性での RAID LV の作成
- RAID の管理

20.7. インストール済みシステムでのソフトウェア RAID の作成

mdadm ユーティリティーを使用して、既存のシステムにソフトウェア Redundant Array of Independent Disks (RAID) を作成できます。

前提条件

- mdadm** パッケージがインストールされている。
- システム上に 2 つ以上のパーティションを作成している。詳細な手順は、[parted を使用したパーティションの作成](#) を参照してください。

手順

- /dev/sda1 と /dev/sdc1 などの 2 つのブロックデバイスの RAID を作成します。

```
# mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sda1 /dev/sdc1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

level_value オプションは、RAID レベルを定義します。

2. オプション: RAID のステータスを確認します。

```
# mdadm --detail /dev/md0
/dev/md0:
  Version : 1.2
  Creation Time : Thu Oct 13 15:17:39 2022
  Raid Level : raid0
  Array Size : 18649600 (17.79 GiB 19.10 GB)
  Raid Devices : 2
  Total Devices : 2
  Persistence : Superblock is persistent

  Update Time : Thu Oct 13 15:17:39 2022
  State : clean
  Active Devices : 2
  Working Devices : 2
  Failed Devices : 0
  Spare Devices : 0
[...]
```

3. オプション: RAID 内の各デバイスに関する詳細情報を確認します。

```
# mdadm --examine /dev/sda1 /dev/sdc1
/dev/sda1:
  Magic : a92b4efc
  Version : 1.2
  Feature Map : 0x1000
  Array UUID : 77ddfb0a:41529b0e:f2c5cde1:1d72ce2c
  Name : 0
  Creation Time : Thu Oct 13 15:17:39 2022
  Raid Level : raid0
  Raid Devices : 2
[...]
```

4. RAID ドライブにファイルシステムを作成します。

```
# mkfs -t xfs /dev/md0
```

xfs を、ドライブをフォーマットするために選択したファイルシステムに置き換えます。

5. RAID ドライブのマウントポイントを作成してマウントします。

```
# mkdir /mnt/raid1
# mount /dev/md0 /mnt/raid1
```

/mnt/raid1 をマウントポイントに置き換えます。

システムの起動時に RHEL が **md0** RAID デバイスを自動的にマウントするようにするには、デバイスのエントリを **/etc/fstab** ファイルに追加します。

```
/dev/md0  /mnt/raid1 xfs defaults 0 0
```

20.8. WEB コンソールで RAID の作成

RHEL 8 Web コンソールで RAID を設定します。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。
- 物理ディスクが接続されており、システムに認識されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、メニュー ボタンをクリックし、**Create MDRAID device** を選択します。
4. **Create RAID Device** フィールドに、新しい RAID の名前を入力します。
5. **RAID レベル** ドロップダウンリストで、使用する RAID レベルを選択します。
6. **Chunk Size** ドロップダウンリストから、使用可能なオプションのリストからサイズを選択します。
Chunk Size の値は、データ書き込み用の各ブロックの大きさを指定します。たとえば、チャンクサイズが 512 KiB の場合、システムは最初の 512 KiB を最初のディスクに書き込み、次の 512 KiB を次のディスクに書き込み、その次の 512 KiB をその次のディスクに書き込みます。RAID に 3 つのディスクがある場合は、4 つ目の 512 KiB が最初のディスクに再度書き込まれます。
7. RAID に使用するディスクを選択します。
8. **Create** をクリックします。

検証

- ストレージ セクションに移動し、RAID デバイス ボックスに新しい RAID が表示されることを確認します。

20.9. WEB コンソールで RAID のフォーマット

RHEL 8 Web コンソールでソフトウェア RAID デバイスをフォーマットおよびマウントできます。

ボリュームのサイズや、選択するオプションによって、フォーマットに数分かかることがあります。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。

- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがインストールされている。
- 物理ディスクが接続されており、システムに認識されている。
- RAID が作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、フォーマットする RAID デバイスのメニュー ボタン  をクリックします。
4. ドロップダウンメニューから **Format** を選択します。
5. **Format** フィールドに名前を入力します。
6. **Mount Point** フィールドにマウントパスを追加します。
7. **Type** ドロップダウンリストから、ファイルシステムのタイプを選択します。
8. オプション: ディスクに機密データが含まれており、それを上書きする場合は、**Overwrite existing data with zeros** オプションをオンにします。オンにしない場合、ディスクヘッダーだけが書き換えられます。
9. **Encryption** ドロップダウンメニューで、暗号化の種類を選択します。ボリュームを暗号化しない場合は、**No encryption** を選択します。
10. **At boot** ドロップダウンメニューで、ボリュームをマウントするタイミングを選択します。
11. **Mount options** セクションで以下を実行します。
 - ボリュームを読み取り専用論理ボリュームとしてマウントする場合は、**Mount read only** チェックボックスをオンにします。
 - デフォルトのマウントオプションを変更する場合は、**Custom mount options** チェックボックスをオンにして、マウントオプションを追加します。
12. RAID パーティションをフォーマットします。
 - パーティションをフォーマットしてマウントする場合は、**Format and mount** ボタンをクリックします。
 - パーティションのみをフォーマットする場合は、**Format only** ボタンをクリックします。

検証

- フォーマットが正常に完了すると、**Storage** ページの **Storage** テーブルでフォーマットされた論理ボリュームの詳細を確認できます。

20.10. WEB コンソールを使用した RAID 上のパーティションテーブルの作成

RHEL 8 インターフェイスに作成した新しいソフトウェア RAID デバイスで、パーティションテーブルを有する RAID をフォーマットします。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがインストールされている。
- 物理ディスクが接続されており、システムに認識されている。
- RAID が作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. Storage テーブルで、パーティションテーブルを作成する RAID デバイスをクリックします。
4. **MDRAID device** セクションのメニュー ボタン **:** をクリックします。
5. ドロップダウンメニューから、**Create partition table** を選択します。
6. **Initialize disk** ダイアログボックスで、以下を選択します。
 - a. **パーティション設定:**
 - すべてのシステムおよびデバイスとの互換性をパーティションに持たせる必要がある場合は、**MBR** を選択します。
 - 最新のシステムとの互換性をパーティションに持たせる必要があり、2 TB を超えるハードディスクが必要な場合は、**GPT** を選択します。
 - パーティション設定が必要ない場合は、**No partitioning** を選択します。
 - b. **オーバーライト:**
 - ディスクに機密データが含まれており、それを上書きする場合は、**Overwrite existing data with zeros** オプションをオンにします。オンにしない場合、ディスクヘッダーだけが書き換えられます。
7. **Initialize** をクリックします。

20.11. WEB コンソールを使用した RAID 上のパーティションの作成

既存のパーティションテーブルにパーティションを作成します。パーティションを作成した後、さらにパーティションを作成できます。

前提条件

- RHEL 8 Web コンソールをインストールし、アクセスできる。詳細は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。
- RAID 上にパーティションテーブルが作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. パーティションを作成する RAID デバイスをクリックします。
4. RAID デバイスページで、**GPT partitions** セクションまでスクロールし、メニュー ボタン [:] をクリックします。
5. **Create partition** をクリックし、**Create partition** フィールドにファイルシステムの名前を入力します。名前にスペースは使用しないでください。
6. **Mount Point** フィールドにマウントパスを入力します。
7. **Type** ドロップダウンリストで、ファイルシステムのタイプを選択します。
8. **Size** スライダーで、パーティションのサイズを設定します。
9. オプション: ディスクに機密データが含まれており、それを上書きする場合は、**Overwrite existing data with zeros** を選択します。オンにしない場合、ディスクヘッダーだけが書き換えられます。
10. **Encryption** ドロップダウンメニューで、暗号化の種類を選択します。ボリュームを暗号化しない場合は、**No encryption** を選択します。
11. **At boot** ドロップダウンメニューで、ボリュームをマウントするタイミングを選択します。
12. **Mount options** セクションで以下を実行します。
 - ボリュームを読み取り専用論理ボリュームとしてマウントする場合は、**Mount read only** チェック ボックスをオンにします。
 - デフォルトのマウントオプションを変更する場合は、**Custom mount options** チェック ボックスをオンにして、マウントオプションを追加します。
13. パーティションを作成します。
 - パーティションを作成してマウントする場合は、**Create and mount** ボタンをクリックします。
 - パーティションのみを作成する場合は、**Create only** ボタンをクリックします。

ボリュームのサイズや、選択するオプションによって、フォーマットに数分かかることがあります。

検証

- フォーマットされた論理ボリュームの詳細は、メインストレージページの **Storage** テーブルで確認できます。

20.12. WEB コンソールを使用した RAID 上のボリュームグループの作成

ソフトウェア RAID からボリュームグループを構築

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザー アカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- cockpit-storaged** パッケージがインストールされている。
- フォーマットされておらず、マウントされていない RAID デバイスがある。

手順

- RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
- パネルで、**Storage** をクリックします。
- Storage** テーブルで、メニュー ボタン [:] をクリックし、**Create LVM2 volume group** を選択します。
- Create LVM2 volume group** フィールドに、新しいボリュームグループの名前を入力します。
- Disks** リストで、RAID デバイスを選択します。
リストに RAID が表示されない場合は、システムから RAID のマウントを解除します。RAID デバイスは、RHEL 8 システムでは使用できません。
- Create** をクリックします。

20.13. STORAGE RHEL システムロールを使用した RAID ボリュームの設定

storage システムロールを使用すると、Red Hat Ansible Automation Platform と Ansible-Core を使用して RHEL に RAID ボリュームを設定できます。要件に合わせて RAID ボリュームを設定するためのパラメーターを使用して、Ansible Playbook を作成します。



警告

特定の状況でデバイス名が変更する場合があります。たとえば、新しいディスクをシステムに追加するときなどです。したがって、データの損失を防ぐために、Playbook では永続的な命名属性を使用してください。永続的な命名属性の詳細は、[永続的な命名属性の概要](#) を参照してください。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: `~/playbook.yml`) を作成します。

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_safe_mode: false
        storage_volumes:
          - name: data
            type: raid
            disks: [sdd, sde, sdf, sdg]
            raid_level: raid0
            raid_chunk_size: 32 KiB
            mount_point: /mnt/data
            state: present
```

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.storage/README.md` ファイルを参照してください。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- アレイが正しく作成されたことを確認します。

```
# ansible managed-node-01.example.com -m command -a 'mdadm --detail /dev/md/data'
```

関連情報

- [/usr/share/ansible/roles/rhel-system-roles.storage/README.md](#) ファイル
- [/usr/share/doc/rhel-system-roles/storage/](#) ディレクトリー

20.14. RAID の拡張

mdadm ユーティリティーの **--grow** オプションを使用して RAID を拡張できます。

前提条件

- 十分なディスク領域
- parted** パッケージがインストールされている

手順

- RAID パーティションを拡張します。詳細は、[parted を使用したパーティションのサイズ変更](#) を参照してください。
- RAID をパーティション容量の最大値まで拡張します。

```
# mdadm --grow --size=max /dev/md0
```

特定のサイズを設定するには、**--size** パラメータの値を kB で記述します (例: **--size=524228**)。

- ファイルシステムのサイズを拡大します。たとえば、ボリュームが XFS を使用し、**/mnt/** にマウントされている場合は、次のように入力します。

```
# xfs_growfs /mnt/
```

関連情報

- システム上の **mdadm(8)** man ページ
- [ファイルシステムの管理](#)

20.15. RAID を縮小

mdadm ユーティリティの **--grow** オプションを使用して RAID を縮小できます。



重要

XFS ファイルシステムは縮小に対応していません。

前提条件

- **parted** パッケージがインストールされている

手順

1. ファイルシステムを縮小します。詳細は、[ファイルシステムの管理](#) を参照してください。
2. RAID のサイズを 512 MB などに減らします。

```
# mdadm --grow --size=524228 /dev/md0
```

--size パラメータを kB で記述します。

3. パーティションのサイズを、必要なサイズまで縮小します。

関連情報

- システム上の **mdadm(8)** man ページ
- [parted でパーティションのサイズ変更](#)

20.16. インストール後にルートディスクを RAID1 に変換する

Red Hat Enterprise Linux 8 をインストールした後、非 RAID ルートディスクを RAID1 ミラーに変換できます。

PowerPC (PPC) アーキテクチャーでは、以下の追加手順を行う必要があります。

前提条件

- Red Hat ナレッジベースソリューション [How do I convert my root disk to RAID1 after installation of Red Hat Enterprise Linux 7?](#) の手順を完了した。



