Red Hat Enterprise Linux 8

ファイルシステムの管理

Red Hat Enterprise Linux 8 でのファイルシステムの作成、変更、管理

Last Updated: 2024-04-18

Red Hat Enterprise Linux 8 ファイルシステムの管理

Red Hat Enterprise Linux 8 でのファイルシステムの作成、変更、管理

法律上の通知

概要

Red Hat Enterprise Linux は、さまざまなファイルシステムに対応します。各タイプのファイルシステムがさまざまな問題を解決し、その使用方法はアプリケーションによって異なります。主な相違点と考慮事項に関する情報を使用し、特定のアプリケーション要件に基づいて適切なファイルシステムを選択してデプロイします。 サポートされるファイルシステムには、ローカルのディスク上のファイルシステム XFS および ext4、ネットワークファイルシステムおよびクライアント/サーバーファイルシステム NFS および SMB が含まれます。ファイルシステムでは、作成、マウント、バックアップ、復元、チェック、修復などのさまざまな操作を実行できるだけでなく、クォータを使用してストレージ領域を制限することもできます。

目次

多様性を受け入れるオープンソースの強化 7

RED HAT ドキュメントへのフィードバック (英語のみ) 8

第1章 利用可能なファイルシステムの概要 9
1.1. ファイルシステムの種類 9
1.2. ローカルファイルシステム 10
1.3. XFS ファイルシステム 10
1.4. EXT4 ファイルシステム 11
1.5. XFS と EXT4 の比較 12
1.6. ローカルファイルシステムの選択 13
1.7. ネットワークファイルシステム 14
1.8. 共有ストレージファイルシステム 14
1.9. ネットワークと共有ストレージファイルシステムの選択 15
1.10. ボリューム管理ファイルシステム 16

第2章 RHEL システムロールを使用したローカルストレージの管理 17
2.1. STORAGE RHEL システムロールの概要 17
2.2. STORAGE RHEL システムロールのストレージデバイスを識別するパラメーター 18
2.3. STORAGE RHEL システムロールを使用してブロックデバイスに XFS ファイルシステムを作成する 18
2.4. STORAGE RHEL システムロールを使用してファイルシステムを永続的にマウントする 19
2.5. STORAGE RHEL システムロールを使用して論理ボリュームを管理する 20
2.6. STORAGE RHEL システムロールを使用してオンラインのブロック破棄を有効にする 21
2.7. STORAGE RHEL システムロールを使用して EXT4 ファイルシステムを作成およびマウントする 22
2.8. STORAGE RHEL システムロールを使用して EXT3 ファイルシステムを作成およびマウントする 23
2.9. STORAGE RHEL システムロールを使用して LVM 上の既存のファイルシステムのサイズを変更する 24
2.10. 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 26
2.11. STORAGE システムロールを使用して RAID ボリュームを設定する 27
2.12. STORAGE RHEL システムロールを使用して RAID を備えた LVM プールを設定する 28
2.13. STORAGE RHEL システムロールを使用して RAID LVM ボリュームのストライプサイズを設定する 29
2.14. STORAGE RHEL システムロールを使用して LVM 上の VDO ボリュームを圧縮および重複排除する 30
2.15. 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 31
2.16. STORAGE RHEL システムロールを使用してプールボリュームのサイズをパーセンテージで表す 33

第3章 NFS 共有のマウント 35
3.1. NFS ホスト名の形式 35
3.2. ファイアウォールの内側で動作するように NFSV3 クライアントを設定する手順 35
3.3. ファイアウォールの内側で動作するように NFSV4 クライアントを設定する手順 37
3.4. NFS エクスポートの検出 37
3.5. MOUNT で NFS 共有のマウント 38
3.6. クライアントで PNFS SCSI の設定 39
3.7. MOUNTSTATS でクライアントからの PNFS SCSI 操作のチェック 39
3.8. 一般的な NFS マウントオプション 40
3.9. NFS でユーザー設定の保存 41
3.10. FS-CACHE の使用 41

第4章 NFS サーバーのデプロイ 49
4.1. NFSV4 のマイナーバージョンの主な機能 49
4.2. AUTH_SYS 認証方式 50
4.3. AUTH_GSS 認証方式 51
4.4. エクスポートされたファイルシステム上のファイル権限 51
4.5. NFS サーバーに必要なサービス 51

4.6. /ETC/EXPORTS 設定ファイル 53
4.7. NFSV4 専用サーバーの設定 54
4.8. オプションの NFSV4 サポートを備えた NFSV3 サーバーの設定 56
4.9. NFS サーバーでクォータサポートを有効にする 58
4.10. NFS サーバーで RDMA 経由の NFS を有効にする 60
4.11. RED HAT IDENTITY MANAGEMENT ドメインで KERBEROS を使用した NFS サーバーを設定する 61

第5章 SMB 共有のマウント 64
5.1. 対応している SMB プロトコルのバージョン 64
5.2. UNIX 拡張機能のサポート 65
5.3. SMB 共有の手動マウント 65
5.4. システム起動時の SMB 共有の自動マウント 66
5.5. SMB 共有に対して認証するための認証情報ファイルの作成 67
5.6. マルチユーザー SMB マウントの実行 67
5.7. よく使用される SMB マウントオプション 69

第6章 永続的な命名属性の概要 71
6.1. 非永続的な命名属性のデメリット 71
6.2. ファイルシステムおよびデバイスの識別子 72
6.3. /DEV/DISK/ にある UDEV メカニズムにより管理されるデバイス名 72
6.4. DM MULTIPATH を使用した WORLD WIDE IDENTIFIER 74
6.5. UDEV デバイス命名規則の制約 75
6.6. 永続的な命名属性のリスト表示 76
6.7. 永続的な命名属性の変更 77

第7章 PARTED でのパーティション操作 78
7.1. PARTED でパーティションテーブルの表示 78
7.2. PARTED でディスクにパーティションテーブルを作成 79
7.3. PARTED でパーティションの作成 80
7.4. PARTED でパーティションの削除 82
7.5. PARTED でパーティションのサイズ変更 83

第8章 ディスクを再設定するストラテジー 85
8.1. パーティションが分割されていない空き領域の使用 85
8.2. 未使用パーティションの領域の使用 85
8.3. アクティブなパーティションの空き領域の使用 86

第9章 XFS の使用 90
9.1. XFS ファイルシステム 90
9.2. EXT4 および XFS で使用されるツールの比較 91

第10章 XFS ファイルシステムの作成 92
10.1. MKFS.XFS で XFS ファイルシステムの作成 92

第11章 XFS ファイルシステムのバックアップ 93
11.1. XFS バックアップの機能 93
11.2. XFSDUMP で XFS ファイルシステムのバックアップ 93
11.3. 関連情報 94

第12章 バックアップからの XFS ファイルシステムの復元 95
12.1. バックアップから XFS を復元する機能 95
12.2. XFSRESTORE を使用してバックアップから XFS ファイルシステムを復元 95
12.3. テープから XFS バックアップを復元するときの情報メッセージ 96
12.4. 関連情報 97

第13章 XFS ファイルシステムのサイズの拡大 98

13.1. XFS_GROWFS で XFS ファイルシステムのサイズの拡大 98

第14章 XFS エラー動作の設定 99
14.1. XFS で設定可能なエラー処理 99
14.2. 特定の、未定義の XFS エラー条件の設定ファイル 99
14.3. 特定の条件に対する XFS 動作の設定 100
14.4. 未定義の条件に対する XFS 動作の設定 100
14.5. XFS アンマウント動作の設定 101

第15章 ファイルシステムの検査と修復 102
15.1. ファイルシステムの検査が必要なシナリオ 102
15.2. FSCK の実行による潜在的な悪影響 103
15.3. XFS のエラー処理メカニズム 103
15.4. XFS_REPAIR で XFS ファイルシステムの検査 104
15.5. XFS_REPAIR で XFS ファイルシステムの修復 105
15.6. EXT2、EXT3、および EXT4 でエラー処理メカニズム 106
15.7. E2FSCK で EXT2、EXT3、または EXT4 ファイルシステムの検査 106
15.8. E2FSCK で EXT2、EXT3、または EXT4 ファイルシステムの修復 107

第16章 ファイルシステムのマウント 109
16.1. LINUX のマウントメカニズム 109
16.2. 現在マウントされているファイルシステムのリスト表示 109
16.3. MOUNT でファイルシステムのマウント 110
16.4. マウントポイントの移動 111
16.5. UMOUNT でファイルシステムのアンマウント 111
16.6. 一般的なマウントオプション 112

第17章 複数のマウントポイントでのマウント共有 114
17.1. 共有マウントのタイプ 114
17.2. プライベートマウントポイントの複製の作成 114
17.3. 共有マウントポイントの複製の作成 115
17.4. スレーブマウントポイントの複製の作成 117
17.5. マウントポイントが複製されないようにする 118

第18章 ファイルシステムの永続的なマウント 120
18.1. /ETC/FSTAB ファイル 120
18.2. /ETC/FSTAB へのファイルシステムの追加 120

第19章 オンデマンドでのファイルシステムのマウント 122
19.1. AUTOFS サービス 122
19.2. AUTOFS 設定ファイル 122
19.3. AUTOFS マウントポイントの設定 124
19.4. AUTOFS サービスを使用した NFS サーバーユーザーのホームディレクトリーの自動マウント 125
19.5. AUTOFS サイトの設定ファイルの上書き/拡張 125
19.6. LDAP で自動マウント機能マップの格納 127
19.7. SYSTEMD.AUTOMOUNT を使用して、/ETC/FSTAB を使用してオンデマンドでファイルシステムをマウントします 128
19.8. SYSTEMD.AUTOMOUNT を使用して、マウントユニットを使用してファイルシステムをオンデマンドでマウントします 129

第20章 IDM からの SSSD コンポーネントを使用した AUTOFS マップのキャッシュ 131
20.1. IDM サーバーを LDAP サーバーとして使用するように AUTOFS を手動で設定する 131
20.2. AUTOFS マップをキャッシュする SSSD の設定 132

第21章 ROOT ファイルシステムに対する読み取り専用パーミッションの設定 134

21.1. 書き込みパーミッションを保持するファイルおよびディレクトリー 134
21.2. ブート時に読み取り専用パーミッションでマウントするように ROOT ファイルシステムの設定 135

第22章 クォータを使用した XFS でのストレージ領域の使用の制限 137
22.1. ディスククォータ 137
22.2. XFS_QUOTA ツール 137
22.3. XFS でのファイルシステムクォータ管理 137
22.4. XFS のディスククォータの有効化 138
22.5. XFS 使用量の報告 138
22.6. XFS クォータ制限の変更 139
22.7. XFS のプロジェクト制限の設定 140

第23章 クォータを使用した EXT4 でのストレージ領域の使用の制限 142
23.1. クォータツールのインストール 142
23.2. ファイルシステム作成でクォータ機能の有効化 142
23.3. 既存のファイルシステムでのクォータ機能の有効化 142
23.4. クォータ強制適用の有効化 143
23.5. ユーザーごとにクォータの割り当て 144
23.6. グループごとにクォータの割り当て 145
23.7. プロジェクトごとにクォータの割り当て 146
23.8. ソフト制限の猶予期間の設定 147
23.9. ファイルシステムのクォータをオフにする 147
23.10. ディスククォータに関するレポート 148

第24章 未使用ブロックの破棄 149
要件 149
24.1. ブロック破棄操作のタイプ 149
24.2. バッチブロック破棄の実行 149
24.3. オンラインブロック破棄の有効化 150
24.4. 定期的なブロック破棄の有効化 150

第25章 STRATIS ファイルシステムの設定 152
25.1. STRATIS とは 152
25.2. STRATIS ボリュームの設定要素 152
25.3. STRATIS で使用可能なブロックデバイス 153
25.4. STRATIS のインストール 154
25.5. 暗号化されていない STRATIS プールの作成 154
25.6. 暗号化された STRATIS プールの作成 155
25.7. 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 156
25.8. STRATIS プールの NBDE へのバインド 158
25.9. STRATIS プールの TPM へのバインド 158
25.10. 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 159
25.11. 補助暗号化からの STRATIS プールのバインド解除 159
25.12. STRATIS プールの開始および停止 160
25.13. STRATIS ファイルシステムの作成 161
25.14. STRATIS ファイルシステムのマウント 162
25.15. STRATIS ファイルシステムの永続的なマウント 162
25.16. SYSTEMD サービスを使用した /ETC/FSTAB での非 ROOT STRATIS ファイルシステムの設定 163

第26章 追加のブロックデバイスでの STRATIS ボリュームの拡張 165
26.1. STRATIS ボリュームの設定要素 165
26.2. STRATIS プールへのブロックデバイスの追加 166
26.3. 関連情報 166

第27章 STRATIS ファイルシステムの監視 167

27.1. さまざまなユーティリティーが報告する STRATIS のサイズ 167
27.2. STRATIS ボリュームの情報表示 167
27.3. 関連情報 168

第28章 STRATIS ファイルシステムでのスナップショットの使用 169
28.1. STRATIS スナップショットの特徴 169
28.2. STRATIS スナップショットの作成 169
28.3. STRATIS スナップショットのコンテンツへのアクセス 170
28.4. STRATIS ファイルシステムを以前のスナップショットに戻す 170
28.5. STRATIS スナップショットの削除 171
28.6. 関連情報 171

第29章 STRATIS ファイルシステムの削除 172
29.1. STRATIS ボリュームの設定要素 172
29.2. STRATIS ファイルシステムの削除 173
29.3. STRATIS プールの削除 173
29.4. 関連情報 174

第30章 EXT3 ファイルシステムの使用 175
30.1. EXT3 ファイルシステムの機能 175
30.2. EXT3 ファイルシステムの作成 175
30.3. EXT3 ファイルシステムのマウント 176
30.4. EXT3 ファイルシステムのサイズ変更 177

第31章 EXT4 ファイルシステムの使用 179
31.1. EXT4 ファイルシステムの機能 179
31.2. EXT4 ファイルシステムの作成 179
31.3. EXT4 ファイルシステムのマウント 181
31.4. EXT4 ファイルシステムのサイズ変更 182
31.5. EXT4 および XFS で使用されるツールの比較 183

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

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 利用可能なファイルシステムの概要

利用可能な選択肢と、関連するトレードオフが多数あるため、アプリケーションに適したファイルシステムを選択することが重要になります。

次のセクションでは、Red Hat Enterprise Linux 8 にデフォルトで含まれるファイルシステムと、アプリケーションに最適なファイルシステムに関する推奨事項について説明します。

1.1. ファイルシステムの種類

Red Hat Enterprise Linux 8 は、さまざまなファイルシステム (FS) に対応します。さまざまな種類のファイルシステムがさまざまな問題を解決し、その使用はアプリケーションによって異なります。最も一般的なレベルでは、利用可能なファイルシステムを以下の主要なタイプにまとめることができます。

表1.1 ファイルシステムの種類とそのユースケース

タイプ	ファイルシステム	属性とユースケース
ディスクまたはローカルのファイルシステム	XFS	XFS は、RHEL におけるデフォルトファイルシステムです。Red Hat は、互換性や、パフォーマンスおいて稀に発生する難しい問題などの特別な理由がない限り、XFS をローカルファイルシステムとしてデプロイすることを推奨します。
	ext4	従来の ext2 および ext3 ファイルシステムから進化した ext4 には、Linux で馴染みやすいという利点があります。多くの場合、パフォーマンスでは XFS に匹敵します。ext4 ファイルシステムとファイルサイズのサポート制限は、XFS よりも小さくなっています。
ネットワーク、またはクライアント/サーバーのファイルシステム	NFS	NFS は、同じネットワークにある複数のシステムでのファイル共有に使用します。
	SMB	SMB は、Microsoft Windows システムとのファイル共有に使用します。
共有ストレージまたは共有ディスクのファイルシステム	GFS2	GFS2 は、コンピュートクラスターのメンバーに共有書き込みアクセスを提供します。可能な限り、ローカルファイルシステムの機能的経験を備えた安定性と信頼性に重点が置かれています。SAS Grid、Tibco MQ、IBM Websphere MQ、および Red Hat Active MQ は、GFS2 に問題なくデプロイされています。
ボリューム管理ファイルシステム	Stratis (テクノロジープレビュー)	Stratis は、XFS と LVM を組み合わせて構築されたボリュームマネージャーです。Stratis の目的は、Btrfs や ZFS などのボリューム管理ファイルシステムにより提供される機能をエミュレートすることです。このスタックを手動で構築することは可能ですが、Stratis は設定の複雑さを軽減し、ベストプラクティスを実装し、エラー情報を統合します。

1.2. ローカルファイルシステム

ローカルファイルシステムは、1 台のローカルサーバーで実行し、ストレージに直接接続されているファイルシステムです。

たとえば、ローカルファイルシステムは、内部 SATA ディスクまたは SAS ディスクにおける唯一の選択肢であり、ローカルドライブを備えた内蔵ハードウェア RAID コントローラーがサーバーにある場合に使用されます。ローカルファイルシステムは、SAN にエクスポートされたデバイスが共有されていない場合に、SAN が接続したストレージに最もよく使用されているファイルシステムです。

ローカルファイルシステムはすべて POSIX に準拠しており、サポートされているすべての Red Hat Enterprise Linux リリースと完全に互換性があります。POSIX 準拠のファイルシステムは、**read()**、**write()**、**seek()** など、明確に定義されたシステムコールのセットに対応します。

アプリケーションプログラマーの観点では、ローカルファイルシステム間の違いは比較的少なくなります。ユーザーの観点で最も重要な違いは、スケーラビリティーとパフォーマンスに関するものです。ファイルシステムの選択を検討する際に、ファイルシステムのサイズ、必要な固有の機能、実際のワークロードにおける実行方法を考慮してください。

利用可能なローカルファイルシステム

- XFS
- ext4

1.3. XFS ファイルシステム

XFS は、拡張性が高く、高性能で堅牢な、成熟した 64 ビットのジャーナリングファイルシステムで、1 台のホストで非常に大きなファイルおよびファイルシステムに対応します。Red Hat Enterprise Linux 8 ではデフォルトのファイルシステムになります。XFS は、元々 1990 年代の前半に SGI により開発され、極めて大規模なサーバーおよびストレージアレイで実行されてきた長い歴史があります。

XFS の機能は次のとおりです。

信頼性

- メタデータジャーナリング - システムの再起動時、およびファイルシステムの再マウント時に再生できるファイルシステム操作の記録を保持することで、システムクラッシュ後のファイルシステムの整合性を確保します。
- 広範囲に及ぶランタイムメタデータの整合性チェック
- 拡張性が高く、高速な修復ユーティリティー
- クォータジャーナリングクラッシュ後に行なわれる、時間がかかるクォータの整合性チェックが不要になります。

スケーラビリティーおよびパフォーマンス

- 対応するファイルシステムのサイズが最大 1024 TiB
- 多数の同時操作に対応する機能
- 空き領域管理のスケーラビリティーに関する B-Tree インデックス
- 高度なメタデータ先読みアルゴリズム

- ストリーミングビデオのワークロードの最適化

割り当てスキーム

- エクステント (領域) ベースの割り当て
- ストライプを認識できる割り当てポリシー
- 遅延割り当て
- 領域の事前割り当て
- 動的に割り当てられる inode

その他の機能

- Reflink ベースのファイルのコピー
- 密接に統合されたバックアップおよび復元のユーティリティー
- オンラインのデフラグ
- オンラインのファイルシステム拡張
- 包括的な診断機能
- 拡張属性 (**xattr**)。これにより、システムが、ファイルごとに、名前と値の組み合わせを追加で関連付けられるようになります。
- プロジェクトまたはディレクトリーのクォータ。ディレクトリーツリー全体にクォータ制限を適用できます。
- サブセカンド (一秒未満) のタイムスタンプ

パフォーマンスの特徴

XFS は、エンタープライズレベルのワークロードがある大規模なシステムで優れたパフォーマンスを発揮します。大規模なシステムとは、相対的に CPU 数が多く、さらには複数の HBA、および外部ディスクアレイへの接続を備えたシステムです。XFS は、マルチスレッドの並列 I/O ワークロードを備えた小規模のシステムでも適切に実行します。

XFS は、シングルスレッドで、メタデータ集約型のワークロードのパフォーマンスが比較的低くなります。たとえば、シングルスレッドで小さなファイルを多数作成し、削除するワークロードがこれに当てはまります。

1.4. EXT4 ファイルシステム

ext4 ファイルシステムは、ext ファイルシステムファミリーの第 4 世代です。これは、Red Hat Enterprise Linux 6 でデフォルトのファイルシステムです。

ext4 ドライバーは、ext2 および ext3 のファイルシステムの読み取りと書き込みが可能ですが、ext4 ファイルシステムのフォーマットは、ext2 ドライバーおよび ext3 ドライバーと互換性がありません。

ext4 には、以下のような新機能、および改善された機能が追加されました。

- 対応するファイルシステムのサイズが最大 50 TiB

- エクステントベースのメタデータ
- 遅延割り当て
- ジャーナルのチェックサム
- 大規模なストレージサポート

エクステントベースのメタデータと遅延割り当て機能は、ファイルシステムで使用されている領域を追跡する、よりコンパクトで効率的な方法を提供します。このような機能により、ファイルシステムのパフォーマンスが向上し、メタデータが使用する領域が低減します。遅延割り当てにより、ファイルシステムは、データがディスクにフラッシュされるまで、新しく書き込まれたユーザーデータの永続的な場所の選択を保留できます。これにより、より大きく、より連続した割り当てが可能になり、より優れた情報に基づいてファイルシステムが決定を下すことができるため、パフォーマンスが向上します。

ext4 で **fsck** ユーティリティーを使用するファイルシステムの修復時間は、ext2 と ext3 よりも高速です。一部のファイルシステムの修復では、最大 6 倍のパフォーマンスの向上が実証されています。

1.5. XFS と EXT4 の比較

XFS は、RHEL におけるデフォルトファイルシステムです。このセクションでは、XFS および ext4 の使用方法と機能を比較します。

メタデータエラーの動作

ext4 では、ファイルシステムがメタデータのエラーに遭遇した場合の動作を設定できます。デフォルトの動作では、操作を継続します。XFS が復旧できないメタデータエラーに遭遇すると、ファイルシステムをシャットダウンし、**EFSCORRUPTED** エラーを返します。

クォータ

ext4 では、既存のファイルシステムにファイルシステムを作成する場合にクォータを有効にできます。次に、マウントオプションを使用してクォータの適用を設定できます。
XFS クォータは再マウントできるオプションではありません。初期マウントでクォータをアクティブにする必要があります。

XFS ファイルシステムで **quotacheck** コマンドを実行すると影響しません。クォータアカウンティングを初めてオンにすると、XFS はクォータを自動的にチェックします。

ファイルシステムのサイズ変更

XFS には、ファイルシステムのサイズを縮小するユーティリティーがありません。XFS ファイルシステムのサイズのみを増やすことができます。ext4 は、ファイルシステムの拡張と縮小の両方をサポートします。

Inode 番号

ext4 ファイルシステムは、2^{32} 個を超える inode をサポートしません。
XFS は inode を動的に割り当てます。XFS ファイルシステムは、ファイルシステムに空き領域がある限り、inode からは実行できません。

特定のアプリケーションは、XFS ファイルシステムで 2^{32} 個を超える inode を適切に処理できません。このようなアプリケーションでは、戻り値 **EOVERFLOW** で 32 ビットの統計呼び出しに失敗する可能性があります。Inode 番号は、以下の条件下で 2^{32} 個を超えます。

- ファイルシステムが 256 バイトの inode を持つ 1 TiB を超える。
- ファイルシステムが 512 バイトの inode を持つ 2 TiB を超える。

inode 番号が大きくてアプリケーションが失敗した場合は、**-o inode32** オプションを使用して XFS ファイルシステムをマウントし、2^{32} 未満の inode 番号を実施します。**inode32** を使用しても、すでに 64 ビットの数値が割り当てられている inode には影響しません。

重要

特定の環境に必要な場合を除き、**inode32** オプション は**使用しないで**ください。**inode32** オプションは割り当ての動作を変更します。これにより、下層のディスクブロックに inode を割り当てるための領域がない場合に、**ENOSPC** エラーが発生する可能性があります。

1.6. ローカルファイルシステムの選択

アプリケーションの要件を満たすファイルシステムを選択するには、ファイルシステムをデプロイするターゲットシステムを理解する必要があります。以下の項目で、選択肢を確認できます。

- 大容量のサーバーがあるか
- ストレージの要件は大きいか、ローカルで低速な SATA ドライブが存在するか
- アプリケーションで期待される I/O ワークロードの種類
- スループットとレイテンシーの要件
- サーバーおよびストレージハードウェアの安定性
- ファイルとデータセットの標準的なサイズ
- システムで障害が発生した場合のダウンタイムの長さ

サーバーとストレージデバイスの両方が大きい場合は、XFS が最適です。ストレージアレイが小さくても、XFS は、平均のファイルサイズが大きい場合 (たとえば、数百メガバイト) に、非常に優れたパフォーマンスを発揮します。

既存のワークロードが ext4 で良好に機能している場合は、ext4 を引き続き使用することで、ユーザーとアプリケーションに非常に馴染みのある環境を提供できます。

ext4 ファイルシステムは、I/O 機能が制限されているシステムでパフォーマンスが向上する傾向があります。限られた帯域幅 (200MB/s 未満) と、最大約 1000 の IOPS 機能でパフォーマンスが向上します。より高い機能を備えたものであれば、XFS はより高速になる傾向があります。

XFS は、ext4 と比較して、メタデータあたりの CPU の動作を約 2 倍消費します。そのため、同時に処理できることがほとんどない、CPU にバインドされたワークロードがあると、ext4 の方が高速になります。通常、アプリケーションが 1 つの読み取り/書き込みスレッドと小さなファイルを使用する場合は ext4 の方が優れていますが、アプリケーションが複数の読み取り/書き込みスレッドと大きなファイルを使用する場合は、XFS の方が優れています。

XFS ファイルシステムを縮小することはできません。ファイルシステムを縮小できるようにする必要がある場合は、オフライン縮小に対応する ext4 を使用することを検討してください。

通常、Red Hat は、ext4 に対する特別なユースケースがない限り、XFS を使用することを推奨します。また、ターゲットサーバーとストレージシステムで特定のアプリケーションのパフォーマンスを測定して、適切なタイプのファイルシステムを選択するようにしてください。

表1.2 ローカルファイルシステムに関する推奨事項の概要

シナリオ	推奨されるファイルシステム
特別なユースケースなし	XFS
大規模サーバー	XFS
大規模なストレージデバイス	XFS
大規模なファイル	XFS
マルチスレッド I/O	XFS
シングルスレッド I/O	ext4
制限された I/O 機能 (1000 IOPS 未満)	ext4
制限された帯域幅 (200MB/s 未満)	ext4
CPU にバインドされているワークロード	ext4
オフラインの縮小への対応	ext4

1.7. ネットワークファイルシステム

クライアント/サーバーファイルシステムとも呼ばれるネットワークファイルシステムにより、クライアントシステムは、共有サーバーに保存されているファイルにアクセスできます。これにより、複数のシステムの、複数のユーザーが、ファイルやストレージリソースを共有できます。

このようなファイルシステムは、ファイルシステムのセットを 1 つ以上のクライアントにエクスポートする、1 つ以上のサーバーから構築されます。クライアントノードは、基盤となるブロックストレージにアクセスできませんが、より良いアクセス制御を可能にするプロトコルを使用してストレージと対話します。

利用可能なネットワークファイルシステム

- RHEL で最も一般的なクライアント/サーバーファイルシステムは、NFS ファイルシステムです。RHEL は、ネットワーク経由でローカルファイルシステムをエクスポートする NFS サーバーコンポーネントと、このようなファイルシステムをインポートする NFS クライアントの両方を提供します。
- RHEL には、Windows の相互運用性で一般的に使用されている Microsoft SMB ファイルサーバーに対応する CIFS クライアントも含まれています。ユーザー空間 Samba サーバーは、RHEL サーバーから Microsoft SMB サービスを使用する Windows クライアントを提供します。

1.8. 共有ストレージファイルシステム

クラスターファイルシステムとも呼ばれる共有ストレージファイルシステムにより、クラスター内の各サーバーは、ローカルストレージエリアネットワーク (SAN) を介して共有ブロックデバイスに直接アクセスできます。

ネットワークファイルシステムとの比較

クライアント/サーバーのファイルシステムと同様、共有ストレージファイルシステムは、クラスターのすべてのメンバーであるサーバーのセットで機能します。ただし、NFS とは異なり、1 台のサーバーでは、その他のメンバーにデータまたはメタデータへのアクセスを提供しません。クラスターの各メンバーが同じストレージデバイス (**共有ストレージ**) に直接アクセスし、すべてのクラスターメンバーノードが同じファイルセットにアクセスできるようになります。

同時並行性

キャッシュの一貫性は、データの一貫性と整合性を確保するためにクラスター化されたファイルシステムで重要になります。クラスター内のすべてのノードに表示される、クラスター内のすべてのファイルのバージョンが 1 つ必要です。ファイルシステムは、クラスターのメンバーが同じストレージブロックを同時に更新して、データ破損を引き起こさないようにする必要があります。共有ストレージファイルシステムは、クラスター全体のロックメカニズムを使用して、同時実行制御メカニズムとしてストレージへのアクセスを調整します。たとえば、新しいファイルを作成したり、複数のサーバーで開いているファイルに書き込む前に、サーバーにあるファイルシステムコンポーネントが正しいロックを取得する必要があります。
クラスターファイルシステムの要件は、Apache Web サーバーのような可用性の高いサービスを提供することです。クラスターのすべてのメンバーに、共有ディスクのファイルシステムに保存されているデータに関する、完全に一貫した表示が提供され、すべての更新がロックメカニズムにより正しく調整されます。

パフォーマンスの特徴

共有ディスクファイルシステムは、ロックオーバーヘッドの計算コストのため、同じシステムで実行しているローカルファイルシステムと同じように機能するとは限りません。共有ディスクのファイルシステムは、各ノードが、その他のノードと共有していない特定のファイルセットにほぼ排他的に書き込むか、ファイルセットが、ノードセット間でほぼ排他的に読み取り専用で共有されるワークロードで良好に機能します。これにより、ノード間のキャッシュの無効化が最小限に抑えられ、パフォーマンスを最大化できます。
共有ディスクファイルシステムの設定は複雑で、共有ディスクのファイルシステムで適切に動作するようにアプリケーションを調整することが困難な場合があります。

利用可能な共有ストレージファイルシステム

- Red Hat Enterprise Linux は、GFS2 ファイルシステムを提供します。GFS2 は、Red Hat Enterprise Linux High Availability Add-On および Resilient Storage Add-On と密接に統合されています。

Red Hat Enterprise Linux は、サイズが 2 ノードから 16 ノードのクラスターで GFS2 に対応します。

1.9. ネットワークと共有ストレージファイルシステムの選択

ネットワークと共有ストレージのファイルシステムのいずれかを選択する際は、以下の点を考慮してください。

- NFS ベースのネットワークファイルシステムは、NFS サーバーを提供する環境において、ごく一般的で評判が良い選択肢です。

- ネットワークファイルシステムは、Infiniband や 10 ギガビットイーサネットなど、非常に高性能なネットワークテクノロジーを使用してデプロイできます。これは、ストレージに、生の帯域幅を取得するだけのために、共有ストレージのファイルシステムを有効にすべきではないことを意味します。アクセスの速度が非常に重要な場合は、NFS を使用して、XFS などのローカルファイルシステムをエクスポートします。

- 共有ストレージのファイルシステムは、設定や維持が容易ではないため、ローカルまたはネットワークのファイルシステムのいずれかで必要な可用性を提供できない場合に限りデプロイしてください。

- クラスター環境の共有ストレージのファイルシステムは、高可用性サービスの再配置を伴う一般的なフェイルオーバーシナリオで、マウント解除およびマウントに必要な手順を省くことで、ダウンタイムを短縮できます。

Red Hat は、共有ストレージのファイルシステムに対する特別なユースケースがない限り、ネットワークのファイルシステムを使用することを推奨します。共有ストレージのファイルシステムは、主に、最小限のダウンタイムで高可用性サービスを提供する必要があり、サービスレベルの要件が厳しいデプロイメントに使用します。

1.10. ボリューム管理ファイルシステム

ボリューム管理ファイルシステムは、簡素化とスタック内の最適化の目的で、ストレージスタック全体を統合します。

利用可能なボリューム管理ファイルシステム

- Red Hat Enterprise Linux 8 は、Stratis ボリュームマネージャーがテクノロジープレビューとして提供します。Stratis は、ファイルシステム層に XFS を使用し、LVM、Device Mapper、およびその他のコンポーネントと統合します。

