



Red Hat Enterprise Linux 9

仮想化の設定および管理

ホストのセットアップ、仮想マシンの作成と管理、仮想化機能の詳細

Red Hat Enterprise Linux 9 仮想化の設定および管理

ホストのセットアップ、仮想マシンの作成と管理、仮想化機能の詳細

法律上の通知

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

Red Hat Enterprise Linux (RHEL) システムを仮想化ホストとして使用するには、このドキュメントの手順に従ってください。提供される情報には以下が含まれます。仮想化の機能およびユースケース コマンドラインユーティリティーと Web コンソールを使用して、ホストと仮想マシンを管理する方法 Intel 64、AMD64、IBM Z など、さまざまなシステムアーキテクチャーにおける仮想化のサポート制限

目次

RED HAT ドキュメントへのフィードバック (英語のみ)	6
第1章 RHEL における仮想化について	7
1.1. 仮想化とは	7
1.2. 仮想化の利点	7
1.3. 仮想マシンコンポーネントおよびその相互作用	8
1.4. 仮想管理に使用するツールおよびインターフェイス	9
1.5. RED HAT の仮想化ソリューション	10
第2章 仮想化の有効化	11
2.1. AMD64 および INTEL 64 での仮想化の有効化	11
2.2. IBM Z での仮想化の有効化	12
2.3. ARM 64 での仮想化の有効化	13
2.4. 仮想マシンでの QEMU ゲストエージェント機能の有効化	15
第3章 仮想マシンの作成	19
3.1. コマンドラインインターフェイスを使用した仮想マシンの作成	19
3.2. WEB コンソールを使用した仮想マシンの作成、およびゲストのオペレーティングシステムのインストール	23
第4章 仮想マシンの起動	30
4.1. コマンドラインインターフェイスでの仮想マシンの起動	30
4.2. WEB コンソールを使用した仮想マシンの起動	31
4.3. ホストの起動時に仮想マシンを自動的に起動する	31
第5章 仮想マシンへの接続	34
5.1. WEB コンソールを使用した仮想マシンとの相互作用	34
5.2. VIRT VIEWER で仮想マシンのグラフィカルコンソールを開く方法	38
5.3. SSH を使用した仮想マシンへの接続	39
5.4. 仮想マシンのシリアルコンソールを開く	41
5.5. リモートの仮想化ホストへの簡単なアクセスの設定	42
第6章 仮想マシンのシャットダウン	45
6.1. コマンドラインインターフェイスを使用した仮想マシンのシャットダウン	45
6.2. WEB コンソールを使用した仮想マシンのシャットダウンおよび再起動	45
第7章 仮想マシンの削除	48
7.1. コマンドラインインターフェイスを使用した仮想マシンの削除	48
7.2. WEB コンソールを使用した仮想マシンの削除	48
第8章 WEB コンソールでの仮想マシンの管理	50
8.1. WEB コンソールを使用した仮想マシンの管理の概要	50
8.2. 仮想マシンを管理するために WEB コンソールを設定	50
8.3. WEB コンソールを使用した仮想マシンの名前の変更	51
8.4. WEB コンソールで利用可能な仮想マシンの管理機能	52
第9章 仮想マシンに関する情報の表示	54
9.1. コマンドラインインターフェイスを使用した仮想マシン情報の表示	54
9.2. WEB コンソールを使用した仮想マシン情報の表示	56
9.3. 仮想マシンの XML 設定例	62
第10章 仮想マシンの保存および復元	67
10.1. 仮想マシンの保存および復元の仕組み	67
10.2. コマンドラインインターフェイスを使用した仮想マシンの保存	67

10.3. コマンドラインインターフェイスでの仮想マシンの起動	68
10.4. WEB コンソールを使用した仮想マシンの起動	69
第11章 仮想マシンのクローン作成	71
11.1. 仮想マシンのクローン作成の仕組み	71
11.2. 仮想マシンテンプレートの作成	71
11.3. コマンドラインインターフェイスを使用した仮想マシンのクローン作成	75
11.4. WEB コンソールを使用した仮想マシンのクローン作成	76
第12章 仮想マシンの移行	78
12.1. 仮想マシンの移行の仕組み	78
12.2. 仮想マシンの移行の利点	79
12.3. 仮想マシンの移行の制限事項	79
12.4. 仮想マシンの移行におけるホスト CPU の互換性の確認	80
12.5. 他のホストとの仮想マシンディスクイメージの共有	83
12.6. コマンドラインインターフェイスを使用した仮想マシンの移行	85
12.7. WEB コンソールを使用した仮想マシンのライブ移行	88
12.8. MELLANOX VIRTUAL FUNCTION が割り当てられた仮想マシンのライブマイグレーション	90
12.9. 仮想マシンの移行に関するトラブルシューティング	96
12.10. 仮想マシンの移行で対応しているホスト	98
第13章 スナップショットを使用した仮想マシンの状態の保存と復元	100
13.1. 仮想マシンのスナップショットのサポート制限	100
13.2. コマンドラインインターフェイスを使用した仮想マシンのスナップショットの作成	101
13.3. WEB コンソールを使用した仮想マシンのスナップショットの作成	104
13.4. コマンドラインインターフェイスを使用して仮想マシンのスナップショットに戻す	105
13.5. WEB コンソールを使用して仮想マシンのスナップショットに戻す	105
13.6. コマンドラインインターフェイスを使用して仮想マシンのスナップショットを削除する	106
13.7. WEB コンソールを使用して仮想マシンのスナップショットを削除する	107
第14章 仮想デバイスの管理	108
14.1. 仮想デバイスの動作	108
14.2. 仮想デバイスの種類	109
14.3. CLI を使用した仮想マシンに接続されたデバイスの管理	111
14.4. WEB コンソールを使用したホストデバイスの管理	115
14.5. 仮想 USB デバイスの管理	119
14.6. 仮想光学ドライブの管理	121
14.7. SR-IOV デバイスの管理	126
14.8. IBM Z の仮想マシンへの DASD デバイスの割り当て	131
14.9. WEB コンソールを使用した仮想マシンへのウォッチドッグデバイスの接続	134
14.10. IBM Z の仮想マシンへの PCI デバイスの接続	135
第15章 仮想マシン用のストレージの管理	138
15.1. 仮想マシンのストレージの概要	138
15.2. CLI を使用した仮想マシンストレージプールの管理	141
15.3. WEB コンソールを使用した仮想マシンストレージプールの管理	154
15.4. ストレージプールを作成するパラメーター	166
15.5. CLI を使用した仮想マシンのストレージボリュームの管理	175
15.6. CLI を使用した仮想ディスクイメージの管理	178
15.7. WEB コンソールを使用した仮想マシンのストレージボリュームの管理	183
15.8. WEB コンソールを使用した仮想マシンストレージディスクの管理	186
15.9. LIBVIRT シークレットを使用した ISCSI ストレージプールのセキュリティー保護	191
15.10. VHBA の作成	193
第16章 仮想マシンでの GPU デバイスの管理	196

16.1. 仮想マシンへの GPU の割り当て	196
16.2. NVIDIA vGPU デバイスの管理	199
第17章 仮想マシンのネットワーク接続の設定	206
17.1. 仮想ネットワークの概要	206
17.2. WEB コンソールで仮想マシンのネットワークインターフェイスの管理	208
17.3. 推奨される仮想マシンネットワーク設定	211
17.4. 仮想マシンのネットワーク接続の種類	214
17.5. PXE サーバーから仮想マシンの起動	219
17.6. PASST ユーザー空間接続の設定	222
17.7. 関連情報	224
第18章 仮想マシンのパフォーマンスの最適化	225
18.1. 仮想マシンのパフォーマンスに影響を及ぼすもの	225
18.2. TUNED を使用した仮想マシンのパフォーマンスの最適化	226
18.3. LIBVIRT デモンの最適化	227
18.4. 仮想マシンのメモリの設定	229
18.5. 仮想マシンの I/O パフォーマンスの最適化	241
18.6. 仮想マシンの CPU パフォーマンスの最適化	244
18.7. 仮想マシンのネットワークパフォーマンスの最適化	256
18.8. 仮想マシンのパフォーマンス監視ツール	257
18.9. 関連情報	259
第19章 仮想マシンの保護	260
19.1. 仮想マシンでセキュリティーが機能する仕組み	260
19.2. 仮想マシンのセキュリティー保護に関するベストプラクティス	261
19.3. SECUREBOOT の仮想マシンの作成	262
19.4. 仮想マシンユーザーが使用できるアクションの制限	263
19.5. 仮想マシンのセキュリティーの自動機能	265
19.6. 仮想化用の SELINUX ブール値	265
19.7. IBM Z での IBM SECURE EXECUTION の設定	267
19.8. IBM Z 上の仮想マシンへの暗号化コプロセッサの割り当て	270
19.9. WINDOWS 仮想マシンでの標準ハードウェアセキュリティーの有効化	274
19.10. WINDOWS 仮想マシンでの拡張ハードウェアセキュリティーの有効化	275
第20章 ホストとその仮想マシン間でのファイルの共有	277
20.1. NFS を使用したホストとその仮想マシン間でのファイルの共有	277
20.2. VIRTIOFS を使用したホストとその仮想マシン間でのファイルの共有	280
第21章 WINDOWS 仮想マシンのインストールおよび管理	286
21.1. WINDOWS 仮想マシンのインストール	286
21.2. WINDOWS 仮想マシンの最適化	288
21.3. WINDOWS 仮想マシンでの標準ハードウェアセキュリティーの有効化	304
21.4. WINDOWS 仮想マシンでの拡張ハードウェアセキュリティーの有効化	305
21.5. 次のステップ	306
第22章 入れ子仮想マシンの作成	307
22.1. ネストされた仮想化とは	307
22.2. ネストされた仮想化に対するサポート制限	308
22.3. INTEL でのネスト化された仮想マシンの作成	310
22.4. AMD でのネスト化された仮想マシンの作成	312
22.5. IBM Z でのネストされた仮想マシンの作成	313
第23章 仮想マシンの問題診断	315
23.1. LIBVIRT デバッグログの生成	315

23.2. 仮想マシンのコアのダンプ	318
23.3. 仮想マシンプロセスのバックトレース	319
第24章 RHEL 9 仮想化における機能のサポートおよび制限	321
24.1. RHEL 9 仮想化サポートの動作	321
24.2. RHEL 9 仮想化で推奨される機能	321
24.3. RHEL 9 仮想化で対応していない機能	323
24.4. RHEL 9 仮想化におけるリソース割り当ての制限	326
24.5. IBM Z の仮想化と、AMD64 および INTEL 64 の仮想化の相違点	327
24.6. ARM 64 での仮想化が AMD64 および INTEL 64 とどのように異なるか	329
24.7. RHEL 9 における仮想化機能のサポートの概要	332

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 RHEL における仮想化について

本章では、仮想化の概念や、Linux における仮想化の実装について参考になるように、RHEL 9 における仮想化の概要、基本な内容、利点、コンポーネントなど、Red Hat が提供する仮想化ソリューションを説明します。

1.1. 仮想化とは

RHEL 9 では **仮想化機能** が提供され、RHEL 9 を実行するマシンが、複数の仮想マシン (VM) (**ゲスト** と呼ばれます) を **ホスト** できるようにします。仮想マシンは、ホストの物理ハードウェアとコンピューティングリソースを使用して、独立した仮想化オペレーティングシステム (**ゲスト OS**) を、ホストのオペレーティングシステムのユーザー空間プロセスとして実行します

つまり、仮想化により、オペレーティングシステム内にオペレーティングシステムを追加できます。

仮想マシンを使用すると、ソフトウェアの設定や機能を安全にテストしたり、レガシーソフトウェアを実行したり、ハードウェアのワークロードの効率を最適化したりできます。利点の詳細は、[仮想化の利点](#) を参照してください。

仮想化の詳細は、[仮想化のトピックページ](#) を参照してください。

次のステップ

- Red Hat Enterprise Linux 9 で仮想化の使用を開始するには、Red Hat Enterprise Linux 9 での [仮想化の有効化](#) を参照してください。
- Red Hat は、Red Hat Enterprise Linux 9 の仮想化以外にも、専門化した仮想化ソリューションを多数提供しています。各ソリューションには、さまざまなユーザーフォーカスおよび機能があります。詳細は、[Red Hat virtualization solutions](#) を参照してください。

1.2. 仮想化の利点

仮想マシンの使用には、物理マシンを使用する場合と比較して、以下の利点があります。

- **リソースの柔軟性と詳細な割り当て**

仮想マシンは、通常、物理マシンであるホストマシンで稼働し、使用するゲスト OS に物理ハードウェアを割り当てることもできます。ただし、仮想マシンへの物理リソースの割り当てはソフトウェアレベルで行うため、柔軟性が非常に高くなります。仮想マシンは、ホストメモリ、CPU、またはストレージ領域で設定可能な割合を指定して、非常に詳細なリソース要求を指定できます。

たとえば、ゲスト OS がディスクとして見るものは、ホストファイルシステムではファイルとして表示され、そのディスクのサイズは、物理ディスクで利用可能なサイズよりも少なくなります。

- **ソフトウェアで制御される設定**

仮想マシン全体の設定は、ホスト上のデータとして保存され、ソフトウェア制御下にあります。したがって、仮想マシンの作成、削除、クローン作成、移行、リモートからの操作、リモートストレージへの接続などを簡単に行うことができます。

- **ホストからの分離**

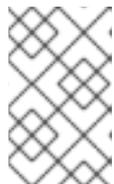
ゲスト OS は、ホストの OS とは別の仮想化カーネルで実行します。つまり、任意の OS を仮想マシンにインストールでき、ゲスト OS が不安定になっても、または不正アクセスされても、ホストには影響を及ぼしません。

- **領域とコスト効率**

1 台の物理マシンで仮想マシンを多数ホストできます。したがって、複数の物理マシンが同じタスクを実行する必要がないため、物理ハードウェアに対する領域、電力、およびメンテナンスの要件が低くなります。

- **ソフトウェアの互換性**

仮想マシンは、ホストとは異なる OS を使用できるため、仮想化により、本来はホスト OS 用にリリースされていないアプリケーションを実行できるようになります。たとえば、RHEL 7 のゲスト OS を使用すると、RHEL 7 用にリリースされたアプリケーションを RHEL 9 ホストシステムで実行できます。



注記

RHEL 9 ホストでは、すべてのオペレーティングシステムがゲスト OS としてサポートされているわけではありません。詳細は、[Recommended features in RHEL 9 virtualization](#) を参照してください。

1.3. 仮想マシンコンポーネントおよびその相互作用

RHEL 9 の仮想化は、以下の主要ソフトウェアコンポーネントで設定されています。

ハイパーバイザー

RHEL 9 で仮想マシンを作成する基礎となる部分は、ハードウェアを制御し、ホストマシンで複数のオペレーティングシステムを実行できるようにするソフトウェア層で、**ハイパーバイザー** と呼ばれます。

ハイパーバイザーには、**KVM (Kernel-based Virtual Machine)** モジュールと仮想化カーネルドライバが含まれます。このコンポーネントでは、ホストマシンの Linux カーネルにより、ユーザー空間のソフトウェアに仮想化のリソースが提供されます。

ユーザー空間レベルでは、**QEMU** エミュレーターが、ゲスト OS を実行できる完全に仮想化されたハードウェアプラットフォームをシミュレートし、リソースがホストでどのように割り当てられ、ゲストに示されるかを管理します。

さらに、**libvirt** ソフトウェアスイートが管理層および通信層として機能し、QEMU とのやり取りを容易にし、セキュリティルールを適用し、仮想マシンを設定して実行するための追加ツールを多数提供します。

XML 設定

ホストベースの XML 設定ファイル (**ドメイン XML** ファイルとも呼ばれます) では、個別の仮想マシンの設定およびデバイスをすべて決定します。設定には以下が含まれます。

- メタデータ (仮想マシンの名前、タイムゾーン、その他の仮想マシンの情報など)
- 仮想マシンのデバイスの説明 (仮想 CPU (vCPU)、ストレージデバイス、入出力デバイス、ネットワークインターフェイスカード、その他の物理ハードウェアおよび仮想ハードウェアなど)
- 仮想マシンの設定 (使用可能な最大メモリー量、再起動設定、仮想マシンの動作に関するその他の設定など)

XML 設定の内容の詳細は、[仮想マシンの XML 設定例](#) を参照してください。

コンポーネントのインタラクション

仮想マシンが起動すると、ハイパーバイザーは XML 設定を使用して、ホストのユーザー空間プロセスとして仮想マシンのインスタンスを作成します。ハイパーバイザーは、仮想マシンプロセスが、ホスト

ベースのインターフェイス (**virsh** ユーティリティー、**virt-install** ユーティリティー、**guestfish** ユーティリティー、Web コンソールの GUI など) にアクセスできるようにします。

このような仮想化ツールを使用すると、libvirt が、入力を QEMU の命令に変換します。QEMU が命令を KVM に伝え、カーネルが命令を実行するのに必要なリソースを適切に割り当てるようになります。これにより、QEMU が、仮想マシンの作成や修正、仮想マシンのオペレーティングシステムでのアクションの実行など、対応するユーザー空間を変更します。

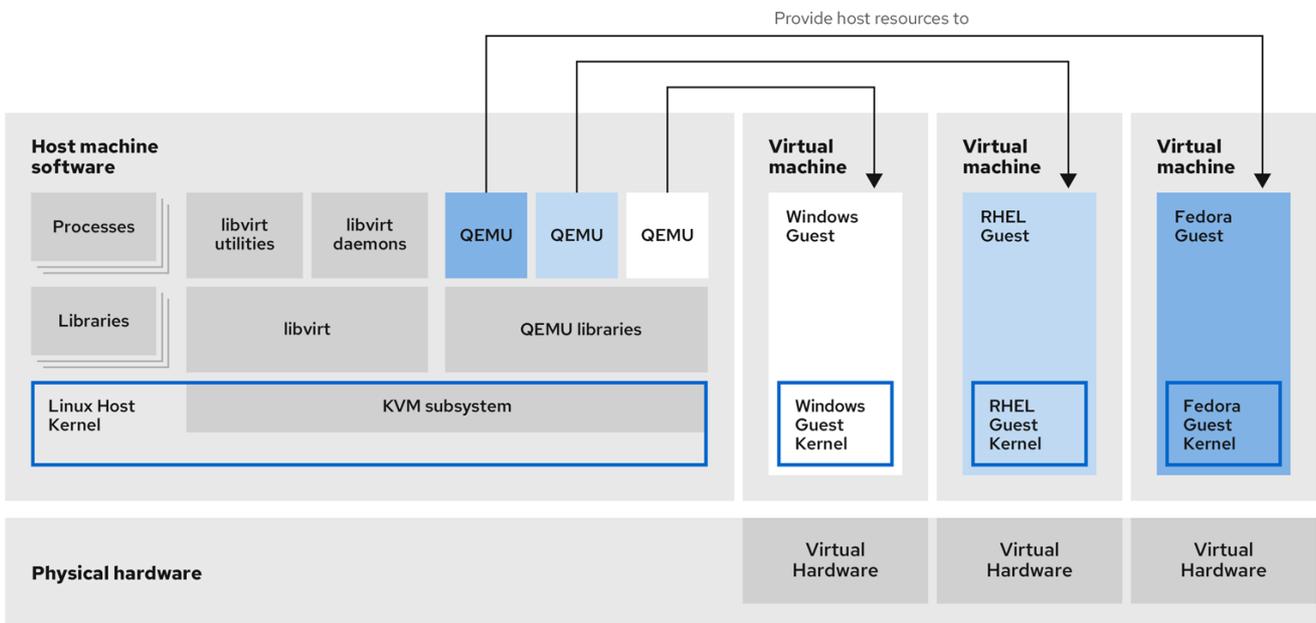


注記

QEMU はアーキテクチャーの必須コンポーネントですが、セキュリティに関する懸念があるため、RHEL 9 システムで直接使用することは意図されていません。したがって、Red Hat は、**qemu-*** コマンドをサポート対象外としており、libvirt を使用して QEMU と相互作用することを強く推奨します。

ホストベースのインターフェイスの詳細は、[仮想管理に使用するツールおよびインターフェイス](#) を参照してください。

図1.1 RHEL 9 の仮想アーキテクチャー



244_RHEL_0422

1.4. 仮想管理に使用するツールおよびインターフェイス

RHEL 9 の仮想化は、コマンドラインインターフェイス (CLI) または複数のグラフィカルユーザーインターフェイス (GUI) を使用して管理できます。

コマンドラインインターフェイス

CLI は、RHEL 9 で仮想化を管理する最も強力な方法です。仮想マシン (VM) 管理用の CLI コマンドでは、以下のものがよく知られています。

- **virsh** - 指定した引数に応じて、多種多様な目的を持つ多目的仮想コマンドラインユーティリティーおよびシェル。以下に例を示します。
 - 仮想マシンの起動およびシャットダウン - **virsh start** および **virsh shutdown**

- 利用可能な仮想マシンのリスト表示 - **virsh list**
- 設定ファイルからの仮想マシンの作成 - **virsh create**
- 仮想化シェルへの入力 - **virsh**

詳細は、**virsh(1)** man ページを参照してください。

- **virt-install** - 新しい仮想マシンを作成する CLI ユーティリティー。詳細は、**virt-install(1)** man ページを参照してください。
- **virt-xml** - 仮想マシンの設定を編集するユーティリティー。
- **guestfish** - 仮想マシンのディスクイメージを調べ、修正するユーティリティー。詳細は、**guestfish(1)** man ページを参照してください。

グラフィカルユーザーインターフェイス

以下の GUI を使用して、RHEL 9 で仮想化を管理できます。

- RHEL 9 の Web コンソール (Cockpit と呼ばれています) は、仮想マシンおよび仮想化ホストの管理用に、リモートからアクセスでき、簡単に使用できるグラフィカルユーザーインターフェイスを提供します。Web コンソールを使用した基本的な仮想化管理の手順については、[Managing virtual machines in the web console](#) を参照してください。

1.5. RED HAT の仮想化ソリューション

以下の Red Hat 製品は、RHEL 9 仮想化機能に構築されており、RHEL 9 で利用可能な KVM 仮想化機能を拡張します。また、[RHEL 9 仮想化の制限](#) の多くが、このような製品には適用されません。

OpenShift Virtualization

KubeVirt テクノロジーに基づいて、OpenShift Virtualization は Red Hat OpenShift Container Platform の一部であり、仮想マシンをコンテナで実行することができます。

OpenShift Virtualization の詳細は、[Red Hat ハイブリッドクラウド](#) のページを参照してください。

Red Hat OpenStack Platform (RHOSP)

Red Hat OpenStack Platform は、安全で信頼性の高いパブリックまたはプライベートの OpenStack クラウドを作成、デプロイ、および拡張するための統合基盤を提供します。

Red Hat OpenStack Platform の詳細は、[Red Hat OpenStack Platform の製品ページ](#)、または [Red Hat OpenStack Platform ドキュメントスイート](#) を参照してください。



注記

RHEL ではサポートされていませんが、他の Red Hat 仮想化ソリューションでサポートされている仮想化機能の詳細は、[RHEL 9 仮想化で対応していない機能](#) を参照してください。

第2章 仮想化の有効化

RHEL 9 で仮想化を使用するには、仮想化パッケージをインストールして、仮想マシンをホストするようにシステムを設定する必要があります。これを行うための具体的な手順は、CPU アーキテクチャーによって異なります。

2.1. AMD64 および INTEL 64 での仮想化の有効化

KVM ハイパーバイザーを設定し、RHEL 9 を実行している AMD64 または Intel 64 システムで仮想マシンを作成するには、以下の手順に従います。

前提条件

- Red Hat Enterprise Linux 9 が、ホストマシンに [インストールされ登録されている](#)。
- システムが仮想ホストとして機能するように、以下のハードウェア要件を満たしている。
 - ホストマシンのアーキテクチャーが [KVM 仮想化](#) に対応している。
 - 最低でも、以下のシステムリソースが利用できる。
 - ホスト用に 6 GB と、各仮想マシン用に 6 GB の空きディスク容量。
 - ホスト用に 2 GB と、各仮想マシン用に 2 GB の RAM。

手順

1. 仮想化ハイパーバイザーパッケージをインストールします。

```
# dnf install qemu-kvm libvirt virt-install virt-viewer
```

2. 仮想化サービスを起動します。

```
# for drv in qemu network nodedev nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done
```

検証

1. システムが仮想ホストとして準備されていることを確認します。

```
# virt-host-validate
[...]  
QEMU: Checking for device assignment IOMMU support      : PASS  
QEMU: Checking if IOMMU is enabled by kernel            : WARN (IOMMU appears to be  
disabled in kernel. Add intel_iommu=on to kernel cmdline arguments)  
LXC: Checking for Linux >= 2.6.26                      : PASS  
[...]  
LXC: Checking for cgroup 'blkio' controller mount-point : PASS  
LXC: Checking if device /sys/fs/fuse/connections exists : FAIL (Load the 'fuse' module to  
enable /proc/ overrides)
```

2. `virt-host-validate` のすべての項目で **PASS** 値が返された場合は、システムに [仮想マシンを作成する](#) 準備ができています。

いずれかの項目で **FAIL** が返された場合は、表示される指示に従って問題を解決してください。

いずれかの項目で **WARN** が返された場合は、表示される指示に従って仮想化機能を向上させることを検討してください。

トラブルシューティング

- KVM 仮想化がホスト CPU でサポートされていない場合は、`virt-host-validate` は以下の出力を生成します。

```
QEMU: Checking for hardware virtualization: FAIL (Only emulated CPUs are available,
performance will be significantly limited)
```

ただし、このようなホストシステムにある仮想マシンは、パフォーマンス上の問題が発生するのではなく、起動に失敗します。

これを回避するには、仮想マシンの XML 設定の `<domain type>` 値を `qemu` に変更します。ただし、Red Hat は `qemu` ドメインタイプを使用する仮想マシンに対応していないため、実稼働環境ではこれを設定しないことを強く推奨している点に注意してください。

次のステップ

- [RHEL 9 ホスト上での仮想マシンの作成](#)

2.2. IBM Z での仮想化の有効化

KVM ハイパーバイザーを設定し、RHEL 9 を実行している IBM Z システムで仮想マシンを作成するには、以下の手順に従います。

前提条件

- 最低でも、以下のシステムリソースが利用できる。
 - ホスト用に 6 GB と、各仮想マシン用に 6 GB の空きディスク容量。
 - ホスト用に 2 GB と、各仮想マシン用に 2 GB の RAM。
 - ホスト上の 4 つの CPU 通常、仮想マシンは、割り当てられた 1 つの vCPU で実行できますが、Red Hat は、高負荷時に仮想マシンが応答しなくならないように、仮想マシンごとに 2 つ以上の vCPU を割り当てることを推奨します。
- IBM Z ホストシステムでは、z13 以降の CPU を使用している。
- RHEL 9 が論理パーティション (LPAR) にインストールされている。また、LPAR が `start-interpretive execution (SIE)` 仮想機能に対応している。これを確認するには、`/proc/cpuinfo` ファイルで `sie` を検索します。

```
# grep sie /proc/cpuinfo
features      : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgrps te sie
```

手順

1. 仮想化パッケージをインストールします。

```
# dnf install qemu-kvm libvirt virt-install
```

2. 仮想化サービスを起動します。

```
# for drv in qemu network nodedev nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done
```

検証

1. システムが仮想ホストとして準備されていることを確認します。

```
# virt-host-validate
[...]
QEMU: Checking if device /dev/kvm is accessible      : PASS
QEMU: Checking if device /dev/vhost-net exists       : PASS
QEMU: Checking if device /dev/net/tun exists        : PASS
QEMU: Checking for cgroup 'memory' controller support : PASS
QEMU: Checking for cgroup 'memory' controller mount-point : PASS
[...]
```

2. `virt-host-validate` のすべての項目で **PASS** 値が返された場合は、システムに [仮想マシンを作成する](#) 準備ができています。
いずれかの項目で **FAIL** が返された場合は、表示される指示に従って問題を解決してください。

いずれかの項目で **WARN** が返された場合は、表示される指示に従って仮想化機能を向上させることを検討してください。

トラブルシューティング

- KVM 仮想化がホスト CPU でサポートされていない場合は、`virt-host-validate` は以下の出力を生成します。

```
QEMU: Checking for hardware virtualization: FAIL (Only emulated CPUs are available,
performance will be significantly limited)
```

ただし、このようなホストシステムにある仮想マシンは、パフォーマンス上の問題が発生するのではなく、起動に失敗します。

これを回避するには、仮想マシンの XML 設定の `<domain type>` 値を `qemu` に変更します。ただし、Red Hat は `qemu` ドメインタイプを使用する仮想マシンに対応していないため、実稼働環境ではこれを設定しないことを強く推奨している点に注意してください。

関連情報

- [IBM Z の仮想化と、AMD64 および Intel 64 の仮想化の相違点](#)

2.3. ARM 64 での仮想化の有効化

RHEL 9 を実行する ARM 64 システム (**AArch64** と呼ばれます) 上で仮想マシン (VM) を作成するための KVM ハイパーバイザーをセットアップするには、以下の手順に従います。

前提条件

- ホストシステムとゲストシステムは、64 KB のメモリーページサイズのカーネルを使用します。このようなカーネルを RHEL システムにインストールするには、[Kernel-64k を使用した ARM への RHEL のインストール](#) を参照してください。
- 最低でも、以下のシステムリソースが利用できる。
 - ホスト用に 6 GB と、各ゲスト用に 6 GB の空きディスク容量
 - ホスト用に 4 GB の RAM と、対象のゲストごとにさらに 4 GB。

手順

1. 仮想化パッケージをインストールします。

```
# dnf install qemu-kvm libvirt virt-install
```

2. 仮想化サービスを起動します。

```
# for drv in qemu network nodedev nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done
```

検証

1. システムが仮想ホストとして準備されていることを確認します。

```
# virt-host-validate
[...]
QEMU: Checking if device /dev/vhost-net exists      : PASS
QEMU: Checking if device /dev/net/tun exists       : PASS
QEMU: Checking for cgroup 'memory' controller support : PASS
QEMU: Checking for cgroup 'memory' controller mount-point : PASS
[...]
QEMU: Checking for cgroup 'blkio' controller support : PASS
QEMU: Checking for cgroup 'blkio' controller mount-point : PASS
QEMU: Checking if IOMMU is enabled by kernel       : WARN (Unknown if this platform
has IOMMU support)
```

2. `virt-host-validate` のすべての項目で **PASS** 値が返された場合は、システムに [仮想マシンを作成](#) できます。
いずれかの項目で **FAIL** が返された場合は、表示される指示に従って問題を解決してください。

いずれかの項目で **WARN** が返された場合は、表示される指示に従って仮想化機能を向上させることを検討してください。

次のステップ

- [仮想マシンの作成](#)

関連情報

- [ARM 64 での仮想化が AMD64 および Intel 64 とどのように異なるか](#)

2.4. 仮想マシンでの QEMU ゲストエージェント機能の有効化

RHEL 9 システムでホストされている仮想マシンの特定の機能を使用するには、まず QEMU ゲストエージェント (GA) を使用するように仮想マシンを設定する必要があります。

これらの機能の完全なリストについては、[QEMU ゲストエージェントを必要とする仮想化機能](#) を参照してください。

仮想マシン上で QEMU GA を設定するために必要な具体的な手順は、仮想マシンが使用するゲストオペレーティングシステムによって異なります。

- Linux 仮想マシンの場合は、[Linux ゲストでの QEMU ゲストエージェントの有効化](#) を参照してください。
- Windows 仮想マシンの場合は、[Windows ゲストでの QEMU ゲストエージェントの有効化](#) を参照してください。

2.4.1. Linux ゲストでの QEMU ゲストエージェントの有効化

RHEL ホストが Linux 仮想マシン上で [特定の操作のサブセット](#) を実行できるようにするには、QEMU ゲストエージェント (GA) を有効にする必要があります。

実行中の仮想マシンとシャットダウンした仮想マシンの両方で、QEMU GA を有効にできます。

手順

1. QEMU GA の XML 設定ファイル (例: `qemuga.xml`) を作成します。

```
# touch qemuga.xml
```

2. ファイルに以下の行を追加します。

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/f16x86_64.agent' />
  <target type='virtio' name='org.qemu.guest_agent.0' />
</channel>
```

3. XML ファイルを使用して、仮想マシンの設定に QEMU GA を追加します。

- 仮想マシンが実行中の場合は、次のコマンドを使用します。

```
# virsh attach-device <vm-name> qemuga.xml --live --config
```

- 仮想マシンがシャットダウンされている場合は、次のコマンドを使用します。

```
# virsh attach-device <vm-name> qemuga.xml --config
```

4. Linux ゲストオペレーティングシステムで、QEMU GA をインストールします。

```
# dnf install qemu-guest-agent
```

5. ゲストで QEMU GA サービスを起動します。

```
# systemctl start qemu-guest-agent
```

検証

QEMU GA が Linux 仮想マシンで有効化および実行されていることを確認するには、次のいずれかを実行します。

- ゲストオペレーティングシステムで、**systemctl status qemu-guest-agent | grep Loaded** コマンドを使用します。出力に **enabled** が含まれる場合、仮想マシン上で QEMU GA がアクティブになっています。
- ホストで **virsh domfsinfo <vm-name>** コマンドを使用します。何らかの出力が表示された場合、指定した仮想マシン上で QEMU GA がアクティブになっています。

関連情報

- [QEMU ゲストエージェントを必要とする仮想化機能](#)

2.4.2. Windows ゲストでの QEMU ゲストエージェントの有効化

RHEL ホストが Windows 仮想マシン上で [特定の操作のサブセット](#) を実行できるようにするには、QEMU ゲストエージェント (GA) を有効にする必要があります。これを行うには、QEMU ゲストエージェントインストーラーを含むストレージデバイスを、既存の仮想マシンに追加するか、新しい仮想マシンを作成するときに追加し、Windows ゲストオペレーティングシステムにドライバーをインストールします。

グラフィカルインターフェイスを使用してゲストエージェント (GA) をインストールするには、以下の手順を参照してください。コマンドラインインターフェイスで GA をインストールするには、[Microsoft Windows Installer \(MSI\)](#) を使用してください。

前提条件

- ゲストエージェントを含むインストールメディアが仮想マシンに接続されている。メディアの準備手順は、[Preparing virtio driver installation media on a host machine](#) を参照してください。

手順

1. Windows ゲストオペレーティングシステムで、**File Explorer** アプリケーションを開きます。
2. **この PC** をクリックします。
3. **デバイスおよびドライブ** ペインで、**virtio-win** メディアを開きます。
4. **guest-agent** フォルダーを開きます。
5. 仮想マシンにインストールされているオペレーティングシステムに基づいて、次のいずれかのインストーラーを実行します。
 - 32 ビットオペレーティングシステムを使用している場合は、**qemu-ga-i386.msi** インストーラーを実行します。
 - 64 ビットオペレーティングシステムを使用している場合は、**qemu-ga-x86_64.msi** インストーラーを実行します。
6. **オプション**: ホストと Windows ゲスト間の通信インターフェイスとして準仮想化シリアルドライバー (**virtio-serial**) を使用する場合は、**virtio-serial** ドライバーが Windows ゲストにインストールされていることを確認します。**virtio** ドライバーのインストールの詳細は、[Windows ゲ](#)

ストへの virtio ドライバーのインストールを参照してください。

検証

1. Windows 仮想マシンで、**Services** ウィンドウに移動します。
Computer Management > Services
2. **QEMU Guest Agent** のステータスが **Running** であることを確認します。

関連情報

- [QEMU ゲストエージェントを必要とする仮想化機能](#)

2.4.3. QEMU ゲストエージェントを必要とする仮想化機能

仮想マシン (VM) で QEMU ゲストエージェント (GA) を有効にすると、ホスト上で次のコマンドを使用して仮想マシンを管理できます。

virsh shutdown --mode=agent

このシャットダウン方法は、**virsh shutdown --mode=acpi** よりも信頼性が高くなります。これは、QEMU GA で使用する **virsh shutdown** は、確実にクリーンな状態で協調ゲストをシャットダウンするためです。

virsh domfsfreeze および virsh domfsthaw

ゲストファイルシステムを分離してフリーズします。

virsh domfstrim

ゲストにファイルシステムをトリミングするように指示します。これにより、移行中に転送する必要のあるデータを削減できます。



重要

このコマンドを使用して Linux 仮想マシンを管理する場合は、ゲストオペレーティングシステムで次の SELinux ブール値も設定する必要があります。

```
# setsebool virt_qemu_ga_read_nonsecurity_files on
```

virsh domtime

ゲストの時計をクエリーまたは設定します。

virsh setvcpus --guest

ゲストに CPU をオフラインにするように指示します。これは、CPU をホットアンプラグできない場合に便利です。

virsh domifaddr --source agent

QEMU GA を使用してゲストオペレーティングシステムの IP アドレスをクエリーします。たとえば、ゲストインターフェイスがホストインターフェイスに直接接続されている場合に便利です。

virsh domfsinfo

実行中のゲストにマウントされているファイルシステムのリストを表示します。

virsh set-user-password

ゲストの特定のユーザーアカウントのパスワードを設定します。

virsh set-user-sshkeys

ゲストの特定のユーザーの認可された SSH 鍵ファイルを編集します。



重要

このコマンドを使用して Linux 仮想マシンを管理する場合は、ゲストオペレーティングシステムで次の SELinux ブール値も設定する必要があります。

```
# setsebool virt_qemu_ga_manage_ssh on
```

関連情報

- [Linux ゲストでの QEMU ゲストエージェントの有効化](#)
- [Windows ゲストでの QEMU ゲストエージェントの有効化](#)

第3章 仮想マシンの作成

RHEL 9 で仮想マシンを作成する場合は、[コマンドラインインターフェイス](#) または [RHEL 9 Web コンソール](#) を使用します。

3.1. コマンドラインインターフェイスを使用した仮想マシンの作成

virt-install ユーティリティを使用して、RHEL 9 ホストで仮想マシンを作成するには、以下の手順に従ってください。

前提条件

- ホストシステムで仮想化が **有効** になっている。
- ディスク領域、RAM、CPU など、仮想マシンに割り当てるのに十分なシステムリソースがある。推奨される値は、仮想マシンで行うタスクやワークロードにより大きく異なる可能性があります。
- オペレーティングシステム (OS) のインストールソースがローカルまたはネットワークで利用できる。これには、次のいずれかを使用できます。
 - インストールメディアの ISO イメージ
 - 既存の仮想マシンインストールのディスクイメージ



警告

RHEL 9 では、ホストの CD-ROM デバイスまたは DVD-ROM デバイスからインストールすることができません。RHEL 9 で利用可能な仮想マシンのインストール方法を使用する際に、インストールソースに CD-ROM または DVD-ROM を選択するとインストールに失敗します。詳細は [Red Hat ナレッジベース](#) を参照してください。

また、Red Hat は、[限られたゲストオペレーティングシステムのセット](#) のみをサポートしていることにも注意してください。

- 任意: インストールをより速く、簡単に設定するために、キックスタートファイルを利用できません。

手順

仮想マシンを作成して OS のインストールを開始するには、以下の必須引数を指定して、**virt-install** コマンドを使用します。

- **--name**: 新しいマシンの名前
- **--memory**: 割り当てるメモリーの量
- **--vcpus**: 割り当てる仮想 CPU の数
- **--disk**: 割り当てるストレージのタイプとサイズ

- **--cdrom** または **--location**: OS インストールソースのタイプと場所

選択したインストール方法に応じて、必要なオプションと値が異なります。例については、以下のコマンドを参照してください。

- 次のコマンドでは、**demo-guest1** という名前の仮想マシンを作成し、ローカルの **/home/username/Downloads/Win10install.iso** ファイルに保存されている ISO イメージから、Windows 10 OS をインストールします。この仮想マシンには、2048 MiB の RAM と 2 つの vCPU が割り当てられ、80 GiB の qcow2 仮想ディスクも自動的に割り当てられます。

```
# virt-install \
  --name demo-guest1 --memory 2048 \
  --vcpus 2 --disk size=80 --os-variant win10 \
  --cdrom /home/username/Downloads/Win10install.iso
```

- 次のコマンドは、**demo-guest2** という名前の仮想マシンを作成し、**/home/username/Downloads/rhel9.iso** イメージを使用して、ライブ CD から RHEL 9 OS を実行します。この仮想マシンにはディスク領域が割り当てられないため、セッション中に行った変更は保持されません。また、仮想マシンには、4096 MiB の RAM と、4 つの vCPU が割り当てられます。

```
# virt-install \
  --name demo-guest2 --memory 4096 --vcpus 4 \
  --disk none --livecd --os-variant rhel9.0 \
  --cdrom /home/username/Downloads/rhel9.iso
```

- 次のコマンドは、**demo-guest3** という名前の RHEL 9 仮想マシンを作成し、既存のディスクイメージ **/home/username/backup/disk.qcow2** に接続します。これは、マシン間でハードドライブを物理的に移動するのと似ています。したがって、**demo-guest3** で使用できる OS およびデータは、イメージが処理された方法により決定します。また、仮想マシンには、2048 MiB の RAM および 2 つの vCPU が割り当てられます。

```
# virt-install \
  --name demo-guest3 --memory 2048 --vcpus 2 \
  --os-variant rhel9.0 --import \
  --disk /home/username/backup/disk.qcow2
```

ディスクイメージをインポートする場合は、**--os-variant** オプションを使用することが強く推奨されます。このオプションを指定しないと、作成された仮想マシンのパフォーマンスに影響を及ぼします。

- 次のコマンドは、**demo-guest4** という名前の仮想マシンを作成し、URL **http://example.com/OS-install** からインストールします。インストールを開始するには、作業中の OS インストールツリーを URL に指定する必要があります。さらに、OS は、キックスタートファイル **/home/username/ks.cfg** で自動的に設定されます。この仮想マシンには、2048 MiB の RAM、2 つの vCPU、および 160 GiB の qcow2 仮想ディスクも割り当てられます。

```
# virt-install \
  --name demo-guest4 --memory 2048 --vcpus 2 --disk size=160 \
  --os-variant rhel9.0 --location http://example.com/OS-install \
  --initrd-inject /home/username/ks.cfg --extra-args="inst.ks=file:/ks.cfg console=tty0 console=ttyS0,115200n8"
```

さらに、ARM 64 ホスト上の RHEL 9 で demo-guest4 をホストする場合は、キックスタートファイルによって **kernel-64k** パッケージが確実にインストールされるように、次の行を追加します。

```
%packages
-kernel
kernel-64k
%end
```

- 次のコマンドは、**demo-guest5** という名前の仮想マシンを作成し、グラフィックスがない、テキストのみのモードである **RHEL9.iso** イメージファイルからインストールします。ゲストコンソールをシリアルコンソールに接続します。仮想マシンには、16384 MiB のメモリ、16 個の vCPU、および 280 GiB のディスクが割り当てられます。このようなインストールは、低速なネットワークリンクを介してホストに接続する際に便利です。

```
# virt-install \
  --name demo-guest5 --memory 16384 --vcpus 16 --disk size=280 \
  --os-variant rhel9.0 --location RHEL9.iso \
  --graphics none --extra-args='console=ttyS0'
```

- 次のコマンドは、**demo-guest6** という名前の仮想マシンを作成します。この仮想マシンの設定は demo-guest5 と同じですが、リモートホスト 192.0.2.1 に置かれます。

```
# virt-install \
  --connect qemu+ssh://root@192.0.2.1/system --name demo-guest6 --memory 16384 \
  --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso \
  --graphics none --extra-args='console=ttyS0'
```

- 次のコマンドは、**demo-guest-7** という名前の仮想マシンを作成します。この仮想マシンの設定は demo-guest5 と同じですが、ストレージとして DASD 仲介デバイス **mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8** を使用し、デバイス番号 **1111** を割り当てます。

```
# virt-install \
  --name demo-guest7 --memory 16384 --vcpus 16 --disk size=280 \
  --os-variant rhel9.0 --location RHEL9.iso --graphics none \
  --disk none --hostdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8,address.type=ccw,address.cssid
=0xfe,address.ssid=0x0,address.devno=0x1111,boot-order=1 \
  --extra-args 'rd.dasd=0.0.1111'
```

インストールに利用可能な仲介デバイスの名前は、**virsh nodedev-list --cap mdev** コマンドを使用して取得できることに注意してください。

検証

- 仮想マシンが問題なく作成されると、仮想マシンのグラフィカルコンソールで **virt-viewer** 画面が開き、ゲスト OS のインストールが開始します。

トラブルシューティング

- **virt-install** が **cannot find default network** エラーを出力する場合は、以下のようにします。
 - **libvirt-daemon-config-network** 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。

-

```
# {PackageManagerCommand} info libvirt-daemon-config-network
Installed Packages
Name      : libvirt-daemon-config-network
[...]
```

- **libvirt** のデフォルトネットワークがアクティブで、自動的に起動するように設定されていることを確認します。

```
# virsh net-list --all
Name    State  Autostart  Persistent
-----
default active   yes        yes
```

- そうでない場合は、デフォルトのネットワークをアクティブにし、自動起動に設定します。

```
# virsh net-autostart default
Network default marked as autostarted

# virsh net-start default
Network default started
```

- デフォルトのネットワークをアクティベートしても以下のエラーが出て失敗する場合は、**libvirt-daemon-config-network** パッケージが正常にインストールされていません。

```
error: failed to get network 'default'
error: Network not found: no network with matching name 'default'
```

この問題を修正するには、以下のコマンドで **libvirt-daemon-config-network** を再インストールします。

```
# {PackageManagerCommand} reinstall libvirt-daemon-config-network
```

- 以下のようなエラーでデフォルトのネットワークをアクティベートできない場合には、デフォルトネットワークのサブネットとホストの既存インターフェイスで競合が発生しています。

```
error: Failed to start network default
error: internal error: Network is already in use by interface ens2
```

これを修正するには、**virsh net-edit default** コマンドを使用して、設定の **192.0.2.*** の値を、ホストで使用していないサブネットに変更します。

関連情報

- [virt-install\(1\) man ページ](#)
- [Web コンソールを使用した仮想マシンの作成、およびゲストのオペレーティングシステムのインストール](#)
- [仮想マシンのクローン作成](#)

3.2. WEB コンソールを使用した仮想マシンの作成、およびゲストのオペレーティングシステムのインストール

RHEL 9 ホストの GUI で仮想マシンを管理するには、Web コンソールを使用します。次のセクションでは、RHEL 9 Web コンソールを使用して仮想マシンを作成し、仮想マシンにゲストオペレーティングシステムをインストールする方法を説明します。

3.2.1. Web コンソールを使用した仮想マシンの作成

RHEL 9 Web コンソールが接続しているホストマシン上に仮想マシン (VM) を作成するには、以下の手順を使用します。

前提条件

- [ホストシステムで仮想化が有効になっている](#)。
- [Web コンソールの仮想マシンプラグインがホストシステムにインストールされている](#)。
- ディスク領域、RAM、CPU など、仮想マシンに割り当てるのに十分なシステムリソースがある。推奨される値は、仮想マシンで行うタスクやワークロードにより大きく異なる可能性がある。

手順

1. Web コンソールの **Virtual Machines** インターフェイスで、**Create VM** をクリックします。**Create new virtual machine** ダイアログが表示されます。

Create new virtual machine ✕

Name

Details Automation

Connection ⓘ System User session

Installation type ▼

Operating system ▼

Storage ▼

Storage limit ▼

2. 作成する仮想マシンの基本設定を入力します。
 - **名前** - 仮想マシンの名前

- **接続** - セッションに付与される権限のレベル。詳細は、Web コンソールで関連するダイアログボックスをデプロイメントしてください。
- **インストールタイプ** - インストールでは、ローカルのインストールメディア、URL、PXE ネットワークブート、クラウドベースイメージを使用したり、または限定されたオペレーティングシステムのセットからオペレーティングシステムをダウンロードしたりできます。
- **Operating system** - 仮想マシン上で実行されているゲストオペレーティングシステム。Red Hat がサポートするのは、[限られたゲストオペレーティングシステムのセット](#)のみです。



注記

Web コンソールから Red Hat Enterprise Linux を直接ダウンロードしてインストールする場合は、**Offline token** フィールドにオフライントークンを追加する必要があります。

- **Storage** - ストレージのタイプ。
 - **Storage Limit** - ストレージ領域の容量。
 - **Memory** - メモリーの容量。
3. 仮想マシンを作成します。
- 仮想マシンでオペレーティングシステムを自動的にインストールする場合は、**Create and run** をクリックします。
 - オペレーティングシステムをインストールする前に仮想マシンを編集する場合は、**Create and edit** をクリックします。

次のステップ

- [Web コンソールを使用したゲストのオペレーティングシステムのインストール](#)

関連情報

- [コマンドラインインターフェイスを使用した仮想マシンの作成](#)

3.2.2. Web コンソールでディスクイメージをインポートして仮想マシンを作成する手順

RHEL 9 Web コンソールで既存の仮想マシンインストールのディスクイメージをインポートすることで、仮想マシン (VM) を作成できます。

前提条件

- [Web コンソールの仮想マシンプラグインがシステムにインストールされている](#)。
- ディスク領域、RAM、CPU など、仮想マシンに割り当てするのに十分なシステムリソースがある。推奨値は、仮想マシンで行うタスクやワークロードにより大きく異なる可能性があります。
- 既存の仮想マシンインストールのディスクイメージがダウンロードされている。

手順

1. Web コンソールの **Virtual Machines** インターフェイスで、**Import VM** をクリックします。
Import a virtual machine ダイアログが表示されます。

2. 作成する仮想マシンの基本設定を入力します。
 - **名前** - 仮想マシンの名前
 - **ディスクイメージ** - ホストシステム上の仮想マシンに存在するディスクイメージのパスです。
 - **Operating system** - 仮想マシンディスク上で実行されているオペレーティングシステム。Red Hat がサポートするのは、[限られたゲストオペレーティングシステムのセット](#)のみです。
 - **Memory** - 仮想マシンによる使用のために割り当てるメモリーの容量。
3. 仮想マシンをインポートします。
 - 仮想マシン設定をさらに編集せずに仮想マシンにオペレーティングシステムをインストールするには、**Import and run** をクリックします。
 - オペレーティングシステムのインストール前に仮想マシン設定を編集するには、**Import and edit** をクリックします。

3.2.3. Web コンソールを使用したゲストのオペレーティングシステムのインストール

仮想マシンを初めて起動するときは、仮想マシンにオペレーティングシステムをインストールする必要があります。



注記

新しい仮想マシンを作成するときに **Create and run** または **Import and run** をクリックすると、仮想マシン作成時にオペレーティングシステムのインストールルーチンが自動的に開始されます。

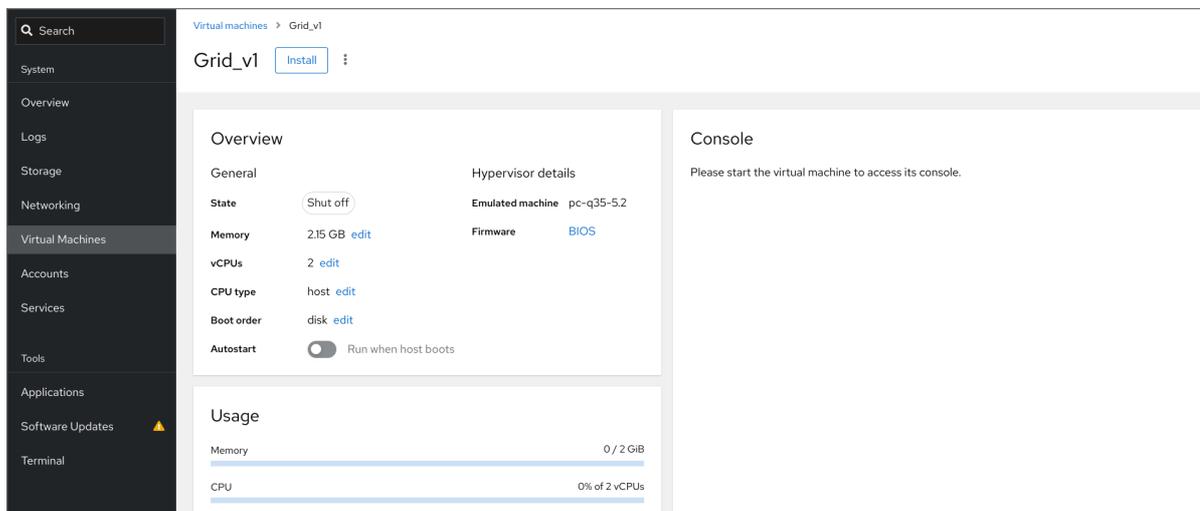
前提条件

- Web コンソールの仮想マシンプラグインがホストシステムにインストールされている。

手順

1. **Virtual Machines** インターフェイスで、ゲスト OS をインストールする仮想マシンをクリックします。

選択した仮想マシンの基本情報を含む新しいページが開き、仮想マシンのさまざまな側面を管理するための制御を行います。



2. **任意:** ファームウェアを変更します。



注記

新しい仮想マシンの作成時に **Create and edit** または **Import and edit** を選択し、かつ仮想マシンに OS がまだインストールされていない場合にのみ、ファームウェアを変更できます。

+ ..ファームウェアをクリックします。

- a. **Change Firmware** ウィンドウで、必要なファームウェアを選択します。
 - b. **Save** をクリックします。
3. **インストール** をクリックします。
仮想マシンコンソールで、オペレーティングシステムのインストールルーチンが実行します。

トラブルシューティング

- インストールルーチンが失敗した場合は、インストールを再度開始する前に、仮想マシンを削除して再作成します。

3.2.4. Web コンソールを使用したクラウドイメージ認証による仮想マシンの作成

デフォルトでは、ディストリビューションクラウドイメージにはログインアカウントがありません。ただし、RHEL Web コンソールを使用して、仮想マシンを作成し、root アカウントとユーザーアカウントのログイン認証情報を指定して、cloud-init に渡すことができるようになりました。

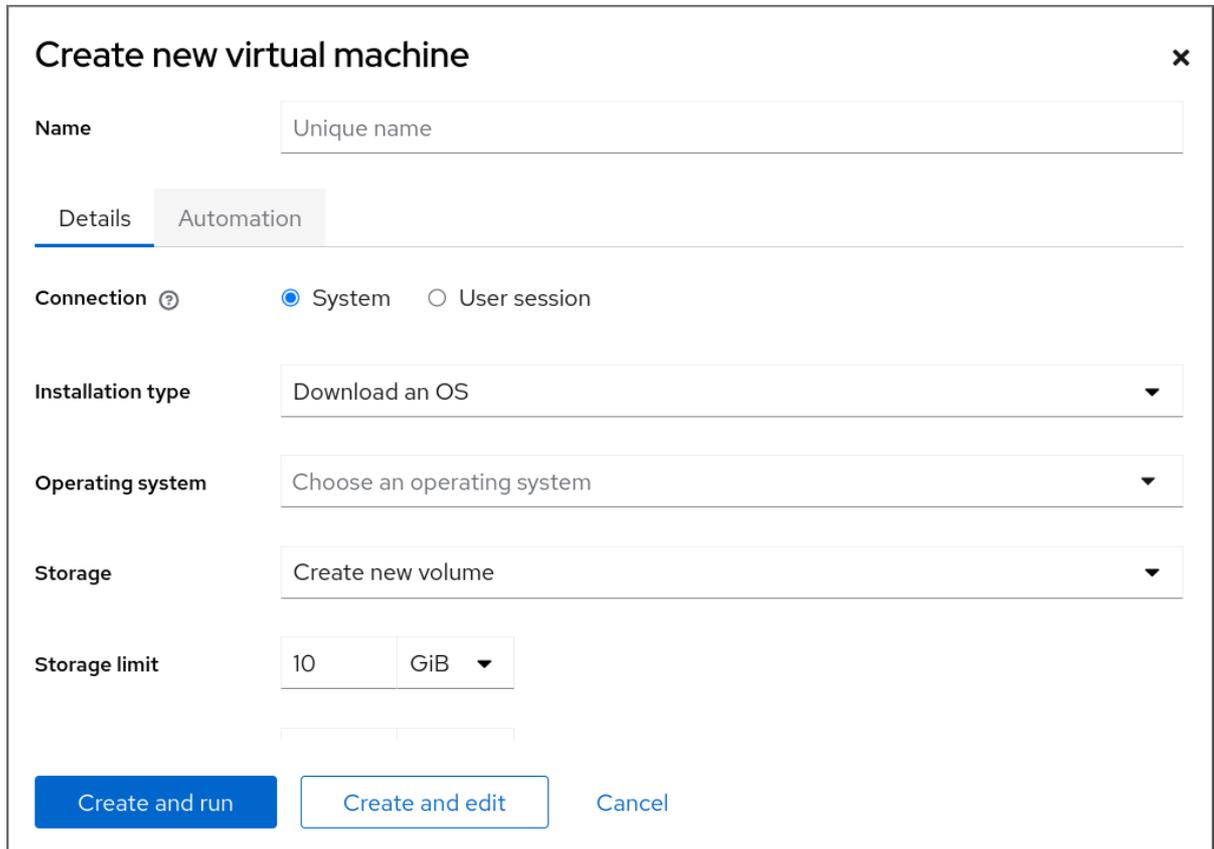
前提条件

- Web コンソールの仮想マシンプラグインが **システムにインストールされている**。
- ホストシステムで仮想化が **有効** になっている。

- ディスク領域、RAM、CPU など、仮想マシンに割り当てるのに十分なシステムリソースがある。推奨される値は、仮想マシンで行うタスクやワークロードにより大きく異なる可能性があります。

手順

1. Web コンソールの **Virtual Machines** インターフェイスで、**Create VM** をクリックします。仮想マシンの新規作成ダイアログが表示されます。



Create new virtual machine ×

Name

Details Automation

Connection ? System User session

Installation type ▼

Operating system ▼

Storage ▼

Storage limit ▼

2. **名前** フィールドに、仮想マシンの名前を入力します。
3. **Details** タブの **Installation type** フィールドで、**Cloud base image** を選択します。

Create new virtual machine ✕

Name

Details Automation

Installation type

Installation source

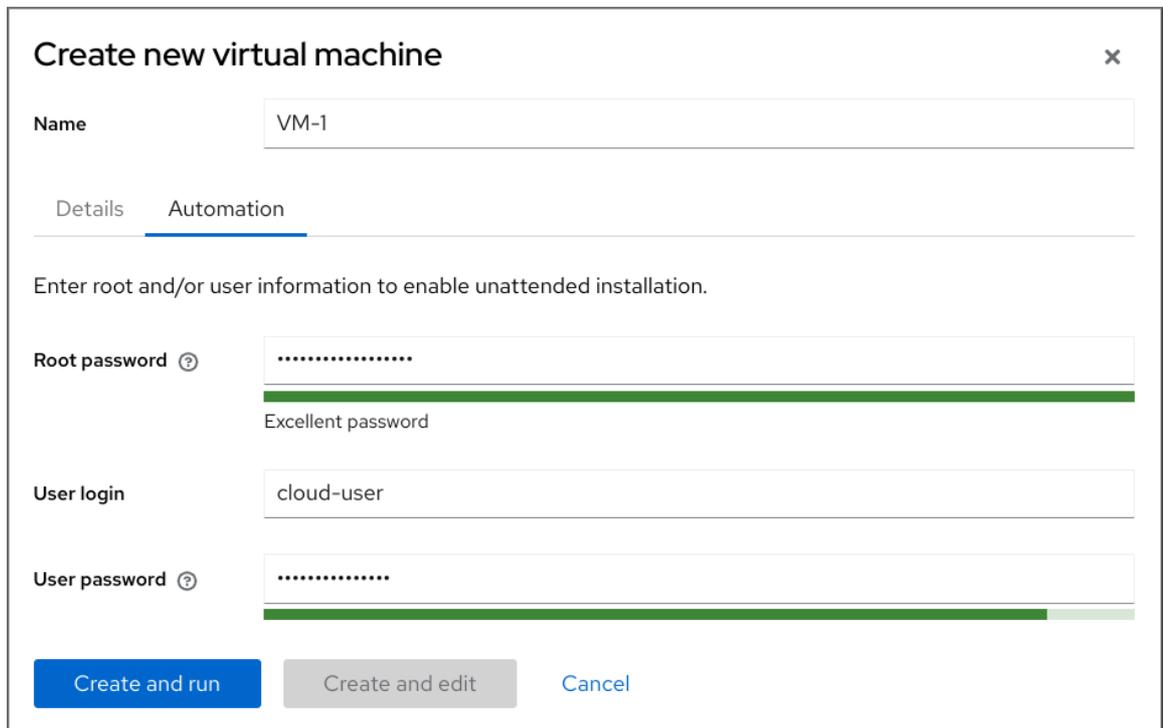
Operating system

Storage

Storage Limit 198.8 GiB available at default location

Memory 15.2 GiB available on host

4. インストールソース フィールドで、ホストシステム上のイメージファイルへのパスを設定します。
5. 作成する仮想マシンの設定を入力します。
 - **オペレーティングシステム** - 仮想マシンのオペレーティングシステム。Red Hat がサポートするのは、[限られたゲストオペレーティングシステムのセット](#)のみです。
 - **ストレージ** - 仮想マシンを設定するストレージの種類
 - **ストレージのサイズ** - 仮想マシンを設定するストレージ容量
 - **メモリー** - 仮想マシンを設定するメモリーのサイズ
6. **Automation** タブをクリックします。
クラウド認証の認証情報を設定します。
 - **root パスワード** - 仮想マシンの root パスワードを入力します。root パスワードを設定しない場合は、フィールドを空白のままにします。
 - **ユーザーログイン** - cloud-init ユーザーログインを入力します。ユーザーアカウントを作成しない場合は、このフィールドを空白のままにします。
 - **ユーザーパスワード** - パスワードを入力します。ユーザーアカウントを作成しない場合は、このフィールドを空白のままにします。



Create new virtual machine ×

Name

Details Automation

Enter root and/or user information to enable unattended installation.

Root password ⓘ
Excellent password

User login

User password ⓘ

7. **Create and run** をクリックします。
仮想マシンが作成されます。

関連情報

- [仮想マシンへのオペレーティングシステムのインストール](#)

第4章 仮想マシンの起動

RHEL 9 で仮想マシンを起動する場合は、[コマンドインターフェイス](#) または [Web コンソール GUI](#) を使用できます。

前提条件

- 仮想マシンを起動する前に仮想マシンを作成しておく。理想としては、OS をインストールしておく。手順は、[仮想マシンの作成](#) を参照してください。

4.1. コマンドラインインターフェイスでの仮想マシンの起動

コマンドラインインターフェイス (CLI) を使用して、シャットダウンした仮想マシン (VM) を起動するか、保存した仮想マシンを復元します。CLI を使用すると、ローカル仮想マシンとリモート仮想マシンの両方を起動できます。

前提条件

- すでに定義されている非アクティブな仮想マシン
- 仮想マシンの名前
- リモート仮想マシンの場合は、以下も設定されている。
 - 仮想マシンが置かれているホストの IP アドレス
 - ホストへの root アクセス権限

手順

- ローカルの仮想マシンには、**virsh start** ユーティリティーを使用します。たとえば、次のコマンドは仮想マシン **demo-guest1** を起動します。

```
# virsh start demo-guest1
Domain 'demo-guest1' started
```

- リモートホストにある仮想マシンでは、ホストへの QEMU+SSH 接続と共に **virsh start** ユーティリティーを使用します。たとえば、次のコマンドは、ホスト 192.0.2.1 にある仮想マシン **demo-guest1** を起動します。

```
# virsh -c qemu+ssh://root@192.0.2.1/system start demo-guest1

root@192.0.2.1's password:

Domain 'demo-guest1' started
```

関連情報

- **virsh start --help** コマンド
- [リモートの仮想化ホストへの簡単なアクセスの設定](#)
- [ホストの起動時に仮想マシンを自動的に起動する](#)

4.2. WEB コンソールを使用した仮想マシンの起動

仮想マシンが **停止** 状態にある場合は、RHEL 9 Web コンソールを使用して起動できます。ホストの起動時に、仮想マシンが自動的に起動するように設定することもできます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- すでに定義されている非アクティブな仮想マシン
- 仮想マシンの名前

手順

1. **仮想マシン** インターフェイスで、起動する仮想マシンをクリックします。
選択した仮想マシンの詳細情報を含む新しいページが開き、仮想マシンのシャットダウンおよび削除を制御できます。
2. **Run** をクリックします。
仮想マシンが起動し、[そのコンソールまたはグラフィカル出力に接続](#) できます。
3. **オプション**: ホスト起動時に仮想マシンが自動的に起動するように設定するには、**Overview** セクションの **Autostart** チェックボックスを切り替えます。
libvirt が管理していないネットワークインターフェイスを使用する場合は、systemd 設定も変更する必要があります。そうしないと、影響を受ける仮想マシンが起動できなくなる可能性があります。[starting virtual machines automatically when the host starts](#) を参照してください。

関連情報

- [Web コンソールで仮想マシンのシャットダウン](#)
- [Web コンソールを使用した仮想マシンの再起動](#)

4.3. ホストの起動時に仮想マシンを自動的に起動する

実行中の仮想マシン (VM) のホストが再起動すると、仮想マシンはシャットダウンされるため、デフォルトで手動で再起動する必要があります。ホストの実行中に仮想マシンがアクティブであることを確認するには、仮想マシンが自動的に起動するように設定できます。

前提条件

- [作成された仮想マシン](#) がある

手順

1. **virsh autostart** ユーティリティを使用して、ホストの起動時に仮想マシンが自動的に起動するように設定します。
たとえば、次のコマンドは、**demo-guest1** 仮想マシンを自動的に起動するように設定します。

```
# virsh autostart demo-guest1
Domain 'demo-guest1' marked as autostarted
```

2. **libvirt** が管理していないネットワークインターフェイスを使用する場合は、**systemd** 設定にも追加の変更を行う必要があります。これを行わないと、影響を受ける仮想マシンの起動に失敗する可能性があります。



注記

このようなインターフェイスには、以下の例が含まれます。

- **NetworkManager** が作成したブリッジデバイス
- `<forward mode='bridge'/>` を使用するように設定されたネットワーク

- a. **systemd** 設定ディレクトリツリーに、**virtqemud.service.d** ディレクトリが存在しない場合は作成します。

```
# mkdir -p /etc/systemd/system/virtqemud.service.d/
```

- b. 以前に作成したディレクトリに、**10-network-online.conf** **systemd** ユニットオーバーライドファイルを作成します。このファイルのコンテンツは、**virtqemud** サービスのデフォルトの **systemd** 設定を上書きします。

```
# touch /etc/systemd/system/virtqemud.service.d/10-network-online.conf
```

- c. **10-network-online.conf** ファイルに以下の行を追加します。この設定変更により、ホストのネットワークの準備ができてから、**systemd** が **virtqemud** サービスを起動するようになります。

```
[Unit]
After=network-online.target
```

検証

1. 仮想マシンの設定を表示し、**自動開始** オプションが有効になっていることを確認します。たとえば、次のコマンドは、**自動開始** オプションなど、**demo-guest1** 仮想マシンの基本情報を表示します。

```
# virsh dominfo demo-guest1
Id:          2
Name:        demo-guest1
UUID:        e46bc81c-74e2-406e-bd7a-67042bae80d1
OS Type:     hvm
State:       running
CPU(s):      2
CPU time:    385.9s
Max memory:  4194304 KiB
Used memory: 4194304 KiB
Persistent:  yes
Autostart:   enable
Managed save: no
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c873,c919 (enforcing)
```

2. libvirt が管理していないネットワークインターフェイスを使用する場合は、**10-network-online.conf** ファイルの内容が次の出力と一致するかどうかを確認してください。

```
$ cat /etc/systemd/system/virtqemu.service.d/10-network-online.conf
[Unit]
After=network-online.target
```

関連情報

- **virsh autostart --help** コマンド
- [Web コンソールを使用した仮想マシンの起動](#)

第5章 仮想マシンへの接続

RHEL 9 で仮想マシンと相互作用するには、以下のいずれかの方法で接続する必要があります。

- Web コンソールインターフェイスを使用する場合は、Web コンソールインターフェイスの仮想マシンペインを使用します。詳細は、[Web コンソールを使用した仮想マシンとの相互作用](#) を参照してください。
- Web コンソールを使用せずに、仮想マシンのグラフィカル表示と相互作用する必要がある場合は、Virt Viewer アプリケーションを使用します。詳細は、[Virt Viewer で仮想マシンのグラフィカルコンソールを開く方法](#) を参照してください。
- グラフィック表示ができない、または必要ない場合は、[SSH の端末接続](#) を使用します。
- ネットワークを使用してシステムから仮想マシンに到達できない場合は、[virsh コンソール](#) を使用します。

接続先の仮想マシンがローカルホストではなくリモートホストにある場合は、[リモートホストにより便利にアクセスできるように](#)、システムを設定することもできます。

前提条件

- 相互作用する仮想マシンが [インストールされ、起動している](#)。

5.1. WEB コンソールを使用した仮想マシンとの相互作用

RHEL 9 Web コンソールで仮想マシンと相互作用するには、仮想マシンのコンソールに接続する必要があります。グラフィカルコンソールおよびシリアルコンソールの両方が含まれます。

- Web コンソールで仮想マシンのグラフィカルインターフェイスを操作するには、[グラフィカルコンソール](#) を使用します。
- リモートビューアーで仮想マシンのグラフィカルインターフェイスを操作する場合は、[リモートビューアーでグラフィカルコンソールの表示](#) を参照してください。
- Web コンソールで仮想マシンの CLI を操作するには、[シリアルコンソール](#) を使用します。

5.1.1. Web コンソールで仮想マシンのグラフィカルコンソールの表示

仮想マシンのコンソールインターフェイスを使用すると、RHEL 9 Web コンソールに、選択した仮想マシンのグラフィカル出力を表示できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- ホストおよび仮想マシンの両方が、グラフィカルインターフェイスに対応している。

手順

1. **仮想マシン** インターフェイスで、グラフィカルコンソールを表示する仮想マシンをクリックします。
仮想マシンの**概要**と**コンソール**セクションがある新しいページが開きます。
2. コンソールドロップダウンメニューで **VNC コンソール** を選択します。

Web インターフェイスのメニューの下に VNC コンソールが表示されます。

グラフィカルコンソールが Web インターフェイスに表示されます。

3. **Expand** をクリックします。

実際のマシンの場合と同じように、マウスとキーボードを使用して仮想マシンのコンソールと相互作用できるようになりました。仮想マシンコンソールには、仮想マシンで実行しているアクティビティが表示されます。



注記

Web コンソールを実行しているホストで、特定の鍵の組み合わせ (**Ctrl+Alt+Del** など) を傍受して、仮想マシンに送信しないようにできます。

このようなキーの組み合わせを送信する場合は、**キーの送信** メニューをクリックして、送信するキーシーケンスを選択します。

たとえば、仮想マシンに **Ctrl+Alt+Del** の組み合わせを送信するには、**キーの送信** メニューをクリックして、**Ctrl+Alt+F1** メニューエントリを選択します。

トラブルシューティング

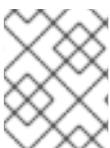
- グラフィカルコンソールをクリックしても効果がない場合は、コンソールを全画面表示にします。これは、マウスカーソルオフセットの既知の問題です。

関連情報

- [Web コンソールを使用して、リモートビューアーでグラフィカルコンソールを表示する方法](#)
- [Web コンソールで仮想マシンのシリアルコンソールの表示](#)

5.1.2. Web コンソールを使用して、リモートビューアーでグラフィカルコンソールを表示する方法

Web コンソールインターフェイスを使用して、選択した仮想マシンのグラフィカルコンソールを Virt Viewer などのリモートビューアーに表示することができます。



注記

Web コンソールから Virt Viewer を起動できます。他の VNC リモートビューアーは手動で起動できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- ホストおよび仮想マシンの両方が、グラフィカルインターフェイスに対応している。
- Virt Viewer でグラフィカルコンソールを表示する前に、Web コンソールが接続しているマシンに Virt Viewer をインストールする必要があります。
 1. **Launch remote viewer** をクリックします。
virt ビューアー (.vv) ファイルをダウンロードします。
 2. ファイルを開き、Virt Viewer を起動します。

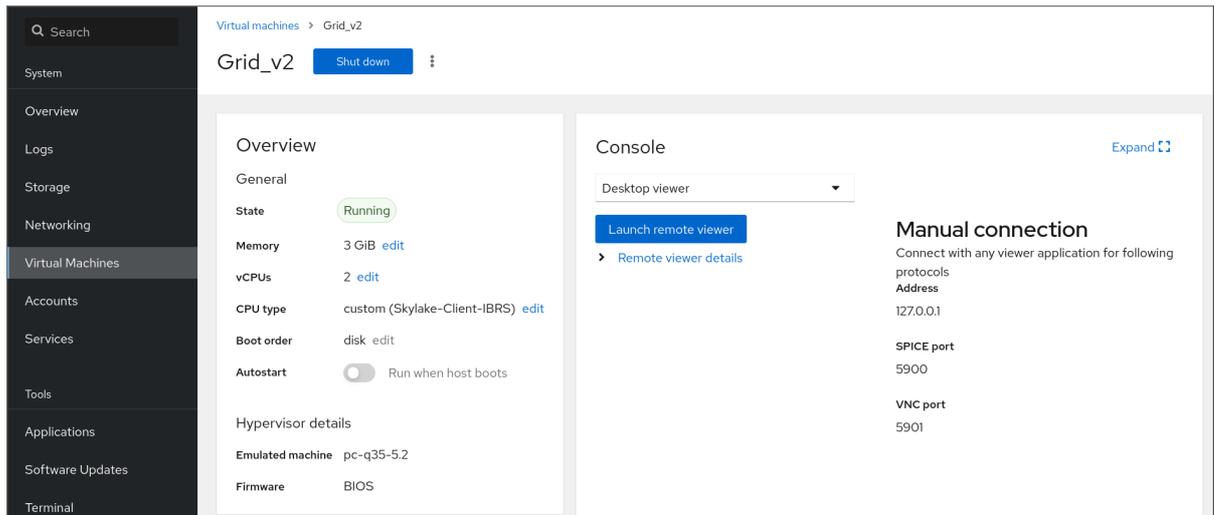


注記

リモートビューアーは、ほとんどのオペレーティングシステムで使用できます。ブラウザーの拡張機能やプラグインによっては、Web コンソールで Virt Viewer を開けないことがあります。

手順

1. **仮想マシン** インターフェイスで、グラフィカルコンソールを表示する仮想マシンをクリックします。
仮想マシンの**概要**と**コンソール**セクションがある新しいページが開きます。
2. コンソールドロップダウンメニューで **デスクトップビューアー** を選択します。



3. **Launch Remote Viewer** をクリックします。
Virt Viewer でグラフィカルコンソールが開きます。

実際のマシンの場合と同じように、マウスとキーボードを使用して仮想マシンのコンソールと相互作用できます。仮想マシンコンソールには、仮想マシンで実行しているアクティビティーが表示されます。



注記

Web コンソールを実行しているサーバーで、特定の鍵の組み合わせ (**Ctrl+Alt+Del** など) を傍受して、仮想マシンに送信しないようにできます。

このようなキーの組み合わせを送信する場合は、**キーの送信** メニューをクリックして、送信するキーシーケンスを選択します。

たとえば、仮想マシンに **Ctrl+Alt+F1** の組み合わせを送信するには、**キーの送信** メニューをクリックして、**Ctrl+Alt+F1** メニューエントリを選択します。

トラブルシューティング

- グラフィカルコンソールをクリックしても効果がない場合は、コンソールを全画面表示にします。これは、マウスカーソルオフセットの既知の問題です。
- Web コンソールでリモートビューアーを起動することができない場合、または最適ではない場合は、以下のプロトコルを使用して、任意のビューアーアプリケーションに手動で接続できます。

- アドレス - デフォルトのアドレスは **127.0.0.1** です。`/etc/libvirt/qemu.conf` の `vnc_listen` パラメーターを変更して、ホストの IP アドレスに変更できます。
- VNC ポート - 5901

関連情報

- [Web コンソールで仮想マシンのグラフィカルコンソールの表示](#)
- [Web コンソールで仮想マシンのシリアルコンソールの表示](#)

5.1.3. Web コンソールで仮想マシンのシリアルコンソールの表示

RHEL 9 Web コンソールで、選択した仮想マシンのシリアルコンソールを表示できます。これは、グラフィカルインターフェイスでホストマシンまたは仮想マシンを設定していない場合に便利です。

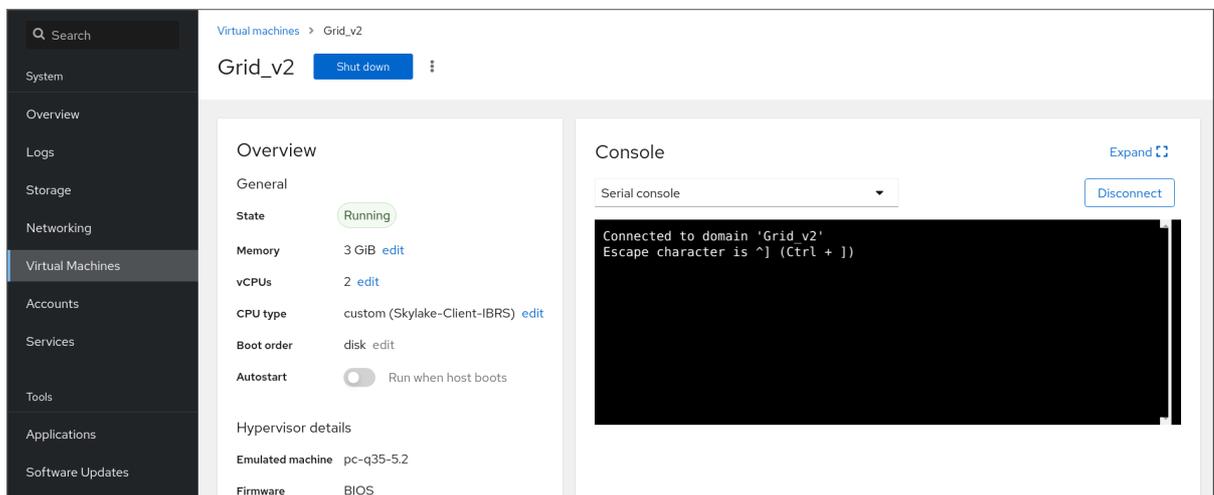
シリアルコンソールの詳細は、[Opening a virtual machine serial console](#) を参照してください。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** ペインで、シリアルコンソールを表示する仮想マシンをクリックします。仮想マシンの **概要** と **コンソール** セクションがある新しいページが開きます。
2. コンソールドロップダウンメニューで **シリアルコンソール** を選択します。グラフィカルコンソールが Web インターフェイスに表示されます。



仮想マシンからシリアルコンソールへの接続を切断して、再接続できます。

- 仮想マシンからシリアルコンソールへの接続を切断するには、**Disconnect** をクリックします。
- シリアルコンソールを仮想マシンに再接続するには、**Reconnect** をクリックします。

関連情報

- [Web コンソールで仮想マシンのグラフィカルコンソールの表示](#)

- [Web コンソールを使用して、リモートビューアーでグラフィカルコンソールを表示する方法](#)

5.1.4. Web コンソールで SPICE リモートディスプレイプロトコルを VNC に置き換える

RHEL 9 ホストでは、SPICE リモートディスプレイプロトコルのサポートが削除されました。SPICE プロトコルを使用するように設定された仮想マシン (VM) がある場合は、Web コンソールを使用して SPICE プロトコルを VNC プロトコルに置き換えることができます。そうしないと、仮想マシンが起動に失敗します。ただし、オーディオや USB パススルーなどの一部の SPICE デバイスは、VNC プロトコルに適切な代替機能が存在しないため、仮想マシンから削除されます。



重要

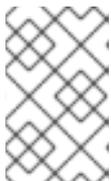
デフォルトでは、RHEL 8 仮想マシンは SPICE プロトコルを使用するように設定されています。RHEL 9 ホストでは、SPICE から VNC に切り替えない限り、RHEL 8 仮想マシンは起動に失敗します。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- SPICE リモートディスプレイプロトコルを使用するように設定され、すでにシャットダウンされている既存の仮想マシンがある。

手順

1. Web コンソールの仮想マシンインターフェイスで、SPICE プロトコルを使用するように設定されている仮想マシンのメニューボタン  をクリックします。
さまざまな仮想マシン操作を制御するためのドロップダウンメニューが開きます。
2. **Replace SPICE devices** をクリックします。
Replace SPICE devices ダイアログが開きます。



注記

SPICE プロトコルを使用する既存の仮想マシンが複数ある場合は、このダイアログにそれらがリストされます。このダイアログで、1つのステップで SPICE から VNC の使用に切り替える仮想マシンを複数選択できます。

3. **Replace** をクリックします。
操作が成功したことを確認するメッセージが表示されます。

5.2. VIRT VIEWER で仮想マシンのグラフィカルコンソールを開く方法

KVM 仮想マシンのグラフィカルコンソールに接続して、**Virt Viewer** デスクトップアプリケーションで開く場合は、以下の手順を行います。

前提条件

- システム、および接続している仮想マシンが、グラフィカルディスプレイに対応している。
- ターゲットの仮想マシンがリモートホストにある場合は、そのホストへの接続およびルートアクセス権限が確保されている。

- (必要に応じて) ターゲットの仮想マシンがリモートホストにある場合は、[リモートホストにアクセスしやすくなる](#) ように libvirt と SSH を設定している。

手順

- ローカルの仮想マシンに接続するには、次のコマンドを使用して、**guest-name** を、接続する仮想マシンの名前に置き換えます。

```
# virt-viewer guest-name
```

- リモートの仮想マシンに接続するには、SSH プロトコルで **virt-viewer** コマンドを実行します。たとえば、次のコマンドは、root 権限で、リモートシステム 192.0.2.1 にある **guest-name** という名前の仮想マシンに接続します。接続には、192.0.2.1 用の root 認証も必要になります。

```
# virt-viewer --direct --connect qemu+ssh://root@192.0.2.1/system guest-name  
root@192.0.2.1's password:
```

検証

接続が正しく機能している場合は、**Virt Viewer** 画面に仮想マシンのディスプレイが表示されます。

実際のマシンの場合と同じように、マウスとキーボードを使用して仮想マシンのコンソールと相互作用できます。仮想マシンコンソールには、仮想マシンで実行しているアクティビティーが表示されます。

トラブルシューティング

- グラフィカルコンソールをクリックしても効果がない場合は、コンソールを全画面表示にします。これは、マウスカーソルオフセットの既知の問題です。

関連情報

- **virt-viewer** の man ページ
- [リモートの仮想化ホストへの簡単なアクセスの設定](#)
- [Web コンソールを使用した仮想マシンとの相互作用](#)

5.3. SSH を使用した仮想マシンへの接続

SSH 接続プロトコルを使用して仮想マシンの端末と相互作用するには、以下の手順に従います。

前提条件

- ターゲットの仮想マシンへのネットワーク接続および root アクセス権がある。
- ターゲットの仮想マシンがリモートホストにある場合は、そのホストへの接続およびルートのアクセス権限もある。
- 仮想マシンネットワークは、libvirt が生成した **dnsmasq** により IP アドレスを割り当てます。これは、たとえば、[libvirt NAT ネットワーク](#) などに該当します。特に、仮想マシンが次のネットワーク設定のいずれかを使用している場合、SSH を使用して仮想マシンに接続することはできません。
 - **hostdev** インターフェイス

- ダイレクトインターフェイス
- ブリッジインターフェイス
- **libvirt-nss** コンポーネントを仮想マシンのホストにインストールして有効にしている。そうでない場合は、以下を行います。
 - a. **libvirt-nss** パッケージをインストールします。