注記

PowerPC マシンでは **grub2-install /dev/sda** コマンドを実行しても動作せず、エラーが返されますが、システムは想定どおりに起動します。

手順

1. PowerPC Reference Platform (PReP) 起動パーティションの内容を `/dev/sda1` から `/dev/sdb1` にコピーします。

```
# dd if=/dev/sda1 of=/dev/sdb1
```

2. 両方のディスクの最初のパーティションで **prep** フラグと **boot** フラグを更新します。

```
$ parted /dev/sda set 1 prep on
```

```
$ parted /dev/sda set 1 boot on
$ parted /dev/sdb set 1 prep on
$ parted /dev/sdb set 1 boot on
```

20.17. 高度な RAID デバイスの作成

場合によっては、インストールが完了する前に作成されたアレイにオペレーティングシステムをインストールすることを推奨します。通常、これは複雑な RAID デバイスに **/boot** または root ファイルシステムアレイを設定することを意味します。このような場合、Anaconda インストーラーでサポートされていないアレイオプションを使用する必要がある場合があります。これを回避するには、以下の手順を行います。



注記

インストーラーの制限されたレスキュー モードには man ページは含まれません。 **mdadm** と **md** の両方の man ページには、カスタム RAID アレイを作成するための有用な情報が含まれており、回避策全体で必要になる場合があります。

手順

1. インストールディスクを挿入します。
2. 初回起動時に、**Install** または **Upgrade** ではなく、**Rescue Mode** を選択します。システムが **Rescue mode** で完全に起動すると、コマンドラインインターミナルが表示されます。
3. このターミナルから、次のコマンドを実行します。
 - a. **parted** コマンドを使用して、ターゲットハードドライブに RAID パーティションを作成します。
 - b. 使用可能なすべての設定とオプションを使用して、これらのパーティションから **mdadm** コマンドを使用して手動で RAID アレイを作成します。
4. オプション: アレイを作成したら、アレイ上にもファイルシステムを作成します。
5. コンピューターを再起動して、**インストール** または **更新** を選択して通常通りにインストールします。Anaconda インストーラーはシステム内のディスクを検索するため、既存の RAID デバイスが見つかります。
6. システムのディスクの使い方を求められたら、**カスタムレイアウト** を選択して **次へ** をクリックします。デバイス一覧に、既存の MD RAID デバイスが表示されます。
7. RAID デバイスを選択し、**Edit** をクリックします。
8. マウントポイントを設定し、必要に応じて、以前に作成していない場合は使用するファイルシステムのタイプを設定し、**Done** をクリックします。Anaconda は、この既存の RAID デバイスにインストールし、Rescue モードで作成したときに選択したカスタムオプションを保持します。

20.18. RAID を監視するための電子メール通知の設定

mdadm ツールを使用して RAID を監視するように電子メールアラートを設定できます。 **MAILADDR** 変数が必要な電子メールアドレスに設定されると、監視システムは追加された電子メールアドレスにアラートを送信します。

前提条件

- **mdadm** パッケージがインストールされている。
- メールサービスが設定されている。

手順

1. RAID の詳細をスキャンして、アレイを監視するための **/etc/mdadm.conf** 設定ファイルを作成します。

```
# mdadm --detail --scan >> /etc/mdadm.conf
```

ARRAY および **MAILADDR** は必須の変数であることに注意してください。

2. 任意のテキストエディターで **/etc/mdadm.conf** 設定ファイルを開き、 **MAILADDR** 変数に通知用のメールアドレスを追加します。たとえば、次の行を追加します。

```
MAILADDR example@example.com
```

example@example.com は、アレイの監視からアラートを受信するためのメールアドレスです。

3. **/etc/mdadm.conf** ファイルに変更を保存して、閉じます。

関連情報

- システム上の **mdadm.conf(5)** man ページ

20.19. RAID での障害のあるディスクの置き換え

残りのディスクを使用して、故障したディスクからデータを再構築できます。データを正常に再構築するためには最低限必要な残りのディスクの量は、RAID レベルとディスクの総数によって決まります。

この手順では、**/dev/md0** RAID に 4 つのディスクが含まれています。 **/dev/sdd** ディスクに障害が発生したため、**/dev/sdf** ディスクと交換する必要があります。

前提条件

- 交換用スペアディスク。
- **mdadm** パッケージがインストールされている。

手順

1. 障害が発生したディスクを確認します。

- a. カーネルログを表示します。

```
# journalctl -k -f
```

- b. 次のようなメッセージを検索します。

```
md/raid:md0: Disk failure on sdd, disabling device.
md/raid:md0: Operation continuing on 3 devices.
```

- c. キーボードの **Ctrl+C** を押して、**journalctl** プログラムを終了します。

2. 障害の発生したディスクに **faulty** のマークを付けます。

```
# mdadm --manage /dev/md0 --fail /dev/sdd
```

3. オプション: 障害が発生したディスクが正しくマークされているかどうかを確認します。

```
# mdadm --detail /dev/md0
```

出力の最後には、ディスク **/dev/sdd** のステータスが **faulty** の **/dev/md0** RAID 内にあるディスクのリストが表示されます。

Number	Major	Minor	RaidDevice	State
0	8	16	0	active sync /dev/sdb
1	8	32	1	active sync /dev/sdc
-	0	0	2	removed
3	8	64	3	active sync /dev/sde
2	8	48	-	faulty /dev/sdd

4. 障害が発生したディスクを RAID から取り外します。

```
# mdadm --manage /dev/md0 --remove /dev/sdd
```



警告

RAID が別のディスク障害に耐えられない場合は、新しいディスクのステータスが **active sync** になるまでディスクを取り外さないでください。 **watch cat /proc/mdstat** コマンドを使用すると、進捗を監視できます。

5. 新しいディスクを RAID に追加します。

```
# mdadm --manage /dev/md0 --add /dev/sdf
```

/dev/md0 RAID には新しいディスク **/dev/sdf** が含まれるようになり、**mdadm** サービスは他のディスクからデータのコピーを自動的に開始します。

検証

- アレイの詳細を確認します。

```
# mdadm --detail /dev/md0
```

このコマンドの出力の最後に表示される `/dev/md0` RAID 内のディスクのリストで、新しいディスクのステータスが `spare rebuilding` である場合、データはまだ他のディスクからコピーされています。

Number	Major	Minor	RaidDevice	State
0	8	16	0	active sync /dev/sdb
1	8	32	1	active sync /dev/sdc
4	8	80	2	spare rebuilding /dev/sdf
3	8	64	3	active sync /dev/sde

データのコピーが完了すると、新しいディスクは `active sync` 状態になります。

関連情報

- RAID を監視するための電子メール通知の設定

20.20. RAID ディスクの修復

`repair` オプションを使用して、RAID アレイ内のディスクを修復できます。

前提条件

- `mdadm` パッケージがインストールされている。

手順

- 障害が発生したディスクの動作についてアレイを確認します。

```
# echo check > /sys/block/	md0/md/sync_action
```

これによりアレイがチェックされ、`/sys/block/md0/md/sync_action` ファイルに同期アクションが表示されます。

- `/sys/block/md0/md/sync_action` ファイルを任意のテキストエディターで開き、ディスク同期の失敗に関するメッセージがあるかどうかを確認します。
- `/sys/block/md0/md/mismatch_cnt` ファイルを表示します。`mismatch_cnt` パラメーターが `0` でない場合は、RAID ディスクを修復する必要があることを意味します。
- アレイ内のディスクを修復します。

```
# echo repair > /sys/block/	md0/md/sync_action
```

これにより、アレイ内のディスクが修復され、結果が `/sys/block/md0/md/sync_action` ファイルに書き込まれます。

- 同期の進行状況を表示します。

```
# cat /sys/block/md0/md/sync_action
repair
```

```
# cat /proc/mdstat
Personalities : [raid0] [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdg[1] dm-3[0]
      511040 blocks super 1.2 [2/2] [UU]
unused devices: <none>
```

第21章 LUKS を使用したブロックデバイスの暗号化

ディスク暗号化を使用すると、ブロックデバイス上のデータを暗号化して保護できます。デバイスの復号化されたコンテンツにアクセスするには、認証としてパスフレーズまたは鍵を入力します。これは、デバイスがシステムから物理的に取り外された場合でも、デバイスのコンテンツを保護するのに役立ちます。モバイルコンピューターやリムーバブルメディアにとって重要です。LUKS 形式は、Red Hat Enterprise Linux におけるブロックデバイスの暗号化のデフォルト実装です。

21.1. LUKS ディスクの暗号化

Linux Unified Key Setup-on-disk-format (LUKS) は、暗号化されたデバイスの管理を簡素化するツールセットを提供します。LUKS を使用すると、ブロックデバイスを暗号化し、複数のユーザーキーでマスターキーを復号化できるようになります。パーティションの一括暗号化には、このマスターキーを使用します。

Red Hat Enterprise Linux は、LUKS を使用してブロックデバイスの暗号化を実行します。デフォルトではインストール時に、ブロックデバイスを暗号化するオプションが指定されていません。ディスクを暗号化するオプションを選択すると、コンピューターを起動するたびにパスフレーズの入力が求められます。このパスフレーズは、パーティションを復号化するバルク暗号鍵のロックを解除します。デフォルトのパーティションテーブルを変更する場合は、暗号化するパーティションを選択できます。この設定は、パーティションテーブル設定で行われます。

Ciphers

LUKS に使用されるデフォルトの暗号は **aes-xts-plain64** です。LUKS のデフォルトの鍵サイズは 512 ビットです。Anaconda XTS モードを使用した LUKS のデフォルトの鍵サイズは 512 ビットです。使用可能な暗号は次のとおりです。

- 高度暗号化標準 (Advanced Encryption Standard, AES)
- Twofish
- Serpent

LUKS によって実行される操作

- LUKS は、ブロックデバイス全体を暗号化するため、脱着可能なストレージメディアやラップトップのディスクドライブといった、モバイルデバイスのコンテンツを保護するのに適しています。
- 暗号化されたブロックデバイスの基本的な内容は任意であり、スワップデバイスの暗号化に役立ちます。また、とりわけデータストレージ用にフォーマットしたブロックデバイスを使用する特定のデータベースに関しても有用です。
- LUKS は、既存のデバイスマッパーのカーネルサブシステムを使用します。
- LUKS はパスフレーズのセキュリティを強化し、辞書攻撃から保護します。
- LUKS デバイスには複数のキースロットが含まれているため、バックアップキーやパスフレーズを追加できます。



重要

LUKS は次のシナリオには推奨されません。

- LUKS などのディスク暗号化ソリューションは、システムの停止時にしかデータを保護しません。システムの電源がオンになり、LUKS がディスクを復号化すると、そのディスクのファイルは、そのファイルにアクセスできるすべてのユーザーが使用できます。
- 同じデバイスに対する個別のアクセスキーを複数のユーザーが持つ必要があるシナリオ。LUKS1 形式はキースロットを 8 個提供し、LUKS2 形式はキースロットを最大 32 個提供します。
- ファイルレベルの暗号化を必要とするアプリケーション。

関連情報

- [LUKS プロジェクトのホームページ](#)
- [LUKS オンディスクフォーマットの仕様](#)
- [FIPS 197: Advanced Encryption Standard \(AES\)](#)

21.2. RHEL の LUKS バージョン

Red Hat Enterprise Linux では、LUKS 暗号化のデフォルト形式は LUKS2 です。古い LUKS1 形式は引き続き完全にサポートされており、以前の Red Hat Enterprise Linux リリースと互換性のある形式で提供されます。LUKS2 再暗号化は、LUKS1 再暗号化と比較して、より堅牢で安全に使用できる形式と考えられています。

LUKS2 形式を使用すると、バイナリー構造を変更することなく、さまざまな部分を後に更新できます。LUKS2 は、内部的にメタデータに JSON テキスト形式を使用し、メタデータの冗長性を提供し、メタデータの破損を検出し、メタデータのコピーから自動的に修復します。



重要

LUKS2 と LUKS1 はディスクの暗号化に異なるコマンドを使用するため、LUKS1 のみをサポートするシステムでは LUKS2 を使用しないでください。LUKS バージョンに誤ったコマンドを使用すると、データが失われる可能性があります。