Stratis は、Red Hat Enterprise Linux 8.0 で初めてリリースされました。Red Hat が Btrfs を非推奨にした時に生じたギャップを埋めると考えられています。Stratis 1.0 は、ユーザーによる複雑さを隠しつつ、重要なストレージ管理操作を実行できる直感的なコマンドラインベースのボリュームマネージャーです。

- ボリュームの管理
- プールの作成
- シンストレージプール
- スナップショット
- 自動化読み取りキャッシュ

Stratis は強力な機能を提供しますが、現時点では Btrfs や ZFS といったその他の製品と比較される可能性がある機能をいつくか欠いています。たとえば、セルフ修復を含む CRC には対応していません。

第2章 RHEL システムロールを使用したローカルストレージの管理

Ansible を使用して LVM およびローカルファイルシステム (FS) を管理するには、RHEL 8 で使用可能な RHEL システムロールの 1 つである **ストレージ** ロールを使用できます。

storage ロールを使用すると、ディスク上のファイルシステム、複数のマシンにある論理ボリューム、および RHEL 7.7 以降の全バージョンでのファイルシステムの管理を自動化できます。

RHEL システムロールとその適用方法の詳細は、RHEL システムロールの概要 を参照してください。

2.1. STORAGE RHEL システムロールの概要

storage ロールは以下を管理できます。

- パーティションが分割されていないディスクのファイルシステム
- 論理ボリュームとファイルシステムを含む完全な LVM ボリュームグループ
- MD RAID ボリュームとそのファイルシステム

storage ロールを使用すると、次のタスクを実行できます。

- ファイルシステムを作成する
- ファイルシステムを削除する
- ファイルシステムをマウントする
- ファイルシステムをアンマウントする
- LVM ボリュームグループを作成する
- LVM ボリュームグループを削除する
- 論理ボリュームを作成する
- 論理ボリュームを削除する
- RAID ボリュームを作成する
- RAID ボリュームを削除する
- RAID で LVM ボリュームグループを作成する
- RAID で LVM ボリュームグループを削除する
- 暗号化された LVM ボリュームグループを作成する
- RAID で LVM 論理ボリュームを作成する

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.2. STORAGE RHEL システムロールのストレージデバイスを識別するパラメーター

storage ロールの設定は、以下の変数に記載されているファイルシステム、ボリューム、およびプールにのみ影響します。

storage_volumes

マネージドのパーティションが分割されていない全ディスク上のファイルシステムのリスト
storage_volumes には **raid** ボリュームを含めることもできます。

現在、パーティションはサポートされていません。

storage_pools

管理するプールのリスト
現在、サポートされている唯一のプールタイプは LVM です。LVM では、プールはボリュームグループ (VG) を表します。各プールの下には、ロールで管理されるボリュームのリストがあります。
LVM では、各ボリュームは、ファイルシステムを持つ論理ボリューム (LV) に対応します。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.3. STORAGE RHEL システムロールを使用してブロックデバイスに XFS ファイルシステムを作成する

Ansible Playbook の例では、**storage** ロールを適用して、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上に XFS ファイルシステムを作成します。

注記

storage ロールは、パーティションが分割されていないディスク全体または論理ボリューム (LV) でのみファイルシステムを作成できます。パーティションにファイルシステムを作成することはできません。

前提条件

- 制御ノードと管理ノードを準備している
- 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。

```
---
- hosts: managed-node-01.example.com
  roles:
```

```
  - rhel-system-roles.storage
vars:
  storage_volumes:
    - name: barefs
      type: disk
      disks:
        - sdb
      fs_type: xfs
```

- 現在、ボリューム名 (この例では **barefs**) 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。
- XFS は RHEL 8 のデフォルトファイルシステムであるため、**fs_type: xfs** 行を省略することができます。
- 論理ボリュームにファイルシステムを作成するには、エンクロージングボリュームグループを含む **disks:** 属性の下に LVM 設定を指定します。詳細は、ストレージ RHEL システムロールを使用した論理ボリュームの管理 を参照してください。
  LV デバイスへのパスを指定しないでください。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.4. STORAGE RHEL システムロールを使用してファイルシステムを永続的にマウントする

Ansible の例では、**storage** ロールを適用して、XFS ファイルシステムを即時かつ永続的にマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 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。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - この Playbook は、ファイルシステムを **/etc/fstab** ファイルに追加し、ファイルシステムを即座にマウントします。
   - **/dev/sdb** デバイス上のファイルシステム、またはマウントポイントのディレクトリーが存在しない場合は、Playbook により作成されます。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.5. STORAGE RHEL システムロールを使用して論理ボリュームを管理する

Ansible Playbook の例では、**storage** ロールを適用して、ボリュームグループに LVM 論理ボリュームを作成します。

前提条件

- 制御ノードと管理ノードを準備している
- 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。

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt/dat
```

   - **myvg** ボリュームグループは、ディスク **/dev/sda**、**/dev/sdb**、および **/dev/sdc** で構成されています。
   - **myvg** ボリュームグループがすでに存在する場合は、Playbook により論理ボリュームがボリュームグループに追加されます。
   - **myvg** ボリュームグループが存在しない場合は、Playbook により作成されます。
   - この Playbook は、**mylv** 論理ボリュームに Ext4 ファイルシステムを作成し、そのファイルシステムを **/mnt** に永続的にマウントします。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

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

Ansible Playbook の例では、**storage** ロールを適用して、オンラインのブロック破棄を有効にして XFS ファイルシステムをマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 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。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_options: discard
```

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.7. STORAGE RHEL システムロールを使用して EXT4 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して、Ext4 ファイルシステムを作成してマウントします。

前提条件

- 制御ノードと管理ノードを準備している

- 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。

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
```

   - Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを **/mnt/data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.8. STORAGE RHEL システムロールを使用して EXT3 ファイルシステムを作成およびマウントする

Ansible Playbook の例では、**storage** ロールを適用して Ext3 ファイルシステムを作成してマウントします。

前提条件

- 制御ノードと管理ノードを準備している
- 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。

```
---
- hosts: all
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - Playbook は、**/dev/sdb** ディスクにファイルシステムを作成します。
   - この Playbook は、ファイルシステムを **/mnt/data** ディレクトリーに永続的にマウントします。
   - ファイルシステムのラベルは **label-name** です。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.9. STORAGE RHEL システムロールを使用して LVM 上の既存のファイルシステムのサイズを変更する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用して、ファイルシステムを持つ LVM 論理ボリュームのサイズを変更します。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Create LVM pool over three disks
  hosts: managed-node-01.example.com
  tasks:
    - name: Resize LVM logical volume with file system
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_pools:
          - name: myvg
            disks:
              - /dev/sda
              - /dev/sdb
              - /dev/sdc
            volumes:
              - name: mylv1
                size: 10 GiB
                fs_type: ext4
                mount_point: /opt/mount1
              - name: mylv2
                size: 50 GiB
                fs_type: ext4
                mount_point: /opt/mount2
```

   この Playbook は、以下の既存のファイルシステムのサイズを変更します。

   - /**opt**/**mount1** にマウントされる **mylv1** ボリュームの Ext4 ファイルシステムは、そのサイズを 10 GiB に変更します。
   - /**opt**/**mount2** にマウントされる **mylv2** ボリュームの Ext4 ファイルシステムは、そのサイズを 50 GiB に変更します。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.10. 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

本セクションでは、Ansible Playbook の例を紹介します。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

- 制御ノードと管理ノードを準備している
- 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: 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。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.11. STORAGE システムロールを使用して RAID ボリュームを設定する

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 で特定のディスク名を使用しないでください。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Configure the storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: 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
```

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.12. STORAGE RHEL システムロールを使用して RAID を備えた LVM プールを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RAID を備えた LVM プールを RHEL に設定できます。利用可能なパラメーターを使用して Ansible Playbook をセットアップし、LVM pool with RAID を設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Configure LVM pool with RAID
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_volume
```

```
        size: "1 GiB"
        mount_point: "/mnt/app/shared"
        fs_type: xfs
        state: present
```

RAID を備えた LVM プールを作成するには、**raid_level** パラメーターを使用して RAID タイプを指定する必要があります。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.13. STORAGE RHEL システムロールを使用して RAID LVM ボリュームのストライプサイズを設定する

storage システムロールを使用すると、Red Hat Ansible Automation Platform を使用して、RHEL の RAID LVM ボリュームのストライプサイズを設定できます。利用可能なパラメーターを使用して Ansible Playbook をセットアップし、LVM pool with RAID を設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
```

```
storage_pools:
  - name: my_pool
    type: lvm
    disks: [sdh, sdi]
    volumes:
      - name: my_volume
        size: "1 GiB"
        mount_point: "/mnt/app/shared"
        fs_type: xfs
        raid_level: raid1
        raid_stripe_size: "256 KiB"
        state: present
```

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。

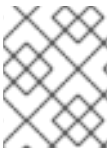
```
$ ansible-playbook ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー
- RAID の管理

2.14. STORAGE RHEL システムロールを使用して LVM 上の VDO ボリュームを圧縮および重複排除する

このサンプル Ansible Playbook は、**storage** RHEL システムロールを適用し、Virtual Data Optimizer (VDO) を使用した論理ボリューム (LVM) の圧縮と重複排除を有効にします。



注記

storage システムロールが LVM VDO を使用するため、圧縮と重複排除を使用できるのはプールごとに 1 つのボリュームのみです。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Create LVM VDO volume under volume group 'myvg'
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: mylv1
            compression: true
            deduplication: true
            vdo_pool_size: 10 GiB
            size: 30 GiB
            mount_point: /mnt/app/shared
```

この例では、**compression** プールおよび **deduplication** プールを true に設定します。これは、VDO が使用されることを指定します。以下では、このパラメーターの使用方法を説明します。

- **deduplication** は、ストレージボリュームに保存されている重複データの重複排除に使用されます。
- 圧縮は、ストレージボリュームに保存されているデータを圧縮するために使用されます。これにより、より大きなストレージ容量が得られます。
- vdo_pool_size は、ボリュームがデバイスで使用する実際のサイズを指定します。VDO ボリュームの仮想サイズは、**size** パラメーターで設定します。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

2.15. 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

- 制御ノードと管理ノードを準備している
- 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: Create and configure a volume encrypted with LUKS
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
        - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: <password>
```

また、**encryption_key**、**encryption_cipher**、**encryption_key_size**、**encryption_luks** など、他の暗号化パラメーターを Playbook ファイルに追加することもできます。

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

1. 暗号化ステータスを表示します。

```
# cryptsetup status sdb

/dev/mapper/sdb is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
```

```
key location: keyring
device: /dev/sdb
...
```

2. 作成された LUKS 暗号化ボリュームを確認します。

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards

Data segments:
  0: crypt
        offset: 33554432 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 4096 [bytes]
...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/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

2.16. STORAGE RHEL システムロールを使用してプールボリュームのサイズをパーセンテージで表す

このサンプル Ansible Playbook は、**storage** システムロールを適用して、論理マネージャーボリューム (LVM) のボリュームサイズをプールの合計サイズのパーセンテージで表現できるようにします。

前提条件

- 制御ノードと管理ノードを準備している
- 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: Express volume sizes as a percentage of the pool's total size
```

```
hosts: managed-node-01.example.com
roles:
  - rhel-system-roles.storage
vars:
  storage_pools:
    - name: myvg
      disks:
        - /dev/sdb
      volumes:
        - name: data
          size: 60%
          mount_point: /opt/mount/data
        - name: web
          size: 30%
          mount_point: /opt/mount/web
        - name: cache
          size: 10%
          mount_point: /opt/cache/mount
```

この例では、LVM ボリュームのサイズをプールサイズのパーセンテージで指定します (例: **60%**)。LVM ボリュームのサイズは、人間が判読できるファイルシステムのサイズ (例: **10g** または **50 GiB**) に占めるプールサイズのパーセンテージで指定することもできます。

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
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** ディレクトリー

第3章 NFS 共有のマウント

システム管理者は、システムにリモート NFS 共有をマウントすると、共有データにアクセスできます。

3.1. NFS ホスト名の形式

本セクションでは、NFS 共有をマウントまたはエクスポートするときにホストの指定に使用するさまざまな形式を説明します。

次の形式でホストを指定できます。

単独のマシン

次のいずれかになります。

- 完全修飾ドメイン名 (サーバーにより解決)
- ホスト名 (サーバーにより解決)
- IP アドレス

IP ネットワーク

以下のいずれかの形式が有効です。

- **a.b.c.d/z** - **a.b.c.d** がネットワークで、**z** がネットマスクのビット数になります (例: **192.168.0.0/24**)。
- **a.b.c.d/netmask** - **a.b.c.d** がネットワークで、**netmask** がネットマスクになります (例: **192.168.100.8/255.255.255.0**)。

Netgroup

@group-name 形式 - **group-name** は NIS netgroup 名です。

3.2. ファイアウォールの内側で動作するように NFSV3 クライアントを設定する手順

ファイアウォールの内側で実行するように NFSv3 クライアントを設定する手順は、ファイアウォールの内側で実行するように NFSv3 サーバーを設定する手順と似ています。

設定するマシンが NFS クライアントと NFS サーバーの両方である場合は、オプションの NFSv4 サポートを備えた NFSv3 サーバーの設定 で説明されている手順に従ってください。

以下の手順では、ファイアウォールの内側でのみ NFS クライアントマシンを設定する方法を説明します。

手順

1. クライアントがファイアウォールの内側で NFS クライアントにコールバックを実行できるようにするには、NFS クライアントで以下のコマンドを実行して **rpc-bind** サービスをファイアウォールに追加します。

```
firewall-cmd --permanent --add-service rpc-bind
```

2. **/etc/nfs.conf** ファイルで、RPC サービス **nlockmgr** が使用するポートを次のように指定します。

```
[lockd]

port=port-number
udp-port=upd-port-number
```

または、**/etc/modprobe.d/lockd.conf** ファイルで、**nlm_tcpport** および **nlm_udpport** を指定することもできます。

3. NFS クライアントで以下のコマンドを実行して、ファイアウォールで指定したポートを開きます。

```
firewall-cmd --permanent --add-port=<lockd-tcp-port>/tcp
firewall-cmd --permanent --add-port=<lockd-udp-port>/udp
```

4. 以下のように、**/etc/nfs.conf** ファイルの **[statd]** セクションを編集して、**rpc.statd** の静的ポートを追加します。

```
[statd]

port=port-number
```

5. NFS クライアントで以下のコマンドを実行して、ファイアウォールに追加したポートを開きます。

```
firewall-cmd --permanent --add-port=<statd-tcp-port>/tcp
firewall-cmd --permanent --add-port=<statd-udp-port>/udp
```

6. ファイアウォール設定を再読み込みします。

```
firewall-cmd --reload
```

7. **rpc-statd** サービスを再起動します。

```
# systemctl restart rpc-statd.service
```

または、**/etc/modprobe.d/lockd.conf** ファイルの **lockd** ポートを指定した場合は、次のコマンドを実行します。

a. **/proc/sys/fs/nfs/nlm_tcpport** と **/proc/sys/fs/nfs/nlm_udpport** の現在の値を更新します。

```
# sysctl -w fs.nfs.nlm_tcpport=<tcp-port>
# sysctl -w fs.nfs.nlm_udpport=<udp-port>
```

b. **rpc-statd** サービスを再起動します。

```
# systemctl restart rpc-statd.service
```

3.3. ファイアウォールの内側で動作するように NFSV4 クライアントを設定する手順

この手順は、クライアントが NFSv4.0 を使用している場合に限り行います。その場合は、NFSv4.0 コールバックのポートを開く必要があります。

この手順は、NFSv4.1 以降では必要ありません。これは、新しいプロトコルバージョンでは、クライアントによって開始された同じ接続でサーバーがコールバックを実行するためです。

手順

1. NFSv4.0 コールバックがファイアウォールを通過できるようにするには、以下のように **/proc/sys/fs/nfs/nfs_callback_tcpport** を設定して、サーバーがクライアントのそのポートに接続できるようにします。

```
# echo "fs.nfs.nfs_callback_tcpport = <callback-port>" >/etc/sysctl.d/90-nfs-callback-port.conf
# sysctl -p /etc/sysctl.d/90-nfs-callback-port.conf
```

2. NFS クライアントで以下のコマンドを実行して、ファイアウォールの指定のポートを開きます。

```
firewall-cmd --permanent --add-port=<callback-port>/tcp
```

3. ファイアウォール設定を再読み込みします。

```
firewall-cmd --reload
```

3.4. NFS エクスポートの検出

この手順では、特定の NFSv3 または NFSv4 サーバーがエクスポートしているファイルシステムを検出します。

手順

- NFSv3 に対応しているサーバーで、**showmount** ユーティリティーを使用します。

```
$ showmount --exports my-server

Export list for my-server
/exports/foo
/exports/bar
```

- NFSv4 に対応しているサーバーで、root ディレクトリーをマウントして確認します。

```
# mount my-server:/ /mnt/
# ls /mnt/

exports

# ls /mnt/exports/
```

```
foo
bar
```

NFSv4 と NFSv3 の両方に対応するサーバーでは、上記の方法はいずれも有効で、同じ結果となります。

関連情報

- **showmount(8)** man ページ

3.5. MOUNT で NFS 共有のマウント

mount ユーティリティーを使用して、サーバーからエクスポートされた NFS 共有をマウントします。

警告

NFS クライアントが同じ短いホスト名を使用している場合は、NFSv4 **clientid** で競合が発生し、それらが突然期限切れになることがあります。NFSv4 **clientid** が突然期限切れになる可能性を回避するには、使用しているシステムに応じて、NFS クライアントに一意のホスト名を使用するか、各コンテナーで識別子を設定する必要があります。詳細については、ナレッジベース記事 NFSv4 clientid was expired suddenly due to use same hostname on several NFS clients を参照してください。

手順

- NFS 共有をマウントするには、次のコマンドを使用します。

```
# mount -t nfs -o options host:/remote/export /local/directory
```

このコマンドは、以下のような変数を使用します。

options
: マウントオプションのコンマ区切りリスト

host
: マウントするファイルシステムをエクスポートするサーバーのホスト名、IP アドレス、または完全修飾ドメイン名。

/remote/export
: サーバーからエクスポートされるファイルシステムまたはディレクトリー、つまりマウントするディレクトリー。

/local/directory
: **/remote/export** がマウントされているクライアントの場所

関連情報

- 一般的な NFS マウントオプション

- NFS ホスト名の形式
- **mount(8)** の man ページ
- **exports(5)** man ページ

3.6. クライアントで PNFS SCSI の設定

この手順では、pNFS SCSI レイアウトをマウントするように NFS クライアントを設定します。

前提条件

- NFS サーバーは、pNFS SCSI で XFS ファイルシステムをエクスポートするように設定されています。

手順

- クライアントで、NFS バージョン 4.1 以降を使用して、エクスポートした XFS ファイルシステムをマウントします。

```
# mount -t nfs -o nfsvers=4.1 host:/remote/export /local/directory
```

NFS なしで XFS ファイルシステムを直接マウントしないでください。

3.7. MOUNTSTATS でクライアントからの PNFS SCSI 操作のチェック

この手順では、**/proc/self/mountstats** ファイルを使用して、クライアントから pNFS SCSI 操作を監視します。

手順

1. マウントごとの操作カウンターをリスト表示します。

```
# cat /proc/self/mountstats \
      | awk /scsi_lun_0/,/^$/ \
      | egrep device\|READ\|WRITE\|LAYOUT

device 192.168.122.73:/exports/scsi_lun_0 mounted on /mnt/rhel7/scsi_lun_0 with fstype
nfs4 statvers=1.1
    nfsv4:
bm0=0xfdffbfff,bm1=0x40f9be3e,bm2=0x803,acl=0x3,sessions,pnfs=LAYOUT_SCSI
            READ: 0 0 0 0 0 0 0 0
           WRITE: 0 0 0 0 0 0 0 0
       READLINK: 0 0 0 0 0 0 0 0
         READDIR: 0 0 0 0 0 0 0 0
       LAYOUTGET: 49 49 0 11172 9604 2 19448 19454
    LAYOUTCOMMIT: 28 28 0 7776 4808 0 24719 24722
    LAYOUTRETURN: 0 0 0 0 0 0 0 0
     LAYOUTSTATS: 0 0 0 0 0 0 0 0
```

2. 結果は以下のようになります。

   - **LAYOUT** 統計は、クライアントとサーバーが pNFS SCSI 操作を使用する要求を示します。

- **READ** および **WRITE** の統計は、クライアントとサーバーが NFS 操作にフォールバックする要求を示します。

3.8. 一般的な NFS マウントオプション

以下は、NFS 共有をマウントするときに一般的に使用されるオプションです。これらのオプションは、手動 **mount** コマンド、**/etc/fstab** 設定、および **autofs** で使用できます。

lookupcache=mode

任意のマウントポイントに対して、カーネルがディレクトリーエントリーのキャッシュを管理する方法を指定します。**mode** の有効な引数は、**all**、**none**、または **positive** です。

nfsvers=version

使用する NFS プロトコルのバージョンを指定します。**version** は、**3**、**4**、**4.0**、**4.1**、または **4.2** になります。これは、複数の NFS サーバーを実行しているホストや、より低いバージョンでのマウントの再試行を無効にするのに役立ちます。バージョンを指定しないと、NFS により、カーネルと **mount** ユーティリティーで対応している最新バージョンが使用されます。
vers オプションは **nfsvers** と同じで、互換性のためにこのリリースに含まれています。

noacl

ACL の処理をすべてオフにします。古いバージョンの Red Hat Enterprise Linux、Red Hat Linux、または Solaris と連動させる場合に必要となることがあります。こうした古いシステムには、最新の ACL テクノロジーに対する互換性がないためです。

nolock

ファイルのロック機能を無効にします。この設定は、非常に古いバージョンの NFS サーバーに接続する場合に必要となる場合があります。

noexec

マウントしたファイルシステムでバイナリーが実行されないようにします。互換性のないバイナリーを含む、Linux 以外のファイルシステムをマウントしている場合に便利です。

nosuid

set-user-identifier ビットおよび **set-group-identifier** ビットを無効にします。これにより、リモートユーザーは、**setuid** プログラムを実行してより高い権限を取得できなくなります。

port=num

NFS サーバーポートの数値を指定します。**num** が **0** (デフォルト値) の場合、**mount** は、使用するポート番号を、リモートホストの **rpcbind** サービスに問い合わせます。リモートホストの NFS サービスがその **rpcbind** サービスに登録されていない場合は、代わりに TCP 2049 の標準 NFS ポート番号が使用されます。

rsize=num and **wsize=num**

このオプションは、1 回の NFS 読み取り操作または書き込み操作で転送される最大バイト数を設定します。
rsize と **wsize** には、固定のデフォルト値がありません。デフォルトでは、NFS はサーバーとクライアントの両方がサポートしている最大の値を使用します。Red Hat Enterprise Linux 8 では、クライアントとサーバーの最大値は 1,048,576 バイトです。詳細は、NFS マウントを使用した場合の rsize と wsize のデフォルト値と最大値 参照してください。KBase の記事。

sec=flavors

マウントされたエクスポート上のファイルにアクセスするために使用するセキュリティーフレーバーです。**flavors** の値は、複数のセキュリティーフレーバーのコロンで区切られたリストです。
デフォルトでは、クライアントは、クライアントとサーバーの両方をサポートするセキュリティーフレーバーの検索を試みます。サーバーが選択したフレーバーのいずれかに対応していない場合、マウント操作は失敗します。

利用可能なフレーバー:

- **sec=sys** は、ローカルの UNIX UID および GID を使用します。 **AUTH_SYS** を使用して NFS 操作を認証します。
- **sec=krb5** は、ユーザー認証に、ローカルの UNIX の UID と GID ではなく、Kerberos V5 を使用します。
- **sec=krb5i** は、ユーザー認証に Kerberos V5 を使用し、データの改ざんを防ぐ安全なチェックサムを使用して、NFS 操作の整合性チェックを行います。
- **sec=krb5p** は、ユーザー認証に Kerberos V5 を使用し、整合性チェックを実行し、トラフィックの傍受を防ぐため NFS トラフィックの暗号化を行います。これが最も安全な設定になりますが、パフォーマンスのオーバーヘッドも最も高くなります。

tcp

NFS マウントが TCP プロトコルを使用するよう指示します。

関連情報

- **mount(8)** の man ページ
- **nfs(5)** man ページ

3.9. NFS でユーザー設定の保存

NFS ホームディレクトリーを使用するシステムで GNOME を使用する場合は、**dconf** データベースの **keyfile** バックエンドを設定する必要があります。そうしないと、**dconf** が正常に機能しない可能性があります。この設定では、**dconf** は設定を **~/.config/dconf-keyfile/user** ファイルに保存します。

手順

1. システムに **glib2-fam** パッケージがインストールされていることを確認します。

```
# yum install glib2-fam
```

このパッケージがインストールされていないと、リモートマシンで変更した設定に関する通知が適切に表示されません。

2. すべてのクライアントで **/etc/dconf/profile/user** ファイルを作成または編集します。
3. **/etc/dconf/profile/user** ファイルの先頭に、次の行を追加します。

```
service-db:keyfile/user
```

4. ユーザーはログアウトしてから再度ログインする必要があります。
dconf は **keyfile** バックエンドをポーリングして更新が行われたかどうかを判断するため、設定がすぐに更新されない可能性があります。

3.10. FS-CACHE の使用

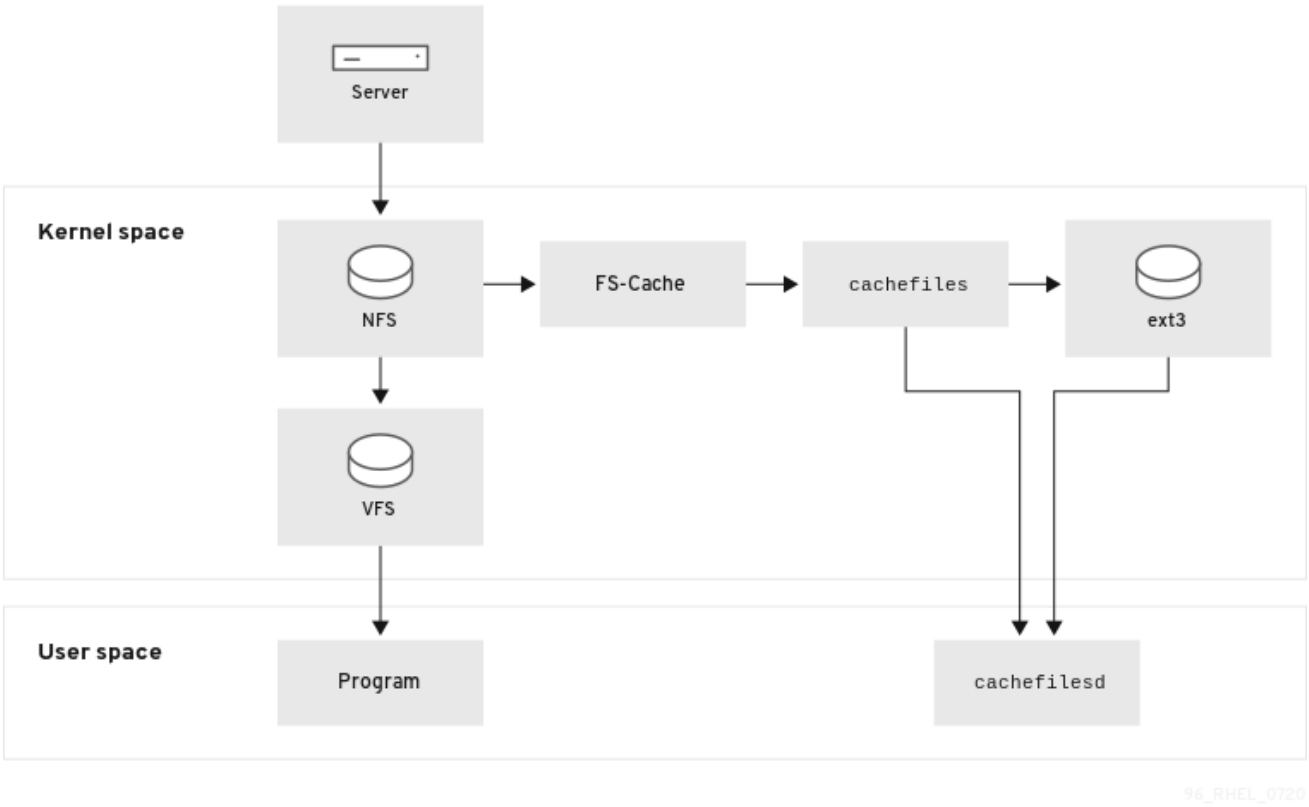
FS-Cache は、ファイルシステムがネットワーク経由で取得したデータをローカルディスクにキャッシュするために使用できる永続的なローカルキャッシュです。これは、ネットワーク経由でマウントされたファイルシステムからデータにアクセスするユーザーのネットワークトラフィックを最小限に抑え

ます (例: NFS)。

3.10.1. FS-Cache の概要

以下の図は、FS-Cache の仕組みの概要を示しています。

図3.1 FS-Cache の概要



FS-Cache は、システムのユーザーおよび管理者が可能な限り透過的になるように設計されています。Solaris では **cachefs** とは異なり、サーバー上のファイルシステムは、オーバーマウントしたファイルシステムを作成せずに、クライアントのローカルキャッシュと直接対話できます。NFS では、マウントオプションにより、FS-cache が有効になっている NFS 共有をマウントするようにクライアントに指示します。マウントポイントにより、**fscache** と **cachefiles** の 2 つのカーネルモジュールの自動アップロードが実行します。**cachefilesd** デーモンは、カーネルモジュールと通信してキャッシュを実装します。

FS-Cache はネットワーク上で機能するファイルシステムの基本操作を変更せず、単にデータをキャッシュできる永続的な場所でファイルシステムを提供するだけです。たとえば、クライアントは FS-Cache が有効になっているかどうかに関わらず、NFS 共有をマウントできます。さらに、キャッシュされた NFS は、ファイルが部分的にキャッシュされ、事前完全に読み込む必要がないため、ファイル (個別または一括) に収まらないファイルを処理できます。また、FS-Cache は、クライアントファイルシステムドライバーからキャッシュで発生するすべての I/O エラーも非表示にします。

キャッシングサービスを提供するには、**キャッシュバックエンド** が必要です。キャッシュバックエンドは、**cachefiles** であるキャッシングサービスを提供するように設定されたストレージドライバーです。この場合、FS-Cache には、キャッシュバックエンドとして **bmap** および拡張属性をサポートするマウントされたブロックベースのファイルシステム (**ext3** など) が必要です。

FS-Cache のキャッシュバックエンドで必要とされる機能に対応するファイルシステムには、以下のファイルシステムの Red Hat Enterprise Linux 8 実装が含まれます。

- ext3 (拡張属性が有効)

- ext4
- XFS

FS-Cache は、ネットワークを介するかどうかに関係なく、ファイルシステムを任意にキャッシュすることはできません。共有ファイルシステムのドライバーを変更して、FS-Cache、データストレージ/検索、メタデータのセットアップと検証を操作できるようにする必要があります。FS-Cache では、永続性に対応するためにキャッシュされたファイルシステムの **インデックスキー** と **一貫性データ** が必要になります。インデックスキーはファイルシステムオブジェクトをキャッシュオブジェクトに一致させ、一貫性データを使用してキャッシュオブジェクトが有効のままかどうかを判断します。

注記

Red Hat Enterprise Linux 8 では、**cachefilesd** パッケージはデフォルトでインストールされていないため、手動でインストールする必要があります。

3.10.2. パフォーマンスに関する保証

FS-Cache は、パフォーマンスの向上を **保証しません**。キャッシュを使用するとパフォーマンスが低下します。たとえば、キャッシュされた NFS 共有では、ネットワーク間のルックアップにディスクアクセスが追加されます。FS-Cache は可能な限り非同期となりますが、非同期にできない同期パス (**read** 操作など) があります。

たとえば、FS-Cache を使用して、通常は負荷のない GigE ネットワークを介して 2 台のコンピューター間の NFS 共有をキャッシュしても、ファイルアクセスのパフォーマンスは向上しない可能性があります。代わりに、NFS 要求はローカルディスクからではなく、サーバーメモリーより早く満たされます。