```
# dnf install libvirt-nss
```

- b. `/etc/nsswitch.conf` ファイルを編集し、**libvirt_guest** を **hosts** 行に追加します。

```
...
passwd:  compat
shadow:  compat
group:   compat
hosts:   files libvirt_guest dns
...
```

手順

1. リモート仮想マシンに接続する場合は、最初に SSH でその物理ホストに接続します。以下の例は、root 認証情報を使用してホストマシン **192.0.2.1** に接続する方法を示しています。

```
# ssh root@192.0.2.1
root@192.0.2.1's password:
Last login: Mon Sep 24 12:05:36 2021
root~#
```

2. 仮想マシンの名前とユーザーアクセスの認証情報を使用して、仮想マシンに接続します。たとえば、以下は、root 認証情報を使用して、仮想マシン **testguest1** に接続します。

```
# ssh root@testguest1
root@testguest1's password:
Last login: Wed Sep 12 12:05:36 2018
root~]#
```

トラブルシューティング

- 仮想マシンの名前が分からない場合は、**virsh list --all** コマンドを使用すると、ホストで利用可能な仮想マシンのリストを表示できます。

```
# virsh list --all
Id   Name                State
-----
 2   testguest1         running
-   testguest2         shut off
```

関連情報

- [アップストリームの libvirt ドキュメント](#)

5.4. 仮想マシンのシリアルコンソールを開く

virsh console コマンドを使用すると、仮想マシンのシリアルコンソールに接続できます。

これは、仮想マシンが次のような場合に役に立ちます。

- VNC プロトコルは提供されないため、GUI ツールのビデオ表示には対応していません。
- ネットワークに接続されていないため、[SSH を使用して](#) 相互作用できない

前提条件

- ホスト上の GRUB ブートローダーは、シリアルコンソールを使用するように設定する必要があります。確認するには、ホスト上の `/etc/default/grub` ファイルに **GRUB_TERMINAL=serial** パラメーターが含まれていることを確認します。

```
$ sudo grep GRUB_TERMINAL /etc/default/grub
GRUB_TERMINAL=serial
```

- 仮想マシンには、**console type='pty'** などのシリアルコンソールデバイスが設定されている必要がある。確認するには、以下の手順を実施します。

```
# virsh dumpxml vm-name | grep console

<console type='pty' tty='/dev/pts/2'>
</console>
```

- 仮想マシンに、カーネルコマンドラインでシリアルコンソールが設定されている。これを確認するには、仮想マシン上の **cat /proc/cmdline** コマンド出力に **console=<console-name>** が含まれている必要があります。<console-name> はアーキテクチャー固有です。
 - AMD64 および Intel 64 の場合: **ttyS0**
 - ARM 64 の場合: **ttyAMA0**



注記

この手順の次のコマンドは **ttyS0** を使用します。

```
# cat /proc/cmdline
BOOT_IMAGE=vmlinuz-3.10.0-948.el7.x86_64 root=/dev/mapper/rhel-root ro
console=tty0 console=ttyS0,9600n8 rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb
```

シリアルコンソールが仮想マシンに正しく設定されていない場合は、**virsh コンソール** を仮想マシンに接続すると、応答のないゲストコンソールに接続できます。ただし、**Ctrl+]** ショートカットを使用して、応答しないコンソールを終了することができます。

- 仮想マシンでシリアルコンソールを設定するには、以下を行います。
 - i. 仮想マシンで、**console=ttyS0** カーネルオプションを有効にします。

```
# grubby --update-kernel=ALL --args="console=ttyS0"
```

- ii. 変更を反映させない可能性があるカーネルオプションをクリアします。

```
# grub2-editenv - unset kernelopts
```

iii. 仮想マシンを再起動します。

- **serial-getty@<console-name>** サービスを有効にする必要があります。たとえば、AMD64 および Intel 64 の場合:

```
# systemctl status serial-getty@ttyS0.service
```

- serial-getty@ttyS0.service - Serial Getty on ttyS0
Loaded: loaded (/usr/lib/systemd/system/serial-getty@.service; enabled; preset: enabled)

手順

1. ホストシステムで、**virsh console** コマンドを使用します。次の例では、libvirt ドライバーが安全なコンソール処理に対応していると、仮想マシン **guest1** に接続します。

```
# virsh console guest1 --safe
Connected to domain 'guest1'
Escape character is ^]
```

Subscription-name

```
Kernel 3.10.0-948.el7.x86_64 on an x86_64
```

```
localhost login:
```

2. virsh コンソールは、標準のコマンドラインインターフェイスと同じ方法で相互作用できます。

関連情報

- **virsh** の man ページ

5.5. リモートの仮想化ホストへの簡単なアクセスの設定

libvirt ユーティリティを使用してリモートホストシステムの仮想マシンを管理する場合は、**-c qemu+ssh://root@hostname/system** 構文を使用することが推奨されます。たとえば、ホスト **192.0.2.1** で、root で **virsh list** コマンドを実行します。

```
# virsh -c qemu+ssh://root@192.0.2.1/system list
root@192.0.2.1's password:
```

```
Id Name      State
-----
1  remote-guest  running
```

ただし、SSH および libvirt の設定を変更すれば、接続の詳細を完全に指定する必要がなくなります。以下に例を示します。

```
# virsh -c remote-host list
root@192.0.2.1's password:
```

```

Id Name          State
-----
1  remote-guest  running

```

この改善機能を有効にするには、以下の手順を行います。

手順

1. `~/.ssh/config` ファイルを以下のように編集します。ここで、`host-alias` は特定のリモートホストに関連付けられた短縮名および `root@192.0.2.1` のエイリアス、`hosturl` は `host` の URL アドレスです。

```

# vi ~/.ssh/config
Host example-host-alias
  User          root
  Hostname      192.0.2.1

```

2. `/etc/libvirt/libvirt.conf` ファイルを以下のように編集します。`example-qemu-host-alias` は、QEMU および libvirt ユーティリティーが `qemu+ssh://192.0.2.1/system` に目的のホスト `example-host-alias` を関連付けるホストエイリアスです。

```

# vi /etc/libvirt/libvirt.conf
uri_aliases = [
  "example-qemu-host-alias=qemu+ssh://example-host-alias/system",
]

```

検証

1. ローカルシステムで libvirt ベースのユーティリティーを使用し、`-c qemu-host-alias` パラメータを追加することで、リモートの仮想マシンを管理できることを確認します。これにより、リモートホストの SSH でコマンドが自動的に実行されます。
たとえば、以下のコマンドにより、前の手順で `example-qemu-host-alias` としてセットアップした接続である、`192.0.2.1` リモートホスト上の仮想マシンがリスト表示されることを確認します。

```

# virsh -c example-qemu-host-alias list

root@192.0.2.1's password:

Id Name          State
-----
1  example-remote-guest  running

```



注記

`virsh` の他に、`-c` (または `--connect`) オプションと、上記のリモートホストアクセス設定は、以下のユーティリティーで使用できます。

- [virt-install](#)
- [virt-viewer](#)

次のステップ

libvirt ユーティリティーを、1台のリモートホストで排他的に使用する場合は、libvirt ベースのユーティリティーのデフォルトターゲットとして特定の接続を設定することもできます。ただし、ローカルホストまたは別のリモートホストでも仮想マシンを管理する場合、この方法は推奨されません。

- `/etc/libvirt/libvirt.conf` ファイルを編集して、`uri_default` パラメーターの値を、デフォルトの libvirt ターゲットとして `example-qemu-host-alias` に設定できます。

```
# These can be used in cases when no URI is supplied by the application
# (@uri_default also prevents probing of the hypervisor driver).
#
uri_default = "example-qemu-host-alias"
```

これにより、指定したリモートホストで、libvirt ベースのコマンドがすべて自動的に実行されます。

```
$ virsh list
root@192.0.2.1's password:

Id Name          State
-----
1  example-remote-guest  running
```

- リモートホストに接続する場合、リモートシステムへの root パスワードの入力を回避できます。そのためには、以下の方法を1つ以上行います。
 - [リモートホストへのキーベースの SSH アクセスを設定する](#)
 - SSH 接続の多重化を使用して、リモートシステムに接続する。
 - [Identity Management における Kerberos 認証](#)
- `-c` (または `--connect`) オプションを使用して、リモートホストで `virt-install`、`virt-viewer`、および `virsh` コマンドを実行できます。

第6章 仮想マシンのシャットダウン

RHEL 9 でホストされた実行中の仮想マシンをシャットダウンする場合は、[コマンドラインインターフェイス](#) または [Web コンソールの GUI](#) を使用します。

6.1. コマンドラインインターフェイスを使用した仮想マシンのシャットダウン

応答している仮想マシンをシャットダウンするには、以下のいずれかを行います。

- [ゲストに接続している](#) 場合に、ゲスト OS に適したシャットダウンコマンドを使用
- ホストで **virsh shutdown** コマンドを使用
 - 仮想マシンがローカルホストにある場合は、以下のコマンドを実行します。

```
# virsh shutdown demo-guest1
Domain 'demo-guest1' is being shutdown
```

- 仮想マシンがリモートホスト (この例では 192.0.2.1) にある場合は、以下のコマンドを実行します。

```
# virsh -c qemu+ssh://root@192.0.2.1/system shutdown demo-guest1

root@192.0.2.1's password:
Domain 'demo-guest1' is being shutdown
```

応答しない場合など、仮想マシンを強制的にシャットダウンする場合は、そのホストで **virsh destroy** コマンドを実行します。

```
# virsh destroy demo-guest1
Domain 'demo-guest1' destroyed
```

注記

virsh destroy コマンドは、仮想マシンの設定またはディスクイメージを削除するわけではありません。物理マシンから電源コードを抜くのと同様に、仮想マシンの実行中の仮想マシンインスタンスを終了するだけになります。したがって、まれに、**virsh destroy** により、仮想マシンのファイルシステムが破損することがあるため、その他のシャットダウン方法がすべて失敗した場合に限り、このコマンドを使用することが推奨されません。

6.2. WEB コンソールを使用した仮想マシンのシャットダウンおよび再起動

RHEL 9 Web コンソールを使用して、実行中の仮想マシンを [シャットダウン](#) または [再起動](#) できます。仮想マシンが応答しない場合は、マスク不可割り込みを送信できます。

6.2.1. Web コンソールで仮想マシンのシャットダウン

仮想マシンが **稼働** 状態であれば、RHEL 9 Web コンソールを使用してシャットダウンできます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、シャットダウンする仮想マシンがある行を見つけます。
2. 行の右側で、**Shut Down** をクリックします。
仮想マシンがシャットダウンします。

トラブルシューティング

- 仮想マシンがシャットダウンしない場合には、**シャットダウン** ボタンの横にある **⋮** をクリックして、**シャットダウンの強制** を選択します。
- 応答しない仮想マシンをシャットダウンするには、[マスク不可割り込みを送信](#) することもできます。

関連情報

- [Web コンソールを使用した仮想マシンの起動](#)
- [Web コンソールを使用した仮想マシンの再起動](#)

6.2.2. Web コンソールを使用した仮想マシンの再起動

仮想マシンが **稼働** 状態であれば、RHEL 9 Web コンソールを使用して再起動できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **Virtual Machines** インターフェイスで、再起動する仮想マシンの行を見つけます。
2. 行の右側にあるメニューボタン **⋮** をクリックします。
アクションのドロップダウンメニューが表示されます。
3. ドロップダウンメニューで、**Reboot** をクリックします。
仮想マシンがシャットダウンして再起動します。

トラブルシューティング

- 仮想マシンが再起動しない場合には **Reboot** ボタンのとりにある **⋮** をクリックして **Force Reboot** を選択します。
- 応答しない仮想マシンをシャットダウンするには、[マスク不可割り込みを送信](#) することもできます。

関連情報

- [Web コンソールを使用した仮想マシンの起動](#)
- [Web コンソールで仮想マシンのシャットダウン](#)

6.2.3. Web コンソールでマスク不可割り込みを仮想マシンに送信する手順

NMI (マスク不可割り込み) を送信すると、応答しない稼働中の仮想マシンが応答またはシャットダウンする可能性があります。たとえば、**Ctrl+Alt+Del** の NMI を、標準入力に応答しない仮想マシンに送信できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、NMI を送信する仮想マシンの行を見つけます。
2. 行の右側にあるメニューボタン **⋮** をクリックします。
アクションのドロップダウンメニューが表示されます。
3. ドロップダウンメニューで、**Send non-maskable interrupt** をクリックします。
NMI が仮想マシンに送信されます。

関連情報

- [Web コンソールを使用した仮想マシンの起動](#)
- [Web コンソールを使用した仮想マシンの再起動](#)
- [Web コンソールで仮想マシンのシャットダウン](#)

第7章 仮想マシンの削除

RHEL 9 で仮想マシンを削除する場合は、[コマンドラインインターフェイス](#) または [Web コンソールの GUI](#) を使用します。

7.1. コマンドラインインターフェイスを使用した仮想マシンの削除

仮想マシンを削除するには、コマンドラインでその XML 設定および関連するストレージファイルをホストから削除します。以下の手順を実施します。

前提条件

- 仮想マシンからの重要なデータのバックアップを作成する。
- 仮想マシンをシャットダウンしている。
- その他の仮想マシンが、同じ関連ストレージを使用しないようにしている。

手順

- **virsh undefine** ユーティリティーを使用します。
たとえば、次のコマンドは、**guest1** 仮想マシン、関連のあるストレージボリューム、および不揮発性 RAM が存在する場合はそれを削除します。

```
# virsh undefine guest1 --remove-all-storage --nvram
Domain 'guest1' has been undefined
Volume 'vda'(/home/images/guest1.qcow2) removed.
```

関連情報

- **virsh undefine --help** コマンド
- **virsh** の man ページ

7.2. WEB コンソールを使用した仮想マシンの削除

RHEL 9 Web コンソールが接続しているホストから、仮想マシンおよび関連ストレージファイルを削除する場合は、以下の手順を行います。

前提条件

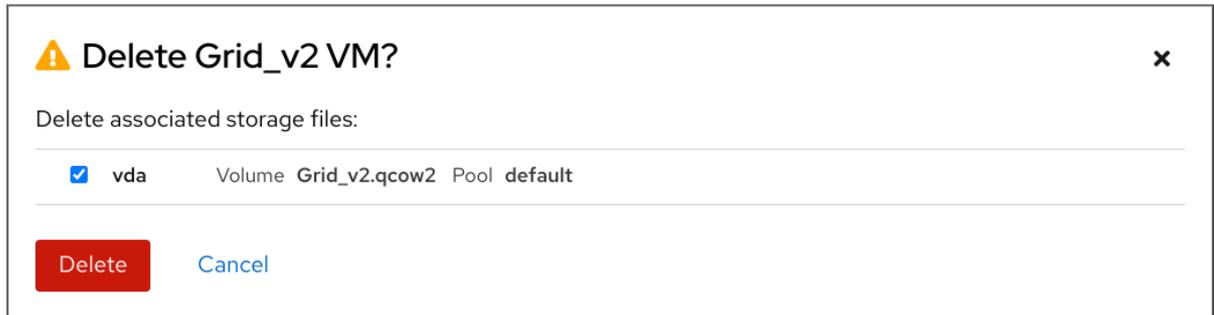
- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 仮想マシンからの重要なデータのバックアップを作成する。
- 他の仮想マシンが同じ関連ストレージを使用していないことを確認します。
- **オプション:** 仮想マシンをシャットダウンします。

手順

1. **仮想マシン** インターフェイスで、削除する仮想マシンのメニューボタン **⋮** をクリックします。

仮想マシン操作を制御するためのドロップダウンメニューが表示されます。

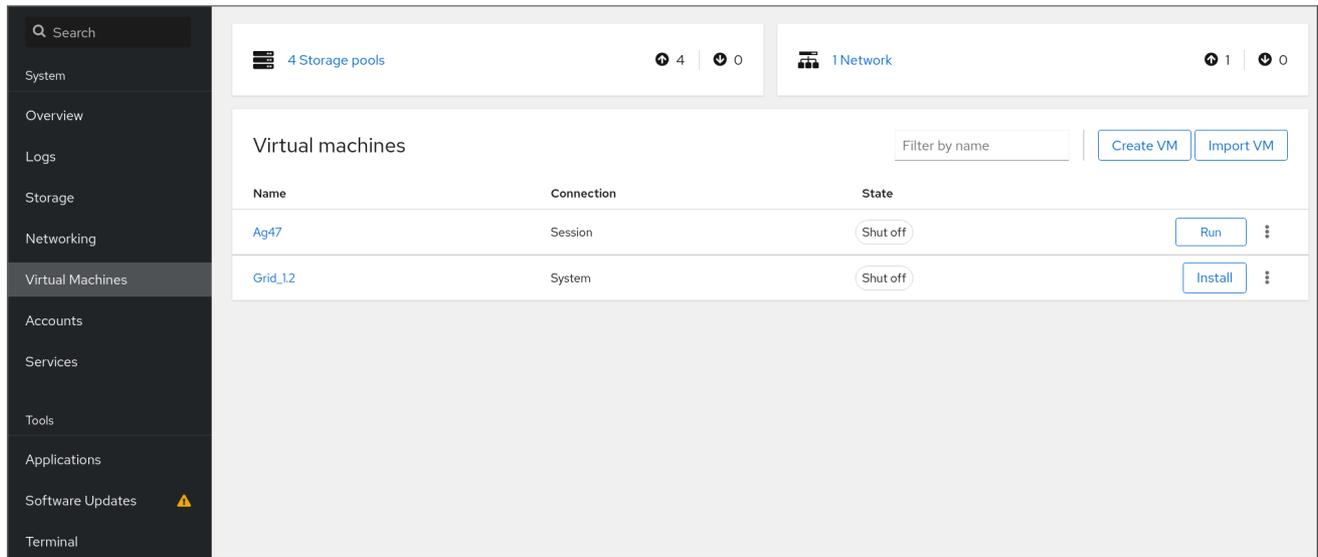
2. **Delete** をクリックします。
確認ダイアログが表示されます。



3. **(必要に応じて)** 仮想マシンに関連するすべてまたは一部のストレージファイルを削除する場合は、削除するストレージファイルの横にあるチェックボックスを選択します。
4. **Delete** をクリックします。
仮想マシンと、選択したストレージファイルが削除されます。

第8章 WEB コンソールでの仮想マシンの管理

RHEL 9 ホストのグラフィカルインターフェイスで仮想マシンを管理する場合は、RHEL 9 Web コンソールの **Virtual Machines** ペインを使用できます。



8.1. WEB コンソールを使用した仮想マシンの管理の概要

RHEL 9 Web コンソールは、Web ベースのシステム管理インターフェイスです。Web コンソールは、その機能の1つとして、ホストシステムで仮想マシンをグラフィカルに表示してその仮想マシンの作成、アクセス、および設定を可能にします。

Web コンソールを使用して RHEL 9 で仮想マシンを管理するには、最初に、仮想化用の [Web コンソールプラグイン](#) をインストールする必要があります。

次のステップ

- Web コンソールで仮想マシンの管理を有効にする手順は、[Web コンソールの設定による仮想マシンの管理](#) を参照してください。
- Web コンソールで利用できる仮想マシン管理アクションの包括的なリストは、[Virtual machine management features available in the web console](#) を参照してください。

8.2. 仮想マシンを管理するために WEB コンソールを設定

Web コンソールの仮想マシン (VM) プラグインをインストールして、RHEL 9 Web コンソールを使用してホストで仮想マシンを管理できるようにしてある。

前提条件

- Web コンソールがマシンにインストールされ、有効化されている。

```
# systemctl status cockpit.socket
cockpit.socket - Cockpit Web Service Socket
Loaded: loaded (/usr/lib/systemd/system/cockpit.socket
[...]
```

このコマンドが、**Unit cockpit.socket could not be found** を返す場合は、[Web コンソールのインストールおよび有効化](#) のドキュメントに従って Web コンソール を有効にします。

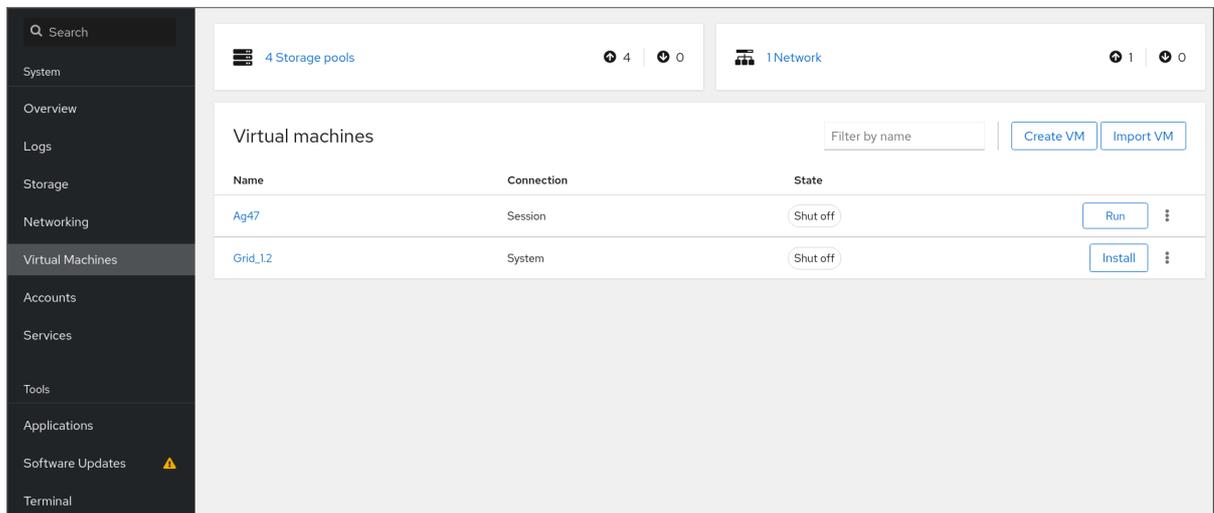
手順

- **cockpit-machines** プラグインをインストールします。

```
# dnf install cockpit-machines
```

検証

1. ブラウザーに **https://localhost:9090** のアドレスを入力するなどして、Web コンソールにアクセスします。
2. ログインします。
3. インストールに成功すると、**仮想マシン** が Web コンソールのサイドメニューに表示されます。



関連情報

- [RHEL 9 Web コンソールを使用したシステムの管理](#)

8.3. WEB コンソールを使用した仮想マシンの名前の変更

名前の競合を避けるために、またはユースケースに基づいて新しい一意の名前を割り当てるために、既存の仮想マシンの名前を変更することが必要な場合があります。RHEL Web コンソールを使用して仮想マシンの名前を変更できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 仮想マシンがシャットダウンされている。

手順

1. **Virtual Machines** インターフェイスで、名前を変更する仮想マシンのメニューボタン **⋮** をクリックします。
仮想マシン操作を制御するためのドロップダウンメニューが表示されます。

2. **Rename** をクリックします。
Rename a VM ダイアログが表示されます。



3. **New name** フィールドに、仮想マシンの名前を入力します。
4. **Rename** をクリックします。

検証

- 新しい仮想マシン名が **Virtual Machines** インターフェイスに表示されていることを確認します。

8.4. WEB コンソールで利用可能な仮想マシンの管理機能

RHEL 9 Web コンソールを使用すると、システム上の仮想マシンを管理する以下のアクションを実行できます。

表8.1 RHEL 9 Web コンソールで実行できる仮想マシン管理タスク

タスク	詳細は以下参照
仮想マシンの作成およびゲストオペレーティングシステムでのインストール	Web コンソールを使用した仮想マシンの作成、およびゲストのオペレーティングシステムのインストール
仮想マシンの削除	Web コンソールを使用した仮想マシンの削除
仮想マシンを起動、シャットダウンし、再起動	Web コンソールを使用した仮想マシンの起動と Web コンソールを使用した仮想マシンのシャットダウンおよび再起動
さまざまなコンソールを使用した仮想マシンへの接続および操作	Web コンソールを使用した仮想マシンとの相互作用
仮想マシンに関するさまざまな情報の表示	Web コンソールを使用した仮想マシン情報の表示
仮想マシンに割り当てられたホストメモリーの調整	Web コンソールを使用した仮想マシンのメモリーの追加および削除

タスク	詳細は以下参照
仮想マシンのネットワーク接続管理	Web コンソールで仮想マシンのネットワークインターフェイスの管理
ホストでの利用可能な仮想マシンストレージ管理および仮想ディスクを仮想マシンへの割り当て	Web コンソールを使用した仮想マシン用のストレージの管理
仮想マシンの仮想 CPU 設定	Web コンソールを使用した仮想 CPU の管理
仮想マシンのライブマイグレーション	Web コンソールを使用した仮想マシンのライブ移行
ホストデバイスの管理	Web コンソールを使用したホストデバイスの管理
仮想光学ドライブを管理する	仮想光学ドライブの管理
ウォッチドッグデバイスを接続する	Web コンソールを使用した仮想マシンへのウォッチドッグデバイスの接続

第9章 仮想マシンに関する情報の表示

RHEL 9 での仮想化デプロイメントのあらゆる側面を調整またはトラブルシューティングする必要がある場合、通常、最初の手順として実行しなければならないのは、仮想マシンの現在の状態および設定に関する情報を確認することです。これには、[the command-line interface](#) または [the web console](#) を使用できます。仮想マシンの [XML 設定](#) で情報を表示することもできます。

9.1. コマンドラインインターフェイスを使用した仮想マシン情報の表示

ホストおよびその設定で仮想マシンに関する情報を取得するには、以下のコマンドのいずれかまたは複数コマンドを使用します。

手順

- ホストで仮想マシンのリストを取得するには、次のコマンドを実行します。

```
# virsh list --all
Id Name          State
-----
 1 testguest1     running
- testguest2     shut off
- testguest3     shut off
- testguest4     shut off
```

- 特定の仮想マシンに関する基本的な情報を取得するには、次のコマンドを実行します。

```
# virsh dominfo testguest1
Id:          1
Name:        testguest1
UUID:        a973666f-2f6e-415a-8949-75a7a98569e1
OS Type:     hvm
State:       running
CPU(s):      2
CPU time:    188.3s
Max memory:  4194304 KiB
Used memory: 4194304 KiB
Persistent:  yes
Autostart:   disable
Managed save: no
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c486,c538 (enforcing)
```

- 特定の仮想マシンの XML 設定をすべて取得するには、次のコマンドを実行します。

```
# virsh dumpxml testguest2

<domain type='kvm' id='1'>
  <name>testguest2</name>
  <uuid>a973434f-2f6e-4ěša-8949-76a7a98569e1</uuid>
  <metadata>
  [...]
```

- 仮想マシンのディスクおよびその他のブロックデバイスに関する情報は、次のコマンドを実行します。

```
# virsh domblklist testguest3
Target Source
-----
vda    /var/lib/libvirt/images/testguest3.qcow2
sda    -
sdb    /home/username/Downloads/virt-p2v-1.36.10-1.el7.iso
```

- 仮想マシンのファイルシステムとそのマウントポイントに関する情報を取得するには、次のコマンドを実行します。

```
# virsh domfsinfo testguest3
Mountpoint Name Type Target
-----
/          dm-0 xfs
/boot     vda1 xfs
```

- 特定の仮想マシンの vCPU に関する詳細を取得するには、次のコマンドを実行します。

```
# virsh vcpuinfo testguest4
VCPU:      0
CPU:       3
State:     running
CPU time:  103.1s
CPU Affinity: yyyy

VCPU:      1
CPU:       0
State:     running
CPU time:  88.6s
CPU Affinity: yyyy
```

仮想マシンで vCPU を設定し、最適化するには、[仮想マシンの CPU パフォーマンスの最適化](#)を参照してください。

- ホスト上の仮想ネットワークインターフェイスのリストを表示するには、次のコマンドを実行します。

```
# virsh net-list --all
Name      State  Autostart Persistent
-----
default   active yes      yes
labnet    active yes      yes
```

特定のインターフェイスに関する情報は、次のコマンドを実行します。

```
# virsh net-info default
Name:      default
UUID:     c699f9f6-9202-4ca8-91d0-6b8cb9024116
Active:    yes
Persistent: yes
Autostart: yes
Bridge:    virbr0
```

- ネットワークインターフェイス、仮想マシンネットワーク、およびこれらの設定手順の詳細は、[仮想マシンのネットワーク接続の設定](#)を参照してください。

9.2. WEB コンソールを使用した仮想マシン情報の表示

RHEL 9 Web コンソールを使用して、Web コンソールセッションがアクセスできるすべての [仮想マシン](#) および [ストレージプール](#) に関する情報を表示することができます。

Web コンソールセッションの接続先である [選択した仮想マシンに関する情報](#) を表示できます。これには、[ディスク](#)、[仮想ネットワークインターフェイス](#)、および [リソースの使用量](#) に関する情報が含まれます。

9.2.1. Web コンソールで仮想化の概要を表示

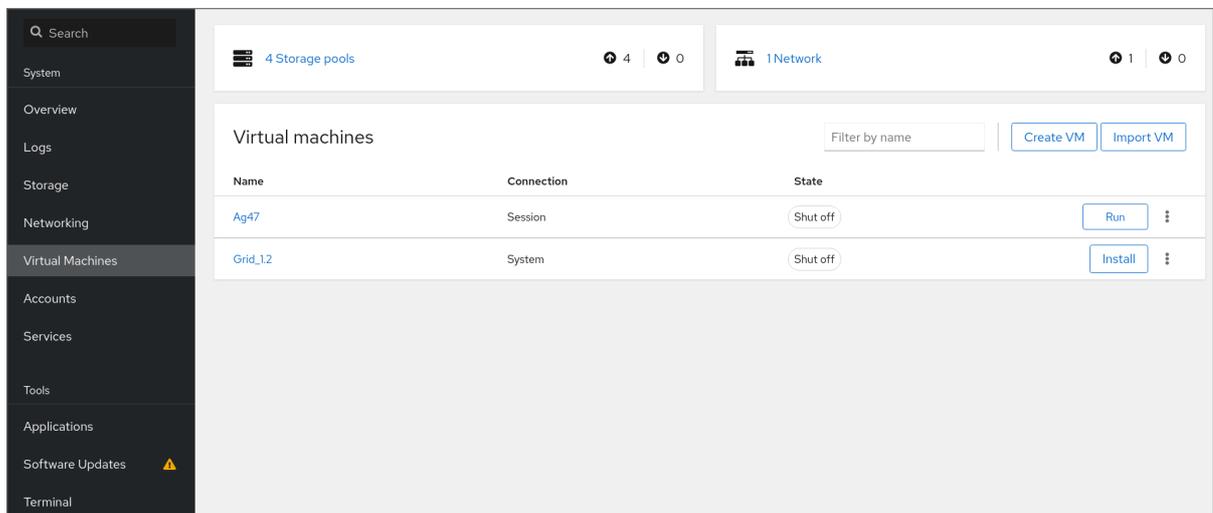
Web コンソールを使用して、仮想マシン、ストレージプール、およびネットワークに関する概要情報を含む仮想化の概要にアクセスできます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

- Web コンソールのサイドメニューで、[仮想マシン](#) をクリックします。利用可能なストレージプール、利用可能なネットワーク、および Web コンソールが接続する仮想マシンに関する情報を含むダイアログボックスが表示されます。



この情報には以下が含まれます。

- **ストレージプール** - Web コンソールからアクセス可能なストレージプールの数とその状態です (アクティブまたは非アクティブ)。
- **ネットワーク** - Web コンソールからアクセス可能なネットワークの数とその状態です (アクティブまたは非アクティブ)。
- **名前** - 仮想マシンの名前
- **接続** - libvirt 接続、システム、またはセッションの種類。

- **状態** - 仮想マシンの状態

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

9.2.2. Web コンソールを使用したストレージプール情報の表示

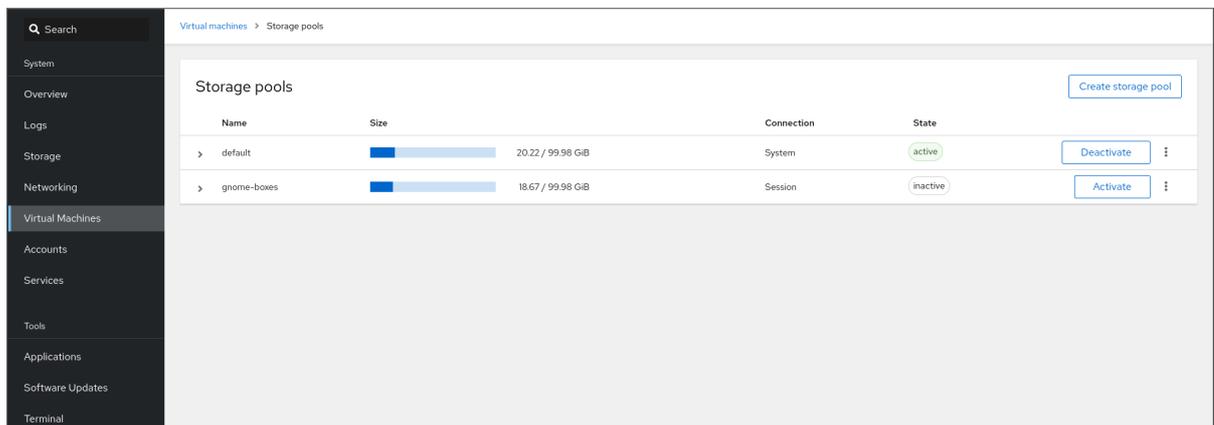
Web コンソールを使用して、システムで利用可能なストレージプールの詳細情報を表示できます。ストレージプールを使用すると、仮想マシンのディスクイメージを作成できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

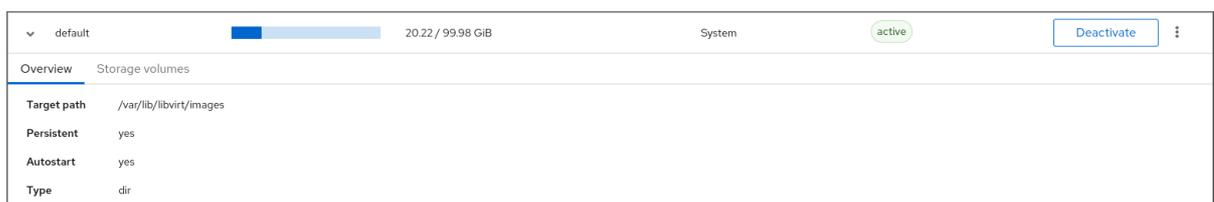
手順

1. **仮想マシン インターフェイス**で **ストレージプール** をクリックします。
Storage pools画面が表示され、設定されているストレージプールの一覧が示されます。



この情報には以下が含まれます。

- **名前** - ストレージプールの名前
 - **サイズ** - 現在の割り当てとストレージプールの合計容量。
 - **接続** - ストレージプールのアクセスに使用される接続
 - **状態** - ストレージプールのステータス
2. 情報を表示するストレージプールの横にある矢印をクリックします。
行がデプロイメントされ、選択したストレージプールに関する詳細情報を含む概要ペインが表示されます。



この情報には以下が含まれます。

- **ターゲットパス** - ストレージプール の場所です。
 - **永続的** - ストレージプールの設定が永続的であるかどうかを示します。
 - **自動起動** - システムの起動時にストレージプールが自動的に起動するかどうかを示します。
 - **種類** - ストレージプールの種類。
3. ストレージプールに関連付けられているストレージボリュームのリストを表示する場合は、**ストレージボリューム** をクリックします。
ストレージボリュームペインが表示され、設定したストレージボリュームのリストが表示されます。

Name	Used by	Size
<input type="checkbox"/> volume1		0 / 1 GB
<input type="checkbox"/> volume2		0 / 1 GB

この情報には以下が含まれます。

- **名前** - ストレージボリュームの名前。
- **使用者** - 現在ストレージボリュームを使用している仮想マシン。
- **サイズ** - ボリュームのサイズ。

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

9.2.3. Web コンソールで仮想マシン基本情報の表示

Web コンソールを使用して、選択した仮想マシンに関する基本情報 (割り当てられたリソース、ハイパーバイザーの詳細など) を表示できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. Web コンソールのサイドメニューで、**仮想マシン** をクリックします。
2. 情報を表示する仮想マシンをクリックします。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。

概要セクションには、次の一般的な仮想マシンの詳細が記載されています。

- **状態** - 仮想マシンの状態 (実行中またはシャットオフ)。

- **メモリー** - 仮想マシンに割り当てるメモリー容量
- **CPU** - 仮想マシンに設定されている仮想 CPU の数とタイプ。
- **ブート順序** - 仮想マシンに設定されたブート順序
- **自動起動** - 仮想マシンで自動起動が有効になっているかどうか

この情報には、以下のハイパーバイザーの詳細も含まれます。

- **エミュレートされたマシン** - 仮想マシンによりエミュレートされたマシンタイプ
- **ファームウェア** - 仮想マシンのファームウェア。

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)
- [Web コンソールを使用した仮想 CPU の管理](#)

9.2.4. Web コンソールで仮想マシンのリソース使用状況の表示

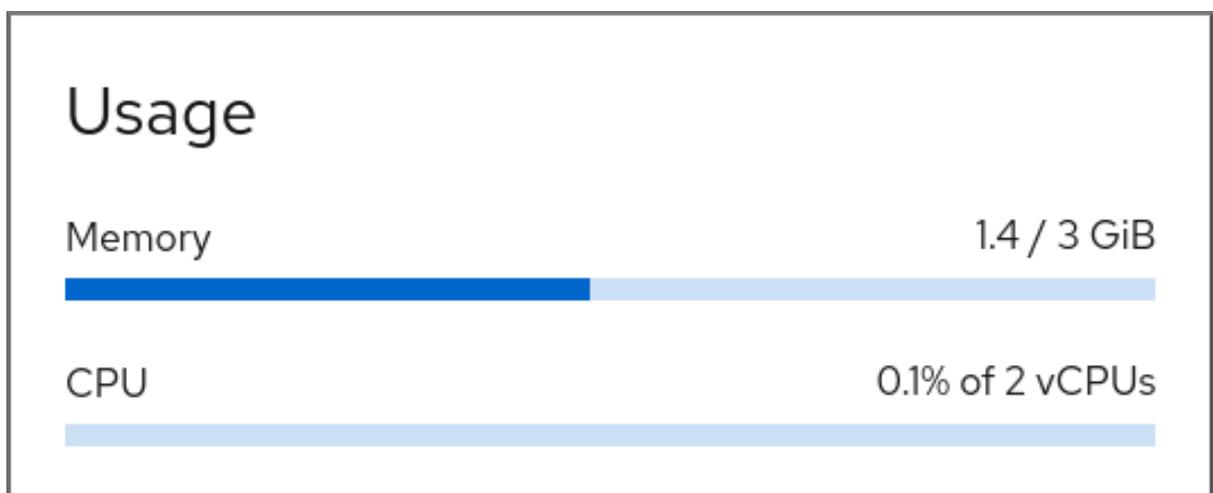
Web コンソールを使用して、選択した仮想マシンのメモリーと仮想 CPU 使用率を表示できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **使用方法** までスクロールします。
使用率セクションには、仮想マシンのメモリーおよび仮想 CPU 使用率に関する情報が表示されます。



関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

9.2.5. Web コンソールで仮想マシンのディスク情報の表示

Web コンソールを使用して、選択した仮想マシンに割り当てられたディスクの詳細情報を表示できません。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. 情報を表示する仮想マシンをクリックします。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ディスク** までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

この情報には以下が含まれます。

- **デバイス** - ディスクのデバイスの種類。
- **使用済み** - 現在割り当てられているディスク容量。
- **容量** - ストレージボリュームの最大サイズ。
- **バス** - エミュレートされるディスクデバイスの種類。
- **アクセス** - ディスクが **書き込み可能** かどうか、**読み取り専用** であるか。**raw** ディスクの場合は、**書き込み可能および共有** へのアクセスを設定することもできます。
- **ソース** - ディスクデバイスまたはファイル

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

9.2.6. Web コンソールで仮想ネットワークインターフェイス情報の表示および編集

RHEL 9 Web コンソールを使用して、選択した仮想マシンで仮想ネットワークインターフェイスを表示および変更することができます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ネットワークインターフェイス** までスクロールします。
ネットワークインターフェイスセクションには、仮想マシンに設定された仮想ネットワークインターフェイスに関する情報と、ネットワークインターフェイスの**追加**、**削除**、**編集**、または**アンプラグ**のオプションが表示されます。

Network interfaces						Add network interface
Type	Model type	MAC address	IP address	Source	State	
network	virtio	52:54:00	inet 192.168.122.9/24	default	up	Delete Unplug Edit

この情報には以下が含まれます。

- **種類** - 仮想マシンのネットワークインターフェイスの種類。タイプには、仮想ネットワーク、LAN へのブリッジ、および直接割り当てが含まれます。



注記

RHEL 9 以降では、汎用イーサネット接続はサポートされていません。

- **モデルタイプ** - 仮想ネットワークインターフェイスのモデル。
 - **MAC アドレス** - 仮想ネットワークインターフェイスの MAC アドレス。
 - **IP アドレス** - 仮想ネットワークインターフェイスの IP アドレス。
 - **ソース** - ネットワークインターフェイスのソース。これはネットワークの種類によって異なります。
 - **状態** - 仮想ネットワークインターフェイスの状態。
3. 仮想ネットワークインターフェイスの設定を編集するには、**編集** をクリックします。仮想ネットワークインターフェイスの設定ダイアログが開きます。

52:54:00:b4:2a:62 virtual network interface settings ✕

Interface type ? Virtual network

Source default

Model (Linux, perf)

MAC address 52:64:00:b4:2a:63

Save
Cancel

4. インターフェイスの種類、ソース、モデル、または MAC アドレスを変更します。
5. **Save** をクリックします。ネットワークインターフェイスが変更しました。



注記

仮想ネットワークインターフェイス設定の変更は、仮想マシンを再起動しないと有効になりません。

また、MAC アドレスは、仮想マシンがシャットダウンしている場合にのみ変更できます。

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

9.3. 仮想マシンの XML 設定例

仮想マシンの XML 設定 (ドメイン XML と呼ばれる) は、仮想マシンの設定およびコンポーネントを決定します。以下の表は、仮想マシンの XML 設定例の各セクションと、コンテンツについて説明しています。

仮想マシンの XML 設定を取得するには、**virsh dumpxml** コマンドの後に仮想マシンの名前を指定します。

```
# virsh dumpxml testguest1
```

表9.1 XML 設定のサンプル

ドメイン XML セクション	説明
<pre><domain type='kvm'> <name>Testguest1</name> <uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid> <memory unit='KiB'>1048576</memory> <currentMemory unit='KiB'>1048576</currentMemory></pre>	<p>これは、1024 MiB のメモリーが割り当てられている KVM 仮想マシン Testguest1 です。</p>

ドメイン XML セクション	説明
<pre><vcpu placement='static'>1</vcpu></pre>	<p>仮想マシンには、仮想 CPU (vCPU) が1つ割り当てられます。</p> <p>vCPU の設定に関する詳細は、仮想マシンの CPU パフォーマンスの最適化 を参照してください。</p>
<pre><os> <type arch='x86_64' machine='pc-q35- rhel9.0.0'>hvm</type> <boot dev='hd'> </os></pre>	<p>マシンアーキテクチャーは AMD64 および Intel 64 のアーキテクチャーに設定され、Intel Q35 マシン種別を使用して機能の互換性を決定します。OS は、ハードディスクドライブから起動するように設定されています。</p> <p>OS がインストールされた仮想マシンの作成については、Web コンソールを使用した仮想マシンの作成、およびゲストのオペレーティングシステムのインストール を参照してください。</p>
<pre><features> <acpi/> <apic/> </features></pre>	<p>acpi および apic ハイパーバイザー機能が無効になっています。</p>
<pre><cpu mode='host-model' check='partial'></pre>	<p>機能 XML (virsh domcapabilities で取得可能) からのホストの CPU 定義は、仮想マシンの XML 設定に自動的にコピーされます。したがって、仮想マシンの起動時に、libvirt はホストの CPU と似た CPU モデルを選択し、さらに機能を追加してホストモデルに可能な限り近づけます。</p>
<pre><clock offset='utc'> <timer name='rtc' tickpolicy='catchup'> <timer name='pit' tickpolicy='delay'> <timer name='hpet' present='no'> </clock></pre>	<p>仮想マシンの仮想ハードウェアクロックは UTC タイムゾーンを使用します。また、QEMU ハイパーバイザーと同期するために、異なるタイマーが3つ設定されます。</p>

ドメイン XML セクション	説明
<pre data-bbox="212 264 692 365"><on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash></pre>	<p data-bbox="1015 224 1430 465">仮想マシンの電源が切れた場合や、仮想マシンの OS が突然終了すると、libvirt が仮想マシンを終了し、割り当てられているリソースをすべて解放します。仮想マシンの再起動時に、libvirt は同じ設定で仮想マシンを起動します。</p>
<pre data-bbox="212 566 643 696"><pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm></pre>	<p data-bbox="1015 524 1430 618">この仮想マシンでは、S3 および S4 ACPI のスリープ状態が無効になっています。</p>
<pre data-bbox="212 840 895 1256"><devices> <emulator>/usr/libexec/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='vda' bus='virtio'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='sdb' bus='sata'/> <readonly/> </disk></pre>	<p data-bbox="1015 799 1430 969">仮想マシンは、エミュレーションに /usr/libexec/qemu-kvm バイナリーファイルを使用し、これには2つのディスクデバイスが割り当てられています。</p> <p data-bbox="1015 1008 1430 1317">最初のディスクは、ホストに保存されている /var/lib/libvirt/images/Testguest.qcow2 をベースにした仮想ハードドライブで、その論理デバイス名は vda に設定されています。Windows ゲストでは、virtio の代わりに sata バスを使用することが推奨されます。</p> <p data-bbox="1015 1355 1430 1449">2番目のディスクは仮想化 CD-ROM で、その論理デバイス名は sdb に設定されています。</p>

ドメイン XML セクション	説明
<pre> <controller type='usb' index='0' model='qemu-xhci' ports='15'/> <controller type='sata' index='0'/> <controller type='pci' index='0' model='pcie-root'/> <controller type='pci' index='1' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='1' port='0x10'/> </controller> <controller type='pci' index='2' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='2' port='0x11'/> </controller> <controller type='pci' index='3' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='3' port='0x12'/> </controller> <controller type='pci' index='4' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='4' port='0x13'/> </controller> <controller type='pci' index='5' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='5' port='0x14'/> </controller> <controller type='pci' index='6' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='6' port='0x15'/> </controller> <controller type='pci' index='7' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='7' port='0x16'/> </controller> <controller type='virtio-serial' index='0'/> </pre>	<p>仮想マシンは、USB デバイスの割り当てにコントローラーを1つ、PCI-Express (PCIe) デバイスにルートコントローラーを使用します。さらに、virtio-serial コントローラーが利用できるため、仮想マシンは、シリアルコンソールなど、各種方法でホストを操作できます。</p> <p>仮想デバイスの詳細は、仮想デバイスの種類 を参照してください。</p>
<pre> <interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='virtio'/> </interface> </pre>	<p>ネットワークインターフェイスは、default の仮想ネットワークおよび virtio ネットワークデバイスモデルを使用する仮想マシンに設定されます。Windows ゲストでは、virtio の代わりに e1000e モデルを使用することが推奨されます。</p> <p>ネットワークインターフェイスの設定に関する詳細は、仮想マシンのネットワークパフォーマンスの最適化 を参照してください。</p>

ドメイン XML セクション	説明
<pre> <serial type='pty'> <target type='isa-serial' port='0'> <model name='isa-serial'> </target> </serial> <console type='pty'> <target type='serial' port='0'> </console> <channel type='unix'> <target type='virtio' name='org.qemu.guest_agent.0'> <address type='virtio-serial' controller='0' bus='0' port='1'> </channel> </target> </channel> </console> </serial> </pre>	<p>pty シリアルコンソールが仮想マシンに設定されているので、ホストとの基本的な仮想マシン通信が可能になります。コンソールは、ポート1で UNIX チャンネルを使用します。この設定は自動で設定されており、設定の変更は推奨されません。</p> <p>仮想マシンと相互作用する方法の詳細は、Web コンソールを使用した仮想マシンとの相互作用 を参照してください。</p>
<pre> <input type='tablet' bus='usb'> <address type='usb' bus='0' port='1'> </input> <input type='mouse' bus='ps2'> <input type='keyboard' bus='ps2'> </pre>	<p>仮想マシンは、タブレット入力を受信するように設定された仮想 usb ポートと、マウスとキーボード入力を受け取るように設定された仮想 ps2 ポートを使用します。この設定は自動で設定されており、設定の変更は推奨されません。</p>
<pre> <graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'> <listen type='address' address='127.0.0.1'> </graphics> </pre>	<p>仮想マシンは、グラフィカル出力をレンダリングするために vnc プロトコルを使用します。</p>
<pre> <redirdev bus='usb' type='tcp'> <source mode='connect' host='localhost' service='4000'> <protocol type='raw'> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'> </memballoon> </devices> </domain> </pre>	<p>仮想マシンは、USB デバイスのリモート接続に tcp リダイレクターを使用し、メモリーバルーンがオンになっています。この設定は自動で設定されており、設定の変更は推奨されません。</p>

第10章 仮想マシンの保存および復元

システムリソースを解放するには、そのシステムで実行中の仮想マシンをシャットダウンできます。ただし、仮想マシンが再び必要になった場合は、ゲストオペレーティングシステム (OS) を起動し、アプリケーションを再起動する必要があります。これにはかなりの時間がかかる場合があります。このダウンタイムを短縮し、仮想マシンワークロードをより早く実行できるようにする場合は、保存機能および復元機能を使用して、OS のシャットダウンと起動シーケンスを完全に回避できます。

本セクションでは、仮想マシンの保存、および仮想マシンの完全な起動を行わずに仮想マシンを同じ状態に復元する方法を説明します。

10.1. 仮想マシンの保存および復元の仕組み

仮想マシンを保存すると、そのメモリーとデバイス状態がホストのディスクに保存され、仮想マシンプロセスがすぐに停止します。実行中または一時停止状態の仮想マシンを保存できます。復元すると、仮想マシンがその状態に戻ります。

このプロセスにより、ディスク容量と引き換えにホストシステムの RAM および CPU のリソースが解放され、ホストシステムのパフォーマンスが向上する場合があります。仮想マシンが復元する場合にゲスト OS を起動する必要がないため、時間がかかる起動時間も回避できます。

仮想マシンを保存するには、コマンドラインインターフェイス (CLI) を使用します。手順は、[コマンドラインインターフェイスを使用した仮想マシンの保存](#) を参照してください。

仮想マシンを復元するには、CLI または [Web コンソールの GUI](#) を使用します。

スナップショットを使用して仮想マシンの状態を保存および復元することもできます。詳細は、[スナップショットを使用した仮想マシンの状態の保存と復元](#) を参照してください。

10.2. コマンドラインインターフェイスを使用した仮想マシンの保存

仮想マシン (VM) と現在の状態をホストのディスクに保存できます。これは、たとえば、その他の目的でホストのリソースを使用する必要がある場合に便利です。保存した仮想マシンは、すぐに以前の稼働状態に復元できます。

コマンドラインを使用して仮想マシンを保存するには、以下の手順を行います。

前提条件

- 仮想マシンと設定を保存するのに十分なディスク領域がある。仮想マシンが占有する領域は、その仮想マシンに割り当てられている RAM のサイズによって異なることに注意してください。
- 仮想マシンが永続的である。
- (必要に応じて) 仮想マシンの重要なデータのバックアップを作成する。

手順

- **virsh managedsave** ユーティリティを使用します。
たとえば、次のコマンドは、仮想マシン **demo-guest1** を停止し、その設定を保存します。

```
# virsh managedsave demo-guest1
Domain 'demo-guest1' saved by libvirt
```

保存された仮想マシンファイルは、デフォルトで `/var/lib/libvirt/qemu/save` ディレクトリーに `demo-guest1.save` として置かれます。

次に仮想マシンを [起動](#) すると、上記のファイルから、保存された状態が自動的に復元します。

検証

- 管理保存が有効になっている仮想マシンを一覧表示します。以下の例では、`saved` として一覧表示されている仮想マシンで、管理保存が有効になっています。

```
# virsh list --managed-save --all
Id   Name                State
-----
-   demo-guest1         saved
-   demo-guest2         shut off
```

管理保存のイメージがある仮想マシンのリストを表示するには、次のコマンドを使用します。

```
# virsh list --with-managed-save --all
Id   Name                State
-----
-   demo-guest1         shut off
```

停止状態にある保存された仮想マシンのリストを表示するには、コマンドで `--all` オプションまたは `--inactive` オプションを使用する必要があります。

トラブルシューティング

- 保存した仮想マシンファイルが破損したり、読み込めない場合は、仮想マシンを復元すると、代わりに標準の仮想マシン起動が起動します。

関連情報

- [virsh managedsave --help](#) コマンド
- [コマンドラインインターフェイスを使用した保存済みの仮想マシンの復元](#)
- [Web コンソールを使用した保存済みの仮想マシンの復元](#)

10.3. コマンドラインインターフェイスでの仮想マシンの起動

コマンドラインインターフェイス (CLI) を使用して、シャットダウンした仮想マシン (VM) を起動するか、保存した仮想マシンを復元します。CLI を使用すると、ローカル仮想マシンとリモート仮想マシンの両方を起動できます。

前提条件

- すでに定義されている非アクティブな仮想マシン
- 仮想マシンの名前
- リモート仮想マシンの場合は、以下も設定されている。
 - 仮想マシンが置かれているホストの IP アドレス

- ホストへの root アクセス権限

手順

- ローカルの仮想マシンには、**virsh start** ユーティリティを使用します。たとえば、次のコマンドは仮想マシン **demo-guest1** を起動します。

```
# virsh start demo-guest1
Domain 'demo-guest1' started
```

- リモートホストにある仮想マシンでは、ホストへの QEMU+SSH 接続と共に **virsh start** ユーティリティを使用します。たとえば、次のコマンドは、ホスト 192.0.2.1 にある仮想マシン **demo-guest1** を起動します。

```
# virsh -c qemu+ssh://root@192.0.2.1/system start demo-guest1

root@192.0.2.1's password:

Domain 'demo-guest1' started
```

関連情報

- **virsh start --help** コマンド
- [リモートの仮想化ホストへの簡単なアクセスの設定](#)
- [ホストの起動時に仮想マシンを自動的に起動する](#)

10.4. WEB コンソールを使用した仮想マシンの起動

仮想マシンが **停止** 状態にある場合は、RHEL 9 Web コンソールを使用して起動できます。ホストの起動時に、仮想マシンが自動的に起動するように設定することもできます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- すでに定義されている非アクティブな仮想マシン
- 仮想マシンの名前

手順

1. **仮想マシン** インターフェイスで、起動する仮想マシンをクリックします。選択した仮想マシンの詳細情報を含む新しいページが開き、仮想マシンのシャットダウンおよび削除を制御できます。
2. **Run** をクリックします。仮想マシンが起動し、[そのコンソールまたはグラフィカル出力に接続](#) できます。
3. **オプション**: ホスト起動時に仮想マシンが自動的に起動するように設定するには、**Overview** セクションの **Autostart** チェックボックスを切り替えます。

libvirt が管理していないネットワークインターフェイスを使用する場合は、systemd 設定も変更する必要があります。そうしないと、影響を受ける仮想マシンが起動できなくなる可能性があります。[starting virtual machines automatically when the host starts](#) を参照してください。

関連情報

- [Web コンソールで仮想マシンのシャットダウン](#)
- [Web コンソールを使用した仮想マシンの再起動](#)

第11章 仮想マシンのクローン作成

特定のプロパティセットで仮想マシンの新規作成を行うには、既存の仮想マシンの **クローン** を作成します。

クローンを作成すると、ストレージ用に独自のディスクイメージを使用する新しい仮想マシンが作成されますが、クローン設定と保存データのほとんどはソース仮想マシンと同じです。これにより、各仮想マシンを個別に最適化せずに、特定のタスクに最適化された複数の仮想マシンを準備できます。

11.1. 仮想マシンのクローン作成の仕組み

仮想マシンのクローンを作成すると、ソース仮想マシンとそのディスクイメージの XML 設定がコピーされるため、新しい仮想マシンの一意性を確保するように設定を調整します。これには、仮想マシンの名前を変更して、ディスクイメージのクローンを使用するようにすることが含まれます。ただし、クローンの仮想ディスクに保存されているデータは、ソース仮想マシンと同じです。

このプロセスは、新しい仮想マシンを作成してゲストオペレーティングシステムと一緒にインストールするよりも高速であり、特定の設定およびコンテンツを持つ仮想マシンを迅速に生成するために使用できます。

仮想マシンの複数のクローンを作成することを計画している場合は、最初に、以下を含まない仮想マシン **テンプレート** を作成します。

- 永続的なネットワーク MAC 設定などの一意の設定。これにより、クローンが適切に機能しなくなる可能性があります。
- SSH キーやパスワードファイルなどの機密データ。

手順は、[Creating virtual machines templates](#) を参照してください。

関連情報

- [コマンドラインインターフェイスを使用した仮想マシンのクローン作成](#)
- [Web コンソールを使用した仮想マシンのクローン作成](#)

11.2. 仮想マシンテンプレートの作成

正常に機能する複数のクローン仮想マシンを作成するには、SSH 鍵や永続的なネットワーク MAC 設定などの、ソース仮想マシンに固有の情報および設定を削除します。これにより、仮想マシンのクローンを簡単かつ安全に作成するのに使用できる仮想マシン**テンプレート**が作成されます。

仮想マシンのテンプレートは、**virt-sysprep** ユーティリティーを使用して作成するか、要件に基づいて**手動で作成**することができます。

11.2.1. virt-sysprep を使用した仮想マシンテンプレートの作成

既存の仮想マシン (VM) から複製テンプレートを作成するには、**virt-sysprep** ユーティリティーを使用できます。これにより、特定のネットワーク設定やシステム登録メタデータなど、クローンが正しく機能しない可能性がある特定の設定が削除されます。その結果、**virt-sysprep** は仮想マシンのクローンをより効率的に作成し、クローンがより確実に動作するようにします。

前提条件

- **virt-sysprep** ユーティリティを含む **guestfs-tools** パッケージがホストにインストールされます。

```
# dnf install guestfs-tools
```

- テンプレートとして使用するソース仮想マシンがシャットダウンしている。
- ソース仮想マシンのディスクイメージの場所を把握しており、その仮想マシンのディスクイメージファイルの所有者である。
libvirt の [システムコネクション](#) で作成した仮想マシンのディスクイメージが、デフォルトで **/var/lib/libvirt/images** ディレクトリにあり、root ユーザーが所有している。

```
# ls -la /var/lib/libvirt/images
-rw-----. 1 root root 9665380352 Jul 23 14:50 a-really-important-vm.qcow2
-rw-----. 1 root root 8591507456 Jul 26 2017 an-actual-vm-that-i-use.qcow2
-rw-----. 1 root root 8591507456 Jul 26 2017 totally-not-a-fake-vm.qcow2
-rw-----. 1 root root 10739318784 Sep 20 17:57 another-vm-example.qcow2
```

- **オプション:** ソース仮想マシンのディスク上の重要なデータがすべてバックアップされている。ソース仮想マシンをそのまま保持する場合は、[クローン](#) を作成してから、そのクローンをテンプレートに変換します。

手順

1. 仮想マシンのディスクイメージの所有者としてログインしていることを確認します。

```
# whoami
root
```

2. (**必要に応じて**) 仮想マシンのディスクイメージをコピーします。

```
# cp /var/lib/libvirt/images/a-really-important-vm.qcow2 /var/lib/libvirt/images/a-really-important-vm-original.qcow2
```

これは後で、仮想マシンが正常にテンプレートに変換されたことを確認するために使用されません。

3. 次のコマンドを使用し、**/var/lib/libvirt/images/a-really-important-vm.qcow2** を、ソース仮想マシンのディスクイメージへのパスに置き換えます。

```
# virt-sysprep -a /var/lib/libvirt/images/a-really-important-vm.qcow2
[ 0.0] Examining the guest ...
[ 7.3] Performing "abrt-data" ...
[ 7.3] Performing "backup-files" ...
[ 9.6] Performing "bash-history" ...
[ 9.6] Performing "blkid-tab" ...
[...]
```

検証

- プロセスが成功したことを確認するには、変更したディスクイメージを元のイメージと比較します。次の例は、テンプレートの作成例を示しています。

```
# virt-diff -a /var/lib/libvirt/images/a-really-important-vm-orig.qcow2 -A
```

```

/var/lib/libvirt/images/a-really-important-vm.qcow2
-- 0644    1001 /etc/group-
-- 0000    797 /etc/gshadow-
= - 0444    33 /etc/machine-id
[...]
-- 0600    409 /home/username/.bash_history
- d 0700    6 /home/username/.ssh
-- 0600    868 /root/.bash_history
[...]

```

関連情報

- `virt-sysprep` の man ページの OPERATIONS セクション
- [コマンドラインインターフェイスを使用した仮想マシンのクローン作成](#)

11.2.2. 仮想マシンテンプレートの手動による作成

既存の仮想マシンからテンプレートを作成する場合は、ゲスト仮想マシンを手動でリセットまたは設定解除して、クローン作成の準備をします。

前提条件

- ソースの仮想マシンのディスクイメージの場所を把握しており、仮想マシンのディスクイメージファイルの所有者であることを確認します。
libvirt の [システムコネクション](#) で作成した仮想マシンのディスクイメージが、デフォルトで `/var/lib/libvirt/images` ディレクトリにあり、root ユーザーが所有している。

```

# ls -la /var/lib/libvirt/images
-rw-----. 1 root root 9665380352 Jul 23 14:50 a-really-important-vm.qcow2
-rw-----. 1 root root 8591507456 Jul 26 2017 an-actual-vm-that-i-use.qcow2
-rw-----. 1 root root 8591507456 Jul 26 2017 totally-not-a-fake-vm.qcow2
-rw-----. 1 root root 10739318784 Sep 20 17:57 another-vm-example.qcow2

```

- 仮想マシンがシャットダウンしていることを確認します。
- (必要に応じて) 仮想マシンのディスクにある重要なデータのバックアップが作成されている。ソースの仮想マシンをそのまま保持する場合は、[クローン](#) を作成してから、そのクローンを編集してテンプレートを作成します。

手順

- クローンを作成するように仮想マシンを設定します。
 - クローンに必要なソフトウェアをインストールします。
 - オペレーティングシステムに一意でない設定を設定します。
 - 固有でないアプリケーション設定を設定します。
- ネットワーク設定を削除します。
 - 以下のコマンドを使用して、永続的な udev ルールを削除します。

```
# rm -f /etc/udev/rules.d/70-persistent-net.rules
```



注記

udev ルールを削除しないと、最初の NIC の名前が **eth0** ではなく **eth1** になる場合があります。

- b. **/etc/NetworkManager/system-connections/** ディレクトリーの **NMConnection** ファイルから一意の情報を削除します。

- i. MAC アドレス、IP アドレス、DNS、ゲートウェイ、およびその他の一意の情報または望ましくない設定を削除します。

```
*ID=ExampleNetwork
BOOTPROTO="dhcp"
HWADDR="AA:BB:CC:DD:EE:FF"          <- REMOVE
NM_CONTROLLED="yes"
ONBOOT="yes"
TYPE="Ethernet"
UUID="954bd22c-f96c-4b59-9445-b39dd86ac8ab" <- REMOVE
```

- ii. 同様の一意の情報と望ましくない設定を **/etc/hosts** および **/etc/resolv.conf** ファイルから削除します。

3. 登録の詳細を削除します。

- Red Hat ネットワーク (RHN) に登録されている仮想マシンの場合:

```
# rm /etc/sysconfig/rhn/systemid
```

- Red Hat Subscription Manager (RHSM) に登録されている仮想マシンの場合:

- 元の仮想マシンを使用しない場合は、次のコマンドを実行します。

```
# subscription-manager unsubscribe --all # subscription-manager unregister #
subscription-manager clean
```

- 元の仮想マシンを使用する場合は、以下を行います。

```
# subscription-manager clean
```



注記

元の RHSM プロファイルは、ID コードとともにポータルに残ります。クローンの作成後、次のコマンドを使用して仮想マシンで RHSM 登録を再アクティブ化します。

```
# subscription-manager register --consumerid=71rd64fx-6216-4409-
bf3a-e4b7c7bd8ac9
```

4. その他の固有の詳細を削除します。

- a. SSH 公開鍵と秘密鍵のペアを削除します。

```
# rm -rf /etc/ssh/ssh_host_example
```

- b. LVM デバイスの設定を削除します。

```
# rm /etc/lvm/devices/system.devices
```

- c. 複数のマシンで実行している場合に、競合する可能性があるその他のアプリケーション固有の識別子や設定を削除します。

5. **gnome-initial-setup-done** ファイルを削除し、次回のシステムの起動時に設定ウィザードを実行するように仮想マシンを設定します。

```
# rm ~/.config/gnome-initial-setup-done
```



注記

次回の起動時に実行するウィザードは、仮想マシンから削除された設定によって異なります。また、クローンの初回起動時には、ホスト名を変更することが推奨されます。

11.3. コマンドラインインターフェイスを使用した仮想マシンのクローン作成

テストのために、特定のプロパティセットで新しい仮想マシンを作成するには、CLI を使用して既存の仮想マシンのクローンを作成します。

前提条件

- 移行元の仮想マシンがシャットダウンしている。
- クローンとして作成したディスクイメージを保存するのに十分なディスク領域があることを確認します。
- (必要に応じて) 仮想マシンのクローンを複数作成する場合は、ソースの仮想マシンから一意のデータと設定を削除して、クローンとして作成した仮想マシンが正しく機能することを確認することを推奨します。手順は、[仮想マシンテンプレートの作成](#) を参照してください。

手順

- 環境とユースケースに適したオプションを指定して **virt-clone** ユーティリティーを使用します。

サンプルのユースケース

- 次のコマンドは、**example-VM-1** という名前のローカル仮想マシンのクローンを作成し、**example-VM-1-clone** 仮想マシンを作成します。また、元の仮想マシンのディスクイメージと同じ場所に、同じデータで **example-VM-1-clone.qcow2** ディスクイメージを作成して割り当てます。

```
# virt-clone --original example-VM-1 --auto-clone
Allocating 'example-VM-1-clone.qcow2' | 50.0 GB 00:05:37

Clone 'example-VM-1-clone' created successfully.
```

- 次のコマンドは、**example-VM-2** という名前で仮想マシンのクローンを作成し、**example-VM-3** という名前でローカル仮想マシンを作成します。この仮想マシンは、**example-VM-2** の複数ディスクのうち 2 つだけを使用します。

```
# virt-clone --original example-VM-2 --name example-VM-3 --file
/var/lib/libvirt/images/disk-1-example-VM-2.qcow2 --file /var/lib/libvirt/images/disk-2-
example-VM-2.qcow2
Allocating 'disk-1-example-VM-2-clone.qcow2' | 78.0 GB 00:05:37
Allocating 'disk-2-example-VM-2-clone.qcow2' | 80.0 GB 00:05:37

Clone 'example-VM-3' created successfully.
```

- 仮想マシンを別のホストにクローンするには、ローカルホストで定義を解除せずに仮想マシンを移行します。たとえば、次のコマンドは、以前に作成した仮想マシン **example-VM-3** を **192.0.2.1** リモートシステムにローカルディスクを含めてクローンします。**192.0.2.1** に対して次のコマンドを実行するには root 権限が必要であることに注意してください。

```
# virsh migrate --offline --persistent example-VM-3 qemu+ssh://root@192.0.2.1/system
root@192.0.2.1's password:

# scp /var/lib/libvirt/images/<disk-1-example-VM-2-clone>.qcow2
root@192.0.2.1/<user@remote_host.com>:/var/lib/libvirt/images/

# scp /var/lib/libvirt/images/<disk-2-example-VM-2-clone>.qcow2
root@192.0.2.1/<user@remote_host.com>:/var/lib/libvirt/images/
```

検証

1. 仮想マシンのクローンが正常に作成され、正しく機能していることを確認するには、以下を行います。
 - a. クローンが、ホストの仮想マシンのリストに追加されていることを確認します。

```
# virsh list --all
Id Name State
-----
- example-VM-1 shut off
- example-VM-1-clone shut off
```

- b. クローンを起動し、起動しているかどうかを確認します。

```
# virsh start example-VM-1-clone
Domain 'example-VM-1-clone' started
```

関連情報

- [virt-clone\(1\) man ページ](#)
- [仮想マシンの移行](#)

11.4. WEB コンソールを使用した仮想マシンのクローン作成

特定のプロパティセットで新しい仮想マシンを作成するには、Web コンソールを使用して事前に設定した仮想マシンのクローンを作成します。



注記

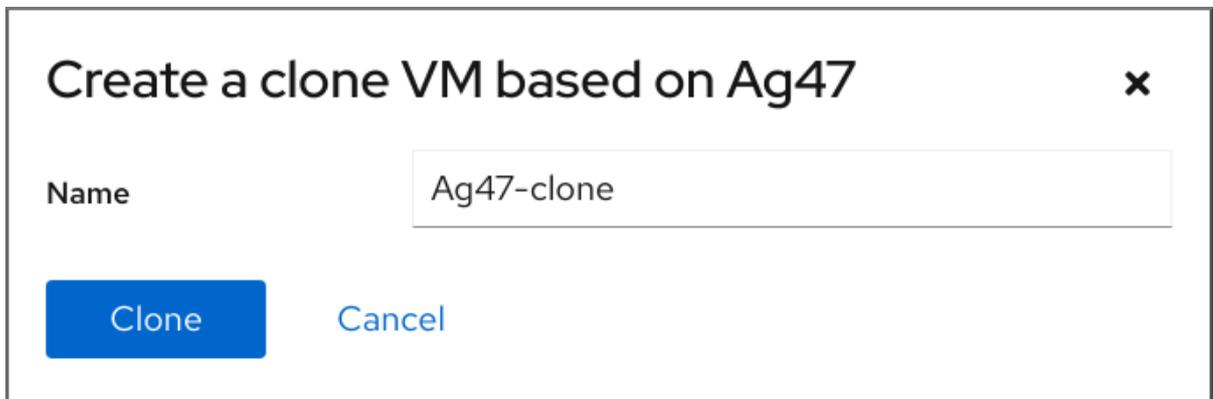
仮想マシンのクローンを作成すると、その仮想マシンに関連付けられたディスクのクローンも作成されます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- クローンを作成する仮想マシンがシャットダウンしていることを確認します。

手順

1. Web コンソールの仮想マシンインターフェイスで、クローンを作成する仮想マシンのメニューボタン  をクリックします。
仮想マシン操作を制御するためのドロップダウンメニューが表示されます。
2. **Clone** をクリックします。
仮想マシンのクローンの作成ダイアログが表示されます。



Create a clone VM based on Ag47 ×

Name

Clone Cancel

3. **オプション:** 仮想マシンクローンの新しい名前を入力します。
4. **Clone** をクリックします。
ソースの仮想マシンに基づいて、新しい仮想マシンが作成されます。

検証

- クローンとして作成された仮想マシンが、ホストで利用可能な仮想マシンのリストに表示されるかどうかを確認します。

第12章 仮想マシンの移行

仮想マシンの現在のホストが不安定な場合や使用できない場合や、ホストワークロードを再分散する場合は、仮想マシンを別の KVM ホストに移行できます。

12.1. 仮想マシンの移行の仕組み

仮想マシンの移行は、仮想マシンの XML 設定を別のホストマシンにコピーします。移行した仮想マシンがシャットダウンしていない場合、移行では、仮想マシンのメモリーと仮想デバイスの状態も移行先ホストマシンに転送されます。移行先ホストで仮想マシンが機能し続けるには、仮想マシンのディスクイメージが利用可能なままである必要があります。

デフォルトでは移行された仮想マシンは、移行先ホスト上では一時的なもので、移行元ホストでもそのまま定義されたままとなります。

ライブマイグレーションまたは **ライブ以外の** マイグレーションを使用して、実行中の仮想マシンを移行できます。シャットダウンした仮想マシンを移行するには、**オフライン** マイグレーションを使用する必要があります。詳細は、以下の表を参照してください。

表12.1 仮想マシンの移行タイプ

移行タイプ	説明	ユースケース	ストレージ要件
ライブマイグレーション	仮想マシンは移行元ホストマシンでそのまま実行を続け、KVM が仮想マシンのメモリーページを移行先ホストに転送します。移行がほぼ完了すると、KVM はごく短い間仮想マシンを中断し、移行先ホストで再開します。	常に稼働する必要がある仮想マシンに役に立ちます。ただし、I/O 負荷の多い仮想マシンなど、KVM の転送時間よりも、メモリーページの変更が早く行われる仮想マシンでは、ライブマイグレーションは使用できないので、 ライブマイグレーション以外の移行 を使用する必要があります。	仮想マシンのディスクイメージが 共有ネットワーク に存在し、移行元ホストと移行先ホストの両方からアクセスできる必要があります。
ライブマイグレーション以外の移行	仮想マシンを一時停止して、その設定とメモリーを移行先ホストにコピーし、仮想マシンを再開します。	仮想マシンに対するダウンタイムが発生しますが、一般的にはライブマイグレーションよりも信頼性が高くなります。メモリー負荷が大きい仮想マシンに推奨されます。	仮想マシンのディスクイメージが 共有ネットワーク に存在し、移行元ホストと移行先ホストの両方からアクセスできる必要があります。
オフラインマイグレーション	仮想マシンの設定を移行先ホストに移動します。	シャットダウンした仮想マシンや、仮想マシンをシャットダウンしてもワークロードを中断しない場合に推奨されます。	仮想マシンのディスクイメージは、共有ネットワークに配置する必要はなく、移行先ホストに手動でコピーまたは移動できます。

ライブマイグレーションと **ライブマイグレーション以外の移行** を組み合わせることもできます。これ

は、(移行の完了を阻止する)非常に多くの vCPU や大量のメモリーを使用する仮想マシンをライブマイグレーションする場合などに推奨されます。このようなシナリオでは、ソース仮想マシンを一時停止できます。これにより、追加のダーティーメモリーページが生成されなくなり、移行が完了する可能性が大幅に高くなります。ゲストのワークロードと移行中の静的ページ数に基づくと、このようなハイブリッド移行では、ライブマイグレーション以外の移行よりも、ダウンタイムが大幅に削減される可能性があります。

関連情報

- [仮想マシンの移行の利点](#)
- [他のホストとの仮想マシンディスクイメージの共有](#)

12.2. 仮想マシンの移行の利点

仮想マシンの移行は、以下の場合に役に立ちます。

ロードバランシング

ホストがオーバーロードするか、別のホストの使用率が低くなっている場合に、仮想マシンを使用率の低いホストマシンに移動できます。

ハードウェアの非依存性

ホストマシンでハードウェアデバイスのアップグレード、追加、削除などを行う必要がある場合は、仮想マシンをその他のホストに安全に移動できます。つまり、仮想マシンは、ハードウェアを改善する際にダウンタイムが生じることはありません。

エネルギー節約

仮想マシンはその他のホストに再配布できるため、電力使用量の少ない時間帯に、アンロードしたホストシステムの電源を切ることで、節電やコスト削減が可能になります。

地理的な移行

待ち時間の短縮や他の理由により、別の物理的な場所に仮想マシンを移動できます。

12.3. 仮想マシンの移行の制限事項

RHEL 9 で仮想マシンを移行する前に、移行の制限に注意してください。

- 仮想マシンと [libvirt のセッションコネクション間の移行](#) は信頼できないため、推奨されません。
- 特定の機能と設定を使用する仮想マシンは、移行すると正しく機能しなくなるか、移行が失敗します。このような機能は次のとおりです。
 - デバイスパススルー
 - SR-IOV デバイスの割り当て
 - vGPU などの仲介デバイス
- NUMA (Non-Uniform Memory Access) ピニングを使用するホスト間の移行は、ホストのトポロジーが類似している場合にのみ機能します。ただし、実行中のワークロードのパフォーマンスは、移行の影響を受ける可能性があります。
- 移行元仮想マシンと移行先仮想マシンの両方で、エミュレートしている CPU が同一である必要があります。同一でないと、移行が失敗します。以下の CPU 関連領域の仮想マシン間で相違があると、移行の問題が発生する可能性があります。

- CPU モデル
 - Intel 64 ホストと AMD64 ホスト間の移行は、x86-64 命令セットを共有している場合でも サポートされていません。
 - 別の CPU モデルを持つホストに移行した後に仮想マシンが正しく機能することを確認する手順は、[仮想マシン移行のためのホスト CPU の互換性の確認](#) を参照してください。
- ファームウェア設定
- Microcode バージョン
- BIOS バージョン
- BIOS 設定
- QEMU バージョン
- カーネルバージョン
- 1TB を超えるメモリーを使用する仮想マシンのライブマイグレーションは、一部のケースでは信頼できない場合があります。この問題を回避または修正する方法は、[仮想マシンのライブマイグレーションに長時間かかり、完了しない](#) を参照してください。

12.4. 仮想マシンの移行におけるホスト CPU の互換性の確認

移行した仮想マシン (VM) が移行先ホストで正しく機能するには、移行元および移行先のホストの CPU の互換性が必要です。これを確認するには、移行を開始する前に、共通の CPU ベースラインを計算します。



注記

本セクションの手順では、以下のホスト CPU で移行シナリオの例を使用します。

- 移行元ホスト : Intel Core i7-8650U
- 移行先ホスト : Intel Xeon CPU E5-2620 v2

前提条件

- 仮想化がシステムに [インストールされ有効になっている](#)。
- 移行元ホストと移行先ホストへの管理者アクセスがある。

手順

1. 移行元ホストで、CPU 機能を取得し、**domCaps-CPU.xml** などの新しい XML ファイルに貼り付けます。

```
# virsh domcapabilities | xmllint --xpath "//cpu/mode[@name='host-model']" - > domCaps-CPU.xml
```

2. XML ファイルで、**<mode> </mode>** タグを **<cpu> </cpu>** に置き換えます。

3. オプション : **domCaps-CPUs.xml** ファイルの内容が以下のようにになっていることを確認します。

```
# cat domCaps-CPUs.xml

<cpu>
  <model fallback="forbid">Skylake-Client-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy="require" name="ss"/>
  <feature policy="require" name="vmx"/>
  <feature policy="require" name="pdcml"/>
  <feature policy="require" name="hypervisor"/>
  <feature policy="require" name="tsc_adjust"/>
  <feature policy="require" name="clflushopt"/>
  <feature policy="require" name="umip"/>
  <feature policy="require" name="md-clear"/>
  <feature policy="require" name="stibp"/>
  <feature policy="require" name="arch-capabilities"/>
  <feature policy="require" name="ssbd"/>
  <feature policy="require" name="xsaves"/>
  <feature policy="require" name="pdpe1gb"/>
  <feature policy="require" name="invtscl"/>
  <feature policy="require" name="ibpb"/>
  <feature policy="require" name="ibrsl"/>
  <feature policy="require" name="amd-stibp"/>
  <feature policy="require" name="amd-ssbd"/>
  <feature policy="require" name="rsba"/>
  <feature policy="require" name="skip-l1dfl-vmentry"/>
  <feature policy="require" name="pschange-mc-no"/>
  <feature policy="disable" name="hle"/>
  <feature policy="disable" name="rtml"/>
</cpu>
```

4. 移行先ホストで以下のコマンドを使用して CPU 機能を取得します。

```
# virsh domcapabilities | xmllint --xpath "//cpu/mode[@name='host-model']" -

<mode name="host-model" supported="yes">
  <model fallback="forbid">IvyBridge-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy="require" name="ss"/>
  <feature policy="require" name="vmx"/>
  <feature policy="require" name="pdcml"/>
  <feature policy="require" name="pcid"/>
  <feature policy="require" name="hypervisor"/>
  <feature policy="require" name="arat"/>
  <feature policy="require" name="tsc_adjust"/>
  <feature policy="require" name="umip"/>
  <feature policy="require" name="md-clear"/>
  <feature policy="require" name="stibp"/>
  <feature policy="require" name="arch-capabilities"/>
  <feature policy="require" name="ssbd"/>
  <feature policy="require" name="xsaveopt"/>
  <feature policy="require" name="pdpe1gb"/>
  <feature policy="require" name="invtscl"/>
  <feature policy="require" name="ibpb"/>
```

```

    <feature policy="require" name="amd-ssbd"/>
    <feature policy="require" name="skip-l1dfl-vmentry"/>
    <feature policy="require" name="pschange-mc-no"/>
</mode>

```

- 移行先ホストから移行元ホストの **domCaps-CPUs.xml** ファイルに取得した CPU 機能を追加します。ここでも、**<mode> </mode>** タグを **<cpu> </cpu>** に置き換え、ファイルを保存します。
- オプション** : XML ファイルに両方のホストの CPU 機能が含まれていることを確認します。

```
# cat domCaps-CPUs.xml
```

```

<cpu>
  <model fallback="forbid">Skylake-Client-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy="require" name="ss"/>
  <feature policy="require" name="vmx"/>
  <feature policy="require" name="pdcn"/>
  <feature policy="require" name="hypervisor"/>
  <feature policy="require" name="tsc_adjust"/>
  <feature policy="require" name="clflushopt"/>
  <feature policy="require" name="umip"/>
  <feature policy="require" name="md-clear"/>
  <feature policy="require" name="stibp"/>
  <feature policy="require" name="arch-capabilities"/>
  <feature policy="require" name="ssbd"/>
  <feature policy="require" name="xsaves"/>
  <feature policy="require" name="pdpe1gb"/>
  <feature policy="require" name="invtsc"/>
  <feature policy="require" name="ibpb"/>
  <feature policy="require" name="ibrs"/>
  <feature policy="require" name="amd-stibp"/>
  <feature policy="require" name="amd-ssbd"/>
  <feature policy="require" name="rsba"/>
  <feature policy="require" name="skip-l1dfl-vmentry"/>
  <feature policy="require" name="pschange-mc-no"/>
  <feature policy="disable" name="hle"/>
  <feature policy="disable" name="rtm"/>
</cpu>
<cpu>
  <model fallback="forbid">IvyBridge-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy="require" name="ss"/>
  <feature policy="require" name="vmx"/>
  <feature policy="require" name="pdcn"/>
  <feature policy="require" name="pcid"/>
  <feature policy="require" name="hypervisor"/>
  <feature policy="require" name="arat"/>
  <feature policy="require" name="tsc_adjust"/>
  <feature policy="require" name="umip"/>
  <feature policy="require" name="md-clear"/>
  <feature policy="require" name="stibp"/>
  <feature policy="require" name="arch-capabilities"/>
  <feature policy="require" name="ssbd"/>
  <feature policy="require" name="xsaveopt"/>

```