表21.1 LUKS バージョンに応じた暗号化コマンド

LUKS バージョン	暗号化コマンド
LUKS2	cryptsetup reencrypt
LUKS1	cryptsetup-reencrypt

オンラインの再暗号化

LUKS2 形式は、デバイスが使用中の間に、暗号化したデバイスの再暗号化に対応します。たとえば、以下のタスクを実行するにあたり、デバイスでファイルシステムをアンマウントする必要はありません。

- ボリュームキーの変更
- 暗号化アルゴリズムの変更
暗号化されていないデバイスを暗号化する場合は、ファイルシステムのマウントを解除する必要があります。暗号化の短い初期化後にファイルシステムを再マウントできます。

LUKS1 形式は、オンライン再暗号化に対応していません。

変換

特定の状況では、LUKS1 を LUKS2 に変換できます。具体的には、以下のシナリオでは変換ができません。

- LUKS1 デバイスが、Policy-Based Decryption (PBD) Clevis ソリューションにより使用されているとマークされている。**cryptsetup** ツールは、**luksmeta** メタデータが検出されると、そのデバイスを変換することを拒否します。
- デバイスがアクティブになっている。デバイスが非アクティブ状態でなければ、変換することはできません。

21.3. LUKS2 再暗号化中のデータ保護のオプション

LUKS2 では、再暗号化プロセスで、パフォーマンスやデータ保護の優先度を設定する複数のオプションを選択できます。**resilience** オプションには次のモードが用意されています。**cryptsetup reencrypt --resilience resilience-mode /dev/<device_ID>** コマンドを使用すると、これらのモードのいずれかを選択できます。<device_ID> は、デバイスの ID に置き換えてください。

checksum

デフォルトのモード。データ保護とパフォーマンスのバランスを取ります。

このモードでは、再暗号化領域内のセクターのチェックサムが個別に保存されます。チェックサムは、LUKS2 によって再暗号化されたセクターについて、復旧プロセスで検出できます。このモードでは、ブロックデバイスセクターの書き込みがアトミックである必要があります。

journal

最も安全なモードですが、最も遅いモードでもあります。このモードでは、再暗号化領域をバイナリ領域にジャーナル化するため、LUKS2 はデータを 2 回書き込みます。

none

none モードではパフォーマンスが優先され、データ保護は提供されません。**SIGTERM** シグナルやユーザーによる **Ctrl+C** キーの押下など、安全なプロセス終了からのみデータを保護します。予期しないシステム障害やアプリケーション障害が発生すると、データが破損する可能性があります。

LUKS2 の再暗号化プロセスが強制的に突然終了した場合、LUKS2 は以下のいずれかの方法で復旧を実行できます。

自動

次のいずれかのアクションを実行すると、次回の LUKS2 デバイスを開くアクション中に自動復旧アクションがトリガーされます。

- cryptsetup open** コマンドを実行する。
- systemd-cryptsetup** コマンドを使用してデバイスを接続する。

手動

LUKS2 デバイスで **cryptsetup repair /dev/<device_ID>** コマンドを使用します。

関連情報

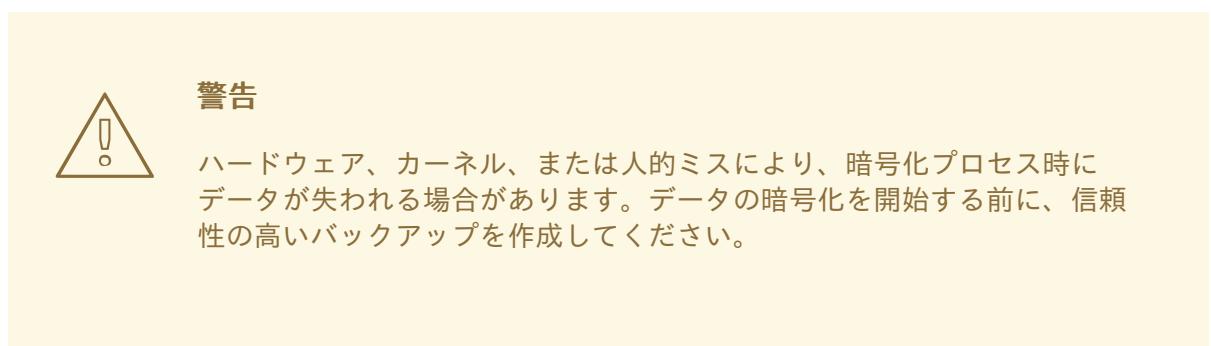
- システム上の **cryptsetup-reencrypt(8)** および **cryptsetup-repair(8)** man ページ

21.4. LUKS2 を使用したブロックデバイスの既存データの暗号化

LUKS2 形式を使用して、まだ暗号化されていないデバイスの既存のデータを暗号化できます。新しい LUKS ヘッダーは、デバイスのヘッドに保存されます。

前提条件

- ブロックデバイスにファイルシステムがある。
- データのバックアップを作成している。



手順

- 暗号化するデバイスにあるファイルシステムのマウントをすべて解除します。次に例を示します。

```
# umount /dev/mapper/vg00-lv00
```

- LUKS ヘッダーを保存するための空き容量を確認します。シナリオに合わせて、次のいずれかのオプションを使用します。

- 論理ボリュームを暗号化する場合は、以下のように、ファイルシステムのサイズを変更せずに、論理ボリュームを拡張できます。以下に例を示します。

```
# lvextend -L+32M /dev/mapper/vg00-lv00
```

- parted** などのパーティション管理ツールを使用してパーティションを拡張します。
- このデバイスのファイルシステムを縮小します。ext2、ext3、またはext4のファイルシステムには **resize2fs** ユーティリティーを使用できます。XFS ファイルシステムは縮小できないことに注意してください。

- 暗号化を初期化します。

```
# cryptsetup reencrypt --init-only --reduce-device-size 32M /dev/mapper/vg00-lv00_lv00_encrypted
```

```
■ /dev/mapper/lv00_encrypted is now active and ready for online encryption.
```

4. デバイスをマウントします。

```
■ # mount /dev/mapper/lv00_encrypted /mnt/lv00_encrypted
```

5. 永続的なマッピングのエントリーを **/etc/crypttab** ファイルに追加します。

a. **luksUUID** を見つけます。

```
■ # cryptsetup luksUUID /dev/mapper/vg00-lv00
```

```
a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
```

b. 任意のテキストエディターで **/etc/crypttab** を開き、このファイルにデバイスを追加します。

```
■ $ vi /etc/crypttab
```

```
lv00_encrypted UUID=a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325 none
```

`a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325` は、デバイスの **luksUUID** に置き換えます。

c. **dracut** で initramfs を更新します。

```
■ $ dracut -f --regenerate-all
```

6. **/etc/fstab** ファイルに永続的なマウントのエントリーを追加します。

a. アクティブな LUKS ブロックデバイスのファイルシステムの UUID を見つけます。

```
■ $ blkid -p /dev/mapper/lv00_encrypted
```

```
/dev/mapper/lv00-encrypted: UUID="37bc2492-d8fa-4969-9d9b-bb64d3685aa9"
BLOCK_SIZE="4096" TYPE="xfs" USAGE="filesystem"
```

b. 任意のテキストエディターで **/etc/fstab** を開き、このファイルにデバイスを追加します。次に例を示します。

```
■ $ vi /etc/fstab
```

```
UUID=37bc2492-d8fa-4969-9d9b-bb64d3685aa9 /home auto rw,user,auto 0
```

`37bc2492-d8fa-4969-9d9b-bb64d3685aa9` は、ファイルシステムの UUID に置き換えます。

7. オンライン暗号化を再開します。

```
■ # cryptsetup reencrypt --resume-only /dev/mapper/vg00-lv00
```

```
Enter passphrase for /dev/mapper/vg00-lv00:
```

```
Auto-detected active dm device 'lv00_encrypted' for data device /dev/mapper/vg00-lv00.
```

```
Finished, time 00:31.130, 10272 MiB written, speed 330.0 MiB/s
```

検証

- 既存のデータが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump /dev/mapper/vg00-lv00

LUKS header information
Version: 2
Epoch: 4
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID: a52e2cc9-a5be-47b8-a95d-6bdf4f2d9325
Label: (no label)
Subsystem: (no subsystem)
Flags: (no flags)

Data segments:
 0: crypt
  offset: 33554432 [bytes]
  length: (whole device)
  cipher: aes-xts-plain64
  [...]
```

- 暗号化された空のブロックデバイスのステータスを表示します。

```
# cryptsetup status lv00_encrypted

/dev/mapper/lv00_encrypted is active and is in use.
  type:  LUKS2
  cipher: aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device: /dev/mapper/vg00-lv00
```

関連情報

- システム上の **cryptsetup(8)**、**cryptsetup-reencrypt(8)**、**lvextend(8)**、**resize2fs(8)**、および **parted(8)** man ページ

21.5. 独立したヘッダーがある LUKS2 を使用してブロックデバイスの既存データの暗号化

LUKS ヘッダーを保存するための空き領域を作成せずに、ブロックデバイスの既存のデータを暗号化できます。ヘッダーは、追加のセキュリティ層としても使用できる、独立した場所に保存されます。この手順では、LUKS2 暗号化形式を使用します。

前提条件

- ブロックデバイスにファイルシステムがある。
- データがバックアップ済みである。



警告

ハードウェア、カーネル、または人的ミスにより、暗号化プロセス時にデータが失われる場合があります。データの暗号化を開始する前に、信頼性の高いバックアップを作成してください。

手順

1. 以下のように、そのデバイスのファイルシステムをすべてアンマウントします。

```
# umount /dev/<nvme0n1p1>
```

<nvme0n1p1> は、アンマウントするパーティションに対応するデバイス識別子に置き換えます。

2. 暗号化を初期化します。

```
# cryptsetup reencrypt --init-only --header </home/header> /dev/<nvme0n1p1>
<nvme_encrypted>
```

WARNING!

=====

Header file does not exist, do you want to create it?

Are you sure? (Type 'yes' in capital letters): YES

Enter passphrase for </home/header>:

Verify passphrase:

/dev/mapper/<nvme_encrypted> is now active and ready for online encryption.

以下のように置き換えます。

- </home/header> には、独立した LUKS ヘッダーを含むファイルへのパスを指定します。後で暗号化したデバイスのロックを解除するために、独立した LUKS ヘッダーにアクセスできる必要があります。
- <nvme_encrypted> は、暗号化後に作成されるデバイスマッパーの名前に置き換えます。

3. デバイスをマウントします。

```
# mount /dev/mapper/<nvme_encrypted> /mnt/<nvme_encrypted>
```

4. 永続的なマッピングのエントリーを /etc/crypttab ファイルに追加します。

```
# <nvme_encrypted> /dev/disk/by-id/<nvme-partition-id> none header=</home/header>
```

<nvme-partition-id> は、NVMe パーティションの識別子に置き換えます。

5. dracut を使用して initramfs を再生成します。

```
# dracut -f --regenerate-all -v
```

6. **/etc/fstab** ファイルに永続的なマウントのエントリーを追加します。

a. アクティブな LUKS ブロックデバイスのファイルシステムの UUID を見つけます。

```
$ blkid -p /dev/mapper/<nvme_encrypted>
```

```
/dev/mapper/<nvme_encrypted>: UUID="37bc2492-d8fa-4969-9d9b-bb64d3685aa9"
BLOCK_SIZE="4096" TYPE="xfs" USAGE="filesystem"
```

b. テキストエディターで **/etc/fstab** を開き、このファイルにデバイスを追加します。次に例を示します。

```
UUID=<file_system_UUID> /home auto rw,user,auto 0
```

<file_system_UUID> は、前の手順で見つかったファイルシステムの UUID に置き換えます。

7. オンライン暗号化を再開します。

```
# cryptsetup reencrypt --resume-only --header </home/header> /dev/<nvme0n1p1>
```

Enter passphrase for /dev/<nvme0n1p1>:

```
Auto-detected active dm device '<nvme_encrypted>' for data device /dev/<nvme0n1p1>.
Finished, time 00m51s, 10 GiB written, speed 198.2 MiB/s
```

検証

1. 独立したヘッダーがある LUKS2 を使用するブロックデバイスの既存のデータが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump </home/header>
```

LUKS header information

Version: 2

Epoch: 88

Metadata area: 16384 [bytes]

Keyslots area: 16744448 [bytes]

UUID: c4f5d274-f4c0-41e3-ac36-22a917ab0386

Label: (no label)

Subsystem: (no subsystem)

Flags: (no flags)

Data segments:

0: crypt

offset: 0 [bytes]

length: (whole device)

cipher: aes-xts-plain64

sector: 512 [bytes]

[...]