したがって、FS-Cache の使用は、さまざまな要因における **妥協** です。たとえば、NFS トラフィックのキャッシュに FS-Cache を使用すると、クライアントは多少遅くなりますが、ネットワークの帯域幅を消費せずにローカルに読み取り要求を満たすことでネットワークおよびサーバーの読み込み負荷が大幅に削減されます。

3.10.3. NFS でのキャッシュの使用

明示的に指示されない限り、NFS はキャッシュを使用しません。ここでは、FS-Cache を使用して NFS マウントを設定する方法を説明します。

NFS インデックスは NFS ファイルハンドルを使用してコンテンツをキャッシュします。**ファイル名ではなく**、ハードリンクされたファイルはキャッシュを正しく共有します。

NFS バージョン 3、4.0、4.1、および 4.2 はキャッシュに対応します。ただし、各バージョンはキャッシュに異なるブランチを使用します。

前提条件

- **cachefilesd** パッケージがインストールされ、実行している。これを実行していることを確認するには、次のコマンドを使用します。

```
# systemctl start cachefilesd
# systemctl status cachefilesd
```

ステータスは **active (running)** である必要があります。

手順

- 以下のオプションで NFS 共有をマウントします。

```
# mount nfs-share:/ /mount/point -o fsc
```

ファイルがダイレクト I/O や書き込みのために開いていない限り、**/mount/point** の下にあるファイルへのアクセスはすべてキャッシュを経由します。

3.10.4. キャッシュの設定

現在、Red Hat Enterprise Linux 8 は **cachefiles** キャッシュバックエンドのみを提供します。**cachefilesd** デーモンは **cachefiles** を開始し、管理します。**/etc/cachefilesd.conf** ファイルは、**cachefiles** によるキャッシュサービスの提供方法を制御します。

キャッシュバックエンドは、キャッシュをホストしているパーティション上の一定の空き領域を維持することで動作します。空き領域を使用する他の要素に応じてキャッシュを増大および縮小し、root ファイルシステム (ラップトップなど) で安全に使用できるようにします。FS-Cache ではこの動作でデフォルト値を設定します。これは、**キャッシュカリング制限** で設定できます。キャッシュカリング制限の設定方法は、キャッシュカリング制限の設定 を参照してください。

この手順では、キャッシュを設定する方法を説明します。

前提条件

- **cachefilesd** パッケージがインストールされ、サービスが正常に起動しました。サービスが実行中であることを確認するには、次のコマンドを使用します。

```
# systemctl start cachefilesd
# systemctl status cachefilesd
```

ステータスは **active (running)** である必要があります。

手順

1. キャッシュとして使用するディレクトリーをキャッシュバックエンドで設定するには、次のパラメーターを使用します。

```
$ dir /path/to/cache
```

2. 一般的に、キャッシュバックエンドディレクトリーは、以下のように **/etc/cachefilesd.conf** 内に **/var/cache/fscache** として設定されます。

```
$ dir /var/cache/fscache
```

3. キャッシュバックエンドのディレクトリーを変更する場合、selinux コンテキストは **/var/cache/fscache** と同じである必要があります。

```
# semanage fcontext -a -e /var/cache/fscache /path/to/cache
# restorecon -Rv /path/to/cache
```

4. キャッシュを設定する際に、**/path/to/cache** をディレクトリー名に置き換えます。

5. selinux コンテキストを設定するコマンドが機能しない場合は、以下のコマンドを使用します。

```
# semanage permissive -a cachefilesd_t
# semanage permissive -a cachefiles_kernel_t
```

FS-Cache は、**/path/to/cache** をホストするファイルシステムにキャッシュを保存します。ラップトップでは、root ファイルシステム (/) をホストのファイルシステムとして使用することが推奨されますが、デスクトップマシンの場合は、キャッシュ専用のディスクパーティションをマウントするより慎重に行ってください。

6. ホストのファイルシステムは、ユーザー定義の拡張属性をサポートする必要があります。FS-Cache はこれらの属性を使用して、一貫性維持情報を保存します。ext3 ファイルシステムを備えたデバイスでユーザー定義の拡張属性を有効にするには、次のように入力します。

```
# tune2fs -o user_xattr /dev/device
```

7. 代わりに、マウント時にファイルシステムの拡張属性を有効にするには、次のコマンドを使用します。

```
# mount /dev/device /path/to/cache -o user_xattr
```

8. 設定ファイルを置いたら、**cachefilesd** サービスを起動します。

```
# systemctl start cachefilesd
```

9. 起動時に **cachefilesd** が起動するように設定するには、root で次のコマンドを実行します。

```
# systemctl enable cachefilesd
```

3.10.5. NFS キャッシュ共有の設定

NFS キャッシュの共有には潜在的な問題がいくつかあります。キャッシュは永続的であるため、キャッシュ内のデータブロックは 4 つのキーのシーケンスでインデックス化されます。

- レベル 1: サーバーの詳細
- レベル 2: 一部のマウントオプション、セキュリティータイプ、FSID、識別子
- レベル 3: ファイルハンドル
- レベル 4: ファイル内のページ番号

スーパーブロック間の整合性の管理に関する問題を回避するには、データのキャッシュを必要とする NFS のすべてのスーパーブロックに、固有のレベル 2 キーを設定します。通常、同じソースボリュームとオプションを持つ 2 つの NFS マウントはスーパーブロックを共有しているため、そのボリューム内に異なるディレクトリーをマウントする場合でもキャッシュを共有することになります。

以下は、異なるオプションでキャッシュ共有を設定する方法の例になります。

手順

1. 次のコマンドで NFS 共有をマウントします。

```
mount home0:/disk0/fred /home/fred -o fsc
mount home0:/disk0/jim /home/jim -o fsc
```

/home/fred および **/home/jim** には同じオプションがあるため、スーパーブロックを共有する可能性が高くなります。特に NFS サーバー上の同じボリュームやパーティションから作成されている場合は共有する可能性が高くなります (**home0**)。

2. スーパーブロックを共有しないようにするには、**mount** コマンドに以下のオプションを付けて実行します。

```
mount home0:/disk0/fred /home/fred -o fsc,rsize=8192
mount home0:/disk0/jim /home/jim -o fsc,rsize=65536
```

この場合、**/home/fred** と **/home/jim** は、レベル 2 キーの異なるネットワークアクセスパラメーターを持つため、スーパーブロックを共有しません。

3. 2 つのサブツリー (**/home/fred1** と **/home/fred2**) のコンテンツを **2 回** キャッシュしてスーパーブロックを共有しないようにするには、次のコマンドを使用します。

```
mount home0:/disk0/fred /home/fred1 -o fsc,rsize=8192
mount home0:/disk0/fred /home/fred2 -o fsc,rsize=65536
```

4. スーパーブロックの共有を回避するもう 1 つの方法は、**nosharecache** パラメーターで明示的に共有を回避することです。同じ例を使用します。

```
mount home0:/disk0/fred /home/fred -o nosharecache,fsc
mount home0:/disk0/jim /home/jim -o nosharecache,fsc
```

ただし、この場合は、レベル 2 キーの **home0:/disk0/fred** および **home0:/disk0/jim** を区別することができないため、使用できるスーパーブロックは 1 つだけとなります。

5. スーパーブロックにアドレスを指定するには、**fsc=unique-identifier** マウントオプションを使用して、少なくとも 1 つのマウントに **一意の識別子** を設定します。次に例を示します。

```
mount home0:/disk0/fred /home/fred -o nosharecache,fsc
mount home0:/disk0/jim /home/jim -o nosharecache,fsc=jim
```

/home/jim のキャッシュで使用されるレベル 2 キーに固有識別子の **jim** が追加されます。

重要

ユーザーは、異なる通信またはプロトコルパラメーターを持つスーパーブロック間でキャッシュを共有することはできません。たとえば、NFSv4.0 と NFSv3 の間、NFSv4.1 と NFSv4.2 間で共有することはできません。これは、強制されるスーパーブロックが異なるためです。また、読み込みサイズ (rsize) などのパラメーターを設定すると、キャッシュの共有が回避されます。これは、別のスーパーブロックを強制するためです。

3.10.6. NFS でのキャッシュの制限

NFS にはキャッシュの制限がいくつかあります。

- ダイレクト I/O で共有ファイルシステムからファイルを開くと、自動的にキャッシュが回避されます。これは、この種のアクセスがサーバーに直接行なわれる必要があるためです。
- ダイレクト I/O または書き込みのいずれかで共有ファイルシステムからファイルを開くと、キャッシュされたファイルのコピーがフラッシュされます。ダイレクト I/O や書き込みのためにファイルが開かれなくなるまで、FS-Cache はファイルを再キャッシュしません。

- さらに、FS-Cache の今回のリリースでは、通常の NFS ファイルのみをキャッシュします。FS-Cache はディレクトリー、シンボリックリンク、デバイスファイル、FIFO、ソケットを**キャッシュしません**。

3.10.7. キャッシュカリング制限の設定

cachefilesd デーモンは、共有ファイルシステムからディスクの空き領域にリモートデータをキャッシュすることで機能します。これにより、利用可能な空き領域がすべて消費される可能性があり、ディスクがルートパーティションも格納している場合は問題になる可能性があります。これを制御するために、**cachefilesd** は、最近のアクセスが少ないオブジェクトなどの古いオブジェクトをキャッシュから破棄することで、一定量の空き領域を維持しようとします。この動作は **キャッシュカリング** と呼ばれます。

キャッシュカリングは、基盤となるファイルシステムで使用可能なブロックのパーセンテージとファイルのパーセンテージに基づいて行われます。**/etc/cachefilesd.conf** には、6 つの制限を制御する設定が存在します。

brun N% (ブロックのパーセンテージ)、frun N% (ファイルのパーセンテージ)

キャッシュの空き領域と利用可能なファイルの数がこれらの制限を上回ると、カリングはオフになります。

bcull N% (ブロックのパーセンテージ)、fcull N% (ファイルのパーセンテージ)

キャッシュの空き領域と利用可能なファイルの数がこれらの制限のいずれかを下回ると、カリング動作が開始します。

bstop N% (ブロックのパーセンテージ)、fstop N% (ファイルのパーセンテージ)

キャッシュ内の使用可能な領域または使用可能なファイルの数がこの制限のいずれかを下回ると、カリングによってこれらの制限を超える状態になるまで、ディスク領域またはファイルのそれ以上の割り当ては許可されません。

各設定の **N** のデフォルト値は以下の通りです。

- **brun**/**frun** - 10%
- **bcull**/**fcull** - 7%
- **bstop**/**fstop** - 3%

この設定を行う場合は、以下の条件を満たす必要があります。

- 0 ≤ **bstop** < **bcull** < **brun** < 100
- 0 ≤ **fstop** < **fcull** < **frun** < 100

これは、空き領域と利用可能なファイルの割合であり、100 から、**df** プログラムで表示される割合を引いたものではありません。

重要

カリングは、b**xxx** と f**xxx** のペアを同時に依存します。ユーザーが個別に処理することはできません。

3.10.8. fscache カーネルモジュールからの統計情報の取得

FS-Cache は一般的な統計情報も追跡します。以下の手順では、この情報を取得する方法を説明します。

手順

1. FS-Cache に関する統計情報を表示するには、次のコマンドを使用します。

```
# cat /proc/fs/fscache/stats
```

FS-Cache 統計には、デシジョンポイントとオブジェクトカウンターに関する情報が含まれます。詳細は、以下のカーネルドキュメントを参照してください。

/usr/share/doc/kernel-doc-4.18.0/Documentation/filesystems/caching/fscache.txt

3.10.9. FS-Cache の参考資料

本セクションでは、FS-Cache の参考情報を詳細します。

1. **achefilesd** とその設定方法の詳細については、**man queuefilesd** および **man queuefilesd.conf** を参照してください。その他にも、以下のカーネルドキュメントを参照してください。
   - **/usr/share/doc/cachefilesd/README**
   - **/usr/share/man/man5/cachefilesd.conf.5.gz**
   - **/usr/share/man/man8/cachefilesd.8.gz**
2. 設計上の制約、利用可能な統計、機能など、FS-Cache に関する一般的な情報は、以下のカーネルドキュメントを参照してください。
   /usr/share/doc/kernel-doc-4.18.0/Documentation/filesystems/caching/fscache.txt

第4章 NFS サーバーのデプロイ

ネットワークファイルシステム (NFS) プロトコルを使用すると、リモートユーザーはネットワーク経由で共有ディレクトリーをマウントし、ローカルにマウントされているのと同じように使用できます。また、リソースを、ネットワークの集中化サーバーに統合できるようになります。

4.1. NFSV4 のマイナーバージョンの主な機能

NFSv4 の各マイナーバージョンでは、パフォーマンスとセキュリティーの向上を目的とした機能強化が導入されています。これらの改善点を利用して NFSv4 の可能性を最大限に活用し、ネットワーク全体で効率的かつ信頼性の高いファイル共有を実現します。

NFSv4.2 の主な機能

サーバー側コピー

サーバー側コピーは、ネットワーク経由でデータを転送せずにサーバー上のファイルをコピーする NFS サーバーの機能です。

スパースファイル

ファイルに 1 つ以上の空きスペース、つまりギャップ (ゼロのみで設定される未割り当てまたは初期化されていないデータブロック) を含めることができます。これにより、アプリケーションはスパースファイル内のホールの位置をマップできるようになります。

領域の予約

クライアントは、データを書き込む前にストレージサーバー上のスペースを予約または割り当てることができます。これにより、サーバーの容量不足が防止されます。

ラベル付き NFS

データアクセス権を強制し、NFS ファイルシステム上の個々のファイルに対して、クライアントとサーバーとの間の SELinux ラベルを有効にします。

レイアウトの機能強化

Parallel NFS (pNFS) サーバーがより優れたパフォーマンス統計を収集できるようにする機能を提供します。

NFSv4.1 の主な機能

pNFS のクライアント側サポート

クラスター化されたサーバーへの高速 I/O のサポートにより、複数のマシンにデータを保存し、データに直接アクセスし、メタデータの更新を同期できるようになります。

Sessions

セッションは、クライアントに属する接続に対するサーバーの状態を維持します。これらのセッションは、各リモートプロシージャコール (RPC) 操作の接続の確立と終了に関連するオーバーヘッドを削減することで、パフォーマンスと効率を向上させます。

NFSv4.0 の主な機能

RPC とセキュリティー

RPCSEC_GSS フレームワークは RPC セキュリティーを強化します。NFSv4 プロトコルは、インバンドセキュリティーネゴシエーションのための新しい操作を導入します。これにより、クライアントはファイルシステムリソースに安全にアクセスするためのサーバーポリシーを照会できるようになります。

手順と運用構造

NFS 4.0 では、**COMPOUND** プロシージャが導入され、クライアントが複数の操作を 1 つの要求にマージして RPC を削減できるようになりました。

ファイルシステムモデル

NFS 4.0 は階層型ファイルシステムモデルを保持し、ファイルをバイトストリームとして扱い、国際化のために名前を UTF-8 でエンコードします。

- **ファイルハンドルの種類**
  揮発性ファイルハンドルを使用すると、サーバーはファイルシステムの変更に適応でき、永続的なファイルハンドルを必要とせずにクライアントが必要に応じて適応できるようになります。

- **属性タイプ**
  ファイル属性構造には、必須属性、推奨属性、および名前付き属性が含まれており、それぞれが異なる目的を果たします。NFSv3 から派生した必須属性はファイルタイプを区別するために不可欠ですが、ACL などの推奨属性は強化されたアクセス制御を提供します。

- **マルチサーバー名前空間**
  名前空間は複数のサーバーにまたがり、属性に基づいてファイルシステム転送を簡素化し、参照、冗長性、シームレスなサーバー移行をサポートします。

OPEN および CLOSE 操作

これらの操作により、ファイルの検索、作成、セマンティック共有を 1 カ所で組み合わせることができ、ファイルアクセス管理をより効率的に行うことができます。

ファイルロック

ファイルロックはプロトコルの一部であるため、RPC コールバックは不要になります。ファイルロックの状態は、リースベースのモデルに基づいてサーバーによって管理されます。リースの更新に失敗すると、サーバーによって状態が解放される可能性があります。

クライアントのキャッシュと移譲

キャッシュは以前のバージョンと似ており、属性とディレクトリーのキャッシュのタイムアウトはクライアントによって決定されます。NFS 4.0 の移譲により、サーバーはクライアントに特定の責任を割り当てることができるようになり、特定のファイル共有セマンティクスが保証され、サーバーとの直接のやり取りなしでローカルファイル操作が可能になります。

4.2. AUTH_SYS 認証方式

AUTH_SYS メソッド (**AUTH_UNIX** とも呼ばれます) は、クライアント認証メカニズムです。**AUTH_SYS** を使用すると、クライアントはファイルにアクセスするときにユーザーのアイデンティティーと権限を確認するために、ユーザーのユーザー ID (UID) とグループ ID (GID) をサーバーに送信します。クライアントが提供する情報に依存するため、誤って設定された場合に不正アクセスを受ける可能性があり、安全性が低いと考えられています。

マッピングメカニズムにより、UID と GID の割り当てがシステム間で異なる場合でも、NFS クライアントがサーバー上の適切な権限でファイルにアクセスできるようになります。UID と GID は、次のメカニズムによって NFS クライアントとサーバーの間でマッピングされます。

直接マッピング

UID と GID は、NFS サーバーとクライアントによってローカルシステムとリモートシステム間で直接マッピングされます。これには、NFS ファイル共有に参加しているすべてのシステム間で一貫した UID と GID の割り当てが必要です。たとえば、クライアント上の UID 1000 を持つユーザーは、サーバー上の UID 1000 を持つユーザーがアクセスできる共有上のファイルにのみアクセスできます。

NFS 環境での ID 管理を簡素化するために、管理者は多くの場合、LDAP やネットワーク情報サービス (NIS) などの集中型サービスに依存して、複数のシステムにわたる UID と GID のマッピングを管理します。

ユーザーとグループ ID のマッピング

NFS サーバーおよびクライアントは、**idmapd** サービスを使用して、異なるシステム間で UID と GID を変換し、一貫した識別と権限の割り当てを実現できます。

4.3. AUTH_GSS 認証方式

Kerberos は、安全でないネットワーク上でクライアントとサーバーの安全な認証を可能にするネットワーク認証プロトコルです。対称キー暗号化を使用し、ユーザーとサービスを認証するには信頼できるキー配布センター (KDC) が必要です。

AUTH_SYS とは異なり、**RPCSEC_GSS** Kerberos メカニズムでは、サーバーはどのユーザーがファイルにアクセスしているかを正しく表すためにクライアントに依存しません。代わりに、暗号を使用してサーバーにユーザーを認証し、悪意のあるクライアントがそのユーザーの Kerberos 認証情報を持たずにユーザーになりすますことがないようにします。

/etc/exports ファイルでは、**sec** オプションは共有が提供する Kerberos セキュリティーの 1 つまたは複数の方法を定義し、クライアントはこれらの方法のいずれかを使用して共有をマウントできます。**sec** オプションは次の値をサポートします。

- **sys** : 暗号化保護なし (デフォルト)
- **krb5** - 認証のみ
- **krb5i** : 認証と整合性の保護
- **krb5p** : 認証、整合性チェック、トラフィック暗号化

メソッドが提供する暗号化機能が多ければ多いほど、パフォーマンスは低下することに注意してください。

4.4. エクスポートされたファイルシステム上のファイル権限

エクスポートされたファイルシステムのファイル権限によって、NFS 経由でファイルとディレクトリーにアクセスするクライアントのアクセス権が決まります。

NFS ファイルシステムがリモートホストによってマウントされると、各共有ファイルに対する保護はファイルシステムのアクセス許可のみになります。同じユーザー ID (UID) 値を共有する 2 人のユーザーが、異なるクライアントシステムに同じ NFS ファイルシステムをマウントすると、お互いのファイルを変更できます。

NFS は、クライアント上の **root** ユーザーをサーバー上の **root** ユーザーと同等として扱います。ただし、デフォルトでは、NFS サーバーは NFS 共有にアクセスするときに **root** を **nobody** アカウントにマップします。**root_squash** オプションはこの動作を制御します。

関連情報

- **exports(5)** man ページ

4.5. NFS サーバーに必要なサービス

Red Hat Enterprise Linux (RHEL) は、カーネルモジュールとユーザー空間プロセスの組み合わせを使用して NFS ファイル共有を提供します。

表4.1 NFS サーバーに必要なサービス

サービス名	NFS バージョン	説明
nfsd	3, 4	共有 NFS ファイルシステムに対する要求を処理する NFS カーネルモジュール。
rpcbind	3	このプロセスは、ローカルのリモートプロシージャコール (RPC) サービスからのポート予約を受け入れ、それらを使用可能にしたりアドバタイズしたりして、対応するリモート RPC サービスがそれらにアクセスできるようにします。**rpcbind** サービスは要求に応答し、指定された RPC サービスへの接続を設定します。
rpc.mountd	3, 4	このサービスは NFSv3 クライアントからの **MOUNT** 要求を処理し、NFSv4 サーバーはこのサービスの内部機能を使用します。 要求された NFS 共有が現在 NFS サーバーによってエクスポートされており、クライアントがアクセスを許可されているかどうかを確認します。
rpc.nfsd	3, 4	このプロセスは、サーバーが定義する明示的な NFS バージョンとプロトコルをアドバタイズします。カーネルと連携して、NFS クライアントが接続するたびにサーバースレッドを提供するなど、NFS クライアントの動的な要求を満たします。 **nfs-server** サービスがこのプロセスを開始します。
lockd	3	このカーネルモジュールは、クライアントがサーバー上のファイルをロックできるようにする Network Lock Manager (NLM) プロトコルを実装します。RHEL は、NFS サーバーの実行時にモジュールを自動的にロードします。
rpc.rquotad	3, 4	このサービスは、リモートユーザーのユーザークォータ情報を提供します。
rpc.idmapd	4	このプロセスは、NFSv4 名 (`user@domain` 形式の文字列) とローカルユーザーおよびグループ ID をマッピングする NFSv4 クライアントおよびサーバーアップコールを提供します。
gssproxy	3, 4	このサービスは、**rpc.nfsd** に代わって **krb5** 認証を処理します。
nfsdcld	4	このサービスは、ネットワーク分割とサーバーの再起動中に他のクライアントが競合するロックを取得した場合に、サーバーがロックの再利用を許可しないようにする NFSv4 クライアント追跡デーモンを提供します。

サービス名	NFS バージョン	説明
rpc.statd	3	このサービスは、ローカルホストが再起動したときに他の NFSv3 クライアントに通知し、リモート NFSv3 ホストが再起動したときにカーネルに通知します。

関連情報

- **rpcbind (8)**、**rpc.mountd (8)**、**rpc.nfsd (8)**、**rpc.statd (8)**、**rpc.rquotad (8)**、**rpc.idmapd (8)**、**nfsdcld (8)** man ページ

4.6. /ETC/EXPORTS 設定ファイル

/etc/exports ファイルは、サーバーがエクスポートするディレクトリーを制御します。各行には、エクスポートポイント、ディレクトリーのマウントが許可されているクライアントの空白区切りのリスト、および各クライアントのオプションが含まれます。

```
<directory> <host_or_network_1>(<options_1>) <host_or_network_n>(<options_n>)...
```

以下は **/etc/exports** エントリーの個々の部分です。

<export>
エクスポートされるディレクトリー。

<host_or_network>
エクスポートが共有されるホストまたはネットワーク。たとえば、ホスト名、IP アドレス、または IP ネットワークを指定できます。

<options>
ホストまたはネットワークのオプション。

クライアントとオプションの間にスペースを追加すると、動作が変わります。たとえば、次の行は同じ意味を持ちません。

```
/projects client.example.com(rw)
/projects client.example.com (rw)
```

最初の行では、サーバーは **client.example.com** のみが **/projects** ディレクトリーを読み取り/書き込みモードでマウントすることを許可し、他のホストは共有をマウントできません。ただし、2 行目の **client.example.com** と **(rw)** の間にスペースがあるため、サーバーはディレクトリーを読み取り専用モード (デフォルト設定) で **client.example.com** にエクスポートしますが、他のすべてのホストは共有を読み取り/書き込みモードでマウントできます。

NFS サーバーは、エクスポートされた各ディレクトリーに対して次のデフォルト設定を使用します。

表4.2 /etc/exports のエントリーのデフォルトオプション

デフォルト設定	説明
ro	ディレクトリーを読み取り専用モードでエクスポートします。

デフォルト設定	説明
sync	NFS サーバーは、以前の要求によって行われた変更がディスクに書き込まれるまで、要求に応答しません。
wdelay	別の書き込み要求が保留中であると疑われる場合、サーバーはディスクへの書き込みを遅延します。
root_squash	クライアントの **root** ユーザーがエクスポートされたディレクトリーに対して **root** 権限を持つことを防ぎます。**root_squash** を有効にすると、NFS サーバーは **root** からユーザー **nobody** へのアクセスをマップします。

4.7. NFSV4 専用サーバーの設定

ネットワーク内に NFSv3 クライアントが存在しない場合は、NFSv4 またはその特定のマイナープロトコルバージョンのみをサポートするように NFS サーバーを設定できます。サーバー上で NFSv4 のみを使用すると、ネットワークに開かれるポートの数が減ります。

手順

1. **nfs-utils** パッケージをインストールします。

```
# dnf install nfs-utils
```

2. **/etc/nfs.conf** ファイルを編集し、次の変更を加えます。

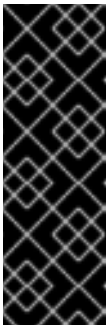
   a. NFSv3 を無効にするには、**[nfsd]** セクションの **vers3** パラメーターを無効にします。

```
[nfsd]
vers3=n
```

   b. オプション: 特定の NFSv4 マイナーバージョンのみが必要な場合は、すべての **vers4.<minor_version>** パラメーターのコメントを解除し、それに応じて設定します。次に例を示します。

```
[nfsd]
vers3=n
# vers4=y
vers4.0=n
vers4.1=n
vers4.2=y
```

   この設定では、サーバーは NFS バージョン 4.2 のみを提供します。



重要

特定の NFSv4 マイナーバージョンのみが必要な場合は、マイナーバージョンのパラメーターのみを設定します。マイナーバージョンの予期しないアクティブ化または非アクティブ化を回避するために、**vers4** パラメーターのコメントを解除しないでください。デフォルトでは、**vers4** パラメーターはすべての NFSv4 マイナーバージョンを有効または無効にします。ただし、**vers4** を他の **vers** パラメーターと組み合わせて設定すると、この動作は変わります。

3. NFSv3 関連のすべてのサービスを無効にします。

```
# systemctl mask --now rpc-statd.service rpcbind.service rpcbind.socket
```

4. オプション: 共有するディレクトリーを作成します。例:

```
# mkdir -p /nfs/projects/
```

既存のディレクトリーを共有する場合は、この手順をスキップしてください。

5. **/nfs/projects/** ディレクトリーに必要な権限を設定します。

```
# chmod 2770 /nfs/projects/
# chgrp users /nfs/projects/
```

これらのコマンドは、**/nfs/projects/** ディレクトリーの **ユーザー** グループの書き込み権限を設定し、このディレクトリーに作成される新しいエントリーに同じグループが自動的に設定されるようにします。

6. 共有するディレクトリーごとに、**/etc/exports** ファイルにエクスポートポイントを追加します。

```
/nfs/projects/     192.0.2.0/24(rw) 2001:db8::/32(rw)
```

このエントリーは 、**/nfs/projects/** ディレクトリーを共有し、**192.0.2.0/24** および **2001:db8::/32** サブネット内のクライアントが読み取りおよび書き込みアクセスできるようにします。

7. **Firewalld** で関連するポートを開きます:

```
# firewall-cmd --permanent --add-service nfs
# firewall-cmd --reload
```

8. NFS サーバーを有効にして起動します。

```
# systemctl enable --now nfs-server
```

検証

- サーバー上で、設定した NFS バージョンのみがサーバーから提供されていることを確認します。

```
# cat /proc/fs/nfsd/versions
-3 +4 -4.0 -4.1 +4.2
```

- クライアントで、次の手順を実行します。

  1. **nfs-utils** パッケージをインストールします。

     ```
     # dnf install nfs-utils
     ```

  2. エクスポートされた NFS 共有をマウントします。

     ```
     # mount server.example.com:/nfs/projects/ /mnt/
     ```

  3. **users** グループのメンバーであるユーザーとして、**/mnt/** にファイルを作成します。

     ```
     # touch /mnt/file
     ```

  4. ファイルが作成されたことを確認するためにディレクトリーをリスト表示します。

     ```
     # ls -l /mnt/
     total 0
     -rw-r--r--. 1 demo users 0 Jan 16 14:18 file
     ```

4.8. オプションの NFSV4 サポートを備えた NFSV3 サーバーの設定

NFSv3 クライアントをまだ使用しているネットワークでは、NFSv3 プロトコルを使用して共有を提供するようにサーバーを設定します。ネットワーク内に新しいクライアントもある場合は、さらに NFSv4 を有効にすることもできます。デフォルトでは、Red Hat Enterprise Linux NFS クライアントはサーバーが提供する最新の NFS バージョンを使用します。

手順

1. **nfs-utils** パッケージをインストールします。

   ```
   # dnf install nfs-utils
   ```

2. オプション: デフォルトでは、NFSv3 と NFSv4 が有効になっています。NFSv4 が必要ない場合、または特定のマイナーバージョンのみが必要な場合は、すべての **vers4.<minor_version>** パラメーターのコメントを解除し、それに応じて設定します。

   ```
   [nfsd]
   # vers3=y
   # vers4=y
   vers4.0=n
   vers4.1=n
   vers4.2=y
   ```

   この設定では、サーバーは NFS バージョン 3 と 4.2 のみを提供します。

重要

特定の NFSv4 マイナーバージョンのみが必要な場合は、マイナーバージョンのパラメーターのみを設定します。マイナーバージョンの予期しないアクティブ化または非アクティブ化を回避するために、**vers4** パラメーターのコメントを解除しないでください。デフォルトでは、**vers4** パラメーターはすべての NFSv4 マイナーバージョンを有効または無効にします。ただし、**vers4** を他の **vers** パラメーターと組み合わせて設定すると、この動作は変わります。

3. デフォルトでは、NFSv3 RPC サービスはランダムポートを使用します。ファイアウォール設定を有効にするには、**/etc/nfs.conf** ファイルで固定ポート番号を設定します。

   a. **[lockd]** セクションで、**nlockmgr** RPC サービスの固定ポート番号を設定します。例:

   ```
   [lockd]
   port=5555
   ```

   この設定により、サービスは UDP プロトコルと TCP プロトコルの両方にこのポート番号を自動的に使用します。

   b. **[statd]** セクションで、**rpc.statd** サービスの固定ポート番号を設定します。例:

   ```
   [statd]
   port=6666
   ```

   この設定により、サービスは UDP プロトコルと TCP プロトコルの両方にこのポート番号を自動的に使用します。

4. オプション: 共有するディレクトリーを作成します。例:

   ```
   # mkdir -p /nfs/projects/
   ```

   既存のディレクトリーを共有する場合は、この手順をスキップしてください。

5. **/nfs/projects/** ディレクトリーに必要な権限を設定します。

   ```
   # chmod 2770 /nfs/projects/
   # chgrp users /nfs/projects/
   ```

   これらのコマンドは、**/nfs/projects/** ディレクトリーの **ユーザー** グループの書き込み権限を設定し、このディレクトリーに作成される新しいエントリーに同じグループが自動的に設定されるようにします。

6. 共有するディレクトリーごとに、**/etc/exports** ファイルにエクスポートポイントを追加します。

   ```
   /nfs/projects/     192.0.2.0/24(rw) 2001:db8::/32(rw)
   ```

   このエントリーは 、**/nfs/projects/** ディレクトリーを共有し、**192.0.2.0/24** および **2001:db8::/32** サブネット内のクライアントが読み取りおよび書き込みアクセスできるようにします。

7. **Firewalld** で関連するポートを開きます:

```
# firewall-cmd --permanent --add-service={nfs,rpc-bind,mountd}
# firewall-cmd --permanent --add-port={5555/tcp,5555/udp,6666/tcp,6666/udp}
# firewall-cmd --reload
```

8. NFS サーバーを有効にして起動します。

```
# systemctl enable --now rpc-statd nfs-server
```

検証

- サーバー上で、設定した NFS バージョンのみがサーバーから提供されていることを確認します。

```
# cat /proc/fs/nfsd/versions
+3 +4 -4.0 -4.1 +4.2
```

- クライアントで、次の手順を実行します。

1. **nfs-utils** パッケージをインストールします。

```
# dnf install nfs-utils
```

2. エクスポートされた NFS 共有をマウントします。

```
# mount -o vers=<version> server.example.com:/nfs/projects/ /mnt/
```