```

<feature policy="require" name="pdpe1gb"/>
<feature policy="require" name="invtscl"/>
<feature policy="require" name="ibpb"/>
<feature policy="require" name="amd-ssbd"/>
<feature policy="require" name="skip-l1dfl-vmentry"/>
<feature policy="require" name="pschange-mc-no"/>
</cpu>

```

7. XML ファイルを使用して、移行する仮想マシンの CPU 機能ベースラインを計算します。

```
# virsh hypervisor-cpu-baseline domCaps-CPUs.xml
```

```

<cpu mode='custom' match='exact'>
<model fallback='forbid'>IvyBridge-IBRS</model>
<vendor>Intel</vendor>
<feature policy='require' name='ss'>
<feature policy='require' name='vmx'>
<feature policy='require' name='pdcml'>
<feature policy='require' name='pcid'>
<feature policy='require' name='hypervisor'>
<feature policy='require' name='arat'>
<feature policy='require' name='tsc_adjust'>
<feature policy='require' name='umip'>
<feature policy='require' name='md-clear'>
<feature policy='require' name='stibp'>
<feature policy='require' name='arch-capabilities'>
<feature policy='require' name='ssbd'>
<feature policy='require' name='xsaveopt'>
<feature policy='require' name='pdpe1gb'>
<feature policy='require' name='invtscl'>
<feature policy='require' name='ibpb'>
<feature policy='require' name='amd-ssbd'>
<feature policy='require' name='skip-l1dfl-vmentry'>
<feature policy='require' name='pschange-mc-no'>
</cpu>

```

8. 移行する仮想マシンの XML 設定を開き、**<cpu>** セクションの内容を直前の手順で取得した設定に置き換えます。

```
# virsh edit VM-name
```

9. 仮想マシンが実行中の場合は再起動します。

```
# virsh reboot VM-name
```

次のステップ

- [他のホストとの仮想マシンディスクイメージの共有](#)
- [コマンドラインインターフェイスを使用した仮想マシンの移行](#)
- [Web コンソールを使用した仮想マシンのライブ移行](#)

12.5. 他のホストとの仮想マシンディスクイメージの共有

対応している KVM ホスト 間で仮想マシンのライブマイグレーションを実行するには、仮想マシンの共有ストレージが必要です。次の手順では、NFS プロトコルを使用して、ローカルに保存された仮想マシンイメージをソースホストおよび宛先ホストと共有する方法について説明します。

前提条件

- 移行に使用する仮想マシンがシャットダウンしている。
- (必要に応じて) 移行元ホストまたは移行先ホストではないストレージをホストするのにホストシステムを使用できるが、移行元ホストと移行先ホストの両方がネットワーク経由でアクセスできる。これは共有ストレージに最適なソリューションで、Red Hat が推奨しています。
- KVM では対応していないため、NFS ファイルのロック機能を使用しない。
- NFS が移行元および移行先ホストにインストールされ、有効になっている。詳細は、以下を参照してください。
- [NFS サーバーのデプロイ](#)

手順

1. 共有ストレージを提供するホストに接続します。この例では、**example-shared-storage** ホストです。

```
# ssh root@example-shared-storage
root@example-shared-storage's password:
Last login: Mon Sep 24 12:05:36 2019
root~#
```

2. ディスクイメージを保持し、移行ホストと共有されるディレクトリーを移行元ホスト上に作成します。

```
# mkdir /var/lib/libvirt/shared-images
```

3. 移行元ホストから新規作成されたディレクトリーに仮想マシンのディスクイメージをコピーします。次の例では、仮想マシンのディスクイメージ **example-disk-1** を **example-shared-storage** ホストの **/var/lib/libvirt/shared-images/** ディレクトリーにコピーします。

```
# scp /var/lib/libvirt/images/example-disk-1.qcow2 root@example-shared-storage:/var/lib/libvirt/shared-images/example-disk-1.qcow2
```

4. ストレージを共有するのに使用するホストで、**/etc/exports** ファイルに共有ディレクトリーを追加します。次の例では、**/var/lib/libvirt/shared-images** ディレクトリーを **example-source-machine** ホストおよび **example-destination-machine** ホストと共有します。

```
# /var/lib/libvirt/shared-images example-source-machine(rw,no_root_squash) example-destination-machine(rw,no_root_squash)
```

5. 移行元ホストと移行先ホストの両方で、共有ディレクトリーを **/var/lib/libvirt/images** ディレクトリーにマウントします。

```
# mount example-shared-storage:/var/lib/libvirt/shared-images /var/lib/libvirt/images
```

検証

- 移行元ホストで仮想マシンを起動し、正常に起動するかどうかを確認します。

関連情報

- [NFS サーバーのデプロイ](#)

12.6. コマンドラインインターフェイスを使用した仮想マシンの移行

仮想マシンの現在のホストが不安定な場合や使用できない場合や、ホストワークロードを再分散する場合は、仮想マシンを別の KVM ホストに移行できます。次の手順では、このような移行のさまざまなシナリオの手順と例を示します。

前提条件

- 移行元ホストと移行先ホストはいずれも KVM ハイパーバイザーを使用します。
- 移行元ホストと移行先のホストは、ネットワーク経由で相互に通信できなければなりません。 **ping** ユーティリティを使用してこれを確認します。
- 移行先ホストで以下のポートが開いていることを確認します。
 - ポート 22 は、SSH を使用して宛先ホストに接続するために必要です。
 - ポート 16509 は、TLS を使用して宛先ホストに接続するために必要です。
 - ポート 16514 は、TCP を使用して宛先ホストに接続するために必要です。
 - ポート 49152-49215 は、QEMU がメモリーおよびディスク移行データを転送するために必要です。
- Red Hat が移行に対応できるようにするには、移行元ホストと移行先のホストが特定のオペレーティングシステムとマシンタイプを使用している必要があります。これを確認するには、[Supported hosts for virtual machine migration](#) を参照してください。
- 仮想マシンは、移行先ホストの CPU 機能と互換性がある必要があります。これを確認するには、[仮想マシン移行のホスト CPU の互換性の確認](#) を参照してください。
- 移行する仮想マシンのディスクイメージが、ソースホストと宛先ホストの両方にアクセスできる別のネットワーク上の場所にある。オフラインマイグレーションの場合は任意ですが、実行中の仮想マシンの移行に必要になります。
このような仮想マシンの共有ストレージを設定する手順は、[Sharing virtual machine disk images with other hosts](#) を参照してください。
- 仮想マシンの実行中に移行する場合は、ネットワークの帯域幅が、仮想マシンがダーティーメモリーページを生成する速度を超える必要があります。
ライブマイグレーションを開始する前に仮想マシンのダーティーページ速度を取得するには、次の手順を実行します。
 - 短期間、仮想マシンのダーティーページ生成速度を監視します。

```
# virsh domdirtyrate-calc example-VM 30
```

- 監視が終了したら、結果を取得します。

```
# virsh domstats example-VM --dirtyrate
Domain: 'example-VM'
dirtyrate.calc_status=2
dirtyrate.calc_start_time=200942
dirtyrate.calc_period=30
dirtyrate.megabytes_per_second=2
```

この例では、仮想マシンが1秒あたり 2MB のダーティメモリーページを生成しています。帯域幅が 2MB/s 以下のネットワーク上でこのような仮想マシンをライブマイグレーションしようとする、仮想マシンを一時停止したり、ワークロードを低くしたりしないと、ライブマイグレーションが進行しません。

ライブマイグレーションが正常に終了するように、Red Hat では、ネットワークの帯域幅が仮想マシンのダーティページの生成速度を大幅に上回ることを推奨しています。

- パブリックブリッジネットワークの既存の仮想マシンで移行を行う場合は、移行元ホストと移行先ホストが同じネットワークにある必要があります。そうでない場合は、移行後に仮想マシンのネットワークが機能しなくなります。



注記

calc_period オプションの値は、ワークロードとダーティページ速度により異なる場合があります。いくつかの **calc_period** 値を試して、環境のダーティページ速度に合わせた最適な期間を決定できます。

- VM 移行を実行する場合、ソースホスト上の **virsh** クライアントは、いくつかのプロトコルの1つを使用して、宛先ホスト上の libvirt デーモンに接続できます。次の手順の例では SSH 接続を使用していますが、別の接続を選択することもできます。
 - libvirt で SSH 接続を使用する場合は、**virtqemud** ソケットが有効になっていて、宛先ホストで実行されていることを確認してください。

```
# systemctl enable --now virtqemud.socket
```

- libvirt で TLS 接続を使用する場合は、**virtproxyd-tls** ソケットが有効になっていて、宛先ホストで実行していることを確認してください。

```
# systemctl enable --now virtproxyd-tls.socket
```

- libvirt で TCP 接続を使用する場合は、**virtproxyd-tcp** ソケットが有効になっていて、宛先ホストで実行していることを確認してください。

```
# systemctl enable --now virtproxyd-tcp.socket
```

手順

- virsh migrate** コマンドで、移行の要件に適したオプションを指定します。
 - 次のコマンドは、SSH トンネルを使用して、ローカルホストから **example-destination** ホストのシステム接続に **example-VM-1** 仮想マシンを移行します。仮想マシンは移行中も稼働し続けます。

```
# virsh migrate --persistent --live example-VM-1 qemu+ssh://example-destination/system
```

- b. 次のコマンドを使用すると、ローカルホストで実行している **example-VM-2** 仮想マシンの設定を手動で調整し、その仮想マシンを **example-destination** ホストに移行できます。移行した仮想マシンが更新された設定を自動的に使用します。

```
# virsh dumpxml --migratable example-VM-2 > example-VM-2.xml
# vi example-VM-2.xml
# virsh migrate --live --persistent --xml example-VM-2.xml example-VM-2
qemu+ssh://example-destination/system
```

この手順は、たとえば、移行先ホストが別のパスを使用して仮想マシンの共有ストレージにアクセスする必要がある場合、または移行先ホストに固有の機能を設定する場合に役立ちます。

- c. 次のコマンドは、**example-VM-3** 仮想マシンを **example-source** ホストで一時停止して **example-destination** ホストに移行し、**example-VM-3-alt.xml** ファイルが提供する調整済みの XML 設定を使用するように当該仮想マシンに指示します。移行が終了すると、**libvirt** は移行先ホストで仮想マシンを再開します。

```
# virsh migrate example-VM-3 qemu+ssh://example-source/system
qemu+ssh://example-destination/system --xml example-VM-3-alt.xml
```

移行後、仮想マシンはソースホストでシャットオフ状態になり、移行されたコピーはシャットダウン後に削除されます。

- d. 次の例では、シャットダウンされた **example-VM-4** 仮想マシンを **example-source** ホストから削除し、その設定を **example-destination** ホストに移動します。

```
# virsh migrate --offline --persistent --undefinesource example-VM-4
qemu+ssh://example-source/system qemu+ssh://example-destination/system
```

このタイプの移行では、仮想マシンのディスクイメージを共有ストレージに移動する必要がないことに注意してください。ただし、移行先ホストで仮想マシンを使用するには、仮想マシンのディスクイメージも移行する必要があります。以下に例を示します。

```
# scp root@example-source:/var/lib/libvirt/images/example-VM-4.qcow2
root@example-destination:/var/lib/libvirt/images/example-VM-4.qcow2
```

- e. 次のコマンドは、**example-VM-5** 仮想マシンを **example-destination** ホストに移行し、複数の並列接続 (マルチファイル記述子 (マルチ FD) 移行とも呼ばれます) を使用します。マルチ FD 移行では、移行プロセスに利用可能なネットワーク帯域幅をすべて利用することで、移行を高速化できます。

```
# virsh migrate --parallel --parallel-connections 4 <example-VM-5>
qemu+ssh://<example-destination>/system
```

この例では、4つのマルチ FD チャンネルを使用して **example-VM-5** 仮想マシンを移行します。利用可能なネットワーク帯域幅 10 Gbps ごとに1つのチャンネルを使用することを推奨します。デフォルト値は2チャンネルです。

2. 移行が完了するまで待ちます。ネットワーク帯域幅、システムの負荷、仮想マシンのサイズによっては、プロセスに時間がかかる場合があります。**virsh migrate** で **--verbose** オプションが使用されていないと、CLI はエラー以外の進捗インジケータを表示しません。移行中は、**virsh domjobinfo** ユーティリティを使用して移行の統計を表示できます。

検証

- 移行先ホストで、使用可能な仮想マシンのリストを表示して、仮想マシンが移行されたかどうかを確認します。

```
# virsh list
Id   Name           State
-----
10   example-VM-1   running
```

移行がまだ実行中であれば、このコマンドは、**paused** の仮想マシンのリストを表示します。

トラブルシューティング

- ターゲットのホストは、ネットワーク名や CPU タイプなど、移行した仮想マシンの XML 設定で使用される特定の値と互換性がない場合があります。そのため、仮想マシンがターゲットホストで起動できなくなります。この問題を修正するには、**virsh edit** コマンドを使用して問題のある値を更新します。値を更新した後、変更を適用するには仮想マシンを再起動する必要があります。
- ライブマイグレーションの完了に時間がかかっている場合は、仮想マシンの負荷が高く、ライブマイグレーションを実行するために変更しているメモリーページ多すぎる可能性があります。この問題を修正するには、仮想マシンを停止して、ライブ以外への移行に変更します。

```
# virsh suspend example-VM-1
```

関連情報

- **virsh migrate --help** コマンド
- **virsh(1)** man ページ

12.7. WEB コンソールを使用した仮想マシンのライブ移行

継続的に実行する必要があるタスクを実行している仮想マシンを移行する場合は、シャットダウンせずに、その仮想マシンを別の KVM ホストに移行できます。これはライブマイグレーションとも呼ばれます。以下の手順では、Web コンソールを使用した移行方法を説明します。



警告

I/O 負荷が高いタスクなど、KVM がメモリーページを転送するよりも速い速度でメモリーページを変更するタスクには、仮想マシンをライブマイグレーションしないことが推奨されます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 移行元ホストと移行先ホストが実行中である。

- 移行先ホストで以下のポートが開いていることを確認します。
 - ポート 22 は、SSH を使用して宛先ホストに接続するために必要です。
 - ポート 16509 は、TLS を使用して宛先ホストに接続するために必要です。
 - ポート 16514 は、TCP を使用して宛先ホストに接続するために必要です。
 - ポート 49152-49215 は、QEMU がメモリーおよびディスク移行データを転送するために必要です。
- 仮想マシンは、移行先ホストの CPU 機能と互換性がある必要があります。これを確認するには、[仮想マシン移行のホスト CPU の互換性の確認](#) を参照してください。
- 仮想マシンのディスクイメージは、移行元ホストおよび移行先ホストからアクセス可能な [共有ストレージ](#) に配置されています。
- 仮想マシンの実行中に移行する場合は、ネットワークの帯域幅が、仮想マシンがダーティメモリーページを生成する速度を超える必要があります。ライブマイグレーションを開始する前に仮想マシンのダーティページ速度を取得するには、コマンドラインインターフェイスで次の手順を行います。
 - a. 短期間、仮想マシンのダーティページ生成速度を監視します。

```
# virsh domdirtyrate-calc vm-name 30
```

- b. 監視が終了したら、結果を取得します。

```
# virsh domstats vm-name --dirtyrate
Domain: 'vm-name'
dirtyrate.calc_status=2
dirtyrate.calc_start_time=200942
dirtyrate.calc_period=30
dirtyrate.megabytes_per_second=2
```

この例では、仮想マシンが1秒あたり 2MB のダーティメモリーページを生成しています。帯域幅が 2MB/s 以下のネットワーク上でこのような仮想マシンをライブマイグレーションしようとする、仮想マシンを一時停止したり、ワークロードを低くしたりしないと、ライブマイグレーションが進行しません。

ライブマイグレーションが正常に終了するように、Red Hat では、ネットワークの帯域幅が仮想マシンのダーティページの生成速度を大幅に上回ることを推奨しています。

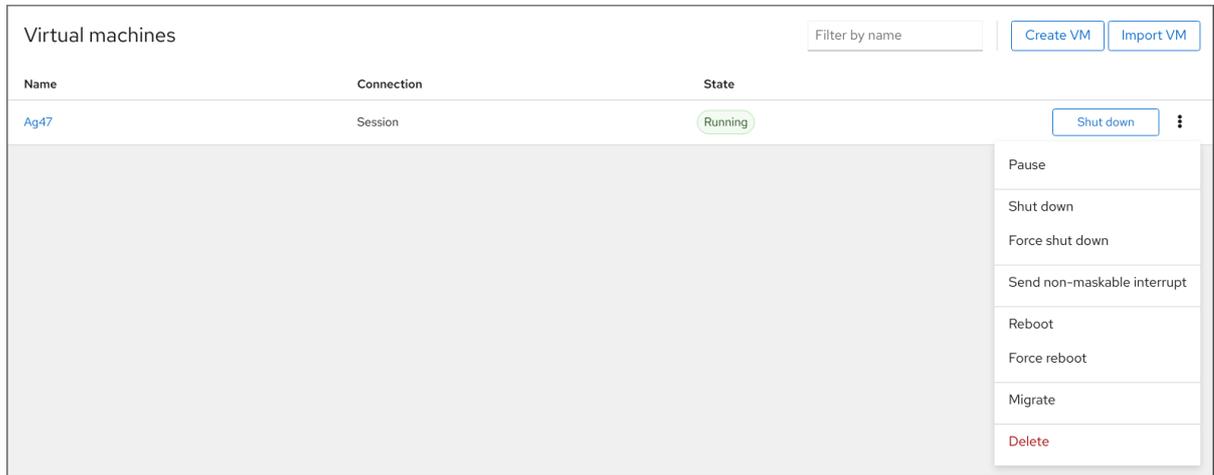


注記

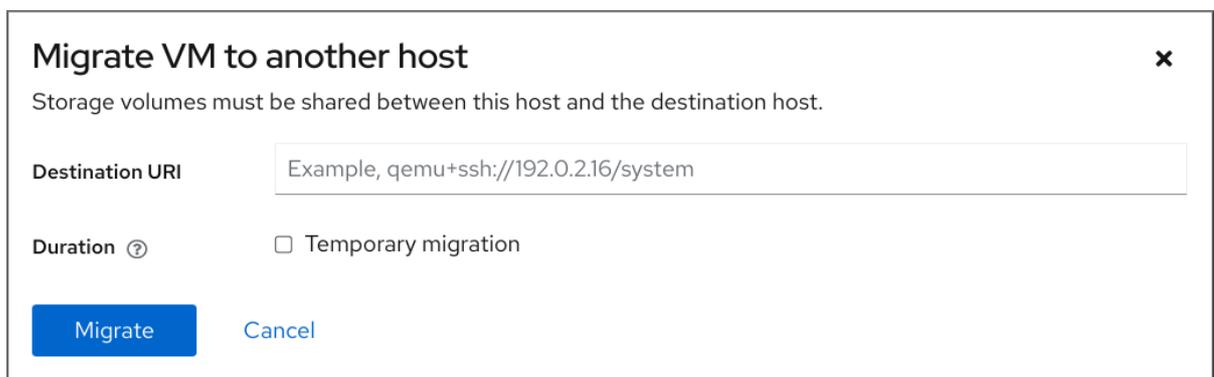
`calc_period` オプションの値は、ワークロードとダーティページ速度により異なる場合があります。いくつかの `calc_period` 値を試して、環境のダーティページ速度に合わせた最適な期間を決定できます。

手順

1. Web コンソールの仮想マシンインターフェイスで、移行する仮想マシンのメニュー ボタン  をクリックします。仮想マシン操作を制御するためのドロップダウンメニューが表示されます。



2. **Migrate** をクリックします。
仮想マシンを別のホストに移行ダイアログボックスが表示されます。



3. 宛先ホストの URI を入力します。
4. 移行の期間を設定します。
 - **Permanent** - 仮想マシンを永続的に移行する場合はチェックを外します。永続的な移行では、移行元ホストから仮想マシンの設定が完全に削除されます。
 - **Temporary** - 一時的な移行では、仮想マシンのコピーを移行先ホストに移行します。このコピーは、仮想マシンのシャットダウン時に移行先ホストから削除されます。元の仮想マシンは、ソースホストに残ります。
5. **Migrate** をクリックします。
仮想マシンが移行先ホストに移行されます。

検証

仮想マシンの移行に成功し、正常に機能しているかどうかを確認するには、次のコマンドを実行します。

- 移行先ホストで利用可能な仮想マシンのリストに仮想マシンが表示されているかどうかを確認します。
- 移行した仮想マシンを起動し、起動するかどうかを確認します。

12.8. MELLANOX VIRTUAL FUNCTION が割り当てられた仮想マシンのライブマイグレーション

テクノロジープレビューとして、Mellanox ネットワークデバイスの Virtual Function (VF) が割り当てられた仮想マシン (VM) のライブマイグレーションを利用できます。現在、これは Mellanox CX-7 ネットワークデバイスを使用している場合にのみ可能です。Mellanox CX-7 ネットワークデバイス上の VF は、ライブマイグレーションに必要な機能を追加する新しい `mlx5_vfio_pci` ドライバーを使用します。この新しいドライバーは、`libvirt` によって VF に自動的にバインドされます。

制限事項

現在、Mellanox Virtual Function が割り当てられた仮想マシンをライブマイグレーションする場合、以下の仮想化機能を使用できません。

- 仮想マシンのダーティメモリーページレート生成の計算
- コピー後のライブマイグレーションの使用
- 仮想マシンでの仮想 I/O Memory Management Unit (vIOMMU) デバイスの使用



重要

この機能は RHEL 9 に [テクノロジープレビュー](#) のみの機能として組み込まれているため、サポート対象外です。

前提条件

- ファームウェアバージョンが `28.36.1010` 以上の Mellanox CX-7 ネットワークデバイスを使用している。
ファームウェアバージョンの詳細は、[Mellanox のドキュメント](#) を参照してください。

- `mstflint` パッケージが、ソースホストと宛先ホストの両方にインストールされている。

```
# dnf install mstflint
```

- Mellanox CX-7 ネットワークデバイスで、`VF_MIGRATION_MODE` が `MIGRATION_ENABLED` に設定されている。

```
# mstconfig -d <device_pci_address> query | grep -i VF_migration
```

```
VF_MIGRATION_MODE          MIGRATION_ENABLED(2)
```

- 次のコマンドを使用して、`VF_MIGRATION_MODE` を `MIGRATION_ENABLED` に設定できます。

```
# mstconfig -d <device_pci_address> set VF_MIGRATION_MODE=2
```

- `openvswitch` パッケージが、ソースホストと宛先ホストの両方にインストールされている。

```
# dnf install openvswitch
```

- ホストの CPU およびファームウェアは、IOMMU (I/O Memory Management Unit) に対応している。
 - Intel CPU を使用している場合は、Intel VT-d (Virtualization Technology for Directed I/O) に対応する必要があります。
 - AMD CPU を使用している場合は、AMD-Vi 機能に対応している必要があります。

- ホストシステムが、アクセス制御サービス (ACS) を使用して PCIe トポロジーの DMA (Direct Memory Access) 分離を提供している。この点をシステムベンダーに確認してください。詳細は、[SR-IOV 実装に関するハードウェアの考慮事項](#) を参照してください。
- VF の作成に使用するホストのネットワークインターフェイスが実行中である。たとえば、**eth1** インターフェイスをアクティブにして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。

```
# ip link set eth1 up
# ip link show eth1
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT qlen 1000
    link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

- SR-IOV デバイス割り当てを有効にするには、ホスト BIOS およびカーネルで IOMMU 機能を有効にする必要があります。これを行うには、以下を行います。

- Intel ホストで、Intel Virtualization Technology for Directed I/O (VT-d) を有効にします。
 - i. **intel_iommu=on** および **iommu=pt** パラメーターを使用して GRUB 設定を再生成します。

```
# grubby --args="intel_iommu=on iommu=pt" --update-kernel=ALL
```

- ii. ホストを再起動します。

- AMD ホストで、AMD-Vi を有効にします。
 - i. **iommu=pt** パラメーターで GRUB 設定を再生成します。

```
# grubby --args="iommu=pt" --update-kernel=ALL
```

- ii. ホストを再起動します。

- 移行元ホストと移行先ホストはいずれも KVM ハイパーバイザーを使用します。
- 移行元ホストと移行先のホストは、ネットワーク経由で相互に通信できなければなりません。**ping** ユーティリティを使用してこれを確認します。
- 宛先ホストで次のポートが開いている。
 - ポート 22 は、SSH を使用して宛先ホストに接続するために必要です。
 - ポート 16509 は、TLS を使用して宛先ホストに接続するために必要です。
 - ポート 16514 は、TCP を使用して宛先ホストに接続するために必要です。
 - ポート 49152 - 49215 は、QEMU がメモリーおよびディスク移行データを転送するために必要です。

- ソースホストと宛先ホストが、移行可能なオペレーティングシステムとマシンタイプを使用している。これを確認するには、[Supported hosts for virtual machine migration](#) を参照してください。
- 仮想マシンは、移行先ホストの CPU 機能と互換性がある必要があります。これを確認するには、[仮想マシン移行のホスト CPU の互換性の確認](#) を参照してください。
- 移行する仮想マシンのディスクイメージが、ソースホストと宛先ホストの両方にアクセスできる別のネットワーク上の場所にある。オフラインマイグレーションの場合は任意ですが、実行中の仮想マシンの移行に必要になります。
このような仮想マシンの共有ストレージを設定する手順は、[Sharing virtual machine disk images with other hosts](#) を参照してください。
- 仮想マシンの実行中に移行する場合は、ネットワークの帯域幅が、仮想マシンがダーティメモリーページを生成する速度を超える必要があります。
- 接続プロトコルに対応する仮想ネットワークソケットが有効になっている。
VM 移行を実行する場合、ソースホスト上の **virsh** クライアントは、いくつかのプロトコルの1つを使用して、宛先ホスト上の libvirt デーモンに接続できます。次の手順の例では SSH 接続を使用していますが、別の接続を選択することもできます。* libvirt で SSH 接続を使用する場合は、宛先ホストで **virtqemud** ソケットが有効になっていて実行中であることを確認してください。

+

```
# systemctl enable --now virtqemud.socket
```

- libvirt で TLS 接続を使用する場合は、**virtproxyd-tls** ソケットが有効になっていて、宛先ホストで実行していることを確認してください。

```
# systemctl enable --now virtproxyd-tls.socket
```

- libvirt で TCP 接続を使用する場合は、**virtproxyd-tcp** ソケットが有効になっていて、宛先ホストで実行していることを確認してください。

```
# systemctl enable --now virtproxyd-tcp.socket
```

手順

1. ソースホストで、Mellanox ネットワークデバイスを **switchdev** モードに設定します。

```
# devlink dev eswitch set pci/<device_pci_address> mode switchdev
```

2. ソースホストで、Mellanox デバイス上に Virtual Function を作成します。

```
# echo 1 > /sys/bus/pci/devices/0000:e1:00.0/sriov_numvfs
```

ファイルパスの **/0000:e1:00.0/** の部分は、デバイスの PCI アドレスに基づいています。この例では、**0000:e1:00.0** です。

3. ソースホストで、VF をそのドライバーからアンバインドします。

```
# virsh nodedev-detach <vf_pci_address> --driver pci-stub
```

次のコマンドを使用して、VF の PCI アドレスを表示できます。

```
# lshw -c network -businfo
```

Bus info	Device	Class	Description
pci@0000:e1:00.0	enp225s0np0	network	MT2910 Family [ConnectX-7]
pci@0000:e1:00.1	enp225s0v0	network	ConnectX Family mlx5Gen Virtual Function

4. ソースホストで、VF の移行機能を有効にします。

```
# devlink port function set pci/0000:e1:00.0/1 migratable enable
```

この例の **pci/0000:e1:00.0/1** は、指定の PCI アドレスを持つ Mellanox デバイス上の最初の VF を示しています。

5. ソースホストで、VF の移行用に Open vSwitch (OVS) を設定します。Mellanox デバイスが **switchdev** モードの場合、ネットワーク経由でデータを転送できません。

- a. **openvswitch** サービスが実行中であることを確認します。

```
# systemctl start openvswitch
```

- b. ネットワークのパフォーマンスを向上させるために、ハードウェアオフロードを有効にします。

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

- c. 移行中にネットワーク接続が開いたままになるように、最大アイドル時間を増やします。

```
# ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
```

- d. OVS インスタンスに新しいブリッジを作成します。

```
# ovs-vsctl add-br <bridge_name>
```

- e. **openvswitch** サービスを再起動します。

```
# systemctl restart openvswitch
```

- f. 物理的な Mellanox デバイスを OVS ブリッジに追加します。

```
# ovs-vsctl add-port <bridge_name> enp225s0np0
```

この例では、**<bridge_name>** はステップ d で作成したブリッジの名前であり、**enp225s0np0** は Mellanox デバイスのネットワークインターフェイス名です。

- g. Mellanox デバイスの VF を OVS ブリッジに追加します。

```
# ovs-vsctl add-port <bridge_name> enp225s0npf0vf0
```

この例では、**<bridge_name>** はステップ d で作成したブリッジの名前であり、**enp225s0npf0vf0** は VF のネットワークインターフェイス名です。

- 宛先ホスト でステップ 1-5 を繰り返します。
- ソースホストで、**mlx_vf.xml** などの新しいファイルを開き、次のような VF の XML 設定を追加します。

```
<interface type='hostdev' managed='yes'>
  <mac address='52:54:00:56:8c:f7'/>
  <source>
    <address type='pci' domain='0x0000' bus='0xe1' slot='0x00' function='0x1'/>
  </source>
</interface>
```

この例では、VF のパススルーを仮想マシンのネットワークインターフェイスとして設定します。MAC アドレスが一意であることを確認し、ソースホスト上の VF の PCI アドレスを使用します。

- ソースホストで、VF の XML ファイルを仮想マシンに割り当てます。

```
# virsh attach-device <vm_name> mlx_vf.xml --live --config
```

この例の **mlx_vf.xml** は、VF 設定を含む XML ファイルの名前です。実行中の仮想マシンにデバイスを割り当てるために、**--live** オプションを使用します。

- ソースホストで、VF が割り当てられた実行中の仮想マシンのライブマイグレーションを開始します。

```
# virsh migrate --live --domain <vm_name> --desturi
qemu+ssh://<destination_host_ip_address>/system
```

検証

- 移行された仮想マシンで、Mellanox VF のネットワークインターフェイス名を表示します。

```
# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.1.10 netmask 255.255.255.0 broadcast 192.168.1.255
  inet6 fe80::a00:27ff:fe4e:66a1 prefixlen 64 scopeid 0x20<link>
  ether 08:00:27:4e:66:a1 txqueuelen 1000 (Ethernet)
  RX packets 100000 bytes 6543210 (6.5 MB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 100000 bytes 6543210 (6.5 MB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp4s0f0v0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.3.10 netmask 255.255.255.0 broadcast 192.168.3.255
  inet6 fe80::a00:27ff:fe4e:66c3 prefixlen 64 scopeid 0x20<link>
  ether 08:00:27:4e:66:c3 txqueuelen 1000 (Ethernet)
  RX packets 200000 bytes 12345678 (12.3 MB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 200000 bytes 12345678 (12.3 MB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

2. 移行した仮想マシンで、Mellanox VF が動作することを確認します。次に例を示します。

```
# ping -I <VF_interface_name> 8.8.8.8

PING 8.8.8.8 (8.8.8.8) from 192.168.3.10 <VF_interface_name>: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=57 time=27.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=57 time=26.9 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 26.944/27.046/27.148/0.102 ms
```

関連情報

- [コマンドラインインターフェイスを使用した仮想マシンの移行](#)
- [仮想マシンの移行に関するトラブルシューティング](#)

12.9. 仮想マシンの移行に関するトラブルシューティング

仮想マシン (VM) を移行する際に、以下のいずれかの問題が発生した場合は、手順を参照して問題を修正または回避してください。

12.9.1. 仮想マシンのライブマイグレーションに長時間かかり、完了しない

原因

場合によっては、実行中の仮想マシンを移行することにより、その仮想マシンは、**ダーティーメモリーページ**の移行速度よりも速いスピードで、**ダーティーメモリーページ**を生成することがあります。このような場合、移行は正常に完了できません。

この問題は、以下のシナリオにより、頻繁に発生します。

- 負荷が大きい仮想マシンのライブマイグレーション
- 大量のメモリー (1TB 以上) を使用する仮想マシンのライブマイグレーション



重要

Red Hat は、最大 6 TB のメモリーを搭載した仮想マシンのライブマイグレーションを正常にテストしました。ただし、メモリーが 1TB を超える仮想マシンのライブマイグレーションに関しては、[Red Hat テクニカルサポート](#) までお問い合わせください。

診断

仮想マシンのライブマイグレーションにかかる時間が予想よりも長い場合は、**virsh domjobinfo** コマンドを使用して、仮想マシンのメモリーページデータを取得します。

```
# virsh domjobinfo vm-name

Job type:      Unbounded
Operation:     Outgoing migration
Time elapsed:  168286974 ms
```

```

Data processed: 26.106 TiB
Data remaining: 34.383 MiB
Data total: 10.586 TiB
Memory processed: 26.106 TiB
Memory remaining: 34.383 MiB
Memory total: 10.586 TiB
Memory bandwidth: 29.056 MiB/s
Dirty rate: 17225 pages/s
Page size: 4096 bytes

```

この出力では、**Dirty rate** と **Page size** を乗算すると、**Memory bandwidth** より大きくなります。これは、ネットワークがダーティーページを移行できるよりも速い速度で、仮想マシンがダーティーメモリーページを生成していることを意味します。そのため、移行先ホストの仮想マシンの状態を移行元ホストの仮想マシンの状態に収束することができません。これにより、移行が完了しません。

修正

停止状態のライブマイグレーションが正常に終了する確率を高めるには、以下のいずれかを行います。

- 仮想マシンのワークロード、特にメモリー更新を減らします。
 - これを行うには、元の仮想マシンのゲストオペレーティングシステムで必須ではないプロセスを停止またはキャンセルします。
- ライブマイグレーションで許容されるダウンタイムを増やします。
 - a. 移行中の仮想マシンのライブマイグレーションの最後に、現在の最大ダウンタイムを表示します。

```
# virsh migrate-getmaxdowntime vm-name
```

- b. 最大ダウンタイムを長く設定します。

```
# virsh migrate-setmaxdowntime vm-name downtime-in-milliseconds
```

最大ダウンタイムを長く設定すればするほど、移行の完了までにかかり時間が長くなる可能性があります。

- ライブマイグレーションを **post-copy** モードに切り替えます。

```
# virsh migrate-start-postcopy vm-name
```

- これにより、仮想マシンのメモリーページが移行先ホストで収束し、移行が完了できるようになります。
ただし、**post-copy** モードがアクティブになると、移行先ホストから移行元ホストへのリモートページ要求が原因で、仮想マシンが大幅に遅くなる可能性があります。さらに、**post-copy** マイグレーション中に移行元ホストと移行先ホスト間のネットワーク接続が動作しなくなった場合、メモリーページが不足しているために一部の仮想マシンプロセスが停止することがあります。

したがって、仮想マシンの可用性が重要である場合や、移行ネットワークが不安定な場合は、**post-copy** マイグレーションを使用しないでください。

- ワークロードで許可されている場合は、仮想マシンを一時停止し、移行を **ライブ以外** の移行として完了させます。これにより、仮想マシンのダウンタイムは長くなりますが、ほとんどの場合で、移行が正常に完了するようになります。

阻止

仮想マシンのライブマイグレーションが正常に完了する確率は、以下によって変わってきます。

- 移行中の仮想マシンのワークロード
 - 移行を開始する前に、仮想マシンのゲストオペレーティングシステムで必須でないプロセスを停止またはキャンセルします。
- ホストが移行に使用できるネットワーク帯域幅
 - ライブマイグレーションの最適な結果を得るには、移行に使用するネットワークの帯域幅を、仮想マシンのダーティーページ生成よりも、はるかに広くする必要があります。仮想マシンのダーティーページの生成速度を取得する手順は、[コマンドラインインターフェイスを使用した仮想マシンの移行](#)の前提条件を参照してください。
 - 移行元ホストと移行先ホストの両方に、移行用の専用のネットワークインターフェイスコントローラー (NIC) が必要です。メモリーが1TBを超える仮想マシンのライブマイグレーションの場合、Red Hat では、25 Gb/s 以上の速度を持つ NIC を推奨しています。
 - また、移行の開始時に **--bandwidth** オプションを使用して、ライブマイグレーションに割り当てるネットワーク帯域幅を指定することもできます。非常に大きな仮想マシンを移行するには、デプロイメントに実行可能な帯域幅をできるだけ多く割り当てます。
- ライブマイグレーションのモード
 - デフォルトの **pre-copy** 移行モードでは、メモリーページがダーティーになると、繰り返しメモリーページをコピーします。
 - **Post-copy** 移行は、メモリーページを1回だけコピーします。移行が停止した場合にライブマイグレーションが **post-copy** モードに切り替わるようにするには、移行の開始時に **virsh migrate** を指定した **--postcopy** オプションを使用します。
- デプロイメント用に指定されたダウンタイム
 - 前述のように、**virsh migrate-setmaxdowntime** を使用して移行中にこれを調整できません。

12.10. 仮想マシンの移行で対応しているホスト

仮想マシンの移行が適切に機能し、Red Hat でサポートされるようにするには、移行元ホストと移行先ホストが特定の RHEL バージョンおよびマシンタイプである必要があります。以下の表は、対応している仮想マシンの移行パスを示しています。

表12.2 ライブマイグレーションの互換性

移行の方法	リリースタイプ	将来バージョンの例	サポート状況
前方	マイナーリリース	9.0.1 → 9.1	対応している RHEL 9 システム - マシンタイプ q35
後方	マイナーリリース	9.1 → 9.0.1	対応している RHEL 9 システム - マシンタイプ q35



注記

RHOSP や OpenShift Virtualization など、Red Hat が提供する他の仮想化ソリューションのサポートレベルは異なります。

第13章 スナップショットを使用した仮想マシンの状態の保存と復元

仮想マシン (VM) の現在の状態を保存するには、仮想マシンの **スナップショット** を作成します。その後、スナップショットに戻すことで、仮想マシンを保存した状態に戻すことができます。

仮想マシンのスナップショットには、仮想マシンのディスクイメージが含まれます。実行中の仮想マシンからスナップショット (**ライブスナップショット**とも呼ばれます) を作成すると、そのスナップショットには、実行中のプロセスやアプリケーションを含む仮想マシンのメモリー状態も含まれます。

スナップショットを作成すると、たとえば次のタスクに役立ちます。

- ゲストオペレーティングシステムのクリーンな状態を保存する
- 仮想マシン上で破壊的な影響を与える可能性のある操作を実行する前に復元ポイントを確保する

13.1. 仮想マシンのスナップショットのサポート制限

Red Hat は、お客様が **外部** スナップショットを使用する場合にのみ、RHEL 上の仮想マシン (VM) のスナップショット機能をサポートします。現在、外部スナップショットは、次の要件をすべて満たしている場合にのみ RHEL で作成されます。

- ホストが RHEL 9.4 以降を使用している。
- 仮想マシンがファイルベースのストレージを使用している。
- 次のいずれかの条件の下で仮想マシンのスナップショットを作成する。
 - 仮想マシンがシャットダウンされている。
 - 仮想マシンが実行中の場合は、**--disk-only --quiesce** オプションまたは **--live --memspec** オプションを使用する。

他のほとんどの設定では、**内部** スナップショットが作成されます。これは RHEL 9 では非推奨です。内部スナップショットはお客様のユースケースに適している可能性がありますが、Red Hat は内部スナップショットの完全なテストとサポートを提供していません。



警告

実稼働環境では内部スナップショットを使用しないでください。

スナップショットがサポートされていることを確認するには、スナップショットの XML 設定を表示し、スナップショットの種類とストレージを確認します。

```
# virsh snapshot-dumpxml <vm-name> <snapshot-name>
```

- サポートされているスナップショットの出力例:

```
<domainsnapshot>
```

```

<name>sample-snapshot-name-1</name>
<state>shutoff</state>
<creationTime>1706658764</creationTime>
<memory snapshot='no'/>
<disks>
  <disk name='vda' snapshot='external' type='file'>
    <driver type='qcow2'/>
    <source file='/var/lib/libvirt/images/vm-name.sample-snapshot-name-1'>
  </disk>
</disks>
<domain type='kvm'>
[...]
```

- サポートされていないスナップショットの出力例:

```

<domainsnapshot>
  <name>sample-snapshot-name-2</name>
  <state>running</state>
  <creationTime>1653396424</creationTime>
  <memory snapshot='internal'/>
  <disks>
    <disk name='vda' snapshot='internal'/>
    <disk name='sda' snapshot='no'/>
  </disks>
  <domain type='kvm'>
[...]
```

13.2. コマンドラインインターフェイスを使用した仮想マシンのスナップショットの作成

仮想マシン (VM) の状態をスナップショットに保存するには、**virsh snapshot-create-as** コマンドを使用できます。

前提条件

- ホストが RHEL 9.4 以降を使用している。
- 仮想マシンがファイルベースのストレージを使用している。これが当てはまるかどうかを確認するには、次のコマンドを使用して、**disk** デバイスの **disk type** が **file** と表示されることを確認します。

```

# virsh dumpxml <vm-name> | grep "disk type"
  <disk type='file' device='disk'>
  <disk type='file' device='cdrom'>
```

- 実行中の仮想マシンのメモリーを含む仮想マシンスナップショットを作成する場合は、仮想マシンのメモリーを保存するための十分なディスク領域が必要です。
 - 仮想マシンのメモリーを保存するための推奨最小容量は、仮想マシンに割り当てられた RAM と同じ容量です。たとえば、32 GB の RAM を搭載した仮想マシンのメモリーを保存するには、最大 32 GB のディスク領域が必要です。
 - 仮想マシンの I/O 負荷が大きい場合、大幅な追加ディスク領域が必要になる可能性があります。

- 仮想マシンに VFIO パススルーデバイスが割り当てられている場合、追加のディスク領域が必要になる可能性があります。
- 仮想マシンを一時停止せずにスナップショットを作成すると、追加のディスク領域が必要になる場合があります。



警告

Red Hat では、非常に高いワークロードがかかっている実行中の仮想マシンのメモリーや、VFIO パススルーデバイスを使用している実行中の仮想マシンのメモリーを保存しないことを推奨しています。このような仮想マシンのメモリーを保存すると、ホストディスクがいっぱいになり、システムのデグレードが発生する可能性があります。このような仮想マシンについては、代わりにメモリーなしでスナップショットを作成することを検討してください。

また、すべての VFIO デバイスがメモリーを含むスナップショットの作成に対応しているわけではないことに注意してください。現在、メモリーを含むスナップショットの作成は、接続されている VFIO デバイスが、移行機能が有効な Mellanox VF である場合にのみ正しく機能します。

手順

- 必要なパラメーターを指定して仮想マシンのスナップショットを作成するには、**virsh snapshot-create-as** コマンドを使用します。

```
# virsh snapshot-create-as <vm-name> <snapshot-name> <optional-description>
<additional-parameters>
```

- シャットダウンされた仮想マシンのスナップショットを作成するには、**--disk-only** パラメーターを使用します。たとえば、次のコマンドは、シャットダウンされた **Testguest1** 仮想マシンの現在のディスク状態から **Snapshot1** を作成します。

```
# virsh snapshot-create-as Testguest1 Snapshot1 --disk-only
Domain snapshot Snapshot1 created.
```

- 実行中の仮想マシンのディスク状態をメモリーを除いて保存するスナップショットを作成するには、**--disk-only --quiesce** パラメーターを使用します。たとえば、次のコマンドは、実行中の **Testguest2** 仮想マシンの現在のディスク状態から、**clean system install** という説明を持つ **Snapshot2** を作成します。

```
# virsh snapshot-create-as Testguest2 Snapshot2 "clean system install" --disk-only --
quiesce
Domain snapshot Snapshot2 created.
```

- 実行中の仮想マシンを一時停止して、ディスク状態とメモリーを保存するスナップショットを作成するには、**--memspec** パラメーターを使用します。たとえば、次のコマンドは、**Testguest3** 仮想マシンを一時停止して、仮想マシンの現在のディスクとメモリーの状

態から **Snapshot3** を作成します。仮想マシンのメモリーは、`/var/lib/libvirt/images/saved_memory.img` ファイルに保存されます。スナップショットが完成すると、仮想マシンが自動的に操作を再開します。

```
# virsh snapshot-create-as Testguest3 Snapshot3 --memspec
/var/lib/libvirt/images/saved_memory.img
Domain snapshot Snapshot3 created.
```

スナップショット作成時に仮想マシンを一時停止すると、ダウンタイムが発生します。しかし、特に負荷の高い仮想マシンの場合は、一時停止したほうが、実行中の仮想マシンのライブスナップショットを (`--live` オプションを使用して) 作成するよりも、確実に機能する可能性があります。

- 実行中の仮想マシンのディスク状態とライブメモリーを保存するスナップショットを作成するには、`--live --memspec` パラメーターを使用します。たとえば、次のコマンドは、実行中の **Testguest4** 仮想マシンの現在のディスクとメモリーの状態から **Snapshot4** を作成し、メモリーの状態を `/var/lib/libvirt/images/saved_memory2.img` ファイルに保存します。

```
# virsh snapshot-create-as Testguest4 Snapshot4 --live --memspec
/var/lib/libvirt/images/saved_memory2.img
Domain snapshot Snapshot4 created.
```



警告

仮想マシンのメモリーをスナップショットに保存すると、仮想マシンのゲストオペレーティングシステムで実行中のプロセスの状態が保存されます。ただし、このようなスナップショットに戻したときに、ネットワーク接続の喪失やシステム時間の同期の欠如など、さまざまな要因によりプロセスが失敗する可能性があります。

検証

1. 指定した仮想マシンに関連付けられているスナップショットをリスト表示します。

```
# virsh snapshot-list <Testguest1>

Name                Creation Time          State
-----
Snapshot1           2024-01-30 18:34:58 +0100  shutoff
```

2. スナップショットが **外部** として作成されたことを確認します。

```
# virsh snapshot-dumpxml <Testguest1> <Snapshot1> | grep external

<disk name='vda' snapshot='external' type='file'>
```

このコマンドの出力に `snapshot='external'` が含まれている場合、スナップショットは外部スナップショットであり、Red Hat によって完全にサポートされます。

次のステップ

- [CLI を使用して仮想マシンスナップショットに戻す](#)
- [Web コンソールを使用して仮想マシンスナップショットに戻す](#)

関連情報

- [スナップショットのメタデータに関するアップストリームの libvirt の情報](#)
- [virsh の man ページ](#)

13.3. WEB コンソールを使用した仮想マシンのスナップショットの作成

仮想マシン (VM) の状態をスナップショットに保存するには、RHEL Web コンソールを使用できます。

前提条件

- ホストが RHEL 9.4 以降を使用している。
- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 仮想マシンがファイルベースのストレージを使用している。この条件を満たしていることを確認するには、次の手順を実行します。
 - a. Web コンソールの **Virtual machines** インターフェイスで、スナップショットを作成する仮想マシンをクリックします。
 - b. 管理概要の **Disks** ペインで、リストされているデバイスの **Source** 列を確認します。ソースが表示されているすべてのデバイスで、このソースが **File** である必要があります。

手順

1. Web コンソールの **Virtual machines** インターフェイスで、スナップショットを作成する仮想マシンをクリックします。
仮想マシンの管理概要が開きます。
2. 管理概要の **Snapshots** ペインで、**Create snapshot** ボタンをクリックします。
3. スナップショットの名前を入力し、必要に応じて説明を入力します。
4. **Create** をクリックします。

検証

1. スナップショットの作成が成功したことを確認するには、スナップショットが仮想マシンの **Snapshots** ペインに表示されていることを確認します。
2. スナップショットが **外部** として作成されたことを確認します。これを行うには、ホストのコマンドラインインターフェイスで次のコマンドを使用します。

```
# virsh snapshot-dumpxml <Testguest1> <Snapshot1> | grep external  
  
<disk name='vda' snapshot='external' type='file'>
```

このコマンドの出力に **snapshot='external'** が含まれている場合、スナップショットは外部スナップショットであり、Red Hat によってサポートされます。

次のステップ

- [Web コンソール](#)を使用して仮想マシンスナップショットに戻す
- [コマンドラインインターフェイス](#)を使用して仮想マシンスナップショットに戻す

13.4. コマンドラインインターフェイスを使用して仮想マシンのスナップショットに戻す

仮想マシン (VM) をスナップショットに保存された状態に戻すには、コマンドラインインターフェイス (CLI) を使用できます。

前提条件

- 以前に [Web コンソール](#) または [コマンドラインインターフェイス](#)を使用して作成した仮想マシンのスナップショットが利用可能である。
- **オプション:** 仮想マシンの現在の状態のスナップショットを作成した。現在の状態を保存せずに以前のスナップショットに戻すと、最後のスナップショット以降に仮想マシンで実行された変更が失われます。

手順

- **virsh snapshot-revert** ユーティリティを使用して、仮想マシンの名前と、復元先のスナップショットの名前を指定します。以下に例を示します。

```
# virsh snapshot-revert Testguest2 clean-install
Domain snapshot clean-install reverted
```

検証

- 元に戻した仮想マシンの現在アクティブなスナップショットを表示します。

```
# virsh snapshot-current Testguest2 --name
clean-install
```

13.5. WEB コンソールを使用して仮想マシンのスナップショットに戻す

仮想マシン (VM) をスナップショットに保存された状態に戻すには、RHEL Web コンソールを使用できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 以前に [Web コンソール](#) または [コマンドラインインターフェイス](#)を使用して作成した仮想マシンのスナップショットが利用可能である。

- **オプション:** 仮想マシンの現在の状態のスナップショットを作成した。現在の状態を保存せずに以前のスナップショットに戻すと、最後のスナップショット以降に仮想マシンで実行された変更が失われます。

手順

1. Web コンソールの **Virtual machines** インターフェイスで、状態に戻す仮想マシンをクリックします。
仮想マシンの管理概要が開きます。
2. 管理概要の **Snapshots** ペインで、復元先のスナップショットの横にある **Revert** ボタンをクリックします。
3. 元に戻す操作が完了するまで待ちます。スナップショットのサイズや現在の状態との違いによっては、数分かかる場合があります。

検証

- **Snapshots** ペインで、選択したスナップショットの左側に緑色のチェック記号が表示されていれば、そのスナップショットに正常に戻されています。

13.6. コマンドラインインターフェイスを使用して仮想マシンのスナップショットを削除する

仮想マシン (VM) スナップショットが不要になった場合は、コマンドラインインターフェイスでスナップショットを削除して、そのスナップショットが使用しているディスク領域を解放できます。

前提条件

- **オプション:** 削除するスナップショットの子スナップショットがある。
アクティブなスナップショットがあるときに新しいスナップショットを作成すると、子スナップショットが自動的に作成されます。子を持たないスナップショットを削除すると、親スナップショットから作成された後にそのスナップショットに保存された変更がすべて失われます。

仮想マシン内のスナップショットの親子構造を表示するには、**virsh snapshot-list --tree** コマンドを使用します。次の例では、**Latest-snapshot** が **Redundant-snapshot** の子として表示されています。

```
# virsh snapshot-list --tree <vm-name>

Clean-install-snapshot
|
+- Redundant-snapshot
|
+- Latest-snapshot
```

手順

- スナップショットを削除するには、**virsh snapshot-delete** コマンドを使用します。たとえば、次のコマンドは、**Testquest1** 仮想マシンから **Redundant-snapshot** を削除します。

```
# virsh snapshot-delete Testquest1 Redundant-snapshot
Domain snapshot Redundant-snapshot deleted
```

検証

- 削除したスナップショットがなくなったことを確認するには、該当する仮想マシンの既存のスナップショットとその親子構造を表示します。

```
# virsh snapshot-list --tree <Testguest1>

Clean-install-snapshot
|
+- Latest-snapshot
```

この例では、**Redundant-snapshot** が削除され、**Latest-snapshot** が **Clean-install-snapshot** の子になっています。

13.7. WEB コンソールを使用して仮想マシンのスナップショットを削除する

仮想マシン (VM) スナップショットが不要になった場合は、Web コンソールでスナップショットを削除して、そのスナップショットが使用しているディスク領域を解放できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- **オプション:** 削除するスナップショットの子スナップショットがある。
アクティブなスナップショットがあるときに新しいスナップショットを作成すると、子スナップショットが自動的に作成されます。子を持たないスナップショットを削除すると、親スナップショットから作成された後にそのスナップショットに保存された変更がすべて失われます。

スナップショットに子があるかどうかを確認するには、仮想マシンの Web コンソールの概要にあるスナップショットの **Parent snapshot** 列に **Snapshots** がリストされていることを確認します。

手順

1. Web コンソールの **Virtual machines** インターフェイスで、スナップショットを削除する仮想マシンをクリックします。
仮想マシンの管理概要が開きます。
2. 管理概要の **Snapshots** ペインで、削除するスナップショットの横にある **Delete** ボタンをクリックします。
3. 削除操作が完了するまで待ちます。スナップショットのサイズによっては、数分かかる場合があります。

検証

- スナップショットが **Snapshots** ペインに表示されなくなれば、正常に削除されています。

第14章 仮想デバイスの管理

仮想マシンの機能、特徴、およびパフォーマンスを管理する最も効果的な方法の1つは、**仮想デバイス**を調整することです。

以下のセクションでは、仮想デバイスの **一般的な概要** と、CLI または **Web コンソール** を使用して仮想デバイスを管理する方法について説明します。

14.1. 仮想デバイスの動作

物理マシンと同様、仮想マシンでは、処理能力、メモリー、ストレージ、ネットワーク、グラフィックスなどの機能をシステムに提供する特殊なデバイスが必要になります。物理システムでは通常、これらの目的でハードウェアデバイスを使用します。ただし、仮想マシンはソフトウェア実装として機能するため、代わりにそのようなデバイスのソフトウェアの抽象化を使用する必要があります。これは、**仮想デバイス**と呼ばれています。

基本情報

仮想マシンの作成 時に、仮想マシンに接続されている仮想デバイスを設定でき、既存の仮想マシンでも管理できます。通常、仮想デバイスは、仮想マシンが停止している場合に限り仮想マシンに接続または切断できますが、仮想マシンの実行中に追加または削除できるものもあります。この機能は、デバイスの **ホットプラグ** および **ホットアンプラグ** と呼ばれています。

新しい仮想マシンを作成すると、特に指定しない限り、**libvirt** は、必須の仮想デバイスのデフォルトセットを自動的に作成して設定します。これは、ホストシステムのアーキテクチャーとマシンタイプに基づいており、通常は以下のものが含まれます。

- CPU
- メモリー
- キーボード
- ネットワークインターフェイスコントローラー (NIC)
- さまざまなデバイスコントローラー
- ビデオカード
- サウンドカード

仮想マシンの作成後に仮想デバイスを管理するには、コマンドラインインターフェイス (CLI) を使用します。ただし、仮想ストレージデバイスおよび NIC を管理する場合は、RHEL 9 Web コンソールを使用することもできます。

パフォーマンスまたは柔軟性

デバイスの種類によっては、RHEL 9 が複数の実装に対応し、しばしばパフォーマンスと柔軟性にトレードオフが伴います。

たとえば、仮想ディスクに使用される物理ストレージは、**qcow2**、**raw** などのさまざまな形式のファイルで示され、次のようなさまざまなコントローラーを使用して仮想マシンに提示されます。

- エミュレートされたコントローラー
- **virtio-scsi**

- **virtio-blk**

virtio デバイスは、仮想化を目的として特別に設計されているため、エミュレートされたコントローラーは、**virtio** コントローラーよりも遅くなります。一方、エミュレートされたコントローラーは、**virtio** デバイスに対するドライバーがないオペレーティングシステムを実行するのを可能にします。同様に、**virtio-scsi** は、SCSI コマンドへのより完全な対応を提供しており、仮想マシンにより多くのディスクを割り当てることができるようにします。最後に、**virtio-blk** は、**virtio-scsi** とエミュレートされたコントローラーよりも高いパフォーマンスを提供しますが、ユースケースは範囲がより限定されます。たとえば、**virtio-blk** を使用する場合には、物理ディスクを LUN デバイスとして仮想マシンに割り当ててすることはできません。

仮想デバイスの種類の詳細は、[仮想デバイスの種類](#) を参照してください。

14.2. 仮想デバイスの種類

RHEL 9 の仮想化では、仮想マシン (VM) に接続できるいくつかの異なるタイプの仮想デバイスを提示できます。

エミュレートされたデバイス

エミュレートされたデバイスは、広く使用されている物理デバイスのソフトウェア実装です。物理デバイス用に設計されたドライバーは、エミュレートされたデバイスとも互換性があります。そのため、エミュレートされたデバイスは柔軟性に非常に優れています。

ただし、特定のタイプのハードウェアを正確にエミュレートする必要があるため、エミュレートされたデバイスは、対応する物理デバイス、またはより最適化された仮想デバイスと比較すると、パフォーマンスが大幅に低下する可能性があります。

以下のタイプのエミュレートされたデバイスに対応します。

- 仮想 CPU (vCPU) があり、利用可能な CPU モデルが多数あります。エミュレーションのパフォーマンスへの影響は、ホストの CPU とエミュレートされた vCPU の差異に大きく左右されます。
- PCI バスコントローラーなどのエミュレートされたシステムコンポーネント。
- SATA、SCSI、IDE などのエミュレートされたストレージコントローラー。
- ICH9、ICH6、AC97 などのエミュレートされたサウンドデバイス。
- VGA カードなどのエミュレートされたグラフィックカード。
- rtl8139 などのエミュレートされたネットワークデバイス。

準仮想化デバイス

準仮想化は、仮想デバイスを仮想マシンに公開する高速かつ効率的な方法を提供します。準仮想化デバイスは、仮想マシンで使用するために特別に設計されたインターフェイスを公開するため、デバイスのパフォーマンスが大幅に向上します。RHEL 9 では、**virtio** API を、ハイパーバイザーと仮想マシンとの間のレイヤーとして使用して、仮想マシンに準仮想化デバイスを提供します。このアプローチの欠点は、ゲストオペレーティングシステムで特定のデバイスドライバーが必要になることです。

可能な場合、特に I/O 集約型アプリケーションを実行している場合は、仮想マシンにエミュレートされたデバイスの代わりに準仮想化デバイスを使用することが推奨されます。準仮想化デバイスは、I/O レイテンシーを低減し、I/O スループットを増加させます。場合によっては、ベアメタルのパフォーマンスに非常に近づくことがあります。その他の準仮想化デバイスも、他の方法では利用できない機能を仮想マシンに追加します。

以下のタイプの準仮想化デバイスに対応します。

- 準仮想化ネットワークデバイス (**virtio-net**)
- 準仮想化ストレージコントローラー:
 - **virtio-blk** - ブロックデバイスエミュレーションを提供します。
 - **virtio-scsi** - より完全な SCSI エミュレーションを提供します。
- 準仮想化されたクロック
- 準仮想化されたシリアルデバイス (**virtio-serial**)
- 仮想マシンとそのホスト間でメモリーを動的に分散するために使用されるバルーンデバイス (**virtio-balloon**)。
- 準仮想化された乱数ジェネレーター (**virtio-rng**)

物理的に共有されているデバイス

特定のハードウェアプラットフォームにより、仮想マシンはさまざまなハードウェアデバイスやコンポーネントに直接アクセスできます。このプロセスは、**デバイスの割り当て**として、または **パススルー** として知られています。

この方法で接続すると、物理マシンの場合と同様に、物理デバイスの一部の側面が仮想マシンで直接利用できます。これにより、仮想マシンで使用されるデバイスのパフォーマンスが向上します。ただし、仮想マシンに物理的に接続されているデバイスはホストからは利用できず、移行もできません。

それにもかかわらず、いくつかのデバイスは、複数の仮想マシンで **共有** できます。たとえば、場合によっては、1台の物理デバイスが複数の **仲介デバイス** を提供します。これは、異なる仮想マシンに割り当てることができます。

以下の種類のパススルーデバイスに対応します。

- USB、PCI、および SCSI のパススルー - ゲストソフトウェアで特定の機能が利用できるようにするために、一般的な業界標準のバスを仮想マシンに直接公開します。
- シングルルート I/O 仮想化 (SR-IOV) - PCI Express リソースのハードウェアで強制された分離を可能にする仕様です。これにより、1つの物理 PCI リソースを、複数の仮想 PCI 機能に分割する、安全かつ効率的な作業が可能になります。これは、通常、ネットワークインターフェイスカード (NIC) に使用されます。
- NPIV (N_Port ID virtualization) - 1つの物理ホストバスアダプター (HBA) を、複数の仮想ポートと共有するファイバーチャネル技術です。
- GPU および vGPU - 特定のタイプのグラフィックスまたは計算ワークロード用のアクセラレーター。GPU によっては仮想マシンに直接接続できるものもありますが、一部のタイプでは、基本となる物理ハードウェアを共有する仮想 GPU (vGPU) を作成する機能も提供されます。



注記

これらのタイプの一部のデバイスはサポート対象外であるか、RHEL と互換性がない可能性があります。仮想デバイスのセットアップについてサポートが必要な場合は、Red Hat サポートにお問い合わせください。

14.3. CLI を使用した仮想マシンに接続されたデバイスの管理

仮想マシンの機能を変更するには、コマンドラインインターフェイス (CLI) を使用して、仮想マシンに接続されているデバイスを管理します。

CLI を使用して次のことができます。

- デバイスを接続する
- デバイスを変更する
- デバイスを削除する

14.3.1. 仮想マシンへのデバイスの割り当て

新しい仮想デバイスを割り当てることで、仮想マシンに特定の機能を追加できます。

次の手順では、コマンドラインインターフェイス (CLI) を使用して仮想デバイスを作成し、仮想マシンに接続します。一部のデバイスは、[RHEL Web コンソールを使用](#)して仮想マシンに接続することもできます。

たとえば、仮想マシンに新しい仮想ディスクデバイスを割り当てることで、仮想マシンのストレージ容量を増やすことができます。これは、[メモリーのホットプラグ](#)とも呼ばれます。



警告

仮想マシンからのメモリーデバイスの削除 ([メモリーのホットアンプラグ](#)とも呼ばれる) は、RHEL 9 ではサポートされておらず、Red Hat ではその使用を推奨していません。

前提条件

- 仮想マシンに接続するデバイスに必要なオプションを取得します。特定のデバイスで利用可能なオプションを確認するには、**virt-xml --device=?** コマンドを使用します。以下に例を示します。

```
# virt-xml --network=?
--network options:
[...]
address.unit
boot_order
clearxml
driver_name
[...]
```

手順

1. デバイスを仮想マシンに接続するには、デバイスと必要なオプションの定義を含む **virt-xml --add-device** コマンドを使用します。

- たとえば、次は、`/var/lib/libvirt/images/` ディレクトリーに 20GB の `newdisk` qcow2 ディスクイメージを作成し、仮想マシンの次の起動時にそれを仮想マシンとして、実行中の仮想マシン `testguest` に接続します。

```
# virt-xml testguest --add-device --disk
/var/lib/libvirt/images/newdisk.qcow2,format=qcow2,size=20
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

- 以下は、仮想マシンの稼働時に、ホストでバス 002 のデバイス 004 として、仮想マシン `testguest2` に接続した USB フラッシュドライブを接続します。

```
# virt-xml testguest2 --add-device --update --hostdev 002.004
Device hotplug successful.
Domain 'testguest2' defined successfully.
```

USB を定義するバスとデバイスの組み合わせは、`lsusb` コマンドを使用して取得できません。

検証

デバイスが追加されたことを確認するには、次のいずれかを行います。

- `virsh dumpxml` コマンドを実行し、デバイスの XML 定義が、仮想マシンの XML 設定の `<devices>` セクションに追加されました。たとえば、以下の出力は、仮想マシン `testguest` の設定を表示し、002.004 USB フラッシュディスクドライブが追加されていることを確認します。

```
# virsh dumpxml testguest
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x4146' />
    <product id='0x902e' />
    <address bus='2' device='4' />
  </source>
  <alias name='hostdev0' />
  <address type='usb' bus='0' port='3' />
</hostdev>
[...]
```

- 仮想マシンを実行し、デバイスが存在し、正しく機能しているかどうかをテストします。

関連情報

- `man virt-xml` コマンド

14.3.2. 仮想マシンに接続されているデバイスの変更

接続している仮想デバイスの設定を編集することで、仮想マシンの機能を変更できます。たとえば、仮想マシンのパフォーマンスを最適化する場合は、ホストの CPU に合わせて仮想 CPU モデルを変更できます。

以下の手順は、コマンドラインインターフェイス (CLI) を使用して仮想デバイスを修正する一般的な手順を示しています。ディスクや NIC など、仮想マシンに接続されている一部のディスクは、[RHEL 9 Web コンソール](#) で修正できます。

前提条件

- 仮想マシンに接続するデバイスに必要なオプションを取得します。特定のデバイスで利用可能なオプションを確認するには、**virt-xml --device=?** コマンドを使用します。以下に例を示します。

```
# virt-xml --network=?
--network options:
[...]
address.unit
boot_order
clearxml
driver_name
[...]
```

- (必要に応じて) **virsh dumpxml vm-name** を使用してファイルに出力を送って、仮想マシンの XML 設定のバックアップを作成します。たとえば、以下は、**testguest1** 仮想マシンの設定のバックアップファイル **testguest1.xml** を作成します。

```
# virsh dumpxml testguest1 > testguest1.xml
# cat testguest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testguest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

手順

- デバイスの定義および必要なオプションを追加して、**virt-xml --edit** コマンドを使用します。たとえば、次のようにすると、停止する仮想マシン **testguest** の **<cpu>** 設定を削除し、**host-model** に設定します。

```
# virt-xml testguest --edit --cpu host-model,clearxml=yes
Domain 'testguest' defined successfully.
```

検証

デバイスが変更されたことを確認するには、次のいずれかを行います。

- デバイスが存在し、変更を反映する場合は、仮想マシンを実行してテストします。
- virsh dumpxml** コマンドを使用して、デバイスの XML 定義が、仮想マシンの XML 設定で変更されているかどうかを確認します。たとえば、次の出力は、仮想マシン **testguest** の設定を表示し、CPU モードが **host-model** として設定されていることを確認します。

```
# virsh dumpxml testguest
[...]
<cpu mode='host-model' check='partial'>
```

```
<model fallback='allow'/>
</cpu>
[...]
```

トラブルシューティング

- デバイスを変更すると仮想マシンが起動できなくなる場合は、**virsh define** ユーティリティを使用して、バックアップとして作成しておいた XML 設定ファイルを再読み込みして XML 設定を復元します。

```
# virsh define testguest.xml
```



注記

仮想マシンの XML 設定を変更する場合は、**virsh edit** コマンド (**virsh edit testguest** など) も使用できます。ただし、より詳細な変更にはこの方法を使用しないでください。設定を壊し、仮想マシンの起動を妨げる可能性が高くなります。

関連情報

- **man virt-xml** コマンド

14.3.3. 仮想マシンからのデバイスの削除

仮想デバイスを削除することで、仮想マシンの機能を変更できます。たとえば、仮想マシンから仮想ディスクデバイスが不要になった場合は、削除できます。

次の手順は、コマンドラインインターフェイス (CLI) を使用して、仮想マシンから仮想デバイスを削除する方法を示しています。ディスクや NIC などの一部のデバイスは、[using the RHEL 9 web console](#) 仮想マシンから削除することもできます。

前提条件

- (必要に応じて) **virsh dumpxml vm-name** を使用してファイルに出力を送って、仮想マシンの XML 設定のバックアップを作成します。たとえば、以下は、**testguest1** 仮想マシンの設定のバックアップファイル **testguest1.xml** を作成します。

```
# virsh dumpxml testguest1 > testguest1.xml
# cat testguest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testguest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

手順

1. デバイスの定義を付けて、**virt-xml --remove-device** コマンドを使用します。以下に例を示します。
 - 以下は、シャットダウン後に、稼働中の仮想マシン **testguest** から **vdb** としてマークされているストレージデバイスを削除します。

```
# virt-xml testguest --remove-device --disk target=vdb
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

- 次は、稼働中の稼働マシン `testguest2` からすぐに USB フラッシュドライブデバイスを削除します。

```
# virt-xml testguest2 --remove-device --update --hostdev type=usb
Device hotunplug successful.
Domain 'testguest2' defined successfully.
```

トラブルシューティング

- デバイスを取り外すと仮想マシンが起動できなくなる場合は、`virsh define` ユーティリティーを使用して、バックアップとして作成しておいた XML 設定ファイルを再読み込みして XML 設定を復元します。

```
# virsh define testguest.xml
```

関連情報

- `man virt-xml` コマンド

14.4. WEB コンソールを使用したホストデバイスの管理

仮想マシンの機能を変更するには、Red Hat Enterprise Linux 9 Web コンソールを使用して、仮想マシンに接続されているホストデバイスを管理します。

ホストデバイスは、ホストシステムに接続されている物理デバイスです。要件に基づいて、仮想マシンがこれらのハードウェアデバイスおよびコンポーネントに直接アクセスできるようにすることができます。

Web コンソールを使用して以下を行うことができます。

- [デバイスを表示する](#)
- [デバイスを接続する](#)
- [デバイスを削除する](#)

14.4.1. Web コンソールを使用した仮想マシンに接続されているデバイスの表示

仮想マシンに接続されているデバイスを追加または変更する前に、仮想マシンに接続されているデバイスを表示できます。以下の手順では、Web コンソールを使用してこのようなデバイスを表示する方法を説明します。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。

仮想マシンの詳細情報を含む新しいページが開きます。

2. ホストデバイス セクションまでスクロールします。

Host devices				
Type	Class	Model	Vendor	Source
usb		CHERRY Corded Device	Cherry GmbH	Device 002 Bus 001
usb		Optical Mouse	Lenovo	Device 003 Bus 001
pci	Network controller	Ethernet Connection I219-LM	Intel Corporation	Slot 0000:00:1f6

関連情報

- [仮想デバイスの管理](#)

14.4.2. Web コンソールを使用した仮想マシンへのデバイスの接続

仮想マシン (VM) に特定の機能を追加するには、Web コンソールを使用してホストデバイスを仮想マシンに接続します。



注記

複数のホストデバイスを同時に接続することはできません。一度に接続できるデバイスは1つだけです。

詳細は、[RHEL 9 Known Issues](#) を参照してください。

前提条件

- PCI デバイスを接続している場合は、**hostdev** 要素の **managed** 属性のステータスが、**yes** に設定されていることを確認してください。



注記

PCI デバイスを仮想マシンに接続するときは、**hostdev** 要素の **managed** 属性を省略したり、**no** に設定したりしないでください。設定している場合は、PCI デバイスを仮想マシンに渡すときに、PCI デバイスをホストから自動的に切り離すことができなくなります。また、仮想マシンをオフにしたときに、ホストに自動的に再接続することもできません。

その結果、ホストが応答しなくなったり、予期せずシャットダウンしたりする可能性があります。

managed 属性のステータスは、仮想マシンの XML 設定で確認できます。次の例では、**example-VM-1** 仮想マシンの XML 設定を開きます。

```
# virsh edit example-VM-1
```

- 仮想マシンからの重要なデータのバックアップを作成する。
- **オプション:** 仮想マシンの XML 設定をバックアップします。たとえば、**example-VM-1** 仮想マシンをバックアップするには、次のようにします。

```
# virsh dumpxml example-VM-1 > example-VM-1.xml
```

- Web コンソールの仮想マシンプラグインがシステムにインストールされている。

手順

1. **Virtual Machines** インターフェイスで、ホストデバイスを接続する仮想マシンをクリックします。
新しいページが開き、選択した仮想マシンに関する基本情報を含む **Overview** セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための **Console** セクションが表示されます。
2. **Host devices** までスクロールします。
Host devices セクションには、仮想マシンに接続されているデバイスに関する情報と、デバイスを **追加** または **削除** するためのオプションが表示されます。

Host devices				
Type	Class	Model	Vendor	Source
usb		CHERRY Corded Device	Cherry GmbH	Device 002 Bus 001
usb		Optical Mouse	Lenovo	Device 003 Bus 001
pci	Network controller	Ethernet Connection I219-LM	Intel Corporation	Slot 0000:00:1f6

3. **ホストデバイスの追加** をクリックします。
ホストデバイスの追加 ダイアログが表示されます。

Add host device ✕

Type USB PCI

Device	Product	Vendor	Location
<input type="checkbox"/>	Card Reader	Realtek Semiconductor Corp.	Device 002 Bus 002
<input type="checkbox"/>	3.0 root hub	Linux Foundation	Device 001 Bus 004
<input type="checkbox"/>	Bluetooth wireless interface	Intel Corp.	Device 002 Bus 001
<input type="checkbox"/>	2.0 root hub	Linux Foundation	Device 001 Bus 003
<input type="checkbox"/>	Integrated Camera (1280x720@30)	Chicony Electronics Co., Ltd	Device 003

4. VM に接続するデバイスを選択します。
5. **追加** をクリックします。
選択したデバイスが仮想マシンに接続されます。

検証

- VM を実行し、デバイスが **ホストデバイス** セクションに表示されるかどうかを確認します。

14.4.3. Web コンソールを使用した仮想マシンからのデバイスの削除

リソースを解放するか、仮想マシンの機能を変更するか、その両方を行うには、Web コンソールを使用して仮想マシンを変更し、不要になったホストデバイスを削除します。



警告

デバイスと USB デバイスのバス番号の相関が正しくないことが原因で、接続された USB ホストデバイスを Web コンソールで削除することができない場合があります。

詳細は、[RHEL 9 Known Issues](#) を参照してください。

回避策として、`virsh` ユーティリティーを使用して、仮想マシンの XML 設定から USB デバイスの `<hostdev>` 部分を削除します。次の例では、**example-VM-1** 仮想マシンの XML 設定を開きます。

```
# virsh edit <example-VM-1>
```

前提条件

- [Web コンソールの仮想マシンプラグイン](#)がシステムにインストールされている。
- **オプション:** `virsh dumpxml example-VM-1` を使用してファイルに出力を送信し、仮想マシンの XML 設定のバックアップを作成します。たとえば、以下は、`testquest1` 仮想マシンの設定のバックアップファイル `testquest1.xml` を作成します。

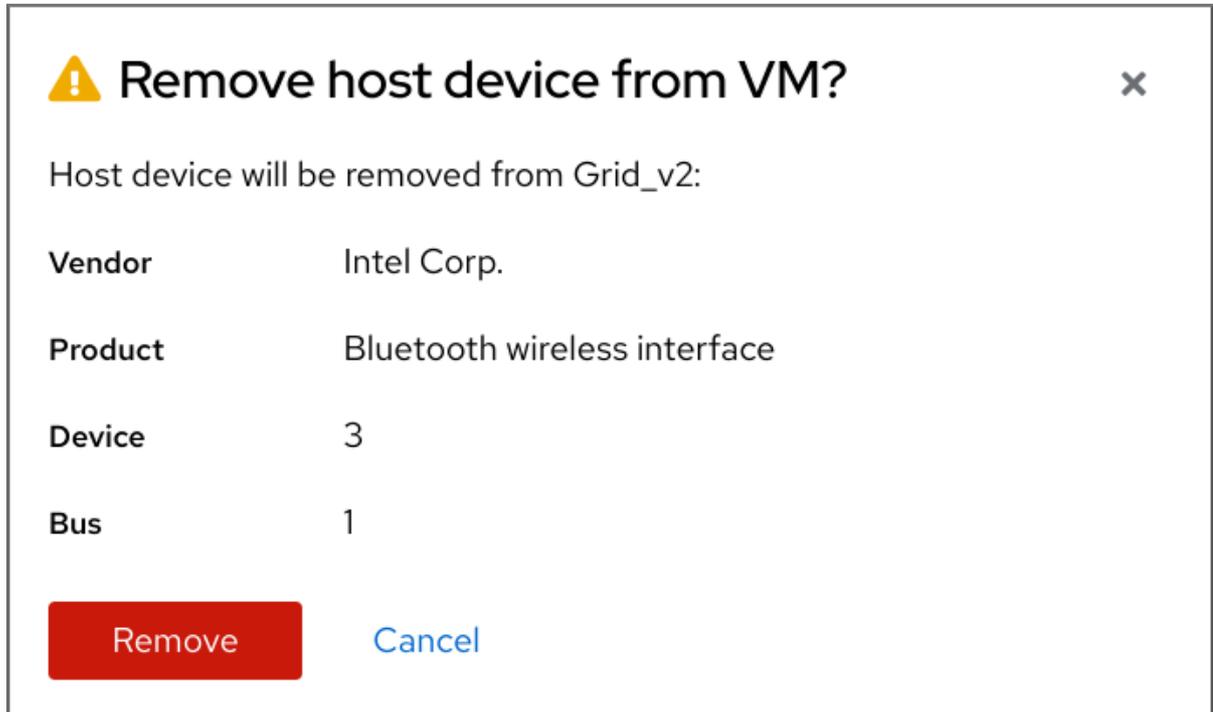
```
# virsh dumpxml testquest1 > testquest1.xml
# cat testquest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testquest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

手順

1. **Virtual Machines** インターフェイスで、ホストデバイスを削除する仮想マシンをクリックします。新しいページが開き、選択した仮想マシンに関する基本情報を含む **Overview** セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための **Console** セクションが表示されます。
2. **Host devices** までスクロールします。**Host devices** セクションには、仮想マシンに接続されているデバイスに関する情報と、デバイスを **追加** または **削除** するためのオプションが表示されます。

Host devices				
Type	Class	Model	Vendor	Source
usb		CHERRY Corded Device	Cherry GmbH	Device 002 Bus 001
usb		Optical Mouse	Lenovo	Device 003 Bus 001
pci	Network controller	Ethernet Connection I219-LM	Intel Corporation	Slot 0000:00:1f6

3. VM から削除するデバイスの横にある **削除** ボタンをクリックします。デバイスの削除確認ダイアログが表示されます。



4. **削除** をクリックします。デバイスが VM から削除されます。

トラブルシューティング

- ホストデバイスを取り外すことで、仮想マシンが起動できなくなる場合は、**virsh define** ユーティリティを使用して、以前にバックアップした XML 設定ファイルを再ロードして XML 設定を復元します。

```
# virsh define testguest1.xml
```

14.5. 仮想 USB デバイスの管理

仮想マシンを使用する場合は、ホストシステムに接続されているフラッシュドライブや Web マッピングなどの USB デバイスにアクセスし、制御できます。このシナリオでは、ホストシステムはデバイスの制御を仮想マシンに渡します。これは USB パススルーとしても知られています。

次のセクションでは、コマンドラインを使用して次のことを行う方法を説明します。

- 仮想マシンに **USB デバイスを接続する**
- 仮想マシンから **USB デバイスを削除する**

14.5.1. 仮想マシンへの USB デバイスの割り当て

USB デバイスを仮想マシンに割り当てるには、仮想マシンの XML 設定ファイルに USB デバイス情報を追加してください。

前提条件

- 仮想マシンにパススルーするデバイスがホストに接続されていることを確認します。

手順

1. 仮想マシンに接続する USB のバスおよびデバイス値を見つけます。
たとえば、次のコマンドは、ホストに接続されている USB デバイスのリストを表示します。この例で使用するデバイスは、デバイス 005 としてバス 001 にアタッチされています。

```
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```

2. **--add-device** 引数を指定して **virt-xml** ユーティリティーを使用します。
たとえば、次のコマンドは、USB フラッシュドライブを **example-VM-1** 仮想マシンに接続します。

```
# virt-xml example-VM-1 --add-device --hostdev 001.005
Domain 'example-VM-1' defined successfully.
```



注記

実行中の仮想マシンに USB デバイスを接続するには、**--update** 引数を直前のコマンドに追加します。

検証

- 仮想マシンを実行し、デバイスが存在し、予想通りに機能しているかどうかをテストします。
- **virsh dumpxml** コマンドを実行し、デバイスの XML 定義が、仮想マシンの XML 設定ファイルの `<devices>` セクションに追加されたかどうかを確認します。

```
# virsh dumpxml example-VM-1
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x0407'>
    <product id='0x6252'>
    <address bus='1' device='5'>
  </source>
  <alias name='hostdev0'>
  <address type='usb' bus='0' port='3'>
</hostdev>
[...]
```

関連情報

- [virt-xml\(1\) man ページ](#)
- [仮想マシンへのデバイスの割り当て](#)

14.5.2. 仮想マシンからの USB デバイスの削除

仮想マシンから USB デバイスを削除するには、仮想マシンの XML 設定から USB デバイス情報を削除してください。

手順

1. 仮想マシンから削除する USB のバスおよびデバイス値を見つけます。
たとえば、次のコマンドは、ホストに接続されている USB デバイスのリストを表示します。この例で使用するデバイスは、デバイス 005 としてバス 001 にアタッチされています。

```
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```

2. **--remove-device** 引数を指定して **virt-xml** ユーティリティーを使用します。
たとえば、次のコマンドは、**example-VM-1** 仮想マシンから、バス 001 でデバイス 005 としてホストに接続されている USB フラッシュドライブを削除します。

```
# virt-xml example-VM-1 --remove-device --hostdev 001.005
Domain 'example-VM-1' defined successfully.
```



注記

実行中の仮想マシンから USB デバイスを削除するには、**--update** 引数を直前のコマンドに追加します。

検証

- 仮想マシンを実行して、デバイスのリストから、このデバイスが削除されたかどうかを確認します。

関連情報

- [virt-xml\(1\) man ページ](#)
- [仮想マシンへのデバイスの割り当て](#)

14.6. 仮想光学ドライブの管理

仮想マシンを使用する場合は、ホストの ISO イメージに保存されている情報にアクセスできます。これを行うには、CD ドライブや DVD ドライブなどの仮想光学ドライブとして、ISO イメージを仮想マシンに割り当てます。

次のセクションでは、コマンドラインを使用して次のことを行う方法を説明します。

- 仮想マシンへの [ドライブと ISO イメージの接続](#)
- 実行中の仮想マシンに [CD-ROM を接続する](#)
- 仮想光学ドライブでの [ISO イメージの置き換え](#)
- 仮想光学ドライブからの [ISO イメージの削除](#)
- 仮想マシンからの [ドライブの削除](#)
- 実行中の仮想マシンから [CD-ROM を削除する](#)

14.6.1. 仮想マシンへの光学ドライブの割り当て

ISO イメージを仮想光学ドライブとして割り当てるには、仮想マシンの XML 設定ファイルを編集し、新しいドライブを追加します。

前提条件

- ISO イメージのパスをホストマシンに保存してコピーしている。

手順

- `--add-device` 引数を指定して `virt-xml` ユーティリティーを使用します。
たとえば、次のコマンドは、`/home/username/Downloads` ディレクトリーに保存されている `example-ISO-name` ISO イメージを `example-VM-name` 仮想マシンに接続します。

```
# virt-xml example-VM-name --add-device --disk /home/username/Downloads/example-ISO-name.iso,device=cdrom
Domain 'example-VM-name' defined successfully.
```

検証

- 仮想マシンを実行し、デバイスが存在し、予想通りに機能しているかどうかをテストします。

関連情報

- `man virt-xml` コマンド
- [仮想マシンへのデバイスの割り当て](#)