2. 暗号化された空のブロックデバイスのステータスを表示します。

```
# cryptsetup status <nvme_encrypted>
```

```
/dev/mapper/<nvme_encrypted> is active and is in use.
  type:  LUKS2
  cipher: aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device: /dev/<nvme0n1p1>
```

関連情報

- システム上の **cryptsetup(8)** および **cryptsetup-reenrypt(8)** man ページ

21.6. LUKS2 を使用した空のブロックデバイスの暗号化

LUKS2 形式を使用して、空のブロックデバイスを暗号化して、暗号化ストレージとして使用できます。

前提条件

- 空のブロックデバイス。**lsblk** などのコマンドを使用して、そのデバイス上に実際のデータ(ファイルシステムなど)がないかどうかを確認できます。

手順

- 暗号化した LUKS パーティションとしてパーティションを設定します。

```
# cryptsetup luksFormat /dev/nvme0n1p1

WARNING!
=====
This will overwrite data on /dev/nvme0n1p1 irreversibly.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/nvme0n1p1:
Verify passphrase:
```

- 暗号化した LUKS パーティションを開きます。

```
# cryptsetup open /dev/nvme0n1p1 nvme0n1p1_encrypted

Enter passphrase for /dev/nvme0n1p1:
```

これにより、パーティションのロックが解除され、デバイスマッパーを使用してパーティションが新しいデバイスにマッピングされます。暗号化されたデータを上書きしないように、このコマンドは、デバイスが暗号化されたデバイスであり、**/dev/mapper/device_mapped_name** パスを使用して LUKS を通じてアドレス指定されることをカーネルに警告します。

- 暗号化されたデータをパーティションに書き込むためのファイルシステムを作成します。このパーティションには、デバイスマップ名を介してアクセスする必要があります。

```
# mkfs -t ext4 /dev/mapper/nvme0n1p1_encrypted
```

- デバイスをマウントします。

```
# mount /dev/mapper/nvme0n1p1_encrypted mount-point
```

検証

- 空のブロックデバイスが暗号化されているかどうかを確認します。

```
# cryptsetup luksDump /dev/nvme0n1p1

LUKS header information
Version:      2
Epoch:        3
Metadata area: 16384 [bytes]
Keyslots area: 16744448 [bytes]
UUID:         34ce4870-ffdf-467c-9a9e-345a53ed8a25
Label:        (no label)
Subsystem:    (no subsystem)
Flags:        (no flags)

Data segments:
 0: crypt
  offset: 16777216 [bytes]
  length: (whole device)
  cipher: aes-xts-plain64
  sector: 512 [bytes]
  [...]
```

- 暗号化された空のブロックデバイスのステータスを表示します。

```
# cryptsetup status nvme0n1p1_encrypted

/dev/mapper/nvme0n1p1_encrypted is active and is in use.
  type:  LUKS2
  cipher: aes-xts-plain64
  keysize: 512 bits
  key location: keyring
  device: /dev/nvme0n1p1
  sector size: 512
  offset: 32768 sectors
  size:  20938752 sectors
  mode:  read/write
```

関連情報

- システム上の **cryptsetup(8)**、**cryptsetup-open(8)**、および **cryptsetup-luksFormat(8)** man ページ

21.7. WEB コンソールでの LUKS パスフレーズの設定

システムの既存の論理ボリュームに暗号化を追加する場合は、ボリュームをフォーマットすることですか実行できません。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。

- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。
- 暗号化なしで、既存の論理ボリュームを利用できます。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、暗号化するストレージデバイスのメニューボタン **:** をクリックし、**Format** をクリックします。
4. **Encryption field** で、暗号化仕様 **LUKS1** または **LUKS2** を選択します。
5. 新しいパスフレーズを設定し、確認します。
6. オプション: さらなる暗号化オプションを変更します。
7. フォーマット設定の最終処理
8. **Format** をクリックします。

21.8. WEB コンソールで LUKS パスフレーズの変更

Web コンソールで、暗号化されたディスクまたはパーティションで LUKS パスフレーズを変更します。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **cockpit-storaged** パッケージがシステムにインストールされている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. パネルで、**Storage** をクリックします。
3. **Storage** テーブルで、暗号化されたデータを含むディスクを選択します。
4. ディスクページで、**Keys** セクションまでスクロールし、**編集** ボタンをクリックします。
5. **パスフレーズの変更** ダイアログウィンドウで、以下を行います。

- a. 現在のパスフレーズを入力します。
 - b. 新しいパスフレーズを入力します。
 - c. 新しいパスフレーズを確認します。
6. **Save** をクリックします。

21.9. コマンドラインを使用した LUKS パスフレーズの変更

コマンドラインを使用して、暗号化されたディスクまたはパーティションの LUKS パスフレーズを変更します。**cryptsetup** ユーティリティーを使用すると、さまざまな設定オプションと機能を使用して暗号化プロセスを制御し、既存の自動化ワークフローにプロセスを統合できます。

前提条件

- **root** 特権、または **sudo** を使用して管理コマンドを入力する権限がある。

手順

1. LUKS 暗号化デバイスの既存のパスフレーズを変更します。

```
# cryptsetup luksChangeKey /dev/<device_ID>
```

<device_ID> は、デバイス指定子 (例: **sda**) に置き換えます。

複数のキースロットが設定されている場合は、使用するスロットを指定できます。

```
# cryptsetup luksChangeKey /dev/<device_ID> --key-slot <slot_number>
```

<slot_number> は、変更するキースロットの番号に置き換えます。

2. 現在のパスフレーズと新しいパスフレーズを入力します。

```
Enter passphrase to be changed:  
Enter new passphrase:  
Verify passphrase:
```

3. 新しいパスフレーズを検証します。

```
# cryptsetup --verbose open --test-passphrase /dev/<device_ID>
```

検証

1. 新しいパスフレーズでデバイスのロックを解除できることを確認します。

```
Enter passphrase for /dev/<device_ID>:  
Key slot <slot_number> unlocked.  
Command successful.
```

21.10. STORAGE RHEL システムロールを使用して LUKS2 暗号化ボリュームを作成する

storage ロールを使用し、Ansible Playbook を実行して、LUKS で暗号化されたボリュームを作成および設定できます。

前提条件

- コントロールノードと管理対象ノードの準備が完了している。
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 機密性の高い変数を暗号化されたファイルに保存します。

- a. vault を作成します。

```
$ ansible-vault create ~/vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- b. **ansible-vault create** コマンドでエディターが開いたら、機密データを **<key>: <value>** 形式で入力します。

```
luks_password: <password>
```

- c. 変更を保存して、エディターを閉じます。Ansible は vault 内のデータを暗号化します。

2. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- name: Manage local storage
  hosts: managed-node-01.example.com
  vars_files:
    - ~/vault.yml
  tasks:
    - name: Create and configure a volume encrypted with LUKS
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.storage
      vars:
        storage_volumes:
          - name: barefs
            type: disk
            disks:
              - sdb
            fs_type: xfs
            fs_label: <label>
            mount_point: /mnt/data
            encryption: true
            encryption_password: "{{ luks_password }}"
```

Playbook で使用されるすべての変数の詳細は、コントロールノードの `/usr/share/ansible/roles/rhel-system-roles.storage/README.md` ファイルを参照してください。

- Playbook の構文を検証します。

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

- Playbook を実行します。

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

検証

- LUKS 暗号化ボリュームの `luksUUID` 値を見つけます。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksUUID /dev/sdb'  
4e4e7970-1822-470e-b55a-e91efe5d0f5c
```

- ボリュームの暗号化ステータスを表示します。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup status luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c'  
  
/dev/mapper/luks-4e4e7970-1822-470e-b55a-e91efe5d0f5c is active and is in use.  
  type:  LUKS2  
  cipher: aes-xts-plain64  
  keysize: 512 bits  
  key location: keyring  
  device: /dev/sdb  
  ...
```

- 作成された LUKS 暗号化ボリュームを確認します。

```
# ansible managed-node-01.example.com -m command -a 'cryptsetup luksDump /dev/sdb'  
  
LUKS header information  
Version: 2  
Epoch: 3  
Metadata area: 16384 [bytes]  
Keyslots area: 16744448 [bytes]  
UUID: 4e4e7970-1822-470e-b55a-e91efe5d0f5c  
Label: (no label)  
Subsystem: (no subsystem)  
Flags: (no flags)  
  
Data segments:  
0: crypt
```

```
offset: 16777216 [bytes]
length: (whole device)
cipher: aes-xts-plain64
sector: 512 [bytes]
```

...

関連情報

- [/usr/share/ansible/roles/rhel-system-roles.storage/README.md](#) ファイル
- [/usr/share/doc/rhel-system-roles/storage/](#) ディレクトリー
- [LUKS を使用したブロックデバイスの暗号化](#)
- [Ansible vault](#)

第22章 テープデバイスの管理

テープデバイスは、データの保存先で順次アクセスされる磁気テープです。データは、テープドライブを使用してこのテープデバイスに書き込まれます。テープデバイスにデータを保存するためにファイルシステムを作成する必要はありません。テープドライブは、SCSI、FC、USB、SATAなどのさまざまなインターフェイスを備えたホストコンピューターに接続できます。

22.1. テープデバイスの種類

以下は、さまざまなタイプのテープデバイスのリストです。

- **/dev/st0** は、巻き戻しありのテープデバイスです。
- **/dev/nst0** は、巻き戻しなしのテープデバイスです。日次バックアップには、巻き戻しなしのデバイスを使用します。

テープデバイスを使用するメリットは複数あります。コスト効率が高く、安定しています。テープデバイスは、データの破損に対しても回復力があり、データの保持に適しています。

22.2. テープドライブ管理ツールのインストール

テープドライブ操作用の **mt-st** パッケージをインストールします。磁気テープドライブの操作を制御するには **mt** ユーティリティーを使用します。SCSI テープドライバーを制御するには **st** ユーティリティーを使用します。

手順

- **mt-st** パッケージをインストールします。

```
# yum install mt-st
```

関連情報

- システム上の **mt(1)** および **st(4)** man ページ

22.3. テープコマンド

以下は、一般的な **mt** コマンドです。

表22.1 mt コマンド

コマンド	説明
mt -f /dev/st0 status	テープデバイスの状態を表示します。
mt -f /dev/st0 erase	テープ全体を消去します。
mt -f /dev/nst0 rewind	テープデバイスを巻き戻します。

コマンド	説明
<code>mt -f /dev/nst0 fsf n</code>	テープヘッドを正引きレコードに切り替えます。ここでは、 <code>n</code> はオプションのファイル数です。ファイル数を指定すると、テープヘッドは <code>n</code> 件のレコードをスキップします。
<code>mt -f /dev/nst0 bsmf n</code>	テープヘッドを以前のレコードに切り替えます。
<code>mt -f /dev/nst0 eod</code>	テープヘッドをデータの最後に切り替えます。

22.4. 巻き戻しテープデバイスへの書き込み

巻き戻しテープデバイスは、操作のたびにテープを巻き戻します。データをバックアップするには、`tar` コマンドを使用できます。デフォルトでは、テープデバイスの ブロックサイズ は 10KB (`bs=10k`) です。`export TAPE=/dev/st0` 属性を使用して `TAPE` 環境変数を設定できます。代わりに `-f` デバイスオプションを使用してテープデバイスファイルを指定します。このオプションは、複数のテープデバイスを使用する場合に役立ちます。

前提条件

1. `mt-st` パッケージがインストールされている。詳細は、[テープドライブ管理ツールのインストール](#) を参照。
2. テープドライブが読み込まれている。

```
# mt -f /dev/st0 load
```

手順

1. テープヘッドを確認します。

```
# mt -f /dev/st0 status

SCSI 2 tape drive:
File number=-1, block number=-1, partition=0.
Tape block size 0 bytes. Density code 0x0 (default).
Soft error count since last status=0
General status bits on (50000):
DR_OPEN IM_REP_EN
```