3. 指定された NFS バージョンで共有がマウントされたことを確認します。

```
# mount | grep "/mnt"
server.example.com:/nfs/projects/ on /mnt type nfs (rw,relatime,vers=3,...
```

4. **users** グループのメンバーであるユーザーとして、**/mnt/** にファイルを作成します。

```
# touch /mnt/file
```

5. ファイルが作成されたことを確認するためにディレクトリーをリスト表示します。

```
# ls -l /mnt/
total 0
-rw-r--r--. 1 demo users 0 Jan 16 14:18 file
```

4.9. NFS サーバーでクォータサポートを有効にする

ユーザーまたはグループが保存できるデータの量を制限する場合は、ファイルシステムにクォータを設定できます。NFS サーバーでは、**rpc-rquotad** サービスにより、クォータが NFS クライアント上のユーザーにも適用されるようになります。

前提条件

- NFS サーバーは実行され、設定されています。
- ext または XFS ファイルシステムでクォータが設定されています。

手順

1. エクスポートするディレクトリーでクォータが有効になっていることを確認します。

   - ext ファイルシステムの場合は、次のように入力します。

     ```
     # quotaon -p /nfs/projects/
     group quota on /nfs/projects (/dev/sdb1) is on
     user quota on /nfs/projects (/dev/sdb1) is on
     project quota on /nfs/projects (/dev/sdb1) is off
     ```

   - XFS ファイルシステムの場合は、次のように入力します。

     ```
     # findmnt /nfs/projects
     TARGET     SOURCE FSTYPE OPTIONS
     /nfs/projects /dev/sdb1 xfs
     rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,usrquota,grpquota
     ```

2. **クォータ -rpc** パッケージをインストールします。

   ```
   # dnf install rpc-quotad
   ```

3. オプション: デフォルトでは、クォータ RPC サービスはポート 875 で実行されます。別のポートでサービスを実行する場合は、**/etc/sysconfig/rpc-rquotad** ファイルの **RPCRQUOTADOPTS** 変数に **-p <port_number>** を追加します。

   ```
   RPCRQUOTADOPTS="-p __<port_number>__"
   ```

4. オプション: デフォルトでは、リモートホストはクォータの読み取りのみが可能です。クライアントがクォータを設定できるようにするには、**/etc/sysconfig/rpc-rquotad** ファイルの **RPCRQUOTADOPTS** 変数に **-S** オプションを追加します。

   ```
   RPCRQUOTADOPTS="-S"
   ```

5. **FirewallD** でポートを開きます:

   ```
   # firewall-cmd --permanent --add-port=875/udp
   # firewall-cmd --reload
   ```

6. **rpc-quotad** サービスを有効にして起動します。

   ```
   # systemctl enable --now rpc-rquotad
   ```

検証

1. クライアント上:

   a. エクスポートした共有をマウントします。

      ```
      # mount server.example.com:/nfs/projects/ /mnt/
      ```

   b. クォータを表示します。コマンドは、エクスポートされたディレクトリーのファイルシステムによって異なります。以下に例を示します。

- マウントされたすべての ext ファイルシステム上の特定のユーザーのクォータを表示するには、次のように入力します。

```
# quota -u <user_name>
Disk quotas for user demo (uid 1000):
     Filesystem  space  quota  limit  grace  files  quota  limit  grace
server.example.com:/nfs/projects
               0K      100M     200M              0        0        0
```

- XFS ファイルシステム上のユーザーおよびグループのクォータを表示するには、次のように入力します。

```
# xfs_quota -x -c "report -h" /mnt/
User quota on /nfs/projects (/dev/vdb1)
                Blocks
User ID     Used     Soft     Hard     Warn/Grace
---------- ---------------------------------
root        0        0        0        00 [------]
demo        0        100M     200M     00 [------]
```

関連情報

- **quota (1)** man ページ
- **xfs_quota(8)** man page

4.10. NFS サーバーで RDMA 経由の NFS を有効にする

リモートダイレクトメモリーアクセス (RDMA) は、クライアントシステムがストレージサーバーのメモリーから自身のメモリーにデータを直接転送できるようにするプロトコルです。これにより、ストレージのスループットが向上し、サーバーとクライアント間のデータ転送の遅延が減少し、両端の CPU 負荷が軽減されます。NFS サーバーとクライアントの両方が RDMA 経由で接続されている場合、クライアントは NFSoRDMA を使用してエクスポートされたディレクトリーをマウントできます。

前提条件

- NFS サービスが実行中であり、設定されている
- InfiniBand または RDMA over Converged Ethernet (RoCE) デバイスがサーバーにインストールされています。
- サーバー上で IP over InfiniBand (IPoIB) が設定され、InfiniBand デバイスに IP アドレスが割り当てられています。

手順

1. **rdma-core** パッケージをインストールします。

```
# dnf install rdma-core
```

2. パッケージがすでにインストールされている場合は、**/etc/rdma/** modules/rdma.conf ファイル内の **xprtrdma** および **svcrdma** モジュールのコメントが解除されていることを確認します。

```
# NFS over RDMA client support
```

```
xprtrdma
# NFS over RDMA server support
svcrdma
```

3. オプション: デフォルトでは、NFS over RDMA はポート 20049 を使用します。別のポートを使用する場合は、**/etc/nfs.conf** ファイルの **[nfsd]** セクションで **rdma-port** 設定を設定します。

```
rdma-port=_<port>_
```

4. **Firewalld** で NFSoRDMA ポートを開きます。

```
# firewall-cmd --permanent --add-port={20049/tcp,20049/udp}
# firewall-cmd --reload
```

20049 以外のポートを設定する場合は、ポート番号を調整してください。

5. **nfs-server** サービスを再起動します。

```
# systemctl restart nfs-server
```

検証

1. InfiniBand ハードウェアを搭載したクライアントで、次の手順を実行します。
   a. 以下のパッケージをインストールします。

```
# dnf install nfs-utils rdma-core
```

   b. エクスポートされた NFS 共有を RDMA 経由でマウントします。

```
# mount -o rdma server.example.com:/nfs/projects/ /mnt/
```

デフォルト (20049) 以外のポート番号を設定する場合は、コマンドに **port=<port_number>** を渡します。

```
# mount -o rdma,port=<port_number> server.example.com:/nfs/projects/ /mnt/
```

   c. 共有が **rdma** オプションでマウントされたことを確認します。

```
# mount | grep "/mnt"
server.example.com:/nfs/projects/ on /mnt type nfs (...,proto=rdma,...)
```

関連情報

- InfiniBand ネットワークおよび RDMA ネットワークの設定

4.11. RED HAT IDENTITY MANAGEMENT ドメインで KERBEROS を使用した NFS サーバーを設定する

Red Hat Identity management (IdM) を使用する場合は、NFS サーバーを IdM ドメインに参加させることができます。これにより、ユーザーとグループを集中管理し、認証、整合性保護、トラフィック暗号化に Kerberos を使用できるようになります。

前提条件

- NFS サーバーは Red Hat Identity management (IdM) ドメインに 登録されて います。
- NFS サーバーは実行され、設定されています。

手順

1. IdM 管理者として Kerberos チケットを取得します。

```
# kinit admin
```

2. **nfs/<FQDN>** サービスプリンシパルを作成します。

```
# ipa service-add nfs/nfs_server.idm.example.com
```

3. IdM から **nfs** サービスプリンシパルを取得し、**/etc/krb5.keytab** ファイルに保存します。

```
# ipa-getkeytab -s idm_server.idm.example.com -p nfs/nfs_server.idm.example.com -k
/etc/krb5.keytab
```

4. オプション: **/etc/krb5.keytab** ファイル内のプリンシパルを表示します。

```
# klist -k /etc/krb5.keytab
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
---- --------------------------------------------------------------------------
   1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   1 nfs/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
   7 host/nfs_server.idm.example.com@IDM.EXAMPLE.COM
```

デフォルトでは、ホストを IdM ドメインに参加させると、IdM クライアントはホストプリンシパルを **/etc/krb5.keytab** ファイルに追加します。ホストプリンシパルが見つからない場合は、**ipa-getkeytab -s idm_server.idm.example.com -p host/nfs_server.idm.example.com -k/etc/krb5.keytab** コマンドを使用して追加します。

5. IdM ID のマッピングを設定するには、**ipa-client-automount** ユーティリティーを使用します。

```
#  ipa-client-automount
Searching for IPA server...
IPA server: DNS discovery
Location: default
Continue to configure the system with these values? [no]: yes
Configured /etc/idmapd.conf
Restarting sssd, waiting for it to become available.
Started autofs
```

6. **/etc/exports** ファイルを更新し、クライアントオプションに Kerberos セキュリティーメソッドを追加します。以下に例を示します。

```
/nfs/projects/       192.0.2.0/24(rw,sec=krb5i)
```

クライアントが複数のセキュリティーメソッドを選択できるようにするには、それらをコロンで区切って指定します。

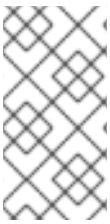
```
/nfs/projects/       192.0.2.0/24(rw,sec=krb5:krb5i:krb5p)
```

7. エクスポートされたファイルシステムを再ロードします。

```
# exportfs -r
```

第5章 SMB 共有のマウント

Server Message Block (SMB) プロトコルは、アプリケーション層のネットワークプロトコルを実装します。これは、ファイル共有や共有プリンターなど、サーバー上のリソースにアクセスするために使用されます。



注記

SMB のコンテキストでは、SMB ダイアレクトである CIFS (Common Internet File System) プロトコルが言及されています。SMB と CIFS の両方のプロトコルがサポートされており、SMB 共有と CIFS 共有のマウントに関連するカーネルモジュールとユーティリティーはどちらも **cifs** という名前を使用します。

cifs-utils パッケージには、以下を行うユーティリティーがあります。

- SMB 共有と CIFS 共有をマウントする
- カーネルのキーリングで、NT LAN Manager (NTLM) の認証情報を管理する
- SMB 共有および CIFS 共有のセキュリティー記述子で、アクセス制御リスト (ACL) を設定して、表示する

5.1. 対応している SMB プロトコルのバージョン

cifs.ko カーネルモジュールは、以下の SMB プロトコルバージョンをサポートします。

- SMB 1

警告

SMB1 プロトコルは既知のセキュリティー問題により非推奨となり、**プライベートネットワークでのみ安全に使用する** ことができます。SMB1 がサポートされているオプションとして推奨される主な理由は、現在 UNIX 拡張機能をサポートする唯一の SMB プロトコルバージョンであるためです。SMB で UNIX 拡張を使用する必要がない場合は、Red Hat は、SMB2 以降を使用することを強く推奨します。

- SMB 2.0
- SMB 2.1
- SMB 3.0
- SMB 3.1.1

注記

プロトコルのバージョンによっては、一部の SMB 機能しか実装されていません。

5.2. UNIX 拡張機能のサポート

Samba は、SMB プロトコルの **CAP_UNIX** 機能ビットを使用して UNIX 拡張機能を提供します。これらの拡張機能は、**cifs.ko** カーネルモジュールでも対応します。ただし、Samba とカーネルモジュールはいずれも、SMB 1 プロトコルでのみ UNIX 拡張機能に対応します。

前提条件

- **cifs-utils** パッケージがインストールされている。

手順

1. **/etc/samba/smb.conf** ファイルの **[global]** セクションにある **server min protocol** パラメーターを **NT1** に設定します。
2. マウントコマンドに **-o vers=1.0** オプションを指定し、SMB 1 プロトコルを使用して共有をマウントします。以下に例を示します。

```
# mount -t cifs -o vers=1.0,username=<user_name> //<server_name>/<share_name> /mnt/
```

デフォルトで、カーネルモジュールは、SMB 2 またはサーバーでサポートされている最新のプロトコルバージョンを使用します。**-o vers=1.0** オプションを **mount** コマンドに渡すと、UNIX 拡張機能の使用に必要な SMB 1 プロトコルをカーネルモジュールが使用することが強制されます。

検証

- マウントされた共有のオプションを表示します。

```
# mount
...
//<server_name>/<share_name> on /mnt type cifs (...,unix,...)
```

マウントオプションのリストに **unix** エントリーが表示されている場合は、UNIX 拡張機能が有効になっています。

5.3. SMB 共有の手動マウント

SMB 共有のみを一時的にマウントする必要がある場合は、**mount** ユーティリティーを使用して手動でマウントできます。

注記

手動でマウントされた共有は、システムを再起動しても自動的にはマウントされません。システムの起動時に、Red Hat Enterprise Linux が自動的に共有をマウントするように設定する場合は、システムの起動時に自動的に SMB 共有をマウントする を参照してください。

前提条件

- **cifs-utils** パッケージがインストールされている。

手順

- **-t cifs** パラメーターを指定して **mount** ユーティリティーを使用して、SMB 共有をマウントします。

```
# mount -t cifs -o username=<user_name> //<server_name>/<share_name> /mnt/
Password for <user_name>@//<server_name>/<share_name>:  password
```

-o パラメーターでは、共有のマウントに使用されるオプションを指定できます。詳細は、**mount.cifs(8)** の man ページおよび 頻繁に使用されるマウントオプション の **OPTIONS** セクションを参照してください。

例5.1 暗号化された SMB 3.0 接続を使用した共有のマウント

暗号化された SMB 3.0 接続で、**DOMAIN\Administrator** ユーザーとして **\\server\example** 共有を **/mnt/** ディレクトリーにマウントする場合は、次の手順を実行します。

```
# mount -t cifs -o username=DOMAIN\Administrator,seal,vers=3.0 //server/example
/mnt/
Password for DOMAIN\Administrator@//server_name/share_name:  password
```

検証

- マウントされた共有の内容をリスト表示します。

```
# ls -l /mnt/
total 4
drwxr-xr-x.  2 root root 8748 Dec  4 16:27 test.txt
drwxr-xr-x. 17 root root 4096 Dec  4 07:43 Demo-Directory
```

5.4. システム起動時の SMB 共有の自動マウント

マウントされた SMB 共有へのアクセスがサーバー上で恒久的に必要とされる場合は、システムの起動時に共有を自動的にマウントします。

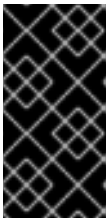
前提条件

- **cifs-utils** パッケージがインストールされている。

手順

1. 共有のエントリーを **/etc/fstab** ファイルに追加します。以下に例を示します。

```
//<server_name>/<share_name>  /mnt  cifs  credentials=/root/smb.cred  0 0
```



重要

システムが自動的に共有をマウントできるようにするには、ユーザー名、パスワード、およびドメイン名を認証情報ファイルに保存する必要があります。詳細は、SMB 共有に対して認証するための認証情報ファイルの作成 を参照してください。

/etc/fstab の行の 4 つ目のフィールドで、認証情報ファイルへのパスなど、マウントオプションを指定します。詳細は、**mount.cifs(8)** の man ページおよび 頻繁に使用されるマウントオプション の **OPTIONS** セクションを参照してください。

検証

- マウントポイントを指定して共有をマウントします。

```
# mount /mnt/
```

5.5. SMB 共有に対して認証するための認証情報ファイルの作成

特定の状況 (システムの起動時に共有を自動的にマウントする場合など) では、ユーザー名とパスワードを入力することなく共有がマウントされる必要があります。これを実装するには、認証情報ファイルを作成します。

前提条件

- **cifs-utils** パッケージがインストールされている。

手順

1. **/root/smb.cred** などのファイルを作成し、そのファイルのユーザー名、パスワード、およびドメイン名を指定します。

```
username=user_name
password=password
domain=domain_name
```

2. 所有者だけがファイルにアクセスできるようにパーミッションを設定します。

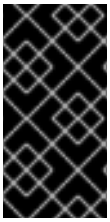
```
# chown user_name /root/smb.cred
# chmod 600 /root/smb.cred
```

mount ユーティリティーに **credentials=file_name** マウントオプションを渡すか、**/etc/fstab** ファイルでこのオプションを使用して、ユーザー名とパスワードの入力を求められずに共有をマウントできます。

5.6. マルチユーザー SMB マウントの実行

共有をマウントするために指定した認証情報により、デフォルトでマウントポイントのアクセス権が決まります。たとえば、共有をマウントするときに **DOMAIN\example** ユーザーを使用した場合は、どのローカルユーザーが操作を実行しても、共有に対するすべての操作はこのユーザーとして実行されます。

ただし特定の状況では、システムの起動時に管理者が自動的に共有をマウントしたい場合でも、ユーザーは自分の認証情報を使用して共有のコンテンツに対して操作を実行する必要があります。このとき、**multiuser** マウントオプションを使用すると、このシナリオを設定できます。



重要

multiuser マウントオプションを使用するには、認証情報ファイルの **krb5** オプションや **ntlmssp** オプションなど、非対話式の方法で認証情報の提供に対応するセキュリティータイプに、**sec** マウントオプションを追加で設定する必要があります。詳細は、ユーザーとしての共有へのアクセス を参照してください。

root ユーザーは、**multiuser** オプションと、共有内のコンテンツへの最低限のアクセスを持つアカウントを使用して、共有をマウントします。通常のユーザーは、**cifscreds** ユーティリティーを使用して、現在のセッションのカーネルキーリングに、自身のユーザー名とパスワードを渡すことができます。マウントされた共有のコンテンツにユーザーがアクセスすると、カーネルは、共有のマウントに最初に使用されたものではなく、カーネルキーリングからの認証情報を使用します。

この機能の使用は、以下の手順で設定されます。

- **multiuser** オプションを使用して共有をマウント
- 任意で、**multiuser** オプションを使用して共有が正常にマウントされたかを確認
- ユーザーとして共有にアクセス

前提条件

- **cifs-utils** パッケージがインストールされている。

5.6.1. multiuser オプションを使用した共有のマウント

ユーザーが自身の認証情報を使用して共有にアクセスする場合は、パーミッションが制限されたアカウントを使用して、**root** ユーザーとして共有をマウントする必要があります。

手順

システムの起動時に、**multiuser** オプションを使用して自動的に共有をマウントするには、次の手順を実行します。

1. **/etc/fstab** ファイルに共有のエントリーを作成します。以下に例を示します。

```
//server_name/share_name  /mnt  cifs
multiuser,sec=ntlmssp,credentials=/root/smb.cred  0 0
```

2. 共有をマウントします。

```
# mount /mnt/
```

システムの起動時に共有を自動的にマウントしない場合は、**-o multiuser,sec=security_type** を **mount** コマンドに渡して手動で共有をマウントします。SMB 共有を手動でマウントする方法は、SMB 共有の手動マウント を参照してください。

5.6.2. SMB 共有が multiuser オプションを使用してマウントされているかどうかの確認

共有が **multiuser** オプションを使用してマウントされているかどうかを確認するには、マウントオプションを表示します。

手順

```
# mount
...
//server_name/share_name on /mnt type cifs (sec=ntlmssp,multiuser,...)
```

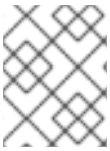
マウントオプションのリストに **multiuser** エントリーが表示されている場合は、機能が有効になっています。

5.6.3. ユーザーとして共有へのアクセス

SMB 共有が **multiuser** オプションを使用してマウントされている場合、ユーザーはサーバーの認証情報をカーネルのキーリングに提供できます。

```
# cifscreds add -u SMB_user_name server_name
Password: password
```

マウントされた SMB 共有を含むディレクトリーでユーザーが操作を実行すると、サーバーは、共有がマウントされたときに最初に使用されたものではなく、このユーザーのファイルシステムのパーミッションを適用します。



注記

複数のユーザーが、マウントされた共有で、自身の認証情報を使用して同時に操作を実行できます。

5.7. よく使用される SMB マウントオプション

SMB 共有をマウントすると、マウントオプションにより次のことが決まります。

- サーバーとの接続がどのように確立されるか。たとえば、サーバーに接続するときに使用される SMB プロトコルバージョンはどれか。
- 共有が、ローカルファイルシステムにどのようにマウントされるか。たとえば、複数のローカルユーザーが、サーバーのコンテンツにアクセスできるようにするために、システムがリモートファイルとディレクトリーのパーミッションを上書きする場合など。

/etc/fstab ファイルの 4 番目のフィールド、またはマウントコマンドの **-o** パラメーターで複数のオプションを設定するには、オプションをコンマで区切ります。たとえば、multiuser オプションを使用した共有のマウント を参照してください。

次のリストは、よく使用されるマウントオプションを示しています。

オプション	説明
credentials=**file_name**	認証情報ファイルへのパスを設定します。認証情報ファイルを使用した SMB 共有への認証 を参照してください。
dir_mode=**mode**	サーバーが CIFS UNIX 拡張機能をサポートしていない場合は、ディレクトリーモードを設定します。
file_mode=**mode**	サーバーが CIFS UNIX 拡張機能をサポートしていない場合は、ファイルモードを設定します。

オプション	説明
password=**password**	SMB サーバーへの認証に使用されるパスワードを設定します。あるいは、**credentials** オプションを使用して認証情報ファイルを指定します。
seal	SMB 3.0 以降のプロトコルバージョンを使用した接続に対する暗号化サポートを有効にします。そのため、**seal** は **3.0** 以降に設定された **vers** マウントオプションと一緒に使用します。SMB 共有の手動マウント の例を参照してください。
sec=**security_mode**	**ntlmsspi** などのセキュリティーモードを設定して、NTLMv2 パスワードハッシュとパケット署名を有効にします。対応している値のリストは、man ページの **mount.cifs(8)** にあるオプションの説明を参照してください。 サーバーが **ntlmv2** セキュリティーモードに対応していない場合は、**sec=ntlmssp** (デフォルト) を使用します。 セキュリティー上の理由から、安全でない **ntlm** セキュリティーモードは使用しないでください。
username=**user_name**	SMB サーバーへの認証に使用されるユーザー名を設定します。あるいは、**credentials** オプションを使用して認証情報ファイルを指定します。
vers=**SMB_protocol_version**	サーバーとの通信に使用される SMB プロトコルバージョンを設定します。

完全なリストは、man ページの **mount.cifs(8)** の **OPTIONS** セクションを参照してください。

第6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、再起動を何度も行っ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。

6.1. 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、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 コントローラー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 コントローラー (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。

6.2. ファイルシステムおよびデバイスの識別子

このセクションでは、ファイルシステムおよびブロックデバイスを識別する永続的な属性の相違点を説明します。

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。一方、**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ものは、次のとおりです。

- 一意の ID (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ものは、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。

6.3. /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
- 一意の ID
- 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、設定可能なものもあります。

6.3.1. 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 **一意の ID** (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。

6.3.2. 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6.1 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
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	**/dev/sdb**	ページ **0x80** 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
/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3	**/dev/sdc3**	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6.2 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
/dev/disk/by-partuuid/4cd1448a-01	**/dev/sda1**
/dev/disk/by-partuuid/4cd1448a-02	**/dev/sda2**
/dev/disk/by-partuuid/4cd1448a-03	**/dev/sda3**

/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。

6.4. 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6.3 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][hwhandler=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追加の手順が必要です。

6.5. 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。

6.6. 永続的な命名属性のリスト表示

この手順では、非永続的なストレージデバイスの永続命名属性を確認する方法を説明します。

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6.4 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6.5 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6.6 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
```

6.7. 永続的な命名属性の変更

この手順では、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する方法を説明します。

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。これにより、次のコマンドが新しい属性を正しく利用できるようになります。

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** ユーティリティーを使用します。

```
# 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** ユーティリティーを使用します。

```
# 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、**swaplabel** ユーティリティーを使用します。

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

第7章 PARTED でのパーティション操作

parted は、ディスクパーティションを操作するプログラムです。MS-DOS や GPT など、複数のパーティションテーブル形式をサポートしています。これは、新しいオペレーティングシステム用のスペースの作成、ディスクの使用方法の再編成、および新しいハードディスクへのデータのコピーに役立ちます。

7.1. 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

1. **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
```

2. 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
```

3. 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については、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

- **parted(8)** man ページ

7.2. 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。
  - **msdo** (MBR の場合)
  - **gpt** (GPT の場合)

例7.1 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。

4. 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
```

5. **parted** シェルを終了します。

```
# (parted) quit
```

関連情報

- **parted(8)** man ページ

7.3. PARTED で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

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、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** ユーティリティーは、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7.2 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