14.6.2. Web コンソールを使用して実行中の仮想マシンに CD-ROM を追加する

Web コンソールを使用すると、メディアを指定せずに、実行中の仮想マシン (VM) に CD-ROM を挿入できます。

前提条件

- システムに [Web コンソール仮想マシンプラグイン](#)がインストールされている。

手順

1. 仮想マシンをシャットダウンします。

2. ソースイメージを指定せずに仮想 CD-ROM デバイスを接続します。

```
# virt-xml vmname --add-device --disk target.dev=sda,device=cdrom
```

3. 仮想マシンを実行します。
4. Web コンソールを開き、**仮想マシン** インターフェイスで、CD-ROM を接続する仮想マシンをクリックします。
5. **ディスク** までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。
6. **cdrom** デバイスの **Insert** オプションをクリックします。

Disks						Add disk
Devi...	Used	Capaci...	Bus	Access	Source	
cdrom			scsi	Read-only		Insert Edit ⋮

7. 添付するファイルの **Source** を選択します。
 - **カスタムパス**: ファイルはホストマシン上のカスタムディレクトリーにあります。
 - **既存のものを使用**: ファイルは、作成したストレージプールにあります。
8. **Insert** をクリックします。

検証

- **仮想マシン** インターフェイスの **Disks** セクションにファイルが表示されます。

14.6.3. 仮想光学ドライブでの ISO イメージの置き換え

仮想マシンに仮想光学ドライブとして割り当てられた ISO イメージを置き換えるには、仮想マシンの XML 設定ファイルを編集し、別のイメージを指定します。

前提条件

- ISO イメージをホストマシンに保存している。
- ISO イメージへのパスを知っている。

手順

1. CD-ROM が仮想マシンに接続されているターゲットデバイスを見つけます。この情報は、仮想マシンの XML 設定ファイルにあります。
たとえば、次のコマンドは、**example-VM-name** 仮想マシンの XML 設定ファイルを表示します。ここでは、CD-ROM のターゲットデバイスは **sda** です。

```
# virsh dumpxml example-VM-name
...
<disk>
...
  <source file='$(/home/username/Downloads/example-ISO-name.iso)'/>
```

```
<target dev='sda' bus='sata'/>
...
</disk>
...
```

2. `--edit` 引数を指定して `virt-xml` ユーティリティを使用します。
たとえば、次のコマンドは、ターゲットの `sda` の `example-VM-name` 仮想マシンに接続されている `example-ISO-name` ISO イメージを、`/dev/cdrom` ディレクトリーに保存されている `example-ISO-name-2` ISO イメージに置き換えます。

```
# virt-xml example-VM-name --edit target=sda --disk /dev/cdrom/example-ISO-name-2.iso
Domain 'example-VM-name' defined successfully.
```

検証

- 仮想マシンを実行して、デバイスが置き換えられ、想定どおりに機能しているかどうかを確認します。

関連情報

- `man virt-xml` コマンド

14.6.4. 仮想光学ドライブからの ISO イメージの削除

仮想マシンに接続されている仮想光学ドライブから ISO イメージを削除するには、仮想マシンの XML 設定ファイルを編集します。

手順

1. CD-ROM が仮想マシンに接続されているターゲットデバイスを見つけます。この情報は、仮想マシンの XML 設定ファイルにあります。
たとえば、次のコマンドは、`example-VM-name` 仮想マシンの XML 設定ファイルを表示します。ここでは、CD-ROM のターゲットデバイスは `sda` です。

```
# virsh dumpxml example-VM-name
...
<disk>
...
<source file='$(/home/username/Downloads/example-ISO-name.iso)'/>
<target dev='sda' bus='sata'/>
...
</disk>
...
```

2. `--edit` 引数を指定して `virt-xml` ユーティリティを使用します。
たとえば、次のコマンドは、`example-VM-name` 仮想マシンに接続されている CD ドライブから `example-ISO-name` ISO イメージを削除します。

```
# virt-xml example-VM-name --edit target=sda --disk path=
Domain 'example-VM-name' defined successfully.
```

検証

- 仮想マシンを実行し、イメージが使用できなくなっていることを確認します。

関連情報

- `man virt-xml` コマンド

14.6.5. 仮想マシンからの光学ドライブの削除

仮想マシンに接続されている光学ドライブを削除するには、仮想マシンの XML 設定ファイルを編集します。

手順

1. CD-ROM が仮想マシンに接続されているターゲットデバイスを見つけます。この情報は、仮想マシンの XML 設定ファイルにあります。
たとえば、次のコマンドは、**example-VM-name** 仮想マシンの XML 設定ファイルを表示します。ここでは、CD-ROM のターゲットデバイスは **sda** です。

```
# virsh dumpxml example-VM-name
...
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. **--remove-device** 引数を指定して **virt-xml** ユーティリティーを使用します。
たとえば、次のコマンドは、ターゲット **sda** として接続された光学ドライブを、仮想マシン **example-VM-name** から削除します。

```
# virt-xml example-VM-name --remove-device --disk target=sda
Domain 'example-VM-name' defined successfully.
```

検証

- デバイスが仮想マシンの XML 設定ファイルにリスト表示されていないことを確認します。

関連情報

- `man virt-xml` コマンド

14.6.6. Web コンソールを使用した実行中の仮想マシンからの CD-ROM の削除

Web コンソールを使用して、実行中の仮想マシン (VM) から CD-ROM デバイスを取り出すことができます。

前提条件

- システムに [Web コンソール仮想マシンプラグイン](#)がインストールされている。

手順

1. **仮想マシン** インターフェイスで、CD-ROM を削除する仮想マシンをクリックします。
2. **ディスク** までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Additional	
Device	Used	Capacity	Bus	Access	Source			
cdrom			sata	Read-only	File	/home/test/	Format	raw

⋮

3. **CDROM** デバイスの **Eject** オプションをクリックします。
Eject media from VM? ダイアログボックスが開きます。
4. **Eject** をクリックします。

検証

- **仮想マシン** インターフェイスでは、添付ファイルが **Disks** セクションに表示されなくなりました。

14.7. SR-IOV デバイスの管理

エミュレートされた仮想デバイスは、多くの場合、ハードウェアネットワークデバイスよりも多くの CPU およびメモリーを使用します。これにより、仮想マシンのパフォーマンスを制限できます。ただし、仮想化ホストのデバイスが SR-IOV (Single Root I/O Virtualization) に対応する場合は、この機能を使用してデバイスのパフォーマンスを向上し、仮想マシンの全体的なパフォーマンスを向上させることができます。

14.7.1. SR-IOV とは

SR-IOV (Single-root I/O virtualization) は、1つの PCIe (PCI Express) デバイスが、ホストに、複数の個別の PCI デバイス (**仮想機能** (VF) と呼ばれます) をホストシステムに表示できるようにする仕様です。このデバイスはそれぞれ以下ようになります。

- 元の PCIe デバイスと同一または同様のサービスを提供できます。
- ホストの PCI バス上にある別のアドレスに表示されます。
- VFIO の割り当てを使用して、別の仮想マシンに割り当てることができます。

たとえば、1つの SR-IOV 対応ネットワークデバイスが、VF を複数の仮想マシンに提示できます。すべての VF は同じ物理カード、同じネットワーク接続、同じネットワークケーブルを使用しますが、各仮想マシンは直接そのハードウェアネットワークデバイスを制御し、ホストのリソースは使用しません。

SR-IOV の仕組み

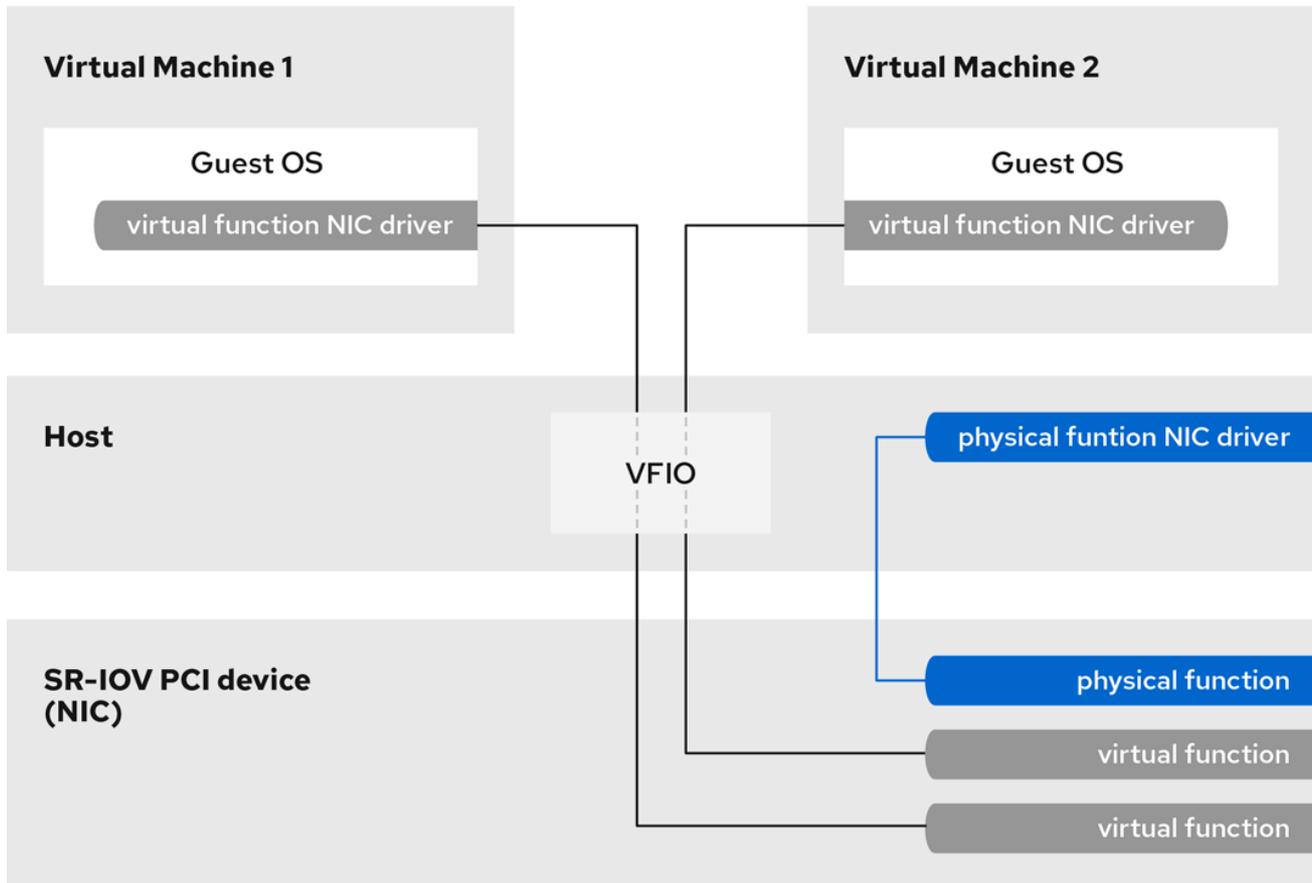
SR-IOV 機能は、以下の PCIe 機能の導入により可能になりました。

- **Physical Function (PF)** - デバイス (ネットワークなど) の機能をホストに提供しますが、一連の VF を作成して管理することもできる PCIe 機能。SR-IOV 対応の各デバイスには、1つ以上の PF があります。
- **Virtual Function (VF)** - 独立したデバイスとして動作する軽量の PCIe 機能。各 VF は PF から派生します。デバイスが持つことができる VF の最大数は、デバイスのハードウェアによって

異なります。各 VF は、一度に1台の仮想マシンにのみ割り当てることができますが、1台の仮想マシンには複数の VF を割り当てることができます。

仮想マシンは、VF を仮想デバイスとして認識します。たとえば、SR-IOV ネットワークデバイスによって作成された VF は、物理ネットワークカードがホストシステムに表示されるのと同じように、割り当てられた仮想マシンへのネットワークカードとして表示されます。

図14.1 SR-IOV アーキテクチャー



メリット

エミュレートされたデバイスではなく SR-IOV VF を使用する主な利点は以下のとおりです。

- パフォーマンスが向上する
- ホストの CPU およびメモリーリソースの使用が減少する

たとえば、vNIC として仮想マシンに接続する VF は、物理 NIC とほぼ同じレベルで実行され、準仮想化またはエミュレートされた NIC よりもはるかに適しています。特に、複数の VF を1台のホスト上で同時に使用する場合に、パフォーマンス上のメリットは重要です。

デメリット

- PF の設定を変更する場合は、最初に PF により公開される VF の数をゼロに変更する必要があります。したがって、このような VF が提供するデバイスを、デバイスが割り当てられている仮想マシンから削除する必要があります。
- SR-IOV VF など、VFIO が割り当てられたデバイスが接続された仮想マシンは、別のホストに移行することができません。場合によっては、割り当てられたデバイスをエミュレートされたデバイスとペアにすることにより、この制限を回避できます。たとえば、割り当てられたネッ

トワーク VF をエミュレートされた vNIC に [ボンディング](#) を行い、移行前に VF を削除できません。

- さらに、VFIO が割り当てたデバイスには仮想マシンのメモリの固定 (ピンング) が必要になるため、仮想マシンのメモリ消費が増加し、仮想マシンのメモリバルーンが使用できなくなります。

関連情報

- [SR-IOV 割り当てに対応しているデバイス](#)
- [IBM Z でのパススルー PCI デバイスの設定](#)

14.7.2. SR-IOV ネットワークデバイスの仮想マシンへの割り当て

Intel ホストまたは AMD ホストの仮想マシンに SR-IOV ネットワークデバイスを割り当てるには、VF (Virtual Function) をホストの SR-IOV 対応ネットワークインターフェイスから作成し、VF をデバイスとして、指定された仮想マシンに割り当てます。詳細は、次の手順を参照してください。

前提条件

- ホストの CPU およびファームウェアは、IOMMU (I/O Memory Management Unit) に対応している。
 - Intel CPU を使用している場合は、Intel VT-d (Virtualization Technology for Directed I/O) に対応する必要があります。
 - AMD CPU を使用している場合は、AMD-Vi 機能に対応する必要があります。
- ホストシステムが、アクセス制御サービス (ACS) を使用して PCIe トポロジーの DMA (Direct Memory Access) 分離を提供している。この点をシステムベンダーに確認してください。詳細は、[SR-IOV 実装に関するハードウェアの考慮事項](#) を参照してください。
- 物理ネットワークデバイスが SR-IOV をサポートしている。システムのネットワークデバイスが SR-IOV に対応しているかどうかを確認するには、**lspci -v** コマンドを使用して、出力で **Single Root I/O Virtualization (SR-IOV)** を探します。

```
# lspci -v
[...]
02:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
Subsystem: Intel Corporation Gigabit ET Dual Port Server Adapter
Flags: bus master, fast devsel, latency 0, IRQ 16, NUMA node 0
Memory at fcb00000 (32-bit, non-prefetchable) [size=128K]
[...]
Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
Kernel driver in use: igb
Kernel modules: igb
[...]
```

- VF の作成に使用するホストのネットワークインターフェイスが実行中である。たとえば、**eth1** インターフェイスをアクティブにして、実行していることを確認するには、次のコマンドを実行します。

```
# ip link set eth1 up
```

```
# ip link show eth1
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT qlen 1000
    link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

- SR-IOV デバイス割り当てを有効にするには、ホスト BIOS およびカーネルで IOMMU 機能を有効にする必要があります。これを行うには、以下を行います。

- Intel ホストで VT-d を有効にします。

- i. **intel_iommu=on** および **iommu=pt** パラメーターを使用して GRUB 設定を再生成します。

```
# grubby --args="intel_iommu=on iommu=pt" --update-kernel=ALL
```

- ii. ホストを再起動します。

- AMD ホストで、AMD-Vi を有効にします。

- i. **iommu=pt** パラメーターで GRUB 設定を再生成します。

```
# grubby --args="iommu=pt" --update-kernel=ALL
```

- ii. ホストを再起動します。

手順

1. (必要に応じて) ネットワークデバイスが使用できる VF の最大数を確認します。これを実行するには、次のコマンドを使用して、**eth1** を SR-IOV 互換のネットワークデバイスに置き換えます。

```
# cat /sys/class/net/eth1/device/sriov_totalvfs
7
```

2. 次のコマンドを実行して、Virtual Function (VF) を作成します。

```
# echo VF-number > /sys/class/net/network-interface/device/sriov_numvfs
```

上記コマンドでは、以下ようになります。

- **VF-number** には、PF に作成する VF の数を入力します。
- **network-interface** は、VF が作成されるネットワークインターフェイスの名前に置き換えます。

以下の例では、eth1 ネットワークインターフェイスから 2 つの VF を作成します。

```
# echo 2 > /sys/class/net/eth1/device/sriov_numvfs
```

3. VF が追加されたことを確認します。

■

```
# lspci | grep Ethernet
82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev
01)
82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev
01)
```

4. VF の作成に使用したネットワークインターフェイス用の udev ルールを作成して、作成した VF を永続化します。たとえば、**eth1** インターフェイスの場合は、**/etc/udev/rules.d/eth1.rules** ファイルを作成し、以下の行を追加します。

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

これにより、ホストの起動時に **ixgbe** ドライバーを使用する 2 つの VF が **eth1** インターフェイスで自動的に利用できるようになります。永続的な SR-IOV デバイスが必要ない場合は、この手順を省略します。



警告

現在、Broadcom NetXtreme II BCM57810 アダプターで VF を永続化しようとする、上記の設定が正しく機能しません。また、このアダプターに基づく VF を Windows 仮想マシンに接続することは、現在信頼性がありません。

5. 新しく追加された VF インターフェイスデバイスの 1 つを実行中の仮想マシンにホットプラグします。

```
# virsh attach-interface testquest1 hostdev 0000:82:10.0 --managed --live --config
```

検証

- この手順が成功すると、ゲストオペレーティングシステムが新しいネットワークインターフェイスカードを検出します。

14.7.3. SR-IOV 割り当てに対応しているデバイス

すべてのデバイスを SR-IOV に使用できるわけではありません。以下のデバイスは、RHEL 9 の SR-IOV との互換性がテストおよび検証されています。

ネットワークデバイス

- Intel 82599ES 10 Gigabit Ethernet Controller - **ixgbe** ドライバーを使用します。
- Intel Ethernet Controller XL710 Series - **i40e** ドライバーを使用します。

- Intel Ethernet Network Adapter XXV710 - **i40e** ドライバーを使用します。
- Intel 82576 Gigabit Ethernet Controller - **igb** ドライバーを使用します。
- Broadcom NetXtreme II BCM57810 - **bnx2x** ドライバーを使用します。
- Ethernet Controller E810-C for QSFP - **ice** ドライバーを使用します。
- SFC9220 10/40G Ethernet Controller - **sfc** ドライバーを使用します。
- FastLinQ QL41000 Series 10/25/40/50GbE Controller - **qed** ドライバーを使用します。
- Mellanox ConnectX-5 Ethernet Adapter Cards
- Mellanox MT2892 Family [ConnectX-6 Dx]

14.8. IBM Z の仮想マシンへの DASD デバイスの割り当て

vfio-ccw 機能を使用すると、直接アクセスストレージデバイス (DASD) を仲介デバイスとして IBM Z ホスト上の仮想マシンに割り当てることができます。これにより、たとえば仮想マシンは z/OS データセットにアクセスできるか、割り当てられた DASD を z/OS マシンに提供できるようになります。

前提条件

- FICON プロトコルでサポートされる IBM Z ハードウェアアーキテクチャーを備えたシステムがある。
- Linux オペレーティングシステムのターゲット仮想マシンがある。
- **driverctl** パッケージがインストールされている。

```
# dnf install driverctl
```

- ホストに必要な **vfio** カーネルモジュールがロードされている。

```
# lsmod | grep vfio
```

このコマンドの出力には、以下のモジュールが含まれている必要があります。

- **vfio_ccw**
 - **vfio_mdev**
 - **vfio_iommu_type1**
- 仮想マシンによる排他的使用のために予備の DASD デバイスがあり、そのデバイスの識別子を把握している。
次の手順では、例として **0.0.002c** を使用しています。コマンドを実行する場合は、**0.0.002c** を DASD デバイスの ID に置き換えます。

手順

1. DASD デバイスのサブチャネル識別子を取得します。

```
# lscss -d 0.0.002c
```

```
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
-----
0.0.002c 0.0.29a8 3390/0c 3990/e9 yes f0 f0 ff 02111221 00000000
```

この例では、サブチャンネル ID は **0.0.29a8** として検出されます。以下のコマンドでは、**0.0.29a8** を、検出されたデバイスのサブチャンネル ID に置き換えます。

2. 前の手順の **lscss** コマンドでヘッダー出力のみが表示され、デバイスインフォメーションが表示されない場合は、以下の手順を実行します。

- a. **cio_ignore** リストからデバイスを削除します。

```
# cio_ignore -r 0.0.002c
```

- b. ゲスト OS で、仮想マシンの [edit the kernel command line](#) を編集し、! マークを使用して、**cio_ignore=** で始まる行にデバイス識別子を追加します (まだ存在しない場合)。

```
cio_ignore=all,!condev,!0.0.002c
```

- c. ホストで手順 1 を繰り返し、サブチャンネル識別子を取得します。

3. サブチャンネルは **vfio_ccw** パススルードライバーにバインドします。

```
# driverctl -b css set-override 0.0.29a8 vfio_ccw
```



注記

これにより、**0.0.29a8** サブチャンネルが **vfio_ccw** に永続的にバインドされます。つまり、DASD はホストコンピューターでは使用できなくなります。ホストでデバイスを使用する必要がある場合は、まず 'vfio_ccw' への自動バインディングを削除し、サブチャンネルをデフォルトドライバーに再バインドする必要があります。

```
# driverctl -b css unset-override 0.0.29a8
```

4. DASD 仲介デバイスを定義して起動します。

```
# cat nodedev.xml
<device>
  <parent>css_0_0_29a8</parent>
  <capability type="mdev">
    <type id="vfio_ccw-io"/>
  </capability>
</device>

# virsh nodedev-define nodedev.xml
Node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8' defined from
'nodedev.xml'

# virsh nodedev-start mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Device mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8 started
```

5. 実行中の場合は、仮想マシンをシャットダウンします。

6. 以前に定義したデバイスの UUID を表示し、次の手順のために保存します。

```
# virsh nodedev-dumpxml mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8

<device>
  <name>mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8</name>
  <parent>css_0_0_29a8</parent>
  <capability type='mdev'>
    <type id='vfio_ccw-io'>
      <uuid>30820a6f-b1a5-4503-91ca-0c10ba12345a</uuid>
      <iommuGroup number='0'>
        <attr name='assign_adapter' value='0x02'>
          <attr name='assign_domain' value='0x002b'>
            </capability>
          </device>
        </device>
      </device>
    </device>
  </device>
</device>
```

7. 仲介デバイスを仮想マシンに接続します。これを行うには、**virsh edit** ユーティリティを使用して仮想マシンの XML 設定を編集し、以下のセクションを XML に追加します。**uuid** の値は、前の手順で取得した UUID に置き換えます。

```
<hostdev mode='subsystem' type='mdev' model='vfio-ccw'>
  <source>
    <address uuid="30820a6f-b1a5-4503-91ca-0c10ba12345a"/>
  </source>
</hostdev>
```

8. **オプション**: ホストの起動時に自動的に開始するように仲介デバイスを設定します。

```
# virsh nodedev-autostart mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
```

検証

1. 仲介デバイスが正しく設定されていることを確認します。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Name:      mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Parent:    css_0_0_0121
Active:    yes
Persistent: yes
Autostart: yes
```

2. **libvirt** が仲介 DASD デバイスに割り当てた識別子を取得します。これを行うには、仮想マシンの XML 設定を表示して、**vfio-ccw** デバイスを見つけます。

```
# virsh dumpxml vm-name

<domain>
[...]
```

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ccw'>
  <source>
    <address uuid='10620d2f-ed4d-437b-8aff-beda461541f9'>
  </source>
  <alias name='hostdev0'>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0009'>
```

```
</hostdev>
[...]
</domain>
```

この例では、デバイスに割り当てられた識別子は **0.0.0009** です。

- 仮想マシンを起動し、ゲスト OS にログインします。
- ゲスト OS で、DASD デバイスがリストされていることを確認します。以下に例を示します。

```
# lscss | grep 0.0.0009
0.0.0009 0.0.0007 3390/0c 3990/e9   f0 f0 ff 12212231 00000000
```

- ゲスト OS で、デバイスをオンラインに設定します。以下に例を示します。

```
# chccwdev -e 0.0009
Setting device 0.0.0009 online
Done
```

関連情報

- [cio_ignore](#) に関する IBM のドキュメント
- [ランタイム時のカーネルパラメーターの設定](#)

14.9. WEB コンソールを使用した仮想マシンへのウォッチドッグデバイスの接続

仮想マシン (VM) が応答を停止したときに指定されたアクションを強制的に実行するには、仮想ウォッチドッグデバイスを仮想マシンに接続します。

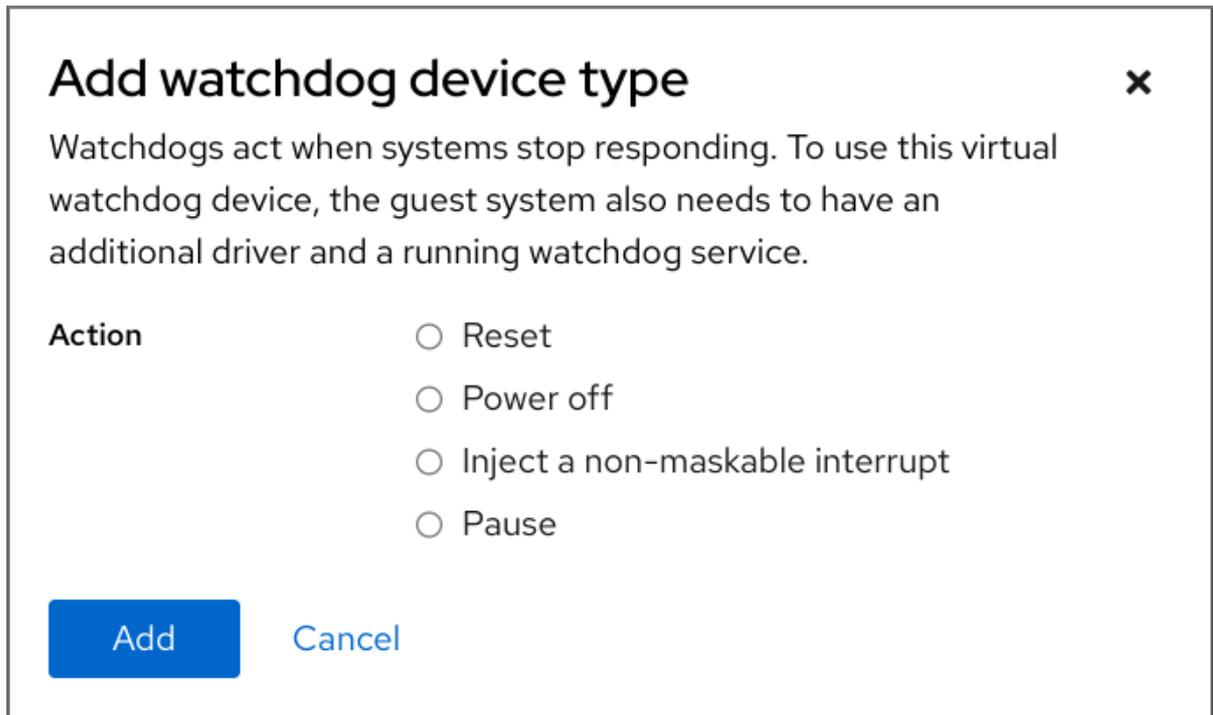
前提条件

- システムに Web コンソール仮想マシンプラグインがインストールされている。詳細は、「[仮想マシンを管理するために Web コンソールを設定](#)」を参照すること。

手順

- コマンドラインインターフェイスで、ウォッチドッグサービスをインストールします。
`# dnf install watchdog`
- 仮想マシンをシャットダウンします。
- ウォッチドッグサービスを仮想マシンに追加します。
`# virt-xml vmname --add-device --watchdog action=reset --update`
- 仮想マシンを実行します。
- Web コンソールを開き、Web コンソールの **仮想マシン** インターフェイスで、ウォッチドッグデバイスを追加する仮想マシンをクリックします。
- 概要ペインの **Watchdog** フィールドの横にある **add** をクリックします。
Add watchdog device type ダイアログが表示されます。

7. 仮想マシンが応答を停止した場合にウォッチドッグデバイスが実行するアクションを選択します。



8. **Add** をクリックします。

検証

- 選択したアクションは、Overview ペインの **Watchdog** フィールドの横に表示されます。

14.10. IBM Z の仮想マシンへの PCI デバイスの接続

vfio-pci デバイスドライバーを使用すると、パススルーモードで PCI デバイスを IBM Z ホストの仮想マシンに割り当てることができます。たとえば、これにより、仮想マシンはデータベースの処理に NVMe フラッシュディスクを使用できるようになります。

前提条件

- IBM Z ハードウェアアーキテクチャを使用するホストシステムがある。
- Linux オペレーティングシステムのターゲット仮想マシンがある。
- ホストに必要な **vfio** カーネルモジュールがロードされている。

```
# lsmod | grep vfio
```

このコマンドの出力には、以下のモジュールが含まれている必要があります。

- **vfio_pci**
- **vfio_pci_core**
- **vfio_iommu_type1**

手順

1. 使用するデバイスの PCI アドレス識別子を取得します。

```
# lspci -nkD
0000:00:00.0 0000: 1014:04ed
Kernel driver in use: ism
Kernel modules: ism
0001:00:00.0 0000: 1014:04ed
Kernel driver in use: ism
Kernel modules: ism
0002:00:00.0 0200: 15b3:1016
Subsystem: 15b3:0062
Kernel driver in use: mlx5_core
Kernel modules: mlx5_core
0003:00:00.0 0200: 15b3:1016
Subsystem: 15b3:0062
Kernel driver in use: mlx5_core
Kernel modules: mlx5_core
```

2. PCI デバイスを接続する仮想マシンの XML 設定を開きます。

```
# virsh edit vm-name
```

3. 以下の **<hostdev>** 設定を XML ファイルの **<devices>** セクションに追加します。
address 行の値を、デバイスの PCI アドレスに置き換えます。たとえば、デバイスアドレスが **0003:00:00.0** の場合は、以下の設定を使用します。

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <driver name="vfio"/>
  <source>
    <address domain="0x0003" bus="0x00" slot="0x00" function="0x0"/>
  </source>
  <address type="pci"/>
</hostdev>
```

4. **オプション:** ゲストオペレーティングシステムが PCI デバイスを検出する方法を変更するには、**<zpci>** サブ要素を **<address>** 要素に追加することもできます。**<zpci>** 行では、**uid** 値と **fid** 値を調整できます。これにより、ゲストオペレーティングシステムのデバイスの PCI アドレスと機能 ID が変更されます。

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <driver name="vfio"/>
  <source>
    <address domain="0x0003" bus="0x00" slot="0x00" function="0x0"/>
  </source>
  <address type="pci">
    <zpci uid="0x0008" fid="0x001807"/>
  </address>
</hostdev>
```

この例では、以下が適用されます。

- **uid="0x0008"** は、仮想マシンのデバイスのドメイン PCI アドレスを **0008:00:00.0** に設定します。

- `fid="0x001807"` は、デバイスのスロット値を `0x001807` に設定します。これにより、仮想マシンのファイルシステムのデバイス設定が `/sys/bus/pci/slots/00001087/address` に保存されます。
これらの値が指定されていない場合は、`libvirt` がこれらの値を自動的に設定します。

5. XML 設定を保存します。
6. 仮想マシンが実行中の場合はシャットダウンします。

```
# virsh shutdown vm-name
```

検証

1. 仮想マシンを起動し、ゲストオペレーティングシステムにログインします。
2. ゲストオペレーティングシステムで、PCI デバイスがリストされていることを確認します。
たとえば、デバイスアドレスが `0003:00:00.0` の場合は、次のコマンドを使用します。

```
# lspci -nkD | grep 0003:00:00.0  
  
0003:00:00.0 8086:9a09 (rev 01)
```

第15章 仮想マシン用のストレージの管理

仮想マシンは、物理マシンと同様に、データ、プログラム、およびシステムファイル用にストレージを必要とします。仮想マシン管理者は、物理ストレージまたはネットワークベースのストレージを仮想マシンに仮想ストレージとして割り当てることができます。また、基本となるハードウェアに関係なく、ストレージを仮想マシンに表示する方法を変更することもできます。

次のセクションでは、仮想マシンのストレージの種類、その機能、および CLI または Web コンソールを使用してそれらを管理する方法を説明します。

15.1. 仮想マシンのストレージの概要

仮想マシンのストレージを初めて使用するユーザー、またはその仕組みがよくわからないユーザー向けに、次のセクションでは仮想マシンストレージのさまざまなコンポーネントの概要、その機能、管理の基本、Red Hat が提供するサポートされるソリューションを説明します。

以下の情報が記載されています。

- [ストレージプール](#)
- [ストレージボリューム](#)
- [libvirt を使用したストレージの管理](#)
- [仮想マシンのストレージの概要](#)
- [対応しているストレージプールのタイプと、対応していないストレージプールのタイプ](#)

15.1.1. ストレージプールの概要

ストレージプールは、仮想マシンにストレージを提供するために、**libvirt** が管理するファイル、ディレクトリー、またはストレージデバイスです。ストレージプールは、ストレージボリュームに分割できます。ストレージボリュームは、仮想マシンイメージを保存するか、追加のストレージとして仮想マシンに割り当てられます。

さらに、複数の仮想マシンが同じストレージプールを共有できるため、ストレージリソースの割り当てが改善されます。

- ストレージプールは永続的または一時的なものにできます。
 - 永続ストレージプールは、ホストマシンのシステムを再起動しても維持します。この **virsh pool-define** を使用して、永続ストレージプールを作成できます。
 - 一時的なストレージプールは、ホストが再起動すると削除されます。**virsh pool-create** コマンドを使用すると、一時的なストレージプールを作成できます。

ストレージプールのストレージタイプ

ストレージプールは、ローカルまたはネットワークベース (共有) にできます。

- **ローカルストレージのプール**
ローカルストレージプールは、ホストサーバーに直接割り当てることができます。これには、ローカルデバイスのローカルディレクトリー、直接接続したディスク、物理パーティション、および論理ボリューム管理 (LVM) ボリュームグループが含まれます。

ローカルストレージプールは、移行を必要としない、または仮想マシンが多数存在する、開発、テスト、および小規模なデプロイメントに役立ちます。

- **ネットワーク (共有) ストレージプール**

ネットワークストレージプールには、標準プロトコルを使用してネットワーク経由で共有されるストレージデバイスが含まれます。

15.1.2. ストレージボリュームの概要

ストレージプールは、**ストレージボリューム** に分類されます。ストレージボリュームは、**libvirt** が処理する物理パーティション、LVM 論理ボリューム、ファイルベースのディスクイメージ、その他のストレージタイプの抽象化です。ストレージボリュームは、基盤となるハードウェアに関係なく、ローカルのストレージデバイス (ディスクなど) として仮想マシンに表示されます。

ホストマシンでは、ストレージボリュームは、その名前と、そこから派生するストレージプールの識別子で参照されます。**virsh** コマンドラインでは、**--pool storage_pool volume_name** の形式を取ります。

たとえば、**guest_images** プールにある **firstimage** という名前のボリュー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。

```
# virsh vol-info --pool guest_images firstimage
Name:      firstimage
Type:      block
Capacity:  20.00 GB
Allocation: 20.00 GB
```

15.1.3. libvirt を使用したストレージ管理

libvirt リモートプロトコルを使用して、仮想マシンストレージのあらゆる側面を管理できます。これらの操作は、リモートホストで実行することもできます。したがって、RHEL Web コンソールなどの **libvirt** を使用する管理アプリケーションを使用して、仮想マシンのストレージを設定するために必要なすべてのタスクを実行できます。

libvirt の API を使用すると、ストレージプールのボリュームリストを照会したり、そのストレージプールの容量、割り当て、利用可能なストレージに関する情報を取得したりできます。それに対応するストレージプールの場合、**libvirt** の API を使用して、ストレージボリュームを作成、クローン作成、サイズ変更、および削除することもできます。また、**libvirt** API を使用してデータをストレージボリュームにアップロードしたり、ストレージボリュームからデータをダウンロードしたり、ストレージボリュームのデータを消去したりできます。

15.1.4. ストレージ管理の概要

ストレージの管理で利用可能なオプションを説明するため、以下の例では、**mount -t nfs nfs.example.com:/path/to/share /path/to/data** を使用するサンプルの NFS サーバーを説明します。

ストレージ管理者は、以下を実行できます。

- 仮想ホストに NFS ストレージプールを定義し、エクスポートするサーバーパスと、クライアントのターゲットパスを記述できます。その結果、**libvirt** は、**libvirt** の起動時に自動的に、または **libvirt** の実行中に必要に応じてストレージをマウントできます。
- ストレージプールとストレージボリュームは、名前だけで仮想マシンに追加するだけです。ターゲットパスをボリュームに追加する必要はありません。そのため、ターゲットのクライアントパスが変更しても、仮想マシンには影響を及ぼしません。

- ストレージプールを自動起動するように設定できます。これを行うと、**libvirt** は、**libvirt** の起動時に指定したディレクトリーに、NFS 共有ディスクを自動的にマウントします。**libvirt** は、コマンド `mount nfs.example.com:/path/to/share /vmdata` と同様に、指定したディレクトリーに共有をマウントします。
- **libvirt** の API を使用して、ストレージボリュームパスをクエリーできます。このようなストレージボリュームは、基本的には NFS 共有ディスクにあるファイルです。その後、これらのパスを、仮想マシンのブロックデバイスのソースストレージを説明する仮想マシンの XML 定義のセクションにコピーできます。
- NFS の場合は、**libvirt** の API を使用するアプリケーションを使用して、ストレージプール (NFS 共有内のファイル) にあるストレージボリュームを、プールのサイズ (共有のストレージ容量) の上限まで作成および削除できます。
すべてのストレージプールタイプがボリュームの作成および削除に対応しているわけではないことに注意してください。
- ストレージプールは、不要になったときに停止できます。ストレージプールを停止する (**pool-destroy**) と、開始操作が取り消されます。この場合は、NFS 共有のマウントが解除されます。コマンドの名前が記載されているにも関わらず、共有上のデータは `destroy` 操作で修正されません。詳細は、**man virsh** を参照してください。

15.15. 対応しているストレージプールのタイプと、対応していないストレージプールのタイプ

対応しているストレージプールの種類

以下は、RHEL で対応しているストレージプールタイプのリストです。

- ディレクトリーベースのストレージプール
- ディスクベースのストレージプール
- パーティションベースのストレージプール
- iSCSI ベースのストレージプール
- LVM ベースのストレージプール
- NFS ベースのストレージプール
- vHBA デバイスを使用した SCSI ベースのストレージプール
- マルチパスベースのストレージプール
- RBD ベースのストレージプール

対応していないストレージプールの種類

以下は、RHEL で対応していない **libvirt** ストレージプールタイプのリストです。

- sheepdog ベースのストレージプール
- vstorage ベースのストレージプール
- ZFS ベースのストレージプール
- iSCSI-direct ストレージプール

- GlusterFS ストレージプール

15.2. CLI を使用した仮想マシンストレージプールの管理

CLI を使用して、ストレージプールの次の側面を管理し、仮想マシン (VM) にストレージを割り当てることができます。

- [ストレージプール情報の表示](#)
- [ストレージプールの作成](#)
 - [CLI を使用したディレクトリーベースのストレージプールの作成](#)
 - [CLI を使用したディスクベースのストレージプールの作成](#)
 - [CLI を使用したファイルシステムベースのストレージプールの作成](#)
 - [CLI を使用した iSCSI ベースのストレージプールの作成](#)
 - [CLI を使用した LVM ベースのストレージプールの作成](#)
 - [CLI を使用した NFS ベースのストレージプールの作成](#)
 - [CLI で vHBA デバイスを使用した SCSI ベースのストレージプールを作成する手順](#)
- [ストレージプールの削除](#)

15.2.1. CLI を使用したストレージプール情報の表示

CLI を使用して、ストレージプールに関する詳細の一部またはすべてが含まれるストレージプールのリストを表示できます。また、リスト表示されているストレージプールにフィルターをかけることもできます。

手順

- `virsh pool-list` コマンドを使用して、ストレージプール情報を表示します。

```
# virsh pool-list --all --details
Name           State  Autostart Persistent Capacity Allocation Available
default        running yes      yes      48.97 GiB 23.93 GiB 25.03 GiB
Downloads      running yes      yes      175.62 GiB 62.02 GiB 113.60 GiB
RHEL-Storage-Pool running yes      yes      214.62 GiB 93.02 GiB 168.60 GiB
```

関連情報

- `virsh pool-list --help` コマンド

15.2.2. CLI を使用したディレクトリーベースのストレージプールの作成

ディレクトリーベースのストレージプールは、マウントされている既存のファイルシステムのディレクトリーを基にしています。これは、たとえば、ファイルシステムの残りの領域を他の目的で使用する場合に役立ちます。`virsh` ユーティリティを使用して、ディレクトリーベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーがディレクトリーのストレージプールをサポートしていることを確認します。

```
# virsh pool-capabilities | grep "'dir' supported='yes'"
```

コマンドの出力が表示される場合には、ディレクトリープールはサポートの対象です。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用し、ディレクトリータイプのストレージプールを定義して作成します。たとえば、`/guest_images` ディレクトリーを使用する **guest_images_dir** という名前のストレージプールを作成するには以下を実行します。

```
# virsh pool-define-as guest_images_dir dir --target "/guest_images"
Pool guest_images_dir defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[Directory-based storage pool parameters](#) を参照してください。

2. ストレージプールのターゲットパスの作成

virsh pool-build コマンドを使用して、フォーマット済みファイルシステムのストレージプール用のストレージプールターゲットパスを作成し、ストレージソースデバイスを初期化し、データのフォーマットを定義します。

```
# virsh pool-build guest_images_dir
Pool guest_images_dir built

# ls -la /guest_images
total 8
drwx-----. 2 root root 4096 May 31 19:38 .
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name           State   Autostart
-----
default        active  yes
guest_images_dir  inactive no
```

4. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_dir
Pool guest_images_dir started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

5. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_dir
Pool guest_images_dir marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_dir
Name:      guest_images_dir
UUID:      c7466869-e82a-a66c-2187-dc9d6f0877d0
State:     running
Persistent: yes
Autostart: yes
Capacity:  458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

15.2.3. CLI を使用したディスクベースのストレージプールの作成

ディスクベースのストレージプールでは、プールはディスクパーティションに基づいています。これは、たとえば、ディスクパーティション全体を仮想マシン (VM) ストレージ専用にする場合に便利です。**virsh** ユーティリティを使用して、ディスクベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーがディスクベースのストレージプールをサポートしていることを確認します。

```
# virsh pool-capabilities | grep "'disk' supported='yes'"
```

コマンドの出力が表示される場合には、ディスクベースのプールはサポートの対象です。

- ストレージプールのベースとなるデバイスを準備します。この目的のために、パーティション (**/dev/sdb1** など) または LVM ボリュームを優先します。ディスク全体またはブロックデバイス (**/dev/sdb** など) への書き込みアクセスを仮想マシンに提供すると、その仮想マシンはそれをパーティション分割するか、その上に独自の LVM グループを作成する可能性があります。これにより、ホストでシステムエラーが発生する可能性があります。ただし、ストレージプールにブロックデバイス全体を使用する必要がある場合、Red Hat は、デバイス上の重要なパーティションを GRUB の **os-prober** 機能から保護することを推奨します。これを行うには、**/etc/default/grub** ファイルを編集して、次のいずれかの設定を適用します。

- **os-prober** を無効にします。

```
GRUB_DISABLE_OS_PROBER=true
```

- **os-prober** が特定のパーティションを検出しないようにします。以下に例を示します。

```
GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
```

- ストレージプールを作成する前に、選択したストレージデバイス上のデータをバックアップします。使用されている **libvirt** のバージョンに応じて、ディスクをストレージプール専用にする、現在ディスクデバイスに格納されているすべてのデータが再フォーマットされて消去される可能性があります。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用し、ディスクタイプのストレージプールを定義して作成します。次の例では、`/dev/sdb` デバイスを使用する **guest_images_disk** という名前のストレージプールを作成します。

```
# virsh pool-define-as guest_images_disk disk --source-format=gpt --source-dev=/dev/sdb --target /dev
Pool guest_images_disk defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[Disk-based storage pool parameters](#) を参照してください。

2. ストレージプールのターゲットパスの作成

virsh pool-build コマンドを使用して、フォーマット済みファイルシステムのストレージプール用のストレージプールターゲットパスを作成し、ストレージソースデバイスを初期化し、データのフォーマットを定義します。

```
# virsh pool-build guest_images_disk
Pool guest_images_disk built
```



注記

ターゲットパスの構築は、ディスクベース、ファイルシステムベース、論理ストレージプールにのみ必要です。**libvirt** は、**overwrite** オプションが指定されている場合を除き、ソースストレージデバイスのデータフォーマットが、選択したストレージプールタイプと異なることを検出すると、ビルドに失敗します。

3. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name           State   Autostart
-----
default        active  yes
guest_images_disk  inactive no
```

4. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

5. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_disk
Pool guest_images_disk marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_disk
Name:      guest_images_disk
UUID:      c7466869-e82a-a66c-2187-dc9d6f0877d0
State:     running
Persistent: yes
Autostart: yes
Capacity:  458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

15.2.4. CLI を使用したファイルシステムベースのストレージプールの作成

マウントされていないファイルシステムにストレージプールを作成する場合は、ファイルシステムベースのストレージプールを使用します。このストレージプールは、指定のファイルシステムのマウントポイントを基にしています。**virsh** ユーティリティを使用すると、ファイルシステムベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーがファイルシステムベースのストレージプールをサポートしていることを確認します。

```
# virsh pool-capabilities | grep "'fs' supported='yes'"
```

コマンドの出力が表示される場合には、ファイルベースのストレージプールはサポートの対象です。

- ストレージプールのベースとなるデバイスを準備します。この目的のために、パーティション (`/dev/sdb1` など) または LVM ボリュームを優先します。ディスク全体またはブロックデバイス (`/dev/sdb` など) への書き込みアクセスを仮想マシンに提供すると、その仮想マシンはそれをパーティション分割するか、その上に独自の LVM グループを作成する可能性があります。これにより、ホストでシステムエラーが発生する可能性があります。

ただし、ストレージプールにブロックデバイス全体を使用する必要がある場合、Red Hat は、デバイス上の重要なパーティションを GRUB の **os-prober** 機能から保護することを推奨します。これを行うには、`/etc/default/grub` ファイルを編集して、次のいずれかの設定を適用します。

- **os-prober** を無効にします。

```
GRUB_DISABLE_OS_PROBER=true
```

- **os-prober** が特定のパーティションを検出しないようにします。以下に例を示します。

```
GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
```

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用し、ファイルシステムタイプのストレージプールを定義して作成します。たとえば、`/dev/sdc1` パーティションを使用し、`/guest_images` ディレクトリにマウントされるストレージプールに **guest_images_fs** という名前を指定して作成するには以下を実行します。

```
# virsh pool-define-as guest_images_fs fs --source-dev /dev/sdc1 --target /guest_images
Pool guest_images_fs defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[Filesystem-based storage pool parameters](#) を参照してください。

2. ストレージプールのターゲットパスの定義

virsh pool-build コマンドを使用して、フォーマット済みファイルシステムのストレージプール用のストレージプールターゲットパスを作成し、ストレージソースデバイスを初期化し、データのフォーマットを定義します。

```
# virsh pool-build guest_images_fs
Pool guest_images_fs built

# ls -la /guest_images
total 8
drwx-----. 2 root root 4096 May 31 19:38 .
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name          State  Autostart
-----
default       active yes
guest_images_fs  inactive no
```

4. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_fs
Pool guest_images_fs started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

5. オプション: 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_fs
Pool guest_images_fs marked as autostarted
```

検証

1. **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_fs
Name:      guest_images_fs
UUID:      c7466869-e82a-a66c-2187-dc9d6f0877d0
State:     running
Persistent: yes
Autostart: yes
Capacity:  458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

2. ファイルシステムのターゲットパスに **lost+found** ディレクトリーがあることを確認します。これは、デバイスがマウントされていることを示しています。

```
# mount | grep /guest_images
/dev/sdc1 on /guest_images type ext4 (rw)

# ls -la /guest_images
total 24
drwxr-xr-x. 3 root root 4096 May 31 19:47 .
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
drwx-----. 2 root root 16384 May 31 14:18 lost+found
```

15.2.5. CLI を使用した iSCSI ベースのストレージプールの作成

iSCSI (Internet Small Computer Systems Interface) は、データストレージ施設をリンクするための IP ベースのストレージネットワーク標準です。iSCSI サーバーにストレージプールを置く場合は、**virsh** ユーティリティーを使用して、iSCSI ベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーが iSCSI ベースのストレージプールをサポートしていることを確認します。

```
# virsh pool-capabilities | grep "'iscsi' supported='yes'"
```

コマンドの出力が表示される場合には、iSCSI ベースのストレージプールはサポートの対象です。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用し、iSCSI タイプのストレージプールを定義して作成します。たとえば、**server1.example.com** で **iqn.2010-05.com.example.server1:iscsirhel7guest** IQN を使用し、**/dev/disk/by-path** パスにマウントされるストレージプールに **guest_images_iscsi** という名前を指定して作成するには、以下を実行します。

```
# virsh pool-define-as --name guest_images_iscsi --type iscsi --source-host
server1.example.com --source-dev iqn.2010-05.com.example.server1:iscsirhel7guest --
target /dev/disk/by-path
Pool guest_images_iscsi defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[iSCSI-based storage pool parameters](#) を参照してください。

2. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name          State   Autostart
-----
default       active  yes
guest_images_iscsi  inactive no
```

3. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_iscsi
Pool guest_images_iscsi started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

4. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_iscsi
Pool guest_images_iscsi marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_iscsi
Name:      guest_images_iscsi
UUID:      c7466869-e82a-a66c-2187-dc9d6f0877d0
State:     running
Persistent: yes
Autostart: yes
Capacity:  458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

15.2.6. CLI を使用した LVM ベースのストレージプールの作成

LVM ボリュームグループに含まれるストレージプールが必要な場合は、**virsh** ユーティリティーを使用して LVM ベースのストレージプールを作成できます。

推奨事項

LVM ベースのストレージプールを作成する前に、以下の点に注意してください。

- LVM ベースのストレージプールは、LVM の柔軟性を完全には提供しません。
- **libvirt** は、シン論理ボリュームに対応しますが、シンストレージプールの機能は提供しません。
- LVM ベースのストレージプールは、ボリュームグループです。**virsh** ユーティリティーを使用してボリュームグループを作成できますが、この方法では、作成したボリュームグループには1つのデバイスしか作成できません。複数のデバイスを持つボリュームグループを作成する場合は、代わりに LVM ユーティリティーを使用します。詳細は、[How to create a volume group in Linux with LVM](#) を参照してください。
ボリュームグループの詳細は、[Red Hat Enterprise Linux Logical Volume Manager Administration Guide](#) を参照してください。
- LVM ベースのストレージプールには、完全なディスクパーティションが必要です。**virsh** コマンドを使用して新しいパーティションまたはデバイスをアクティブにすると、パーティションがフォーマットされ、すべてのデータが消去されます。この手順で説明しているように、ホストの既存のボリュームグループを使用している場合は、何も消去されません。

前提条件

- ハイパーバイザーが LVM ベースのストレージプールをサポートしていることを確認します。

```
# virsh pool-capabilities | grep "'logical' supported='yes'"
```

コマンドの出力が表示される場合には、LVM ベースのストレージプールはサポートの対象です。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用して、LVM タイプのストレージプールを定義して作成します。たとえば、次のコマンドは、**lvm_vg** ボリュームグループを使用し、**/dev/lvm_vg** ディレクトリーにマウントされている **guest_images_lvm** という名前のストレージプールを作成します。

```
# virsh pool-define-as guest_images_lvm logical --source-name lvm_vg --target
/dev/lvm_vg
Pool guest_images_lvm defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[LVM-based storage pool parameters](#) を参照してください。

2. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name          State  Autostart
-----
default       active yes
guest_images_lvm  inactive no
```

3. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_lvm
Pool guest_images_lvm started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

4. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_lvm
Pool guest_images_lvm marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_lvm
Name:      guest_images_lvm
UUID:     c7466869-e82a-a66c-2187-dc9d6f0877d0
State:    running
Persistent: yes
Autostart: yes
Capacity: 458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

15.2.7. CLI を使用した NFS ベースのストレージプールの作成

ネットワークファイルシステム (NFS) サーバーにストレージプールを置く場合は、**virsh** ユーティリティを使用して、NFS ベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーが NFS ベースのストレージプールに対応していることを確認します。

```
# virsh pool-capabilities | grep "<value>nfs</value>"
```

コマンドの出力が表示される場合には、NFS ベースのストレージプールはサポートの対象です。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドを使用し、NFS タイプのストレージプールを定義して作成します。たとえば、ターゲットディレクトリ `/var/lib/libvirt/images/nfspool` を使用してサーバーディレクトリ `/home/net_mount` にマウントされ、IP が `111.222.111.222` の NFS サーバーを使用するストレージプールを、**guest_images_netfs** という名前で作成するには、以下を実行します。

```
# virsh pool-define-as --name guest_images_netfs --type netfs --source-
host='111.222.111.222' --source-path='/home/net_mount' --source-format='nfs' --
target='/var/lib/libvirt/images/nfspool'
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[NFS-based storage pool parameters](#) を参照してください。

2. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

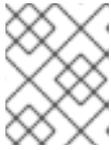
```
# virsh pool-list --all

Name          State   Autostart
-----
default       active  yes
guest_images_netfs  inactive no
```

3. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_netfs
Pool guest_images_netfs started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

4. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_netfs
Pool guest_images_netfs marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_netfs
Name:      guest_images_netfs
UUID:     c7466869-e82a-a66c-2187-dc9d6f0877d0
State:    running
Persistent: yes
Autostart: yes
Capacity: 458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
```

15.2.8. CLI で vHBA デバイスを使用した SCSI ベースのストレージプールを作成する手順

Small Computer System Interface (SCSI) デバイスにストレージプールを設定する場合は、ホストが、仮想ホストバスアダプター (vHBA) を使用して SCSI デバイスに接続できる必要があります。その後、**virsh** ユーティリティを使用して、SCSI ベースのストレージプールを作成できます。

前提条件

- ハイパーバイザーが SCSI ベースのストレージプールをサポートしている。

```
# virsh pool-capabilities | grep "'scsi' supported='yes'"
```

コマンドの出力が表示される場合には、SCSI ベースのストレージプールはサポートの対象です。

- 先に vHBA を作成し、vHBA デバイスで SCSI ベースのストレージプールを作成できるようにしてある。詳細は、[Creating vHBAs](#) を参照してください。

手順

1. ストレージプールを作成します。

virsh pool-define-as コマンドで、vHBA を使用して SCSI ストレージプールを定義して作成します。たとえば、以下は、**guest_images_vhba** という名前のストレージプールを作成します。このストレージプールは、親アダプター (**scsi_host3**)、ワールドワイドポート番号 (**5001a4ace3ee047d**)、ワールドワイドノード番号 (**5001a4a93526d0a1**) で識別される vHBA を使用します。ストレージプールは **/dev/disk/** ディレクトリーにマウントされます。

```
# virsh pool-define-as guest_images_vhba scsi --adapter-parent scsi_host3 --adapter-wwnn 5001a4a93526d0a1 --adapter-wwpn 5001a4ace3ee047d --target /dev/disk/
Pool guest_images_vhba defined
```

作成するストレージプールの XML 設定がすでにある場合は、XML を基にプールを定義することもできます。詳細は、[Parameters for SCSI-based storage pools with vHBA devices](#) を参照してください。

2. プールが作成されたことを確認します。

virsh pool-list コマンドを使用して、プールが作成されたことを確認します。

```
# virsh pool-list --all

Name           State   Autostart
-----
default        active  yes
guest_images_vhba  inactive no
```

3. ストレージプールを起動します。

virsh pool-start コマンドを使用して、ストレージプールをマウントします。

```
# virsh pool-start guest_images_vhba
Pool guest_images_vhba started
```



注記

virsh pool-start コマンドは、永続ストレージプールにのみ必要です。一時的なストレージプールは、作成時に自動的に起動します。

4. (必要に応じて) 自動起動をオンにします。

デフォルトでは、**virsh** コマンドで定義されたストレージプールは、仮想化サービスが起動するたびに自動的に起動するようには設定されていません。**virsh pool-autostart** コマンドを使用して、ストレージプールが自動的に起動するように設定します。

```
# virsh pool-autostart guest_images_vhba
Pool guest_images_vhba marked as autostarted
```

検証

- **virsh pool-info** コマンドを使用して、ストレージプールが **running** 状態であることを確認します。報告されるサイズが期待どおりであるか、また、自動開始が正しく設定されているかを確認してください。

```
# virsh pool-info guest_images_vhba
Name:      guest_images_vhba
```

```

UUID:      c7466869-e82a-a66c-2187-dc9d6f0877d0
State:     running
Persistent: yes
Autostart: yes
Capacity:  458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB

```

15.2.9. CLI を使用したストレージプールの削除

ホストシステムからストレージプールを削除するには、プールを停止して、その XML 定義を削除する必要があります。

手順

1. **virsh pool-list** コマンドを使用して、定義済みストレージプールをリスト表示します。

```

# virsh pool-list --all
Name           State   Autostart
-----
default        active yes
Downloads      active yes
RHEL-Storage-Pool active yes

```

2. **virsh pool-destroy** コマンドを使用して、削除するストレージプールを停止します。

```

# virsh pool-destroy Downloads
Pool Downloads destroyed

```

3. **任意:** ストレージプールの種類によっては、**virsh pool-delete** コマンドを使用して、ストレージプールが含まれるディレクトリーを削除できます。これを実行するには、ディレクトリーが空である必要があります。

```

# virsh pool-delete Downloads
Pool Downloads deleted

```

4. **virsh pool-undefine** コマンドを使用して、ストレージプールの定義を削除します。

```

# virsh pool-undefine Downloads
Pool Downloads has been undefined

```

検証

- ストレージプールが削除されたことを確認します。

```

# virsh pool-list --all
Name           State   Autostart
-----
default        active yes
rhel-Storage-Pool active yes

```

15.3. WEB コンソールを使用した仮想マシンストレージプールの管理

RHEL Web コンソールを使用すると、ストレージプールを管理して、仮想マシンにストレージを割り当てることができます。

Web コンソールを使用して以下を行うことができます。

- [ストレージプール情報の表示](#)。
- ストレージプールの作成:
 - [ディレクトリーベースのストレージプールの作成](#)
 - [NFS ベースのストレージプールの作成](#)
 - [iSCSI ベースのストレージプールの作成](#)
 - [LVM ベースのストレージプールの作成](#)
- [ストレージプールの削除](#)
- [ストレージプールの非アクティブ化](#)

15.3.1. Web コンソールを使用したストレージプール情報の表示

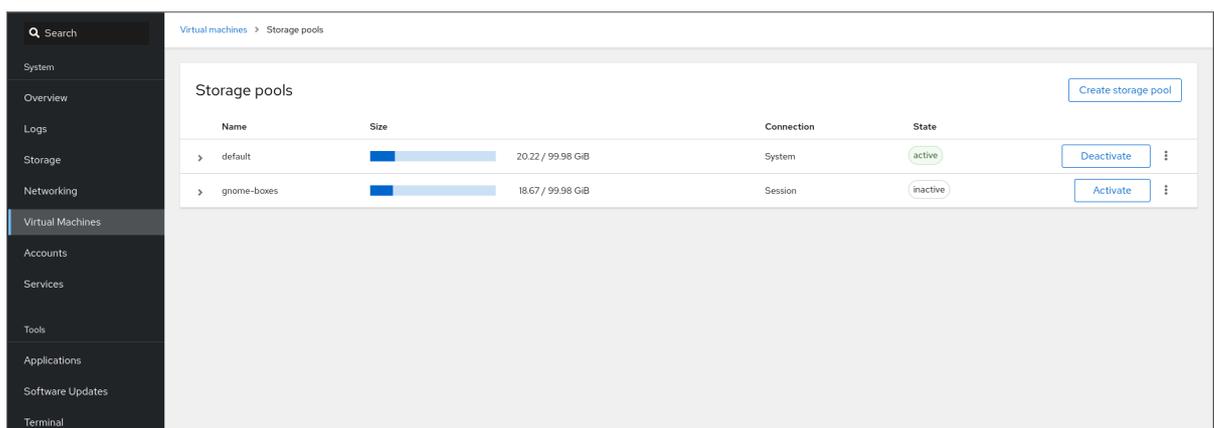
Web コンソールを使用して、システムで利用可能なストレージプールの詳細情報を表示できます。ストレージプールを使用すると、仮想マシンのディスクイメージを作成できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン インターフェイスで [ストレージプール](#) をクリックします。**
Storage pools 画面が表示され、設定されているストレージプールの一覧が示されます。



この情報には以下が含まれます。

- **名前** - ストレージプールの名前
- **サイズ** - 現在の割り当てとストレージプールの合計容量。
- **接続** - ストレージプールのアクセスに使用される接続

- **状態** - ストレージプールのステータス
2. 情報を表示するストレージプールの横にある矢印をクリックします。行がデプロイメントされ、選択したストレージプールに関する詳細情報を含む概要ペインが表示されます。

Property	Value
Target path	/var/lib/libvirt/images
Persistent	yes
Autostart	yes
Type	dir

この情報には以下が含まれます。

- **ターゲットパス** - ストレージプールの場所です。
 - **永続的** - ストレージプールの設定が永続的であるかどうかを示します。
 - **自動起動** - システムの起動時にストレージプールが自動的に起動するかどうかを示します。
 - **種類** - ストレージプールの種類。
3. ストレージプールに関連付けられているストレージボリュームのリストを表示する場合は、**ストレージボリューム** をクリックします。ストレージボリュームペインが表示され、設定したストレージボリュームのリストが表示されます。

Name	Used by	Size
<input type="checkbox"/> volume1		0 / 1GB
<input type="checkbox"/> volume2		0 / 1GB

この情報には以下が含まれます。

- **名前** - ストレージボリュームの名前。
- **使用者** - 現在ストレージボリュームを使用している仮想マシン。
- **サイズ** - ボリュームのサイズ。

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

15.3.2. Web コンソールを使用したディレクトリベースのストレージプールの作成

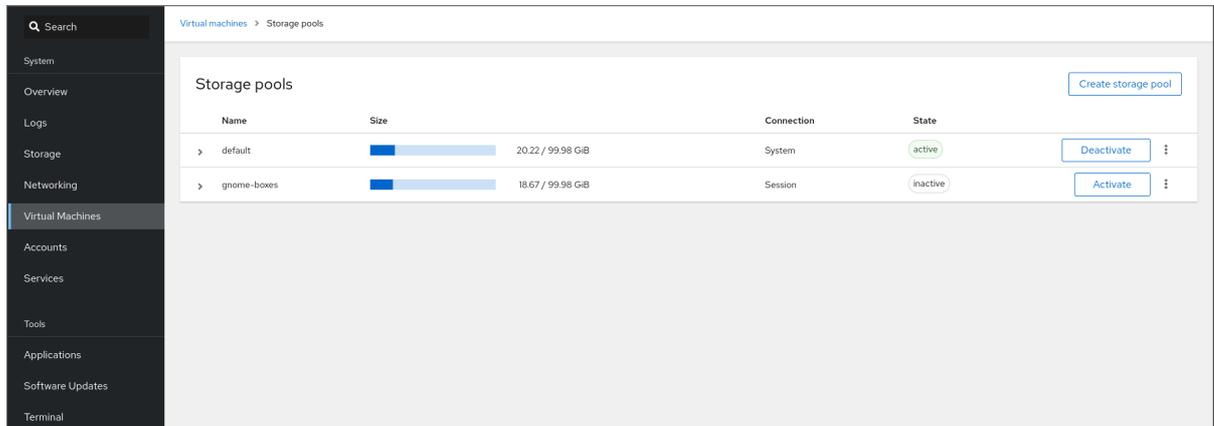
ディレクトリベースのストレージプールは、マウントされている既存のファイルシステムのディレクトリを基にしています。これは、たとえば、ファイルシステムの残りの領域を他の目的で使用する場合に役立ちます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. RHEL Web コンソールで、**Virtual Machines** タブの **Storage pools** をクリックします。
Storage pools 画面が表示され、設定されているストレージプールのリストが示されます。



2. **Create storage pool** をクリックします。
Create storage pool ダイアログボックスが表示されます。
3. ストレージプールの名前を入力します。
4. **Type** ドロップダウンメニューで、**Filesystem directory** を選択します。

Create storage pool ✕

Name

Type

Target path

Startup Start pool when host boots



注記

ドロップダウンメニューに **Filesystem directory** オプションが表示されない場合、ハイパーバイザーはディレクトリーベースのストレージプールをサポートしていません。

5. 以下の情報を入力します。
 - **ターゲットパス** - ストレージプールの場所です。
 - **起動** - ホストの起動時にストレージプールが起動するかどうか
6. **Create** をクリックします。

ストレージプールが作成され、**Create Storage Pool** ダイアログが閉じて、新しいストレージプールがストレージプールのリストに表示されます。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.3. Web コンソールを使用した NFS ベースのストレージプールの作成

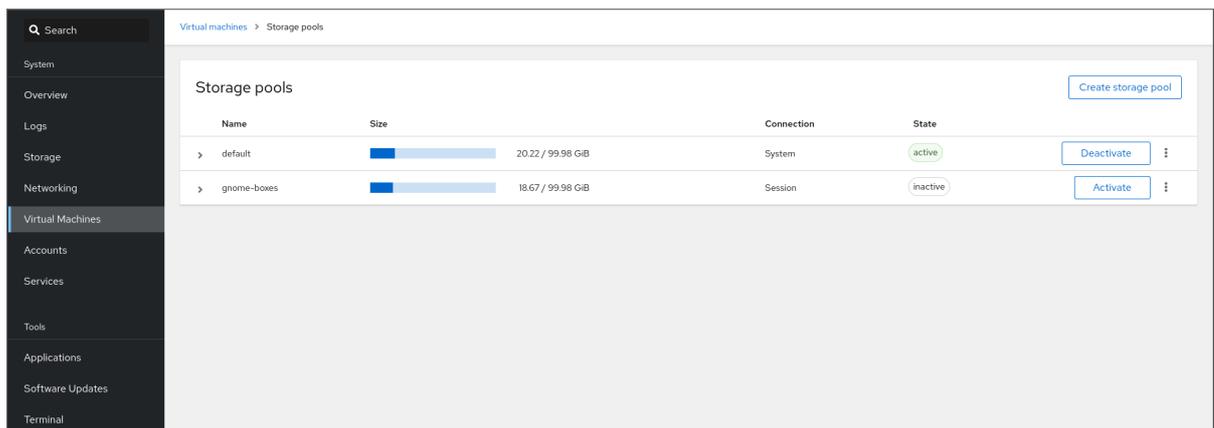
NFS ベースのストレージプールは、サーバーでホストされているファイルシステムに基づいています。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. RHEL Web コンソールで、**Virtual Machines** タブの **Storage pools** をクリックします。**Storage pools** 画面が表示され、設定されているストレージプールのリストが示されます。



2. **Create storage pool** をクリックします。**Create storage pool** ダイアログボックスが表示されます。
3. ストレージプールの名前を入力します。
4. **Type** ドロップダウンメニューで、**Network file system** を選択します。

Create storage pool ×

Name

Type Network file system ▼

Target path Path on host's filesystem ▼

Host

Source path

Startup Start pool when host boots

Create
Cancel



注記

ドロップダウンメニューに **Network file system** オプションが表示されない場合、ハイパーバイザーは nfs ベースのストレージプールをサポートしていません。

5. 残りの情報を入力します。

- **Target path** - ターゲットを指定するパス。ストレージプールに使用されるパスになります。
- **Host** - マウントポイントがあるネットワークサーバーのホスト名。これは、ホスト名または IP アドレスになります。
- **Source path** - ネットワークサーバーで使用されるディレクトリー。
- **起動** - ホストの起動時にストレージプールが起動するかどうか

6. **Create** をクリックします。

ストレージプールが作成されます。**Create storage pool** ダイアログが閉じ、新しいストレージプールがストレージプールのリストに表示されます。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.4. Web コンソールを使用した iSCSI ベースのストレージプールの作成

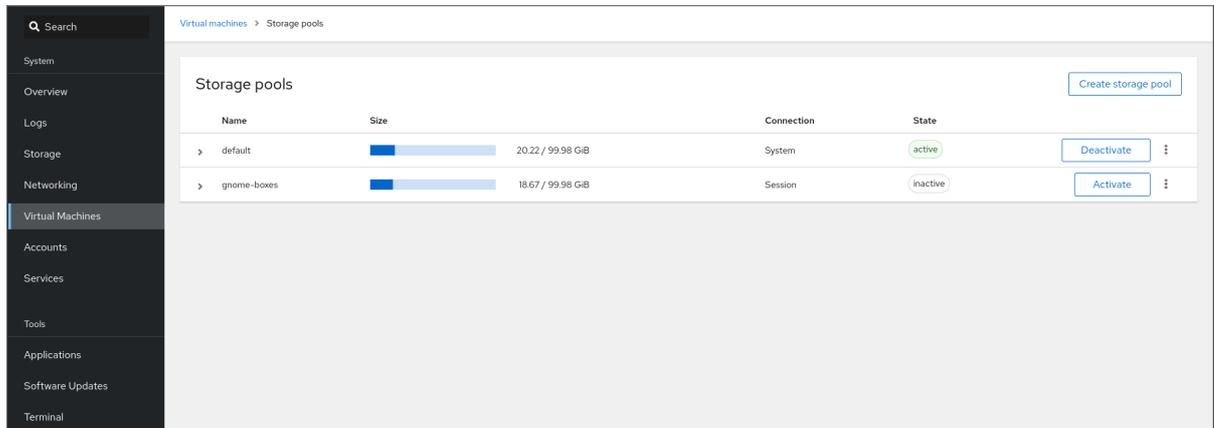
iSCSI ベースのストレージプールは、iSCSI (Internet Small Computer Systems Interface) をベースとする、データストレージ施設をリンクするための IP ベースのストレージネットワーク規格です。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. RHEL Web コンソールで、**Virtual Machines** タブの **Storage pools** をクリックします。
Storage pools 画面が表示され、設定されているストレージプールのリストが示されます。



2. **Create storage pool** をクリックします。
Create storage pool ダイアログボックスが表示されます。
3. ストレージプールの名前を入力します。
4. **Type** ドロップダウンメニューで、**iSCSI target** を選択します。

Create storage pool ×

Name

Type

Target path

Host

Source path

Startup Start pool when host boots

5. 残りの情報を入力します。
 - **Target Path** - ターゲットを指定するパス。ストレージプールに使用されるパスになります。
 - **Host** - iSCSI サーバーのホスト名または IP アドレス。
 - **Source path** - iSCSI ターゲットの一意の iSCSI 修飾名 (IQN)。
 - **起動** - ホストの起動時にストレージプールが起動するかどうか

6. **Create** をクリックします。
ストレージプールが作成されます。**Create storage pool** ダイアログが閉じ、新しいストレージプールがストレージプールのリストに表示されます。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.5. Web コンソールを使用したディスクベースのストレージプールの作成

ディスクベースのストレージプールは、ディスクパーティション全体を使用します。



警告

- 使用されている libvirt のバージョンに応じて、ディスクをストレージプール専用にする、現在ディスクデバイスに格納されているすべてのデータが再フォーマットされて消去される可能性があります。ストレージプールを作成する前に、ストレージデバイスのデータのバックアップを作成することを強く推奨します。
- ディスク全体またはブロックデバイスが仮想マシンに渡されると、仮想マシンはそれをパーティション分割するか、その上に独自の LVM グループを作成する可能性があります。これにより、ホストマシンがこのようなパーティションまたは LVM グループを検出し、エラーが発生する可能性があります。
これらのエラーは、パーティションまたは LVM グループを手動で作成して仮想マシンに渡す場合にも発生する可能性があります。

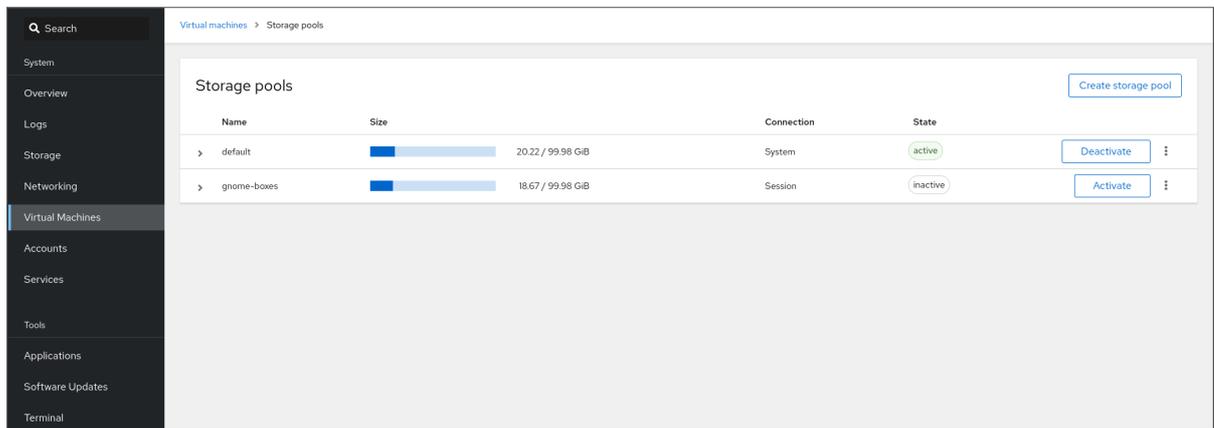
これらのエラーを回避するには、代わりにファイルベースのストレージプールを使用します。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. RHEL Web コンソールで、**Virtual Machines** タブの **Storage pools** をクリックします。
Storage pools 画面が表示され、設定されているストレージプールのリストが示されます。



2. **Create storage pool** をクリックします。
Create storage pool ダイアログボックスが表示されます。
3. ストレージプールの名前を入力します。
4. **Type** ドロップダウンメニューで、**Physical disk device** を選択します。



注記

ドロップダウンメニューに **Physical disk device** オプションが表示されない場合、ハイパーバイザーはデータベースのストレージプールをサポートしていません。

5. 残りの情報を入力します。
 - **Target Path** - ターゲットデバイスを指定するパス。ストレージプールに使用されるパスになります。
 - **Source path** - ストレージデバイスを指定するパス。たとえば、**/dev/sdb** です。
 - **Format** - パーティションテーブルのタイプ。
 - **起動** - ホストの起動時にストレージプールが起動するかどうか
6. **Create** をクリックします。

ストレージプールが作成されます。**Create storage pool** タイアログが閉じ、新しいストレージプールがストレージプールのリストに表示されます。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.6. Web コンソールを使用した LVM ベースのストレージプールの作成

LVM ベースのストレージプールはボリュームグループに基づいており、論理ボリュームマネージャー (LVM) を使用して管理できます。ボリュームグループは、単一のストレージ構造を作成する複数の物理ボリュームの組み合わせです。

注記

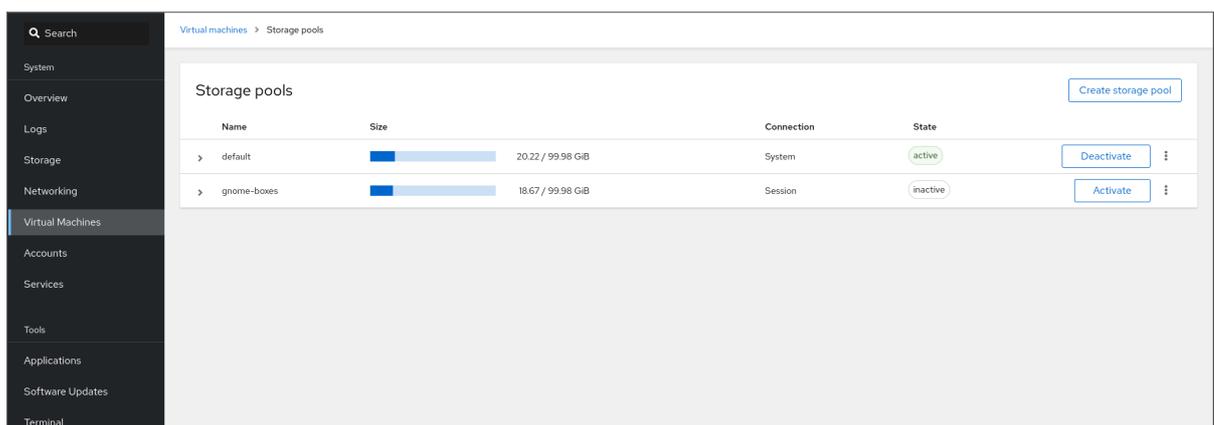
- LVM ベースのストレージプールは、LVM の柔軟性を完全には提供しません。
- **libvirt** は、シン論理ボリュームに対応しますが、シンストレージプールの機能は提供しません。
- LVM ベースのストレージプールには、完全なディスクパーティションが必要です。**virsh** コマンドを使用して新しいパーティションまたはデバイスをアクティブにすると、パーティションがフォーマットされ、すべてのデータが消去されます。この手順で説明しているように、ホストの既存のボリュームグループを使用している場合は、何も消去されません。
- 複数のデバイスを持つボリュームグループを作成する場合は、代わりに LVM ユーティリティを使用します。詳細は、[How to create a volume group in Linux with LVM](#) を参照してください。
ボリュームグループの詳細は、[Red Hat Enterprise Linux Logical Volume Manager Administration Guide](#) を参照してください。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. RHEL Web コンソールで、**Virtual Machines** タブの **Storage pools** をクリックします。**Storage pools** 画面が表示され、設定されているストレージプールのリストが示されます。



2. **Create storage pool** をクリックします。
Create storage pool ダイアログボックスが表示されます。
3. ストレージプールの名前を入力します。
4. **Type** ドロップダウンメニューで、**LVM volume group** を選択します。



注記

ドロップダウンメニューに **LVM volume group** オプションが表示されない場合、ハイパーバイザーは LVM ベースのストレージプールをサポートしていません。

5. 残りの情報を入力します。
 - **Source volume group** - 使用する LVM ボリュームグループの名前。
 - **起動** - ホストの起動時にストレージプールが起動するかどうか
6. **Create** をクリックします。
ストレージプールが作成されます。Create storage pool ダイアログが閉じ、新しいストレージプールがストレージプールのリストに表示されます。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.7. Web コンソールを使用したストレージプールの削除

ストレージプールを削除してホストまたはネットワーク上のリソースを解放し、システムパフォーマンスを向上させることができます。ストレージプールを削除すると、他の仮想マシンで使用できるようにリソースが解放されます。



重要

明示的に指定されていない限り、ストレージプールを削除しても、そのプール内のストレージボリュームは同時に削除されません。

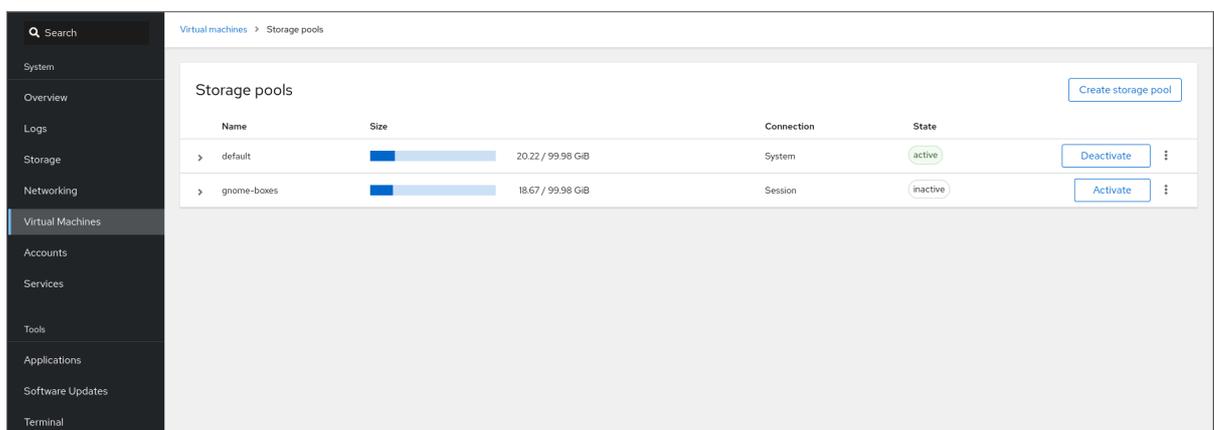
ストレージプールを削除するのではなく一時的に非アクティブにする場合は、[Web コンソールを使用したストレージプールの非アクティブ化](#)を参照してください。

前提条件

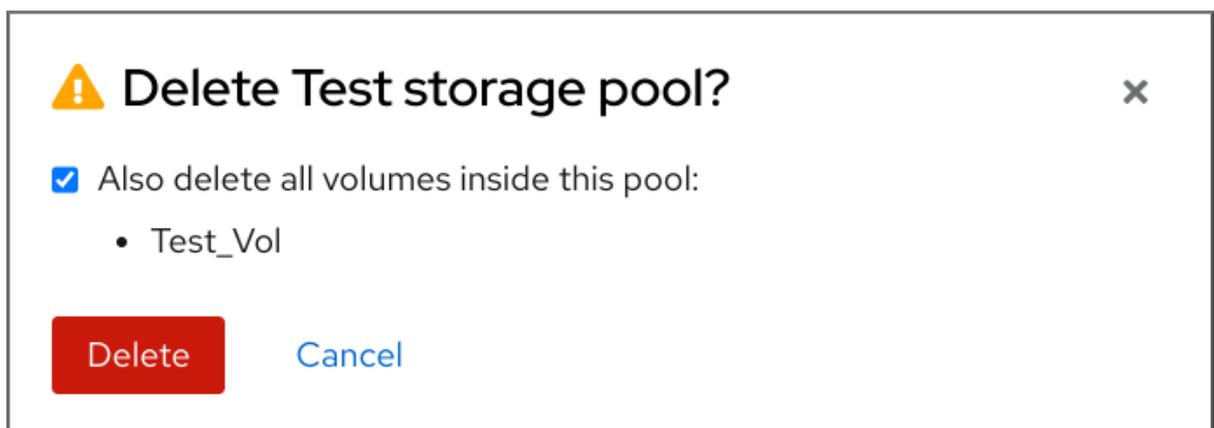
- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 仮想マシンから [ディスクを切り離し](#)ます。
- 関連するストレージボリュームをプールと共に削除する場合は、プールをアクティブ化します。

手順

1. **Virtual Machines** タブの **Storage Pools** をクリックします。
Storage Pools 画面が表示され、設定されているストレージプールのリストが示されます。



2. 削除するストレージプールのメニューボタン **⋮** をクリックし、**Delete** をクリックします。
確認ダイアログが表示されます。



3. **Optional:** プール内のストレージボリュームを削除するには、ダイアログで対応するチェックボックスをオンにします。
4. **Delete** をクリックします。
ストレージプールが削除されます。直前の手順でチェックボックスを選択した場合は、関連付けられたストレージボリュームも削除されます。

関連情報

- [Understanding storage pools](#)

- [Web コンソールを使用したストレージプール情報の表示](#)

15.3.8. Web コンソールを使用したストレージプールの非アクティブ化

ストレージプールを永続的に削除しない場合は、代わりに一時的に無効にできます。

ストレージプールを無効にすると、そのプールに新しいボリュームを作成できません。ただし、そのプールにボリュームがある仮想マシンは、引き続き実行されます。この設定は、プールに作成できるボリューム数を制限してシステムパフォーマンスを向上させる場合などに役立ちます。

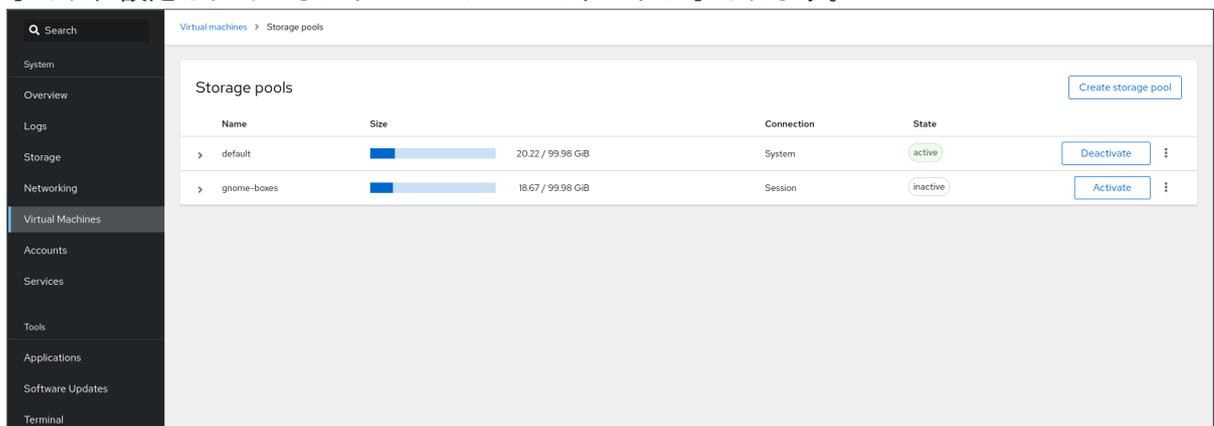
RHEL Web コンソールを使用してストレージプールを無効にするには、以下の手順を参照してください。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. 仮想マシンタブの上部にある **ストレージプール** をクリックします。ストレージプール画面が表示され、設定されているストレージプールのリストが示されます。



2. ストレージプールの行で **Deactivate** をクリックします。ストレージプールは非アクティブになります。

関連情報

- [Understanding storage pools](#)
- [Web コンソールを使用したストレージプール情報の表示](#)

15.4. ストレージプールを作成するパラメーター

必要なストレージプールのタイプに基づいて、その XML 設定ファイルを変更し、特定のタイプのストレージプールを定義できます。本セクションは、さまざまなタイプのストレージプールを作成するために必要な XML パラメーターと、例を説明します。

15.4.1. ディレクトリーベースのストレージプールのパラメーター

XML 設定ファイルを使用してディレクトリーベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

virsh pool-define を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_dir
```