ここでは、以下のようになります。

- 現在の **ファイル番号** は `-1` です。
- **block number** はテープヘッドを定義します。デフォルトでは、`-1` に設定されます。
- **block size 0** は、テープデバイスのブロックサイズが固定されていないことを示します。
- **Soft error count** は、`mt status` コマンドの実行後に発生したエラーの数を示します。
- **General status bits** は、テープデバイスの統計を表示します。

- **DR_OPEN** は、ドアが開き、テープデバイスが空であることを示します。 **IM REP EN** は即時レポートモードです。

2. テープデバイスが空でない場合は、それを上書きします。

```
# tar -czf /dev/st0 _/source/directory
```

このコマンドは、テープデバイスのデータを **/source/directory** の内容で上書きします。

3. **/source/directory** をテープデバイスにバックアップします。

```
# tar -czf /dev/st0 _/source/directory
tar: Removing leading '/' from member names
/source/directory
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

4. テープデバイスのステータスを表示します。

```
# mt -f /dev/st0 status
```

検証

- テープデバイスにあるすべてのファイルのリストを表示します。

```
# tar -tf /dev/st0
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

関連情報

- システム上の **mt(1)**、**st(4)**、および **tar(1)** man ページ
- [Tape drive media detected as write protected](#) (Red Hat ナレッジベース)
- [How to check if tape drives are detected in the system](#) (Red Hat ナレッジベース)

22.5. 巻き戻しなしのテープデバイスへの書き込み

特定のコマンドの実行を完了した後、巻き戻しなしのテープデバイスはテープをその状態のままにします。たとえば、巻き戻しなしのテープデバイスでは、バックアップの後にさらにデータを追加できます。また、これを使用して予期しない巻き戻しを回避することもできます。

前提条件

1. **mt-st** パッケージがインストールされている。 詳細は、[テープドライブ管理ツールのインストール](#) を参照。

2. テープドライブが読み込まれている。

```
# mt -f /dev/nst0 load
```

手順

1. 巻き戻しなしのテープデバイス /dev/nst0 のテープヘッドを確認します。

```
# mt -f /dev/nst0 status
```

2. テープのヘッドまたはテープの最後にポインターを指定します。

```
# mt -f /dev/nst0 rewind
```

3. テープデバイスにデータを追加するには、次のコマンドを実行します。

```
# mt -f /dev/nst0 eod
# tar -czf /dev/nst0 /source/directory/
```

4. /source/directory/ をテープデバイスにバックアップします。

```
# tar -czf /dev/nst0 /source/directory/
tar: Removing leading `/' from member names
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

5. テープデバイスのステータスを表示します。

```
# mt -f /dev/nst0 status
```

検証

- テープデバイスにあるすべてのファイルのリストを表示します。

```
# tar -tzf /dev/nst0
/source/directory/
/source/directory/man_db.conf
/source/directory/DIR_COLORS
/source/directory/rsyslog.conf
[...]
```

関連情報

- システム上の **mt(1)**、**st(4)**、および **tar(1)** man ページ
- [Tape drive media detected as write protected](#) (Red Hat ナレッジベース)
- [How to check if tape drives are detected in the system](#) (Red Hat ナレッジベース)

22.6. テープデバイスでのテープヘッドの切り替え

eod オプションを使用して、テープデバイス内のテープヘッドを切り替えることができます。

前提条件

1. **mt-st** パッケージがインストールされている。詳細は、[テープドライブ管理ツールのインストール](#) を参照。
2. データはテープデバイスに書き込まれる。詳細は、[Writing to rewinding tape devices](#) または [Writing to non-rewinding tape devices](#) を参照。

手順

- テープポインターの現在の位置を表示するには、次のコマンドを実行します。


```
# mt -f /dev/nst0 tell
```
- データをテープデバイスに追加する際にテープヘッドを切り替えるには、次のコマンドを実行します。


```
# mt -f /dev/nst0 eod
```
- 前のレコードに移動するには、以下を実行します。


```
# mt -f /dev/nst0 bsm 1
```
- 次のレコードに移動するには、以下を行います。


```
# mt -f /dev/nst0 fsf 1
```

関連情報

- システム上の **mt(1)** man ページ

22.7. テープデバイスからのデータの復元

tar コマンドを使用して、テープデバイスからデータを復元できます。

前提条件

1. **mt-st** パッケージがインストールされている。詳細は、[テープドライブ管理ツールのインストール](#) を参照。
2. データはテープデバイスに書き込まれる。詳細は、[Writing to rewinding tape devices](#) または [Writing to non-rewinding tape devices](#) を参照。

手順

- 巻き戻しありのテープデバイス **/dev/st0** の場合、以下を実行します。
 - **/source/directory/** を復元します。

```
# tar -xzf /dev/st0 /source/directory/
```

- 巻き戻しなしのテープデバイス /dev/nst0 の場合、以下を実行します。

- テープデバイスを巻き戻します。

```
# mt -f /dev/nst0 rewind
```

- /etc ディレクトリーを復元します。

```
# tar -xzf /dev/nst0 /source/directory/
```

関連情報

- システム上の **mt(1)** および **tar(1)** man ページ

22.8. テープデバイスのデータの消去

erase オプションを使用して、テープデバイスからデータを消去できます。

前提条件

1. **mt-st** パッケージがインストールされている。詳細は、[テープドライブ管理ツールのインストール](#) を参照。
2. データはテープデバイスに書き込まれる。詳細は、[Writing to rewinding tape devices](#) または [Writing to non-rewinding tape devices](#) を参照。

手順

1. テープデバイスからデータを削除します。

```
# mt -f /dev/st0 erase
```

2. テープデバイスをアンロードします。

```
# mt -f /dev/st0 offline
```

関連情報

- システム上の **mt(1)** man ページ

第23章 ストレージデバイスの削除

実行中のシステムからストレージデバイスを安全に削除することで、システムメモリーのオーバーロードやデータ損失を防ぐことができます。次のシステムではストレージデバイスを削除しないでください。

- 空きメモリーが合計メモリーの 5 % 未満 (サンプル 100 件の内 10 件以上)。
- スワップが有効になっている (**vmstat** コマンドの出力で **si** と **so** のコラムが 0 以外の値)。

前提条件

- I/O フラッシュ中にシステムメモリーの負荷が増加するため、ストレージデバイスを削除する前に、システムメモリーが十分にあることを確認する。システムの現在のメモリー負荷と空きメモリーを表示するには、次のコマンドを使用します。

```
# vmstat 1 100
# free
```

23.1. ストレージデバイスの安全な削除

稼働中のシステムからストレージデバイスを安全に取り外すには、上から下へのアプローチが必要です。アプリケーションやファイルシステムなどの最上位層から始め、物理デバイスなどの最下位層に向かって作業を進めます。

ストレージデバイスは複数の方法で使用でき、物理デバイスの上層に別の仮想設定を指定できます。例えば、デバイスの複数のインスタンスをマルチパスデバイスにグループ化したり、RAID の一部にしたり、LVM グループの一部にしたりすることができます。さらに、デバイスはファイルシステムを介してアクセスすることもできるし、“raw” デバイスのように直接アクセスすることもできます。

上から下へのアプローチを用いながら、次のことを確認する必要があります。

- 削除したいデバイスが使用中でないこと
- デバイスへの保留中の I/O がすべてフラッシュされる
- オペレーティングシステムがストレージデバイスを参照していない

23.2. ブロックデバイスと関連メタデータの削除

実行中のシステムからブロックデバイスを安全に削除するには、システムメモリーのオーバーロードとデータ損失を防ぐために、最初にブロックデバイスからメタデータを削除する必要があります。ファイルシステムから始めて、スタック内の各レイヤーに対処し、ディスクに進みます。これらのアクションにより、システムが不整合な状態になるのを防ぎます。

削除するデバイスのタイプに応じて異なる特定のコマンドを使用します。

- lvremove**、**vgremove**、および **pvremove** は LVM に固有です。
- ソフトウェア RAID の場合、**mdadm** を実行してアレイを削除します。詳細は、[RAID の管理](#) を参照してください。

- LUKS を使用して暗号化されたブロックデバイスの場合、特定の追加手順があります。次の手順は、LUKS を使用して暗号化されたブロックデバイスでは機能しません。詳細は、[LUKS を使用したブロックデバイスの暗号化](#) を参照してください。

警告



SCSI バスを再びスキャンしたり、ここで説明されている手順に従わずにオペレーティングシステムを変更する別のアクションを実行すると、I/O タイムアウトが原因で遅延が発生したり、デバイスやデータが予期せず削除されたりする可能性があります。

前提条件

- ファイルシステム、論理ボリューム、およびボリュームグループを含む既存のブロックデバイススタックがある。
- 削除するデバイスを他のアプリケーションやサービスが使用していないことを確認した。
- 削除するデバイスからデータをバックアップした。
- オプション: マルチパスデバイスを削除する必要があり、そのパスデバイスにアクセスできない場合は、次のコマンドを実行してマルチパスデバイスのキューリングを無効にしておく。

```
# multipathd disablequeueing map multipath-device
```

無効にすることで、デバイスの I/O が失敗し、デバイスを使用しているアプリケーションがシャットダウンできるようになります。



注記

メタデータを含むデバイスを一度に1レイヤーずつ削除することで、ディスクに古い署名が残らないようにします。

手順

- ファイルシステムをアンマウントします。

```
# umount /mnt/mount-point
```

- ファイルシステムを削除します。

```
# wiperfs -a /dev/vg0/myvol
```

/etc/fstab ファイルにエントリーを追加して、ファイルシステムとマウントポイントの間の永続的な関連付けを作成した場合は、この時点で **/etc/fstab** を編集してそのエントリーを削除します。

削除するデバイスのタイプに応じて、次の手順に進みます。

- ファイルシステムを含む論理ボリューム (LV) を削除します。

```
# lvremove vg0/myvol
```

4. ボリュームグループ (VG) に他の論理ボリュームが残っていない場合は、デバイスを含む VG を安全に削除できます。

```
# vgremove vg0
```

5. 物理ボリューム (PV) メタデータを PV デバイスから削除します。

```
# pvremove /dev/sdc1
```

```
# wipefs -a /dev/sdc1
```

6. PV が含まれていたパーティションを削除します。

```
# parted /dev/sdc rm 1
```

7. デバイスを完全に消去する場合は、パーティションテーブルを削除します。

```
# wipefs -a /dev/sdc
```

8. デバイスを物理的に取り外す場合にのみ、次の手順を実行します。

- マルチパスデバイスを削除する場合は、次のコマンドを実行します。

- デバイスへの全パスを表示します。

```
# multipath -l
```

このコマンドの出力は、後のステップで必要になります。

- I/O をフラッシュして、マルチパスデバイスを削除します。

```
# multipath -f multipath-device
```

- デバイスがマルチパスデバイスとして設定されていない場合や、デバイスがマルチパスデバイスとして設定されていて、過去に I/O を個別のパスに渡している場合は、未処理の I/O を、使用されている全デバイスパスにフラッシュします。

```
# blockdev --flushbufs device
```

この操作は、**umount** コマンドまたは **vgreduce** コマンドで I/O がフラッシュされないデバイスに直接アクセスする場合に重要になります。

- SCSI デバイスを取り外す場合は、以下のコマンドを実行します。

- システム上のアプリケーション、スクリプト、またはユーティリティーで、**/dev/sd**、**/dev/disk/by-path**、または **major:minor** 番号など、デバイスのパスベースの名前への参照をすべて削除します。参照を削除することで、今後追加される別のデバイスが現在のデバイスと混同されないようにします。

- SCSI サブシステムからデバイスへの各パスを削除します。

```
# echo 1 > /sys/block/device-name/device/delete
```

デバイスが以前にマルチパスデバイスとして使用されていた場合、**device-name** は、**multipath -l** コマンドの出力からの内容に置き換えます。

9. 稼働中のシステムから物理デバイスを削除します。このデバイスを削除しても、他のデバイスへの I/O は停止しないことに注意してください。

検証

- 削除したデバイスが **lsblk** コマンドの出力に表示されないことを確認します。出力例を以下に示します。

```
# lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
sda	8:0	0	5G	0	disk	
sr0	11:0	1	1024M	0	rom	
vda	252:0	0	10G	0	disk	
-vda1	252:1	0	1M	0	part	
-vda2	252:2	0	100M	0	part	/boot/efi
`-vda3	252:3	0	9.9G	0	part	/