- **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

7.4. 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。

警告

パーティションを削除すると、そのパーティション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
```

   - **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

- **parted(8)** man ページ

7.5. 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に、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

- **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
- ext3 ファイルシステムのサイズ変更
- ext4 ファイルシステムのサイズ変更
- XFS ファイルシステムのサイズの拡大

第8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。

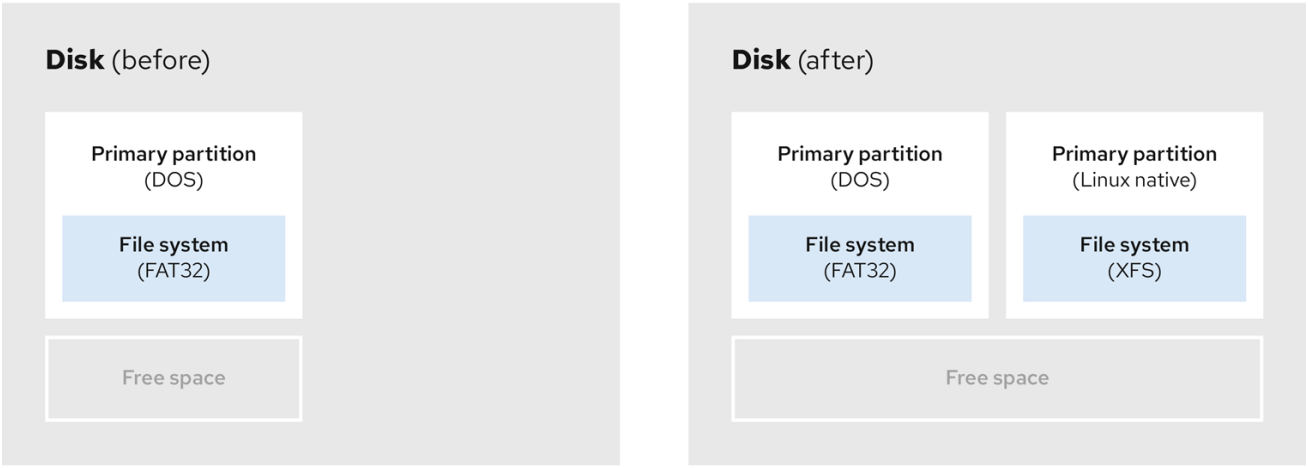
注記

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。

8.1. 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8.1 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



最初の図は、1つのプライマリー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。

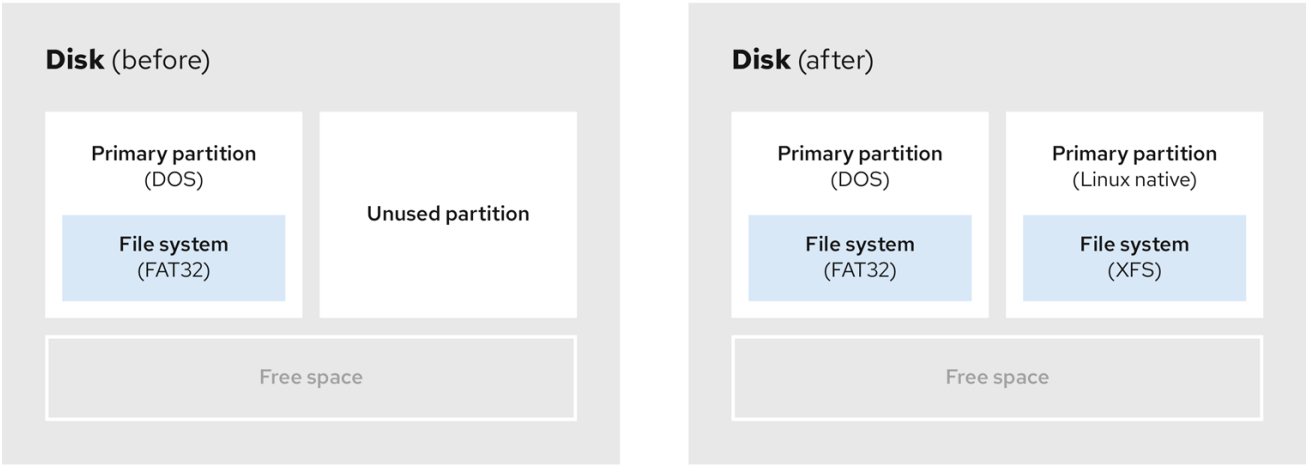
未使用のハードディスクもこのカテゴリーに分類されます。唯一の違いは、**すべて** の領域が定義されたパーティションの一部ではないことです。

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。

8.2. 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8.2 未使用のパーティションがあるディスク



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。

8.3. 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。

警告

アクティブなパーティションでオペレーティングシステム (OS) を使用する場合は、OS を再インストールする必要があります。ソフトウェアが事前にインストールされている一部のコンピューターには、元の OS を再インストールするためのインストールメディアが含まれていないことに注意してください。元のパーティションと OS インストールを破棄する前に、これが OS に当てはまるか確認してください。

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。

8.3.1. 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。

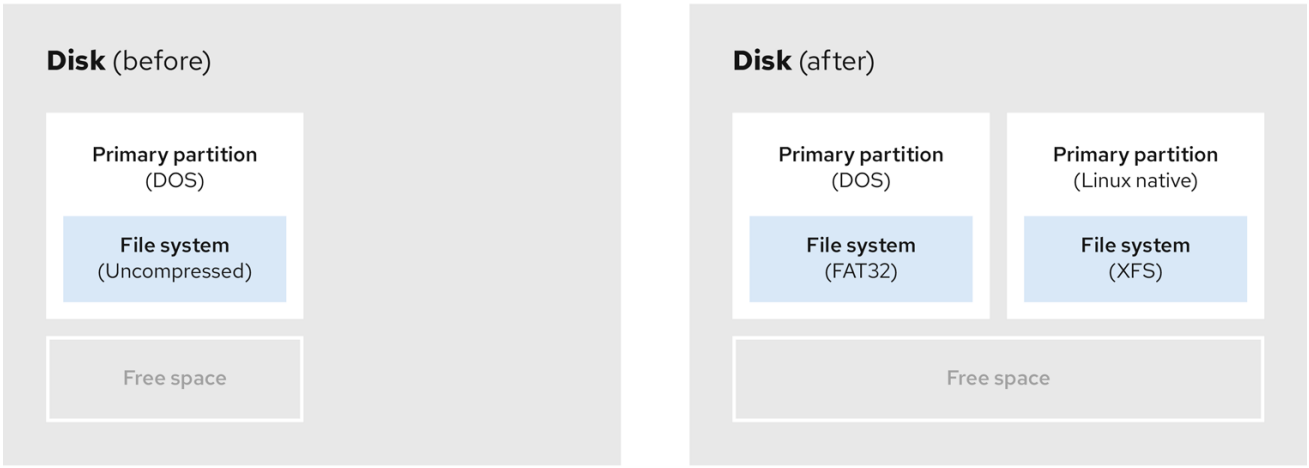
既存のオペレーティングシステム用に小規模なパーティションを作成すると、以下が可能になります。

- ソフトウェアをの再インストール。

- 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8.3 ディスク上での破壊的な再パーティション処理



警告

このメソッドは、元のパーティションに保存されたデータをすべて削除します。

8.3.2. 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。

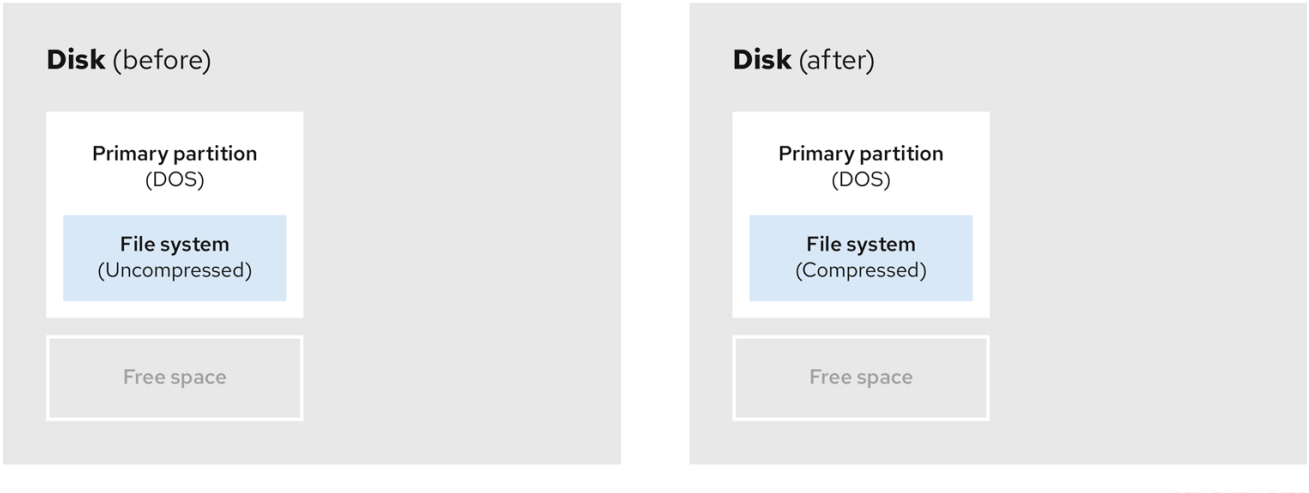
以下は、破壊的なパーティション再設定の開始に役立つメソッドのリストです。

- 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8.4 ディスク上でのデータ圧縮



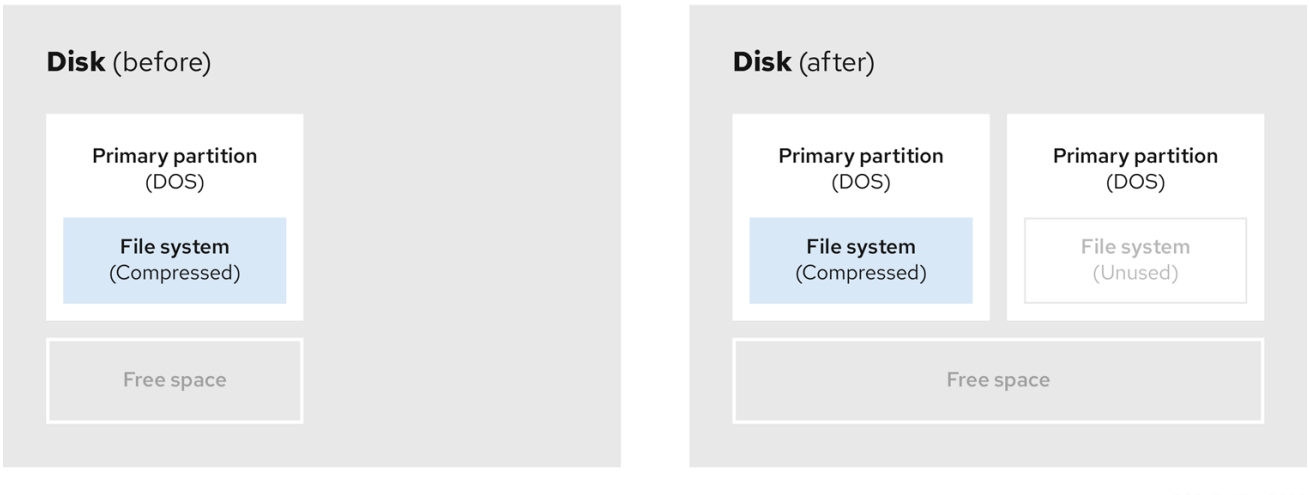
データ損失の可能性を回避するには、圧縮プロセスを続行する前にバックアップを作成します。

- 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8.5 ディスク上でのパーティションのサイズ変更

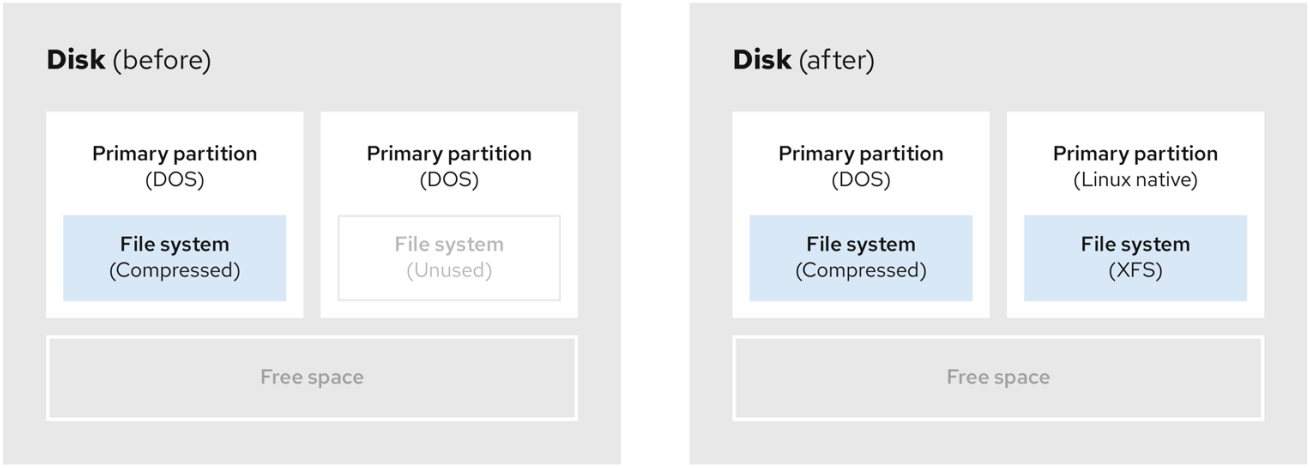


- オプション: 新規パーティションの作成

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8.6 最終パーティション設定のディスク



第9章 XFS の使用

これは、XFS ファイルシステムを作成および維持する方法の概要です。

9.1. XFS ファイルシステム

XFS は、拡張性が高く、高性能で堅牢な、成熟した 64 ビットのジャーナリングファイルシステムで、1 台のホストで非常に大きなファイルおよびファイルシステムに対応します。Red Hat Enterprise Linux 8 ではデフォルトのファイルシステムになります。XFS は、元々 1990 年代の前半に SGI により開発され、極めて大規模なサーバーおよびストレージアレイで実行されてきた長い歴史があります。

XFS の機能は次のとおりです。

信頼性

- メタデータジャーナリング - システムの再起動時、およびファイルシステムの再マウント時に再生できるファイルシステム操作の記録を保持することで、システムクラッシュ後のファイルシステムの整合性を確保します。
- 広範囲に及ぶランタイムメタデータの整合性チェック
- 拡張性が高く、高速な修復ユーティリティー
- クォータジャーナリングクラッシュ後に行なわれる、時間がかかるクォータの整合性チェックが不要になります。

スケーラビリティーおよびパフォーマンス

- 対応するファイルシステムのサイズが最大 1024 TiB
- 多数の同時操作に対応する機能
- 空き領域管理のスケーラビリティーに関する B-Tree インデックス
- 高度なメタデータ先読みアルゴリズム
- ストリーミングビデオのワークロードの最適化

割り当てスキーム

- エクステント (領域) ベースの割り当て
- ストライプを認識できる割り当てポリシー
- 遅延割り当て
- 領域の事前割り当て
- 動的に割り当てられる inode

その他の機能

- Reflink ベースのファイルのコピー
- 密接に統合されたバックアップおよび復元のユーティリティー

- オンラインのデフラグ
- オンラインのファイルシステム拡張
- 包括的な診断機能
- 拡張属性 (**xattr**)。これにより、システムが、ファイルごとに、名前と値の組み合わせを追加で関連付けられるようになります。
- プロジェクトまたはディレクトリーのクォータ。ディレクトリーツリー全体にクォータ制限を適用できます。
- サブセカンド (一秒未満) のタイムスタンプ

パフォーマンスの特徴

XFS は、エンタープライズレベルのワークロードがある大規模なシステムで優れたパフォーマンスを発揮します。大規模なシステムとは、相対的に CPU 数が多く、さらには複数の HBA、および外部ディスクアレイへの接続を備えたシステムです。XFS は、マルチスレッドの並列 I/O ワークロードを備えた小規模のシステムでも適切に実行します。

XFS は、シングルスレッドで、メタデータ集約型のワークロードのパフォーマンスが比較的低くなります。たとえば、シングルスレッドで小さなファイルを多数作成し、削除するワークロードがこれに当てはまります。

9.2. EXT4 および XFS で使用されるツールの比較

本セクションでは、ext4 ファイルシステムおよび XFS ファイルシステムで一般的なタスクを行うのに使用するツールを比較します。

タスク	ext4	XFS
ファイルシステムを作成する	**mkfs.ext4**	**mkfs.xfs**
ファイルシステム検査	**e2fsck**	**xfs_repair**
ファイルシステムのサイズを変更する	**resize2fs**	**xfs_growfs**
ファイルシステムのイメージを保存する	**e2image**	**xfs_metadump** および **xfs_mdrestore**
ファイルシステムのラベル付けまたはチューニングを行う	**tune2fs**	**xfs_admin**
ファイルシステムのバックアップを作成する	**dump** および **restore**	**xfsdump** および **xfsrestore**
クォータ管理	**quota**	**xfs_quota**
ファイルマッピング	**filefrag**	**xfs_bmap**

第10章 XFS ファイルシステムの作成

システム管理者は、ブロックデバイスに XFS ファイルシステムを作成して、ファイルやディレクトリーを格納できます。

10.1. MKFS.XFS で XFS ファイルシステムの作成

この手順では、ブロックデバイスに XFS ファイルシステムを作成する方法を説明します。

手順

1. ファイルシステムを作成する場合は、以下の手順を実行します。
   - デバイスが通常のパーティション、LVM ボリューム、MD ボリューム、ディスク、または類似デバイスである場合は、次のコマンドを使用します。

     ```
     # mkfs.xfs block-device
     ```

     - **block-device** を、ブロックデバイスへのパスに置き換えます。たとえば、**/dev/sdb1**、**/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**、または **/dev/my-volgroup/my-lv** です。
     - 通常、デフォルトのオプションは、一般的な使用に最適なものです。
     - 既存のファイルシステムを含むブロックデバイスで **mkfs.xfs** を使用する場合は、そのファイルシステムを上書きする **-f** オプションを追加してください。
   - ハードウェア RAID デバイスにファイルシステムを作成する場合は、システムがデバイスのストライプジオメトリーを正しく検出しているかどうかを確認します。
     - ストライプジオメトリー情報が正しい場合は、追加のオプションが必要ありません。ファイルシステムを作成します。

       ```
       # mkfs.xfs block-device
       ```

     - 情報が正しくない場合は、**-d** オプションの **su** パラメーターおよび **sw** パラメーターを使用して、ストライプジオメトリーを手動で指定します。**su** パラメーターは RAID チャンクサイズを指定し、**sw** パラメーターは RAID デバイス内のデータディスクの数を指定します。
       以下に例を示します。

       ```
       # mkfs.xfs -d su=64k,sw=4 /dev/sda3
       ```

2. 次のコマンドを使用して、システムが新しいデバイスノードを登録するまで待機します。

   ```
   # udevadm settle
   ```

関連情報

- **mkfs.xfs(8)** の man ページ。

第11章 XFS ファイルシステムのバックアップ

システム管理者は、**xfsdump** を使用して XFS ファイルシステムをファイルまたはテープにバックアップできます。これは、簡単なバックアップメカニズムを提供します。

11.1. XFS バックアップの機能

本セクションでは、**xfsdump** ユーティリティーを使用して XFS ファイルシステムをバックアップする場合の主な概念と機能を説明します。

xfsdump ユーティリティーを使用すると次のことができます。

- 通常のファイルイメージへのバックアップ
  通常のファイルに書き込むことができるバックアップは 1 つだけです。

- テープドライブへのバックアップ
  xfsdump ユーティリティーを使用すると、同じテープに複数のバックアップを書き込むこともできます。バックアップは、複数のテープを分割して書き込むことができます。

  複数のファイルシステムのバックアップを 1 つのテープデバイスに作成するには、XFS バックアップがすでに含まれているテープにバックアップを書き込みます。これにより、古いバックアップに、新しいバックアップが追加されます。**xfsdump** は、デフォルトでは既存のバックアップを上書しません。

- 増分バックアップの作成
  xfsdump ユーティリティーはダンプレベルを使用して、その他のバックアップの相対的なベースバックアップを決定します。0 から 9 までの数字は、ダンプレベルの増加を表します。増分バックアップは、下位レベルの最後のダンプ以降に変更したファイルのみが対象となります。

  - フルバックアップを実行する場合は、ファイルシステムでレベル 0 のダンプを実行します。

  - レベル 1 のダンプは、フルバックアップ後の最初の増分バックアップです。次の増分バックアップはレベル 2 になります。これは、前回のレベル 1 のダンプ以降に変更したファイルのみが対象となります。レベル 9 まで同様です。

- ファイルを絞り込むサイズ、サブツリー、または inode のフラグを使用して、バックアップからファイルを除外

関連情報

- **xfsdump(8)** man ページ

11.2. XFSDUMP で XFS ファイルシステムのバックアップ

この手順では、XFS ファイルシステムのコンテンツのバックアップを、ファイルまたはテープに作成する方法を説明します。

前提条件

- バックアップが可能な XFS ファイルシステム

- バックアップを保存できる別のファイルシステムまたはテープドライブ

手順

- 次のコマンドを使用して、XFS ファイルシステムのバックアップを作成します。

```
# xfsdump -l level [-L label] \
        -f backup-destination path-to-xfs-filesystem
```

  - level を、バックアップのダンプレベルに置き換えます。フルバックアップを実行する場合は **0** を使用し、それに続く増分バックアップを実行する場合は **1** から **9** を使用します。
  - backup-destination を、バックアップを保存する場所のパスに置き換えます。保存場所は、通常のファイル、テープドライブ、またはリモートテープデバイスです。たとえば、ファイルの場合は **/backup-files/Data.xfsdump**、テープドライブの場合は **/dev/st0** に置き換えます。
  - path-to-xfs-filesystem を、バックアップを作成する XFS ファイルシステムのマウントポイントに置き換えます。たとえば、**/mnt/data/** に置き換えます。ファイルシステムをマウントする必要があります。
  - 複数のファイルシステムのバックアップを作成して 1 つのテープデバイスに保存する場合は、復元時にそれらを簡単に識別できるように **-L label** オプションを使用して、各バックアップにセッションラベルを追加します。label を、バックアップの名前 (例: **backup_data**) に置き換えます。

例11.1 複数の XFS ファイルシステムのバックアップ

- **/boot/** ディレクトリーおよび **/data/** ディレクトリーにマウントされている XFS ファイルシステムのコンテンツのバックアップを作成し、作成したバックアップ内容をファイルとして **/backup-files/** ディレクトリーに保存するには、次のコマンドを実行します。

```
# xfsdump -l 0 -f /backup-files/boot.xfsdump /boot
# xfsdump -l 0 -f /backup-files/data.xfsdump /data
```

- 1 つのテープデバイスにある複数のファイルシステムのバックアップを作成する場合は、**-L label** オプションを使用して、各バックアップにセッションラベルを追加します。

```
# xfsdump -l 0 -L "backup_boot" -f /dev/st0 /boot
# xfsdump -l 0 -L "backup_data" -f /dev/st0 /data
```

関連情報

- **xfsdump(8)** man ページ

11.3. 関連情報

- **xfsdump(8)** man ページ

第12章 バックアップからの XFS ファイルシステムの復元

システム管理者は、**xfsrestore** ユーティリティーを使用して、**xfsdump** ユーティリティーで作成され、ファイルまたはテープに保存されている XFS バックアップを復元できます。

12.1. バックアップから XFS を復元する機能

xfsrestore ユーティリティーは、**xfsdump** により作成されたバックアップからファイルシステムを復元します。**xfsrestore** ユーティリティーには 2 つのモードがあります。

- **simple** モードでは、ユーザーはレベル 0 のダンプからファイルシステム全体を復元できます。これがデフォルトのモードです。
- **cumulative** モードでは、増分バックアップ (つまりレベル 1 からレベル 9) からファイルシステムを復元できます。

各バックアップは、**session ID** または **session label** で一意に識別されます。複数のバックアップを含むテープからバックアップを復元するには、対応するセッション ID またはラベルが必要です。

バックアップから特定のファイルを抽出、追加、または削除するには、**xfsrestore** インタラクティブモードを起動します。インタラクティブモードでは、バックアップファイルを操作する一連のコマンドが提供されます。

関連情報

- **xfsrestore(8)** man ページ

12.2. XFSRESTORE を使用してバックアップから XFS ファイルシステムを復元

この手順では、XFS ファイルシステムの内容を、ファイルまたはテープのバックアップから復元する方法を説明します。

前提条件

- XFS ファイルシステムのバックアップの作成 の説明に従って、XFS ファイルシステムのファイルまたはテープのバックアップ
- バックアップを復元できるストレージデバイス。

手順

- バックアップを復元するコマンドは、フルバックアップから復元するか、増分バックアップから復元するか、1 つのテープデバイスから複数のバックアップを復元するかによって異なります。

```
# xfsrestore [-r] [-S session-id] [-L session-label] [-i]
           -f backup-location restoration-path
```

  - **backup-location** を、バックアップの場所に置き換えます。これは、通常のファイル、テープドライブ、またはリモートテープデバイスになります。たとえば、ファイルの場合は **/backup-files/Data.xfsdump**、テープドライブの場合は **/dev/st0** に置き換えます。

- **restoration-path** を、ファイルシステムを復元するディレクトリーへのパスに置き換えます。たとえば、**/mnt/data/** に置き換えます。

- ファイルシステムを増分 (レベル 1 からレベル 9) バックアップから復元するには、**-r** オプションを追加します。

- 複数のバックアップを含むテープデバイスからバックアップを復元するには、**-S** オプションまたは **-L** オプションを使用してバックアップを指定します。
  -S オプションではセッション ID でバックアップを選択でき、**-L** オプションではセッションラベルで選択できます。セッション ID とセッションラベルを取得するには、**xfsrestore -I** コマンドを使用します。

  session-id を、バックアップのセッション ID に置き換えます。たとえば、**b74a3586-e52e-4a4a-8775-c3334fa8ea2c** に置き換えます。**session-label** を、バックアップのセッションラベルに置き換えます。たとえば、**my_backup_session_label** に置き換えます。

- **xfsrestore** をインタラクティブに使用するには、**-i** オプションを使用します。
  インタラクティブダイアログは、指定されたデバイスの、**xfsrestore** による読み取りが終了してから始まります。インタラクティブな **xfsrestore** シェルの使用可能なコマンドには、**cd**、**ls**、**add**、**delete**、**extract** があります。コマンドの全リストを見るには、**help** コマンドを使用します。

例12.1 複数の XFS ファイルシステムの復元

- XFS バックアップファイルを復元し、その内容を **/mnt/** 配下のディレクトリーに保存するには、次のコマンドを実行します。

```
# xfsrestore -f /backup-files/boot.xfsdump /mnt/boot/
# xfsrestore -f /backup-files/data.xfsdump /mnt/data/
```

- 複数のバックアップを含むテープデバイスから復元するには、各バックアップをセッションラベルまたはセッション ID で指定します。

```
# xfsrestore -L "backup_boot" -f /dev/st0 /mnt/boot/
# xfsrestore -S "45e9af35-efd2-4244-87bc-4762e476cbab" \
        -f /dev/st0 /mnt/data/
```

関連情報

- **xfsrestore(8)** man ページ

12.3. テープから XFS バックアップを復元するときの情報メッセージ

複数のファイルシステムのバックアップを使用してテープからバックアップを復元するとき、**xfsrestore** ユーティリティーがメッセージを出力することがあります。メッセージは、**xfsrestore** がテープ上の各バックアップを順番に調べたときに、要求されたバックアップと一致するものが見つかったかどうかを通知します。以下に例を示します。

```
xfsrestore: preparing drive
xfsrestore: examining media file 0
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match the
media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
xfsrestore: examining media file 1
```

```
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a) does not match the
media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-c50467912408)
[...]
```

情報メッセージは、一致するバックアップが見つかるまで継続して表示されます。

12.4. 関連情報

- **xfsrestore(8)** man ページ

第13章 XFS ファイルシステムのサイズの拡大

システム管理者は、XFS ファイルシステムのサイズを増やして、より大きなストレージ容量を最大限に活用できます。

重要

現在、XFS ファイルシステムのサイズを縮小することはできません。

13.1. XFS_GROWFS で XFS ファイルシステムのサイズの拡大

この手順では、**xfs_growfs** ユーティリティーを使用して XFS ファイルシステムを拡張する方法を説明します。

前提条件

- 基礎となるブロックデバイスのサイズが、後でファイルシステムのサイズを変更するのに十分な大きさである。該当するブロックデバイスのサイズを変更する場合は、ブロックデバイスに適した方法を選択してください。
- XFS ファイルシステムをマウントしている。

手順

- XFS ファイルシステムのマウント時に、**xfs_growfs** ユーティリティーを使用してサイズを大きくします。

```
# xfs_growfs file-system -D new-size
```

  - **file-system** を、XFS ファイルシステムのマウントポイントに置き換えます。
  - **-D** オプションを指定して、**new-size** を、ファイルシステムブロックの数で指定されているファイルシステムの新しいサイズに置き換えます。
    特定の XFS ファイルシステムのブロックサイズ (KB 単位) を調べるには、**xfs_info** ユーティリティーを使用します。

```
# xfs_info block-device

...
data     =               bsize=4096
...
```

  - **xfs_growfs** は、**-D** オプションを指定しないと、基となるデバイスがサポートする最大サイズまでファイルシステムを拡張します。

関連情報

- **xfs_growfs(8)** man ページ

第14章 XFS エラー動作の設定

異なる I/O エラーが発生すると、XFS ファイルシステムの動作を設定できます。

14.1. XFS で設定可能なエラー処理

XFS ファイルシステムは、I/O 操作中にエラーが発生すると、以下のいずれかの方法で応答します。

- XFS は、操作が成功するまで、または XFS が設定制限に到達するまで I/O 操作を繰り返し再試行します。
  この制限は、再試行の最大数または再試行の最大時間を基にしています。

- XFS は、エラーを永続的に考慮し、ファイルシステムで操作を停止します。

XFS が以下のエラー条件に反応する方法を設定できます。

EIO
　読み取りまたは書き込み時のエラー

ENOSPC
　デバイスに空き容量がない

ENODEV
　デバイスが見つからない

最大再試行回数と、XFS がエラーを永続的と見なすまでの最大時間を秒単位で設定できます。XFS は、いずれかの制限に達すると操作の再試行を停止します。

また、ファイルシステムのマウントを解除するときに、他の設定に関係なく XFS が再試行を即座にキャンセルするように XFS を設定することもできます。この設定により、永続的なエラーを出しても、マウント解除操作は成功します。

デフォルトの動作

各 XFS エラー条件のデフォルトの動作は、エラーコンテキストによって異なります。**ENODEV** などの XFS エラーは、リトライ回数に関係なく致命的で回復不能とみなされます。デフォルトの再試行制限は 0 です。

14.2. 特定の、未定義の XFS エラー条件の設定ファイル

以下のディレクトリーは、さまざまなエラー状態に対して XFS エラー動作を制御する設定ファイルを保存します。

/sys/fs/xfs/device/error/metadata/EIO/
　EIO エラー条件の場合

/sys/fs/xfs/device/error/metadata/ENODEV/
　ENODEV エラー条件の場合

/sys/fs/xfs/device/error/metadata/ENOSPC/
　ENOSPC エラー条件の場合

/sys/fs/xfs/device/error/default/
　その他のすべての未定義エラー条件の共通設定

各ディレクトリーには、再試行制限を設定するために以下の設定ファイルが含まれています。

max_retries

XFS が操作を再試行する最大回数を制御します。

retry_timeout_seconds

XFS が操作の再試行を停止するまでの時間制限を秒単位で指定します。

14.3. 特定の条件に対する XFS 動作の設定

この手順では、XFS が特定のエラー条件にどのように反応するかを設定します。

手順

- 再試行の最大数、再試行時間制限、またはその両方を設定します。
  - 再試行の最大数を設定するには、必要な数を **max_retries** ファイルに書き込みます。

```
# echo value > /sys/fs/xfs/device/error/metadata/condition/max_retries
```

  - 時間制限を設定するには、希望する秒数を **retry_timeout_seconds** ファイルに書き込みます。

```
# echo value > /sys/fs/xfs/device/error/metadata/condition/retry_timeout_second
```

value は、-1 から C 符号付き整数型の可能な最大値です。これは、64 ビットの Linux では 2147483647 です。

いずれの制限も、**-1** の値は継続的な再試行に使用され、**0** は即座に停止するために使用されます。

device は、**/dev/** ディレクトリーにあるデバイス名です。たとえば、**sda** です。

14.4. 未定義の条件に対する XFS 動作の設定

この手順では、XFS が、共通の設定を共有するすべての未定義のエラー条件に反応する方法を設定します。

手順

- 再試行の最大数、再試行時間制限、またはその両方を設定します。
  - 再試行の最大数を設定するには、必要な数を **max_retries** ファイルに書き込みます。

```
# echo value > /sys/fs/xfs/device/error/metadata/default/max_retries
```

  - 時間制限を設定するには、希望する秒数を **retry_timeout_seconds** ファイルに書き込みます。

```
# echo value > /sys/fs/xfs/device/error/metadata/default/retry_timeout_seconds
```

value は、-1 から C 符号付き整数型の可能な最大値です。これは、64 ビットの Linux では 2147483647 です。

いずれの制限も、**-1** の値は継続的な再試行に使用され、**0** は即座に停止するために使用されます。

device は、**/dev/** ディレクトリーにあるデバイス名です。たとえば、**sda** です。

14.5. XFS アンマウント動作の設定

この手順では、ファイルシステムのアンマウント時に XFS がエラー状態に対応するように設定します。

ファイルシステムに **fail_at_unmount** オプションを設定すると、マウント解除時にその他すべてのエラー設定が上書きされ、I/O 操作を再試行せずにすぐにファイルシステムをマウント解除します。これにより、永続的なエラーが発生した場合でも、マウント解除操作を成功できます。

警告

マウント解除プロセスの開始後に **fail_at_unmount** の値を変更することはできません。アンマウントプロセスにより、設定ファイルが各ファイルシステムの **sysfs** インターフェイスから削除されます。ファイルシステムのマウント解除を開始する前に、マウント解除動作を設定する必要があります。

手順

- **fail_at_unmount** オプションを有効または無効にします。
  - ファイルシステムのマウント解除時にすべての操作を再試行する場合は、オプションを有効にします。

    ```
    # echo 1 > /sys/fs/xfs/device/error/fail_at_unmount
    ```

  - ファイルシステムのマウント解除時に、再試行制限の **max_retries** および **retry_timeout_seconds** を回避するには、オプションを無効にします。

    ```
    # echo 0 > /sys/fs/xfs/device/error/fail_at_unmount
    ```

  device は、**/dev/** ディレクトリーにあるデバイス名です。たとえば、**sda** です。

第15章 ファイルシステムの検査と修復

RHEL は、ファイルシステムの検査および修復が可能なファイルシステム管理ユーティリティーを提供します。このツールは、通常 **fsck** ツールと呼ばれることが多く、**fsck** は、**file system check** を短くした名前になります。ほとんどの場合、このようなユーティリティーは、必要に応じてシステムの起動時に自動的に実行しますが、必要な場合は手動で呼び出すこともできます。

重要

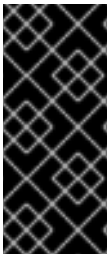
ファイルシステムチェッカーは、ファイルシステム全体のメタデータの整合性のみを保証します。チェッカーは、ファイルシステムに含まれる実際のデータを認識しないため、データのリカバリーツールではありません。

15.1. ファイルシステムの検査が必要なシナリオ

以下のいずれかが発生した場合は、関連する **fsck** ツールを使用してシステムを検査できます。

- システムが起動しない
- 特定ディスクのファイルが破損する
- 不整合によりファイルシステムがシャットダウンするか、読み取り専用に変更する
- ファイルシステムのファイルにアクセスできない

ファイルシステムの不整合は、ハードウェアエラー、ストレージ管理エラー、ソフトウェアバグなどのさまざまな理由で発生する可能性があります。

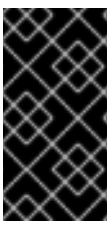


重要

ファイルシステムの検査ツールは、ハードウェアの問題を修復できません。修復を正常に動作させるには、ファイルシステムが完全に読み取り可能かつ書き込み可能である必要があります。ハードウェアエラーが原因でファイルシステムが破損した場合は、まず **dd(8)** ユーティリティーなどを使用して、ファイルシステムを適切なディスクに移動する必要があります。

ジャーナリングファイルシステムの場合、システムの起動時に通常必要なのは、必要に応じてジャーナルを再生することだけで、これは通常非常に短い操作になります。

ただし、ジャーナリングファイルシステムであっても、ファイルシステムの不整合や破損が発生した場合は、ファイルシステムチェッカーを使用してファイルシステムを修復する必要があります。



重要

/etc/fstab の 6 番目のフィールドを **0** に設定すると、システムの起動時にファイルシステムの検査を無効にできます。ただし、Red Hat は、システムの起動時に **fsck** に問題がある場合 (非常に大きなファイルシステムやリモートファイルシステムなど) を除いて、無効にすることを推奨しません。

関連情報

- **fstab(5)** man ページ
- **fsck(8)** man ページ

- **dd(8)** man ページ

15.2. FSCK の実行による潜在的な悪影響

通常、ファイルシステムの検査および修復のツールを実行すると、検出された不整合の少なくとも一部が自動的に修復されることが期待できます。場合によっては、以下の問題が発生する場合があります。

- inode やディレクトリーが大幅に損傷し、修復できない場合は、破棄される場合があります。
- ファイルシステムが大きく変更する場合があります。

予期しない変更や、望ましくない変更が永続的に行われないようにするには、この手順にまとめられている予防手順を行ってください。

15.3. XFS のエラー処理メカニズム

本セクションでは、XFS がファイルシステム内のさまざまな種類のエラーを処理する方法を説明します。

不完全なアンマウント
ジャーナリングは、ファイルシステムで発生したメタデータの変更のトランザクション記録を保持します。

システムクラッシュ、電源障害、またはその他の不完全なアンマウントが発生した場合、XFS はジャーナル (ログとも呼ばれる) を使用してファイルシステムを復旧します。カーネルは XFS ファイルシステムをマウントするときにジャーナルの復旧を実行します。

破損
この文脈での **破損** は、次のような原因によるファイルシステムのエラーを意味します。

- ハードウェア障害
- ストレージファームウェア、デバイスドライバー、ソフトウェアスタック、またはファイルシステム自体のバグ
- ファイルシステムの一部が、ファイルシステム外の何かにより上書きされる問題

XFS は、ファイルシステムまたはファイルシステムメタデータの破損を検出すると、ファイルシステムをシャットダウンして、システムログにインシデントを報告することがあります。**/var** ディレクトリーが置かれているファイルシステムで破損が発生すると、このログは再起動後に利用できなくなります。

例15.1 XFS の破損を報告するシステムログエントリー

```
# dmesg --notime | tail -15

XFS (loop0): Mounting V5 Filesystem
XFS (loop0): Metadata CRC error detected at xfs_agi_read_verify+0xcb/0xf0 [xfs], xfs_agi block 0x2
XFS (loop0): Unmount and run xfs_repair
XFS (loop0): First 128 bytes of corrupted metadata buffer:
00000000027b3b56: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000005f9abc7a: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000005b0aef35: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000da9d2ded: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000001e265b07: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
000000006a40df69: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000000b272907: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000e484aac5: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
XFS (loop0): metadata I/O error in "xfs_trans_read_buf_map" at daddr 0x2 len 1 error 74
XFS (loop0): xfs_imap_lookup: xfs_ialloc_read_agi() returned error -117, agno 0
XFS (loop0): Failed to read root inode 0x80, error 11
```

ユーザー空間ユーティリティーは通常、破損した XFS ファイルシステムにアクセスしようとすると **Input/output error** メッセージを報告します。破損したログを使用して XFS ファイルシステムをマウントすると、マウントに失敗し、次のエラーメッセージが表示されます。

```
mount: /mount-point: mount(2) system call failed: Structure needs cleaning.
```

破損を修復するには、手動で **xfs_repair** ユーティリティーを使用する必要があります。

関連情報

- **xfs_repair(8)** man ページ

15.4. XFS_REPAIR で XFS ファイルシステムの検査

この手順では、**xfs_repair** ユーティリティーを使用して XFS ファイルシステムの読み取り専用の検査を実行します。破損を修復するには、手動で **xfs_repair** ユーティリティーを使用する必要があります。**xfs_repair** は、その他のファイルシステム修復ユーティリティーとは異なり、XFS ファイルシステムが正しくアンマウントされていなくても起動時には動作しません。不完全なアンマウントが発生した場合、XFS は単にマウント時にログを再生して、一貫したファイルシステムを確保します。**xfs_repair** は、最初にマウントし直さずに、ダーティーログがある XFS ファイルシステムを修復することはできません。

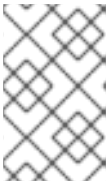
注記

xfsprogs パッケージには **fsck.xfs** バイナリーがありますが、これは、システムの起動時に **fsck.file** システムバイナリーを検索する **initscripts** を満たすためにのみ存在します。**fsck.xfs** は、すぐに終了コード 0 で終了します。

手順

1. ファイルシステムをマウントおよびアンマウントしてログを再生します。