パラメーター

以下の表は、ディレクトリベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.1 ディレクトリベースのストレージプールのパラメーター

説明	XML
ストレージプールの種類	<code><pool type='dir'></code>
ストレージプールの名前	<code><name>name</name></code>
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<code><target></code> <code> <path>target_path</path></code> <code></target></code>

例

以下は、`/guest_images` ディレクトリに基づいたストレージプールの XML ファイルの例です。

```
<pool type='dir'>
  <name>dirpool</name>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

関連情報

- [CLI を使用したディレクトリベースのストレージプールの作成](#)

15.4.2. ディスクベースのストレージプールのパラメーター

XML 設定ファイルを使用してディスクベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

virsh pool-define を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_disk
```

パラメーター

以下の表は、ディスクベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

以下の表は、ディスクベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.2 ディスクベースのストレージプールのパラメーター

説明	XML
ストレージプールの種類	<code><pool type='disk'></code>
ストレージプールの名前	<code><name>name</name></code>
ストレージデバイスを指定するパス。たとえば、 <code>/dev/sdb</code> です。	<code><source></code> <code> <path>source_path</path></code> <code></source></code>
ターゲットデバイスを指定するパス。ストレージプールに使用されるパスになります。	<code><target></code> <code> <path>target_path</path></code> <code></target></code>

例

以下は、ディスクに基づいたストレージプールに対する XML ファイルの例です。

```
<pool type='disk'>
  <name>phy_disk</name>
  <source>
    <device path='/dev/sdb'>
      <format type='gpt'>
    </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

関連情報

- [CLI を使用したディスクベースのストレージプールの作成](#)

15.4.3. ファイルシステムベースのストレージプールパラメーター

XML 設定ファイルを使用してファイルシステムベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

`virsh pool-define` を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_fs
```

パラメーター

次の表は、ファイルシステムベースのストレージプールのXMLファイルに必要なパラメーターのリストです。

表15.3 ファイルシステムベースのストレージプールパラメーター

説明	XML
ストレージプールの種類	<code><pool type='fs'></code>
ストレージプールの名前	<code><name>name</name></code>
パーミッションを指定するパス。たとえば、 <code>/dev/sdc1</code> です。	<code><source></code> <code><device path=device_path /></code>
ファイルシステムのタイプ (ext4 など)。	<code><format type=fs_type /></code> <code></source></code>
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<code><target></code> <code><path>path-to-pool</path></code> <code></target></code>

例

以下は、`/dev/sdc1` パーティションに基づいたストレージプールに対するXMLファイルの例です。

```
<pool type='fs'>
  <name>guest_images_fs</name>
  <source>
    <device path='/dev/sdc1' />
    <format type='auto' />
  </source>
  <target>
    <path>guest_images</path>
  </target>
</pool>
```

関連情報

- [CLIを使用したファイルシステムベースのストレージプールの作成](#)

15.4.4. iSCSI ベースのストレージプールパラメーター

XML 設定ファイルを使用して iSCSI ベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

`virsh pool-define` を使用すると、指定したファイルのXML設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_iscsi
```

パラメーター

以下の表は、iSCSI ベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.4 iSCSI ベースのストレージプールパラメーター

説明	XML
ストレージプールの種類	<code><pool type='iscsi'></code>
ストレージプールの名前	<code><name>name</name></code>
ホストの名前	<code><source> <host name=hostname /></code>
iSCSI IQN	<code><device path= iSCSI_IQN /> </source></code>
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<code><target> <path>/dev/disk/by-path</path> </target></code>
(必要に応じて) iSCSI イニシエーターの IQN。これは、ACL が LUN を特定のイニシエーターに制限する場合に限り必要です。	<code><initiator> <iqn name='initiator0' /> </initiator></code>



注記

iSCSI イニシエーターの IQN は、`virsh find-storage-pool-sources-as iscsi` コマンドを使用して指定できます。

例

以下は、指定した iSCSI に基づいたストレージプールに対する XML ファイルの例です。

```
<pool type='iscsi'>
  <name>iscsi_pool</name>
  <source>
    <host name='server1.example.com' />
    <device path='iqn.2010-05.com.example.server1:iscsirhel7guest' />
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

関連情報

- [CLI を使用した iSCSI ベースのストレージプールの作成](#)

15.4.5. LVM ベースのストレージプールパラメーター

XML 設定ファイルを使用して LVM ベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

virsh pool-define を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_logical
```

パラメーター

以下の表は、LVM ベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.5 LVM ベースのストレージプールパラメーター

説明	XML
ストレージプールの種類	<code><pool type='logical'></code>
ストレージプールの名前	<code><name>name</name></code>
ストレージプールのデバイスのパス	<code><source> <device path='device_path' /></code>
ボリュームグループの名前	<code><name>VG-name</name></code>
仮想グループの形式	<code><format type='lvm2' /> </source></code>
ターゲットパス	<code><target> <path=target_path /> </target></code>

注記

論理ボリュームグループが複数のディスクパーティションで作成されている場合は、複数のソースデバイスがリスト表示されている可能性があります。以下に例を示します。

```
<source>
  <device path='/dev/sda1'/>
  <device path='/dev/sdb3'/>
  <device path='/dev/sdc2'/>
  ...
</source>
```

例

以下は、指定した LVM に基づいたストレージプールに対する XML ファイルの例です。

```
<pool type='logical'>
  <name>guest_images_lvm</name>
```

```

<source>
  <device path='/dev/sdc'/>
  <name>libvirt_lvm</name>
  <format type='lvm2'/>
</source>
<target>
  <path>/dev/libvirt_lvm</path>
</target>
</pool>

```

関連情報

- [CLI を使用した LVM ベースのストレージプールの作成](#)

15.4.6. NFS ベースのストレージプールパラメーター

XML 設定ファイルを使用して NFS ベースのストレージプールを作成または変更する場合は、必要なパラメーターを指定する必要があります。これらのパラメーターの詳細は、以下の表を参照してください。

virsh pool-define を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```

# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_netfs

```

パラメーター

以下の表は、NFS ベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.6 NFS ベースのストレージプールパラメーター

説明	XML
ストレージプールの種類	<pool type='netfs'>
ストレージプールの名前	<name>name</name>
マウントポイントがあるネットワークサーバーのホスト名。これは、ホスト名または IP アドレスになります。	<source> <host name=hostname />
ストレージプールの形式	次のいずれかになります。 <format type='nfs' /> <format type='cifs' />
ネットワークサーバーで使用されるディレクトリー	<dir path=source_path /> </source>

説明	XML
ターゲットを指定するパス。ストレージプールに使用されるパスになります。	<pre><target> <path>target_path</path> </target></pre>

例

以下は、NFS サーバー `file_server` の `/home/net_mount` ディレクトリーでのストレージプールの XML ファイルの例です。

```
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='file_server'/>
    <format type='nfs'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

関連情報

- [CLI を使用した NFS ベースのストレージプールの作成](#)

15.4.7. vHBA デバイスを使用した SCSI ベースのストレージプールのパラメーター

仮想ホストアダプターバス (vHBA) デバイスを使用する SCSI ベースのストレージプール用の XML 設定ファイルを作成または変更する場合は、XML 設定ファイルに必要な特定のパラメーターを含める必要があります。必要なパラメーターの詳細は、以下の表を参照してください。

`virsh pool-define` を使用すると、指定したファイルの XML 設定を基にしてストレージプールを作成できます。以下に例を示します。

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_vhba
```

パラメーター

以下の表は、vHBA を使用する iSCSI ベースのストレージプールの XML ファイルに必要なパラメーターのリストです。

表15.7 vHBA デバイスを使用した SCSI ベースのストレージプールのパラメーター

説明	XML
ストレージプールの種類	<code><pool type='scsi'></code>
ストレージプールの名前	<code><name>name</name></code>

説明	XML
vHBA の識別子。parent 属性はオプションです。	<pre><source> <adapter type='fc_host' [parent=parent_scsi_device] wwnn='WWNN' wwpn='WWPN' /> </source></pre>
ターゲットパス。ストレージプールに使用されるパスになります。	<pre><target> <path=target_path /> </target></pre>

重要

<path> フィールドが /dev/ の場合、libvirt は、ボリュームデバイスパスで一意的な短いデバイスパスを生成します。たとえば、/dev/sdc です。それ以外の場合は、物理ホストパスが使用されます。たとえば、/dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0 などです。一意的な短いデバイスパスを使用すると、複数のストレージプールで、同じボリュームを複数の仮想マシン (VM) にリスト表示できません。物理ホストのパスを複数の仮想マシンで使用すると、デバイスタイプが重複していることを示す警告が発生することがあります。

注記

parent 属性は、パスを変更して NPIV LUN の使用元となる物理 HBA の親を識別する <adapter> フィールドで使用できます。scsi_hostN フィールドは、vports 属性および max_vports 属性と合わせて、親 ID を作成します。parent、parent_wwnn、parent_wwpn、または parent_fabric_wwn の属性は、ホストの再起動後に同じ HBA が使用されることを保証するさまざまなレベルを提供します。

- 親を指定しないと、libvirt は、NPIV に対応する最初の scsi_hostN アダプターを使用します。
- 親のみを指定し、設定に SCSI ホストアダプターを追加すると、問題が発生する可能性があります。
- parent_wwnn または parent_wwpn を指定すると、ホストの再起動後に同じ HBA が使用されます。
- ホストの再起動後、parent_fabric_wwn を使用すると、scsi_hostN が使用されたかどうかにかかわらず、同じファブリックの HBA が選択されます。

例

以下は、vHBA を使用する SCSI ベースのストレージプールの XML ファイルの例です。

- HBA にある唯一のストレージプール

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
```

```
<source>
  <adapter type='fc_host' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
</source>
<target>
  <path>/dev/disk/by-path</path>
</target>
</pool>
```

- **parent** 属性を使用して SCSI ホストデバイスを識別し、vHBA を1つ使用する複数のストレージプールの1つ。

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

関連情報

- [CLI で vHBA デバイスを使用した SCSI ベースのストレージプールを作成する手順](#)

15.5. CLI を使用した仮想マシンのストレージボリュームの管理

CLI を使用して、ストレージボリュームの次の側面を管理し、仮想マシン (VM) にストレージを割り当てることができます。

- [ストレージボリューム情報の表示](#)
- [ストレージボリュームの作成](#)
- [ストレージボリュームの削除](#)

15.5.1. CLI を使用したストレージボリューム情報の表示

コマンドラインを使用して、ホストで利用可能なすべてのストレージプールのリストと、指定したストレージプールの詳細を表示できます。

手順

1. **virsh vol-list** コマンドを使用して、指定したストレージプールにあるストレージボリュームをリスト表示します。

```
# virsh vol-list --pool RHEL-Storage-Pool --details
Name          Path                                          Type  Capacity Allocation
-----
.bash_history  /home/VirtualMachines/.bash_history        file  18.70 KiB  20.00 KiB
.bash_logout  /home/VirtualMachines/.bash_logout        file   18.00 B   4.00 KiB
.bash_profile  /home/VirtualMachines/.bash_profile        file  193.00 B   4.00 KiB
.bashrc       /home/VirtualMachines/.bashrc             file   1.29 KiB   4.00 KiB
```

```
.git-prompt.sh /home/VirtualMachines/.git-prompt.sh file 15.84 KiB 16.00 KiB
.gitconfig /home/VirtualMachines/.gitconfig file 167.00 B 4.00 KiB
RHEL_Volume.qcow2 /home/VirtualMachines/RHEL8_Volume.qcow2 file 60.00 GiB
13.93 GiB
```

2. **virsh vol-info** コマンドを使用して、指定したストレージプール内のストレージボリュームをリスト表示します。

```
# virsh vol-info --pool RHEL-Storage-Pool --vol RHEL_Volume.qcow2
Name:      RHEL_Volume.qcow2
Type:      file
Capacity:  60.00 GiB
Allocation: 13.93 GiB
```

15.5.2. CLI を使用したストレージボリュームの作成と割り当て

ディスクイメージを取得して、仮想ディスクとして仮想マシンに割り当てるには、ストレージボリュームを作成し、その XML 設定を仮想マシンに割り当てます。

前提条件

- 空き領域が割り当てられていないストレージプールがホストに存在する。
 - ホストのストレージプールをリスト表示して確認します。

```
# virsh pool-list --details
```

Name	State	Autostart	Persistent	Capacity	Allocation	Available
default	running	yes	yes	48.97 GiB	36.34 GiB	12.63 GiB
Downloads	running	yes	yes	175.92 GiB	121.20 GiB	54.72 GiB
VM-disks	running	yes	yes	175.92 GiB	121.20 GiB	54.72 GiB

- 既存のストレージプールがない場合は、作成します。詳細は、[仮想マシンのストレージの管理](#) を参照してください。

手順

1. **virsh vol-create-as** コマンドを使用してストレージボリュームを作成します。たとえば、**guest-images-fs** ストレージプールをもとに 20 GB qcow2 ボリュームを作成するには以下を実行します。

```
# virsh vol-create-as --pool guest-images-fs --name vm-disk1 --capacity 20 --format qcow2
```

重要: ストレージプールタイプによっては、**virsh vol-create-as** コマンドがサポートされないため、代わりにストレージボリューム作成の特定のプロセスが必要になります。

- **iSCSI ベース** - iSCSI サーバーに事前に iSCSI LUN を準備します。
- **マルチパスベース** - **multipathd** コマンドを使用して、マルチパスを準備または管理します。
- **vHBA ベース** - ファイバーチャネルカードを事前に準備します。

- XML ファイルを作成し、そのファイルに以下の行を追加します。このファイルは、ストレージボリュームをディスクとして仮想マシンに追加するために使用します。

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='qcow2' />
  <source pool='guest-images-fs' volume='vm-disk1' />
  <target dev='hdk' bus='ide' />
</disk>
```

この例では、前の手順で作成した **vm-disk1** ボリュームを使用する仮想ディスクを指定し、このボリュームを **ide** バスに **hdk** ディスクとして指定するように設定します。実際の環境に応じてそれぞれのパラメーターを変更します。

重要: 特定のストレージプールタイプでは、別の XML 形式を使用してストレージボリュームディスクを記述する必要があります。

- マルチパスベースのプールの場合:

```
<disk type='block' device='disk'>
<driver name='qemu' type='raw' />
<source dev='/dev/mapper/mpatha' />
<target dev='sda' bus='scsi' />
</disk>
```

- RBD ベースのストレージプールの場合:

```
<disk type='network' device='disk'>
<driver name='qemu' type='raw' />
<source protocol='rbd' name='pool/image'>
  <host name='mon1.example.org' port='6321' />
</source>
<target dev='vdc' bus='virtio' />
</disk>
```

- XML ファイルを使用して、ストレージボリュームをディスクとして仮想マシンに割り当てます。たとえば、`~/vm-disk1.xml` で定義されたディスクを **testguest1** 仮想マシンに割り当てるには、次のコマンドを使用します。

```
# virsh attach-device --config testguest1 ~/vm-disk1.xml
```

検証

- 仮想マシンのゲストオペレーティングシステムで、ディスクイメージが未フォーマットかつ未割り当てのディスクとして利用できるようになっていることを確認します。

15.5.3. CLI を使用したストレージボリュームの削除

ホストシステムからストレージボリュームを削除するには、プールを停止して、その XML 定義を削除する必要があります。

前提条件

- 削除するストレージボリュームを使用する仮想マシンがすべてシャットダウンしている。

手順

1. **virsh vol-list** コマンドを使用して、指定したストレージプールにあるストレージボリュームをリスト表示します。

```
# virsh vol-list --pool RHEL-SP
Name          Path
-----
.bash_history  /home/VirtualMachines/.bash_history
.bash_logout  /home/VirtualMachines/.bash_logout
.bash_profile  /home/VirtualMachines/.bash_profile
.bashrc        /home/VirtualMachines/.bashrc
.git-prompt.sh /home/VirtualMachines/.git-prompt.sh
.gitconfig     /home/VirtualMachines/.gitconfig
vm-disk1       /home/VirtualMachines/vm-disk1
```

2. オプション - **virsh vol-wipe** コマンドを使用して、ストレージボリュームをワイプします。たとえば、ストレージプール **RHEL-SP** に関連付けられている **vm-disk1** という名前のストレージボリュームを削除する場合は、次のコマンドを実行します。

```
# virsh vol-wipe --pool RHEL-SP vm-disk1
Vol vm-disk1 wiped
```

3. **virsh vol-delete** コマンドを使用して、ストレージボリュームを削除します。たとえば、ストレージプール **RHEL-SP** に関連付けられている **vm-disk1** という名前のストレージボリュームを削除する場合は、次のコマンドを実行します。

```
# virsh vol-delete --pool RHEL-SP vm-disk1
Vol vm-disk1 deleted
```

検証

- **virsh vol-list** を再度実行して、ストレージボリュームが削除されたことを確認します。

```
# virsh vol-list --pool RHEL-SP
Name          Path
-----
.bash_history  /home/VirtualMachines/.bash_history
.bash_logout  /home/VirtualMachines/.bash_logout
.bash_profile  /home/VirtualMachines/.bash_profile
.bashrc        /home/VirtualMachines/.bashrc
.git-prompt.sh /home/VirtualMachines/.git-prompt.sh
.gitconfig     /home/VirtualMachines/.gitconfig
```

15.6. CLI を使用した仮想ディスクイメージの管理

仮想ディスクイメージは、[仮想ストレージボリューム](#) の一種であり、ハードドライブが物理マシンにストレージを提供するのと同様に、仮想マシンにストレージを提供します。

[新しい仮想マシンを作成する](#) 場合、特に指定しない限り、**libvirt** は新しいディスクイメージを自動的に作成します。ただし、ユースケースによっては、仮想マシンとは別にディスクイメージを作成して管理することが必要になる場合があります。

15.6.1. qemu-img を使用した仮想ディスクイメージの作成

新しい仮想マシンとは別に新しい仮想ディスクイメージを作成する必要があり、[ストレージボリュームの作成](#) が実行できない場合は、**qemu-img** コマンドラインユーティリティーを使用できます。

手順

- **qemu-img** ユーティリティーを使用して仮想ディスクイメージを作成します。

```
# qemu-img create -f <format> <image-name> <size>
```

たとえば、次のコマンドは、**test-image** という名前の、30 GB の qcow2 ディスクイメージを作成します。

```
# qemu-img create -f qcow2 test-image 30G
```

```
Formatting 'test-img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib
size=32212254720 lazy_refcounts=off refcount_bits=16
```

検証

- 作成したイメージに関する情報を表示して、必要なサイズであること、および破損していないことを確認します。

```
# qemu-img info <test-img>
image: test-img
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
  compat: 1.1
  compression type: zlib
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
  extended l2: false
```

関連情報

- [CLI を使用したストレージボリュームの作成と割り当て](#)
- [Web コンソールを使用した仮想マシンへの新しいディスクの追加](#)
- **qemu-img** の man ページ

15.6.2. 仮想ディスクイメージの整合性のチェック

ディスクイメージを仮想マシンにアタッチする前に、ディスクイメージに破損や著しい断片化などの問題がないことを確認します。確認には、**qemu-img check** コマンドを使用します。

必要に応じて、このコマンドを使用してディスクイメージの修復を試みることもできます。

前提条件

- ディスクイメージを使用する仮想マシンがすべてシャットダウンしている。

手順

1. テストするイメージに対して **qemu-img check** コマンドを使用します。以下に例を示します。

```
# qemu-img check <test-name.qcow2>
```

```
No errors were found on the image.
327434/327680 = 99.92% allocated, 0.00% fragmented, 0.00% compressed clusters
Image end offset: 21478375424
```

チェックでディスクイメージに問題が見つかった場合、コマンドの出力は次のようになります。

```
167 errors were found on the image.
Data may be corrupted, or further writes to the image may corrupt it.
```

```
453368 leaked clusters were found on the image.
This means waste of disk space, but no harm to data.
```

```
259 internal errors have occurred during the check.
Image end offset: 21478375424
```

2. **qemu-img check** コマンドで **-r all** オプションを指定して、問題を修復します。ただし、問題の一部しか解決されない可能性があることに注意してください。



警告

ディスクイメージを修復すると、データの破損やその他の問題が発生する可能性があります。修復を試みる前に、ディスクイメージをバックアップしてください。

```
# qemu-img check -r all <test-name.qcow2>
```

```
[...]
122 errors were found on the image.
Data may be corrupted, or further writes to the image may corrupt it.
```

```
250 internal errors have occurred during the check.
Image end offset: 27071414272
```

この出力は、修復後にディスクイメージで見つかった問題の数を示しています。

3. さらにディスクイメージの修復が必要な場合は、**guestfish シェル** のさまざまな **libguestfs** ツールを使用できます。

関連情報

- **qemu-img** の man ページ
- **guestfish** の man ページ

15.6.3. 仮想ディスクイメージのサイズ変更

既存のディスクイメージに追加の領域が必要な場合は、**qemu-img resize** ユーティリティーを使用して、ユースケースに合わせてイメージのサイズを変更できます。

前提条件

- ディスクイメージのバックアップを作成している。
- ディスクイメージを使用する仮想マシンがすべてシャットダウンしている。



警告

実行中の仮想マシンのディスクイメージのサイズを変更すると、データの破損やその他の問題が発生する可能性があります。

- ホストのハードディスクに、意図したディスクイメージサイズに対して十分な空き領域がある。
- オプション: ディスクイメージにデータの破損や同様の問題がないことを確認済みである。手順については、[仮想ディスクイメージの整合性のチェック](#) を参照してください。

手順

1. サイズを変更する仮想マシンのディスクイメージファイルの場所を判別します。以下に例を示します。

```
# virsh dombllist <vm-name>

Target Source
-----
vda    /home/username/disk-images/example-image.qcow2
```

2. オプション: 現在のディスクイメージをバックアップします。

```
# cp <example-image.qcow2> <example-image-backup.qcow2>
```

3. **qemu-img resize** ユーティリティーを使用して、イメージのサイズを変更します。たとえば、<example-image.qcow2>のサイズを 10 GB 増やすには、次のようにします。

```
# qemu-img resize <example-image.qcow2> +10G
```

4. ディスクイメージ内のファイルシステム、パーティション、または物理ボリュームのサイズを変更して、使用する領域を追加します。RHEL ゲストオペレーティングシステムでこれを行うには、[ストレージデバイスの管理](#) および [ファイルシステムの管理](#) の手順を使用します。

検証

1. サイズを変更したイメージに関する情報を表示し、意図したサイズになっているかを確認します。

```
# qemu-img info <converted-image.qcow2>

image: converted-image.qcow2
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
  compat: 1.1
  compression type: zlib
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
  extended l2: false
```

2. サイズを変更したディスクイメージに潜在的なエラーがないか確認します。手順については、[仮想ディスクイメージの整合性のチェック](#)を参照してください。

関連情報

- [qemu-img の man ページ](#)
- [ストレージデバイスの管理](#)
- [ファイルシステムの管理](#)

15.6.4. 仮想ディスクイメージの形式の変換

qemu-img convert コマンドを使用して、仮想ディスクイメージを別の形式に変換できます。たとえば、別のハイパーバイザーで実行している仮想マシンにディスクイメージをアタッチする場合、仮想ディスクイメージの形式の変換が必要になることがあります。

前提条件

- ディスクイメージを使用する仮想マシンがすべてシャットダウンしている。

手順

- **qemu-im convert** コマンドを使用して、既存の仮想ディスクイメージを別の形式に変換します。たとえば、**raw** ディスクイメージを **QCOW2** ディスクイメージに変換するには、次のようにします。

```
# qemu-img convert -f raw <original-image.img> -O qcow2 <converted-image.qcow2>
```

検証

1. 変換したイメージに関する情報を表示し、意図した形式とサイズであるかどうかを確認します。

```
# qemu-img info <converted-image.qcow2>

image: converted-image.qcow2
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
  compat: 1.1
  compression type: zlib
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
  extended l2: false
```

2. ディスクイメージに潜在的なエラーがないか確認します。手順については、[仮想ディスクイメージの整合性のチェック](#) を参照してください。

関連情報

- [仮想ディスクイメージの整合性のチェック](#)
- `qemu-img` の man ページ

15.7. WEB コンソールを使用した仮想マシンのストレージボリュームの管理

RHEL を使用すると、仮想マシンにストレージを割り当てるために使用されるストレージボリュームを管理できます。

RHEL Web コンソールを使用して以下を実行できます。

- [ストレージボリュームの作成](#)
- [ストレージボリュームの削除](#)

15.7.1. Web コンソールを使用したストレージボリュームの作成

機能している仮想マシンを作成するには、仮想マシンイメージと仮想マシン関連のデータを保存できるローカルストレージデバイスが仮想マシンに割り当てられている必要があります。ストレージプールにストレージボリュームを作成し、ストレージディスクとして仮想マシンに割り当てることができます。

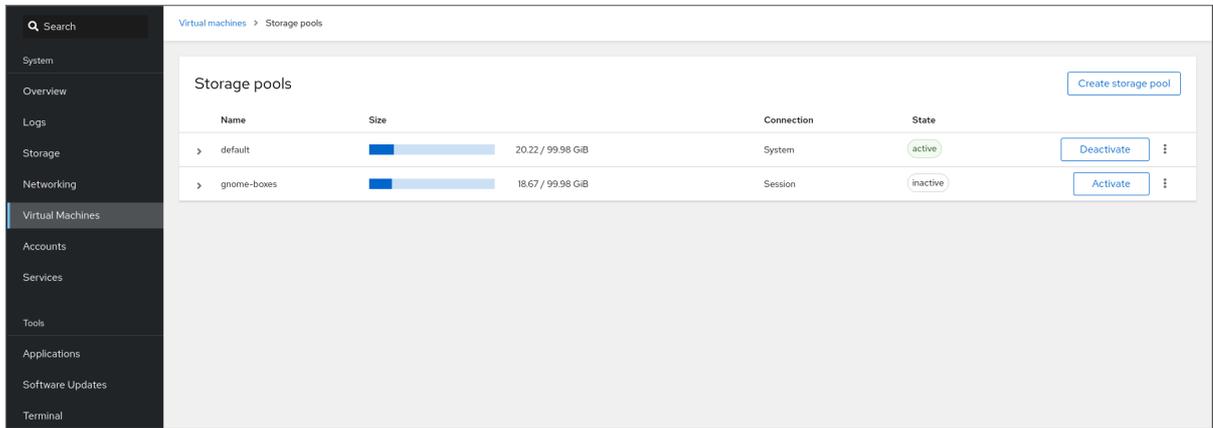
Web コンソールを使用してストレージボリュームを作成するには、以下の手順を参照してください。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

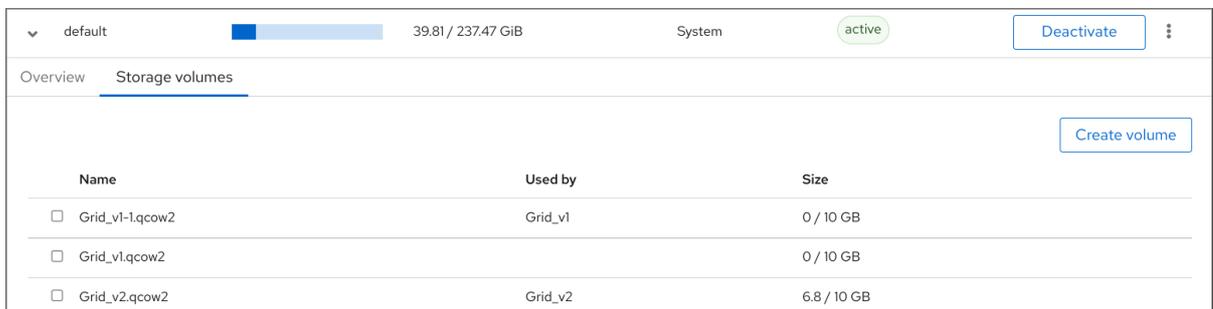
1. 仮想マシンタブの上部にある [ストレージプール](#) をクリックします。ストレージプール画面が表示され、設定されているストレージプールのリストが示されます。



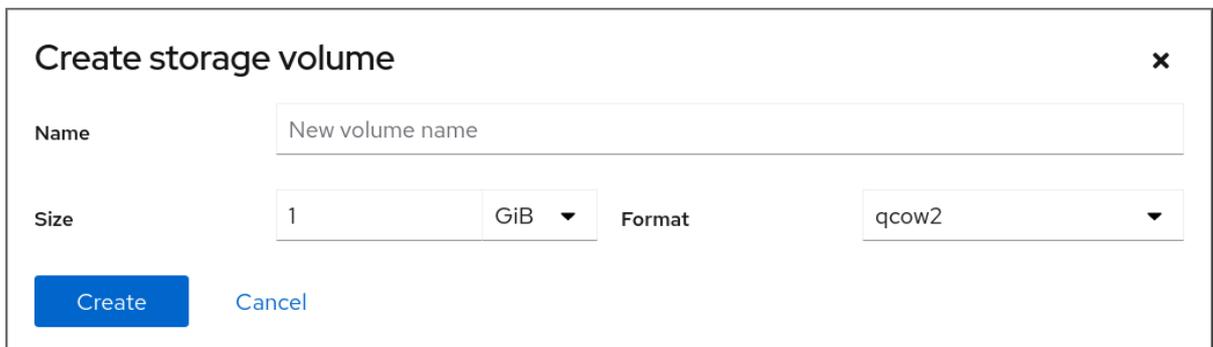
2. **ストレージプール** 画面で、ストレージボリュームを作成するストレージプールをクリックします。
行がデプロイメントされ、選択したストレージプールに関する基本情報を含む概要ペインが表示されます。



3. 拡張された行の概要タブの横にある **ストレージボリューム** をクリックします。
ストレージボリュームタブが表示され、既存のストレージボリュームが存在する場合は、そのボリュームに関する基本的な情報が表示されます。



4. **ボリュームの作成** をクリックします。
Create storage volume ダイアログが表示されます。



5. ストレージボリュームの作成ダイアログに、次の情報を入力します。
 - **名前** - ストレージボリュームの名前。

- **サイズ** - ストレージボリュームのサイズ (MiB または GiB)。
- **フォーマット** - ストレージボリュームの形式。サポートされているタイプは **qcow2** および **raw** です。

6. **Create** をクリックします。

ストレージボリュームが作成されると、ストレージボリュームの作成ダイアログが終了し、ストレージボリュームのリストに新しいストレージボリュームが表示されます。

関連情報

- [Understanding storage volumes](#)
- [Web コンソールを使用した仮想マシンへの新しいディスクの追加](#)

15.7.2. Web コンソールを使用したストレージボリュームの削除

ストレージプールの領域を解放するためにストレージボリュームを削除したり、仮想マシンが切断された場合に関連付けられたストレージ項目を削除したりできます。

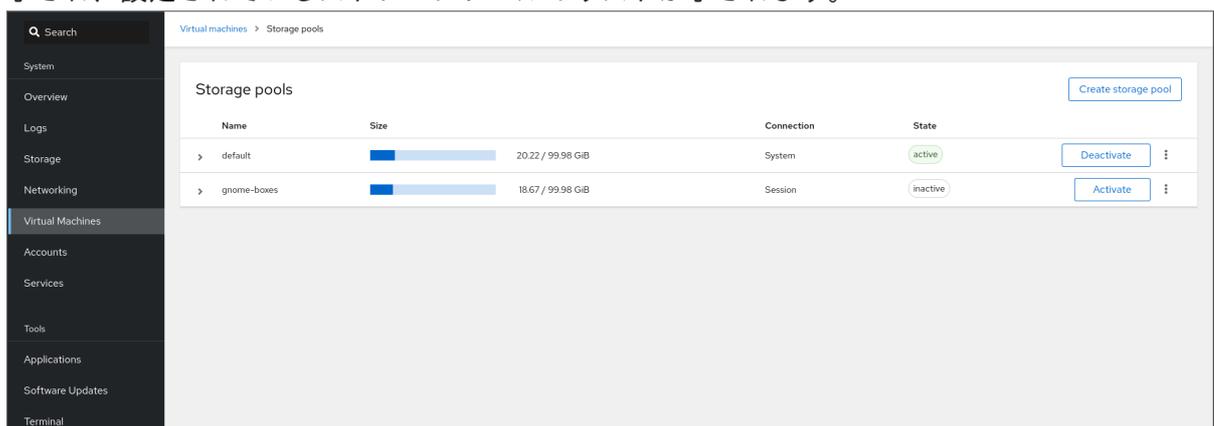
RHEL Web コンソールを使用してストレージボリュームを削除するには、以下の手順を参照してください。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 削除するストレージボリュームを使用する仮想マシンがすべてシャットダウンしている。

手順

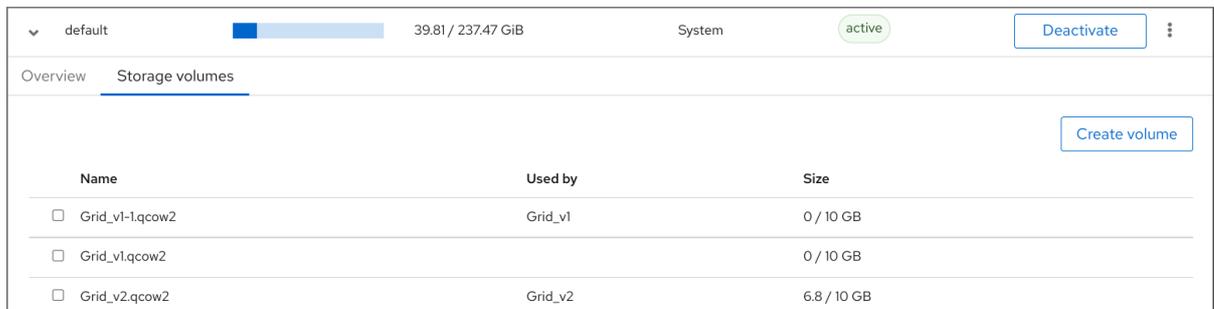
1. 仮想マシンタブの上部にある **ストレージプール** をクリックします。ストレージプール画面が表示され、設定されているストレージプールのリストが示されます。



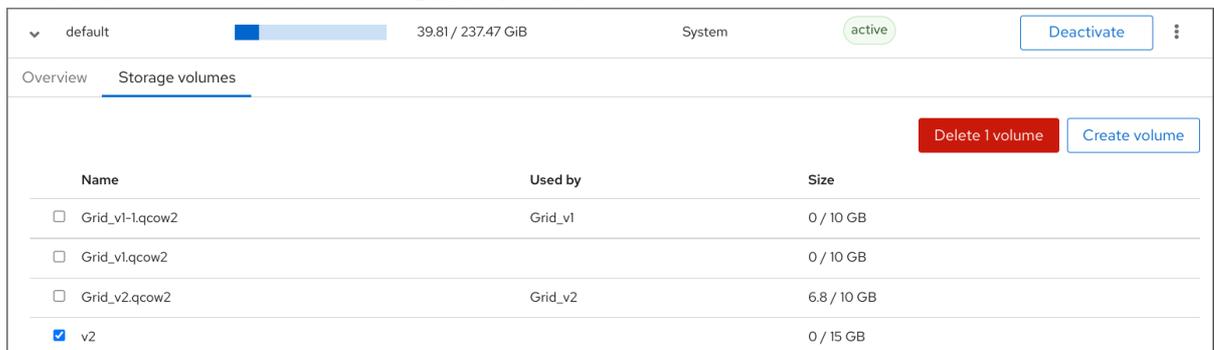
2. **ストレージプール** 画面で、ストレージボリュームを削除するストレージプールをクリックします。行がデプロイメントされ、選択したストレージプールに関する基本情報を含む概要ペインが表示されます。



3. 拡張された行の概要タブの横にある **ストレージボリューム** をクリックします。ストレージボリュームタブが表示され、既存のストレージボリュームが存在する場合は、そのボリュームに関する基本的な情報が表示されます。



4. 削除するストレージボリュームを選択します。



5. **1 ボリュームの削除** をクリックします。

関連情報

- [Understanding storage volumes](#)

15.8. WEB コンソールを使用した仮想マシンストレージディスクの管理

RHEL を使用すると、仮想マシンに接続されているストレージディスクを管理できます。

RHEL Web コンソールを使用して以下を実行できます。

- [仮想マシンのディスク情報の表示](#)
- [仮想マシンへの新しいディスクの追加](#)
- [仮想マシンへのディスクの割り当て](#)
- [仮想マシンからのディスクの切り離し](#)

15.8.1. Web コンソールで仮想マシンのディスク情報の表示

Web コンソールを使用して、選択した仮想マシンに割り当てられたディスクの詳細情報を表示できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. 情報を表示する仮想マシンをクリックします。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. ディスク までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

この情報には以下が含まれます。

- **デバイス** - ディスクのデバイスの種類。
- **使用済み** - 現在割り当てられているディスク容量。
- **容量** - ストレージボリュームの最大サイズ。
- **バス** - エミュレートされるディスクデバイスの種類。
- **アクセス** - ディスクが **書き込み可能** かどうか、**読み取り専用** であるか。**raw** ディスクの場合は、**書き込み可能および共有** へのアクセスを設定することもできます。
- **ソース** - ディスクデバイスまたはファイル

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

15.8.2. Web コンソールを使用した仮想マシンへの新しいディスクの追加

新しいディスクを仮想マシンに追加するには、RHEL 9 Web コンソールを使用して、新しいストレージボリュームを作成し、仮想マシンに割り当てます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、新しいディスクを作成して割り当てる仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ディスク** までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

3. **ディスクの追加** をクリックします。
ディスクの追加ダイアログが表示されます。

Add disk ✕

Source Create new Use existing Custom path

Pool

Name

Size Format

Persistence Always attach

[Show additional options](#)

4. **新規作成** オプションを選択します。
5. 新しいディスクを設定します。
 - **プール** - 仮想ディスクの作成元であるストレージプールを選択します。
 - **名前** - 作成する仮想ディスクの名前を入力します。
 - **サイズ** - 作成する仮想ディスクのサイズを入力し、単位 (MiB または GiB) を選択します。
 - **フォーマット** - 作成する仮想ディスクの形式を選択します。サポートされているタイプは **qcow2** および **raw** です。
 - **永続** - 選択すると、仮想ディスクが永続化されます。選択しないと、仮想ディスクは一時的になります。



注記

一時的なデバイスは、稼働中の仮想マシンにのみ追加できます。

- **追加オプション** - 仮想ディスクの追加設定を指定します。
 - **キャッシュ** - キャッシュメカニズムを選択します。
 - **バス** - エミュレートするディスクデバイスの種類を選択します。
 - **ディスク識別子** - マルチパスストレージの設定に使用できる、接続されているディスクの識別子を設定します。識別子は、特定のディスクシリアル番号にライセンスされた独自のソフトウェアを使用する場合にも役立ちます。

6. **Add** をクリックします。
仮想ディスクが作成され、仮想マシンに接続します。

関連情報

- [Web コンソールで仮想マシンのディスク情報の表示](#)
- [Web コンソールで既存ディスクを仮想マシンに割り当てる手順](#)
- [Web コンソールを使用した仮想マシンからのディスクの割り当て解除](#)

15.8.3. Web コンソールで既存ディスクを仮想マシンに割り当てる手順

Web コンソールを使用して、既存のストレージボリュームをディスクとして仮想マシンに割り当てることができます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、新しいディスクを作成して割り当てる仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ディスク** までスクロールします。
ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

3. **ディスクの追加** をクリックします。

ディスクの追加ダイアログが表示されます。

4. **既存の使用** ラジオボタンをクリックします。
ディスクの追加ダイアログに、適切な設定フィールドが表示されます。
5. 仮想マシンのディスクを設定します。
 - **プール** - 仮想ディスクを割り当てるストレージプールを選択します。
 - **ボリューム** - 割り当てるストレージボリュームを選択します。
 - **永続性**: 仮想マシンの実行中に利用できます。常に**接続** チェックボックスを選択して、仮想ディスクを永続化します。仮想ディスクを一時的にするには、チェックボックスをオフにします。
 - **追加オプション** - 仮想ディスクの追加設定を指定します。
 - **キャッシュ** - キャッシュメカニズムを選択します。
 - **バス** - エミュレートするディスクデバイスの種類を選択します。
 - **ディスク識別子** - マルチパスストレージの設定に使用できる、接続されているディスクの識別子を設定します。識別子は、特定のディスクシリアル番号にライセンスされた独自のソフトウェアを使用する場合にも役立ちます。
6. **追加** をクリックします。
選択した仮想ディスクが仮想マシンに割り当てられます。

関連情報

- [Web コンソールで仮想マシンのディスク情報の表示](#)
- [Web コンソールを使用した仮想マシンへの新しいディスクの追加](#)
- [Web コンソールを使用した仮想マシンからのディスクの割り当て解除](#)

15.8.4. Web コンソールを使用した仮想マシンからのディスクの割り当て解除

Web コンソールを使用して、仮想マシンからディスクの割り当てを解除できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、ディスクの割り当てを解除する仮想マシンを選択します。新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ディスク** までスクロールします。ディスクセクションには、仮想マシンに割り当てられたディスクに関する情報と、ディスクの **Add**、または **Edit** のオプションが表示されます。

Disks							Add disk
Device	Used	Capacity	Bus	Access	Source		
disk	8.9 GiB	10 GiB	virtio	Writeable	File	/var/lib/libvirt/images/Grid_v2.qcow2	Remove Edit
disk	0 GiB	15 GiB	virtio	Writeable	Pool	default	Remove Edit
					Volume	v2	

3. 切り離すディスクの行の右側にあるメニューボタン **⋮** をクリックします。
4. 表示されるドロップダウンメニューで、**Remove** ボタンをクリックします。**Remove disk from VM?** 確認ダイアログボックスが表示されます。
5. 確認ダイアログボックスで、**削除** をクリックします。オプションで、ディスクイメージも削除する場合は、**Remove and delete file** をクリックします。仮想マシンから、仮想ディスクの割り当てが解除されます。

関連情報

- [Web コンソールで仮想マシンのディスク情報の表示](#)
- [Web コンソールを使用した仮想マシンへの新しいディスクの追加](#)
- [Web コンソールで既存ディスクを仮想マシンに割り当てる手順](#)

15.9. LIBVIRT シークレットを使用した ISCSI ストレージプールのセキュリティー保護

ユーザー名とパスワードのパラメーターは、iSCSI ストレージプールをセキュリティー保護するため、**virsh** で設定できます。プールの定義前または後に設定できますが、認証設定を有効にするにはプールを起動する必要があります。

ここでは、**libvirt** シークレットを使用して、iSCSI ベースのストレージプールのセキュリティーを保護する手順を説明します。



注記

この手順は、iSCSI ターゲットの作成時に **user_ID** および **password** が定義される場合に必要です。

前提条件

- iSCSI ベースのストレージプールが作成されていることを確認します。詳細は、[CLI を使用した iSCSI ベースのストレージプールの作成](#) を参照してください。

手順

1. チャレンジハンドシェイク認証プロトコル (CHAP) のユーザー名を使用して、libvirt シークレットファイルを作成します。以下に例を示します。

```
<secret ephemeral='no' private='yes'>
  <description>Passphrase for the iSCSI example.com server</description>
  <usage type='iscsi'>
    <target>iscsirhel7secret</target>
  </usage>
</secret>
```

2. **virsh secret-define** コマンドを使用して、libvirt シークレットを定義します。

```
# virsh secret-define secret.xml
```

3. **virsh secret-list** コマンドで UUID を確認します。

```
# virsh secret-list
UUID                               Usage
-----
2d7891af-20be-4e5e-af83-190e8a922360  iscsi iscsirhel7secret
```

4. **virsh secret-set-value** コマンドを使用して、前の手順の出力の UUID に、シークレットを割り当てます。これにより、CHAP ユーザー名とパスワードが、libvirt が制御するシークレットリストにあることが保証されます。以下に例を示します。

```
# virsh secret-set-value --interactive 2d7891af-20be-4e5e-af83-190e8a922360
Enter new value for secret:
Secret value set
```

5. **virsh edit** コマンドを使用してストレージプールの XML ファイルに認証エントリーを追加し、**<auth>** 要素を追加して **authentication type**、**username**、および **secret usage** を指定します。以下に例を示します。

```
<pool type='iscsi'>
  <name>iscsirhel7pool</name>
  <source>
    <host name='192.0.2.1'>
      <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'>
        <auth type='chap' username='_example-user_'>
          <secret usage='iscsirhel7secret'>
        </auth>
      </source>
```

```
<target>
  <path>/dev/disk/by-path</path>
</target>
</pool>
```

注記

サブ要素 **<auth>** は仮想マシンの **<pool>** および **<disk>** XML 要素内の異なる場所に存在します。**<pool>** の場合は、**<auth>** が **<source>** 要素に指定されます。認証は一部のプールソース (iSCSI および RBD) のプロパティであるため、これはプールソースの検索場所を説明する要素となります。ドメインのサブ要素である **<disk>** の場合、iSCSI ディスクまたは RBD ディスクに対する認証は、ディスクのプロパティです。また、ディスクのサブ要素 **<auth>** は、ストレージプールのサブ要素とは異なります。

```
<auth username='redhat'>
  <secret type='iscsi' usage='iscsirhel7secret' />
</auth>
```

- 変更を有効にするには、ストレージプールを有効にします。プールがすでに起動している場合は、ストレージプールを停止して再起動します。

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

15.10. VHBA の作成

仮想ホストバスアダプター (vHBA) デバイスは、ホストシステムを SCSI デバイスに接続し、SCSI ベースのストレージプールを作成するために必要です。

vHBA デバイスは、XML 設定ファイルで定義することで作成できます。

手順

- virsh nodedev-list --cap vports** コマンドを使用して、ホストシステムの HBA を見つけます。以下の例は、vHBA に対応する HBA が 2 つ搭載されているホストを示しています。

```
# virsh nodedev-list --cap vports
scsi_host3
scsi_host4
```

- virsh nodedev-dumpxml HBA_device** コマンドを使用して、HBA の詳細を表示します。

```
# virsh nodedev-dumpxml scsi_host3
```

コマンドからの出力には、**<name>**、**<wwnn>**、および **<wwpn>** フィールドのリストが表示されます。これは、vHBA を作成するために使用されます。**<max_vports>** は、対応している vHBA の最大数を示します。以下に例を示します。

```
<device>
  <name>scsi_host3</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
```

```

<parent>pci_0000_10_00_0</parent>
<capability type='scsi_host'>
  <host>3</host>
  <unique_id>0</unique_id>
  <capability type='fc_host'>
    <wwnn>20000000c9848140</wwnn>
    <wwpn>10000000c9848140</wwpn>
    <fabric_wwn>2002000573de9a81</fabric_wwn>
  </capability>
  <capability type='vport_ops'>
    <max_vports>127</max_vports>
    <vports>0</vports>
  </capability>
</capability>
</device>

```

この例では、**<max_vports>** 値は、HBA 設定で使用できる合計 127 の仮想ポートがあることを示します。**<vports>** の値は、現在使用中の仮想ポートの数を示します。この値は、vHBA の作成後に更新されます。

3. vHBA ホスト用に、以下のいずれかの XML ファイルを作成します。この例では、ファイルの名前は **vhba_host3.xml** です。
次の例では、**scsi_host3** を使用して親 vHBA を説明します。

```

<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
      </capability>
    </capability>
  </device>

```

次の例では、WWNN/WWPN のペアを使用して親 vHBA を説明します。

```

<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140'>
    <capability type='scsi_host'>
      <capability type='fc_host'>
        </capability>
      </capability>
    </device>
  </device>

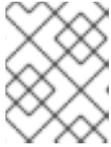
```



注記

WWNN および WWPN の値は、前の手順で確認した HBA の詳細の値と一致する必要があります。

<parent> フィールドは、この vHBA デバイスに関連付ける HBA デバイスを指定します。**<device>** タグの詳細は、ホスト用の新しい vHBA デバイスを作成するために、次の手順で使用されます。**nodedev** の XML 形式の詳細は、[アップストリームの libvirt ページ](#) を参照してください。



注記

virsh コマンドでは、**parent_wwnn** 属性、**parent_wwpn** 属性、または **parent_fabric_wwn** 属性を定義する方法が提供されません。

4. **virsh nodedev-create** コマンドを使用して、前の手順で作成した XML ファイルに基づいて VHBA を作成します。

```
# virsh nodedev-create vhba_host3
Node device scsi_host5 created from vhba_host3.xml
```

検証

- **virsh nodedev-dumpxml** コマンドで、新しい vHBA の詳細 (scsi_host5) を確認します。

```
# virsh nodedev-dumpxml scsi_host5
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

関連情報

- [CLI で vHBA デバイスを使用した SCSI ベースのストレージプールを作成する手順](#)

第16章 仮想マシンでの GPU デバイスの管理

RHEL 9 ホストで仮想マシンのグラフィカルパフォーマンスを向上させるために、仮想マシンにホスト GPU を割り当てることができます。

- ホストから GPU を取り外し、GPU の完全な制御を仮想マシンに直接渡すことができます。
- 物理 GPU から複数の仲介デバイスを作成し、これらのデバイスを仮想 GPU (vGPU) として複数のゲストに割り当てることができます。現在、これは選択した NVIDIA GPU でのみ対応しており、1つのゲストに割り当てることができる仲介デバイスは1つだけです。



重要

GPU の割り当ては現在、Intel 64 システムおよび AMD64 システムでのみサポートされています。

16.1. 仮想マシンへの GPU の割り当て

ホストシステムに接続されている GPU にアクセスして制御するには、GPU の直接制御を仮想マシンに渡すようにホストシステムを設定する必要があります。



注記

仮想 GPU の割り当て方法の詳細は、[Managing NVIDIA vGPU devices](#) を参照してください。

前提条件

- ホストマシンカーネルで IOMMU サポートを有効にする必要があります。
 - Intel ホストでは、VT-d を有効にする必要があります。
 1. **intel_iommu=on** および **iommu=pt** パラメーターを使用して GRUB 設定を再生成します。


```
# grubby --args="intel_iommu=on iommu_pt" --update-kernel DEFAULT
```
 2. ホストを再起動します。
 - AMD ホストでは、AMD-Vi を有効にする必要があります。AMD ホストでは、IOMMU はデフォルトで有効になっているため、**iommu=pt** を追加してパススルーモードに切り替えることができます。
 1. **iommu=pt** パラメーターで GRUB 設定を再生成します。

```
# grubby --args="iommu=pt" --update-kernel DEFAULT
```



注記

pt オプションは、パススルーモードで使用されるデバイスにのみ IOMMU を有効にし、ホストパフォーマンスを向上させます。ただし、すべてのハードウェアがこのオプションに対応しているわけではありません。このオプションが有効になっていない場合でも、デバイスを割り当てることは可能です。

2. ホストを再起動します。

手順

1. ドライバーが GPU にバインドしないようにします。

- a. GPU の接続先である PCI バスアドレスを特定します。

```
# lspci -Dnn | grep VGA
0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL [Quadro
K4000] [10de:11fa] (rev a1)
```

- b. ホストのグラフィックドライバーが GPU を使用しないようにします。これには、pci-stub ドライバーで GPU の PCI ID を使用します。

たとえば、次のコマンドは、ドライバーが **10de: 11fa** バスに接続されている GPU にバインドしないようにします。

```
# grubby --args="pci-stub.ids=10de:11fa" --update-kernel DEFAULT
```

- c. ホストを再起動します。

2. **オプション:** サポートの制限により、オーディオなどの特定の GPU 機能が仮想マシンに渡せない場合は、IOMMU グループ内のエンドポイントのドライバーバインドを変更して、必要な GPU 機能のみを通過させることができます。

- a. GPU 設定を XML に変換し、ホストドライバーに接続しないようにするエンドポイントの PCI アドレスを書き留めておきます。

これを行うには、アドレスに **pci_** 接頭辞を追加し、区切り文字をアンダースコアに変換することにより、GPU の PCI バスアドレスを libvirt 互換形式に変換します。

たとえば、次のコマンドは、**0000:02:00.0** バスアドレスに接続されている GPU の XML 設定を表示します。

```
# virsh nodedev-dumpxml pci_0000_02_00_0

<device>
  <name>pci_0000_02_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
  <parent>pci_0000_00_03_0</parent>
  <driver>
    <name>pci-stub</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>2</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x11fa'>GK106GL [Quadro K4000]</product>
    <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <iommuGroup number='13'>
      <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'>
        <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'>
      </iommuGroup>
    <pci-express>
      <link validity='cap' port='0' speed='8' width='16'>
```

```
<link validity='sta' speed='2.5' width='16'/>
</pci-express>
</capability>
</device>
```

- b. エンドポイントがホストドライバーに接続されないようにします。
この例では、GPU を仮想マシンに割り当て、オーディオ機能である **<address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>** に対応するエンドポイントがホストオーディオドライバーに接続されないようにし、代わりにエンドポイントを VFIO-PCI に接続します。

```
# driverctl set-override 0000:02:00.1 vfio-pci
```

3. GPU の仮想マシンへの接続

- a. PCI バスアドレスを使用して GPU 用の XML 設定ファイルを作成します。
たとえば、GPU のバスアドレスからパラメーターを使用して、次の XML ファイル GPU-Assign.xml を作成できます。

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
  </source>
</hostdev>
```

- b. ホストシステムにファイルを保存します。
c. ファイルを仮想マシンの XML 設定とマージします。
たとえば、次のコマンドは、GPU XML ファイルの GPU-Assign.xml を、**System1** 仮想マシンの XML 設定ファイルにマージします。

```
# virsh attach-device System1 --file /home/GPU-Assign.xml --persistent
Device attached successfully.
```



注記

GPU は、セカンダリーグラフィックスデバイスとして仮想マシンに接続されています。GPU をプライマリーグラフィックスデバイスとして割り当てることには対応していません。Red Hat では、仮想マシンの XML 設定で、エミュレートしているプライマリーグラフィックスデバイスを削除することは推奨しません。

検証

- デバイスが、仮想マシンの XML 設定の **<devices>** セクションに表示されます。詳細は、[Sample virtual machine XML configuration](#) を参照してください。

既知の問題

- 仮想マシンに接続できる GPU の数は、割り当てられた PCI デバイスの最大数 (RHEL 9 では現在 64) によって制限されます。ただし、仮想マシンに複数の GPU を接続すると、ゲストのメモリーマップド I/O (MMIO) で問題が発生する可能性があり、その結果、GPU が仮想マシンで

使用できなくなる可能性があります。

これらの問題を回避するには、より大きな 64 ビット MMIO 空間を設定し、vCPU 物理アドレスビットを設定して、拡張された 64 ビット MMIO 空間をアドレス指定可能にします。

- 現在 RHEL 9 ゲストオペレーティングシステムを使用している仮想マシンに NVIDIA GPU デバイスを接続すると、その仮想マシンで Wayland セッションが無効になり、代わりに Xorg セッションが読み込まれます。これは、NVIDIA ドライバーと Wayland の間の非互換性が原因です。

16.2. NVIDIA vGPU デバイスの管理

vGPU 機能により、**仲介デバイス**として参照される物理的な NVIDIA GPU デバイスを複数の仮想デバイスに分割できます。この仲介デバイスは、仮想 GPU として複数の仮想マシンに割り当てることができます。これにより、この仮想マシンが、1つの物理 GPU のパフォーマンスを共有できます。



重要

仲介デバイスの使用にかかわらず、仮想マシンに物理 GPU を割り当てると、ホストが GPU を使用できなくなります。

16.2.1. NVIDIA vGPU デバイスの設定

NVIDIA の vGPU 機能を設定するには、GPU デバイスの NVIDIA vGPU ドライバーをダウンロードして、仲介デバイスを作成し、使用する仮想マシンに割り当てする必要があります。詳細な手順は、以下を参照してください。

前提条件

- GPU が vGPU 仲介デバイスをサポートしている。vGPU の作成をサポートする NVIDIA GPU の最新のリストについては、[NVIDIA vGPU ソフトウェアのドキュメント](#)を参照してください。
- ホストが使用している GPU が分からない場合は、**lshw** パッケージをインストールして、**lshw -C display** コマンドを使用します。以下の例は、システムが、vGPU と互換性がある NVIDIA Tesla P4 GPU を使用していることを示しています。

```
# lshw -C display
*-display
  description: 3D controller
  product: GP104GL [Tesla P4]
  vendor: NVIDIA Corporation
  physical id: 0
  bus info: pci@0000:01:00.0
  version: a1
  width: 64 bits
  clock: 33MHz
  capabilities: pm msi pciexpress cap_list
  configuration: driver=vfio-pci latency=0
  resources: irq:16 memory:f6000000-f6ffffff memory:e0000000-efffffff
  memory:f0000000-f1ffffff
```

手順

1. NVIDIA vGPU ドライバーをダウンロードして、システムにインストールします。手順は [NVIDIA ドキュメント](#) を参照してください。
2. NVIDIA ソフトウェアのインストーラーで `/etc/modprobe.d/nvidia-installer-disable-nouveau.conf` ファイルが作成されなかった場合は、`/etc/modprobe.d/` に任意の名前で `conf` ファイルを作成し、そのファイルに以下の行を追加します。

```
blacklist nouveau
options nouveau modeset=0
```

3. 現在のカーネル用に初期 ramdisk を再生成してから再起動します。

```
# dracut --force
# reboot
```

4. カーネルで `nvidia_vgpu_vfio` モジュールが読み込まれていること、`nvidia-vgpu-mgr.service` サービスが実行されていることを確認してください。

```
# lsmod | grep nvidia_vgpu_vfio
nvidia_vgpu_vfio 45011 0
nvidia 14333621 10 nvidia_vgpu_vfio
mdev 20414 2 vfio_mdev,nvidia_vgpu_vfio
vfio 32695 3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1

# systemctl status nvidia-vgpu-mgr.service
nvidia-vgpu-mgr.service - NVIDIA vGPU Manager Daemon
  Loaded: loaded (/usr/lib/systemd/system/nvidia-vgpu-mgr.service; enabled; vendor preset: disabled)
  Active: active (running) since Fri 2018-03-16 10:17:36 CET; 5h 8min ago
  Main PID: 1553 (nvidia-vgpu-mgr)
  [...]
```

さらに、NVIDIA Ampere GPU デバイスに基づいて vGPU を作成する場合は、物理 GPU で仮想機能が有効になっていることを確認してください。手順は [NVIDIA ドキュメント](#) を参照してください。

5. デバイスの UUID を生成します。

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
```

6. 検出された GPU ハードウェアに基づいて、仲介されたデバイスの設定を含む XML ファイルを準備します。たとえば、次の例では、0000:01:00.0 PCI バスで実行され、前の手順で生成された UUID を使用する NVIDIA Tesla P4 カードで `nvidia-63` vGPU タイプの仲介デバイスを設定します。

```
<device>
  <parent>pci_0000_01_00_0</parent>
  <capability type="mdev">
    <type id="nvidia-63"/>
    <uuid>30820a6f-b1a5-4503-91ca-0c10ba58692a</uuid>
  </capability>
</device>
```

7. 準備した XML ファイルに基づいて vGPU 仲介デバイスを定義します。以下に例を示します。

```
# virsh nodedev-define vgpu-test.xml
Node device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 created
from vgpu-test.xml
```

8. オプション: 仲介デバイスが非アクティブとしてリストされていることを確認します。

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

9. 作成した vGPU 仲介デバイスを起動します。

```
# virsh nodedev-start mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 started
```

10. オプション: 仲介デバイスがアクティブとしてリストされていることを確認します。

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

11. ホストの再起動後に自動的に起動するように vGPU デバイスを設定します。

```
# virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_d196754e_d8ed_4f43_bf22_684ed698b08b_0000_9b_00_0 marked as
autostarted
```

12. vGPU リソースを共有する仮想マシンに仲介デバイスを割り当てます。これを行うには、以下の行を、仮想マシンの XML 設定の `<devices/>` セクションに追加します。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a' />
  </source>
</hostdev>
```

各 UUID は、一度に1つの仮想マシンにしか割り当てることができないのでご注意ください。さらに、仮想マシンに **virtio-vga** などの QEMU ビデオデバイスがない場合は、`<hostdev>` 行に **ramfb='on'** パラメーターも追加します。

13. 割り当てられた仮想マシンに vGPU 仲介デバイスの全機能が提供されるように、これらの仮想マシンに NVIDIA vGPU ゲストソフトウェアのライセンスを設定します。詳細および手順は、[NVIDIA の仮想 GPU ソフトウェアのライセンスサーバーユーザーガイド](#) を参照してください。

検証

1. 作成した vGPU の機能をクエリーし、アクティブで永続的な機能としてリストされていることを確認します。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:          virsh nodedev-autostart
```

```
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:      pci_0000_01_00_0
Active:      yes
Persistent:  yes
Autostart:   yes
```

- 仮想マシンを起動し、ゲストオペレーティングシステムが仲介デバイスを NVIDIA GPU として検出することを確認します。たとえば、仮想マシンが Linux を使用している場合は、以下のとおりとなります。

```
# lspci -d 10de: -k
07:00.0 VGA compatible controller: NVIDIA Corporation GV100GL [Tesla V100 SXM2 32GB]
(rev a1)
    Subsystem: NVIDIA Corporation Device 12ce
    Kernel driver in use: nvidia
    Kernel modules: nouveau, nvidia_drm, nvidia
```

既知の問題

- 現在 RHEL 9 ゲストオペレーティングシステムを使用している仮想マシンに NVIDIA vGPU 仲介デバイスを割り当てると、その仮想マシンで Wayland セッションが無効になり、代わりに Xorg セッションが読み込まれます。これは、NVIDIA ドライバーと Wayland の間の非互換性が原因です。

関連情報

- [NVIDIA vGPU ソフトウェアのドキュメント](#)
- `man virsh` コマンド

16.2.2. NVIDIA vGPU デバイスの削除

[割り当てられた vGPU 仲介デバイス](#) の設定を変更する場合は、割り当てられた仮想マシンから既存のデバイスを削除する必要があります。手順は、以下を参照してください。

前提条件

- デバイスを削除する仮想マシンがシャットダウンしている。

手順

- 削除する仲介デバイスの ID を取得します。

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

- vGPU 仲介デバイスの実行中のインスタンスを停止します。

```
# virsh nodedev-destroy mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Destroyed node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

- オプション:** 仲介デバイスが非アクティブであることを確認します。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:      virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:    pci_0000_01_00_0
Active:    no
Persistent: yes
Autostart: yes
```

4. 仮想マシンの XML 設定からデバイスを削除します。これには、**virsh edit** ユーティリティーを使用して仮想マシンの XML 設定を編集し、mdev の設定セグメントを削除します。このセグメントは、以下ようになります。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

仲介デバイスを停止してデタッチしても、このデバイスは削除されずに **定義された** とおりに保持されるのでご注意ください。したがって、デバイスを **再起動** して、別の仮想マシンに **割り当てる** ことができます。

5. **オプション:** 停止した仲介デバイスを削除するには、デバイスの定義を削除します。

```
# virsh nodedev-undefine
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Undefined node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

検証

- デバイスを停止して切り離しただけの場合は、仲介デバイスが非アクティブとしてリストされていることを確認してください。

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

- デバイスも削除した場合は、次のコマンドでデバイスが表示されないことを確認してください。

```
# virsh nodedev-list --cap mdev
```

関連情報

- **man virsh** コマンド

16.2.3. システムに関する NVIDIA vGPU 情報の取得

利用可能な vGPU 機能の機能を評価するには、お使いのシステムの仲介デバイスに関する以下のような追加情報を取得してください。

- 特定タイプの仲介デバイスを何個作成できるか
- お使いのシステムに設定済みの仲介デバイスはどれか

手順

- vGPU 仲介デバイスをサポートできるホストで使用可能な GPU デバイスを確認するには、**virsh nodedev-list--capmdev_types** コマンドを使用します。たとえば、以下は 2 つの NVIDIA Quadro RTX6000 デバイスを備えたシステムを示しています。

```
# virsh nodedev-list --cap mdev_types
pci_0000_5b_00_0
pci_0000_9b_00_0
```

- 特定の GPU デバイスでサポートされている vGPU タイプと追加のメタデータを表示するには、**virsh nodedev-dumpxml** コマンドを使用します。

```
# virsh nodedev-dumpxml pci_0000_9b_00_0
<device>
  <name>pci_0000_9b_00_0</name>
  <path>/sys/devices/pci0000:9a/0000:9a:00.0/0000:9b:00.0</path>
  <parent>pci_0000_9a_00_0</parent>
  <driver>
    <name>nvidia</name>
  </driver>
  <capability type='pci'>
    <class>0x030000</class>
    <domain>0</domain>
    <bus>155</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x1e30'>TU102GL [Quadro RTX 6000/8000]</product>
    <vendor id='0x10de'>NVIDIA Corporation</vendor>
    <capability type='mdev_types'>
      <type id='nvidia-346'>
        <name>GRID RTX6000-12C</name>
        <deviceAPI>vfio-pci</deviceAPI>
        <availableInstances>2</availableInstances>
      </type>
      <type id='nvidia-439'>
        <name>GRID RTX6000-3A</name>
        <deviceAPI>vfio-pci</deviceAPI>
        <availableInstances>8</availableInstances>
      </type>
      [...]
      <type id='nvidia-440'>
        <name>GRID RTX6000-4A</name>
        <deviceAPI>vfio-pci</deviceAPI>
        <availableInstances>6</availableInstances>
      </type>
      <type id='nvidia-261'>
        <name>GRID RTX6000-8Q</name>
        <deviceAPI>vfio-pci</deviceAPI>
        <availableInstances>3</availableInstances>
      </type>
    </capability>
  <iommuGroup number='216'>
    <address domain='0x0000' bus='0x9b' slot='0x00' function='0x3' />
    <address domain='0x0000' bus='0x9b' slot='0x00' function='0x1' />
    <address domain='0x0000' bus='0x9b' slot='0x00' function='0x2' />
  </iommuGroup>
</device>
```

```
<address domain='0x0000' bus='0x9b' slot='0x00' function='0x0'/>
</iommuGroup>
<numa node='2'/>
<pci-express>
  <link validity='cap' port='0' speed='8' width='16'/>
  <link validity='sta' speed='2.5' width='8'/>
</pci-express>
</capability>
</device>
```

関連情報

- **man virsh** コマンド

16.2.4. NVIDIA vGPU のリモートデスクトップストリーミングサービス

次のリモートデスクトップストリーミングサービスは、NVIDIA vGPU または NVIDIA GPU パススルーが有効になっている RHEL 9 ハイパーバイザーでサポートされています。

- HP ZCentral Remote Boost/Teradici
- NICE DCV
- Mechdyne TGX

サポートの詳細については、適切なベンダーサポートマトリックスを参照してください。

16.2.5. 関連情報

- [NVIDIA vGPU ソフトウェアのドキュメント](#)

第17章 仮想マシンのネットワーク接続の設定

ホストや、ホスト上の他の仮想マシン、外部ネットワークの場所に、ネットワーク経由で仮想マシンを接続する場合には、仮想マシンネットワークをそれぞれに合わせて設定する必要があります。仮想マシンのネットワークを提供するために、RHEL 9 ハイパーバイザーおよび新規に作成された仮想マシンには、デフォルトのネットワーク設定があります。これは、さらに変更することもできます。以下に例を示します。

- 仮想マシンがホストと同じネットワーク上にあるかのように、ホスト上の仮想マシンを検出し、ホスト外の場所に接続できます。
- 仮想マシンを受信ネットワークトラフィックから部分的に分離するか、完全に分離して、セキュリティを強化し、仮想マシンに影響を及ぼすリスクを最小限に抑えることができます。

以下のセクションでは、仮想マシンネットワーク設定と、選択した仮想マシンネットワークオプションの設定について説明します。

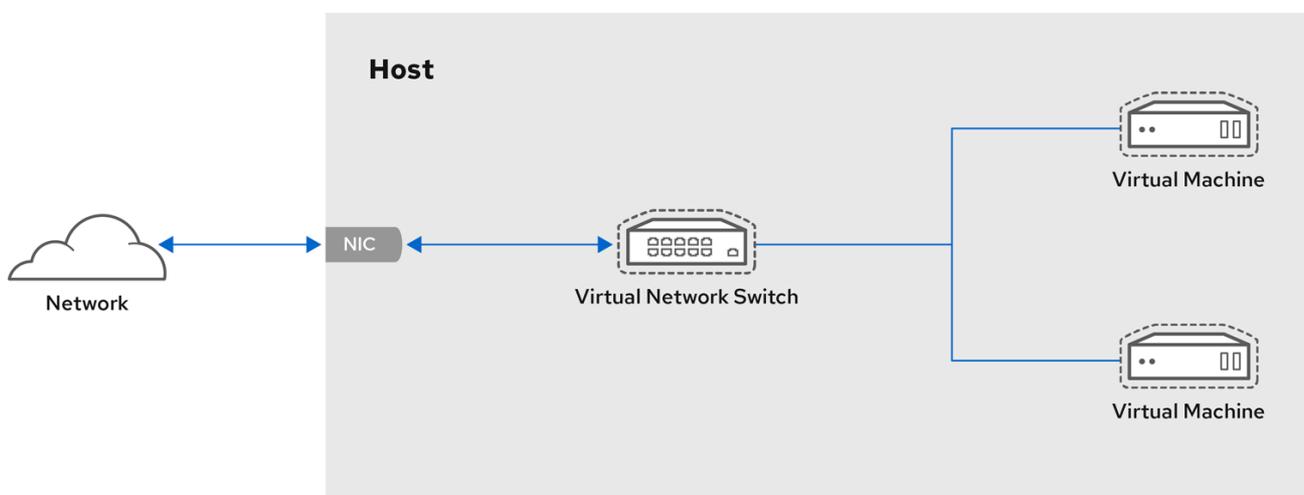
17.1. 仮想ネットワークの概要

ホストハードウェアにより、ネットワーク上の他のデバイスや場所への仮想マシンの接続が容易になります。以下のセクションでは、仮想マシンのネットワーク接続のメカニズムや、デフォルトの仮想マシンのネットワーク設定について説明します。

17.1.1. 仮想ネットワークの仕組み

仮想ネットワークは、仮想ネットワークスイッチの概念を使用します。仮想ネットワークスイッチは、ホストマシンで動作するソフトウェア設定です。仮想マシンは、仮想ネットワークスイッチを介してネットワークに接続します。仮想スイッチの設定に基づいて、仮想マシンはハイパーバイザーによって管理される既存の仮想ネットワーク、または別のネットワーク接続メソッドを使用できます。

以下の図は、2つの仮想マシンをネットワークに接続する仮想ネットワークスイッチを示しています。



RHEL_52_1219

ゲストオペレーティングシステムの視点から見ると、仮想ネットワーク接続は物理ネットワーク接続と同じです。ホストのマシンサーバーは、仮想ネットワークスイッチをネットワークインターフェイスとして表示します。**virtnetworkd** サービスを最初にインストールして起動すると、仮想マシンのデフォルトのネットワークインターフェイスである **virbr0** が作成されます。

このインターフェイスに関する情報を表示するには、ホストで **ip** ユーティリティを使用します。

-

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.0.2.255 scope global virbr0
```

デフォルトでは、1台のホストにあるすべての仮想マシンが、**virbr0** インターフェイスを使用する **default** という名前の同じ **NAT タイプ** の仮想ネットワークに接続されています。詳細は、[Virtual networking default configuration](#) を参照してください。

仮想マシンからの基本的なアウトバウンドのみのネットワークアクセスでは、デフォルトのネットワークが **libvirt-daemon-config-network** パッケージと一緒にインストールされ、**virtnetworkd** サービスが起動すると自動的に開始するため、通常は追加のネットワーク設定は必要ありません。

別の仮想マシンのネットワーク機能が必要な場合は、仮想ネットワークおよびネットワークインターフェイスを追加で作成し、仮想マシンがその機能を使用するように設定できます。デフォルトの NAT に加えて、これらのネットワークとインターフェイスは以下のいずれかのモードを使用するように設定できます。

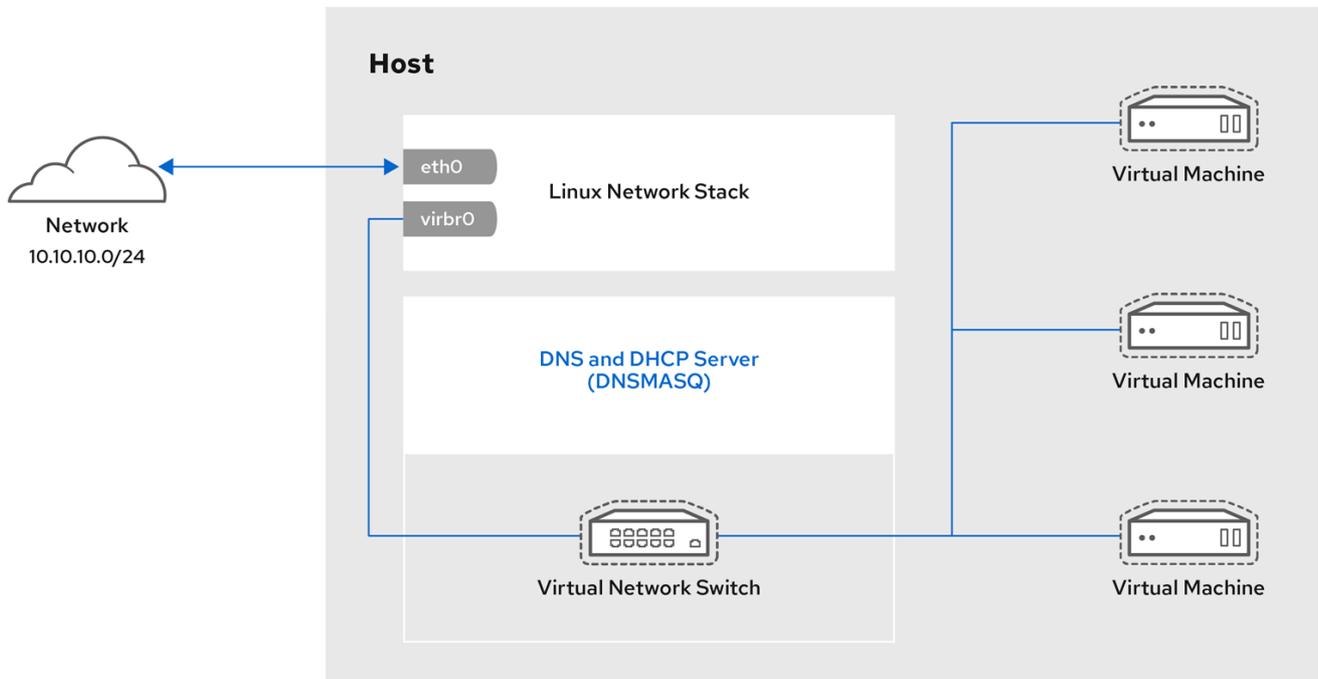
- ルーティングモード
- ブリッジモード
- 分離モード
- オープンモード

17.1.2. デフォルトの仮想ネットワーク設定

仮想化ホストに **virtnetworkd** サービスが最初にインストールされる際、これにはネットワークアドレス変換 (NAT) モードの仮想ネットワークの初期設定が含まれます。デフォルトでは、1台のホストにあるすべての仮想マシンが、同じ **libvirt** 仮想ネットワーク (**default**) に接続されています。このネットワーク上の仮想マシンは、ホスト上およびホスト外のネットワーク上の場所の両方に接続できますが、以下の制限があります。

- ネットワーク上の仮想マシンはホストと、ホスト上の他の仮想マシンに表示されますが、ネットワークトラフィックはゲストオペレーティングシステムのネットワークスタックのファイアウォールと、ゲストインターフェイスに接続されている **libvirt** ネットワークフィルタールールの影響を受けます。
- ネットワーク上の仮想マシンは、ホスト外の場所に接続可能ですが、表示されません。送信トラフィックは NAT ルールおよびホストシステムのファイアウォールの影響を受けます。

以下の図は、仮想マシンのデフォルトのネットワーク設定を示しています。



RHEL_52_1219

17.2. WEB コンソールで仮想マシンのネットワークインターフェイスの管理

RHEL 9 Web コンソールを使用して、Web コンソールが接続している仮想マシンの仮想ネットワークインターフェイスを管理できます。これにより、以下が可能になります。

- ネットワークインターフェイスに関する情報を表示および編集
- 仮想マシンへのネットワークインターフェイスの追加 および インターフェイスの切断または削除

17.2.1. Web コンソールで仮想ネットワークインターフェイス情報の表示および編集

RHEL 9 Web コンソールを使用して、選択した仮想マシンで仮想ネットワークインターフェイスを表示および変更することができます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

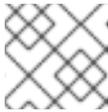
手順

1. **仮想マシン インターフェイス** で、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ネットワークインターフェイス** までスクロールします。
ネットワークインターフェイスセクションには、仮想マシンに設定された仮想ネットワークインターフェイスに関する情報と、ネットワークインターフェイスの **追加**、**削除**、**編集**、または **アンプラグ** のオプションが表示されます。

Network interfaces						Add network interface
Type	Model type	MAC address	IP address	Source	State	
network	virtio	52:54:00	inet 192.168.122.9/24	default	up	Delete Unplug Edit

この情報には以下が含まれます。

- **種類** - 仮想マシンのネットワークインターフェイスの種類。タイプには、仮想ネットワーク、LAN へのブリッジ、および直接割り当てが含まれます。



注記

RHEL 9 以降では、汎用イーサネット接続はサポートされていません。

- **モデルタイプ** - 仮想ネットワークインターフェイスのモデル。
 - **MAC アドレス** - 仮想ネットワークインターフェイスの MAC アドレス。
 - **IP アドレス** - 仮想ネットワークインターフェイスの IP アドレス。
 - **ソース** - ネットワークインターフェイスのソース。これはネットワークの種類によって異なります。
 - **状態** - 仮想ネットワークインターフェイスの状態。
3. 仮想ネットワークインターフェイスの設定を編集するには、**編集** をクリックします。仮想ネットワークインターフェイスの設定ダイアログが開きます。

52:54:00:b4:2a:62 virtual network interface settings ✕

Interface type ⓘ

Source

Model

MAC address

[Save](#) [Cancel](#)

4. インターフェイスの種類、ソース、モデル、または MAC アドレスを変更します。
5. **Save** をクリックします。ネットワークインターフェイスが変更しました。



注記

仮想ネットワークインターフェイス設定の変更は、仮想マシンを再起動しないと有効になりません。

また、MAC アドレスは、仮想マシンがシャットダウンしている場合にのみ変更できます。

関連情報

- [Web コンソールを使用した仮想マシン情報の表示](#)

17.2.2. Web コンソールでの仮想ネットワークインターフェイスの追加および接続

RHEL 9 Web コンソールを使用して、仮想ネットワークインターフェイスを作成し、これに仮想マシンを接続できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ネットワークインターフェイス** までスクロールします。
ネットワークインターフェイスセクションには、仮想マシンに設定された仮想ネットワークインターフェイスに関する情報と、ネットワークインターフェイスの**追加**、**編集**、または **プラグ** のオプションが表示されます。
3. 接続する仮想ネットワークインターフェイスの行の **プラグ** をクリックします。
選択した仮想ネットワークインターフェイスが仮想マシンに接続します。

17.2.3. Web コンソールでの仮想ネットワークインターフェイスの切断および削除

RHEL 9 Web コンソールを使用して、選択した仮想マシンに接続した仮想ネットワークインターフェイスの接続を解除できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. **ネットワークインターフェイス** までスクロールします。
ネットワークインターフェイスセクションには、仮想マシンに設定された仮想ネットワークインターフェイスに関する情報と、ネットワークインターフェイスの**追加**、**削除**、**編集**、または **アンプラグ** のオプションが表示されます。

Network interfaces						Add network interface
Type	Model type	MAC address	IP address	Source	State	
network	virtio	52:54:00	inet 192.168.122.9/24	default	up	Delete Unplug Edit

3. 切断する仮想ネットワークインターフェイスの行で **アンプラグ** をクリックします。選択した仮想ネットワークインターフェイスが仮想マシンから切断されます。

17.3. 推奨される仮想マシンネットワーク設定

多くのシナリオでは、デフォルトの仮想マシンのネットワーク設定だけで十分です。ただし、設定の調整が必要な場合は、コマンドラインインターフェイス (CLI) または RHEL 9 Web コンソールを使用して調整できます。次のセクションでは、このような状況での仮想マシンのネットワーク設定を一部説明します。

17.3.1. コマンドラインインターフェイスを使用した外部に表示される仮想マシンの設定

デフォルトでは、新規作成された仮想マシンは、NAT タイプのネットワークに接続されます。このネットワークは、ホストのデフォルトの仮想ブリッジである **virbr0** を使用します。これにより、仮想マシンはホストのネットワークインターフェイスコントローラー (NIC) を使用して外部ネットワークに接続できますが、外部システムから仮想マシンには到達できません。

仮想マシンをハイパーバイザーと同じ外部ネットワークに表示する必要がある場合は、代わりに **ブリッジモード** を使用する必要があります。これには、仮想マシンを、ハイパーバイザーの物理ネットワークデバイスに接続されているブリッジデバイスに割り当てます。コマンドラインインターフェイスを使用するには、以下の手順に従います。

前提条件

- デフォルトの NAT 設定を持つ **既存の仮想マシン** のシャットダウン。
- ハイパーバイザーの IP 設定。これは、ホストのネットワーク接続によって異なります。たとえば、以下の手順では、イーサネットケーブルを使用してホストがネットワークに接続され、ホストの物理 NIC MAC アドレスが DHCP サーバーの静的 IP に割り当てられるシナリオを使用します。したがって、イーサネットインターフェイスはハイパーバイザー IP として扱われます。イーサネットインターフェイスの IP 設定を取得するには、**ip addr** ユーティリティーを使用します。