関連情報

- システム上の **multipath(8)**、**pvremove(8)**、**vgremove(8)**、**lvremove(8)**、**wipefs(8)**、**parted(8)**、**blockdev(8)**、および **umount(8)** man ページ

第24章 STRATIS ファイルシステムの設定

Stratis は、Red Hat Enterprise Linux 用のローカルストレージ管理ソリューションです。これは、シンプルさと使いやすさを重視し、高度なストレージ機能にアクセスできます。

Stratis は、物理ストレージデバイスのプールを管理するためにサービスとして実行され、複雑なストレージ設定のセットアップと管理を支援しながら、ローカルストレージ管理を使いやすく簡素化します。



重要

Stratis はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat では、実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、今後予定されている製品の機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲の詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

Stratis は、以下のために使用できます。

- ストレージの初期設定
- その後の変更
- 高度なストレージ機能の使用

Stratis の中心的な概念はストレージプールです。このプールは 1 つ以上のローカルディスクまたはパーティションから作成され、ファイルシステムはプールから作成されます。プールでは次のような機能が有効になります。

- ファイルシステムのスナップショット
- シンプロビジョンング
- キャッシュ
- 暗号化

24.1. STRATIS ファイルシステムのコンポーネント

外部的には、Stratis はコマンドラインと API を通じて次のファイルシステムコンポーネントを提供します。

blockdev

ディスクやディスクパーティションなどのブロックデバイス。

pool

1 つ以上のブロックデバイスで構成されます。

プールの合計サイズは固定で、ブロックデバイスのサイズと同じです。

プールには、**dm-cache** ターゲットを使用した不揮発性データキャッシュなど、ほとんどの Stratis レイヤーが含まれています。

Stratis は、各プールの `/dev/stratis/my-pool/` ディレクトリーを作成します。このディレクトリーには、プール内の Stratis ファイルシステムを表すデバイスへのリンクが含まれています。

filesystem

各プールには 0 個以上のファイルシステムを含めることができます。ファイルシステムを含むプールには、任意の数のファイルを保存できます。

ファイルシステムはシンプロビジョニングされており、合計サイズは固定されていません。ファイルシステムの実際のサイズは、そこに格納されているデータとともに大きくなります。データのサイズがファイルシステムの仮想サイズに近づくと、Stratis はシンボリュームとファイルシステムを自動的に拡張します。

ファイルシステムは XFS ファイルシステムでフォーマットされます。



重要

Stratis は、XFS が認識しない、作成されたファイルシステムに関する情報を追跡します。また、XFS を使用して行われた変更によって、Stratis が自動的に更新されることはありません。ユーザーは、Stratis が管理する XFS ファイルシステムを再フォーマットまたは再設定しないでください。

Stratis は、`/dev/stratis/my-pool/my-fs` パスにファイルシステムへのリンクを作成します。

Stratis は、**dmsetup** リストと **/proc/partitions** ファイルに表示される多くの Device Mapper デバイスを使用します。同様に、**lsblk** コマンドの出力は、Stratis の内部の仕組みとレイヤーを反映します。

24.2. STRATIS と互換性のあるブロックデバイス

Stratis で使用可能なストレージデバイス。

対応デバイス

Stratis プールは、次の種類のブロックデバイスで動作するかどうかをテスト済みです。

- LUKS
- LVM 論理ボリューム
- MD RAID
- DM Multipath
- iSCSI
- HDD および SSD
- NVMe デバイス

対応していないデバイス

Stratis にはシンプロビジョニングレイヤーが含まれているため、Red Hat はすでにシンプロビジョニングされているブロックデバイスに Stratis プールを配置することを推奨しません。

24.3. STRATIS のインストール

Stratis に必要なパッケージをインストールします。

手順

- Stratis サービスとコマンドラインユーティリティーを提供するパッケージをインストールします。

```
# dnf install stratisd stratis-cli
```

2. **stratisd** サービスを開始し、ブート時に起動できるようにするには、以下を実行します。

```
# systemctl enable --now stratisd
```

検証

- **stratisd** サービスが有効になっていて実行されていることを確認します。

```
# systemctl status stratisd
stratisd.service - Stratis daemon
  Loaded: loaded (/usr/lib/systemd/system/stratisd.service; enabled; preset:>)
  Active: active (running) since Tue 2025-03-25 14:04:42 CET; 30min ago
    Docs: man:stratisd(8)
   Main PID: 24141 (stratisd)
     Tasks: 22 (limit: 99365)
    Memory: 10.4M
      CPU: 1.436s
     CGroup: /system.slice/stratisd.service
             └─24141 /usr/libexec/stratisd --log-level debug
```

24.4. 暗号化されていない STRATIS プールの作成

1つ以上のブロックデバイスから暗号化されていない Stratis プールを作成できます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。
- IBM Z アーキテクチャーで、**/dev/dasd*** ブロックデバイスがパーティション設定されている。Stratis プールの作成には、パーティションデバイスを使用します。DASD デバイスのパーティション設定の詳細は、[IBM Z での Linux インスタンスの設定](#) を参照してください。



注記

Stratis プールは作成時にのみ暗号化でき、後から暗号化することはできません。

手順

- Stratis プールで使用する各ブロックデバイスに存在するファイルシステム、パーティション テーブル、または RAID 署名をすべて削除します。

```
# wipefs --all block-device
```

block-device の値は、ブロックデバイスへのパスです (例: `/dev/sdb`)。

- 選択したブロックデバイスに新しい暗号化されていない Stratis プールを作成します。

```
# stratis pool create my-pool block-device
```

block-device の値は、空または消去されたブロックデバイスへのパスです。

次のコマンドを使用して、1行に複数のブロックデバイスを指定することもできます。

```
# stratis pool create my-pool block-device-1 block-device-2
```

検証

- 新しい Stratis プールが作成されていることを確認します。

```
# stratis pool list
```

24.5. WEB コンソールを使用した暗号化されていない STRATIS プールの作成

Web コンソールを使用して、1つ以上のブロックデバイスから暗号化されていない Stratis プールを作成できます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザー アカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。



注記

暗号化されていない Stratis プールの作成後に、当該 Stratis プールを暗号化することはできません。

手順

- RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。

2. **Storage** をクリックします。
3. **Storage** テーブルで、メニュー ボタンをクリックし、**Create Stratis pool** を選択します。
4. **Name** フィールドに、Stratis プールの名前を入力します。
5. Stratis プールの作成元となる **Block devices** を選択します。
6. オプション: プールに作成する各ファイルシステムの最大サイズを指定する場合は、**Manage filesystem sizes** を選択します。
7. **Create** をクリックします。

検証

- **Storage** セクションに移動し、**Devices** テーブルに新しい Stratis プールが表示されていることを確認します。

24.6. カーネルキーリング内のキーを使用して暗号化された STRATIS プールを作成する

データを保護するために、カーネルキーリングを使用して、1つ以上のブロックデバイスから暗号化された Stratis プールを作成できます。

この方法で暗号化された Stratis プールを作成すると、カーネルキーリングはプライマリー暗号化メカニズムとして使用されます。その後のシステムを再起動すると、このカーネルキーリングは、暗号化された Stratis プールのロックを解除します。

1つ以上のブロックデバイスから暗号化された Stratis プールを作成する場合は、次の点に注意してください。

- 各ブロックデバイスは **cryptsetup** ライブラリーを使用して暗号化され、**LUKS2** 形式を実装します。
- 各 Stratis プールは、一意の鍵を持つか、他のプールと同じ鍵を共有できます。これらのキーはカーネルキーリングに保存されます。
- Stratis プールを構成するブロックデバイスは、すべて暗号化または暗号化されていないデバイスである必要があります。同じ Stratis プールに、暗号化したブロックデバイスと暗号化されていないブロックデバイスの両方を含めることはできません。
- 暗号化 Stratis プールのデータキャッシュに追加されるブロックデバイスは、自動的に暗号化されます。

前提条件

- Stratis v2.1.0 以降がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。
- IBM Z アーキテクチャーで、**/dev/dasd*** ブロックデバイスがパーティション設定されている。Stratis プールでパーティションを使用します。

DASD デバイスのパーティション設定の詳細は、[IBM Z での Linux インスタンスの設定](#) を参照してください。

手順

- Stratis プールで使用する各ブロックデバイスに存在するファイルシステム、パーティションテーブル、または RAID 署名をすべて削除します。

```
# wipefs --all block-device
```

block-device の値は、ブロックデバイスへのパスです (例: `/dev/sdb`)。

- キーをまだ設定していない場合には、以下のコマンドを実行してプロンプトに従って、暗号化に使用するキーセットを作成します。

```
# stratis key set --capture-key key-description
```

key-description は、カーネルキーリングで作成されるキーへの参照になります。コマンドラインで、キー値の入力を求められます。キー値をファイルに配置し、**--capture-key** オプションの代わりに **--keyfile-path** オプションを使用することもできます。

- 暗号化した Stratis プールを作成し、暗号化に使用すキーの説明を指定します。

```
# stratis pool create --key-desc key-description my-pool block-device
```

key-description

直前の手順で作成したカーネルキーリングに存在するキーを参照します。

my-pool

新しい Stratis プールの名前を指定します。

block-device

空のブロックデバイスまたは消去したブロックデバイスへのパスを指定します。

次のコマンドを使用して、1行に複数のブロックデバイスを指定することもできます。

```
# stratis pool create --key-desc key-description my-pool block-device-1 block-device-2
```

検証

- 新しい Stratis プールが作成されていることを確認します。

```
# stratis pool list
```

24.7. WEB コンソールを使用した暗号化された STRATIS プールの作成

データを保護するために、Web コンソールを使用して、1つ以上のブロックデバイスから暗号化された Stratis プールを作成できます。

1つ以上のブロックデバイスから暗号化された Stratis プールを作成する場合は、次の点に注意してください。

- 各ブロックデバイスは cryptsetup ライブラリーを使用して暗号化され、LUKS2 形式を実装します。
- 各 Stratis プールは、一意の鍵を持つか、他のプールと同じ鍵を共有できます。これらのキーはカーネルキーリングに保存されます。
- Stratis プールを構成するブロックデバイスは、すべて暗号化または暗号化されていないデバイスである必要があります。同じ Stratis プールに、暗号化したブロックデバイスと暗号化されていないブロックデバイスの両方を含めることはできません。
- 暗号化 Stratis プールのデータ層に追加されるブロックデバイスは、自動的に暗号化されます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザー アカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- Stratis v2.1.0 以降がインストールされ、**stratisd** サービスが実行されている。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。

手順

- RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
- Storage** をクリックします。
- Storage テーブルで、メニュー ボタンをクリックし、**Create Stratis pool** を選択します。
- Name** フィールドに、Stratis プールの名前を入力します。
- Stratis プールの作成元となる **Block devices** を選択します。
- 暗号化のタイプを選択します。パスフレーズ、Tang キーサーバー、またはその両方を使用できます。
 - パスフレーズ:
 - パスフレーズを入力します。
 - パスフレーズを確定します。
 - Tang キーサーバー:
 - キーサーバーのアドレスを入力します。詳細は、[SELinux を Enforcing モードで有効にした Tang サーバーのデプロイメント](#) を参照してください。
- オプション: プールに作成する各ファイルシステムの最大サイズを指定する場合は、**Manage filesystem sizes** を選択します。
- Create** をクリックします。

検証

- Storage セクションに移動し、Devices テーブルに新しい Stratis プールが表示されていることを確認します。

24.8. WEB コンソールを使用した STRATIS プールの名前変更

Web コンソールを使用して、既存の Stratis プールの名前を変更できます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザー アカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- Stratis がインストールされ、**stratisd** サービスが実行されている。
Web コンソールがデフォルトで Stratis を検出してインストールしている。ただし、Stratis を手動でインストールする場合は、[Stratis のインストール](#) を参照してください。
- Stratis プールが作成されている。

手順

- RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
- Storage** をクリックします。
- Storage テーブルで、名前を変更する Stratis プールをクリックします。
- Stratis pool ページで、Name フィールドの横にある **edit** をクリックします。
- Rename Stratis pool ダイアログボックスで、新しい名前を入力します。
- Rename** をクリックします。