```
# mount file-system
# umount file-system
```



注記

マウントが structure needs cleaning (構造のクリーニングが必要) エラーで失敗した場合は、ログが破損しているため再生できません。ドライランは、結果として、より多くのディスク上の破損を検出して報告する必要があります。

2. **xfs_repair** ユーティリティーを使用してドライランを実行し、ファイルシステムを検査します。エラーが表示され、ファイルシステムを変更せずに実行できるアクションが示されます。

```
# xfs_repair -n block-device
```

3. ファイルシステムをマウントします。

```
# mount file-system
```

関連情報

- **xfs_repair(8)** man ページ
- **xfs_metadump(8)** man ページ

15.5. XFS_REPAIR で XFS ファイルシステムの修復

この手順では、**xfs_repair** ユーティリティーを使用して破損した XFS ファイルシステムを修復します。

手順

1. **xfs_metadump** ユーティリティーを使用して、診断またはテストの目的で、修復する前にメタデータイメージを作成します。修復前のファイルシステムメタデータイメージは、破損がソフトウェアのバグによるものであるかどうかのサポート調査に役立ちます。修復前のイメージに含まれる破損のパターンは、根本的な分析に役に立つ場合があります。

   - **xfs_metadump** デバッグツールを使用して、XFS ファイルシステムからファイルにメタデータをコピーします。サポートに大きな **メタダンプ** ファイルを送信する必要がある場合は、標準の圧縮ユーティリティーを使用して生成された **メタダンプ** ファイルを圧縮してファイルサイズを縮小できます。

     ```
     # xfs_metadump block-device metadump-file
     ```

2. ファイルシステムを再マウントしてログを再生します。

```
# mount file-system
# umount file-system
```

3. アンマウントしたファイルシステムを修復するには、**xfs_repair** ユーティリティーを使用します。

   - マウントが成功した場合、追加のオプションは必要ありません。

     ```
     # xfs_repair block-device
     ```

   - マウントが **Structure needs cleaning** エラーで失敗した場合は、ログが破損しているため再生できません。ログを消去するには、**-L** オプション (**force log zeroing**) を使用します。

警告

このコマンドを実行すると、クラッシュ時に進行中だったすべてのメタデータの更新が失われます。これにより、ファイルシステムに重大な損傷やデータ損失が生じる可能性があります。これは、ログを再生できない場合に最後の手段としてのみ使用してください。

```
# xfs_repair -L block-device
```

4. ファイルシステムをマウントします。

```
# mount file-system
```

関連情報

- **xfs_repair(8)** man ページ

15.6. EXT2、EXT3、および EXT4 でエラー処理メカニズム

ext2、ext3、および ext4 のファイルシステムは、**e2fsck** ユーティリティーを使用して、ファイルシステムの検査と修復を実行します。ファイル名の **fsck.ext2**、**fsck.ext3**、および **fsck.ext4** は、**e2fsck** ユーティリティーへのハードリンクです。これらのバイナリーは、システムの起動時に自動的に実行し、その動作は確認されるファイルシステムと、そのファイルシステムの状態によって異なります。

完全なファイルシステムの検査および修復は、メタデータジャーナリングファイルシステムではない ext2 や、ジャーナルのない ext4 ファイルシステムに対して呼び出されます。

メタデータジャーナリング機能のある ext3 ファイルシステムおよび ext4 ファイルシステムの場合、ジャーナルはユーザー空間で再生され、ユーティリティーは終了します。これは、ジャーナルの再生によりクラッシュ後のファイルシステムの整合性が確保されるためのデフォルト動作になります。

このファイルシステムで、マウント中にメタデータの不整合が生じると、その事実がファイルシステムのスーパーブロックに記録されます。**e2fsck** が、このようなエラーでファイルシステムがマークされていることを検出すると、**e2fsck** はジャーナル (がある場合) の再生後にフルチェックを実行します。

関連情報

- **fsck(8)** man ページ
- **e2fsck(8)** man ページ

15.7. E2FSCK で EXT2、EXT3、または EXT4 ファイルシステムの検査

この手順では、**e2fsck** ユーティリティーを使用して、ext2 ファイルシステム、ext3 ファイルシステム、または ext4 ファイルシステムを検査します。

手順

1. ファイルシステムを再マウントしてログを再生します。

```
# mount file-system
# umount file-system
```

2. ドライランを実行して、ファイルシステムを検査します。

```
# e2fsck -n block-device
```

注記

エラーが表示され、ファイルシステムを変更せずに実行できるアクションが示されます。整合性チェック後のフェーズでは、修復モードで実行していた場合に前のフェーズで修正されていた不整合が検出される可能性があるため、追加のエラーが出力される場合があります。

関連情報

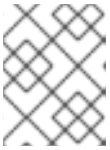
- **2eimage(8)** man ページ
- **e2fsck(8)** man ページ

15.8. E2FSCK で EXT2、EXT3、または EXT4 ファイルシステムの修復

この手順では、**e2fsck** ユーティリティーを使用して、破損した ext2、ext3、または ext 4 のファイルシステムを修復します。

手順

1. サポート調査のためにファイルシステムイメージを保存します。修復前のファイルシステムメタデータイメージは、破損がソフトウェアのバグによるものであるかどうかのサポート調査に役立ちます。修復前のイメージに含まれる破損のパターンは、根本的な分析に役に立つ場合があります。



注記

ファイルシステムが大幅に損傷している場合は、メタデータイメージの作成に関連して問題が発生する可能性があります。

- テスト目的でイメージを作成する場合は、**-r** オプションを指定して、ファイルシステム自体と同じサイズのスパースファイルを作成します。その後、**e2fsck** は作成されたファイルで直接操作できます。

```
# e2image -r block-device image-file
```

- 診断用にアーカイブまたは提供するイメージを作成する場合は、**-Q** オプションを使用して、転送に適したよりコンパクトなファイル形式を作成します。

```
# e2image -Q block-device image-file
```

2. ファイルシステムを再マウントしてログを再生します。

```
# mount file-system
# umount file-system
```

3. ファイルシステムを自動的に修復します。ユーザーの介入が必要な場合は、**e2fsck** が出力の未修正の問題を示し、このステータスを終了コードに反映させます。

```
# e2fsck -p block-device
```

関連情報

- **2eimage(8)** man ページ
- **e2fsck(8)** man ページ

第16章 ファイルシステムのマウント

システム管理者は、システムにファイルシステムをマウントすると、ファイルシステムのデータにアクセスできます。

16.1. LINUX のマウントメカニズム

本セクションでは、Linux でのファイルシステムのマウントに関する基本概念を説明します。

Linux、UNIX、および類似のオペレーティングシステムでは、さまざまなパーティションおよびリムーバブルデバイス (CD、DVD、USB フラッシュドライブなど) にあるファイルシステムをディレクトリーツリーの特定のポイント (マウントポイント) に接続して、再度切り離すことができます。ファイルシステムがディレクトリーにマウントされている間は、そのディレクトリーの元の内容にアクセスすることはできません。

Linux では、ファイルシステムがすでに接続されているディレクトリーにファイルシステムをマウントできます。

マウント時には、次の方法でデバイスを識別できます。

- UUID (universally unique identifier): **UUID=34795a28-ca6d-4fd8-a347-73671d0c19cb** など
- ボリュームラベル - **LABEL=home** など
- 非永続的なブロックデバイスへのフルパス - **/dev/sda3** など

デバイス名、目的のディレクトリー、ファイルシステムタイプなど、必要な情報をすべて指定せずに **mount** コマンドを使用してファイルシステムをマウントすると、**mount** ユーティリティーは **/etc/fstab** ファイルの内容を読み取り、指定のファイルシステムが記載されているかどうかを確認します。**/etc/fstab** ファイルには、選択したファイルシステムがマウントされるデバイス名およびディレクトリーのリスト、ファイルシステムタイプ、およびマウントオプションが含まれます。そのため、**/etc/fstab** で指定されたファイルシステムをマウントする場合は、以下のコマンド構文で十分です。

- マウントポイントによるマウント:

```
# mount directory
```

- ブロックデバイスによるマウント:

```
# mount device
```

関連情報

- **mount(8)** の man ページ
- UUID などの永続的な命名属性のリストを表示する方法。

16.2. 現在マウントされているファイルシステムのリスト表示

この手順では、コマンドラインに、現在マウントされているファイルシステムのリストを表示する方法を説明します。

手順

- マウントされているファイルシステムのリストを表示するには、**findmnt** ユーティリティーを使用します。

```
$ findmnt
```

- リスト表示されているファイルシステムを、特定のファイルシステムタイプに制限するには、**--types** オプションを追加します。

```
$ findmnt --types fs-type
```

以下に例を示します。

例16.1 XFS ファイルシステムのみを表示

```
$ findmnt --types xfs

TARGET  SOURCE                                          FSTYPE OPTIONS
/       /dev/mapper/luks-5564ed00-6aac-4406-bfb4-c59bf5de48b5 xfs    rw,relatime
├─/boot /dev/sda1                                       xfs    rw,relatime
└─/home /dev/mapper/luks-9d185660-7537-414d-b727-d92ea036051e xfs    rw,relatime
```

関連情報

- **findmnt(8)** man ページ

16.3. MOUNT でファイルシステムのマウント

この手順では、**mount** ユーティリティーを使用してファイルシステムをマウントする方法を説明します。

前提条件

- 選択したマウントポイントにファイルシステムがマウントされていない。

```
$ findmnt mount-point
```

手順

1. 特定のファイルシステムを添付する場合は、**mount** ユーティリティーを使用します。

```
# mount device mount-point
```

例16.2 XFS ファイルシステムのマウント

たとえば、UUID により識別されるローカル XFS ファイルシステムをマウントするには、次のコマンドを実行します。

```
# mount UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b /mnt/data
```

2. **mount** がファイルシステムタイプを自動的に認識できない場合は、**--types** オプションで指定します。

```
# mount --types type device mount-point
```

例16.3 NFS ファイルシステムのマウント

たとえば、リモートの NFS ファイルシステムをマウントするには、次のコマンドを実行します。

```
# mount --types nfs4 host:/remote-export /mnt/nfs
```

関連情報

- **mount(8)** の man ページ

16.4. マウントポイントの移動

この手順では、マウントされたファイルシステムのマウントポイントを、別のディレクトリーに変更する方法を説明します。

手順

1. ファイルシステムがマウントされているディレクトリーを変更するには、以下のコマンドを実行します。

```
# mount --move old-directory new-directory
```

例16.4 ホームファイルシステムの移動

たとえば、**/mnt/userdirs/** ディレクトリーにマウントされたファイルシステムを **/home/** マウントポイントに移動するには、以下のコマンドを実行します。

```
# mount --move /mnt/userdirs /home
```

2. ファイルシステムが想定どおりに移動したことを確認します。

```
$ findmnt
$ ls old-directory
$ ls new-directory
```

関連情報

- **mount(8)** の man ページ

16.5. UMOUNT でファイルシステムのアンマウント

この手順では、**umount** ユーティリティーを使用してファイルシステムをアンマウントする方法を説明します。

手順

1. 次のいずれかのコマンドを使用してファイルシステムをアンマウントします。
   - マウントポイントで行う場合は、以下のコマンドを実行します。

     ```
     # umount mount-point
     ```

   - デバイスで行う場合は、以下のコマンドを実行します。

     ```
     # umount device
     ```

   コマンドが次のようなエラーで失敗した場合は、プロセスがリソースを使用しているため、ファイルシステムが使用中であることを意味します。

   ```
   umount: /run/media/user/FlashDrive: target is busy.
   ```

2. ファイルシステムが使用中の場合は、**fuser** ユーティリティーを使用して、ファイルシステムにアクセスしているプロセスを特定します。以下に例を示します。

   ```
   $ fuser --mount /run/media/user/FlashDrive

   /run/media/user/FlashDrive: 18351
   ```

   その後、ファイルシステムを使用してプロセスを終了し、マウント解除を再度試みます。

16.6. 一般的なマウントオプション

次の表に、**mount** ユーティリティーの最も一般的なオプションを示します。次の構文を使用して、これらのマウントオプションを適用できます。

```
# mount --options option1,option2,option3 device mount-point
```

表16.1 一般的なマウントオプション

オプション	説明
async	ファイルシステムで非同期の入出力を可能にします。
auto	**mount -a** コマンドを使用したファイルシステムの自動マウントを可能にします。
defaults	オプション **async、auto、dev、exec、nouser、rw、suid** のエイリアスを指定します。
exec	特定のファイルシステムでのバイナリーファイルの実行を許可します。
loop	イメージをループデバイスとしてマウントします。
noauto	デフォルトでは、**mount -a** コマンドを使用したファイルシステムの自動マウントを無効します。

オプション	説明
noexec	特定のファイルシステムでのバイナリーファイルの実行は許可しません。
nouser	普通のユーザー (つまり root 以外のユーザー) によるファイルシステムのマウントおよびアンマウントは許可しません。
remount	ファイルシステムがすでにマウントされている場合は再度マウントを行います。
ro	読み取り専用でファイルシステムをマウントします。
rw	ファイルシステムを読み取りと書き込み両方でマウントします。
user	普通のユーザー (つまり root 以外のユーザー) によるファイルシステムのマウントおよびアンマウントを許可します。

第17章 複数のマウントポイントでのマウント共有

システム管理者は、マウントポイントを複製して、複数のディレクトリーからファイルシステムにアクセスするようにできます。

17.1. 共有マウントのタイプ

使用できる共有マウントには複数のタイプがあります。共有マウントポイントの種類によって、マウントポイントに別のファイルシステムをマウントしたときに発生する内容がなります。共有マウントは、**共有サブツリー** 機能を使用して実装されます。

次のマウントタイプを使用できます。

プライベート

このタイプは、伝播イベントを受信または転送しません。
複製マウントポイントまたは元のマウントポイントのどちらかに別のファイルシステムをマウントしても、それは他方には反映されません。

shared

このタイプは、指定したマウントポイントの正確なレプリカを作成します。
マウントポイントが **shared** マウントとしてマークされている場合は、元のマウントポイント内のすべてのマウントが複製マウントポイントに反映されます (その逆も同様です)。

これは、root ファイルシステムのデフォルトのマウントタイプです。

slave

このタイプは、指定したマウントポイントの限定的な複製を作成します。
マウントポイントが **slave** マウントとしてマークされている場合は、元のマウントポイント内のすべてのマウントがそれに反映されますが、**slave** マウント内のマウントは元のマウントに反映されません。

unbindable

このタイプは、指定のマウントポイントの複製をまったく行いません。

関連情報

- Linux Weekly News の **Shared subtrees** 記事

17.2. プライベートマウントポイントの複製の作成

この手順では、マウントポイントをプライベートマウントとして複製します。複製後に、複製または元のマウントポイントにマウントするファイルシステムは、他方のマウントポイントには反映されません。

手順

1. 元のマウントポイントから仮想ファイルシステム (VFS) ノードを作成します。

```
# mount --bind original-dir original-dir
```

2. 元のマウントポイントをプライベートとしてマークします。

```
# mount --make-private original-dir
```

あるいは、選択したマウントポイントと、その下のすべてのマウントポイントのマウントタイプを変更するには、**--make-private** ではなく、**--make-rprivate** オプションを使用します。

3. 複製を作成します。

```
# mount --bind original-dir duplicate-dir
```

例17.1 プライベートマウントポイントとして /mnt に /media を複製

1. **/media** ディレクトリーから VFS ノードを作成します。

```
# mount --bind /media /media
```

2. **/media** ディレクトリーをプライベートとしてマークします。

```
# mount --make-private /media
```

3. そのコピーを **/mnt** に作成します。

```
# mount --bind /media /mnt
```

4. これで、**/media** と **/mnt** はコンテンツを共有してますが、**/media** 内のマウントはいずれも **/mnt** に現れていないことが確認できます。たとえば、CD-ROM ドライブに空でないメディアがあり、**/media/cdrom/** ディレクトリーが存在する場合は、以下のコマンドを実行します。

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
#
```

5. また、**/mnt** ディレクトリーにマウントされているファイルシステムが **/media** に反映されていないことを確認することもできます。たとえば、**/dev/sdc1** デバイスを使用する、空でない USB フラッシュドライブをプラグインしており、**/mnt/flashdisk/** ディレクトリーが存在する場合は、次のコマンドを実行します。

```
# mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
# ls /mnt/flashdisk
en-US  publican.cfg
```

関連情報

- **mount(8)** の man ページ

17.3. 共有マウントポイントの複製の作成

この手順では、マウントポイントを共有マウントとして複製します。複製後に、元のディレクトリーまたは複製にマウントしたファイルシステムは、他方のマウントポイントに常に反映されます。

手順

1. 元のマウントポイントから仮想ファイルシステム (VFS) ノードを作成します。

```
# mount --bind original-dir original-dir
```

2. 元のマウントポイントを共有としてマークします。

```
# mount --make-shared original-dir
```

あるいは、選択したマウントポイントとその下のすべてのマウントポイントのマウントタイプを変更する場合は、**--make-shared** ではなく、**--make-rshared** オプションを使用します。

3. 複製を作成します。

```
# mount --bind original-dir duplicate-dir
```

例17.2 共有マウントポイントとして /mnt に /media を複製

/media ディレクトリーと **/mnt** ディレクトリーが同じコンテンツを共有するようにするには、次の手順を行います。

1. **/media** ディレクトリーから VFS ノードを作成します。

```
# mount --bind /media /media
```

2. **/media** ディレクトリーを共有としてマークします。

```
# mount --make-shared /media
```

3. そのコピーを **/mnt** に作成します。

```
# mount --bind /media /mnt
```

4. これで、**/media** 内のマウントが **/mnt** にも現れていることを確認できます。たとえば、CD-ROM ドライブに空でないメディアがあり、**/media/cdrom/** ディレクトリーが存在する場合は、以下のコマンドを実行します。

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

5. 同様に、**/mnt** ディレクトリー内にマウントされているファイルシステムが **/media** に反映されていることを確認することもできます。たとえば、**/dev/sdc1** デバイスを使用する、空でない USB フラッシュドライブをプラグインしており、**/mnt/flashdisk/** ディレクトリーが存在する場合は、次のコマンドを実行します。

```
# mount /dev/sdc1 /mnt/flashdisk
```

```
# ls /media/flashdisk
en-US  publican.cfg
# ls /mnt/flashdisk
en-US  publican.cfg
```

関連情報

- **mount(8)** の man ページ

17.4. スレーブマウントポイントの複製の作成

この手順では、マウントポイントを **slave** マウントタイプとして複製します。複製後に、元のマウントポイントにマウントしたファイルシステムは複製に反映されますが、その逆は反映されません。

手順

1. 元のマウントポイントから仮想ファイルシステム (VFS) ノードを作成します。

```
# mount --bind original-dir original-dir
```

2. 元のマウントポイントを共有としてマークします。

```
# mount --make-shared original-dir
```

あるいは、選択したマウントポイントとその下のすべてのマウントポイントのマウントタイプを変更する場合は、**--make-shared** ではなく、**--make-rshared** オプションを使用します。

3. 複製を作成し、これを **slave** タイプとしてマークします。

```
# mount --bind original-dir duplicate-dir
# mount --make-slave duplicate-dir
```

例17.3 スレーブマウントポイントとして /mnt に /media を複製

この例は、**/media** ディレクトリーのコンテンツが **/mnt** にも表示され、**/mnt** ディレクトリーのマウントが **/media** に反映されないようにする方法を示しています。

1. **/media** ディレクトリーから VFS ノードを作成します。

```
# mount --bind /media /media
```

2. **/media** ディレクトリーを共有としてマークします。

```
# mount --make-shared /media
```

3. その複製を **/mnt** に作成し、**slave** としてマークします。

```
# mount --bind /media /mnt
# mount --make-slave /mnt
```

4. **/media** 内のマウントが **/mnt** にも表示されていることを確認します。たとえば、CD-ROM ドライブに空でないメディアがあり、**/media/cdrom/** ディレクトリーが存在する場合は、以下のコマンドを実行します。

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

5. また、**/mnt** ディレクトリー内にマウントされているファイルシステムが **/media** に反映されていないことを確認します。たとえば、**/dev/sdc1** デバイスを使用する、空でない USB フラッシュドライブをプラグインしており、**/mnt/flashdisk/** ディレクトリーが存在する場合は、次のコマンドを実行します。

```
# mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
# ls /mnt/flashdisk
en-US  publican.cfg
```

関連情報

- **mount(8)** の man ページ

17.5. マウントポイントが複製されないようにする

この手順では、別のマウントポイントに複製されないように、マウントポイントをバインド不可としてマークします。

手順

- マウントポイントのタイプをバインド不可なマウントに変更するには、以下のコマンドを使用します。

```
# mount --bind mount-point mount-point
# mount --make-unbindable mount-point
```

あるいは、選択したマウントポイントとその下のすべてのマウントポイントのマウントタイプを変更する場合は、**--make-unbindable** の代わりに、**--make-runbindable** オプションを使用します。

これ以降、このマウントの複製を作成しようとすると、以下のエラーが出て失敗します。

```
# mount --bind mount-point duplicate-dir

mount: wrong fs type, bad option, bad superblock on mount-point,
missing codepage or helper program, or other error
In some cases useful info is found in syslog - try
dmesg | tail  or so
```

例17.4 /media が複製されないようにする

- **/media** ディレクトリーが共有されないようにするには、以下のコマンドを実行します。

```
# mount --bind /media /media
# mount --make-unbindable /media
```

関連情報

- **mount(8)** の man ページ

第18章 ファイルシステムの永続的なマウント

システム管理者は、ファイルシステムを永続的にマウントして、非リムーバブルストレージを設定できます。

18.1. /ETC/FSTAB ファイル

/etc/fstab 設定ファイルを使用して、ファイルシステムの永続的なマウントポイントを制御します。**/etc/fstab** ファイルの各行は、ファイルシステムのマウントポイントを定義します。

空白で区切られた 6 つのフィールドが含まれています。

1. **/dev** ディレクトリーの永続的な属性またはパスで識別されるブロックデバイス。
2. デバイスがマウントされるディレクトリー。
3. デバイス上のファイルシステム。
4. ファイルシステムのマウントオプション。これには、ブート時にデフォルトオプションでパーティションをマウントする **defaults** オプションが含まれます。マウントオプションフィールドは、**x-systemd.option** 形式の **systemd** マウントユニットオプションも認識します。
5. **dump** ユーティリティーのオプションのバックアップを作成します。
6. **fsck** ユーティリティーの順序を確認します。

注記

systemd-fstab-generator は、エントリーを **/etc/fstab** ファイルから **systemd-mount** ユニットに動的に変換します。**systemd-mount** ユニットがマスクされていない限り、**systemd** は手動アクティベーション中に **/etc/fstab** から LVM ボリュームを自動マウントします。

例18.1 /etc/fstab の /boot ファイルシステム

ブロックデバイス	マウントポイント	ファイルシステム	オプション	バックアップ	チェック
UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b	**/boot**	**xfs**	**defaults**	**0**	**0**

systemd サービスは、**/etc/fstab** のエントリーからマウントユニットを自動的に生成します。

関連情報

- **fstab(5)** および **systemd.mount(5)** の man ページ

18.2. /ETC/FSTAB へのファイルシステムの追加

この手順では、**/etc/fstab** 設定ファイルでファイルシステムの永続マウントポイントを設定する方法を説明します。

手順

1. ファイルシステムの UUID 属性を調べます。

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

以下に例を示します。

例18.2 パーティションの UUID の表示

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

NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs    Boot  ea74bbec-536d-490c-b8d9-5b40bbd7545b /boot
```

2. このマウントポイントのディレクトリーがない場合は、作成します。

```
# mkdir --parents mount-point
```

3. root で **/etc/fstab** ファイルを編集し、ファイルシステムに行を追加します (UUID で識別されます)。
以下に例を示します。

例18.3 /etc/fstab の /boot マウントポイント

```
UUID=ea74bbec-536d-490c-b8d9-5b40bbd7545b /boot xfs defaults 0 0
```

4. 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
```

5. ファイルシステムをマウントして、設定が機能することを確認します。

```
# mount mount-point
```

関連情報

- 永続的な命名属性の概要

第19章 オンデマンドでのファイルシステムのマウント

システム管理者は、NFS などのファイルシステムをオンデマンドで自動的にマウントするように設定できます。

19.1. AUTOFS サービス

本セクションでは、ファイルシステムをオンデマンドでマウントするのに使用する **autofs** サービスの利点と基本概念を説明します。

/etc/fstab 設定を使用した永続的なマウントの欠点の 1 つは、マウントされたファイルシステムにユーザーがアクセスする頻度に関わらず、マウントされたファイルシステムを所定の場所で維持するために、システムがリソースを割り当てる必要があることです。これは、システムが一度に多数のシステムへの NFS マウントを維持している場合などに、システムのパフォーマンスに影響を与える可能性があります。

/etc/fstab に代わるのは、カーネルベースの **autofs** サービスの使用です。これは以下のコンポーネントで設定されています。

- ファイルシステムを実装するカーネルモジュール
- 他のすべての機能を実行するユーザー空間サービス

autofs サービスは、ファイルシステムの自動マウントおよび自動アンマウントが可能なため (オンデマンド)、システムのリソースを節約できます。このサービスは、NFS、AFS、SMBFS、CIFS、およびローカルなどのファイルシステムをマウントする場合にも使用できます。

関連情報

- man ページの **autofs(8)**

19.2. AUTOFS 設定ファイル

本セクションでは、**autofs** サービスで使用される設定ファイルの使用方法と構文を説明します。

マスターマップファイル

autofs サービスは、デフォルトの主要設定ファイルとして、**/etc/auto.master** (マスターマップ) を使用します。これは、**/etc/autofs.conf** 設定ファイルの **autofs** 設定を Name Service Switch (NSS) メカニズムとともに使用することで、対応している別のネットワークソースと名前を使用するように変更できます。

すべてのオンデマンドマウントポイントはマスターマップで設定する必要があります。マウントポイント、ホスト名、エクスポートされたディレクトリー、オプションはすべて、ホストごとに手動で設定するのではなく、一連のファイル (またはサポートされているその他のネットワークソース) で指定できます。

マスターマップファイルには、**autofs** により制御されるマウントポイントと、それに対応する設定ファイルまたは自動マウントマップと呼ばれるネットワークソースがリスト表示されます。マスターマップの形式は次のとおりです。

```
mount-point  map-name  options
```

この形式で使用されている変数を以下に示します。

mount-point

autofs マウントポイント (例: **/mnt/data/**) です。

map-file

マウントポイントのリストと、マウントポイントがマウントされるファイルシステムの場所が記載されているマップソースファイルです。

options

指定した場合に、エントリーにオプションが指定されていなければ、指定されたマップ内のすべてのエントリーに適用されます。

例19.1 /etc/auto.master ファイル

以下は **/etc/auto.master** ファイルのサンプル行です。

```
/mnt/data  /etc/auto.data
```

マップファイル

マップファイルは、個々のオンデマンドマウントポイントのプロパティーを設定します。

ディレクトリーが存在しない場合、自動マウント機能はディレクトリーを作成します。ディレクトリーが存在している状況で自動マウント機能が起動した場合は、自動マウント機能の終了時にディレクトリーが削除されることはありません。タイムアウトを指定した場合は、タイムアウト期間中ディレクトリーにアクセスしないと、ディレクトリーが自動的にアンマウントされます。

マップの一般的な形式は、マスターマップに似ています。ただし、マスターマップでは、オプションフィールドはエントリーの末尾ではなく、マウントポイントと場所の間に表示されます。

```
mount-point  options  location
```

この形式で使用されている変数を以下に示します。

mount-point

これは、**autofs** のマウントポイントを参照しています。これは 1 つのインダイレクトマウント用の 1 つのディレクトリー名にすることも、複数のダイレクトマウント用のマウントポイントの完全パスにすることもできます。ダイレクトマップとインダイレクトマップの各エントリーキー (**mount-point**) の後に空白で区切られたオフセットディレクトリー (/ で始まるサブディレクトリー名) が記載されます。これがマルチマウントエントリーと呼ばれるものです。

options

このオプションを指定すると、マスターマップエントリーのオプション (存在する場合) に追加されます。設定エントリーの **append_options** が **no** に設定されている場合は、マスターマップのオプションの代わりにこのオプションが使用されます。

location

ローカルファイルシステムのパス (Sun マップ形式のエスケープ文字 **:** が先頭に付き、マップ名が / で始まります)、NFS ファイルシステム、他の有効なファイルシステムの場所などのファイルシステムの場所を参照します。

例19.2 マップファイル

以下は、マップファイルのサンプルです (例: **/etc/auto.misc**)。

```
payroll  -fstype=nfs4  personnel:/exports/payroll
sales    -fstype=xfs   :/dev/hda4
```

マップファイルの最初の列は、**autofs** マウントポイント (**personnel** サーバーからの **sales** と **payroll**) を示しています。2 列目は、**autofs** マウントのオプションを示しています。3 列目はマウントのソースを示しています。

任意の設定に基づき、**autofs** マウントポイントは、**/home/payroll** と **/home/sales** になります。**-fstype=** オプションは多くの場合省略されており、ファイルシステムが NFS の場合は必要ありません。これには、システムのデフォルトが NFS マウント用の NFSv4 である場合の NFSv4 のマウントも含まれます。

与えられた設定を使用して、プロセスが **/home/payroll/2006/July.sxc** などのアンマウントされたディレクトリー **autofs** へのアクセスを要求すると、**autofs** サービスは自動的にディレクトリーをマウントします。

amd マップ形式

autofs サービスは、**amd** 形式のマップ設定も認識します。これは Red Hat Enterprise Linux から削除された、**am-utils** サービス用に書き込まれた既存の自動マウント機能の設定を再利用する場合に便利です。

ただし、Red Hat は、前述のセクションで説明した簡単な **autofs** 形式の使用を推奨しています。

関連情報

- **autofs(5)** man ページ
- **autofs.conf(5)** man ページ
- **auto.master(5)** man ページ
- **/usr/share/doc/autofs/README.amd-maps** ファイル

19.3. AUTOFS マウントポイントの設定

この手順では、**autofs** サービスを使用してオンデマンドマウントポイントを設定する方法を説明します。

前提条件

- **autofs** パッケージをインストールしている。

```
# yum install autofs
```

- **autofs** サービスを起動して有効にしている。

```
# systemctl enable --now autofs
```

手順

1. **/etc/auto.identifier** にあるオンデマンドマウントポイント用のマップファイルを作成します。**identifier** を、マウントポイントを識別する名前に置き換えます。

2. マップファイルで、autofs 設定ファイル の説明に従って、マウントポイント、オプション、および場所の各フィールドを入力します。

3. autofs 設定ファイル セクションの説明に従って、マップファイルをマスターマップファイルに登録します。

4. 設定の再読み込みを許可し、新しく設定した **autofs** マウントを管理できるようにします。

```
# systemctl reload autofs.service
```

5. オンデマンドディレクトリーのコンテンツへのアクセスを試みます。

```
# ls automounted-directory
```

19.4. AUTOFS サービスを使用した NFS サーバーユーザーのホームディレクトリーの自動マウント

この手順では、ユーザーのホームディレクトリーを自動的にマウントするように autofs サービスを設定する方法を説明します。

前提条件

- autofs パッケージがインストールされている。
- autofs サービスが有効で、実行している。

手順

1. ユーザーのホームディレクトリーをマウントする必要があるサーバーの **/etc/auto.master** ファイルを編集して、マップファイルのマウントポイントと場所を指定します。これを行うには、以下の行を **/etc/auto.master** ファイルに追加します。

```
/home /etc/auto.home
```

2. ユーザーのホームディレクトリーをマウントする必要があるサーバー上で、**/etc/auto.home** という名前のマップファイルを作成し、以下のパラメーターでファイルを編集します。

```
* -fstype=nfs,rw,sync host.example.com:/home/&
```

fstype パラメーターはデフォルトで **nfs** であるため、このパラメーターは飛ばして次に進むことができます。詳細は、**autofs(5)** man ページを参照してください。