```
# ip addr
[...]
enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s25
```

手順

1. ホスト上の物理インターフェイスのブリッジ接続を作成して設定します。手順は、**ネットワークブリッジの設定** を参照してください。静的 IP 割り当てが使用されるシナリオでは、物理イーサネットインターフェイスの IPv4 設定をブリッジインターフェイスに移行する必要があることに注意してください。
2. 作成したブリッジインターフェイスを使用するように仮想マシンのネットワークを変更します。たとえば、以下のコマンドは、**bridge0** を使用するように **testguest** を設定します。

```
# virt-xml testguest --edit --network bridge=bridge0
Domain 'testguest' defined successfully.
```

3. 仮想マシンを起動します。

```
# virsh start testguest
```

4. ゲストオペレーティングシステムで、仮想マシンがハイパーバイザーと同じネットワーク内の別の物理システムであるかのように、システムのネットワークインターフェイスの IP および DHCP 設定を調整します。

これに関する具体的な手順は、仮想マシンが使用するゲスト OS によって異なります。たとえば、ゲスト OS が RHEL 9 の場合は、[イーサネット接続の設定](#) を参照してください。

検証

1. 新たに作成されたブリッジが実行中で、ホストの物理インターフェイスと仮想マシンのインターフェイスの両方が含まれていることを確認します。

```
# ip link show master bridge0
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
10: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bridge0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether fe:54:00:89:15:40 brd ff:ff:ff:ff:ff:ff
```

2. 仮想マシンがハイパーバイザーと同じ外部ネットワークに表示されることを確認します。

- a. ゲストオペレーティングシステムで、システムのネットワーク ID を取得します。たとえば、これが Linux ゲストの場合は、次のコマンドを実行します。

```
# ip addr
[...]
enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP group default qlen 1000
    link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s0
```

- b. ローカルネットワークに接続された外部システムから、取得した ID を使用して仮想マシンに接続します。

```
# ssh root@192.0.2.1
root@192.0.2.1's password:
Last login: Mon Sep 24 12:05:36 2019
root~#*
```

接続が機能している場合にはネットワークが正常に設定されています。

トラブルシューティング

- 仮想マシンがクライアントでホストされる間に、クライアントからサイトへの VPN を使用するなどの特定の状況では、外部ロケーションで仮想マシンを利用可能にするブリッジモードを使用することはできません。
この問題を回避するには、仮想マシンの [nftables](#) を使用して宛先 NAT を設定 します。

関連情報

- [Web コンソールを使用した外部に表示される仮想マシンの設定](#)

- [ブリッジモードの仮想ネットワーク](#)

17.3.2. Web コンソールを使用した外部に表示される仮想マシンの設定

デフォルトでは、新規作成された仮想マシンは、NAT タイプのネットワークに接続されます。このネットワークは、ホストのデフォルトの仮想ブリッジである **virbr0** を使用します。これにより、仮想マシンはホストのネットワークインターフェイスコントローラー (NIC) を使用して外部ネットワークに接続できますが、外部システムから仮想マシンには到達できません。

仮想マシンをハイパーバイザーと同じ外部ネットワークに表示する必要がある場合は、代わりに [ブリッジモード](#) を使用する必要があります。これには、仮想マシンを、ハイパーバイザーの物理ネットワークデバイスに接続されているブリッジデバイスに割り当てます。これを行うために RHEL 9 Web コンソールを使用するには、以下の手順に従います。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- デフォルトの NAT 設定を持つ [既存の仮想マシン](#) のシャットダウン。
- ハイパーバイザーの IP 設定。これは、ホストのネットワーク接続によって異なります。たとえば、以下の手順では、イーサネットケーブルを使用してホストがネットワークに接続され、ホストの物理 NIC MAC アドレスが DHCP サーバーの静的 IP に割り当てられるシナリオを使用します。したがって、イーサネットインターフェイスはハイパーバイザー IP として扱われます。イーサネットインターフェイスの IP 設定を取得するには、Web コンソールの **Networking** タブに移動し、**Interfaces** セクションを確認します。

手順

1. ホスト上の物理インターフェイスのブリッジ接続を作成して設定します。手順は、[Web コンソールでネットワークブリッジの設定](#) を参照してください。
静的 IP 割り当てが使用されるシナリオでは、物理イーサネットインターフェイスの IPv4 設定をブリッジインターフェイスに移行する必要があることに注意してください。
2. ブリッジインターフェイスを使用するように仮想マシンのネットワークを変更します。仮想マシンの [ネットワークインターフェイス](#) タブで、以下を行います。
 - a. **ネットワークインターフェイスの追加** をクリックします。
 - b. **仮想ネットワークインターフェイスの追加** ダイアログで、以下を設定します。
 - インターフェイスタイプ: **LAN へのブリッジ**
 - ソース: **新規作成ブリッジ** (例: **bridge0**)
 - c. **追加** をクリックします。
 - d. **任意**: 仮想マシンに接続するその他の全インターフェイスに対して **Unplug** をクリックします。
3. **実行** をクリックして、仮想マシンを起動します。
4. ゲストオペレーティングシステムで、仮想マシンがハイパーバイザーと同じネットワーク内の別の物理システムであるかのように、システムのネットワークインターフェイスの IP および DHCP 設定を調整します。

これに関する具体的な手順は、仮想マシンが使用するゲスト OS によって異なります。たとえば、ゲスト OS が RHEL 9 の場合は、[イーサネット接続の設定](#) を参照してください。

検証

1. ホストの Web コンソールの **Networking** タブで、新たに作成されたブリッジがある行をクリックして、ホストの物理インターフェイスと仮想マシンのインターフェイスの両方が含まれていることを確認します。
2. 仮想マシンがハイパーバイザーと同じ外部ネットワークに表示されることを確認します。
 - a. ゲストオペレーティングシステムで、システムのネットワーク ID を取得します。たとえば、これが Linux ゲストの場合は、次のコマンドを実行します。

```
# ip addr
[...]
enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP group default qlen 1000
    link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s0
```

- b. ローカルネットワークに接続された外部システムから、取得した ID を使用して仮想マシンに接続します。

```
# ssh root@192.0.2.1
root@192.0.2.1's password:
Last login: Mon Sep 24 12:05:36 2019
root~#*
```

接続が機能している場合にはネットワークが正常に設定されています。

トラブルシューティング

- 仮想マシンがクライアントでホストされる間に、クライアントからサイトへの VPN を使用するなどの特定の状況では、外部ロケーションで仮想マシンを利用可能にするブリッジモードを使用することはできません。

関連情報

- [コマンドラインインターフェイスを使用した外部に表示される仮想マシンの設定](#)
- [ブリッジモードの仮想ネットワーク](#)

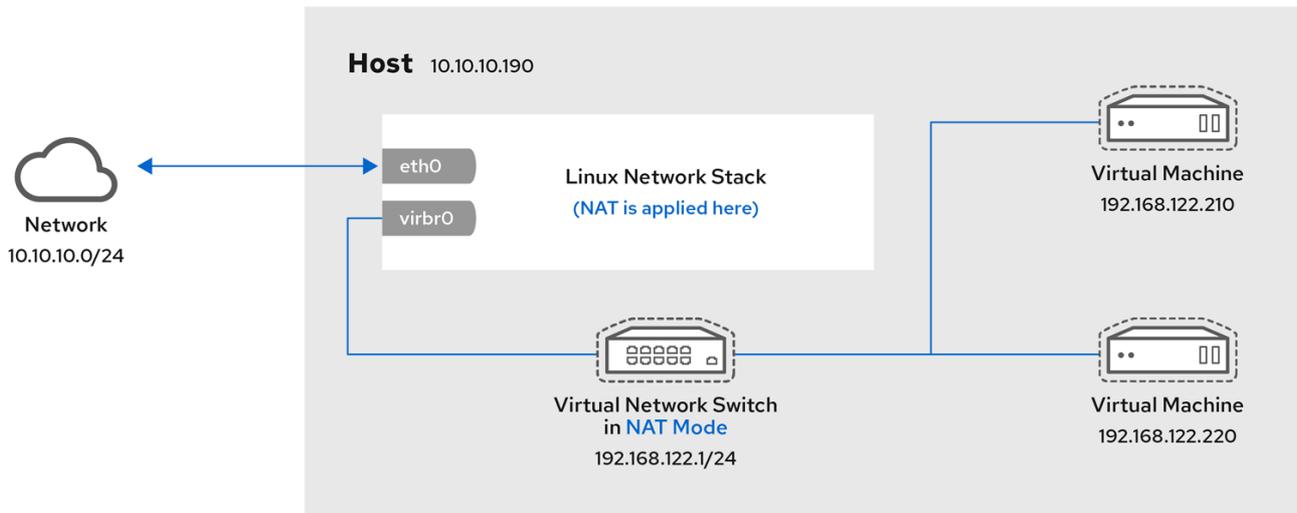
17.4. 仮想マシンのネットワーク接続の種類

仮想マシンのネットワークプロパティと動作を変更するには、仮想マシンが使用する仮想ネットワークまたはインターフェイスの種類を変更します。次のセクションでは、RHEL 9 の仮想マシンで利用可能な接続の種類を説明します。

17.4.1. ネットワークアドレス変換のある仮想ネットワーク

デフォルトでは、仮想ネットワークスイッチはネットワークアドレス変換 (NAT) モードで動作します。Source-NAT (SNAT) または Destination-NAT (DNAT) の代わりに IP マスカレードを使用します。IP マスカレードで接続されている仮想マシンは、外部ネットワークとの通信にホストマシンの IP アド

レスを使用できるようになります。ホスト外のコンピューターは、仮想ネットワークスイッチが NAT モードで動作している場合に、ホスト内部の仮想マシンと通信できません。



RHEL_52_1219



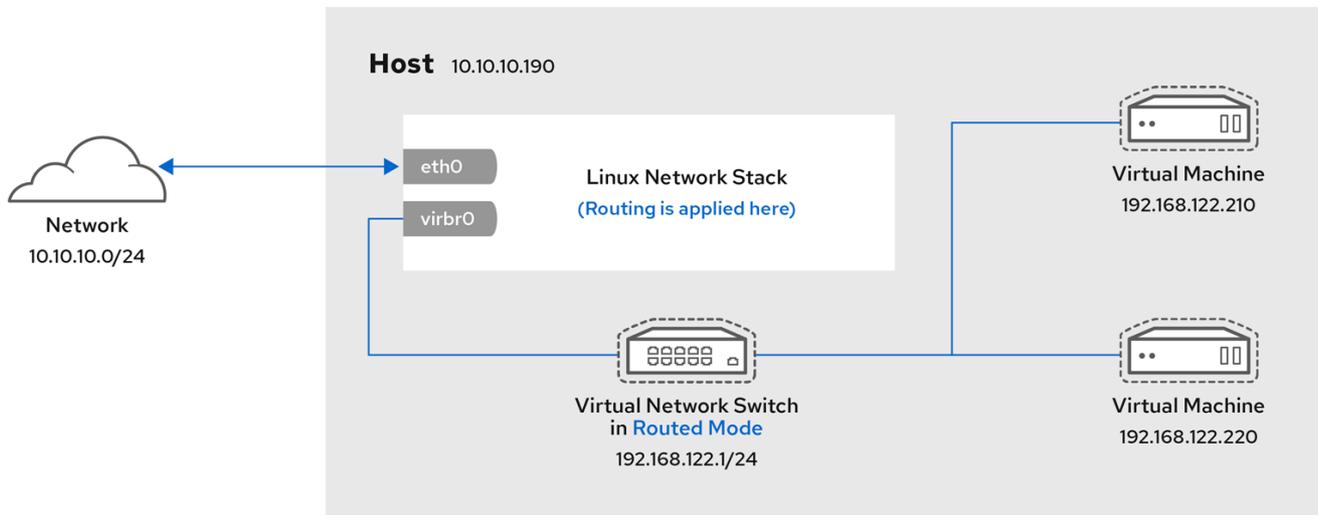
警告

仮想ネットワークスイッチは、ファイアウォールルールで設定された NAT を使用します。スイッチの実行中にこのルールを編集することは推奨されていません。誤ったルールがあると、スイッチが通信できなくなる可能性があるためです。

17.4.2. ルーティングモードの仮想ネットワーク

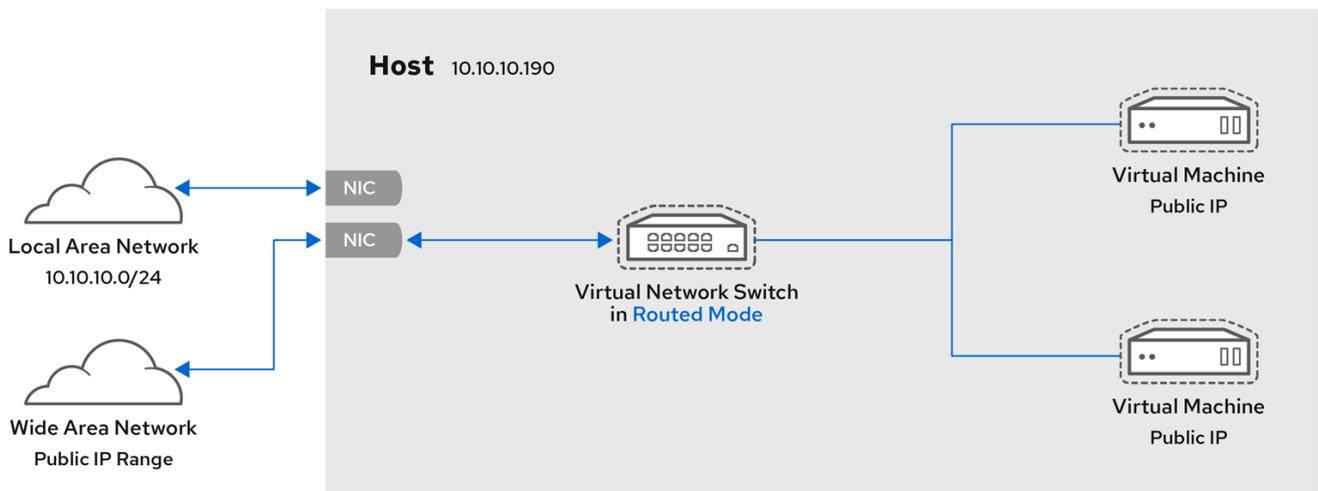
ルーティングモードを使用する場合は、仮想スイッチを、ホストマシンに接続された物理 LAN に接続し、NAT を使用せずにトラフィックをやり取りします。仮想スイッチは、すべてのトラフィックを調べ、ネットワークパケットに含まれる情報を使用して、ルーティングの決定を行うことができます。このモードを使用すると、仮想マシンはすべて1つのサブネット内に入り、ホストマシンから分離されません。仮想マシンサブネットは、ホストマシンにある仮想スイッチを介してルーティングされます。これにより、着信接続が有効になりますが、外部ネットワークのシステムに追加のルーティングテーブルエントリが必要になります。

ルーティングモードは、IP アドレスベースのルーティングを使用します。



RHEL_52_1219

ルーティングモードを使用する一般的なトポロジは、仮想サーバーホスティング (VSH) です。VSH プロバイダーには複数のホストマシンがあり、それぞれに2つの物理ネットワーク接続がある場合があります。管理とアカウンティングにはいずれかのインターフェイスが使用されており、もう1つは仮想マシンによる接続に使用されます。各仮想マシンには独自のパブリック IP アドレスがありますが、ホストマシンはプライベート IP アドレスを使用するため、内部管理者のみが仮想マシンを管理できます。

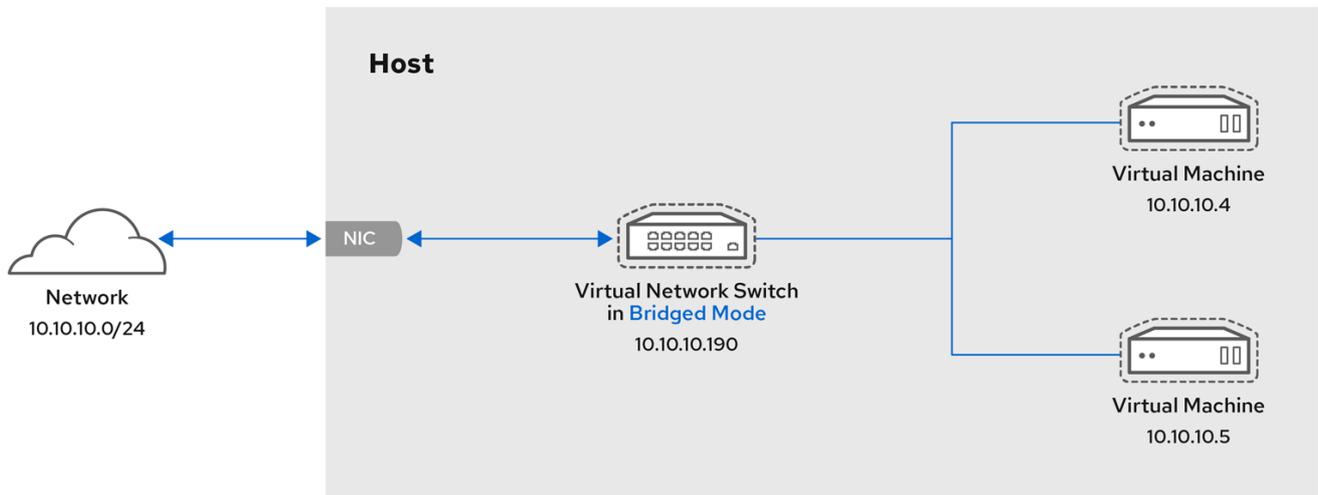


93_RHEL_0520

17.4.3. ブリッジモードの仮想ネットワーク

その他の仮想マシンネットワークモードは、仮想ブリッジ **virbr0** を自動的に作成して接続します。一方、**ブリッジ** モードでは、仮想マシンはホストの既存の Linux ブリッジに接続します。これにより、仮想マシンが物理ネットワークに直接表示されます。これにより、着信接続が有効になりますが、追加のルーティングテーブルエントリは必要ありません。

ブリッジモードは、MAC アドレスをベースにした接続スイッチを使用します。



RHEL_52_1219

ブリッジモードでは、仮想マシンがホストマシンと同じサブネットに表示されます。同じ物理ネットワーク上にある他の物理マシンはすべて、仮想マシンを検出してアクセスできます。

ブリッジネットワークボンディング

ハイパーバイザーで複数の物理ブリッジインターフェイスを使用する場合は、ボンドで複数のインターフェイスを結合します。ボンドをブリッジに追加すると、仮想マシンをブリッジに追加できるようになります。ただし、ボンディングドライバーにはいくつかの動作モードがあり、このモードのすべてが、仮想マシンが使用されているブリッジで機能するわけではありません。

以下の **ボンディングモード** を使用できます。

- モード 1
- モード 2
- モード 4

対照的に、モード 0、3、5、または 6 を使用すると、接続が失敗する可能性が高くなります。また、アドレス解決プロトコル (ARP) の監視が正しく機能しないため、MII (Media-Independent Interface) 監視を使用してボンディングモードを監視する必要があります。

ボンディングモードの詳細は、[Red Hat ナレッジベース](#) を参照してください。

一般的なシナリオ

ブリッジモードにおける最も一般的なユースケースには、たとえば以下のようなものがあります。

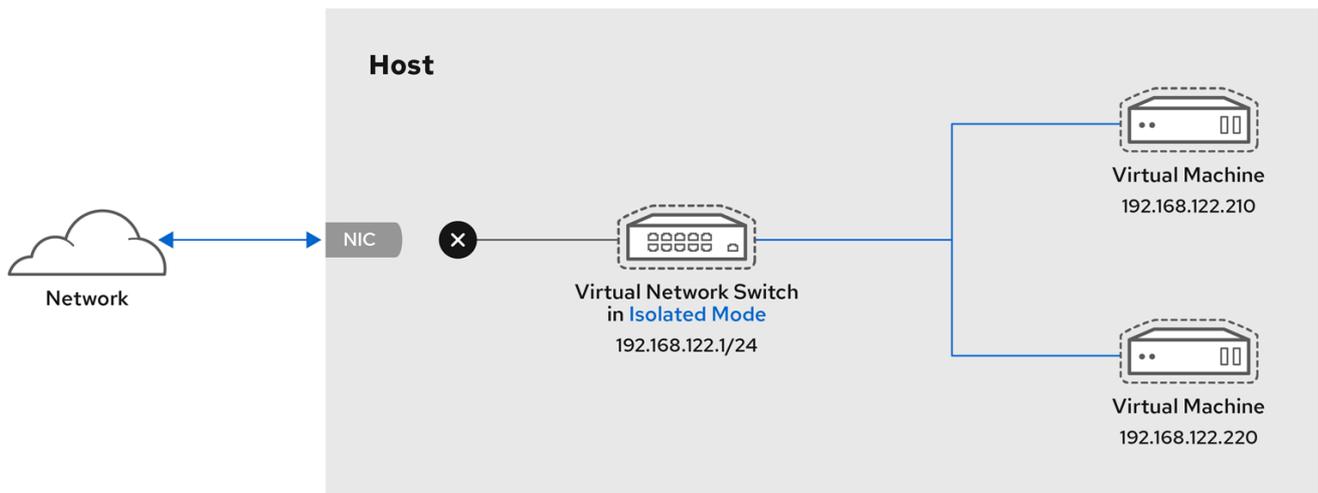
- ホストマシンとともに既存のネットワークに仮想マシンをデプロイし、仮想マシンと物理マシンの相違点をエンドユーザーに見えないようにする。
- 既存の物理ネットワーク設定を変更せずに仮想マシンをデプロイする。
- 既存の物理ネットワークから簡単にアクセスできる必要がある仮想マシンをデプロイする。また、DHCP サービスにアクセスする必要のある物理ネットワークに仮想マシンを配置する。
- 仮想 LAN (VLAN) が使用されている既存のネットワークに仮想マシンを接続する。
- 非武装地帯 (DMZ) ネットワーク。仮想マシンを使用した DMZ デプロイメントの場合、Red Hat は、物理ネットワークのルーターとスイッチで DMZ を設定し、ブリッジモードを使用して仮想マシンを物理ネットワークに接続することを推奨しています。

関連情報

- [コマンドラインインターフェイスを使用した外部に表示される仮想マシンの設定](#)
- [Web コンソールを使用した外部に表示される仮想マシンの設定](#)
- [Explanation of `bridge_opts` parameters](#)

17.4.4. 分離モードの仮想ネットワーク

分離モードを使用すると、仮想スイッチに接続されている仮想マシンは相互に通信でき、ホストマシンとも通信できますが、トラフィックはホストマシンの外部を通過せず、ホストマシンの外部からトラフィックを受信することができません。DHCP などの基本的な機能には、このモードの `dnsmasq` を使用する必要があります。



RHEL_52_1219

17.4.5. オープンモードの仮想ネットワーク

ネットワークにオープンモードを使用する場合、`libvirt` はネットワークにファイアウォールルールを生成しません。したがって、`libvirt` はホストが提供するファイアウォールルールを上書きせず、仮想マシンのファイアウォールルールを手動で管理できます。

17.4.6. 仮想マシンの接続タイプの比較

以下の表では、選択したタイプの仮想マシンネットワーク設定が接続できる場所と、表示できる内容を示します。

表17.1 仮想マシンの接続タイプ

	ホストへの接続	ホスト上の他の仮想マシンへの接続	外部ロケーションへの接続	外部の場所に表示可能か
ブリッジモード	はい	はい	はい	はい
NAT	はい	はい	はい	いいえ
ルーティングモード	はい	はい	はい	はい

	ホストへの接続	ホスト上の他の仮想マシンへの接続	外部ロケーションへの接続	外部の場所に表示可能か
分離モード	はい	はい	いいえ	いいえ
オープンモード	ホストのファイアウォールルールにより異なります。			

17.5. PXE サーバーから仮想マシンの起動

PXE (Preboot Execution Environment) を使用する仮想マシンは、ネットワークから起動して設定を読み込むことができます。本章では、**libvirt** を使用して、仮想ネットワークまたはブリッジネットワークの PXE サーバーから仮想マシンを起動する方法を説明します。



警告

以下の手順は、例としてのみ提供されます。続行する前に、十分なバックアップがあることを確認してください。

17.5.1. 仮想ネットワークで PXE ブートサーバーの設定

この手順では、PXE (Preboot Execution Environment) を提供するように **libvirt** 仮想ネットワークを設定する方法を説明します。これにより、ホストの仮想マシンを、仮想ネットワークで利用可能な起動イメージから起動するように設定できます。

前提条件

- 次のようなローカルの PXE サーバー (DHCP および TFTP)
 - libvirt 内部サーバー
 - 手動で設定した dhcpd および tftpd
 - dnsmasq
 - Cobbler サーバー
- Cobbler が設定した **PXELINUX** など、または手動で設定した PXE 起動イメージ。

手順

1. PXE ブートイメージおよび設定を **/var/lib/tftpboot** フォルダに置きます。
2. フォルダのパーミッションを設定する:

```
# chmod -R a+r /var/lib/tftpboot
```

3. フォルダの所有権を設定する:

```
# chown -R nobody: /var/lib/tftpboot
```

- SELinux コンテキストを更新します。

```
# chcon -R --reference /usr/sbin/dnsmasq /var/lib/tftpboot
# chcon -R --reference /usr/libexec/libvirt_leasehelper /var/lib/tftpboot
```

- 仮想ネットワークをシャットダウンします。

```
# virsh net-destroy default
```

- デフォルトエディターで仮想ネットワーク設定ファイルを開きます。

```
# virsh net-edit default
```

- <ip> 要素を編集して、適切なアドレス、ネットワークマスク、DHCP アドレス範囲、および起動ファイルを追加します。example-pxelinux は、ブートイメージファイルの名前になります。

```
<ip address='192.0.2.1' netmask='255.255.255.0'>
  <tftp root='/var/lib/tftpboot' />
  <dhcp>
    <range start='192.0.2.2' end='192.0.2.254' />
    <bootp file='example-pxelinux' />
  </dhcp>
</ip>
```

- 仮想ネットワークを起動します。

```
# virsh net-start default
```

検証

- default** 仮想ネットワークが有効であることを確認します。

```
# virsh net-list
Name          State  Autostart  Persistent
-----
default      active no         no
```

関連情報

- [PXE を使用してネットワークからインストールするための準備](#)

17.5.2. PXE および仮想ネットワークを使用した仮想マシンの起動

仮想ネットワークで利用可能な PXE (Preboot Execution Environment) サーバーから仮想マシンを起動するには、PXE ブートを有効にする必要があります。

前提条件

- PXE ブートサーバーは、[Setting up a PXE boot server on a virtual network](#) 説明されているように、仮想ネットワークでセットアップされます。

手順

- PXE 起動が有効になっている新しい仮想マシンを作成します。たとえば、**default** 仮想ネットワークで利用可能な PXE から、新しい 10GB の qcow2 イメージファイルにインストールする場合は、次のコマンドを実行します。

```
# virt-install --pxe --network network=default --memory 2048 --vcpus 2 --disk size=10
```

- または、既存の仮想マシンの XML 設定ファイルを手動で編集できます。
 - i. **<os>** 要素の内部に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
  <boot dev='hd'/>
</os>
```

- ii. 仮想ネットワークを使用するようにゲストネットワークが設定されている。

```
<interface type='network'>
  <mac address='52:54:00:66:79:14'/>
  <source network='default'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

検証

- **virsh start** コマンドを使用して仮想マシンを起動します。PXE が正しく設定されていると、仮想マシンは、PXE サーバーで利用可能な起動イメージから起動します。

17.5.3. PXE およびブリッジネットワークを使用した仮想マシンの起動

ブリッジネットワークで利用可能な PXE (Preboot Execution Environment) サーバーから仮想マシンを起動するには、PXE ブートを有効にする必要があります。

前提条件

- ネットワークブリッジが有効になっている。
- ブリッジネットワークでは、PXE ブートサーバーが利用できます。

手順

- PXE 起動が有効になっている新しい仮想マシンを作成します。たとえば、**breth0** ブリッジネットワークで利用可能な PXE から、新しい 10GB の qcow2 イメージファイルにインストールする場合は、次のコマンドを実行します。

```
# virt-install --pxe --network bridge=breth0 --memory 2048 --vcpus 2 --disk size=10
```

- または、既存の仮想マシンの XML 設定ファイルを手動で編集できます。

- i. **<os>** 要素の内部に **<boot dev='network'/>** 要素があることを確認します。

```
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
  <boot dev='hd'/>
</os>
```

- ii. ブリッジネットワークを使用するように仮想マシンが設定されていることを確認します。

```
<interface type='bridge'>
  <mac address='52:54:00:5a:ad:cb'/>
  <source bridge='breth0'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

検証

- **virsh start** コマンドを使用して仮想マシンを起動します。PXE が正しく設定されていると、仮想マシンは、PXE サーバーで利用可能な起動イメージから起動します。

関連情報

- [ネットワークブリッジの設定](#)

17.6. PASST ユーザー空間接続の設定

仮想ネットワークへの非特権アクセスが必要な場合 (たとえば、**libvirt** の **session** 接続を使用する場合)、**passt** ネットワークバックエンドを使用するように仮想マシン (VM) を設定できます。

前提条件

- **passt** パッケージがシステムにインストールされている。

```
# dnf install passt
```

手順

1. **passt** 接続を使用する仮想マシンの XML 設定を開きます。以下に例を示します。

```
# virsh edit <testquest1>
```

2. **<devices>** セクションに、バックエンドタイプとして **passt** を使用する **<interface type='user'>** 要素を追加します。

たとえば、次の設定では、最初のデフォルトルートに関連付けられたホストインターフェイスからコピーされたアドレスとルートを使用する **passt** 接続を設定します。

```
<devices>
  [...]
```

```
<interface type='user'>
  <backend type='passt'/>
</interface>
</devices>
```

passt を使用する場合は、必要に応じて、複数の **<portForward>** 要素を指定して、ホストへの受信ネットワークトラフィックをこの仮想マシンインターフェイスに転送できます。インターフェイスの IP アドレスをカスタマイズすることもできます。以下に例を示します。

```
<devices>
[... ]
<interface type='user'>
  <backend type='passt'/>
  <mac address="52:54:00:98:d8:b7"/>
  <source dev='eth0'/>
  <ip family='ipv4' address='192.0.2.1' prefix='24'/>
  <ip family='ipv6' address='::ffff:c000:201'/>
  <portForward proto='tcp'>
    <range start='2022' to='22'/>
  </portForward>
  <portForward proto='udp' address='1.2.3.4'>
    <range start='5000' end='5020' to='6000'/>
    <range start='5010' end='5015' exclude='yes'/>
  </portForward>
  <portForward proto='tcp' address='2001:db8:ac10:fd01::1:10' dev='eth0'>
    <range start='8080'/>
    <range start='4433' to='3444'/>
  </portForward>
</interface>
</devices>
```

この設定例では、次のパラメーターを使用して **passt** 接続を設定します。

- 仮想マシンは、**eth0** ホストインターフェイスからトラフィックを転送するためのネットワークルートをコピーします。
- インターフェイス MAC は **52:54:00:98:d8:b7** に設定します。設定されていない場合は、ランダムなものが生成されます。
- IPv4 アドレスは **192.0.2.1/24** に設定し、IPv6 アドレスは **::ffff:c000:201** に設定します。
- ホスト上の TCP ポート **2022** は、そのネットワークトラフィックを仮想マシン上のポート **22** に転送します。
- ホストインターフェイス **eth0** の TCP アドレス **2001:db8:ac10:fd01::1:10** とポート **8080** は、ネットワークトラフィックを仮想マシンのポート **8080** に転送します。ポート **4433** は仮想マシンのポート **3444** に転送します。
- ホスト上の UDP アドレス **1.2.3.4** とポート **5000 - 5009** および **5016 - 5020** は、ネットワークトラフィックを仮想マシン上のポート **6000 - 6009** および **6016 - 6020** に転送します。

3. XML 設定を保存します。

検証

- **passt** を使用して設定した仮想マシンを起動または再起動します。

```
# virsh reboot <vm-name>  
# virsh start <vm-name>
```

仮想マシンが正常に起動すると、**passt** ネットワークバックエンドが使用されます。

関連情報

- [libvirt](#) でのシステムおよびセッションの接続

17.7. 関連情報

- [ネットワークの設定および管理](#)
- [特定のネットワークインターフェイスカードを SR-IOV デバイスとして割り当て、仮想マシンのパフォーマンスを向上させます。](#)

第18章 仮想マシンのパフォーマンスの最適化

仮想マシンでは、ホストと比べて、パフォーマンス低下が常に見られます。以下のセクションでは、この低下の理由を説明します。また、ハードウェアのインフラストラクチャーリソースを可能な限り効率的に使用できるように、RHEL 9での仮想化によるパフォーマンスへの影響を最小限に抑える方法を説明します。

18.1. 仮想マシンのパフォーマンスに影響を及ぼすもの

仮想マシンは、ホストのユーザー空間プロセスとして実行します。したがって、ハイパーバイザーは、仮想マシンがホストシステムのリソースを使用できるように、ホストのシステムリソースを変換する必要があります。したがって、変換によりリソースの一部が消費されるため、仮想マシンのパフォーマンス効率は、ホストと同じにはなりません。

システムパフォーマンスにおける仮想化の影響

仮想マシンのパフォーマンス低下の理由には、以下のようなものがあります。

- 仮想 CPU (vCPU) がホスト上のスレッドとして実装され、Linux スケジューラーで処理される。
- 仮想マシンは、ホストカーネルから NUMA や Huge Page などの最適化機能を自動的に継承しない。
- ホストのディスクおよびネットワーク I/O の設定が、仮想マシンのパフォーマンスに大きく影響する可能性がある。
- ネットワークトラフィックは、一般的に、ソフトウェアベースのブリッジから仮想マシンに流れる。
- ホストデバイスとそのモデルによっては、その特定のハードウェアのエミュレーションにより、オーバーヘッドが著しくなる可能性がある。

仮想化が仮想マシンのパフォーマンスに与える影響の重大度は、次のようなさまざまな要因の影響を受けます。

- 同時に実行している仮想マシンの数
- 各仮想マシンで使用される仮想デバイスのサイズ
- 仮想マシンが使用するデバイスの種類

仮想マシンのパフォーマンス損失を減らす

RHEL 9 は、仮想化のパフォーマンスへの悪影響を減らすのに使用できる多くの機能を提供します。以下に例を示します。

- **tuned サービス** は、仮想マシンのリソースディストリビューションとパフォーマンスを自動的に最適化できる。
- **ブロック I/O チューニング** により、ディスクなどの仮想マシンのブロックデバイスのパフォーマンスを改善できる。
- **NUMA のチューニング** により、vCPU のパフォーマンスを向上させることができる。
- **仮想ネットワーク** は、さまざまな方法で最適化できる。



重要

仮想マシンのパフォーマンスのチューニングは、その他の仮想化機能に悪影響を与える可能性があります。たとえば、変更した仮想マシンの移行がより困難になります。

18.2. TUNED を使用した仮想マシンのパフォーマンスの最適化

TuneD ユーティリティーは、CPU 集中型タスクや、ストレージネットワークスループットの応答などの特定のワークロードの特性に対して RHEL を調整するプロファイル配信メカニズムです。これにより、特定のユースケースで、パフォーマンスを強化し、電力消費を減らすように事前設定されたチューニングプロファイルを多数利用できます。これらのプロファイルを編集するか、新規プロファイルを作成して、仮想化環境に適したパフォーマンスソリューション (仮想化環境を含む) を作成できます。

RHEL 9 を仮想化に最適化するには、次のプロファイルを使用します。

- RHEL 9 仮想マシンの場合は、**virtual-guest** プロファイルを使用します。これは、一般的に適用された **throughput-performance** プロファイルをベースにしていますが、仮想メモリーのスワップは減少します。
- RHEL 9 仮想ホストの場合は、**virtual-host** プロファイルを使用します。これにより、ダーティメモリーページのより集中的なライトバックが有効になり、ホストのパフォーマンスを活用できます。

前提条件

- **TuneD** サービスがインストールされており、有効になっている。

手順

特定の **TuneD** プロファイルを有効にするには、以下を実行します。

1. 使用可能な **Tuned** プロファイルをリスト表示します。

```
# tuned-adm list
```

```
Available profiles:
```

```
- balanced          - General non-specialized TuneD profile
- desktop          - Optimize for the desktop use-case
[...]
- virtual-guest    - Optimize for running inside a virtual guest
- virtual-host     - Optimize for running KVM guests
Current active profile: balanced
```

2. (必要に応じて) 新しい **TuneD** プロファイルを作成するか、既存の **TuneD** プロファイルを編集します。
詳細は [TuneD プロファイルのカスタマイズ](#) を参照してください。
3. **TuneD** プロファイルをアクティベートします。

```
# tuned-adm profile selected-profile
```

- 仮想化ホストを最適化するには、**virtual-host** プロファイルを使用します。

```
# tuned-adm profile virtual-host
```

- RHEL ゲストオペレーティングシステムで、`virtual-guest` プロファイルを使用します。

```
# tuned-adm profile virtual-guest
```

関連情報

- [システムの状態とパフォーマンスの監視と管理](#)

18.3. LIBVIRT デーモンの最適化

libvirt 仮想化スイートは、RHEL ハイパーバイザーの管理層として機能し、**libvirt** の設定は仮想化ホストに大きな影響を与えます。特に、RHEL 9 には、モノリシックまたはモジュラーの2つのタイプの **libvirt** デーモンが含まれており、使用するデーモンのタイプは、個々の仮想化ドライバーをどの程度細かく設定できるかに影響します。

18.3.1. libvirt デーモンのタイプ

RHEL 9 は、以下の **libvirt** デーモンタイプをサポートします。

モノリシックな libvirt

従来の **libvirt** デーモンである **libvirtd** は、単一の設定ファイル `/etc/libvirt/libvirtd.conf` を使用して、さまざまな仮想化ドライバーを制御します。

このため、**libvirtd** は一元化されたハイパーバイザー設定を可能にしますが、システムリソースの使用が非効率的となる可能性があります。したがって、**libvirtd** は、RHEL の今後のメジャーリリースではサポートされなくなる予定です。

ただし、RHEL 8 から RHEL 9 に更新した場合、ホストはデフォルトで引き続き **libvirtd** を使用します。

モジュラー libvirt

RHEL 9 で新たに導入されたモジュラー **libvirt** は、仮想化ドライバーごとに特定のデーモンを提供します。これらには以下が含まれます。

- **virtqemud** - ハイパーバイザー管理用のプライマリーデーモン
- **virtinterfaced** - ホストの NIC 管理用のセカンダリーデーモン
- **virtnetworkd** - 仮想ネットワーク管理用のセカンダリーデーモン
- **virtnodevd** - ホストの物理デバイス管理用のセカンダリーデーモン
- **virtnwfilterd** - ホストのファイアウォール管理用のセカンダリーデーモン
- **virtsecret** - ホストシークレット管理用のセカンダリーデーモン
- **virtstoraged** - ストレージ管理用のセカンダリーデーモン

デーモンごとに個別の設定ファイル (`/etc/libvirt/virtqemud.conf` など) があります。したがって、モジュラーの **libvirt** デーモンは、**libvirt** リソース管理を細かく調整するためのより良いオプションを提供します。

RHEL 9 を新規インストールした場合、モジュラー **libvirt** はデフォルトで設定されています。

次のステップ

- RHEL 9 で **libvirtd** を使用する場合、Red Hat は、モジュール式デーモンへの切り替えを推奨しています。手順は、[モジュラー libvirt デーモンの有効化](#) を参照してください。

18.3.2. モジュラー libvirt デーモンの有効化

RHEL 9 では、**libvirt** ライブラリーは、ホスト上の個々の仮想化ドライバーセットを処理するモジュラーデーモンを使用します。たとえば、**virtqemu** デーモンは QEMU ドライバーを処理します。

RHEL 9 ホストの新規インストールを実行すると、ハイパーバイザーはデフォルトでモジュラー **libvirt** デーモンを使用します。ただし、ホストを RHEL 8 から RHEL 9 にアップグレードした場合、ハイパーバイザーは RHEL 8 のデフォルトであるモノリシックな **libvirtd** デーモンを使用します。

その場合、Red Hat は、代わりにモジュラー **libvirt** デーモンを有効にすることを推奨します。これは、**libvirt** リソース管理を微調整するためのより良いオプションを提供するためです。また、RHEL の今後のメジャーリリースでは **libvirtd** はサポートされなくなる予定です。

前提条件

- ハイパーバイザーがモノリシックな **libvirtd** サービスを使用している。

```
# systemctl is-active libvirtd.service
active
```

このコマンドで **active** が表示される場合、**libvirtd** を使用していることとなります。

- 仮想マシンがシャットダウンしている。

手順

1. **libvirtd** とそのソケットを停止します。

```
$ systemctl stop libvirtd.service
$ systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

2. **libvirtd** を無効にして、システムの起動時に開始されないようにします。

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

3. モジュラーの **libvirt** デーモンを有効にします。

```
# for drv in qemu interface network nodedev nwfilter secret storage; do systemctl unmask
virt${drv}d.service; systemctl unmask virt${drv}d{,-ro,-admin}.socket; systemctl enable
virt${drv}d.service; systemctl enable virt${drv}d{,-ro,-admin}.socket; done
```

4. モジュラーデーモンのソケットを起動します。

```
# for drv in qemu network nodedev nwfilter secret storage; do systemctl start virt${drv}d{,-ro,-
admin}.socket; done
```

5. **オプション:** リモートホストからホストに接続する必要がある場合は、仮想化プロキシーデーモンを有効にして起動します。

- a. システムで **libvirtd-tls.socket** サービスが有効になっているかどうかを確認します。

```
# grep listen_tls /etc/libvirt/libvirtd.conf

listen_tls = 0
```

- b. **libvirtd-tls.socket** が有効になっていない場合 (**listen_tls = 0**)、次のように **virtproxyd** をアクティブにします。

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin}.socket
# systemctl start virtproxyd{,-ro,-admin}.socket
```

- c. **libvirtd-tls.socket** が有効になっている場合 (**listen_tls = 1**)、次のように **virtproxyd** をアクティブにします。

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

virtproxyd の TLS ソケットを有効にするには、**libvirt** で使用できるように設定された TLS 証明書がホストに必要です。詳細は、[アップストリームの libvirt ドキュメント](#) を参照してください。

検証

1. 有効化された仮想化デーモンをアクティブにします。

```
# virsh uri
qemu:///system
```

2. ホストが **virtqemud** モジュールデーモンを使用していることを確認します。

```
# systemctl is-active virtqemud.service
active
```

ステータスが **active** の場合、**libvirt** モジュールデーモンは正常に有効になっています。

18.4. 仮想マシンのメモリーの設定

仮想マシンのパフォーマンスを改善するために、追加のホスト RAM を仮想マシンに割り当てることができます。同様に、仮想マシンに割り当てるメモリー量を減らして、ホストメモリーを他の仮想マシンやタスクに割り当てることができます。

これらのアクションを実行するには、[Web コンソール](#) または [コマンドラインインターフェイス](#) を使用します。

18.4.1. Web コンソールを使用した仮想マシンのメモリーの追加および削除

仮想マシンのパフォーマンスを向上させるか、仮想マシンが使用するホストリソースを解放するために、Web コンソールを使用して、仮想マシンに割り当てられたメモリーの量を調整できます。

前提条件

- ゲスト OS がメモリーバルンドライバーを実行している。これを確認するには、以下を実行します。
 1. 仮想マシンの設定に **memballoon** デバイスが含まれていることを確認します。


```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

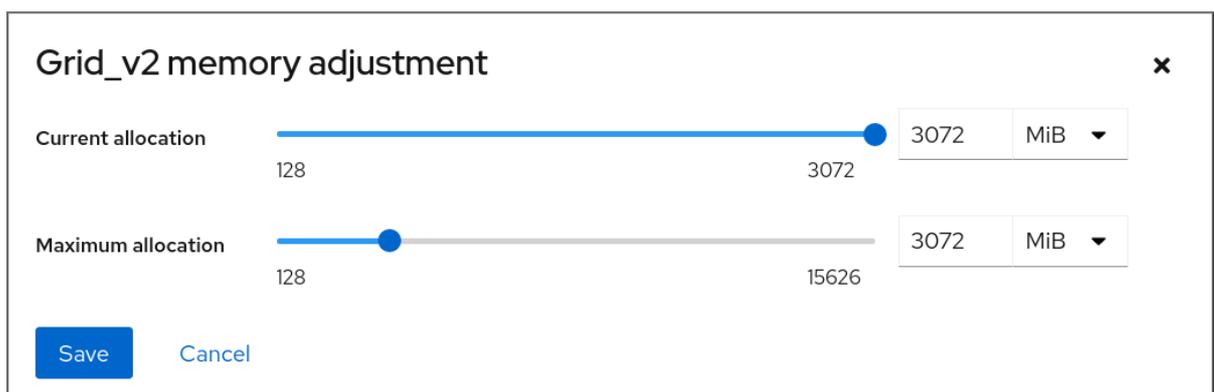
このコマンドで出力が表示され、モデルが **none** に設定されていない場合は、**memballoon** デバイスが存在します。
 2. バルンドライバーがゲスト OS で実行していることを確認します。
 - Windows ゲストでは、ドライバーは **virtio-win** ドライバーパッケージの一部としてインストールされます。手順は、[Installing KVM paravirtualized drivers for Windows virtual machines](#) を参照してください。
 - Linux ゲストでは、通常、このドライバーはデフォルトで含まれており、**memballoon** デバイスがあれば、アクティベートされます。
- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **任意:** 最大メモリーと、仮想マシンに現在使用されている最大メモリーの情報を取得します。これは、変更のベースラインとしても、検証のためにも機能します。

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

2. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
3. 概要ペインで、**Memory** 行の横にある **編集** をクリックします。**メモリー調整** ダイアログが表示されます。



4. 選択した仮想マシンの仮想メモリーを設定します。

- **最大割り当て**: 仮想マシンがそのプロセスに使用できるホストメモリーの最大量を設定します。VMの作成時に最大メモリーを指定することも、後で増やすこともできます。メモリーは、MiB または GiB の倍数で指定できます。
仮想マシンをシャットダウンしてからでないと、最大メモリー割り当てを調整できません。
- **現在の割り当て** - 仮想マシンに割り当てる実際のメモリー量を設定します。この値は、最大割り当てより小さい値にすることができますが、上限を超えることはできません。値を調整して、仮想マシンで利用可能なメモリーをプロセス用に調整できます。メモリーは、MiB または GiB の倍数で指定できます。
この値を指定しない場合、デフォルトの割り当ては**最大割り当て**の値になります。

5. **Save** をクリックします。

仮想マシンのメモリー割り当てが調整されます。

関連情報

- [コマンドラインインターフェイスを使用した仮想マシンのメモリーの追加と削除](#)
- [仮想マシンの CPU パフォーマンスの最適化](#)

18.4.2. コマンドラインインターフェイスを使用した仮想マシンのメモリーの追加と削除

仮想マシンのパフォーマンスを改善したり、使用しているホストリソースを解放したりするために、CLI を使用して仮想マシンに割り当てられたメモリーの量を調整できます。

前提条件

- ゲスト OS がメモリーバルーンドライバーを実行している。これを確認するには、以下を実行します。

1. 仮想マシンの設定に **memballoon** デバイスが含まれていることを確認します。

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

このコマンドで出力が表示され、モデルが **none** に設定されていない場合は、**memballoon** デバイスが存在します。

2. balloon ドライバーがゲスト OS で実行されていることを確認します。

- Windows ゲストでは、ドライバーは **virtio-win** ドライバーパッケージの一部としてインストールされます。手順は、[Installing KVM paravirtualized drivers for Windows virtual machines](#) を参照してください。
- Linux ゲストでは、通常、このドライバーはデフォルトで含まれており、**memballoon** デバイスがあれば、アクティベートされます。

手順

1. **任意**: 最大メモリーと、仮想マシンに現在使用されている最大メモリーの情報を取得します。これは、変更のベースラインとしても、検証のためにも機能します。

■

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

- 仮想マシンに割り当てる最大メモリーを調整します。この値を増やすと、仮想マシンのパフォーマンスが低下する可能性が向上し、値を減らすことで、仮想マシンがホスト上にあるパフォーマンスフットプリントが低減します。この変更は、停止している仮想マシンでのみ実行できるため、実行中の仮想マシンを調整するには再起動する必要があります。たとえば、仮想マシン **testguest** が使用可能な最大メモリーを 4096 MiB に変更するには、次のコマンドを実行します。

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

実行中の仮想マシンの最大メモリーを増やすには、仮想マシンにメモリーデバイスを割り当てます。これは、**メモリーのホットプラグ**とも呼ばれます。詳細は、Attaching memory devices to virtual machines を参照してください。



警告

実行中の仮想マシン (メモリーのホットアンプラグとも呼ばれる) から、メモリーデバイスを削除することはサポートされておらず、Red Hat では推奨していません。

- 任意:** 仮想マシンが現在使用しているメモリーを最大割り当てまで調整することもできます。これにより、仮想マシンの最大割り当てを変更せずに、仮想マシンが次の再起動までホスト上にあるメモリー負荷が調整されます。

```
# virsh setmem testguest --current 2048
```

検証

- 仮想マシンが使用するメモリーが更新されていることを確認します。

```
# virsh dominfo testguest
Max memory: 4194304 KiB
Used memory: 2097152 KiB
```

- (**必要に応じて**) 現在の仮想マシンメモリーを調整すると、仮想マシンのメモリーバルーンの統計を取得して、そのメモリー使用量をどの程度効果的に調整するかを評価できます。

```
# virsh domstats --balloon testguest
Domain: 'testguest'
balloon.current=365624
balloon.maximum=4194304
balloon.swap_in=0
balloon.swap_out=0
balloon.major_fault=306
```

```

balloon.minor_fault=156117
balloon.unused=3834448
balloon.available=4035008
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456

```

関連情報

- [Web コンソールを使用した仮想マシンのメモリの追加および削除](#)
- [仮想マシンの CPU パフォーマンスの最適化](#)

18.4.3. virtio-mem を使用した仮想マシンメモリの追加および削除

RHEL 9 は、**virtio-mem** 準仮想化メモリーデバイスを提供します。このデバイスを使用すると、仮想マシン (VM) 内のホストメモリーを動的に追加または削除できます。たとえば、**virtio-mem** を使用して、実行中の仮想マシン間でメモリーリソースを移動したり、現在の要件に基づいてクラウドセットアップの仮想マシンメモリーのサイズを変更したりできます。

18.4.3.1. virtio-mem の概要

virtio-mem は、仮想マシンでホストメモリーを動的に追加または削除するために使用できる準仮想化メモリーデバイスです。たとえば、このデバイスを使用して、実行中の仮想マシン間でメモリーリソースを移動したり、現在の要件に基づいてクラウドセットアップの仮想マシンメモリーのサイズを変更したりできます。

virtio-mem を使用すると、4 から数百メガバイト (MiB) の単位で、仮想マシンのメモリーを初期サイズより増やしたり、元のサイズに縮小したりできます。ただし、**virtio-mem** は、特にメモリーを確実にアンプラグするために、特定のゲストオペレーティングシステム設定にも依存していることに注意してください。

virtio-mem 機能の制限

virtio-mem は現在、以下の機能と互換性がありません。

- ホスト上のリアルタイムアプリケーションのメモリーロックの使用
- ホストでの暗号化された仮想化の使用
- **virtio-mem** とホスト上での **memballoon** 膨張および収縮の組み合わせ
- 仮想マシンでの **virtio_mem** ドライバーのアンロードまたはリロード
- **virtiofs** を除く **vhost-user** デバイスの使用

関連情報

- [仮想マシンでのメモリーのオンライン化設定](#)
- [Attaching a virtio-mem device to virtual machines](#)

18.4.3.2. 仮想マシンでのメモリのオンライン化設定

virtio-mem を使用して実行中の仮想マシンにメモリーを接続する (メモリーのホットプラグとも呼ばれます) 前に、ホットプラグされたメモリーが自動的にオンライン状態に設定されるように仮想マシン (VM) オペレーティングシステムを設定する必要があります。そうしないと、ゲストオペレーティングシステムは追加メモリーを使用できなくなります。メモリーのオンライン化については、次のいずれかの設定から選択できます。

- **online_movable**
- **online_kernel**
- **auto-movable**

これらの設定の違いについては、[メモリーのオンライン化設定の比較](#) を参照してください。

RHEL では、メモリーのオンライン化はデフォルトで udev ルールで設定されます。ただし、**virtio-mem** を使用する場合は、カーネル内でメモリーのオンライン化を直接設定することを推奨します。

前提条件

- ホストに Intel 64 または AMD64 CPU アーキテクチャーがある。
- ホストがオペレーティングシステムとして RHEL 9.4 以降を使用している。
- ホスト上で実行されている仮想マシンが、次のいずれかのオペレーティングシステムバージョンを使用している。
 - RHEL 8.10



重要

RHEL 8.10 仮想マシンでは、実行中の仮想マシンからメモリーをアンプラグすることはデフォルトで無効になっています。

- RHEL 9

手順

- 仮想マシンで **online_movable** 設定を使用するようにメモリーオンライン化を設定するには、以下を実行します。
 1. **memhp_default_state** カーネルコマンドラインパラメーターを **online_movable** に設定します。


```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --args=memhp_default_state=online_movable
```
 2. 仮想マシンを再起動します。
- 仮想マシンで **online_kernel** 設定を使用するようにメモリーオンライン化を設定するには、以下を実行します。
 1. 以下のように、**memhp_default_state** カーネルコマンドラインパラメーターを **online_kernel** に設定します。

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online_kernel
```

2. 仮想マシンを再起動します。

- 仮想マシンで **auto-movable** メモリーオンライン化ポリシーを使用するには、以下の手順を実行します。

1. **memhp_default_state** カーネルコマンドラインパラメーターを **online** に設定します。

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online
```

2. **memory_hotplug.online_policy** カーネルコマンドラインパラメーターを **auto-movable** に設定します。

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.online_policy" --
args=memory_hotplug.online_policy=auto-movable
```

3. オプション:**auto-movable** オンライン化ポリシーをさらに調整するには、**memory_hotplug.auto_movable_ratio** パラメーターと **memory_hotplug.auto_movable_numa_aware** パラメーターを変更します。

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.auto_movable_ratio" --
args=memory_hotplug.auto_movable_ratio=<percentage>
```

```
# grubby --update-kernel=ALL --remove-
args="memory_hotplug.memory_auto_movable_numa_aware" --
args=memory_hotplug.auto_movable_numa_aware=<y/n>
```

- **memory_hotplug.auto_movable_ratio parameter** は、任意の割り当てに使用できるメモリーと比較すると、移動可能な割り当てにのみ使用できるメモリーの最大比率を設定します。比率はパーセントで表され、デフォルト値は 3:1 の比率である 301 (%) です。
- **memory_hotplug.auto_movable_numa_aware** パラメーターは、**memory_hotplug.auto_movable_ratio** パラメーターを使用可能なすべての NUMA ノードのメモリーに適用するか、単一の NUMA ノード内のメモリーのみに適用するかを制御します。デフォルト値は y (yes) です。
たとえば、最大比率を 301% に設定し、**memory_hotplug.auto_movable_numa_aware** が y (yes) に設定されている場合は、アタッチされた **virtio-mem** デバイスを持つ NUMA ノード内でも 3:1 の比率が適用されます。パラメーターが n (no) に設定されている場合、最大 3:1 の比率はすべての NUMA ノード全体に対してのみ適用されます。

また、比率を超えていない場合、新しくホットプラグされたメモリーは、移動可能な割り当てに対してのみ利用できます。それ以外の場合では、新しくホットプラグされたメモリーは、移動可能な割り当てと移動不可能な割り当ての両方に使用できます。

4. 仮想マシンを再起動します。

検証

- **online_movable** 設定が正しく設定されているかを確認するには、**memhp_default_state** カーネルパラメーターの現在の値を確認します。

```
# cat /sys/devices/system/memory/auto_online_blocks
online_movable
```

- **online_kernel** 設定が正しく設定されているかを確認するには、**memhp_default_state** カーネルパラメーターの現在の値を確認します。

```
# cat /sys/devices/system/memory/auto_online_blocks
online_kernel
```

- **auto-movable** 設定が正しく設定されているかを確認するには、以下のカーネルパラメーターを確認してください。

- **memhp_default_state:**

```
# cat /sys/devices/system/memory/auto_online_blocks
online
```

- **memory_hotplug.online_policy:**

```
# cat /sys/module/memory_hotplug/parameters/online_policy
auto-movable
```

- **memory_hotplug.auto_movable_ratio:**

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_ratio
301
```

- **memory_hotplug.auto_movable_numa_aware:**

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_numa_aware
y
```

関連情報

- [virtio-mem の概要](#)
- [Attaching a virtio-mem device to virtual machines](#)
- [Configuring Memory Hot\(Un\)Plug](#)

18.4.3.3. Attaching a virtio-mem device to virtual machines

実行中の仮想マシンに追加のメモリーをアタッチ (メモリーのホットプラグとも呼ばれます) し、その後ホットプラグされたメモリーのサイズを変更できるようにするには、**virtio-mem** デバイスを使用できます。具体的には、libvirt XML 設定ファイルと **virsh** コマンドを使用し、**virtio-mem** デバイスを定義して仮想マシン (VM) に割り当てることができます。

並列名”

前提条件

- ホストに Intel 64 または AMD64 CPU アーキテクチャーがある。
- ホストがオペレーティングシステムとして RHEL 9.4 以降を使用している。
- ホスト上で実行されている仮想マシンが、次のいずれかのオペレーティングシステムバージョンを使用している。
 - RHEL 8.10



重要

RHEL 8.10 仮想マシンでは、実行中の仮想マシンからメモリーをアンプラグすることはデフォルトで無効になっています。

- RHEL 9
- VM にメモリーオンライン化が設定されている。手順は、[仮想マシンでのメモリーのオンライン化設定](#)を参照してください。

手順

1. ターゲット仮想マシンの XML 設定に **maxMemory** パラメーターが含まれるようにします。

```
# virsh edit testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <maxMemory unit='GiB'>128</maxMemory>
  ...
</domain>
```

この例では、**testguest1** 仮想マシンの XML 設定で、128 ギビバイト (GiB) の **maxMemory** パラメーターが定義されています。**maxMemory** サイズは、仮想マシンが使用できる最大メモリーを指定します。これには、初期メモリーとホットプラグされたメモリーの両方が含まれます。

2. XML ファイルを作成して開き、ホスト上の **virtio-mem** デバイスを定義します。次に例を示します。

```
# vim virtio-mem-device.xml
```

3. **virtio-mem** デバイスの XML 定義をファイルに追加し、保存します。

```
<memory model='virtio-mem'>
  <target>
    <size unit='GiB'>48</size>
    <node>0</node>
    <block unit='MiB'>2</block>
    <requested unit='GiB'>16</requested>
    <current unit='GiB'>16</current>
  </target>
  <alias name='ua-virtiomem0' />
```

```

    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </memory>
  <memory model='virtio-mem'>
    <target>
      <size unit='GiB'>48</size>
      <node>1</node>
      <block unit='MiB'>2</block>
      <requested unit='GiB'>0</requested>
      <current unit='GiB'>0</current>
    </target>
    <alias name='ua-virtiomem1'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </memory>
  </memory>

```

この例では、2つの **virtio-mem** デバイスが以下のパラメーターで定義されます。

- **size**: これは、デバイスの最大サイズです。この例では 48 GiB です。 **size** は **block** サイズの倍数である必要があります。
 - **node**: これは、 **virtio-mem** デバイ스에割り当てられた vNUMA ノードです。
 - **block**: これはデバイスのブロックサイズです。これは、少なくとも Transparent Huge Page (THP) のサイズである必要があります。THP は、Intel 64 または AMD64 CPU アーキテクチャーでは 2 MiB です。通常、Intel 64 または AMD64 アーキテクチャーでは、2 MiB ブロックサイズが適切なデフォルトの選択となります。 **virtio-mem** を **Virtual Function I/O (VFIO)** または **仲介デバイス (mdev)** で使用する場合、すべての **virtio-mem** デバイスにまたがるブロックの合計数は 32768 を超えることはできません。超えると、RAM のプラグインに失敗する可能性があります。
 - **requested**: これは、 **virtio-mem** デバイスを使用して仮想マシンに割り当てるメモリー量です。ただし、これは VM に対する単なるリクエストであり、VM が適切に設定されていない場合など、正常に解決されない可能性があります。 **requested** サイズは **block** サイズの倍数である必要があります、定義された最大 **size** を超えることはできません。
 - **current**: これは、 **virtio-mem** デバイスが仮想マシンに提供する現在のサイズを表します。たとえば、リクエストを完了できない場合や VM を再起動する場合など、 **current** サイズは、 **requested** サイズとは異なる場合があります。
 - **alias**: これは、 **libvirt** コマンドでデバイスを編集する場合など、目的の **virtio-mem** デバイスを指定するために使用できるオプションのユーザー定義のエイリアスです。 **libvirt** のすべてのユーザー定義のエイリアスは、"ua-" 接頭辞で始まる必要があります。これらの特定のパラメーターとは別に、 **libvirt** は、 **virtio-mem** デバイスを他の PCI デバイスと同様に処理します。VM に接続された PCI デバイスの管理の詳細については、[仮想デバイスの管理](#) を参照してください。
4. XML ファイルを使用して、定義された **virtio-mem** デバイスを仮想マシンにアタッチします。たとえば、 **virtio-mem-device.xml** で定義された 2 つのデバイスを実行中の仮想マシン **testguest1** に永続的にアタッチするには、次のコマンドを実行します。

```
# virsh attach-device testguest1 virtio-mem-device.xml --live --config
```

--live オプションは、実行中の仮想マシンにのみデバイスを接続します。再起動後に永続性は維持されません。 **--config** オプションは、設定の変更を永続化します。 **--live** オプションを指定せずに、デバイスをシャットダウンした仮想マシンに接続することもできます。

5. **Optional:** 実行中の仮想マシンに接続されている **virtio-mem** デバイスの **requested** サイズを動的に変更するには、**virsh update-memory-device** コマンドを使用します。

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 4GiB
```

この例では、以下が適用されます。

- **testguest1** は、更新する仮想マシンです。
- **--alias ua-virtiomem0** は、以前に定義されたエイリアスで指定された **virtio-mem** デバイスです。
- **--requested-size 4GiB** は、**virtio-mem** デバイスの **requested** サイズを 4 GiB に変更します。



警告

requested サイズを減らして実行中の仮想マシンからメモリーをアンプラグすると、信頼性が低下する可能性があります。このプロセスが成功するかどうかは、使用中のメモリーラインニングポリシーなど、さまざまな要因によって決まります。

場合によっては、その時点でホットプラグされたメモリーの量を変更できないため、ゲストオペレーティングシステムが要求を正常に完了できないことがあります。

さらに、RHEL 8.10 仮想マシンでは、実行中の仮想マシンからメモリーをアンプラグすることはデフォルトで無効になっています。

6. **オプション:** シャットダウンした仮想マシンから **virtio-mem** デバイスを取り外すには、**virsh detach-device** コマンドを使用します。

```
# virsh detach-device testguest1 virtio-mem-device.xml
```

7. **オプション:** 実行中の仮想マシンから **virtio-mem** デバイスを取り外すには、以下を実行します。

- a. **virtio-mem** デバイスの **requested** サイズを 0 に変更します。そうしないと、実行中の仮想マシンから **virtio-mem** デバイスを取り外す試行が失敗します。

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 0
```

- b. 実行中の仮想マシンから **virtio-mem** デバイスを取り外します。

```
# virsh detach-device testguest1 virtio-mem-device.xml
```

検証

- 仮想マシンで、利用可能な RAM を確認し、合計量にホットプラグされたメモリーが含まれているかを確認します。

```
# free -h
```

```
      total used free shared buff/cache available
Mem:  31Gi  5.5Gi 14Gi  1.3Gi  11Gi    23Gi
Swap:  8.0Gi  0B   8.0Gi
```

```
# numactl -H
```

```
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 29564 MB
node 0 free: 13351 MB
node distances:
node 0
  0: 10
```

- 実行中の仮想マシンの XML 設定を表示して、プラグイン RAM の現在の容量をホストで表示することもできます。

```
# virsh dumpxml testguest1
```

```
<domain type='kvm'>
  <name>testguest1</name>
  ...
  <currentMemory unit='GiB'>31</currentMemory>
  ...
  <memory model='virtio-mem'>
    <target>
      <size unit='GiB'>48</size>
      <node>0</node>
      <block unit='MiB'>2</block>
      <requested unit='GiB'>16</requested>
      <current unit='GiB'>16</current>
    </target>
    <alias name='ua-virtiomem0'>
      <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function='0x00'>
    ...
  </domain>
```

この例では、以下が適用されます。

- **<currentMemory unit='GiB'>31</currentMemory>** は、すべてのソースから VM で利用可能な合計 RAM を表します。
- **<current unit='GiB'>16</current>** は、**virtio-mem** デバイスが提供するプラグイン RAM の現在のサイズを表します。

関連情報

- [virtio-mem の概要](#)
- [仮想マシンでのメモリーのオンライン化設定](#)

18.4.3.4. メモリーのオンライン化設定の比較

実行中の RHEL 仮想マシンにメモリーをアタッチする場合 (メモリーのホットプラグとも呼ばれます)、仮想マシン (VM) のオペレーティングシステムでホットプラグされたメモリーをオンライン状態に設定する必要があります。そうしないと、システムはメモリーを使用できなくなります。

次の表は、利用可能なメモリーのオンライン化設定を選択する際の主な考慮事項をまとめたものです。

表18.1 メモリーのオンライン化設定の比較

設定名	仮想マシンからのメモリーのアンプラグ	メモリーゾーンの不均等性が発生するリスク	潜在的なユースケース	目的のワークロードのメモリー要件
online_movable	ホットプラグされたメモリーを確実に取り外すことができます。	あり	比較的少量のメモリーのホットプラグ	ほとんどがユーザー空間のメモリー
auto-movable	ホットプラグされたメモリーの可動部分は確実に取り外すことができます。	最小	大量のメモリーのホットプラグ	ほとんどがユーザー空間のメモリー
online_kernel	ホットプラグされたメモリーは確実に取り外すことができません。	なし	信頼性の低いメモリーの取り外しは許容されます。	ユーザー空間またはカーネル空間のメモリー

ゾーン不均衡 とは、Linux メモリーゾーンの1つに、使用可能なメモリーページがないことです。**ゾーン不均衡** になると、システムのパフォーマンスに悪影響を及ぼす可能性があります。たとえば、移動不可能な割り当てが原因で空きメモリーが不足すると、カーネルがクラッシュする可能性があります。通常、移動可能な割り当てには、主にユーザー空間のメモリーページが含まれ、移動不可能な割り当てには、主にカーネル空間のメモリーページが含まれています。

関連情報

- [Onlining and Offlining Memory Blocks](#)
- [Zone Imbalances](#)
- [仮想マシンでのメモリーのオンライン化設定](#)

18.4.4. 関連情報

- [Attaching devices to virtual machines.](#)

18.5. 仮想マシンの I/O パフォーマンスの最適化

仮想マシンの入出力 (I/O) 機能は、仮想マシンの全体的な効率を大幅に制限する可能性があります。これに対処するために、ブロック I/O パラメーターを設定して、仮想マシンの I/O を最適化できます。

18.5.1. 仮想マシンにおけるブロック I/O のチューニング

複数のブロックデバイスが、複数の仮想マシンで使用されている場合は、I/O ウェイトを変更して特定の仮想デバイスの I/O の優先度を調整することが重要になる場合があります。

デバイスの I/O ウェイトを上げると、I/O 帯域幅の優先度が高まるため、より多くのホストリソースが提供されます。同様に、デバイスのウェイトを下げると、ホストのリソースが少なくなります。



注記

各デバイスの **ウェイト** の値は **100** から **1000** の範囲内でなければなりません。もしくは、値を **0** にすると、各デバイスのリストからそのデバイスを削除できます。

手順

仮想マシンのブロック I/O パラメーターを表示および設定するには、以下を行います。

1. 仮想マシンの現在の **<blkio>** パラメーターを表示します。

```
# virsh dumpxml VM-name
```

```
<domain>
[...]
<blkiotune>
  <weight>800</weight>
  <device>
    <path>/dev/sda</path>
    <weight>1000</weight>
  </device>
  <device>
    <path>/dev/sdb</path>
    <weight>500</weight>
  </device>
</blkiotune>
[...]
</domain>
```

2. 指定したデバイスの I/O ウェイトを編集します。

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

たとえば、次の例では、**testguest1** 仮想マシンの **/dev/sda** デバイスの重みを 500 に変更します。

```
# virsh blkiotune testguest1 --device-weights /dev/sda, 500
```

18.5.2. 仮想マシンのディスク I/O スロットリング

複数の仮想マシンが同時に実行する場合は、過剰なディスク I/O により、システムパフォーマンスに影響が及ぶ可能性があります。KVM 仮想化のディスク I/O スロットリングでは、仮想マシンからホストマシンに送られるディスク I/O 要求に制限を設定する機能を利用できます。これにより、仮想マシンが共有リソースを過剰に使用し、その他の仮想マシンのパフォーマンスに影響を及ぼすことを防ぐことができます。

ディスク I/O スロットリングを有効にするには、仮想マシンに割り当てられた各ブロックデバイスからホストマシンに送られるディスク I/O 要求に制限を設定します。

手順

1. **virsh domblklist** コマンドを使用して、指定された仮想マシン上のすべてのディスクデバイスの名前をリスト表示します。

```
# virsh domblklist rollin-coal
Target  Source
-----
vda     /var/lib/libvirt/images/rollin-coal.qcow2
sda     -
sdb     /home/horridly-demanding-processes.iso
```

2. スロットルする仮想ディスクがマウントされているホストブロックデバイスを見つけます。たとえば、前の手順の **sdb** 仮想ディスクをスロットリングする場合は、以下の出力では、ディスクが **/dev/nvme0n1p3** パーティションにマウントされていることを示しています。

```
$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
zram0                               252:0  0    4G  0 disk [SWAP]
nvme0n1                             259:0  0 238.5G  0 disk
├─nvme0n1p1                         259:1  0   60M  0 part /boot/efi
├─nvme0n1p2                         259:2  0    1G  0 part /boot
├─nvme0n1p3                         259:3  0 236.9G  0 part
└─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0  0 236.9G  0 crypt /home
```

3. **virsh blkiotune** コマンドを使用して、ブロックデバイスの I/O 制限を設定します。

```
# virsh blkiotune VM-name --parameter device,limit
```

以下の例は、**rollin-coal** 仮想マシン上の **sdb** ディスクを毎秒 1000 の読み書き操作にスロットリングし、毎秒 50 MB の読み書きスループットにスロットリングします。

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec /dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

関連情報

- ディスク I/O スロットリングは、異なる顧客に属する仮想マシンが同じホストで実行されている場合や、異なる仮想マシンに QoS 保証が提供されている場合など、さまざまな状況で役立ちます。ディスク I/O スロットリングは、低速なディスクをシミュレートするために使用することもできます。
- I/O スロットリングは、仮想マシンに割り当てられた各ブロックデバイスに個別に適用でき、スループットおよび I/O 操作の制限に対応します。
- Red Hat は、**virsh blkdeviotune** コマンドを使用した仮想マシンでの I/O スロットリングの設定はサポートしていません。RHEL 9 を VM ホストとして使用する場合にサポートされていない機能の詳細は、[RHEL 9 仮想化でサポートされていない機能](#) を参照してください。

18.5.3. マルチキュー virtio-scsi の有効化

仮想マシンで **virtio-scsi** ストレージデバイスを使用する場合は、マルチキュー **virtio-scsi** 機能により、ストレージパフォーマンスおよびスケーラビリティが向上します。このため、各仮想 CPU

(vCPU) に別のキューを持たせることが可能になります。また仮想 CPU は、その他の vCPU に影響を及ぼすことなく使用するために、割り込みできるようになります。

手順

- 特定の仮想マシンに対してマルチキュー virtio-scsi サポートを有効にするには、仮想マシンの XML 設定に以下を追加します。ここでの **N** は、vCPU キューの合計数です。

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

18.6. 仮想マシンの CPU パフォーマンスの最適化

vCPU は、ホストマシンの物理 CPU と同様、仮想マシンのパフォーマンスにおいて極めて重要です。したがって、vCPU を最適化すると、仮想マシンのリソース効率に大きな影響を及ぼす可能性があります。vCPU を最適化するには、以下を実行します。

1. 仮想マシンに割り当てられているホスト CPU の数を調整します。これは、[CLI](#) または [Web コンソール](#) を使用して実行できます。
2. vCPU モデルが、ホストの CPU モデルに調整されていることを確認します。たとえば、仮想マシン `testguest1` を、ホストの CPU モデルを使用するように設定するには、次のコマンドを実行します。

```
# virt-xml testguest1 --edit --cpu host-model
```

ARM 64 システムでは、`--cpu host-passthrough` を使用します。

3. [カーネルの同一ページマージ \(KSM\)](#) を管理します。
4. ホストマシンが Non-Uniform Memory Access (NUMA) を使用する場合は、その仮想マシンに対して **NUMA** を設定することもできます。これにより、ホストの CPU およびメモリープロセスが、仮想マシンの CPU およびメモリープロセスにできるだけ近くにマッピングされます。事実上、NUMA チューニングにより、仮想マシンに割り当てられたシステムメモリーへのより効率的なアクセスが可能になります。これにより、vCPU 処理の効果が改善されます。詳細は、[仮想マシンで NUMA の設定](#) および [サンプルの vCPU パフォーマンスチューニングシナリオ](#) を参照してください。

18.6.1. コマンドラインインターフェイスを使用した仮想 CPU の追加と削除

仮想マシンの CPU パフォーマンスを増減するには、仮想マシンに割り当てられた仮想 CPU (vCPU) を追加または削除します。

実行中の仮想マシンで実行する場合、これは vCPU ホットプラグおよびホットアンプラグとも呼ばれます。ただし、RHEL 9 では vCPU のホットアンプラグに対応しておらず、Red Hat ではその使用を強く推奨していません。

前提条件

- **オプション:** ターゲット仮想マシン内の vCPU の現在の状態を表示します。たとえば、仮想マシン `testguest` 上の仮想 CPU 数を表示するには、以下を実行します。

```
# virsh vcpucount testguest
```

```

maximum  config    4
maximum  live       2
current  config    2
current  live       1

```

この出力は、**testguest** が現在 1vCPU を使用していることを示し、1つ以上の vCPU をホットプラグして仮想マシンのパフォーマンスを向上できることを示しています。ただし、再起動後に使用される vCPU の **testguest** 数は 2 に変更され、2 以上の vCPU のホットプラグが可能になります。

手順

1. 仮想マシンに割り当てることができる vCPU の最大数を調整します。これは、仮想マシンの次回起動時に有効になります。

たとえば、仮想マシン **testguest** の vCPU の最大数を 8 に増やすには、次のコマンドを実行します。

```
# virsh setvcpus testguest 8 --maximum --config
```

最大値は、CPU トポロジー、ホストハードウェア、ハイパーバイザー、およびその他の要素によって制限される可能性があることに注意してください。

2. 仮想マシンに割り当てられている現在の仮想 CPU の数を調整し、直前のステップで設定された最大数まで調整します。以下に例を示します。

- 実行中の仮想マシン **testguest** にアタッチされている vCPU を 4 に増やすには、以下を実行します。

```
# virsh setvcpus testguest 4 --live
```

これにより、仮想マシンの次の起動まで、仮想マシンのパフォーマンスおよび **testguest** のホスト負荷のフットプリントが高まります。

- **testguest** 仮想マシンにアタッチされている vCPU の数を永続的に 1 に減らすには、次のコマンドを実行します。

```
# virsh setvcpus testguest 1 --config
```

これにより、仮想マシンの次の起動後に、仮想マシンのパフォーマンスおよび **testguest** のホスト負荷のフットプリントが低下します。ただし、必要に応じて、仮想マシンに追加の vCPU をホットプラグして、一時的にパフォーマンスを向上させることができます。

検証

- 仮想マシンの vCPU の現在の状態に変更が反映されていることを確認します。

```

# virsh vcpucount testguest
maximum  config    8
maximum  live       4
current  config    1
current  live       4

```

関連情報

- [Web コンソールを使用した仮想 CPU の管理](#)

18.6.2. Web コンソールを使用した仮想 CPU の管理

RHEL 9 Web コンソールを使用して、Web コンソールが接続している仮想マシンが使用する仮想 CPU を確認し、設定できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。
2. 概要ペインで、vCPU の数の横にある **編集** をクリックします。
vCPU の詳細ダイアログが表示されます。

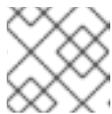
Grid_v2 vCPU details ✕

vCPU count ⓘ <input style="width: 80%;" type="text" value="2"/>	Sockets ⓘ <input style="border-bottom: 1px solid black;" type="text" value="1"/>
vCPU maximum ⓘ <input style="width: 80%;" type="text" value="2"/>	Cores per socket <input style="border-bottom: 1px solid black;" type="text" value="1"/>
	Threads per core <input style="border-bottom: 1px solid black;" type="text" value="1"/>

Apply Cancel

1. 選択した仮想マシンの仮想 CPU を設定します。

- **vCPU 数**: 現在使用中の vCPU の数



注記

vCPU 数は、vCPU 最大値以下にする必要があります。

- **vCPU 最大値** - 仮想マシンに設定できる仮想 CPU の最大数を入力します。この値が **vCPU 数** よりも大きい場合には、vCPU を追加で仮想マシンに割り当てることができます。
- **ソケット** - 仮想マシンに公開するソケットの数を選択します。
- **ソケットごとのコア** - 仮想マシンに公開する各ソケットのコア数を選択します。
- **コアあたりのスレッド** - 仮想マシンに公開する各コアのスレッド数を選択します。
Sockets、**Cores per socket** および **Threads per core** オプションは、仮想マシンの CPU トポロジーを調整することに注意してください。これは、vCPU のパフォーマンスにメリットがあり、ゲスト OS の特定のソフトウェアの機能に影響を与える可能性があります。デプロイメントで別の設定が必要ない場合は、デフォルト値のままにします。

2. **Apply** をクリックします。
仮想マシンに仮想 CPU が設定されます。



注記

仮想 CPU 設定の変更は、仮想マシンの再起動後にのみ有効になります。

関連情報

- [コマンドラインインターフェイスを使用した仮想 CPU の追加と削除](#)

18.6.3. 仮想マシンでの NUMA の設定

以下の方法は、RHEL 9 ホストで、仮想マシンの Non-Uniform Memory Access (NUMA) 設定の設定に使用できます。

前提条件

- ホストが NUMA 対応のマシンである。これを確認するには、**virsh nodeinfo** コマンドを使用して、**NUMA cell(2)** の行を確認します。

```
# virsh nodeinfo
CPU model:      x86_64
CPU(s):         48
CPU frequency:  1200 MHz
CPU socket(s):  1
Core(s) per socket: 12
Thread(s) per core: 2
NUMA cell(s):  2
Memory size:    67012964 KiB
```

行の値が 2 以上であると、そのホストは NUMA に対応しています。

手順

使いやすさのため、自動化ユーティリティとサービスを使用して、仮想マシンの NUMA を設定できます。ただし、手動で NUMA を設定すると、パフォーマンスが大幅に向上する可能性が高くなります。

自動方式

- 仮想マシンの NUMA ポリシーを **Preferred** に設定します。たとえば、仮想マシン **testguest5** に対してこれを行うには、次のコマンドを実行します。

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- ホストで NUMA の自動負荷分散を有効にします。

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- **umad** サービスを起動して、メモリーリソースで仮想マシンの CPU を自動的に調整します。

```
# systemctl start numad
```

手動方式

1. 特定ホストの CPU、またはある範囲の CPU に特定の vCPU スレッドをピンニングします。これは、NUMA 以外のホストおよび仮想マシンでも可能で、vCPU のパフォーマンスを向上させる安全な方法として推奨されています。

たとえば、次のコマンドでは、仮想マシン **testguest6** の vCPU スレッドの 0 から 5 を、ホストの CPU 1、3、5、7、9、11 にそれぞれピンニングします。

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

その後、これが成功したかどうかを確認できます。

```
# virsh vcpupin testguest6
VCPU CPU Affinity
-----
0    1
1    3
2    5
3    7
4    9
5   11
```

2. vCPU スレッドのピンニング後に、指定の仮想マシンに関連付けられた QEMU プロセススレッドを、特定ホスト CPU、またはある範囲の CPU に固定することもできます。たとえば、以下のコマンドは、**testguest6** の QEMU プロセススレッドを CPU 13 および 15 にピンニングし、これが成功したことを確認します。

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
-----
*: 13,15
```

3. これで、特定の仮想マシンに対して割り当てられるホストの NUMA ノードを指定することができます。これにより、仮想マシンの vCPU によるホストメモリーの使用率が向上します。たとえば、次のコマンドでは、ホスト NUMA ノード 3 ~ 5 を使用するように **testguest6** を設定し、これが成功したかどうかを確認します。

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```



注記

最善のパフォーマンス結果を得るためにも、上記の手動によるチューニングメソッドをすべて使用することが推奨されます。

既知の問題

- NUMA チューニングは現在、IBM Z ホストでは実行できません。

関連情報

- [vCPU のパフォーマンスチューニングシナリオ例](#)
- [View the current NUMA configuration of your system](#) using the **numastat** utility

18.6.4. vCPU のパフォーマンスチューニングシナリオ例

最適な vCPU パフォーマンスを得るためにも、たとえば以下のシナリオのように、手動で **vcpupin**、**emulatorpin**、および **numatune** 設定をまとめて使用することが推奨されます。