24.9. STRATIS ファイルシステムでのオーバープロビジョニングモードの設定

デフォルトでは、すべての Stratis プールはオーバープロビジョニングされており、論理ファイルシステムのサイズが物理的に割り当てられた領域を超える可能性があります。Stratis はファイルシステムの使用状況を監視し、必要に応じて使用可能な領域を使用して割り当てを自動的に増やします。ただし、すでに使用可能な領域がすべて割り当てられており、プールがいっぱいの場合は、ファイルシステムに追加領域を割り当てるることはできません。



注記

ファイルシステムの容量が不足すると、ユーザーはデータを失う可能性があります。データ損失のリスクがオーバープロビジョニングの利点を上回るアプリケーションの場合、この機能を無効にできます。

Stratis はプールの使用状況を継続的に監視し、D-Bus API を使用して値を報告します。ストレージ管理者はこれらの値を監視し、容量に達しないように必要に応じてデバイスをプールに追加する必要があります。

前提条件

- Stratis がインストールされている。詳細は、[Stratis のインストール](#) を参照してください。

手順

プールを正しく設定するには、次の 2 つの方法があります。

- 1つ以上のブロックデバイスからプールを作成します。

```
# stratis pool create pool-name /dev/sdb
```

- 既存のプールにオーバープロビジョニングモードを設定します。

```
# stratis pool overprovision pool-name <yes|no>
```

- "yes" に設定すると、プールへのオーバープロビジョニングが有効になります。これは、プールによってサポートされる Stratis ファイルシステムの論理サイズの合計が、利用可能なデータ領域の量を超える可能性があることを意味します。プールがオーバープロビジョニングされ、すべてのファイルシステムの論理サイズの合計がプールで使用可能な領域を超える場合、システムはオーバープロビジョニングをオフにできず、エラーを返します。

検証

- Stratis プールの完全なリストを表示します。

```
# stratis pool list
```

Name	Total Physical	Properties	UUID	Alerts
pool-name	1.42 TiB / 23.96 MiB / 1.42 TiB	~Ca,~Cr,~Op	cb7cb4d8-9322-4ac4-a6fd-eb7ae9e1e540	

- stratis pool list** の出力に、プールのオーバープロビジョニングモードフラグが表示されているかどうかを確認します。"~" は "NOT" を表す数学記号であるため、**~Op** はオーバープロビジョニングなしという意味です。

- オプション: 特定のプールのオーバープロビジョニングを確認します。

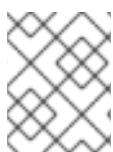
```
# stratis pool overprovision pool-name yes
```

```
# stratis pool list
```

Name	Total Physical	Properties	UUID	Alerts
pool-name	1.42 TiB / 23.96 MiB / 1.42 TiB	~Ca,~Cr,~Op	cb7cb4d8-9322-4ac4-a6fd-eb7ae9e1e540	

24.10. STRATIS プールの NBDE へのバインド

暗号化された Stratis プールを Network Bound Disk Encryption (NBDE) にバインドするには、Tang サーバーが必要です。Stratis プールを含むシステムが再起動すると、Tang サーバーに接続して、カーネルキーリングの説明を指定しなくても、暗号化したプールのロックを自動的に解除します。



注記

Stratis プールを補助 Clevis 暗号化メカニズムにバインドすると、プライマリーカーネルキーリング暗号化は削除されません。

前提条件

- Stratis v2.3.0 以降がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- 暗号化した Stratis プールを作成し、暗号化に使用したキーの説明がある。詳細は、[カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。
- Tang サーバーに接続できる。詳細は、[SELinux を Enforcing モードで有効にした Tang サーバーのデプロイメント](#) を参照してください。

手順

- 暗号化された Stratis プールを NBDE にバインドする。

```
# stratis pool bind nbde --trust-url my-pool tang-server
```

my-pool

暗号化された Stratis プールの名前を指定します。

tang-server

Tang サーバーの IP アドレスまたは URL を指定します。

関連情報

- [ポリシーベースの復号を使用して暗号化ボリュームの自動アンロックの設定](#)

24.11. STRATIS プールの TPM へのバインド

暗号化された Stratis プールを Trusted Platform Module (TPM) 2.0 にバインドすると、プールを含むシステムが再起動され、カーネルキーリングの説明を指定しなくても、プールは自動的にロック解除されます。

前提条件

- Stratis v2.3.0 以降がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- 暗号化した Stratis プールを作成し、暗号化に使用したキーの説明がある。詳細は、[カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。
- 使用しているシステムは TPM 2.0 をサポートしている。

手順

- 暗号化された Stratis プールを TPM にバインドします。

```
# stratis pool bind tpm my-pool
```

my-pool

暗号化された Stratis プールの名前を指定します。

key-description

暗号化された Stratis プールの作成時に生成されたカーネルキーリングに存在するキーを参照します。

24.12. カーネルキーリングを使用した暗号化 STRATIS プールのロック解除

システムの再起動後、暗号化した Stratis プール、またはこれを構成するブロックデバイスが表示されない場合があります。プールの暗号化に使用したカーネルキーリングを使用して、プールのロックを解除できます。

前提条件

- Stratis v2.1.0 がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- 暗号化された Stratis プールが作成されている。詳細は、[カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。

手順

1. 以前使用したものと同じキー記述を使用して、キーセットを再作成します。

```
# stratis key set --capture-key key-description
```

key-description は、暗号化された Stratis プールの作成時に生成されたカーネルキーリングに存在するキーを参照します。

2. Stratis プールが表示されることを確認します。

```
# stratis pool list
```

24.13. 補助暗号化からの STRATIS プールのバインド解除

暗号化した Stratis プールを、サポート対象の補助暗号化メカニズムからバインドを解除すると、プライマリーカーネルキーリングの暗号化はそのまま残ります。これは、最初から Clevis 暗号化を使用して作成されたプールには当てはまりません。

前提条件

- Stratis v2.3.0 以降がシステムにインストールされている。詳細は、[Stratis のインストール](#) を参照してください。
- 暗号化された Stratis プールが作成されている。詳細は、[カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。
- 暗号化した Stratis プールは、サポート対象の補助暗号化メカニズムにバインドされます。

手順

- 補助暗号化メカニズムから暗号化された Stratis プールのバインドを解除します。

```
# stratis pool unbind clevis my-pool
```

my-pool は、バインドを解除する Stratis プールの名前を指定します。

関連情報

- 暗号化された Stratis プールの NBDE へのバインド
- 暗号化された Stratis プールの TPM へのバインド

24.14. STRATIS プールの開始および停止

Stratis プールを開始および停止できます。この操作により、ファイルシステム、キャッシングデバイス、シンプール、暗号化されたデバイスなど、プールの構築に使用されたすべてのオブジェクトを解体または停止できます。プールがデバイスまたはファイルシステムをアクティブに使用している場合は、警告が表示され、停止できない可能性があることに注意してください。

停止状態は、プールのメタデータに記録されます。これらのプールは、プールが開始コマンドを受信するまで、次のブートでは開始されません。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- 暗号化されていない、または暗号化された Stratis プールを作成した。詳細は、[暗号化されていない Stratis プールの作成](#) または [カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。

手順

- 次のコマンドを使用して、Stratis プールを停止します。これにより、ストレージスタックが切断されますが、メタデータはすべて保持されます。

```
# stratis pool stop --name pool-name
```

- 以下のコマンドを使用して Stratis プールを起動します。**--unlock-method** オプションは、プールが暗号化されている場合にプールのロックを解除する方法を指定します。

```
# stratis pool start --unlock-method <keyring|clevis> --name pool-name
```



注記

プール名またはプール UUID のいずれかを使用してプールを開始できます。

検証

- 次のコマンドを使用して、システム上のすべてのアクティブなプールをリスト表示します。

```
# stratis pool list
```

- 次のコマンドを使用して、停止されたプールをすべてリスト表示します。

```
# stratis pool list --stopped
```

- 次のコマンドを使用して、停止したプールの詳細情報を表示します。UUID を指定すると、このコマンドは UUID に対応するプールに関する詳細情報を出力します。

```
# stratis pool list --stopped --uuid UUID
```

24.15. STRATIS ファイルシステムの作成

既存の Stratis プールに Stratis ファイルシステムを作成します。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールが作成されている。詳細は、[暗号化されていない Stratis プールの作成](#) または [カーネルキーリング内のキーの使用](#) を参照してください。

手順

- プールに Stratis ファイルシステムを作成します。

```
# stratis filesystem create --size number-and-unit my-pool my-fs
```

number-and-unit

ファイルシステムのサイズを指定します。仕様形式は、入力の標準サイズ指定形式 (B、KiB、MiB、GiB、TiB、またはPiB) に準拠する必要があります。

my-pool

Stratis プールの名前を指定します。

my-fs

ファイルシステムの任意名を指定します。

以下に例を示します。

例24.1 Stratis ファイルシステムの作成

```
# stratis filesystem create --size 10GiB pool1 filesystem1
```

検証

- プール内のファイルシステムをリスト表示して、Stratis ファイルシステムが作成されているかどうかを確認します。

```
# stratis fs list my-pool
```

関連情報

- [Stratis ファイルシステムのマウント](#)

24.16. WEB コンソールを使用した STRATIS プール上のファイルシステムの作成

Web コンソールを使用して、既存の Stratis プール上にファイルシステムを作成できます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **stratisd** サービスを実行している。
- Stratis プールが作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. **Storage** をクリックします。
3. ファイルシステムを作成する Stratis プールをクリックします。
4. Stratis pool ページで、Stratis filesystems セクションまでスクロールし、**Create new filesystem** をクリックします。
5. ファイルシステムの名前を入力します。
6. ファイルシステムのマウントポイントを入力します。
7. マウントオプションを選択します。
8. At boot ドロップダウンメニューで、ファイルシステムをマウントするタイミングを選択します。
9. ファイルシステムを作成します。
 - ファイルシステムを作成してマウントする場合は、**Create and mount** をクリックします。
 - ファイルシステムの作成のみを行う場合は、**Create only** をクリックします。

検証

- 新しいファイルシステムは、Stratis pool ページの Stratis filesystems タブに表示されます。

24.17. STRATIS ファイルシステムのマウント

既存の Stratis ファイルシステムをマウントして、コンテンツにアクセスします。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis ファイルシステムが作成されます。詳細は、[Stratis ファイルシステムの作成](#) を参照してください。

手順

- ファイルシステムをマウントするには、**/dev/stratis/** ディレクトリーに Stratis が維持するエンターを使用します。

```
# mount /dev/stratis/my-pool/my-fs mount-point
```

これでファイルシステムは **mount-point** ディレクトリーにマウントされ、使用できるようになりました。



注記

プールを停止する前に、プールに属するすべてのファイルシステムをアンマウントします。ファイルシステムがまだマウントされている場合、プールは停止しません。

24.18. SYSTEMD サービスを使用した /ETC/FSTAB での非ルート STRATIS ファイルシステムの設定

systemd サービスを使用して、**/etc/fstab** で非ルートファイルシステムの設定を管理できます。

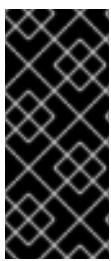
前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis ファイルシステムが作成されます。詳細は、[Stratis ファイルシステムの作成](#) を参照してください。

手順

- root** として、**/etc/fstab** ファイルを編集し、非ルートファイルシステムを設定するための行を追加します。

```
/dev/stratis/my-pool/my-fs mount-point xfs defaults,x-systemd.requires=stratis-fstab-
setup@pool-uuid.service,x-systemd.after=stratis-fstab-setup@pool-uuid.service dump-
value fsck_value
```



重要

暗号化された Stratis プールから Stratis ファイルシステムを永続的にマウントすると、パスワードが入力されるまでブートプロセスが停止する可能性があります。プールが NBDE や TPM2 などの無人メカニズムを使用して暗号化されている場合、Stratis プールは自動的にロック解除されます。そうでない場合、ユーザーはコンソールにパスワードを入力する必要があります。

関連情報

- [ファイルシステムの永続的なマウント](#)

第25章 追加のブロックデバイスで STRATIS プールを拡張する

Stratis ファイルシステムのストレージ容量を増やすために、追加のブロックデバイスを Stratis プールに追加できます。手動で行うことも、Web コンソールを使用して行うこともできます。



重要

Stratis はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat では、実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、今後予定されている製品の機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲の詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

25.1. STRATIS プールへのブロックデバイスの追加

Stratis プールに1つ以上のブロックデバイスを追加できます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。