3. **autofs** サービスを再読み込みします。

```
# systemctl reload autofs
```

19.5. AUTOFS サイトの設定ファイルの上書き/拡張

クライアントシステムの特定のマウントポイントで、サイトのデフォルトを上書きすることが役に立つ場合があります。

例19.3 初期条件

たとえば、次の条件を検討します。

- 自動マウント機能のマップが NIS に格納され、**/etc/nsswitch.conf** ファイルに次のようなディレクティブがある。

```
automount:    files nis
```

- **auto.master** ファイルに以下を含む。

```
+auto.master
```

- NIS の **auto.master** マップファイルに以下を含む。

```
/home auto.home
```

- NIS の **auto.home** マップには以下が含まれている。

```
beth    fileserver.example.com:/export/home/beth
joe     fileserver.example.com:/export/home/joe
*       fileserver.example.com:/export/home/&
```

- **autofs** 設定オプションの **BROWSE_MODE** は **yes** に設定されています。

```
BROWSE_MODE="yes"
```

- **/etc/auto.home** ファイルマップが存在しない。

手順

本セクションでは、別のサーバーからホームディレクトリーをマウントし、選択したエントリーのみで **auto.home** を強化する例を説明します。

例19.4 別のサーバーからのホームディレクトリーのマウント

上記の条件で、クライアントシステムが NIS マップの **auto.home** を上書きして、別のサーバーからホームディレクトリーをマウントする必要があるとします。

- この場合、クライアントは次の **/etc/auto.master** マップを使用する必要があります。

```
/home /etc/auto.home
+auto.master
```

- **/etc/auto.home** マップにエントリーが含まれています。

```
*    host.example.com:/export/home/&
```

自動マウント機能は最初に出現したマウントポイントのみを処理するため、**/home** ディレクトリーには NIS **auto.home** マップではなく、**/etc/auto.home** の内容が含まれます。

例19.5 選択されたエントリーのみを使用した auto.home の拡張

別の方法として、サイト全体の **auto.home** マップを少しのエントリーを使用して拡張するには、次の手順を行います。

1. **/etc/auto.home** ファイルマップを作成し、そこに新しいエントリーを追加します。最後に、NIS の **auto.home** マップを含めます。これにより、**/etc/auto.home** ファイルマップは次のようになります。

```
mydir someserver:/export/mydir
+auto.home
```

2. この NIS の **auto.home** マップ条件で、**/home** ディレクトリーの出力内容をリスト表示すると次のようになります。

```
$ ls /home

beth joe mydir
```

autofs は、読み取り中のファイルマップと同じ名前のファイルマップの内容を組み込まないため、上記の例は期待どおりに動作します。このように、**autofs** は、**nsswitch** 設定内の次のマップソースに移動します。

19.6. LDAP で自動マウント機能マップの格納

この手順では、**autofs** マップファイルではなく、LDAP 設定で自動マウント機能マップを格納するように **autofs** を設定します。

前提条件

- LDAP から自動マウント機能マップを取得するように設定されているすべてのシステムに、LDAP クライアントライブラリーをインストールする必要があります。Red Hat Enterprise Linux では、**openldap** パッケージは、**autofs** パッケージの依存関係として自動的にインストールされます。

手順

1. LDAP アクセスを設定するには、**/etc/openldap/ldap.conf** ファイルを変更します。**BASE**、**URI**、**schema** の各オプションがサイトに適切に設定されていることを確認します。

2. 自動マウント機能マップを LDAP に格納するためにデフォルトされた最新のスキーマが、**rfc2307bis** ドラフトに記載されています。このスキーマを使用する場合は、スキーマの定義のコメント文字を取り除き、**/etc/autofs.conf** 設定ファイル内に設定する必要があります。以下に例を示します。

例19.6 autofs の設定

```
DEFAULT_MAP_OBJECT_CLASS="automountMap"
DEFAULT_ENTRY_OBJECT_CLASS="automount"
DEFAULT_MAP_ATTRIBUTE="automountMapName"
DEFAULT_ENTRY_ATTRIBUTE="automountKey"
DEFAULT_VALUE_ATTRIBUTE="automountInformation"
```

3. 他のすべてのスキーマエントリーが設定内でコメントされていることを確認してください。**rfc2307bis** スキーマの **automountKey** 属性は、**rfc2307** スキーマの **cn** 属性に置き換わります。以下は、LDAP データ交換形式 (LDIF) 設定の例です。

例19.7 LDIF 設定

```
# auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: top
objectClass: automountMap
automountMapName: auto.master

# /home, auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: automount
automountKey: /home
automountInformation: auto.home

# auto.home, example.com
dn: automountMapName=auto.home,dc=example,dc=com
objectClass: automountMap
automountMapName: auto.home

# foo, auto.home, example.com
dn: automountKey=foo,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: foo
automountInformation: filer.example.com:/export/foo

# /, auto.home, example.com
dn: automountKey=/,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: /
automountInformation: filer.example.com:/export/&
```

関連情報

- **rfc2307bis** ドラフト

19.7. SYSTEMD.AUTOMOUNT を使用して、/ETC/FSTAB を使用してオンデマンドでファイルシステムをマウントします

この手順は、マウントポイントが **/etc/fstab** で定義されている場合に、automount systemd ユニットを使用してオンデマンドでファイルシステムをマウントする方法を示しています。マウントごとに自動マウントユニットを追加して有効にする必要があります。

手順

1. ファイルシステムの永続的なマウント の説明に従って、目的の fstab エントリーを追加します。以下に例を示します。

```
/dev/disk/by-id/da875760-edb9-4b82-99dc-5f4b1ff2e5f4  /mount/point  xfs  defaults  0 0
```

2. 前の手順で作成したエントリーの options フィールドに **x-systemd.automount** を追加します。
3. システムが新しい設定を登録するように、新しく作成されたユニットをロードします。

```
# systemctl daemon-reload
```

4. 自動マウントユニットを起動します。

```
# systemctl start mount-point.automount
```

検証

1. **mount-point.automount** が実行されていることを確認します。

```
# systemctl status mount-point.automount
```

2. 自動マウントされたディレクトリーに目的のコンテンツが含まれていることを確認します。

```
# ls /mount/point
```

関連情報

- **systemd.automount(5)** man ページ
- **systemd.mount(5)** man ページ
- systemd の管理

19.8. SYSTEMD.AUTOMOUNT を使用して、マウントユニットを使用してファイルシステムをオンデマンドでマウントします

この手順は、マウントポイントがマウントユニットによって定義されている場合に、automount systemd ユニットを使用してオンデマンドでファイルシステムをマウントする方法を示しています。マウントごとに自動マウントユニットを追加して有効にする必要があります。

手順

1. マウントユニットを作成します。以下に例を示します。

```
mount-point.mount
[Mount]
What=/dev/disk/by-uuid/f5755511-a714-44c1-a123-cfde0e4ac688
Where=/mount/point
Type=xfs
```

2. マウントユニットと同じ名前で、拡張子が **.automount** のユニットファイルを作成します。
3. ファイルを開き、**[Automount]** セクションを作成します。**Where=** オプションをマウントパスに設定します。

```
[Automount]
Where=/mount/point
```

```
[Install]
WantedBy=multi-user.target
```

4. システムが新しい設定を登録するように、新しく作成されたユニットをロードします。

```
# systemctl daemon-reload
```

5. 代わりに、自動マウントユニットを有効にして起動します。

```
# systemctl enable --now mount-point.automount
```

検証

1. **mount-point.automount** が実行されていることを確認します。

```
# systemctl status mount-point.automount
```

2. 自動マウントされたディレクトリーに目的のコンテンツが含まれていることを確認します。

```
# ls /mount/point
```

関連情報

- **systemd.automount(5)** man ページ
- **systemd.mount(5)** man ページ
- systemd の管理

第20章 IDM からの SSSD コンポーネントを使用した AUTOFS マップのキャッシュ

システムセキュリティーサービスデーモン (System Security Services Daemon: SSSD) は、リモートサービスディレクトリーと認証メカニズムにアクセスするシステムサービスです。データキャッシュは、ネットワーク接続が遅い場合に役立ちます。SSSD サービスが autofs マップをキャッシュするように設定するには、本セクションの以下の手順に従います。

20.1. IDM サーバーを LDAP サーバーとして使用するように AUTOFS を手動で設定する

この手順では、IdM サーバーを LDAP サーバーとして使用するように **autofs** を設定する方法を説明します。

手順

1. **/etc/autofs.conf** ファイルを編集し、**autofs** が検索するスキーマ属性を指定します。

```
#
# Other common LDAP naming
#
map_object_class = "automountMap"
entry_object_class = "automount"
map_attribute = "automountMapName"
entry_attribute = "automountKey"
value_attribute = "automountInformation"
```

注記

ユーザーは、**/etc/autofs.conf** ファイルに小文字と大文字の両方で属性を書き込むことができます。

2. オプションで、LDAP 設定を指定します。これには 2 通りの方法があります。最も簡単な方法は、自動マウントサービスが LDAP サーバーと場所を自分で発見するようにすることです。

```
ldap_uri = "ldap:///dc=example,dc=com"
```

このオプションでは、DNS に検出可能なサーバーの SRV レコードが含まれている必要があります。

別の方法では、使用する LDAP サーバーと LDAP 検索のベース DN を明示的に設定します。

```
ldap_uri = "ldap://ipa.example.com"
search_base = "cn=location,cn=automount,dc=example,dc=com"
```

3. autofs が IdM LDAP サーバーによるクライアント認証を許可するように **/etc/autofs_ldap_auth.conf** ファイルを編集します。

- **authrequired** を yes に変更します。

- プリンシパルを IdM LDAP サーバー (**host/fqdn@REALM**) の Kerberos ホストプリンシパルに設定します。プリンシパル名は、GSS クライアント認証の一部として IdM ディレクトリーへの接続に使用されます。

```
<autofs_ldap_sasl_conf
    usetls="no"
    tlsrequired="no"
    authrequired="yes"
    authtype="GSSAPI"
    clientprinc="host/server.example.com@EXAMPLE.COM"
    />
```

ホストプリンシパルの詳細は、IdM での正規化された DNS ホスト名の使用 を参照してください。

必要に応じて **klist -k** を実行して、正確なホストプリンシパル情報を取得します。

20.2. AUTOFS マップをキャッシュする SSSD の設定

SSSD サービスを使用すると、IdM サーバーに保存されている **autofs** マップを、IdM サーバーを使用するように **autofs** を設定することなくキャッシュできます。

前提条件

- **sssd** パッケージがインストールされている。

手順

1. SSSD 設定ファイルを開きます。

```
# vim /etc/sssd/sssd.conf
```

2. SSSD が処理するサービスリストに **autofs** サービスを追加します。

```
[sssd]
domains = ldap
services = nss,pam,autofs
```

3. **[autofs]** セクションを新規作成します。**autofs** サービスのデフォルト設定はほとんどのインフラストラクチャーに対応するため、これを空白のままにすることができます。

```
[nss]

[pam]

[sudo]

[autofs]

[ssh]

[pac]
```

詳細は man ページの **sssd.conf** を参照してください。

4. オプションとして、**autofs** エントリーの検索ベースを設定します。デフォルトでは、これは LDAP 検索ベースですが、**ldap_autofs_search_base** パラメーターでサブツリーを指定できます。

```
[domain/EXAMPLE]

ldap_search_base = "dc=example,dc=com"
ldap_autofs_search_base = "ou=automount,dc=example,dc=com"
```

5. SSSD サービスを再起動します。

```
# systemctl restart sssd.service
```

6. SSSD が自動マウント設定のソースとしてリスト表示されるように、**/etc/nsswitch.conf** ファイルを確認します。

```
automount: sss files
```

7. **autofs** サービスを再起動します。

```
# systemctl restart autofs.service
```

8. **/home** のマスターマップエントリーがあると想定し、ユーザーの **/home** ディレクトリーをリスト表示して設定をテストします。

```
# ls /home/userName
```

リモートファイルシステムをマウントしない場合は、**/var/log/messages** ファイルでエラーを確認します。必要に応じて、**logging** パラメーターを **debug** に設定して、**/etc/sysconfig/autofs** ファイルのデバッグレベルを増やします。

第21章 ROOT ファイルシステムに対する読み取り専用パーミッションの設定

場合によっては、root ファイルシステム (/) を読み取り専用パーミッションでマウントする必要があります。ユースケースの例には、システムの予期せぬ電源切断後に行うセキュリティーの向上またはデータ整合性の保持が含まれます。

21.1. 書き込みパーミッションを保持するファイルおよびディレクトリー

システムが正しく機能するためには、一部のファイルやディレクトリーで書き込みパーミッションが必要とされます。root ファイルシステムが読み取り専用モードでマウントされると、このようなファイルは、**tmpfs** 一時ファイルシステムを使用して RAM にマウントされます。

このようなファイルおよびディレクトリーのデフォルトセットは、**/etc/rwtab** ファイルから読み込まれます。このファイルをシステムに存在させるには、**readonly-root** パッケージが必要であることに注意してください。

```
dirs /var/cache/man
dirs /var/gdm
<content truncated>

empty /tmp
empty /var/cache/foomatic
<content truncated>

files /etc/adjtime
files /etc/ntp.conf
<content truncated>
```

/etc/rwtab ファイルのエントリーは、以下の形式に従います。

```
copy-method    path
```

この構文で、以下のことを行います。

- **copy-method** を、ファイルまたはディレクトリーを tmpfs にコピーする方法を指定するキーワードの 1 つに置き換えます。
- **path** を、ファイルまたはディレクトリーへのパスに置き換えます。

/etc/rwtab ファイルは、ファイルまたはディレクトリーを **tmpfs** にコピーする方法として以下を認識します。

empty

空のパスが **tmpfs** にコピーされます。以下に例を示します。

```
empty /tmp
```

dirs

ディレクトリーツリーが空の状態で **tmpfs** にコピーされます。以下に例を示します。

```
dirs /var/run
```

files

ファイルやディレクトリーツリーはそのまま **tmpfs** にコピーされます。以下に例を示します。

```
files /etc/resolv.conf
```

カスタムパスを **/etc/rwtab.d/** に追加する場合も同じ形式が適用されます。

21.2. ブート時に読み取り専用パーミッションでマウントするように ROOT ファイルシステムの設定

この手順を行うと、今後システムが起動するたびに、root ファイルシステムが読み取り専用としてマウントされます。

手順

1. **/etc/sysconfig/readonly-root** ファイルで、**READONLY** オプションを **yes** に設定して、ファイルシステムを読み取り専用としてマウントします。

```
READONLY=yes
```

2. **/etc/fstab** ファイルの root エントリー (/) に **ro** オプションを追加します。

```
/dev/mapper/luks-c376919e... /  xfs  x-systemd.device-timeout=0,ro  1  1
```

3. **ro** kernel オプションを有効にします。

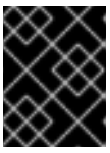
```
# grubby --update-kernel=ALL --args="ro"
```

4. **rw** カーネルオプションが無効になっていることを確認します。

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

5. **tmpfs** ファイルシステムに書き込みパーミッションでマウントするファイルとディレクトリーを追加する必要がある場合は、**/etc/rwtab.d/** ディレクトリーにテキストファイルを作成し、そこに設定を置きます。
たとえば、**/etc/example/file** ファイルを書き込みパーミッションでマウントするには、この行を **/etc/rwtab.d/example** ファイルに追加します。

```
files /etc/example/file
```



重要

tmpfs のファイルおよびディレクトリーの変更内容は、再起動後は持続しません。

6. システムを再起動して変更を適用します。

トラブルシューティング

- 誤って読み取り専用パーミッションで root ファイルシステムをマウントした場合は、次のコマンドを使用して、読み書きパーミッションで再度マウントできます。

```
# mount -o remount,rw /
```

第22章 クォータを使用した XFS でのストレージ領域の使用の制限

ディスククォータを実装して、ユーザーまたはグループに利用可能なディスク領域のサイズを制限できます。ユーザーがディスク領域を過剰に消費したり、パーティションが満杯になる前に、システム管理者に通知を出す警告レベルを定義することもできます。

XFS クォータサブシステムは、ディスク領域 (ブロック) およびファイル (inode) の使用量の制限を管理します。XFS クォータは、ユーザー、グループ、ディレクトリーレベル、またはプロジェクトレベルでこれらの項目の使用を制御または報告します。グループおよびプロジェクトのクォータは、古いデフォルト以外の XFS ディスクフォーマットでのみ相互に排他的です。

ディレクトリーまたはプロジェクトごとに管理する場合、XFS は特定のプロジェクトに関連付けられたディレクトリー階層のディスク使用量を管理します。

22.1. ディスククォータ

ほとんどのコンピューティング環境では、ディスク領域は無限ではありません。クォータサブシステムは、ディスク領域の使用量を制御するメカニズムを提供します。

ディスククォータは、ローカルファイルシステムの個々のユーザーおよびユーザーグループに設定できます。これにより、ユーザー固有のファイル (電子メールなど) に割り当てられる領域を、ユーザーが作業するプロジェクトに割り当てられた領域とは別に管理できます。クォータサブシステムは、割り当てられた制限を超えるとユーザーに警告しますが、現在の作業に追加領域を許可します (ハード制限/ソフト制限)。

クォータが実装されている場合は、クォータを超過しているかどうかを確認して、クォータが正しいことを確認する必要があります。ユーザーが繰り返しクォータを超過するか、常にソフト制限に達している場合、システム管理者は、ユーザーが使用するディスク領域を減らすか、ユーザーのディスククォータを増やす方法を決定するのを助けることができます。

クォータは、以下を制御するように設定できます。

- 消費されるディスクブロックの数。
- UNIX ファイルシステムのファイルに関する情報を含むデータ構造である inode の数。inode はファイル関連の情報を保存するため、作成可能なファイルの数を制御できます。

22.2. XFS_QUOTA ツール

xfs_quota ツールを使用して、XFS ファイルシステム上のクォータを管理できます。さらに、有効なディスク使用量のアカウンティングシステムとして、制限の強制適用をオフにして XFS ファイルシステムを使用できます。

XFS クォータシステムは、他のファイルシステムとはさまざまな点で異なります。最も重要な点として、XFS はクォータ情報をファイルシステムのメタデータとみなし、ジャーナリングを使用して一貫性のより高いレベルの保証を提供します。

関連情報

- **xfs_quota(8)** man ページ。

22.3. XFS でのファイルシステムクォータ管理

XFS クォータサブシステムは、ディスク領域 (ブロック) およびファイル (inode) の使用量の制限を管理

します。XFS クォータは、ユーザー、グループ、ディレクトリーレベル、またはプロジェクトレベルでこれらの項目の使用を制御または報告します。グループおよびプロジェクトのクォータは、古いデフォルト以外の XFS ディスクフォーマットでのみ相互に排他的です。

ディレクトリーまたはプロジェクトごとに管理する場合、XFS は特定のプロジェクトに関連付けられたディレクトリー階層のディスク使用量を管理します。

22.4. XFS のディスククォータの有効化

この手順では、XFS ファイルシステムのユーザー、グループ、およびプロジェクトのディスククォータを有効にします。クォータを有効にすると、**xfs_quota** ツールを使用して制限を設定し、ディスク使用量を報告できます。

手順

1. ユーザーのクォータを有効にします。

```
# mount -o uquota /dev/xvdb1 /xfs
```

uquota を **uqnoenforce** に置き換えて、制限を強制適用せずに使用状況の報告を可能にします。

2. グループのクォータを有効にします。

```
# mount -o gquota /dev/xvdb1 /xfs
```

gquota を **gqnoenforce** に置き換えて、制限を強制適用せずに使用状況の報告を可能にします。

3. プロジェクトのクォータを有効にします。

```
# mount -o pquota /dev/xvdb1 /xfs
```

pquota を **pqnoenforce** に置き換え、制限を強制適用せずに使用状況の報告を可能にします。

4. または、**/etc/fstab** ファイルにクォータマウントオプションを追加します。以下の例は、XFS ファイルシステムでユーザー、グループ、およびプロジェクトのクォータを有効にする **/etc/fstab** ファイルのエントリーを示しています。以下の例では、読み取り/書き込みパーミッションでファイルシステムもマウントします。

```
# vim /etc/fstab
/dev/xvdb1    /xfs    xfs    rw,quota       0  0
/dev/xvdb1    /xfs    xfs    rw,gquota      0  0
/dev/xvdb1    /xfs    xfs    rw,prjquota    0  0
```

関連情報

- **mount(8)** man ページ。
- **xfs_quota(8)** man ページ。

22.5. XFS 使用量の報告

xfs_quota ツールを使用して制限を設定し、ディスク使用量を報告できます。**xfs_quota** は、デフォルトでは対話形式で基本モードで実行されます。基本モードのサブコマンドは使用量を報告するだけで、すべてのユーザーが使用できます。

前提条件

- XFS ファイルシステムに対してクォータが有効になっている。XFS のディスククォータの有効化 を参照してください。

手順

1. **xfs_quota** シェルを起動します。

```
# xfs_quota
```

2. 指定したユーザーの使用状況および制限を表示します。

```
# xfs_quota> quota username
```

3. ブロックおよび inode の空きおよび使用済みの数を表示します。

```
# xfs_quota> df
```

4. help コマンドを実行して、**xfs_quota** で利用可能な基本的なコマンドを表示します。

```
# xfs_quota> help
```

5. **q** を指定して **xfs_quota** を終了します。

```
# xfs_quota> q
```

関連情報

- **xfs_quota(8)** man ページ。

22.6. XFS クォータ制限の変更

-x オプションを指定して **xfs_quota** ツールを起動し、エキスパートモードを有効にして、クォータシステムを変更できる管理者コマンドを実行します。このモードのサブコマンドは、制限を実際に設定することができるため、昇格した特権を持つユーザーのみが利用できます。

前提条件

- XFS ファイルシステムに対してクォータが有効になっている。XFS のディスククォータの有効化 を参照してください。

手順

1. エキスパートモードを有効にするには、**-x** オプションを指定して **xfs_quota** シェルを起動します。

```
# xfs_quota -x
```

2. 特定のファイルシステムのクォータ情報を表示します。

```
# xfs_quota> report /path
```

たとえば、(/**dev**/**blockdevice** の) /**home** のクォータレポートのサンプルを表示するには、**report -h** /**home** コマンドを使用します。これにより、以下のような出力が表示されます。

```
User quota on /home (/dev/blockdevice)
Blocks
User ID      Used   Soft   Hard Warn/Grace
---------- ---------------------------------
root            0      0      0  00 [------]
testuser   103.4G      0      0  00 [------]
```

3. クォータの制限を変更します。

```
# xfs_quota> limit isoft=500m ihard=700m user /path
```

たとえば、ホームディレクトリーが /**home**/**john** のユーザー **john** に対して、inode 数のソフト制限およびハード制限をそれぞれ 500 と 700 に設定するには、次のコマンドを使用します。

```
# xfs_quota -x -c 'limit isoft=500 ihard=700 john' /home/
```

この場合は、マウントされた xfs ファイルシステムである **mount_point** を渡します。

4. help コマンドを実行して、**xfs_quota -x** で利用可能なエキスパートコマンドを表示します。

```
# xfs_quota> help
```

関連情報

- **xfs_quota(8)** man ページ。

22.7. XFS のプロジェクト制限の設定

以下の手順では、プロジェクトが制御するディレクトリーに制限を設定します。

手順

1. プロジェクトが制御するディレクトリーを /**etc**/**projects** に追加します。たとえば、以下は一意の ID が 11 の /**var**/**log** パスを /**etc**/**projects** に追加します。プロジェクト ID には、プロジェクトにマッピングされる任意の数値を指定できます。

```
# echo 11:/var/log >> /etc/projects
```

2. /**etc**/**projid** にプロジェクト名を追加して、プロジェクト ID をプロジェクト名にマップします。たとえば、以下は、前のステップで定義されたように **logfiles** というプロジェクトをプロジェクト ID 11 に関連付けます。

```
# echo logfiles:11 >> /etc/projid
```

3. プロジェクトのディレクトリーを初期化します。たとえば、以下はプロジェクトディレクトリー **/var** を初期化します。

```
# xfs_quota -x -c 'project -s logfiles' /var
```

4. 初期化したディレクトリーでプロジェクトのクォータを設定します。

```
# xfs_quota -x -c 'limit -p bhard=1g logfiles' /var
```

関連情報

- **xfs_quota(8)** man ページ。
- **projid(5)** man ページ。
- **projects(5)** man ページ。

第23章 クォータを使用した EXT4 でのストレージ領域の使用の制限

ディスククォータを割り当てる前に、システムでディスククォータを有効にする必要があります。ユーザーごと、グループごと、またはプロジェクトごとにディスククォータを割り当てることができます。ただし、ソフト制限が設定されている場合は、猶予期間として知られる設定可能な期間として、これらのクォータを超過できます。

23.1. クォータツールのインストール

ディスククォータを実装するには、RPM パッケージ **quota** をインストールする必要があります。

手順

- **quota** パッケージをインストールします。

```
# yum install quota
```

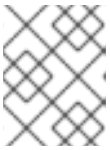
23.2. ファイルシステム作成でクォータ機能の有効化

この手順では、ファイルシステムの作成時にクォータを有効にする方法を説明します。

手順

1. ファイルシステムの作成時にクォータを有効にします。

```
# mkfs.ext4 -O quota /dev/sda
```



注記

デフォルトでは、ユーザーとグループのクォータのみが有効になり、初期化されます。

2. ファイルシステムの作成時にデフォルトを変更します。

```
# mkfs.ext4 -O quota -E quotatype=usrquota:grpquota:prjquota /dev/sda
```

3. ファイルシステムをマウントします。

```
# mount /dev/sda
```

関連情報

- **ext4(5)** man ページ。

23.3. 既存のファイルシステムでのクォータ機能の有効化

この手順では、**tune2fs** コマンドを使用して、既存のファイルシステムでクォータ機能を有効にする方法を説明します。

手順

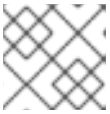
手順

1. ファイルシステムをアンマウントします。

```
# umount /dev/sda
```

2. 既存のファイルシステムでクォータを有効にします。

```
# tune2fs -O quota /dev/sda
```



注記

デフォルトでは、ユーザーとグループのクォータのみが初期化されます。

3. デフォルトを変更します。

```
# tune2fs -Q usrquota,grpquota,prjquota /dev/sda
```

4. ファイルシステムをマウントします。

```
# mount /dev/sda
```

関連情報

- **ext4(5)** man ページ。

23.4. クォータ強制適用の有効化

クォータアカウンティングは、追加のオプションを使用せずにファイルシステムをマウントした後にデフォルトで有効になりますが、クォータの強制適用は行いません。

前提条件

- クォータ機能が有効になり、デフォルトのクォータが初期化されます。

手順

- ユーザークォータに対して、**quotaon** によるクォータの強制適用を有効にします。

```
# mount /dev/sda /mnt
```

```
# quotaon /mnt
```

注記

クォータの強制適用は、マウントオプション **usrquota**、**grpquota**、または **prjquota** を使用して、マウント時に有効にできます。

```
# mount -o usrquota,grpquota,prjquota /dev/sda /mnt
```

- すべてのファイルシステムのユーザー、グループ、およびプロジェクトのクォータを有効にします。

```
# quotaon -vaugP
```

  - **-u** オプション、**-g** オプション、または **-P** オプションがいずれも指定されていないと、ユーザーのクォータのみが有効になります。
  - **-g** オプションのみを指定すると、グループのクォータのみが有効になります。
  - **-P** オプションのみを指定すると、プロジェクトのクォータのみが有効になります。

- /**home** などの特定のファイルシステムのクォータを有効にします。

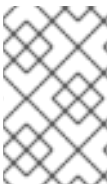
```
# quotaon -vugP /home
```

関連情報

- **quotaon(8)** man ページ。

23.5. ユーザーごとにクォータの割り当て

ディスククォータは、**edquota** コマンドでユーザーに割り当てられます。



注記

EDITOR 環境変数により定義されたテキストエディターは、**edquota** により使用されます。エディターを変更するには、~/**.bash_profile** ファイルの **EDITOR** 環境変数を、使用するエディターのフルパスに設定します。

前提条件

- ユーザーは、ユーザークォータを設定する前に存在する必要があります。

手順

1. ユーザーにクォータを割り当てます。

```
# edquota username
```

   username を、クォータを割り当てるユーザーに置き換えます。

   たとえば、/**dev**/**sda** パーティションのクォータを有効にし、**edquota testuser** コマンドを実行すると、システムに設定したデフォルトエディターに以下が表示されます。

```
Disk quotas for user testuser (uid 501):
Filesystem   blocks   soft   hard   inodes   soft   hard
/dev/sda      44043      0      0    37418      0      0
```

2. 必要な制限を変更します。
   いずれかの値が 0 に設定されていると、制限は設定されません。テキストエディターでこれらを変更します。

たとえば、以下は、testuser のソフトブロック制限とハードブロック制限をそれぞれ 50000 と 55000 に設定していることを示しています。

```
Disk quotas for user testuser (uid 501):
Filesystem   blocks   soft   hard   inodes   soft   hard
/dev/sda      44043  50000  55000    37418      0      0
```

- 最初の列は、クォータが有効になっているファイルシステムの名前です。
- 2 列目には、ユーザーが現在使用しているブロック数が示されます。
- その次の 2 列は、ファイルシステム上のユーザーのソフトブロック制限およびハードブロック制限を設定するのに使用されます。
- **inodes** 列には、ユーザーが現在使用している inode 数が表示されます。
- 最後の 2 列は、ファイルシステムのユーザーに対するソフトおよびハードの inode 制限を設定するのに使用されます。
  - ハードブロック制限は、ユーザーまたはグループが使用できる最大ディスク容量 (絶対値) です。この制限に達すると、それ以上のディスク領域は使用できなくなります。
  - ソフトブロック制限は、使用可能な最大ディスク容量を定義します。ただし、ハード制限とは異なり、ソフト制限は一定時間超過する可能性があります。この時間は **猶予期間** として知られています。猶予期間の単位は、秒、分、時間、日、週、または月で表されます。

検証手順

- ユーザーのクォータが設定されていることを確認します。

```
# quota -v testuser
Disk quotas for user testuser:
Filesystem  blocks  quota  limit  grace  files  quota  limit  grace
/dev/sda      1000*  1000   1000           0      0      0
```

23.6. グループごとにクォータの割り当て

グループごとにクォータを割り当てることができます。

前提条件

- グループは、グループクォータを設定する前に存在している必要があります。

手順

1. グループクォータを設定します。

```
# edquota -g groupname
```

たとえば、**devel** グループのグループクォータを設定するには、以下を実行します。

```
# edquota -g devel
```

このコマンドにより、グループの既存クォータがテキストエディターに表示されます。

```
Disk quotas for group devel (gid 505):
Filesystem   blocks  soft  hard  inodes  soft  hard
/dev/sda     440400     0     0   37418     0     0
```

2. 制限を変更し、ファイルを保存します。

検証手順

- グループクォータが設定されていることを確認します。

```
# quota -vg groupname
```

23.7. プロジェクトごとにクォータの割り当て

以下の手順では、プロジェクトごとにクォータを割り当てます。

前提条件

- プロジェクトクォータがファイルシステムで有効になっている。

手順

1. プロジェクトが制御するディレクトリーを **/etc/projects** に追加します。たとえば、以下は一意の ID が 11 の **/var/log** パスを **/etc/projects** に追加します。プロジェクト ID には、プロジェクトにマッピングされる任意の数値を指定できます。

```
# echo 11:/var/log >> /etc/projects
```

2. **/etc/projid** にプロジェクト名を追加して、プロジェクト ID をプロジェクト名にマップします。たとえば、以下は、前のステップで定義されたように **Logs** というプロジェクトをプロジェクト ID 11 に関連付けます。

```
# echo Logs:11 >> /etc/projid
```

3. 必要な制限を設定します。

```
# edquota -P 11
```

注記

プロジェクトは、プロジェクト ID (この場合は **11**)、または名前 (この場合は **Logs**) で選択できます。

4. **quotaon** を使用して、クォータの強制適用を有効にします。
クォータ強制の有効化を 参照してください。

検証手順

- プロジェクトのクォータが設定されていることを確認します。

```
# quota -vP 11
```

注記

プロジェクト ID またはプロジェクト名のいずれかで検証できます。

関連情報

- **edquota(8)** man ページ。
- **projid(5)** man ページ。
- **projects(5)** man ページ。

23.8. ソフト制限の猶予期間の設定

特定のクォータにソフト制限がある場合、猶予期間 (ソフト制限を超過できる期間) を編集できます。ユーザー、グループ、またはプロジェクトの猶予期間を設定できます。

手順

- 猶予期間を編集します。

```
# edquota -t
```

重要

他の **edquota** コマンドは特定のユーザー、グループ、またはプロジェクトのクォータで機能しますが、**-t** オプションはクォータが有効になっているすべてのファイルシステムで機能します。