開始シナリオ

- ホストには以下のハードウェア仕様があります。
 - 2つの NUMA ノード
 - 各ノードにある 3つの CPU コア
 - 各コアにある 2スレッド

このようなマシンの **virsh nodeinfo** の出力は以下のようになります。

```
# virsh nodeinfo
CPU model:      x86_64
CPU(s):         12
CPU frequency:  3661 MHz
CPU socket(s):  2
Core(s) per socket: 3
Thread(s) per core: 2
NUMA cell(s):  2
Memory size:    31248692 KiB
```

- 既存の仮想マシンを変更して、8つの vCPU を使用できるようにします。これは、1つの NUMA ノードに収まらないことを意味します。したがって、各 NUMA ノードに 4つの vCPU を分散し、vCPU トポロジをホストトポロジに可能な限り近づけるようにする必要があります。つまり、指定の物理 CPU のシブリングスレッドとして実行される vCPU は、同じコア上のホストスレッドに固定 (ピンング) される必要があります。詳細は、以下の **ソリューション** を参照してください。

解決方法

1. ホストトポロジに関する情報を取得します。

```
# virsh capabilities
```

この出力には、以下のようなセクションが含まれます。

```
<topology>
<cells num="2">
<cell id="0">
<memory unit="KiB">15624346</memory>
<pages unit="KiB" size="4">3906086</pages>
<pages unit="KiB" size="2048">0</pages>
<pages unit="KiB" size="1048576">0</pages>
```

```

<distances>
  <sibling id="0" value="10" />
  <sibling id="1" value="21" />
</distances>
<cpus num="6">
  <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
  <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
  <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
  <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
  <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
  <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
</cpus>
</cell>
<cell id="1">
  <memory unit="KiB">15624346</memory>
  <pages unit="KiB" size="4">3906086</pages>
  <pages unit="KiB" size="2048">0</pages>
  <pages unit="KiB" size="1048576">0</pages>
  <distances>
    <sibling id="0" value="21" />
    <sibling id="1" value="10" />
  </distances>
  <cpus num="6">
    <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
    <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
    <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
    <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
    <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
    <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
  </cpus>
</cell>
</cells>
</topology>

```

2. (必要に応じて) 適用可能なツールおよびユーティリティーを使用して、仮想マシンのパフォーマンスをテストします。
3. ホストに 1 GiB の Huge Page を設定してマウントします。



注記

1 GiB huge page は、ARM 64 ホストなどの一部のアーキテクチャーおよび設定では使用できない場合があります。

- a. ホストのカーネルコマンドラインに次の行を追加します。

```
default_hugepagesz=1G hugepagesz=1G
```

- b. `/etc/systemd/system/hugetlb-gigantic-pages.service` ファイルを以下の内容で作成します。

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
```

```
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G
```

```
[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/etc/systemd/hugetlb-reserve-pages.sh
```

```
[Install]
WantedBy=sysinit.target
```

- c. `/etc/systemd/hugetlb-reserve-pages.sh` ファイルを以下の内容で作成します。

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
  echo "ERROR: $nodes_path does not exist"
  exit 1
fi

reserve_pages()
{
  echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages 4 node1
reserve_pages 4 node2
```

これにより、4つの1GiBのHuge Pageが `node1` から予約され、さらに別の4つの1GiBのHuge Pageが `node2` から予約されます。

- d. 前の手順で作成したスクリプトを実行ファイルにします。

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

- e. システムの起動時にHuge Page予約を有効にします。

```
# systemctl enable hugetlb-gigantic-pages
```

4. `virsh edit` コマンドを使用して、最適化する仮想マシンのXML設定(この例では `super-VM`)を編集します。

```
# virsh edit super-vm
```

5. 次の方法で仮想マシンのXML設定を調整します。

- 仮想マシンが8つの静的vCPUを使用するように設定します。これを行うには、`<vcpu/>`要素を使用します。
- トポロジーでミラーリングする、対応するホストCPUスレッドに、各vCPUスレッドをピンニングします。これを行うには、`<cputune>` セクションの `<vcpupin/>` 要素を使用します。

上記の **virsh 機能** ユーティリティで示されているように、ホストのCPUスレッドは、各コアで連続的に順次付けされません。また、vCPUスレッドは、同じNUMAノード上のホ

ストのコアの利用可能な最大セットに固定される必要があります。表の図については、以下のトポロジーの例 セクションを参照してください。

手順 a と b の XML 設定は次のようになります。

```
<cputune>
  <vcupin vcpu='0' cpuset='1'>
  <vcupin vcpu='1' cpuset='4'>
  <vcupin vcpu='2' cpuset='2'>
  <vcupin vcpu='3' cpuset='5'>
  <vcupin vcpu='4' cpuset='7'>
  <vcupin vcpu='5' cpuset='10'>
  <vcupin vcpu='6' cpuset='8'>
  <vcupin vcpu='7' cpuset='11'>
  <emulatorpin cpuset='6,9'>
</cputune>
```

- c. 1 GiB の Huge Page を使用するように仮想マシンを設定します。

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'>
  </hugepages>
</memoryBacking>
```

- d. ホスト上で対応する NUMA ノードからメモリーを使用するように、仮想マシンの NUMA ノードを設定します。これを行うには、`<numatune/>` セクションの `<memnode/>` 要素を使用します。

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

- e. CPU モードが **host-passthrough** に設定され、CPU が **passthrough** モードでキャッシュを使用していることを確認します。

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
```

ARM 64 システムでは、`<cache mode="passthrough"/>` 行を省略します。

検証

1. 仮想マシンの XML 設定に、以下のようなセクションが含まれていることを確認します。

```
[...]
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'>
  </hugepages>
</memoryBacking>
```

```

<vcpu placement='static'>8</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'>
  <vcpupin vcpu='1' cpuset='4'>
  <vcpupin vcpu='2' cpuset='2'>
  <vcpupin vcpu='3' cpuset='5'>
  <vcpupin vcpu='4' cpuset='7'>
  <vcpupin vcpu='5' cpuset='10'>
  <vcpupin vcpu='6' cpuset='8'>
  <vcpupin vcpu='7' cpuset='11'>
  <emulatorpin cpuset='6,9'>
</cputune>
<numatune>
  <memory mode="preferred" nodeset="1">
  <memnode cellid="0" mode="strict" nodeset="0">
  <memnode cellid="1" mode="strict" nodeset="1">
</numatune>
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2">
  <cache mode="passthrough">
  <numa>
    <cell id="0" cpus="0-3" memory="2" unit="GiB">
      <distances>
        <sibling id="0" value="10">
        <sibling id="1" value="21">
      </distances>
    </cell>
    <cell id="1" cpus="4-7" memory="2" unit="GiB">
      <distances>
        <sibling id="0" value="21">
        <sibling id="1" value="10">
      </distances>
    </cell>
  </numa>
</cpu>
</domain>

```

2. (必要に応じて) アプリケーションツールおよびユーティリティを使用して仮想マシンのパフォーマンスをテストし、仮想マシンの最適化への影響を評価します。

トポロジーの例

- 以下の表は、ピンングされる必要のある vCPU とホスト CPU 間の接続を示しています。

表18.2 ホストトポロジー

CPU スレッド	0	3	1	4	2	5	6	9	7	10	8	11
コア	0		1		2		3		4		5	
ソケット	0						1					
NUMA ノード	0						1					

表18.3 仮想マシントポロジー

vCPU スレッド	0	1	2	3	4	5	6	7
コア	0		1		2		3	
ソケット	0				1			
NUMA ノード	0				1			

表18.4 ホストと仮想マシントポロジーの組み合わせ

vCPU スレッド			0	1	2	3			4	5	6	7
ホストの CPU スレッド	0	3	1	4	2	5	6	9	7	10	8	11
コア	0		1		2		3		4		5	
ソケット	0				1							
NUMA ノード	0				1							

このシナリオでは、2つの NUMA ノードと 8つの vCPU があります。したがって、4つの vCPU スレッドは各ノードに固定 (ピンング) される必要があります。

また、Red Hat では、ホストシステムの操作のために、各ノードで少なくとも1つの CPU スレッドを使用できるようにしておくことを推奨します。

以下の例では、NUMA ノードにはそれぞれ 3 コアで、2 個のホスト CPU スレッドがあるため、ノード 0 のセットは、以下のように変換できます。

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

18.6.5. カーネルの同一ページマージの管理

Kernel Same-Page Merging (KSM) は、仮想マシン (VM) 間で同一のメモリーページを共有することにより、メモリー密度を向上させます。ただし、KSM を有効にすると CPU 使用率が増加し、ワークロードによっては全体的なパフォーマンスに悪影響を与える可能性があります。

要件に応じて、単一のセッションに対して、または永続的に KSM を有効または無効にすることができます。



注記

RHEL 9 以降では、KSM はデフォルトで無効になっています。

前提条件

- ホストシステムへのルートアクセス。

手順

- KSM を無効にします。
 - KSM をセッション1回分無効にするには、**systemctl** ユーティリティーを使用して **ksm** サービスおよび **ksmtuned** サービスを停止します。

```
# systemctl stop ksm
# systemctl stop ksmtuned
```

- KSM を永続的に無効にするには、**systemctl** ユーティリティーを使用して **ksm** サービスおよび **ksmtuned** サービスを無効にします。

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```



注記

KSM を無効にする前に仮想マシン間で共有されていたメモリーページは、そのまま共有されます。共有を停止するには、以下のコマンドを使用して、システムの **PageKSM** ページをすべて削除します。

```
# echo 2 > /sys/kernel/mm/ksm/run
```

KSM ページを匿名ページに置き換えると、**khugepaged** カーネルサービスは仮想マシンの物理メモリーに透過的なヒュージページを再ビルドします。

- KSM を有効にします。



警告

KSM を有効にすると、CPU 使用率が増大し、CPU 全体のパフォーマンスに影響を及ぼします。

1. **ksmtuned** サービスをインストールします。


```
# dnf install ksmtuned
```
2. サービスを起動します。
 - 単一セッションで KSM を有効にするには、**systemctl** ユーティリティーを使用して **ksm** および **ksmtuned** サービスを開始します。

```
# systemctl start ksm
# systemctl start ksmtuned
```

- KSM を永続的に有効にするには、**systemctl** ユーティリティーを使用して **ksm** サービスおよび **ksmtuned** サービスを有効にします。

```
# systemctl enable ksm
Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service →
/usr/lib/systemd/system/ksm.service

# systemctl enable ksmtuned
Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
/usr/lib/systemd/system/ksmtuned.service
```

18.7. 仮想マシンのネットワークパフォーマンスの最適化

仮想マシンのネットワークインターフェイスカード (NIC) の性質上、仮想マシンは、割り当てられているホストネットワークの帯域幅の一部を失います。これにより、仮想マシンの全体的なワークロード効率が削減されることがあります。以下のヒントは、仮想 NIC (vNIC) のスループットで仮想化の影響を最小限に抑えることができます。

手順

以下の方法のいずれかを使用し、仮想マシンのネットワークパフォーマンスにメリットがあるかどうかを調べます。

vhost_net モジュールの有効化

ホストで **vhost_net** カーネル機能が有効になっていることを確認します。

```
# lsmod | grep vhost
vhost_net      32768  1
vhost          53248  1 vhost_net
tap            24576  1 vhost_net
tun            57344  6 vhost_net
```

このコマンドの出力が空白である場合は、**vhost_net** カーネルモジュールを有効にします。

```
# modprobe vhost_net
```

マルチキュー virtio-net の設定

仮想マシンにマルチキュー **virtio-net** 機能を設定するには、**virsh edit** コマンドを使用して、仮想マシンの XML 設定を編集します。XML で、以下を **<devices>** セクションに追加し、**N** を、仮想マシンの vCPU 数 (最大 16) に変更します。

```
<interface type='network'>
  <source network='default'/>
  <model type='virtio'/>
  <driver name='vhost' queues='N'/>
</interface>
```

仮想マシンが実行中の場合は、再起動して変更を適用します。

ネットワークパケットのバッチ処理

転送パスが長い Linux の仮想マシン設定では、パケットをバッチ処理してからカーネルに送信することで、キャッシュが有効に活用される場合があります。パケットバッチ機能を設定するには、ホストで次のコマンドを実行し、**tap0** を、仮想マシンが使用するネットワークインターフェイスの名前に置き換えます。

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

ホスト NIC が SR-IOV に対応している場合は、vNIC に SR-IOV デバイス割り当てを使用します。詳しくは、[Managing SR-IOV devices](#) を参照してください。

関連情報

- [仮想ネットワークの概要](#)

18.8. 仮想マシンのパフォーマンス監視ツール

最も多くの仮想マシンリソースを消費するものと、仮想マシンで最適化を必要とする部分を認識するために、一般的なパフォーマンス診断ツールや仮想マシン固有のパフォーマンス診断ツールを使用できます。

デフォルトの OS パフォーマンス監視ツール

標準のパフォーマンス評価には、ホストおよびゲストのオペレーティングシステムでデフォルトで提供されるユーティリティーを使用できます。

- RHEL 9 ホストで、root として **top** ユーティリティーまたは **システムモニター** アプリケーションを使用し、出力結果から **qemu** と **virt** を見つけます。これは、仮想マシンが消費しているホストシステムのリソースのサイズを示します。
 - 監視ツールにおいて、**qemu** プロセスまたは **virt** プロセスのいずれかで、ホストの CPU またはメモリの容量を大幅に消費していることが示されている場合は、**perf** ユーティリティーを使用して調査を行います。詳細は以下を参照してください。
 - また、**vhost_net** スレッドプロセス (例: **vhost_net-1234**) が、ホストの CPU 容量を過剰に消費する際に表示される場合は、**multi-queue virtio-net** などの [仮想ネットワークの最適化機能](#) を使用することを検討してください。
- ゲストオペレーティングシステムでは、システムで利用可能なパフォーマンスユーティリティーとアプリケーションを使用して、どのプロセスが最も多くのシステムリソースを消費するかを評価します。
 - Linux システムでは、**top** ユーティリティーを使用できます。
 - Windows システムでは、**Task Manager** アプリケーションを使用できます。

perf kvm

perf ユーティリティーを使用して、RHEL 9 ホストのパフォーマンスに関する仮想化固有の統計を収集および分析できます。これを行うには、以下を行います。

1. ホストに、**perf** パッケージをインストールします。

```
# dnf install perf
```

2. **perf kvm stat** コマンドの1つを使用して、仮想化ホストの **perf** 統計を表示します。

- お使いのハイパーバイザーのリアルタイム監視には、**perf kvm stat live** コマンドを使用します。
 - 一定期間でハイパーバイザーの perf データをログに記録するには、**perf kvm stat record** コマンドを使用してロギングを有効にします。コマンドをキャンセルまたは中断した後、データは **perf.data.guest** ファイルに保存されます。これは、**perf kvm stat report** コマンドを使用して分析できます。
3. **VM-EXIT** イベントとそのディストリビューションのタイプについて **perf** 出力を分析します。たとえば、**PAUSE_INSTRUCTION** イベントは頻繁に存在すべきではありませんが、以下の出力では、このイベントが頻繁に現れ、ホスト CPU が vCPU を適切に処理していないことを示しています。このようなシナリオでは、アクティブな一部の仮想マシンの電源オフ、その仮想マシンからの vCPU の削除、または [vCPU のパフォーマンスの調整](#) を検討してください。

perf kvm stat report

Analyze events for all VMs, all VCPUs:

```

VM-EXIT  Samples Samples%  Time%  Min Time  Max Time  Avg time
EXTERNAL_INTERRUPT  365634  31.59%  18.04%  0.42us  58780.59us
204.08us (+- 0.99%)
MSR_WRITE  293428  25.35%  0.13%  0.59us  17873.02us  1.80us (+-
4.63%)
PREEMPTION_TIMER  276162  23.86%  0.23%  0.51us  21396.03us  3.38us (+-
5.19%)
PAUSE_INSTRUCTION  189375  16.36%  11.75%  0.72us  29655.25us  256.77us
(+ 0.70%)
HLT  20440  1.77%  69.83%  0.62us  79319.41us  14134.56us (+- 0.79%
)
VMCALL  12426  1.07%  0.03%  1.02us  5416.25us  8.77us (+- 7.36%
)
EXCEPTION_NMI  27  0.00%  0.00%  0.69us  1.34us  0.98us (+-
3.50%)
EPT_MISCONFIG  5  0.00%  0.00%  5.15us  10.85us  7.88us (+-
11.67%)

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm stat の出力で問題を知らせる他のイベントタイプには、以下が含まれます。

- **INSN_EMULATION** - 準最適な [仮想マシンの I/O 設定](#) を示します。

perf を使用した仮想化パフォーマンスを監視する方法は、**perf-kvm** man ページを参照してください。

numastat

システムの現在の NUMA 設定を表示するには、**numastat** ユーティリティを使用できます。これは **numactl** パッケージをインストールすることで利用できます。

以下は、4 つの実行中の仮想マシンが含まれるホストを示しています。それぞれは、複数の NUMA ノードからメモリーを取得しています。これは、vCPU のパフォーマンスに対して最適なのではなく、[保証調整](#) です。

```
# numastat -c qemu-kvm
```

```

Per-node process memory usage (in MBs)
PID      Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
-----
51722 (qemu-kvm)  68  16  357 6936  2  3  147  598 8128
51747 (qemu-kvm)  245  11  5  18 5172 2532  1  92 8076
53736 (qemu-kvm)  62  432 1661 506 4851 136  22  445 8116
53773 (qemu-kvm) 1393  3  1  2  12  0  0 6702 8114
-----
Total      1769  463 2024 7462 10037 2672  169 7837 32434

```

一方、以下では、1つのノードで各仮想マシンに提供されているメモリーを示しています。これは、より一層効率的です。

numastat -c qemu-kvm

```

Per-node process memory usage (in MBs)
PID      Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
-----
51747 (qemu-kvm)  0  0  7  0 8072  0  1  0 8080
53736 (qemu-kvm)  0  0  7  0  0  0 8113  0 8120
53773 (qemu-kvm)  0  0  7  0  0  0  1 8110 8118
59065 (qemu-kvm)  0  0 8050  0  0  0  0  0 8051
-----
Total      0  0 8072  0 8072  0 8114 8110 32368

```

18.9. 関連情報

- [Windows 仮想マシンの最適化](#)

第19章 仮想マシンの保護

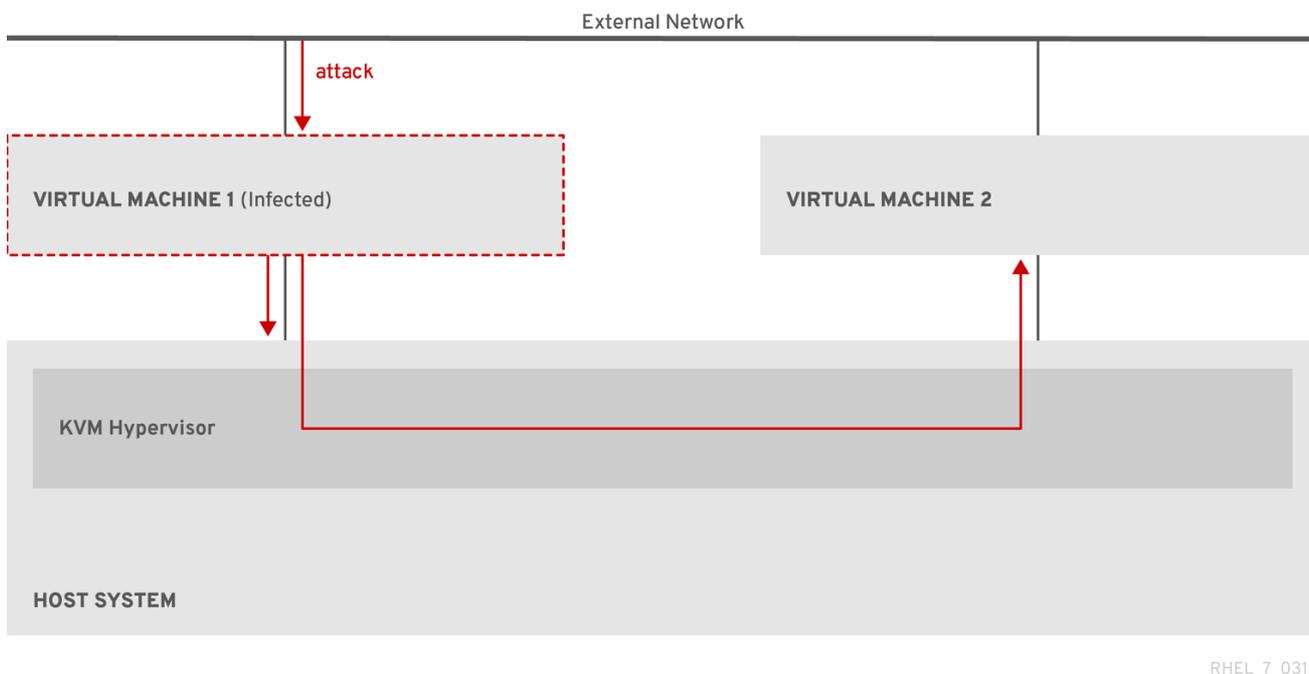
仮想マシン (VM) を使用する RHEL 9 システムの管理者は、仮想マシンのセキュリティをできる限り確保することで、ゲストおよびホストの OS が悪意のあるソフトウェアに感染するリスクを大幅に低減することができます。

本書は、RHEL 9 ホストで [仮想マシンを保護するメカニズム](#) の概要を説明し、仮想マシンのセキュリティを強化する [方法のリスト](#) を提供します。

19.1. 仮想マシンでセキュリティが機能する仕組み

仮想マシンを使用する場合は、複数のオペレーティングシステムを1台のホストマシンに格納できます。このシステムは、ハイパーバイザーを介してホストに接続しますが、通常は仮想ネットワークを介して接続します。したがって、各仮想マシンを、悪意のあるソフトウェアでホストを攻撃するベクトルとして使用できます。また、ホストも、仮想マシンを攻撃するベクトルとして使用できます。

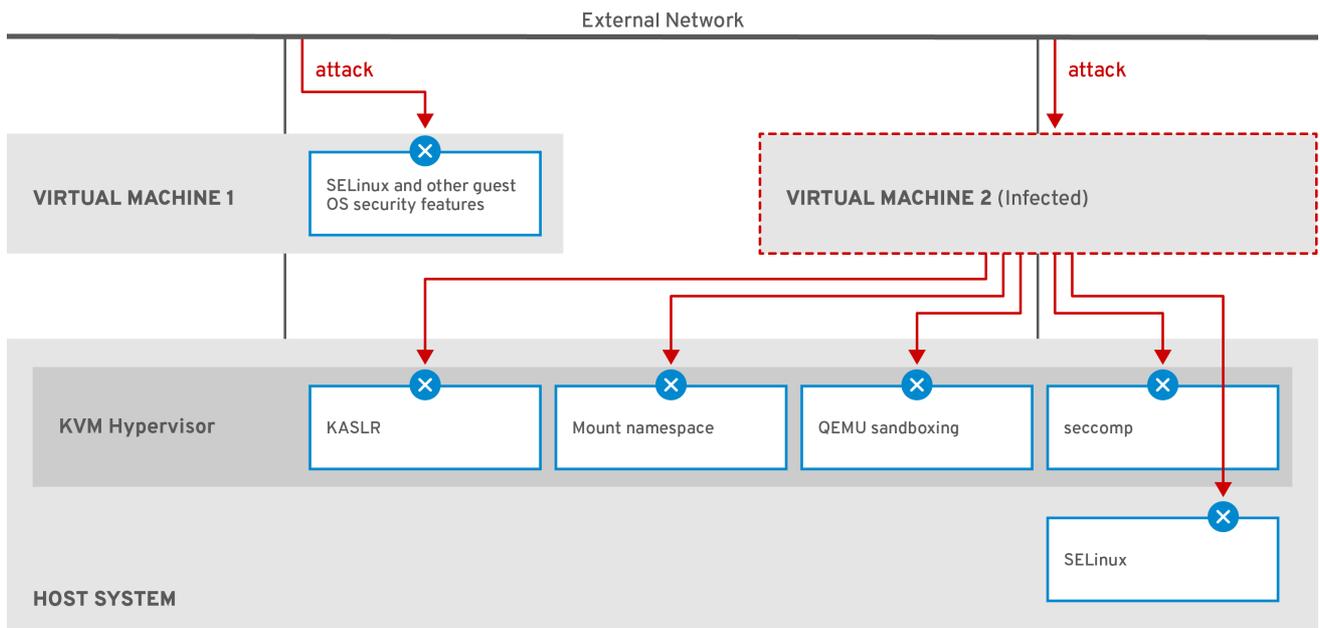
図19.1 仮想化ホストの潜在的なマルウェア攻撃ベクトル



ハイパーバイザーは、ホストカーネルを使用して仮想マシンを管理するため、仮想マシンのオペレーティングシステムで実行しているサービスは、悪意のあるコードをホストシステムに挿入するのによく使用されます。ただし、ホストおよびゲストのシステムで [多数のセキュリティ機能](#) を使用して、このようなセキュリティの脅威からシステムを保護できます。

このような SELinux や QEMU サンドボックスなどの機能は、悪意のあるコードがハイパーバイザーを攻撃し、ホストと仮想マシンとの間の転送をより困難にするさまざまな対策を提供します。

図19.2 仮想化ホストでマルウェア攻撃を阻止



RHEL_7_0319

仮想マシンのセキュリティに対して RHEL 9 が提供する機能の多くは、常にアクティブで、有効または設定する必要がありません。詳細は、[Automatic features for virtual machine security](#) を参照してください。

さらに、仮想マシンおよびハイパーバイザーの脆弱性を最小限に抑えるために、さまざまなベストプラクティスを実行することもできます。詳細は、[Best practices for securing virtual machines](#) を参照してください。

19.2. 仮想マシンのセキュリティ保護に関するベストプラクティス

以下の手順を行うと、仮想マシンが悪意のあるコードに感染し、ホストシステムに侵入するための攻撃ベクトルとして使用されるリスクが大幅に低減します。

ゲストで以下を行います。

- 仮想マシンを、物理マシンと同じように保護します。セキュリティを強化するのに使用できる方法は、ゲスト OS によって異なります。
仮想マシンで RHEL 9 を実行している場合、ゲストシステムのセキュリティを強化する方法の詳細は、[Securing Red Hat Enterprise Linux 9](#) を参照してください。

ホストで以下を行います。

- 仮想マシンをリモートで管理する場合は、SSH などの暗号化ユーティリティと、SSL などのネットワークプロトコルを使用して仮想マシンに接続します。
- SELinux が Enforcing モードであることを確認します。

```
# getenforce
Enforcing
```

SELinux が無効または **Permissive** モードになっている場合は、[SELinux の使用](#) で Enforcing モードを有効にする手順を参照してください。



注記

SELinux の Enforcing モードでは、sVirt RHEL 9 機能も有効になります。これは、仮想化に使用される特別な SELinux ブール値のセットです。この値は [手動で調整](#) でき、仮想マシンのセキュリティーを詳細に管理できます。

- **SecureBoot** で仮想マシンを使用します。
SecureBoot は、仮想マシンが暗号化で署名された OS を実行していることを確認する機能です。これにより、マルウェア攻撃が変更した OS の仮想マシンを起動できなくなります。

SecureBoot は、AMD64 または Intel 64 ホストで OVMF ファームウェアを使用する Linux 仮想マシンをインストールする場合にのみ適用できます。手順は、[Creating a SecureBoot virtual machine](#) を参照してください。

- **qemu-kvm** などの **qemu-*** コマンドは使用しないでください。
QEMU は、RHEL 9 における仮想化アーキテクチャーの必須コンポーネントですが、手動で管理することが難しく、QEMU 設定に誤りがあるとセキュリティーの脆弱性を引き起こす可能性があります。したがって、Red Hat では、大半の **qemu-*** コマンドの使用をサポートしていません。代わりに、ベストプラクティスに従って QEMU のオーケストレーションを行うため、**virsh**、**virt-install**、**virt-xml** などの **libvirt** ユーティリティーを使用します。

ただし、[仮想ディスクイメージの管理](#) には **qemu-img** ユーティリティーがサポートされていることに注意してください。

関連情報

- [SELinux booleans for virtualization in RHEL](#)

19.3. SECUREBOOT の仮想マシンの作成

SecureBoot 機能を使用する Linux 仮想マシンを作成できます。これにより、仮想マシンで暗号で署名された OS を実行できるようになります。これは、仮想マシンのゲスト OS がマルウェアにより変更された場合に役立ちます。このようなシナリオでは、SecureBoot により仮想マシンが起動しなくなり、ホストマシンへのマルウェアの潜在的な拡散を阻止します。

前提条件

- 仮想マシンが Q35 マシンタイプである。
- ホストシステムは AMD64 または Intel 64 アーキテクチャーを使用します。
- **edk2-OVMF** パッケージがインストールされている。

```
# dnf install edk2-ovmf
```

- オペレーティングシステム (OS) のインストールソースがローカルまたはネットワークで利用できる。これには、以下のいずれかの形式を使用できます。
 - インストールメディアの ISO イメージ
 - 既存の仮想マシンインストールのディスクイメージ



警告

RHEL 9 では、ホストの CD-ROM デバイスまたは DVD-ROM デバイスからインストールすることができません。RHEL 9 で利用可能な仮想マシンのインストール方法を使用する際に、インストールソースに CD-ROM または DVD-ROM を選択するとインストールに失敗します。詳細は [Red Hat ナレッジベース](#) を参照してください。

- 任意: インストールをより速く、簡単に設定するために、キックスタートファイルを利用できません。

手順

1. `virt-install` コマンドを使用して、[コマンドラインインターフェイスを使用した仮想マシンの作成](#) で説明されているとおりに仮想マシンを作成します。`--boot` オプションには、`uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd` を使用します。これは、`OVMF_VARS.secboot.fd` ファイルおよび `OVMF_CODE.secboot.fd` ファイルをテンプレートとして使用します。仮想マシンの不揮発性 RAM (NVRAM) 設定のテンプレートとして使用します。これにより、SecureBoot 機能を有効にします。以下に例を示します。

```
# virt-install --name rhel8sb --memory 4096 --vcpus 4 --os-variant rhel9.0 --boot
uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd --disk
boot_order=2,size=10 --disk boot_order=1,device=cdrom,bus=scsi,path=/images/RHEL-9.0-
installation.iso
```

2. 画面の指示に従って、OS のインストール手順を進めます。

検証

1. ゲスト OS がインストールされたら、[グラフィカルゲストコンソール](#) で端末を開いて仮想マシンのコマンドラインにアクセスするか、[SSH を使用](#) してゲスト OS へ接続します。
2. 仮想マシンで SecureBoot が有効になっていることを確認するには、`mokutil --sb-state` コマンドを使用します。

```
# mokutil --sb-state
SecureBoot enabled
```

関連情報

- [Installing RHEL 9 on AMD64, Intel 64, and 64-bit ARM](#)

19.4. 仮想マシンユーザーが使用できるアクションの制限

場合によっては、RHEL 9 でホストされる仮想マシン (VM) のユーザーがデフォルトで実行できるアクションにより、セキュリティリスクが発生する可能性があります。この場合は、ホストマシンで `polkit` ポリシーツールキットを使用するように `libvirt` デーモンを設定して、仮想マシンユーザーに利用可能なアクションを制限できます。

手順

1. **オプション:** お使いの設定に基づいて、**libvirt** に関連するシステムの **polkit** コントロールポリシーが設定されていることを確認してください。

- a. `/usr/share/polkit-1/actions/` ディレクトリーおよび `/usr/share/polkit-1/rules.d/` ディレクトリーにある **libvirt** 関連のファイルすべてを検索します。

```
# ls /usr/share/polkit-1/actions | grep libvirt
# ls /usr/share/polkit-1/rules.d | grep libvirt
```

- b. ファイルを開き、ルール設定を確認します。
polkit 制御ポリシーの構文の読み取りに関する詳細は、**man polkit** を使用します。

- c. **libvirt** 制御ポリシーを変更します。これを行うには、以下を行います。

- i. `/etc/polkit-1/rules.d/` ディレクトリーに新しい **.rules** ファイルを作成します。

- ii. このファイルにカスタムポリシーを追加して保存します。

libvirt コントロールポリシーの詳細および例については、[アップストリームの libvirt ドキュメント](#) を参照してください。

2. **polkit** で決定されるアクセスポリシーを使用するように仮想マシンを設定します。
これを行うには、`/etc/libvirt/` ディレクトリーで仮想化ドライバーのすべての設定ファイルを見つけて、それらの `access_drivers = ["polkit"]` 行のコメントを外します。

```
# find /etc/libvirt/ -name virt*d.conf -exec sed -i 's/#access_drivers = \[ "polkit"
\]/access_drivers = \[ "polkit" \]/g' {} +
```

3. 前の手順で変更した各ファイルで、対応するサービスを再起動します。
たとえば、`/etc/libvirt/virtqemud.conf` を変更した場合には、**virtqemud** サービスを再起動します。

```
# systemctl try-restart virtqemud
```

検証

- VM アクションを制限する予定だったユーザーとして、制限されたアクションの1つを実行します。
たとえば、特権のないユーザーがシステムセッションで作成された VM の表示を制限されている場合は、以下を実行します。

```
$ virsh -c qemu:///system list --all
Id Name      State
-----
```

お使いのシステムに1つ以上の仮想マシンが存在していても、このコマンドで仮想マシンがリスト表示されない場合は、**polkit** は特権のないユーザーのアクションを正常に制限します。

トラブルシューティング

- 現在、**polkit** を使用するように **libvirt** を設定すると、**libvirt-dbus** サービスとの互換性がないため、[RHEL 9 Web コンソールを使用](#) する VM に接続できなくなります。

Web コンソールで仮想マシンのきめ細かいアクセス制御が必要な場合は、カスタム D-Bus ポリシーを作成します。手順については、Red Hat ナレッジベースの [How to configure fine-grained control of Virtual Machines in Cockpit](#) を参照してください。

関連情報

- `man polkit` コマンド
- [polkit アクセス制御ポリシー](#) に関する `libvirt` アップストリーム情報

19.5. 仮想マシンのセキュリティの自動機能

[Best practices for securing virtual machines](#) に記載されている、仮想マシンのセキュリティを向上させる手動での手段に加えて、RHEL 9 で仮想化を使用する場合は、`libvirt` ソフトウェアスイートにより、多くのセキュリティ機能が提供され、自動的に有効になります。これには以下が含まれます。

システムおよびセッションの接続

RHEL 9 で仮想マシン管理に使用できるすべてのユーティリティーにアクセスするには、`libvirt` のシステム接続 (`qemu:///system`) を使用する必要があります。そのためには、システムで root 権限を持っているか、`libvirt` ユーザーグループの一部である必要があります。

`libvirt` グループに属していない root 以外のユーザーは、`libvirt` (`qemu:///session`) のセッション接続にのみアクセスできます。これは、リソースにアクセスする際に、ローカルユーザーのアクセス権限が有効である必要があります。たとえば、セッション接続を使用しても、システム接続で作成された仮想マシンや、その他のユーザーを検出したり、アクセスしたりすることはできません。また、利用可能な仮想マシンネットワーク設定オプションも大幅に制限されます。



注記

RHEL 9 のドキュメントでは、システムの接続特権があることを前提としています。

仮想マシンの分離

個々の仮想マシンは、ホストで孤立したプロセスとして動作し、ホストカーネルにより強制されるセキュリティに依存します。したがって、仮想マシンは、同じホストにある他の仮想マシンのメモリーやストレージを読み取ったり、アクセスすることができません。

QEMU サンドボックス

QEMU コードが、ホストのセキュリティを侵害する可能性のあるシステムコールを実行できない機能です。

KASLR (Kernel Address Space Randomization)

カーネルイメージをデプロイメントする物理アドレスおよび仮想アドレスをランダム化できるようにします。したがって、KASLR は、カーネルオブジェクトの場所に基づいて、ゲストのセキュリティが悪用されるのを防ぎます。

19.6. 仮想化用の SELINUX ブール値

RHEL 9 は、SELinux が Enforcing モードのホストで自動的に有効になる特殊な SELinux ブール値のセットである `sVirt` 機能を提供します。

RHEL 9 システムにおける仮想マシンのセキュリティの詳細な設定には、ハイパーバイザーが特定の 방법으로機能するように、ホストで SELinux のブール値を設定できます。

仮想化関連のブール値とそのステータスのリストを表示するには、`getsebool -a | grep virt` コマンドを実行します。

```
$ getsebool -a | grep virt
[...]
virt_sandbox_use_netlink --> off
virt_sandbox_use_sys_admin --> off
virt_transition_userdomain --> off
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
[...]
```

特定のブール値を有効にするには、root で **setsebool -P boolean_name on** コマンドを実行します。ブール値を無効にするには、**setsebool -P boolean_name off** を使用します。

以下の表は、RHEL 9 で利用可能な仮想化関連のブール値と、その値が有効な場合の動作を示しています。

表19.1 SELinux 仮想化ブール値

SELinux のブール値	説明
staff_use_svirt	非 root ユーザーが仮想マシンを作成して、sVirt に移行できるようになります。
unprivuser_use_svirt	非特権ユーザーが仮想マシンを作成して、sVirt に移行できるようになります。
virt_sandbox_use_audit	サンドボックスコンテナが監査メッセージを送信できるようになります。
virt_sandbox_use_netlink	サンドボックスコンテナでネットリンクシステム呼び出しが使用できるようになります。
virt_sandbox_use_sys_admin	サンドボックスコンテナで sys_admin システム呼び出し (mount 等) が使用できるようになります。
virt_transition_userdomain	仮想プロセスをユーザードメインとして実行できるようになります。
virt_use_comm	virt でシリアルおよびパラレルの通信ポートが使用できるようになります。
virt_use_execmem	制限された仮想ゲストが実行可能メモリーおよび実行可能スタックを使用できるようになります。
virt_use_fusefs	FUSE がマウントしたファイルを virt が読み取りできるようになります。
virt_use_nfs	NFS がマウントしたファイルを virt が管理できるようになります。

SELinux のブール値	説明
virt_use_rawip	virt で rawip ソケットとの通信ができるようになります。
virt_use_samba	CIFS がマウントしたファイルを virt が管理できるようになります。
virt_use_sanlock	制限された仮想ゲストが sanlock と相互作用できるようになります。
virt_use_usb	virt で USB デバイスが使用できるようになります。
virt_use_xserver	仮想マシンで X Window System と相互作用できるようになります。

19.7. IBM Z での IBM SECURE EXECUTION の設定

IBM Z ハードウェアを使用して RHEL 9 ホストを実行する場合は、仮想マシンの IBM Secure Execution を設定して、仮想マシンのセキュリティを強化できます。

IBM Secure Execution (Protected Virtualization と呼ばれる) は、ホストシステムが仮想マシンの状態とメモリーのコンテンツにアクセスできないようにします。その結果、ホストが危険にさらされても、ゲストオペレーティングシステムを攻撃するベクトルとして使用できません。さらに、セキュア実行を使用して、信頼できないホストが仮想マシンから機密情報を取得しないようにすることもできます。

次の手順では、IBM Z ホストの既存の仮想マシンを、セキュアな仮想マシンに変換する方法を説明します。

前提条件

- システムハードウェアに以下のいずれかを使用している。
 - IBM z15 以降
 - IBM LinuxONE III 以降
- Secure Execution 機能がお使いのシステムで有効になっている。確認するには、次のコマンドを実行します。

```
# grep facilities /proc/cpuinfo | grep 158
```

このコマンドで出力が表示された場合には、お使いの CPU は Secure Execution と互換性があります。

- カーネルに Secure Execution のサポートが含まれている。これを確認するには、次のコマンドを実行します。

```
# ls /sys/firmware | grep uv
```

このコマンドで出力が表示された場合には、カーネルで Secure Execution がサポートされています。

- ホストの CPU モデルに **unpack** 機能が含まれている。これを確認するには、次のコマンドを実行します。

```
# virsh domcapabilities | grep unpack
<feature policy='require' name='unpack'/>
```

このコマンドで上記の出力が表示された場合には、お使いの CPU ホストモデルは Secure Execution と互換性があります。

- 仮想マシンの CPU モードが **host-model** に設定されている。これを確認するには、以下を使用します。**vm-name** は、仮想マシンの名前に置き換えます。

```
# virsh dumpxml vm-name | grep "<cpu mode='host-model'/>"
```

このコマンドで出力が表示された場合には、仮想マシンの CPU モデルは正しく設定されています。

- genprotimg** パッケージがホストにインストールされている必要がある。

```
# dnf install genprotimg
```

- IBM Z のホストキーのドキュメントを取得および確認している。この方法は、IBM ドキュメントの [ホストキードキュメントの確認](#) を参照してください。

手順

お使いのホストで、以下の手順を実行します。

- prot_virt=1** カーネルパラメーターをホストの [ブート設定](#) に追加します。

```
# grubby --update-kernel=ALL --args="prot_virt=1"
```

- ブートメニューを更新します。

```
# zipl
```

- virsh edit** を使用して、セキュリティー保護する仮想マシンの XML 設定を変更します。
- <launchSecurity type="s390-pv"/>** を **</devices>** 行の下に追加します。以下に例を示します。

```
[...]
  </memballoon>
</devices>
<launchSecurity type="s390-pv"/>
</domain>
```

- 設定の **<devices>** セクションに **virtio-rng** デバイス(**<rng model="virtio">**)が含まれている場合は、**<rng>** **</rng>** ブロックのすべての行を削除します。
- オプション:** セキュリティー保護する仮想マシンが 32 GiB 以上の RAM を使用している場合は、XML 設定の **<features>****</features>** セクションに **<async-teardown enabled='yes'/>** 行を追加します。
これにより、そのような Secure Execution ゲストの再起動または停止のパフォーマンスが向上します。

セキュリティーを保護する仮想マシンの **ゲストオペレーティングシステム** で、以下の手順を実行します。

1. パラメーターファイルを作成します。以下に例を示します。

```
# touch ~/secure-parameters
```

2. **/boot/loader/entries** ディレクトリーで、最新バージョンのブートローダーエントリーを特定します。

```
# ls /boot/loader/entries -l
[...]
-rw-r--r--. 1 root root 281 Oct 9 15:51 3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
```

3. ブートローダーエントリーからカーネルオプションの行を取得します。

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf | grep options
options root=/dev/mapper/rhel-root
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
```

4. オプションの行の内容と **swiotlb=262144** を作成したパラメーターのファイルに追加します。

```
# echo "root=/dev/mapper/rhel-root rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap swiotlb=262144" > ~/secure-parameters
```

5. IBM Secure Execution イメージを生成します。

たとえば、以下は **secure-parameters** ファイル、**/boot/initramfs-4.18.0-240.el8.s390x.img** 初期 RAM ディスクファイル、および **HKD-8651-000201C048.crt** ホストキードキュメントを使用して、**/boot/vmlinuz-4.18.0-240.el8.s390x** イメージをもとに、セキュアなイメージ (**/boot/secure-image**) を作成します。

```
# genprotimg -i /boot/vmlinuz-4.18.0-240.el8.s390x -r /boot/initramfs-4.18.0-240.el8.s390x.img -p ~/secure-parameters -k HKD-8651-00020089A8.crt -o /boot/secure-image
```

genprotimg ユーティリティーを使用すると、カーネルパラメーター、初期 RAM ディスク、ブートイメージを含む、セキュアなイメージが作成されます。

6. 仮想マシンのブートメニューを更新して、セキュアなイメージから起動します。さらに、**initrd** および **オプション** で始まる行は必要ないので削除します。

たとえば、RHEL 8.3 仮想マシンでは、**/boot/loader/entries/** ディレクトリーでブートメニューの編集が可能です。

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
title Red Hat Enterprise Linux 8.3
version 4.18.0-240.el8.s390x
linux /boot/secure-image
[...]
```

7. ブート可能なディスクイメージの作成

```
# zipl -V
```

- 保護されていない元のファイルを安全に削除します。以下に例を示します。

```
# shred /boot/vmlinuz-4.18.0-240.el8.s390x
# shred /boot/initramfs-4.18.0-240.el8.s390x.img
# shred secure-parameters
```

元のブートイメージ、初期 RAM イメージ、およびカーネルパラメーターファイルは保護されていません。削除しない場合には、Secure Execution が有効になっている仮想マシンで、ハッキングまたは機密データマイニングの攻撃を受ける可能性があります。

検証

- ホストで、**virsh dumpxml** ユーティリティを使用して、セキュアな仮想マシンの XML 設定を確認します。設定には **<launchSecurity type="s390-pv"/>** 要素を含み、**<rng model="virtio">** 行は使用しないでください。

```
# virsh dumpxml vm-name
[...]
<cpu mode='host-model'/>
<devices>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none' io='native'>
      <source file='/var/lib/libvirt/images/secure-guest.qcow2'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <model type='virtio'/>
    </interface>
    <console type='pty'/>
    <memballoon model='none'/>
  </devices>
  <launchSecurity type="s390-pv"/>
</domain>
```

関連情報

- [IBM documentation on starting a secure virtual server](#)
- [IBM documentation on **genprotimg**](#)
- [カーネルコマンドラインパラメーターの設定](#)

19.8. IBM Z 上の仮想マシンへの暗号化コプロセッサの割り当て

IBM Z ホストの仮想マシンでハードウェア暗号化を使用するには、暗号化プロセッサデバイスから仲介デバイスを作成して目的の仮想マシンに割り当てます。詳細な手順は、以下を参照してください。

前提条件

- お使いのホストを IBM Z ハードウェアで実行している。

- 暗号化コプロセッサは、デバイスの割り当てと互換性があります。これを確認するには、コプロセッサの **タイプ** が **CEX4** 以降として表示されているかをチェックします。

```
# lszcrypt -V

CARD.DOMAIN TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH
FUNCTIONS DRIVER
-----
05 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4card
05.0004 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
05.00ab CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
```

- **vfio_ap** カーネルモジュールが読み込まれている。確認するには、次のコマンドを実行します。

```
# lsmod | grep vfio_ap
vfio_ap      24576 0
[...]
```

モジュールを読み込むには、以下を使用します。

```
# modprobe vfio_ap
```

- **s390utils** バージョンは **ap** 処理をサポートしています。

```
# lsudev --list-types
...
ap      Cryptographic Adjunct Processor (AP) device
...
```

手順

1. 仮想マシンに割り当てるデバイスの10進数値を取得します。たとえば、デバイス **05.0004** および **05.00ab** の場合は以下ようになります。

```
# echo "obase=10; ibase=16; 04" | bc
4
# echo "obase=10; ibase=16; AB" | bc
171
```

2. ホストで、デバイスを **vfio-ap** ドライバーに再割り当てします。

```
# chzdev -t ap apmask=-5 aqmask=-4,-171
```



注記

デバイスを永続的に割り当てるには、**-p** フラグを使用します。

3. 暗号化デバイスが正しく再割り当てされていることを確認します。

```
# lszcrypt -V
```

```

CARD.DOMAIN TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH
FUNCTIONS DRIVER
-----
05 CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- cex4card
05.0004 CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- vfio_ap
05.00ab CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- vfio_ap

```

ドメインキューの DRIVER の値が **vfio_ap** に変更されると、再割り当ては成功します。

- 新しい仲介デバイスを定義する XML スニペットを作成します。

以下の例では、永続的な仲介デバイスを定義してそのデバイスにキューを割り当てます。具体的には、この例の **vfio_ap.xml** XML スニペットは、ドメインアダプター **0x05**、ドメインキュー **0x0004** および **0x00ab**、および制御ドメイン **0x00ab** を仲介デバイスに割り当てます。

```

# vim vfio_ap.xml

<device>
  <parent>ap_matrix</parent>
  <capability type="mdev">
    <type id="vfio_ap-passthrough"/>
    <attr name='assign_adapter' value='0x05'/>
    <attr name='assign_domain' value='0x0004'/>
    <attr name='assign_domain' value='0x00ab'/>
    <attr name='assign_control_domain' value='0x00ab'/>
  </capability>
</device>

```

- vfio_ap.xml** XML スニペットから新しい仲介デバイスを作成します。

```

# virsh nodedev-define vfio_ap.xml
Node device 'mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix' defined from
'vfio_ap.xml'

```

- 前の手順で作成した仲介デバイス (この場合は **mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix**) を起動します。

```

# virsh nodedev-start mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix
Device mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix started

```

- 設定が正しく適用されたことを確認します。

```

# cat /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-
passthrough/devices/669d9b23-fe1b-4ecb-be08-a2fabca99b71/matrix
05.0004
05.00ab

```

出力に **vfio-ap** に割り当てられたキューの数値が含まれる場合には、プロセスは成功です。

- 仲介デバイスを仮想マシンに接続します。

- 作成した仲介デバイスの UUID を表示し、次の手順のために保存します。

```

# virsh nodedev-dumpxml mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix

```

```
<device>
  <name>mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix</name>
  <parent>ap_matrix</parent>
  <capability type='mdev'>
    <type id='vfio_ap-passthrough'/>
    <uuid>8f9c4a73-1411-48d2-895d-34db9ac18f85</uuid>
    <iommuGroup number='0'/>
    <attr name='assign_adapter' value='0x05'/>
    <attr name='assign_domain' value='0x0004'/>
    <attr name='assign_domain' value='0x00ab'/>
    <attr name='assign_control_domain' value='0x00ab'/>
  </capability>
</device>
```

- b. 暗号化カード仲介デバイスの XML ファイルを作成して開きます。以下に例を示します。

```
# vim crypto-dev.xml
```

- c. 以下の行をファイルに追加して保存します。**uuid** 値は、手順 a で取得した UUID に置き換えます。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='8f9c4a73-1411-48d2-895d-34db9ac18f85'/>
  </source>
</hostdev>
```

- d. XML ファイルを使用して、仲介デバイスを仮想マシンに接続します。たとえば、**crypto-dev.xml** ファイルで定義されたデバイスを、実行中の **testguest1** 仮想マシンに永続的に接続するには、次のコマンドを実行します。

```
# virsh attach-device testguest1 crypto-dev.xml --live --config
```

--live オプションは、実行中の仮想マシンにのみデバイスを接続します。再起動後に永続性は維持されません。**--config** オプションは、設定の変更を永続化します。**--config** オプションのみを使用すると、デバイスをシャットダウンした仮想マシンに接続できます。

各 UUID は、一度に1つの仮想マシンにしか割り当てることができないのでご注意ください。

検証

1. ゲストオペレーティングシステムが、割り当てられた暗号化デバイスを検出していることを確認します。

```
# lszcrypt -V
```

```
CARD.DOMAIN TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH
FUNCTIONS DRIVER
-----
```

```
05 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4card
05.0004 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
05.00ab CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
```

ゲストオペレーティングシステムでのこのコマンドの出力は、利用可能な暗号化コプロセッサデバイスが同じホストの論理パーティションで表示される出力と同じです。

2. ゲストオペレーティングシステムで、制御ドメインが暗号化デバイスに正常に割り当てられていることを確認します。

```
# lszcrypt -d C

DOMAIN 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
-----
00 . . . . . U . . . . .
10 . . . . .
20 . . . . .
30 . . . . .
40 . . . . .
50 . . . . .
60 . . . . .
70 . . . . .
80 . . . . .
90 . . . . .
a0 . . . . . B . . . . .
b0 . . . . .
c0 . . . . .
d0 . . . . .
e0 . . . . .
f0 . . . . .
-----

C: Control domain
U: Usage domain
B: Both (Control + Usage domain)
```

lszcrypt -d C で暗号化デバイスマトリックスに **U** と **B** の交差が表示された場合、制御ドメインの割り当ては成功しています。

19.9. WINDOWS 仮想マシンでの標準ハードウェアセキュリティの有効化

Windows 仮想マシンを保護するには、Windows デバイスの標準ハードウェア機能を使用して基本的なレベルのセキュリティを有効にします。

前提条件

- 最新の WHQL 認定 VirtIO ドライバーがインストールされている。
- 仮想マシンのファームウェアが UEFI ブートに対応している。
- **edk2-OVMF** パッケージをホストマシンにインストールしている。

```
# {PackageManagerCommand} install edk2-ovmf
```

- ホストマシンに **vTPM** パッケージをインストールしている。

```
# {PackageManagerCommand} install swtpm libtpms
```

- 仮想マシンが Q35 マシンアーキテクチャーを使用している。

- Windows インストールメディアを使用している。

手順

1. TPM 2.0 を有効にするには、仮想マシンの XML 設定の **<devices>** セクションに以下のパラメーターを追加します。

```
<devices>
[...]
<tpm model='tpm-crb'>
  <backend type='emulator' version='2.0'>
</tpm>
[...]
</devices>
```

2. UEFI モードで Windows をインストールします。詳細は、[SecureBoot の仮想マシンの作成](#) を参照してください。
3. Windows 仮想マシンに virtio ドライバーをインストールします。詳細は、[Windows ゲストへの virtio ドライバーのインストール](#) を参照してください。
4. UEFI でセキュアブートを有効にします。詳細は、[セキュアブート](#) を参照してください。

検証

- Windows マシンの [デバイスのセキュリティー](#) ページに、以下のメッセージが表示されていることを確認します。

Settings > Update & Security > Windows Security > Device Security

Your device meets the requirements for standard hardware security.

19.10. WINDOWS 仮想マシンでの拡張ハードウェアセキュリティーの有効化

Windows 仮想マシンをさらにセキュアにするために、コード整合性の仮想化ベースの保護 (HVCI (Hypervisor-Protected Code Integrity) と呼ばれます) を有効にできます。

前提条件

- 標準のハードウェアセキュリティーが有効になっていることを確認します。詳細は、[Windows 仮想マシンでの標準ハードウェアセキュリティーの有効化](#) を参照してください。
- Hyper-V enlightenments が有効になっていることを確認します。詳細は、[Hyper-V enlightenment の有効化](#) を参照してください。

手順

1. Windows VM の XML 設定を開きます。次の例では、**Example-L1 VM** の設定を開きます。

```
# virsh edit Example-L1
```

2. **<cpu>** セクションで、CPU モードを指定し、ポリシーフラグを追加します。



重要

- Intel CPU の場合は、**vmx** ポリシーフラグを有効にします。
- AMD CPU の場合は、**svm** ポリシーフラグを有効にします。
- カスタム CPU を指定したくない場合は、**<cpu mode>** を **host-passthrough** として設定できます。

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'>
  <feature policy='require' name='vmx'>
</cpu>
```

3. XML 設定を保存し、仮想マシンを再起動します。
4. 仮想マシンオペレーティングシステムで、**Core isolation details** ページに移動します。
Settings > Update & Security > Windows Security > Device Security > Core isolation details
5. スイッチを切り替えて、**メモリーの整合性** を有効にします。
6. 仮想マシンを再起動します。



注記

HVCI を有効にするその他の方法は、関連する Microsoft ドキュメントを参照してください。

検証

- Windows 仮想マシンの **デバイスのセキュリティー** ページに、以下のメッセージが表示されていることを確認します。

Settings > Update & Security > Windows Security > Device Security

Your device meets the requirements for enhanced hardware security.

- または、Windows 仮想マシンのシステム情報を確認します。
 - a. コマンドプロンプトで **msinfo32.exe** を実行します。
 - b. **Virtualization-based security Services Running** の下に **Credential Guard, Hypervisor enforced Code Integrity** がリスト表示されているかどうかを確認します。

第20章 ホストとその仮想マシン間でのファイルの共有

ホストシステムと、そのホストが実行する仮想マシンとの間で、データを共有することが頻繁に必要なになります。これを迅速かつ効率的に行うために、システムに NFS ファイル共有をセットアップできます。または、**virtiofs** を使用して、Linux および Windows 仮想マシンとデータを共有することもできます。

20.1. NFS を使用したホストとその仮想マシン間でのファイルの共有

RHEL 9 ホストシステムと仮想マシン間でファイルを効率的に共有するために、仮想マシンがマウントしてアクセスできる NFS 共有をエクスポートできます。

ただし、Linux VM では、通常、**virtiofs** 機能を使用する方が便利です。

前提条件

- **nfs-utils** パッケージがホストにインストールされている。

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

- **NAT** または **bridge** タイプの仮想ネットワークが、ホストを仮想マシンに接続するように設定されている。
- (必要に応じて) セキュリティーを強化する場合は、仮想マシンが NFS バージョン 4 以降と互換性があることを確認してください。

手順

1. ホストで、ネットワークファイルシステム (NFS) として共有するファイルを含むディレクトリーをエクスポートします。
 - a. 既存のディレクトリーを仮想マシンと共有します。既存のディレクトリーを共有したくない場合は、新しいディレクトリーを作成します。

```
# mkdir shared-files
```

- b. ホストからファイルを共有するために各仮想マシンの IP アドレスを取得します (例: **testguest1** と **testguest2**)。

```
# virsh domifaddr testguest1
Name      MAC address      Protocol  Address
-----
vnet0     52:53:00:84:57:90  ipv4     192.0.2.2/24

# virsh domifaddr testguest2
Name      MAC address      Protocol  Address
-----
vnet1     52:53:00:65:29:21  ipv4     192.0.2.3/24
```

- c. ホスト上の **/etc/exports** ファイルを編集し、共有するディレクトリー、共有する仮想マシンの IP、および追加のオプションを含む行を追加します。

```
/home/<username>/Downloads/<shared_directory>/<VM1-IP(options)> <VM2-IP(options)>
```

```
...
```

たとえば、以下は、`testguest1` および `testguest2` があるホストの `/usr/local/shared-files` ディレクトリーを共有し、仮想マシンがディレクトリーのコンテンツを編集できるようにします。

```
/usr/local/shared-files/ 192.0.2.2(rw,sync) 192.0.2.3(rw,sync)
```



注記

Windows の仮想マシンとディレクトリーを共有するには、Windows NFS クライアントに共有ディレクトリーへの書き込み権限があることを確認する必要があります。`/etc/exports` ファイルでは、`all_squash`、`anonuid`、および `anongid` オプションを使用できます。

```
/usr/local/shared-files/
192.0.2.2(rw,sync,all_squash,anonuid=<directory-owner-UID>,anongid=<directory-owner-GID>)
```

`<directory-owner-UID>` と `<directory-owner-GID>` は、ホスト上の共有ディレクトリーを所有するローカルユーザーの UID と GID です。

NFS クライアントの権限を管理するその他のオプションについては、[NFS サービスの保護](#) ガイドに従ってください。

- d. 更新したファイルシステムをエクスポートします。

```
# exportfs -a
```

- e. `nfs-server` サービスを起動します。

```
# systemctl start nfs-server
```

- f. ホストシステムの IP アドレスを取得して、仮想マシンに共有ディレクトリーをマウントします。

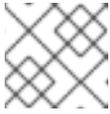
```
# ip addr
...
5: virbr0: [BROADCAST,MULTICAST,UP,LOWER_UP] mtu 1500 qdisc noqueue state UP group default qlen 1000
link/ether 52:54:00:32:ff:a5 brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.0.2.255 scope global virbr0
valid_lft forever preferred_lft forever
...
```

関連するネットワークはホストと仮想マシンを接続してファイルを共有することに注意してください。通常、これは `virbr0` です。

2. `/etc/exports` ファイルで指定されている Linux 仮想マシンに共有ディレクトリーをマウントします。

```
# mount 192.0.2.1:/usr/local/shared-files /mnt/host-share
```

- **192.0.2.1**: ホストの IP アドレスです。
- **/usr/local/shared-files**: ホスト上のエクスポートされたディレクトリーへのファイルシステムパスです。
- **/mnt/host-share**: 仮想マシン上のマウントポイントです。



注記

マウントポイントは空のディレクトリーである必要があります。

3. **/etc/exports** ファイルに指定されているように Windows 仮想マシンに共有ディレクトリーをマウントするには、次の手順を実行します。
 - a. 管理者として PowerShell シェルプロンプトを開きます。
 - b. Windows に **NFS-Client** パッケージをインストールします。
 - i. サーバーバージョンにインストールするには、次のように入力します。

```
# Install-WindowsFeature NFS-Client
```

- ii. デスクトップバージョンにインストールするには、次のように入力します。

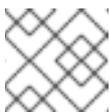
```
# Enable-WindowsOptionalFeature -FeatureName ServicesForNFS-ClientOnly,  
ClientForNFS-Infrastructure -Online -NoRestart
```

- c. ホストによってエクスポートされたディレクトリーを Windows 仮想マシンにマウントします。

```
# C:\Windows\system32\mount.exe -o anon \\192.0.2.1\usr\local\shared-files Z:
```

この例では、以下が適用されます。

- **192.0.2.1**: ホストの IP アドレスです。
- **/usr/local/shared-files**: ホスト上のエクスポートされたディレクトリーへのファイルシステムパスです。
- **Z:**: マウントポイントのドライブ文字です。



注記

システムで使用されていないドライブ文字を選択する必要があります。

検証

- ホストと仮想マシン間でファイルを共有できるように、仮想マシン上の共有ディレクトリーの内容をリスト表示します。

```
$ ls <mount_point>  
shared-file1 shared-file2 shared-file3
```

この例では、`<mount_point>` を、マウントされた共有ディレクトリーへのファイルシステムパスに置き換えます。

関連情報

- [NFS サーバーのデプロイ](#)

20.2. VIRTIOFS を使用したホストとその仮想マシン間でのファイルの共有

virtiofs を使用すると、ホストと仮想マシン (VM) の間で、ローカルファイルシステムの構造と同じように機能するディレクトリーツリーとしてファイルを共有できます。virtiofs を使用して、次のタスクを実行できます。

- [ホストと仮想マシンの間でファイルを共有する](#)
- [ホストと Windows 仮想マシンの間でファイルを共有する](#)
- [Web コンソールを使用してホストと仮想マシンの間でファイルを共有する](#)
- [Web コンソールを使用してホストと仮想マシンの間の共有ファイルを削除する](#)

20.2.1. virtiofs を使用したホストとその仮想マシン間でのファイルの共有

RHEL 9 をハイパーバイザーとして使用する場合は、**virtiofs** 機能を使用して、ホストシステムとその仮想マシン間でファイルを効率的に共有できます。

前提条件

- 仮想化は、RHEL 9 ホストでインストールされ、有効になります。
- 仮想マシンと共有するディレクトリーがある。既存のディレクトリーを共有しない場合は、`shared-files` などの新しいディレクトリーを作成します。

```
# mkdir /root/shared-files
```

- データを共有する仮想マシンは、ゲストオペレーティングシステムとして Linux ディストリビューションを使用します。

手順

1. 仮想マシンと共有するホストの各ディレクトリーを、仮想マシンの XML 設定の virtiofs ファイルシステムとして設定します。
 - a. 目的の仮想マシンの XML 設定を開きます。

```
# virsh edit vm-name
```

- b. 仮想マシンの XML 設定の `<devices>` に、以下のようなエントリーを追加します。

```
<filesystem type='mount' accessmode='passthrough'>  
  <driver type='virtiofs'/>  
  <binary path='/usr/libexec/virtiofsd' xattr='on'/>
```

```
<source dir='/root/shared-files'/>
<target dir='host-file-share'/>
</filesystem>
```

この例では、ホストの **/root/shared-files** ディレクトリーを、仮想マシンの **host-file-share** として表示するように設定します。

2. 仮想マシンの共有メモリーを設定します。これを行うには、XML 設定の **<domain>** セクションに共有メモリーバックアップを追加します。

```
<domain>
[...]
<memoryBacking>
  <access mode='shared'/>
</memoryBacking>
[...]
</domain>
```

3. 仮想マシンを起動します。

```
# virsh start vm-name
```

4. ゲストオペレーティングシステムにファイルシステムをマウントします。以下の例では、Linux ゲストオペレーティングシステムで事前に設定した **host-file-share** ディレクトリーをマウントします。

```
# mount -t virtiofs host-file-share /mnt
```

検証

- 共有ディレクトリーが仮想マシンからアクセス可能になり、ディレクトリーに保存されているファイルを開けるようになりました。

既知の問題と制限

- **noatime**、**strictatime** など、アクセス時間に関連するファイルシステムのマウントオプションは **virtiofs** では機能しない可能性が高く、Red Hat はその使用を推奨しません。

トラブルシューティング

- **virtiofs** がユースケースに最適でない場合、またはシステムでサポートされていない場合は、代わりに **NFS** を使用できます。

20.2.2. virtiofs を使用したホストと Windows 仮想マシン間でのファイルの共有

RHEL 9 をハイパーバイザーとして使用する場合、**virtiofs** 機能と **virtio-win** パッケージを使用して、ホストシステムと Windows 仮想マシン (VM) の間でファイルを効率的に共有できます。



注記

virtiofs.exe コマンドと **-i** パラメーターを使用して、Windows 仮想マシン上で **virtiofs** サービスを大文字と小文字を区別しないモードで実行できます。

前提条件

- virtiofs を使用するように仮想マシンの XML 設定ファイルを設定している。詳細は、[「virtiofs を使用したホストとその仮想マシン間でのファイルの共有」](#) を参照してください。
- **virtio** ドライバーインストールメディアを仮想マシンに割り当てている。
- Windows 仮想マシンに **virtio-win** パッケージがインストールされている。詳細は、[Installing virtio drivers on a Windows guest](#) を参照してください。

手順

1. Windows 仮想マシンで WinFsp をインストールします。これを行うには、**virtio-win** ISO イメージをマウントし、**winfsp** MSI インストーラーを起動して、プロンプトに従います。インストールウィザードの **Custom Setup** ウィンドウで、仮想マシンにインストールする機能を選択します。

2. virtiofs サービスを起動します。

```
# sc start VirtioFsSvc
```

3. **This PC** に移動します。
File Explorer → **This PC**

virtiofs は、Windows 仮想マシン上で、**z:** から逆順に遡る、使用可能な最初のドライブ文字として使用できます。たとえば、**my_viofs (Z:)** です。



重要

共有ディレクトリーにアクセスするには、仮想マシンを再起動するたびに virtiofs サービスを再起動する必要があります。

4. **オプション:** 追加の virtiofs インスタンスを設定するには、以下を実行します。

- a. virtiofs サービスを停止します。

```
# sc stop VirtioFsSvc
# sc config VirtioFsSvc start=demand
```

- b. 複数の virtiofs インスタンスをセットアップするように WinFSP.Launcher サービスを設定します。

```
# "C:\Program Files (x86)\WinFsp\bin\fsreg.bat" virtiofs "<path to the binary>\virtiofs.exe" "-t %1 -m %2"
```

- c. virtiofs インスタンスをドライブにマウントします。
たとえば、**mount_tag0** タグを持つ virtiofs を **Y:** ドライブにマウントするには、以下を実行します。

```
"C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" start virtiofs viofsY mount_tag0 Y:
```

- d. 前の手順を繰り返して、すべての virtiofs インスタンスをマウントします。

- e. virtiofs インスタンスをアンマウントするには、以下を実行します。

```
"C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" stop virtiofs viofsY
```

検証

1. Windows 仮想マシンで、**This PC**に移動します。

File Explorer → This PC

- virtiofs サービスのセットアップ時にマウントポイントを指定しなかった場合は、**z:** から逆順に遡る、使用可能な最初のドライブ文字が使用されます。
- 複数の virtiofs インスタンスをセットアップした場合、それらはインスタンスに割り当てた文字を持つドライブとして表示されます。

20.2.3. Web コンソールで virtiofs を使用してホストとその仮想マシン間でファイルを共有する

RHEL Web コンソールを使用すると、**virtiofs** 機能を使用して、ホストシステムとその仮想マシン (VM) 間でファイルを効率的に共有できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- 仮想マシンと共有するディレクトリーがある。既存のディレクトリーを共有したくない場合は、たとえば **centurion** という名前の新しいディレクトリーを作成します。

```
# mkdir /home/centurion
```

- データを共有する仮想マシンは、ゲストオペレーティングシステムとして Linux ディストリビューションを使用します。

手順

1. **仮想マシン** インターフェイスで、ファイルを共有する VM をクリックします。新しいページが開き、選択した VM に関する基本情報を含む **概要** セクションと、**コンソール** セクションが表示されます。
2. **共有ディレクトリー** までスクロールします。**共有ディレクトリー** セクションには、その VM と共有されているホストファイルとディレクトリーに関する情報、および共有ディレクトリーを **追加** または **削除** するためのオプションが表示されます。

Shared directories ⓘ		Add shared directory
Source path	Mount tag	
/home/centurion/	Ace	Remove

3. **共有ディレクトリーの追加** をクリックします。**ホストディレクトリーをゲストと共有する** ダイアログが表示されます。

Share a host directory with the guest ×

Shared host directories need to be manually mounted inside the VM ?

Source path ?

Mount tag ?

▼ Hide additional options

Extended attributes Enable/disable extended attributes (xattr) on files and directories

4. 以下の情報を入力します。

- ソースパス - 共有するホストディレクトリーへのパス。
- マウントタグ - VM がディレクトリーをマウントするために使用するタグ。

5. 追加オプションを設定します。

- 拡張属性 - 共有ファイルとディレクトリーで拡張属性 **xattr** を有効にするかどうかを設定します。

6. **共有** をクリックします。

選択したディレクトリーは仮想マシンと共有されます。

検証

- 共有ディレクトリーが VM でアクセス可能であり、そのディレクトリーに保存されているファイルを開くことができるようになったことを確認します。

20.2.4. virtiofs を使用したホストとその仮想マシン間での共有ファイルの削除のための Web コンソールの使用

RHEL Web コンソールを使用すると、**virtiofs** 機能を使用してホストシステムとその仮想マシン (VM) 間で共有されているファイルを削除できます。

前提条件

- Web コンソールの仮想マシンプラグインが [システムにインストールされている](#)。
- ディレクトリーは仮想マシンによって使用されなくなりました。

手順

1. **仮想マシン** インターフェイスで、共有ファイルを削除する VM をクリックします。新しいページが開き、選択した VM に関する基本情報を含む **概要** セクションと、**コンソール** セクションが表示されます。
2. **共有ディレクトリー** までスクロールします。

共有ディレクトリー セクションには、その VM と共有されているホストファイルとディレクトリーに関する情報、および共有ディレクトリーを **追加** または **削除** するためのオプションが表示されます。

Shared directories ⓘ		Add shared directory
Source path	Mount tag	
/home/centurion/	Ace	Remove

- 仮想マシンとの共有を解除するディレクトリーの横にある **Remove** をクリックします。ファイルシステムの削除 ダイアログが表示されます。

 **Remove filesystem?** ×

This filesystem will be removed from Grid_v2:

Source path	/home/centurion/
Mount tag	Ace

Remove Cancel

- 削除** をクリックします。
選択したディレクトリーは VM と共有されていません。

検証

- 共有ディレクトリーは、VM で使用できなくなり、アクセスできなくなります。

第21章 WINDOWS 仮想マシンのインストールおよび管理

RHEL 9 ホスト上の仮想マシン (VM) でゲストオペレーティングシステムとして Microsoft Windows を使用するには、Red Hat は、これらの VM が正しく実行されるように追加の手順を実行することを推奨します。

このため、以下のセクションでは、ホストマシンに Windows 仮想マシンをインストールし、最適化する方法を説明します。また、Windows 仮想マシンにドライバーをインストールし、設定する方法を説明します。

21.1. WINDOWS 仮想マシンのインストール

RHEL 9 ホスト上に完全に仮想化された Windows マシンを作成し、仮想マシン (VM) 内でグラフィカルな Windows インストーラーを起動し、インストールされた Windows ゲストオペレーティングシステム (OS) を最適化できます。

VM を作成し、Windows ゲスト OS をインストールするには、**virt-install** コマンドまたは RHEL 9 Web コンソールを使用します。

前提条件

- ローカルまたはネットワークで利用可能な OS のインストールソースがある。次のいずれかになります。
 - インストールメディアの ISO イメージ
 - 既存の仮想マシンインストールのディスクイメージ
- KVM **virtio** ドライバーを備えた記憶媒体がある。
このメディアを作成するには、[Preparing virtio driver installation media on a host machine](#) を参照してください。
- Windows 11 をインストールする場合は、**edk2-ovmf** パッケージ、**swtpm** パッケージ、および **libtpms** パッケージをホストにインストールする必要があります。

手順

1. 仮想マシンを作成します。手順については、[仮想マシンの作成](#) を参照してください。ただし、次の詳細に注意してください。
 - **virt-install** ユーティリティを使用して仮想マシンを作成する場合は、次のオプションをコマンドに追加します。
 - KVM **virtio** ドライバーを備えた記憶媒体。以下に例を示します。

```
--disk path=/usr/share/virtio-win/virtio-win.iso,device=cdrom
```

- インストールする Windows バージョン。たとえば、Windows 10 および 11 の場合は、以下ようになります。

```
--os-variant win10
```

Windows のバージョンと、適切なオプションのリストを表示するには、次のコマンドを実行します。

```
# osinfo-query os
```

- Windows 11 をインストールする場合は、**Unified Extensible Firmware Interface(UEFI)** および **仮想 Trusted Platform Module (vTPM)** を有効にします。

```
--boot uefi
```

- Web コンソールを使用して仮想マシンを作成する場合は、**仮想マシンの新規作成** 画面の **オペレーティングシステム** フィールドで Windows のバージョンを指定します。
 - Windows 11 および Windows Server 2022 より前のバージョンの Windows をインストールする場合は、**Create and run** をクリックしてインストールを開始します。
 - Windows 11 をインストールする場合、または追加の Windows Server 2022 機能を使用する場合は、**Create and edit** をクリックして確認し、CLI を使用して UEFI および vTPM を有効にします。
 - 仮想マシンの XML 設定を開きます。

```
# virsh edit windows-vm
```

- firmware='efi'** オプションを **os** 要素に追加します。

```
<os firmware='efi'>
  <type arch='x86_64' machine='pc-q35-6.2'>hvm</type>
  <boot dev='hd'/>
</os>
```

- devices** 要素内に **tpm** デバイスを追加します。

```
<devices>
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
</devices>
```

- Virtual machines** テーブルで **Install** をクリックして、Windows のインストールを開始します。
- 仮想マシンに Windows OS をインストールします。
Windows オペレーティングシステムのインストール方法は、関連する Microsoft インストールドキュメントを参照してください。
 - Web コンソールを使用して仮想マシンを作成する場合は、**Disks** インターフェイスを使用して、virtio ドライバーを含むストレージメディアを仮想マシンに接続します。手順は、[Web コンソールで既存ディスクを仮想マシンに割り当てる手順](#) を参照してください。
 - Windows ゲスト OS で、KVM **virtio** ドライバーを設定します。詳細は、[Installing KVM paravirtualized drivers for Windows virtual machines](#) を参照してください。

関連情報

- [Windows 仮想マシンの最適化](#)

- [Windows 仮想マシンでの標準ハードウェアセキュリティの有効化](#)
- [Windows 仮想マシンでの拡張ハードウェアセキュリティの有効化](#)
- [仮想マシンの XML 設定例](#)

21.2. WINDOWS 仮想マシンの最適化

RHEL 9 でホストされる仮想マシンで Microsoft Windows をゲストオペレーティングシステムとして使用すると、ゲストのパフォーマンスが悪影響を受ける可能性があります。

そのため、Red Hat は、以下のいずれかを組み合わせて実行して Windows 仮想マシンを最適化することを推奨しています。

- 準仮想化ドライバーの使用。詳細は、[Installing KVM paravirtualized drivers for Windows virtual machines](#) を参照してください。
- Hyper-V Enlightenment の有効化。詳細は、[Hyper-V enlightenment の有効化](#) を参照してください。
- NetKVM ドライバーパラメーターの設定。詳細は、[Configuring NetKVM driver parameters](#) を参照してください。
- Windows バックグラウンドプロセスの最適化または無効化。詳細は、[Optimizing background processes on Windows virtual machines](#) を参照してください。

21.2.1. Windows 仮想マシン用の KVM 準仮想化ドライバーのインストール

Windows 仮想マシンのパフォーマンスを改善する主な方法は、Windows 用の KVM 準仮想化 (**virtio**) ドライバーをゲストオペレーティングシステムにインストールすることです。



注記

virtio-win ドライバーは、各 **virtio-win** リリースの時点で利用可能な Windows 10 および 11 の最新リリースに対して認定 (WHQL) されています。ただし、**virtio-win** ドライバーは広くテストされており、Windows 10 および 11 の以前のビルドでも正しく機能することが期待されます。

Windows VM にドライバーをインストールするには、次の操作を実行します。

1. ホストマシンにインストールメディアを準備します。詳細は、[Preparing virtio driver installation media on a host machine](#) を参照してください。
2. インストールメディアを既存の Windows 仮想マシンに接続するか、新しい Windows 仮想マシンを作成するときに接続します。詳細については、[RHEL への Windows 仮想マシンのインストール](#) を参照してください。
3. Windows ゲストオペレーティングシステムに **virtio** ドライバーをインストールします。詳細は、[Installing virtio drivers on a Windows guest](#) を参照してください。
4. **QEMU Guest Agent** を Windows ゲストオペレーティングシステムにインストールします。詳細は、[Windows ゲストへの QEMU ゲストエージェントのインストール](#) を参照してください。

21.2.1.1. Windows virtio ドライバーの仕組み

準仮想化ドライバーは仮想マシンのパフォーマンスを向上し、I/O レイテンシーを下げ、ベアメタルレベルまでスループットを増加させます。Red Hat は、I/O 負荷の高いタスクとアプリケーションを実行する仮想マシンには、準仮想化ドライバーを使用することを推奨します。

virtio ドライバーは、KVM ホストで実行する Windows 仮想マシンで利用可能な、KVM の準仮想化デバイスドライバーです。これらのドライバーは、**virtio-win** パッケージにより提供されます。これには、以下のドライバーが含まれます。

- ブロック (ストレージ) デバイス
- ネットワークインターフェイスコントローラー
- ビデオコントローラー
- メモリーバルーニングデバイス
- 準仮想化シリアルポートデバイス
- エントロピーソースデバイス
- 準仮想化パニックデバイス
- マウス、キーボード、タブレットなどの入力デバイス
- エミュレートされたデバイスの小規模セット



注記

エミュレートされたデバイス、**virtio** デバイス、および割り当てられたデバイスの詳細は、[仮想デバイスの管理](#) を参照してください。

KVM の virtio ドライバーを使用すると、以下の Microsoft Windows バージョンが、物理システムのように動作することが見込まれます。

- Windows Server バージョン - Red Hat ナレッジベースの [Certified guest operating systems for Red Hat Enterprise Linux with KVM](#) を参照してください。
- Windows デスクトップ (サーバー以外) バージョン:
 - Windows 10 (32 ビット版および 64 ビット版)
 - Windows 11 (64 ビット)

21.2.1.2. ホストマシンでの virtio ドライバーインストールメディアの準備

KVM **virtio** ドライバーを Windows 仮想マシン (VM) にインストールまたは更新するには、最初にホストマシンで **virtio** ドライバーインストールメディアを準備する必要があります。これを行うには、**virtio-win** パッケージで提供される **.iso** ファイルをストレージデバイスとして Windows VM に接続します。

前提条件

- RHEL 9 ホストシステムで仮想化が有効になっていることを確認する。詳細は、[仮想化の有効化](#) を参照してください。
- 仮想マシンへのルートアクセス権限があることを確認します。

手順

1. サブスクリプションデータを更新します。

```
# subscription-manager refresh
All local data refreshed
```

2. **virtio-win** パッケージの最新バージョンを入手します。

- **virtio-win** がインストールされていない場合:

```
# dnf install -y virtio-win
```

- **virtio-win** がインストールされている場合:

```
# dnf upgrade -y virtio-win
```

インストールが成功すると、**virtio-win** ドライバーファイルが `/usr/share/virtio-win/` ディレクトリーで使用可能になります。これには、**ISO** ファイルと、ディレクトリーにドライバーファイルを持つ **drivers** ディレクトリー (各アーキテクチャーと対応している Windows バージョン用のファイル) が含まれます。

```
# ls /usr/share/virtio-win/
drivers/ guest-agent/ virtio-win-1.9.9.iso virtio-win.iso
```

3. **virtio-win.iso** ファイルをストレージデバイスとして Windows VM に接続します。

- [新しい Windows 仮想マシンを作成する](#) ときは、**virt-install** コマンドオプションを使用してファイルをアタッチします。
- 既存の Windows 仮想マシンにドライバーをインストールする場合は、**virt-xml** ユーティリティーを使用してファイルを CD-ROM としてアタッチします。