手順

- 1つ以上のブロックデバイスをプールに追加するには、以下を使用します。

```
# stratis pool add-data my-pool device-1 device-2 device-n
```

関連情報

- システム上の **stratis(8)** man ページ

25.2. WEB コンソールを使用した STRATIS プールへのブロックデバイスの追加

Web コンソールを使用して、既存の Stratis プールにブロックデバイスを追加できます。キャッシュをブロックデバイスとして追加することもできます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
 - cockpit サービスが有効になっている。
 - ユーザー アカウントが Web コンソールにログインできる。
- 手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。

- **stratisd** サービスを実行している。
- Stratis プールが作成されている。
- Stratis プールを作成するブロックデバイスは、使用もマウントもされておらず、1GB 以上の領域がある。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. **Storage** をクリックします。
3. **Storage** テーブルで、ブロックデバイスを追加する Stratis プールをクリックします。
4. **Stratis pool** ページで、**Add block devices** をクリックし、ブロックデバイスをデータまたはキャッシュとして追加する **Tier** を選択します。
5. パスフレーズで暗号化された Stratis プールにブロックデバイスを追加する場合は、パスフレーズを入力します。
6. **Block devices** で、プールに追加するデバイスを選択します。
7. **Add** をクリックします。

第26章 STRATIS ファイルシステムの監視

Stratis ユーザーは、システムにある Stratis ファイルシステムに関する情報を表示して、その状態と空き容量を監視できます。



重要

Stratis はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat では、実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、今後予定されている製品の機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲の詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

26.1. STRATIS ファイルシステムに関する情報の表示

stratis ユーティリティーを使用すると、Stratis ファイルシステムの合計サイズ、使用サイズ、空きサイズ、プールに属するファイルシステムとブロックデバイスなどに関する統計情報をリスト表示できます。

XFS ファイルシステムのサイズは、管理できるユーザーデータの合計量です。シンプロビジョニングされた Stratis プールでは、Stratis ファイルシステムのサイズが、割り当てられた領域よりも大きいように見える場合があります。XFS ファイルシステムのサイズは、この見かけのサイズに合わせて設定されます。つまり、通常は割り当てられた領域よりも大きくなります。df などの標準 Linux ユーティリティーは、XFS ファイルシステムのサイズを報告します。この値は通常、XFS ファイルシステムに必要な領域、つまり Stratis によって割り当てられた領域を過大評価します。



重要

オーバープロビジョニングされた Stratis プールの使用状況を定期的に監視してください。ファイルシステムの使用量が割り当てられた領域に近づくと、Stratis はプール内の使用可能な領域を使用して割り当てを自動的に増加します。ただし、すでに使用可能な領域がすべて割り当てられていて、プールがいっぱいの場合は、追加の領域を割り当てることができず、ファイルシステムの領域が不足することになります。これにより、Stratis ファイルシステムを使用するアプリケーションでデータが失われるリスクが生じる可能性があります。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。

手順

- システムで Stratis に使用されているすべての ブロックデバイス に関する情報を表示する場合は、次のコマンドを実行します。

```
# stratis blockdev
```

Pool Name	Device	Node	Physical Size	State	Tier
my-pool	/dev/sdb		9.10 TiB	In-use	Data

- システムにあるすべての Stratis プールに関する情報を表示するには、次のコマンドを実行します。

```
# stratis pool

Name  Total Physical Size  Total Physical Used
my-pool  9.10 TiB  598 MiB
```

- システムにあるすべての Stratis ファイルシステムに関する情報を表示するには、次のコマンドを実行します。

```
# stratis filesystem

Pool Name  Name  Used  Created  Device
my-pool  my-fs  546 MiB  Nov 08 2018 08:03  /dev/stratis/my-pool/my-fs
```

関連情報

- システム上の **stratis(8)** man ページ

26.2. WEB コンソールを使用した STRATIS プールの表示

Web コンソールを使用して、既存の Stratis プールとそれに含まれるファイルシステムを表示できます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- stratisd** サービスを実行している。
- 既存の Stratis プールがある。

手順

- RHEL 8 Web コンソールにログインします。
- Storage** をクリックします。
- Storage** テーブルで、表示する Stratis プールをクリックします。
Stratis プールページには、プールおよびプール内に作成したファイルシステムに関するすべての情報が表示されます。

第27章 STRATIS ファイルシステムでのスナップショットの使用

Stratis ファイルシステムのスナップショットを使用して、ファイルシステムの状態を任意の時点でキャプチャーし、後でそれを復元できます。



重要

Stratis はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat では、実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、今後予定されている製品の機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲の詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

27.1. STRATIS スナップショットの特徴

Stratis では、スナップショットは、別の Stratis ファイルシステムのコピーとして作成した通常の Stratis ファイルシステムです。

Stratis の現在のスナップショット実装は、次のような特徴があります。

- ファイルシステムのスナップショットは別のファイルシステムです。
- スナップショットと元のファイルシステムのリンクは、有効期間中は行われません。スナップショットされたファイルシステムは、元のファイルシステムよりも長く存続します。
- スナップショットを作成するためにファイルシステムをマウントする必要はありません。
- 各スナップショットは、XFS ログに必要となる実際のバックингストレージの約半分のギガバイトを使用します。

27.2. STRATIS スナップショットの作成

既存の Stratis ファイルシステムのスナップショットとして Stratis ファイルシステムを作成できます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis ファイルシステムを作成している。詳細は、[Stratis ファイルシステムの作成](#) を参照してください。

手順

- Stratis スナップショットを作成します。

```
# stratis fs snapshot my-pool my-fs my-fs-snapshot
```

スナップショットは、中心的な Stratis ファイルシステムです。Stratis スナップショットは複数作成できます。単一のオリジンファイルシステム、または別のスナップショットファイルシステムのスナップショットがあります。ファイルシステムがスナップショットの場合、**origin** フィールドには、詳細な

ファイルシステムリストの元のファイルシステムの UUID が表示されます。

関連情報

- システム上の **stratis(8)** man ページ

27.3. STRATIS スナップショットのコンテンツへのアクセス

Stratis ファイルシステムのスナップショットを、読み取りおよび書き込み操作でアクセスできるようにマウントすることができます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis スナップショットを作成している。詳細は、[Stratis スナップショットの作成](#) を参照してください。

手順

- スナップショットにアクセスするには、`/dev/stratis/my-pool/` ディレクトリーから通常のファイルシステムとしてマウントします。

```
# mount /dev/stratis/my-pool/my-fs-snapshot mount-point
```

関連情報

- [Stratis ファイルシステムのマウント](#)
- システム上の **mount(8)** man ページ

27.4. STRATIS ファイルシステムを以前のスナップショットに戻す

Stratis ファイルシステムの内容を、Stratis スナップショットでキャプチャされた状態に戻すことができます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis スナップショットを作成している。詳細は、[Stratis スナップショットの作成](#) を参照してください。

手順

- オプション: 後でアクセスできるように、ファイルシステムの現在の状態をバックアップします。

```
# stratis filesystem snapshot my-pool my-fs my-fs-backup
```

- 元のファイルシステムをアンマウントして削除します。

```
# umount /dev/stratis/my-pool/my-fs
# stratis filesystem destroy my-pool my-fs
```

3. 元のファイルシステムの名前でスナップショットのコピーを作成します。

```
# stratis filesystem snapshot my-pool my-fs-snapshot my-fs
```

4. 元のファイルシステムと同じ名前でアクセスできるようになったスナップショットをマウントします。

```
# mount /dev/stratis/my-pool/my-fs mount-point
```

`my-fs` という名前のファイルシステムの内容は、スナップショット `my-fs-snapshot` と同じになりました。

関連情報

- システム上の **stratis(8)** man ページ

27.5. STRATIS スナップショットの削除

プールから Stratis スナップショットを削除できます。スナップショットのデータは失われます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis スナップショットを作成している。詳細は、[Stratis スナップショットの作成](#) を参照してください。

手順

1. スナップショットをアンマウントします。

```
# umount /dev/stratis/my-pool/my-fs-snapshot
```

2. スナップショットを破棄します。

```
# stratis filesystem destroy my-pool my-fs-snapshot
```

関連情報

- システム上の **stratis(8)** man ページ

第28章 STRATIS ファイルシステムの削除

既存の Stratis ファイルシステムまたはプールを削除できます。削除した Stratis ファイルシステムやプールは復元できません。



重要

Stratis はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat では、実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、今後予定されている製品の機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲の詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

28.1. STRATIS ファイルシステムの削除

既存の Stratis ファイルシステムを削除できます。そこに保存されているデータは失われます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis ファイルシステムを作成している。詳細は、[Stratis ファイルシステムの作成](#) を参照してください。

手順

- ファイルシステムをアンマウントします。

```
# umount /dev/stratis/my-pool/my-fs
```

- ファイルシステムを破棄します。

```
# stratis filesystem destroy my-pool my-fs
```

検証

- ファイルシステムがもう存在しないことを確認します。

```
# stratis filesystem list my-pool
```

関連情報

- システム上の **stratis(8)** man ページ

28.2. WEB コンソールを使用した STRATIS プールからのファイルシステムの削除

Web コンソールを使用して、既存の Stratis プールからファイルシステムを削除できます。



注記

Stratis プールのファイルシステムを削除すると、そこに含まれるすべてのデータが消去されます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- Stratis がインストールされ、**stratisd** サービスが実行されている。
Web コンソールがデフォルトで Stratis を検出してインストールしている。ただし、Stratis を手動でインストールする場合は、[Stratis のインストール](#) を参照してください。
- 既存の Stratis プールがあり、その Stratis プール上にファイルシステムが作成されている。

手順

1. RHEL 8 Web コンソールにログインします。
詳細は、[Web コンソールへのログイン](#) を参照してください。
2. **Storage** をクリックします。
3. **Storage** テーブルで、ファイルシステムを削除する Stratis プールをクリックします。
4. **Stratis pool** ページで、**Stratis filesystems** セクションまでスクロールし、削除するファイルシステムのメニュー ボタン **:** をクリックします。
5. ドロップダウンメニューから **Delete** を選択します。
6. **Confirm deletion** ダイアログ ボックスで、**Delete** をクリックします。

28.3. STRATIS プールの削除

既存の Stratis プールを削除できます。そこに保存されているデータは失われます。

前提条件

- Stratis がインストールされ、**stratisd** サービスが実行されている。詳細は、[Stratis のインストール](#) を参照してください。
- Stratis プールを作成している。
 - 暗号化されていないプールを作成するには、[暗号化されていない Stratis プールの作成](#) を参照してください。
 - 暗号化されたプールを作成するには、[カーネルキーリング内のキーを使用して暗号化された Stratis プールを作成する](#) を参照してください。

手順

1. プールにあるファイルシステムのリストを表示します。

```
# stratis filesystem list my-pool
```

2. プール上のすべてのファイルシステムをアンマウントします。

```
# umount /dev/stratis/my-pool/my-fs-1 \
/dev/stratis/my-pool/my-fs-2 \
/dev/stratis/my-pool/my-fs-n
```

3. ファイルシステムを破棄します。

```
# stratis filesystem destroy my-pool my-fs-1 my-fs-2
```

4. プールを破棄します。

```
# stratis pool destroy my-pool
```

検証

- プールがなくなったことを確認します。

```
# stratis pool list
```

関連情報

- システム上の **stratis(8)** man ページ

28.4. WEB コンソールを使用した STRATIS プールの削除

Web コンソールを使用して、既存の Stratis プールを削除できます。



注記

Stratis プールを削除すると、そこに含まれるすべてのデータが消去されます。

前提条件

- RHEL 8 Web コンソールがインストールされている。
- cockpit サービスが有効になっている。
- ユーザーアカウントが Web コンソールにログインできる。
手順は、[Web コンソールのインストールおよび有効化](#) を参照してください。
- **stratisd** サービスを実行している。
- 既存の Stratis プールがある。

手順

1. RHEL 8 Web コンソールにログインします。

詳細は、[Web コンソールへのログイン](#) を参照してください。

2. **Storage** をクリックします。
3. **Storage** テーブルで、削除する Stratis プールのメニュー ボタン  をクリックします。
4. ドロップダウンメニューから **Delete pool** を選択します。
5. **Permanently delete pool** ダイアログ ボックスで、**Delete** をクリックします。