関連情報

- **edquota(8)** man ページ。

23.9. ファイルシステムのクォータをオフにする

quotaoff を使用して、指定されたファイルシステムでディスククォータの強制適用をオフにします。クォータアカウンティングは、このコマンド実行後も有効のままになります。

手順

- すべてのユーザーとグループのクォータをオフにするには、次のコマンドを実行します。

```
# quotaoff -vaugP
```

  - **-u** オプション、**-g** オプション、または **-P** オプションがいずれも指定されていないと、ユーザーのクォータのみが無効になります。
  - **-g** オプションのみを指定すると、グループクォータのみが無効になります。
  - **-P** オプションのみを指定すると、プロジェクトのクォータのみが無効になります。

- **-v** スイッチにより、コマンドの実行時に詳細なステータス情報が表示されます。

関連情報

- **quotaoff(8)** man ページ。

23.10. ディスククォータに関するレポート

repquota ユーティリティーを使用してディスククォータレポートを作成できます。

手順

1. **repquota** コマンドを実行します。

```
# repquota
```

たとえば、**repquota /dev/sda** コマンドは次のような出力を生成します。

```
*** Report for user quotas on device /dev/sda
Block grace time: 7days; Inode grace time: 7days
  Block limits   File limits
User  used soft hard grace used soft hard grace
----------------------------------------------------------------------
root      --      36       0       0              4     0     0
kristin   --     540       0       0            125     0     0
testuser  --  440400  500000  550000          37418     0     0
```

2. クォータが有効化された全ファイルシステムのディスク使用状況レポートを表示します。

```
# repquota -augP
```

各ユーザーに続いて表示される **--** 記号で、ブロックまたは inode の制限を超えたかどうかを簡単に判断できます。ソフト制限のいずれかを超えると、対応する **-** 文字の代わりに **+** 文字が表示されます。最初の **-** 文字はブロック制限を表し、次の文字は inode 制限を表します。

通常、**grace** 列は空白です。ソフト制限が超過した場合、その列には猶予期間に残り時間量に相当する時間指定が含まれます。猶予期間の期間が過ぎると、その時間には **何も** 表示されません。

関連情報

詳細は、**repquota(8)** man ページを参照してください。

第24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。ブロック破棄操作は、マウントされたファイルシステムでファイルシステムブロックが使用されなくなった基礎となるストレージと通信します。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。

24.1. 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

24.2. 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設定され、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

- **fstrim(8)** man ページ。

24.3. 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

- **mount(8)** man ページ。
- **fstab(5)** man ページ

24.4. 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25章 STRATIS ファイルシステムの設定

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 製品サポート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の機能は、最新の製品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についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

25.1. STRATIS とは

Stratis は、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に力を入れており、高度なストレージ機能にアクセスできます。

Stratis を使用すると、以下の活動をより簡単に行うことができます。

- 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 は、高度なストレージ機能をサポートするローカルストレージ管理システムです。Stratis は、ストレージ **プール** の概念を中心としています。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。

プールにより、次のような多くの便利な機能を使用できます。

- ファイルシステムのスナップショット
- シンプロビジョニング
- 階層化
- 暗号化

関連情報

- Stratis Web サイト

25.2. STRATIS ボリュームの設定要素

Stratis ボリュームを設定するコンポーネントについて説明します。

外部的には、Stratis は、コマンドラインインターフェイスおよび API に次のボリュームコンポーネントを表示します。

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設定されています。
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、ファイルを格納する 1 つ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 でフォーマットされています。

重要

Stratis は、Stratis を使用して作成したファイルシステムに関する情報を追跡し、XFS はそれを認識しません。また、XFS を使用して変更を行っても、自動的に Stratis に更新を作成しません。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。

注記

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。

25.3. STRATIS で使用可能なブロックデバイス

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。

25.4. 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

1. 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。

```
# yum install stratisd stratis-cli
```

2. **stratisd** サービスが有効になっていることを確認します。

```
# systemctl enable --now stratisd
```

25.5. 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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis プールを作成するブロックデバイスは使用されておらず、マウントされていない。
- Stratis プールを作成する各ブロックデバイスが、1 GB 以上である。
- IBM Z アーキテクチャーでは、**/dev/dasd*** ブロックデバイスをパーティションに分割している。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分割の詳細は、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

暗号化されていない Stratis プールを暗号化することはできません。

手順

1. 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
```

ここで、**block-device** は、ブロックデバイスへのパスになります (例: **/dev/sdb**)。

2. 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
```

ここで、**block-device** は、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になります。

注記

1 行に複数のブロックデバイスを指定します。

```
# stratis pool create my-pool block-device-1 block-device-2
```

3. 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
```

25.6. 暗号化された STRATIS プールの作成

データを保護するために、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。

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 プールを設定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

- Stratis v2.1.0 以降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。
- **stratisd** サービスを実行している。
- Stratis プールを作成するブロックデバイスは使用されておらず、マウントされていない。
- Stratis プールを作成するブロックデバイスが、それぞれ 1GB 以上である。

- IBM Z アーキテクチャーでは、**/dev/dasd*** ブロックデバイスをパーティションに分割している。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分割の詳細は、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

1. 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
```

ここで、**block-device** は、ブロックデバイスへのパスになります (例: **/dev/sdb**)。

2. キーセットをまだ作成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
```

ここでの **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。

3. 暗号化した Stratis プールを作成し、暗号化に使用する鍵の説明を指定します。**key-description** オプションを使用する代わりに、**--keyfile-path** オプションを使用してキーのパスを指定することもできます。

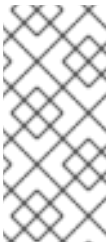
```
# stratis pool create --key-desc key-description my-pool block-device
```

ここでは、以下のようになります。

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。



注記

1 行に複数のブロックデバイスを指定します。

```
# stratis pool create --key-desc key-description my-pool block-device-1
block-device-2
```

4. 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
```

25.7. 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

ストレージスタックは、オーバープロビジョニングの状態になる可能性があります。ファイルシステムのサイズが、そのファイルシステムをサポートするプールよりも大きい場合には、プールがいっぱいになります。これを回避するには、オーバープロビジョニングを無効にし、プール上のすべてのファイルシステムのサイズが、プールが提供する利用可能な物理ストレージを超えないようにします。重要なアプリケーションまたは root ファイルシステムに Stratis を使用する場合は、このモードでは特定の障害ケースが阻止されます。

オーバープロビジョニングを有効にすると、ストレージが完全に割り当てられたことを API シグナルに通知します。通知は、残りのプールスペースがすべていっぱいになると、Stratis に拡張するスペースが残っていないことをユーザーに通知する警告として機能します。

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. 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
```

2. 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。

検証

1. 以下のコマンドを実行し、Stratis プールの全一覧を表示します。

```
# 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
```

2. ubuntu 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。

3. オプション: 以下のコマンドを実行して、特定のプールで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
```

関連情報

- Stratis Storage の Web ページ

25.8. 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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- 暗号化した Stratis プールを作成し、暗号化に使用されたキーの説明がある。詳細は、暗号化された Stratis プールの作成 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
```

ここでは、以下のようになります。

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

25.9. 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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- 暗号化された Stratis プールを作成している。詳細は、暗号化された Stratis プールの作成 を参照してください。

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 key-description
```

ここでは、以下のようになります。

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。

25.10. 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、またはこれを設定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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- 暗号化された Stratis プールを作成している。詳細は、暗号化された Stratis プールの作成 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
```

ここで、**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
```

25.11. 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 プールを作成している。詳細は、暗号化された Stratis プールの作成 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
```

ここでは、以下のようになります。

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

25.12. 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。これにより、ファイルシステム、キャッシュデバイス、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オプションとして分解するか、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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- 暗号化されていない、または暗号化された Stratis プールを作成している。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。

または 暗号化された Stratis プールの作成

手順

- 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 pool-uuid --unlock-method <keyring|clevis>
```

- または、以下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 pool-name
```

検証手順

- 以下のコマンドを使用して、システム上のプールを一覧表示します。

```
# stratis pool list
```

- 以下のコマンドを使用して、以前に起動していないプールの一覧を表示します。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
```

25.13. 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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis プールを作成している。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。

または 暗号化された Stratis プールの作成

手順

1. Stratis ファイルシステムをプールに作成するには、次のコマンドを実行します。

```
# stratis filesystem create --size number-and-unit my-pool my-fs
```

ここでは、以下のようになります。

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25.1 Stratis ファイルシステムの作成

```
# stratis filesystem create --size 10GiB pool1 filesystem1
```

検証手順

- プール内のファイルシステムを一覧表示して、Stratis ファイルシステムが作成されているか確認します。

```
# stratis fs list my-pool
```

関連情報

- Stratis ファイルシステムのマウント

25.14. 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ている。詳細は、Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- 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 が維持するエントリーを使用します。

```
# 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。

関連情報

- Stratis ファイルシステムの作成

25.15. STRATIS ファイルシステムの永続的なマウント

この手順では、Stratis ファイルシステムを永続的にマウントして、システムが起動した後に自動的に利用できるようにします。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。

- Stratis ファイルシステムを作成している。Stratis ファイルシステムの作成 を参照してください。

手順

1. ファイルシステムの UUID 属性を調べます。

```
$ lsblk --output=UUID /dev/stratis/my-pool/my-fs
```

以下に例を示します。

例25.2 Stratis ファイルシステムの UUID の表示

```
$ lsblk --output=UUID /dev/stratis/my-pool/fs1

UUID
a1f0b64a-4ebb-4d4e-9543-b1d79f600283
```

2. このマウントポイントのディレクトリーがない場合は、作成します。

```
# mkdir --parents mount-point
```

3. root で **/etc/fstab** ファイルを編集し、ファイルシステムに行を追加します (UUID で識別されます)。**xfs** をファイルシステムのタイプとして使用し、**x-systemd.requires=stratisd.service** オプションを追加します。
以下に例を示します。

例25.3 /etc/fstab の /fs1 マウントポイント

```
UUID=a1f0b64a-4ebb-4d4e-9543-b1d79f600283 /fs1 xfs defaults,x-
systemd.requires=stratisd.service 0 0
```

4. 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
```

5. ファイルシステムをマウントして、設定が機能することを確認します。

```
# mount mount-point
```

関連情報

- ファイルシステムの永続的なマウント

25.16. SYSTEMD サービスを使用した /ETC/FSTAB での非 ROOT 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 root 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ている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis ファイルシステムを作成している。Stratis ファイルシステムの作成 を参照してください。

手順

- すべての非 root Stratis ファイルシステムでは、次を使用します。

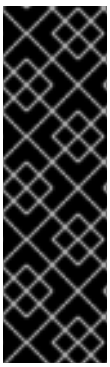
```
# /dev/stratis/[STRATIS_SYMLINK] [MOUNT_POINT] xfs defaults, x-
systemd.requires=stratis-fstab-setup@[POOL_UUID].service,x-systemd.after=stratis-stab-
setup@[POOL_UUID].service <dump_value> <fsck_value>
```

関連情報

- ファイルシステムの永続的なマウント

第26章 追加のブロックデバイスでの STRATIS ボリュームの拡張

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。



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 製品サポート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の機能は、最新の製品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についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

26.1. STRATIS ボリュームの設定要素

Stratis ボリュームを設定するコンポーネントについて説明します。

外部的には、Stratis は、コマンドラインインターフェイスおよび API に次のボリュームコンポーネントを表示します。

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設定されています。
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、ファイルを格納する 1 つ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 でフォーマットされています。

重要

Stratis は、Stratis を使用して作成したファイルシステムに関する情報を追跡し、XFS はそれを認識しません。また、XFS を使用して変更を行っても、自動的に Stratis に更新を作成しません。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。

注記

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。

26.2. STRATIS プールへのブロックデバイスの追加

この手順では、Stratis ファイルシステムで使用できるように、1 つ以上のブロックデバイスを Stratis プールに追加します。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis プールに追加するブロックデバイスは使用されておらず、マウントされていない。
- Stratis プールに追加するブロックデバイスは使用されておらず、それぞれ 1 GiB 以上である。

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

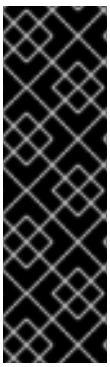
- **stratis(8)** man ページ

26.3. 関連情報

- **Stratis Storage** の Web サイト

第27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 ボリュー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 製品サポート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の機能は、最新の製品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についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

27.1. さまざまなユーティリティーが報告する STRATIS のサイズ

本セクションでは、**df** などの標準的なユーティリティーと、**stratis** ユーティリティーにより報告される Stratis サイズの相違点を説明します。

df などの標準的な Linux ユーティリティーは、Stratis 上の 1TiB の XFS ファイルシステムレイヤーのサイズを報告します。これは 1 TiB です。Stratis の実際のストレージ使用量は、シンプロビジョニングにより少なくなっており、また XFS レイヤーが満杯に近くなると Stratis が自動的にファイルシステムを拡張するため、これは特に有用な情報ではありません。

重要

Stratis ファイルシステムに書き込まれているデータ量を定期的に監視します。これは **Total Physical Used** の値として報告されます。これが **Total Physical Size** の値を超えていないことを確認してください。

関連情報

- **stratis (8)** man ページ

27.2. STRATIS ボリュームの情報表示

この手順では、Stratis ボリュームに関する合計サイズ、使用済みサイズ、空きサイズ、ファイルシステム、プールに属するブロックデバイスなどの統計情報をリスト表示します。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。

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

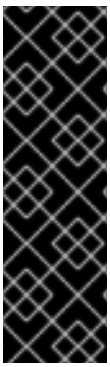
- **stratis (8)** man ページ

27.3. 関連情報

- Stratis Storage の Web サイト

第28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 製品サポート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の機能は、最新の製品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についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

28.1. 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。スナップショットには、元のファイルシステムと同じファイルの内容が含まれていますが、スナップショットが変更するときにファイル内容が変更する可能性があります。スナップショットにどんな変更を加えても、元のファイルシステムには反映されません。

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キングストレージの約半分のギガバイトを使用します。

28.2. STRATIS スナップショットの作成

この手順では、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します。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis ファイルシステムを作成している。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するには、次のコマンドを実行します。

```
# stratis fs snapshot my-pool my-fs my-fs-snapshot
```

関連情報

- **stratis (8)** man ページ

28.3. STRATIS スナップショットのコンテンツへのアクセス

この手順では、Stratis ファイルシステムのスナップショットをマウントして、読み書き操作にアクセスできるようにします。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis スナップショットを作成している。Stratis ファイルシステムの作成 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
- **mount(8)** man ページ。

28.4. STRATIS ファイルシステムを以前のスナップショットに戻す

この手順では、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ーされた状態に戻します。

前提条件

- Stratis がインストールされている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis スナップショットを作成している。Stratis スナップショットの作成 を参照してください。

手順

1. 必要に応じて、後でそれにアクセスできるように、ファイルシステムの現在の状態のバックアップを作成します。

```
# stratis filesystem snapshot my-pool my-fs my-fs-backup
```

2. 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

- **stratis (8)** man ページ

28.5. STRATIS スナップショットの削除

この手順では、Stratis スナップショットをプールから削除します。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ている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis スナップショットを作成している。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

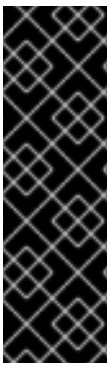
- **stratis (8)** man ページ

28.6. 関連情報

- Stratis Storage の Web サイト

第29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 Stratis プールは、そこに含まれるデータを破棄することで削除できます。



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 製品サポート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の機能は、最新の製品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についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

29.1. STRATIS ボリュームの設定要素

Stratis ボリュームを設定するコンポーネントについて説明します。

外部的には、Stratis は、コマンドラインインターフェイスおよび API に次のボリュームコンポーネントを表示します。

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設定されています。
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。

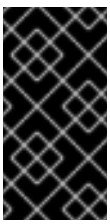
プールには、**dm-cache** ターゲットを使用した不揮発性データキャッシュなど、ほとんどの Stratis レイヤーが含まれています。

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、ファイルを格納する 1 つ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 でフォーマットされています。



重要

Stratis は、Stratis を使用して作成したファイルシステムに関する情報を追跡し、XFS はそれを認識しません。また、XFS を使用して変更を行っても、自動的に Stratis に更新を作成しません。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。

注記

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。

29.2. STRATIS ファイルシステムの削除

この手順では、既存の Stratis ファイルシステムを削除します。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ている。Stratis のインストール を参照してください。
- **stratisd** サービスを実行している。
- Stratis ファイルシステムを作成している。Stratis ファイルシステムの作成 を参照してください。

手順

1. 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
```

2. ファイルシステムを破棄します。

```
# stratis filesystem destroy my-pool my-fs
```

3. 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

- **stratis (8)** man ページ

29.3. STRATIS プールの削除

この手順では、既存の Stratis プールを削除します。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ている。Stratis のインストール 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、暗号化された Stratis プールの作成 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
```

5. プールがなくなったことを確認します。

```
# stratis pool list
```

関連情報

- **stratis (8)** man ページ

29.4. 関連情報

- Stratis Storage の Web サイト

第30章 EXT3 ファイルシステムの使用

システム管理者は、ext3 ファイルシステムの作成、マウント、サイズ変更、バックアップ、および復元が可能です。ext3 ファイルシステムは、基本的に、ext2 ファイルシステムが拡張されたバージョンです。

30.1. EXT3 ファイルシステムの機能

ext3 ファイルシステムの機能は以下のとおりです。

- 可用性 - 予期しない電源障害やシステムクラッシュの後、ジャーナリングが提供されているため、ファイルシステムを検査する必要はありません。デフォルトのジャーナルサイズは、ハードウェアの速度に応じて、復旧するのに約1秒かかります

注記

ext3 で対応しているジャーナリングモードは **data=ordered** (デフォルト) のみです。詳細は、EXT ジャーナリングオプション "data=writeback" は RHEL でサポートされますか？を参照してください。ナレッジベース記事。

- データの整合性 - ext3 ファイルシステムは、予期しない電源障害やシステムクラッシュが発生したときに、データの整合性が失われないようにします。
- 速度 - 一部のデータを複数回書き込みますが、ext3 のジャーナリングにより、ハードドライブのヘッドモーションが最適化されるため、ほとんどの場合、ext3 のスループットは ext2 よりも高くなります。
- 簡単な移行 - ext2 から ext3 に簡単に移行でき、再フォーマットをせずに、堅牢なジャーナリングファイルシステムの恩恵を受けることができます。

関連情報

- **ext3** man ページ

30.2. EXT3 ファイルシステムの作成

システム管理者は、**mkfs.ext3** コマンドを使用して、ブロックデバイスに ext3 ファイルシステムを作成できます。

前提条件

- ディスクにパーティションがある。MBR または GPT パーティションの作成は、parted を使用してディスク上にパーティションテーブルを作成する を参照してください。

.

+ もしくは、LVM ボリュームまたは MD ボリュームを使用します。

手順

1. ext3 ファイルシステムを作成する場合は、以下の手順を実行します。
    - デバイスが通常のパーティションの場合、LVM ボリューム、MD ボリューム、または類似デバイスは次のコマンドを使用します。

```
# mkfs.ext3 /dev/block_device
```

/dev/**block_device** を、ブロックデバイスへのパスに置き換えます。

たとえば、**/dev/sdb1**、**/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**、または **/dev/my-volgroup/my-lv** です。一般的な用途では、デフォルトのオプションが最適です。

- ストライプ化されたブロックデバイス (RAID5 アレイなど) の場合は、ファイルシステムの作成時にストライプジオメトリーを指定できます。適切なストライプジオメトリーを使用することで、ext3 ファイルシステムのパフォーマンスが向上します。たとえば、4k ブロックのファイルシステムで、64k ストライド (16 x 4096) のファイルシステムを作成する場合は、次のコマンドを使用します。

```
# mkfs.ext3 -E stride=16,stripe-width=64 /dev/block_device
```

この例では、以下のようになります。

  - stride=value - RAID チャンクサイズを指定します。
  - stripe-width=value - 1 RAID デバイス内のデータディスク数、または 1 ストライプ内のストライプユニット数を指定します。

注記

- ファイルシステムの作成時に UUID を指定する場合は、次のコマンドを実行します。

```
# mkfs.ext3 -U UUID /dev/block_device
```

UUID を、設定する UUID (例: **7cd65de3-e0be-41d9-b66d-96d749c02da7**) に置き換えます。

/dev/**block_device** を、ext3 ファイルシステムへのパス (例: **/dev/sda8**) に置き換え、UUID を追加します。

- ファイルシステムの作成時にラベルを指定するには、以下のコマンドを実行します。

```
# mkfs.ext3 -L label-name /dev/block_device
```

2. 作成した ext3 ファイルシステムを表示する場合は、以下のコマンドを実行します。

```
# blkid
```

関連情報

- **ext3** man ページ
- **mkfs.ext3** man ページ

30.3. EXT3 ファイルシステムのマウント

システム管理者は、**mount** ユーティリティーを使用して、ext3 ファイルシステムをマウントできます。

前提条件

- ext3 ファイルシステム。ext3 ファイルシステムの作成については、ext3 ファイルシステムの作成 を参照してください。

手順

1. ファイルシステムをマウントするためのマウントポイントを作成するには、以下のコマンドを実行します。

```
# mkdir /mount/point
```

   /mount/point を、パーティションのマウントポイントを作成するディレクトリー名に置き換えます。

2. ext3 ファイルシステムをマウントするには、以下の手順を実行します。
   - ext3 ファイルシステムを追加のオプションなしでマウントするには、以下のコマンドを実行します。

```
# mount /dev/block_device /mount/point
```

   - ファイルシステムを永続的にマウントするには、ファイルシステムの永続的なマウント を参照してください。

3. マウントされたファイルシステムを表示するには、次のコマンドを実行します。

```
# df -h
```

関連情報

- **mount** man ページ
- **ext3** man ページ
- **fstab** man ページ
- ファイルシステムのマウント

30.4. EXT3 ファイルシステムのサイズ変更

システム管理者は、**resize2fs** ユーティリティーを使用して、ext3 ファイルシステムのサイズを変更できます。**resize2fs** ユーティリティーは、特定の単位を示す接尾辞が使用されていない限り、ファイルシステムのブロックサイズの単位でサイズを読み取ります。以下の接尾辞は、特定の単位を示しています。

- s (セクター) - **512** バイトのセクター
- K (キロバイト) - **1,024** バイト
- M (メガバイト) - **1,048,576** バイト

- G (ギガバイト) - **1,073,741,824** バイト
- T (テラバイト) - **1,099,511,627,776** バイト

前提条件

- ext3 ファイルシステム。ext3 ファイルシステムの作成については、ext3 ファイルシステムの作成 を参照してください。
- サイズ変更後にファイルシステムを保持するための、適切なサイズの基本ブロックデバイス

手順

1. ext3 ファイルシステムのサイズを変更する場合は、以下の手順に従ってください。
   - アンマウントされている ext3 ファイルシステムのサイズを縮小および拡張するには、以下のコマンドを実行します。

     ```
     # umount /dev/block_device
     # e2fsck -f /dev/block_device
     # resize2fs /dev/block_device size
     ```

     /dev/block_device を、ブロックデバイスへのパス (例: **/dev/sdb1**) に置き換えます。

     size を、**s**、**K**、**M**、**G**、および **T** の接尾辞を使用して必要なサイズ変更値に置き換えます。
   - ext3 ファイルシステムは、**resize2fs** コマンドを使用して、マウントしたままの状態でサイズを大きくすることができます。

     ```
     # resize2fs /mount/device size
     ```

     注記

     拡張時のサイズパラメーターは任意です (多くの場合は必要ありません)。**resize2fs** は、コンテナーの使用可能な領域 (通常は論理ボリュームまたはパーティション) を埋めるように、自動的に拡張します。
2. サイズを変更したファイルシステムを表示するには、次のコマンドを実行します。

   ```
   # df -h
   ```

関連情報

- **resize2fs** man ページ
- **e2fsck** man ページ
- **ext3** man ページ

第31章 EXT4 ファイルシステムの使用

システム管理者は、ext4 ファイルシステムの作成、マウント、サイズ変更、バックアップ、および復元が可能です。ext4 ファイルシステムは、ext3 ファイルシステムの拡張性を高めたファイルシステムです。Red Hat Enterprise Linux 8 では、最大 **16** テラバイトの個別のファイルサイズと、最大 **50** テラバイトのファイルシステムに対応します。

31.1. EXT4 ファイルシステムの機能

以下は、ext4 ファイルシステムの機能です。

- エクステントの使用 - ext4 ファイルシステムはエクステントを使用します。これにより、サイズが大きいファイルを使用する場合のパフォーマンスが向上し、サイズが大きいファイルのメタデータのオーバーヘッドが削減されます。
- ext4 は、未割り当てのブロックグループと、inode テーブルセクションに適宜ラベルを付けます。これにより、ファイルシステムの検査時に、ブロックグループとテーブルセクションをスキップできます。ファイルシステムの検査が簡単に行われ、ファイルシステムがサイズが大きくなるとより有益になります。
- メタデータチェックサム - この機能は、デフォルトでは Red Hat Enterprise Linux 8 で有効になっています。
- 以下は、ext4 ファイルシステムの割り当て機能です。
  - 永続的な事前割り当て
  - 遅延割り当て
  - マルチブロック割り当て
  - ストライプ認識割り当て
- 拡張属性 (**xattr**) - これにより、システムは、ファイルごとに、名前と値の組み合わせを追加で関連付けられるようになります。
- クォータジャーナリング - クラッシュ後に行なわれる時間がかかるクォータの整合性チェックが不要になります。

注記

ext4 で対応しているジャーナリングモードは **data=ordered** (デフォルト) のみです。詳細は、EXT ジャーナリングオプション "data=writeback" は RHEL でサポートされますか？を参照してください。ナレッジベース記事。

- サブセカンド (一秒未満) のタイムスタンプ - サブセカンドのタイムスタンプを指定します。

関連情報

- **ext4** man ページ

31.2. EXT4 ファイルシステムの作成

システム管理者は、**mkfs.ext4** コマンドを使用して、ブロックデバイスに ext4 ファイルシステムを作成できます。

前提条件

- ディスクにパーティションがある。MBR または GPT パーティションの作成は、parted を使用してディスク上にパーティションテーブルを作成する を参照してください。
- もしくは、LVM ボリュームまたは MD ボリュームを使用します。

手順

1. ext4 ファイルシステムを作成する場合は、以下の手順を実行します。
   - デバイスが通常のパーティションの場合、LVM ボリューム、MD ボリューム、または類似デバイスは次のコマンドを使用します。

     ```
     # mkfs.ext4 /dev/block_device
     ```

     /dev/**block_device** を、ブロックデバイスへのパスに置き換えます。

     たとえば、**/dev/sdb1**、**/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**、または **/dev/my-volgroup/my-lv** です。一般的な用途では、デフォルトのオプションが最適です。
   - ストライプ化されたブロックデバイス (RAID5 アレイなど) の場合は、ファイルシステムの作成時にストライプジオメトリーを指定できます。適切なストライプジオメトリーを使用することで、ext4 ファイルシステムのパフォーマンスが向上します。たとえば、4k ブロックのファイルシステムで、64k ストライド (16 x 4096) のファイルシステムを作成する場合は、次のコマンドを使用します。

     ```
     # mkfs.ext4 -E stride=16,stripe-width=64 /dev/block_device
     ```

     この例では、以下のようになります。
     - stride=value - RAID チャンクサイズを指定します。
     - stripe-width=value - 1 RAID デバイス内のデータディスク数、または 1 ストライプ内のストライプユニット数を指定します。

注記

- ファイルシステムの作成時に UUID を指定する場合は、次のコマンドを実行します。

```
# mkfs.ext4 -U UUID /dev/block_device
```

UUID を、設定する UUID (例: **7cd65de3-e0be-41d9-b66d-96d749c02da7**) に置き換えます。

/dev/**block_device** を、ext4 ファイルシステムへのパス (例: **/dev/sda8**) に置き換え、UUID を追加します。

- ファイルシステムの作成時にラベルを指定するには、以下のコマンドを実行します。

```
# mkfs.ext4 -L label-name /dev/block_device
```

2. 作成した ext4 ファイルシステムを表示するには、以下のコマンドを実行します。

```
# blkid
```

関連情報

- **ext4** man ページ
- **mkfs.ext4** man ページ

31.3. EXT4 ファイルシステムのマウント

システム管理者は、**mount** ユーティリティーを使用して、ext4 ファイルシステムをマウントできます。

前提条件

- ext4 ファイルシステム。ext4 ファイルシステムの作成は、ext4 ファイルシステムの作成 を参照してください。

手順

1. ファイルシステムをマウントするためのマウントポイントを作成するには、以下のコマンドを実行します。

```
# mkdir /mount/point
```

/mount/point を、パーティションのマウントポイントを作成するディレクトリー名に置き換えます。

2. ext4 ファイルシステムをマウントするには、以下を行います。
   - ext4 ファイルシステムを追加のオプションなしでマウントするには、次のコマンドを実行します。

```
# mount /dev/block_device /mount/point
```

- ファイルシステムを永続的にマウントするには、ファイルシステムの永続的なマウント を参照してください。

3. マウントされたファイルシステムを表示するには、次のコマンドを実行します。

```
# df -h
```

関連情報

- **mount** man ページ
- **ext4** man ページ
- **fstab** man ページ
- ファイルシステムのマウント

31.4. EXT4 ファイルシステムのサイズ変更

システム管理者は、**resize2fs** ユーティリティーを使用して、ext4 ファイルシステムのサイズを変更できます。**resize2fs** ユーティリティーは、特定の単位を示す接尾辞が使用されていない限り、ファイルシステムのブロックサイズの単位でサイズを読み取ります。以下の接尾辞は、特定の単位を示しています。

- s (セクター) - **512** バイトのセクター
- K (キロバイト) - **1,024** バイト
- M (メガバイト) - **1,048,576** バイト
- G (ギガバイト) - **1,073,741,824** バイト
- T (テラバイト) - **1,099,511,627,776** バイト

前提条件

- ext4 ファイルシステム。ext4 ファイルシステムの作成は、ext4 ファイルシステムの作成 を参照してください。
- サイズ変更後にファイルシステムを保持するための、適切なサイズの基本ブロックデバイス

手順

1. ext4 ファイルシステムのサイズを変更するには、以下の手順に従ってください。
   - アンマウントされている ext4 ファイルシステムのサイズを縮小および拡張するには、次のコマンドを実行します。

```
# umount /dev/block_device
# e2fsck -f /dev/block_device
# resize2fs /dev/block_device size
```

/dev/block_device を、ブロックデバイスへのパス (例: /**dev/sdb1**) に置き換えます。

size を、**s**、**K**、**M**、**G**、および **T** の接尾辞を使用して必要なサイズ変更値に置き換えます。

- ext4 ファイルシステムは、**resize2fs** を使用して、マウントしたままの状態でサイズを大きくすることができます。

```
# resize2fs /mount/device size
```

注記

拡張時のサイズパラメーターは任意です (多くの場合は必要ありません)。**resize2fs** は、コンテナーの使用可能な領域 (通常は論理ボリュームまたはパーティション) を埋めるように、自動的に拡張します。

2. サイズを変更したファイルシステムを表示するには、次のコマンドを実行します。

```
# df -h
```

関連情報

- **resize2fs** man ページ
- **e2fsck** man ページ
- **ext4** man ページ

31.5. EXT4 および XFS で使用されるツールの比較

本セクションでは、ext4 ファイルシステムおよび XFS ファイルシステムで一般的なタスクを行うのに使用するツールを比較します。

タスク	ext4	XFS
ファイルシステムを作成する	**mkfs.ext4**	**mkfs.xfs**
ファイルシステム検査	**e2fsck**	**xfs_repair**
ファイルシステムのサイズを変更する	**resize2fs**	**xfs_growfs**
ファイルシステムのイメージを保存する	**e2image**	**xfs_metadump** および **xfs_mdrestore**
ファイルシステムのラベル付けまたはチューニングを行う	**tune2fs**	**xfs_admin**
ファイルシステムのバックアップを作成する	**dump** および **restore**	**xfsdump** および **xfsrestore**

タスク	ext4	XFS
クォータ管理	**quota**	**xfs_quota**
ファイルマッピング	**filefrag**	**xfs_bmap**