```
# virt-xml WindowsVM --add-device --disk virtio-win.iso,device=cdrom
Domain 'WindowsVM' defined successfully.
```

関連情報

- [Installing the virtio driver on the Windows guest operating system](#)

21.2.1.3. Windows ゲストへの virtio ドライバーのインストール

KVM **virtio** ドライバーを Windows ゲストオペレーティングシステムにインストールするには、ドライバーを含むストレージデバイスを (仮想マシン (VM) の作成時または作成後に) 追加し、Windows ゲストオペレーティングシステムにドライバーをインストールする必要があります。

この手順では、グラフィカルインターフェイスを使用してドライバーをインストールする手順を説明します。[Microsoft Windows インストーラー \(MSI\)](#) コマンドラインインターフェイスを使用することもできます。

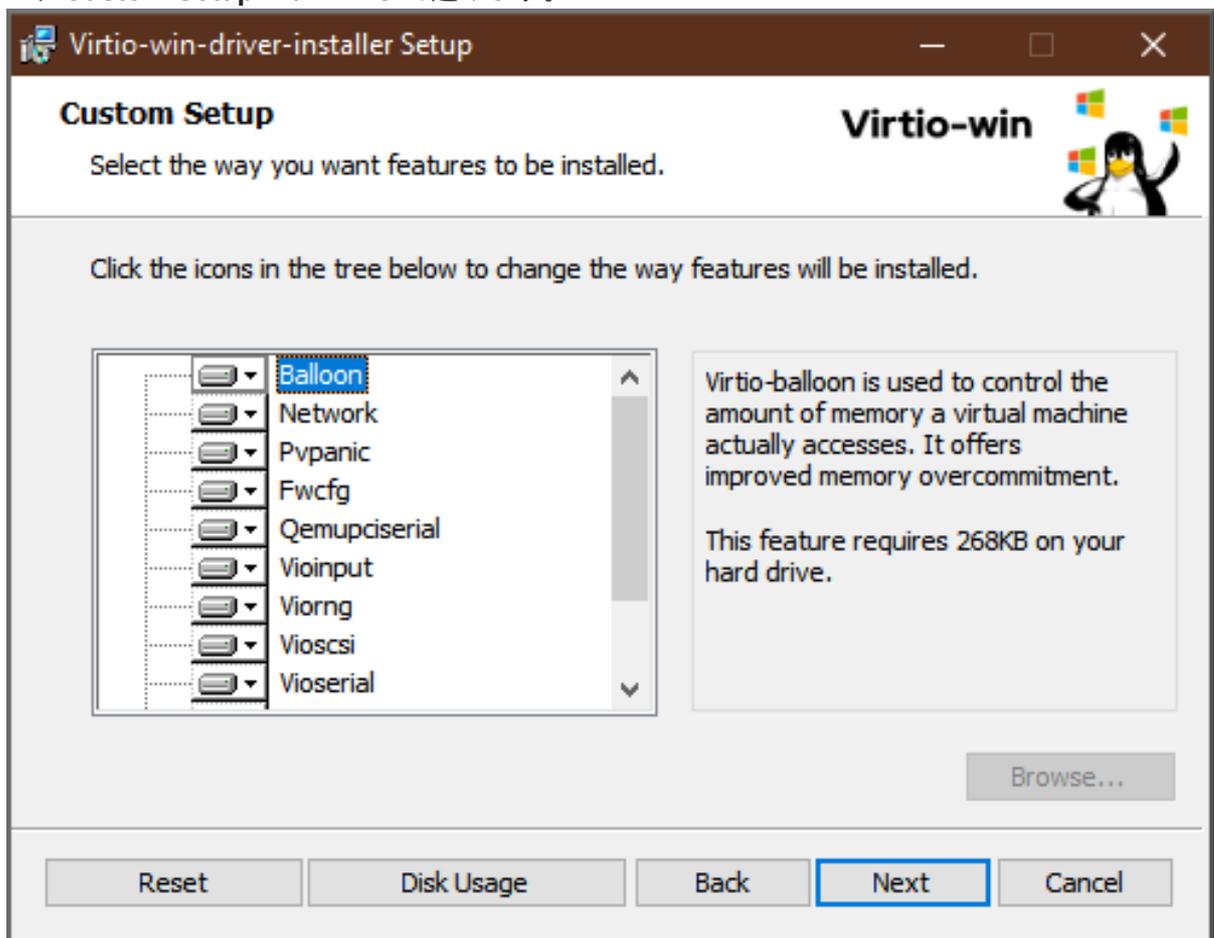
前提条件

この手順では、仮想マシンをインストールする前に、仮想マシンをインストールする必要があることを確認する必要があります。

- KVM **virtio** ドライバーを備えたインストールメディアを仮想マシンに接続する必要があります。メディアの準備手順は、[Preparing virtio driver installation media on a host machine](#) を参照してください。

手順

1. Windows ゲストオペレーティングシステムで、**File Explorer** アプリケーションを開きます。
2. **この PC** をクリックします。
3. **デバイスおよびドライブ** ペインで、**virtio-win** メディアを開きます。
4. 仮想マシンにインストールされているオペレーティングシステムに基づいて、次のいずれかのインストーラーを実行します。
 - 32 ビットオペレーティングシステムを使用している場合は、**virtio-win-gt-x86.msi** インストーラーを実行します。
 - 64 ビットオペレーティングシステムを使用している場合は、**virtio-win-gt-x64.msi** インストーラーを実行します。
5. 表示された **Virtio-win-driver-installer** セットアップウィザードで、表示される指示に従い、**Custom Setup** ステップまで進みます。



6. カスタムセットアップ画面で、インストールするデバイスドライバーを選択します。推奨されるドライバーセットが自動的に選択され、ドライバーの説明がリストの右側に表示されます。
7. **次へ** をクリックして、**インストール** をクリックします。
8. インストールが完了したら、**完了** をクリックします。

9. 仮想マシンを再起動してドライバーのインストールを完了します。

検証

1. Windows 仮想マシンで、**Device Manager** に移動します。
 - a. Start をクリックします。
 - b. **Device Manager** を検索します。
2. デバイスが正しいドライバーを使用していることを確認します。
 - a. デバイスをクリックして **Driver Properties** ウィンドウを開きます。
 - b. **Driver** タブに移動します。
 - c. **Driver Details** をクリックします。

次のステップ

- NetKVM ドライバーをインストールした場合は、Windows ゲストのネットワークパラメーターの設定も必要になる場合があります。詳細は、[Configuring NetKVM driver parameters](#) を参照してください。

21.2.1.4. Windows ゲストでの virtio ドライバーの更新

Windows ゲストオペレーティングシステム (OS) で KVM **virtio** ドライバーを更新するには、Windows OS バージョンがサポートしている場合、**Windows Update** サービスを使用できます。そうでない場合は、Windows 仮想マシン (VM) に接続されている **virtio** ドライバーインストールメディアからドライバーを再インストールします。

前提条件

- [virtio ドライバーがインストールされた Windows ゲスト OS](#)。
- **Windows Update** を使用しない場合は、最新の KVM **virtio** ドライバーを含むインストールメディアを Windows 仮想マシンに接続する必要があります。メディアの準備手順は、[Preparing virtio driver installation media on a host machine](#) を参照してください。

手順 1: Windows Update を使用してドライバーを更新する

Windows 10、Windows Server 2016 以降のオペレーティングシステムでは、**Windows Update** グラフィカルインターフェイスを使用して、ドライバーの更新が利用可能かどうかを確認します。

1. Windows 仮想マシンを起動し、ゲスト OS にログインします。
2. **Optional updates** ページに移動します。
Settings → **Windows Update** → **Advanced options** → **Optional updates**
3. Red Hat, Inc. からのすべての更新をインストールします。

手順 2: ドライバーを再インストールして更新する

Windows 10 および Windows Server 2016 より前のオペレーティングシステムの場合、または OS が **Windows Update** にアクセスできない場合は、ドライバーを再インストールします。これにより、Windows ゲスト OS のネットワーク設定がデフォルト (DHCP) に復元されます。カスタマイズした

ネットワーク設定を保持する場合は、バックアップを作成し、**netsh** ユーティリティーを使用して復元する必要があります。

1. Windows 仮想マシンを起動し、ゲスト OS にログインします。
2. Windows コマンドプロンプトを開きます。
 - a. **Super+R** キーボードショートカットを使用します。
 - b. 表示されるウィンドウで、**cmd** と入力し、**Ctrl+Shift+Enter** を押して管理者として実行します。
3. Windows コマンドプロンプトを使用して、OS ネットワーク設定をバックアップします。

```
C:\WINDOWS\system32\netsh dump > backup.txt
```

4. 付属のインストールメディアから KVM **virtio** ドライバーを再インストールします。次のいずれかを行います。
 - Windows コマンドプロンプトを使用してドライバーを再インストールします。ここで、**X** はインストールメディアのドライブ文字です。次のコマンドは、すべての **virtio** ドライバーをインストールします。
 - 64 ビット vCPU を使用している場合:

```
C:\WINDOWS\system32\msiexec.exe /i X:\virtio-win-gt-x64.msi /passive /norestart
```

- 32 ビット vCPU を使用している場合:

```
C:\WINDOWS\system32\msiexec.exe /i X:\virtio-win-gt-x86.msi /passive /norestart
```

- VM を再起動せずに、[グラフィカルインターフェイスを使用して](#) ドライバーを再インストールします。
5. Windows コマンドプロンプトを使用して、OS ネットワーク設定を復元します。

```
C:\WINDOWS\system32\netsh -f backup.txt
```

6. 仮想マシンを再起動してドライバーのインストールを完了します。

関連情報

- [Windows Update に関する Microsoft ドキュメント](#)

21.2.1.5. Windows ゲストでの QEMU ゲストエージェントの有効化

RHEL ホストが Windows 仮想マシン上で [特定の操作のサブセット](#) を実行できるようにするには、QEMU ゲストエージェント (GA) を有効にする必要があります。これを行うには、QEMU ゲストエージェントインストーラーを含むストレージデバイスを、既存の仮想マシンに追加するか、新しい仮想マシンを作成するときに追加し、Windows ゲストオペレーティングシステムにドライバーをインストールします。

グラフィカルインターフェイスを使用してゲストエージェント (GA) をインストールするには、以下の手順を参照してください。コマンドラインインターフェイスで GA をインストールするには、[Microsoft Windows Installer \(MSI\)](#) を使用してください。

前提条件

- ゲストエージェントを含むインストールメディアが仮想マシンに接続されている。メディアの準備手順は、[Preparing virtio driver installation media on a host machine](#) を参照してください。

手順

1. Windows ゲストオペレーティングシステムで、**File Explorer** アプリケーションを開きます。
2. **この PC** をクリックします。
3. **デバイスおよびドライブ** ペインで、**virtio-win** メディアを開きます。
4. **guest-agent** フォルダーを開きます。
5. 仮想マシンにインストールされているオペレーティングシステムに基づいて、次のいずれかのインストーラーを実行します。
 - 32 ビットオペレーティングシステムを使用している場合は、**qemu-ga-i386.msi** インストーラーを実行します。
 - 64 ビットオペレーティングシステムを使用している場合は、**qemu-ga-x86_64.msi** インストーラーを実行します。
6. **オプション:** ホストと Windows ゲスト間の通信インターフェイスとして準仮想化シリアルドライバ (**virtio-serial**) を使用する場合は、**virtio-serial** ドライバーが Windows ゲストにインストールされていることを確認します。**virtio** ドライバーのインストールの詳細は、[Windows ゲストへの virtio ドライバーのインストール](#) を参照してください。

検証

1. Windows 仮想マシンで、**Services** ウィンドウに移動します。
Computer Management > Services
2. **QEMU Guest Agent** のステータスが **Running** であることを確認します。

関連情報

- [QEMU ゲストエージェントを必要とする仮想化機能](#)

21.2.2. Hyper-V Enlightenment の有効化

Hyper-V Enlightenment では、KVM が Microsoft Hyper-V ハイパーバイザーをエミュレートするための方法を利用できます。これにより、Windows 仮想マシンのパフォーマンスが向上します。

以下のセクションは、対応している Hyper-V Enlightenment と、その有効化に関する情報を提供します。

21.2.2.1. Windows 仮想マシンでの Hyper-V Enlightenment の有効化

Hyper-V Enlightenment により、RHEL 9 ホストで実行している Windows 仮想マシン (VM) でパフォーマンスが向上します。それを有効にする方法は、次を参照してください。

手順

1. **virsh edit** コマンドを使用して、仮想マシンの XML 設定を表示します。以下に例を示します。

```
# virsh edit windows-vm
```

2. XML の **<features>** セクションに、以下の **<hyperv>** サブセクションを追加します。

```
<features>
[...]
```

```
<hyperv>
  <relaxed state='on' />
  <vapic state='on' />
  <spinlocks state='on' retries='8191' />
  <vpindex state='on' />
  <runtime state='on' />
  <synic state='on' />
  <stimer state='on'>
    <direct state='on' />
  </stimer>
  <frequencies state='on' />
  <reset state='on' />
  <relaxed state='on' />
  <time state='on' />
  <tlbflush state='on' />
  <reenlightenment state='on' />
  <stimer state='on'>
    <direct state='on' />
  </stimer>
  <ipi state='on' />
  <crash state='on' />
  <evmcs state='on' />
</hyperv>
[...]
```

```
</features>
```

XML に **<hyperv>** サブセクションが含まれている場合は、上記のように変更します。

3. 以下のように、設定の **クロック** セクションを変更します。

```
<clock offset='localtime'>
...
  <timer name='hypervclock' present='yes' />
</clock>
```

4. XML 設定を保存して終了します。
5. 仮想マシンが実行中の場合は再起動します。

検証

- **virsh dumpxml** コマンドを使用して、実行中の仮想マシンの XML 設定を表示します。次のセグメントが含まれている場合は、Hyper-V Enlightenment が仮想マシンで有効になります。

```
<hyperv>
```

```

<relaxed state='on'/>
<vapic state='on'/>
<spinlocks state='on' retries='8191'/>
<vpindex state='on'/>
<runtime state='on' />
<synic state='on'/>
<stimer state='on'/>
<frequencies state='on'/>
<reset state='on'/>
<relaxed state='on'/>
<time state='on'/>
<tlbflush state='on'/>
<reenlightenment state='on'/>
<stimer state='on'>
  <direct state='on'/>
</stimer>
<ipi state='on'/>
<crash state='on'/>
<evmcs state='on'/>
</hyperv>

<clock offset='localtime'>
...
  <timer name='hypervclock' present='yes'/>
</clock>

```

21.2.2.2. 設定可能な Hyper-V Enlightenment

特定の Hyper-V 機能を設定して Windows 仮想マシンを最適化できます。以下の表では、設定可能な Hyper-V 機能およびその値に関する情報を提供します。

表21.1 設定可能な Hyper-V 機能

Enlightenment	説明	値
crash	<p>仮想マシンがクラッシュした場合には、情報とログを保存するために使用できる MSR を仮想マシンに提供します。QEMU ログの情報を利用できます。</p> <div style="display: flex; align-items: center;">  <div> <p>注記</p> <p>hv_crash が有効になっている場合、Windows クラッシュダンプは作成されません。</p> </div> </div>	on、off

Enlightenment	説明	値
evmcs	<p>L0 (KVM) と L1 (Hyper-V) ハイパーバイザーとの間で準仮想化プロトコルを実装し、L2 を終了してハイパーバイザーに移動する時間を短縮できます。</p> <div data-bbox="598 443 707 607" style="display: inline-block; vertical-align: middle;">  </div> <p style="margin-left: 20px;">注記</p> <p style="margin-left: 20px;">この機能は Intel プロセッサのみを対象とします。</p>	on、off
frequencies	Hyper-V 周波数 MSR (Machine Specific Registeres) を有効にします。	on、off
ipi	IPI (準仮想化された相互プロセッサ割り込み) サポートを有効にします。	on、off
no-nonarch-coresharing	仮想プロセッサが SMT シブリングスレッドとして報告されない限り、物理コアを共有しないように、ゲスト OS に指示します。この情報は、SMT (同時マルチスレッド) に関連する CPU の脆弱性を適切に軽減するために、Windows および Hyper-V ゲストで必要です。	on、off、auto
reenlightenment	タイムスタンプカウンター (TSC) 周波数の変更がある場合に (移行時のみ) 通知します。新しい周波数に切替える準備ができるまで、ゲストでそのまま以前の周波数を使用することも可能です。	on、off
relaxed	仮想マシンを高負荷のホストで実行すると、一般的に BSOD に陥る Windows のサニティーチェックを無効化します。これは、Linux カーネルオプション <code>no_timer_check</code> と似ています。これは、Linux が KVM で実行している場合に自動的に有効になります。	on、off

Enlightenment	説明	値
runtime	ゲストコードの実行に費やすプロセッサ時間および、ゲストコードの代わりに費やすプロセッサ時間を設定します。	on、off
spinlocks	<ul style="list-style-type: none"> ● 仮想マシンのオペレーティングシステムによって使用され、呼び出し仮想プロセッサが同じパーティション内の別の仮想プロセッサで保持する可能性があるリソースを取得することを Hyper-V に通知します。 ● Hyper-V に過度のスピン状況を示す前に、スピンのロックの取得が施行されるべき回数についてを仮想マシンのオペレーティングシステムに示すために Hyper-V によって使用されます。 	on、off
stimer	仮想プロセッサの合成タイマーを有効にします。この Enlightenment が指定されない場合には、特定の Windows バージョンが、HPET (HPET が利用できない場合には RTC も使用) を使用するように戻すため、仮想 CPU がアイドル状態であっても、CPU の消費量が大幅に消費される可能性があることに注意してください。	on、off
stimer-direct	有効期限イベントが通常の割り込みで配信されると合成タイマーを有効にします。	on、off
syncic	stimer とともに、syncic タイマーをアクティブにします。Windows 8 では、この機能は定期的なモードで使用します。	on、off

Enlightenment	説明	値
時間	仮想マシンでできるように、以下の Hyper-V 固有のクロックソースを有効にします。 <ul style="list-style-type: none"> MSR ベースの 82 Hyper-V クロックソース (HV_X64_MSR_TIME_REF_COUNT, 0x40000020) MSR で有効にされる Reference TSC 83 ページ (HV_X64_MSR_REFERENCE_TSC, 0x40000021) 	on、off
tlbflush	仮想プロセッサの TLB をフラッシュします。	on、off
vapic	仮想 APIC を有効にして、高負荷のメモリーマッピングされた APIC (Advanced Programmable Interrupt Controller) レジスターへのアクセラレート MSR アクセスを提供します。	on、off
vendor_id	Hyper-V ベンダー ID を設定します。	<ul style="list-style-type: none"> on、off ID 値: 文字列 (最大 12 文字)
vpindex	仮想プロセッサのインデックスを有効にします。	on、off

21.2.3. NetKVM ドライバーパラメーターの設定

NetKVM ドライバーをインストールした後は、ご使用の環境に応じて設定を行うことができます。次の手順にリストされているパラメーターは、Windows デバイスマネージャー (**devmgmt.msc**) を使用して設定できます。



重要

ドライバーのパラメーターを変更すると、Windows はそのドライバーを再読み込みします。これにより、既存のネットワークアクティビティーが中断します。

前提条件

- NetKVM ドライバーが仮想マシンにインストールされている。

詳細は、[Installing KVM paravirtualized drivers for Windows virtual machines](#) を参照してください。

手順

1. Windows デバイスマネージャーを開きます。
Device Manager を開く方法は、Windows のドキュメントを参照してください。
2. **Red Hat VirtIO Ethernet Adapter**を見つけます。
 - a. Device Manager 画面で、Network アダプターの隣にある **+** をクリックします。
 - b. ネットワークアダプターのリストの下で、**Red Hat VirtIO Ethernet Adapter**をダブルクリックします。
デバイスの **プロパティ** ウィンドウが開きます。
3. デバイスパラメーターを表示します。
プロパティ ウィンドウで、**詳細設定** タブをクリックします。
4. デバイスパラメーターを変更します。
 - a. 変更するパラメーターをクリックします。
そのパラメーターのオプションが表示されます。
 - b. 必要に応じてオプションを変更します。
NetKVM パラメーターオプションの詳細は、[NetKVM ドライバーパラメーター](#) を参照してください。
 - c. **OK** をクリックして変更を保存します。

21.2.4. NetKVM ドライバーパラメーター

次の表に、設定可能な NetKVM ドライバーのロギングパラメーターに関する情報を示します。

表21.2 ロギングパラメーター

パラメーター	説明 2
logging.Enable	ロギングが有効であるかどうかを決定するブール値。デフォルト値は Enabled です。

パラメーター	説明 2
logging.Level	<p>ロギングレベルを定義する整数。この整数を高くすると、ログの詳細度が上がります。</p> <ul style="list-style-type: none"> ● デフォルト値は 0 (エラーのみ) です。 ● 1-2 は設定メッセージを追加します。 ● 3-4 は、パケットフロー情報を追加します。 ● 5-6 は割り込みおよび DPC レベルのトレース情報を追加します。 <p> 注記</p> <p>ロギングレベルが高くなると、仮想マシンの速度が低下します。</p>

次の表に、設定可能な NetKVM ドライバーの初期パラメーターに関する情報を示します。

表21.3 初期パラメーター

パラメーター	説明
Assign MAC	<p>準仮想化 NIC のローカル管理 MAC アドレスを定義する文字列。これはデフォルトでは設定されません。</p>
Init.Do802.1PQ	<p>Priority/VLAN タグポピュレーションと削除サポートを有効にするブール値。デフォルト値は Enabled です。</p>
Init.MaxTxBuffers	<p>割り当てられた TX リング記述子の数を表す整数。この値は、QEMU の Tx キューのサイズによって制限されます。</p> <p>デフォルト値は 1024 です。</p> <p>有効な値は、16、32、64、128、256、512、1024 です。</p>

パラメーター	説明
Init.MaxRxBuffers	<p>割り当てられた RX リング記述子の数を表す整数。この値は、QEMU の Tx キューのサイズによって制限されます。</p> <p>デフォルト値は 1024 です。</p> <p>有効な値は、16、32、64、128、256、512、1024、2048、および 4096 です。</p>
Offload.Tx.Checksum	<p>TX チェックサムオフロード機能を指定します。</p> <p>Red Hat Enterprise Linux 9 では、このパラメーターの有効な値は次のとおりです。</p> <ul style="list-style-type: none"> ● IPv4 と IPv6 の両方で IP、TCP、および UDP チェックサムオフロードを有効にする All (デフォルト) ● IPv4 と IPv6 の両方で TCP および UDP チェックサムオフロードを有効にする TCP/UDP (v4,v6) ● IPv4 でのみ TCP および UDP チェックサムオフロードを有効にする TCP/UDP (v4) ● IPv4 でのみ TCP (v4) チェックサムオフロードを有効にする TCP (v4)
Offload.Rx.Checksum	<p>RX チェックサムオフロード機能を指定します。</p> <p>Red Hat Enterprise Linux 9 では、このパラメーターの有効な値は次のとおりです。</p> <ul style="list-style-type: none"> ● IPv4 と IPv6 の両方で IP、TCP、および UDP チェックサムオフロードを有効にする All (デフォルト) ● IPv4 と IPv6 の両方で TCP および UDP チェックサムオフロードを有効にする TCP/UDP (v4,v6) ● IPv4 でのみ TCP および UDP チェックサムオフロードを有効にする TCP/UDP (v4) ● IPv4 でのみ TCP (v4) チェックサムオフロードを有効にする TCP (v4)

パラメーター	説明
Offload.Tx.LSO	<p>TX ラージセグメントオフロード (LSO) 機能を指定します。</p> <p>Red Hat Enterprise Linux 9 では、このパラメーターの有効な値は次のとおりです。</p> <ul style="list-style-type: none"> ● TCPv4 と TCPv6 の両方で LSO オフロードを有効にする Maximal (デフォルト) ● TCPv4 でのみ LSO オフロードを有効にする IPv4 ● LSO オフロードを無効にする Disable
MinRxBufferPercent	<p>RX キュー内の使用可能なバッファの最小量を、RX バッファの合計量に対するパーセントで指定します。使用可能なバッファの実際の数がこの値よりも少ない場合、NetKVM ドライバーは、オペレーティングシステムにリソース不足状態を通知します (できるだけ早く RX バッファを返すように要求します)。</p> <p>最小値 (デフォルト) - 0。ドライバーがリソース不足状態を示さないことを意味します。</p> <p>最大値 - 100。ドライバーが常にリソース不足状態を示すことを意味します。</p>

関連情報

- [INF enumeration keywords](#)
- [INF keywords that can be edited](#)

21.2.5. Windows 仮想マシンでのバックグラウンドプロセスの最適化

Windows OS を実行している仮想マシンのパフォーマンスを最適化するには、さまざまな Windows プロセスを設定するか、無効化してください。



警告

設定を変更すると、プロセスによっては、予想通り機能しない場合があります。

手順

次の組み合わせを実行すると、Windows 仮想マシンを最適化できます。

- USB や CD-ROM などの未使用のデバイスを削除して、ポートを無効にします。
- SuperFetch や Windows Search などのバックグラウンドサービスを無効にします。サービスの停止に関する詳細は、[システムサービスの無効化](#) または [サービス停止](#) を参照してください。
- **useplatformclock** を無効にします。これには以下のコマンドを実行します。

```
# bcdedit /set useplatformclock No
```

- スケジュール済みのディスクのデフラグなど、不要なスケジュールタスクを確認して無効にします。方法は、[スケジュール済みタスクの無効化](#) を参照してください。
- ディスクが暗号化されていないことを確認します。
- 定期的なサーバーアプリケーションのアクティビティーを減らします。これには、各タイマーを編集します。詳細は [Multimedia Timers](#) を参照してください。
- 仮想マシンで Server Manager アプリケーションを閉じます。
- ウイルス対策ソフトウェアを無効にします。ウイルス対策ソフトウェアを無効にすると、仮想マシンのセキュリティーが侵害される可能性があることに注意してください。
- スクリーンセーバーを無効にします。
- 使用時以外は、Windows OS のサインイン画面のままにします。

21.3. WINDOWS 仮想マシンでの標準ハードウェアセキュリティーの有効化

Windows 仮想マシンを保護するには、Windows デバイスの標準ハードウェア機能を使用して基本的なレベルのセキュリティーを有効にします。

前提条件

- 最新の WHQL 認定 VirtIO ドライバーがインストールされている。
- 仮想マシンのファームウェアが UEFI ブートに対応している。
- **edk2-OVMF** パッケージをホストマシンにインストールしている。

```
# {PackageManagerCommand} install edk2-ovmf
```

- ホストマシンに **vTPM** パッケージをインストールしている。

```
# {PackageManagerCommand} install swtpm libtpms
```

- 仮想マシンが Q35 マシンアーキテクチャーを使用している。
- Windows インストールメディアを使用している。

手順

1. TPM 2.0 を有効にするには、仮想マシンの XML 設定の **<devices>** セクションに以下のパラメーターを追加します。

```
<devices>
[...]
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'>
  </tpm>
[...]
</devices>
```

2. UEFI モードで Windows をインストールします。詳細は、[SecureBoot の仮想マシンの作成](#) を参照してください。
3. Windows 仮想マシンに virtio ドライバーをインストールします。詳細は、[Windows ゲストへの virtio ドライバーのインストール](#) を参照してください。
4. UEFI でセキュアブートを有効にします。詳細は、[セキュアブート](#) を参照してください。

検証

- Windows マシンの [デバイスのセキュリティー](#) ページに、以下のメッセージが表示されていることを確認します。
Settings > Update & Security > Windows Security > Device Security

Your device meets the requirements for standard hardware security.

21.4. WINDOWS 仮想マシンでの拡張ハードウェアセキュリティーの有効化

Windows 仮想マシンをさらにセキュアにするために、コード整合性の仮想化ベースの保護 (HVCI (Hypervisor-Protected Code Integrity) と呼ばれます) を有効にできます。

前提条件

- 標準のハードウェアセキュリティーが有効になっていることを確認します。詳細は、[Windows 仮想マシンでの標準ハードウェアセキュリティーの有効化](#) を参照してください。
- Hyper-V enlightenments が有効になっていることを確認します。詳細は、[Hyper-V enlightenment の有効化](#) を参照してください。

手順

1. Windows VM の XML 設定を開きます。次の例では、**Example-L1 VM** の設定を開きます。

```
# virsh edit Example-L1
```

2. **<cpu>** セクションで、CPU モードを指定し、ポリシーフラグを追加します。



重要

- Intel CPU の場合は、**vmx** ポリシーフラグを有効にします。
- AMD CPU の場合は、**svm** ポリシーフラグを有効にします。
- カスタム CPU を指定したくない場合は、**<cpu mode>** を **host-passthrough** として設定できます。

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'>
  <feature policy='require' name='vmx'!>
</cpu>
```

- XML 設定を保存し、仮想マシンを再起動します。
- 仮想マシンオペレーティングシステムで、**Core isolation details** ページに移動します。
Settings > Update & Security > Windows Security > Device Security > Core isolation details
- スイッチを切り替えて、**メモリーの整合性** を有効にします。
- 仮想マシンを再起動します。



注記

HVCI を有効にするその他の方法は、関連する Microsoft ドキュメントを参照してください。

検証

- Windows 仮想マシンの **デバイスのセキュリティー** ページに、以下のメッセージが表示されていることを確認します。

Settings > Update & Security > Windows Security > Device Security

Your device meets the requirements for enhanced hardware security.

- または、Windows 仮想マシンのシステム情報を確認します。
 - コマンドプロンプトで **msinfo32.exe** を実行します。
 - Virtualization-based security Services Running** の下に **Credential Guard, Hypervisor enforced Code Integrity** がリスト表示されているかどうかを確認します。

21.5. 次のステップ

- Windows 仮想マシンの仮想マシンディスクまたは他のディスクイメージへのアクセス、編集、および作成にユーティリティーを使用するには、ホストマシンに **libguestfs-tools** パッケージおよび **libguestfs-winsupport** パッケージをインストールします。

```
$ sudo dnf install libguestfs-tools libguestfs-winsupport
```

- Windows VM の仮想マシンディスクまたは他のディスクイメージへのアクセス、編集、および作成にユーティリティーを使用するには、ホストマシンに **guestfs-tools** パッケージおよび **guestfs-winsupport** パッケージをインストールします。

```
$ sudo dnf install guestfs-tools guestfs-winsupport
```

- RHEL 9 ホストとその Windows VM の間でファイルを共有するには、**virtiofs** または **NFS** を使用できます。

第22章 入れ子仮想マシンの作成

ローカルホストが実行しているものとは異なるホストオペレーティングシステムが必要な場合は、ネストされた仮想マシンを使用できます。これにより、追加の物理ハードウェアが不要になります。



警告

ほとんどの環境では、ネストされた仮想化は RHEL 9 の [テクノロジープレビュー](#) としてのみ利用できます。

サポート対象の環境とサポート対象外の環境の詳細は、[ネストされた仮想化に対するサポート制限](#) を参照してください。

22.1. ネストされた仮想化とは

ネストされた仮想化を使用すると、仮想マシンを他の仮想マシン内で実行できます。物理ホストで実行される標準の仮想マシンは、2 番目のハイパーバイザーとして機能し、独自の仮想マシンを作成することもできます。

ネストされた仮想化に関する用語

レベル 0 (L0)

物理ホスト、ベアメタルマシン。

レベル 1 (L1)

L0 物理ホスト上で実行され、追加の仮想ホストとして機能できる標準の仮想マシン。

レベル 2 (L2)

L1 仮想ホスト上で実行されるネストされた仮想マシン。

重要: 第 2 レベルの仮想化では、**L2** 仮想マシンのパフォーマンスが大幅に制限されます。したがって、ネストされた仮想化は、主に以下のような開発およびテストシナリオを対象としています。

- 制限された環境でのハイパーバイザーのデバッグ
- 限られた物理リソースでの大規模な仮想デプロイメントのテスト



警告

ほとんどの環境では、ネストされた仮想化は RHEL 9 の [テクノロジープレビュー](#) としてのみ利用できます。

サポート対象の環境とサポート対象外の環境の詳細は、[ネストされた仮想化に対するサポート制限](#) を参照してください。

関連情報

- [ネストされた仮想化に対するサポート制限](#)

22.2. ネストされた仮想化に対するサポート制限

ほとんどの環境では、ネストされた仮想化は RHEL 9 のテクノロジープレビュー としてのみ利用できます。

ただし、Windows Subsystem for Linux (WSL2) を備えた Windows 仮想マシンを使用して、Windows 仮想マシン内に仮想 Linux 環境を作成できます。このユースケースは、特定の条件下では RHEL 9 で完全にサポートされます。

ネストされた仮想化に関連する用語の詳細は、[ネストされた仮想化とは](#) を参照してください。

サポート対象の環境

ネストされた仮想化のサポート対象のデプロイメントを作成するには、RHEL 9 **L0** ホスト上に **L1** Windows 仮想マシンを作成し、WSL2 を使用して **L1** Windows 仮想マシン内に仮想 Linux 環境を作成します。現在、この環境は、唯一サポート対象となっているネストされた環境です。



重要

L0 ホストは、Intel または AMD システムである必要があります。現在、ARM や IBM Z などの他のアーキテクチャーはサポート対象外です。

次のオペレーティングシステムのバージョンのみを使用する必要があります。

L0 ホストの場合:	L1 仮想マシンの場合:
RHEL 9.2 以降	WSL2 搭載 Windows Server 2019
	WSL2 搭載 Windows Server 2022
	WSL2 搭載 Windows 10
	WSL2 搭載 Windows 11

WSL2 のインストール手順とサポート対象の Linux ディストリビューションの選択手順については、[Microsoft のドキュメント](#) を参照してください。

サポート対象のネストされた環境を作成するには、次のいずれかの手順を使用します。

- [Intel でのネスト化された仮想マシンの作成](#)
- [AMD でのネスト化された仮想マシンの作成](#)

テクノロジープレビュー環境

これらのネストされた環境はテクノロジープレビューとしてのみ利用可能であり、サポート対象外です。



重要

L0 ホストは、Intel、AMD、または IBM Z システムである必要があります。ネストされた仮想化は現在、ARM などの他のアーキテクチャーでは動作しません。

次のオペレーティングシステムのバージョンのみを使用する必要があります。

L0 ホストの場合:	L1 仮想マシンの場合:	L2 仮想マシンの場合:
RHEL 9.2 以降	RHEL 8.8 以降	RHEL 8.8 以降
	RHEL 9.2 以降	RHEL 9.2 以降
	Hyper-V 搭載 Windows Server 2016	Windows Server 2019
	Hyper-V 搭載 Windows Server 2019	Windows Server 2022
	Hyper-V 搭載 Windows Server 2022	
	Hyper-V 搭載 Windows 10	
	Hyper-V 搭載 Windows 11	



注記

他の Red Hat 仮想化オフリングで使用する場合、RHEL **L1** 仮想マシンの作成はテストされません。これには以下が含まれます。

- Red Hat Virtualization
- Red Hat OpenStack Platform
- OpenShift Virtualization

テクノロジープレビューのネストされた環境を作成するには、次のいずれかの手順を使用します。

- [Intel でのネスト化された仮想マシンの作成](#)
- [AMD でのネスト化された仮想マシンの作成](#)
- [IBM Z でのネストされた仮想マシンの作成](#)

Hypervisor の制限

- 現在、Red Hat は RHEL-KVM でのみネスト化のテストを行っています。RHEL を **L0** ハイパーバイザーとして使用する場合は、RHEL または Windows を **L1** ハイパーバイザーとして使用できます。

- KVM 以外の **L0** ハイパーバイザー (VMware ESXi や Amazon Web Services (AWS) など) で **L1** RHEL 仮想マシンを使用する場合、RHEL ゲストオペレーティングシステムでの **L2** 仮想マシンの作成はテストされておらず、機能しない可能性があります。

機能の制限

- **L2** 仮想マシンをハイパーバイザーとして使用し、**L3** ゲストを作成することは適切にテストされていないため、機能することは想定されていません。
- 現在、AMD システムでの仮想マシンの移行は、**L0** ホストでネストされた仮想化が有効になっている場合には機能しません。
- IBM Z システムでは、huge-page バッキングストレージとネストされた仮想化を同時に使用することはできません。

```
# modprobe kvm hpage=1 nested=1
modprobe: ERROR: could not insert 'kvm': Invalid argument
# dmesg |tail -1
[90226.508366] kvm-s390: A KVM host that supports nesting cannot back its KVM guests
with huge pages
```

- **L0** ホストで利用可能な機能は、**L1** ハイパーバイザーでは利用できない場合があります。

関連情報

- [What is Windows Subsystem for Linux?](#)
- [Intel でのネスト化された仮想マシンの作成](#)
- [AMD でのネスト化された仮想マシンの作成](#)
- [IBM Z でのネストされた仮想マシンの作成](#)

22.3. INTEL でのネスト化された仮想マシンの作成

以下の手順に従って、Intel ホストでネストされた仮想化を有効にし、設定します。



警告

ほとんどの環境では、ネストされた仮想化は RHEL 9 の [テクノロジープレビュー](#) としてのみ利用できます。

サポート対象の環境とサポート対象外の環境の詳細は、[ネストされた仮想化に対するサポート制限](#) を参照してください。

前提条件

- L1 仮想マシンを実行している L0 RHEL 9 ホスト。
- ハイパーバイザー CPU でネストされた仮想化をサポートしている。L0 ハイパーバイザーで `cat /proc/cpuinfo` コマンドを使用して、サポートがあるか確認します。コマンドの出力に **vmx**

および **ept** フラグが含まれる場合は、L2 仮想マシンを作成できます。通常、Intel Xeon v3 コア以降が対象です。

- L0 ホストでネストされた仮想化が有効になっていることを確認します。

```
# cat /sys/module/kvm_intel/parameters/nested
```

- コマンドが **1** または **Y** を返すと、この機能は有効になっています。残りの前提条件の手順を省略し、手順セクションに進みます。
- コマンドで **0** または **N** が返されたにも拘らず、システムがネストされた仮想化に対応している場合は、以下の手順に従って機能を有効にします。

- i. **kvm_intel** モジュールをアンロードします。

```
# modprobe -r kvm_intel
```

- ii. ネスト機能をアクティブにします。

```
# modprobe kvm_intel nested=1
```

- iii. ネスト機能は有効になりましたが、L0 ホストの次回起動後は無効になります。永続的に有効にするには、以下の行を **/etc/modprobe.d/kvm.conf** ファイルに追加します。

```
options kvm_intel nested=1
```

手順

1. ネストされた仮想化用に L1 仮想マシンを設定します。

- a. 仮想マシンの XML 設定を開きます。以下の例では、**Intel-L1** 仮想マシンの設定が開きます。

```
# virsh edit Intel-L1
```

- b. **<cpu>** 要素を編集して、**host-passthrough** CPU モードを使用するように仮想マシンを設定します。

```
<cpu mode='host-passthrough'/>
```

仮想マシンで特定の CPU モデルを使用する必要がある場合は、**custom** CPU モードを使用するように仮想マシンを設定します。**<cpu>** 要素内に、**<feature policy='require' name='vmx'/>** 要素と **<model>** 要素を追加し、内部で CPU モデルを指定します。以下に例を示します。

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Haswell-noTSX</model>
  <feature policy='require' name='vmx'/>
  ...
</cpu>
```

2. L1 仮想マシン内に L2 仮想マシンを作成します。作成するには、[L1 仮想マシンの作成](#) と同じ手順に従います。

22.4. AMD でのネスト化された仮想マシンの作成

以下の手順に従って、AMD ホストでネストされた仮想化を有効にして設定します。



警告

ほとんどの環境では、ネストされた仮想化は RHEL 9 の [テクノロジープレビュー](#) としてのみ利用できます。

サポート対象の環境とサポート対象外の環境の詳細は、[ネストされた仮想化に対するサポート制限](#) を参照してください。

前提条件

- L1 仮想マシンを実行している L0 RHEL 9 ホスト。
- ハイパーバイザー CPU でネストされた仮想化をサポートしている。L0 ハイパーバイザーで **cat /proc/cpuinfo** コマンドを使用して、サポートがあるか確認します。コマンドの出力に **svm** および **npt** フラグが含まれる場合は、L2 仮想マシンを作成できます。通常、これは AMD EPYC コア以降が対象です。
- L0 ホストでネストされた仮想化が有効になっていることを確認します。

```
# cat /sys/module/kvm_amd/parameters/nested
```

- コマンドが **1** または **Y** を返すと、この機能は有効になっています。残りの前提条件の手順を省略し、手順セクションに進みます。
- コマンドで **0** または **N** が返された場合は、以下の手順に従ってこの機能を有効にします。
 - i. L0 ホストで実行中の仮想マシンをすべて停止します。
 - ii. **kvm_amd** モジュールをアンロードします。

```
# modprobe -r kvm_amd
```

- iii. ネスト機能をアクティブにします。

```
# modprobe kvm_amd nested=1
```

- iv. ネスト機能は有効になりましたが、L0 ホストの次回起動後は無効になります。永続的に有効にするには、以下を **/etc/modprobe.d/kvm.conf** ファイルに追加します。

```
options kvm_amd nested=1
```

手順

1. ネストされた仮想化用に L1 仮想マシンを設定します。
 - a. 仮想マシンの XML 設定を開きます。以下の例では、**AMD-L1** 仮想マシンの設定が開きま

す。

```
# virsh edit AMD-L1
```

- b. **<cpu>** 要素を編集して、**host-passthrough** CPU モードを使用するように仮想マシンを設定します。

```
<cpu mode='host-passthrough'/>
```

仮想マシンで特定の CPU モデルを使用する必要がある場合は、**custom** CPU モードを使用するように仮想マシンを設定します。**<cpu>** 要素内に、**<feature policy='require' name='svm'/>** 要素と **<model>** 要素を追加し、内部で CPU モデルを指定します。以下に例を示します。

```
<cpu mode="custom" match="exact" check="none">
  <model fallback="allow">EPYC-IBPB</model>
  <feature policy="require" name="svm"/>
  ...
</cpu>
```

2. L1 仮想マシン内に L2 仮想マシンを作成します。作成するには、[L1 仮想マシンの作成](#) と同じ手順に従います。

22.5. IBM Z でのネストされた仮想マシンの作成

以下の手順に従って、IBM Z ホストでネストされた仮想化を有効にして設定します。



注記

実際には、IBM Z はベアメタル **L0** ホストを提供しません。代わりに、ユーザーシステムは、論理パーティション (LPAR) にセットアップされます。論理パーティションはすでに仮想化されたシステムであるため、しばしば **L1** と呼ばれます。ただし、このガイドの他のアーキテクチャーとの整合性を高めるために、次の手順では IBM Z が **L0** ホストを提供するものとします。

ネストされた仮想化の詳細は、[ネストされた仮想化とは](#) を参照してください。



警告

ほとんどの環境では、ネストされた仮想化は RHEL 9 の [テクノロジープレビュー](#) としてのみ利用できます。

サポート対象の環境とサポート対象外の環境の詳細は、[ネストされた仮想化に対するサポート制限](#) を参照してください。

前提条件

- L1 仮想マシンを実行している L0 RHEL 9 ホスト。

- ハイパーバイザー CPU でネストされた仮想化をサポートしている。これを確認するには、LO ハイパーバイザーで `cat /proc/cpuinfo` コマンドを使用します。コマンドの出力に `smi` フラグが含まれる場合は、L2 仮想マシンを作成できます。
- LO ホストでネストされた仮想化が有効になっていることを確認します。

```
# cat /sys/module/kvm/parameters/nested
```

- コマンドが `1` または `Y` を返すと、この機能は有効になっています。残りの前提条件の手順を省略し、手順セクションに進みます。
- コマンドで `0` または `N` が返された場合は、以下の手順に従ってこの機能を有効にします。
 - i. LO ホストで実行中の仮想マシンをすべて停止します。
 - ii. **kvm** モジュールをアンロードします。

```
# modprobe -r kvm
```

- iii. ネスト機能をアクティブにします。

```
# modprobe kvm nested=1
```

- iv. ネスト機能は有効になりましたが、LO ホストの次回起動後は無効になります。永続的に有効にするには、以下の行を `/etc/modprobe.d/kvm.conf` ファイルに追加します。

```
options kvm nested=1
```

手順

- L1 仮想マシン内に L2 仮想マシンを作成します。作成するには、[L1 仮想マシンの作成](#) と同じ手順に従います。

第23章 仮想マシンの問題診断

仮想マシンを使用していると、さまざまな重大度の問題が発生する可能性があります。問題によってはすぐ簡単に修正できる可能性があります。仮想マシン関連のデータおよびログを取得して、問題を報告または診断しなければならない場合もあります。

以下のセクションでは、ログの生成および一般的な仮想マシンの問題の診断方法と、このような問題の報告方法を説明します。

23.1. LIBVIRT デバッグログの生成

仮想マシンの問題を診断するには、libvirt デバッグログを生成し、確認すると便利です。また仮想マシン関連の問題解決のサポートを受ける場合には、デバッグログを添付すると役立ちます。

次のセクションでは、[デバッグログとは何か](#)、[デバッグログの永続設定](#)、[実行時の有効化](#)、[問題報告時の添付方法](#)について説明します。

23.1.1. libvirt デバッグログについて

デバッグログは、仮想マシンランタイム時に発生するイベント関連のデータが含まれるテキストファイルです。ログには、ホストライブラリーや libvirt デーモンなどの、サーバー側の基本機能に関する情報が含まれます。ログファイルには、実行中の全仮想マシンの標準エラー出力 (**stderr**) も含まれます。

デバッグロギングはデフォルトでは有効ではないので、libvirt の起動時に有効にする必要があります。セッション1回分のロギングを有効にしたり、[永続的](#)にロギングを有効にすることができます。また、[デーモンのランタイム設定を変更](#)することにより、libvirt デーモンセッションがすでに実行中の場合にロギングを有効にすることもできます。

仮想マシンの問題のサポートを受ける場合に、[libvirt デバッグログ](#)を添付すると役立ちます。

23.1.2. libvirt デバッグログの永続的な設定の有効化

libvirt の起動時に毎回、libvirt デバッグロギングを自動的に有効にするように設定できます。デフォルトでは、**virtqemud** は RHEL 9 の主な libvirt デーモンになります。libvirt 設定で永続的な変更を行うには、**/etc/libvirt** ディレクトリーにある **virtqemud.conf** ファイルを編集する必要があります。



注記

場合によっては、たとえば、RHEL 8 からアップグレードするときに、**libvirtd** は依然として有効な libvirt デーモンである可能性があります。この場合は、代わりに **libvirtd.conf** ファイルを編集する必要があります。

手順

1. エディターで **virtqemud.conf** ファイルを開きます。
2. 要件に応じてフィルターを置き換えるか、設定します。

表23.1 フィルター値のデバッグ

1	libvirt が生成したすべてのメッセージをログに記録します。
2	すべての非デバッグ情報をログに記録します。

3	すべての警告およびエラーメッセージをログに記録します。これはデフォルト値です。
4	エラーメッセージのみをログに記録します。

例23.1 ログフィルターのデーモン設定例

以下の設定を行います。

- **remote**、**util.json**、および **rpc** 層からのすべてのエラーメッセージおよび警告メッセージをログに記録します。
- **event** レイヤーからのエラーメッセージのみを記録します。
- フィルターされたログを **/var/log/libvirt/libvirt.log** に保存します。

```
log_filters="3:remote 4:event 3:util.json 3:rpc"
log_outputs="1:file:/var/log/libvirt/libvirt.log"
```

3. 保存して終了します。
4. libvirt デーモンを再起動します。

```
$ systemctl restart virtqemud.service
```

23.1.3. ランタイム時の libvirt デバッグログの有効化

libvirt デーモンのランタイム設定を変更し、デバッグログを有効にして出力ファイルに保存できます。

これは、再起動すると問題が解決するため、libvirt デーモンを再起動できない場合や、マイグレーションやバックアップなどの別のプロセスが同時に実行されている場合などに便利です。設定ファイルを編集したり、デーモンを再起動せずにコマンドを試行したりする場合にも、ランタイム設定を変更すると便利です。

前提条件

- **libvirt-admin** パッケージがインストールされている。

手順

1. **任意:** アクティブなログフィルターのバックアップを作成します。

```
# virt-admin -c virtqemud:///system daemon-log-filters >> virt-filters-backup
```



注記

ログの生成後に復元できるように、アクティブなフィルターセットをバックアップすることが推奨されます。フィルターを復元しないと、メッセージがログに記録され、システムパフォーマンスに影響する可能性があります。

2. **virt-admin** ユーティリティーを使用してデバッグを有効にし、要件に応じてフィルターを設定します。

表23.2 フィルター値のデバッグ

1	libvirt が生成したすべてのメッセージをログに記録します。
2	すべての非デバッグ情報をログに記録します。
3	すべての警告およびエラーメッセージをログに記録します。これはデフォルト値です。
4	エラーメッセージのみをログに記録します。

例23.2 ロギングフィルターの virt-admin 設定の例

以下のコマンドを実行します。

- **remote**、**util.json**、および **rpc** レイヤーからのエラーメッセージおよび警告メッセージをすべてログに記録します。
- イベント レイヤーからのエラーメッセージのみを記録します。

```
# virt-admin -c virtqemud:///system daemon-log-filters "3:remote 4:event 3:util.json 3:rpc"
```

3. **virt-admin** ユーティリティーを使用して、ログを特定のファイルまたはディレクトリーに保存します。
たとえば、以下のコマンドはログ出力を `/var/log/libvirt/` ディレクトリーの **libvirt.log** ファイルに保存します。

```
# virt-admin -c virtqemud:///system daemon-log-outputs "1:file:/var/log/libvirt/libvirt.log"
```

4. **任意:** フィルターを削除して、仮想マシン関連の情報をすべて含むログファイルを生成することもできます。ただし、このファイルには libvirt のモジュールが生成した多くの冗長情報が含まれる可能性があるため、推奨されません。

- **virt-admin** ユーティリティーを使用して空のフィルターを指定します。

```
# virt-admin -c virtqemud:///system daemon-log-filters
Logging filters:
```

5. **任意:** バックアップファイルを使用して、フィルターを元の状態に復元します。
保存した値を使用して 2 番目の手順を実行し、フィルターを復元します。

23.1.4. サポートリクエストへの libvirt デバッグログの添付

仮想マシンの問題の診断および解決に追加のサポートを依頼する必要がある場合があります。仮想マシン関連の問題を迅速に解決するために、サポートチームが必要な情報すべてにアクセスできるように、サポートリクエストにデバッグログを添付することを強く推奨します。

手順

- 問題およびサポートを報告するには、[サポートケースを作成](#) してください。
- 発生した問題に応じて、レポートに以下のログを添付します。
 - libvirt サービスに問題がある場合は、ホストから `/var/log/libvirt/libvirt.log` ファイルを添付します。
 - 特定の仮想マシンに関する問題は、該当するログファイルを添付します。たとえば、仮想マシン `testguest1` の場合は、`/var/log/libvirt/qemu/testguest1.log` にある `testguest1.log` ファイルを添付します。

関連情報

- [How to provide log files to Red Hat Support?](#)

23.2. 仮想マシンのコアのダンプ

仮想マシンがクラッシュしたり、誤作動した理由を分析する場合は、後で分析と診断を行えるように仮想マシンのコアをディスクのファイルにダンプします。

本セクションでは、[コアダンプ概要](#) と、[仮想マシンのコア](#) を特定のファイルにダンプする方法を説明します。

23.2.1. 仮想マシンのコアダンプの仕組み

仮想マシンでは、数多くの実行中のプロセスを正確かつ効率的に機能させる必要があります。場合によっては、実行中の仮想マシンが、使用中に予期せず終了したり、誤作動したりすることがあります。仮想マシンを再起動すると、データがリセットされたり失われてしまう可能性があり、仮想マシンがクラッシュした問題の正確な診断が困難になります。

このような場合は、`virsh dump` ユーティリティーを使用して、仮想マシンを再起動する前に仮想マシンのコアをファイルに保存 (または **ダンプ**) できます。コアダンプファイルには、仮想マシンの生の物理メモリーイメージが含まれ、その中に仮想マシンの詳細情報が含まれます。この情報は、手動、または `crash` ユーティリティーなどのツールで、仮想マシンの問題診断に使用できます。

関連情報

- `crash` の man ページ
- `crash` の Github リポジトリ

23.2.2. 仮想マシンのコアダンプファイルの作成

仮想マシンのコアダンプには、任意の時点における仮想マシンの状態に関する詳細情報が含まれます。この情報は、VM のスナップショットに似ており、VM が誤動作したり、突然シャットダウンしたりした場合に問題を検出するのに役立ちます。

前提条件

- ファイルを保存するのに十分なディスク領域があることを確認してください。仮想マシンが占有する領域は、その仮想マシンに割り当てられている RAM のサイズによって異なることに注意してください。

手順

- **virsh dump** ユーティリティーを使用します。
たとえば、次のコマンドは、仮想マシンのコア **lander1**、メモリー、CPU 共通レジスターファイル、**/core/file** の **gargantua.file** にダンプします。

```
# virsh dump lander1 /core/file/gargantua.file --memory-only
Domain 'lander1' dumped to /core/file/gargantua.file
```

重要

crash ユーティリティーは、**virsh dump** コマンドのデフォルトファイル形式に対応しなくなりました。**crash** を使用してコアダンプファイルを分析するには、**--memory-only** オプションを使用してファイルを作成する必要があります。

また、コアダンプファイルを作成して Red Hat サポートケースに添付する場合は、**--memory-only** オプションを使用する必要があります。

トラブルシューティング

virsh dump コマンドが **System is deadlocked on memory** エラーで失敗する場合、コアダンプファイルに十分なメモリーが割り当てられていることを確認してください。これを行うには、以下の **crashkernel** オプション値を使用します。または、コアダンプメモリーを自動的に割り当てる **crashkernel** は一切使用しないでください。

```
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
```

関連情報

- **virsh dump --help** コマンド
- **virsh** の man ページ
- [サポートケースの作成](#)

23.3. 仮想マシンプロセスのバックトレース

仮想マシンの誤作動に関連するプロセスが機能する場合には、プロセス識別子 (PID) を指定して **gstack** コマンドを使用し、誤作動しているプロセスの実行スタックトレースを生成できます。プロセスがスレッドグループの一部である場合は、スレッドもすべてトレースされます。

前提条件

- **GDB** パッケージがインストールされている。
GDB および利用可能なコンポーネントのインストール方法は、[Installing the GNU Debugger](#) を参照してください。
- バックトレースするプロセスの PID を把握している。
pgrep コマンドの後にプロセス名を使用すると、PID を検索できます。以下に例を示します。

```
# pgrep libvirt
22014
22025
```

手順

- **gstack** ユーティリティの後にバックトレースするプロセスの PID を指定して使用します。たとえば、以下のコマンドは、PID 22014 で libvirt プロセスをバックトレースします。

```
# gstack 22014
Thread 3 (Thread 0x7f33edaf7700 (LWP 22017)):
#0  0x00007f33f81aef21 in poll () from /lib64/libc.so.6
#1  0x00007f33f89059b6 in g_main_context_iterate.isra () from /lib64/libglib-2.0.so.0
#2  0x00007f33f8905d72 in g_main_loop_run () from /lib64/libglib-2.0.so.0
...
```

関連情報

- **gstack** の man ページ
- [GNU デバッガー \(GDB\)](#)

仮想マシンの問題報告およびログ提供に使用する追加のリソース

追加でヘルプおよびサポートを依頼するには、以下を行います。

- **redhat-support-tool** コマンドラインオプション、Red Hat Portal UI、またはいくつかの FTP の方法を使用して、サービスリクエストを提出してください。
 - 問題およびサポートを報告する方法は、[サポートケースの作成](#) を参照してください。
- サービス依頼の送信時に SOS Report およびログファイルをアップロードします。これにより、Red Hat サポートエンジニアが必要な診断情報をすべて参照できるようになります。
 - SOS レポートに関する詳細は、[What is an SOS Report and how to create one in Red Hat Enterprise Linux?](#) を参照してください。
 - ログファイルの添付方法に関する詳細は、[How to provide files to Red Hat Support?](#) を参照してください。

第24章 RHEL 9 仮想化における機能のサポートおよび制限

このドキュメントでは、Red Hat Enterprise Linux 9 (RHEL 9) 仮想化の機能のサポートと制限事項について説明します。

24.1. RHEL 9 仮想化サポートの動作

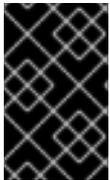
一連のサポート制限は、Red Hat Enterprise Linux 9 (RHEL 9) の仮想化に適用されます。つまり、RHEL 9 で仮想マシンを使用する際に、特定の機能を使用したり、割り当てられたリソースの量のある程度を超えたりすると、特別なサブスクリプションプランがない限り、Red Hat はこれらのゲストをサポートしません。

[Recommended features in RHEL 9 仮想化](#) に記載されている機能は、RHEL 9 システムの KVM ハイパーバイザーと連携するように、Red Hat によりテストおよび認定されています。したがって、RHEL 9 の仮想化の使用が完全にサポートされ、推奨されます。

[Unsupported features in RHEL 9 仮想化](#) に記載されている機能は動作する場合がありますが、サポート対象外であり、RHEL 9 での使用は意図されていません。したがって、Red Hat は、KVM を使用する RHEL 9 でこの機能を使用しないことを強く推奨します。

[RHEL 9 仮想化のリソースの割り当ての制限](#) は、RHEL 9 の KVM ゲストで対応する特定のリソースの最大数を一覧表示します。この制限を超えるゲストは、Red Hat ではサポートされていません。

さらに、特に記載がない限り、RHEL 9 の仮想化のドキュメントで使用されるすべての機能とソリューションがサポートされます。ただし、その一部は完全にテストされていないため、完全には最適化されない場合があります。



重要

この制限の多くは、Red Hat が提供するその他の仮想化ソリューション (OpenShift Virtualization や Red Hat OpenStack Platform (RHOSP) など) には適用されません。

24.2. RHEL 9 仮想化で推奨される機能

以下の機能は、Red Hat Enterprise Linux 9 (RHEL 9) に含まれる KVM ハイパーバイザーで使用することが推奨されます。

ホストシステムのアーキテクチャー

KVM を使用した RHEL 9 は、以下のホストアーキテクチャーでのみ対応します。

- AMD64 および Intel 64
- IBM Z - IBM z13 システムおよびそれ以降
- ARM 64

その他のハードウェアアーキテクチャーは、KVM 仮想化ホストとして RHEL 9 の使用に対応していないため、Red Hat では推奨していません。

ゲストのオペレーティングシステム

Red Hat は、特定のゲストオペレーティングシステム (OS) を使用する KVM 仮想マシンのサポートを提供します。サポートされているゲスト OS の詳細なリストについては [Red Hat ナレッジベース](#) の Certified Guest Operating Systems を参照してください。

ただし、デフォルトでは、ゲスト OS とホストとは、同じサブスクリプションを使用しない点に注意してください。したがって、ゲスト OS を適切に機能させるには、別のライセンスまたはサブスクリプションをアクティベートする必要があります。

さらに、仮想マシンにアタッチするパススルーデバイスは、ホスト OS とゲスト OS の両方でサポートされる必要があります。

同様に、デプロイメントの最適な機能を得るには、Red Hat では、仮想マシンの XML 設定で定義する CPU モデルおよび機能が、ホスト OS とゲスト OS の両方でサポートされることを推奨します。

さまざまなバージョンの RHEL で認定された CPU およびその他のハードウェアを表示するには、[Red Hat Ecosystem Catalog](#) を参照してください。

マシンタイプ

VM がホストアーキテクチャーと互換性があり、ゲスト OS が最適に実行されるようにするには、仮想マシンで適切なマシンタイプを使用する必要があります。



重要

RHEL 9 では、RHEL の以前のメジャーバージョンでデフォルトであった **pc-i440fx-rhel7.5.0** およびそれ以前のマシンタイプはサポートされなくなりました。その結果、RHEL 9 ホストでそのようなマシンタイプの VM を起動しようとする、**unsupported configuration** エラーで失敗します。ホストを RHEL 9 にアップグレードした後にこの問題が発生した場合は、[Red Hat KnowledgeBase](#) を参照してください。

[コマンドラインを使用して仮想マシンを作成](#) する場合、**virt-install** ユーティリティーはマシンタイプを設定する複数の方法を提供します。

- **--os-variant** オプションを使用すると、**virt-install** は、使用しているホスト CPU に対して推奨され、ゲスト OS でサポートされているマシンタイプを自動的に選択します。
- **--os-variant** を使用しない場合、または別のマシンタイプが必要な場合は、**-machine** オプションを使用してマシンタイプを明示的に指定します。
- サポートされていない、またはホストと互換性のない **--machine** 値を指定すると、**virt-install** が失敗し、エラーメッセージが表示されます。

サポートされているアーキテクチャー上の KVM 仮想マシンに推奨されるマシンタイプ、および **--machine** オプションに対応する値は次のとおりです。Y は、RHEL 9 の最新のマイナーバージョンを表します。

- Intel 64 および AMD64 (x86_64) の場合: **pc-q35-rhel9.Y.0** → **--machine=q35**
- IBM Z (s390x) の場合: **s390-ccw-virtio-rhel9.Y.0** → **--machine=s390-ccw-virtio**
- ARM 64 の場合: **virt-rhel9.Y.0** → **--machine=virt**

既存の仮想マシンを取得する場合:

```
# virsh dumpxml VM-name | grep machine=
```

ホストでサポートされているマシンタイプの完全なリストを表示する場合:

```
# /usr/libexec/qemu-kvm -M help
```

関連情報

- [RHEL 9 仮想化で対応していない機能](#)
- [RHEL 9 仮想化におけるリソース割り当ての制限](#)

24.3. RHEL 9 仮想化で対応していない機能

以下の機能は、Red Hat Enterprise Linux 9 (RHEL 9) に含まれる KVM ハイパーバイザーでは対応していません。



重要

この制限の多くは、Red Hat が提供するその他の仮想化ソリューション (Red Hat Virtualization (RHV) や Red Hat OpenStack Platform (RHOSP) など) には適用されない場合があります。

他の仮想化ソリューションでサポートされる機能については、次の段落で説明します。

ホストシステムのアーキテクチャー

KVM を使用した RHEL 9 は、[RHEL 9 仮想化で推奨される機能](#) に記載されていないホストアーキテクチャーではサポートされません。

ゲストのオペレーティングシステム

RHEL 9 ホストは、次のゲストオペレーティングシステム (OS) を使用する KVM 仮想マシンには対応していません。

- Windows 8.1 以前
- Windows Server 2012 R2 以前
- MacOS
- x86 システム用の Solaris
- 2009 年以前にリリースされたオペレーティングシステム

RHEL ホストおよびその他の仮想化ソリューションでサポートされるゲスト OS の一覧は、[Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization, OpenShift Virtualization and Red Hat Enterprise Linux with KVM](#) を参照してください。

コンテナでの仮想マシンの作成

Red Hat は、RHEL 9 ハイパーバイザーの要素を含むコンテナの種類 (**QEMU** エミュレーターや **libvirt** パッケージなど) に関係なく、KVM 仮想マシンの作成に対応していません。

コンテナに仮想マシンを作成する場合は、[OpenShift Virtualization](#) オファリングの使用を推奨します。

特定の virsh コマンドおよびオプション

virsh ユーティリティで利用できるすべてのパラメーターが Red Hat によってテストされ、本番環境で使用できると認定されているわけではありません。したがって、Red Hat のドキュメントで明示的に推奨されていない **virsh** コマンドおよびオプションは、正しく機能しない可能性があります。Red Hat では、実稼働環境でのこれらの使用を推奨しません。

特に、サポートされていない **virsh** コマンドには次のものがあります。

- **virsh iface-*** コマンド (**virsh iface-start** や **virsh iface-destroy** など)
- **virsh blkdeviotune**
- **virsh snapshot-*** コマンド (**virsh snapshot-create** や **virsh snapshot-revert** など)

QEMU コマンドライン

QEMU は、RHEL 9 における仮想化アーキテクチャーの必須コンポーネントですが、手動で管理することが難しく、QEMU 設定に誤りがあるとセキュリティーの脆弱性を引き起こす可能性があります。したがって、**qemu-kvm** などの **qemu-*** コマンドラインユーティリティーの使用は、Red Hat ではサポートされていません。代わりに、**virt-install**、**virt-xml**、およびサポートされている **virsh** コマンドなどの **libvirt** ユーティリティーを使用してください。これらはベストプラクティスに従って QEMU を調整します。ただし、[仮想ディスクイメージの管理](#) には **qemu-img** ユーティリティーがサポートされています。

vCPU のホットアンプラグ

実行中の仮想マシンから仮想 CPU (vCPU) を削除することは vCPU ホットアンプラグと呼ばれますが、RHEL 9 では対応していません。

メモリーのホットアンプラグ

実行中の仮想マシンに接続されているメモリーデバイスの削除 (メモリーのホットアンプラグとも呼ばれる) は、RHEL 9 では対応していません。

QEMU 側の I/O スロットリング

virsh blkdeviotune ユーティリティーを使用して、仮想ディスクでの操作の最大入出力レベルを設定することは、QEMU 側の I/O スロットリングとしても知られていますが、RHEL 9 では対応していません。

RHEL 9 で I/O スロットリングを設定するには、**virsh blkiotune** を使用します。これは、**libvirt** 側の I/O スロットリングとしても知られています。手順は、[Disk I/O throttling in virtual machines](#) を参照してください。

その他のソリューション:

- QEMU 側の I/O スロットリングは RHOSP でもサポートされています。詳細は、[RHOSP ストレージガイドの ディスクでのリソース制限の設定](#) および [サービス品質の仕様の使用セクション](#) を参照してください。
- さらに、OpenShift Virtualization は QEMU 側の I/O スロットリングもサポートします。

ストレージのライブマイグレーション

実行している仮想マシンのディスクイメージをホスト間で移行することは、RHEL 9 では対応していません。

その他のソリューション:

- ストレージライブマイグレーションは RHOSP でサポートされますが、制限がいくつかあります。詳細は [ボリュームの移行](#) を参照してください。

内部スナップショット

仮想マシンの内部スナップショットの作成と使用は、RHEL 9 では非推奨であり、実稼働環境では使用しないことを強く推奨します。代わりに、外部スナップショットを使用してください。詳細は、[仮想マシンのスナップショットのサポート制限](#) を参照してください。

その他のソリューション:

- RHOSP はライブスナップショットをサポートしています。詳細は、[Importing virtual machines into the overcloud](#) を参照してください。
- ライブスナップショットは OpenShift Virtualization でもサポートされています。

vHost Data Path Acceleration

RHEL 9 ホストでは、virtio デバイス用に vHost Data Path Acceleration (vDPA) を設定できますが、Red Hat は現在この機能をサポートしておらず、実稼働環境では使用しないことを強く推奨します。

vhost-user

RHEL 9 は、ユーザー空間の vHost インターフェイスの実装には対応していません。

その他のソリューション:

- **vhost-user** は RHOSP でサポートされていますが、対象は、**virtio-net** インターフェイスのみです。詳細は [virtio-net implementation](#) および [vhost user ports](#) を参照してください。
- OpenShift Virtualization は **vhost-user** もサポートします。

S3 および S4 のシステムの電力状態

仮想マシンを **Suspend to RAM (S3)** または **Suspend to disk (S4)** のシステム電源状態にサスペンドすることには対応していません。この機能はデフォルトでは無効になっており、有効にすると、仮想マシンが Red Hat のサポート対象外となります。

現在、S3 および S4 の状態は、Red Hat が提供する他の仮想化ソリューションでもサポートされていないことに注意してください。

マルチパス化された vDisk の S3-PR

マルチパス化された vDisk の SCSI3 の永続的な予約 (S3-PR) は、RHEL 9 では対応していません。これにより、RHEL 9 では、Windows Cluster に対応していません。

virtio-crypto

RHEL 9 での **virtio-crypto** デバイスの使用はサポートされておらず、RHEL では使用しないことを強く推奨します。

virtio-crypto デバイスは、Red Hat が提供する他の仮想化ソリューションでもサポートされていないことに注意してください。

virtio-multitouch-device、virtio-multitouch-pci

RHEL 9 での **virtio-multitouch-device** および **virtio-multitouch-pci** デバイスの使用はサポートされておらず、RHEL では使用しないことを強く推奨します。

インクリメンタルバックアップ

最後のバックアップ (増分ライブバックアップとも呼ばれる) 以降の VM の変更のみを保存する VM バックアップの設定は、RHEL 9 ではサポートされておらず、Red Hat はこれを使用しないことを強く推奨しています。

net_failover

RHEL 9 では、**net_failover** ドライバーを使用した自動ネットワークデバイスフェイルオーバーメカニズムの設定はサポートされていません。

現在、**net_failover** は、Red Hat が提供する他の仮想化ソリューションでもサポートされていないことに注意してください。

TCG

QEMU および libvirt には、QEMU Tiny Code Generator (TCG) を使用した動的な変換モードが含まれます。このモードでは、ハードウェアの仮想化のサポートは必要ありません。ただし、TCG は Red Hat ではサポートされていません。

TCG ベースのゲストは、**virsh dumpxml** コマンドを使用するなど、XML 設定を調べることで確認できます。

- TCG ゲストの設定ファイルでは、以下の行が含まれます。

```
<domain type='qemu'>
```

- KVM ゲストの設定ファイルでは、以下の行が含まれます。

```
<domain type='kvm'>
```

SR-IOV InfiniBand ネットワークデバイス

シングルルート I/O 仮想化 (SR-IOV) を使用した VM への InfiniBand ネットワークデバイスの接続はサポートされていません。

関連情報

- [RHEL 9 仮想化で推奨される機能](#)
- [RHEL 9 仮想化におけるリソース割り当ての制限](#)

24.4. RHEL 9 仮想化におけるリソース割り当ての制限

以下の制限は、Red Hat Enterprise Linux 9 (RHEL 9) ホストの 1 台の KVM 仮想マシンに割り当てることができる仮想化リソースに適用されます。



重要

この制限の多くは、Red Hat が提供するその他の仮想化ソリューション (OpenShift Virtualization や Red Hat OpenStack Platform (RHOSP) など) には適用されません。

仮想マシンごとの vCPU の最大数

RHEL 9 ホストで実行されている単一の VM でサポートされる vCPU とメモリーの最大量については、[KVM による Red Hat Enterprise Linux の仮想化の制限](#) を参照してください。

各仮想マシンの PCI デバイス

RHEL 9 は、各仮想マシンバスごとに 32 個の PCI デバイススロットをサポートし、デバイススロットごとに 8 個の PCI 機能をサポートします。これにより、仮想マシンで多機能の性能が有効になり、PCI

ブリッジが使用されていない場合に、理論上は1つのバスあたり最大 256 個の PCI 機能が提供されます。

各 PCI ブリッジは新しいバスを追加します。これにより、別の 256 個のデバイスアドレスが有効になる可能性があります。ただし、一部のバスでは、256 個のデバイスアドレスがすべて利用できるようなになっていません。たとえば、ルートバスには、スロットを占有する複数の組み込みデバイスがあります。

仮想化 IDE デバイス

KVM は、各仮想マシンで仮想化されている IDE デバイスの最大数を 4 に制限します。

24.5. IBM Z の仮想化と、AMD64 および INTEL 64 の仮想化の相違点

IBM Z システムの RHEL 9 の KVM 仮想化は、以下の点で AMD64 システムおよび Intel 64 システムの KVM とは異なります。

PCI デバイスおよび USB デバイス

IBM Z は、仮想 PCI デバイスおよび USB デバイスに対応していません。したがって、**virtio-*-pci** デバイスに対応していないため、代わりに **virtio-*-ccw** デバイスを使用してください。たとえば、**virtio-net-pci** の代わりに **virtio-net-ccw** を使用します。

PCI パススルーとも呼ばれる PCI デバイスの直接アタッチに対応しています。

サポートされているゲスト OS

Red Hat が IBM Z でホストする仮想マシンをサポートするのは、仮想マシンがゲストオペレーティングシステムとして RHEL 7、8、または 9 を使用している場合のみです。

デバイスの起動順序

IBM Z は、XML 設定要素 **<boot dev='device'>** に対応していません。デバイスの起動順序を定義するには、XML の **<devices>** セクションで、**<boot order='number'>** 要素を使用します。

さらに、**<boot>** 要素でアーキテクチャー固有の **loadparm** 属性を使用して、必要なブートエントリーを選択できます。たとえば、次の例では、ブートシーケンスでディスクを最初に使用する必要があります、そのディスクで Linux ディストリビューションが利用可能な場合は、2 番目のブートエントリーが選択されます。

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' />
  <source file='/path/to/qcow2' />
  <target dev='vda' bus='virtio' />
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000' />
  <boot order='1' loadparm='2' />
</disk>
```



注記

また、AMD64 および Intel 64 のホストでは、ブート手順管理に **<boot order='number'>** を使用することが推奨されます。

メモリーのホットプラグ

IBM Z では、実行中の仮想マシンにメモリーを追加することはできません。実行中の仮想マシン (メモリーの **ホットアンプラグ**) からメモリーを削除することは、IBM Z や、AMD64 および Intel 64 でもできないことに注意してください。

NUMA トポロジー

CPU の Non-Uniform Memory Access (NUMA) トポロジーは、IBM Z 上の **libvirt** では対応していません。したがって、このシステムでは、NUMA を使用して vCPU パフォーマンスをチューニングすることはできません。

GPU devices

IBM Z システムでは [GPU デバイスの割り当て](#) はサポートされていません。

vfio-ap

IBM Z ホストの仮想マシンは、**vfio-ap** 暗号デバイスパススルーを使用しますが、その他のアーキテクチャーでは対応していません。

vfio-ccw

IBM Z ホスト上の仮想マシンは、他のアーキテクチャーではサポートされていない **vfio-ccw** ディスクデバイスパススルーを使用できます。

SMBIOS

IBM Z では、SMBIOS 設定は利用できません。

Watchdog デバイス

IBM Z ホストの仮想マシンでウォッチドッグデバイスを使用する場合は、**diag288** モデルを使用します。以下に例を示します。

```
<devices>
  <watchdog model='diag288' action='poweroff'/>
</devices>
```

kvm-clock

kvm-clock サービスは、AMD64 システムおよび Intel 64 システムに固有のものであり、IBM Z の仮想マシンの時間管理用に設定する必要はありません。

v2v および p2v

virt-v2v ユーティリティおよび **virt-p2v** ユーティリティは、AMD64 および Intel 64 のアーキテクチャーでのみ対応しており、IBM Z では提供されません。

移行

後のホストモデル (たとえば IBM z14 から z15) に正常に移行したり、ハイパーバイザーを更新したりするには、**host-model** の CPU モードを使用します。**host-passthrough** および **maximum** CPU モードは、一般的に移行に対して安全ではないため、推奨しません。

custom CPU モードで明示的な CPU モデルを指定する場合は、次のガイドラインに従ってください。

- **-base** で終わる CPU モデルは使用しない。
- **qemu**、**max**、または **host** CPU モデルは使用しない。

古いホストモデルへの移行 (z15 から z14 など)、または以前のバージョンの QEMU、KVM、RHEL カーネルへの移行の場合、最後に **-base** が付いていない使用可能な最も古いホストモデルの CPU タイプを使用します。

- ソースホストと宛先ホストの両方を実行している場合は、代わりに宛先ホストで **virsh hypervisor-cpu-baseline** コマンドを使用して、適切な CPU モデルを取得できます。詳細は、[仮想マシン移行のホスト CPU の互換性の確認](#) を参照してください。
- RHEL 9 でサポートされているマシンタイプの詳細については、[RHEL 9 仮想化で推奨される機能](#) を参照してください。

PXE インストールおよび起動

PXE を使用して IBM Z で仮想マシンを実行する場合は、`pxelinux.cfg/default` ファイルに特定の設定が必要です。以下に例を示します。

```
# pxelinux
default linux
label linux
kernel kernel.img
initrd initrd.img
append ip=dhcp inst.repo=example.com/redhat/BaseOS/s390x/os/
```

安全な実行

仮想マシンの XML 設定で `<launchSecurity type="s390-pv"/>` を定義することにより、準備されたセキュアなゲストイメージで仮想マシンを起動できます。これにより、仮想マシンのメモリーが暗号化され、ハイパーバイザーによる不要なアクセスから保護されます。

仮想マシンを安全な実行モードで実行している場合、次の機能はサポートされないことに注意してください。

- `vfio` を使用したデバイスパススルー
- `virsh domstats` および `virsh memstat` を使用したメモリー情報の取得
- `memballoon` および `virtio-rng` 仮想デバイス
- huge page を使用したメモリーバッキング
- ライブマイグレーションおよびライブ以外の移行
- 仮想マシンの保存および復元
- メモリースナップショットを含む仮想マシンスナップショット (`--memspec` オプションを使用)
- 完全なメモリーダンプ。代わりに、`virsh dump` コマンドに `--memory-only` オプションを指定してください。
- 248 以上の vCPU。セキュアゲストの vCPU 制限は 247 です。

関連情報

- [アーキテクチャー全体でサポートされる仮想化機能の概要](#)

24.6. ARM 64 での仮想化が AMD64 および INTEL 64 とどのように異なるか

ARM 64 システム (AArch64 と呼ばれる) 上の RHEL 9 の KVM 仮想化は、AMD64 および Intel 64 システム上の KVM とはいくつかの点で異なります。これには以下が含まれますが、以下に限定されません。

ゲストのオペレーティングシステム

現在、ARM 64 仮想マシン (VM) でサポートされているゲストオペレーティングシステムは RHEL 9 のみです。

Web コンソール管理

[RHEL 9 Web コンソールの仮想マシン管理](#) の一部の機能は、ARM 64 ハードウェアで正しく動作しない場合があります。

vCPU のホットプラグとホットアンプラグ

仮想 CPU (vCPU) を実行中の仮想マシン (vCPU ホットプラグとも呼ばれる) に接続することは、ARM 64 ホストではサポートされていません。さらに、AMD64 および Intel 64 ホストと同様に、実行中の仮想マシンからの vCPU の削除 (vCPU ホットアンプラグ) は、ARM 64 ではサポートされていません。

SecureBoot

SecureBoot 機能は、ARM 64 システムでは使用できません。

Migration

現在、ARM 64 ホスト間での仮想マシンの移行はサポートされていません。

メモリーページサイズ

ARM 64 は現在、64 KB のメモリーページサイズでのみ、また 64 KB のメモリーページサイズのホスト上でのみ仮想マシンの実行をサポートしています。ホストまたはゲスト内の 4 KB ページサイズは現在サポートされていません。

ARM 64 上で仮想マシンを正常に作成するには、ホストが [64 KB のメモリーページサイズのカーネルを使用](#) している必要があります。仮想マシンを作成するときに、たとえばキックスタートファイルに次のパラメーターを含めるなどして、**kernel-64k パッケージ** を使用して仮想マシンをインストールする必要があります。

```
%packages
-kernel
kernel-64k
%end
```

ヒュージページ

64 KB のメモリーページサイズを持つ ARM 64 ホストは、次のサイズの巨大なメモリーページをサポートします。

- 2 MB
- 512 MB
- 16 GB

ARM 64 ホストで透過的 huge page (THP) ファイルシステムを使用する場合は、512 MB の huge page のみがサポートされます。

SVE

ARM 64 アーキテクチャーは、Scalable Vector Expansion (SVE) 機能を提供します。ホストがこの機能をサポートしている場合、仮想マシンで SVE を使用すると、これらの仮想マシンでのベクトル数学計算と文字列操作の速度が向上します。

SVE のベースラインレベルは、それをサポートするホスト CPU でデフォルトで有効になっています。ただし、Red Hat では、各ベクトルの長さを明示的に設定することを推奨しています。これにより、仮想マシンは互換性のあるホスト上でのみ起動できるようになります。これを行うには、以下を行います。

1. CPU に SVE 機能があることを確認します。

```
# grep -m 1 Features /proc/cpuinfo | grep -w sve
```

```
Features: fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid
asimdrdm fcma dcpop sve
```

このコマンドの出力に **sve** が含まれている場合、または終了コードが 0 の場合は、CPU が SVE をサポートしています。

2. 変更する仮想マシンの XML 設定を開きます。

```
# virsh edit vm-name
```

3. **<cpu>** 要素を次のように編集します。

```
<cpu mode='host-passthrough' check='none'>
  <feature policy='require' name='sve' />
  <feature policy='require' name='sve128' />
  <feature policy='require' name='sve256' />
  <feature policy='disable' name='sve384' />
  <feature policy='require' name='sve512' />
</cpu>
```

この例では、SVE ベクトルの長さ 128、256、および 512 を明示的に有効にし、ベクトルの長さ 384 を明示的に無効にします。

CPU モデル

ARM 64 上の仮想マシンは現在、**host** CPU モデルのみをサポートしています。

仮想マシンの保存および復元

仮想マシンの [保存と復元](#) は現在、ARM 64 ホストではサポートされていません。

PXE

PXE (Preboot Execution Environment) での起動は機能しますがサポートされていないため、実稼働環境では使用しないことを Red Hat は強く推奨します。

PXE ブートが必要な場合は、**virtio-net-pci** ネットワークインターフェイスコントローラー (NIC) のみ可能です。さらに、PXE ブートには、仮想マシン UEFI プラットフォームファームウェア (**edk2-aarch64** パッケージでインストールされる) の組み込み **VirtioNetDxe** ドライバーを使用する必要があります。iPXE オプションの ROM はサポートされません。

デバイスメモリー

DIMM (Dual In-line Memory Module) や NVDIMM (Non-Volatile DIMM) などのデバイスメモリー機能は、ARM 64 では動作しません。

pvpanic

pvpanic デバイスは現在 ARM 64 ではサポートされていません。ARM 64 のゲスト XML 設定の **<devices>** セクションから **<panic>** 要素を必ず削除してください。この要素が存在すると、VM の起動に失敗する場合があります。

OVMF

ARM 64 ホスト上の仮想マシンは、**edk2-ovmf** パッケージに含まれる AMD64 および Intel 64 で使用される OVMF UEFI ファームウェアを使用できません。代わりに、これらの仮想マシンは **edk2-aarch64** パッケージに含まれる UEFI ファームウェアを使用します。これは、同様のインターフェイスを提供し、同様の機能セットを実装します。

具体的には、**edk2-aarch64** は組み込みの UEFI シェルを提供しますが、次の機能はサポートしていません。

- SecureBoot
- 管理モード

kvm-clock

kvm-clock サービスは、ARM 64 の仮想マシンで時間管理用に設定する必要はありません。

周辺機器

ARM 64 システムは、AMD64 および Intel 64 デバイスとは部分的に異なる周辺機器のセットをサポートします。

- PCIe トポロジーのみがサポートされます。
- ARM 64 システムは、**virtio-*-pci** ドライバーを使用して **virtio** デバイスをサポートします。さらに、**virtio-iommu** デバイスおよび **virtio-input** デバイスはサポートされていません。
- **virtiofs** 機能はテクノロジープレビューとしてのみ提供されているため、サポートされていません。
- **virtio-gpu** ドライバーはグラフィカルインストールでのみサポートされます。
- ARM 64 システムは、グラフィカルインストールのみで **usb-mouse** および **usb-tablet** デバイスをサポートします。その他の USB デバイス、USB パススルー、または USB リダイレクトはサポートされていません。
- Virtual Function/I/O (VFIO) を使用するデバイスの割り当ては、NIC (物理および Virtual Function) でのみサポートされます。

エミュレートされたデバイス

次のデバイスは ARM 64 ではサポートされていません。

- ICH9、ICH6、AC97 などのエミュレートされたサウンドデバイス。
- VGA カードなどのエミュレートされたグラフィックカード。
- **rtl8139** などのエミュレートされたネットワークデバイス。

GPU devices

ARM 64 システムでは **GPU デバイスの割り当て** はサポートされていません。

シリアルコンソールの設定

VM でシリアルコンソールをセットアップする 場合、**grubby** ユーティリティーで **console=ttyS0** の代わりに **console=ttyAMA0** カーネルオプションを使用します。

マスク不可割り込み

現在、ARM 64 仮想マシンにマスク不可割り込み (NMI) を送信することはできません。

ネストされた仮想化

現在、ARM 64 ホストではネストされた仮想マシンを作成できません。

v2v および p2v

virt-v2v ユーティリティーおよび **virt-p2v** ユーティリティーは、AMD64 および Intel 64 のアーキテクチャーでのみ対応しており、ARM 64 では提供されません。

24.7. RHEL 9 における仮想化機能のサポートの概要

以下の表は、利用可能なシステムアーキテクチャーで、RHEL 9 で選択された仮想化機能のサポート状態に関する比較情報を示しています。

表24.1 一般的なサポート

Intel 64 および AMD64	IBM Z	ARM 64
サポート対象	サポート対象	サポート対象

表24.2 デバイスのホットプラグとホットアンプラグ

	Intel 64 および AMD64	IBM Z	ARM 64
CPU ホットプラグ	サポート対象	サポート対象	サポート対象外
CPU のホットアンプラグ	サポート対象外	サポート対象外	サポート対象外
メモリーのホットプラグ	サポート対象	サポート対象外	サポート対象外
メモリーのホットアンプラグ	サポート対象外	サポート対象外	サポート対象外
周辺機器のホットプラグ	サポート対象	サポート対象 [a]	サポート対象
周辺機器のホットアンプラグ	サポート対象	サポート対象 [b]	サポート対象

[a] **virtio-*-pci** ではなく **virtio-*-ccw** デバイスを使用する必要があります。

[b] **virtio-*-pci** ではなく **virtio-*-ccw** デバイスを使用する必要があります。

表24.3 選択したその他の機能

	Intel 64 および AMD64	IBM Z	ARM 64
NUMA チューニング	サポート対象	サポート対象外	サポート対象
SR-IOV デバイス	サポート対象	サポート対象外	サポート対象
virt-v2v および p2v	サポート対象	サポート対象外	利用できません

サポートされていない機能の一部は、Red Hat Virtualization や Red Hat OpenStack プラットフォームなどのその他の Red Hat 製品でサポートしていることに注意してください。詳細は、[RHEL 9 仮想化で対応していない機能](#) を参照してください。

関連情報

- [RHEL 9 仮想化で対応していない機能](#)

